Visualization Methods for Molecular Studies on the Web Platform

Marco Callieri*
Visual Computing Lab
ISTI-CNR

Raluca Mihaela Andrei†
Scuola Normale Superiore, Pisa
Scientific Visualization Unit
IFC-CNR

Marco Di Benedetto‡
Visual Computing Lab
ISTI-CNR

Monica Zoppè§
Scientific Visualization Unit
IFC-CNR

Roberto Scopigno¶
Visual Computing Lab
ISTI-CNR

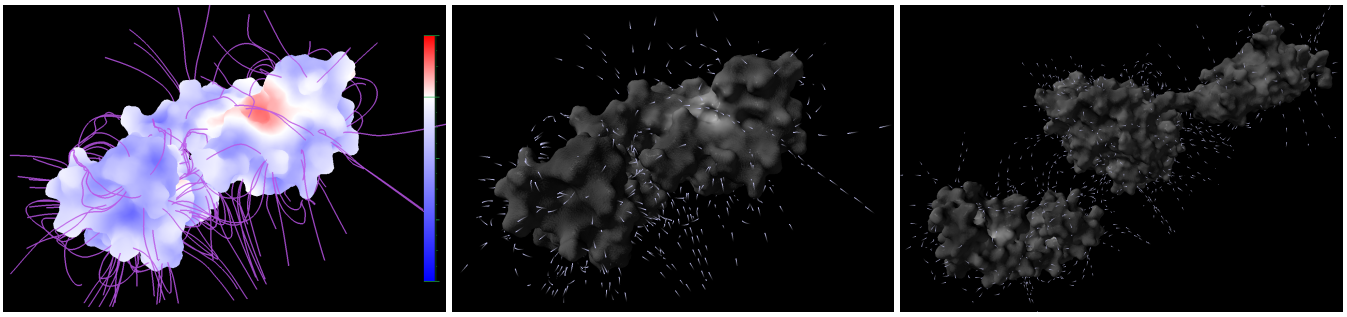

Figure 1: *Standard representation of molecular surface properties using color ramps and field lines (leftmost), the same properties drawn using complex shading techniques (center) and the electrical interaction of two proteins (rightmost), rendered on a Web Page by using SpiderGL and WebGL.*

Abstract

This work presents a technical solution for the creation of visualization schemes for biological data on the web platform. The proposed technology tries to overcome the standard approach of molecular/biochemical visualization tools, which generally provide a fixed set of visualization methods. This goal is reached by exploiting the capabilities of the WebGL API and the high level objects of the SpiderGL library, these features will give the users the possibility to implement an arbitrary visualization scheme, while keeping simple the implementation process. To better explain the philosophy and capabilities of this technology, we will describe the implementation of the web version of a specific visualization method, demonstrating how it can deal with both the requirements of scientific rigor in manipulating the data and the necessity to produce flexible and appealing rendering styles.

CR Categories: I.3.2 [Graphics Systems]: Distributed/network graphics—; J.3 [Life and Medical Sciences]: Biology and genetics —;

Keywords: Web platform,Molecular Biology, Molecular Surface Visualization, Protein Structure, physico-chemical properties, Interactive 3D, WebGL

1 Introduction

Interactive visualization of molecular structures and physico-chemical data is an important and interesting research field which span from the Computer Graphics world to the Biological and Molecular studies. The amount of complex structures that is available through public repositories and the level of detail of biochemical datasets which can be manipulated by physico-chemical tools has greatly increased in the last years, making it essential to employ dedicated visualization techniques to make an effective use of these data. While it may be easy to draw even large molecular datasets as a series of atoms (Van der Waals spheres) using simple rendering methods based on impostors and other tricks, the precise rendering of a high resolution molecular surface involves the management of more complex geometry. Furthermore, when it is necessary to represent interaction between different molecules or to introduce the rendering of further 3D elements and data layers (as in the examples of Figure 1), the required computational and rendering capabilities do increase significantly.

A previous work in the field of visualization of molecular structures, QuteMol [Tarini et al. 2006], has shown that, by using advanced shading techniques, it is much easier to convey the information regarding the geometry and structure of the molecule. We believe the same reasoning may also be applied to the visualization of other physico-chemical data: by using custom shading and rendering techniques the same improvement in clarity and expressiveness can be attained. This is however quite difficult at the moment,

*e-mail: callieri@isti.cnr.it
†e-mail: r.andrei@sns.it
‡e-mail: dibenedetto@isti.cnr.it
§e-mail: mzoppe@ifc.cnr.it
¶e-mail: scopigno@isti.cnr.it

since it is rarely possible to finely control the rendering pipeline and shading process inside existing visualization tools, especially when working on on-line platforms. Here are some possible scenarios we are considering:

- a research group interested in proposing a new visualization method may want to publish a web page which shows an interactive version of such visualization;
- a public repository of biological data structures may want to let its users view the available data using a custom shading, designed to effectively show the characteristics of the provided information;
- an educational-oriented entity may want to present biological phenomena to a large public, possibly including a non-specialist audience, by using visualization method that are not just scientifically accurate, but also visually pleasant.

These three scenarios exemplify the need for advanced visualization methods, but also for the use of the web platform. Pervasive and easy to access, the web platform is becoming more and more important for sharing data, processing methods and visualization techniques. It is easy to foresee that the research community working on visualization for cellular and molecular biology will find in the web platform the ideal media for the purposes of research, education and science divulgation.

Up to now, the capabilities of web browsers to efficiently manipulate and display 3D content have been very limited. The task of putting online three-dimensional data has exploited the use of commercial or custom browsers plugins, and has been characterized by a series of problems, like low portability (each plugin/extension would work only on a subset of browsers and operating systems), scarce flexibility (in most cases the visualization plugins offered no way to configure the drawing pipeline or add special rendering modes) and poor performance (the different software layers of network, O.S., sandbox and plugin introduced lag and computational overhead, separating too much the rendering from the hardware layer). A suitable solution for this problem may reside in the upcoming standard of WebGL [Group 2009b], which is an API specification which defines the web-oriented analogous of the OpenGL API. The most interesting feature of this API is that it is implemented directly inside the browser, with direct control over the graphics hardware. This will help overcoming the compatibility problems and result in a much more efficient and performing platform. Deriving from the specifications of OpenGL-ES, WebGL provides a completely customizable rendering pipeline and the entire shading process is controlled through hardware-level GLSL shaders. This shader-based nature of WebGL is perfectly suited to cope with the need of creating a custom visualization scheme. Direct access to the hardware layer means not only better performances, but also the possibility to exploit the full repertoire of techniques and experience accumulated in years of computer graphics research.

