Scuola Normale Superiore

TESI DI PERFEZIONAMENTO IN CHIMICA
(Ph.D. Thesis in Chemistry)

Design and development of a cross-platform molecular viewer for Immersive Virtual Reality systems

Candidato *(Candidate)*	Relatori *(Advisors)*
Andrea Salvadori	Prof. Vincenzo Barone
	Dr. Giordano Mancini

Anno accademico 2016/2017

"Visualization is daydreaming with a purpose"

Robert “Bo” Bennett

Contents

Introduction

Scientific discoveries and technological evolution from the second half of the XX century made available computers capable of performing complex numerical simulations and measuring instruments ever more accurate and sophisticated. These tools brought an essential contribution to our understanding of the physical universe but at the cost of increasing the size and complexity of the data sets. While this situation led to the development of automatic data analysis procedures, it does not diminish the need for the insight and reasoning capabilities of the human mind.

Visualization is a discipline that tries to exploit the capabilities of the human visual system to quickly identify structures, patterns, an anomalies in images in order to stimulate the mental processes of insight and understanding about large sets of data or complex phenomena. The field of *Scientific Visualization* focusing on the graphical representation of molecular structures and properties is known as *Molecular Graphics*. The rapid evolution of computer graphics and *Virtual Reality* technologies has made feasible their employment in Scientific Visualization. In particular, the exploitation of *Immersive Virtual Reality* (IVR) may "*provide substantial benefit to computational molecular sciences*" [1]. The reason is that, since most of the analyzed data have are inherently three-dimensional, stereoscopic displays can definitely have a positive impact on their understanding, as well as on the visual identification of relations, patterns and anomalies in them. Important is also the ability to physically explore the world with the movements of our our body, in order to both observe the objects from different perspectives and to encourage spatial judgments (of sizes, distances and angles) based on proprioception.

Although scientists are trying to exploit of IVR technologies in molecular sciences since the late 60's (e. g., "Project GROPE" started in 1967 [2]), their use within scientific fields is still limited, partly due to limits and costs of the specialized hardware and partly to the infancy of software using such technologies [3]. However, the recent introduction of a new generation of consumer-grade immersive helmets (such as the *Oculus Rift* and the *Vive* from HTC and Valve) may change this scenario, hopefully leading to a wide adoption of IVR technologies both for research and dissemination purposes. The only obstacle, at this point, is the availability of proper software capable to fully exploit the potential of these technologies.

In this thesis, I will discuss the features, the design choices and the algorithmic details of *Caffeine*, a new molecular viewer developed at the SMART Laboratory of Scuola Normale Superiore. The project has two main objectives:

1. To exploit modern Immersive Virtual Reality (IVR) technologies.
2. To reduce the gap between the state-of-the-art research in molecular graphics and the actual molecular graphics systems used day-by-day by researchers and students.

This latest point is often underestimated: the use of advanced rendering algorithms is not aimed only at producing pleasant images, but mainly to obtain superior performance. In a VR environment, high frame-rates are essential to provide a fluid and immersive experience, and to avoid common symptoms like dizziness, headache and nausea. Furthermore, if carefully chosen, pleasant graphical effects can contribute to produce clearer and more informative images.

Caffeine supports both standard desktop computers as well as IVR systems such as the CAVE theater installed at SNS. It allows to visualize both static and dynamic structures using the most widespread representations (all-atoms and ribbons), iso-surfaces extracted interactively by volume data sets, and line charts displaying additional scalar data resulting from further data analysis. In an IVR context, these charts are visualized as part of the 3D scene and kept always in front of the user in an "augmented reality fashion". This vis-a-vis comparison between represented structures and charted data helps in perceiving interesting features and encourages the user to exploit his "chemical intuition". Finally, *Caffeine* has been successfully used in the VIS (Virtual Immersions in Science, http://vis.sns.it), project which was awarded a prize for science outreach [4].

This thesis is based on the following works:

- [5] "Graphical Interfaces and Virtual Reality for Molecular Sciences", A. Salvadori, D. Licari, G. Mancini, A. Brogni, N. De Mitri, and V. Barone, in Reference Module in Chemistry, Molecular Sciences and Chemical Engineering, Elsevier, 2014. DOI: 10.1016/B978-0-12-409547-2.11045-5
- [6] "Moka: Designing a Simple Scene Graph Library for Cluster-Based Virtual Reality Systems", A. Salvadori, A. Brogni, G. Mancini, and V. Barone, in Augmented and Virtual Reality: First International Conference, AVR 2014,

Lecce, Italy, September 17-20, 2014, Revised Selected Papers (L. T. De Paolis and A. Mongelli, eds.), vol. 8853 of Lecture Notes in Computer Science, pp. 333–350, Springer International Publishing, 2014. DOI: 10.1007/978-3-319-13969-2_25

- [7] “Immersive virtual reality in computational chemistry: Applications to the analysis of QM and MM data”, A. Salvadori, G. Del Frate, M. Pagliai, G. Mancini, and V. Barone, International Journal of Quantum Chemistry, vol. 116, pp. 1731–1746, nov 2016. DOI: 10.1002/qua.25207

A good part of the contents of [5] have been heavily re-elaborated, extended and updated to form the firsts three chapters of this thesis. The works [6] and [7], instead, have been included respectively as Chapter 4 and 7.

In particular, this thesis is organized as follows. Chapter 1 discusses the basic concepts of Scientific Visualization. Chapter 2 present a brief history of Molecular Graphics, followed by a survey on the graphical representations of molecular structures and properties and the related state of the art rendering algorithms. Chapter 3 provides a brief overview about Virtual Reality and discuss its employment in molecular sciences. Chapter 4 is constituted by the paper [6], which describes the motivations for the development of a Distributed Scene Graph as base of *Caffeine*, its design and implementation. The reasons of its recent removal from *Caffeine* are also discussed. Chapter 5 present *Caffeine* from the user’s point of view, illustrating its features and its use. Chapter 6 describes in detail some relevant aspects about the implementation of *Caffeine* and the related algorithms. Chapter 7 is constituted by the paper [7], which discuss some case studies specifically conducted to illustrate the usefulness and advantages that can be gained by analyzing the resulting data in a VR environment with *Caffeine*, and present some benchmarks conducted to test the performance of the program. This thesis concludes discussing the future perspectives of *Caffeine.*

1 Basic concepts of Scientific Visualization

1.1 Introduction to Visualization

Vision has an important role in human beings for the acquisition of knowledge : about one-third of the neurons of our brain cortex is devoted to this task [8]. From what said it follows that humans have a natural predisposition to to use pictures to communicate information: the earliest form of artworks we know of so far are cave paintings from about 40 thousand years ago [9], long before the appearance of the first known forms of writing, which are dated back to the second half of the fourth millennium B.C. [10].

Scientific discoveries and technological evolution of the XX century made available measuring instruments ever more accurate and sophisticated and computers capable of performing complex numerical simulations. These tools brought an essential contribution to our understanding of the physical universe but at the cost of the increasing size and complexity of the resulting data. While this situation led to the development of automatic data analysis procedures, it does not diminish the need for the insight and reasoning capabilities of the human mind. Citing Richard Hamming [11]: "*The purpose of computing is insight, not numbers*".

Visualization tries to exploit the capabilities of the human visual system to quickly identify structures, patterns, and anomalies in an image, so to stimulate the mental processes of insight and understanding about large sets of data or complex phenomena. To this end, *Visualization* studies the principles and techniques to meaningfully represent complex data sets by means of interactive computer graphics technologies. Visualization is widely used by scientists, engineers and managers both to *extract knowledge* (e.g. identify structures, trends, relationships and anomalies) about a phenomenon under study and for *communication* purposes (in order to effectively present and explain such results to an audience).

The discipline of Visualization can be classified in various sub-fields, according to the nature of the data they focus on, which often require dedicated strategies and algorithms for their management, analysis and graphical representation. Traditionally, the two major sub-fields are *Scientific Visualization* (*SciVis*) and *Information Visualization* (*InfoVis*): Scientific Visualization deals with quantities having an intrinsic spatial and/or temporal nature and resulting from measurements or numerical simulations; Information Visualization, on the other hand, focuses on abstract data and relations which are non-inherently spatial. Of course, being part of the same discipline, SciVis, InfoVis and other sub-fields share many goals and basic principles, and the techniques developed in one specific context could be successfully applied to other sub-fields. After all, as pointed out by Prof. Tamara Munzner[1] [12]: "*scientific visualization isn't uninformative, and information visualization isn't unscientific*". In the following, the discussion will focus on Scientific Visualization, being of primary interest for this thesis.

Until few decades ago, representation of scientific data usually consisted in illustrations on paper or models made of wood, metal or plastic (several examples are reported in [13]). Today, thanks to the advances in the field of real-time computer graphics, researchers can analyze the results of their simulations and experiments by means of *interactive* software. *Interactivity* plays a fundamental role in Visualization: allowing the user to change the representation of the data, filter the less relevant aspects, inspect different quantities related to a same phenomenon at the same time etc., all performed in *real-time*, provides a crucial help in understanding the behavior and the peculiarities of the phenomena under study. In this regard, a popular guideline for designing user interfaces for visualization systems is the so-called "*visual information seeking mantra*" coined by Shneiderman [14]: "*Overview first, zoom and filter, details on demand*". In fact, many data sets are so large that even if would be possible (from a technological point of view) to represent them in their entirety and in every detail, the obtained images will convey so many information that will result (almost) incomprehensible. For this reason it is convenient, in the first place, to provide to the user a "first look" of the entire data set (e.g. by hiding of most details or by aggregating the information related many data items and present the summarized info a single visual element), so that he can notice patterns and regions containing information of interest in the data. Then the user can focus on those regions through navigation (zooming) and discard unneeded data by filtering. Finally, the user can select a particular group of items to visualize

[1]Department of Computer Science, University of British Columbia, Canada

in detail the value of their attributes. Of course, this can't be a one-way process, since the user will probably need to switch back and forth between different levels of detail during his investigation [15].

1.2 The Visualization pipeline

Visualization can be thought as a process that takes raw data as input and produce an image on screen as output. Conceptually, this process can be described as a sequence of transformation steps, each one taking a set of data as input, and producing a different set of data as output (see Figure 1.1). This "visualization pipeline", first introduced by Upson et al. [16] and further refined by Haber and McNabb [17], is usually described in literature in terms of the following macro-steps (briefly discussed in the following sub-sections) : *acquisition*, *data filtering and processing*, *mapping* and *rendering*.

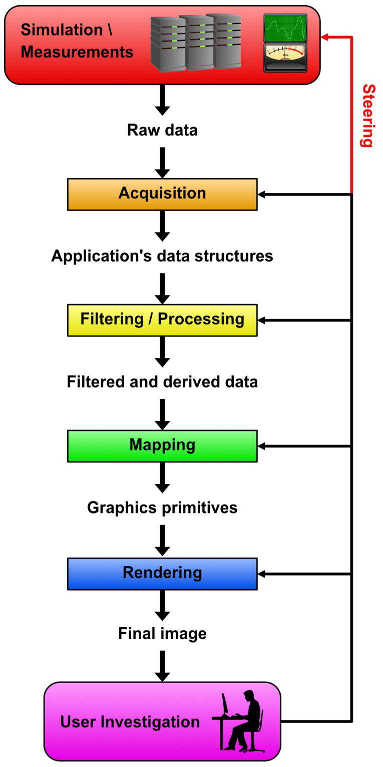


Figure 1.1: The visualization pipeline.

1.2.1 Acquisition

In the case of Scientific Visualizations the data to be visualized are obtained from real-world measurements or computer simulations. Once produced, they can be stored on file or a database (so to allow their analysis at a later time) or can fed into the visualization application as a live stream. In either case, the first step of the visualization process consists in acquiring these data and store them in data structures defined by the program. Usually this involves a conversion between the format in which these information are provided and the one adopted by the application (and possibly chosen by the user). In both choosing which data structures to employ and in implementing the conversion procedure, care should be taken in order to minimize the loss of information, which could led from inaccurate up to erroneous final results which can not be rectified in later stages. Furthermore, often the user is given the opportunity to import only a subset of the original data set. While this is sometimes necessary in the case of massive data sets (if the program does not implements "*out-of-core*" strategies), this pre-filtering may compromise the possibility to extract some derived information. In conclusion, as pointed out by Prof. Alexandru C. Telea[2] in his book about data visualization [18]: "*data importing step should try to preserve as much of the available input information as possible, and make as few assumptions as possible about what is important and what is not*".

As an example, consider the Protein Data Bank (PDB) file format [19], a widely employed file format for storing the structure of macromolecules. PDB files are constituted by a list of records having a fixed length of 80 characters. The ATOM and HETATM records provides information about an atom of the considered molecular system. When di erent alternate locations are reported for an atom, the file contains multiple records for the same atom, having the same atom name (that according to the file format specification *should* be unique only within a residue) but di erent serial numbers. In order to correctly identifying the alternate locations of an atom, a special field is allocated in each ATOM/HETATM record, denoting the location "name" with a character. Unfortunately, many cheminformatics libraries (including OpenBabel [20]) and molecular viewers just ignore this field. This deficiency, together with the not always strict observance of the rule of uniqueness of the atom name within a residue, makes impossible to recover information about alternate locations subsequently. An usual consequence of this misinterpretation of

[2]Department of Computer Science, Institute Johann Bernoulli, University of Groningen, Netherlands.

the acquired data is the construction of erroneous structures, presenting additional inexistent atoms and bonds.

1.2.2 Data filtering and processing

Since the imported information usually consist in an huge amount of data, their entire processing would preclude the responsiveness (and hence the interactivity) of the application. Furthermore, by also having a computer capable to process large data sets in real-time, the quantity of information displayed to the user would be overwhelming, thus preventing their understanding. To solve both problems, data are subject to *filtering*, so to select the portion of the entire data set to be further processed and finally displayed. Many filtering strategies exists. As examples, data can be filtered by spatial location (so to obtain detailed information about a set of elements lying within a specific region of space), by selecting only a subset of the attributes/quantities associated to each element or, in the case of time-varying data, according to a given time interval.

Apart from filtering and according to the aims of the user's analysis, imported data can be processed in a number of ways before being visualized. An example is the *aggregation*, a data reduction method in which the original data set is partitioned and each partition is replaced by an aggregate value (computed as a function of the elements of the partition). In other cases, instead, there might be the need to interpolate the data, so to obtain a finer-grained data set. Finally, derived quantities can be computed.

Multiple filtering and processing transformations, combined in various ways, can be employed as part of a single visualization, depending on the information of interest for the specific analysis performed by the user and on the characteristics of the related data.

1.2.3 Mapping

The purpose of this step is to map data resulting from filtering/processing procedures in sets of graphics primitives (e.g. points, lines, triangles) and related attributes (e.g. color, transparency, reflectance, surface textures) to be fed into the rendering engine to generate an image on screen. The *mapping functions* (also known as “*transfer functions*”) implemented by this stage of the pipeline are the core of the visualization process, since they define how numerical data will be represented in the resulting image.

1.2.4 Rendering

The *rendering* process generates 2D images starting from the graphics primitives and related attributes produced by the mapping stage. It will be discussed in section 1.3.

1.2.5 Results investigation and user interaction

The images generated by the visualization process are analyzed by the user with the aim to gain further knowledge regarding the phenomenon under investigation. It is important to note that, in most cases, a single attempt will not be sufficient to obtain a full comprehension of the data: the user will probably iterate over the visualization process multiple times, e.g. by changing the filtering parameters so to visualize different portions of the data set or acting on the mapping functions in order to produce different representations. The ability of the user to interact with the visualization systems transform visualization from a linear to a cyclic process [16].

In theory, and as shown in Figure 1.1, the user could affect every stage of the visualization process. In practice, however, how and how much the user can affect the process depends on the specific visualization system. Particularly interesting are those systems in which the user not only can monitor in real-time the results produced by a running simulation, but can also interactively "*steer*" the course of the simulation by adjusting some of its parameters. Systems providing such feature are known as "*interactive steering systems*".

1.2.6 Implementing the visualization pipeline

Visualization pipeline also provides a good abstract model for the implementation of visualization software libraries and tools, since it maps well with the *data-flow programming paradigm*. According to this paradigm, the code is structured in the form of a Directed Acyclic Graph (DAG), in which each node is an executable entity that wait for incoming data in its "*input ports*", transforms them and forwards the results to the nodes "connected" to its "*output ports*". The arcs of the graphs, instead, are unidirectional "*channels*" connecting the output ports of a node to the input ports of a following node. In other words, in this model the data "flow" from one or more "*source nodes*" to one or more "*sink nodes*", passing through a sequence of "*processing nodes*" that performs a transformation on them and forwards the results to the subsequent stages. An advantage of the dataflow paradigm is its inherent support to concurrency: in fact, since each node is an processing module that (once received the proper input) can perform its task independently by the other nodes, the execution of each node can be delegated to a separate thread or process. Furthermore, programs following this paradigm are well suited to be implemented graphically, by means of so called "*visual programming languages*". Further details on the *data-flow programming paradigm* can be found in [21].

In the context of visualization, each nodes of the data-flow will implement an acquisition, data filtering and processing, mapping or rendering procedure. Several popular scientific visualization libraries and tools employ this paradigms, for example the Application Visualization System (AVS) [16], its successor AVS/Express [22], IRIS Explorer [23] and the Visualization Toolkit (VTK) [24].

1.3 Overview of the rendering process in real-time computer graphics

The process of generating 2D images starting from the description of a 3D scene is called "*rendering*". The input of this process is usually constituted by:

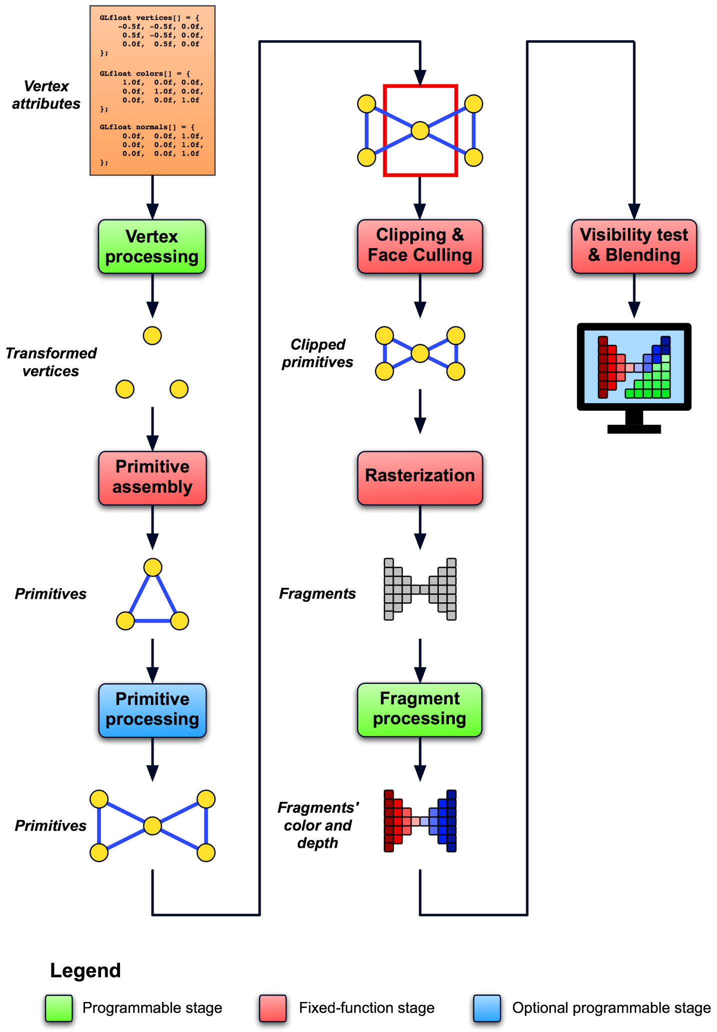
- The description of one or more virtual cameras. In fact, the rendering process tries to simulate the result that would be obtained by using cameras to film a virtual world. Although in the simplest case only one camera is used, more advanced scenarios where multiple cameras are employed are possible. In such cases, multiple rendering "*passes*" are performed for each frame, one pass for each active camera. The final image will then be obtained by *compositing* the images related to each camera (e.g. by dividing the final image in multiple regions). It follows that the properties of the virtual cameras, such as their position, orientation, field of view etc., are part of the input of the rendering process.
- One or more sets of geometric primitives (e.g. points, lines and triangles) representing the objects of the virtual world. For each geometric primitive a set of attributes associated to each of its vertices is provided. The type of

these attributes is application dependent, but common choices are the position of the vertex, its color and an associated normal vector. In the case of 3D objects, it is common to describe only their external surface, approximated by a set of polygons. Since graphics hardware is heavily optimized to process triangles, the use of this kind of polygons is the de-facto standard in real-time computer graphics. One relevant exception is constituted by *direct volume rendering* representation, in which a volume of scalar data is drawn by employing an optical model which defines how the light is reflected, scattered, absorbed, occluded or emitted by the volume, as a function of the scalar value associated to each *voxel* [25].

- A set of transformation matrices defining the position, the orientation and the size of the objects within the 3D world.
- A set of light sources with their properties (type, position, color, ...).
- Etc.

In the following the functioning of the rendering process is briefly described, with particular reference to *real-time computer graphics*, which is of primary interest for this thesis. In this field, the rendering process is often described in terms of a *pipeline*, in which each stage takes as input a sequence of elements, performs a specific elaboration on each element, and produces as output another sequence of elements. Since each stage operates on different elements and each element is considered in isolation, stages can operate in parallel and (in most cases) they can be internally parallelized by replicating their functionality on multiple processing units, so that each stage can process multiple elements at the same time. In order to boost the performance, *Graphics Processing Units* (*GPU*s) which implements in hardware the rendering pipeline are usually exploited. While earlier generation of *GPU*s was specialized processor with a hard-wired behavior (although configurable by means of a graphics API such as *OpenGL* or *DirectX*), their evolution led to an increasing programmability of their functioning, by means of dedicated programs known as *shaders.* Ever more stages of rendering pipeline have become programmable, giving the opportunity to replace their standard behavior with a custom one defined in a dedicated *shader.* It is important to note that one instance of shader is executed for each element to be processed by the related stage of the pipeline and that these instances are executed in parallel on different *cores* (processing units) of the *GPU.*

A simplified rendering pipeline is summarized in Figure 1.2 and its stages will be described in the following sub-sections. Although the discussion will primarily refers to *OpenGL 3.3*, the exposed concepts are mostly generals. Interested readers can refer to [26] for an in depth discussion about the rendering pipeline of the latest version of OpenGL.



```
GLfloat vertices[] = {
    -0.5f, -0.5f, 0.0f,
     0.5f, -0.5f, 0.0f,
     0.0f,  0.5f, 0.0f
};

GLfloat colors[] = {
     1.0f,  0.0f, 0.0f,
     0.0f,  1.0f, 0.0f,
     0.0f,  0.0f, 1.0f
};

GLfloat normals[] = {
     0.0f,  0.0f, 1.0f,
     0.0f,  0.0f, 1.0f,
     0.0f,  0.0f, 1.0f
};
```

Figure 1.2: A simplified rendering pipeline.

1.3.1 Vertex processing

The primary purpose of this stage is to compute the coordinates of each vertex of each geometric primitive in a normalized reference frame. Since modern graphics hardware allows to program the operation performed by this stage by means of a so called "*vertex shader*", there are no restriction on how the vertex coordinates are computed. However, a common procedure to compute such coordinates is the following:

- Vertex coordinates are provided as input to the pipeline expressed in their own reference frame.
- Further three matrices are passed as input to this stage of the pipeline, commonly known as "*model matrix*", "*view matrix*" and "*projection matrix*".
- The *model matrix* transforms the vertex coordinates in the "global" reference frame of the virtual world.
- Then the *view matrix* is applied to transform these coordinates in the camera's reference frame. In fact, the view matrix is computed as a function of the position and on the orientation of the camera with respect to the "global" reference frame.
- Finally, the vertex coordinates are once again transformed by means of the projection matrix. Despite its name, the projection matrix does not projects the three-dimensional vertices on a planar surface. Instead, its purpose is to pass from a three-dimensional coordinate system (the camera's reference frame) to another three-dimensional coordinate system in which the "*view volume*" (i.e. the closed region of space delimited by six planes which defines the field of view of the camera) is distorted in a normalized cube (or rectangular parallelepiped). This last transformation has two main purposes [27]:
  - It simplifies the definition of the following stages of the pipeline, since both the perspective and orthographic view volumes are reduced to a canonical orthogonal one.
  - It simplifies the "*clipping*" process (taking place later in the pipeline), since the six clipping planes (bounding the normalized view volume) are aligned with the coordinate axes.

Apart from position, others per-vertex attributes (such as the associated color and normal vector) can be fetched and/or computed in this stage and propagate through the pipeline. Finally, it is important to note that the *vertex shader* is executed once for each each vertex and it can access only the data related to that vertex, without any knowledge about the other vertices nor about the geometric primitive the processed vertex belongs to.

1.3.2 Primitive assembly and processing

The output of the first stage of the pipeline is fed into the "*primitive assembly*" stage, where the vertices are collected and converted in a sequence of primitives (points, lines or triangles). Then an optional "*primitive processing*" stage may take place. The operations performed by this stage must be defined by the developer by means of an appropriate "*geometry shader*". The *geometry shader* is executed once for each primitive, have access to all the data related to the vertices forming that primitive and can output zero or more primitives, possibly of a different type. That implies that a geometry shader not only can transform the data of the vertices of the primitive, but may also discard (i.e. remove from the pipeline) the primitive, generate multiple instances of the same primitive type, or output one or more primitives of a different type. As an example, a geometry shader may take a line as input and output zero, one or more triangles. The type of the input / output primitives accepted / generated by a *geometry shader* is fixed as part of its definition. However the number of the outputted primitives is not fixed (although upper-bounded), since it depends on the logic implemented by the shader and on its input.

1.3.3 Clipping and face culling

The "*clipping*" and "*face culling*" processes are performed by a non-programmable (yet configurable) stage of the pipeline. Their purpose is to detect and discard primitives that will not be visible in the final image, so to not waste further processing on them.

The *clipping* process tests if the primitive lies completely inside, partially inside or completely outside the normalized view volume. In the first case the primitive is passed to the next stage of the pipeline without modifications. If instead the primitive lies completely outside the view volume it is discarded. Finally, if the primitive lies across the boundaries of the view volume, then the part lying outside is clipped away. In the simpler cases this operation only requires the alteration of some vertices. However, in the case of triangles, it may result in the generation of multiple triangles replacing the original primitive.

As previously mentioned, in real time computer graphics the objects of the virtual world are commonly described by approximating their external surface with a set of triangles, called "*triangle mesh*". Since these meshes are usually closed and opaque, and since triangles in 3D space have two sides, each triangle of the mesh will have one side oriented towards the interior of the object (known as "*back*" side) and the other side pointing outwards (known as "front" side). It follows that a triangle belonging to a closed opaque mesh can be visible only if it's "front" side is oriented toward the virtual camera: in fact the triangles whose "back" side point toward the camera depict the internal side of a closed surface, so they will not appear in the final image. This fact is exploited by the "*face culling*" process, who discards from the pipeline the triangles whose "back" side points toward the camera, so to avoid useless further processing. In the case of *OpenGL*, the developer is responsible for defining which is the "front" and the "back" side of each triangle. This is done by providing the vertices of triangles in a conventional order. Finally note that, since there are cases in which open or semi-transparent surfaces have to be drawn, "*face culling*" can be enabled or disabled on a "per-object" basis.

1.3.4 Rasterization and fragment processing

"*Rasterization*" is the process of determining which pixels of the final image might be covered by the primitive being processed. The result of such process is a set of "*fragments*" corresponding to the input primitive. *Fragments* are a sort of "proto-pixels" whose processing will result in the generation of real pixels on final image. In particular, fragments not only have a 2D coordinate identifying the position on final image they are trying to occupy, but they also maintains a *depth* value. Together, these 3D coordinates are known as "*window coordinates*". Furthermore, each fragment carries with it a set of attributes obtained by interpolating vertex attributes across the primitive.

The set of *fragments* resulting from rasterization are fed into the subsequent stage of the pipeline, in charge of their processing. The operation performed by this stage can be programmed by means of a "*fragment shader*". Similar to other *shaders* an instance of *fragment shader* is executed for each fragment (usually multiple instances run in parallel on dedicated *cores* of the *GPU*), and each instance is responsible for outputting a color and a depth value for the processed fragment, without any knowledge about the other fragments. Although the main purpose of fragments shaders is to compute a color for the processed fragment as a function of its attributes, they can also discard the fragment or adjust its depth. However they can't modify the fragment's position within the final image (i.e. they can modify only on the z component of the *window coordinates* associated to the fragment).

1.3.5 Visibility test and blending

The last step of the rendering process consists in adding the processed fragments to the "*color buffer*", a block of memory (usually allocated on the *GPU*'s memory) storing the color for each pixel of the final image. Before storing the color of an incoming fragment into the color buffer, however, a visibility test is performed. This is necessary because the incoming fragment could be occluded by another fragment (processed earlier) belonging to a primitive closer to the virtual camera in the 3D scene. The standard algorithm employed in interactive computer graphics for visibility testing is known as "*Z-Buffering*". This algorithm requires the use of another buffer, the *"Z-Buffer"*, having the same resolution of the color buffer. Each element of the *Z-Buffer* stores the minimum depth value among all the fragments already processed for the corresponding pixel. When a new fragment is fed into this stage, its depth value is compared with the one stored in the corresponding element of the Z-Buffer: if the fragment's depth is smaller than the one contained in the Z-buffer (i.e. the fragment is closer to the camera than the previous ones), then the new color and depth are written respectively into the *color buffer* and *Z-buffer*. Otherwise the fragment is discarded. It is important to note that, although this algorithm works perfectly when drawing opaque objects occluding each others (the common scenario), there are cases where this behavior is not appropriate (e.g. rendering of semi-transparent objects or other advanced graphical effects). For that reason, as in the case of every "*fixed-function*" stage of the pipeline, the functioning of this stage is configurable: for example it is possible to disable the depth test for a subset of the 3D objects, or to keep the test active while disabling the writing on the *Z-Buffer* for some objects, or even change the testing condition. Furthermore, while the default behavior in the case of a fragment passing the depth test is to store its color in the corresponding element of the color buffer (thus replacing the previous content of that element), it is also possible to configure this stage such that the old and the new color will be combined according to a specified "*blend equation*". This procedure is known as "*blending*", and it is used to simulate semi-transparent materials as well as other various kinds of graphical effects.

1.3.6 Texture mapping

Texture mapping was born as an efficient method to simulate materials in three-dimensional virtual scenes and, in its basic form, consists in “attaching” one or more bi-dimensional images to the surface of a three-dimensional object (like a decal). This techniques was introduced in 1974 by Ed Catmull [28] and refined by Blinn and Newell in 1976 [29]. However, graphics hardware natively supporting texture mapping was not available until early 1990s (the first was the Silicon Graphics RealityEngine released in 1992 [30]). Today, texture mapping is widely supported (i.e. accelerated in hardware) by every GPU.

In order to select a particular piece of image to be applied over a surface, a coordinate system must be associated to the *texture* (image), known as *texture space.* In this way it is possible to pick the color corresponding to a particular point of the texture by knowing the point’s coordinates in texture space. It follows that, when defining the geometry of a 3D object to texturize, proper texture coordinates must be included among the attributes of each vertex. During *rasterization*, each fragment resulting from processing a geometric primitive (e.g. a triangle) will be coupled with proper texture coordinates, computed by interpolating those of the vertices of the primitive. Then, during *fragment processing*, the texture is sampled as a function of the texture coordinates of the fragment. The resulting color will be then employed in the computation of the final color for the fragment.

Nowadays, texturing capabilities provided by modern hardware and graphics libraries allows to use texture mapping for purposes that go far beyond to apply a preloaded image to surface. By allowing to define 1D, 2D, or 3D textures storing integer or floating-point numbers representing arbitrary quantities, texture mapping is actually a mechanism by means of which to transfer sets of sampled values to the GPU. Note that, starting from these discrete data sets and thanks to the automatic interpolation performed in hardware when sampling a texture, it is very easy and efficient to approximate the value of the considered quantity for a generic point (within the ranges of the dataset).

Texture mapping is widely employed in scientific visualization algorithms, for example to implement color mapping, direct volume rendering, volume slicing etc. (see section 2.2 for examples of these visualization techniques in the context of molecular properties).

2 Molecular Graphics

Molecular Graphics is a field of Scientific Visualization focusing on the graphical representation of molecular structures and related properties. According to the "Glossary of terms used in computational drug design (IUPAC Recommendations 1997)" [31], *Molecular Graphics* can be defined as:

"*a technique for the visualization and manipulation of molecules on a graphical display device*"

Although the modern conception of Molecular Graphics (as well as Scientific Visualization) implies the use of computer graphics technologies, the description of molecules in a graphical way has been a fundamental topic of chemistry since the XIX century. In fact, graphical representations provide a concise and convenient way to record and communicate knowledge about molecules, and in some cases they also lead to further discoveries (some important examples are quoted in the following).

2.1 A brief history of Molecular Graphics

This section briefly retrace the history and the evolution of the conventions and methods proposed from the mid of the XIX century to visually represent molecular structures: from bi-dimensional diagrams on paper (structural formulas) to the advent of the computer graphics, passing through the construction of physical models. The discussion is based on the timeline elaborated by James A. Perkins in his work on the history of molecular representations [32, 33].

2.1.1 Structural diagrams

Although the origin of diagrams to depict the arrangement of atoms in compounds is unknown, early structural diagrams can be found in a paper by Archibald Scott

Couper published in 1858 [34], in which atoms were indicated by their chemical symbol and bonds was depicted as dotted lines. This notation was later refined by Alexander Crum Brown [35, 36] and by August Kekulé [37]. However, these representations lacked of stereochemical information because the spatial arrangement of atoms was unknown at the time. Near the end of the nineteenth century, Hermann Emil Fischer proposed a new bi-dimensional notation for the representation of carbohydrates, today known as *Fischer projection*, in which horizontal lines depicts bonds pointing above the plane of the paper, while vertical lines depicts bonds pointing below the plane of the paper [38]. Today, a common way to represent molecules is according to the *Natta projection* [39] (named after Giulio Natta and also known as "*zigzag structure*") in which stereochemical information are provided by means of wedge lines (representing bonds bonds pointing above the plane of the paper) and dashed lines (representing bonds bonds pointing above the plane of the paper). An example of Natta projection is shown in Figure 2.1.

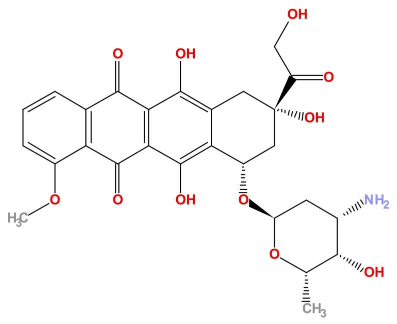


Figure 2.1: Natta projection ("*zigzag structure*") of doxorubicin, an anti-cancer drug.

2.1.2 Physical models and molecular illustrations

Although structural formulas are a convenient way to represent small molecules and functional groups, they are not suited to visualize the structure of macromolecules, such as proteins and nucleic acids. Before the availability of computer graphics, three-dimensional physical models were build, using wood, metal, paper or plastics. An important example of discoveries lead by physical models, was the intuition by Linus Pauling about the alpha-helix structure. In 1948, while Pauling was in bed recovering from a cold when...

"I took a sheet of paper and sketched the atoms with the bonds between them and then folded the paper to bend one bond at the right angle, what

I thought it should be relative to the other, and kept doing this, making a helix, until I could form hydrogen bonds between one turn of the helix and the next turn of the helix, and it only took a few hours of doing that to discover the alpha-helix." [40]

A formal presentation of the alpha-helices was published in 1951 in a paper by Pauling, Corey and Branson [41], rapidly followed by another paper by Pauling and Corey introducing the beta-sheets [42]. A couple of years later, in 1953, Corey and Pauling published a paper in which they describe a method to build accurate physical models of amino acids, peptides and proteins [43]. In these models atoms was represented as a spheres with radius proportional to the corresponding van der Waals radius ("*space-filling*" representation). Later, Walter Koltun refined the method for the construction of these types of physical models and called them "*Corey-Pauling-Koltun*" (CPK) models [44].

In 1953, James Watson and Francis Crick published a paper on Nature describing the double helix structure of deoxyribose nucleic acid (DNA) [45]. They also build a six feet tall physical model in metal of part of a DNA (see Figure 2.2).

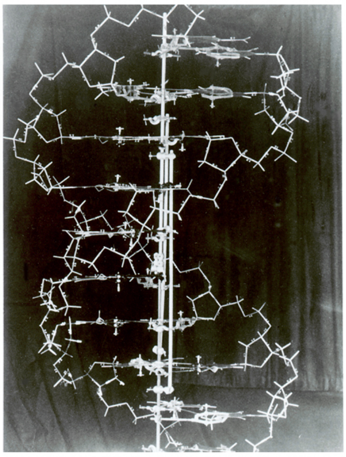

Courtesy of Cold Spring Harbor Archives. Noncommercial, educational use only.

Figure 2.2: Model of part of a DNA molecule build by Watson and Crick (1953).
Courtesy of and copyright by the Cold Spring Harbor Laboratory (http://www.dnalc.org).
Licensed under a Creative Commons Attribution - Noncommercial - No Derivative Works 3.0 United States License (http://creativecommons.org/licenses/by-nc-nd/3.0/us/).

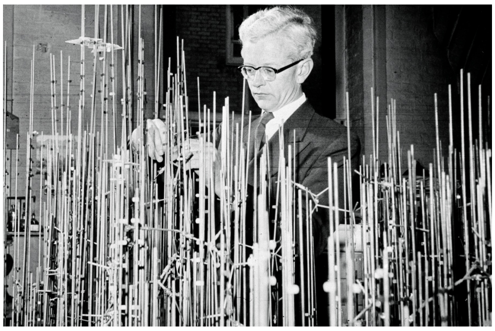
At the end of the 1950s, John Kendrew and co-workers built the first models of a macro-molecule, the Myoglobin. They subjected to X-ray diffraction a crystal of muscle cell of sperm whale, and computed the three-dimensional inverse Fourier transformation at six angstroms resolution on a EDSAC Mark I computer at Cambridge [46]. Contour maps was derived from the results and traced on sixteen lucite sheets, each one representing a bi-dimensional slice of the electron density distribution within the cell. The electron density across the cell was approximated by stacking the sheets on top of one another. These data was employed to build a rough model of the protein (Figure 2.3a), showing the regions of space having high electron density. Since these regions are mainly due to the presence of the backbone of the polypeptide chain or of the heme group, the resulting model provided a coarse approximation of the tertiary structure of the protein. However, the Fourier synthesis at six angstroms resolution was too coarse to observe details of the polypeptide at the atomic level. Driven by the desire to build a detailed three-dimensional structural model of a protein, Kendrew and co-workers repeated their analysis at a finer resolution [47]. They pushed the new EDSAC Mark II computer to its limits by computing a Fourier synthesis at two angstroms resolution, achieving a detailed representation of the electron density distribution traced on fifty lucite sheets. In order to build the skeletal model, they erected a “forest” of steel rods on which they placed colored clips to represent points of high density (various colors was used to represent different density values). Finally they built the skeleton model by careful observing the spatial distribution of the clips (see Figure 2.3b).

The 1960s also saw the affirmation of “molecular illustration”, thanks to very talented artists like Irving Geis and Roger Hayward. A couple of paintings by Geis illustrating molecular structures are shown in Figure 2.4. Probably the most famous drawings by Geis is the one depicting the skeletal structure of myoglobin (Figure 2.4a), accompanying the Kendrew’s article of 1961 [47].

Skeleton models like the one made by Kendrew and co-workers gained some popularity, although difficult to build. To simplify their construction, Frederic Richards proposed in 1968 a device (later nicknamed “*Richard’s box*”) capable of providing an *optical* superposition between the electron-density maps and the skeleton model under construction [49].

In the 1970s another type of physical models of polypeptides take hold, called “*alpha-Carbon models*”. In these models, the tertiary structure of polypeptide chains was approximated by bending and/or soldering metal rods so to obtain the trace of the alpha-carbons. In other words, in these models, each linear trait of rod represented a segment connecting the alpha-carbons of two consecutive amino-acid residues in the chain. While early construction methods (such as the one proposed

(a) "Sausage" model of myoglobin [46]. The white tubes are a coarse approximation of the tertiary structure of the protein, while the red disk represents the heme group.
Courtesy of and copyright by the Medical Research Council Laboratory of Molecular Biology in Cambridge, United Kingdom.



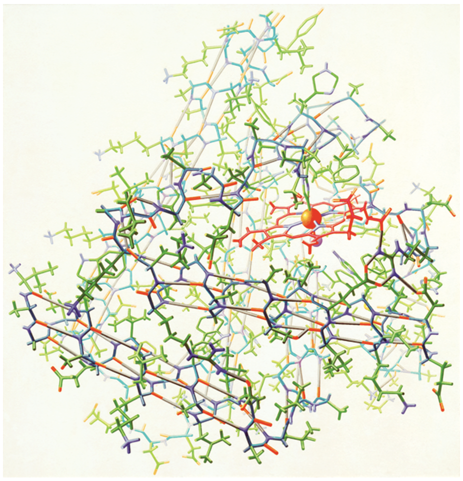
(b) Kendrew with his "forest of rods" model of myoglobin [47].
Courtesy of and copyright by the Medical Research Council Laboratory of Molecular Biology in Cambridge, United Kingdom.

Figure 2.3: Kendrew's models of myoglobin.

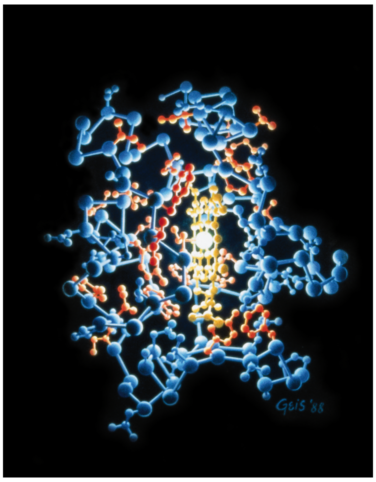
by Haas [50]) required to weld together linear traits of rod, in 1972 Byron Rubin and Jane Richardson invented a machine able to produce an alpha-Carbon model of a polypeptide by bending a single metal wire [51]. Alpha-Carbon models becomes popular thanks to their small size and weight (hence portability), and their analysis and comparison led to significant scientific discoveries, as Eric Martz[1] recount on his website on the history of visualization of biological macro-molecules [52]:

> "*An example illustrating the importance of models from Byron's Bender occurred at a scientific meeting in the mid 1970's. At this time, less than two dozen protein structures had been solved. David Davies brought a Bender model of an immunoglobulin Fab fragment, and Jane and David Richardson brought a Bender model of superoxide dismutase. While comparing these physical models at the meeting, they realized that both proteins use a similar fold, despite having only about 9% sequence identity. This incident was the first recognition of the occurrence of what is now recognized as the immunoglobulin super-family domain in*

[1]Professor Emeritus, Department of Microbiology, University of Massachusetts, Amherst.



(a) "Myoglobin" by Irving Geis. This painting was published in [47] *Illustration, Irving Geis. Used with permission from the Howard Hughes Medical Institute (www.hhmi.org). All rights reserved.*



(b) "Cytochrome C" by Irving Geis. This painting was published in [48] *Illustration, Irving Geis. Used with permission from the Howard Hughes Medical Institute (www.hhmi.org). All rights reserved.*

Figure 2.4: Paintings depicting molecular structures by Irving Geis.

proteins that are apparently unrelated by sequence. The insight was published in a paper entitled 'Similarity of three-dimensional structure between the immunoglobulin domain and the copper, zinc superoxide dismutase subunit' [53]".

The evolution of the alpha-Carbon models are the nowadays called "*ribbon diagrams*", in which the backbone of a polypeptide is represented by a smooth ribbon whose spatial arrangement approximate the trace of the alpha-carbons, while different colors and/or shapes are used to visualize secondary structures. Early ribbon representations was employed since the end of the 1960s (e.g. they appears in [54–56] etc.), but they becomes popular in the 1980s thanks to a survey on protein structures by Jane Richardson, in which the author illustrated the full range of known protein structures (75 at the time) with a consistent representation (the ribbon representation still in use today), so to allow their visual comparison and classification [57].

2.1.3 Computer-based Molecular Graphics

The first computer system capable of providing a graphical and interactive representation of a molecular structure was developed at the Massachusetts Institute of Technology in the early 1960s [58]. The structure was displayed in a “wireframe” representation by a computer-controlled monochrome oscilloscopes. The program was actually a molecular modeler for proteins, since polypeptide chains could be built by sequentially specifying the type of the amino-acid residues, the angles formed by subsequent peptide planes and the rotation angles of the side chains. This choice derives from the need to avoid to specify the three-dimensional coordinates for each atom: these coordinates were then automatically calculated by the program by employing a fixed predefined structure for the standard amino-acid residues. The program was also able to compute an approximate folding for the protein, by minimizing the total free energy of the system. Since the automatic procedure usually resulted in a configuration with a local minimum energy, the experience and the insight of the user was exploited to understand the reason for the local minimum. The user could then chose among a set of predefined procedures to introduce pseudo-energy terms in the system, so to pull or push parts of the structure. Users could interact with the system by typing commands on a keyboard, by acting on a “globe” to adjust the direction and speed of rotation and by pointing with a light pen.

In the same period, Carroll Johnson developed the *Oak Ridge Thermal Ellipsoid Plot Program* (ORTEP) [59]. This program allowed to draw molecular structures by means of a pen plotter. The structures was depicted in a “*ball-and-stick*” representation, but with the ability to replace spheres with ellipsoids to show the thermal-motion probability on the atomic sites. Furthermore, stereoscopic pairs of a same model could be drawn. Being able to automatically produce drawings of molecular structures, ORTEP was widely employed by crystallographers to produce illustrations for presentations and publications.

In 1971 Lee and Richards introduced the concept of “*Solvent-Accessible Surface*” (SAS) and implemented a program capable of computing the accessible surface area of atoms, or groups of atoms, with respect to solvent molecules [60]. The program was also able to plot slices of the van der Waal’s and/or of the Solvent-Accessible surfaces of a molecular system. By reproducing these slices on a stack of clear plastic sheets, the authors was able to built physical models representing these two types of surfaces.

Until the early 1970s, computers were employed to compute electron density maps from X-ray diffraction patterns and began to be used to draw molecular structures thanks to the early molecular graphics systems. However, the determination of the atomic structure from electron density maps was still a manual process, involving the construction of a Kendrew's-like physical model from which the coordinates of the atoms had to be measured and then passed as input to the molecular graphics system. In the following years several computer system was designed with the aim of analyze electron density maps and helping the users to construct three-dimensional molecular structures from them. Because of their purpose, these systems was initially called "*electronic Richard's box*es". Some of the most popular were *GRIP-75* [61] (said to be the computer graphics system thanks to which the complete three-dimensional structure of a large molecule was determined for the first time [62]), *Molecular Modeling System-X* (*MMS-X*) [63] and *FRODO* [64].

The advent of raster displays pushed the development of algorithms to create CPK representations of molecules. Among the several proposed techniques, Porter presented in 1978 a fast algorithm to generate images of shaded intersecting spheres without creating polygonal models [65]. The importance of Porter's work is also due to the fact that it represent the starting point on top of which Roger Sayle developed the rendering algorithm of *RasMol* (see below).

In the same year, Greer and Bush produced the first computer generated images of "*Solvent-Excluded Surface*" (SES) [66], a concept introduced just a year earlier by Richards [67]. In particular, Greer and Bush proposed a numerical method for computing the SES of a molecule according to a chosen view direction, and to plot the resulting surface topography. These plots could also be enriched by others atomic or molecular attributes (such as polarity, charge distribution etc.). In 1983 Connolly presented the first analytical description of the SES and a method for its computation [68]. He also described two applications of this formulation: the computation of the area of the SES and its visualization by means of vector computer graphics. A couple of years later, Connolly proposed the first algorithm to triangulate the SES starting from its analytical piecewise description [69]. Thanks to these and others contributions on the subject, the SES is also known as "*Connolly surface*". After the Connolly's original algorithm, several others have been developed to compute the SES analytically in a more fast and/or reliable way. Particularly relevant is the one proposed by Sanner in 1996 [70]. By exploiting the concept of "Reduced Surface" (RS, a compact way to describe the geometrical characteristics of molecular surfaces, developed by Sanner for his PhD thesis) he was able to implement a fast and reliable program to compute the analytical description of both SES and SAS. The program was named MSMS, and it was capable of computing and exporting the RS, the analytical description of the SAS/SES and a possible triangulation. The MSMS software package is currently used by many popular molecular graphics systems (such as VMD [71] and UCSF Chimera [72]) to build molecular surfaces. In the same year, another notable algorithm for deriving the analytical description of the SES, called "*contour-buildup*", algorithm presented by Totrov and Abagyan [73]. As discussed in section section 2.2.3, both the RS and the "*contour-buildup*" algorithm are at the base of several state-of-the-at methods for computing and visualizing the SES.

At the end of the 1980s Mike Carson presented an algorithm to efficiently draw three-dimensional ribbon diagrams of proteins [74, 75]. Although other computer programs have been previously implemented to generate ribbon drawings (see [76] as an example), the one proposed by Carson can be considered a milestone in the history of Molecular Graphics, since many of today's molecular graphics systems (including *Caffeine*) use an improved version of his algorithm. In his papers, Carson describes how to smooth the alpha-Carbon trace of a polypeptide by means of a B-spline curve and how to generate a 3D ribbon following that curve and oriented according to the peptide planes. Later Carson refined his algorithm and implemented it in a program named "*RIBBONS 2.0*" [77], capable of drawing customizables shaded ribbons diagrams in real-time on a Silicon Graphics 4D workstation.

In 1991 the first version of *MolScript* [78] was released. This program was focused on the production of publication-quality illustrations of molecular structures. The program required as input one or more coordinate files and a script file containing the parameters for generating the 3D scene (which part of the structure to draw and in which representation, the position and orientation of the resulting 3D object etc.) and other graphical settings. *MolScript* supported several representations such as wire-frame, ball-and-stick, CPK and ribbons that could be mixed freely. Furthermore, text could be added to label the displayed elements. The resulting image was outputted as a *PostScript* file (a file format for describing vector graphics, supported by many printers of the time), but the program also allowed to export the resulting 3D scene as input files for *Raster3D* [79] (a ray-tracing software for molecular graphics). Although *MolScript* was not an interactive program, it's use in conjunction with *Raster3D* becomes one of the most popular ways to produce molecular graphics for publications.

In 1992, David and Jane Richardson released a software package for the creation and the fruition of "*kinemages*" (short for "kinetic images") [80]. With this term the authors referred to "*a scientific illustration presented as an interactive computer display*" [80], i.e. a sort of an interactive three-dimensional illustration/presentation that can be played on personal computers. The software package consisted in two programs: "*PREKIN*" to create a kinemage from a PDB file, and "*MAGE*" to both view and authoring kinemages. The aim of the project was to provide molecular scientists a tool for creating interactive content "*to better communicate ideas that depend on three-dimensional information*" [80], e.g. by providing the kinemage file as supplement material accompanying a paper. In accordance to such aim, both the programs and some kinemages was released in a floppy-disk accompanying the 1992 paper on the first issue of the journal "*Protein Science*". The software was initially developed for Macintosh computers (a popular series of personal computer featuring the *Windows, Icons, Menus, Pointer* (WIMP) human-computer interface), a choice that has facilitated its adoption. On the other hand, due to the performance constraints and in order to ensure interactivity, the first version of *MAGE* was limited to wire-frame graphics.

Another program that made history in molecular graphics is *RasMol* (short for "*Raster Molecules*") [81, 82]. It was originally developed in 1989 by Roger Sayle (an undergraduate student at the time) as part of a final-year project. Initially it was focused on the rendering of large (for the time) molecules in space-filling representation, being able to display a fair amount of intersecting shaded spheres with specular highlights and shadows. At this end, *RasMol* implemented an hybrid scan line-based rendering algorithm. In particular, the rendering of each row of pixels of the final image (scan line) was computed by two stages: The first stage had the purpose to determine, for each pixel, which sphere was closest to the virtual camera (if any) and the depth of the spherical surface at that point. In the second stage shading and lighting model calculations was performed: in addition to the usual local shading model, a sort of ray-tracing algorithm was performed to determine, for each pixel of the scan line, if the corresponding point on the sphere was in shadow or not (i.e. if the line segment between the point and the light sources was intersected by other spheres). As mentioned before, the rendering algorithm implemented by *RasMol* was based on the one proposed by Porter [65], apart from the two stages approach and the ray-tracing procedure used to determine the shadows. Furthermore, thanks to its scan line-based nature, the *RasMol*'s algorithm was easily parallelizable: that was a key feature, since the program was originally developed for execution on distributed memory systems. The program was further developed during the 90s, by adding several features (such as additional graphical representations, see Figure 2.5) and the support for Unix workstations and Windows / Macintosh PCs, up to become one of the most widespread interactive molecular visualizers.

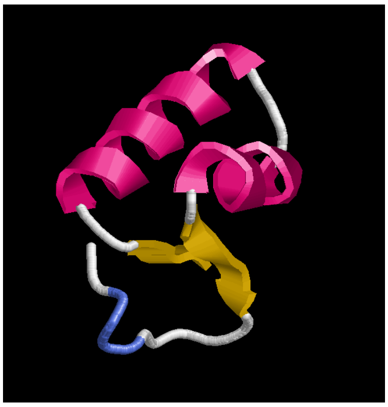


Figure 2.5: Ribbon diagram of a plant seed protein (PDB ID: 1CRN, [83]) rendered with *RasMol* 2.7.5

Nowadays many molecular graphics system are available; many of them are free to use (at least for personal or educational/research purposes) and provide the ability to extend their functionality through scripts or plug-ins. Among those oriented to biochemistry and structural biology, probably the most popular are VMD [71], UCSF Chimera [72], PyMOL [84] and Jmol [85]. Other quite popular software in these fields are Avogadro [86], YASARA [87] and CCP4mg [88]. Recently, a promising new molecular viewer and modeler called SAMSON (Software for Adaptive Modeling and Simulation Of Nanosystems) was developed by the NANO-D group at the French Institute for Research in Computer Science and Automation (INRIA)[2]. It has been developed using modern technologies (such as the Qt framework [89] and OpenGL 3.2) and it is accompanied by a Software Developed Kit (SDK) for the development of new modules. These modules, called “Elements”, can be shared with the SAMSON community via the “SAMSON Connect” website [90]. However, apart from basic stereo rendering without head-tracking, it has no documented support for advanced virtual reality visual display systems, such as CAVEs or Head Mounted Displays. As regard to quantum chemistry, some popular molecular graphics system are Molden *[91],* GaussView [92] and Molekel [93]; XCrySDen [94] and PLATON [95] focus instead on crystallography and material

[2]https://team.inria.fr/nano-d/

science.

Most of the cited software originally exploited the state-of-the-art computer graphics technologies available at the time of their conception, However, many of them are still bound to those technologies (that in some cases dates back to twenty years ago) or update themselves with non-negligible difficulties. That situation creates a disparity between the state-of-the-art research in molecular graphics, which rapidly exploit the advances in computer graphics, and the actual molecular graphics systems used day-by-day by researchers and students. One of the aims of Caffeine is therefore to try to reduce this gap.

2.2 Representation of molecular systems and state of the art rendering algorithms

As illustrated in section 2.1, in the last two centuries many conventions has been developed to graphically represent molecular structures and related properties. Different representations allows to depict the system under investigation with different levels of chemical detail and to highlight different peculiarities of the system. Furthermore it is common to combine more representations in the same image to communicate different types of information and properties about the molecular system.

In the following, a brief description about the commonly used and the recently proposed representations of molecular structures and related properties is provided, accompanied by a survey about the state-of-art algorithms for their rendering in real time.

2.2.1 Atomistic models

This category includes the representations of molecular structures depicting the position of each atom in the three-dimensional space. Atoms are usually represented as spheres having a radius proportional to the van der Waals (or the covalent) radius of the element they represent, while covalent bonds are usually represented by lines or cylinders. Other types of interactions between atoms (such as hydrogen bonds) can also be represented, using lines or similar visual elements. Widely used representations such as “wire-frame”, “licorice”, “ball-and-stick” and “CPK” belongs to this category. Atoms (spheres) can be drawn according to various color schemes, in order to represent their chemical element, the residue/fragment they belongs to or according to some atomic or molecular property, such as temperature factor, charge, hydrophobicity, solvent accessibility etc. (see Figure 2.6). Bonds can be colored in accordance to the related atoms or using a color map to visualize their intensity / order. It is also common to represent double and triple covalent bonds respectively with two and three cylinders connecting the same pair of atoms.

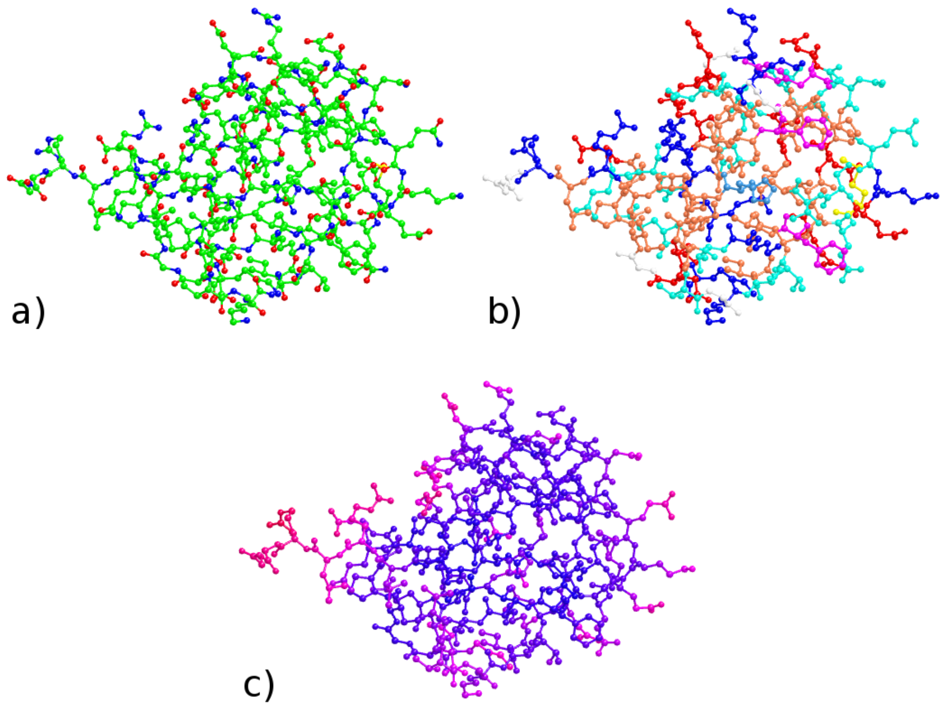


Figure 2.6: Ball-and-stick representation of ubiquitin (PDB ID: 1UBQ, [96]). Different color schemes have been employed in drawing atoms: (a) by element; (b) by residue; (c) by temperature factor. Images rendered with CCP4mg [88].

Although the common way in interactive computer graphics to describe a three-dimensional model is by means of a set of triangles approximating its external surface (since graphics hardware is heavily optimized to process triangles), this traditional approach does not suits well for atomistic models. The reason is that, to obtain a good approximation of curved surfaces, like spheres or cylinders, a significant number of triangles is required. When dealing with large molecular assemblies, this fact may compromise the frame rate and, in turn, the responsiveness of the application. On the other hand, approximating these surfaces with a low number of triangles saves the performance at the cost of a much lower visual quality. Although traditional techniques like multi-resolution tessellations (in order to reduce the number of polygons as a function of the distance from the virtual camera) or visibility-based culling (i.e. trying to detect the elements that will not be visible in the final image so to avoid drawing them, or at least process them as little as possible) can be employed to contain this problem (see [97, 98] as examples), a technique known as "*GPU-based ray casting of implicit surfaces*" has proven to be a better solution. First introduced by Gumhold [99] in 2003 to render a large number of ellipsoids representing tensor fields, this technique consists in feeding the GPU with a simple bounding geometry (e.g. a quad or a box) in place of each implicit surface to be draw and to analytically compute, in a *fragment shader*, the intersection between the view direction and the implicit surface for each fragment resulting from the rasterization of the bounding geometry. In 2004, Bajaj et al. presented TexMol [100], a molecular viewer exploiting a texture-based depth-corrected impostor rendering for both ball-and-stick and ribbon visualizations. The rendering technique employed by TexMol can be considered an early approximation of the actual GPU-based ray casting, since depths and normals are not calculated on a per-fragment basis, but stored in pre-computed textures which are simply fetched by the fragment shader. In the same year, Toledo and Levy generalized GPU-based ray casting to any quadric surface [101]; the method was then further improved by Sigg et al. in 2006 [102]. It should be noted that, although this technique can be applied to any implicit surface, it provides excellent results in the case of quadrics (such as spheres and cylinders), since the intersection test requires solving a simple second degree equation. For that reason, it has been successfully employed in numerous research works about the visualization of molecular structures (several of which are cited in the following), and implemented in several molecular viewers such as VMD [71] (for spheres only), UnityMol [103], MegaMol [104] and Caffeine. Further details about this algorithm, with particular reference to the implementation used in Caffeine, will be given in section 6.1.

