Game On, Science - How Video Game Technology May Help Biologists Tackle Visualization Challenges

Zhihan Lv[1,2], Alex Tek[1,4], Franck Da Silva[1], Charly Empereur-mot[1], Matthieu Chavent[3]*, Marc Baaden[1]*

1 Laboratoire de Biochimie Théorique, CNRS, UPR9080, Univ Paris Diderot Sorbonne Paris Cité, Paris, France, 2 Ocean University of China, QingDao, People's Republic of China, 3 CEA, DAM, DIF, Arpajon, France, 4 Université Pierre et Marie Curie, UPMC-Sorbonne Universités, Paris, France

Abstract

The video games industry develops ever more advanced technologies to improve rendering, image quality, ergonomics and user experience of their creations providing very simple to use tools to design new games. In the molecular sciences, only a small number of experts with specialized know-how are able to design interactive visualization applications, typically static computer programs that cannot easily be modified. Are there lessons to be learned from video games? Could their technology help us explore new molecular graphics ideas and render graphics developments accessible to non-specialists? This approach points to an extension of open computer programs, not only providing access to the source code, but also delivering an easily modifiable and extensible scientific research tool. In this work, we will explore these questions using the Unity3D game engine to develop and prototype a biological network and molecular visualization application for subsequent use in research or education. We have compared several routines to represent spheres and links between them, using either built-in Unity3D features or our own implementation. These developments resulted in a stand-alone viewer capable of displaying molecular structures, surfaces, animated electrostatic field lines and biological networks with powerful, artistic and illustrative rendering methods. We consider this work as a proof of principle demonstrating that the functionalities of classical viewers and more advanced novel features could be implemented in substantially less time and with less development effort. Our prototype is easily modifiable and extensible and may serve others as starting point and platform for their developments. A webserver example, standalone versions for MacOS X, Linux and Windows, source code, screen shots, videos and documentation are available at the address: http://unitymol.sourceforge.net/.

Citation: Lv Z, Tek A, Da Silva F, Empereur-mot C, Chavent M, et al. (2013) Game On, Science - How Video Game Technology May Help Biologists Tackle Visualization Challenges. PLoS ONE 8(3): e57990. doi:10.1371/journal.pone.0057990

Editor: Paul Taylor, University of Edinburgh, United Kingdom

Received October 30, 2012; **Accepted** January 31, 2013; **Published** March 6, 2013

Funding: This work was supported by the French Agency for Research (grants ANR-07-CIS7-003 and ANR-11-MONU-003) and the China Scholarship Council (scholarship 2010633093). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

* E-mail: baaden@smplinux.de (MB); chavent@ibpc.fr (MC)

These authors contributed equally to this work.

Introduction

Biology currently undergoes a rapid expansion, calling for tools to visualize huge and complex systems, such as macromolecular structures, -omics networks or even organs and organisms [1–3]. It is a particular challenge for academic researchers to develop software solutions meeting these demands. At the same time, the video game and movie industries face similar needs in terms of complexity and efficiency to release products (i.e. respectively games and movies) for an increasingly wider audience. These industries benefit from more and more advanced tools enabling a quick development cycle, making the most of latest hardware and software. Can the scientific community use these tools to overcome complexity and efficiency issues related to software development by using solutions already provided by the entertainment industry? During recent years, this question was partially answered with respect to the movie industry as several research groups around the world have begun to use dedicated tools such as Maya (http://usa.autodesk.com/maya/), Cinema 4D (http://www.maxon.net/products/cinema-4d-studio.html) or Blender (http://www.blender.org/). Scientific projects associated to these programs include ePMV [4], Molecular Maya [5] and BioBlender [6]. Using such tools for animation is particularly beneficial for teaching as well as for communicating with a broad public, even leading to new discoveries [7] and valuable insight by combining experimental data and modelling [8]. Unlike the emerging spread of tools from the movie industry, the use of software to develop games remains largely unexplored in molecular graphics and modelling, despite its enormous potential.

For many years, game developers have gathered routines and frameworks that can be reused for a wide range of games. These code blocks have been aggregated to provide toolkits, called game engines, embedding all the components required to create a game in one package [9]. The core functionality typically includes a rendering engine for 2D or 3D graphics, a physics engine, sound, scripting, animation, artificial intelligence, networking, memory management, multi-threading, etc.… There are many game engines that are designed to generate executables for video game consoles, personal computers and even mobile devices: Unity3D (http://unity3d.com/), Unreal Engine (http://www.unrealengine.com/), CryEngine (http://mycryengine.com/), Blender Game Engine (http://www.blender.org/education-help/tutorials/game-engine/). These engines are free for non-commercial use and very appealing for developing scientific applications.

Here, we focus on molecular and network visualization using surfaces, spheres and links between them. With this test case we assess whether by using a game engine the functionality of classical viewers could be reproduced and extended in substantially less time and with less development effort than with common tools. Current needs include the ability to display hundreds of thousands of distinguishable elements in interactive time, using classical representation schemes. To provide a realistic scenario, inspired by current research, we have implemented our own visualization algorithm based on latest hardware capabilities such as programmable Graphics Processing Units (GPUs) [10]. The aim was to identify eventual restrictions imposed by the game engine programming environment. A user-friendly interface is required to facilitate navigation and interaction with virtual objects. In order to touch a wide audience, multiple platforms should be supported without the need to develop specific ports – ideally computers on Windows, MacOS and Linux operating systems, web pages, handheld devices such as smartphones and tablets. Most of the available game engines provide such features as well, but at different levels of development easiness.

We have chosen to test the Unity3D game engine (http://unity3d.com/) for its ability to deploy multi-platform applications with minimal programming effort. Furthermore, Unity3D provides an easy-to-use interface to develop 3D graphics applications using JavaScript, Boo – a python derived language - or C# code. Unity3D includes advanced functionalities to fully use the capacities of recent programmable graphics cards. This is possible employing optimized Cg functions [11], dedicated for Nvidia graphics cards, or GLSL (OpenGL Shading Language) code, a more universal language, which can be used on all recent graphics cards as well as on mobile devices [12]. The developer community is very active and helpful in providing example code and expert answers when technical problems are encountered.