Of course, WebGL alone is not enough to answer the needs of people interested in biochemical visualization (which are likely not experts in graphical programming); following the design philosophy of OpenGL, WebGL is a very low-level API which requires a good knowledge of computer graphics techniques and coding skills. It is therefore necessary, to ease the use of this technology, to introduce a library able to wrap the most low level function, while giving the user the ability to dive into implementation details, when needed. As the WebGL standard is taking shape, different wrapping libraries are appearing on the web [DeLillo 2009; Brunt 2010; Kay 2009]; one of these libraries, SpiderGL [Di Benedetto 2010], seems to provide the right balance between the ease of use of the higher level functions and the possibility to fully control the rendering pipeline. We believe that the use of WebGL through the SpiderGL library will prove to be a very powerful platform to implement visualization methods on the web for the molecular biology and physico-chemical research community.

We review in Section 2 some of the previous work in the field of both molecular visualization and on-line publishing of 3D content. Then, in Section 3, we introduce the basic ideas of the proposed technology describing its main philosophy and by presenting the core library used. Finally, as an example of the use of this technology, we show in Section 4 how a specific visualization method has been adapted to the web platform.

2 Previous Work

2.1 Molecular visualization Off-Line and On-Line

The solution of the 3D structure of myoglobin in 1958 by Kendrew [Kendrew et al. 1958] marked the beginning of the new era of protein structural biology. Since then, a large number of protein structures have been solved and today the Protein Data Bank counts over 60.000 entries [Berman et al. 2003]. With the availability of all these data and the advance of computer graphics technologies, many research groups have developed tools for the manipulation and visualization of 3D structures such as VMD [Humphrey et al. 1996], SPDBViewer [Guex and Peitsch 1997], Chimera [Pettersen et al. 2004] and PyMOL [Delano 2002]. Beside working on the atomic structure, most programs can nowadays also calculate surface features such as electrostatic potential (using, for example, tools like APBS [Baker et al. 2001] or DelPhi [Rocchia et al. 2002]) and hydropathy [Kyte and Doolittle 1982].

In addition to the many standalone visualization tools, there are also web viewers especially designed for molecular structures, such as Jmol [jmo 2002] and MDL Chime, which represent a simple way to visualize molecules directly on browser. MDL Chime, used by the Protein Explorer website was gradually phased out in favor of Jmol, which is nowadays the most used plugin for molecular visualization, used by websites such as Proteopedia and RCSB PDB Protein Data Bank.

Following the advance of techniques for the generation of CG movies, in the last few years many different groups focused on the creation of animated movies depicting biological molecules and cellular processes. The movies range from the simple representations of the mechanical functioning of a single protein, to complex events involving many subjects. These works are important scientific efforts and add to their educational value the bonus of rising interest in the general public to approach biology. Some of these examples are collected on websites [McGill 2010; SCIVIS 2005].

2.2 3D Content on Web

The web platform has acquired through the years the ability to efficiently incorporate and deliver many different kinds of digital data such as still images, videos and sound. With respect to these additions, the management of 3D content through the web comes with a considerable delay. The reasons for this delay are likely to be found in the higher requirements of 3D graphics in terms of computational power, but also because the lack of a strong unifying standard behind the 3D content.

Several technologies have been developed over the years to achieve this integration. The Virtual Markup Modeling Language (VRML) [Raggett 1994] (then replaced by X3D [Don Brutzmann 2007]) was proposed as a text based format for specifying 3D scenes in terms of geometry and material properties and for the

definition of basic user interaction. The format itself was a standard, but the rendering in the web browser was relaying on specific plugins. The Java Applets are probably the most used method to add dynamic content, not necessarily 3D, in the web browsers. The philosophy of Java applets is that the URL to the applet and its data are put in the HTML page and then executed by the Java Virtual Machine, a third part component. The implementation of JVM on all the operating systems made Java applets ubiquitous and the introduction of binding to OpenGL such as JOGL [JOG] added control on the 3D graphics hardware. A similar idea lies behind the ActiveX [Microsoft Corporation 1996] technology, developed by Microsoft from 1996. Unlike Java Applets, ActiveX controls are not bytecode but dynamic linked Windows libraries which share the same memory space as the calling process (i.e. the browser), and so much faster to execute. These technologies enable the incorporation of 3D graphics in a web page but they all do it by handling a special element of the page itself with a third party component.

WebGL [Group 2009b] is an API specification produced by the Khronos group [Group 2009a] and, as the name suggests, defines the JavaScript analogous of the OpenGL API for C++. WebGL closely matches OpenGL|ES 2.0 and, extremely important, uses GLSL as the language for shader programs, which means that the shader core of existent applications can be reused for their JavaScript/WebGL version. Since WebGL is a specification, it is up to the web browsers developer to implement it. At the time of this writing, WebGL is supported in the nightly build versions of the most used web browsers (Firefox, Chrome, Safari), and a number of JavaScript libraries are being developed to provide higher level functionalities to create 3D graphics applications. For example WebGLU [DeLillo 2009], which is the WebGL correspondent of GLU [OpenGL ARB], provides wrappings for placing the camera in the scene or for creating simple geometric primitives, other libraries such as GLGE [Brunt 2010] or SceneJS [Kay 2009] uses WebGL for implementing a scene graph based rendering and animation engines.