Chavent et al. exploited the GPU-based ray casting of quadric surfaces to design a new atomistic representation called "*HyperBalls*" [105]. HyperBalls is a variant of the "ball-and-stick" where bonds are represented as hyperboloids instead of cylinders (see Figure 2.7). In fact, while cylinders are usually employed to depict a binary relation (presence or absence of a bond), the shape of the hyperboloids can be parameterized as a function of some physical/chemical property, such as the intensity of the bond. By being able to change the shape of each hyperboloid by tuning only a shrink factor, this representation results particularly suited to

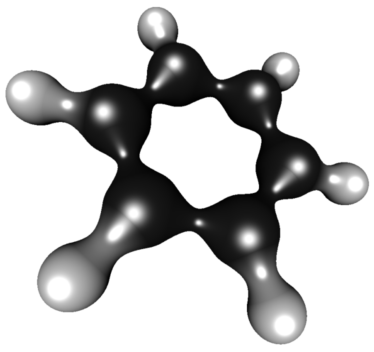


Figure 2.7: Benzene molecule drawn according to the "*HyperBalls*" representation [105]. Bonds are depicted as hyperboloids, whose thickness is a function of the distance between the bonded atoms.

represent dynamic phenomena, such as the the formation and decay of inter-atomic interactions.

Although traditional GPU-based ray casting of quadric surfaces allows to visualize static assembly formed by some millions of atoms at interactive frame rates on commodity hardware, and although the use of proxy geometries requires only a small amount of data to be transferred to the GPU for each quadric surface, special optimizations must be employed when dealing with large time-varying and/or massive systems.

An interesting optimization for visualizing dynamics molecular system has been proposed by Lampe et al. [106]. They focused on the visualization of Normal Modes Analysis (NMA) simulations [107] predicting motions of proteins, in which residues are approximated as points with a mass and connected by springs. The results of such simulations consist in roto-translations to be applied to the backbone elements of residues. It follows that the side chain and the atoms lying on the peptide planes are considered as rigid bodies joined by C_α atoms. Therefore it would be a waste of bandwidth to sent to the GPU a new position for each atom of the protein at each time-step of the simulation. On these premises, Lampe and co-workers designed an algorithm in which, for each residue and for each time-step, only the position of C_α atom, a set of angles and the type of the residue is sent to the GPU. The graphical primitives for each atom are then generated on the fly on the GPU, by exploiting geometry shader capabilities of graphics hardware. Finally, spheres are drawn via ray-casting on the fragment shader.

With aim to visualize time-dependent point-based data sets represented as GPU-based glyphs[3], Grottel et al. conducted an in-depth study to compare how rendering performance are affected by different strategies in constructing the bounding geometry, storing/encoding the parameters and uploading the data to the GPU [108]. Later, the same research team proposed a two-level (coarse- and fine-grained) occlusion culling algorithm to detect those glyphs that would not be visible in the final image and thus limit their processing as much as possible [109]. It is based on the “Hierarchical Z-Buffer Visibility” algorithm by Greene et al. [110]. As preprocessing step, the algorithm construct a regular grid for each frame of the simulation, storing in each cell a reference to the glyphs lying in that region of space. During the rendering, the algorithm keeps track of the cells that was visible in the last frame. When a new frame must be rendered, the algorithm creates a depth map storing the maximum depth of the glyphs lying in the cells that was visible in the previous frame (but using the position, rotation, scale and view parameters of the new frame). This is done by setting the depth map as only render target and by rendering those glyphs with a simplified ray-casting algorithm that only determine if a fragment belongs to the considered glyph and assign to it the maximum depth of the glyph. Then, for each cell of the grid, an occlusion query is requested to the GPU to determine if the bounding box of the considered cell has a depth greater than the ones stored at the corresponding region of the depth map computed in the previous step. If so, the glyphs lying in the cell would not be visible in the final image, so they are not transferred to the GPU. For those cells visible (at least in part) in the new frame, a fine-grained occlusion culling algorithm is applied. In particular, while waiting for the occlusion queries to be resolved by the GPU, an hierarchical occlusion map is constructed by “mip-mapping” the previously computed depth map. Then, when the glyphs lying in the visible cells are rendered, the vertex shader compares the image space bounding box of the glyph against the hierarchical occlusion map. If the glyph results to be occluded, the vertices of its proxy geometry are “culled” (moved outside the view frustum). This fine-grained occlusion culling does not prevent the geometry to be transferred to the GPU, but avoids unnecessary fragment processing. Thanks to this technique, Grottel and coworkers was able to visualize dynamics data sets consisting of up to one hundred millions of glyphs at interactive frame rates on commodity hardware. However, as the authors admit, this technique is effective only in the case of very dense data sets, such as those resulting from molecular dynamics simulations in the field of

[3]A glyph is a small visual object (such as a sphere, an arrow, an ellipsoid etc.) having a precise location in space and used to represent a data element. Its attributes (such as its orientation, color and size) are defined as a function of the data associated to the element it represents.

material science.

In 2012, Lindow et al. proposed a method to perform the rendering of biological structures composed by several billions of atoms at interactive frame rates [111]. When dealing with massive systems, the problem is not only to find a rendering method capable to produce images at a sufficient frame-rate, but also to find a strategy to store the data in the limited memory of the graphics card. To cope with this last problem, Lindow and co-workers exploit the fact that biological structures usually consist of a large number of recurring substructures. By considering these substructures as rigid bodies whose atoms have the same relative position within each instance, it is sufficient to store the information about a single substructure and a list of transformation matrices defining the position and the orientation of each instance. Authors chose to render substructures with a method which mixes the traditional GPU-based ray-casting of quadric surfaces with the GPU-based "*volume ray-marching*" (see section 2.2.4). In particular, for each substructure, information about atoms are stored in a regular grid, whose cells keeps track of the atoms lying in the corresponding regions of space. Grids are encoded in the form of three-dimensional textures, so to be easily passed to the GPU. For each instance of substructure to be drawn, a bounding box (with proper transformations) is feed to the GPU for rendering. In the fragment shader, "*volume ray-marching*" takes places: the cells of the grid traversed by the ray (starting from the camera and crossing the fragment) are sequentially examined along the ray direction, until an atom (sphere) crossed by the ray is found or until the ray exits from the bounding box. If such atom (sphere) is found, the intersection point and the related normal are computed and used to shade the fragment and to adjust its depth, otherwise the fragment is discarded. From the benchmarks presented in the original paper results that, although the traditional GPU-based ray-casting is faster for structures smaller than some million atoms, the proposed technique obtain good performance with larger structures, allowing to render assemblies composed by ~150 million atoms at interactive frame rates on the hardware of the time. The technique proposed by Lindow and co-workers (described so far) was later optimized and extended by Falk et al. [112], both to obtain a further gain in performance and to support the rendering of triangles (in addition to spheres), so to employ further representations (e.g. molecular surfaces).

Le Muzic et al. presented in 2014 a system capable of generate interactive computer-animated simulations of molecular reactions in biological networks for educational and dissemination purposes [113]. The authors exploited several techniques in order to visually represent systems containing up to some million of medium-large molecules. First of all, they extended the method described by Lampe et al. [106] for dynamically constructing (static) molecular structures on the GPU. In particular, their system send to the GPU a single vertex for each molecule to be visualized, delegating the generation of the proxy-geometry for ray casting each atom of the structure (visualized in CPK representation) to the tessellation and geometry shaders. The the positions of the atoms for each type of molecule are stored in texture buffers, so to be retrievable by the shaders. In this way, the authors was able to produce molecules containing up to ~260000 atoms for each vertex sent to the GPU. Of course, they exploited the fact that the simulated biological system contains only a limited number of different molecules, instanced many times and handled as rigid bodies. Further optimizations includes view-frustum culling of molecules by the vertex shader and a Level Of Detail (LOD) technique that reduce the number of generated spheres for each molecule as a function of its distance from the camera. When this happens, the radii of the remaining spheres is enlarged so to closely resemble the shape and volume of the original molecule.

2.2.2 Backbone models

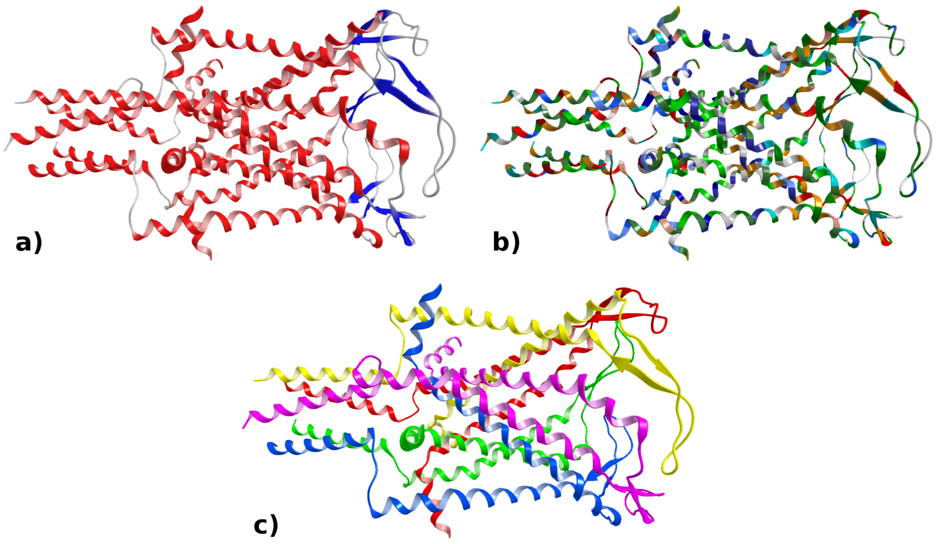


Figure 2.8: Ribbon diagrams of a prokaryotic Mechanosensitive Channel of Large Conductance (MscL) [114]. Three different color schemes are showed: a) by secondary structure; b) by residue; c) by chain. Images rendered with Caffeine.

Backbone models abstracts the structure of polymers so to highlight that spatial conformation of their main chain. The most diffuse backbone model is the so-called “ribbon” representation for proteins, made popular by Richardson in the 1980s [57]. In this representation the spatial conformation of polypeptide is approximated by a curve (usually a B-spline) whose control points are the C_α atoms of the amino acid residues. Secondary structures are represented with a ribbon, oriented according to the peptide planes (see Figure 2.8) : helices are visualized as a spiral, while β-strands as an arrow pointing towards the C-terminus of the chain. Random coils are drawn as tubes. Although in the simplest cases ribbons are colored according to the secondary structure they represent (Figure 2.8a), the residues they are formed by (Figure 2.8b), or the chain they belongs to (Figure 2.8c), additional information about the molecular system can be visualized via color mapping. An example is shown in Figure 2.9, depicting a Human topoisomerase I in ribbon representation, where the color and the thickness of the ribbon are functions of the structural disorder (β factor) [115].

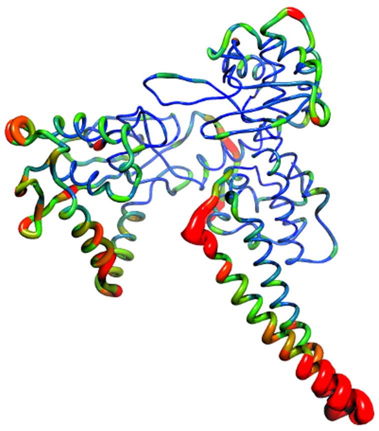


Figure 2.9: Image reproduced from [115]. Ribbon diagram of a Human topoisomerase I. Color (from blue to red) and thickness of the ribbon are proportional to increasing β factor.

While most of molecular graphics software use the same basic rules to draw ribbon diagrams for polypeptides, in the case of nucleic acids the situation is more variegated. In general, a ribbon is used to represent the spatial conformation of the sugar-phosphate backbone. The path of the ribbon is computed by evaluating the equation of a curve: common choices are cubic B-spline (Caffeine, Chimera [116] and VMD [71]) or Catmull-Rom spline (VMD). There are also various possibilities in the choice of the control points for the curve and in the way the orientation of the ribbon is computed. For example, VMD allows to chose between phosphorus and *C3'* atoms as control points, and takes into account the position of some of the phosphate oxygens to compute the normal vectors to the ribbon (i.e. its orientation). Chimera, instead, use *C5'* atoms as control points and computes the orientation as a function of the position of the *C1'* atoms. Finally, Caffeine uses phosphorus atoms as control points and the versor from *C1'* to *C3'* as normal vector for the ribbon. Nucleobase and (deoxy)ribose are represented using filled pentagons and hexagons. However simpler representations are possible, in which the sugar is omitted (replaced by a line connecting the ribbon to the base) and/or in which nucleobase are visualized as boxes, ellipsoids or elliptical tubes [116]. Figure 2.10 shows the ribbon diagram of a fragment of B-form DNA.

Further details about the geometrical construction of ribbon diagrams in Caffeine are described in section 6.3.

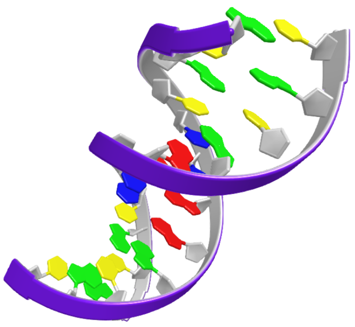


Figure 2.10: Ribbon diagram of a fragment of B-form DNA rendered by Caffeine.

Several works proposed GPU-accelerated algorithms for drawing ribbons diagrams. Among the newest, Bagur et al. proposed a method to draw tubes (coils) and spirals (helices) as quadric surfaces to be ray-cast on the GPU [117]. Although their method is promising for random coils, it is able to produce only flat helices. Krone et al. [118] and Wahle et al. [119] proposed algorithms for the geometrical construction of tessellated ribbons on the GPU. The aim of both methods is to relieve the CPU by most of the computations required to tessellate the ribbons and, at the same time, to minimize the amount of data to be transferred to the GPU. It is important to note that these approaches are convenient only for the visualization of time-varying systems. In fact, when used for static structures, they introduce a non-negligible overhead to the rendering procedure, resulting in a consistent drop of the frame rate with respect to the rendering of pre-computed static geometries. In the case of time-varying systems, instead, both the computation of the tessellation on the CPU and the transfer of the resulting graphics geometry to the GPU may represent a bottleneck, that can be solved by exploiting GPU-acceleration.

2.2.3 Surface models

While atomistic and backbone models provides valuable information about the structure of the molecule, they offer little knowledge (with the exception of the CPK model) about the region of space occupied by the molecule or about the areas of the molecule exposed to solvent or other molecules. Surface models was developed to provide these types of indications. Furthermore, it is common to visualize additional molecular properties over the surface (such as hydrophobicity, atomic charge, electrostatic potential etc.), via color or texture mapping (see also section 2.2.4). Molecular surfaces are widely exploited in biochemical research and structure-based drug design, for example in order to identify cavities or channels accessible by the solvent or suitable to become the binding site of a ligand.

2.2.3.1 Molecular surfaces and their definition

The most diffused surface models are the "*van der Waals Surface*" (vdWS, in the following), the "*Solvent Accessible Surface*" (SAS) and the *Solvent Excluded Surface* (SES). They are usually defined according to the "*hard-sphere*" model illustrated in Figure 2.11, where a "*probe*" sphere approximating a solvent molecule is rolled over the vdWS.

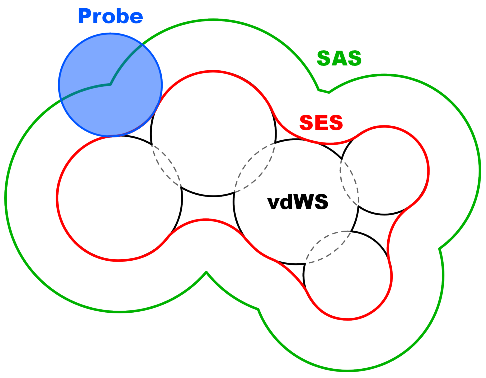


Figure 2.11: Definition of the *van der Waals Surface* (vdWS, in black), *Solvent Excluded Surface* (SES, in red) and *Solvent Accessible Surface* (SAS, in green) according to the "*hard-sphere*" model. SAS and SES are constructed by rolling a spherical probe (representing the solvent) over the van der Waals surface.

The "*van der Waals Surface*" is the simpler surface model and provides an approximation of the volume occupied by the molecule. It is obtained by the union of the spheres representing the atoms of the molecule and having a radius proportional to related the van der Waals radius (see Figure 2.11). In other words, the vdWS is the external surface obtained when representing a molecule according to the CPK model.

In 1971, Lee and Richards proposed another type of molecular surface, today known as "*Solvent Accessible Surface*" [60]. This model is useful to identify which atoms of the considered molecule are accessible by the solvent. The SAS can be defined by rolling a probe sphere over the vdWS of the molecule under investigation. During this process, the probe must be kept in touch with the vdWS without penetrating it. The SAS is the surface traced by the *center* of the probe after rolling over the entire vdWS. The atoms involved in the construction of the SAS (e.g. those touched by the probe) can probably be accessible by solvent molecules equals or smaller then the chosen probe. Another way to obtain the SAS is expanding the radius of the spheres that form the vdWS by the radius of the probe (usually set to 1.4Å).

As in the case of the SAS, also the "*Solvent Excluded Surface*" (also known as "*Connolly surface*" or simply "*molecular surface*") can be defined by rolling a spherical probe over the vdWS. However, instead of taking into account the center of the probe, the SES is defined as the surface traced by the exterior of the probe while rolling overt the vdWS. The SES combine the advantages of both the vdW and solvent accessible surfaces, by approximating the volume occupied by the molecule while providing information about the areas accessible by the solvent. A comparison between the vdW and the solvent excluded surfaces is shown in Figure 2.12.

The SES was first proposed by Richards in 1977 [67]. Actually, the name was coined one year later by Greer and Bush, who also proposed an alternative (equivalent) definition for it and presented the first computer program capable of generating approximated images of the SES [66]. The first analytical method for computing the SES was presented by Connolly in 1983 [68]. From his formulation results that the SES is formed by three types of curved surface patches, joined by circular arcs:

- *Convex spherical patches*: occurring when the probe is in contact with a single atom. These patches are the regions of the vdWS in direct contact with the solvent probe.
- *Toroidal ("saddle") patches*: generated when the probe is in touch with two

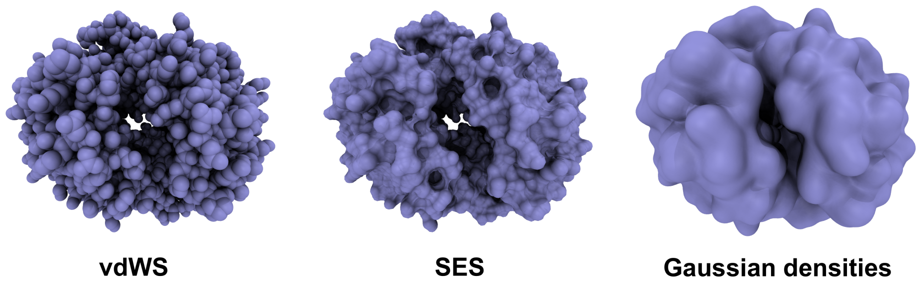


Figure 2.12: Comparison between three different types of molecular surface for a human deoxy-hemoglobin (PDB ID: 2HHB, [120]). Left: *van der Waals Surface* (vdWS). Center: *Solvent Excluded Surface* (SES). Right: *Gaussian density surface* generated according to the algorithm by Krone et al. [121] (named "*QuickSurf*" in VMD). Images created with VMD [71] and the built-in Tachyon Ray Tracing System.

atoms.

- *Concave spherical patches*: formed when the probe is in contact with three or more atoms.

Connolly's paper also presents an algorithm to determine these patches and their parameters.

In the following years, several others algorithms have been developed to compute the patches constituting the SES in a more fast and/or reliable way. Particularly relevant is the method proposed by Sanner et al. [70], which exploit an intermediate geometric description of the molecular surface called "*Reduced Surface*" (RS). The RS is defined by a set of vertices, a set of edges and a set of faces. Recalling the concept of the solvent probe rolling over the atoms of the molecule, a vertex of the RS is generated in correspondence of the center of each atom touched by the probe, an edge of the RS connecting two vertices is generated whenever the probe can roll over a pair of atoms, and a triangular face (delimited by three RS-edges) is created when the probe gets in touch with three atoms. Both the analytical description of the SAS and of the SES can be computed from the RS. In a following work [122], Sanner and co-workers describes how the RS can be updated in in consequence of the movement of a subset of atoms. A similar approach has been proposed by Ryu et al. [123], which exploit a geometric construct called $\beta - shape$ [124] (a generalization of the $\alpha - shape$ [125] which allows to employ spheres in addition to points) as intermediary representation of the molecular surface from which the analytical description of the SES can be derived. Interestingly, the resulting $\beta - shape$ coincide with the Reduced Surface, although the two constructs are built according to different criteria.

Another relevant result is the "*contour-buildup*" algorithm by Totrov and Abagyan [73], which determines a sequence of arcs representing the borders ("*contour*s") of the "*contact surface*" (the regions of the vdWS that the solvent probe can touch without colliding with other atoms) and then derives from them the analytical description of the SES. The key idea of the contour-buildup algorithm is that the spherical patches of the contact surface and those constituting the SAS have the same shape and only differ in radius. Therefore, the algorithm computes the contours of the patches of the SAS and scale them down to obtain the contours of the contact surface. Since the contour of each accessible atom does not depends on the result of the computation for the other atoms, the algorithm is easily parallelizable.

Besides the definitions used so far, it is also possible to define the molecular surfaces in terms of implicit surfaces. Examples of definition of the vdWS (but applicable also to the SAS) as the isosurface of a scalar field can be found in [126, 127], where the implicit function is expressed as the *union* (maximum) of Gaussian density distribution centered at atoms positions and parameterized according to the vdW radii. In their work, Giard and Macq [127] also show how to derive the SES from such density field: the idea is to use the Fourier transform in order to express the density field of the vdWS in the frequency domain and then filter it by an ideal low-pass filter, so to remove the frequencies corresponding to elements smaller than a solvent molecule. Another analytical definition of the SES in terms of implicit surfaces was recently elaborated by Parulek and Viola [128]. They defined a set of implicit functions representing the convex, concave and toroidal patches of the molecular surface. Later, Parulek and Brambilla [129] exploited implicit surfaces to define a new molecular surface having a shape that approximate the SES, but that can be computed more efficiently.

Another interesting class of molecular surfaces are the ones resulting from isosurfacing a density field given by the *summation* of Gaussian distribution functions centered at atoms positions. This approach was originally proposed by Blinn in 1982 [130] and was designed to approximate the model of the electron density maps of molecular structures. An example of "*Gaussian density surfaces*", also known as "*Metaballs*" or "*Blobby surfaces*" is shown in Figure 2.12. The main advantage of these kind of surfaces is that their computation significantly less expensive of the SES. On the other hand, SES provide an accessibility model with respect to the solvent which is much more accurate, although Gaussian density surfaces can be parameterized so to closely approximate the volume and area of a collection of intersecting spheres [131] (such as the vdWS or the SAS).

Several others types of molecular surfaces has been proposed, such as the "*Molecular Skin Surface*" [132] (presenting interesting mathematical properties), the "*Minimal Molecular Surface*" [133] (obtained by a process of free energy minimization simulating the effects of immersing a polar molecule in a polar solvent) or the "*Ligand Excluded Surface*" [134] (which provides an accurate accessibility model with respect to a specific ligand). However, their use is still limited and/or their computation is expensive (e.g. see [135] for a comparison between the computational costs required to construct and render the SES and the MSS). For these reasons, they will not be discussed any further.

2.2.3.2 Rendering of molecular surfaces

The traditional method of rendering molecular surfaces consist in tessellate them as triangle meshes, since graphics hardware is designed (and heavily optimized) to render triangles. Due to the popularity of the SES, most of the methods presented in literature focus on its triangulation, presenting new algorithms to reduce the (still relevant) cost of its construction and/or to handle/avoid the occurrence of self-intersecting parts ("*singularities*"). However, many of them are also able to construct a triangulation of the vdWS and of the SAS. These algorithms can be classified in two families. The first computes the triangulation starting from the analytical description of the surface. Notable examples are the original triangulation algorithm by Connolly [69], the one integrated in the *MSMS* software package by Sanner et al. [70], the fast triangulation based on the $\beta - shape$ of molecules designed by Ryu et al. [136], or the recent work by Zhang et al. [137]. The second family of algorithms are the "*grid-based*" methods, which construct a three-dimensional grid containing the values of some scalar field at discrete locations of space around the molecule, and then compute the triangulation of an isosurface by means of one of the many isosurface extraction algorithms, such as the popular "*Marching Cube*s" [138] or one of its variants [139]. This category includes the already cited works by Laskowski [126] and Giard et al. [127] (which employ a scalar field resulting from the union of Gaussian functions), the method described by Chan et al. [140], as well as the algorithms by Can et al. [141] and Yu [142] which exploit respectively level-set methods and a list-based representation of grid regions in order to speed up the computation of the sampled scalar field. Recently, Decherchi and Rocchia [143] presented an interesting hybrid approach, where the values to be stored on the 3D grid are computed by a ray-casting procedure, which tests the intersection between a set of rays starting at regularly spaced position and the analytical description of the surface (the equations describing set of patches forming the surface). The surface is then tessellated by applying the *Marching Cubes* algorithm on the resulting grid. Finally, it should be noted that most grid-based method can also be employed to detect surface grooves and internal cavities.

An alternative approach for rendering molecular surfaces is the one based on the "*GPU-based ray casting*" method. Besides producing images with a better visual quality with respect to tessellated surfaces, this technique also provides the best performance when rendering the vdWS and the SAS (as opaque surfaces), as already discussed in section 2.2.1. GPU-based ray casting of the SES is relatively efficient, since its constituting patches (identified by Connolly) are portions of algebraic surfaces of the second order (spheres) and fourth order (tori). Ray casting of other types of implicit surfaces on the GPU (like those defined by distance or Gaussian functions) is also possible [128, 129, 144–146], although computationally more expensive.

From the point of view of raw rendering performance, drawing a triangle mesh representing a molecular surface is usually faster then ray-casting implicit surfaces. However, this second approach provides a superior image quality, does not need a triangulation procedure and requires a much smaller amount of data to be sent to the GPU. While for static structures the time spent in triangulating the surface and send the resulting data to the GPU does not constitute a problem (since those procedures must be performed one time only), it usually represent a relevant bottleneck in the case of time-varying systems, as discussed int the following subsection. On the other hand, it is easier to correctly draw semi-transparent and/or open surfaces (e.g. because the camera lies inside the surface or a clipping plane is applied) with triangle meshes. In fact, common implementations of ray-casted molecular surfaces draw "spurious" (portions of) patches lying within the surface. These are not visible when the surface is opaque, but produce erroneous images when it is open or transparent. A simple solution to the transparency problem consists in drawing only the frontmost portion of the surface on a hidden frame buffer and then blending it to the main frame buffer, but it only produces approximated results. Kauker et al. [147] recently discussed how to solve this problem by employing arrays of fragments to obtain both a correct transparency effect and the removal of spurious inner fragments. Further details on the problems arising when trying to simulate transparent surfaces in real-time computer graphics and related solutions are discussed in section 6.5.

2.2.3.3 Visualization of time-varying molecular surfaces

To recap, the traditional process to visualize a molecular surface consists in the following steps:

1. Computation of the analytical description of the surface (e.g. the set of the constituting patches and their parameters).
2. Triangulation of the surface.
3. Transfer of the data to the GPU.
4. Rendering.

When dealing with time-varying structures, a new updated molecular surface must be computed and rendered at very short time intervals. In order to do so, the computations of the previous steps must be parallelized or, when possible, avoided. In the case of short trajectories some of these steps may also be pre-computed and their results stored in memory for later usage. This is not, however, a general solution.

GPU-based ray casting of algebraic surfaces is particularly suited for visualizing dynamic surfaces, since it does not requires triangulation procedures, greatly reduce the data to be sent to the GPU (with respect to triangle meshes) and offer good rendering performance. For these reasons, it is employed by several of state-of-the-art techniques that will be briefly discussed in the following.

In 2009 Krone et al. exploited GPU-based ray casting to visualize SES dynamics [148]. The Sanner's Reduced Surface (RS) [70, 122] was chosen to analytically compute the SES. However this computation was performed on the CPU. The authors tried to speed up this procedure by exploiting the fact that the RS can be partially updated [122] when a small subset of atoms (at most 100) changes their position. In order to apply these partial updates to MD simulations, they tried to impose a filter on the atomic movements: positional changes lower than a threshold chosen by the user was ignored. If the number of moving atoms exceeded the maximum allowed for a partial update, the whole RS was re-computed. On the hardware of the time, Krone and co-workers was able to visualize dynamics systems containing ~4000 atoms at a frame rate equals or lower than 20 fps. In a later work, Krone et al. [149] proposed a parallel algorithm for the RS computation that can be executed on the GPU. However, the algorithm did not support partial updates, leading to lower performance with respect to the version with partial updates computed on the CPU (5 fps for molecular systems composed by ~2500 atoms).

Lindow et al. preferred the "*contour-buildup*" algorithm for the computation of the patches of the SES [135], since it is easily parallelizable. In their work, SES computation was performed on the CPU in a parallel fashion and with the support for partial updates, while the resulting algebraic surfaces was ray-casted on the GPU. The authors compared their parallel "*contour-buildup*" algorithm with the RS algorithm employed in [146]. For a system composed by ~59000 atoms, the contour-buildup algorithm built the SES in 0.4s, while the sequential RS employed 2.6s. In another test involving a dynamic system composed by ~4500 atoms with 500 moving atoms, the described system was able to compute (with partial updates) and draw the SES 33 times per second. According to the authors, partial updates provided a speedup of 2.5 with respect to the entire re-computation of the SES. One year later, Krone et al. presented a new parallel variant of the "*contour-buildup*" algorithm without partial updates, in which the computations was divided in fine-grained tasks [150] suited to be executed on graphics hardware. By implementing the proposed algorithm by means of the CUDA [151] GPGPU technology, they was able to compute and visualize the entire SES of a molecule of ~10000 atoms 20 times per second.

Previous discussed research works shows how SES computation still represent a bottleneck when visualizing dynamics systems. For this reason, in a recent work Krone et al. [121] focussed on the visualization of *Gaussian density surfaces* (see Figure 2.12 right) that, if proper parametrized, can approximate electron density and solvent accessible surfaces. Their algorithm consists in two parts, both implemented in CUDA [151] and thus running on the GPU: the first step computes a volumetric density map as the sum of Gaussian functions centered at atoms positions, while the seconds applies the "*Marching Cubes*" algorithm [138, 139] to triangulate an isosurface from the resulting three-dimensional grid. The authors compared the performance with [150], showing that for a protein of ~59000 atoms the Gaussian density surface can be entirely computed and rendered 19 times per second (with a grid spacing of 1Å), against the 7 times per second obtained by the previous approach for visualizing the SES (on comparable hardware). This algorithm has been integrated in VMD and it is available as "*QuickSurf*" drawing method.

Finally, in an previously cited work of 2013 [129], Parulek and Brambilla defined a new implicit function whose isosurface resemble the SES and that can be efficiently evaluated. The author chose to determine the isosurface on a per-pixel basis by means of a ray-casting procedure implemented in CUDA [151]. The obtained performance are lower to the ones obtained by Krone et al. [121] (9fps for a molecule of ~34500 atoms), although their molecular surface provides a better approximation of the SES and the ray-casting strategy produce pixel-perfect images. However, better performance could be obtained by sampling their implicit function on a 3D grid and triangulating the surface with the *Marching Cubes* (as done in [121]).

2.2.4 Visualization of volumetric molecular properties

Many molecular properties are scalar, vector or tensor fields. In numerical simulations, the value of these properties is computed only for discrete locations of the three-dimensional space, thus producing a *volumetric dataset*. Formally, a volumetric dataset is a set of pairs $< P_i, V_i >$ called "*voxels*" (short for "*volume elements*"), where P_i is a point in space and V_i is its associated value. Although the location of the voxels can be chosen arbitrarily, it is common to chose equally spaced locations, so to obtain a regular three-dimensional grid of values. The value for an arbitrary point lying within a cell of the grid can then be approximated by interpolating the values of the eight voxels delimiting the cell.

Volumetric molecular properties are usually visualized using the representations and techniques developed in the field of Scientific Visualization for scalar, vector and tensor fields. In the following, the discussion will focus on the visualization of *scalar* molecular properties (such as electron density, electrostatic potential, molecular orbitals, solvent density etc.), since they are the properties of most common use. Interested readers can refer to [18, 152] for an in-depth discussion on the visualization of vector and tensor fields, as well as to get an overview of the state-of-the-art on these topics.

Scalar volume data are usually visualized according to the following representations: *isosurfaces*, *slicing* and *direct volume rendering*.

2.2.4.1 Isosurfaces

Given a field and a value in it (called *isovalue*), the resulting *isosurface* is the surface formed by the points having the specified value in the considered field. The main limitation of isosurfaces is that they can only provide limited information about a field: in order to show how the considered quantity varies in space, multiple isosurfaces are required. However, it often happens that these isosurfaces are closed and nested one into the other, so the use of transparency or clipping planes is required (see Figure 2.13). In the case of interactive applications or movies, another possibility is to change the isovalue over time, as an animation.

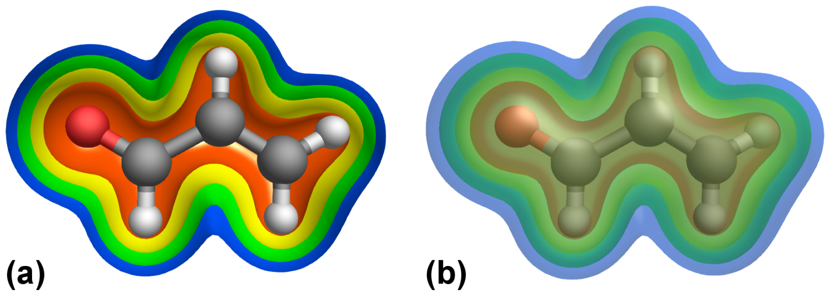
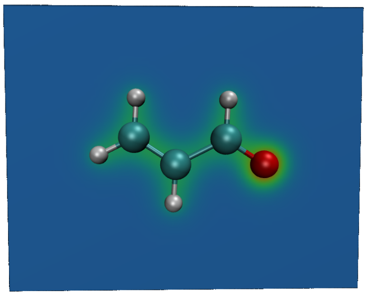


Figure 2.13: Total electron density of an Acrolein molecule represented by multiple nested isosurfaces. a) A clipping plane has been applied to the isosurfaces in order to reveal the internal structure. b) The isosurfaces has been drawn as semi-transparent surfaces. Images generated with Caffeine.

The traditional method to render isosurfaces is by constructing a triangle mesh approximating the surface by means of one of the many algorithms proposed in literature [152, 153], such as the popular "*Marching Cubes*" [138] or one of its improved variants [139]. Alternative approaches exists to render an isosurface without tessellating it. These techniques are known as "*non-polygonal isosurfaces*" [154]. Although rendering a triangle mesh representing an isosurface is usually faster than non-polygonal methods, the latter are better suited in situations in which the isovalue is changed frequently (e.g. by animating it over time), since they do not require to re-compute a polygonal representation of the surface. However several parallel and GPU-accelerated versions of the *Marching Cubes* (or similar algorithms) have been developed over the years to cope with these situations (e.g. [155, 156]). Finally, the non-polygonal approach can be a convenient solution if the visualization system already implement direct volume rendering (see section 2.2.4.3), since the rendering of isosurfaces can be considered as a special case of direct volume rendering.

2.2.4.2 Slicing

A popular way to visualize volume data is by *slicing* the volume by a surface. In this representation, a surface is introduced in the region of space bounded by the three-dimensional grid and every point of the surface is colored as a function of the value sampled from the data set at that point. Common choices for the surface with which to slice the volume include planes (Figure 2.14a), isosurfaces (Figure 2.14b) or molecular surfaces.


(a) Planar slice of the total electron density of an Acrolein molecule.

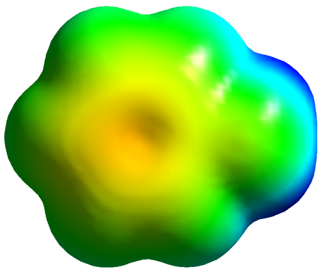

(b) Aniline electron density isosurface colored as a function of the corresponding electrostatic potential. Courtesy of Gianluca Del Frate.

Figure 2.14: Examples of volumetric molecular properties visualized by means of slicing surfaces. Images generated with VMD [71].

This representation is usually implemented by encoding the volume data in a *3D texture*. Furthermore, the surface must be drawn with a proper *fragment shader* that, given the relative position of the fragment within the volume (from *texture coordinates*), sample the volume value from the 3D texture and compute the corresponding value for the fragment according to a color map (usually provided as an additional texture).

2.2.4.3 Direct Volume Rendering

While "*Indirect Volume Rendering*" methods visualize the information contained in the volume as polygonal surfaces (belong of this category the algorithms which construct triangle meshes representing isosurfaces), "*Direct Volume Rendering*" (DVR in the following) methods consider the volume as a sort of "gaseous" light-emitting medium which can scatter or absorb the emitted light according to the data values and the chosen optical model [157]. The function that maps data values to optical properties (usually a color and an opacity factor) is known as "*transfer function*". During rendering, the color of each pixel of the final image is computed by accumulating the effects of the optical properties of the volume region corresponding to (i.e. projected on) the considered pixel .

It is important to note that choosing a proper transfer function is critical in order to produce meaningful images. Furthermore, the quality of a transfer function is strictly related to the nature of the data to be visualized. Therefore, the knowledge and the insight of human experts is often essential to define meaningful transfer functions.

DVR is widely used in Scientific Visualization, especially for medical and engineering data. However, its use in molecular graphics is not very common. An example of DVR employed in the visualization of electron densities is shown in Figure 2.15.

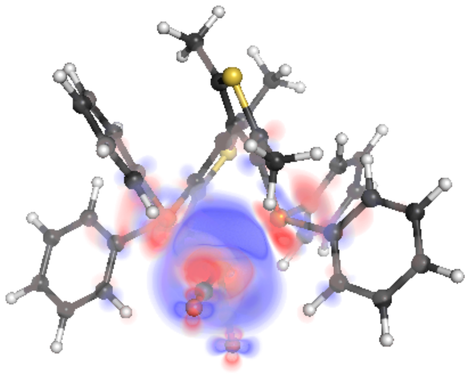


Figure 2.15: Electron density rearrangement after metal-ligand bond formation in a nickel dicarbonyl complex with a chelating diphsofine ligand, visualized via direct volume rendering. Red regions: electron depletion, blue regions: electron accumulation. Courtesy of Dr. Sergio Rampino. Image generated with PyMOL [84].

In the last twenty years, many efforts has been devoted to design and develop fast and visually accurate algorithms for DVR. A complete discussion of these techniques is beyond the aim of this thesis. For this reason, in the following only the basic concepts of a couple of GPU-based methods will be briefly described. Interested readers can refer to [152, 154] for an in-depth discussion on the subject.

The most common GPU-based methods for DVR are the "*texture-based*" and the "*ray-marching*" (also known as "*ray-casting*") methods. They will described under the following assumptions:

- An "*Emission-Absorption*" optical model is used, in which the volume consists in particles that both emit and absorb light [157]. This is the most common optical model in DVR, since an approximate solution for the related rendering integral can be efficiently evaluated using "alpha-blending" [154].
- The volume data is encoded in a 3D texture, in order to be accessible by the

shaders and to take advantage of the hardware-accelerated trilinear interpolation when sampling the texture.

Texture-based volume rendering This method is similar to the slicing plane technique discussed before: the volume is now sliced with multiple planes, and the resulting *semi-transparent* colored slices are "alpha-blended" into the frame buffer. This technique was first proposed by Cullip and Neumann in 1993 [158]. Going into detail, the region of space bounded by the three-dimensional grid is sliced by means of a stack of parallel, view-aligned and equidistant planar polygons, with proper texture coordinates for each vertex (expressing the relative position of the vertex within the volume). For a correct alpha-blending, the polygons must be drawn in back-to-front order[4] (or front-to-back order using an alternative compositing equation). During fragment processing, the texture coordinates of the considered fragment are used to sample the 3D texture storing the volume data, thus obtaining the related data value. The transfer function (that can be "hard-coded" or provided as an additional texture) is then evaluated, thus obtaining the color and the opacity factor for the considered fragment. Color and opacity will then be combined with the previous content of the color buffer according to the alpha-blending equation.

Volume rendering via ray-marching Similarly to the "*ray-casting of implicit surfaces*" discussed in the previous sections, the idea at the base of the DVR via "*ray-marching*" (also called "*ray-casting*" in literature) is to cast an imaginary ray for each pixel of the final image, starting from the virtual camera and crossing the considered pixel. If the ray intersect the volume, the volume data is sampled at multiple locations along the ray, usually "marching" (i.e. advancing along the ray to the next sampling point) in front-to-back order. The transfer function is evaluated for each sampled value, thus obtaining a corresponding color and opacity factor. The final color for the considered pixel is given by compositing all the colors computed during the "march", as a function of their alpha value and sampling order. In the case of front-to-back "marching", the following compositing equation is applied:

$$\begin{cases} C'_n = & C'_{n-1} + \left(1 - \alpha'_{n-1}\right) \cdot \left(\alpha_n \cdot C_n\right) \\ \alpha'_n = & \alpha'_{n-1} + \left(1 - \alpha'_{n-1}\right) \cdot \alpha_n \end{cases} \tag{2.1}$$

[4]The reason is explained in section 6.5.

where C'_n is the color resulting from the composition of the first n sampled points, α'_n is the opacity factor resulting from the composition of the first n sampled points (it is required for computing C'_{n+1}), and (C_n, α_n) are respectively the color and the opacity of the *nth* sampled point. The base case of the recursion is the chosen background color: $C'_0 = C_{background}$ and $\alpha'{}_0 = 1$.

Although CPU-based implementations of *ray-marching* have been used since 1980's, the first GPU-based implementations was proposed in 2003 [159, 160], when graphics hardware with a sufficient fragment shader programmability become available. These first implementations required a multi-pass rendering procedure, due to the lack of support for branching and looping in fragment shaders. A first example of a single-pass GPU-based ray-marching algorithm was included in the samples of the NVIDIA SDK in 2004 [161], while another was described one year later by Stegmaier et al. [162]. In the following a simple single-pass ray-marching algorithm, based on the work of Stegmaier et al. [162], will be briefly described.

As in the case of *ray-casting of implicit surfaces*, a proxy geometry must be constructed and fed into the rendering pipeline, so to generate the fragments upon which the *ray-marching* algorithm will be performed. In this case, the proxy geometry is composed by a bounding box (12 triangles) for the volume, to be draw with "back-face culling" enabled. The fragments resulting from the rasterization of its "front" faces will be then processed by the fragment shader implementing the *ray-marching* algorithm. First of all, the shader computes the direction of the ray (given by the normalized difference between the positions of the fragment and of the virtual camera) and its starting point (given by the intersection between the bounding box and the ray). Then a loop is executed (*"marching procedure"*) in which, at each iteration, the following operation are performed:

- The value of the volume data is sampled for the current position along the ray.
- The transfer function is evaluated for the retrieved value, returning a color and an opacity value.
- The new color is combined with the color accumulated so far according to the equation 2.1.
- If the accumulated opacity exceeds a predefined threshold close to 1, then the contribution of the remaining part of the volume along the ray will be almost invisible in the final image, so the loop can terminate ("*early ray termination*").

- The coordinates of the sampling point for the next iteration is computed by advancing of a predefined distance along the ray.
- If the new point lies outside the bounding box, the loop terminates.

The color accumulated during the marching procedure is finally written in the color buffer at the location associated to the considered fragment.

The *ray-marching algorithm* can also be used for the visualizing isosurfaces without the need to construct a polygonal representation for them ("*non-polygonal isosurfaces*"). In fact, the presence of the isosurface can be detected by looking for a sign change between the difference of the isovalue with the current sampled value and the difference of the isovalue with the previous sampled value [162]. The position of the isosurface within the detected interval can then be computed by a linear interpolation or using an iterative bisection procedure [163]. The normal to the isosurface at the considered point can be computed on the fly during fragment processing by calculating the gradient vector using central differences method. The normal vector can then be used to color the surface according to the common *Blinn-Phong shading model* [164] (or a more sophisticated one), in order to visually enhance its shape. A state-of-the-art method for the visualization of isosurfaces via *ray-marching*, featuring several optimizations and advanced shading, is described in [163].

Noteworthy is also the work by Scharsach et al. [165], presenting a virtual endoscopy systems based on *ray-marching* DVR. In the paper, the authors propose a visualization combining a semi-transparent shaded isosurface (representing the external skin) with an unshaded direct volume rendering of the tissues behind it. The aim is to highlight the external shape of the volume by means of the shaded isosurface, while providing additional visual information about the internal tissues. The work also describe how to combine in the same image DVR with rendering of traditional polygonal geometry (e.g. to visualize parts of the endoscopic device or pointers) as well as to support clipping planes for the volume (including the *near plane*, so to virtually fly through it) [5].

Direct Volume Rendering in Molecular Graphics The use of DVR is quite uncommon in Molecular Graphics. Among the research works that exploit DVR to gain insight on data sets resulting from numerical simulations, worth mentioning

[5] The previously described *ray-marching* algorithm must be properly modified in order to produce correct results in these situations

the paper by Mehta et al. [166], in which authors describes the use of isosurfaces and DVR to detect defects in silicon lattices resulting from Molecular Dynamics simulations. Mehta et al. described a method to automatically detect salient isovalues and to derive meaningful transfer functions from them. Qiao et al. [167] described the realization of "*VolQD*", a system for the interactive DVR of multivariate wave functions in semiconducting quantum dot simulations. Noteworthy is the work by Jang and Varetto [168], who designed a DVR system for the visualization of atomic and molecular orbitals. One of the peculiarity of their system is that it does not simply render a precomputed volumetric dataset. Instead, a per-fragment evaluation of the functional representation of atomic and molecular orbitals is performed on the fly on the GPU. Two types of basis functions are available for this purpose: *pure Gaussian* and *Gaussian Type Orbital*. The coefficients of the functional representation of the orbital are read from the output of quantum chemistry packages such as *Gaussian* and *GAMESS*. Volume rendering is performed by means of a variant of the texture-based method: a stack of polygonal slices intersecting the bounding box is generated in the vertex shader, then, for each fragment resulting from their rasterization, the value of the atomic or molecular orbital is evaluated on the base of the chosen basis function and of the pre-computed orbital coefficients. The transfer function is then evaluated on the resulting value, so to obtain a color and an opacity factor for the fragment. Slices are drawn in front-to-back order and "alpha-blended" in the color buffer, as in the traditional texture-based volume rendering. The authors have also experimented with several transfer functions, including an illustrative rendering technique that produce very clear images of multiple nested isovalue structures. DVR has also been employed to visualize probability density functions representing positional uncertainty of atoms [169] or meta-stable molecular conformations [170]. Finally, Knoll et al. employed DVR to graphically represent uncertain molecular interfaces defined on the base of the electron density distribution of the molecule [171].

2.2.5 Enhancing depth perception of molecular structures

Human visual system exploits multiple cues to perceive the three-dimensionality of space, giving us the ability to estimate sizes and distances of objects of the real world. When exploitation of *stereopsis* (i.e. the ability to infer depth information from two slightly different images perceived by the eyes) is not possible, human brain tries to deduce information about sizes and distances by focusing on the so called "*monocular depth clues*", such as those deriving from the way in which light is reflected or occluded by objects. In absence of stereoscopic displays, is therefore important to reproduce those cues in order to allow the user to better understand the three-dimensional structure of complex molecules.

The illumination models usually employed in interactive computer graphics (such as the standard "*Blinn-Phong*" model [164]) are said "*local*" illumination models, because the color of a surface is computed by keeping into account only the properties of the considered surface and of the light sources, without considering the other objects of the three-dimensional scene. Although reflections of directly incident light are properly simulated, shadows are not produced and indirect lighting is approximated only by means of a constant value, called "*ambient lightning*" (to be summed to the color resulting from reflection calculations). In the years, many techniques have been developed in order to simulate a more realistic illumination exploiting the standard hardware-accelerated rendering pipeline designed for local illumination. The following discussion will focus on the techniques to simulate shadows, since they are one of the main depth cues. As we will see, most of the research works to simulate shadows in molecular graphics focus on atomistic representations of large molecular assemblies. In fact, with these representations, the resulting image consist in a large cluster of small spheres, whose structure is hard to deduce (in absence of stereoscopy and shadows) due to the high variability of surface normals between neighboring pixels (see Figure 2.16 left). In order to obtain a more consistent shading, Grottel et al. described a method to compute a smoothed surface normal by using the coordinates of nine neighboring fragments as control points of a quadratic Bézier patch [109].

Traditionally, molecular viewers employs "*cast shadows*" and the "*fog*" effect (often referred as "*depth cueing*") to enhance depth perception of molecular structures. *Cast shadows* simulate the shadows produced by an object when blocking direct light. One popular method to implement cast shadows is the "*shadow mapping*" technique, first introduced by Lance Williams in 1978 [174]. As regard to the "*fog*" effect, it has the purpose of simulating real fog and atmospheric perspective, in which contrast and saturation of the color of an object decrease as a function of the distance between the object and the camera, up to blend with the background color. The "fog" effect was natively supported by OpenGL since its first version, and can be easily reproduced in modern shader-based OpenGL.The combined effect of cast shadows and fog is shown in Figure 2.16 center.

In the last ten years, an alternate shadowing method known as "*Ambient Occlusion*" (AO in the following) imposed itself as preferred method to enhance depth

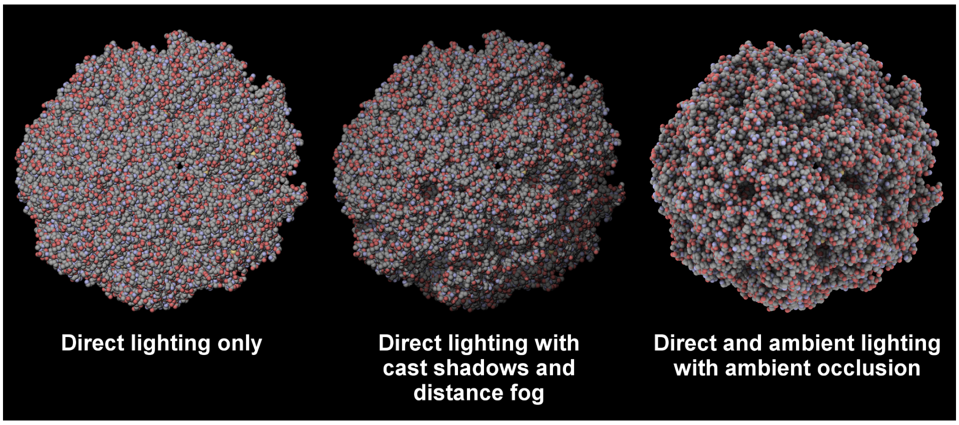

Direct lighting only
Direct lighting with cast shadows and distance fog
Direct and ambient lighting with ambient occlusion

Figure 2.16: Asymmetric chaperonin complex GROEL/GROES/$(ADP)_7$ (PDB ID: 1AON, [172]). Comparison between different rendering techniques in enhancing the perception of the three-dimensional structure of the molecule. Images generated with Qutemol [173]

perception of molecular structures, thanks to visual quality of the resulting images (see Figure Figure 2.16 right). AO was first introduced by Zhukov et al. in 1998 [175]. While cast shadows simulate the occlusion of the light emitted directly from light sources, AO simulate the occlusion of "ambient" (indirect) light. The many AO techniques proposed in literature differ in the way they compute the *ambient occlusion factor*, which provides an estimation of how much the considered point is exposed to the ambient light due to the presence of occluding objects in its neighborhood. In shading computations, AO factor is multiplied to the ambient light in order to darken occluded surfaces.

AO techniques can be classified in two families, called "*Object-space Ambient Occlusion*" and "*Screen-space Ambient Occlusion*". In object-space AO, the AO factors are computed on the base of the information about the elements of the scene. A popular object-space AO method designed for atomistic representations of molecules is the one presented by Tarini et al. and implemented in the *Qutemol* molecular viewer [173]. Although accurate, the estimation of AO factors in "object-space" is usually computationally expensive, making these methods usually unsuitable for the visualization of large time-varying data sets. To solve this problem, Grottel et al. proposed a fast object-space method specifically designed for the visualization of large time-varying set of spheres [176]. AO factors are evaluated by sampling a coarse-grained density volume obtained by aggregating the spheres. The density volume can be computed on the fly for each frame of the dynamics without major impact on the performance, while the estimation of the AO factor requires a single sampling of the density volume for each fragment. The tests conducted by the authors shows that their method allows to obtain full interactive frame-rates (~60fps) in the visualization of dynamic systems composed by two million spheres. The quality the produced images is good in the general case, although some artifacts or inaccuracies may occur. The main drawbacks of this method are that it exploits approximations valid only for spheres (although the authors say that adaptations are possible to handle other types of visual elements) and that the produced results greatly depend on the resolution of the grid (that should be calibrated according to the characteristics of the data set). Recently, Staib et al. presented a technique for combining AO and transparency in the rendering of large assemblies of sphere, by means of an hybrid approach mixing traditional local surface illumination with the emission-absorption model of volume rendering. However, their method requires the spheres to be sorted according to their distance from the camera, a requirement that produce a major impact on performance when dealing with large data sets. Furthermore, visual artifacts occur in the case of intersecting spheres, thus preventing its use in CPK representations.

Screen-space Ambient Occlusion (SSAO) techniques acts as post-processing effects, computing an approximated AO factor only on the base of information about the fragments survived at the depth test at the end of the rendering, such as their depth and normal vector. Being a sort of post-processing procedure, screen-space AO is performed in on the fly on each newly generated frame and it is independent both from the number and the type of objects composing the 3D scene. On the other hand, due to limited knowledge about the scene, the images produced by screen-space techniques are usually noticeably less accurate than the ones obtained with object-space algorithms. With regard to Molecular Graphics, Wahle and Wriggers integrated a SSAO algorithm in the open source modeling software *Sculptor* [177], while Eichelbaum et al. designed an improved SSAO technique for particle-based data visualized as arbitrary glyphs.

A noteworthy application for the AO information obtained with object-space methods have been presented by Borland [178]. Driven by the aim of producing more informative visualizations of the active sites of enzymes, Borland observed that since AO factors measure how much a surface is shielded from the environment light by surrounding objects, it implicitly provides an estimate of the “hiddenness” of an object. Starting from this consideration, Borland invented a new visualization technique to highlight internal cavities of molecular surfaces. The technique, called “Ambient Occlusion Opacity Mapping” (AOOM), modulates the opacity and the color of the molecular surface as a function of the AO factor: areas with an high AO factor are rendered more opaque and with a different color with respect to those with a low AO factor.

Advanced illumination effects like real-time shadowing are not only important in the visualization of molecular structures, but also in the study of volumetric data. Interested readers can refer to [179] for a recent survey on the subject.

As said, interactive computer graphics usually employs *local* illumination models, eventually enriched with additional techniques to simulate shadow, mirror-like reflections and semi-transparent materials. A radically different approach is to make use of rendering systems implementing a *global* illumination model, such as "*ray-tracing*" systems. Until some years ago, *ray-tracing* was considered an "*offline*" rendering method, since it took from several seconds to minutes (or even hours!) to output the final image, thus preventing its use in real-time graphics. However, thanks to the fact that the algorithm is inherently parallelizable, to the diffusion of multi-core CPUs, to the support offered by modern GPUs to general-purpose computations, and to the development of many algorithms and data structure to accelerate the process, real-time *ray-tracing* is becoming possible. In particular, *BallView* and *VMD* have integrated a real-time ray tracer system that can be activated on demand as alternative to the default OpenGL rendering [180–182]. However, the frame rate obtainable with these techniques with today's commodity hardware is significantly lower than the one achievable with traditional graphics API's such as OpenGL or DirectX. This fact makes them probably suitable for desktop uses, but not for VR applications where the scene must be continuously rendered in stereoscopy and at high frequency (due to the movements of the user's head).

3 Virtual Reality in Molecular Sciences

3.1 A brief introduction to Virtual Reality

Numerous definitions have been coined for to synthetically explain the meaning of the term *Virtual Reality* (VR), some of which are reported below.

In their famous book on VR, "*Understanding Virtual Reality*" [183], William Sherman and Alan Craig define VR as:

> "*A medium composed of interactive computer simulations that sense the participant's position and actions and replace or augment the feedback to one or more senses, giving the feeling of being mentally immersed or present in the simulation (a virtual world).*"

Richard Blade and Mary Padgett elaborated the following definition of VR for the "*Handbook of Virtual Environments*" [184] :

> "*Model of reality with which a human can interact, getting information from the model by ordinary human senses such as sight, sound, and touch and/or controlling the model using ordinary human actions such as position and/or motion of body parts and voice.*"

Grigore Burdea and Philippe Coiffet, in their book "*Virtual Reality Technology*" [185], provides the following definition of VR:

> "*Virtual Reality is an high-end user-computer interface that involves real-time simulation and interactions through multiple sensorial channels.*"

Finally, in a famous article titled "*Virtual reality in scientific visualization*" [186], Steve Bryson defines VR as:

"*Virtual reality is the use of computers and human-computer interfaces to create the effect of a three-dimensional world containing interactive objects with a strong sense of three-dimensional presence.*"

From the above definitions it is possible to infer the key elements that characterize VR:

- It is a *computer-generated simulation* of a synthetic (*virtual*) 3D world.
- It is *interactive*, i.e. the simulation must respond to user actions.
- It provides an advanced *sensory feedback*: the simulation produces artificial stimuli directed to one or more sensory organs, sophisticated enough to be perceived by the human brain as "authentic".
- It induces the user in a mental state of *immersion*: the ability of the computer-generated simulation of being interactive and to produce sophisticated artificial stimuli, fools the human brain, giving the user the mental feeling of being "*immersed*" in the *virtual world*.

Sometimes the term "*virtual environment*" is used as a synonym for *virtual reality* or *virtual word* [183].

Computer-based systems capable to satisfy these characteristics are said "*Immersive Virtual Reality*" (IVR) systems. The adjective "*immersive*", often implied, distinguishes these systems from the so-called "*Desktop VR*" or "*non-immersive VR*". Like immersive systems, *Desktop VR* exploit computer graphics and digital audio to simulate a 3D environment that can be navigated and manipulated by the user. However, in standard desktop systems, the user usually observe the virtual world through a monitor and interact with it by means of common input devices such as mouse, keyboard, joystick etc. [187]. Common examples of *Desktop VR* are video-games and visualization software for science and engineering.

Immersive systems, on the other hand, provide realistic stimuli for one or more sensory channels in order to induce the user in a deep mental state of *immersion* in the *virtual world*. Given the current technology, the sensory channels that usually involved in a VR simulation are: *visual*, *acoustic*, *haptic* and *vestibular* channels. "Visual channel is most important sensory channel for the human brain in the real and, hence, in the virtual world. Other channels may be stimulated, according to the type of simulation and to the available hardware. Of course, the greater is the number of sensory channels involved, the greater will be the mental immersion of the user. Tracking system are often employed to detect the position and orientation of parts of the user's body within the VR installation (at least his head and often also his hands or hand-held devices), in order to produce a coherent and realistic sensory feedback.

Until around fifteen years ago, there was a clear distinction between desktop and immersive VR systems: these devices was very costly, that only specialized research centers could afford them. Furthermore, their performance (in terms of latencies, precision, image resolution etc.), was barely sufficient to provide an immersive experience. Thanks to technological evolution, mainly driven by the demand of realistic contents for entertainment, a sequence of increasingly sophisticated low-cost devices have been developed to provide an immersive experience to games enthusiasts: from stereoscopic display technologies such as "*3D Vision*" from NVIDIA [188], to multi-channel audio systems to better simulate the placement of audio sources in the 3D space, up to desktop haptic devices for gaming (such as the "*Falcon*" from Novint [189]). However, these gaming stations still lacked of a tracking system and the visual immersion was limited by the size of the monitor. Today, a new generation of low-cost VR devices designed for gaming have just been (or will be soon) released on the market. These includes, just to cite a few, Head Mounted Displays (HMD) such as the "*Oculus Rift*" [190] and the "*HTC Vive*" [191], headphones with motion tracking to simulate spatial sound, such as the *"3D Sound One"* from 3D Sound Labs [192], and the "*Avatar VR*" gloves by NeuroDigital Technologies [193] (in pre-order at the time of writing), which promise to provide haptic feedback and relative positional tracking of the fingers[1]. Despite their limited cost, these devices are more performant, accurate, lightweight and ergonomic then their costly ancestors of a decade ago, and have the potential to finally bring IVR technologies in homes, schools and small research centers.

3.2 Creating artificial sensory perceptions

This section briefly discuss the devices used in IVR to mislead the human brain, inducing it to interpret computer-generated stimuli as "authentic".

3.2.1 Visual feedback

As said, the visual system is the main sensory channel by means of which humans perceive the external world. The main "*depth cue*" that the brain exploit to perceive

[1]Absolute positional tracking is possible in conjunction with the HMD's own tracking system.

distances and sizes is the so called "*binocular disparity*", that is the fact that the two eyes provides a slightly different view of a same observed object. The ability of the human brain to reconstruct a three-dimensionality mental model by analyzing the differences between these two images is called "*stereopsis*". IVR systems employ stereoscopic displays in order to provide different images to the two eyes. If these images are properly generated by taking into account the position of the user's eyes (see section 6.4 for details), the human visual system can be easily mislead, inducing the user to see a three-dimensional object that does not exists. The stereoscopic displays mainly used in IVR are:

- HMD displays, where a one or two small high-resolution screens are integrated within the HMD helmet and placed in front of the user's eyes.
- *Active* stereo projectors / monitors: they alternate the display of the images directed to the two eyes at high frequency (usually at 120Hz). The user wear a special pair of glasses that alternatively, and in sync with the display, make each of the two lenses opaque (black) or transparent, so that the each eye can see only the related image.
- *Passive* stereo projectors: they exploit light polarization to filter the images for the two eyes. In particular, left and right images are emitted with a different polarization of the light (commonly from two distinct projectors). The user wear a special pair of glasses, having polarized lenses to filter the images directed to the other eye. Requires a projected surface that preserve the polarization of the light. Recently, passive polarized LCD monitors was became available.