In this article, we implement and compare several approaches for ball-and-stick representations using built-in Unity3D features. We also implemented the HyperBalls representation [13] and molecular surfaces. The advantages and limitations of each approach are discussed. Typical molecular properties such as the shape of molecules (surface) or their electrostatic potential can be visualized with original representation schemes. To demonstrate the flexibility of the development tool, we programmed a biological network viewer with interactive features. Using Lit Sphere shading [14], it is possible to achieve an artistic and illustrative rendering for any type of these representations. The outcome is a usable multi-platform stand-alone viewer that produces unique publication-quality figures and may serve as prototype for other developments.

Methods

UnityMol can be deployed in two versions: a stand-alone application and a web applet running on top of the Unity3D web-plugin. We have tested several approaches to achieve the most efficient and appealing visualization of spheres, links and surfaces in order to balance graphical quality and display efficiency. We created a user interface to load and manipulate molecular structures, field lines, surface objects or biological network topologies.

Accessing Data Files

For structural data we use the Protein Data Bank [15] format. Coordinate files can be imported from local storage or can be downloaded directly from the PDB server by entering their molecule ID. Due to security restrictions in the Unity3D web plugin, files have to be loaded through an intermediate server rather than from the local disk. After fetching files from the PDB or the server, they can be used by the web application. This restriction does not apply to the standalone viewer. PDB files are parsed to extract atom information such as atom type, residue name, residue id and coordinates. The molecule topology is built by using typical interatomic distances to detect bonds. Surfaces are generated either externally as meshes in the Wavefront OBJ format (these files contain vertex and face information used to reconstruct a 3D mesh), or internally using a density grid isosurface approach. Electrostatic potential grids computed by APBS [16] (or other tools) and stored in OpenDX format can be read for iso-surface visualization or be pre-treated using the BioBlender pipeline to generate Json files representing field lines. These files describe point coordinates for each field line to be animated. Network topologies are represented using extensible Graph Markup and Modelling Language (XGMML) format that can be obtained from Cytoscape (http://www.cs.rpi.edu/research/groups/pb/punin/public_html/XGMML/) [17]. XGMML files are parsed to extract id, name, position, size, colour and topology of nodes.

Graphical Methods Used

We have tested graphical methods available in Unity3D as well as our HyperBalls representation based on ray casting [13]. The purpose of implementing HyperBalls was to evaluate how difficult it is to port previously developed code to Unity3D and whether this approach facilitates and speeds up the creation of a molecular graphics application. By implementing a molecular surface generation routine based on the classical marching cubes algorithm, we provide an additional proof of the generality of the game-engine approach for visualization applications.

Built-in graphical primitives. Unity3D provides an optimized set of graphical primitives for rendering. We use triangulated spheres, triangulated cubes and lines. Spheres and cubes display atoms (for molecular structures) or nodes (for networks); links between them are rendered using lines (see **Fig. 1-B,D**).

Optimized rendering with a point-sprite particle system. Tessellated graphical primitives require a large number of triangles to represent smooth spheres and are not appropriate to represent molecules within a game engine. Instead, a single square (composed of two triangles only) always oriented perpendicular to the screen plane can be used to represent a single atom. An image of the atom sphere called sprite is pasted onto the square. This point-sprite method is very efficient to display many similar images and can be extended to a so-called particle system (see **Fig. S1 in File S1**). In games, this visual method is commonly used to depict effects involving many small particles such as fire, smoke, clouds, snow, dust, etc. … For UnityMol, we have adapted Unity3D's built-in particle system routine to paste spheres representing atoms on each square (see **Fig. 1-A**). In Unity3D, a single particle system is limited to 16 000 particles. For systems with more components, we implemented a mechanism to create and manage several particle systems transparently. This domain decomposition may lead to minor graphical artefacts as illustrated in **Fig. S2 in File S1**.

Using GPU shaders with the hyperBalls approach. We have developed a molecular representation called HyperBalls [13]. This representation takes advantage of the programmability of graphics cards using high quality ray casting with good display performance. The representation is composed of spheres depicting atoms linked by hyperboloid primitives rather than simple cylinders. Using such hyperboloids, the representation can be