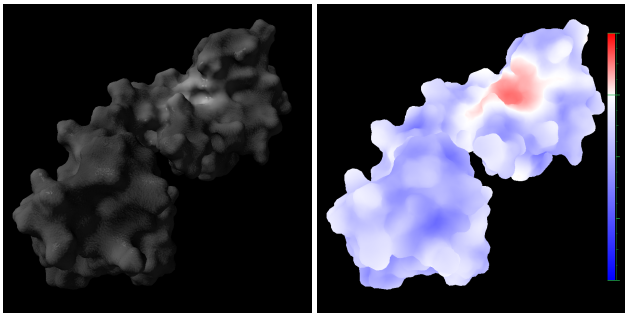

Figure 2: *Lipophilic Potential mapped on the surface of a Calcium-bound Calmodulin. On right, visualization using standard color ramp; on left, visualization using advanced shaders. The light color and the specularity clearly indicates a liphophilic patch on the right part of the molecule, while the dark, dull and rough surface indicates a more hydrophilic area.*

3 Building a custom web-based Visualization Scheme

As stated in the introduction, the aim of this work is to propose a technology for the implementation of advanced visualization schemes for molecular and biochemical data on a web platform. We are interested in a base technology that is able to cope with the needs of a completely customizable rendering while providing enough basic structures and higher level functions to be usable without a major programming effort.

Our idea is that the WebGL standard is able to provide the performances and fine-control over the rendering, since it directly uses the hardware layer for rendering, is built around the idea of a fully customizable rendering pipeline and gives access to the use of GLSL shaders, a really powerful instrument to achieve the desired visual output. While these features are absolutely necessary to reach our goal, they are not sufficient to provide a really usable development platform because the available functions are too low level to be effectively used (especially by a community of people with little or no experience in CG programming). By introducing a wrapping library as SpiderGL, it is possible to enrich this platform with a series of higher level functions that may be used as building blocks to implement the desired rendering method. As a final ingredient, we have also to consider what can be attained by a clever use of the JavaScript. Exploiting the ease of use and the expressive power of this language it is possible to read the source scientific data and do all the needed calculation.

3.1 SpiderGL

The core library used in this work is SpiderGL: a recent, ongoing project which aims to provide an easily usable but powerful wrapping to the lower-level WebGL functions. Most of the available JavaScript graphics libraries and browser plugins for 3D data management are based on the paradigm of *scene graph*. This choice is perfectly natural, in the sense that it mimics the idea of a three-dimensional scene composed of objects, rendered from a given point of view. However, this solution cannot fully answer the need of scientific visualization, where it is often needed to use very diverse data, and render them in a very controlled way. SpiderGL, on the other hand, does not follow this paradigm, but provides a set of data structures and algorithms to support the management of geometric and mathematical entities, in order to simplify the creation of arbitrary visualization prototypes. The idea of this library is to provide a complete wrapping layer to WebGL that, while hiding the details through higher level functions, allows full access to the native API.

To ease the creation of graphical applications, SpiderGL provides a series of classes and functions which cover the various aspects and levels of implementation of a CG program:
Basic structures: linear algebra algorithms for 3D points and vectors are very common tools for the CG developer; the geometry module of SpiderGL implements the essential mathematical objects such as vectors (2,3 and 4 components), quaternions and matrices, along with basic operations on them.
2D/3D Data: one of the fundamental parts of a graphics library is the management of data structures for the definition of 3D objects (meshes), textures and the other components used in the rendering process. While at low level, WebGL works directly on streams of vertex attributes and indices, SpiderGL, to provide a more structured object to manage, implements a mesh object, based on the usual paradigm of vertices+triangle connectivity. For a flexible but efficient use, SpiderGL supplies two different data structures: the first one, `SglMeshJS`, can be freely accessed and modified within the user script; the other, `SglMeshGL`, is generated from the first and used at GPU level for rendering. Management of textures is done through some specific functions which enable the creation of texture from images or raw data, texture sampler options and texture unit binding. A final set of classes is used to manage vertex shaders, fragment shaders and shader programs, with support from compilation feedback, binding and attribute management.
Scene management: while not introducing a scene-graph, SpiderGL provides some specific helpers to place entities in the

3D space, set the viewpoint and simplify the user interaction with the 3D elements. The matrix stack, legacy of the OpenGL library, is still extremely useful when populating a scene and it is implemented in the `SglTransformStack` object, which offers many different methods to manipulate and access these matrices. Other helpful classes include a camera object which implements the typical paradigm used in first-person shooter games (`SglFirstPersonCamera`) and a trackball manipulator (`SglTrackball`) for object inspection with pan, zoom, rotation and scaling operations.

Rendering: WebGL redering is a long series of function used to manage all the data streams, bind streams to attributes, select shaders and control the GL status. In SpiderGL, 3D mesh rendering is managed through the `SglMeshGLRenderer` helper class. This class takes care of all the setup steps required by WebGL and tries to simplify the stream mapping process by automatically match all the most used attributes. Additional helper classes give finer control over the mapping of data streams and deal with the management of shader parameters.

Application: interactivity is one of the focus of this library; for this reason, SpiderGL provides an event-based mechanism which is able to collect events from all the DOM and efficiently dispatch them to multiple listeners. Other application-level support structures like a log system are available through specific classes. Another extremely useful feature on the web platform is the asynchronous loading: many rendering algorithms requires the ability of asynchronous loading of data. Even if JavaScript does still not support multithreading, SpiderGL implements a simple mechanism based on the `XMLHttpRequest` object to queue data to be loaded and set a callback functions which will be invoked whenever the transfer of the requested data has completed.

3.2 Data Importing and Management

While using JavaScript it is not possible to read binary data, this may not be a major problem for the need of importing data from molecular databases or physico-chemical tools. Many biological-related file formats use ASCII coding, which make the parsing really straightforward. As an example, the importer for the PDB file format, which describe the structure of a molecule, is just a few lines long. Here it is possible to see part of the importing code and how the predefined JavaScript functions for tokenization help its parsing:

```
function AtomListFromPDB(atomlist, pdb_txt){
[.....]
var lines = pdb_txt.split("\n");
for (var lineIndex in lines) {
//atom line example
//ATOM 16 O ASP A 2 10.6 5.1 -6.1 0.0 0.0 O

   tokens = line.split(" ");
   if (tokens[0] == "ATOM") {  // atom line
      var atomtype = tokens[11];

      var position = [];
      position.push(parseFloat(tokens[6]));
      position.push(parseFloat(tokens[7]));
      position.push(parseFloat(tokens[8]));
[.....]
```

More recent biochemical tools may also export data in XML format, which is directly readable by JavaScript. Three-dimensional geometries are normally stored using one of the many standard file formats, and can generally be converted from one format to another; SpiderGL does at the moment support OBJ and COLLADA formats (other importers will follow), and different other formats may be parsed by using JavaScript. Less structured data may be imported also by making the data source export in the JSON format, which is quite easy to write and is natively supported by JavaScript interpreters. Since most physico-chemical tools have a scripting layer which can be used to specify custom data exporters, this is often a viable option.