It is important to note that the effectiveness of *binocular disparity* in providing depth perception decrease with the distance from the observed object. For distant objects, or when a single monocular image is available, the human brain is still able to infer depth information by focusing on the so called "*monocular depth clues*". Notable examples of *monocular depth clues* are: the *shadow* cast by an object when occluding a light source; the visual *occlusion* of an object caused by another object lying in front of it; the decrease of the apparent size of an object when its distance from the observer increases, known as *perspective*; the *atmospheric perspective*, that is the effect according to which the saturation of the colors of objects slowly decrease with the distance from the observer, and their contours become increasingly blurred, up to blend with the color of the atmosphere on the horizon; the *motion parallax*, that is the differences in the apparent movement of objects lying at different distances when the observer changes his point of view;

etc. Simulating these cues in computer-generated images is important both to improve the realism and the meaningfulness of the rendered images (as discussed in section 2.2.5), especially in absence of stereoscopic displays.

3.2.2 Acoustic feedback

Besides visual rendering, simulating audio sources for the virtual world can further improve the realism of the simulation. Since VR reality deals with 3D worlds, the positioning of a sound source in space should be simulated. The ability of the human brain to determine the direction and distance of a sound source is known as "*localization*" [183]. As in the case of the visual system, the brain tries to infer the direction and the distance of a sound source on the base of a set of *clues.* As an example, a sound source lying on the right of the listener will be perceived by the right ear earlier and louder with respect to what perceived by the left ear. These and other principles can be used to derive a transfer function that, if applied to an unlocalized sound, produces a modified sound that will appear to the user as coming from a specific location in space. This process is called "*spatialization*" [183]. In order for a sound source to appear fixed within the virtual world, the position and the orientation of the listener's head must be one of the parameters of the transfer function. In simple cases, such as video-games, it is sufficient to use the position and the orientation of the character within the virtual world. In IVR systems such as the CAVE or HMD, where the user can physically turn his head and move within the virtual world, the position and the orientation of his own head (detected by a tracking system) should be used as parameter of sounds spatialization.

If the virtual environment is a model of a real or realistic location (such as a theater or a canyon), emitted sounds can also be manipulated according to a realistic model of propagation of the sound waves, by taking into account how the elements of the virtual scene reflect or absorb sounds.

Finally, sounds can be employed as a as a medium to convey quantitative information, both as alternative to or a complement of visual representation of data: the use of non-speech audio to represent and convey information is called "*sonification*" [194]. Some examples of virtual environments exploiting sonification to provide information about chemical data will be discussed in section section 3.7.

3.2.3 Haptic feedback

The Oxford Dictionary of English[2] defines the term "*haptic*" as:

> "*Relating to the sense of touch, in particular relating to the perception and manipulation of objects using the senses of touch and proprioception*"

Note that, in this context, "*proprioception*" is used as synonym of "*kinesthesia*", defined in [183] as "*the perception of movement or strain from within the muscles, tendons, and joints of the body*".

Haptic devices employed in VR allows the user to perceive the shape, the weight, the roughness etc. of a virtual object. Usually, they acts as both input and output devices, by tracking the position of a part of the body (e.g. the fingers or the whole arm) or of an object manipulated by the user, and providing touch or force feedback in output. Actually, the term "*haptic devices*" is a generalization for two different types of devices, called "*tactile displays*" and "*kinesthetics displays*".

Tactile displays stimulate the nerve sensors under the skin, so to simulate the perception of surface features like temperature, texture and roughness, but does not provide force feedback [183, 195]. Common examples of tactile displays are sort of gloves equipped with actuators under the fingertips, such as the "*Avatar VR*" gloves by NeuroDigital Technologies [193].

Kinesthetic displays, instead, are electromechanical devices in contact with the hands of the user (because directly connected or because provide some sort of object that can be manipulated) that produce forces in response to the user's actions [183, 195]. These devices can be used to simulate the mass of a virtual object manipulated by the user, the resistance of an object to being penetrated, the effects of force field etc. An example of a popular kinesthetic display is the the Geomagic Touch [196] (formerly Sensable Phantom Omni).

3.2.4 Vestibular feedback

The vestibular system helps humans to maintain equilibrium, coordinate movements, and perceive accelerations. Common vestibular "displays" are moving platforms equipped with seats, usually employed in vehicle simulators or theme parks [183, 195].

[2] https://en.oxforddictionaries.com

3.3 Coherency of the sensory feedback and VR sickness

One of the major challenges when developing a VR application is producing a coherent output for the different sensory channels, both from a spatial and temporal point of view. In particular, each generated sensory output must ensure a spatial coherence with the other types of sensory stimuli, with the position of the user and with the position of real-world objects (such as input devices) located within the tracking area. To this end, it is good practice to model the virtual world using a real-world unit of measure for sizes and distances, and to define a common reference frame for both virtual and real worlds. In other words, virtual objects and physical entities (such as the user, input devices and displays) should logically coexist. This usually involves the use of several changes of reference frame transformations, since input and output devices may use a specific local coordinate system, while the application logic should work in the global reference frame. Temporal coherency is another critical factor: latencies introduced by input and output devices and by the program should be short enough to not be noticed by the user. The perception of conflicting stimuli among the sensory channels (and in particular between the visual and vestibular systems) seriously compromise the sense of immersion in the virtual world and, even worse, may cause dizziness, headache or nausea. This latest phenomenon is known as "*VR sickness*", "*simulator sickness*" or "*cybersickness*" [183, 197].

3.4 IVR systems

The most diffused types of IVR systems are the "*Powerwall*", the "*CAVE*" and the "*Head Mounted Displays*".

The *Powerwall* is a large rear-projection stereoscopic display. Usually the user's head is tracked, thus allowing the application to generate correct stereoscopic images by adjusting view and projection parameters in real-time (see section 6.4 for details). User interaction is made possible by the use of hand-held pointing devices called "*wands*" (allowing a complete mobility), *haptic* interfaces (providing touch/force feedback at the cost of reduced freedom of movement) or mobile devices equipped with a touch screen (e.g. tablets).

The *CAVE* (recursive acronym for "CAVE Automatic Virtual Environment") is a room-like installation whose walls (and sometimes the floor and the ceiling) are large stereoscopic displays (see Figure 3.1). The first CAVE system was designed and built at the Electronic Visualization Lab (EVL) at University of Illinois Chicago in early 1990s [198, 199]. The number of projected surfaces can vary, but common installations have from 3 to 6 screens. A CAVE-like system with 9 screens (eight walls forming an octagon, plus the floor), called the "*Octave*", has been built at the University of Salford (UK)[3]. Recently, the CAVE2 was inaugurated at the EVL [200]. It is composed of 72 polarized LCD monitors arranged cylindrically, forming a large panoramic stereoscopic display. A cluster of 36 workstation is used to render images on the 72 monitors.

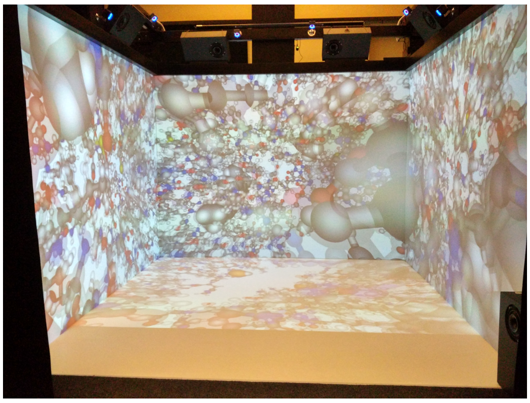


Figure 3.1: CAVE theater at Scuola Normale Superiore.

With respect to the *Powerwall*, the *CAVE* provides a more immersive experience, since the user fells surrounded by the virtual world. Special synchronization of video signals (known as "*Gen-lock*") is required in order to ensure that the images directed to the right and a left eyes are displayed in synchrony across the multiple displays (and with the shutter glasses, if used). Until some years ago, CAVE systems was driven by a specialized workstation equipped with multiple graphics cards or by a cluster of personal computers, since a massive graphics computing power is required to render stereoscopic images for multiple displays at a proper frame rate. Thus special software had to be developed for these systems. Thanks to technological evolution, is now possible to drive a CAVE with 4 screens with a single workstation equipped with high-end commodity hardware, as in the case of the CAVE theater at SNS, allowing to accelerate the development of dedicated applications (as discussed in chapter 4). Finally, as in the case of *Powerwalls*, the

[3]http://www.salford.ac.uk/octave

user is susually tracked, and can interact with the system by means of wands, haptic interfaces or mobile devices.

Head-Mounted Displays (HMD) are helmets equipped with one or two small displays placed in front of the user's eyes. They provide full visual immersion, since the user is not able to see the outside world. The helmet is tracked, so to adjusts the visible region of the virtual world according to the orientation (and in some cases also the position) of the user's head. Previous generations of HMD was characterized by a limited resolution and field of view. Furthermore, technology was not mature enough to provide low-latency tracking and high refresh rates of the images (at least 90Hz) leading to dizziness and nausea. Current generation of high-end HDM, such as the "*Oculus Rift*" [190] and the "*Vive*" from HTC and Valve [191] (Figure 3.2), solved almost completely these problems. Furthermore, they are commercialized at an affordable price (less then 1000$) and are gaining dedicated support by many graphics and game engines, facts that may allow them to gain a leading position in IVR in the near future. User can interact with a large range of devices, such as gamepads, dedicated wands, fingers-tracking devices (e.g. "*Leap Motion*" [201]), desktop haptic devices (e.g. "*Geomagic Touch*" [196]) and even gloves with haptic feedback (e.g. "*Avatar VR*" [193]).

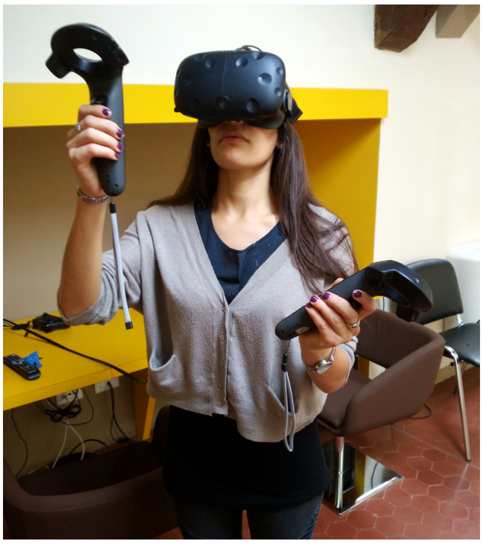


Figure 3.2: User wearing the HTC Vive head-mounted display.

Other types of IVR installations includes "*Dome*" theaters, equipped with an hemispherical display (such as "*iDome*" [202]), and the "*Workbench*" (such as the "*Responsive Workbench*" [203] or the "*ImmersaDesk*" [204]), in which the stereoscopic screen is placed in horizontal or oblique position, so to be used as an interactive desk or workbench.

An interesting IVR installation, built at the EVL of the University of Illinois at the end of the 1990s and designed mainly for chemical applications, was the Protein Interactive Theater (PIT) [205]. The PIT was a VR system for two users, equipped with two stereo displays oriented at 90° to each other and mounted on top of a common desk. The two users sat at the table, in front of each screen, and wore a pair of shutter glasses whose location was monitored by a tracking system. The system was designed so that the visualized 3D objects would appear co-located to the two users. In other worlds, both users saw the same 3D objects as lying at the same position on top of the desk, thus allowing to co-operate on a same task. Each of the two users could interact with the system by means of an hand-held pointing device (wand), and by means of a traditional LCD monitor with mouse and keyboard. The first presented application for the PIT was a crystallography application named *CORWIN* (for “coupled reciprocal windows”), which allowed to manipulate molecules to perform protein-fitting tasks [205].

3.5 Augmented Reality

While the aim of Virtual Reality is to replace the perception of the user of the real world with that of a virtual world, *Augmented Reality* (AR) tries to enrich the real world with additional computer-generated content. In AR systems, the user usually wear a HMD which allows him to see, at the same time, the real world and computer-generated objects. AR generated contents should help the user in performing complex tasks, such as to assemble or repair a complex machinery, or perform a surgical procedure. Although AR is considered a sub-field of VR, it has a different objective (as explained before) and different challenges: AR systems need to acquire images of the physical world, precisely detect the position of both the user and relevant objects in it, and generate images of synthetic elements to be “placed” in the world, with a precise position and alignment. A discussion of AR technologies and applications is beyond the aims of this thesis. Interested readers can refer to the surveys on the subject by Billinghurst et al. [206] and Costanza et al. [207].

3.6 How Scientific Visualization can benefit from IVR

As already discussed, scientific experiments and numerical simulations produce a large amount of data, often in the form of three-dimensional structures and volumetric dataset. Due to their three-dimensional nature, stereoscopic displays can definitely have a positive impact on the understanding of these data, as well as on the visual identification of relations, patterns and anomalies in them. Not surprisingly, almost every computer program for scientific visualization developed so far had/have the ability to produce stereoscopic images. However, as explained in section 3.2, binocular vision is not the only means exploited by our brain to construct a three-dimensionality mental model of the world surrounding us and to provide understanding about phenomena happening in it. Important is also the ability to physically explore the world with the movements of our our body, in order to both observe the objects from different perspectives and to encourage spatial judgments (of sizes, distances and angles) based on proprioception[4]. In VR, this physically exploration is simulated by tracking the position of the user's head. With the same aims, the ability of our brain to process sounds can be exploited through data sonification, while the stimulation of our kinesthetic sensory system by means of haptic interfaces can be a good means for gaining a deep understanding of the forces occurring in the system under study.

Of course, the reasons discussed so far are only expectations on the actual benefits that can be gained through the employment of IVR technologies in scientific visualization. However, several specific studies have been conducted in order to find evidences in support of this hypothesis. An example, is the work of Laha et al. [209] on the effects induced by visual immersion on the visual analysis of volume data. In the study, the candidates had to perform a set of tasks related to the visual analysis of two different datasets visualized in direct volume rendering. The experiments has been conducted in a CAVE system. In order to determine the specific effects deriving from the use of stereoscopy, head tracking, and multiple surrounding displays, the test has been repeated multiple times, enabling or disabling in turn these three technologies. The results proved that the most positive effects of visual immersion occurs in tasks involving complex search of features in the dataset. Overall, all the three technologies shown to lead benefits, but the use of multiple surrounding displays had the most positive effects on the widest range of tasks. Later, a similar study has been conducted, having the same aims and

[4]Defined in [208] as "*the ability to sense the relative positions of parts of our bodies and the amount of muscular effort being involved in moving them*".

methodology, but focussed on the isosurface representation of volume data [210]. The results showed that visual immersion lead to positive effects in 12 of the 15 proposed tasks. In particular, stereoscopy had the strongest effects on performance among the three factors. In another study, related to the visual representation of a class of second-order tensors volumes by means of streamtubes and streamsurfaces, the authors performed a visualization test in a CAVE environment to verify possible benefits for the comprehension of the data in question [211]. The representation of a dataset extracted from a human brain was showed to doctors and medical students. The opinions of the user gathered after that experience suggested that visual immersion and interactivity lead to a better understanding of (the showed) complex geometric models. Other positive feedbacks on the employment of these technologies are reported by many of the works discussed in section 3.7.

Given similar evidences, and despite of the fact that plenty of research projects have been developed with that aim, the penetration of VR in sciences (as well as in other fields) is still very limited. In a famous report on IVR for scientific visualization published in 2000, van Dam et al. [3] identify the high costs and immaturity of both hardware and software among the main factors of the low adoption of IVR technologies. Sixteen years after that report, the situation looks significantly more favorable: the latest generation of IVR devices, having far lower costs and significantly better performance of their ancestors, may be the key for a massive adoption of IVR in sciences, both for research, education or dissemination purposes. The only obstacle, at this point, is the avaiability of proper software capable to fully exploit the potential of these technologies.

3.7 Research on IVR in Molecular Sciences

This section briefly discuss some noteworthy examples, as well as some recent proposal, of research projects having the aim to take advantage IVR technologies in molecular sciences. They will be presented in chronological order. Finally, a brief overview about VR support in well-known molecular graphics system is given. Since visual feedbacks are of primary importance for the aims of this thesis, the following discussion will focus on those system making a appreciable use of IVR visual display technologies. However, also haptic devices are rapidly gaining in importance in this field, as evidenced by the vast research in the field: just to cite a few, studies have been conducted to exploit haptic devices for the interactive steering of molecular mechanics simulators and molecular dynamics [212–218], docking [2, 219–221], real-time Quantum Chemistry models for studying chemical reactivity [222–226], interactive fitting of molecular structures into electron microscopy density maps [227], etc.

Project *GROPE* [2] represents the first attempt to exploit IVR technologies to solve computational chemistry problems. It started in 1967 at the University of North Carolina at Chapel Hill, with the aim to develop a molecular docking system exploiting stereoscopic graphics and an haptic device with 6 degrees of freedom (dof) to provide feedback about (approximated) inter-molecular forces. Due to the complexity of the project and the absence or immaturity of proper hardware, the project was organized in four sequential stages spanning over 20 years, so to gradually develop the required software and hardware technologies. The final goal of the project, i.e. the development molecular docking system equipped with a 6-dof haptic device providing force feedback based on the evaluation of molecular force-fields, was reached at the end of the 1980s [2]. The effectiveness of system was evaluated by asking twelve experienced biochemists to dock four drugs into the active site of a protein. The users had to lower the potential energy of the system by moving and orienting the drug, as well as changing its conformation by adjusting six twistable bonds, up to reach a range of known real energies. The result showed that force-torque feedback only slightly improved the completion time. However, due to the complexity of the task and the large number of degrees of freedom (12 dof) provided to the users, about one third of the time was spent in thinking and observing the system, in order do chose subsequent manipulations. Excluding the time spent in this way, docking with force-torque feedback resulted 1.75 times faster. Furthermore, users reported a better understanding of the details of the receptor site and its force fields as well as of the reasons why each candidate drug docks well or poorly.

Short after the realization of the first CAVE at the Electronic Visualization Laboratory (EVL) of the University of Illinois at Chicago [198, 199], several scientists from various disciplines was invited to visualize their research data in the CAVE. A report on their experience was published in 1993 in a paper entitled “Scientists in wonderland” [228]. Among the presented case studies, two was related to the visualization of pre-computed molecular dynamics.

In 1995, Papka et al. [229] developed a application for CAVE systems for monitoring a running parallel molecular dynamics with interaction decomposition [230]. In this type of molecular dynamics, job decomposition is operated on inter-atomic interactions (forces) instead of atoms or region of space. Since this requires the use of complex scheduling algorithms, visualization was proposed to monitor the correctness of the tested scheduling algorithm and the achieved load balancing.

A year later, Akkiraju et al. presented a software to geometrically construct molecular surfaces and view them inside a CAVE [231].

In the same year, Haase et al. presented *VRMol* [232], a molecular viewer featuring stereoscopic rendering on a Powerwall, the use of a data glove and a 3D mouse as input devices and state-of-the-art rendering algorithms (for the time). In the same period, Cruz-Neira et al. developed the *Virtual Biomolecular Environment* (*VIBE*) [233], an interactive and immersive molecular dynamics simulation. It was constituted by an IBM SP Power Parallel computer running a MD simulator, connected with a high-speed network to the CAVE. Multiple molecules could by shown at the same time. The user could move and rotate them with a wand. Those actions was notified to the MD simulator, who computed the evolution of the system on the base of simulated inter-atomic interactions.

An interesting application called *PaulingWord* [234, 235], initially conceived as an educational virtual environment where students could explore molecular structures and interact with them, was presented in 1998. Supporting different types of VR systems, such as CAVEs and workbenches, this application allowed the user to explore molecular structures (loaded from PDB files) according to various representations. The ability to attach a structure to the wand was also provided, allowing basic didactic docking experiments. The main innovation brought by *PaulingWord*, consists in the fact that is also was a *Distributed Virtual Environment* (DVE): two or more users from different places of the real world could share the same virtual environment and collaboratively interact with the molecule.

In the same year, Ai and Frohlich presented *RealMol* [236], a VR molecular dynamics simulation. With respect to *VIBE*, it supported Powerwalls and HMD in addition to CAVE systems. It also provided some further feature, such as the ability to compute and show hydrogen bonds.

In 1999, Prins et al. [237] modified a previously developed system for the interactive steering of molecular dynamics, called Steered Molecular Dynamics (SMD) [238], in order to be used in the Protein Interactive Theater (PIT) [205] (see section 3.4). Prins et al. decided to port SMD on a VR system due to the difficulties encountered in both understanding the three-dimensional motion of atoms when observed on a 2D monitor and in defining three-dimensional forces with a mouse. SMD made use of VMD [71] as molecular graphics system interfacing the PIT with the MD simulator.

In the same period, a VR application for protein docking for the *Immersadesk* workbench [204], called *VRDD*, was developed by Anderson and Weng [239]. The user could search for a good fit by manipulating the two molecules with the wands, as if it would handling solid models. A collision detection procedure was performed to avoid clashes between ligand and receptor (according to the vdW radii). A metallic sound was also emitted in the case of clashes. The receptor-ligand binding free energy was computed in real time and showed as floating text. The program kept track of the best ten fits (i.e. the positions and orientations of the ligand having the lowest binding free energy) found during the manual docking. The final result was then computed off-line, by an automatic local refinement procedure based on the Metropolis Monte Carlo algorithm [240]. The authors tested the effectiveness of the system on two reasonably difficult cases and a very hard one, obtaining correct and accurate results for the first two, but failing to on the most difficult case.

An alternative to haptic devices in providing feedback when defining external forces for an interactive MD simulation was proposed by Koutek et al. in 2002 [241]. They implemented a VR steering environment called *MolDRIVE*, in which a virtual spring was used as visual feedback for the applied force. The virtual spring, connecting the interaction device handheld by the user with the manipulated atom, was drawn stretched and bent in accordance to the force applied to the atom. *MolDRIVE* used the Responsive Workbench [203] as VR system and was interfaced with a MD simulation program running on a supercomputer.

In 2005, Ghadersohi et al. presented a multi-platform networked collaborative molecular viewer and editor, called *SnB Visualizer*, supporting desktop computers with various operating systems as well as CAVE theaters [242]. The core of the system was a database storing information about molecular structures and their components. Remote clients running on desktop computers or CAVE systems could load these structures from the database, so to visualize and edit them. Every change made to a structure was notified to the database, who propagated the event to the other connected remote clients operating on the same molecule.

Another DVE for molecular modeling was created in the same period by Chastine et al. [243]. Called *AMMP-Vis*, it allowed multiple users from different physical locations to visualize and manipulate molecular structures. The proposed equipment for each user included a HMD, a VR glove and a joystick. The joystick allowed the user to navigate within the virtual world, while the glove was used to select a group of atoms and bonds by drawing a three-dimensional bounding box with a “pinch” gesture. Once selected, the elements cold be moved and rotated with a “grab” gesture. *AMMP-Vis* made use of two servers: one was responsible to propagate the movements and the actions of each user to the clients of the other participants (so that all the participants could have a shared and coherent view of the virtual environment), while the other listened for atoms movements, notified them to a molecular dynamics simulator, and transmitted the updated state of the simulation to the clients.

In 2009, Block et al. presented *KinImmerse*, a molecular viewer for VR systems. It was able to visualize molecular structures and properties from *kinemage* files (see section 2.1.3). *KinImmerse* was primarily developed for the Duke Immersive Virtual Environment (DiVE)[5], a 6-sided CAVE system at the Duke University (USA). By interacting with wand, user was able to move, scale and rotate the molecule, to switch between different conformations for the structure (if present) and to show or hide individual elements. Furthermore, the user was able to take annotations in the form of freehand lines in 3D. Finally, it was possible to pick a point and translate multiple structures on it, so to co-center them. Modifications operated on the 3D scene could then be stored in a kinemage file.

In the same year, the results of the *Combination of Sensorimotor Renderings for the Immersive Analysis of Results* (*CoRSAIRe*) project was published [219]. Its purpose was the development of an interactive protein-docking system exploiting VR technologies. Rendering was performed on a CAVE system using a customized version of PyMol [84]. A 3D mouse and an haptic device was used to manipulate ligand and receptor. Since several types of information are involved in finding a good docking configuration, the authors decided to implement different types of sensory feedback (*multi-sensory rendering*), so to avoid a overload of visual information. As examples, force-feedback produced by the haptic device was used to reproduce Van der Waals and electrostatic interactions, while electrostatic and Van der Waals energies of the complex were sonificated. An accurate series of interviews was conducted with experts in the field, in order to determine advantages and drawbacks of existing tools, to design a proper multi-sensory interaction, and to set up a workflow combining the immersive docking performed by the user with traditional automatic computational approaches. The aim was to exploit the expertise of the user to reduce the conformational search space during the automatic phase, thus reducing the total time required to obtain a small number of good configuration to be experimental validated.

[5]http://virtualreality.duke.edu/

Soon after the inauguration of the CAVE2 system (see section 3.4) at the EVL of the University of Illinois at Chicago, Reda et al. presented a dedicated molecular viewer [244] (2013). This viewer visualized static and dynamic molecular structures with an hybrid representation, combining the traditional ball-and-stick with the volumetric rendering of uncertain molecular interfaces proposed by Knoll et al. [171]. In the paper, the authors also describes their distributed ray-marching algorithm used to render both the electron density based interface of the molecule (as a volume) and the structure in ball-and-stick representation. It was specifically designed as a distributed algorithm, so to be executed on the cluster of workstation driving the CAVE2.

In 2014, Glowacki et al. proposed a system where multiple users could interact with a MD by means of body movements [245]. The system employed an array of commodity depth sensors to track human forms. Such forms were translated in energy landscapes to drive the simulation. A visual representation the ongoing dynamics, including the human-shaped energy landscapes, was displayed on multiple screens. An additional acoustic feedback was provided, resulting from the sonification of the atomic dynamics.

One year later, Marangoni and Wischgoll presented a multi-screen system designed to compare and identify differences among multiple conformations of a same structure [246]. The system employs 27 50” stereo displays arranged in three adjacent walls. With the default settings, a different conformation for the molecule is displayed on each screen. Head-tracking allows both to generate view-dependent stereoscopic images and to select the structure to be manipulated (by simply looking at it). Hand gestures, detected by means of a data glove, allow to select a manipulation modality among panning, rotation, or scaling. Arm movements, instead, are tracked to compute direction and intensity of the transformation applied to the selected model.

Recently, Stone et al. proposed a method for the generation of near real-time high quality omnidirectional stereoscopic images and their fruition via HMD [182]. The proposed method has been implemented in VMD and it is conceptually organized in three modules, running in parallel. The first is a GPU-accelerated ray-tracing engine, having the responsibility to generate the panoramic stereoscopic images. The second module, instead, is responsible for producing images for the HMD. On the base of the orientation of the user’s head, it identifies the portion of the panoramic image to be displayed, applies a proper corrective projection on it, and transmit the result to the HMD. It also detects user inputs and translations of the head, which provoke changes in the 3D scene or in the point of view. When these events occurs, the generation of a new panoramic stereoscopic image is requested to the 3D engine. This decoupling between rendering and HMD visualization is necessary because HMDs needs new images at high frame rates (usually 90Hz), a frequency that ray-tracing engines are not able to satisfy. Actually, the generation of omnidirectional stereoscopic images is so computationally expensive that, in the case of complex scenes, a cluster of GPUs is required. In that scenario, the local rendering engine acts as a *proxy*, that dispatch the request of rendering a new image to the remote GPU cluster, receives and decodes the result, and forward it to the module driving the HMD. The authors point outs that, although in the paper a HMD is used, with minor modifications the same technique could be applied to other types of IVR systems, such as the CAVE.

Among the most popular molecular graphics systems, VMD has a relatively good support for VR systems. Actually, as Stone et al. explains in [247], the original name of the software was "*VRChem*" since it was originally designed primary for IVR systems like the CAVE or the ImmersaDesk [204]. VMD supports those visual rendering systems by means of one of the following libraries: CAVElib (originally developed in early 1990s at EVL for the first CAVE [199] and still distributed by Mechdyne [248]), FreeVR [249] and VR Juggler [250]. However, the first two solutions requires specialized multi-GPU workstations in which the rendering-related data structures are accessible both by the master application and by the rendering slaves via shared memory. Other solutions (such as VR Juggler) allows to use VMD on cluster-baser IVR system (like many modern CAVEs), at the cost of a limited possible range of interactions [247]. The reason is that the internal state of a sophisticated application as VMD is too complex to be kept in sync in real time on multiple processes running on multiple machines (this problem will be discussed in depth in chapter 4). As regard to input devices, VMD supports a vast range of 6-dof devices (such as wands and 3D mouse) and haptic devices, thanks to the VRPN library [251]. Recent versions of VMD allows to create high quality 360° videos that can be viewed with HMDs [181]. Finally, VMD implements a strong support for the interactive steering of molecular dynamics simulations [212, 247].

As regard to other well-known molecular graphics systems, a commercial VR plugin for *PyMOL* [84] is developed and distributed by Virtalis [252], although experimental solutions exists to run *PyMOL* in CAVE-like system by means of the *Chromium* library [253] (as done by the Center for Information Technology of the University of Groningen in Netherlands [254]), or to support the Oculus Rift [255]. *YASARA*

[87] has some support for VR devices in its commercial versions, while experimental support of *UnityMol* [103] for modern HMDs was recently announced [256]. Also *Amira* [257], a commercial 3D visualization and analysis software for Life Sciences, has dedicated extensions for molecular visualization and CAVE-like system.

4 Included Paper: "Moka: Designing a Simple Scene Graph Library for Cluster-Based Virtual Reality Systems"

VR systems employing multiple screens, such as CAVE-like systems [198, 199], are among the most sophisticated IVR installations. Until a few years ago, these system commonly employed of cluster of PCs, equipped with high-end hardware and interconnected in a high bandwidth Local Area Network (LAN), in order to have enough graphics computing power to render stereoscopic images for multiple displays at a proper frame rate. Although widespread, this solution lead to an increase of complexity when developing dedicated software. In fact, in these systems, the application consists in several processes running in parallel on the nodes of the cluster, a subset of which is responsible for performing a specific part of the entire rendering process (e.g. for each screen, the rendering of the related stereoscopic images is delegated to a dedicated process). In order to produce coherent images across the screens, a strict data synchronization between the processes is required.

Since the support for CAVE-like systems, like the one installed at Scuola Normale Superiore, was one of the main requirements for *Caffeine*, one of the firsts problem to solve in its development was to find a reasonable solution to the above problem. Commercial solutions for the development of cluster-based VR application existed at the time, such as UNIGINE [258], MiddleVR [259], TechViz XL [260] or Vizard [261]. However, some of them were expensive solutions, while others they was designed mainly for fields like architecture, vehicles simulation, manufacturing, entertainment, etc., instead of scientific visualization and molecular graphics. As regard to free or open-source alternatives, the scenario was quite discomforting, among simplistic solutions and/or lack of a proper documentation and/or discontinued development. As Professor LaValle[1] points out in his recent book about VR [208]:

> "*In a perfect world, there would be a VR engine, which serves a purpose similar to the game engines available today for creating video games. If the developer follows patterns that many before her have implemented already, then many complicated details can be avoided by simply calling functions from a well-designed software library. However, if the developer wants to try something original, then she would have to design the functions from scratch. Unfortunately, we are currently a long way from having fully functional, general-purpose VR engines.*"

For the above reasons, I designed and developed a distributed scene-graph library called "*Moka*" on top of OpenSceneGraph [262], an open source high performance OpenGL-based graphics engine used in a wide range of applications (from games to scientific visualization). *Moka* was developed mainly as a base layer on top of which to build *Caffeine*. However it was designed with generality in mind, in order to be employed in others scientific visualization applications.

In this chapter, I included a paper which discuss the motivations that lead to the creation of *Moka*, its design and its features. It was presented by Dr. Andrea Brogni at the conference "SALENTO AVR 2014, First International Conference on Augmented and Virtual Reality" which took place in Lecce (Italy) on 17-20 September 2014, and published in the conference proceedings by Springer [6].

Afterwards, thanks to the latest advances in video card technology, it has became possible to drive a 4-sides CAVE system at full resolution with a single computer equipped with multiple NVIDIA Quadro GPUs in Scalable Link Interface (SLI) [263] configuration. With the aim to be able to spend more time on the development of new features for *Caffeine* (instead of spending time on improving and extending *Moka*), as well as to simplify the implementation of other VR applications in development at the SMART laboratory, it was decided to convert the CAVE installation at SNS from a cluster-based to a single machine configuration. As a consequence, the *Moka* library was discontinued: the code related to the distributed scene graph was removed from *Caffeine*, while the remaining components of *Moka* (such as the module for the ray-casting of spheres and cylinders and the configuration tools) are still part of the *Caffeine*'s code.

[1]Steven M. LaValle, Professor at University of Illinois and Chief Scientist of VR/AR/MR at Huawei Technologies Co. Ltd. http://msl.cs.illinois.edu/~lavalle/

Moka: Designing a Simple Scene Graph Library for Cluster-Based Virtual Reality Systems

Andrea Salvadori, Andrea Brogni(✉), Giordano Mancini, and Vincenzo Barone

Scuola Normale Superiore, Piazza dei Cavalieri 7, 56126 Pisa, Italy
{andrea.salvadori,andrea.brogni,giordano.mancini,
vincenzo.barone}@sns.it

Abstract. Clusters of PCs are widely employed in multi-screen immersive virtual reality systems. While this allows reducing the realization costs, it leds to an increase of complexity on the software side, since they require the development of distributed applications. Over the years, many frameworks supporting cluster rendering have been proposed, but none has established itself as the de-facto standard for immersive virtual reality application development. A new trend that is taking place consists in adding cluster-rendering support to one of the many freely available 3D engines. In this paper, we propose a convenient method to develop a lightweight distributed scene graph on top of a generic graphics engine. In particular, we describe the main mechanisms and design choices behind "Moka", a library for the development for cluster-based virtual reality applications. We also present "Caffeine", a virtual reality molecular visualizer based on the Moka library.

Keywords: Virtual reality · Cluster-based rendering · Molecular visualization

1 Introduction

Many immersive virtual reality systems (e.g. CAVE-like systems [1]) use multiple screens to surround the user with graphical representations of virtual environments, in order to increase the sense of immersion. The first systems of this kind employed expensive shared-memory graphics workstations with multiple video outputs. Thanks to the technological advances of personal computers' hardware, it has been possible to replace dedicated graphics workstations with a cluster of PCs equipped with high performance CPUs and GPUs, interconnected in a high bandwidth LAN. While this approach allowed the realization of multi-screen virtual reality systems at a fraction of the original cost, it led to an increase of complexity on the software side. In fact, these systems require the development of distributed applications consisting in at least one process for each node of the cluster, each one responsible to produce images for one of the screens of the Virtual Reality (VR) system, implying strict synchronization in order to keep coherency between screens.

In the following, we use the terms "cluster rendering" (CR) or "clustering" to refer to algorithms and applications related to the development of interactive graphical distributed applications, capable of producing a coherent and synchronized rendering on a cluster-based multi-display system.

L.T. De Paolis and A. Mongelli (Eds.): AVR 2014, LNCS 8853, pp. 333–350, 2014.
DOI: 10.1007/978-3-319-13969-2_25

Many of the techniques described are also employed in the realization of Networked Virtual Environments (NVEs) [2], although there are some important differences that must be taken into account. In particular, NVEs are designed to handle large numbers of network users, which concurrently modify the state of the virtual world. However, since each user has his/her own "vision of the world", transient inconsistencies are tolerated. In the case of applications for multi-screen virtual reality systems, instead, the virtual world is usually observed and manipulated by one or at most few users, but a much more strict synchronization is required to avoid inconsistences between the projected images, that would compromise the sense of immersion. Another related but distinct research field is the "parallel rendering" methods, in which parallel computing techniques and multi-GPUs systems and/or clusters of graphics workstations are exploited to speed up the rendering of complex datasets. The aim of these techniques is therefore orthogonal to the "cluster rendering", although the two methodologies can be combined (e.g. Chromium [3]).

Over the years, many VR frameworks supporting CR have been proposed, but none has been able to establish itself as a standard for immersive VR application development. Thanks to the availability of many mature and freely available 3D engines, a new trend is going on, adding CR support to one of them. We propose an efficient method to develop a lightweight distributed scene graph on top of a 3D graphics engine. In particular, we introduce "Moka", a library to simplify the development of applications for cluster-based immersive VR systems. Although Moka has been thought mainly for scientific visualization (with focus on molecular sciences), most of its mechanisms and design choices are general solutions that can be applied to any application field.

The paper is organized as follows: section 2 provides an overview of the most diffuse solutions to realize CR, while section 3 describes our motivations; in section 4, we present the "Moka" library; finally, in section 5 we present "Caffeine", a VR molecular visualizer based on the "Moka" library.

2 Cluster Rendering Approaches

The main drawback of cluster-based VR systems is that they require the development of distributed applications whose output image streams must be coherent across the screens. This requirement generally involves three different levels of synchronization [4, 5]. First, to obtain a correct stereoscopy, the generation of right and a left eye images must be synchronized across the multiple displays. This involves the synchronization of video signals (Gen-lock), a feature automatically provided by the current high-end GPUs. Second, each node of the cluster generates a new frame only when the others have finished producing the previous one (Frame-lock). This mechanism is not strictly necessary, but its lack may produce discrepancies across the generated images. On the other hand, it introduces a synchronization barrier that may have a negative impact on the frame rate. Finally, we have to synchronize the data used as input of the rendering process. Obviously, it is possible to implement this functionality at the application level, like in a generic distributed application. However, this approach significantly increases the development time, since the code dedicated to the replication of the application model between the nodes grows with the application's features and, furthermore, this code is very application-specific. The development could be simplified by using middleware that provides high-level functionalities such as object replication and remote method invocation (RakNet [6], ReplicaNet [7], Ice [8], Collage [9]).

A better solution consists in employing a framework specifically designed for this type of applications, which takes into account their common peculiarities. Over the years, several approaches have been proposed to realize generic toolkits for CR, trying to perform the data distribution in a way as transparent as possible for the programmer, so that the development of the applications would require only minor modifications with respect to a non-distributed one.

2.1 Input Event Distribution

In this approach, an identical copy of the application runs on each node of the cluster and the synchronization is obtained by broadcasting the data received from input devices (e.g. tracking systems) for all the instances. However, keeping the state of the application instances reasonably synchronized is difficult, because some computations (such as animations and physical simulations) are time dependent, in the sense that their results depend on the time they start/stop and on the time intervals between the executions of consecutives simulation steps. Since each application instance runs on a different machine, divergences may arise. Similar problems are caused by the use of random numbers and in general by the presence of any source of non-deterministic behaviors. To minimize these inconsistences additional synchronization mechanisms (such as the Frame-lock) are required. Finally note that, for similar reasons, the start of the application's instances should be synchronized and it is not possible to add further instances afterwards[1].

Despite its drawbacks, this approach is one of the most popular. It is employed, for example, as main mechanism to support clustering on VR Juggler [10–13].

2.2 Graphics Command Distribution

An interesting approach consists in intercepting the calls made by the application to the graphic API and streaming them over the network. Rendering slaves, running on each node of the cluster, receive these messages, decoding them and calling the corresponding graphic functions.

The main drawback is that, if the application performs many API calls per frame or if these calls manage many data, the network could be saturated, with consequent impact on the performance of all the services and applications sharing the same network. Moreover, like in input event distribution, it is not possible, in general, to "hot-plug" further rendering slaves once the application started. Finally, graphic libraries like OpenGL expose a vast and frequently expanded API, so implementing and keeping updated a corresponding proxy library requires a considerable development effort.

Examples of projects designed to exploit a similar approach are WireGL [14], Chromium [3] and XVR [15].

[1] More precisely, such a feature is possible but it should be explicitly implemented at the application level.

2.3 Scene Graph Distribution

Since scene graph structures are widely used in 3D graphics, another popular approach consists in replicating and keeping synchronized these structures on each node of the cluster. By doing so there is no need to run a complete copy of the application on each node, thus a master-slave architecture is usually employed: the "real" application acts as the master process, implementing the business logic, handling the input, and manipulating a proxy scene graph. The main purpose of this scene graph is not to render the 3D scene, but to keep track of the graph state and notify each change to the slaves. The slave processes, instead, have their own copy of the scene graph, keeping it updated according to the received messages, and drawing it on screen.

This technique has many points in common with the graphics command distribution approach: both employ master-slave architecture in which slaves are lightweight programs, whose only purpose is to render into the screen, and they are independent from the specific application that controls them. On the other side, the application (usually) does not perform rendering, so the workload is better distributed with respect to those approaches in which the whole application is replicated.

In contrast to previous approaches, the run-time insertion of further rendering slaves is quite simple to implement: when a new slave connects to the master, it is sufficient to perform a visit in which the whole graph is serialized and sent to the new slave.

The main drawback of this solution derives from the fact that, if the scene graph has not been designed from the beginning to be distributed, implementing such features afterwards could be very complicated and time consuming. That would usually consists in extending all the nodes' classes and override all the methods that change their internal state. However, real world's scene graphs are very complex libraries, so extending just a small subset of the available nodes would require a considerable effort. In some cases the problem can be circumvented by exploiting specific features of the used library, like in the case of Distributed Open Inventor [16]. Another possibility consists in wrapping the original scene graph within a set of classes that expose a smaller interface and implement data distribution. Although by doing so the interface of the original library is not maintained, it simplifies the addition of clustering support. For that reasons we choose to follow this path, as already done by other libraries like AVANGO [17]. Examples of libraries implementing a distributed scene graph are OpenSG [18, 19] , AVANGO [17], Distributed Open Inventor [16] and Syzygy [20].

3 Motivations

One of the first problems to face when developing a new VR application is to decide if relying on an existing framework, such as the open source projects VR Juggler [10, 13] and OpenSG [18], the free to use but proprietary XVR [15], commercial products like Unigine [21] and MiddleVR [22] for Unity [23] or other solutions. Other options are the development of an entire VR engine from scratch or the adoption of an intermediate solution by implementing a sort of "VR layer" on top of some pre-existing graphics engine. In particular, when choosing the right tools for the development of medium/large sized projects, many different aspects must be considered and evaluated, including: support of critical features in the applications; maturity of the project; documentation and maintenance; integration with other libraries.

Of course, it could be difficult to choose without making relevant compromises. On the other hand, today there are plenty of mature, freely available 3D engines (such as Unity [23], OpenSceneGraph [24], OGRE [25] etc.). Many of them are free to use also in commercial projects and are accompanied by a vast community providing support, documentation and numerous plugins that facilitate their integration with other libraries (such as GUI toolkits, physics engine etc.) and various kinds of devices (including the new low-cost VR-like devices such as the Oculus Rift [26]).

For these reasons, a new trend that is taking place is to add clustering support to these engines. Beside commercial products like MiddleVR [22], a few freely available plugins for popular engines can be found, but most of them are simplistic solutions based on the replication of the input data and/or of the view matrix. Libraries that implement a distributed scene graph on top of an existing engine are even less common and in most cases impose significant constraints, like requiring a static graph.

Use of an established traditional 3D engine would allow to meet all the requirements of our project (with the only important exception of CR), without onerous limitations, finding a way to implement clustering on top of it. However, we had to find a way to implement clustering on top of it in a limited amount of time. We introduced an abstraction layer on top of a pre-existing graphics engine, which exposes a simplified (but not simplistic) distributed scene graph. As expected the clustering functionalities are almost completely transparent to the developer, allowing the porting of an application using Moka to various kind of environments (from desktop to immersive VR systems) with only minor changes to the code. Furthermore, the structure and the content of the distributed scene graph can be dynamically manipulated and extended. It is important to note that we do not claim that our approach is the best possible, nor that the mechanisms we describe are particularly original. Instead, with this work we want to present a reasonable solution to a diffuse problem, in the hope that other people in similar situations may find it useful.

4 The Moka Library

The approach we adopted consists in developing a high-level scene graph on top of an existing graphics engine, providing a simpler and more concise interface and clustering support. The main characteristics and benefits of the approach we adopted are the following:

- By wrapping an existing scene graph instead of extending its classes, it is possible to expose a simpler interface, thus reducing the clustering related code.
- Thanks to the simplified interface, much of the complexity of the native engine can be hidden, thus accelerating the development process and making the library usable also by people with limited experience.
- Our approach does not rely on specific mechanisms provided by a specific engine (as [16]), thus making it applicable to other graphics engines. The substitution of one engine with another has a very limited impact on the applications' code.
- There are no constraints related to the scene graph structure, like it happens in many simplistic solutions: the structure and the content of scene graph can be dynamically modified, by creating and deleting nodes, removing a sub-graph from its

current position and attaching it to another node, loading a mesh from file, creating geometry and the related attributes procedurally, applying shaders etc.
- The mechanisms used to replicate the state of the scene graph can be exploited to create application-specific distributed objects, thus providing further flexibility.
- Frame-lock mechanisms may not be strictly needed.

The main drawback of our solution is that, due to the way it was conceived, it is better suited for applications that require only a subset of the functionality offered by the original engine. However, this fact does not impose constrains or prevent in any way to expose all the needed functionalities. In the worst case, in fact, the designer will define a scene graph as complex as the underlying one.

Moka has been implemented using OpenSceneGraph (OSG) [24] as native graphics engine, the Qt framework [27] as general-purpose library, Enet [28] for networking and GLM [29] as mathematics library. However, the concepts and mechanisms at the basis of Moka do not depend on specific technologies, so they can be easily adapted to other languages and toolkits.

4.1 The Basic Scene Graph

Each node of Moka wraps one of more OpenSceneGraph (OSG) nodes. As usual, the internal nodes of the graph represent a transformation to be applied to their children, while leaf nodes encapsulate the elements of the scene, such as meshes. At this stage, four different nodes have been defined to represent transformations: in addiction to a generic SGMatrixTransform class, specific nodes have been provided for translation, rotation and scaling. These last three classes also provide specific methods to animate the transformation over time (see section 4.5). As far as concerns the elements of the scene, two types of nodes have been defined: the *SGMeshFile* class allows to load a mesh from a file, while the *SGGeometry* class allows to define custom geometry (see section 4.6). Specific classes have been defined to represent materials, shaders and set of uniform variables. These kinds of objects can be dynamically attached to the *SGGeometry* nodes, obtaining custom graphical representations.

These classes can be used directly to develop desktop 3D applications, like any other scene graph. The most interesting part concerns, however, the mechanisms that allow the replication of these objects on each node of the cluster. With the exception of the initialization phase, these mechanisms are completely transparent to the programmer: thanks to polymorphism, the programmer can write his/her cluster-based application in the same way he/she would write desktop ones.

4.2 Distributing the Scene Graph

Suppose to have a set of objects living on the master process and a corresponding set of objects for each slave process. In order to keep all these instances synchronized, we need to inform the slaves about all the changes applied to the objects owned by the master. When a slave receives one of these messages it must be able to identify which one, between the objects he manages, corresponds to the object modified by the master and to update its state accordingly. In order to do so, each object must be associated to a unique global identifier such that:

- All the objects living within a single process must have different identifiers.
- Each object managed by the master and all its replicas must share the same identifier.

A convenient way to enforce these constraints is to generate a unique identifier every time the master creates a new instance. Then, when a message is sent to notify the slaves about the new object, the ID is included in the payload, so that they can assign it to the newly created object.

In Moka, the "*DistributedObject*" class has been defined to generate and store these identifiers. As the name suggests, this is the superclass of all the objects replicated between master and slaves (see Fig. 1). It provides two constructors:

- the first constructor does not accept any parameter; it is used only by the master process, and is responsible for the generation of unique global identifiers. Only positive IDs are generated in this way;
- the second constructor accepts an identifier as parameter, and it is able to create objects with a predetermined identifier. Slave processes use it when creating replicated objects but, as we will see in the following, it can also be employed on the master side to create application-level distributed objects.

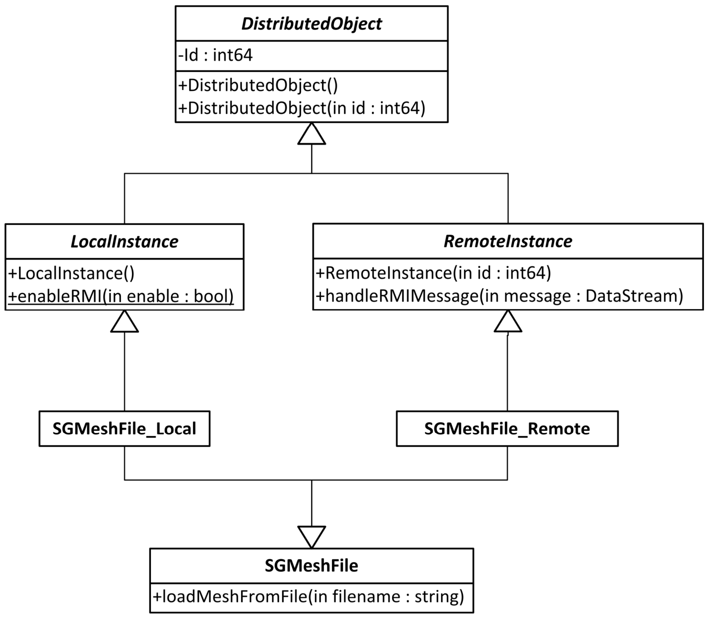

DistributedObject
-Id : int64
+DistributedObject()
+DistributedObject(in id : int64)
LocalInstance
+LocalInstance()
+enableRMI(in enable : bool)
RemoteInstance
+RemoteInstance(in id : int64)
+handleRMIMessage(in message : DataStream)
SGMeshFile_Local
SGMeshFile_Remote
SGMeshFile
+loadMeshFromFile(in filename : string)

Fig. 1. Relation between the classes involved in the definition of a distributed object

In order to distinguish among the functionalities to be used by the master program and the ones designed for the rendering slaves, the DistributedObject class is specialized by the "*LocalInstance*" and "*RemoteInstance*" classes[2]. The purpose of these classes is to establish the basic mechanisms by which modifications applied to a "local" object are notified to the corresponding "remote" instances. Rather than collecting the changes in a list to be periodically distributed, as suggested in [18, 19], we preferred an approach more similar to the Remote Method Invocation (RMI). In particular, *LocalInstance* subclasses act as proxies, i.e. they encode each method call as a network message and broadcast it to the slaves. On the other side *RemoteInstance* subclasses act as "skeletons", decoding the received messages and calling the corresponding methods. Actually, there are some important differences between the classical RMI and the mechanism we employed:

- Unlike "stub" objects, *LocalInstances* maintain a complete internal state so to respond to "getter" methods directly, thus avoiding any unnecessary network communication and the consequent wait for a response.
- As a further consequence of the previous point, any return value can be provided directly by the *LocalInstance*, thus our pseudo-RMI mechanism does not support responses from remote objects.
- The communication is one-to-many and unidirectional, as in the publish/subscribe model.

From what described so far it follows that, in order to obtain a distributed version of the scene graph, a "local" and a "remote" subclass must be defined for each class of Moka. As an example consider Fig. 1, showing the classes involved in the distributed version of the SGMeshFile node, and their relationship.

To better understand what happens when the master program invoke a method on a "proxy" object, consider the following pseudo-code showing the definition of the "loadMeshFromFile" method of the SGMeshFile_Local class:

```
if(isRMIEnabled()) {
   // Sends the RMI message
   byte[] msg = encodeLoadMeshFromFile(this.id, filename);
  NetworkManager::sendMessage(msg); // See section 4.4

   // Prevents that further messages will be generated
   enableRMI(false);
  // Loads the mesh
  SGMeshFile::loadMeshFromFile(filename);
  enableRMI(true);
} else {
   // Loads the mesh without sending any message
  SGMeshFile::loadMeshFromFile(filename);
}
```

[2] Note that, here and in the following, we will use the term "local" class/object to refer to the classes/objects used in the main (master) application, and the term "remote" class/object to refer those used by the rendering slaves.

Since in general these operations may involve other (virtual) methods of *LocalInstance* subclasses, we must avoid that further network messages would be generated in the process. For this reason the *LocalInstance* class provides the “enableRMI” static method that allows to temporarily disable the pseudo-RMI mechanism.

4.3 The Distributed Objects Registry

The *DistributedObjectsRegistry* class, a singleton used only by the rendering slaves, implements several of the key functionalities required by the pseudo-RMI mechanism. Its main purpose is to dispatch the received messages to the corresponding remote object, but it is also responsible for their lifecycle.

In order to perform message dispatching, the *DistributedObjectsRegistry* manages a dictionary that associates all the distributed objects to their global identifier. Whenever a RMI message is received, it extracts the destination ID from the message and performs a look up on the dictionary to retrieve the corresponding remote object. The same mechanism is exploited to allow the manipulation of a remote scene graph. These operations are particularly tricky, involving the dynamic composition of remote objects in complex data structures. A simple solution can be derived by observing that the methods corresponding to these operations usually take one or more (distributed) objects as arguments. Therefore it is possible to insert the IDs of these parameters into the message corresponding to the operation. When the remote instance decodes this message, it queries the DistributedObjectsRegistry to obtain a pointer to the remote counterparts of the involved objects, and performs the requested operation using these pointers as parameters.

Some details must be provided on how our library is intended to be used from the developer’s point of view. In particular, direct instantiation of Moka’s classes is strongly discouraged. Instead the programmer is required to use a set of predefined factory functions that returns a pointer to a new heap-allocated object. These functions takes care of instantiating the right class according to the type of application under development: as an example, the “newSGMeshFile()” function returns an instance of the SGMeshFile class if we are developing a desktop application, a “SGMeshFile_Local” instance when developing the master program and a “SGMeshFile_Remote” object for slave programs. The application type must be specified by the developer by means of an initialization function. In the case of slave programs, this initialization function also takes care to register the factories into the DistributedObjectsRegistry, which manage another dictionary containing the associations between a string, which identifies the type of the nodes, and the corresponding factory function. In this way, whenever a LocalInstance is instantiated, its constructor sends a message containing the type identifier of the new object. The DistributedObjectsRegistry can therefore retrieve the corresponding factory and fulfill the creation request. The destruction of remote objects happens in a similar way: whenever a LocalInstance is being destroyed on the master process, and its destructor is invoked, a message carrying the object’s ID is sent to the DistributedObjectsRegistry, asking for its destruction.

Finally note that, to maintain the same format between the pseudo-RMI messages and those directed to the DistributedObjectsRegistry, it was decided to assign to it the global identifier “0”.

4.4 Network Communication

Most of the messages generated when modifying the scene graph must be delivered reliably, in order to guarantee the correct synchronization of the shared graph. There are cases, however, where unreliability should be preferred, as we will see in section 4.5. If both reliable and unreliable transport protocols are used to implement a single specific functionality (e.g. by means of “raw” TCP and UDP sockets), the implementation should deal with messages interleaved on two distinct communication channels and ensure that they are handled in the correct order. This way to proceed would lead to a significant increase of the complexity of the protocol.

For the reasons just exposed, we decided to make use of Enet [28], a simple and robust network library originally developed for the multiplayer first person shooter Cube [30]. Enet implements a dedicated protocol on top of UDP, providing features like reliable and ordered packet delivery, automatic fragmentation and reassembly of packets, connection monitoring, automatic connection retry etc.

For our purposes, the most important features offered by Enet are those related to reliability and sequencing, which can be enabled or disabled on a per-packet basis. In this way, the messages to be transmitted reliably can be mixed with the unreliable ones on the same logical channel, while preserving the delivery order.

To isolate the networking code from the rest of the system we wrote a small module dedicated to network communication. This module consists in a thread that exploits Enet to send and receive messages to/from the other processes, and in a singleton class (the “NetworkManager”) acting as an interface between the communication thread and the rest of the application.

4.5 Animations

The visualization of dynamic systems is one of the fundamental tasks of the computer graphics. From a computer graphics point of view this corresponds to the generation of an animation, intended as a progressive time-dependent variation of the state of the graphical scene according to some mathematical model and/or to the user’s input.

In this section, we describe an effective method to implement animations in a CR engine. This method is an adaptation of the techniques employed in NVEs [2] and does not employ Frame-lock synchronization, so small discrepancies may occur. However, the animations updates are frequent enough to make such discrepancies acceptable for the user in most of the cases, especially in research fields like molecular sciences. In our case, the aim is having not only visualization of data structures but also have continuous and effective interaction. This means that the collocation between visual graphical feedback and real position of the tracked hand will be a critical point for making this effective and simple. Our choice to do not implement the framelock, at this stage, comes from the priority to have fast visual responses when the user look at a specific point and try to interact with his/her finger.

Let us call “key frame” a given configuration of the animation, constituted by the initial state to be assigned to the node and all the information needed to compute (or approximate) its subsequent evolution. Also let us call “in-between” a portion of an animation between two key frames, which can be computed by applying a known function (e.g. a linear function) to the information provided with the last key frame. Then the master can use reliable messages to communicate only the key frames and, during the in-betweens, send small unreliable packets carrying the essential information to keep the computation adequately synchronized across the slaves.

In Moka, the methods that start an animation on a given node actually set a new key frame and, in the case of “proxy” objects, they generate a reliable pseudo-RMI message (see 4.2 and 4.3). To obtain an animation composed by multiple key frames, it is sufficient to call these methods whenever a new key frame must be set, without stopping the animation before each call. To ensure a correct synchronization in these cases, the “start” messages must include the current state of the node in their payload. For the same reason, the “stop” messages must be delivered reliably and must contain the state of the node at the end of the animation.

The animation steps are performed by the “update” method, which updates the state of the node as a function of the elapsed time from the last key-frame. We call this time span the “animation time”. The “update” method is provided by any node of the scene graph (not only on those that support animations) and it propagates the call to each child. It is thus possible to invoke the update method on the root of the scene graph (letting the call to traverse the entire graph), without the need to keep track of the currently animated nodes. In order to keep the replicated scene graphs synchronized the animation time is periodically sent to each slave. In particular, each time the “update” method is executed on a currently animated local object, a message carrying the new animation time is generated and dispatched to the corresponding remote instances as usual, but with two important differences:

1. These messages are unreliable. However, their ordered delivery is ensured, even with respect to reliable messages.
2. When these messages are handled by a remote object, a special method is called instead of “update”. This method has the only purpose of substituting the stored animation time with the received one.

From the previous discussion, it follows that the periodic update of the scene graph can be performed independently on the master and on each slave with different frequencies. In particular, on the slaves the update visit is performed once per frame, thus ensuring a smooth animation, while the master can adopt a lower frequency (e.g. 5-10 times per second), thus reducing the workload and the generated network traffic. From our experience, these update frequencies are sufficient to ensure a synchronization good enough to fool the human eye.

4.6 Procedurally Generated Geometry

Since our main research field concerns scientific visualization, with a strong focus on molecular structures, it was fundamental to have a way to define a new geometry at runtime (e.g. to visualize data resulting from numerical simulations or generated procedurally) and to apply to them custom graphical representations. For that reason, we defined the SGGeometry class, which allows defining a set of vertices and the related vertex attributes. The geometry is defined by specifying a vertex array, an optional array of indices and the type of geometrical primitives to be generated. Similarly, vertex attributes are assigned by providing an array containing a value for each vertex. Custom representations can be obtained by assigning a shader program to the geometry node. For that purpose, specific classes have been defined to load and encapsulate a shader program and a set of uniform variables. In a similar way, material properties are encapsulated in a dedicated object that can be attached to the geometry node.

An important problem has to be taken into account: since the various attributes and properties are set/added separately, the corresponding messages will be received and handled by the slaves at different times. It follows that the geometry may be rendered when it is still in an inconsistent state. For this reason, SGGeometry objects buffer all the requested changes, and apply them all at once when a specific method is called.

4.7 Local Sub-graphs

The design described so far requires that the master process is the only responsible for managing the application model and its graphical representation (in the form of a scene graph). The scene graph is then replicated and kept synchronized on each rendering slave. However, nothing prevents the slaves from extending the shared graph with specific local additions. This approach, also proposed in other systems like [16], may result useful in several ways, allowing to add ad-hoc functionalities to a specific subset of slaves. Although at first this may seem unnecessary, it can actually become an important feature in the realization of more articulated applications.

Consider, for example, an application running in a CAVE-like system [1] in which the user can interact by means of a tablet computer. Several today's tablets are in fact handheld PCs equipped with a desktop-class operating system, so they are able to run a custom client which acts both as a rendering slave (since it renders the shared scene graph) and as an input device. In that scenario the tablet does not just render a portion of the scene (like the other slaves does), but instead provides a different view of the virtual world, possibly enriched with additional information.

Another important application of this technique is related to head-tracked stereoscopy: to implement stereoscopy each slave must set the view and the projection matrix appropriately, in function of the physical characteristics of the driven display and of the current position of the user with respect to the screen [31]. It follows that it is not possible to use a shared "camera node", since each slave must be able to set the virtual camera parameters according to its individual configuration. For that reason, the "viewing branch" is not included into the shared graph, letting each slave to treat it autonomously as an extension of its local scene graph.

This choice also influences the way the data produced by the tracker are handled, since it would make little sense to gather that information on the master and then propagate them to the slaves. Instead, the data generated by the tracking device are "multicasted" to all the processes via UDP, thus minimizing both the network traffic and the latency [16]. Note that the tracker data can also be received and handled by the master process, since they represent an input for the application.

Our library permits to define custom sub-graphs owned by a slave. In particular, these sub-graphs can be constructed using the native graphics engine or by instantiating directly (i.e. without using factories) the non-distributed version of the Moka nodes. For those tasks commonly required in CAVE-like systems, like head-tracked stereoscopy and keystone correction, the library provides specific utility classes, which transparently manage a local "viewing sub-graph". Tracker data are received and handled directly by the slaves, but the master, if required by the application logic, can also process them.