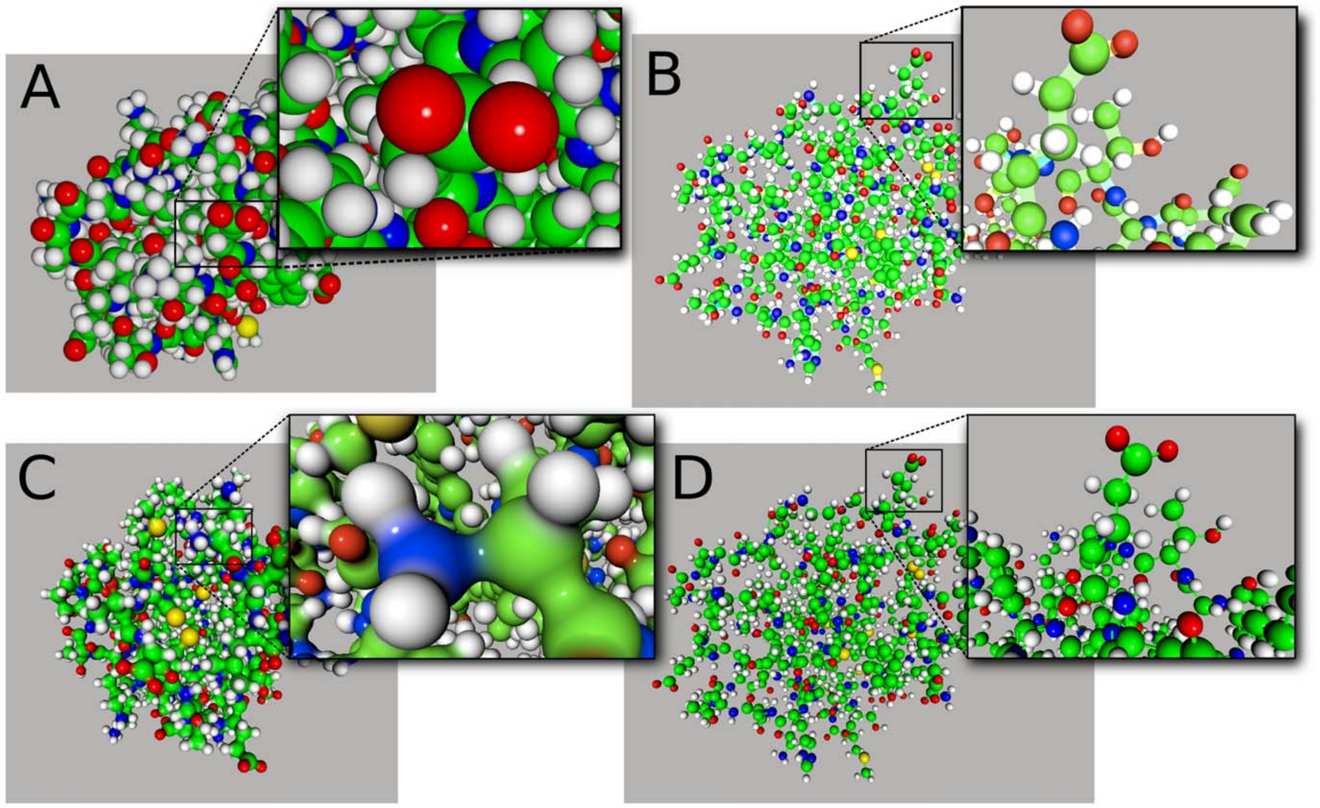


Figure 1. Different molecular representations implemented in UnityMol. A- Particle system depicting atom spheres. B- Unity3D triangulated spheres for atoms and lines for bonds. C- *HyperBalls* spheres and bonds. D- *HyperBalls* spheres for atoms and Unity3D cubes for bonds.
doi:10.1371/journal.pone.0057990.g001

adapted to illustrate continuous bond evolution, passing from a one- to a two-sheeted hyperboloid (see **Fig. 1-C**).

The Unity3D framework enabled us to implement such a graphical method by integrating GPU code – so called shaders – to be run on the graphics card (as Cg or GLSL instructions). We implemented HyperBalls using built-in Cg functions. To create the representation, it is necessary to generate envelopes using cubes and parallelepipeds to define atom and bond primitives, respectively. Then, we represent spheres and hyperboloids within these envelopes using ray casting.

Starting from this previously published implementation, we were able to quickly add improvements to it, such as the lit spheres shading described below.

Visualization of electrostatic potential. Electrostatic potential visualization was implemented in UnityMol either as isosurfaces or as animated particles following field lines. The latter representation is similar to the rendering described in a recent publication by Andrei *et al* [18]. The lines are computed from particle advection inside a gradient vector field derived from an electrostatic potential grid computed by APBS [16]. Currently we use pre-computed field lines from BioBlender [18] stored in a text file. How to generate such files (using APBS or other software) is briefly described in **Text S1 in File S1**. We consider implementing the whole process in a future version of UnityMol, including interactive line selection filters. For now, only some representative lines are selected according to the electrostatic potential value on the molecule surface.

Once the line coordinates are loaded into memory, we reconstruct field lines using the line renderer of Unity3D. A dash-line effect depicting moving particles with a trail is achieved using a shader that animates a sinusoidal function along the field line according to a timer. The user can change the colour, the width, the length and the speed of the particles via the GUI (see discussion and examples of such lines in the results section).

Visualization of molecular surfaces. Molecular surfaces can be imported in Wavefront OBJ format from various molecular modelling programs such as MSMS [19], PyMol [20], Chimera [21], VMD [22] or PMV [23]. The 3D triangulated mesh is easily reconstructed using Unity3D routines and positioned to fit the atomic representation if one is loaded. Different methods can be used to render the surface using shader programs. By default, a simple lighting and a uniform colour are used.

Alternatively, UnityMol is able to generate a Gaussian molecular surface. An atomic density grid is computed by adding Gaussian distributions originating at the atom positions as discussed in [24]. This technique has subsequently been used in the Yasara software [25] and more recently also in the VMD program [26]. Then, the Marching Cubes algorithm [27] is applied to the density grid in order to generate an isosurface. The user can control the coarseness of the surface by changing the surface threshold. While the process is fairly fast for molecules up to 20 000 atoms, more optimizations are to follow for larger systems.

All types of surfaces can be cut according to fixed or moving planes. The interior of the molecule is filled by a uniform colour to enhance the user's impression of cutting a solid object. The implementation uses a shader hiding the pixels of the mesh located above the plane defined by the user. The colour, orientation and position of the cut plane can be changed interactively (see examples in the results section).

For the moment, rendering of molecular surfaces is achieved by triangulation. The quality of such a surface is still limited because of memory restrictions. To improve this issue, it may be possible to build on recent works using the full capacity of modern graphical hardware to depict molecular surfaces combining a high quality of visualization and very good display performances [28–30]. It should be feasible to include such approaches in UnityMol in the near future, as the HyperBalls representation uses similar technology.

Lit sphere shading. On a virtual 3D object, the orientation of one pixel relative to the camera is determined by its normal. All the visible normals for one viewport can be retrieved on a theoretical half-sphere located at the centre of the screen. Thus a lighted sphere, or lit sphere, depicted in 2D contains all the information required to construct a lighting model. Sloan *et al.* [14] used this principle to light 3D meshes by mapping pixel normals with a 2D texture representing a sphere. In UnityMol, we implemented this approach for triangulated meshes and HyperBalls using shader programs to compute the normal for a pixel and retrieve the colour value at the corresponding coordinates on the texture. This value can be used as is or blended with other colours or textures to combine different rendering effects.

A few lit spheres are included as UnityMol program resources. The user can choose a texture for surfaces and HyperBalls representations using the GUI. In the standalone version, it is possible to import custom textures directly from the hard-drive.

Hardware and Software Implementation

Development, benchmarking and testing were carried out on a Mac Pro computer with a quad-core Intel Xeon 3 GHz processor, 8 GB of memory, an NVIDIA GTX 285 graphics card, running 64bit Mac OS X 10.6. Results presented in this article are mostly based on the 3.5 version of Unity3D. All source code is implemented using C# and Cg languages built into Unity3D and is available on the sourceforge project website http://unitymol.sourceforge.net.