Most of the more interesting visualization methods, however are not just based on loading existing data and displaying it in a controlled fashion, but also relies on some kind of data processing. JavaScript may be an effective ally also in this case, thanks to its ease of use, the great flexibility in data structure (dynamic typing, associative arrays, an advanced garbage collector), the presence of many built-in functions and its expressive power. And if it is true that probably JavaScript will never reach the computational efficiency of compiled C++ code, the newest interpreters and the introduction of just-in-time compilers have significantly reduced the gap. It is possible to say that, in this specific scenario, where most of the computational requirements have been moved from CPU to GPU, the difference between the two language is neglectable.

It is also interesting that, since all the computation is done at the JavaScript level and all the visualization code is embedded in the page, it is not possible to effectively hide the data or their processing. This impossibility of building a closed, protected system may be perceived as a serious limitation for industrial-related applications. However, in these context, this same limitation may turn out to be a very positive feature, since the transparency of the data processing (you may check that no hidden tweaking is done on the data) and the possibility of sharing knowledge (by letting others reuse your visualization code) are of capital importance in the fields of research and educational tools.

3.3 Implementation

Having all the necessary building blocks to load, manipulate and render the data, it is possible to build the desired visualization method. The setup of the scene and the definition of the rendering pipeline work similarly to a standard visualization application.

Looking at a webpage with dynamic SpiderGL content, it is possible to see that all of the page logic is defined in the scripting part of the HEAD section, while on the BODY section there is just the page structure and the interface elements that will be used for user interaction (like buttons, text areas and other controls). Among these elements, the most important is an html `canvas` object, that is the place where the WebGL layer does the on-screen rendering.

```
<canvas id="SGL_CANVAS" style="border: 1px solid gray" ↩
     width="900" height="600"></canvas>
```

This canvas is registered as the output area at the end of the scripting; a specific function connects the various events of the canvas to a script object.

```
var glMolViewer = new SpiderGLMolViewer();
sglRegisterCanvas("SGL_CANVAS", glMolViewer, 30.0);
```

The `glMolViewer` object is the main actor for the scene setup and rendering of our molecular visualization. The structure of this object employs the event handling subsystem provided by SpiderGL, which is inspired from the one used by the GLUT library [Kilgard]. Each event coming from the canvas triggers a specific function with a given name and parameters; SpiderGL exploits the JavaScript language feature to give the possibility to dynamically add or remove listeners and redirect events. In this simple example, the only listener is the main object itself.

```
SpiderGLMolViewer.prototype = {
```

```
  load : function(gl) { [...] },
  unload : function(gl) { [...] },

  update : function(gl, dt) { [...] },

  keyDown : function(gl, keyCode, keyString) {[...]},
  keyUp : function(gl, keyCode, keyString) {[...]},
  keyPress : function(gl, keyCode, keyString){[...]},
  mouseDown : function(gl, button, x, y) { [...] },
  mouseUp : function(gl, button, x, y) { [...] },
  mouseMove : function(gl, x, y) { [...] },
  mouseWheel: function(gl, wheelDelta, x, y) { [...] },
  click : function(gl, button, x, y) { [...] },
  dblClick : function(gl, button, x, y) { [...] },

  resize : function(gl, width, height) { [...] },

  draw : function(gl) { [...] },
};
```

Most of the initialization and data loading is done in the `load` function: it is here that the main properties of the rendering are chosen, the input data is loaded and the shaders are compiled.

```
load : function(gl) {
[...]
  this.xform = new SglTransformStack();
  this.camera  = new SglFirstPersonCamera();
  this.camera.lookAt(0.0, 0.0, 1.5, 0.0, 0.0, 0.0, ↩
      sglDegToRad(0.0));
  this.viewMatrix = this.camera.matrix;
  this.trackball  = new SglTrackball();
[...]
  this.prog = new SglProgram(gl, [sglNodeText("↩
      MY_VERTEX_SHADER")],[sglNodeText("↩
      MY_FRAGMENT_SHADER")]);
[...]
  var TextureOptions = {
  generateMipmap : true,
  minFilter : gl.LINEAR_MIPMAP_LINEAR,
  onload : this.ui.requestDraw
  };
  var ColorTexture = new SglTexture2D(gl, "↩
      molecule_color.png", textureOptions);
[...]
  this.meshJS = new SglMeshJS();
  this.meshJS.importOBJ("molecule.obj", true, function(↩
      m, url) { [...]
    this.meshGL_MOL = that.meshJS.toPackedMeshGL(gl, "↩
        triangles", 65000);
  [...] });
[...]
  var pdbtxt = sglLoadFile("mol.pdb");
  this.atomslist = [];
  var res = AtomListFromPDB(this.atomslist, pdbtxt);
[...]
  this.timeOffet = 0.0;  // particle animation offset
  this.stereoEnabled = false; // start with no stereo
  this.particlesEnabled = true; // start with particles
},
```

This monolithic way of managing data is fine for webpages devoted to the visualization of a single, compact dataset. More advanced examples may benefit from the asynchronous loading mechanism which allows efficient use of large datasets, streaming/progressive data or letting user dynamically load remote files.