4.8 Hot-Plug Synchronization

Being able to start a new rendering slave at any time and without particular constraints is a desirable feature, both where the user interacts by means of a dedicated client and when we want to improve the robustness of the system (thanks to the possibility to restart a slave after a crash/fault without restarting the entire application). Only few systems natively support such a feature. In particular, the feasibility of such a mechanism at the framework level also depends on the chosen data distribution model. Scene graph based approaches are naturally predisposed to support this feature, since it would just require a visit of the graph in which, for each encountered node, a sequence of messages is generated in order to reconstruct the node's state on the slave's side. In our implementation, each time a new slave is started, it sends a message to the master asking to be synchronized. On the master, this "hot-plug" synchronization consists in the following three sub-phases:

1. the graph is visited, asking to each node to prepare itself for the imminent synchronization (e.g. by stopping any running animation). Then the "NetworkManager" (i.e. the interface to the network communication module, see section 4.4) is asked to transmit all the future messages only to the newly connected slave;
2. the main visit of the graph is performed, in which each encountered node sends all the messages required to create a corresponding remote instance with the same internal state. Note that the generated messages are the same pseudo-RMI messages described so far, so as to exploit all the related pre-existing mechanisms;
3. a final visit is performed, notifying each node that the synchronization is completed, so that it can resume any possibly pending task. The NetworkManager is finally asked to broadcast all the future messages to any slave.

Except for the initial synchronization request, this entire procedure is completely transparent to the newly connected slave.

4.9 Application Level Distributed Objects

Our library can be extended in various ways. In addition to extend existing ones or add new classes, application-level distributed objects can be defined by using the same mechanisms and techniques we designed for scene graph replication. Given a regular class, the programmer can obtain a distributed version of it by implementing its “local” and “remote” subclasses and paying attention to assign to the local class a fixed, negative identifier. Then a factory function must be defined, returning a pointer to an instance the right class according to the type of application being developed (desktop, master or slave). For slave programs only, the factory function must also be registered to the DistributedObjectsRegistry, so that the remote instances can be automatically generated (see 4.3).

In general, the definition of application-level distributed classes is discouraged, for the reasons explained in section 2 and because the distribution of the scene graph has exactly the purpose to avoid this approach. There may be cases, however, in which this additional option can result useful.

4.10 Configuration Tools

Virtual reality applications usually require an initial configuration phase in which the employed devices are calibrated and the characteristics / settings of these devices and of the other software components are provided to the application. Since these settings are mostly independent from the specific application that use them, Moka provides a set of classes that can be used to load and save to files several common types of configurations, such as stereo settings, physical characteristics of the displays, network parameters, keystone calibration, simple scene parameters, optional transformations to be applied to the tracker data etc. We also developed some graphical programs to help the user in configuring the VR system. The resulting configuration is saved into a set of predefined files, which can be shared between multiple applications running on the same system.

5 Caffeine

We designed Moka as part of the “Caffeine” project, currently under development at the DreamsLab (Dedicated Research Environment for Advanced Modeling and Simulations) at Scuola Normale Superiore in Pisa. The group merges competences on theoretical and computational chemistry with interactive Virtual Reality technologies, working on the production and fruition of scientific and humanistic contents.

The aim of Caffeine project is to develop an integrated system for computational chemistry that will take maximum advantage from VR technologies in order to visualize and model complex molecular structures in a natural and effective way. Although designed for immersive VR systems, versions of Caffeine will be available for various environments, ranging from standard desktop systems to immersive VR environments. At the time of writing Caffeine is at an early alpha stage of development, but a first version of the molecular visualizer is stable enough to be used in our CAVE system (see Fig. 2a). In this scenario, the user can interact with the system by means of a simple application for Android devices we developed.

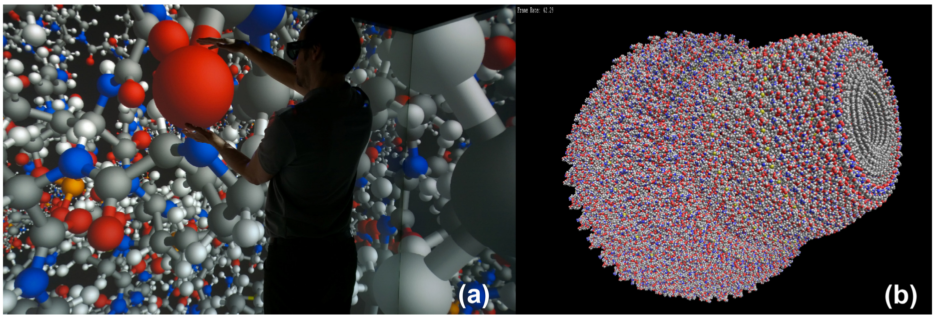


Fig. 2. a) Interacting with a hemoglobin, using Caffeine in our CAVE system. b) Rendering of an half of the Rat Liver Vault [32] molecule using Caffeine.

5.1 GPU-Accelerated Visualization of Molecular Structures

One of the fundamental features that Caffeine had to provide was the ability to visualize "all atoms" representations[3] of complex molecular structures at satisfactory frame rates. To this end we developed a set of shaders implementing a GPU-based ray-casting of implicit surfaces [33, 34], since this method has proven to provide very good results (both in terms of performance and image quality) when applied to the visualization of molecular structures [35–37]. This method consists in generating a simple proxy geometry (e.g. a cuboid or a point sprite) in place of each primitive surface to be drawn. The proxy geometry must enclose the corresponding surface in window space. Then, when the proxy geometry is being rasterized, a fragment shader evaluates the intersection between the surface and a ray starting at the camera position and passing through the center of the generated fragment. The resulting (closer) intersection point and the related normal vector are then used to shade the fragment and adjust its depth. If no intersection is found, the fragment is discarded.

Although a single shader could handle a full class of surfaces (as described in [33, 34, 36] for quadrics), and because we were interested in providing "all-atoms" representations of complex molecular structures, we implemented an optimized version of these shaders for spheres and cylinders. In particular, the proxy geometry is generated on-the-fly by a dedicated geometry shader (similarly to what proposed in [37]) and consists in just a quad for spheres and (at most) two quads for cylinders.

Finally note that since glyphs are common ways to represent scientific data, with Moka we provide specific classes to define sets of spheres and cylinders, in the hope to promote their reuse in other projects. These classes are not part of the scene graph, so they are unknown to the slaves. Instead, they simply encapsulate a SGGeometry node and the shaders implementing the ray-casting algorithm described before.

[3] In this class of representations the atoms are represented as spheres (balls) and the bonds connecting them are depicted as cylinders (sticks). Typical examples are "ball-and-stick", "space filling" and "liquorice" representations, which distinguish themselves according to the radius assigned to the atoms and to the visibility of the bonds.

5.2 A First Qualitative Performance Evaluation

To get a first qualitative evaluation of the performance of our system, we tried to load in Caffeine an half Rat Liver Vault [32] (PDB IDs: 2ZUO, 2ZV4 and 2ZV5), a complex molecule constituted by about 490 thousand atoms and 493 thousand bonds. The test was performed in our CAVE system, constituted by four slave nodes each equipped with 2 Intel Xeon E5645 processors, 24GB of RAM and a Nvidia Quadro 6000 GPU. Each slave drives a stereo projector with a resolution of 1400x1050 pixels.

The molecule was placed in a way to fill the front screen (to maximize the fragment processing workload), while remaining completely visible (to avoid that part of the geometry would be culled/clipped), as shown in figure Fig. 2b. The test was performed with head-tracked stereo enabled and the user was asked to move freely within the CAVE. Since, once finished loading, each slave runs independently from the others, we only measured the frame rate on the front slave.

During the test, we switched between “ball-and-stick” and “space filling” representations: the first produces a greater fill-rate (because of the increased atom’s radius) but does not draw the cylinders, while the second doubles the number of elements to be drawn (because of the cylinders) but less fragments get involved. During the “space filling” test, we got frame rates comprised in the range 30-50 fps. This sensible variation results from the different fill-rates due to the changes of the user’s perspective. The “ball-and-stick” test, instead, provided a more stable frame rate, comprised between 22 and 28 fps. In fact, in this representation, the variation of the fill-rate is more contained. However, the doubled number of elements to be drawn inevitably causes a drop in frame rate.

Although in future we plan to perform a much more accurate and targeted performance evaluation, this first simple test showed encouraging results. In fact, the frame rate remained at interactive levels also when almost two millions quadric surfaces[4] were ray-casted at the same time. We however noted a perceptible lag in the synchronization of the images across the screens when the user moved. This is due to the fact that we decided to not employ any frame-lock mechanism, so the images produced by different slaves may temporary diverge. However, a stringent frame-lock would have caused an inevitable drop in frame rate. Furthermore, these divergences are noticeable only when the system is under heavily stress, like in our test.

6 Conclusions and Future Work

In this work, we proposed a convenient method to develop a lightweight distributed scene graph on top of a 3D graphics engine. In particular, we described “Moka”, a library to simplify the development of distributed VR applications supporting various types of environments: from standard desktop applications to cluster-based multi-screen immersive VR system like the CAVE.

The library is under development and it has been used in the implementation of a molecular visualizer, specifically designed for the research field of our group, solving our

[4] Ball-and-stick representation with “side-by-side” stereoscopy enabled.

needs to visualize structured scientific data in immersive technologies. In the near future, we are planning tests in terms of generic performances, with comparative studies.

In addiction, we are working on the extension of the Moka library both by exposing a much larger set of the features offered by the underlying graphics engine and by supporting some of the newer low-cost VR devices, like the Oculus Rift [26]. We would also like to provide a more comprehensive library for the visualization of scientific data, exploiting GPU-accelerated methods like the GPU-based ray-casting. Finally, we are improving the Caffeine project, with the aim to realize a truly innovative integrated system for computational chemistry, which takes maximum advantage from VR technologies.

References

1. Cruz-Neira, C., Sandin, D.J., DeFanti, T.A.: Surround-screen Projection-based Virtual Reality: The Design and Implementation of the CAVE. In: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, pp. 135–142. ACM, New York (1993)
2. Steed, A., Oliveira, M.F.: Networked Graphics: Building Networked Games and Virtual Environments. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2009)
3. Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P.D., Klosowski, J.T.: Chromium: A Stream-processing Framework for Interactive Rendering on Clusters. In: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, pp. 693–702. ACM, New York (2002)
4. Raffin, B., Soares, L., Ni, T., Ball, R., Schmidt, G.S., Livingston, M.A., Staadt, O.G., May, R.: PC Clusters for Virtual Reality. In: Virtual Reality Conference, pp. 215–222 (2006)
5. Guimarães, M.P., Bressan, P.A., Zuffo, M.K.: Frame lock synchronization for multiprojection immersive environments based on pc graphics clusters. In: Proocedings of the 5th SBC Symposium on Virtual Reality (2002)
6. RakNet 4. http://www.jenkinssoftware.com/
7. ReplicaNet. http://www.replicanet.com/
8. Internet Communications Engine (Ice). http://www.zeroc.com/
9. Collage. http://www.libcollage.net/
10. Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., Cruz-Neira, C.: VR Juggler: a virtual platform for virtual reality application development. In: 2001 Proceedings of the IEEE Virtual Reality, pp. 89–96 (2001)
11. Allard, J., Gouranton, V., Lecointre, L., Melin, E., Raffin, B.: Net Juggler and SoftGenLock: Running VR Juggler with Active Stereo and Multiple Displays on a Commodity Component Cluster. In: Proceeding of IEEE Virtual Reality Conference 2002, pp. 273–274 (2002)
12. Bierbaum, Aron, Hartling, Patrick, Morillo, Pedro, Cruz-Neira, Carolina: Implementing Immersive Clustering with VR Juggler. In: Gervasi, Osvaldo, Gavrilova, Marina L., Kumar, Vipin, Laganá, Antonio, Lee, Heow Pueh, Mun, Youngsong, Taniar, David, Tan, Chih Jeng Kenneth (eds.) ICCSA 2005. LNCS, vol. 3482, pp. 1119–1128. Springer, Heidelberg (2005)
13. VR Juggler: The Programmer's Guide - Version 3.0. http://vrjuggler.org
14. Humphreys, G., Eldridge, M., Buck, I., Stoll, G., Everett, M., Hanrahan, P.: WireGL: A Scalable Graphics System for Clusters. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 129–140. ACM, New York (2001)

15. Carrozzino, M., Tecchia, F., Bacinelli, S., Cappelletti, C., Bergamasco, M.: Lowering the Development Time of Multimodal Interactive Application: The Real-life Experience of the XVR Project. In: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, pp. 270–273. ACM, New York (2005)
16. Hesina, G., Schmalstieg, D., Furhmann, A., Purgathofer, W.: Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology. pp. 74–81. ACM, New York (1999)
17. Kuck, R., Wind, J., Riege, K., Bogen, M.: Improving the AVANGO VR/AR Framework - Lessons Learned. Virtuelle und Erweiterte Realität, 209–220 (2008)
18. Reiners, D.: OpenSG: A scene graph system for flexible and efficient realtime rendering for virtual and augmented reality applications (2002)
19. Roth, M., Voss, G., Reiners, D.: Multi-threading and clustering for scene graph systems. Computers & Graphics. **28**, 63–66 (2004)
20. Schaeffer, B., Goudeseune, C.: Syzygy: native PC cluster VR. In: 2003 Proceedings of the IEEE Virtual Reality, pp. 15–22 (2003)
21. UNIGINE Corp.: Unigine Engine. http://unigine.com
22. i'm in VR: MiddleVR for Unity. http://www.imin-vr.com
23. Unity Technologies: Unity. http://unity3d.com
24. OpenSceneGraph. Version 3.1. http://www.openscenegraph.org
25. OGRE (Object-Oriented Graphics Rendering Engine). http://www.ogre3d.org
26. Oculus VR Inc.: Oculus Rift - Virtual Reality Headset for 3D Gaming. http://www.oculusvr.com/
27. Qt Project. http://qt-project.org/
28. Salzman, L.: ENet Reliable UDP networking library
29. G-Truc Creation: OpenGL Mathematics (GLM). http://glm.g-truc.net/
30. Cube. http://cubeengine.com
31. Kooima, R.: Generalized Perspective Projection (2008). http://csc.lsu.edu/~kooima/articles/genperspective/
32. Tanaka, H., Kato, K., Yamashita, E., Sumizawa, T., Zhou, Y., Yao, M., Iwasaki, K., Yoshimura, M., Tsukihara, T.: The Structure of Rat Liver Vault at 3.5 Angstrom Resolution. Science **323**, 384–388 (2009)
33. Toledo, R., Levy, B.: Extending the graphic pipeline with new GPU-accelerated primitives. In: 24th International gOcad Meeting, Nancy, France (2004)
34. Sigg, C., Weyrich, T., Botsch, M., Gross, M.: GPU-based ray-casting of quadratic surfaces. In: Proceedings of the 3rd Eurographics/IEEE VGTC Conference on Point-Based Graphics, pp. 59–65. Eurographics Association (2006)
35. Tarini, M., Cignoni, P., Montani, C.: Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization. IEEE Transactions on Visualization and Computer Graphics **12**, 1237–1244 (2006)
36. Chavent, M., Vanel, A., Tek, A., Levy, B., Robert, S., Raffin, B., Baaden, M.: GPU-accelerated atom and dynamic bond visualization using hyperballs: A unified algorithm for balls, sticks, and hyperboloids. Journal of Computational Chemistry **32**, 2924–2935 (2011)
37. Bagur, P.D., Shivashankar, N., Natarajan, V.: Improved Quadric Surface Impostors for Large Bio-molecular Visualization. In: Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing, pp. 33:1–33:8. ACM, New York (2012)

5 The Caffeine molecular viewer

5.1 Introduction

This chapter describes the features of Caffeine from the user's point of view.

Caffeine allows to visualize both static and dynamic (trajectories) molecular structures according to the most diffused graphical representations: "*Space Filling*", "*Balls & Sticks*", "*Liquorice*" and "*Ribbons*". Caffeine can also visualize isosurfaces extracted from volume data sets, such as molecular orbitals and electron densities.

Caffeine has been developed in two versions, sharing the same functionalities: one for desktop computers (Windows and Linux operating systems) and one for CAVE-like systems equipped with multiple stereo monitors and a tracking system. In the following, the Desktop version is introduced first, describing in detail all the functionalities it provides. Then the CAVE version will be presented, outlining the differences and the additional features with respect to the desktop version.

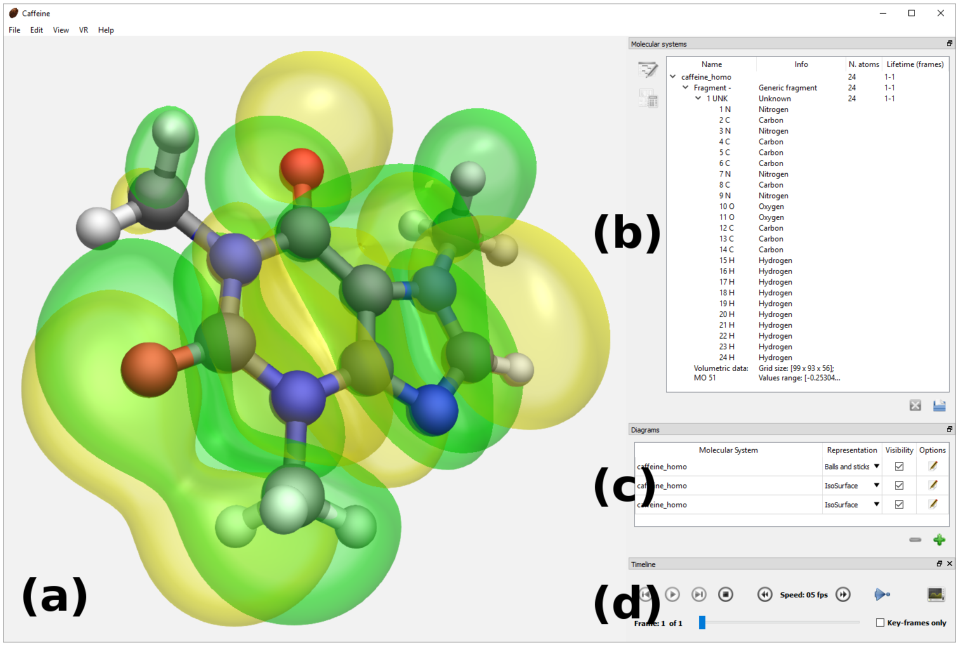
5.2 Development tools

Caffeine has been developed using the C++ programming language; C++ is probably the most widespread programming language in the field of 3D interactive computer graphics, thanks to the good performance of its compiled executable binaries. Caffeine also makes an extensive use of the Qt framework [89] for a vast variety of tasks, such as creating Graphical User Interfaces (GUI), concurrent programming, data structures, 3D graphics-oriented mathematics and so on. Qt provides a comprehensive and *cross-platform* software library that simplifies the development of many tasks not covered by the C++ standard library, such as creating GUI, network programming, database access, OpenGL programming etc. Thanks to its numerous features and its affordable software licenses (it is released under both commercial and open-source licenses), it is probably the most widespread framework for the development of cross-platform graphical applications in C++, both on desktop and embedded platforms. Besides the standard Qt framework, a couple of Qt-based third-part libraries have been employed: Qt Widgets for Technical Applications (Qwt) [264] to plot 2D line chars (see section 5.3.10), and the QxtNetwork module of the Qxt library [265] to implement the communication protocol between the Caffeine process running in a CAVE-like installation and its remote controller application (see section 5.4). The CAVE version of Caffeine employs the NatNet SDK [266] to receive the data sent by the OptiTrack motion capture system [267] of the CAVE installation at Scuola Normale Superiore. The rendering of the 3D scene is made possible by OpenSceneGraph [262], an open source high performance OpenGL-based graphics engine used in a wide range of applications, ranging from games to scientific visualization. Caffeine exploits Open Babel [20] as cheminformatics library, e.g. to read molecular file, detect bonds, add implicit hydrogen atoms and so on. Finally, the icons used in Caffeine belongs to the Oxygen Project [268]. Previous versions made use of the excellent OpenGL Mathematics (GLM) library [269] for graphics-oriented mathematics. However, since latest Qt releases already comes with a copious 3D graphics-oriented mathematics library, and in order to minimize the dependencies of the project, the dependency from GLM has been removed in the latest version of the project.

5.3 Caffeine “desktop” version

5.3.1 Graphical interface overview

Figure 5.1 shows the main window of Caffeine. The rendering of the molecular systems and their associated properties is displayed in the central area while, on the right side, three panels give access to most of the available features. These panels can be rearranged as desired, e.g by snapping them to the other sides of the window, detaching them from the main window or closed. Panel (b) in Figure 5.1 lists the loaded molecular systems. A tree structure allows to inspect the atoms, residues and fragments composing each molecular system. If one or more volume data set are associated to a molecular system, these are listed at the end of the description of the molecule, reporting the name of the data set, the size of the three-dimensional grid and the minimum and maximum values contained in the grid. The buttons on the lower-right corner of the panel allow to load a molecule from file or to delete a previously loaded one. On the top-left corner, two buttons are provided respectively to manage the “*key-frames*” defined for a trajectory and to manage additional scalar data sets associated to the molecular systems. These features will be described in detail in the following sections. Panel (c) shows the list of the generated “*diagrams*”. The term “*diagram*” refers to the graphical depiction of a molecular system, a subset of it, or an associated data set, according to a specified representation (e.g. “*Ball & Stick*”, “*Ribbons*”, “*Isosurface*” and so on). Of course it is possible to generate new diagrams, delete existing ones, as well as hide diagrams without deleting them. For each diagram it is also possible to set various settings, e.g. selecting a subset of fragments to be visualized or specifying various graphical options. Finally, from panel (d) it is possible to control the playback of a trajectory and to visualize line charts representing real functions of a real variable associated to the visualized molecular systems.



Name	Info	N. atoms	Lifetime (frames)
caffeine_homo		24	1-1
Fragment -	Generic fragment	24	1-1
1 UNK	Unknown	24	1-1
1 N	Nitrogen		
2 C	Carbon		
3 N	Nitrogen		
4 C	Carbon		
5 C	Carbon		
6 C	Carbon		
7 N	Nitrogen		
8 C	Carbon		
9 N	Nitrogen		
10 O	Oxygen		
11 O	Oxygen		
12 C	Carbon		
13 C	Carbon		
14 C	Carbon		
15 H	Hydrogen		
16 H	Hydrogen		
17 H	Hydrogen		
18 H	Hydrogen		
19 H	Hydrogen		
20 H	Hydrogen		
21 H	Hydrogen		
22 H	Hydrogen		
23 H	Hydrogen		
24 H	Hydrogen		
Volumetric data: MO 51	Grid size: [99 x 93 x 56]; Values range: [-0.25304...		

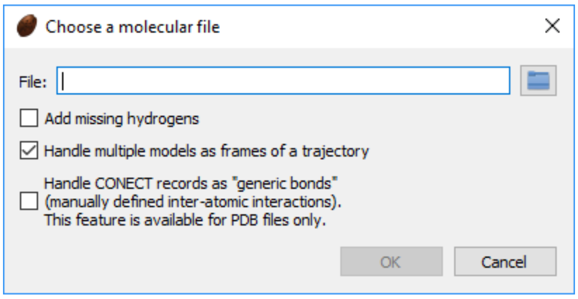
Molecular System	Representation	Visibility	Options
caffeine_homo	Balls and sticks	✓	
caffeine_homo	IsoSurface	✓	
caffeine_homo	IsoSurface	✓	

Figure 5.1: Main window of the desktop version of Caffeine showing a molecular orbital of a caffeine molecule. (a) Rendering area. (b) List of the loaded molecular systems. (c) List of the generated “diagrams”. (d) Trajectories control panel.

5.3.2 Loading a molecular system from file

It is possible to load a molecular file by clicking on the "open" button placed at the bottom-right corner of the (b) panel of Figure 5.1. Alternatively, select the menu item *File* → *Open* or type the associated keystroke *Ctrl+O*. A dialog will show up allowing to select a molecular file and (Figure 5.2).Once loaded, the new molecule will be displayed in "*Space filling*" representation (i.e. a "Space filling" diagram is automatically generated for the molecule).

Open Babel [20] is used to parse the molecular files, detect bonds and to insert the missing hydrogens. Secondary structures of polypeptides are computed by means of Stride [270] (invoked by Caffeine as an external program). The following molecular file formats are supported: *Protein Data Bank* (PDB) [19], *XMol XYZ* [271] and *Gaussian Cube* [272]. It is important to note that the availability of some features depends on the file format from which the molecular system is loaded. In particular, the PDB is the only file format (among those supported) providing topological information, therefore ribbon representations will be available only for this type of files. On the other way, the Gaussian Cube is the only one (among those supported) designed to store volume data. Concerning trajectories, although they could be stored both on PDB or XYZ files, Caffeine can import them only from PDB files at the moment. Actually, neither PDB or XYZ file formats are suited to store large amounts of data; support of compressed file formats will be considered in future.


Choose a molecular file

File:

☐ Add missing hydrogens

☑ Handle multiple models as frames of a trajectory

☐ Handle CONECT records as "generic bonds" (manually defined inter-atomic interactions). This feature is available for PDB files only.

Figure 5.2: Open molecular file dialog.

When loading a molecule from file, some options are provided about how to interpret some class of information or about possible modifications to be applied to the molecular system. First of all, if the file does not contains hydrogen atoms, these can be added to the structure by means of a dedicated algorithm provided by OpenBabel. Furthermore, in the case of molecular files containing multiple structures, it is possible to specify if these must be considered multiple conformations of a same structure (i.e. frames of a trajectory) or separate systems. Finally, in the case of PDB files, CONECT records can be interpreted as user-defined inter-atomic interactions (instead of covalent bonds). These “generic bonds” will be visualized in ball-and-stick representation as dashed lines. This feature can be used, for example, to visualize pre-computed (outside Caffeine) hydrogen bonds, as shown in Figure 5.3.

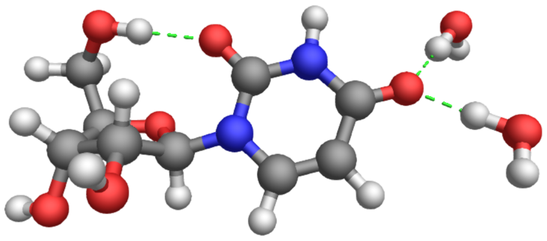


Figure 5.3: Hydrogen bonds between a nucleotide and two water molecules. An intra-molecular hydrogen bond within the nucleotide can also be noted. All the hydrogen bonds are provided as input to Caffeine by means of user-defined CONECT records.

5.3.3 Managing diagrams

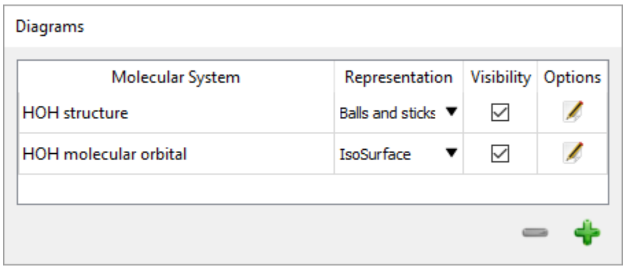
Figure 5.4 shows a detailed view the panel for the management of diagrams. This panel is organized as a table, with rows representing the created diagrams. The columns have the following purposes:

Name of the diagram By default a diagram is named as the molecular system it represents. However, by double-clicking on this field, it is possible to assign a custom (and more informative) name to the diagram.

Representation Combo-box by means of which it is possible to select the representation of the diagram. Available representations are: “*Space Filling*”, “*Balls & Sticks*”, “*Liquorice*”, “*Ribbons*” and “*Isosurface*”. Note that some representations may result in an empty diagram (nothing is shown) if the molecular system does not contains the data required by the selected representation (e.g. “*Ribbons*” only applies to polypeptides and polynucleotides, while the “*Isosurface*” requires one or more volume data sets).

Visibility By acting on this check-box it is possible to temporarily hide the content of a diagram.

Options By clicking on this buttons a configuration window appears, by means of which the user can set various options related to the diagram. These options will be discussed in detail in the following sub-sections.


Diagrams

Molecular System	Representation	Visibility	Options
HOH structure	Balls and sticks ▼	✓	✎
HOH molecular orbital	IsoSurface ▼	✓	✎

Figure 5.4: Diagrams management panel.

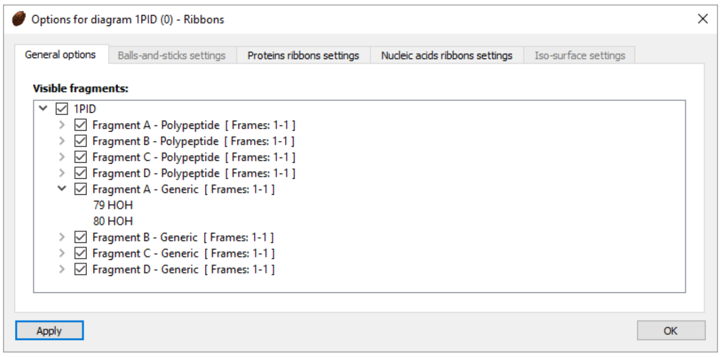
To delete a diagram it is sufficient to select it and click on the "-" button. The "+" button allows instead to create a new diagram. A small dialog appears by means of which the user can specify the molecular system to be depicted and the initial representation. The user can also asks to configure the options of the diagram before its creation.

5.3.4 Filter visible fragments in a diagram

A first category of options concerns the selection of a subset of fragments of a molecular system to be depicted by the diagram. The user interface by means of which the user can filter the fragments is shown in Figure 5.5. The list of fragments composing the molecular system is showed as a tree structure whose items can be enabled or disabled. At present, the filtering is possible only at the fragments level, so it is not possible to show or hide only specific residues. A more flexible and fine-grained filtering policy has been planned in future developments.

5.3.5 Atomistic representations

In atomistic representations ("space filling", "ball-and-stick" and "liquorice"), atoms can be colored according to the most commonly used color schemes: "by element", "by residue" and "by fragment". In addiction, is it possible to set a custom color chosen by the user. From the options dialog is also possible to show or hide the user-defined "generic bonds" read from the CONECT records of PDB files.


Options for diagram 1PID (0) - Ribbons
General options
Balls-and-sticks settings
Proteins ribbons settings
Nucleic acids ribbons settings
Iso-surface settings
Apply
OK

Visible fragments:

- [✓] 1PID
  - [✓] Fragment A - Polypeptide [Frames: 1-1]
  - [✓] Fragment B - Polypeptide [Frames: 1-1]
  - [✓] Fragment C - Polypeptide [Frames: 1-1]
  - [✓] Fragment D - Polypeptide [Frames: 1-1]
  - [✓] Fragment A - Generic [Frames: 1-1]
    - 79 HOH
    - 80 HOH
  - [✓] Fragment B - Generic [Frames: 1-1]
  - [✓] Fragment C - Generic [Frames: 1-1]
  - [✓] Fragment D - Generic [Frames: 1-1]

Figure 5.5: User interface to select a subset of fragments of the molecular system to be depicted by the diagram. In this example, the examined molecule is a bovine despentapeptide insulin (PDB ID: 1PID, [273]). Note how water molecules are separated by polypeptides: the residues which are not part of a polypeptide or a polynucleotide are inserted in their own fragments, so to facilitate their filtering.

A peculiar feature of Caffeine, employed in balls-and-sticks and liquorice representations, consists in highlighting the bonds between different fragments by coloring them in purple, as shown in Figure 5.6.

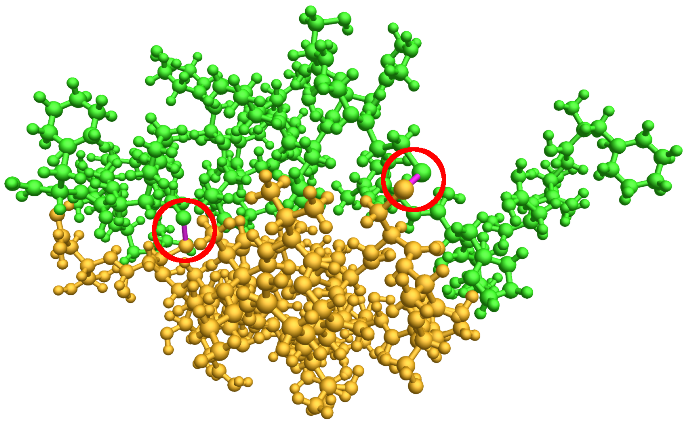


Figure 5.6: Two polypeptide chains of a bovine despentapeptide insulin molecule (PDB ID: 1PID, [273]) represented as balls-and-sticks and colored by chain. The covalent bonds connecting different fragments, colored in purple, are highlighted.

Finally, it is possible to visualize additional information about a molecular systems by means of small green spheres called “*dummy atoms*”. In order to draw such spheres, the user must manually insert in the molecular file a “dummy” atom named “X” with proper coordinates for each sphere to be drawn. Figure 5.7 shows an example of the use of “*dummy atoms*”.

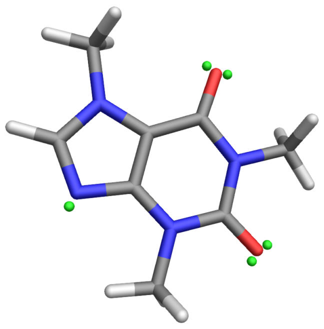


Figure 5.7: Caffeine molecule structure. Lone pair electrons on sp^2 nitrogen and oxygen atoms are visualized as "*dummy atoms*" (small green spheres).

5.3.6 Ribbons diagrams

Caffeine offers many customizations for drawing ribbons diagrams of polypeptides an polynucleotides. First of all, it allows to draw ribbons representing polypeptides according to the most used color schemes, such as by secondary structure, by residue, by chain or using custom colors (see Figure 5.8).

Furthermore, the choice of the color scheme and of the size of the ribbon happens on a per-chain basis, thus allowing to specify different graphical settings for each chain. Finally, the user can highlight one or more sequences of residues within a chain by specifying a custom color scheme and size for them. An example is shown in Figure 5.9, depicting a modified Mechanosensitive Channel of Large Conductance (MscL). MscL is a membrane protein that activates upon sudden changes in membrane tension, creating a large transient and non selective pore to counter osmotic downshock. Due to its ability to react to mechanical stimuli, engineered MscL proteins responsive to pH or light have been prepared to study their possible application as controllable nanovalves. MscL is constituted by five identical trans-membrane domains made up by helices TM1 and TM2. The starting conformational changes induced in MscL have been previously studied by MD [274]. In particular, the effect of creating from one to five equal charges (caused by light activation of mutated residues) into the pore lumen was analyzed. One of the main results of the study was the elucidation of a fast asymmetric sub-unit movement upon single-charge incorporation into the pore, taking place in the ns timescale. Figure 5.9 shows the structure of the MscL including a single light activated charge, where the mutated residue is highlighted by means of an enlarged ribbon colored in azure.

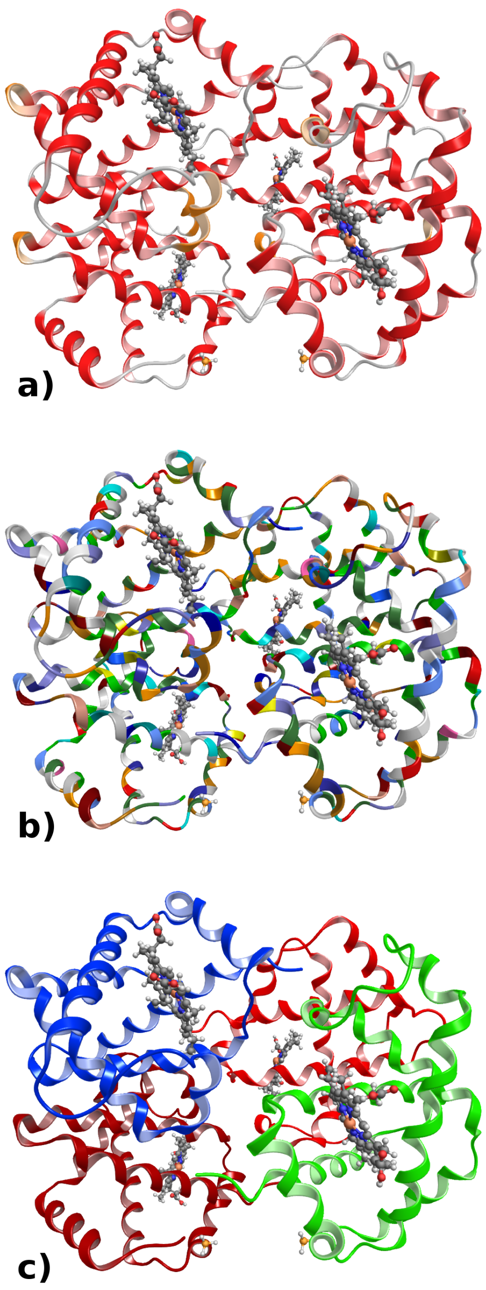
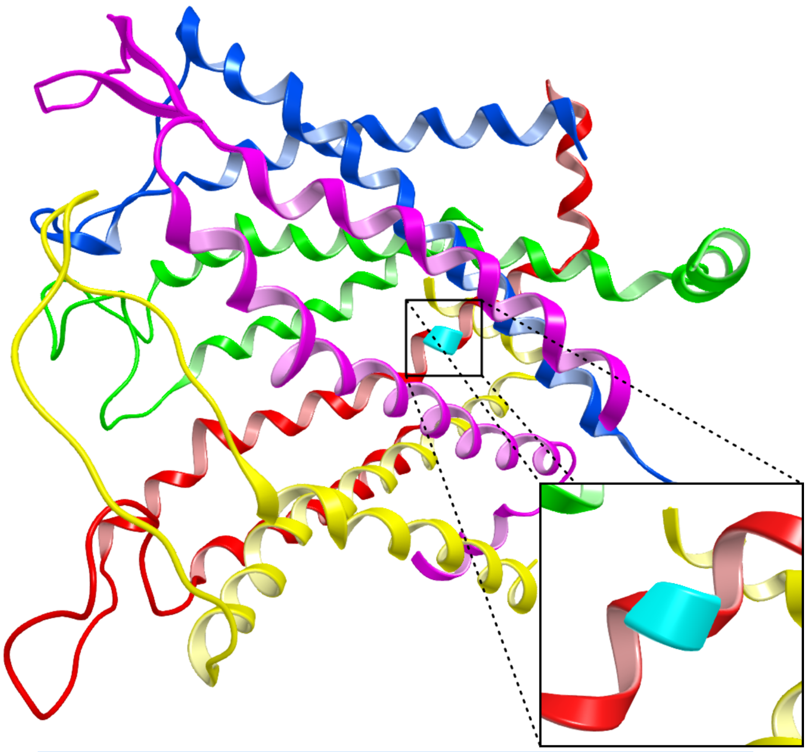
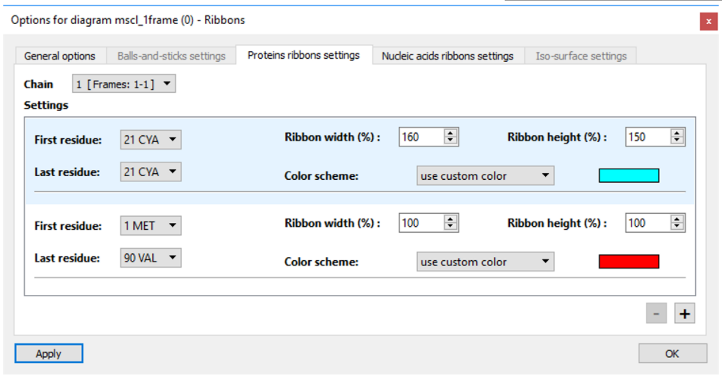


Figure 5.8: Structure of a human deoxy-hemoglobin (PDB ID: 2HHB, [120]). The four polypeptide chains are represented as ribbons, while heme groups are visualized as balls and sticks. In the three sub-figures, ribbons are drawn according to different color schemes: a) by secondary structure; b) by residue; c) by chain’s ID.





Options for diagram mscl_1frame (0) - Ribbons

General options | Balls-and-sticks settings | Proteins ribbons settings | Nucleic acids ribbons settings | Iso-surface settings

Chain: 1 [Frames: 1-1]

Settings

First residue:	21 CYA	Ribbon width (%) :	160	Ribbon height (%) :	150
Last residue:	21 CYA	Color scheme:	use custom color	🟦	
First residue:	1 MET	Ribbon width (%) :	100	Ribbon height (%) :	100
Last residue:	90 VAL	Color scheme:	use custom color	🟥	

Figure 5.9: MscL structure engineered by Chandramouli et al. [274] is shown using ribbons representation. The modified helix including the light activated charge is colored in red and its mutated residue is highlighted by means of an enlarged ribbon colored in azure. The graphical options used to draw the modified chain are shown on the bottom of the figure.

As regard to helices, it can be noted that inner and outer sides are colored with different colors. This feature, provided also by some other molecular visualizers such as Chimera [72] and YASARA [87], helps the user to better understand the spatial structure of the helix. In addiction, this feature is also exploited to visually indicate the handedness of the helix: right-handed helices have a lighter color on the internal side, while left-handed helices are drawn with the lighter color on the outer side (see Figure 5.10). Even if they are uncommon, left-handed helices are actually observed in crystallography [275], therefore detection of handedness is an important feature. Finally note that, in order to be correctly detected, left-handed helices must be composed by at least 4 residues.

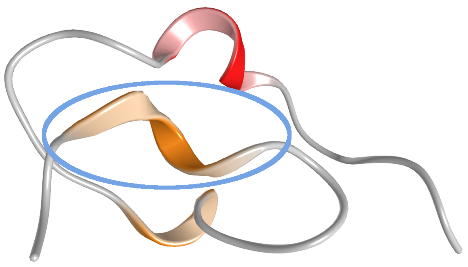
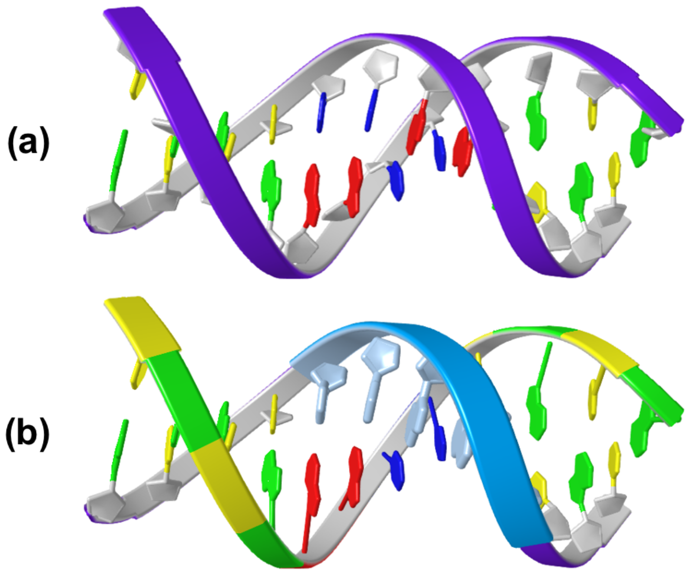


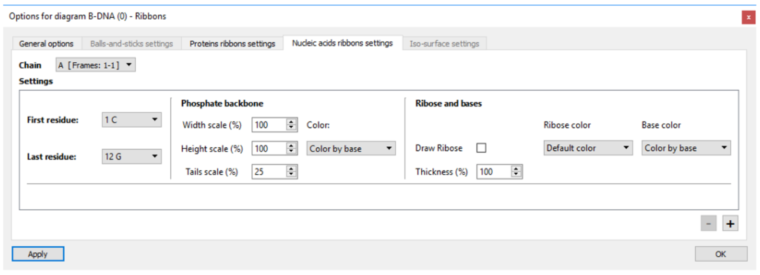
Figure 5.10: Ribbons visualization of a prototype LNR module from human Notch1 (PDB ID: 1PB5, [276]). In the figure a left-handed 3_{10} helix is highlighted. Note that Caffeine draws right-handed helices with lighter color on the internal side while left-handed helices are drawn with a lighter color on the external side.

Figure 5.11 shown an example of ribbons diagram for polynucleotides and the related available options. Many possible customizations are provided, like the ability to assign custom color schemes (default, by base, by fragment and custom color) and sizes to the various elements of the diagram (phosphate backbone, [deoxy]ribose and nucleobases) and to hide the sugar in order to obtain a simplified representation. Like in the case of polypeptides, different settings applies to different chains, and it is possible specify custom graphical properties for specific sequences of nucleotides within the chain. It should be noted that the traits of the ribbon corresponding to the first and the last nucleotides of each chain are, respectively, wider and narrower than the rest of the ribbon. This feature has the purpose to highlight the 5’ to 3’ direction of the chain and, to the best of my knowledge, it’s a unique feature of Caffeine.

Finally, Caffeine allow to select between three different quality levels for the ribbons geometry, allowing to obtain better performance on old hardware by setting an inferior graphics quality. The desired quality level can be selected by the *Edit* →

Diagrams quality menu, and applies to all the ribbons diagrams.





Options for diagram B-DNA (0) - Ribbons

General options | Balls-and-sticks settings | Proteins ribbons settings | Nucleic acids ribbons settings | Iso-surface settings

Chain: A [Frames: 1-1]

Settings

First residue:	1 C
Last residue:	12 G

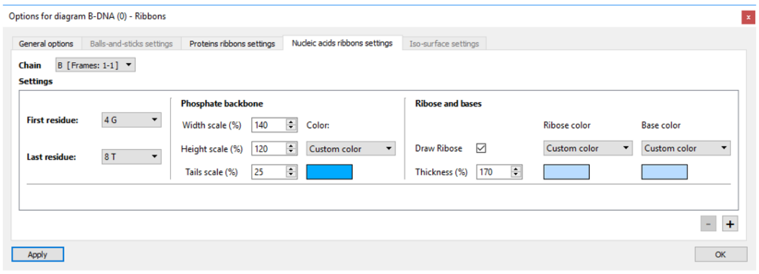
Phosphate backbone

Width scale (%)	100
Height scale (%)	100
Tails scale (%)	25
Color:	Color by base

Ribose and bases

Draw Ribose	☐
Thickness (%)	100
Ribose color	Default color
Base color	Color by base

Apply | OK



Options for diagram B-DNA (0) - Ribbons

General options | Balls-and-sticks settings | Proteins ribbons settings | Nucleic acids ribbons settings | Iso-surface settings

Chain: B [Frames: 1-1]

Settings

First residue:	4 G
Last residue:	8 T

Phosphate backbone

Width scale (%)	140
Height scale (%)	120
Tails scale (%)	25
Color:	Custom color 🟦

Ribose and bases

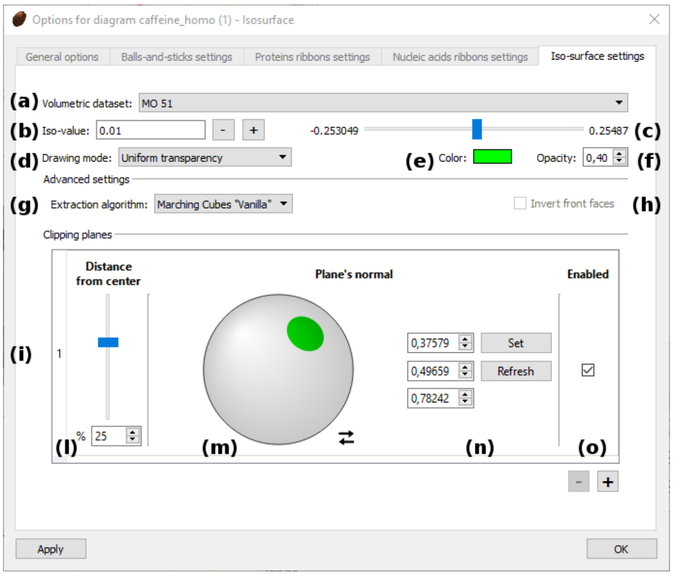
Draw Ribose	☑
Thickness (%)	170
Ribose color	Custom color 🟦
Base color	Custom color 🟦

Apply | OK

Figure 5.11: (a) Ribbons representation of a fragment of B-form DNA with default graphical settings. (b) Same as (a) but applying custom graphical settings to the two chains: the backbone of chain A is colored by base and the pentagon representing the deoxyribose has been replaced with a cylinder spanning from C3' to the nucleobase, thus obtaining a simpler representation. Chain B, instead, has been depicted with default settings, apart from the nucleotides 4-8, which has been drawn enlarged and with custom colors (two shades of light blue). A detailed view of the settings required to obtain figure (b) is shown on the bottom half of the image.

5.3.7 Isosurface diagrams

Three-dimensional scalar fields related to molecular properties, such as molecular orbitals and electron densities, can be visualized in Caffeine as isosurfaces. At present, volume data sets can be imported only from Gaussian Cube files but, apart from file parsing procedures, Caffeine's code does not make assumptions about the source or the type of these data. Once created an isosurface diagram for a molecular system, the user can click on the “option” button shown in Figure 5.4 to chose among the available volume data sets (there can be multiple volumes associated to the same molecular system) and to select an isovalue. The complete list of options related to isosurface visualization is shown in Figure 5.12.


Options for diagram caffeine_homo (1) - Isosurface

(a) Volumetric dataset: MO 51
(b) Iso-value: 0.01
-0.253049 0.25487 (c)
(d) Drawing mode: Uniform transparency
(e) Color: 🟩
Opacity: 0,40 (f)
Advanced settings
(g) Extraction algorithm: Marching Cubes "Vanilla"
Invert front faces (h)
Clipping planes
(i) 1
(l) % 25
0,37579
0,49659
0,78242
(o) Enabled ✓

Figure 5.12: User interface to configure the visualization of an isosurface. (a) Choice of the volume data set. (b)(c) The isovalue can be selected by directly inserting a value in the text field or by acting on the provided buttons and slider. (d) Modality according to which the isosurface must be drawn. (e) Color of the resulting isosurface. (f) Opacity of the isosurface (applies to “uniform transparency” drawing mode only). (g) The algorithm used to generate a tessellation of the isosurface. (h) When a “solid” drawing mode is selected, this check-box allows to invert the “external”/“internal” sides of the surface (the two sides are drawn with different styles). (i) List of “clipping planes” cutting the isosurface. Multiple clipping planes can be defined for a single isosurface. (l) Distance of the clipping plane from the center of the volume. (m) Widget to select a normal vector for the plane in a graphical way. (n) Numerical components of the currently selected normal vector. The user can set the normal vector by inserting its components in the spin-boxes and by pressing the “Set” button. (o) Enables or disables the clipping plane.

Several drawing modes for isosurfaces are available, the most important of which are shown in Figure 5.13. Panels (b) and (c) of Figure 5.13 point out the two different transparency modes provided by Caffeine: "*uniform transparency*" and "*smart transparency*". Uniform transparency simply assign a fixed (user adjustable) opacity to the entire surface. The so called "smart" transparency, instead, assign a different opacity value to the fragments (proto-pixels see section 1.3) resulting from the rasterization of the surface, according to the angle between the normal to the surface and the view direction. It follows that the zones in which the view direction is tangent to the surface are drawn with an higher opacity than those orthogonal to the view direction, thus highlighting the contours of the surface while clearly showing its inner content. The result is a sharper and more understandable image with respect to uniform transparency. Actually this technique, often called "view-dependent transparency" or "X-Ray effect", is a well-known graphics programming "trick" exploited also by other molecular viewers, such as *Molekel* [93] and *Avogadro* [86]. A discussion about the techniques to emulate semitransparent materials in real-time computer graphics and details about its implementation in Caffeine is given in section 6.5.

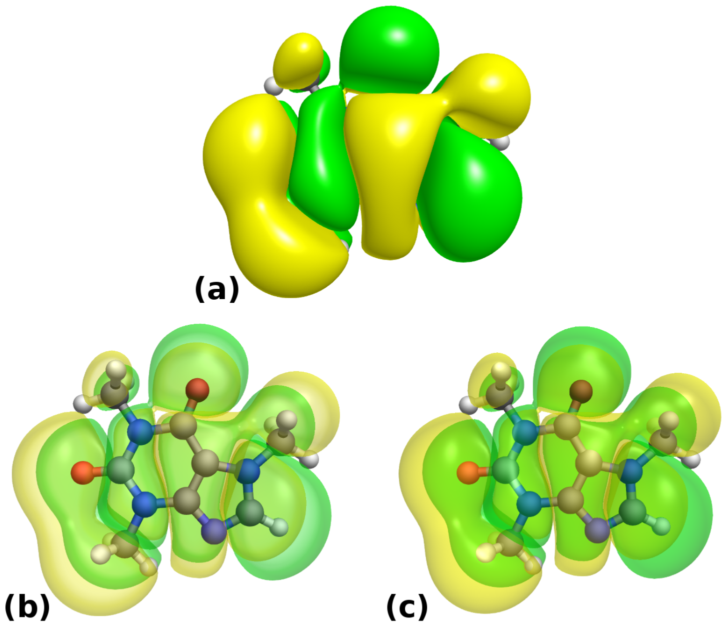


Figure 5.13: Highest Occupied Molecular Orbital (HOMO) of caffeine molecule visualized with different drawing modes: (a) "*solid*"; (b) "*smart transparency*"; (c) "*uniform transparency*". Note that, to visualize the entire orbital, two isosurface diagrams must be created, one with a positive isovalue and the other with the corresponding negative isovalue.

Caffeine provides two different algorithms to generate an isosurface for a given volume data set:

1. The classic “*Marching Cubes*” [138]: The popular implementation by Paul Bourke and Cory Gene Bloyd [277] has been used.
2. A simplified variant of the “*Surface Nets*” [278]: The “*Naive Surface Nets*” by Mikola Lysenko [279] (originally coded in JavaScript) has been re-implemented in C++ so to be employed in Caffeine.

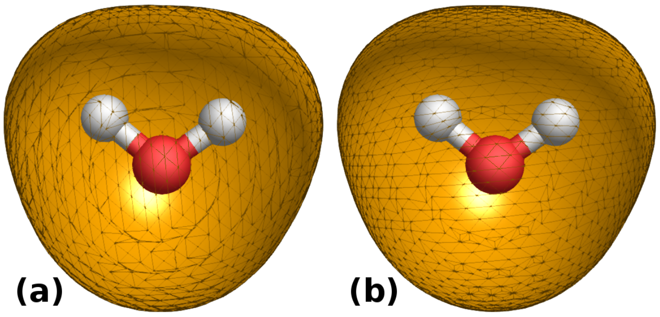


Figure 5.14: Electron density of a water molecule represented by an isosurface. Comparison between the triangulation generated by the Marching Cubes algorithm (a) and the one generated by the Naive Surface Nets algorithm (b).

The classic Marching Cubes provides a more accurate approximation of the isosurface and, for that reason, it has been chosen as the default isosurface extraction algorithm of Caffeine. The (Naive) Surface Nets (NSN), however, exhibits several interesting properties. In particular, as shown in Figure 5.14, NSN produces a more regular triangulation of the isosurface, as opposed to the Marching Cubes whose generated geometry often contains many small, thin triangles. Furthermore, according to some tests performed to compare the two algorithms included in Caffeine, the Naive Surface Nets is slightly faster in constructing the isosurface (see chapter 7 for details). Note that a low construction time is crucial to give the user the possibility to interactively change the isovalue. In conclusion, however, the NSN did not provided enough benefits to be chosen as default extraction algorithm in place of the Marching Cubes.

Caffeine allows to define one or more *clipping planes* cutting the isosurfaces. This feature is useful to visualize multiple nested isosurfaces at the same time, in order to simulate contour plots (see Figure 5.15). Another example is the case of symmetric structures, where clipping planes can be placed along the axis of symmetry to visualize isosurfaces related to different quantities in the same picture.

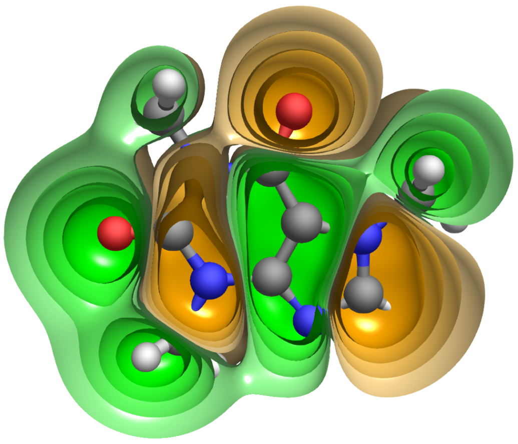
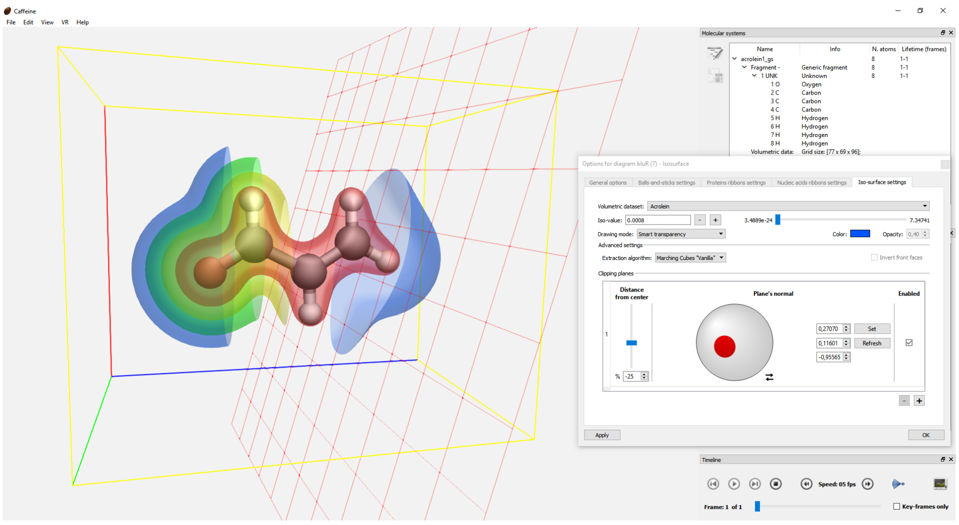


Figure 5.15: Highest Occupied Molecular Orbital (HOMO) of a caffeine molecule visualized as nested, clipped, isosurfaces.

Each clipping plane is defined by means of a normal vector, a reference point and the distance between the plane and the reference point. The reference point is fixed and placed at the center of the volume. The user interface provided by Caffeine to define clipping planes is shown in Figure 5.12(i). In order to help the user to orientate the plane (i.e. to define a proper normal vector), a custom widget (GUI control) has been implemented (Figure 5.12(m)). In this widget, the normal vector is depicted as a point lying on a unit sphere. The user can vary the normal by simply dragging the cursor over the sphere. Since the widget represents only a half of the unit sphere, a special button (having two opposing arrows as icon) is provided to switch between the front and back sides of the sphere. The currently selected side is indicated by the color of the cursor: a green cursor symbolize the front side (those having positive coordinates along the Z axis), while a red cursor symbolize the back side. When editing the clipping planes for a given isosurface, some helpers are drawn on the 3D scene, representing the bounding box of the volume data, the local reference frame, and the clipping plane(s) of the isosurface being edited (see Figure 5.16). Note that there is a limit on the number of clipping planes that can be defined, regardless of how they are distributed across the isosurface diagrams. This limit is imposed by OpenGL, and the actual number of available clipping planes depends on the graphics hardware. Typical values are 6 or 8.



Name	Info	N. atoms	Lifetime (frames)
acrolein1_gs		8	1-1
Fragment -	Generic fragment	8	1-1
1 UNK	Unknown	8	1-1
1 O	Oxygen		
2 C	Carbon		
3 C	Carbon		
4 C	Carbon		
5 H	Hydrogen		
6 H	Hydrogen		
7 H	Hydrogen		
8 H	Hydrogen		
Volumetric data:	Grid size: [77 x 69 x 96];		

Volumetric dataset: Acrolein
Iso-value: 0.0008 3.4889e-24 7.34741
Drawing mode: Smart transparency Color: Opacity: 0,40
Extraction algorithm: Marching Cubes "Vanilla" Invert front faces

	Distance from center	Plane's normal	Enabled
1	% -25	0,27070 0,11601 -0,95565	☑

Speed: 05 fps
Frame: 1 of 1
Key-frames only

Figure 5.16: Creating a representation of the electron density of an acrolein molecule by means of nested, clipped, isosurfaces. It can be noted that, when editing an isosurface, some helpers are drawn on the 3D scene, such as the bounding box of the volume data, the local reference frame, and the clipping plane(s) of the isosurface being edited.

5.3.8 Playing trajectories

Caffeine is able to load trajectories from multi-model PDB files. In particular, when parsing a PDB file, every time that a model (identified by a “MODEL” record in the file) is encountered (after the first), Caffeine checks if this new molecular system is composed by the same atoms of the previously loaded one. If so, the data of the new model are appended to the previous molecular system, so to form a new frame of a trajectory, otherwise it is treated as a different molecular system. In the case of polypeptides, secondary structures are pre-detected for each frame of the trajectory, so to visualize the formation and the decay of these structures over time.

The playback of the loaded trajectory can be controlled by acting on the panel showed in Figure 5.17. As usual, user can play/stop the playback, change the speed of the playback , jump to a specific frame, repeat the reproduction in a loop, and so on.

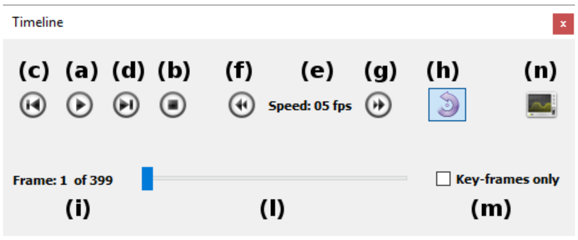


Figure 5.17: Trajectories playback panel. (a) Play/Pause. (b) Stop and rewind. (c) Skip to previous frame. (d) Skip to next frame. (e) Current playback speed expressed in frame per seconds. (f) Slow down the playback. (g) Speed up the playback. (h) Play once / Repeat in a loop. (i) Number of the frame currently played / total number of frames. (l) Timeline showing the frame currently played and allowing to "jump" from a time point to another. (m) Display all frames or only the "*key-frames*". (n) Show / Hide data charts.