Results

Performance of Different Graphical Methods

In this section, we evaluate several Unity3D features on a simple test case and go on to determine the efficiency of the most promising features on a wider range of molecular structures. We then compare these features to our HyperBalls implementation.

Comparison for a simple test case: ferrocytochrome C. We used the molecule ferrocytochrome C [31], PDB ID: 1KX2, as a test case to compare the rendering approaches implemented in Unity3D. This protein is composed of 1 249 atoms and 1 113 bonds, a common size in the PDB.

We evaluated four methods to render atoms (see **Fig. 1**): a particle system, triangulated spheres, triangulated cubes and HyperBalls spheres. We also tested three methods to display bonds: cubes, lines and HyperBalls hyperboloids. The use of cubes is not of interest for representing atoms but it provides reference data for benchmarks allowing us to compare against the creation of simple graphical primitives with Unity3D. **Fig. 2-A** provides a benchmark showing that 3D point-sprites (particles) are the most efficient method to represent spheres in Unity3D. Triangulated cubes are eight times slower to render, closely followed by our implementation of spheres using HyperBalls and Unity's built-in spheres. Rendering bonds in addition to atoms does not change this ranking, but we observe a marked decrease of display performance for the particle system, the display refresh rate being divided by more than a factor three. The impact is less important for the other methods, about a factor 1/3, using cubes for links. This value is divided by more than ten for the particle system when using hyperboloids or lines to render bonds. For the HyperBalls and Unity spheres representations, adding lines or hyperboloids divides the display rate by three. Concerning the particle system, these results can be explained by its high intrinsic efficiency to render spheres. The combination with other graphical methods puts a limit to this efficiency. In the other cases, the decrease in performance can be explained by the fact that the number of graphical primitives is nearly doubled when bonds are added.

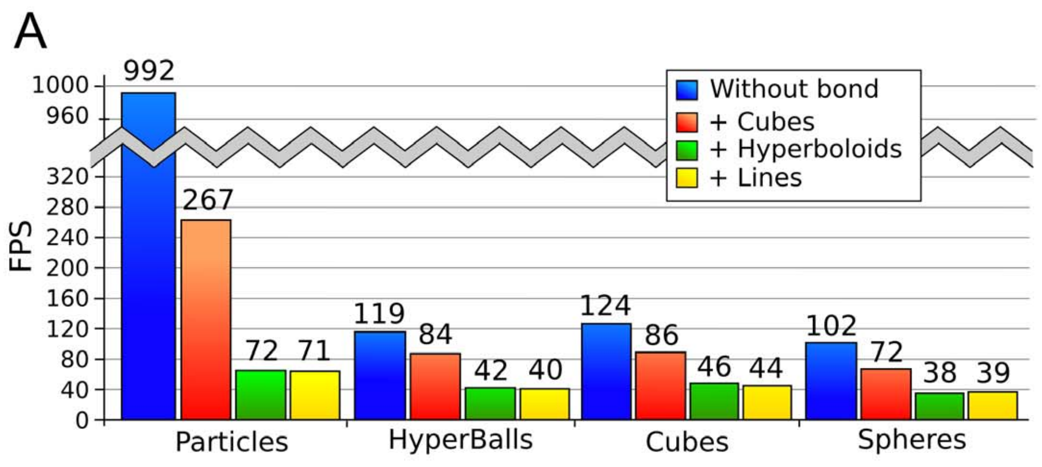
For entirely fluid and interactive display manipulation, it is mandatory to achieve a rendering rate above 30 Frames Per Second (FPS). **Fig. 2-A** shows that the display performance with added bonds is acceptable for all primitives. Depending on the chosen method, the number of triangles and vertices for the graphical primitives may change considerably, directly affecting the display efficiency. The number of function calls to draw these elements is another important bottleneck with less calls leading to better efficiency (see **Table S1 in File S1**). Globally, these results depend on the size of the rendering window, and in particular concerning the HyperBalls representation, intrinsic graphics hardware performance is another key element that governs rendering performance.

Testing rendering efficiency limits and interactivity. We selected the particle system, Unity3D sphere and HyperBalls sphere methodologies to evaluate their respective efficiency limits using a set of molecules with sizes ranging from about 1 200 to 100 000 atoms (see **Table 1**). We focus on spherical atom representations of large macromolecular structures with the aim to render the maximum number of atoms in order to evaluate the brute performance of our own and of Unity3D built-in implementations. We assess the display rate for static molecules (see **Fig. 2-B**). As expected, the performance of the particle system exceeds the other two methodologies. Even for molecules as large as ATP synthase (PDB-id 3OAA), comprising 100 000 atoms, the frame rate remains high. The results for HyperBalls spheres are acceptable for molecules up to a few thousand atoms. Beyond 10 000 atoms, the number of frames per second (FPS) is near ten, the lowest rate at which rendering can be considered useable. We also analysed the impact of user interaction on display performance as described in supplementary material (see **Fig. S3 and Text S2 in File S1**). This lead us to improve the reactivity of UnityMol by implementing an on the fly optimization as described in the following paragraph.

On the fly performance optimization using a level of details approach. To improve the overall reactivity of Unity3D, in particular for slower rendering modes such as HyperBalls spheres and bonds, we have implemented a so-called Level of Details (LoD) method. This functionality automatically switches from a specific representation that is assumed to render slowly to the fast particle system representation while the user is moving the molecule. This option is called "LoD Mode" in the UnityMol Graphical User Interface. LoD is commonly used in other generic data visualization packages such as Paraview (http://www.paraview.org/) to manipulate systems with many particles interactively.