The `draw` function contains the code for the actual rendering:

```
draw : function(gl) {
  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT | ↩
      gl.STENCIL_BUFFER_BIT);
  gl.viewport(0, 0, w, h);
  this.xform.projection.loadIdentity();
  this.xform.projection.perspective(sglDegToRad(45.0), ↩
      w/h, 0.1, 10.0);
[...]
  var uniforms = {
    u_mvp : this.xform.modelViewProjectionMatrix,
    u_normal_mat : this.xform.viewSpaceNormalMatrix,
    u_dotrasp : this.atomsEnabled,
    u_mousepos : [this.ui.mousePos.x, this.ui.mousePos.↩
        y]
  };
  var samplers = {
    s_texture_c : this.ColorTexture,
    s_texture_b : this.BumpTexture
  };
[...]
  gl.enable(gl.DEPTH_TEST);
  gl.enable(gl.CULL_FACE);
  sglRenderMeshGLPrimitives(this.meshGL_MOL, "triangles↩
      ", this.prog, null, uniforms, samplers);
[...]
},
```

This function may be called *continuously* or *on demand*: when registering the canvas with the `sglRegisterCanvas` function, if the last parameter is 0, then the canvas is only redrawn by explicit commands, otherwise, the parameters represent the desired frame rate. At each "tick" the SpiderGL will call the `update` function and then the `draw`. In both cases, the html rendering engine will then issue a page composition operation whenever it detects changes to the associated WebGL framebuffer.

GLSL shaders are included in the web page as script entities in the HEAD section:

```
<script id="MY_VERTEXSHADER" type="x-shader/x-vertex">
[...]
</script>
```

The resulting code is very schematic and organized in such a way that following the various setup and rendering steps is quite easy. This simple example is an optimal starting point for experimentation.

This development process is straightforward for someone with an experience in graphical programming, while may prove to be difficult for users with a different background, like biology, physics or chemistry. This kind of setup is for sure more difficult to master with respect to setup of other existing platforms, like Jmol which, true to their nature, provide much simpler (but restrictive) access to their scene graph, with specific functions to import data and a series of predefined rendering modes. However, the gain in terms of flexibility and expressive power vastly compensate the initial steeper learning curve. Moreover, the learning of this technology is made easier by the possibility of initially use the higher level structures and functions implemented by SpiderGL to easily setup a basic visualization scheme and then start playing with lower level functions to obtain more complex effects. It is also important to note that most of the available JavaScript utility/UI libraries on the net may be used in conjunction with SpiderGL, adding more ready-made components to assemble a powerful, interactive, webpage.

4 Visual Mapping of Molecular Properties

As an example of the strategy described in the previous section, we will describe how a specific visualization scheme may be implemented using the proposed technology in a very straightforward way.

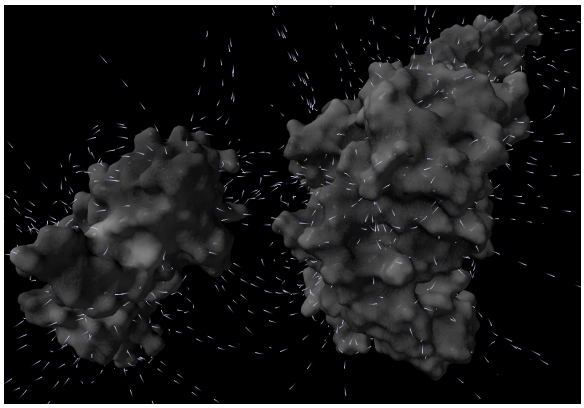

Figure 3: *Interaction between two molecules: the particle flow shows the electrical attraction between the Calmodulin and the MLCK head.*

The aim of this visualization method [Andrei et al. 2010], designed in the framework of the creation of a CG short movie, was to display two specific biochemical properties on the surface of molecules. The two surface properties were the Molecular Lipophilic Potential (MLP) and the Electrostatic Potential (EP). The ability of a molecular surface to establish bonds with water is called Hydrophilicity; its opposite, which is the ability to establish bonds with fat, is called Lipophilicity. The Electrostatic Potential is easier to understand: each atom in a molecule may have a charge, the various charges in the molecule produce an electric field in the surrounding of the molecule. The main idea of this visual mapping has been to exploit perceptual associations between the values to be mapped and visual characterization of real-world objects. Ideally, by using already established perceptual association, the viewer would be able to understand the provided information more naturally, without the use of explicit legends.

For the mapping of the MLP property, it was necessary to choose two opposite surface characterizations, able to convey a sense of affinity to water or to oil. In our real-world experience, a very smooth, hard surface (like porcelain) is completely impervious to water but can be easily coated by oil. The opposite visual feedback is associated to grainy, crumbly, dull surfaces (like clay bricks or biscuits) which can be easily imagined being soaked in water. These considerations led to the association of highly lipophilic areas as white, shiny, smooth material and of highly hydrophilic areas as dark, dull and rough. While the MLP value is obviously only observable on the surface itself, electrical phenomena are associated to the idea of an effect projected in the volume surrounding a charged object, and able to affect other objects (like the high school textbook-favorite amber rod attracting paper bits). Field lines are a common way to describe the effect of the electrical field. EP value is therefore represented by showing small particles, moving along the path defined by field lines, visualizing a higher concentration of particles in areas where the electrical fields is stronger.

A peculiar characteristic of this visual mapping is that it only uses shades of gray to represent the two molecular properties; this choice, which seems restrictive at first glance, is however capable to efficiently convey the two layers of information while leaving the utilization of color space for the description of other biochemical information. This visualization method is perfect to show the capabilities of the proposed strategy, since it involves data coming from BIO tools and rely on a controlled use of shading (bump mapping and specular map for MLP) and rendering effects like particles (moving along the field lines for EP). Moreover, the focus of this visual mapping is not only towards the scientific accuracy, but also towards the visual appeal of the representation.

4.1 From Scientific Data to Rendering

This visual mapping has been designed with the explicit purpose of being used in a CG movie [SCIVIS 2005], produced using the 3D modeling and rendering tool Blender. For this reason, most of the input data, coming from scientific tools, have been heavily processed in order to be converted in a format easily used inside Blender.

The geometry of the molecular surfaces of the depicted proteins has been generated using PyMOL starting from their atomic structure contained in their PDB files. The two properties have been calculated by using scientific tools, starting from atomic structure and reference tables for atomic electrical and lipophilic contributions. The lipophilic potential data, calculated using a dedicated python script (pyMLP) developed by a molecular scientist, is stored as a series of samples in the proximity of the molecule. These samples are then mapped on the molecular 3D surface using interpolation. This mapped value are used to generate color, specular and roughness texture map. The Electrostatic potential, calculated inside another physico-chemical tool (APBS), is basically a volumetric dataset which describes the electrostatic value computed in a regular grid surrounding the molecule. Using this data, it was easy to compute the potential gradient and use it to generate the field lines. The obtained lines were used in the movie rendering to animate a particle system.