Apart from molecular dynamics, it is also possible to visualize chemical reactions, where the connectivity of the system changes in time. In fact, as in the case of secondary structures of polypeptides, bond connectivity is computed (by means of OpenBabel) separately for each frame of the trajectory, so to be able to visualize the formation and the decay of covalent bonds. An example is shown in Figure 5.18, representing 12 states of a S_N2 reaction: $CH_3Br + Cl^- \rightarrow CH_3Cl + Br^-$ (courtesy of Dr. Marco Pagliai).

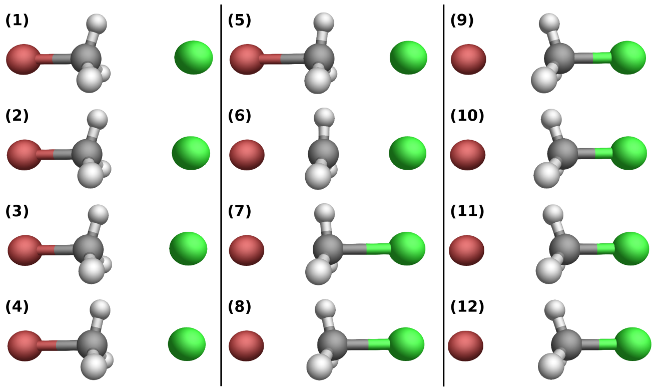
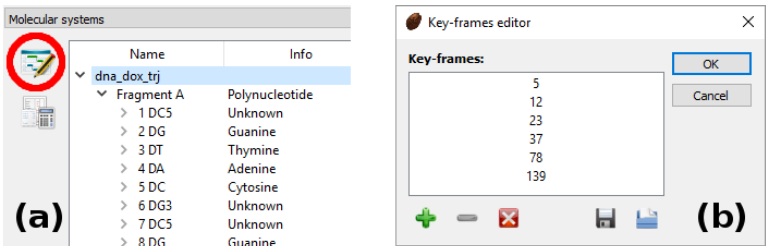


Figure 5.18: Visualization of 12 states of a S_N2 reaction: $CH_3Br + Cl^- \rightarrow CH_3Cl + Br^-$ with Caffeine. Data courtesy of Dr. Marco Pagliai.

If more than one trajectory is loaded, they will be reproduced in parallel. If the trajectories have a different length, the shorter ones will stop on reaching their last frame, while the others will continue their playback. This last rule is also applied if both static and dynamic structures are visualized together: in fact, static structures are managed as dynamic structures composed by one frame.

5.3.9 Key-frames

The term “*key-frames*” refers to a subset of conformations within a single trajectory which are particularly relevant for the study of the system. Caffeine allows to “mark” specific conformations as key-frames as well as save/load them to/from file. Key-frames represent therefore a filtering mechanism for trajectories, allowing the user to visualize only meaningful frames instead of the entire trajectory. In order to do so, it is sufficient to enable the check-box of Figure 5.17(m).



Molecular systems

Name	Info
dna_dox_trj	
Fragment A	Polynucleotide
1 DC5	Unknown
2 DG	Guanine
3 DT	Thymine
4 DA	Adenine
5 DC	Cytosine
6 DG3	Unknown
7 DC5	Unknown
8 DG	Guanine

Key-frames editor

Key-frames:

5
12
23
37
78
139

Figure 5.19: Key-frames editing. By selecting a molecular system and pressing the button highlighted in (a), a new form appears (b) allowing the user to edit, save and load the key-frames.

The set of key-frames for a specific molecular system can be defined, saved to file and loaded from file by means of the dialog showed in Figure 5.19(b), activated when pressing the button highlighted in Figure 5.19(a). The format of the files storing key-frames is extremely simple: these files have “.tkf” as extension and their content is constituted by the list of the indices of the key-frames, one index per line. These indices must be integer numbers in the range [1,N], where N is the total number of frames of the trajectory.

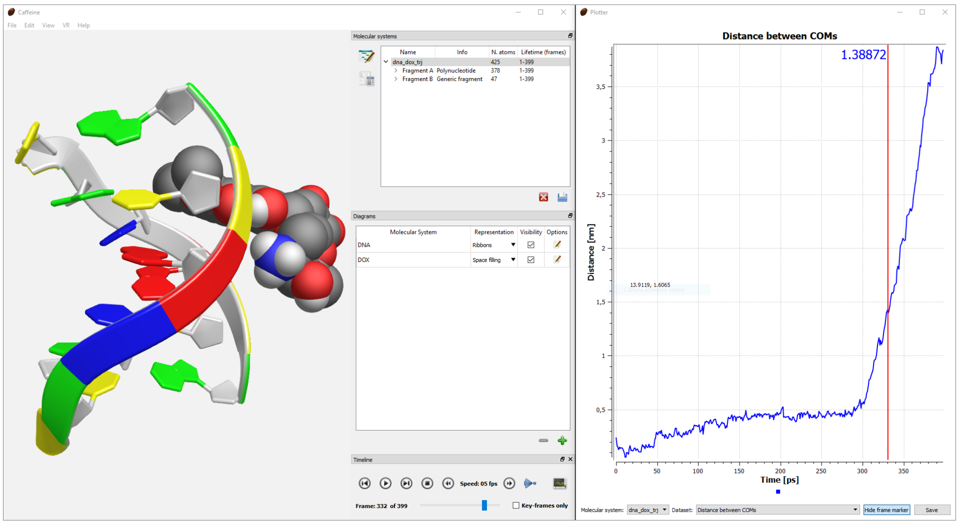
Example of key-frames file

($\backslash n$ represents the "new line" character)

```
1\n
5\n
7\n
12\n
```

5.3.10 Plotting scalar data

Results of simulations of molecular systems does not consist only in a sequence of structures describing the position of the atoms in different conformations or configurations. In fact, they often produce many types of numerical data describing the physicochemical properties of the simulated system. The opportunity to observe at the same time both the conformations and related numerical data, helps the researcher to better understand the behavior of the system under investigation and to foresee its peculiarities. To this end, Caffeine allows to associate one or more data sets to a molecular system and to plot them in charts. At present only two-dimensional scalar data sets are supported, which are interpreted as samples of a real function of a real variable and that can be plotted as line charts. In future developments, however, these functionalities will be extended and generalized, so to support a broader typology of data sets and charts.



Name	Info	N. atoms	Lifetime (frames)
dna_dox_trj		425	1-399
Fragment A	Polynucleotide	378	1-399
Fragment B	Generic fragment	47	1-399

Molecular System	Representation	Visibility	Options
DNA	Ribbons	✓	
DOX	Space filling	✓	

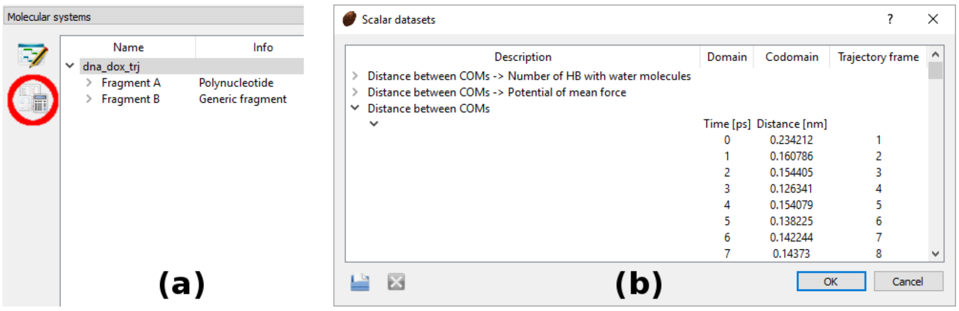
Frame: 332 of 399 ☐ Key-frames only

Molecular system: dna_dox_trj Dataset: Distance between COMs

Figure 5.20: Visualizing the results of a simulation with Caffeine. The simulated phenomena is the unbinding process of a doxorubicin molecule from DNA. On the right, the distance between the centers of mass of the doxorubicin and of the binding site is plotted as a line chart. The red marker shows the distance for the current frame of the trajectory.

An example of this feature is illustrated in Figure 5.20, showing the results of the unbinding process of a doxorubicin molecule (drug acting as inhibitors of Topoisomerase I and/or II) from DNA. The distance between the center of mass of the doxorubicin and the center of mass of the binding site is plotted as a line chart. Furthermore, when a linear relation exists between the conformations of the trajectory and the associated numerical data, a vertical marker appears, pinpointing the value of the scalar quantity associated to the currently visualized frame of the trajectory. Further details on this simulation can be found in chapter 7.

Scalar data sets to be plotted can be loaded by means of the dialog showed in



Name	Info
dna_dox_trj	
Fragment A	Polynucleotide
Fragment B	Generic fragment

Description	Domain	Codomain	Trajectory frame
Distance between COMs -> Number of HB with water molecules			
Distance between COMs -> Potential of mean force			
Distance between COMs			

Time [ps]	Distance [nm]	Trajectory frame
0	0.234212	1
1	0.160786	2
2	0.154405	3
3	0.126341	4
4	0.154079	5
5	0.138225	6
6	0.142244	7
7	0.14373	8

Figure 5.21: Loading numerical data associated to a molecular system. By selecting a molecular system and pressing the button highlighted in (a), a new dialog appears (b), allowing the user to load one or more two-dimensional scalar data sets.

Figure 5.21(b), activated when pressing the button highlighted in Figure 5.21(a).

The data files supported by Caffeine must have “.d2d” as extension and a proper format described in the following. Each file can contain one or more data sets, each one described by an header followed by one or more data blocks. Sections (headers or data blocks) are separated by a blank line. The header must have the following format:

```
DATASET_<Name of the data set><newline>
DOMAIN_<Unit of measure>_<Name of the quantity><newline>
CODOMAIN_<Unit of measure>_<Name of the quantity><newline>
FRAMESDOMAINRELATION_<Ka>_<Kb><newline>
```

Where:

_ Represents one or more blank characters (space or “tab”).

<newline> Represents a line break.

<Name of the data set> Is the name of the data set. It will be used as title of the line chart. Can include spaces.

<Name of the quantity> Name of the considered physical quantity (e.g. “Potential energy”). Can include spaces. It will be used to form the label of the axes of the chart.

<Unit of measure> Abbreviation of the considered physical quantity (e.g. “KJ/-mol”). Can’t contains spaces. This abbreviation will be used to form the label of the axes of the chart.

<K_a> and <K_b> In the case of dynamic systems (trajectories), if a linear relation exists between the frames of the trajectory and the domain of the data

set, then these two numerical constants are used as coefficients of the linear equation that maps the values of the domain to the frames of the trajectory:
$x = (K_a \cdot F_n) + K_b$
where $F_n \geq 1$ is the number of a frame of the trajectory and x is the corresponding value of the domain. If such relation does not exist, $<K_a>$ and $<K_b>$ must be set to 0.

The header is followed by one or more data blocks, each one representing a different data series in the chart and having the following format:

```
MEASUREMENT_[Description]<newline>
<X value 1>_<Y value 1><newline>
<X value 2>_<Y value 2><newline>
...
<X value N>_<Y value N><newline>
```

where [**Description**] is an optional field containing a brief description of the data set. If such description is provided, it will be used as label for the data series. Note that this field can contains space characters.

Below an example of "d2d" file is provided, containing a small portion of the numerical data obtained from the simulation of the unbinding process of a doxorubicin molecule from DNA (see chapter 7 for details). The file contains two data sets: (i) the number of hydrogen bonds between the nucleobases of the binding site or doxorubicin with water molecules as a function of the distance between centers of mass of the binding site and of the doxorubicin; (ii) the distance between centers of mass of the binding site and of the doxorubicin as a function of time. Note that the first data set contains two data series. As regard to the second data set, instead, a linear relation exists between the value of its domain (e.g. time) and the frames of the trajectories (as shown in Figure 5.20).

Example of a file containing 2D scalar data sets associated to a molecular system

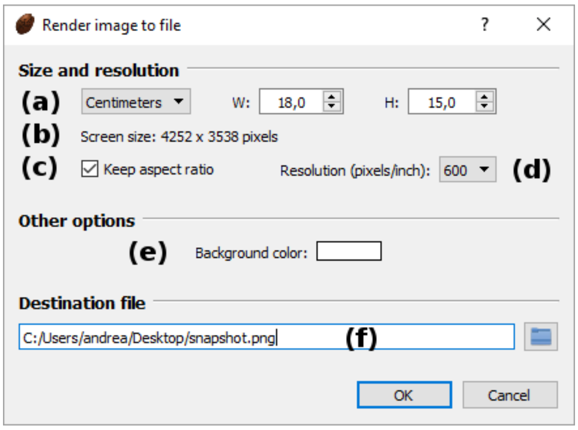
```
DATASET Distance between COMs -> Number of HB with water molecules
DOMAIN nm COMs distance
CODOMAIN - Number of hydrogen bonds
FRAMESDOMAINRELATION 0.0 0.0

MEASUREMENT Num HB formed by the binding site nucleotides
0.1043767 9.877012298770122
0.1618823 9.598240175982403
0.2236353 10.16918308169183
0.2824109 10.013298670132986
```

```
0.3409013 9.791720827917208

MEASUREMENT Num HB formed by the DOX
0.1043767 9.80991900809919
0.1618823 9.966203379662034
0.2236353 9.941105889411059
0.2824109 10.073292670732927
0.3409013 9.891810818918108

DATASET Distance between COMs
DOMAIN ps Time
CODOMAIN nm Distance
FRAMESDOMAINRELATION 1.0 -1.0

MEASUREMENT
0 0.2342122
1 0.1607865
2 0.1544051
3 0.1263411
4 0.1540792
```

5.3.11 Render to file

Caffeine allow to render an high-resolution image of the currently visualized 3D scene and to save it to file. Images with a resolution up to 600 pixels per inch (ppi) can be generated, so to produce images suitable for being included in scientific papers. User can render the visualized scene by selecting the menu item *File* → *Save as Image*. The dialog shown in Figure 5.22 will appear, by means of which the user can select various option about the image to be generated (such as size, resolution and background color). The size of the image can be expressed in pixels or in centimeters. As a general rule, if the final image is intended to be visualized mainly on a screen (e.g. the image will be included in a web page), the user should express the size in pixels. In most of these cases the chosen resolution is not relevant, so the user can leave the default value (72 ppi). On the other hand, if the image is intended to be printed or included in a paper, high resolution values (such as 300 or 600 ppi) are recommended and the size of the image should be expressed in centimeters.

It should be noted that, since the rendering is delegated to OpenGL (instead to be performed by an off-line renderer, like *POV-Ray* [280]) and using a single color buffer for the entire scene, a limit exists on the maximum size of the final image (de-


Render image to file

Size and resolution

(a) Centimeters W: 18,0 H: 15,0

(b) Screen size: 4252 x 3538 pixels

(c) ☑ Keep aspect ratio Resolution (pixels/inch): 600 (d)

Other options

(e) Background color: ⬜

Destination file

C:/Users/andrea/Desktop/snapshot.png (f)

Figure 5.22: Render to file form. (a) Final image size can be expressed in pixels (screen based size) or centimeters (print based size). (b) If the image size has been expressed in centimeters, this label shows the corresponding size in pixels. On the contrary, if the image size has been expressed in pixels, this label shows the corresponding size in centimeters. In both cases the result depends on the selected resolution. (c) If this check-box is enabled, the final image will have the same aspect ratio (width / height) of the image rendered on screen. This constrain ensures that the scene rendered on file will look like the one shown on screen, except for a possibly different resolution. (d) Allows to chose the resolution of the final image, expressed in pixels per inch. (e) Allows to chose a background color for the image. (f) Path of the image file to be rendered.

pending on the specific OpenGL implementation). Actually, this constrain could be overcome by logically splitting the final image in multiple sections, computing the proper projection parameters for each section, rendering each section in a separate memory buffer with OpenGL and finally recomposing the entire image. However, the maximum size allowed by OpenGL is, in most cases, more than sufficient to produce images for web pages and scientific papers, so the rendering of images of arbitrary size is remanded to future developments.

5.4 Caffeine “CAVE” version

The “CAVE” version of Caffeine provides the same functionalities of the “desktop” counterpart, but it also implements several specific features required by this type of installations. In fact, as explained in the following, even if the main goal of this “CAVE” version is to support the CAVE installation at Scuola Normale Superiore, the application has been designed to work in any installation equipped with one or more monoscopic or stereoscopic displays (driven by a single machine), with or without a tracking system (at present only the OptiTrack [267] tracking system is supported). This version of Caffeine has been compiled only for Windows, since the CAVE system at Scuola Normale Superiore is based on Windows and, at the time of writing, some dependencies (such as NatNet [266]) are available only for this operating system.

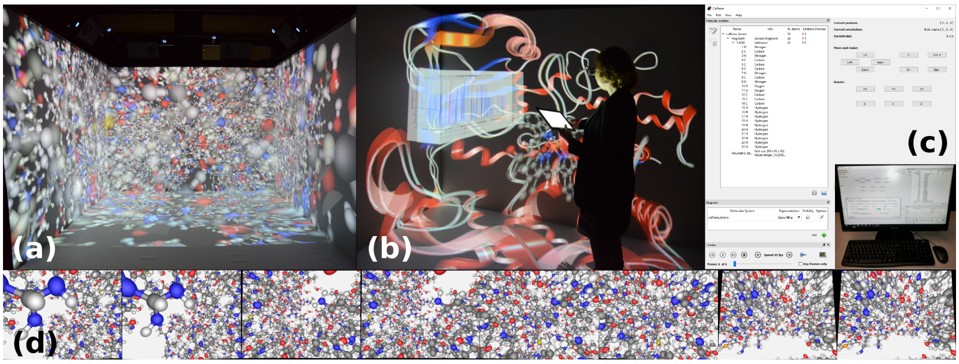


Figure 5.23: Caffeine for CAVE-like systems. (a) Caffeine running on the CAVE installation at Scuola Normale Superiore. (b) User studying a Cytochrome P450-2B4 wild type structure [281] in the CAVE. Basic manipulations of the displayed data can be performed by means of a simple button-based application for mobile devices (e.g. tablet). Line charts of numerical data associated to the molecular system (see section 5.3.10) are displayed in front to the user in a semi-transparent panel, in a way similar to an augmented reality content. (c) The main control panel of the CAVE version of Caffeine. As shown, the control panel is displayed in a monitor outside the CAVE. (d) Example of frame rendered by Caffeine for the CAVE system at SNS. Since this CAVE is equipped with four stereo projectors, Caffeine generates eight different images for each frame: a right-eye and a left-eye view of the scene for each of the four projected sides. The black padding present in some of the views are the due to the keystone correction operated in real time by Caffeine.

The CAVE system at Scuola Normale Superiore (Figure 5.23(a)) employs four projectors to display stereoscopic images on the three walls and on the floor. It is also equipped with an OptiTrack [267] tracking system. Screen are configured to constitute an extended desktop, so that the rendering can be performed on a single, large, border-less window, as shown in Figure 5.23(b). Each frame rendered by Caffeine consists in eight viewports, each one containing a different image: a right-eye and a left-eye view of the scene for each of the four projected sides. In order to provide to the user a convincing and immersive stereoscopic experience, as well as drawing the 3D scene across the screens in a consistent way, proper view and projection parameters of the eight virtual cameras must be computed each time a new frame is rendered (about 60 times per second). By doing so, the user will see the molecule as a real object in front of him/her, will be able to observe it from different positions and even immerse himself/herself within the molecule to inspect it from the inside. Details on how to compute view and projection parameters of the virtual camera for each screen are described in [282] and in section section 6.4. Broadly speaking, these parameters depends on the size, position and orientation of the physical monitors, and on the position and orientation of the user's head. The physical characteristics of the monitors are provided to Caffeine by means of a configuration file (see section 5.4.1), while information on the user's head are provided in real time by the tracking system.

Caffeine also supports real-time keystone correction of the rendered images (see Figure 5.23 (d) and Figure 5.25). In fact, since it is almost impossible to perfectly align the projectors to the projected surfaces, the displayed images may look deformed and/or do not match the borders of the screen. The solution is deform the images in order to compensate the misalignment. Note that even if this feature is often provided by projectors, this is not the case of CAVE system at SNS. Therefore it was necessary to implement such so deformation via software.

Within the CAVE, the user interacts with Caffeine using a simple, button-based application for mobile devices (e.g. tablets), as shown in Figure 5.23(b). This simple remote controller allows to move, rotate and scale the molecule, to control the playback of a trajectory, and to shows/hide between the available line charts. These charts are visualized in front to the user in a semi-transparent panel, in a way similar to an augmented reality content. At present only one chart can be displayed at time, although the user can switch between the available ones. The remaining features of Caffeine are available by means of a dedicated control panel placed outside the CAVE (Figure 5.23(c)). The user interface of this control panel is almost the same of the main window of the "desktop" version of Caffeine, with the only difference that the rendering area is here replaced by a set of controls to move, rotate and scale the molecule and to provide information about the current position, orientation and size of the displayed molecular system. For a better experience this control panel should be managed by a second user. One of the main goals of future developments is to allow a complete interaction with the application within the CAVE, both by means of hand (or wand) based interaction techniques and by creating a new remote controller with a much richer features set.

5.4.1 CAVE configuration tool

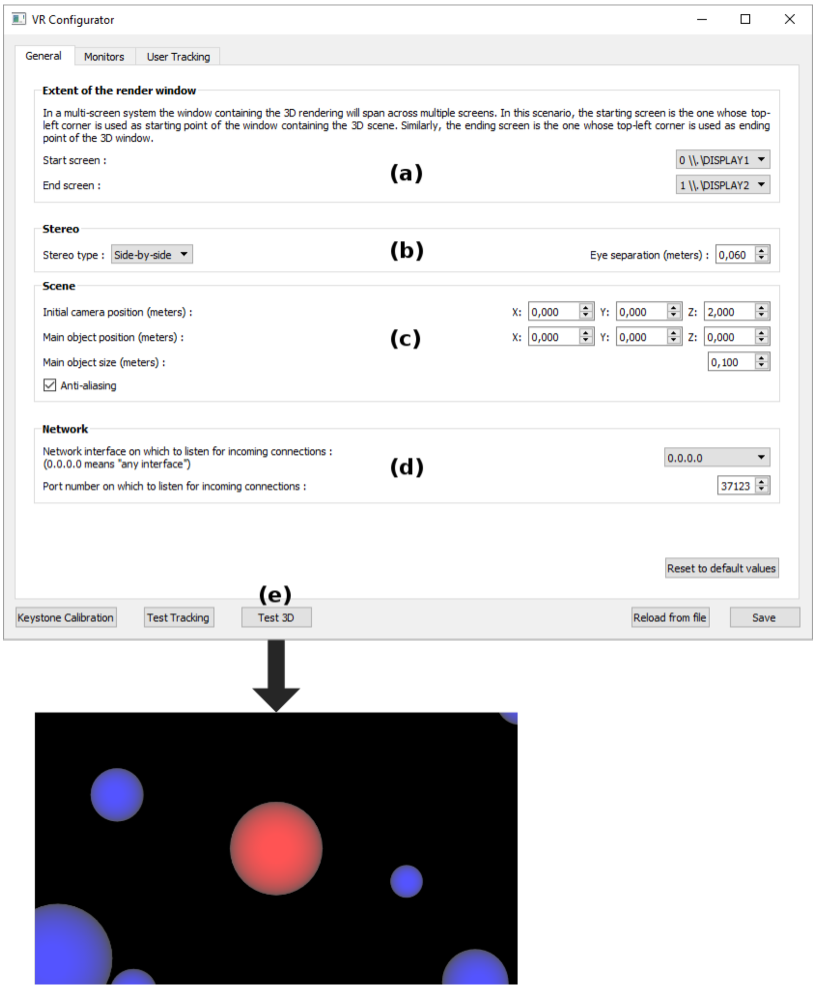
Many information required as input by the “CAVE” version of Caffeine, and in particular those related to the physical installation where it will be executed, are substantially fixed, so are stored in configuration files and loaded when the program starts. These parameters includes the list of the monitors on which the rendering window will be displayed, the physical characteristics of these monitors, the parameters of the keystone correction for each monitor, the stereo configuration etc. In order to simplify the creation of the configuration files, a configuration program has been implemented.

It is important to note that the configuration mechanism here described, as well as the software modules which implements the related functionalities (e.g. head-tracked stereo rendering, keystone correction etc.) in Caffeine, has been designed with the goal to be as general as possible, so to possibly be reused in other applications and to support different types of installations. In fact, these classes and tools was part of the “Moka” library (see chapter 4).

The configuration tool organize the settable parameters in three main groups, associated to a different tab of the user interface. To fully understand these settings it must be kept in mind that, when developing virtual reality applications, it is common choice to model virtual world so to be co-located with the physical world. In particular, a common reference frame (usually the one defined by the tracking system) is used to express sizes and positions of both virtual and physical entities. In this scenario, monitors acts as windows, through which the user can observe the virtual world (although the objects of the virtual world can appear both in front or behind these windows).

The first group of settings is shown in Figure 5.24 and contains several categories of options. First of all, it is possible to choose which monitors should be dedicated to display the 3D rendering (Figure 5.24 (a)). In fact, in a multi-screen environment, the window in which the rendering is performed may span across multiple screens. This window will be generated without borders and will be sized so to cover the entire area of the monitors on which it lays. To this end, the system administrator have to indicate a “starting” and “ending” screen: the starting screen is the one whose top-left corner is used as starting point of the window, while the ending screen is the one whose top-left corner is used as ending point of the 3D window. It follows that, to be supported by Caffeine, all the monitors must be driven by a single computers. Furthermore, the administrator must configure the system so that the screens will form a linear “extended desktop” (i.e. the desktop must span across a single row of screens). Even if the “extended desktop” configuration is the one that provide largest compatibility (by being supported by any modern operating system), it is not the solution that provide the best results. In fact, it is recommended to exploit technologies like NVIDIA Mosaic [283] or AMD Eyefinity [284] whenever possible , which abstract multiple monitors in a single large screen, thus avoiding performance and synchronization problems. Finally note that, even if it is required to *logically* organize the monitors in a single row, there is no correlation with their physical location nor with the portion of the 3D scene they will display. Then stereoscopic visualization parameters must be set (Figure 5.24 (b)). The administrator can enable or disable stereo rendering as well as to adjust the distance between the left and the right virtual cameras (“eye separation”). At present only “side-by-side” stereo is supported, in which images resulting from the left and right cameras are drawn beside each other. Figure 5.24 (c) shows the option related to the 3D scene. In fact, it is possible to specify the initial position of the virtual camera and the initial position and size of the main object of the scene (e.g. a molecular system in the case of Caffeine). Here it is also possible to enable or disable the anti-aliasing post-processing of the rendered images. The adopted anti-aliasing algorithm is *FXAA 3.11*, designed and implemented by Timothy Lottes [285], whose source code was formerly distributed by the author on his personal blog under an open-source license, thus making possible it’s inclusion in Caffeine. Finally, if the application supports some kind of remote controllers (like in the case of Caffeine), the section (d) of the “general” configuration tab allows to set the network interface and the port number on which the application will listen for incoming TCP connection from remote processes.

Once the configuration is complete (including the settings described in the following), it is possible to test the correct functioning of the system by clicking on the “Test 3D” button (Figure 5.24 (e)). A simple 3D scene will appear consisting in a set of colored spheres (see bottom of Figure 5.24). The red sphere represent the “main object” of the scene and, if stereoscopy is enabled, it should appear in the position and of the size specified in the configuration tool. The blue spheres, instead, have a fixed radius and are placed randomly.



Extent of the render window

In a multi-screen system the window containing the 3D rendering will span across multiple screens. In this scenario, the starting screen is the one whose top-left corner is used as starting point of the window containing the 3D scene. Similarly, the ending screen is the one whose top-left corner is used as ending point of the 3D window.

Start screen : 0 \\.\DISPLAY1

End screen : 1 \\.\DISPLAY2

Stereo

Stereo type : Side-by-side

Eye separation (meters) : 0,060

Scene

Initial camera position (meters) : X: 0,000 Y: 0,000 Z: 2,000

Main object position (meters) : X: 0,000 Y: 0,000 Z: 0,000

Main object size (meters) : 0,100

☑ Anti-aliasing

Network

Network interface on which to listen for incoming connections :
(0.0.0.0 means "any interface") 0.0.0.0

Port number on which to listen for incoming connections : 37123

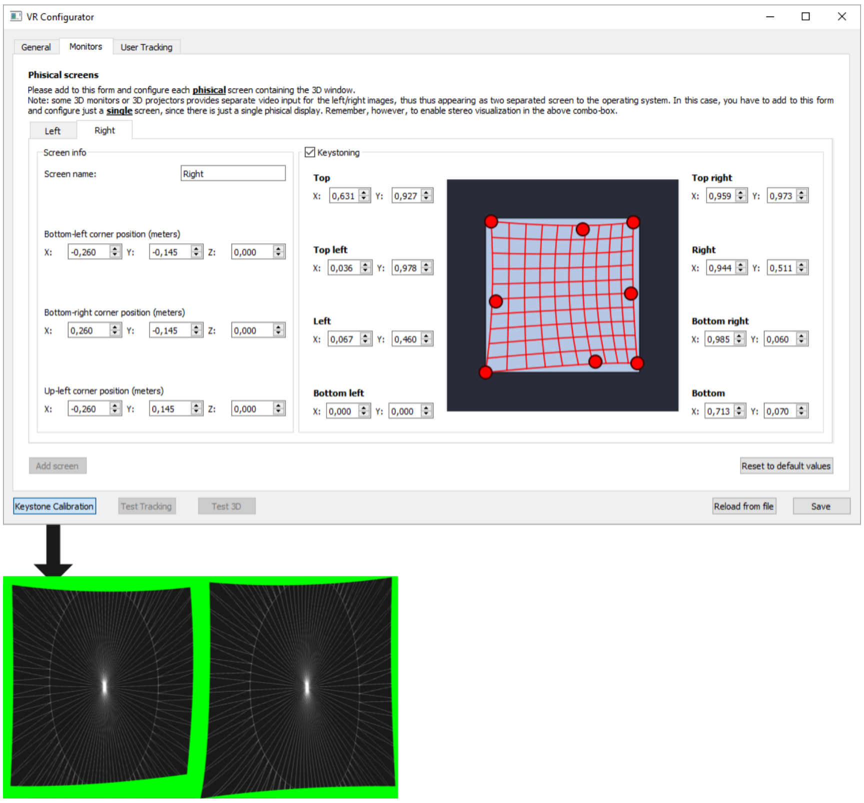
Reset to default values

Keystone Calibration | Test Tracking | Test 3D | Reload from file | Save

Figure 5.24: First group of settings provided by the configuration tool for CAVE-like systems. (a) First and last monitors upon which the rendering window must be spread. (b) Parameters related to the stereoscopic visualization. (c) Various options related to the 3D scene, such as the initial position of the virtual camera and the initial position and size of the main object of the scene. Anti-aliasing of the rendered images can also be enabled. (d) Configuration of the IP address and port number on which the application will listen for connections from remote controllers. (e) By clicking this button a simple 3D scene will be visualized, so to test the correct functioning of the system with the given configuration.

Figure 5.25 shows the user interface to configure the settings related to the physical monitors. The position and the orientation of each monitor must be specified by providing the position of three of its four corners. Since we assume that physical and virtual world are co-located, these information together with the position of the user's eyes will determine the portion of 3D scene visible through each monitor.

For each monitor it is also possible to enable and configure the keystone correction. The deformation of the rendered images is operated by an ad-hoc *fragment shader* by exploiting the equation of the *biquadratic Bézier patch*. The control points of the patch can be easily configured by means of an custom widget created specifically for this purpose[1]. In particular, by pressing "Keystone Calibration" button, a window will appear showing the interior of a sphere drawn in "wire-frame". The wire-frames acts as a grid to help the administrator to align the images on adjacent monitors. The administrator can therefore configure the keystone correction by acting on the interface shown in Figure 5.25: any change applied to the control points affects in real-time the deformation of the rendered grid.



Phisical screens

Please add to this form and configure each **phisical** screen containing the 3D window.
Note: some 3D monitors or 3D projectors provides separate video input for the left/right images, thus thus appearing as two separated screen to the operating system. In this case, you have to add to this form and configure just a **single** screen, since there is just a single phisical display. Remember, however, to enable stereo visualization in the above combo-box.

Screen info

Screen name: Right

Bottom-left corner position (meters)
X: -0,260 Y: -0,145 Z: 0,000

Bottom-right corner position (meters)
X: 0,260 Y: -0,145 Z: 0,000

Up-left corner position (meters)
X: -0,260 Y: 0,145 Z: 0,000

☑ Keystoning

Top
X: 0,631 Y: 0,927

Top left
X: 0,036 Y: 0,978

Left
X: 0,067 Y: 0,460

Bottom left
X: 0,000 Y: 0,000

Top right
X: 0,959 Y: 0,973

Right
X: 0,944 Y: 0,511

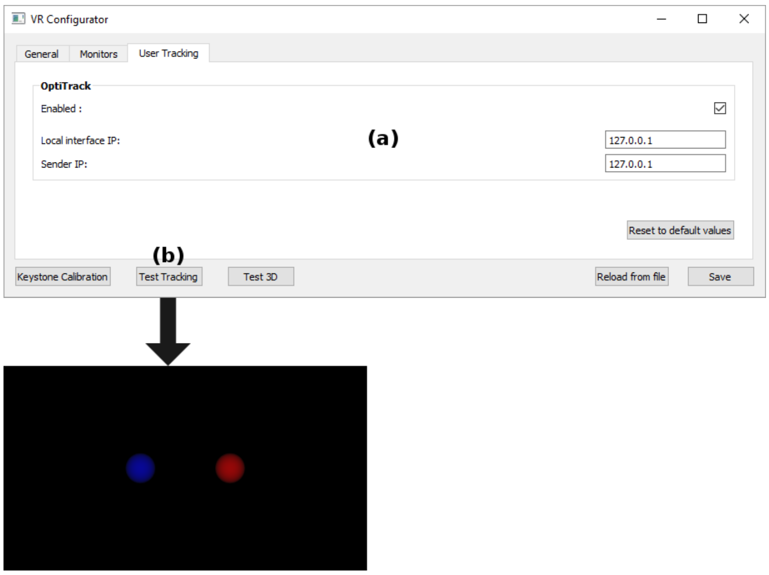
Bottom right
X: 0,985 Y: 0,060

Bottom
X: 0,713 Y: 0,070

Figure 5.25: Second group of settings provided by the configuration tool for CAVE-like systems. Here the physical location and size of each monitor must be set. For each monitor it is also possible to enable the keystone correction: by clicking on the dedicated button a rendering window will appear, showing a live preview of the applied deformation.

[1]Note that only eight of the nine control points for the Bézier patch can be defined. In fact, the central control point is fixed at the center of the surface.

The last settings group is related to the tracking system. Note that at present only OptiTrack [267] tracking systems are supported (configured to stream data compatible with the NatNet library [266]), however adding support for trackers by others manufacturers should be easy. To enable head-tracked stereoscopy, it is sufficient to check the related checkbox showed in Figure 5.26(a) and to provide both the IP address of the network interface from which the data are streamed and the IP address of the network interface from which the data must be received. If the sender process runs on the same machine of the VR application, it is sufficient to use the “*loopback address*” (127.0.0.1) in both fields. Once the configuration is complete (including the settings described previously), it is possible to test the correct configuration of both the tracking system and the VR application by pressing the “Test Tracking” button (Figure 5.26(b)). A window spanning across all the configured monitors will appear, showing the currently detected position of the user’s eye represented as a couple of colored spheres. The eye position is computed as a function of the eye separation (Figure 5.24(b)) and of the detected position and orientation of the rigid body representing the user’s head.


VR Configurator

General | Monitors | User Tracking

OptiTrack

Enabled : ✓

Local interface IP: 127.0.0.1

Sender IP: 127.0.0.1

Figure 5.26: Third group of settings provided by the configuration tool for CAVE-like systems. (a) If the considered installation is equipped with an OptiTrack tracking system, the IP address of the network interface from which the data are streamed and the IP address of the network interface from which the application will receive them must be provided. (b) To test the correct configuration of the tracking system, the system administrator can click on this button: a simple 3D scene will appear, showing the currently detected position of the user’s eye represented as a couple of colored spheres.

5.5 VR Helmets support in Caffeine

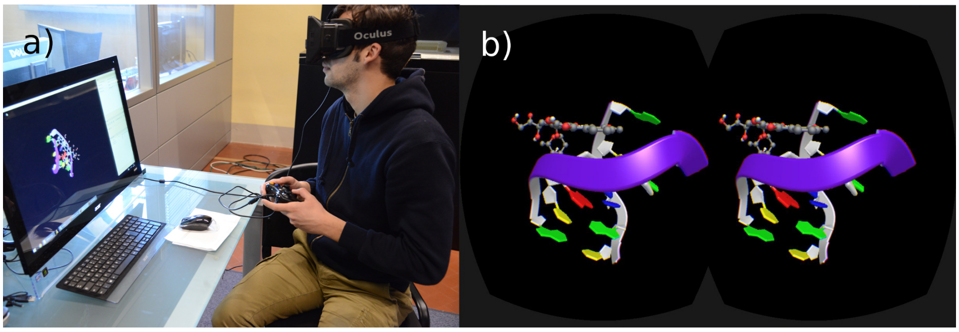


Figure 5.27: Image reproduced from [7]. (a) User wearing the Oculus Rift DK1 helmet. (b) DNA/doxorubicin binary complex rendered for the Oculus Rift DK1 with an previous version of Caffeine.

The support for the *Oculus Rift DK1* [190, 286] was implemented in a previous version of Caffeine still based on the *Moka* library (chapter 4). That version allowed the user to examine static and dynamic structures with the VR helmet, but lacks of the latest features, such as isosurfaces and line charts. The user could interactively play the loaded trajectory with the keyboard or, by using an external program that maps joystick input to keyboard events (such as JoyToKey [287]), with a gamepad.

When the *Moka* library was removed from the project, the code for supporting for the *Oculus Rift DK1* was not updated and thus temporary disabled, both because in the meantime the *Oculus Software Development Kit* [286] experienced relevant changes and because VR helmets produced by other manufactures appeared on the scene (such as the "*Vive*" by *HTC* and *Valve* [191]). For these reasons, VR helmets support in Caffeine was postponed until the release of the commercial version of these devices. At the time of writing, a wrapper integrating the OpenVR library [288] in Caffeine is in development, and should support both the *Oculus Rift* and the *HTC Vive*.

6 Caffeine: Implementation details

6.1 GPU-based ray casting of spheres and cylinders

In section 2.2 a rendering technique known as "*GPU-based ray casting of implicit surfaces*" was discussed, which allows to draw implicit surfaces (and in particular quadric surfaces) with high visual quality and rendering performance. This section provides a high-level description of the the algorithmic aspects of this technique, with particular reference to its implementation in *Caffeine*. Note that, at the time of writing, *Caffeine* implements the ray-casting only of spheres and cylinders. Support for additional types of surfaces is planned in future developments.

The idea at the base of this technique is the following: for each pixel of the final image, a ray is cast starting from the camera and passing through the considered pixel. Then, the intersection point between the ray and the surface nearest to the camera is analytically computed, together with the related normal vector. Finally the color of the pixel is computed, as a function of the material properties of the surface, the normal vector, the information about the light sources present in the scene and the chosen shading model (e.g. the *Blinn-Phong* model [164]). However, this procedure is unsuitable to be executed on graphics hardware (section 1.3). Therefore the algorithm must be re-designed as follows (see Figure 6.1): a proxy geometry is submitted to the rendering pipeline in place of each implicit surface to be drawn. The only requirement for the proxy geometry is to enclose the area occupied by the implicit surface in window space (i.e. when projected on the image plane). Therefore, it is convenient to compose the proxy geometry with a small number of geometric primitives but, at the same time, the proxy geometry should enclose the surface as tightly as possible (in window space), so to minimize the number of superfluous fragments to be processed. Common choices are point sprites, quads composed by two triangles or a parallelepipeds composed by twelve triangles. Optionally, the proxy geometry can also be generated on the fly by a dedicated *geometry shader* (as also proposed in [117, 135]), so to further reduce the geometry to be sent to the GPU and to be stored in graphics memory. By using this optimization, *Caffeine* is able to represent a sphere with a single vertex described by seven floating point numbers (three for the position, three for the color, and one for the radius), and a cylinder/capsule with two vertices described by ten floats (six for the position of the vertices, three for the color, and one for the radius). The actual proxy geometry, consisting in one or two quads, is generated on the fly by the geometry shader. A dedicated *fragment shader* is the executed for each fragment resulting from the rasterization of the proxy geometry. Its purpose is to determine the parameters of the ray starting from the camera and passing through the fragment and then to analytically compute the intersection between the ray and the considered implicit surface. If the ray miss the surface the fragment is discarded, otherwise the intersection point and the related normal vector are computed. These data are then used to compute a color for the fragment and to adjust its depth. Setting a proper depth for the fragment is crucial, since it allows to exploit the standard “*Z-buffer*” algorithm to resolve the visibility problem and to mix ray-casted surfaces with regularly rendered polygons in the same 3D scene.

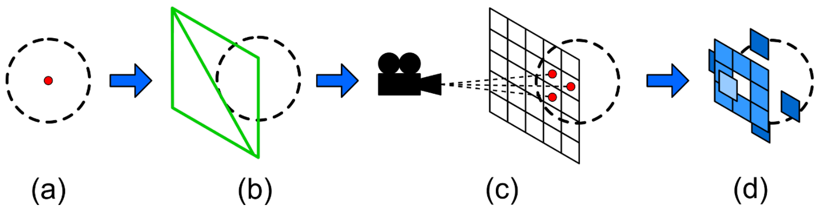


Figure 6.1: GPU-based ray casting of spheres in *Caffeine*. (a) A single vertex (along with its parameters) is submitted to the rendering pipeline for each sphere to be drawn. (b) A dedicated geometry shader transform the vertex in a quad enclosing the sphere in windows space. (c) A dedicated fragment shader is executed for each fragment resulting from the rasterization of the proxy geometry. The fragment shader computes the intersection point between the sphere and a ray starting from the camera and passing through the fragment, and the related normal vector. (d) The coordinates of the intersection point are used to adjust the depth of the fragment, while the normal vector is used in shading computations.

The main benefits of GPU-based ray casting are:

- *High visual quality*: since the surface is not tessellated, its curvature is preserved.
- *Reduced CPU–GPU bus bandwidth consumption and memory usage*, thanks to the fact that proxy geometries are usually much simpler (in terms of polygon-count) then tessellated surfaces. This property is important when the set of surfaces to be drawn changes frequently over time, such as in the case of time-varying molecular surfaces.

- *Sort of Level Of Detail (LOD) strategy built-in into the algorithm*: if the rasterization of the surface produce a small number of fragments, e.g. when the surface is distant from the camera, the rendering cost will be low.

However, this technique also has the following drawbacks:

- *The standard Multisample Anti-Aliasing (MSAA) algorithm does not produce effects on ray-casted surfaces.* For this reason, *Caffeine* uses a screen-space anti-aliasing algorithm: the "*FXAA 3.11*" designed and implemented by Timothy Lottes [285] (the source code was formerly distributed by the author on his personal blog under an open-source license).
- *Fast degradation of the rendering performance when the surfaces covers an high number of fragments.* To reduce this problem, occlusion culling strategies should be employed, like the one proposed by Grottel et al. [109].

6.2 "Tubes" modelling

Ribbons diagrams in Caffeine are made possible by an in-house developed software module capable of generating three-dimensional "*tubes*". Even if at present this module is exploited only to produce ribbons diagrams for polypeptides and polynucleotides, it has been designed to provide a good level of flexibility and, in a future, could be exploited to represent other types of information, such as vector fields.

The path of a *tube* is defined by a sequence of traits. Each trait, in turn, is defined by two endpoints, a normal vector, the shape of the cross section of the trait and at least two set of attributes determining the size and the color of the trait at each endpoint. In fact, both the size and the colors may differ between the stating and the ending point of a trait: in that case they will be linearly interpolated along the extent of the trait, as showed in Figure 6.2. Furthermore different colors can be chosen for the "top" and "bottom" half of the trait. This feature has the purpose to visualize the orientation of the trait along its main axis, defined by the normal vector. Three different shapes are available for the cross section of the trait (see Figure 6.3): rectangular with rounded corners, elliptical and (as a spacial case of this one) circular.

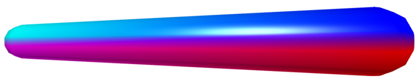


Figure 6.2: Example of a trait with a circular cross section. Different sizes and colors have been assigned to the starting and ending points of this trait. Furthermore, the top and bottom half of the trait have been colored differently, so to give a clue about the orientation of the trait.

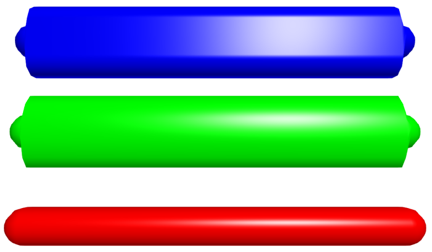


Figure 6.3: The available shapes for the cross section of the trait. From top to bottom: rectangular with rounded corners, elliptical and circular. The semi-spherical caps delimit the begin and the end of the *tube*.

As said a *tube* is defined by a sequence of consecutive traits. However, the actual path swept by the *tube* in the three-dimensional space depends on another factor: its *smoothness*. In fact, when creating the *tube*, it is possible to specify if each trait must be treated as a rectilinear segment (as shown in Figure 6.4(a)) or if the actual path must follow a *cubic B-spline* curve, whose control points are the endpoints of the traits (as shown in Figure 6.4(b)).

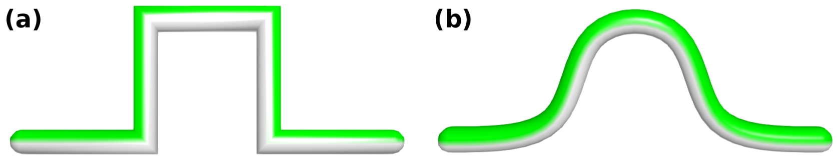


Figure 6.4: Comparison between “*rectilinear*” *tubes* (a) and “*smooth*” *tubes*.

The choice of the *cubic B-spline* to approximate the path of the traits is motivated by the fact that these curves play a major role within the field of molecular graphics, since they are widely employed to approximate the path of the backbone of polypeptides and polynucleotides in ribbons diagrams (as discussed in section 6.3). Like any “*spline*”, *cubic B-splines* are piecewise-defined curves, where each trait is a polynomial function having a certain degree of continuity with the function of the following trait. In particular, *cubic B-splines* have a C^2 continuity at join points, which provide them a considerable smoothness. Each trait of the curve is defined by four control points, according to the following equation:

$$\boldsymbol{p}(t) = \frac{1}{6} \cdot \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_3 \\ P_4 \end{bmatrix} \tag{6.1}$$

where $t \epsilon [0,1]$ and $P_0 = \begin{bmatrix} x_0 & y_0 & z_0 & 1 \end{bmatrix}$, $P_1 = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \end{bmatrix}$, $P_2 = \begin{bmatrix} x_2 & y_2 & z_2 & 1 \end{bmatrix}$, $P_3 = \begin{bmatrix} x_3 & y_3 & z_3 & 1 \end{bmatrix}$ are the control points for the considered trait.

Once the first trait of the curve is defined, following traits can be appended by just adding a control point for each new trait, since the remaining three control points are shared with the trait preceding them. It should be noted, however, that the resulting curve will *approximate* (but not interpolate) the path outlined by the control points. This is, in fact, the main drawback of this class of curves. In order to set a precise starting and ending points, two “dummy” points must be inserted respectively as first and last control points. In particular, and with reference to Figure 6.5, the “dummy” control points are P_{-1} and P_5, and they can be computed as follow:

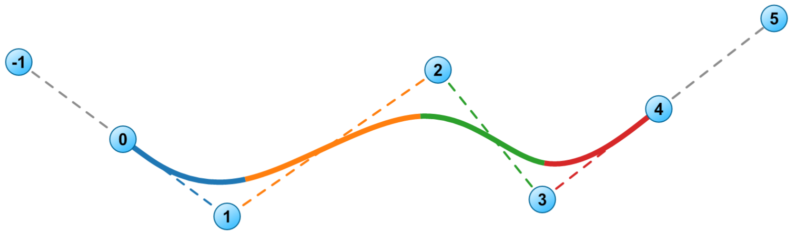


Figure 6.5: B-spline curve defined by six control points and composed by four traits. Each trait of the curve is defined by four control points. As example, the first trait is defined by the points numbered from -1 to 2, the second by the points from 0 to 3, and so on.

$$P_{-1} = P_0 + (P_0 - P_1) = 2 \cdot P_0 - P_1$$
$$P_5 = P_4 + (P_4 - P_3) = 2 \cdot P_4 - P_3$$

With regard to the generation of the graphical geometry for the *tubes*, this involves the following steps:

1. For each trait, an orthonormal basis is computed. Let be $\overrightarrow{T_i}$, $\overrightarrow{N_i}$ and $\overrightarrow{B_i}$ the versors forming that basis for the *i-th* trait of the tube.$\overrightarrow{N}_i$ is the "normal" versor, which defines the "top" and "bottom" sides of the trait; $\overrightarrow{T_i}$ represent the direction of the trait, i.e. the versor directed from the starting to the ending point of the trait: $\overrightarrow{T_i} = (P_{i+1} - P_i)/\left\|P_{i+1} - P_i\right\|$; $\overrightarrow{B_i}$ is a versor orthogonal both to $\overrightarrow{T_i}$ and $\overrightarrow{N_i}$: $\overrightarrow{B_i} = \overrightarrow{N_i} \times \overrightarrow{T_i}$. It is called "binormal" and defines the direction along which the tube extends laterally. Note that, in order to simplify the use of this software module, the *provided* $\overrightarrow{N_i}$ need not necessarily be orthogonal to $\overrightarrow{T_i}$. In other words, the provided $\overrightarrow{N_i}$ is a sort of "first guess" about the normal of the trait. As a consecuence, a correct $\overrightarrow{N_i}$ orthogonal both to $\overrightarrow{T_i}$ and $\overrightarrow{B_i}$ is automatically re-computed as follow: $\overrightarrow{N_i} = \overrightarrow{T_i} \times \overrightarrow{B_i}$.

2. For each trait, the path spanned by the trait is defined by a configurable number of equally spaced points lying on it.

   a) In the case of "*rectilinear*" *tubes*, only two "*path-points*" are generated, coinciding with the starting and ending points of the trait.

   b) For "*smooth*" *tubes*, instead, the "*path-points*" are computed by sampling the B-spline function for the given trait at regular intervals. The number of samples for each section of *B-spline* is configurable, thus allowing to construct *tubes* with different quality levels.

3. For each *path-point “j”* of the *i-th* trait, an orthonormal basis is computed, formed by the *tangent* $\overrightarrow{T_i^j}$, *normal* $\overrightarrow{N_i^j}$ and *binormal* $\overrightarrow{B_i^j}$ versors. $\overrightarrow{T_i^j}$ approximates the tangent to the path in the considered *path-point* and is computed as the direction from the considered *path-point* to the following one of the trait. $\overrightarrow{N_i^j}$ and $\overrightarrow{B_i^j}$ have the same meaning of the corresponding versors of the trait, but referred to the considered *path-point.*

   a) In the case of “*rectilinear*” *tubes*, a simple solution would be using, for both *path-points*, the normal of the trait $\overrightarrow{N_i}$ as $\overrightarrow{N_i^j}$, the direction of the trait $\overrightarrow{T_i}$ as $\overrightarrow{T_i^j}$ and $\overrightarrow{B_i^j} = \overrightarrow{N_i^j} \times \overrightarrow{T_i^j}$. However, to avoid discontinuities between adjacent traits, the tangent vector of the previous and next traits must be taken into account:

      i. For the first *path-point,* $\overrightarrow{T_i^{start}}$ is given by the normalized sum of the direction of the considered trait with the direction of the previous trait:

      $$\overrightarrow{T_i^{start}} = \left(\overrightarrow{T_i} + \overrightarrow{T_{i-1}}\right) / \left\|\overrightarrow{T_i} + \overrightarrow{T_{i-1}}\right\|$$

      ii. For the last *path-points,* $\overrightarrow{T_i^{end}}$ is given by normalizing the sum between the direction of the considered trait and the direction of the next trait:

      $$\overrightarrow{T_i^{end}} = \left(\overrightarrow{T_i} + \overrightarrow{T_{i+1}}\right) / \left\|\overrightarrow{T_i} + \overrightarrow{T_{i+1}}\right\|$$

      iii. In both cases, $\overrightarrow{B_i^j}$ results from the cross-product between the normal associated to the trait and the computed $\overrightarrow{T_i^j}$, while $\overrightarrow{N_i^j} = \overrightarrow{T_i^j} \times \overrightarrow{B_i^j}$:

      $$\overrightarrow{B_i^{start}} = \overrightarrow{N_i} \times \overrightarrow{T_i^{start}} \quad ; \quad \overrightarrow{N_i^{start}} = \overrightarrow{T_i^{start}} \times \overrightarrow{B_i^{start}}$$
      $$\overrightarrow{B_i^{end}} = \overrightarrow{N_i} \times \overrightarrow{T_i^{end}} \quad ; \quad \overrightarrow{N_i^{end}} = \overrightarrow{T_i^{end}} \times \overrightarrow{B_i^{end}}$$

   b) For “*smooth*” *tubes*, *cubic B-splines* are exploited to interpolate normal vectors between two traits, similarly to what done by Krone et al. [118] in their GPU-accelerated algorithm for the construction of ribbons diagrams. In this way it is possible to obtain a smother transition between different orientations with respect to a linear interpolation. To explain in detail how $\overrightarrow{N_i^j}$ is computed for each *path-point* of a given trait, consider Figure 6.6(a). Suppose we want to interpolate the normal vectors along the trait S_i (colored in red in Figure 6.6(a)). $\overrightarrow{N_i}$ is the normal vector associated to S_i and it is considered to be placed at the center of the trait. If we sum independently each of the normal vectors $N_{i-2}, \ldots, N_i, \ldots, N_{i+2}$ to the origin of a Cartesian coordinate systems, we obtain five points, named $P_{N_{i-2}}, \ldots, P_{N_i}, \ldots, P_{N_{i+2}}$ in Figure 6.6(b). These points can then be used as control points of two consecutive *cubic B-spline* sections (colored in orange and green in the figure), whose equations can be evaluated to compute a normal vector for each *path-point* of S_i. It is important to note that the first *B-spline* section (having $P_{N_{i-2}}, \ldots, P_{N_{i+1}}$ as control points) must be considered when computing the normal vector for the path-points lying on the first half of S_i, while the second *B-spline* section (having $P_{N_{i-1}}, \ldots, P_{N_{i+2}}$ as control points) must be considered when computing the normal vector for the path-points of the second half of S_i (Figure 6.6(c)). The computation of an appropriate value of the t parameter (for the function 6.1) for each *path-point* must keep into account this fact. As always, $\overrightarrow{B_i^j} = \overrightarrow{N_i^j} \times \overrightarrow{T_i^j}$. Finally, in order to ensure that $\overrightarrow{N_i^j}$ is orthogonal to $\overrightarrow{T_i^j}$, it is recomputed as follow: $\overrightarrow{N_i^j} = \overrightarrow{T_i^j} \times \overrightarrow{B_i^j}$.

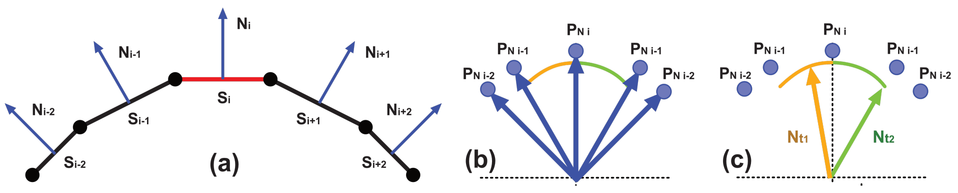


Figure 6.6: Procedure for the interpolation of normal vectors in a "*smooth*" *tube.* (a) Five traits of a tube are shown. We want to interpolate the normal vector along the trait S_i, colored in red. (b) Five points are computed by summing independently each of the normal vectors to the origin of the reference frame. These points are then employed as control points of two consecutive *cubic B-spline* sections (colored in orange and green). (c) The normal vector to be assigned to a given *path-point* of S_i is computed by evaluating the function 6.1 with a proper parameter t corresponding to the *path-point.* Note that a different set of control points and a different value for t must be used depending on whether the *path-point* belongs to the first or second half of S_i.

4. Sizes and colors are computed for each *path-point*, by linearly interpolating those defined for the endpoints of the trait.

5. A set of vertices is generated for each *path-point.* These vertices will form

the cross-section of the *tube* at the considered *path-point*. Their position and color are computed as a function of the shape of the trait and of the attributes associated to the considered *path-point* (sizes, colors and the orthonormal basis described before). The number of vertices for each cross-section of the *tube* is configurable, thus allowing to construct *tubes* with different quality levels.

6. The vertices generated for two consecutive path-points are finally used to define the triangles representing the external surface of the *tube*. When doing this, precautions have been taken in order to avoid "twists" along the *tube* (when two consecutive traits have a significantly different orientation).

From the procedure described before, it follows that "*rectilinear*" *tubes* preserve the orientation of the traits, therefore the orientation varies at discrete steps along the *tube*, as shown in Figure 6.7(a). "*Smooth*" *tubes*, instead, produce a more continuous variation of the orientation along the *tube*, as shown in Figure 6.7(b).

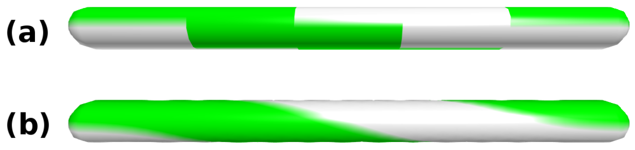


Figure 6.7: Comparison between two types of *tubes* composed by five traits, each with a different orientation. (a) "*Rectilinear*" *tubes* preserve the orientation for along each trait. (b) "*Smooth*" *tubes* smoothly interpolate the orientation of the traits along their extent.

As said before and as showed in Figure 6.2 and Figure 6.8(a), it is possible to obtain smooth transitions in the size and/or in the color of the *tube* by assigning different sizes and/or colors to the two endpoints of a trait, and by taking care of assigning a corresponding set of attributes to the starting/ending point of the subsequent/previous trait. However, its is also possible to produce sharp changes between consecutive traits, as showed in Figure 6.8(b). To do so, it is sufficient to assign different colors, sizes or shape between the ending point of a trait and the starting point of the following one.

There is another way to obtain a similar effect. In fact, it is possible to specify additional sets of attributes *within* a same trait. More formally, let consider a parameter t ranging along the trait in [0,1]. Each trait can contain two or more sets of attributes describing how the size and the color changes along the trait. Each set of attributes is associated to a value of t, defining the point of the trait to which those attributes apply. As minimum, each trait must have a set of attributes

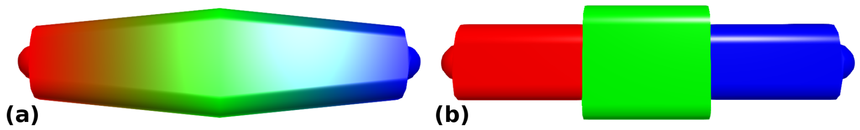


Figure 6.8: (a) Smooth transitions in the size and/or in the color of the *tube*. (b) Sharp variations of the size and/or color of the *tube*. These effects can be obtained by appending traits with different attributes or by using "middle-point attributes" within a single trait. The semi-spherical caps delimit the begin and the end of the *tube*.

for $t = 0$ (starting point) and $t = 1$ (ending point), but additional attributes can be assigned for $t \epsilon (0, 1)$ (internal points of the trait). As an example, the same result of Figure 6.8(a) can be obtained with a single trait, but specifying a larger size and a green color for $t = 0.5$. Furthermore, multiple (usually one or two) sets of attributes can be defined for a same value of t, thus allowing to obtain sharp changes such as those depicted in Figure 6.8(b).

So, what's the difference between using "middle-point attributes" and defining multiple traits with different attributes? For "*rectilinear*" *tubes* there is no difference. However, in the case of "*smooth*" *tubes*, using multiple traits to obtain visual changes in the tube would introduce additional control points that would affect the path of the *tube*! An example of the use of "middle-point attributes" in *Caffeine* is to create the arrow of the beta-strands in ribbons diagrams, without affecting the path of the backbone.

This additional feature affects the algorithm for generating the graphical geometry of tubes in the following ways:

- A *path-point* (and consequently a set of vertices) must be generated for each value of t assigned to a set of attributes.
- For a same point along the path, multiple *path-point* can exists, having different associated sizes and/or colors.
- When computing the attributes for a *path-point* (point 3 of the previous procedure), "middle-point attributes" must be taken into account.

6.3 Ribbon diagrams

This section describes the salient aspects of the algorithms implemented in *Caffeine* for the construction of ribbon diagrams for polypeptides and polynucleotides. Since the actual generation of the triangle meshes representing the ribbons are delegated to the software module for constructing "3D tubes" (described in the previous section), the discussion will focus on how the traits of each tube are defined in order to obtain meaningful and visually pleasant ribbon diagrams.

6.3.1 Polypeptides

The procedure for the construction of ribbon diagrams of polypeptides is based on the popular algorithm presented by Carson at the end of the 1980s [74, 75, 77]. Carson proposed to graphically represent the backbone of a polypeptide by a ribbon whose path is defined by a cubic B-spline having the position of the C_α atoms as control points. Each section of the ribbon spans over the peptide plane formed by two consecutive residues of the chain and must have the same orientation of that plane. For illustration purposes, a schematic representation of a section of polypeptide is shown in Figure 6.9.