Molecular Features Visualization and Artistic Rendering

Electrostatic field lines visualization. Recently, Andrei *et al.* have developed an animated representation to depict electrostatic field lines using the Blender program [18]. This visualization is particularly appealing and useful to represent the pathways that a charged molecule may follow guided by otherwise



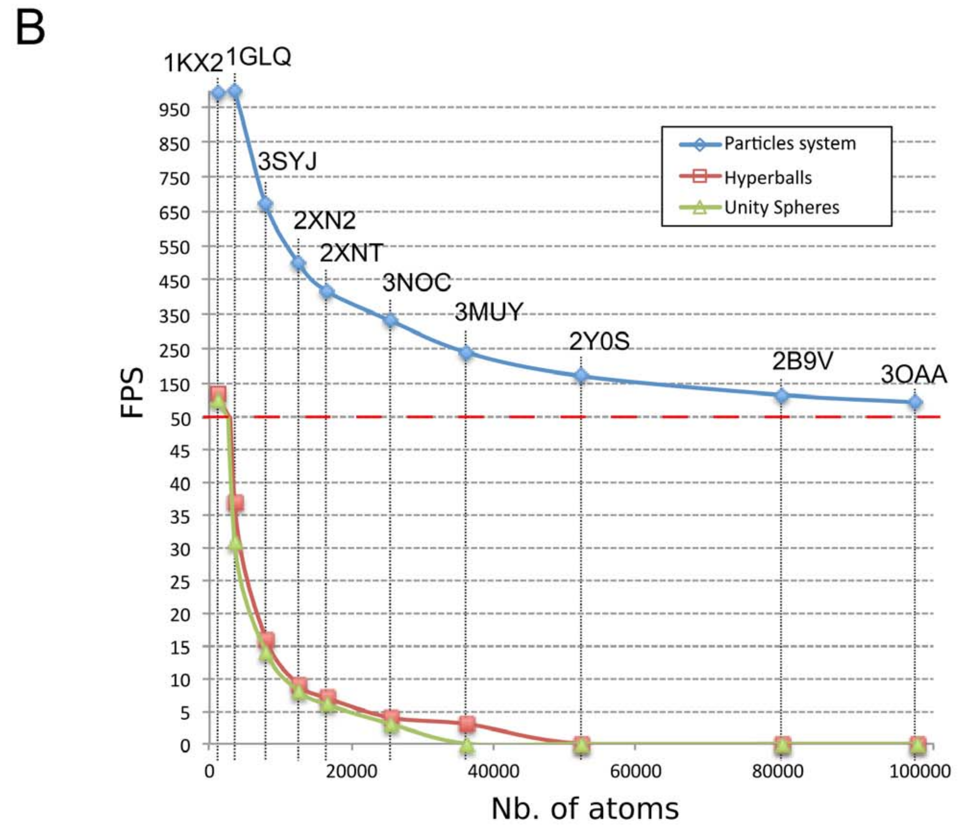


Figure 2. Benchmark of graphical methods. A – Rendering efficiency for ferrocytochrome C (PDB-id 1KX2) shown as atoms and bonds measured in Frames Per Second (FPS) for a 1344×1008 viewport. B - FPS values for molecules of increasing size rendered by a particle system, HyperBalls or Unity spheres. The red dashed line divides the graph in two parts: above this line the FPS step is 50, below this line the FPS step is 5.
doi:10.1371/journal.pone.0057990.g002

invisible electrostatic interactions. Unfortunately, the previous implementation is specific to the Blender package, which has a steep learning curve. Furthermore, it can only be run in an Internet browser window. We have reused this type of animation in the UnityMol viewer so that field lines can be visualized in the standalone version as well as in Internet browsers (see **Fig. 3**). Interactive features enable the user to change the graphical aspect of the field line animation.

In addition to dynamic field lines, classical static isosurface representations of electrostatic potentials can be rendered. The perception of the isosurface shapes may be enhanced using artistic rendering described below (see **Fig. 3E**).

Molecular surface cut planes. Molecular surface rendering is a fairly standard feature in molecular viewers. The addition of cut planes enables the user to easily visualize hidden elements in the interior of a molecule. This feature is particularly useful for proteins with internal cavities or channels (see **Fig. 3-B,C**) and when combined with the electrostatic field lines visualization. Very few molecular viewers implement this feature, which has been rendered particularly efficient and convenient to use in UnityMol.

Table 1. Set of molecules used to evaluate the rendering performance of UnityMol.

Molecule Name	PDB ID	Nb. of atoms	Reference
Ferrocytochrome C	1KX2	1 249	Bartalesi, et al. [31]
Glutathione S-transferase	1GLQ	3 516	Garcia-Saez, et al. [47]
Adhesin	3SYJ	7 968	Meng, et al. [48]
Alpha-galactosidase	2XN2	12 627	Fredslund, et al. [49]
Acetylcholine binding protein	2XNT	16 550	Akdemir, et al. [50]
DARPin bound to AcrB	3NOC	25 558	Monroe, et al. [51]
Beta-galactosidase	3MUY	36 360	Dugdale, et al. [52]
RNA polymerase	2Y0S	52 472	Wojtas, et al. [53]
Hydrolase	2B9V	80 710	Barends, et al. [54]
ATP synthase	3OAA	99 573	Cingolani and Duncan [55]

doi:10.1371/journal.pone.0057990.t001

Artistic rendering by lit sphere shading. Lit sphere shading [14] is a rendering technique inspired from artistic methods to depict light and shadows on a three-dimensional object drawn on a two-dimensional surface. Based on the fact that a 2D depiction of a sphere encodes all the possible light values for a 3D object, we can use a 2D texture representing a sphere and its lighting, a so-called lit sphere, as a basis to light any 3D object. This process ensures a shading consistency and transfers all the artistic value put into the lit sphere onto the 3D object. For example, near ***photorealistic*** rendering is possible using spheres extracted from a photograph of a material such as metal, stone, glass etc… (see **Fig. 4**). As shown on **Fig. 4-C**, using a drawn sphere as reference texture produces an ***illustrative*** rendering close to visualizations generated by Weber [32] with a more demanding technique restricted to secondary structures. Bruckner et al [33] applied this technique to create style transfer functions that map different styles to different parts of a 3D object in a single rendering. This approach could be very useful to generate intuitive visualizations of molecular properties such as hydrophobic patches or to highlight specific binding sites within a molecule.

Lit sphere shading is a non-demanding method to render complex lighting effects in real time, using simple square textures. No performance loss has been observed while applying lit spheres onto a mesh or HyperBalls. As the light orientation is encoded in the lit sphere, the lighting effect is view-dependent which is desirable to achieve a consistent rendering when rotating the camera. An intrinsic inconvenience may be encountered when adding external light sources originating from a different direction than the one in the lit sphere.