In our example, we will start from the processed data, which is somehow in between biochemical data and standard computer graphics data, and then consider the kind of problem and possibilities introduced by the direct use of scientific data. In this conversion from the CG movie to the realtime web environment, it was possible not only to obtain the same look and feel of the rendered movie, as visible in Figure 4, but also to introduce additional elements which are only possible in an interactive context. Beside the usual interactivity which may be attained by the use of simple widgets like a trackball, the ability to configure the rendering pipeline make it possible to change rendering parameters on the fly, mix multiple rendering styles to visualize multiple data layers at the same time and add effects like the direct rendering in anaglyph-stereo. Again, the important point, more than the mentioned effects, is the possibility of overcoming the limits of the predefined rendering that characterize similar systems.

In the next sections we will detail how each component of the render has been implemented in order to obtain the same look and feel presented in the video. For each section we discuss possible alternatives for data source and rendering methods to show how it is possible to directly use biochemical data or create more complex visualizations.

4.2 Geometry

There are many different methods used in molecular biology to visualize the three-dimensionality of a molecule. There is a clear distinction between the representation of the molecular structure and of its surface. The molecular structure is generally displayed atom by atom (using a *Van der Waals spheres*, *sphere+stick* or *licorice* rendering) or as a series of structure elements (*ribbon, rod+arrow*). Conversely, the molecular surface [Connolly 1983], defined as the set of points which are "accessible" to a given solvent (typically water), is a more complex three-dimensional structure and it is generally displayed as a triangulated mesh, or as a series of nurbs patches.

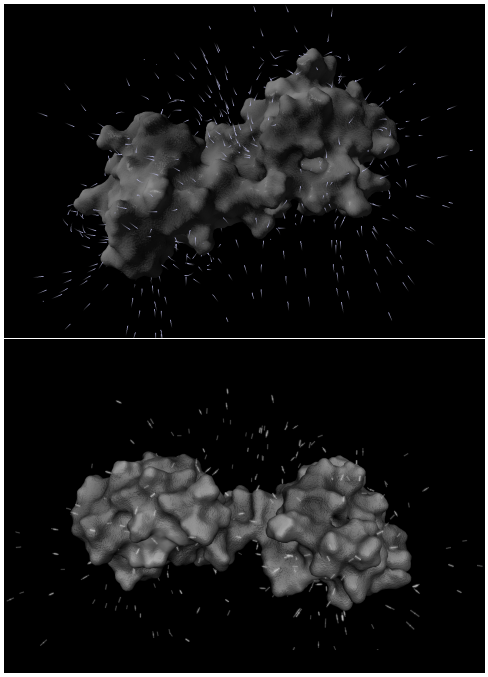

Figure 4: *Comparison of the molecular surface visualization rendered by Blender for the movie (bottom), and rendered using SpiderGL and WebGL (top).*

In this context we are more interested in the rendering of the molecular surface, since the two properties we visualize show their effect in proximity of this surface. Many biochemical tools (like PyMOL, used in this work) are able to compute the geometry of the molecular surface starting from the atomic description of the molecule itself. The result of this process is generally a triangulated mesh, which can be exported, depending on the tool, in different 3D file formats.

For the movie, the used file format was OBJ which is directly readable from SpiderGL: these models were also the starting point for the online visualization. We decided to import precalculated geometries in the scene as they were already available from the pipeline used to create the movie, but also because this is the most sensible option. In theory, it would be possible to compute the molecular surface on the fly starting from the atomic structure of the molecule, but this process would require a non-trivial amount of time and system memory.

As previously stated, to render the structure of the molecule using Van der Waals spheres, it is necessary to know the position, radius and color-coding of each atom. The standard way to represent a molecule structure in biochemical applications is through the use of a PDB file. A PDB file is just an ASCII file which contains (among other molecular-related info) a list of atoms with an associated position. Using JavaScript is quite easy to parse it (as shown in Section 3) and render with SpiderGL a series of colored spheres of appropriate size in the correct position. The atomic representation of the molecule shown in Figure 5 has been generated using this method. A more complex visualization of the molecular structure, like *ribbon*, may also be generated on the fly by starting from the parsed PDB file and a series of pre-defined 3D element templates. As we said before, since there are many molecular databases available online, the PDB file could also be retrieved directly from such a repository.

The availability of alternative representations of a protein structure, makes also possible their combination in a single scene, providing the user the ability to switch between the different representations. Again, this is quite common and nothing new but, since we can configure the rendering pipeline we can, for example, show the superimposition of the molecular surface and the Van der Waals representation by using transparency effects. As shown in Figure 5, it is possible to implement a "fresnel" transparency which depends on the viewing angle, or a more focused "x- ray vision" transparency area which follows the mouse pointer. These kinds of transparency effects are really simple to implement using GLSL shaders and let perceive both representations at the same time, to better understand the relationship between the surface properties and the underlying structure.

4.3 Lipophilic Potential

The visual mapping of lipophilic potential rely on a combination of color, surface roughness and specularity: these three effects are mapped on the molecular surface according to the local lipophilic potential value. For the rendered movie, the potential value has been used to generate the color/specular and bump texture maps inside Blender by baking on the textures a procedural material. To render these effects, we decided to use the same texture maps used in the rendering of the movie and to write a shader which uses simple shading techniques. *Bump Mapping* and *Specular Mapping* are standard shading techniques, but it is possible to apply a fine-control over their appearance by having the full control of the shader setup, which is not generally possible in commercial software for web publishing or in general purpose visualization tools. The result is pleasant and, as visible in the left side of Figure 2, the characterization of the surface is quite effective.

Again, the use of precomputed texture is the fastest way to produce this kind of effect. However, it is also possible to start from the initial data from which those textures have been generated. As in the case of the molecular structure file, the lipophilic information is contained in an ASCII file, which can be parsed using JavaScript and mapped onto the 3D surface as it was done when baking the texture. Once the values are mapped to the surface, a simple shader may be used to produce the same effect of the textures using a procedural approach. Basically, the color of the surface, the intensity of the roughness and the specularity only depends on the lipophilic value: there is nothing which cannot be done in the shader. Having the mapped lipophilic potential makes also possible a more classical rendering style which uses color ramps (right side of Figure 2). This second input method is more generic, since it uses directly the data generated by the biochemical tool, but may be slower (since no precalculation is done and the mapping has to be done at loading time) and less compact (since the lipophilic data may be larger than the textures).