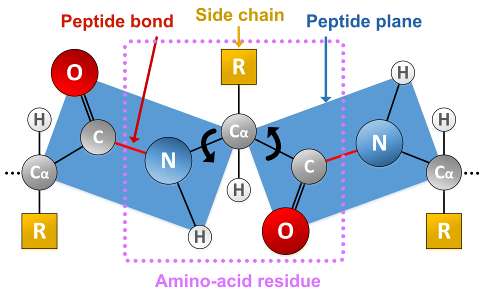


Figure 6.9: Schematic representation of a section of polypeptide.

Formally, let C_α^i and C_α^{i+1} be the position of C_α of two consecutive residues in the chain and let O^i be the position of the oxygen atom of the carboxyl group of the first of these two residues. The following vectors can be defined:

$$\overrightarrow{T_i} = \frac{C_\alpha^{i+1} - C_\alpha^i}{\|C_\alpha^{i+1} - C_\alpha^i\|} \qquad \overrightarrow{N_i} = \frac{(C_\alpha^{i+1} - C_\alpha^i) \times (O^i - C_\alpha^i)}{\|(C_\alpha^{i+1} - C_\alpha^i) \times (O^i - C_\alpha^i)\|} \qquad \overrightarrow{B_i} = \overrightarrow{N_i} \times \overrightarrow{T_i}$$

$\overrightarrow{T_i}$ represent a sort of direction of the backbone in the considered section, $\overrightarrow{N_i}$ is the normal vector of the considered peptide plane, while $\overrightarrow{B_i}$ is a vector orthogonal both to $\overrightarrow{T_i}$ and $\overrightarrow{N_i}$. Let's call $\overrightarrow{B_i}$ the binormal of the peptide plane. For each pair of consecutive residues, the related section of ribbon must be oriented according to $\overrightarrow{B_i}$ (i.e. the ribbon must extend laterally along $\pm\overrightarrow{B_i}$) and must follow the path defined by a section of cubic B-spline (equation 6.1) having $(C_\alpha^{i-1}, C_\alpha^i, C_\alpha^{i+1}, C_\alpha^{i+2})$ as control points. As noticed by Carson, a complication arise from the fact that the orientation of consecutive peptide planes may flip (as in the case of beta-strands), thus resulting in twisted ribbons. This problem can be detected and solved by comparing the binormal vector $\overrightarrow{B_i}$ of the considered peptide plane with the one of the previous peptide plane in the chain ($\overrightarrow{B_{i-1}}$). If the angle between $\overrightarrow{B_i}$ and $\overrightarrow{B_{i-1}}$ is greater than 90° (i.e. if the scalar product between $\overrightarrow{B_i}$ and $\overrightarrow{B_{i-1}}$ is less than 0), then the vectors $\overrightarrow{B_i}$ and $\overrightarrow{N_i}$ must be inverted. As regard to the shape of the ribbon, it varies among the implementations. *Caffeine* employs the most widespread representation, in which random coils are shaped as tubes, helices as spirals of ribbon and beta-strands as ribbons terminated by an arrow pointing in direction of the carboxyl group. The detections of the secondary structure of the polypeptide chains is delegated to Stride [270], invoked by Caffeine as an external program.

As said, *Caffeine* makes use of the software module for constructing "3D tubes" (described in section 6.2) in order to generate a triangle mesh for the ribbon. For each residue of the chain apart from the last one, a new trait is constructed and appended to the 3D tube. The starting and ending point of the trait are C_α^i and C_α^{i+1} respectively, while its orientation is defined by the normal vector of the peptide plane $\overrightarrow{N_i}$, by means of which a proper binormal vector is computed. The algorithm verifies the presence of flips between the orientations of the new and previous trait and, in this case, inverts the normal and binormal vectors of the new trait (to avoid twists in the ribbon). Sizes, colors and shape of each trait depend on the type of secondary structures formed by the two involved residues. As regard to the shape, the cross section of the tube will be a circle if both the residues form a random coil, and a rounded rectangle otherwise. Sizes and colors are specified by defying two or more sets of "attributes" for the trait. As explained in section 6.2, it is mandatory to define at least one set of attributes for both the ends of the trait. Let the two involved residues be respectively be the i-th and the (i+1)-th of the chain. Then, the attributes for the staring point of the trait will depend on the secondary structure and on the user-defined settings of the i-th residue, while the attributes for the ending point will depend on the secondary structure and on the user-defined settings of the (i+1)-th residue. In particular:

- If the considered residue forms a random coil, then the diameter of the circular tube is given by the minimum between the width and the height chosen by the user for the ribbon, while the color depends on the color scheme (e.g. “color by residue”, “color by secondary structure”, etc.) chosen by the user (see section 5.3.6).
- If the considered residue forms a beta strand, in the general case both the color and the size will be assigned according to the user-defined settings. However, if the considered residue is the last of the strand, then its $C_\alpha$lies the middle of the arrow, therefore the width of the start of the trait must be enlarged accordingly. Finally, if the considered residue is both the last of the strand and the last of the chain, then its C_α constitutes the end of the arrow, so the width of the end of the trait is set to be equal to its height (in order to look circular).
- If the considered residue forms a helix, then the width and height of the corresponding starting/ending point of the trait will be the ones chosen by the user. As regard to the color, instead, some additional consideration is necessary. In fact, as explained in section 5.3.6, Caffeine uses different colors to draw the outer and the inner sides of the helix (with the exception of the coloring by residue, where both sides have the same color). The outer color is retrieved by the user-defined settings, while the inner color is computed by decreasing the saturation of the outer color. As explained in section 6.2, it is perfectly valid to assign different colors for the “top” and “bottom” half of the trait. However, it is not said that the “top” side of the trait corresponds to the outer side of the helix! In order to determine how the normal vector $\overrightarrow{N_i}$ of the considered trait is oriented with respect to the helix, the following *heuristic* is employed: consider the vector $\overrightarrow{H_i} = (C_\alpha^{i+1} - C_\alpha^{i-1})$ providing a rough approximation of the direction of the spiral in proximity of the i-th residue. If the angle between $\overrightarrow{H_i}$ and $\overrightarrow{N_i}$ is greater than 90° (i.e. if $\overrightarrow{H_i} \cdot \overrightarrow{N_i} < 0$) then the normal is directed inside the spiral, otherwise is directed outside the spiral. The effectiveness of this heuristic have been confirmed by numerous tests on real-world molecular structures obtained from the Protein Data Bank. According to the result of this test, outer/inner colors are then assigned as top/bottom colors for the trait. Finally, there is one last special case to consider: when the helix is left-handed. In that case, inner and other color are swapped, so to use the more saturated color for the inner side of

the helix (as showed in Figure 5.10). The handedness of the helix is detected according to the following algorithm:

- Let C_α^{first} and C_α^{last} be the position of the C_α atoms of the first and last residue of the helix. The axis of the spiral can be approximated by the versor: $\overrightarrow{A} = (C_\alpha^{last} - C_\alpha^{first}) / \left\| C_\alpha^{last} - C_\alpha^{first} \right\|$.
- For each section of the helix constituted by three consecutive residues, consider the position of their C_α atoms:C_α^{j}, $C_\alpha^{j+1}$and C_α^{j+2}. They form a turn constituted by two consecutive traits having the following directions: $\overrightarrow{T_j} = (C_\alpha^{j+1} - C_\alpha^{j}) / \left\| C_\alpha^{j+1} - C_\alpha^{j} \right\|$, $\overrightarrow{T_{j+1}} = (C_\alpha^{j+2} - C_\alpha^{j+1}) / \left\| C_\alpha^{j+2} - C_\alpha^{j+1} \right\|$.
- The considered turn is right-handed if $\left(\left(\overrightarrow{T_j} \times \overrightarrow{T_{j+1}} \right) \cdot \overrightarrow{A} \right) > 0$, left-handed otherwise.
- Although in theory it should be sufficient to compute the handedness of a single turn (triple of consecutive residues) to determine the handedness of the entire helix, the algorithm computes the handedness of all the turns of the helix, and then assign an handedness to the helix by majority rule. This choice is motivated to compensate structural irregularities that may occur at the begin or at the end of a helix.
- Finally note that the handedness of the helices is pre-computed after loading of the protein and detecting of its secondary structure, in order to avoid any impact on the performance of the algorithm for the construction of the ribbon diagram. This is important in the visualization of dynamic systems, when the graphical geometry for each frame of the trajectory is constructed on-the-fly.

Apart from the sets of attributes for the starting and ending points of the trait, additional attributes may be generated for its middle point. They are necessary only in particular conditions, e.g. in the case of sharp changes in the graphical properties of consecutive residues. In fact, the middle of the trait approximate the position of the peptide bond, that is the “boundary” between adjacent amino acid residues. The following list describe in detail the cases in which “middle-point attributes” are generated and the desired effect:

- Coloring by residue: each residues must be colored with a solid color without transitions (gradients) between adjacent residues.
- Adjacent residues having different graphical settings: a sharp variation of size or color occurs.

- Point in the middle of the last two residues of a beta-strand: the arrow starts at the middle the trait, thus a sharp enlargement of the width is required.
- Middle point between the last residue of a beta-strand and another residue: the arrow ends at the middle the trait, thus a gradual downsizing of the width is required.
- Middle point between the last residue of a helix and the first residue of a random coil or beta-strand: the width of the ribbon is gradually downsized up to resemble a coil.
- Middle point between the last residue of a random coil and the first residue of another structure: the width of the ribbon is gradually enlarged.

Note that only a single set of attributes for the middle point of the trait is generated in the case of a gradual transitions of the graphical properties, while two different sets of attributes are required to obtain sharp changes, as shown in Figure 6.10.

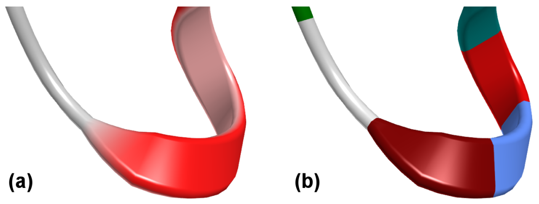


Figure 6.10: Closer look to an alpha-helix followed by a random coil. (a) By coloring the chain by secondary structure, only one set of attributes is applied to the middle point between the last residue of the helix and the first of the random coil, in order to obtain a smooth transition of both size and color. (b) By coloring the chain by residue, a sharp transition in the color happens between the two residues, requiring the use of two set of attributes for the middle point.

6.3.2 Polynucleotides

The sugar-phosphate backbone of nucleic acids is represented by a ribbon, whose path is defined by a cubic B-spline curve having the position of phosphorus atoms as control points (except for specific cases, as explained in the following). The construction of the ribbon is performed by appending a new trait to the “3D tube” for each nucleotide of the chain.

The implemented algorithm requires that the nucleotides are iterated in 5´ to 3´ direction. A specific control has been implemented to verify this condition. In fact, although the PDB file format imposes that nucleotides must be listed from the 5’ to the 3’ terminus [19], non-conforming files may happens in practice. Furthermore, PDB files exists in which amino acid residues are correctly listed from the 5’ to the 3’ terminus, but having decreasing sequence numbers (see, as an example, the chains ’R’ and ’T’ of the PDB file ’1MSW’). Since *Caffeine* stores residues in an ordered data structure (to speed up their lookup by sequence number), the direction of the chain is inverted when the residues are listed with decreasing sequence numbers. The actual direction of the chain is verified by checking the presence of a phosphorus atom in the first and last residue. In fact, since in PDB files the phosphate group included in a residue is the one bonded to C5’ (thus “preceding” the sugar according to the 5´ to 3´ direction) and since usually PDB files does not contain the phosphate group for the 5’-end, then the chain is considered to be in 3´ to 5´ direction if the first residue contains a phosphorous atom and the last residue does not, and in 5´ to 3´ direction otherwise.

Phosphorus atoms are used as starting and ending point of the traits who define the ribbon (i.e. as control points of the B-spline), with the exception of the first and last nucleotide: the position of the C5’ atom of the 5’-end is used as staring point of the first trait, while the position of the C3’ atom of the 3’-end is used as ending point of the last trait. Furthermore, the first and last traits are drawn respectively slightly larger and smaller of the rest of the ribbon, in order to visualize the orientation of the chain (see Figure 6.11). For each nucleotide, the versor directed from the position of C1’ to the position of C3’ ($\overrightarrow{C3' - C1'} / \|C3' - C1'\|$) is used as normal vector to define the orientation of the trait.

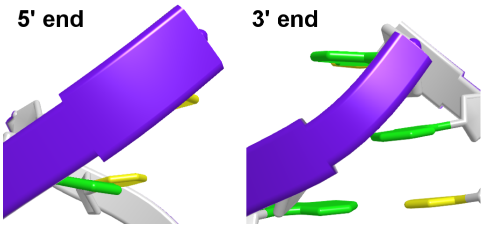


Figure 6.11: In ribbon diagrams of polynucleotides, the nucleotides related to the 5’ and 3’ ends of a chain are visualized slightly larger and smaller of the rest of the ribbon, in order to highlight the direction of the chain.

There are other particular cases that must be explicitly detected and handled in order to provide a correct ribbon representation. First of all, it may happen that the two chains of a DNA helix are encoded as a single chain within the PDB file. Furthermore, during the bio-chemical processes of reading and replication of the DNA, a break in the chain occurs, due to the lack of the phosphate group connecting two nucleotides. *Caffeine* is able to detect both these special cases, as shown in Figure 6.12. In particular, if in the middle of the chain described by the PDB file, a nucleotide without a phosphorus atom is detected, then a break in the chain is assumed: the nucleotide without the phosphorus is considered to be the 5'-end of a new sub-chain, while the previous nucleotide is considered to be the 3'-end of the old sub-chain. The "3D tube" is therefore interrupted, and the two ends are visualized with a slightly smaller and larger ribbon, as explained so far.

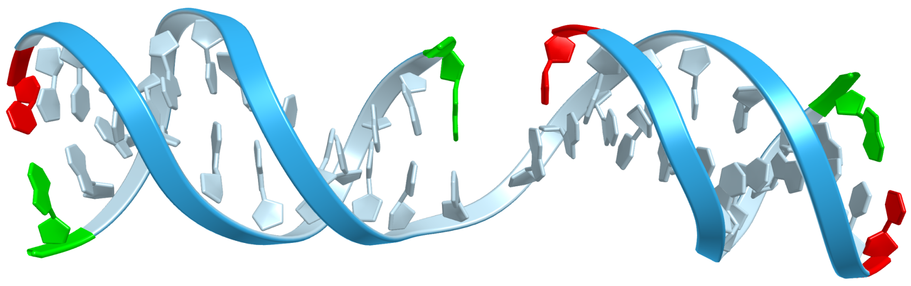


Figure 6.12: Fragment of DNA with a break in one of the two chain of the double helix. The related PDB file lists all the nucleotides as belonging to a single chain. *Caffeine*, instead correctly detects the presence of 3 polynucleotides. In this figure, the nucleotides related to the 5' and 3' ends of each chain are colored respectively in green and red.

Sugar and nucleobases are drawn as 3D polygons connected by a "stick". The borders of the 3D polygons and the stick are constituted by ray-casted "capsules" (cylinders with two semi-spheres at their ends), in order to obtain smooth borders with a minimum required geometry, while the top and bottom faces are properly triangulated. As explained in section section 5.3.6 and shown in Figure 5.11, the sugar can be hidden to obtain a simpler representation. In that case, a stick is drawn from C3' to the nitrogen atom of the nucleobase which is nearest to the sugar.

6.4 Head-tracked stereoscopy

Human visual system exploits multiple cues to perceive the three-dimensionality of space, giving us the ability to estimate sizes and distances of objects of the real world. Among these, the predominant "*depth cue*" arise from the ability of the brain to infer depth and size information by analyzing the differences between the images perceived by the eyes. Virtual Reality (VR) systems employ stereoscopic displays to provide a binocular vision of the projected images, thus simulating depth perception.

In order for an application to provide stereoscopic visualization, it must render the virtual scene twice, one for each eyes. These images will then be presented independently to the related eye. When creating virtual reality applications it is good practice to model the virtual world using a real-world unit of measure for sizes and distances, and to define a common reference frame for both virtual and real worlds. In other words, virtual objects and physical entities (such as the user or the monitor) should logically coexist. Assuming this approach, the virtual cameras must be placed within the 3D scene in correspondence of the position of the user's eyes and must be oriented according to the user's head.

From the point of view of graphics programming, defining the parameters of the two virtual cameras means to compute proper "view" and "projection" matrices. The "*view matrix*" is related to the position and the orientation of the camera in the virtual world. In particular, it is the change of frame matrix from the global reference frame to the local reference frame of the camera. The so called "*projection matrix*", instead, defines the shape of the "*view frustum*", i.e. a truncated rectangular pyramid bounding the region of the virtual world that can be "seen" by the camera (Figure 6.13).

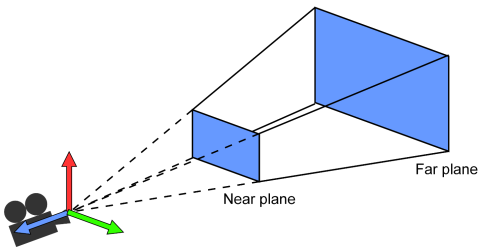


Figure 6.13: Symmetric view frustum in OpenGL.

The way these matrices are computed depends on the characteristics of the considered Virtual Reality installation. For VR systems equipped with multiple stereoscopic displays (such as CAVE systems), the most popular method to compute these matrices is probably the one described by Robert Kooima [282] (also employed by *Caffeine*), that will be summarized in the following.

To begin, consider a system equipped with a *single fixed stereoscopic* monitor and a tracking system (used to detect position and orientation of the user's head). The first step is to define a global reference frame to be employed both for virtual and physical entities. Let us consider a right-handed reference frame, like the one assumed by *OpenGL*. Let also be O the origin of that reference frame. Given the information provided by the tracking system (position and orientation of a tracker bound to the user's head) and knowing that the distance between the two eyes is about 6 - 6.5 cm, it is possible to approximate the position of the two eyes. Let be C_{left} and C_{right} these positions. Finally, let S_{BL}, S_{BR} and S_{TL} be the position respectively of the bottom-left, bottom-right and top-left corners of the screen. Then, the following vectors form an orthonormal basis that, together with S_{BL}, defines a local reference frame for the screen:

$$U = \frac{(S_{BR} - S_{BL})}{\|S_{BR} - S_{BL}\|} \qquad V = \frac{(S_{TL} - S_{BL})}{\|S_{TL} - S_{BL}\|} \qquad W = U \times V$$

This basis can be used to build a rotation matrix expressing the orientation of the screen with respect to the global reference frame:

$$R_{screen} = \begin{bmatrix} U_x & V_x & W_x & 0 \\ U_y & V_y & W_y & 0 \\ U_z & V_z & W_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To give the user the illusion to move within the virtual world, the objects of the scene must be translated by the vector $\overrightarrow{O - C_{left}}$ when generating the image for the left eye, and by the vector $\overrightarrow{O - C_{right}}$ when generating the image for the right eye. The corresponding translation matrices are:

$$T_{left} = \begin{bmatrix} 1 & 0 & 0 & -C_x^{left} \\ 0 & 1 & 0 & -C_y^{left} \\ 0 & 0 & 1 & -C_z^{left} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_{right} = \begin{bmatrix} 1 & 0 & 0 & -C_x^{right} \\ 0 & 1 & 0 & -C_y^{right} \\ 0 & 0 & 1 & -C_z^{right} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For the same reason, the scene must be rotated in a way opposite to the orientation of the camera in the virtual world. In the case of VR systems equipped with a *fixed* monitor, it is common to assume that the view direction is always orthogonal to the screen. An orthonormal basis that satisfy such constrain is $\{U, V, W\}$, (expressing the orientation of the screen with respect to the global reference frame). Thus, the desired rotation matrix is $R_{screen}^{-1} = R_{screen}^{T}$. It is now possible to compute the view matrices for the left and right eyes:

$$V_{left} = R_{screen}^{-1} \cdot T_{left} \qquad V_{right} = R_{screen}^{-1} \cdot T_{right}$$

As regard to the view frustum of the two cameras, they must be shaped so that their edges will converge at some point in space, as shown in Figure 6.14(a). The four intersection points will define the "*projection plane*"[1] within the 3D scene: the objects lying between the cameras and the projection plane will appear to "pop out" from the screen, while those lying behind the projection plane will appear to be behind the screen. In order to match the virtual and real worlds, the view frustums must be shaped so to have the projection plane coinciding with the physical screen (Figure 6.14(a)).

The projection matrix must be computed for each eye separately as follow:

$$P = \begin{bmatrix} \frac{2 \cdot near}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{2 \cdot near}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & \frac{near + far}{near - far} & \frac{2 \cdot near \cdot far}{near - far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where the parameters "*near*" and "*far*" are the distance of the *near plane* and

[1]Despite of the name its a rectangle.

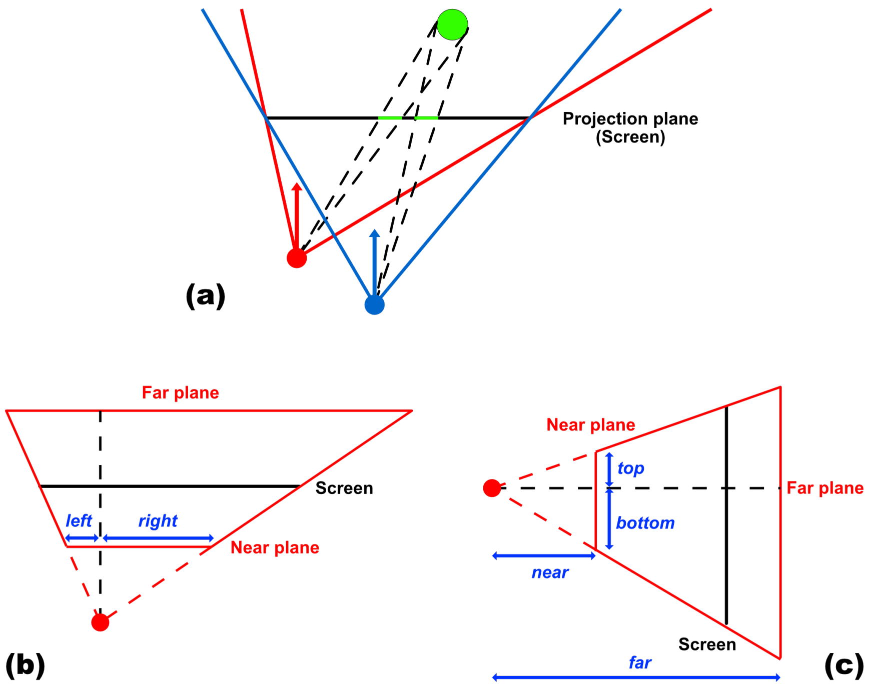


Figure 6.14: Off-axis perspective projection. (a) Asymmetric view frustums for head-tracked stereo rendering on a fixed monitor. Note that the view directions are kept orthogonal to the screen. (b) Visual representation of the parameters for the definition of a asymmetric view frustum, top view. (c) Same as (b) in side view.

far plane from the camera (usually chosen by the developer), while "*left*", "*right*", "*top*" and "*bottom*", "*near*" are the distances shown in Figure 6.14 (b) and (c). They can be easily computed (for each eye separately) by knowing the position of the eye and the position of corners of the screen. Finally, it is important to note that the view and projection matrices of the two eyes must be recomputed every time the user moves or turns his/her head.

For VR systems equipped with *multiple fixed* monitors (such as CAVE-like systems) the procedure to follow is the same: one (for monoscopic displays) or two (for stereoscopic displays) images have to be generated for each monitor, computing the proper view and projection matrices for each image (i.e. virtual camera) as explained so far.

In the case of *Head-Mounted Displays* (HMD) similar reasoning applies, but it must be taken into account the fact that the screen is not fixed anymore: instead it follows the movements of the user's head. For that reason, the projection matrices for the two eyes are fixed and depend on the physical characteristic of the display. As regard to the view matrices, they can be computed similarly to the case of regular (non-VR) 3D applications, by knowing position and orientation of the user's head, but also taking into account the offset of the eyes with respect to the tracked point associated to the user's head. Fortunately, HMD are usually shipped with a dedicated software library which directly provides both projection and view matrices.

6.5 Real-time rendering of semi-transparent surfaces

When representing molecular surfaces and isosurfaces, it is often useful to draw them as semi-transparent objects. In interactive computer graphics, semi-transparent objects are usually simulated using a technique known as “alpha blending”, first introduced by Porter and Duff [289]. In this technique, the color associated to each fragment contains an “alpha” component, representing its opacity. During the rendering process, if the “blending” is active, every time the color of a fragment is going to be stored in a location of the color buffer, it will be “blended” with the one already present at that location (instead of replacing it). In particular, the new color for the considered location of the color buffer is computed as a function of the old color, the fragment’s color, and their associated alpha values. The compositing function usually chosen to simulate semi-transparency is the following (Equation 6.2):

$$\begin{cases} C'_0 = & \alpha_0 \cdot C_0 \\ C'_n = & \alpha_n \cdot C_n + (1-\alpha_n)\cdot C'_{n-1} \end{cases}$$
$$\Longrightarrow \quad C'_n = \alpha_n \cdot C_n + \sum_{i=0}^{n-1} \underbrace{\left(\prod_{j=i+1}^{n} (1-\alpha_j)\right) \cdot \alpha_i \cdot C_i}_{\text{Not commutative with respect to i !}} \tag{6.2}$$

where C'_n is the color to be stored in the considered location of the color buffer (resulting from blending the color of n fragments), (C_0, α_0) are respectively the color and the opacity of the background and (C_n, α_n) are the color and the opacity of the *nth* fragment lying at the considered location of the color buffer.

This function was named “*OVER*” operator by Porter and Duff [289], but is commonly known as “back-to-front alpha blending”. In fact, as proven in Equation 6.2, the OVER operator is not commutative, so it requires to draw fragments in “back-to-front” order with respect to the camera. Ignoring such constrain may result in incorrect colors and/or graphical artifacts, as shown in Figure 6.15(a).

When graphics hardware had no or limited programmability support, this problem was usually handled according to one of the following strategies:

1. Sort transparent *triangles* in back-to-front order before drawing them.
2. Sort transparent *triangle meshes* in back-to-front order and drawing only

those triangles that are oriented towards the camera (by keeping "*backface culling*" active when drawing these meshes).

In both cases, in order to reduce the number of elements to be sorted, opaque objects are drawn in any order in a first rendering pass, while semi-transparent elements are sorted and drawn in a second rendering pass over the same color buffer (with blending active). The "depth test" must be kept enabled during the second rendering pass (since opaque surface can occlude transparent ones), but disabling writing on the Z-buffer (since transparent surfaces must not completely occlude other transparent surfaces). The first solution provides exact results in most cases, but sorting all the transparent triangles of the scene at every camera movement may be result in a major performance hit. The second solution has a much lower impact on performance, but produces roughly approximate results (see Figure 6.15(b)) and, depending on the scene, may still produce graphical artifacts.

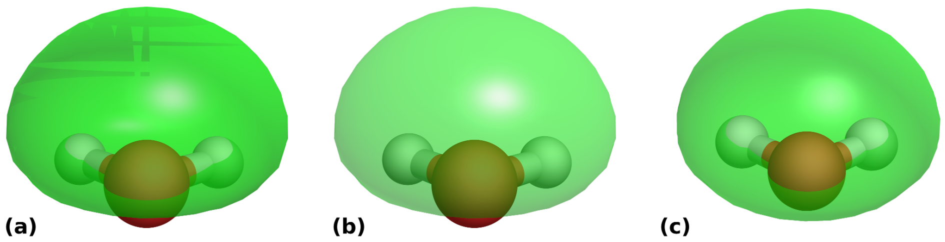


Figure 6.15: Half molecular orbital of a water molecule. Comparison between three different algorithms for simulating semi-transparent surfaces in real time. In all the three cases the polygons are drawn without ordering. (a) Example of graphical artifacts resulting from unordered alpha blending of front and back faces. (b) Unordered alpha blending of front faces only. (c) Weighted Blended Order-Independent Transparency [290].

After the advent of programable GPUs, many hardware-accelerated techniques have been proposed to render transparent surfaces without the need of sorting procedures at the application level. These techniques are usually referred as "*Order-Independent Transparency*" (OIT) methods and can be classified in two families: those producing exact alpha-blending by compositing the fragments in correct order (such as "*Depth peeling*" methods [291–293]) and those employing a commutative compositing operator in place of the *OVER* operator (such as [290, 292, 294]), thus renouncing to produce exact results in order to obtain better performance with respect to "exacts" methods. Interested readers can refer to [295] for survey on OIT methods.

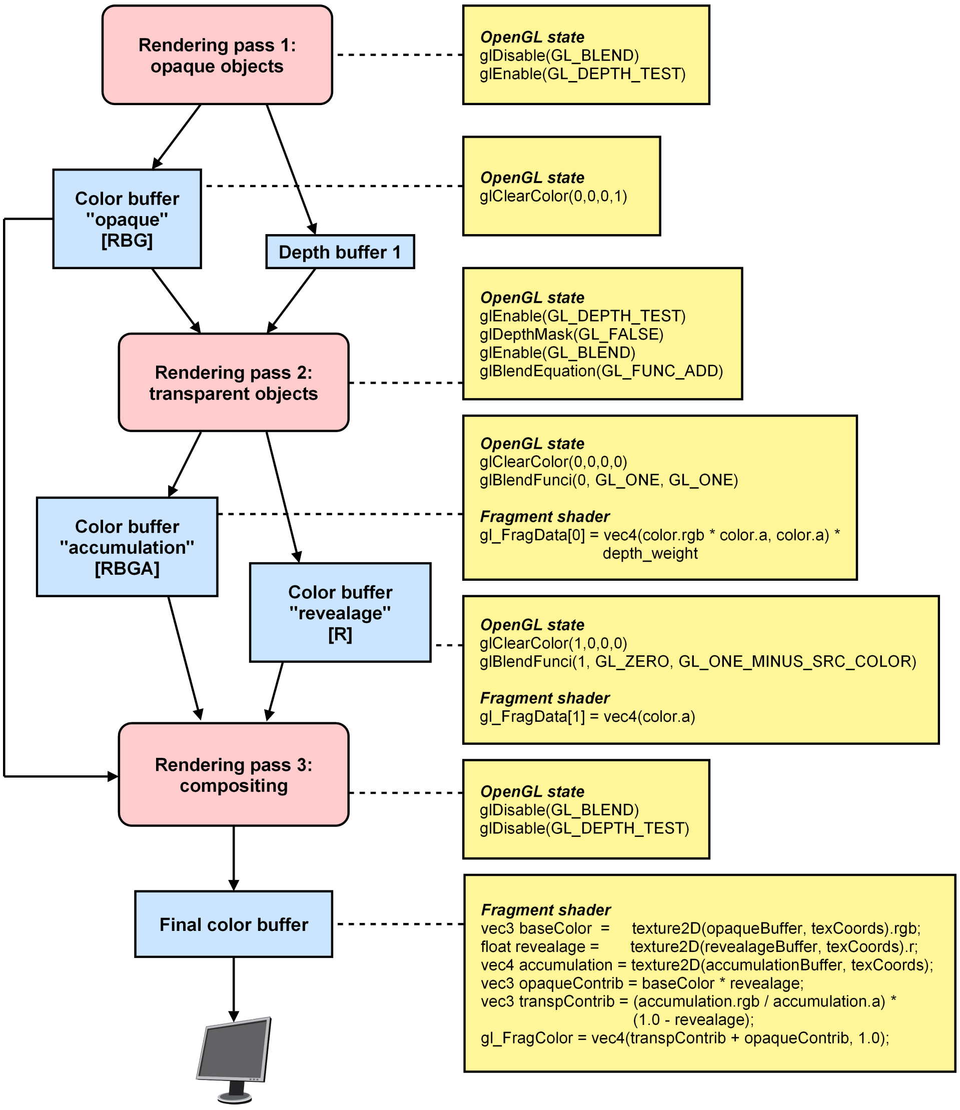
In Caffeine, we employed one approximated OIT method, the "*Weighted Blended*

Order-Independent Transparency" by McGuire and Bavoil [290] (see Figure 6.15(c)), since it provides a good balance between quality of the results, performance, and implementation complexity. The compositing operator used by the " *Weighted Blended Order-Independent Transparency*" is the following (Equation 6.3):

$$C'_n = \underbrace{\underbrace{\frac{\sum_{i=1}^{n} C_i \cdot \alpha_i \cdot w\left(z_i, \alpha_i\right)}{\sum_{i=1}^{n} \alpha_i \cdot w\left(z_i, \alpha_i\right)}}_{\text{Weighted mean of the colors}} \cdot \left(1 - \prod_{i=1}^{n} \left(1 - \alpha_i\right)\right)}_{\text{Contribution of semi-transparent objects}} + \underbrace{C_{opaque} \cdot \left(\prod_{i=1}^{n} \left(1 - \alpha_i\right)\right)}_{\substack{\text{Contribution of opaque objects} \\ \text{and of the background}}} \tag{6.3}$$

where C'_n is the color resulting from blending the color of n fragments, C_i and α_i are respectively the color and the opacity of the i-th fragment, C_{opaque} is the color (at the considered element of the color buffer) resulting from the rendering of the opaque objects (and of the background) and $w(z_i, \alpha_i)\epsilon[0, 1]$ is a monotone decreasing function used to weigh the contribution of a fragment as a function of its depth: the idea is that a fragment near the camera should have a higher weight than a far one. Some possible weighting functions are proposed by the authors in [290] and [296].

The diagram in Figure 6.16 summarize a possible implementation for this technique. As shown, the algorithm consists in three rendering passes. In the first pass the background and the opaque objects are rendered, storing the result in dedicated color and depth buffers. In the second pass only the semi-transparent objects are rendered. The aim of this pass is to accumulate the terms $(C_i \cdot \alpha_i \cdot w_i)$, $(\alpha_i \cdot w_i)$ and α_i of eq. 6.3 in two output color buffers (called "*accumulation*" and "*revealage*" in [290, 296]): each location of the "*accumulation*" buffer will contain the related $\sum(C_i \cdot \alpha_i \cdot w_i)$ in its RGB components and $\sum(\alpha_i \cdot w_i)$ in its alpha component, while each location of the "*revealage*" buffer will contain the related $\sum(\alpha_i)$ term. Note that the depth buffer outputted from the first pass is used to perform the depth test in this second rendering pass (having care to disable the writing on it). In fact, as explained before, opaque objects may occlude transparent ones. Finally, in the third rendering pass, the content of the "*opaque*", "*accumulation*" and "*revealage*" color buffers are combined to form the final image, according to the blending equation 6.3. Further implementation details and alternatives can be found on the original article [290] and in the blog of one of the authors [296].


Rendering pass 1: opaque objects
Color buffer "opaque" [RBG]
Depth buffer 1
Rendering pass 2: transparent objects
Color buffer "accumulation" [RBGA]
Color buffer "revealage" [R]
Rendering pass 3: compositing
Final color buffer

```
OpenGL state
glDisable(GL_BLEND)
glEnable(GL_DEPTH_TEST)
```

```
OpenGL state
glClearColor(0,0,0,1)
```

```
OpenGL state
glEnable(GL_DEPTH_TEST)
glDepthMask(GL_FALSE)
glEnable(GL_BLEND)
glBlendEquation(GL_FUNC_ADD)
```

```
OpenGL state
glClearColor(0,0,0,0)
glBlendFunci(0, GL_ONE, GL_ONE)

Fragment shader
gl_FragData[0] = vec4(color.rgb * color.a, color.a) *
                  depth_weight
```

```
OpenGL state
glClearColor(1,0,0,0)
glBlendFunci(1, GL_ZERO, GL_ONE_MINUS_SRC_COLOR)

Fragment shader
gl_FragData[1] = vec4(color.a)
```

```
OpenGL state
glDisable(GL_BLEND)
glDisable(GL_DEPTH_TEST)
```

```
Fragment shader
vec3 baseColor  =     texture2D(opaqueBuffer, texCoords).rgb;
float revealage =     texture2D(revealageBuffer, texCoords).r;
vec4 accumulation = texture2D(accumulationBuffer, texCoords);
vec3 opaqueContrib = baseColor * revealage;
vec3 transpContrib = (accumulation.rgb / accumulation.a) *
                     (1.0 - revealage);
gl_FragColor = vec4(transpContrib + opaqueContrib, 1.0);
```

Figure 6.16: Implementation scheme of the "*Weighted Blended Order-Independent Transparency*" method [290]. Rendering passes are shown in red. Input and output buffers for each rendering pass are shown in blue. Yellow boxes describes salient settings for the *OpenGL* state and/or sample pseudo-code of the fragment shader. These settings and code always refer to a rendering pass: in figure, when they are visually associated to a color buffer, it means that the rendering procedure must set the specified *OpenGL* settings and/or activate the shown shader when drawing to that buffer.

6.6 Caffeine's Frame Graph

Using the *Qt*'s [89] terminology , a "*frame graph*" is a Directed Acyclic Graph (DAG) describing a rendering procedure. While the "*scene graph*" defines an hierarchical structure describing the content of the three-dimensional scene (i.e. *what* to render), the frame graph describes the sequence of rendering passes, the associated *OpenGL* state and their input/output buffers required to produce a final image of the 3D scene (i.e. *how* to render). Figure 6.16 actually shows the frame graph of a possible implementation of the *Weighted Blended OIT*. The frame graph used by *Caffeine* is very similar and it is shown in Figure 6.17. In particular, support for *keystoning* and "*Fast Approximate Anti-Aliasing*" (FXAA) [285] has been integrate into the frame graph. Both keystoning and FXAA are implemented as "post-processing" effects, that is they operate on the 2D image resulting from the actual rendering of 3D geometries and produce a modified 2D image. It follows that they must be inserted as final stages of the frame graph. In order to reduce these stages (and thus optimize the rendering performance), OIT compositing and keystoning have been integrated in the same rendering pass, i.e. they are performed by the same fragment shader. The final image is written in the "*Frame Buffer Object*" (FBO) used by the Qt OpenGL window (QOpenGLWidget instance), so to be shown on screen.

If multiple "*views*" of the 3D scene are displayed, as in the case of stereoscopic displays and/or multi-screen installations, a frame graph is instantiated for each view. In fact, each "*view*" use a different virtual camera, usually positioned, oriented and/or "tuned" in a different way, so the whole rendering process must be performed separately for each view in accordance to the related viewing parameters. Actually, multiple windows could be created and different views could display different 3D scenes. Although supported by the implementation, these features are not currently exploited in *Caffeine* yet.

It should be noted that when only opaque geometries are rendered, the operations performed by the rendering pass 2 of Figure 6.17 and part of those performed by the rendering pass 3 are useless. While this don't usually constitute a problem in a desktop environment, it may results in a noticeable performance hit in CAVE-like systems driven by a single machine (like the one installed at SNS), where the whole rendering process is performed twice for each screen. Therefore, *Caffeine* periodically verifies if the scene graph contains transparent objects. If not, each frame graph is modified at run-time in order to bypass the second rendering pass of

Figure 6.17 and the compositing code of the third rendering pass. If a transparent object is later added to the scene graph, OIT related operations are restored. In order to distinguish between opaque and transparent objects within the scene graph, the developer is asked to "mark" the related nodes of the scene graph with a proper *bit mask*. This *bit mask* is also checked in the first rendering pass to draw only opaque objects and in the second pass to draw only transparent objects.

As said, the OIT compositing code of the fragment shader of the rendering pass 3 is bypassed if the scene graph does not contain transparent objects. The same happens for the shader code related to keystoning and anti-aliasing. In fact, the CAVE version of *Caffeine* allows to disable these features by means of the configuration tool presented in section 5.4.1, while keystoning is always disabled in the desktop version. In order to alter the behavior of these shaders without writing multiple versions of they (one for each possible configuration) and without introducing in the code conditional branches evaluated at run-time (such as the *if* statement, that are known to affect shaders performance), *conditional preprocessor directives* (such as *#ifdef* , *#else* , *#endif* etc.) have been used, so to exclude unused code during the compilation. However, although *GLSL* compilers provide a preprocessor supporting conditional directives, there is no standard way to externally define identifiers at compile-time (something similar to the "-D" option supported by C/C++ preprocessors). Fortunately, OpenSceneGraph 3.4 [262] introduced a new feature called "*#pragma(tic) composition*" of shaders, that allows (among other things) to define preprocessor macros from C++ code and to preform an automatic re-compilation of the shader as a consequence of such definitions.

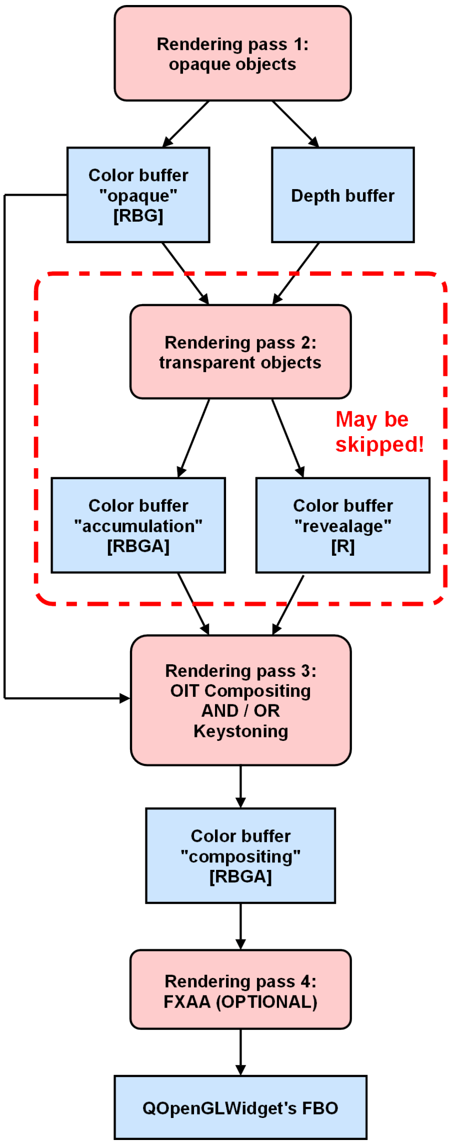


Figure 6.17: *Caffeine*'s frame graph. Rendering passes are shown in red. Input and output buffers for each rendering pass are shown in blue.

7 Included Paper: "Immersive virtual reality in computational chemistry: Applications to the analysis of QM and MM data"

In this chapter, I included the latest paper regarding *Caffeine*, published by the International Journal of Quantum Chemistry in 2016 [7]. Apart from introducing *Caffeine* and its main features (implemented at the time of writing), the paper present some benchmarks conducted to test the performance obtained by *Caffeine* when running on a desktop workstation and in a CAVE system, and to compare them with the ones obtained by VMD [71] (running on a desktop workstation). Finally, the paper discuss some case studies specifically conducted to illustrate the usefulness and advantages that can be gained by analyzing the resulting data in a VR environment with *Caffeine*.

Received: 18 April 2016 | Revised: 21 June 2016 | Accepted: 24 June 2016

DOI 10.1002/qua.25207

SOFTWARE NEWS & UPDATES

WILEY International Journal of QUANTUM CHEMISTRY

Immersive virtual reality in computational chemistry: Applications to the analysis of QM and MM data

Andrea Salvadori | Gianluca Del Frate | Marco Pagliai | Giordano Mancini | Vincenzo Barone

Scuola Normale Superiore, Piazza dei Cavalieri 7, Pisa I-56126, Italy

Correspondence
Giordano Mancini, Scuola Normale Superiore, Classe di Scienze, Piazza dei Cavalieri 7, Pisa, Italy.
Email: giordano.mancini@sns.it

Funding Information
Support from the Italian MIUR (FIRB 2012 Project no. RBFR12ETL5) is acknowledged. The research leading to the results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement n. [320951].

Abstract

The role of Virtual Reality (VR) tools in molecular sciences is analyzed in this contribution through the presentation of the Caffeine software to the quantum chemistry community. Caffeine, developed at Scuola Normale Superiore, is specifically tailored for molecular representation and data visualization with VR systems, such as VR theaters and helmets. Usefulness and advantages that can be gained by exploiting VR are here reported, considering few examples specifically selected to illustrate different level of theory and molecular representation.

KEYWORDS
data interaction, molecular viewers, virtual reality

1 | INTRODUCTION

A detailed, yet compact, representation of molecular structures, together with the inclusion of related properties in formulas and graphs, has always been at the heart of chemistry. Representation plays a key role in the whole discovery process, conveying information to human inspectors, relying on human pattern recognition, and suggesting innovative points of investigation and new, previously unexplored scenarios.[1] From a theoretical chemistry perspective, without molecular graphics, the sheer amount of information provided by current computational power would rather hinder true knowledge acquisition.[2] The importance of molecular graphics in chemistry is demonstrated by its leading role in the adoption of advances in computer graphics for scientific visualization.[3] The evolution of computer technologies for three dimensional immersive virtual reality (IVR) allows nowadays to achieve a further evolution in data representation and visualization.[4] In fact, it is now possible to create 3D virtual environments that extend users perception and increase researchers ability to quickly tackle massive amounts of data coming from multiple and different sources. Within such systems, users can directly interact with visualized data (by means of dedicated devices) in a more natural and friendly way than that achievable on desktop systems with mouse and keyboard.[5,6] IVR technologies include a large panel of devices, from cheap consumer grade ones to very costly specialized hardware. In the first category, we can mention interactive sensors like the *Microsoft Kinect*[7] and the *Leap Motion*,[8] current generation immersive helmets such as the *Oculus Rift*[9] and the *Vive* from *HTC* and *Valve*,[10] or force-feedback devices like the *Novint Falcon 3D Touch controller*.[11] The second category instead includes virtual theaters, such as the Cave Automatic Virtual Environment (CAVE),[12,13] equipped with high-precision tracking sensors and driven by one or more powerful workstations.

With the aim of enabling the employment of different platforms, ranging from desktop computers to more expensive IVR installations, we are developing a new molecular viewer, called *Caffeine*.[14] Beside motivations of performance and the aim to exploit state of the art technologies, one of the reasons for developing a new molecular viewer was not to force IVR features in an application designed within a different scope. Rather, we tried to design the software in such a way that the transition from a familiar 2D desktop environment to an IVR one would be as smooth and easy as possible. A specific feature to which we payed great attention in the development is coupling the visualization of molecular structures with the plotting of numerical data,

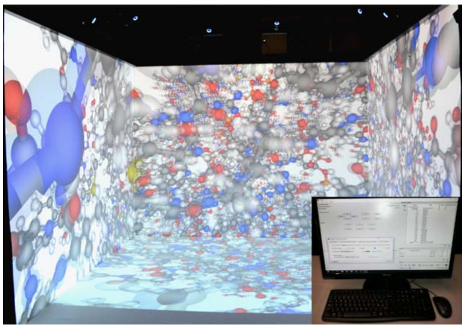


FIGURE 1 A ball-and-stick view of a macromolecular system represented in the CAVE installation at SNS with Caffeine. On the bottom right, the CAVE "control panel" is shown. The user interface of the control panel is, in large part, the same of the "desktop version" of Caffeine (see Figure 5)

for example, those related to the analysis of quantum mechanical (QM) calculations (e.g., the minimum energy path of chemical reactions) or classical simulations (such as the total energy related to a molecular dynamics [MD] trajectory).

In this article, we present the current status of Caffeine, showing its usage through the application to several test cases, which encompass many of the common levels of molecular representation and theory, including both QM studies and molecular mechanics (MM) investigations of large systems. We start from small, isolate molecules, on which high level calculations could be performed, and proceed to intricate macromolecular entities, thus involving different level of data complexity. The article is organized as follows: in a first part, some technical details of the Caffeine program and its versions for desktop and IVRs are presented; in the second part, the graphical representations of different case studies obtained with Caffeine are reported, to demonstrate the program capabilities in chemical visualization and in the analysis of results. We also make the case for potential benefits coming from the use of IVR in *setting up* simulations for medium size or large molecular systems through the examination of the dissociation mechanism of the intercalating drug doxorubicin (hereafter DOX) from a DNA fragment. General remarks and future perspectives on the development of our molecular graphics system are given in the conclusions of the article. Further technical details on the methods and algorithms employed for the visualization of molecular structures and isosurfaces are reported in the appendices.

It is important to highlight here that a rigorous user evaluation study is not our present priority. Here, we want to show the general opportunities offered by the adoption of IVR technologies in molecular modeling and, at the same time, illustrate the features of Caffeine to a wide audience.

2 | RELATED WORK

The representation of molecular structures by means of virtual reality (VR) technologies is not a new methodology: visualizations of atomistic simulations within immersive theaters were already reported at the middle of the 90s (see as an illustrative example the work done by Disz et al.[15]). Nevertheless, IVR tools did not know the diffusion that they deserved within scientific fields for at least a decade, partly due to limits of the underlying hardware and partly to the infancy of software using such technologies.[16] The growth of computer power in the last decade made possible to use IVR for rigorous scientific visualization. However, the adoption of IVR tools in molecular sciences is still an ongoing process, even if the usefulness in visualizing large systems of chemical interest (highlighting both structural and functional properties) within immersive environments has already been demonstrated.[17] Recently, Reda et al.[18] developed an application for the interactive visualization of MD simulations in ultra-resolution immersive environments, exploiting an hybrid representation which combines balls-and-sticks with volume rendering of approximate electron densities.

Among popular molecular viewers, VMD[19] supports several VR technologies[5] such as CAVE systems and ImmersaDesk,[20] using VR toolkits like FreeVR[21] and CAVElib.[22] Recently, Stone et al.[23] implemented an experimental version of VMD combining omni-directional stereoscopic visualization via head-mounted displays (Oculus Rift DK2), with ray-tracing rendering computed by a remote GPU cluster.

To confirm the interest in the use of VR environments, it is worth noticing that some commercial molecular graphics systems like Amira[24] and YASARA[25] support VR technologies. Also PyMOL[26] has a VR plug-in developed by Virtalis.[27]

3 | THE CAFFEINE MOLECULAR VIEWER

Caffeine is a new molecular viewer specifically designed and developed to take advantage of modern IVR technologies. It is implemented in C++, using the Qt framework,[28] Open Babel[29] as base cheminformatics library (we are evaluating its extension or substitution to provide a solution more suited to our needs; however Open Babel provided sufficient flexibility for the early stages of development), OpenSceneGraph[30] as 3D graphics engine, the OpenGL Mathematics library[31] and the Qt Widgets for Technical Applications (Qwt) library.[32] We use Stride[33] (as an external program invoked by our application) for detecting the secondary structures of polypeptides.

Caffeine can visualize both static and dynamic molecular structures (trajectories) read from PDB,[34] XYZ (xmol format),[35] and Gaussian Cube[36] files. Like most molecular viewer, Caffeine supports the most diffused graphical representations of molecular structures, such as "all-atoms" visualization (balls-and-sticks, licorice, and van der Waals spheres) and ribbon diagrams of polypeptides and polynucleotides. In addition, volumetric datasets such as electron densities and molecular orbitals can be imported from Gaussian Cube files and visualized as isosurfaces (several examples are presented in the Case Studies section). In the case of dynamic molecular structures, the graphic geometry is generated on the fly at each time-step, so to avoid to fill the graphic memory in the case of long trajectories. Although this is not a completely "*out-of-core*" solution (since the entire trajectory is completely loaded in main memory), it is a first step in that direction.

There are two main versions of Caffeine (sharing most of the code base and features): one for desktop computers and one designed for

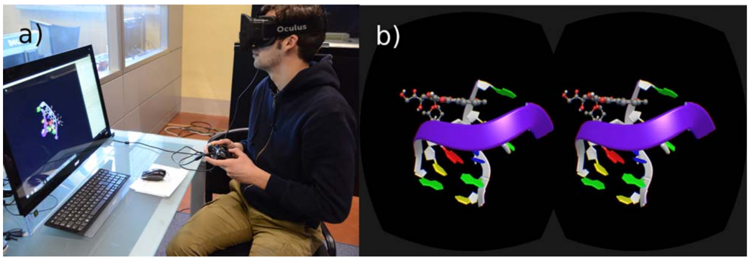


FIGURE 2 (a) User wearing the Oculus Rift DK1 helmet. (b) The DNA/DOX binary complex rendered for the Oculus Rift DK1 with an experimental version of Caffeine

VR systems equipped with multiple displays, such as the CAVE, a cubic room-sized IVR system whose walls (from three to six) are projected with stereoscopic images. Our CAVE installation has four walls/stereo projectors and it is equipped with an OptiTrack[37] tracking system (see Figure 1). The IVR version implements some functionalities which are specifically tailored for CAVE-like systems, like keystone correction of the generated images and the ability to adjust the view point and the projection parameters of the virtual camera as a function of the position and orientation of the user's head (provided by the tracking system), so to obtain a convincing and coherent stereoscopic visualization across the screens.[38] Within the CAVE the user can control the visualizer by means of a simple, "button-based," remote application for mobile devices (e.g., tablet), which allows to rotate, translate and scale the molecular system, and to control the playback of trajectories. The remaining features are accessible via a separate control window, which is displayed, in our installation, in a standard monitor placed outside the CAVE (but driven by the same workstation), as shown in Figure 1.

Note that the "CAVE version" of Caffeine can be configured to be used in systems having a different number, layout and kind (monoscopic or stereoscopic) of displays, with or without tracking systems, as long as all the monitors are driven by a single computer. In the past, our CAVE system was driven by a cluster of workstations (one for each projector) connected through a local network. This configuration required the development of distributed applications, which are notably harder to implement, debug, and extend as compared to their nondistributed counterparts. To facilitate the development of Caffeine, and to minimize the differences in the source code between the CAVE and the desktop version, we designed and developed a *distributed scene graph* library.[14] However, thanks to the latest advances in video card technology, we are now able to drive all the four projectors of our CAVE with a single computer equipped with multiple NVIDIA Quadro GPUs in scalable link interface (SLI)[39] configuration. This solution has allowed us to remove the distributed scene graph from the project, thus saving the time needed for its further improvements and extensions.

While CAVE-like systems are among the most advanced IVR systems available today, they are (very) expensive fixed installations. For that reason, they can be found only in few specialized research centers. However, thanks to the technological evolution, several companies are now developing VR headsets (primarily intended for the video game market) such as the *Oculus Rift*[9] or the *HTC/Valve "Vive."*[10] Thanks to their relatively low cost (less than a thousand US dollars) and good portability, these devices will probably gain a wide adoption in the next few years, not only in the consumer market but also for educational and research purposes. For these reasons, we are interested to support this kind of IVR helmets in Caffeine. Currently, the *Oculus Rift Development Kit 1* VR helmet is supported by an experimental desktop version of Caffeine (see Figure 2). This version is considered "experimental" since it belongs to an older development branch with respect to the current main version and it still lacks some of the latest features, such as charts and key-frames (*vide infra*). We plan to officially support some of these devices in the near future.

3.1 | Visualization of molecular structures

To obtain an interactive visualization of a large number of spheres and cylinders, resulting from the "all atoms" representation of complex molecular structures, we have developed a set of *GPU shaders* (A "*shader*" is a program running on the Graphics Processing Unit (GPU), which allows the programmer to define a specific algorithm for drawing the objects it is applied to) implementing a technique known as "*GPU-based ray-casting of quadric surfaces*." This method has been first introduced by Gumhold[40] to render a large number of ellipsoids representing symmetric tensor fields, and later generalized by Toledo and Lévy[41] so to ray-cast any quadric surface on the GPU. This technique provides very good results (both in terms of performance and image quality) when applied to molecular visualization[42–46] and, according to Kozlíková et al.[47] in a recent survey on the subject, it still represents the state of the art. A brief discussion on the implementation of this method, with specific reference to Caffeine, can be found in Appendix A.

As regards to ribbon diagrams, we implemented an algorithm for generating three-dimensional "paths," formed by a sequence of traits. Each trait is defined by two endpoints, an orientation (normal) vector, the shape of its transversal profile (circular or rectangular with rounded corners) and a set of attributes (such as sizes and colors) for each of its two ends. The traits can be rectilinear or they can be defined by cubic B-Splines, so to obtain a smoother path. In the latter case, also the normal vectors between consecutive traits are interpolated using the cubic B-Spline equation, similarly to what done by Krone et al.[49] in their algorithm for GPU-based ribbons visualization. Although our algorithm could be used to represent various kinds of information (e.g., field lines), it is currently used to generate ribbon diagrams of polypeptides

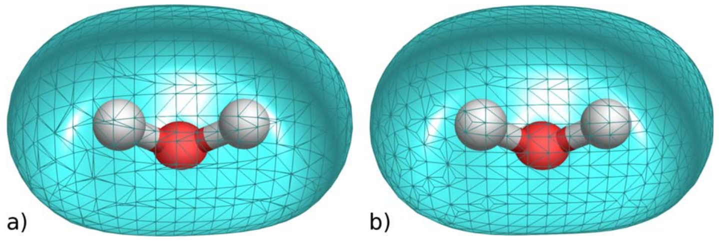


FIGURE 3 Representation of scalar volumetric datasets: comparison between the triangulation of a water molecular orbital generated by the Marching Cubes (a) and the one generated by the Surface Nets algorithm (b). Note that the traditional Marching Cubes generates many thin triangles, while Surface Nets provides a more regular tessellation. The orbitals have been computed at HF/STO-3G level of theory with Gaussian 09[56] suite of programs

and polynucleotides. For polypeptides, we followed the popular procedure described by Carson and Bugg[50]: the path is defined by the sequence of alpha-carbons in the chain, while the oxygen atoms of the backbone are used to compute the normal vectors of the peptide planes, which define the orientation of the traits. Shape, sizes, and colors of each trait are set according to the secondary structures to be presented. Path and normal vectors are finally smoothed using the B-Spline equation. In the case of polynucleotides, we used a similar procedure: phosphorus atoms replace the alpha-carbons in the definition of the path (with the exception of the first and of the last nucleotides of the chain, where the C5′ and C3′ atoms are used respectively as starting and ending points of the path), while the vector (C3′–C1′) of each nucleotide defines the normal vector of the corresponding trait.

3.2 | Isosurfaces extraction and visualization

Several molecular descriptors, such as electron density, electrostatic potential and molecular orbitals are actually examples of *volumetric datasets*. A volumetric dataset can be defined as a set of pairs $<P_i, V_i>$, where P_i is a point in the three-dimensional space and V_i is its associated value (e.g., a scalar value, a vector etc.).[51] The pair $<P_i, V_i>$ is called “voxel” (short for “volume element”). Volumetric datasets can be obtained by sampling the value of some function or measurable quantity at certain locations of the three-dimensional space. Although sampling locations could be chosen randomly, it is common to sample the data at uniformly spaced intervals, so to obtain a regular grid of voxels. It is noteworthy that, even if the values are sampled at discrete locations, it is still possible to approximate the value of a generic point lying inside a cell of the grid by interpolating the values of the eight vertices (voxels) of that cell.

At present, Caffeine visualizes scalar volumetric datasets only in the form of isosurfaces. User can chose between two extraction algoritms: the traditional *Marching Cubes*[52] and a simplified version of the *Surface Nets*.[53] With regard to the Marching Cubes we use (a slightly adapted version of) the popular implementation by Cory Gene Bloyd and Paul Bourke,[54] while for the Surface Nets we re-implemented in C++ the so-called “*Naive*” version by Mikola Lysenko[55] (originally coded in JavaScript). The Surface Nets algorithm produces a more regular triangulation than the traditional Marching Cubes method, as shown in Figure 3. Furthermore, according to some tests performed on our implementations, Naive Surface Nets results slightly faster. A brief description of these algorithms together with a first performance comparison of their implementation in Caffeine is reported in Appendix B. Further details on these and other extraction algorithms can be found in references 58 and 59.

When visualizing surfaces related to molecular properties, such as molecular orbitals, it is important to let the user perceive the relation between these surfaces and the molecular structure they refers to. For this reason surfaces are often drawn as semitransparent objects. In interactive computer graphics, semitransparent objects are usually simulated using a technique known as “*alpha blending*,” first introduced by Porter and Duff[60] in 1984 and nowadays supported natively by the graphics hardware. However, a simplistic use of *alpha blending* may lead to graphical artifacts, as discussed in Appendix C. Several GPU-accelerated techniques has been proposed to properly simulate semitransparent surfaces in real-time. These techniques are generally known as “order independent transparency” (OIT) methods. Interested readers can refer to reference 66 for a comprehensive survey on the subject. We employed a method called “*Weighted Blended Order-Independent Transparency*” by McGuire and Bavoil,[64,67] because it provides a good balance between quality of the results, performance, and implementation complexity.

In Caffeine, the user can choose between two different transparency modes for isosurfaces: the traditional uniform transparency (shown in Figure 4a), where all the fragments resulting from the rasterization of the iso-surface have the same opacity (chosen by the user), or a so-called “*smart transparency*” (shown in Figure 4b), where the opacity of each fragment is a function of the dot product between the normal and the “vector to the viewer” (The vector to the viewer is the normalized direction from the position fragment to the position of the virtual camera. More precisely, both the normal and the vector to the viewer are computed by the *vertex shader* on a per-vertex basis, interpolated by the graphics card, and finally passed to the fragment shader as per-fragment values), so to highlight surface edges while clearly showing the molecular structure behind the surface. This latest method, sometimes called “*X-Ray effect*,” is actually an old trick of computer graphics and it is employed also by other modern molecular viewers (such as Molekel[69] and Avogadro[70]) to represent molecular orbitals and surfaces. From our experience, “*smart*”/”*X-Ray*” transparency produces clearer, more understandable images with respect to

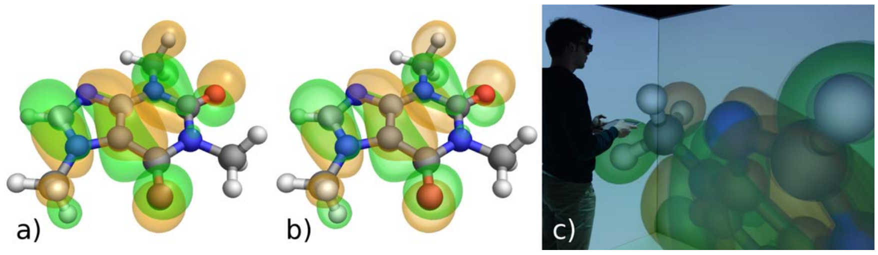


FIGURE 4 Highest Occupied Molecular Orbital (HOMO) of caffeine molecule represented as semitransparent isosurfaces (iso-value ±0.02) and drawn using the *Weighted Blended Order-Independent Transparency* method. (a) Uniform transparency: the entire surface has a constant opacity (35%). (b)"*Smart*" transparency: the regions of the surface which are orthogonal to the view direction have an lower opacity (20%), while the regions tangent to the view direction have an higher opacity (35%). (c) Same as (b) in the CAVE. Optimized molecular structure and HOMO have been calculated with PBE0 exchange and correlation functional[68] and 6-311++G(d,p) basis set with Gaussian 09[56] suite of programs

uniform transparency (see Figure 4), especially in VR systems requiring active shutter glasses (like our CAVE), where original colors are partially filtered/distorted.