Game and Interaction Oriented Features

As a game engine, Unity3D provides pre-configured built-in features such as visual effects, a physics engine and access to peripherals (such as joypads). These functionalities can be reused for molecular visualization and interaction.

Visual effects. There are two kinds of special effects in Unity: image based methods working in 2D screen space and methods working in 3D space. Using screen space effects, it is possible to deform an image, add black lines to outline contours or add glow and blur effects simulating persistence of vision (see **Fig. 5-C,D**). As an example of 3D space effects, we have added ambient occlusion to highlight protein shape (only available with triangulated primitives). Access to these advanced effects requires a Unity3D professional license for development.

A virtual environment may be created by adding a so-called Skybox surrounding the molecule and decorating it with landscape images (see **Fig. 5-A,E**).

Adding physical behaviour with a physics engine. In Unity3D, a physics engine is used to model physical and mechanical behaviour of rigid 3D objects such as collision, dynamics, elastic joints, etc. … To illustrate the potential of such an engine, we have implemented an interaction with the molecule based on a very simple spring network. The user can grab atoms to deform the molecule while the structure is kept coherent ("Interactive Mode" in the UnityMol GUI; see **Movie S1 described in File S1**). A visual feedback is provided by changing atom colours from white to black to represent the potential energy of each part of the elastic system.

Improving the user interaction experience through peripherals and GUI. Unity3D provides routines to use game peripherals such as joypads and joysticks. These game controllers can assist the user in navigating the scene as illustrated in **Movie S1 described in File S1**. UnityMol supports the joystick for control functions such as navigation *via* the integrated manipulator, selection of atoms and bond visualization modes, globally changing atomic radii and more. In the future, we plan to extend these functionalities using the Nintendo Wiimote, the Leap Motion and the Microsoft Kinect devices.

In Unity3D, it is straightforward to create a GUI without using a dedicated and potentially complex toolkit. We used built-in mouse picking functions to highlight selected atoms and access information (such as atom type, residue, etc.) or calculate distances between atoms. The UnityMol GUI, main program features and general usage of the software are described in **Fig. S4 in File S1**.

Reusing Components for Speedy Development of a Systems Biology Application

A particular benefit of a game engine is its modularity and the reusability of the generated routines and scripts. The use of an integrated intuitive editor facilitates the development of applications without writing numerous lines of code. It is possible to position 3D objects in the scene and apply scripts and graphical functions directly via the Unity editor. To illustrate this feature, we have adapted our molecular viewer to represent generic biological networks in 3D. To implement this network viewer (embedded in the demo), we have adapted the file parser to read XGMML (extensible Graph Markup and Modelling Language) format obtained as output of the CytoScape program [17]. The graphical primitives to represent atoms were used to depict network nodes