4.4 Electrostatic Potential

Field lines are a widely used method to depict vector fields, especially for electrical and magnetical phenomena. However, the main problem with field lines is *how many* and *which* lines are needed: too few lines do not convey the necessary information and too many will obscure entirely the object of interest. The visualization method used for the movie tried to compute, from the infinite possible field lines, a "meaningful" subset of lines. The aim was to generate a distribution of lines proportional to the surface EP value: more lines would rise in the more electrically active areas, and the total number of lines would be proportional to the global level of potential of the molecule (in absolute value). This operation was done by using a Monte Carlo sampling, weighted with respect to the potential value of the surface in each area. The selected lines were then exported as a sequence of points, forming various poly-

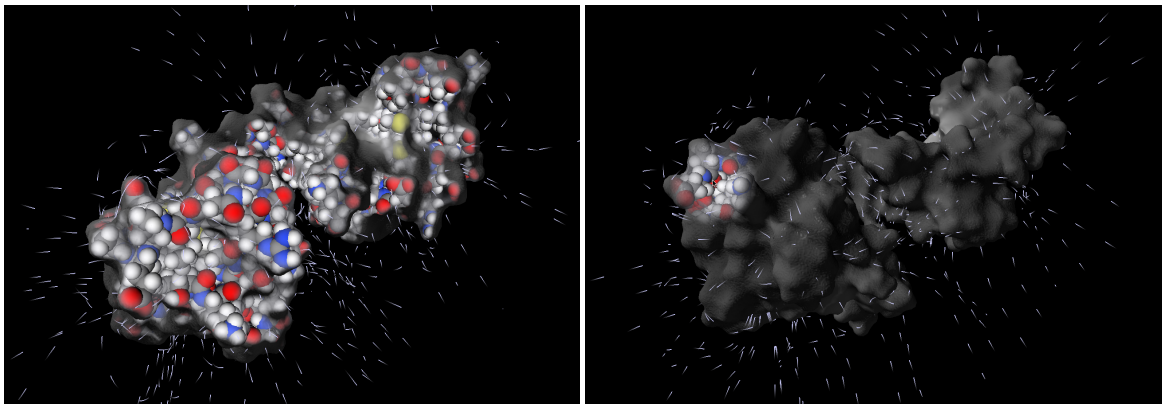

Figure 5: *Showing the superimposition of the Molecular Surface and the underlying Atomic Structure using a transparency based on view angle (left) and a localized transparency area which follows the mouse (right)*

lines. Instead of rendering these entities as solid lines (as visible in Figure 1) each curve was used to drive a particle system. By using moving particles, in fact, it is easier to perceive the flow direction of the field and thanks to their small size and movement, they do not hide the underlying molecular surface. The easiest way to load the lines inside the web implementation was to apply just a small change in the code for line calculation, in order to export the polylines JSON format. It was then possible to parse them using JavaScript. The loaded data may be rendered as a series of solid *line strips*, or used to produce a particle effect similar to the one used in the movie. In this case, since the particles flow on fixed lines, it is not really necessary to create a particle system, but it is possible to visualize the moving particles using a fragment shader which renders only small fragments of the imported polylines according to a periodic function, animated using an offset.

```
uniform float u_timeOffset;
varying float v_texcoord;
void main(void)
{
  const float part_density = 4.0;
  const vec3 part_color = vec3(0.8,0.8,1.0);
  float val = fract((v_texcoord+u_timeOffset)/part_density);
  if (val < 0.7)
    discard;
  else
  {
     val = smoothstep(0.9, 0.7, val);
     gl_FragColor = vec4(part_color * val, val);
  }
}
```

This effect is much more simple and less CPU/GPU demanding than a real particle system, while still effective in conveying the carachteristics of the electrical field surrounding the molecule, the areas of higher electrostatic potential and their polarity. The field particles are also useful to show the electrical interaction between different proteins: in Figure 3 it is shown a calcium-bound Calmodulin approaching an MLCK head, at that distance the two electrostatic field do start an interaction process which will eventually lead to the docking of Calmodulin, and this is shown by the particles flowing from one protein surface to the other. Also in this case it is possible to start directly from the raw physico-chemical data: the volume data of the Electrostatic Potential is saved in ASCII format and can be easily read using JavaScript. With these data, it is possible to compute the potential gradient field and extract the field lines according to the desired parameters. This option would give full control on the lines extraction and make it possible to control the selection parameters on the fly during rendering: given the importance of the line selection, as previously described, this feature may be useful to study the electrical field of the molecule. In any case, being able to load the entire volumetric information may open up new possibilities to visualize the electrostatic field around the molecule. Rendering methods such as ray-casting, interactive slicing and volume splatting are possible on this platform.

5 Future Development

The proposed technology is far from being complete: the WebGL standard is not yet completely finalized and also the SpiderGL wrapping is still an ongoing project. To provide a complete platform for the development of specialized visualization tools for the web platform, some more work will be needed to make this technology accessible to people with not much experience in computer graphics programming. This effort should ideally result in the creation of a reusable library of basic functions which will ease the creation of simple visualization schemes and, at the same time, serve as a code base for more complex results. In perspective, to give a useful instrument to the general public of molecular biology scientist, we will have to work in three different directions:

- a series of *importers* from different file formats which are common to the biology community: more readable formats means more diverse data to play with;
- *utility functions* to manipulate data: because visualization is always a matter of filtering data using standard mathematical/statistical approaches;
- a series of *standard shaders* to be used for rendering: a shader library would save the time required to write simple visualization techniques and give the base for experimentation in creating advanced custom shaders;

An active research problem in the biology community is the calculation of protein motion (i.e. the description of atomic trajectories while transiting from one conformation to another), this kind of online visualization technology would prove quite useful for the evaluation and sharing of new results with the research community. It is however still difficult to display animation of the molecular surface in a way that is biologically accurate but at the same time computationally effective. Animating a structure representation of a molecule (atomic sphere, balls+sticks, ribbon) may be easy enough, since it involves rigid roto-translation of rigid entities. On the contrary, the motion of molecules make the surface undergo major modification and radical change in topology (genus change, merging/dividing parts) thus making it impossible to use animation techniques like skeletal or keyframe. The use of techniques like

metaballs may produce surfaces in realtime, but with very low accuracy from the biological point of view. A better idea could be to exploit the GPU processing power to generate the animated geometry on the fly using, for example, ray-casting methods. An efficient storage and retrieval of such animations is another interesting problem, especially in the context of web-based applications, which present additional constrains of low resources and low bandwidth.