3.3 | Caffeine desktop version

Figure 5a shows a first screenshot of Caffeine running as a standard desktop application, with specific hallmarks highlighted when drawing the DOX/DNA binary complex from a PDB file. Many different visualization features are available, like the possibility to modify ribbons settings and color styles.

Molecular simulations produce a lot of numerical data related to the physicochemical properties of the system taken into account. These measurements, if referred to two-dimensional scalar quantities, can be passed as input to Caffeine (using a specific file format) which plots them as line charts (as shown in Figure 5b). Furthermore, if a linear relationship exists between the measured quantity and the snapshots of a trajectory (i.e., if the quantity is a function of time), this relationship can be explicated in the dataset file. By doing so, a marker is drawn on the line chart during the playing of the trajectory, showing the value of the measured quantity in the currently displayed snapshot (see the top line chart shown in Figure 5b). This direct correlation between the currently represented structure and charted data encourage the user to exploit his "chemical intuition" and pinpoint any perceived interesting feature in the displayed system.

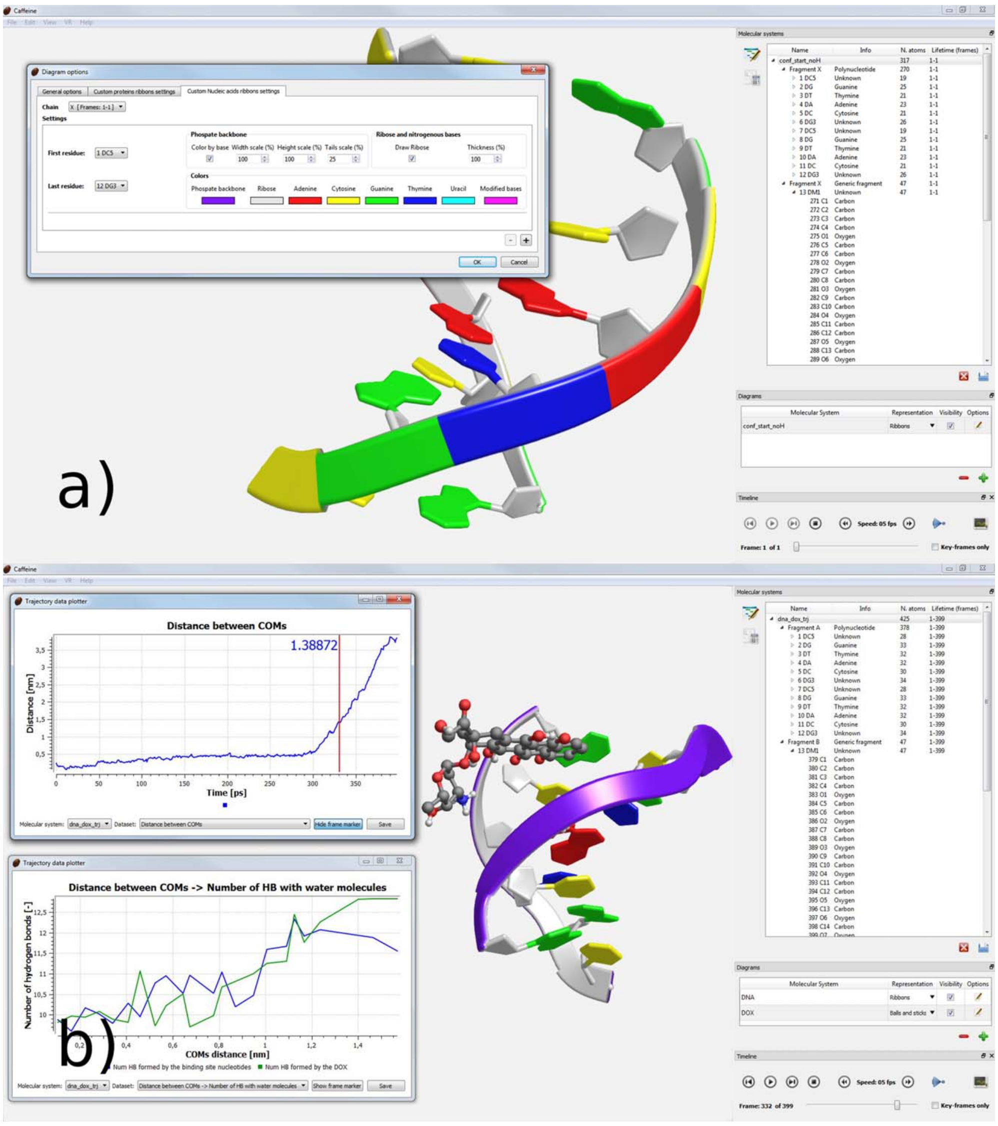
In addition, it is possible to "mark" a subset of the frames of the trajectory, which are deemed to be relevant for the study of the system. These "*key-frames*" may be either supplied using an additional input file or selected within Caffeine. Then, the user can (re-)visualize either the entire trajectory or only these selected key-frames. In both cases, the visualization can be performed as an animation or by explicitly skipping from one (key-)frame to the previous/following one of the sequence. Key-frames may represent particularly relevant conformations along a single trajectory but may also come from different data sources by assembling in one artificial trajectory, for example, the results of a clustering analysis over related systems. This allows for an interactive filtering of a trajectory or of any dataset since it makes it possible to associate different conformations to the same dataset and test for different hypotheses. A word of caution is needed here: while the user is free to associate key-frames to supporting data, these models must (currently) contain the same (sub-)set of atoms, and this requires a limited manipulation of the PDB files.

3.4 | Caffeine within an IVR environment

In immersive environments a new feedback, *proprioception*, is added to the perception of data. Proprioception is the capability to perceive and recognize the position of the own body in space, even without sight: the kinesthetic inputs from mechanoreceptors in muscles, tendons and joints, contribute to the human perception of limb position and limb movement in space.[72] Proprioceptive sense helps the user, without conscious efforts, to understand and evaluate the geometric properties of the visualized objects.

Within the CAVE, 2D data charts are drawn in the 3D scene in front of the user, and follow the movements of the user's head in a way similar to an augmented reality content (see Figure 6). As shown, only one chart is displayed, notwithstanding the ability of the user to switch interactively between the available charts by means of the remote application. The use of key-frames, by itself a useful feature, becomes critically important within an IVR environment since it allows the user to concentrate on the most important properties. As already stated in previous sections, the user is able to interact with the projected system through a mobile device (i.e., a tablet), so to regulate the displayed data according to the need (Figure 6b). The mobile application currently allows the user to rotate, translate and scale the molecular system, and to control the playback of frames.

As an example of the use of Caffeine with another type of IVR system, we can mention the screenshot of the DNA/DOX binary complex rendered for the *Oculus Rift DK1* already shown in Figure 2. In this case the user can interactively play the trajectory with the keyboard or, using a little trick, with a gamepad (to use the joystick an external program that maps gamepad input to keyboard events, such as JoyToKey,[73] is needed).



Chain: X [Frames: 1-1]

Settings

First residue: 1 DC5

Last residue: 12 DG3

Phospate backbone

Color by base	Width scale (%)	Height scale (%)	Tails scale (%)
✓	100	100	25

Ribose and nitrogenous bases

Draw Ribose	Thickness (%)
✓	100

Colors

Phospate backbone	Ribose	Adenine	Cytosine	Guanine	Thymine	Uracil	Modified bases
🟪	⬜	🟥	🟨	🟩	🟦	🟦	🟪

Molecular systems

Name	Info	N. atoms	Lifetime (frames)
conf_start_noH		317	1-1
Fragment X	Polynucleotide	270	1-1
1 DC5	Unknown	19	1-1
2 DG	Guanine	25	1-1
3 DT	Thymine	21	1-1
4 DA	Adenine	23	1-1
5 DC	Cytosine	21	1-1
6 DG3	Unknown	26	1-1
7 DC5	Unknown	19	1-1
8 DG	Guanine	25	1-1
9 DT	Thymine	21	1-1
10 DA	Adenine	23	1-1
11 DC	Cytosine	21	1-1
12 DG3	Unknown	26	1-1
Fragment X	Generic fragment	47	1-1
13 DM1	Unknown	47	1-1
271 C1	Carbon		
272 C2	Carbon		
273 C3	Carbon		
274 C4	Carbon		
275 O1	Oxygen		
276 C5	Carbon		
277 C6	Carbon		
278 O2	Oxygen		
279 C7	Carbon		
280 C8	Carbon		
281 O3	Oxygen		
282 C9	Carbon		
283 C10	Carbon		
284 O4	Oxygen		
285 C11	Carbon		
286 C12	Carbon		
287 O5	Oxygen		
288 C13	Carbon		
289 O6	Oxygen		

Diagrams

Molecular System	Representation	Visibility	Options
conf_start_noH	Ribbons	✓	

Frame: 1 of 1

Molecular system: dna_dox_trj Dataset: Distance between COMs

Molecular system: dna_dox_trj Dataset: Distance between COMs -> Number of HB with water molecules

Molecular systems

Name	Info	N. atoms	Lifetime (frames)
dna_dox_trj		425	1-399
Fragment A	Polynucleotide	378	1-399
1 DC5	Unknown	28	1-399
2 DG	Guanine	33	1-399
3 DT	Thymine	32	1-399
4 DA	Adenine	32	1-399
5 DC	Cytosine	30	1-399
6 DG3	Unknown	34	1-399
7 DC5	Unknown	28	1-399
8 DG	Guanine	33	1-399
9 DT	Thymine	32	1-399
10 DA	Adenine	32	1-399
11 DC	Cytosine	30	1-399
12 DG3	Unknown	34	1-399
Fragment B	Generic fragment	47	1-399
13 DM1	Unknown	47	1-399
379 C1	Carbon		
380 C2	Carbon		
381 C3	Carbon		
382 C4	Carbon		
383 O1	Oxygen		
384 C5	Carbon		
385 C6	Carbon		
386 O2	Oxygen		
387 C7	Carbon		
388 C8	Carbon		
389 O3	Oxygen		
390 C9	Carbon		
391 C10	Carbon		
392 O4	Oxygen		
393 C11	Carbon		
394 C12	Carbon		
395 O5	Oxygen		
396 C13	Carbon		
397 O6	Oxygen		
398 C14	Carbon		
399 O7	Oxygen		

Diagrams

Molecular System	Representation	Visibility	Options
DNA	Ribbons	✓	
DOX	Balls and sticks	✓	

Frame: 332 of 399

FIGURE 5 Screenshots of the current Caffeine desktop version (running in a Windows environment), used in this work for pictures realization and data analysis. The icons used in the program belongs to the Oxygen Project.[71] DNA nucleobases are colored by type. (a) Many interesting features are highlighted, such as the possibility to modify DNA ribbons appearance, according to the user needs. (b) The 2D scalar datasets associated to the molecular system are plotted as line charts. The bottom chart shows the same data of Figure 10d, while the top chart shows the distance between the centers of mass of DOX and of the DNA binding site. In the latter, the possibility to investigate the variation of the scalar quantity frame-by-frame is pointed out

3.5 | Performance evaluation

To evaluate the performance of Caffeine, we performed a benchmark to compare it with VMD.[19] The comparison is relative to the frame per seconds reached by both visualizers when rendering both static and dynamic molecular systems. It is important to note that, while obtaining a high frame rate on desktop is not an essential feature for the user, in the case of IVR systems this is critical to preserve the sense of immersion.

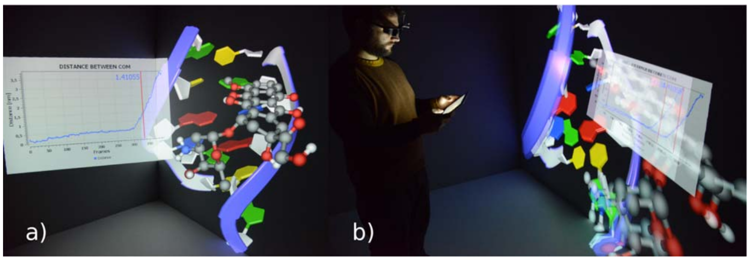


FIGURE 6 Dissociation of DOX from the DNA binding site. (a) Simultaneous representation of charted data and molecules. The binary complex is on the right while a graph showing the distance between COMs is visible on the left with the red marker highlighting the current frame and distance value. For sake of clarity, the stereo mode of projectors was temporarily disabled to shoot this photo. (b) User interacting with the DNA/DOX binary complex using a tablet, the graph is the same of the previous panel

Both applications have been configured so as to produce screen images as close as possible. In particular, this includes:

- Manually setting the "display" options and the transformation matrices of VMD to match the view and projection parameters chosen in Caffeine. Where that was not possible (i.e., the camera position of VMD is not configurable neither via GUI nor via scripting), we changed the configuration of Caffeine instead.
- Disabling "depth cueing" and axis rendering in VMD, since these features are not available in Caffeine.
- Setting the "render-mode" of VMD to "GLSL," so as to enable ray-casting of spheres and high-quality per-pixel lighting of geometry[74] (similarly to Caffeine).
- For static molecular systems, we also enabled the so-called "cache-mode" option of VMD, so as to use a display list caching mechanism to accelerate rendering of static geometry[74] (although we did not notice any variation in performance).

The computer on which the benchmark was performed is equipped with two Intel Xeon E5462 processors with a clock frequency of 2.8 GHz, 24 GB of RAM, a NVIDIA Quadro 6000 GPU, and Windows 7 Professional as operating system. Images were rendered full-screen, at a resolution of 1920 × 1080 pixels. Actually, VMD had a little advantage here, since we were not able to hide the title bar of its rendering window, so it rendered images at a slightly lower resolution.

Table 1 summarizes the frames per seconds rendered by the desktop versions of the two molecular viewers in the case of static molecular structures. To perform this benchmark we chose two medium-large assemblies from the Worldwide Protein Data Bank: (i) 1AON[75] composed by 58,870 atoms (without hydrogens) and (ii) 5AOO[76] composed by 356,280 atoms (without hydrogens). These two systems have been drawn according to the "Space filling" (van der Waals spheres), "Balls-and-Sticks" and "Ribbons" representations (and their equivalents in VMD: "VDW," "CPK," and "New Cartoon"). In VMD, we used the default quality settings for each representation, apart from the use of ray-casted spheres in place of tessellated ones. In Caffeine it is possible to specify quality settings only for ribbon diagrams, and we set them to the maximum quality. As one can see from Table 1, Caffeine is significantly faster than VMD when drawing static molecular systems.

Table 2 shows the results obtained during the visualization of a trajectory related to the dissociation of DOX from the DNA binding site (presented later as a case study). The system is composed by a fragment of DNA (378 atoms), a DOX molecule (69 atoms) and 8987 molecules of water (26,961 atoms). The trajectory contains 201 frames. For this system, we employed a mixed representation: ribbons for DNA ("New Ribbons" in VMD), balls-and-sticks for the DOX ("CPK" in VMD) and licorice for the water molecules. Furthermore we disabled the "cache-mode" option of VMD, as suggested by the user's guide[74] when dealing with trajectories. Both viewers have been configured to play a frame of the trajectory for each rendering frame, to reveal the maximum frame rate for the trajectory. In this scenario, Caffeine is slightly slower than VMD, even if the obtained frame rates have the same same order of magnitude. However, it is clear that visualization of dynamic structures in Caffeine, although fast enough in many cases, needs further optimization.

Tables 3 and 4 show the performance (min/max frames rate) of Caffeine in our CAVE system, when visualizing static and dynamic data. Overall, performances are comparable to those obtained on the

TABLE 1 Comparison of the performance between the desktop versions of Caffeine and VMD when visualizing static molecular structures

		Molecular system	
		1AON	5AOO
	Number of atoms	58,870	356,280
	Representation	fps	fps
Caffeine	Space filling	87	55
	Balls & Sticks	185	71
	Ribbons	273	128
VMD	VDW	17	3
	CPK	12	2
	New Cartoon	230	7

The results are expressed as frames per seconds. Different graphical representations have been considered. In VMD we chose the representations that more closely resemble those of Caffeine.

TABLE 2 Comparison of the performance between the desktop versions of Caffeine and VMD when visualizing a trajectory resulting from molecular dynamics

	DNA-DOX complex trajectory Max trajectory frames per seconds
Caffeine	12
VMD	17

The tested system is the DNA-DOX complex simulation of a single umbrella sampling window (more details in Case Studies section). The results are expressed as the maximum trajectory frames displayed in a second.

desktop system, thus allowing a comfortable and fluid immersive experience (although presenting the limits reported above for trajectories). The computer driving our CAVE is equipped with two Intel Xeon E5645 processors with a clock frequency of 2.4 GHz, 24 GB of RAM, two NVIDIA Quadro M6000 GPUs in SLI configuration, and Windows 10 Enterprise as operating system.

3.6 | Case studies

In this section, we show some useful graphic representations of simulation results obtainable with Caffeine. In the first section, we include a spin-density evaluation of 2,2,6,6-tetramethylpiperidine-1-oxyl-4-amino-4-carboxylic acid (TOAC).[77] In the second one, we used the results of a recent study on Cytochrome P450 2B4.[78] As a last case study, we present the whole DNA-DOX investigation, from the simulation setup performed within the CAVE to the final analysis.

3.7 | Spin density visualization

The analysis of isosurfaces, obtained by QM calculations or classical simulations, is particularly important in molecular sciences. In fact, the graphic representation of molecular orbitals, electron densities or electron localization functions provides a valuable help to characterize structural properties, to describe molecular interactions and to interpret spectroscopic properties. Also in the case of classical simulations the availability of volumetric datasets allows to visualize a series of useful properties such as, for example, electrostatic potential, molecular cavities (see Figure 9) or average density/occupancy (e.g., when calculating spatial distribution functions) near a selected site. As a first example of volumetric data we present the results obtained for the 2,2,6,6-tetramethylpiperidine-1-oxyl-4-amino-4-carboxylic acid (TOAC) molecule, which is characterized by

TABLE 3 Performance of Caffeine in visualizing large assemblies (PDB 5AOO) in our CAVE system; minimum and maximum frame rate for van der Waals, Balls-and-Sticks, and Ribbons representations

	Min fps	5AOO Max fps
Space filling	31	73
Balls-and-Sticks	35	62
Ribbons	51	85

TABLE 4 Performance of Caffeine in visualizing a trajectory resulting from molecular dynamics (DNA-DOX complex) in our CAVE system; minimum and maximum frame rate

DNA-DOX trajectory	
Min fps	**Max fps**
10	12

the presence of a nitroxide moiety and by the possibility to be inserted into polypeptide chains substituting a natural amino acid.[77,79] It is a stable radical, which can be employed as probe in electron spin resonance measurements, allowing to obtain useful information on the conformation of the studied peptide molecule. Density functional theory (DFT) calculations have been revealed effective in the description of the electronic structure of TOAC,[77,80,81] allowing to determine that the unpaired electron occupies an anti-bonding π^* molecular orbital localized on the oxygen and nitrogen atoms of the nitroxide moiety (Figure 7a). The correct computation of the TOAC electronic structure is particularly important, because the magnetic properties, which allow to employ this molecule as spin label, are ruled by its singly occupied molecular orbital (SOMO). A similar method to visualize the unpaired electron localization is through the spin density, sketched in Figure 7b. It is interesting to note (see Figure 7c) that the spin density is essentially the same also introducing TOAC in a polypeptide chain, confirming the importance of this radical in conformation studies.

3.8 | Structure and dynamics of large systems

Enzymes belonging to the Cytochrome P450 hemoproteins family are devoted to the oxidation of a wide range of organic compounds, from drugs to environmental pollutants. Hydrophilic channels, connecting the active site to the protein surface allow the buried heme group to react with the various substrates.[82] The role of the conserved Phe429 on the catalytic activity has been thus investigated through extensive classical MD simulations and clustering analysis of the wild type (WT) together with other four different mutants, highlighting structural and hydrogen bonding observable differences.[78,83] The single point mutation was identified to be responsible of several long-range effects, including the topology of the functional aqueous accesses to the catalytic site. To get light on this, each MD frame of each of the five mutant MD simulations was analyzed to find possible tunnels regulating the access to the heme ion. In a second step such identified tunnels were connected to real pathways by means of an average linkage clustering technique. A total of four major channels were found. In particular, one major pathway was detected in all the five MD simulations, featuring different average bottleneck radius values as we proceed from the WT (lower average radius observed) to the mutants. The different behavior among the mutants was explained in terms of subtle alterations in the hydrogen bond network, that propagated along the systems and affected the whole geometries.

Since IVR environments are well suited for deep investigations on geometries of large systems, we present feasible applications of our immersive tool in the visualization of different cluster members,

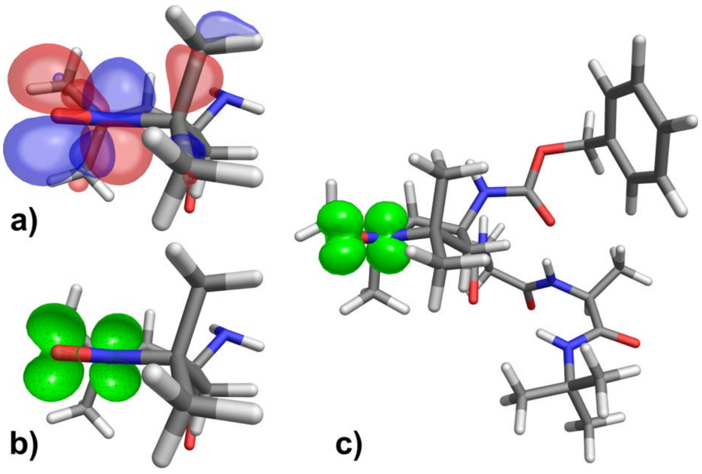


FIGURE 7 (a) and (b) represent SOMO orbital and spin density of TOAC, respectively, while (c) is the spin density of the Z-TOAC-(l-Ala)$_2$-NHtBu. Structural and electronic properties have been computed at PBE0/6-311++G(d,p) level of theory with Gaussian 09[56] suite of programs

pointing out the detection of cluster centroids and the structural differences between them (see Figure 8). Each MD frame, previously included in one specific cluster, is showed together with an index, which specifies its cluster membership. The user can switch from one cluster to another one, and select in this really feasible and nice way eligible key-frames to represent cluster centroids or outlying "extreme" conformers in clusters, to describe the differences within the obtained clusters and the relationships between the various mutations. The detected channels can also be visualized in Caffeine thanks to the support for Gaussian Cube file format.[36] To be supported in Caffeine, the channels identified in the previous study have been thus converted in Guassian Cube format, thanks to an *in-house* script, which computes density values using a simple Gaussian function taken from the literature.[84] As shown in Figure 9b, the difference and peculiarities among the mutants can be highlighted by displaying simultaneously the relative structures within the CAVE. Such investigations could be easily performed thanks to IVR technologies: on the contrary, the complex structures of this dataset could lead to unpleasant misunderstanding using a 2D computer desktop, mostly when the same analysis is explained to nonexperts.

3.9 | Dissociation of DOX from the DNA binding site

Intercalating drugs act as inhibitors of Topoisomerase I or II (or both). DOX is an antraciclynic intercalating drug whose structure can be divided in (i) a planar hydrophobic part, constituted by an antraquinone ring system, and (ii) a hydrophilic aminosugar moiety. Intercalating drugs bearing fused (hydrophobic) ring systems can insert between base pairs through the creation of favorable π-stacking with nearby nucleobases.[85] We report in this section, a study about the unbinding process of DOX from DNA, to show the effectiveness of IVR tools within a computational research project. The unbinding process of DOX from the binary complex was here investigated with umbrella sampling,[86] using the distance between the center of mass (COM) of

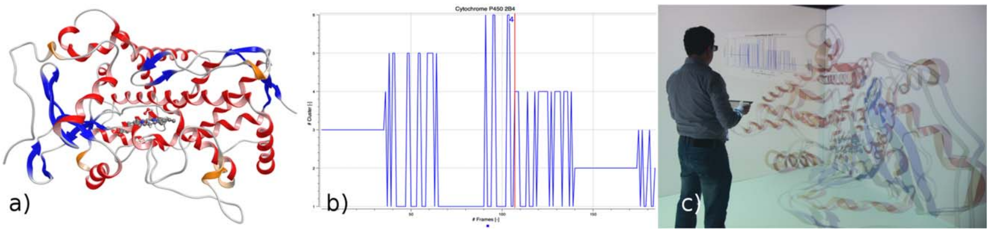


FIGURE 8 (a) Cytochrome P450 2B4 WT structure shown using ribbons. The heme group and OOH$^-$ anion are shown in balls-and-sticks. (b) Distribution of conformations in clusters along the artificial trajectory created by sampling structures from the original WT and the four mutants simulations. Less frames were used with Caffeine, obtained from a uniform re-sample of the original clusters. (c) Comparison of the differences between the obtained clusters using virtual reality with Caffeine

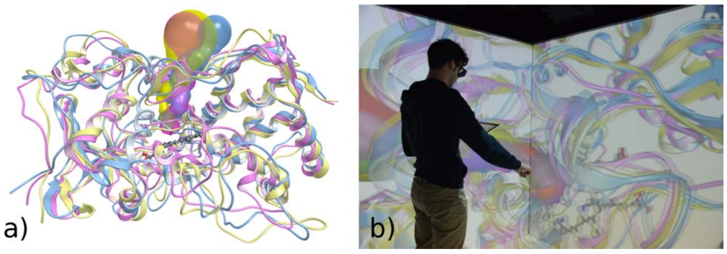


FIGURE 9 Superposition of representative structures, together with relative water channels, of three different mutants of Cytochrome P450 2B4, both on a standard desktop (a) and within the CAVE (b)

DOX and of the binding site as the reaction coordinate. The selection of the starting configurations for the umbrella windows was performed with our IVR environment: the ability to view, at the same time, a molecular conformation and the chart reporting the distance between COMs was exploited to select sensible structures in a very accurate way (see Figure 6).

These structures were marked as key-frames and used for the subsequent umbrella simulations (details about the simulations are

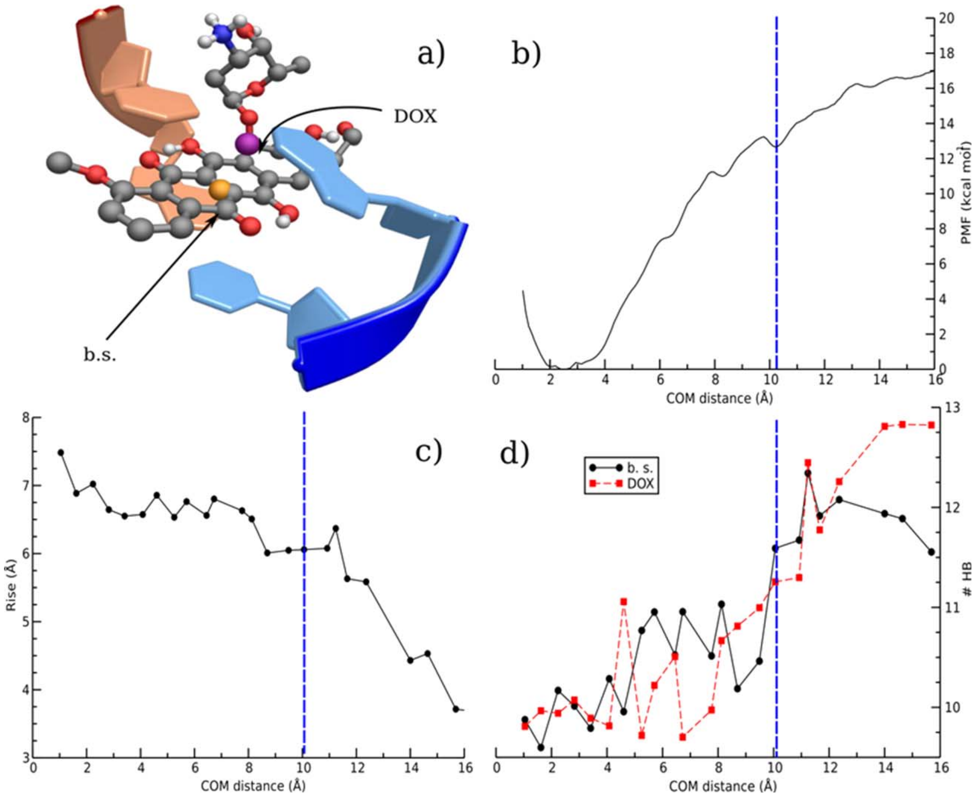


FIGURE 10 Binary complex dissociation process. Note that in panels (b-d) the position of the intermediate at 10 Å and final dissociated state along the sampling coordinate is highlighted with a blue dashed line. (a) Binding site of the DOX compound in the initial conformation. The position of the binding site COM (b.s., gold sphere) and of DOX (purple sphere) is shown. (b) Potential of mean force (PMF) curve associated to the distance between centers of mass of the binding base pairs and of the DOX drug. (c) Calculated change in rise between DNA base pairs. (d) Number of hydrogen bonds between the binding site nucleobases (black) or DOX (red) with water molecules

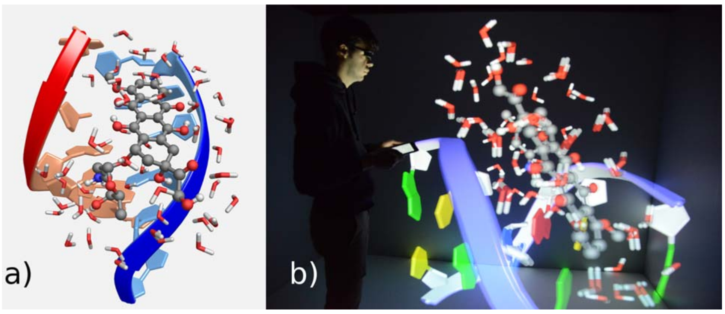


FIGURE 11 (a) Intermediate bound state of DOX with the nearest neighbor water molecules shown as licorice. (b) Same as (a), within the CAVE

reported in the Supporting Information). The obtained free energy ΔG is represented in Figure 10b. Interestingly, a partially stable state was found at 10.2 Å: here the rigid body of the DOX molecule lies on the plane defined by the two DNA backbones, while the intercalation site is still in an opened conformation. This state may be associated to the intermediate one (IM) already found in the case of daunomycin.[87] Roughly, 14 kcal/mol are necessary for DOX to reach the solvent.

Since intercalation, and, subsequently, dissociation of anthracyclines from DNA has been demonstrated to alter the DNA structure,[88] we selected few parameters to measure the structural modifications that take place during the binding process, which could be used in further analysis. Figure 10c shows the average rise distance between base pairs in the intercalation site as a function of the reaction coordinate in the simulated windows. High rise values (of approximately 7.5 Å) are detected for the intercalated state: then, as DOX approaches the bulk solvent, the distance between two consecutive bases decreases, reaching a final value of about 3.5 Å in the unbound state, very close the value of 3.4 Å featured by native B-DNA. The intermediate state (whose structure could be considered as the mean structure of the umbrella window starting at 10 Å of COMs distance) is showed both on a standard desktop and using IVR in Figure 11: it is possible to observe that such conformation seems to be still in an opened conformation (assuming a rise value close to 6 Å), thus being accessible by water molecules.

The hydration of the binary system as a function of the reaction coordinate was taken into account (and plotted in Figure 10d): such property was calculated as the average number of hydrogen bonds between water molecules and either DOX or the four nucleobases that delimit the binding site. The intercalation induces a decrease of the H-bonds since DOX acts as a barrier for water molecules, which cannot enter into the binding site. At the same time, DOX is less hydrated as it approaches the DNA binding task: on average, two hydrogen bonds that take place in the DOX unbound state are not preserved in the intercalated one. It is interesting to observe a peak of average number of hydrogen bonds in proximity of the intermediate state: in fact, at this point, DOX has already left the intercalation site, so as to be considered solvent-exposed whereas the binding site is still opened. Finally, after 10 Å of COM distance, the number of water molecules H-bonded to the binding site bases slowly decreases. This is in agreement with our previous considerations: in fact, after this point, as shown in Figure 10c, the binding site reduces its size, because of the departure of DOX, thus decreasing water accessibility to the intercalation site nucleobases.

Suitably chosen conformations (i.e., the centroids of the single umbrella windows) were used to reconstruct the whole unbinding process, from the intercalated to the completely unbound state, through the IM one, so as to build an artificial trajectory to be used in Caffeine to follow in the meanwhile both chemical structure evolution and related structural parameters. Moreover, considering the PMF chart in Figure 10b, it is always possible to connect the current, visualized snapshot to its associated free energy value just switching from the COMs distance chart to the PMF one, thus increasing the user's understanding of the overall free energy study. It is important to highlight here that a quantitative evaluation of binding/unbinding of DOX was not the ambition of the present study: here we applied a simplified (to limit the computational cost), yet consistent, computational protocol for illustrating the features of Caffeine to a wide audience. It is anyway remarkable that the obtained results are in line with currently available literature data.

4 | CONCLUSION AND OUTLOOKS

In this contribution, the technical details and main features of Caffeine, a novel molecular viewer suited for IVR environments, are presented. Caffeine allows a smooth transition from desktop computers, where the most diffused molecular graphics software work, to IVR systems, such as helmets and the CAVE theater. In our opinion, it is reasonable to expect benefits from systematic use of IVR environments within computational investigations. In fact, molecules can be perceived as three-dimensional objects with a well-established position of atoms in space, thus being characterized in a more precise and effective manner. Moreover, some peculiar features of Caffeine, such as the augmented reality-like visualization of 2D charted data and the interactive filtering of trajectories with "key-frames," envision our idea of possible, productive and realistic employment of IVR in computational chemistry, which could be seen as reliable front-end tool in post-processing analysis.

Caffeine is under active development and there are a number of new features we would like to add in the future releases. It is worth noticing that, although the current version of Caffeine allows to visualize orbitals, spectra, and time evolution of molecular properties, which can be useful in the interpretation of some spectroscopic observable, further improvements are needed for obtaining a full user-friendly virtual spectrometer.

Although, it is sufficiently fluid, the visualization of multiple structures (trajectories) would benefit from further optimization. Such an optimization can be implemented, for example, using some GPU-accelerated methods, like those proposed by Krone et al.[49] and by Wahle and Birmanns[89] for ribbon generation.

Another line of development concerns the interaction with visualized data. Right now, within the CAVE the user is able to interact with the system by means of a simple application for tablet computers. Furthermore, at present, many features of Caffeine cannot be controlled via tablet, and require the help of a second user acting on the external control panel. We want to better exploit our tracking system by enabling hands tracking within the CAVE, thereby allowing the user to manipulate the visualized system, that is, to move, scale, rotate the system and playing back or forward across frames. As for IVR helmets, we plan to officially support this kind of devices in the near future.

From a general perspective, we envision Caffeine as an advanced graphics front-end focused on visualization and interactive data handling, able to communicate with other analysis environments using flexible interchange formats. As an example, we are working on a flexible representation of hydrogen bonds from MD.[90] Other efforts are being directed in more extended visualization of volumetric and spectroscopic quantities, acting as an IVR front-end for other analysis environments.[91]

ACKNOWLEDGMENTS

The authors thank the DREAMS Lab technical staff for managing the computing facilities at SNS and Dr. Balasubramanian Chandramouli for useful discussion.

REFERENCES

[1] M. Valle, *Int. J. Quantum Chem.* **2013**, *113*, 2040.

[2] N. Luehr, A. G. B. Jin, T. J. Martínez, *J. Chem. Theory Comput.* **2015**, *11*, 4536. PMID: 26574246.

[3] C. Casher, C. Leach, C. S. Page, H. S. Rzepa, *J. Mol. Struct. (Theochem)* **1996**, *368*, 49.

[4] J. D. Hirst, D. R. Glowacki, M. Baaden, *Faraday Discuss.* **2014**, *169*, 9.

[5] J. E. Stone, A. Kohlmeyer, K. L. Vandivort, K. Schulten, In *Adv. Visual Comput.: 6th International Symposium, ISVC 2010* (Eds: G. Bebis, R. Boyle, B. Parvin, D. Koracin, R. Chung, R. Hammound, M. Hussain, T. Kar-Han, R. Crawfis, D. Thalmann, D. Kao, L. Avila), Springer Berlin Heidelberg, Las Vegas, NV November 29–December 1 **2010**, pp. 382–393.

[6] S. Rajeev, Z. Michael, V. I. Pavlovic, T. S. Huang, Z. Lo, S. Chu, Y. Zhao, J. C. Phillips, and K. Schulten, *IEEECGA* **2000**, *20*, 29.

[7] Microsoft, Kinect for Xbox One, http://www.xbox.com/en-US/xbox-one/accessories/kinect-for-xbox-one, Accessed 5 July, 2016.

[8] Leap Motion Inc., Leap Motion, https://www.leapmotion.com, Accessed 5 July, 2016.

[9] Oculus VR LLC, Oculus Rift, https://www.oculus.com/en-us/rift/, Accessed 5 July, 2016.

[10] HTC and Valve, Vive, http://www.htcvive.com, Accessed 5 July, 2016.

[11] Novint Technologies Inc., Falcon 3D Touch controller, http://www.novint.com/index.php/novintfalcon, Accessed 5 July, 2016.

[12] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, J. C. Hart, *Commun. ACM* **1992**, *35*, 64.

[13] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, in *Proc. 20th Annu. Conf. Comput. Graph. Interactive Tech.*, SIGGRAPH '93, ACM, New York, NY **1993**, pp. 135–142.

[14] A. Salvadori, A. Brogni, G. Mancini, V. Barone, in *Augmented Virtual Reality: First Int. Conf., AVR 2014* (Eds: T. L. De Paolis, A. Mongelli), Springer International Publishing, Lecce, Italy September 17–20, **2014**, Revised Selected Papers, pp. 333–350.

[15] Terrence, D. Michael, P. Rick, S. Michael, P. Valerie, T. Virtual, *Reality Visualization of Parallel Molecular Dynamics Simulation*, Society for Computer Simulation, **1995**, pp. 483–487.

[16] A. van Dam, A. S. Forsberg, D. H. Laidlaw, J. J. LaViola, R. M. Simpson, *IEEE Comput. Graph. Appl.* **2000**, *20*, 26.

[17] E. Moritz, J. Meyer, in *Proc. Fourth IEEE Symp. Bioinformatics Bioeng.*, **2004**., pp. 503–507.

[18] K. Reda, A. Knoll, K. I. Nomura, M. E. Papka, A. E. Johnson, J. Leigh, in IEEE Symp. Large-Scale Data Anal. Visualization (LDAV), **2013**, pp. 59–65.

[19] W. Humphrey, A. Dalke, K. Schulten, *J. Mol. Graph.* **1996**, *14*, 33.

[20] M. Czernuszenko, D. Pape, D. Sandin, T. DeFanti, G. L. Dawe, M. D. Brown, *SIGGRAPH Comput. Graph.* **1997**, *31*, 46.

[21] W. R. Sherman, *Daniel Coming, and Simon Su*. Freevr: Honoring the Past, Looking to the Future. Proc. SPIE, 8649:864906–864906–15, **2013**.

[22] Mechdyne Corporation, Cavelib: The ultimate solution for 3d virtual reality displays, http://www.mechdyne.com/software.aspx?name=CAVELib, Accessed 5 July, 2016.

[23] J. E. Stone, W. R. Sherman, K. Schulten, in *IEEE Int. Parallel Distributed Processing Symposium Workshop (IPDPSW)*, in Press. http://www.ks.uiuc.edu/Publications/Papers/paper.cgi?tbcode=STON2016A

[24] FEI, Amira 3D Software for Life Sciences, http://www.fei.com/software/amira-3d-for-life-sciences/, Accessed 5 July, 2016.

[25] YASARA Biosciences, YASARA—Yet Another Scientific Artificial Reality Application, http://www.yasara.org/, Accessed 5 July, 2016.

[26] Schrödinger, LLC, The PyMOL Molecular Graphics System, Version 1.8, http://www.pymol.org, Accessed 5 July, 2016.

[27] Virtalis Inc., VR For PyMOL, http://www.virtalis.com/vr-for-pymol/, Accessed 5 July, 2016.

[28] The Qt Company, Qt framework, http://www.qt.io, Accessed 5 July, 2016.

[29] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, G. R. Hutchison, *J. Cheminform.* **2011**, *3*, 1.

[30] OpenSceneGraph, http://www.openscenegraph.org, Accessed 5 July, 2016.

[31] G-Truc Creation, OpenGL Mathematics, http://glm.g-truc.net, Accessed 5 July, 2016.

[32] U. Rathmann, J. Wilgen, Qwt - Qt Widgets for Technical Applications, http://qwt.sourceforge.net, Accessed 5 July, 2016.

[33] D. Frishman, P. Argos, *Proteins* **1995**, *23*, 566.

[34] Worldwide Protein Data Bank, PDB file format, http://www.wwpdb.org/documentation/file-format, Accessed 5 July, 2016.

[35] OpenBabel, XYZ file format, http://openbabel.sourceforge.net/wiki/XYZ, Accessed 5 July, 2016.

[36] Gaussian Inc., The cubegen utility, http://www.gaussian.com/g_tech/g_ur/u_cubegen.htm, Accessed 5 July, 2016.

[37] NaturalPoint Inc., OptiTrack - Motion Capture Systems, https://www.optitrack.com/, Accessed 5 July, 2016.

[38] R. Kooima, Generalized perspective projection, http://csc.lsu.edu/~kooima/articles/genperspective/index.html, 2008, Accessed 5 July, 2016.

[39] NVIDIA Corporation, Quadro SLI Technology, http://www.nvidia.com/object/quadro-sli-technology.html, Accessed 5 July, 2016.

[40] S. Gumhold, in *Proc. Vision, Modeling, Visualization Conf. 2003 (VMV 2003)* (Ed: T. Ertl), Aka GmbH, München, Germany November 19-21, 2003, pp. 245–252.

[41] R. Toledo, B. Lévy, Extending the graphic pipeline with new gpu-accelerated primitives. Technical report, INRIA-ALICE, **2004**.

[42] G. Reina, T. Ertl, in *Proc. Seventh Joint Eurographics/IEEE VGTC Conf. Visualization*, EUROVIS'05, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland **2005**, pp. 177–182.

[43] S. Christian, W. Tim, B. Mario, G. Markus, G, in *Proc. 3rd Eurographics/IEEE VGTC Conf. Point-Based Graphics*, SPBG'06, pages 59–65, Aire-la-Ville, Switzerland, Switzerland, **2006**. Eurographics Association.

[44] M. Tarini, P. Cignoni, C. Montani, *IEEE Transactions on Visualization and Computer Graphics* **2006**, *12*, 1237.

[45] M. Chavent, A. Vanel, A. Tek, B. Levy, S. Robert, B. Raffin, M. Baaden, *Journal of Computational Chemistry* **2011**, *32*, 2924.

[46] P. D. Bagur, N. Shivashankar, V. Natarajan, Improved quadric surface impostors for large bio-molecular visualization. In *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing*, ICVGIP '12, pages 33:1–33:8, New York, NY, USA, **2012**. ACM.

[47] K. Barbora, K. Michael, L. Norbert, F. Martin, B. Marc, B. Daniel, V. Ivan, P. Julius, H. Hans-Christian, in *Eurographics Conference on Visualization (EuroVis) – STARs* (Eds: R. Borgo, F. Ganovelli, I. Viola), The Eurographics Association, **2015**.

[48] J. F. Blinn, in *Proc. 4th Annu. Conf. Comput. Graph. Interactive Tech.*, SIGGRAPH '77, ACM, New York, NY **1977**, pp. 192–198.

[49] K. Michael, B. Katrin, E. Thomas, in *Theory and Practice of Computer Graphics* (Eds: I. S. Lim, W. Tang), The Eurographics Association, **2008**.

[50] M. Carson, C. E. Bugg, *J. Mol. Graph.* **1986**, *4*, 121.

[51] A. Kaufman, K. Mueller. in *Visualization Handbook* (Eds: C. D. HansenChris, R. Johnson), Elsevier Butterworth-Heinemann, **2005**, pp. 127–174.

[52] W. E. Lorensen, H. E. Cline, in *Proc. 14th Annu. Conf. Comput. Graph. Interactive Tech.*, SIGGRAPH '87, ACM, New York, NY **1987**, pp. 163–169.

[53] S. F. F. Gibson, in *Proc. First Int. Conf. Med. Image Comput. Comput.-Assisted Intervention*, MICCAI '98, Springer-Verlag, London, UK **1998**, pp. 888–898.

[54] P. Bourke, Polygonising a scalar field (marching cubes), http://paulbourke.net/geometry/polygonise/, Accessed 5 July, 2016.

[55] L. Mikola, Smooth voxel terrain (part 2), http://0fps.net/2012/07/12/smooth-voxel-terrain-part-2/, Accessed 5 July, 2016.

[56] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, Ö. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, D. J. Fox, *Gaussian* $\sim$*09 Revision D.01*, Gaussian Inc., Wallingford, CT **2009**.

[57] J. H. Han, *3D Graphics for Game Programming*, Chapman and Hall/CRC Press, **2011**.

[58] C. D. Hansen, C. R. Johnson, in *The Visualization Handbook*, Elsevier Butterworth-Heinemann, Burlington, MA, **2005**.

[59] W. Rephael, *Isosurfaces: Geometry, Topology, and Algorithms*, CRC Press, **2013**.

[60] T. Porter, T. Duff, in *Proc. 11th Annu. Conf. Comput. Graph. Interactive Tech.*, SIGGRAPH '84, ACM, New York, NY **1984**, pp. 253–259.

[61] E. Cass, *Technical Report, NVIDIA Corporation* **2001**, *2*, 7.

[62] L. Bavoil, K. Myers, Order independent transparency with dual depth peeling. *Technical Report., NVIDIA Corporation*, pp. 1–12.

[63] B. Liu, L. Y. Wei, Y. Q. Xu, E. Wu, presented at *11th IEEE Int. Conf. Comput. Aided Des. Comput. Graph., CAD/Graphics '09*, 2009, pp. 452–456.

[64] M. McGuire, L. Bavoil, *J. Comput. Graph. Tech.* **2013**, *2*, 122.

[65] M. Houman, Sort-independent alpha blending. *Perpetual Entertainment, GDC Talk*, **2007**.

[66] M. Maule, J. L. D. Comba, R. P. Torchelsen, R. Bastos, *Comput. Graph.* **2011**, *35*, 1023.

[67] M. Morgan, Casual effects: Implementing weighted, blended order-independent transparency, http://casual-effects.blogspot.it/2015/03/implemented-weighted-blended-order.html, Accessed 5 July, 2016.

[68] C. Adamo, V. Barone, *J. Chem. Phys.* **1999**, *110*, 6158.

[69] Ugo Varetto, Molekel 5.4, http://ugovaretto.github.io/molekel/, Accessed 5 July, 2016.

[70] M. D. Hanwell, D. E. Curtis, D. C. Lonie, T. Vandermeersch, E. Zurek, G. R. Hutchison, *J. Cheminform.* **2012**, *4*, 1.

[71] KDE, Oxygen project, https://techbase.kde.org/Projects/Oxygen, Accessed 5 July, 2016.

[72] J. L. Taylor, in *Encyclopedia of Neuroscience* (Ed: Larry R. Squire), Academic Press, **2009**, pp. 1143–1149.

[73] JTKSOFT, JoyToKey, http://joytokey.net/en/, Accessed 5 July, 2016.

[74] Vmd user's guide, version 1.9.2, http://www.ks.uiuc.edu/Research/vmd/current/ug/, Accessed 5 July, 2016.

[75] Z. Xu, A. L. Horwich, P. B. Sigler, *Nature* **1997**, *388*, 741.

[76] C. Sabin, P. Plevka, *Acta Crystallogr. Sect. F* **2016**, *72*, 188.

[77] M. D'Amore, R. Improta, V. Barone, *J. Phys. Chem. A* **2003**, *107*, 6264.

[78] G. Mancini, C. Zazza, *PLoS One* **2015**, *10*, 1.

[79] C. Toniolo, M. Crisma, F. Formaggio, C. Peggion, *Biopolymers (Pept. Sci.)* **2001**, *60*, 396.

[80] R. Improta, V. Barone, *Chem. Rev.* **2004**, *104*, 1231.

[81] S. Carlotto, P. Cimino, M. Zerbetto, L. Franco, C. Corvaja, M. Crisma, F. Formaggio, C. Toniolo, A. Polimeno, V. Barone, *J. Am. Chem. Soc.* **2007**, *129*, 11248.

[82] V. Cojocaru, P. J. Winn, R. C. Wade, *Biochim. Biophys. Acta* **2007**, *1770*, 390.

[83] D. Usharani, C. Zazza, W. Lai, M. Chourasia, L. Waskell, S. Shaik, *J. Am. Chem. Soc.* **2012**, *134*, 4053. PMID: 22356576.

[84] R. A. Laskowski, *J. Mol. Graph.* **1995**, *13*, 323.

[85] W. A. Denny, B. C. Baguley, *Curr. Top. Med. Chem.* **2003**, *3*, 339.

[86] G. M. Torrie, J. P. Valleau, *J. Comput. Phys.* **1977**, *23*, 187.

[87] M. Wilhelm, A. Mukherjee, B. Bouvier, K. Zakrzewska, J. T. Hynes, R. Lavery, *J. Am. Chem. Soc.* **2012**, *134*, 8588. PMID: 22548344.

[88] F. Yang, S. S. Teves, C. J. Kemp, S. Henikoff, *Biochim. Biophys. Acta* **2014**, *1845*, 84.

[89] M. Wahle, S. Birmanns, *Proc. SPIE* **2011**, *7868*, 786805.

[90] M. Pagliai, G. Cardini, R. Righini, V. Schettino, *J. Chem. Phys.* **2003**, *119*, 6655.

[91] D. Licari, A. Baiardi, M. Biczysko, F. Egidi, C. Latouche, V. Barone, *J. Comput. Chem.* **2015**, *36*, 321.

AUTHORS' BIOGRAPHIES

Andrea Salvadori is a PhD student in Chemistry at Scuola Normale Superiore (Pisa, Italy) in the SMART@SNS Laboratory headed by Prof. Vincenzo Barone. His primary research interests are related to the application of computer graphics and virtual reality technologies to the field of Molecular Graphics. He received his Master's Degree in Computer Science ("Laurea Specialistica in Tecnologie Informatiche") from University of Pisa (Pisa, Italy).

Gianluca Del Frate is currently a PhD student at Scuola Normale Superiore, Pisa, in the SMART@SNS Laboratory headed by Prof. Vincenzo Barone. He received his Master Degree in Medicinal Chemistry in 2013 from the University of Pisa. His research interests are related to the computational study of biochemical systems and to the development and validation of accurate force fields for molecular simulations.

Marco Pagliai obtained his PhD in Chemistry (2004) at the University of Florence (Italy), where he did postdoctoral research until 2015. He then joined the group of Prof. Vincenzo Barone at Scuola Normale Superiore in Pisa as a research fellow. His research interests are the application of classical and ab initio molecular dynamics simulations to characterize structural, dynamic and spectroscopic properties of complex systems in condensed phases.

Giordano Mancini obtained his PhD in Physical Chemistry in 2008 from the University of Rome La Sapienza. Since 2013, he has been a post doc researcher at Scuola Normale Superiore in Pisa. He works on the development of accurate force fields for classical molecular dynamics and on the application of IVR technologies to molecular modeling.

Vincenzo Barone is Full Professor in Physical Chemistry since 1994 and has been appointed as Director of the Scuola Normale Superiore in 2016. He is author of more than 700 papers with more than 40,000 citations, an h-factor of 78, and 10 papers with more than 1000 citations each. He has contributed to the development of Density Functional Theory, solvation theory, computational spectroscopy and performed state-of-the-art applications in several fields including cultural heritage and astrochemistry.

SUPPORTING INFORMATION

Additional Supporting Information can be found in the online version of this article at the publisher's website.

How to cite this article: A. Salvadori, G. Del Frate, M. Pagliai, G. Mancini, V. Barone. Int. J. Quantum Chem. 2016, *116*: 1731–1746. DOI: 10.1002/qua.25207

APPENDIX A: NOTES ON THE IMPLEMENTATION OF GPU-BASED RAY-CASTING OF QUADRIC SURFACES IN CAFFEINE

GPU-based ray-casting of quadric surfaces is a technique widely used in scientific visualization which allows to visualize a large amount of glyphs at interactive frame rates by exploiting the huge computing power of the modern GPUs. The use of quadric surfaces as glyphs is motivated by the fact that the intersection between a ray and a surface of this type can be computed by solving a simple quadratic equation.

The key idea of this method is to feed the graphics pipeline with a simple proxy geometry (e.g., a bounding box or a point sprite) in place of the desired implicit surface. The proxy geometry must be sized so to completely enclose the surface in window space, and a ray-casting shader must be enabled when drawing the geometry, so to process each fragment resulting from its rasterization. In particular, the fragment shader analytically computes the intersection between the surface and a ray starting from the virtual camera and passing through the center of the fragment. If there is no intersection then the fragment is discarded, otherwise its depth is adjusted with the one of the intersection point closest to the camera. Usually, the shader also computes the normal vector of the surface at the intersection point, that will be used to calculate the color of the fragment according to a certain shading model (e.g., the Blinn–Phong[48] model). Although it is possible to write a single shader capable of drawing any quadric surface,[41,43,45] we chose to implement separate dedicated shaders for spheres and capsules (cylinders with (or without) semispheres at their ends). Furthermore, we wrote a geometry shader to generate the proxy geometry on the fly, similarly to what done by Bagur et al.[46] By doing so, we are able to represent a sphere with only seven floating point numbers (three for the center, three for the color, and one for its radius), and a capsule with ten floats (two vertex, color, and radius). The geometry shader will then generate a quad for each sphere and up to two quads for each cylinder as proxy geometries.

APPENDIX B: NOTES ON THE IMPLEMENTATION OF THE MARCHING CUBES AND THE SURFACE NETS ALGORITHMS IN CAFFEINE

Caffeine implements, and provides to the user, two different algorithms for the construction of a triangle mesh approximating an isosurface of a volumetric dataset: the traditional *Marching Cubes*[52] and a simplified version of the *Surface Nets*.[53] With regard to the Marching Cubes we use (a slightly adapted version of) the popular implementation by Cory Gene Bloyd and Paul Bourke,[54] while for the Surface Nets we re-implemented in C++ the so-called "*Naive*" version by Mikola Lysenko [55] (originally coded in JavaScript).

In brief, the Marching Cubes algorithm iterates over the cells bordered by the volume grid (called "cubes" even if they can be non-cubical parallelepipeds) and, for each cell, it "marks" the voxels whose value is lower than the *isovalue*. If two voxels connected by an edge have a different marking (because one value is lower than the isovalue, while the other is greater than or equal to it), then the isosurface crosses the edge. In that case, and in function of which edges are crossed, one or more triangles are generated by applying a predefined triangulation scheme. The vertices of these triangles always lies on the intersected edges and their coordinates are computed by linearly interpolating the related voxels in function of their value and of the isovalue. At the end, the algorithm returns the set of the triangles resulting from processing the entire grid, which constitutes a good approximation of the isosurface. The Surface Nets algorithm operates similarly to the Marching Cubes, by iterating over the cells and checking which ones are crossed by the isosurface. The differences between the two approaches resides in the fact that Surface Nets generates only one vertex for each crossed cell, that this vertex lies within the cell (instead to be constrained on an edge) and that the resulting sets of polygons (quads, which can anyway be splitted in triangles) are obtained by connecting each vertex with the vertices of the neighbors cells (sharing a face) crossed by the same isosurface. As regard to the position of the vertex within the cell, the original algorithm[53] initially places it at the center of the cell. Then, an iterative process is applied, which moves the vertices so as to minimize the sum of the squared lengths of the links connecting the vertices, with the constrain to keep each vertex within its original cell. The simplified "*Naive*" version, instead, choses as vertex of the cell the centroid of the approximated intersection points between the isosurface and the edges of the cells (To the best of our knowledge, Mikola Lysenko[55] was the first to propose to place the vertex at the centroid of the intersection points) (computed as in the Marching Cubes algorithm). By avoiding the iterative minimization process, the "*Naive*" Surface Nets method is both faster and easier to implement, although it could produce suboptimal results.

The Surface Nets algorithm produces a more regular triangulation than the traditional Marching Cubes method, as shown in Figure 3. Furthermore, according to some tests performed on our implementations, Naive Surface Nets results slightly faster. Let's consider, as an example, the isosurface representing the molecular orbital of a caffeine molecule with isovalue 0.02, drawn in orange in Figure 4. The traditional Marching Cubes algorithm takes 66 ms to create the corresponding triangle mesh composed by 13,896 triangles and 41,688 vertices, while the Naive Surface Nets method produces a similar isosurface composed by 13,928 triangles and 6980 vertices in 52 ms (These tests has been performed on the same desktop computer described in section "Performance Evaluation"). Note that, while the number of triangles produced by the two algorithms is almost the same, the traditional Marching Cubes method outputs much more vertices than its Naive Surface Nets counterpart. This is due to the fact that the traditional Marching Cubes algorithm does not take into account that the vertices it generates in each cell are shared among multiple triangles, hence the same vertex is repeated multiple times in the output list. In the case of the Surface Nets method, instead, it is easier to obtain an *indexed triangle list*, that is, the output vertex list does only contains non-duplicated vertices and the triangles are defined by an additional index list storing three indices for the vertex list for each triangle to be generated. *Indexed triangle list* allows to reduce the memory required to store a given geometry, and also brings an increase of the rendering performance thanks to a better exploitation of the GPU cache.[57] Regarding to generation times, the difference is due to the fact that the Marching Cubes algorithm computes per-vertex attributes (and in particular the normal vector) for each intersection point between the edges and the isosurface (consequently, several times for the same geometrical vertex within the cell), while in Surface Nets method these attributes are computed one time only.

APPENDIX C: SIMULATING SEMITRANSPARENT SURFACES IN REAL-TIME 3D COMPUTER GRAPHICS

In interactive computer graphics, semitransparent objects are usually simulated using a technique known as "*alpha blending*," first introduced by Porter and Duff[60] in 1984 and nowadays supported natively by the graphics hardware. In this technique, each *fragment* (Pixel prototype resulting from the rasterization of a geometrical primitive [e.g., a triangle]) has an associated *alpha* value representing its opacity. To "blend" a new fragment in the image under construction, a new color is computed as a function of the color of the fragment and of the one already present in the considered location of the frame buffer (Portion of the memory of the graphics card storing the image that will be displayed on screen) (resulting from the processing of the previously submitted fragments). Hence, the new color will replace the old one in the

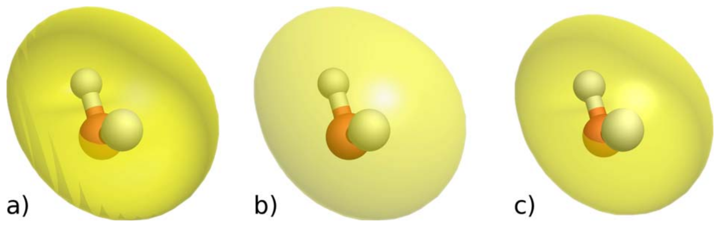


FIGURE 12 Comparison between different rendering techniques for simulating semitransparent surfaces. An orbital of a water molecule is shown. (a) Alpha blending of front and back faces produce graphical artifacts. (b) Alpha blending of front faces only removes (most of) the graphical artifacts but produce inaccurate images (e.g., the oxygen atom does not appear crossed by the orbital). (c) Image obtained using the "*Weighted Blended Order-Independent Transparency*" technique. Even if this is an approximate method, it produces plausible results. The orbitals have been computed at HF/STO-3G level of theory with Gaussian 09[56] suite of programs

frame buffer. The function usually chosen for this task is the following (named "*OVER*" operator in reference 60):

$$\mathrm{RGB}_{\mathrm{new}} = (\mathrm{RGB}_{\mathrm{frag}} \cdot A_{\mathrm{frag}}) + ((1 - A_{\mathrm{frag}}) \cdot \mathrm{RGB}_{\mathrm{old}})$$

Where $\mathrm{RGB}_{\mathrm{new}}$ is the color resulting from the blending operation, $\mathrm{RGB}_{\mathrm{old}}$ is the color previously stored in the frame buffer at the considered location, $\mathrm{RGB}_{\mathrm{frag}}$ is the color of the new fragment and $A_{\mathrm{frag}} \in [0, 1]$ represents the opacity of the fragment. The problem with the *OVER* operator is that the final color for a given pixel depends on the order in which the fragments laying on that pixel are blended. In other words, the rendered image can vary depending on the position of the virtual camera in the 3D scene, on the coordinates of the geometrical primitives in the 3D scene, and on the order in which these primitives are submitted to the rendering system. From the user's point of view this may result in incorrect colors and/or graphical artifacts, as shown in Figure 12a.