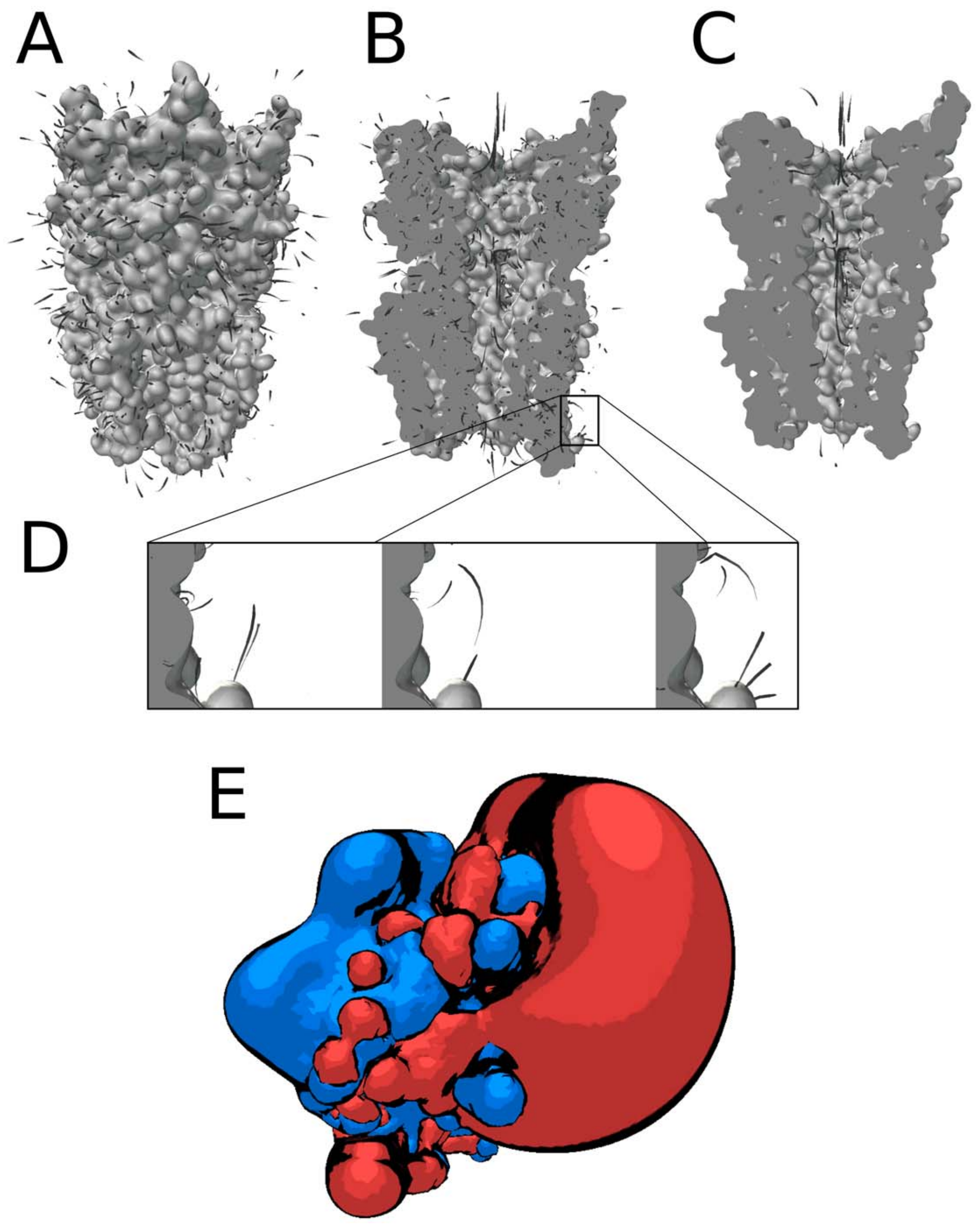


Figure 3. Visualization of animated electrostatic field lines and static iso-surfaces. A – Depiction of the electrostatic potential of the GLIC ion channel (PDB-id 3EAM) by animated trails moving along the field lines. B – Use of a cut plane to reveal the buried channel. C - The built-in Unity3D editor was used to remove some field lines and focus on the ones lining the channel pore. D – Time series showing 3 snapshots of a moving trail. E – Electrostatic potential of ferrocytochrome C shown as iso-surfaces (red : contoured at -2, blue : contoured at $+2$) with a toon shading to enhance the volume perception.
doi:10.1371/journal.pone.0057990.g003

while primitives displaying interatomic bonds are reused to model network edges. Colours and labels were adjusted to mimic conventions from Cytoscape (see **Fig. 6**). We vary the size of the nodes as a function of the radius provided by Cytoscape (depending on connectivity). We noticed that the default flat network view may create graphical artefacts as bigger nodes may hide adjacent smaller ones. To reduce this effect, we have implemented a function distributing nodes along the z-axis according to their radius (see **Fig. 6-B**). Hence, bigger nodes are positioned in the background or in the foreground (using the depth factor scroll bar in the GUI) to improve the visibility of the remaining nodes. We have reused the picking functionality to obtain specific network information when the user clicks on a node. The interactive mode presented previously can be used to manipulate the nodes and position them in a user-defined location (for convenient manipulation, choose a drag value of $\sim$5 and a spring value of $\sim$0).

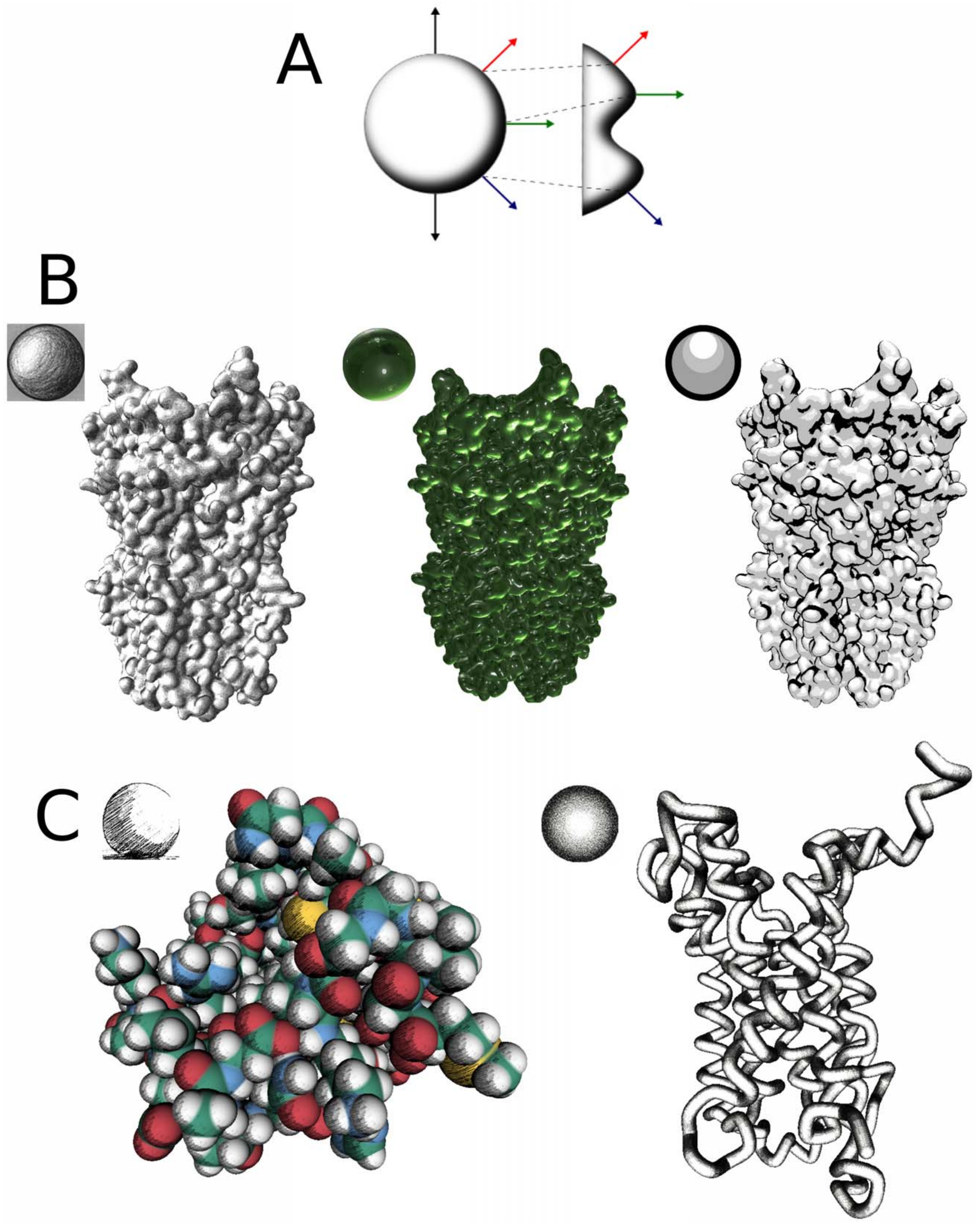


Figure 4. Lit spheres concept. A - All possible lighting and shading components of an arbitrary object (right) can be represented on a 2D image of a sphere (left). Colored arrows depict locations of equivalent surface orientation. B - Different lit-spheres applied to a molecular surface creating different effects : hand drawing (left), green glass (center), cartoon shading (right). C – Illustrative rendering using lit spheres and Hyperballs. Van der Waals (left) and backbone (right) representation of the BLT2 GPCR receptor with hand drawn lit spheres.
doi:10.1371/journal.pone.0057990.g004