6 Conclusions

We have presented here a technology, based on the WebGL standard, which can be profitably used to build, on the web platform, interactive 3D visualization schemes for the scientific data produced by molecular and cellular biology research. By using the low-level features of WebGL, enriched by the utility functions and higher-level classes provided by the SpiderGL library, it is possible to build web-based visualization prototypes which are not only completely custom, but also use advanced shading and rendering techniques. We have discussed the possibilities offered by this technology, describing the available components and how they are used in the creation of an interactive visualization scheme. Moreover, we have shown how it was possible to use this technology to port a specific visualization method on the web platform, and how it was possible to enrich it with additional visual elements, made available by the use of this technology. This technology is still not complete, since the WebGL standard is not yet completely fixed, and the SpiderGL library still an ongoing project; nevertheless, this combination of libraries and working strategy is a promising instrument to deal with the needs of the molecular and cellular biology research community.

Acknowledgements

This work has been financed from Regione Toscana through the project "Studio Animazione 3D" to Monica Zoppè. This work sprouted from the collaboration between the VC Lab of ISTI-CNR and the SCIVIS Group of IFC-CNR, the authors want to thank all components of both groups for their support.

References

ANDREI, R. M., CALLIERI, M., ZINI, M. F., LONI, T., MARAZITI, G., AND ZOPPÈ, M. 2010. Intuitive visualization of surface properties of proteins. *BMC bioinformatics - in review*.

BAKER, N. A., SEPT, D., JOSEPH, S., HOLST, M. J., AND MCCAMMON, J. A. 2001. Electrostatics of nanosystems: application to microtubules and the ribosome. *Proceedings of the National Academy of Sciences of the USA*, 98, 10037–10041.

BERMAN, H., HENRICK, K., AND NAKAMURA, H. 2003. Announcing the worldwide protein data bank. *Nature Structural Biology*, 10, 980.

BRUNT, P., 2010. GLGE: WebGL for the lazy. http://www.glge.org/.

CONNOLLY, M. L. 1983. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 211, 709–713.

DELANO, W. L., 2002. The pymol molecular graphics system.

DELILLO, B., 2009. WebGLU: A utility library for working with WebGL . http://webglu.sourceforge.org/.

DI BENEDETTO, M., 2010. SpiderGL: 3D Graphics for Next-Generation WWW. http://spidergl.org/.

DON BRUTZMANN, L. D. 2007. *X3D: Extensible 3D Graphics for Web Authors*. Morgan Kaufmann.

GROUP, T. K., 2009. Khronos: Open Standards for Media Authoring and Acceleration . http://http://www.khronos.org.

GROUP, T. K., 2009. WebGL - OpenGL ES 2.0 for the Web. http://www.khronos.org/webgl/.

GUEX, N., AND PEITSCH, M. C. 1997. Swiss-model and the swiss-pdbviewer: an environment for comparative protein modeling. *Electrophoresis*, 18, 2714–2723.

HUMPHREY, W., DALKE, A., AND SCHULTEN, K. 1996. Vmd: visual molecular dynamics. *Journal of Molecular Graphics*, 14, 33–38.

2002. Jmol: an open-source Java viewer for chemical structures in 3D. http://www.jmol.org/.

JOGL Java Binding for the OpenGL API. http://kenai.com/projects/jogl/pages/Home.

KAY, L., 2009. SceneJS . http://www.scenejs.com.

KENDREW, J. C., BODO, G., DINTZIS, H. M., PARRISH, R. G., WYCKOFF, H., AND PHILLIPS, D. C. 1958. A three-dimensional model of the myoglobin molecule obtained by x-ray analysis. *Nature*, 181, 662–666.

KILGARD, M. J. GLUT - The OpenGL Utility Toolkit . http://www.opengl.org/resources/libraries/glut/.

KYTE, J., AND DOOLITTLE, R. F. 1982. A simple method for displaying the hydropathic character of a protein. *Journal of Molecolar Biology*, 157, 105–132.

MCGILL, G., 2010. MolecularMovies.org: a Portal to Cell & molecular Animation. http://www.molecularmovies.com/.

MICROSOFT CORPORATION, 1996. Microsoft activex controls. http://msdn.microsoft.com/en-us/library/aa751968(VS.85).aspx.

OPENGL ARB. GLU OpenGL Utility Library . http://www.opengl.org/documentation/specs/glu/glu1_3.pdf.

PETTERSEN, E. F., GODDARD, T. D., HUANG, C. C., COUCH, G. S., GREENBLATT, D. M., MENG, E. C., AND FERRIN, T. E. 2004. Ucsf chimera–a visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, 25, 1605–1612.

RAGGETT, D. 1994. Extending WWW to support platform independent virtual reality. *Proceedings of INET'94, the Annual Conference of the Internet Society*.

ROCCHIA, W., SRIDHARAN, S., NICHOLLS, A., ALEXOV, E., CHIABRERA, A., AND HONIG, B. 2002. Rapid grid-based construction of the molecular surface and the use of induced surface charge to calculate reaction field energies: applications to the molecular systems and geometric objects. *Journal of Computational Chemistry*, 23, 128–137.

SCIVIS, 2005. Scientific Visualization Unit, IFC CNR. http://www.scivis.ifc.cnr.it/index.php/videos.

TARINI, M., CIGNONI, P., AND MONTANI, C. 2006. Ambient occlusion and edge cueing to enhance real time molecular visualization. *IEEE Transaction on Visualization and Computer Graphics 12*, 6 (sep/oct).