One solution to this problem consists in drawing the opaque polygons first (in any order), and then drawing the semitransparent polygons from the farthest to the nearest to the virtual camera (with the writing on the Z-buffer disabled). However, depending on the number of semitransparent polygons, ordering them every frame may not be a viable solution for a real-time interactive application. Furthermore, the problem persists in the case of overlapping polygons. A popular cheaper alternative consists in ordering only the semitransparent "*objects*" of the 3D scene (instead of the triangles they are formed by) and to draw only their polygonal faces that are oriented in the direction of the camera ("*backface culling*"). However, this method produces roughly approximate results (see Figure 12b), works only in the case of non-overlapped semitransparent objects and may not remove entirely the graphical artifacts. In conclusion, the only way to obtain an *exact* alpha blending, using the "*OVER*" operator and in the general case, is to blend the fragments in the correct order. Examples of GPU-accelerated techniques which achieve this goal are the "depth peeling" methods.[61–63] Other GPU-accelerated approaches renounce to produce exact results to obtain better performance, for example, by defining a different compositing operator which is commutative.[62,64,65] Several other OIT methods are present in the literature and a comprehensive survey has been published.[66]

In Caffeine, we employed an approximated method, the "*Weighted Blended Order-Independent Transparency*" by McGuire and Bavoil[64,67] (see Figure 12c), because it provides a good balance between quality of the results, performance, and implementation complexity.

Future perspectives

The development of *Caffeine* will not end with this thesis and there are a number of new features that I, my colleagues and my supervisors would like to implement for future releases.

One of the main priorities is the support to the latest versions of HMDs, and in particular to the *Oculus Rift* and the *HTC Vive*. At the time of writing, that support is being developed by a colleague. Another aspect related to VR that need improvements is user interaction. We would like to replace the use of the tablet with hand-held controllers ("wands") or by directly tracking the user's fingers, and also provide a larger set of commands accessible from within the virtual environment. This could be obtained by means of a dedicated graphical user interface immersed in the 3D scene (similarly to what done with line charts), and would have the advantage of being a unified user interface for both CAVE and HMD systems.

The visualization of dynamic systems (trajectories) would benefit from further optimization. These could be obtained by parallelizing the procedures for the computation of the graphics primitives of each "diagram", as well as by adopting further GPU-accelerated methods, like those discussed in section 2.2. Another missing useful feature is the ability to save to file (and subsequently reload) the visualization state.

As regard to the visualization of molecular data, the main missing feature is the computation of molecular surfaces. Again, we plan to take advantage of the latest GPU-accelerated methods proposed in literature and discussed in section 2.2.3. It would also be interesting to implement Direct Volume Rendering of volumetric data, in particular for the visualization of quantum chemical data (such as electron densities and atomic/molecular orbitals) by exploiting some of the ideas proposed in [168]. The support for plotting supplementary numerical data in charts should be extended, allowing to provide further types of dataset and to represent them with appropriate charts.

Acknowledgments

First of all, I wish to thank Prof. Vincenzo Barone for his guidance and for giving me the opportunity to work on this exiting project in one of the most prestigious universities in Europe.

I am particularly grateful to Dr. Giordano Mancini for the invaluable teachings, guidance, assistance and support he provided me with during my PhD.

I would also like to thank my co-authors, and in particular Dr. Daniele Licari, Dr. Andrea Brogni, Dr. Marco Pagliai and Gianluca Del Frate for their help and feedbacks, as well as for designing and conducting the case studies that has been used to test, improve and illustrate the features of *Caffeine*.

Finally, I would like to extend my thanks to my colleagues and all the technical, logistic and administrative staff of Scuola Normale Superiore, for making it a pleasing place to work.

This thesis is dedicated to my family, for their invaluable help, support and encouragement along all these years. Without them, this thesis would never have been written.

Bibliography

[1] J. D. Hirst, D. R. Glowacki, and M. Baaden, "Molecular simulations and visualization: introduction and overview," *Faraday Discuss.*, vol. 169, pp. 9–22, oct 2014.

[2] F. P. Brooks, M. Ouh-Young, J. J. Batter, P. Jerome Kilpatrick, and P. J. Kilpatrick, "Project GROPE Haptic displays for scientific visualization," *Proceedings of the 17th annual conference on Computer graphics and interactive techniques - SIGGRAPH '90*, vol. 24, no. 4, pp. 177–185, 1990.

[3] A. van Dam, A. S. Forsberg, D. H. Laidlaw, J. J. LaViola, and R. M. Simpson, "Immersive VR for scientific visualization: a progress report," *IEEE Computer Graphics and Applications*, vol. 20, no. 6, pp. 26–52, 2000.

[4] NormaleNews, "Marcos Valdes con il progetto VIS ottiene il "Premio per la Comunicazione Scientifica" della Società Italiana di Fisica." `https://goo.gl/DNkO7z`, 2015.

[5] A. Salvadori, D. Licari, G. Mancini, A. Brogni, N. De Mitri, and V. Barone, "Graphical Interfaces and Virtual Reality for Molecular Sciences," in *Reference Module in Chemistry, Molecular Sciences and Chemical Engineering*, Elsevier, 2014.

[6] A. Salvadori, A. Brogni, G. Mancini, and V. Barone, "Moka: Designing a Simple Scene Graph Library for Cluster-Based Virtual Reality Systems," in *Augmented and Virtual Reality: First International Conference, AVR 2014, Lecce, Italy, September 17-20, 2014, Revised Selected Papers* (L. T. De Paolis and A. Mongelli, eds.), vol. 8853 of *Lecture Notes in Computer Science*, pp. 333–350, Springer International Publishing, 2014.

[7] A. Salvadori, G. Del Frate, M. Pagliai, G. Mancini, and V. Barone, "Immersive virtual reality in computational chemistry: Applications to the analysis of QM and MM data," *International Journal of Quantum Chemistry*, vol. 116, pp. 1731–1746, nov 2016.

[8] C. Casanova and M. Ptito, “Preface,” in *Vision: From Neurons to Cognition* (C. Casanova and M. Ptito, eds.), vol. 134 of *Progress in Brain Research*, pp. ix – xi, Elsevier, 2001.

[9] M. Aubert, A. Brumm, M. Ramli, T. Sutikna, E. W. Saptomo, B. Hakim, M. J. Morwood, G. D. van den Bergh, L. Kinsley, and A. Dosseto, “Pleistocene cave art from Sulawesi, Indonesia,” *Nature*, vol. 514, pp. 223–227, oct 2014.

[10] C. Woods, “The Earliest Mesopotamian Writing,” in *Visible Language: Inventions of Writing in the Ancient Middle East and Beyond (Oriental Institute Museum Publications 32)* (C. Woods, E. Teeter, and G. Emberling, eds.), Oriental Institute of the University of Chicago, 2015.

[11] R. W. Hamming, *Numerical Methods for Scientists and Engineers.* Dover Publications, 2nd revise ed., 1987.

[12] T. Rhyne, M. Tory, T. Munzner, M. Ward, C. Johnson, and D. Laidlaw, “Information and scientific visualization: separate but equal or happy together at last,” *IEEE Visualization, 2003. VIS 2003.*, pp. 611–614, 2003.

[13] P. Dragicevic and Y. Jansen, “List of Physical Visualizations.” `http://dataphys.org/list/`.

[14] B. Shneiderman, “The eyes have it: a task by data type taxonomy for information visualizations,” *Proceedings 1996 IEEE Symposium on Visual Languages*, pp. 336–343, 1996.

[15] T. Munzner, *Visualization Analysis and Design.* A K Peters/CRC Press, 12 2014.

[16] C. Upson, T. Faulhaber Jr., D. Kamins, D. H. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam, “The Application Visualization System: A Computational Environment for Scientific Visualization,” *IEEE Comput. Graph. Appl.*, vol. 9, no. 4, pp. 30–42, 1989.

[17] R. B. Haber and D. A. McNabb, “Visualization idioms: A conceptual model for scientific visualization systems,” in *Visualization in Scientific Computing*, pp. 74–93, IEEE Computer Society Press, 1990.

[18] A. C. Telea, *Data Visualization: Principles and Practice.* A K Peters/CRC Press, 2 ed., 9 2014.

[19] Worldwide Protein Data Bank, “PDB file format.” `http://www.wwpdb.org/documentation/file-format`.

[20] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, and G. R. Hutchison, "Open Babel: An Open chemical toolbox," *Journal of Cheminformatics*, vol. 3, no. 10, pp. 1–14, 2011.

[21] T. B. Sousa, "Dataflow Programming: Concept, Languages and Applications," in *Doctoral Symposium on Informatics Engineering 2012 (DSIE'12)* (E. Oliveira, G. David, and A. A. Sousa, eds.), pp. 323–334, 2012.

[22] Advanced Visual Systems Inc., "AVS/Express." `http://www.avs.com/solutions/express/`.

[23] D. Foulser, "IRIS Explorer: A Framework for Investigation," *SIGGRAPH Comput. Graph.*, vol. 29, no. 2, pp. 13–16, 1995.

[24] W. Schroeder, K. Martin, and B. Lorensen, *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition.* Kitware, 4th ed., 2006.

[25] N. Max, "Optical models for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99–108, 1995.

[26] OpenGL.org, "Rendering Pipeline Overview." `https://www.opengl.org/wiki/Rendering_Pipeline_Overview`.

[27] E. Angel and D. Shreiner, *Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL.* Pearson, 6 ed., 2011.

[28] E. E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, The University of Utah, 1974.

[29] J. F. Blinn and M. E. Newell, "Texture and Reflection in Computer Generated Images," *Commun. ACM*, vol. 19, no. 10, pp. 542–547, 1976.

[30] K. Akeley, "Reality Engine Graphics," in *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, (New York, NY, USA), pp. 109–116, ACM, 1993.

[31] H. van de Waterbeemd, E. R. Carter, G. Grassy, H. Kubinyi, C. Y. Martin, S. M. Tute, and P. Willett, "Glossary of terms used in computational drug design (IUPAC Recommendations 1997)," *Pure and Applied Chemistry*, vol. 69, no. 5, p. 1137, 1997.

[32] J. A. Perkins, "A history of molecular representation part one: 1800 to the 1960s," *The Journal of Biocommunication*, vol. 31, no. 1, 2005.

[33] J. A. Perkins, "A History of Molecular Representation Part 2: The 1960s - Present," *The Journal of Biocommunication*, vol. 31, no. 2, 2005.

[34] A. S. Couper, "On a new chemical theory," *Philosophical Magazine Series 4*, vol. 16, no. 105, pp. 104–116, 1858.

[35] A. Crum Brown, "On the Theory of Isomeric Compounds," *Transactions of the Royal Society of Edinburgh*, vol. 23, no. 3, pp. 707–719, 1864.

[36] A. Crum Brown, "On the Classification of Chemical Substances, by means of Generic Radicals," *Transactions of the Royal Society of Edinburgh*, vol. 24, no. 2, pp. 331–339, 1866.

[37] H. S. Mason, "History of the Use of Graphic Formulas in Organic Chemistry," *Isis*, vol. 34, no. 4, pp. 346–354, 1943.

[38] E. Fischer, "Synthese des Traubenzuckers," *Berichte der deutschen chemischen Gesellschaft*, vol. 23, no. 1, pp. 799–805, 1890.

[39] J. Brecher, "Graphical representation of stereochemical configuration (IUPAC Recommendations 2006)," *Pure and Applied Chemistry*, vol. 78, pp. 1897–1970, jan 2006.

[40] A. Serafini, *Linus Pauling: A Man and His Science.* Paragon House, 1991.

[41] L. Pauling, R. B. Corey, and H. R. Branson, "The structure of proteins: Two hydrogen-bonded helical configurations of the polypeptide chain," *Proceedings of the National Academy of Sciences*, vol. 37, no. 4, pp. 205–211, 1951.

[42] L. Pauling and R. B. Corey, "The Pleated Sheet, A New Layer Configuration of Polypeptide Chains," *Proceedings of the National Academy of Sciences*, vol. 37, pp. 251–256, may 1951.

[43] R. B. Corey and L. Pauling, "Molecular Models of Amino Acids, Peptides, and Proteins," *Review of Scientific Instruments*, vol. 24, no. 8, 1953.

[44] W. L. Koltun, "Precision space-filling atomic models," *Biopolymers*, vol. 3, no. 6, pp. 665–679, 1965.

[45] J. D. Watson and F. H. C. Crick, "Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid," *Nature*, vol. 171, pp. 737–738, apr 1953.

[46] J. C. KENDREW, G. BODO, H. M. DINTZIS, R. G. PARRISH, H. WYCKOFF, and D. C. PHILLIPS, "A Three-Dimensional Model of the Myoglobin Molecule Obtained by X-Ray Analysis," *Nature*, vol. 181, pp. 662–666, mar 1958.

[47] J. C. Kendrew, “The Three-Dimensional Structure of a Protein Molecule,” *Scientific American*, vol. 205, pp. 96–110, dec 1961.

[48] D. Voet and J. G. Voet, *Biochemistry.* Wiley, 2 ed., 1995.

[49] F. M. Richards, “The matching of physical models to three-dimensional electron-density maps: A simple optical device,” *Journal of Molecular Biology*, vol. 37, pp. 225–230, oct 1968.

[50] D. J. Haas, “Polypeptide α-carbon models: A new concept in protein model construction,” *Biopolymers*, vol. 9, pp. 1547–1552, dec 1970.

[51] B. Rubin and J. S. Richardson, “The simple construction of protein alpha-Carbon models,” *Biopolymers*, vol. 11, pp. 2381–2385, nov 1972.

[52] E. Martz and E. Francoeur, “History of Visualization of Biological Macromolecules.” `http://www.umass.edu/microbio/rasmol/history.htm`.

[53] J. S. Richardson, D. C. Richardson, K. A. Thomas, E. W. Silverton, and D. R. Davies, “Similarity of three-dimensional structure between the immunoglobulin domain and the copper, zinc superoxide dismutase subunit,” *Journal of Molecular Biology*, vol. 102, pp. 221–235, apr 1976.

[54] R. E. Dickerson and I. Geis, *The Structure and Action of Proteins.* Addison-Wesley, 1969.

[55] J. W. Campbell, H. C. Watson, and G. I. Hodgson, “Structure of yeast phosphoglycerate mutase,” *Nature*, vol. 250, pp. 301–303, jul 1974.

[56] A. Holmgren, B. O. Söderberg, H. Eklund, and C. I. Brändén, “Three-dimensional structure of Escherichia coli thioredoxin-S2 to 2.8 A resolution,” *Proceedings of the National Academy of Sciences*, vol. 72, no. 6, pp. 2305–2309, 1975.

[57] J. S. Richardson, “The Anatomy and Taxonomy of Protein Structure,” vol. 34 of *Advances in Protein Chemistry*, pp. 167–339, Academic Press, 1981.

[58] C. Levinthal, “Molecular model-building by computer.,” *Scientific American*, vol. 214, no. 6, pp. 42–52, 1966.

[59] C. Johnson, “OR TEP: A FORTRAN Thermal-Ellipsoid Plot Program for Crystal Structure Illustrations,” *Oak Ridge National Laboratory Report*, vol. 3794, 1965.

[60] B. Lee and F. M. Richards, “The interpretation of protein structures: Estimation of static accessibility,” *Journal of Molecular Biology*, vol. 55, no. 3, 1971.

[61] J. S. Lipscomb, *Three-dimensional Cues for a Molecular Computer Graphics System.* PhD thesis, 1981.

[62] D. Tsernoglou, G. A. Petsko, and A. T. Tu, “Protein sequencing by computer graphics,” *Biochimica et Biophysica Acta (BBA) - Protein Structure*, vol. 491, no. 2, pp. 605–608, 1977.

[63] J. R. Miller, S. S. Abdel-Meguid, M. G. Rossmann, and D. C. Anderson, “A computer graphics system for the building of macromolecular models into electron density maps,” *Journal of Applied Crystallography*, vol. 14, pp. 94–100, apr 1981.

[64] T. A. Jones, “A graphics model building and refinement system for macromolecules,” *Journal of Applied Crystallography*, vol. 11, no. 4, pp. 268–272, 1978.

[65] T. K. Porter, “Spherical shading,” in *Proceedings of the 5th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’78*, (New York, New York, USA), pp. 282–285, ACM Press, 1978.

[66] J. Greer and B. L. Bush, “Macromolecular shape and surface maps by solvent exclusion.,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 75, no. 1, pp. 303–307, 1978.

[67] F. M. Richards, “Areas, Volumes, Packing, and Protein Structure,” *Annual Review of Biophysics and Bioengineering*, vol. 6, no. 1, pp. 151–176, 1977.

[68] M. L. Connolly, “Analytical molecular surface calculation,” *Journal of Applied Crystallography*, vol. 16, no. 5, pp. 548–558, 1983.

[69] M. L. Connolly, “Molecular surface Triangulation,” *Journal of Applied Crystallography*, vol. 18, no. 6, pp. 499–505, 1985.

[70] M. F. Sanner, A. J. Olson, J.-C. Spehner, and M. D. Sanner, “REDUCED SURFACE: an Efficient Way to Compute Molecular Surfaces,” *Biopolymers*, vol. 38, no. 383, pp. 305–320, 1996.

[71] W. Humphrey, A. Dalke, and K. Schulten, “VMD: visual molecular dynamics,” *Journal of molecular graphics*, vol. 14, no. 1, pp. 33–38, 1996.

[72] E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin, “UCSF Chimera-A visualization system for exploratory research and analysis,” *Journal of Computational Chemistry*, vol. 25, no. 13, pp. 1605–1612, 2004.

[73] M. Totrov and R. Abagyan, “The Contour-Buildup Algorithm to Calculate the Analytical Molecular Surface,” *Journal of Structural Biology*, vol. 116, no. 1, pp. 138–143, 1996.

[74] M. Carson and C. E. Bugg, “Algorithm for ribbon models of proteins,” *Journal of Molecular Graphics*, vol. 4, no. 2, pp. 121–122, 1986.

[75] M. Carson, “Ribbon models of macromolecules,” *Journal of Molecular Graphics*, vol. 5, no. 2, pp. 103–106, 1987.

[76] A. M. Lesk and K. D. Hardman, “Computer-generated schematic diagrams of protein structures,” *Science*, vol. 216, no. 4545, pp. 539–540, 1982.

[77] M. Carson, “RIBBONS 2.0,” *Journal of Applied Crystallography*, vol. 24, no. 5, pp. 958–961, 1991.

[78] P. J. Kraulis, “MOLSCRIPT: a program to produce both detailed and schematic plots of protein structures,” *Journal of Applied Crystallography*, vol. 24, no. 5, pp. 946–950, 1991.

[79] E. A. Merritt and M. E. P. Murphy, “Raster3D Version 2.0. A program for photorealistic molecular graphics,” *Acta Crystallographica Section D*, vol. 50, pp. 869–873, nov 1994.

[80] D. C. Richardson and J. S. Richardson, “The kinemage: A tool for scientific communication,” *Protein Science*, vol. 1, no. 1, pp. 3–9, 1992.

[81] R. Sayle and A. Bissell, “RasMol: A Program for Fast Realistic Rendering of Molecular Structures with Shadows,” in *Proceedings of the 10th Eurographics UK ’92 Conference*, University of Edinburgh, Scotland, 1992.

[82] R. A. Sayle and E. Milner-White, “RASMOL: biomolecular graphics for all,” *Trends in Biochemical Sciences*, vol. 20, no. 9, pp. 374–376, 1995.

[83] M. M. Teeter, “Water structure of a hydrophobic protein at atomic resolution: Pentagon rings of water molecules in crystals of crambin,” *Proceedings of the National Academy of Sciences*, vol. 81, no. 19, pp. 6014–6018, 1984.

[84] Schrödinger LLC, “The PyMOL Molecular Graphics System, Version∼1.8.” 2015.

[85] A. Herráez, "Biomolecules in the computer: Jmol to the rescue," *Biochemistry and Molecular Biology Education*, vol. 34, no. 4, pp. 255–261, 2006.

[86] M. D. Hanwell, D. E. Curtis, D. C. Lonie, T. Vandermeersch, E. Zurek, and G. R. Hutchison, "Avogadro: an advanced semantic chemical editor, visualization, and analysis platform," *Journal of Cheminformatics*, vol. 4, no. 1, pp. 1–17, 2012.

[87] YASARA Biosciences GmbH, WHAT IF Foundation / CMBI, and Spronk NMR Consultency, "YASARA - Yet Another Scientific Artificial Reality Application." `www.yasara.org`.

[88] S. McNicholas, E. Potterton, K. S. Wilson, and M. E. M. Noble, "Presenting your structures: the CCP4mg molecular-graphics software," *Acta Crystallographica Section D*, vol. 67, pp. 386–394, apr 2011.

[89] The Qt Company, "Qt framework." `https://www.qt.io`.

[90] NANO-D - INRIA, "SAMSON - Software for Adaptive Modeling and Simulation of Nanosystems." `https://www.samson-connect.net`.

[91] G. Schaftenaar and J. H. Noordik, "Molden: a pre- and post-processing program for molecular and electronic structures*," *Journal of Computer-Aided Molecular Design*, vol. 14, no. 2, pp. 123–134, 2000.

[92] Gaussian Inc., "GaussView 6." `http://gaussian.com/gv6new/`.

[93] U. Varetto, "Molekel 5.4." http://ugovaretto.github.io/molekel/.

[94] A. Kokalj, "XCrySDen - a new program for displaying crystalline structures and electron densities," *Journal of Molecular Graphics and Modelling*, vol. 17, no. 3-4, pp. 176–179, 1999.

[95] A. L. Spek, "Structure validation in chemical crystallography," *Acta Crystallographica Section D*, vol. 65, pp. 148–155, feb 2009.

[96] S. Vijay-Kumar, C. E. Bugg, and W. J. Cook, "Structure of ubiquitin refined at 1.8 Å resolution," *Journal of Molecular Biology*, vol. 194, no. 3, pp. 531–544, 1987.

[97] X. Hao, A. Varshney, and S. Sukharev, "Real-time visualization of large time-varying molecules," *Proceedings of the High-Performance Computing Symposium '04*, pp. 1–6, 2004.

[98] A. Sharma, R. K. Kalia, A. Nakano, and P. Vashishta, "Scalable and portable visualization of large atomistic datasets," *Computer Physics Communications*, vol. 163, pp. 53–64, oct 2004.

[99] S. Gumhold, "Splatting Illuminated Ellipsoids with Depth Correction," in *Proceedings of the Vision, Modeling, and Visualization Conference 2003 (VMV 2003), München, Germany, November 19-21, 2003*, pp. 245–252, 2003.

[100] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane, "TexMol: Interactive Visual Exploration of Large Flexible Multi-Component Molecular Complexes," in *Proceedings of the Conference on Visualization '04*, VIS '04, (Washington, DC, USA), pp. 243–250, IEEE Computer Society, 2004.

[101] R. Toledo and B. Levy, "Extending the graphic pipeline with new GPU-accelerated primitives," *International gOcad Meeting, Nancy, France*, 2004.

[102] C. Sigg, T. Weyrich, M. Botsch, and M. Gross, "GPU-Based Ray-Casting of Quadratic Surfaces," *Symposium on Point-Based Graphics*, pp. 59–65, 2006.

[103] S. Doutreligne, T. Cragnolini, S. Pasquali, P. Derreumaux, and M. Baaden, "UnityMol: Interactive scientific visualization for integrative biology," in *Large Data Analysis and Visualization (LDAV), 2014 IEEE 4th Symposium on*, pp. 109–110, nov 2014.

[104] S. Grottel, M. Krone, C. Muller, G. Reina, and T. Ertl, "MegaMol – A Prototyping Framework for Particle-based Visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 21, pp. 201–214, feb 2015.

[105] M. Chavent, A. Vanel, A. Tek, B. Levy, S. Robert, B. Raffin, and M. Baaden, "GPU-Accelerated Atom and Dynamic Bond Visualization Using Hyperballs : A Unified Algorithm for Balls , Sticks , and Hyperboloids," *Journal of Computational Chemistry*, vol. 32, no. 13, pp. 2924–2935, 2011.

[106] O. D. Lampe, I. Viola, N. Reuter, and H. Hauser, "Two-Level Approach to Efficient Visualization of Protein Dynamics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, pp. 1616–1623, nov 2007.

[107] A. Leach, *Molecular Modelling: Principles and Applications (2nd Edition)*. Pearson, 2 ed., 2001.

[108] S. Grottel, G. Reina, and T. Ertl, "Optimized data transfer for time-dependent, GPU-based glyphs," *IEEE Pacific Visualization Symposium, PacificVis 2009 - Proceedings*, pp. 65–72, 2009.

[109] S. Grottel, G. Reina, C. Dachsbacher, and T. Ertl, “Coherent Culling and Shading for Large Molecular Dynamics Visualization,” *Computer Graphics Forum*, vol. 29, pp. 953–962, aug 2010.

[110] N. Greene, M. Kass, and G. Miller, “Hierarchical Z-buffer Visibility,” in *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’93, (New York, NY, USA), pp. 231–238, ACM, 1993.

[111] N. Lindow, D. Baum, and H. C. Hege, “Interactive Rendering of Materials and Biological Structures on Atomic and Nanoscopic Scale,” *Computer Graphics Forum*, vol. 31, no. 3pt4, pp. 1325–1334, 2012.

[112] M. Falk, M. Krone, and T. Ertl, “Atomistic Visualization of Mesoscopic Whole-Cell Simulations Using Ray-Casted Instancing,” *Computer Graphics Forum*, vol. 32, no. 8, pp. 195–206, 2013.

[113] M. Le Muzic, J. Parulek, A. K. Stavrum, and I. Viola, “Illustrative Visualization of Molecular Reactions Using Omniscient Intelligence and Passive Agents,” *Computer Graphics Forum*, vol. 33, no. 3, pp. 141–150, 2014.

[114] S. Steinbacher, R. Bass, P. Strop, and D. C. Rees, “Structures of the Prokaryotic Mechanosensitive Channels MscL and MscS,” in *Mechanosensitive Ion Channels, Part A*, vol. 58 of *Current Topics in Membranes*, pp. 1–24, Academic Press, 2007.

[115] G. Mancini, I. D’Annessa, A. Coletta, N. Sanna, G. Chillemi, and A. Desideri, “Structural and dynamical effects induced by the anticancer drug topotecan on the human topoisomerase i - dna complex,” *PLOS ONE*, vol. 5, pp. 1–10, 06 2010.

[116] G. S. Couch, D. K. Hendrix, and T. E. Ferrin, “Nucleic acid visualization with UCSF Chimera,” *Nucleic Acids Research*, vol. 34, no. 4, p. e29, 2006.

[117] P. D. Bagur, N. Shivashankar, and V. Natarajan, “Improved Quadric Surface Impostors for Large Bio-molecular Visualization,” in *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing*, ICVGIP ’12, (New York, NY, USA), pp. 33:1—-33:8, ACM, 2012.

[118] M. Krone, K. Bidmon, and T. Ertl, “GPU-based Visualisation of Protein Secondary Structure,” in *Theory and Practice of Computer Graphics* (I. S. Lim and W. Tang, eds.), The Eurographics Association, 2008.

[119] M. Wahle and S. Birmanns, “GPU-accelerated visualization of protein dynamics in ribbon mode,” in *Proc. SPIE*, vol. 7868, jan 2011.

[120] G. Fermi, M. F. Perutz, B. Shaanan, and R. Fourme, “The crystal structure of human deoxyhaemoglobin at 1.74 Å resolution,” *Journal of Molecular Biology*, vol. 175, no. 2, pp. 159–174, 1984.

[121] M. Krone, J. Stone, T. Ertl, and K. Schulten, “Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories,” in *EuroVis - Short Papers* (M. Meyer and T. Weinkaufs, eds.), The Eurographics Association, 2012.

[122] M. F. Sanner and A. J. Olson, “Real time surface reconstruction for moving molecular fragments.,” *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, pp. 385–396, 1997.

[123] J. Ryu, R. Park, and D.-S. Kim, “Molecular surfaces on proteins via beta shapes,” *Computer-Aided Design*, vol. 39, no. 12, pp. 1042–1057, 2007.

[124] D.-S. Kim, J. Seo, D. Kim, J. Ryu, and C.-H. Cho, “Three-dimensional beta shapes,” *Computer-Aided Design*, vol. 38, no. 11, pp. 1179–1191, 2006.

[125] H. Edelsbrunner and E. P. Mücke, “Three-dimensional Alpha Shapes,” *ACM Trans. Graph.*, vol. 13, no. 1, pp. 43–72, 1994.

[126] R. A. Laskowski, “SURFNET: A program for visualizing molecular surfaces, cavities, and intermolecular interactions,” *Journal of Molecular Graphics*, vol. 13, no. 5, pp. 323–330, 1995.

[127] J. Giard and B. Macq, “Molecular Surface Mesh Generation by Filtering Electron Density Map,” *International Journal of Biomedical Imaging*, vol. 2010, pp. 1–9, 2010.

[128] J. Parulek and I. Viola, “Implicit Representation of Molecular Surfaces,” in *Proceedings of the 2012 IEEE Pacific Visualization Symposium*, PACIFICVIS ’12, (Washington, DC, USA), pp. 217–224, IEEE Computer Society, 2012.

[129] J. Parulek and A. Brambilla, “Fast Blending Scheme for Molecular Surface Representation,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, pp. 2653–2662, dec 2013.

[130] J. F. Blinn, “A Generalization of Algebraic Surface Drawing,” *ACM Transactions on Graphics*, vol. 1, no. 3, pp. 235–256, 1982.

[131] J. A. Grant and B. T. Pickup, “A Gaussian Description of Molecular Shape,” *The Journal of Physical Chemistry*, vol. 99, no. 11, pp. 3503–3510, 1995.

[132] H. Edelsbrunner, “Deformable Smooth Surface Design,” *Discrete & Computational Geometry*, vol. 21, no. 1, pp. 87–115, 1999.

[133] P. W. Bates, G. W. Wei, and S. Zhao, “Minimal molecular surfaces and their applications,” *Journal of Computational Chemistry*, vol. 29, no. 3, pp. 380–391, 2007.

[134] N. Lindow, D. Baum, and H. C. Hege, “Ligand Excluded Surface: A New Type of Molecular Surface,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2486–2495, 2014.

[135] N. Lindow, D. Baum, S. Prohaska, and H.-C. Hege, “Accelerated Visualization of Dynamic Molecular Surfaces,” *Computer Graphics Forum*, vol. 29, no. 3, pp. 943–952, 2010.

[136] J. Ryu, Y. Cho, and D.-S. Kim, “Triangulation of Molecular Surfaces,” *Computer-Aided Design*, vol. 41, no. 6, pp. 463–478, 2009.

[137] J. Zhang and Z. Shi, “Triangulation of molecular surfaces based on extracting surface atoms,” *Computers & Graphics*, vol. 38, pp. 291–299, 2014.

[138] W. E. Lorensen and H. E. Cline, “Marching Cubes: A High Resolution 3D Surface Construction Algorithm,” in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’87, pp. 163–169, ACM, 1987.

[139] T. S. Newman and H. Yi, “A survey of the marching cubes algorithm,” *Computers & Graphics*, vol. 30, no. 5, pp. 854–879, 2006.

[140] S. L. Chan and E. O. Purisima, “Molecular surface generation using marching tetrahedra,” *Journal of Computational Chemistry*, vol. 19, no. 11, pp. 1268–1277, 1998.

[141] T. Can, C.-I. Chen, and Y.-F. Wang, “Efficient molecular surface generation using level-set methods,” *Journal of Molecular Graphics and Modelling*, vol. 25, no. 4, pp. 442–454, 2006.

[142] Z. Yu, “A list-based method for fast generation of molecular surfaces,” in *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5909–5912, 2009.

[143] S. Decherchi and W. Rocchia, “A general and Robust Ray-Casting-Based Algorithm for Triangulating Surfaces at the Nanoscale,” *PLOS ONE*, vol. 8, no. 4, pp. 1–15, 2013.

[144] Y. Kanamori, Z. Szego, and T. Nishita, “GPU-based Fast Ray Casting for a Large Number of Metaballs,” *Computer Graphics Forum*, vol. 27, no. 2, pp. 351–360, 2008.

[145] L. Szécsi and D. Illés, “Real-Time Metaball Ray Casting with Fragment Lists,” in *Eurographics 2012 - Short Papers* (C. Andujar and E. Puppo, eds.), The Eurographics Association, 2012.

[146] A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen, “Fast Ray Tracing of Arbitrary Implicit Surfaces with Interval and Affine Arithmetic,” *Computer Graphics Forum*, vol. 28, no. 1, pp. 26–40, 2009.

[147] D. Kauker, M. Krone, A. Panagiotidis, G. Reina, and T. Ertl, “Rendering Molecular Surfaces using Order-Independent Transparency,” in *Eurographics Symposium on Parallel Graphics and Visualization* (F. Marton and K. Moreland, eds.), The Eurographics Association, 2013.

[148] M. Krone, K. Bidmon, and T. Ertl, “Interactive Visualization of Molecular Surface Dynamics,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 1391–1398, nov 2009.

[149] M. Krone, C. Dachsbacher, and T. Ertl, “Parallel Computation and Interactive Visualization of Time-varying Solvent Excluded Surfaces,” in *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, BCB ’10, (New York, NY, USA), pp. 402–405, ACM, 2010.

[150] M. Krone, S. Grottel, and T. Ertl, “Parallel Contour-Buildup algorithm for the molecular surface,” in *2011 IEEE Symposium on Biological Data Visualization (BioVis).*, pp. 17–22, 2011.

[151] N. Corporation, “CUDA Parallel Computing Platform.” `http://www.nvidia.com/object/cuda_home_new.html`.

[152] C. D. Hansen and C. R. Johnson, eds., *The Visualization Handbook.* Academic Press, 2004.

[153] R. Wenger, *Isosurfaces: Geometry, Topology, and Algorithms.* A K Peters/CRC Press, 1 ed., 2013.

[154] K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, and D. Weiskopf, *Real-Time Volume Graphics.* A K Peters, 1 ed., 2006.

[155] F. Goetz, T. Junklewitz, and G. Domik, "Real-Time Marching Cubes on the Vertex Shader," in *EG Short Presentations* (J. Dingliana and F. Ganovelli, eds.), The Eurographics Association, 2005.

[156] G. Johansson and H. Carr, "Accelerating Marching Cubes with Graphics Hardware," in *Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '06, IBM Corp., 2006.

[157] N. Max, "Optical models for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99–108, 1995.

[158] T. J. Cullip and U. Neumann, "Accelerating Volume Reconstruction With 3D Texture Hardware," tech. rep., Chapel Hill, NC, USA, 1993.

[159] J. Kruger and R. Westermann, "Acceleration techniques for GPU-based volume rendering," in *IEEE Visualization, 2003. VIS 2003.*, pp. 287–292, 2003.

[160] S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser, "Smart Hardware-accelerated Volume Rendering," in *Proceedings of the Symposium on Data Visualisation 2003*, VISSYM '03, pp. 231–238, Eurographics Association, 2003.

[161] S. Green, "Procedural volumetric fireball effect." NVIDIA Software Development Kit (SDK), 2004.

[162] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl, "A simple and flexible volume rendering framework for graphics-hardware-based raycasting," in *Fourth International Workshop on Volume Graphics, 2005.*, pp. 187–241, 2005.

[163] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross, "Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces," *Computer Graphics Forum*, vol. 24, no. 3, pp. 303–312, 2005.

[164] J. F. Blinn, "Models of Light Reflection for Computer Synthesized Pictures," in *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '77, (New York, NY, USA), pp. 192–198, ACM, 1977.

[165] H. Scharsach, M. Hadwiger, A. Neubauer, S. Wolfsberger, and K. Bühler, "Perspective Isosurface and Direct Volume Rendering for Virtual Endoscopy Applications," in *EUROVIS - Eurographics /IEEE VGTC Symposium on Visualization* (B. S. Santos, T. Ertl, and K. Joy, eds.), The Eurographics Association, 2006.

[166] S. Mehta, K. Hazzard, R. Machiraju, S. Parthasarathy, and J. Wilkins, "Detection and visualization of anomalous structures in molecular dynamics simulation data," in *IEEE Visualization 2004*, pp. 465–472, 2004.

[167] W. Qiao, D. S. Ebert, A. Entezari, M. Korkusinski, and G. Klimeck, "VolQD: direct volume rendering of multi-million atom quantum dot simulations," in *VIS 05. IEEE Visualization, 2005.*, pp. 319–326, 2005.

[168] Y. Jang and U. Varetto, "Interactive Volume Rendering of Functional Representations in Quantum Chemistry," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 1579–5186, nov 2009.

[169] P. Rheingans and S. Joshi, "Visualization of Molecules with Positional Uncertainty," in *Data Visualization '99: Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization in Vienna, Austria, May 26–28, 1999* (E. Gröller, H. Löffelmann, and W. Ribarsky, eds.), pp. 299–306, Vienna: Springer Vienna, 1999.

[170] J. Schmidt-Ehrenberg, D. Baum, and H. C. Hege, "Visualizing dynamic molecular conformations," in *IEEE Visualization, 2002. VIS 2002.*, pp. 235–242, nov 2002.

[171] A. Knoll, M. K. Y. Chan, K. C. Lau, B. Liu, J. Greeley, L. Curtiss, M. Hereld, and M. E. Papka, "UNCERTAINTY CLASSIFICATION AND VISUALIZATION OF MOLECULAR INTERFACES," *International Journal for Uncertainty Quantification*, vol. 3, no. 2, pp. 157–169, 2013.

[172] Z. Xu, A. L. Horwich, and P. B. Sigler, "The crystal structure of the asymmetric GroEL-GroES-(ADP)7 chaperonin complex," *Nature*, vol. 388, pp. 741–750, aug 1997.

[173] M. Tarini, P. Cignoni, and C. Montani, "Ambient occlusion and edge cueing to enhance real time molecular visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1237–1244, 2006.

[174] L. Williams, "Casting Curved Shadows on Curved Surfaces," in *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '78, (New York, NY, USA), pp. 270–274, ACM, 1978.

[175] S. Zhukov, A. Iones, and G. Kronin, "An ambient light illumination model," in *Rendering Techniques '98: Proceedings of the Eurographics Workshop in Vienna, Austria, June 29—July 1, 1998* (G. Drettakis and N. Max, eds.), pp. 45–55, Vienna: Springer Vienna, 1998.

[176] S. Grottel, M. Krone, K. Scharnowski, and T. Ertl, "Object-space ambient occlusion for molecular dynamics," in *Visualization Symposium (PacificVis), 2012 IEEE Pacific*, pp. 209–216, IEEE, feb 2012.

[177] M. Wahle and W. Wriggers, "Multi-scale Visualization of Molecular Architecture Using Real-Time Ambient Occlusion in Sculptor," *PLOS Computational Biology*, vol. 11, no. 10, pp. 1–14, 2015.

[178] D. Borland, "Ambient occlusion opacity mapping for visualization of internal molecular structure," *Journal of WSCG*, vol. 19, no. 1-3, pp. 17–24, 2011.

[179] D. Jönsson, E. Sundén, A. Ynnerman, and T. Ropinski, "A Survey of Volumetric Illumination Techniques for Interactive Volume Rendering," *Computer Graphics Forum*, vol. 33, no. 1, pp. 27–51, 2014.

[180] L. Marsalek, A. K. Dehof, I. Georgiev, H. P. Lenhof, P. Slusallek, and A. Hildebrandt, "Real-Time Ray Tracing of Complex Molecular Scenes," in *2010 14th International Conference Information Visualisation*, pp. 239–245, 2010.

[181] J. E. Stone, M. Sener, K. L. Vandivort, A. Barragan, A. Singharoy, I. Teo, J. V. Ribeiro, B. Isralewitz, B. Liu, B. C. Goh, J. C. Phillips, C. MacGregor-Chatwin, M. P. Johnson, L. F. Kourkoutis, C. N. Hunter, and K. Schulten, "Atomic detail visualization of photosynthetic membranes with GPU-accelerated ray tracing," *Parallel Computing*, vol. 55, pp. 17–27, 2016.

[182] J. E. Stone, W. R. Sherman, and K. Schulten, "Immersive Molecular Visualization with Omnidirectional Stereoscopic Ray Tracing and Remote Rendering," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1048–1057, IEEE, may 2016.

[183] W. R. Sherman and A. B. Craig, *Understanding Virtual Reality: Interface, Application, and Design.* Morgan Kaufmann, 1 ed., 2002.

[184] R. Blade and M. Padgett, "Virtual Environments Standards and Terminology," in *Handbook of Virtual Environments - Design, Implementation, and Applications* (K. S. Hale and K. M. Stanney, eds.), ch. 2, pp. 23–35, CRC Press, 2 ed., 2014.

[185] G. C. Burdea and P. Coiffet, *Virtual Reality Technology.* Wiley-IEEE Press, 2 ed., 2003.

[186] S. Bryson, "Virtual reality in scientific visualization," *Communications of the ACM*, vol. 39, pp. 62–71, may 1996.

[187] G. G. Robertson, S. K. Card, and J. D. Mackinlay, "Three views of virtual reality: nonimmersive virtual reality," *Computer*, vol. 26, pp. 81–83, feb 1993.

[188] NVIDIA Corporation, "3D Vision." `http://www.nvidia.com/object/3d-vision-main.html`.

[189] Wikipedia, "Novint Technologies." `https://en.wikipedia.org/wiki/Novint_Technologies`.

[190] Oculus VR LLC, "Oculus Rift." `https://www.oculus.com/rift/`.

[191] HTC Corporation, "Vive." `https://www.vive.com`.

[192] 3D Sound Labs, "3D Sound One Audio Headphones." `http://www.3dsoundlabs.com/produit/3d-sound-one-headphones/`.

[193] NeuroDigital Technologies, "Avatar VR." `https://www.neurodigital.es/avatarvr/`.

[194] T. Hermann, A. Hunt, and J. G. Neuhoff, eds., *The Sonification Handbook*. Logos Verlag Berlin, 2011.

[195] M. Mihelj, D. Novak, and S. Begus, "Haptic Modality in Virtual Reality," in *Virtual Reality Technology and Applications*, pp. 161–194, Dordrecht: Springer Netherlands, 2014.

[196] Geomagic Inc., "The Geomagic Touch Haptic Device." `http://www.geomagic.com/en/products/phantom-omni/overview/`.

[197] B. Keshavarz, H. Hecht, and B. Lawson, "Visually Induced Motion Sickness: Causes, Characteristics, and Countermeasures," in *Handbook of Virtual Environments* (K. S. . Hale and K. M. . Stanney, eds.), Human Factors and Ergonomics, ch. 26, pp. 647–698, CRC Press, 2 ed., sep 2014.

[198] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart, "The CAVE: Audio Visual Experience Automatic Virtual Environment," *Commun. ACM*, vol. 35, no. 6, pp. 64–72, 1992.

[199] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE," *Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH '93*, pp. 135–142, 1993.

[200] A. Febretti, A. Nishimoto, T. Thigpen, J. Talandis, L. Long, J. D. Pirtle, T. Peterka, A. Verlo, M. Brown, D. Plepys, D. Sandin, L. Renambot,

A. Johnson, and J. Leigh, "CAVE2: a hybrid reality environment for immersive simulation and information analysis," in *Proc. SPIE* (M. Dolinsky and I. E. McDowall, eds.), vol. 8649, p. 864903, mar 2013.

[201] Leap Motion Inc., "Leap Motion." `https://www.leapmotion.com/`.

[202] P. Bourke, "Omni-directional stereoscopic fisheye images for immersive hemispherical dome environments," in *Proceedings of the Computer Games & Allied Technology 09 (CGAT09)* (E. Prakash, ed.), (Singapore), pp. 136–143, Research Publishing Services, 2009.

[203] W. Krueger and B. Froehlich, "Responsive Workbench," in *Virtual Reality '94: Anwendungen & Trends* (H.-J. Warnecke and H.-J. Bullinger, eds.), pp. 73–80, Berlin, Heidelberg: Springer Berlin Heidelberg, 1994.

[204] M. Czernuszenko, D. Pape, D. Sandin, T. DeFanti, G. L. Dawe, and M. D. Brown, "The ImmersaDesk and Infinity Wall Projection-based Virtual Reality Displays," *SIGGRAPH Comput. Graph.*, vol. 31, no. 2, pp. 46–49, 1997.

[205] K. Arthur, T. Preston, R. M. T. Ii, F. P. Brooks, M. C. Whitton, and W. V. Wright, "Designing and building the PIT: a head-tracked stereo workspace for two users," in *2nd International Immersive Projection Technology Workshop*, pp. 11–12, 1998.

[206] M. Billinghurst, A. Clark, and G. Lee, "A Survey of Augmented Reality," *Foundations and Trends® Human-Computer Interaction*, vol. 8, no. 2-3, pp. 73–272, 2015.

[207] E. Costanza, A. Kunz, and M. Fjeld, "Mixed Reality: A Survey," in *Human Machine Interaction: Research Results of the MMI Program* (D. Lalanne and J. Kohlas, eds.), pp. 47–68, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[208] S. M. LaValle, "VIRTUAL REALITY." `http://vr.cs.uiuc.edu/`. Free online book. According to the author, it will be published by Cambridge University Press in 2017.

[209] B. Laha, K. Sensharma, J. D. Schiffbauer, and D. A. Bowman, "Effects of Immersion on Visual Analysis of Volume Data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, pp. 597–606, apr 2012.

[210] B. Laha, D. A. Bowman, and J. J. Socha, "Effects of VR System Fidelity on Analyzing Isosurface Visualization of Volume Datasets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, pp. 513–522, apr 2014.

[211] Song Zhang, C. Demiralp, and D. Laidlaw, "Visualizing diffusion tensor MR images using streamtubes and streamsurfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, pp. 454–462, oct 2003.

[212] J. E. Stone, J. Gullingsrud, and K. Schulten, "A system for interactive molecular dynamics simulation," in *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, (New York, NY, USA), pp. 191–194, ACM, 2001.

[213] Y.-G. Lee and K. W. Lyons, "Smoothing haptic interaction using molecular force calculations," *Computer-Aided Design*, vol. 36, no. 1, pp. 75 – 90, 2004.

[214] A. Bolopion, B. Cagneau, S. Redon, and S. Régnier, "Haptic feedback for molecular simulation," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 237–242, Oct 2009.

[215] O. Delalande, N. Férey, G. Grasseau, and M. Baaden, "Complex molecular assemblies at hand via interactive simulations," *Journal of Computational Chemistry*, vol. 30, no. 15, pp. 2375–2387, 2009.

[216] A. Bolopion, B. Cagneau, S. Redon, and S. Régnier, "Haptic molecular simulation based on force control," in *2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 329–334, July 2010.

[217] M. Dreher, M. Piuzzi, A. Turki, M. Chavent, M. Baaden, N. Férey, S. Limet, B. Raffin, and S. Robert, "Interactive Molecular Dynamics: Scaling up to Large Systems," *Procedia Computer Science*, vol. 18, pp. 20–29, 2013.

[218] N. Luehr, A. G. B. Jin, and T. J. Martínez, "Ab Initio Interactive Molecular Dynamics on Graphical Processing Units (GPUs)," *Journal of Chemical Theory and Computation*, vol. 11, no. 10, pp. 4536–4544, 2015.

[219] N. Férey, J. Nelson, C. Martin, L. Picinali, G. Bouyer, A. Tek, P. Bourdot, J. M. Burkhardt, B. F. G. Katz, M. Ammi, C. Etchebest, and L. Autin, "Multisensory VR interaction for protein-docking in the CoRSAIRe project," *Virtual Reality*, vol. 13, pp. 273–293, dec 2009.

[220] X. Hou and O. Sourina, "Six Degree-of-Freedom Haptic Rendering for Biomolecular Docking," in *Transactions on Computational Science XII: Special Issue on Cyberworlds* (M. L. Gavrilova, C. J. K. Tan, A. Sourin, and O. Sourina, eds.), pp. 98–117, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[221] H. Nagata, H. Mizushima, and H. Tanaka, “Concept and prototype of protein-ligand docking simulator with force feedback technology,” *Bioinformatics*, vol. 18, no. 1, p. 140, 2002.

[222] K. H. Marti and M. Reiher, “Haptic quantum chemistry,” *Journal of Computational Chemistry*, vol. 30, no. 13, pp. 2010–2020, 2009.

[223] M. P. Haag and M. Reiher, “Real-time quantum chemistry,” *International Journal of Quantum Chemistry*, vol. 113, no. 1, pp. 8–20, 2013.

[224] M. P. Haag and M. Reiher, “Studying chemical reactivity in a virtual environment,” *Faraday Discuss.*, vol. 169, pp. 89–118, 2014.

[225] M. P. Haag, A. C. Vaucher, M. Bosson, S. Redon, and M. Reiher, “Interactive chemical reactivity exploration,” *ChemPhysChem*, vol. 15, no. 15, pp. 3301–3319, 2014.

[226] A. C. Vaucher, M. P. Haag, and M. Reiher, “Real-time feedback from iterative electronic structure calculations,” *Journal of Computational Chemistry*, vol. 37, no. 9, pp. 805–812, 2016.

[227] S. Birmanns and W. Wriggers, “Interactive fitting augmented by force-feedback and virtual reality,” *Journal of Structural Biology*, vol. 144, no. 1-2, pp. 123–131, 2003.

[228] C. Cruz-Neira, J. Leigh, M. Papka, C. Barnes, S. Cohen, S. Das, R. Engelmann, R. Hudson, T. Roy, L. Siegel, C. Vasilakis, T. DeFanti, and D. Sandin, “Scientists in wonderland: A report on visualization applications in the CAVE virtual reality environment,” in *Proceedings of 1993 IEEE Research Properties in Virtual Reality Symposium*, pp. 59–66, IEEE Comput. Soc. Press, 1993.

[229] M. E. Papka, T. L. Disz, R. L. Stevens, M. Pellegrino, and V. E. Taylor, “Virtual Reality Visualization of Parallel Molecular Dynamics Simulations,” in *Society for Computer Simulation*, 1995.

[230] V. E. Taylor, R. L. Stevens, and K. E. Arnold, “Parallel molecular dynamics: communication requirements for massively parallel machines,” in *Frontiers of Massively Parallel Computation, 1995. Proceedings. Frontiers ’95., Fifth Symposium on the*, pp. 156–163, feb 1995.

[231] N. Akkiraju, H. Edelsbrunner, P. Fu, and J. Qian, “Viewing geometric protein structures from inside a CAVE,” *IEEE Computer Graphics and Applications*, vol. 16, no. 4, pp. 58–61, 1996.

[232] H. Haase, J. Strassner, and F. Dai, “VR techniques for the investigation of molecule data,” *Computers & Graphics*, vol. 20, pp. 207–217, mar 1996.

[233] C. Cruz-Neira, R. Langley, and P. Bash, “VIBE: A virtual biomolecular environment for interactive molecular modeling,” *Computers & Chemistry*, vol. 20, pp. 469–477, aug 1996.

[234] R. B. Loftin, B. M. Pettitt, S. Su, C. Chuter, J. A. McCammon, C. Dede, B. Bannon, and K. Ash, “PaulingWorld: an immersive environment for collaborative exploration of molecular structures and interactions,” in *Proceedings of the 17th Nordic Internet Conference (NORDUnet ’98)*, 1998.

[235] S. Su, R. Loftin, D. Chen, and Y. Fang, “Distributed collaborative virtual environment: Paulingworld,” in *Proceedings of the 10th International Conference on Artificial Reality and Telexistence*, pp. 112–117, 2000.

[236] Z. Ai and T. Frohlich, “Molecular Dynamics Simulation in Virtual Environments,” *Computer Graphics Forum*, vol. 17, pp. 267–273, aug 1998.

[237] J. F. Prins, J. Hermans, G. Mann, L. S. Nyland, and M. Simons, “A virtual environment for steered molecular dynamics,” *Future Generation Computer Systems*, vol. 15, pp. 485–495, jul 1999.

[238] J. Leech, J. F. Prins, and J. Hermans, “SMD: visual steering of molecular dynamics for protein design,” *IEEE Computational Science and Engineering*, vol. 3, no. 4, pp. 38–45, 1996.

[239] A. Anderson and Z. Weng, “VRDD: applying irtual eality visualization to protein ocking and esign,” *Journal of Molecular Graphics and Modelling*, vol. 17, pp. 180–186, jun 1999.

[240] K. Binder and D. W. Heermann, *Monte Carlo Simulation in Statistical Physics*. Graduate Texts in Physics, Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 5 ed., 2010.

[241] M. Koutek, J. V. Hees, F. H. F. Post, and A. F. Bakker, “Virtual Spring Manipulators for Particle Steering in Molecular Dynamics on the Responsive Workbench,” in *Eighth Eurographics Workshop on Virtual Environments* (S. Mueller and W. Stuerzlinger, eds.), The Eurographics Association, 2002.

[242] A. Ghadersohi, D. E. Pape, C. M. Weeks, M. L. Green, and R. Miller, “Collaborative Scientific Visualization and Real-time Monitoring of Protein Structure Data,” tech. rep., 2005.

[243] J. W. Chastine, J. C. Brooks, Y. Zhu, G. S. Owen, R. W. Harrison, and I. T. Weber, “AMMP-Vis,” in *Proceedings of the ACM symposium on Virtual reality software and technology - VRST ’05*, VRST ’05, (New York, New York, USA), p. 8, ACM Press, 2005.

[244] K. Reda, A. Knoll, K.-i. Nomura, M. E. Papka, A. E. Johnson, and J. Leigh, “Visualizing large-scale atomistic simulations in ultra-resolution immersive environments,” in *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, pp. 59–65, IEEE, oct 2013.

[245] D. R. Glowacki, M. O’Connor, G. Calabro, J. Price, P. Tew, T. Mitchell, J. Hyde, D. P. Tew, D. J. Coughtrie, and S. McIntosh-Smith, “A GPU-accelerated immersive audio-visual framework for interaction with molecular dynamics using consumer depth sensors,” *Faraday Discuss.*, vol. 169, pp. 63–87, 2014.

[246] M. Marangoni and T. Wischgoll, “Comparative visualization of protein conformations using large high resolution displays with gestures and body tracking,” vol. 9397, p. 93970E, feb 2015.

[247] J. E. Stone, A. Kohlmeyer, K. L. Vandivort, and K. Schulten, “Immersive Molecular Visualization and Interactive Modeling with Commodity Hardware,” in *Advances in Visual Computing: 6th International Symposium, ISVC 2010, Las Vegas, NV, USA, November 29 – December 1, 2010, Proceedings, Part II* (G. Bebis, R. Boyle, B. Parvin, D. Koracin, R. Chung, R. Hammound, M. Hussain, T. Kar-Han, R. Crawfis, D. Thalmann, D. Kao, and L. Avila, eds.), pp. 382–393, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[248] Mechdyne Corporation, “CAVELib: The Ultimate Solution for 3D Virtual Reality Displays.” `https://www.mechdyne.com/software.aspx?name=CAVELib`.

[249] W. R. Sherman, D. Coming, and S. Su, “FreeVR: honoring the past, looking to the future,” in *Proc. SPIE* (M. Dolinsky and I. E. McDowall, eds.), p. 864906, mar 2013.

[250] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira, “VR Juggler: a virtual platform for virtual reality application development,” in *Proceedings IEEE Virtual Reality 2001*, pp. 89–96, 2001.

[251] R. M. Taylor II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser, “VRPN: A Device-independent, Network-transparent VR Peripheral

System," in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '01, (New York, NY, USA), pp. 55–61, ACM, 2001.

[252] Virtalis Inc., "VR For PyMOL." `http://www.virtalis.com/vr-for-pymol`.

[253] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski, "Chromium: A Stream-processing Framework for Interactive Rendering on Clusters," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02*, SIGGRAPH '02, (New York, New York, USA), p. 693, ACM Press, 2002.

[254] Center for Information Technology - University of Groningen, "PyMol in the Reality Theatre." `http://www.rug.nl/society-business/centre-for-information-technology/research/hpcv/vr_visualisation/molecular_visualisation/pymol/pymol`.

[255] "OcuMOL_Leap project on GitHub." `https://github.com/lqtza/OcuMOL_Leap`.

[256] "UnityMol Documentation." `http://www.baaden.ibpc.fr/umol/Doc/manual/html/index.html`.

[257] FEI, "Amira for Life Sciences." `https://www.fei.com/software/amira-for-life-sciences/`.

[258] UNIGINE Corp., "UNIGINE." `https://unigine.com/en/`.

[259] MiddleVR, "MiddleVR." `http://www.middlevr.com/`.

[260] TechViz, "TechViz XL." `http://www.techviz.net/techviz-xl`.

[261] WorldViz, "Vizard Virtual Reality Software." `http://www.worldviz.com/vizard-virtual-reality-software/`.

[262] "OpenSceneGraph." `http://www.openscenegraph.org/`.

[263] NVIDIA Corporation, "Quadro SLI Technology." `http://www.nvidia.com/object/sli-certified-systems-and-motherboards.html`.

[264] U. Rathmann and J. Wilgen, "Qwt - Qt Widgets for Technical Applications." `http://qwt.sourceforge.net`.

[265] Qxt Foundation, "Qxt library." `https://bitbucket.org/libqxt/libqxt/wiki/Home`.

[266] NaturalPoint Inc., "NatNet SDK." `http://optitrack.com/products/natnet-sdk/`.

[267] NaturalPoint Inc., "OptiTrack - Motion Capture Systems." `http://www.optitrack.com`.

[268] KDE, "Oxygen Project." `https://techbase.kde.org/Projects/Oxygen`.

[269] G.-T. Creation, "OpenGL Mathematics." `http://glm.g-truc.net`.

[270] D. Frishman and P. Argos, "Knowledge-based protein secondary structure assignment.," *Proteins*, vol. 23, no. 4, pp. 566–79, 1995.

[271] Open Babel, "XYZ file format." `http://openbabel.sourceforge.net/wiki/XYZ`.

[272] Gaussian Inc, "The cubegen utility." `http://www.gaussian.com/g_tech/g_ur/u_cubegen.htm`.

[273] J. Brange, G. G. Dodson, D. J. Edwards, P. H. Holden, and J. L. Whittingham, "A model of insulin fibrils derived from the x-ray crystal structure of a monomeric insulin (despentapeptide insulin)," *Proteins: Structure, Function, and Bioinformatics*, vol. 27, no. 4, pp. 507–516, 1997.

[274] B. Chandramouli, D. Di Maio, G. Mancini, V. Barone, and G. Brancato, "Breaking the Hydrophobicity of the MscL Pore: Insights into a Charge-Induced Gating Mechanism," *PLOS ONE*, vol. 10, no. 3, pp. 1–19, 2015.

[275] M. Novotny and G. J. Kleywegt, "A survey of left-handed helices in protein structures," *Journal of Molecular Biology*, vol. 347, no. 2, pp. 231–241, 2005.

[276] Didem Vardar, Christopher L. North, Cheryll Sanchez-Irizarry, Jon C. Aster, and and Stephen C. Blacklow, "Nuclear Magnetic Resonance Structure of a Prototype Lin12-Notch Repeat Module from Human Notch1," *Biochemistry*, vol. 42, no. 23, pp. 7061–7067, 2003.

[277] P. Bourke, "Polygonising a scalar field." `http://paulbourke.net/geometry/polygonise/`, 1994.

[278] S. Gibson, "Constrained elastic surface nets: Generating smooth surfaces from binary segmented data," *Medical Image Computing and Computer-Assisted Interventation-MICCAI'98*, pp. 888–898, 1998.

[279] M. Lysenko, "Smooth Voxel Terrain (Part 2)." `http://0fps.net/2012/07/12/smooth-voxel-terrain-part-2/`, 2012.

[280] Persistence of Vision Raytracer Pty. Ltd., "Persistence of Vision Raytracer (POV-Ray)." `http://www.povray.org/`.

[281] G. Mancini and C. Zazza, "F429 Regulation of Tunnels in Cytochrome P450 2B4: A Top Down Study of Multiple Molecular Dynamics Simulations," *PLOS ONE*, vol. 10, p. e0137075, sep 2015.

[282] R. Kooima, "Generalized Perspective Projection." `http://csc.lsu.edu/~kooima/articles/genperspective/index.html`, 2008.

[283] NVIDIA Corporation, "Mosaic Technology." `http://www.nvidia.com/object/nvidia-mosaic-technology.html`.

[284] Advanced Micro Devices Inc, "Multi-Monitor Eyefinity Technology." `http://www.amd.com/en-us/innovations/software-technologies/technologies-gaming/eyefinity`.

[285] T. Lottes, "FXAA," *Technical report, NVIDIA Corporation*, 2009.

[286] Oculus VR LLC, "Oculus Development Center." `https://developer.oculus.com/`.

[287] JTKSOFT, "JoyToKey." `http://joytokey.net/en/`.

[288] Valve Software, "OpenVR SDK." `https://github.com/ValveSoftware/openvr`.

[289] T. Porter and T. Duff, "Compositing Digital Images," in *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, (New York, NY, USA), pp. 253–259, ACM, 1984.

[290] M. McGuire and L. Bavoil, "Weighted Blended Order-Independent Transparency," *Journal of Computer Graphics Techniques (JCGT)*, vol. 2, no. 2, pp. 122–141, 2013.

[291] C. Everitt, "Interactive order-independent transparency," *Technical report, NVIDIA Corporation*, vol. 2, no. 6, p. 7, 2001.

[292] L. Bavoil and K. Myers, "Order independent transparency with dual depth peeling," *Technical report, NVIDIA Corporation*, pp. 1–12, 2008.

[293] B. Liu, L. Y. Wei, Y. Q. Xu, and E. Wu, "Multi-layer depth peeling via fragment sort," in *Computer-Aided Design and Computer Graphics, 2009. CAD/Graphics '09. 11th IEEE International Conference on*, pp. 452–456, 2009.

[294] H. Meshkin, "Sort-independent alpha blending," *Perpetual Entertainment, GDC Talk*, 2007.

[295] M. Maule, J. L. D. Comba, R. P. Torchelsen, and R. Bastos, "A survey of raster-based transparency techniques," *Computers & Graphics*, vol. 35, no. 6, pp. 1023–1034, 2011.

[296] M. McGuire, "Casual Effects: Implementing Weighted, Blended Order-Independent Transparency." `http://casual-effects.blogspot.it/2015/03/implemented-weighted-blended-order.html`, 2015.