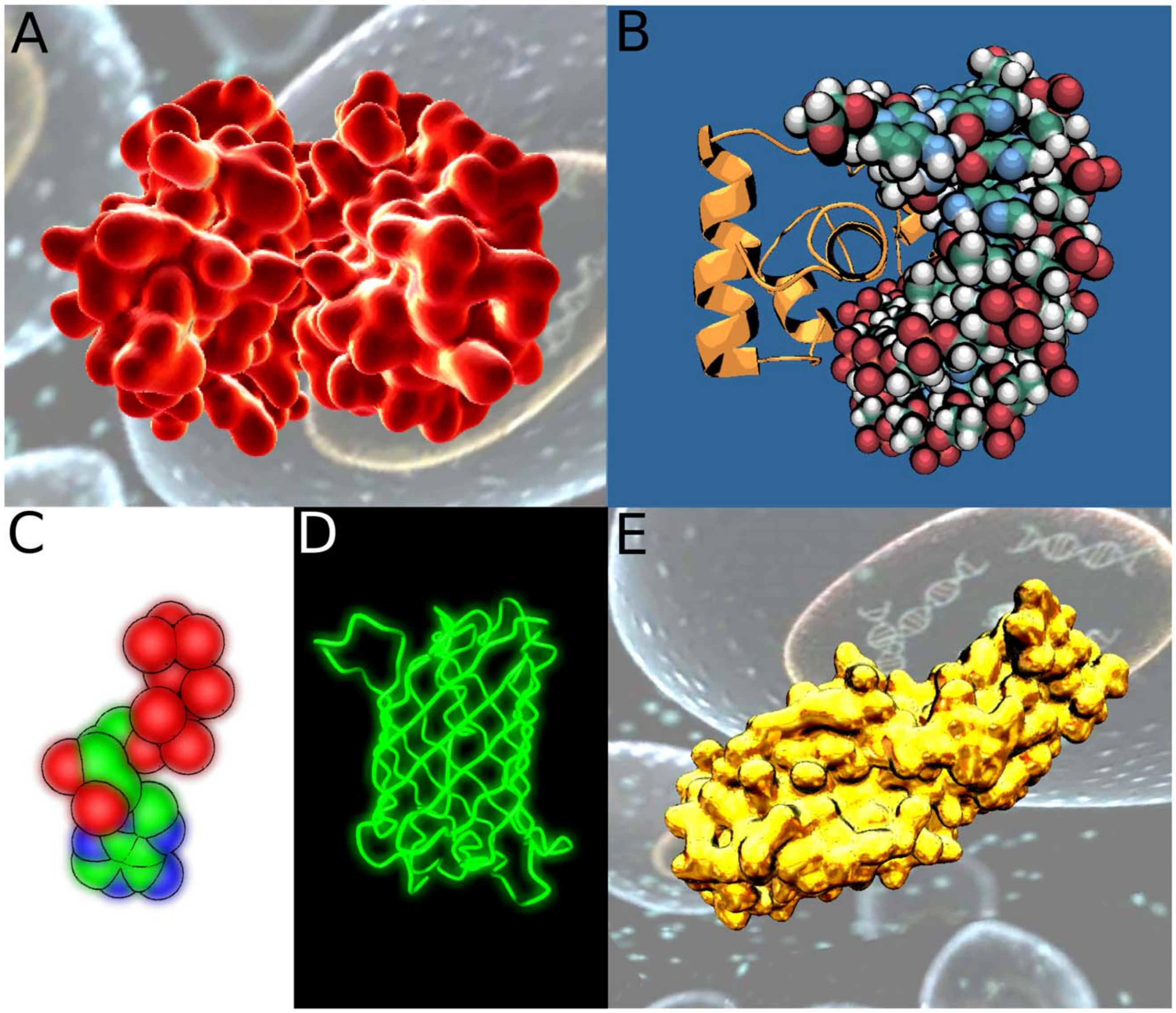


Figure 5. Several visual effects available in Unity. A – Human hemoglobin (PDB id: 1HHO) rendered with a stylized blood effect. B – Binding of a DNA molecule on the DNA-binding domain of Myb (PDB id: 1MSE). The secondary structure object was imported from PyMol. C - Contour outlining combined with glow. D – Green fluorescent protein shown as glowing carbon alpha spline (PDB id: 1KYS). E - Artistic molecule rendering : leucotoxin S component from *Staphylococcus Aureus* (PDB id: 1T5R) with a gold material. A,E – Background creation using a skybox.
doi:10.1371/journal.pone.0057990.g005

Discussion

UnityMol a Graphical Application to Gather Molecular, Network and Electrical Field Lines Visualization

We have gathered recently developed graphical tools in an easy to use distributable software that provides a unique combination of functionalities absent from other publicly available viewers. We have implemented the recently developed HyperBalls representation [13]. In comparison to the original HyperBalls program, it is now possible to dynamically adapt the visual aspect to the user's liking by continuously adjusting radius and shrink factor parameters via the graphical user interface. Furthermore, animation and artistic rendering enabled us to enhance classical molecular representations such as surfaces and electrostatic field lines. Additional visual effects allow the user to intensify a scene and create original images.

We compared our approach with common molecular viewers such as VMD and PyMol (see Table 2). These programs have better performances than our current version of UnityMol using GPU representations. Currently, the UnityMol Ball & Stick representation can interactively render molecules with few thousands of atoms. Hence, this particular representation will be more useful to create 3D graphical scenes for educational demonstration purposes as, in this case, the focus is often on molecules with a relatively small size. To visualize significantly larger systems it is recommended to use particle system spheres and switch on Hyperballs visualizations only when the user wants to obtain a very high quality figure. We have implemented the Level of Details method (see On the fly performance optimization part) precisely for this purpose. We are working to improve this part. The new version of the Unity3D game engine released a few weeks ago has already improved the display performances (see Table 2). We hope that future versions of Unity3D driven by the

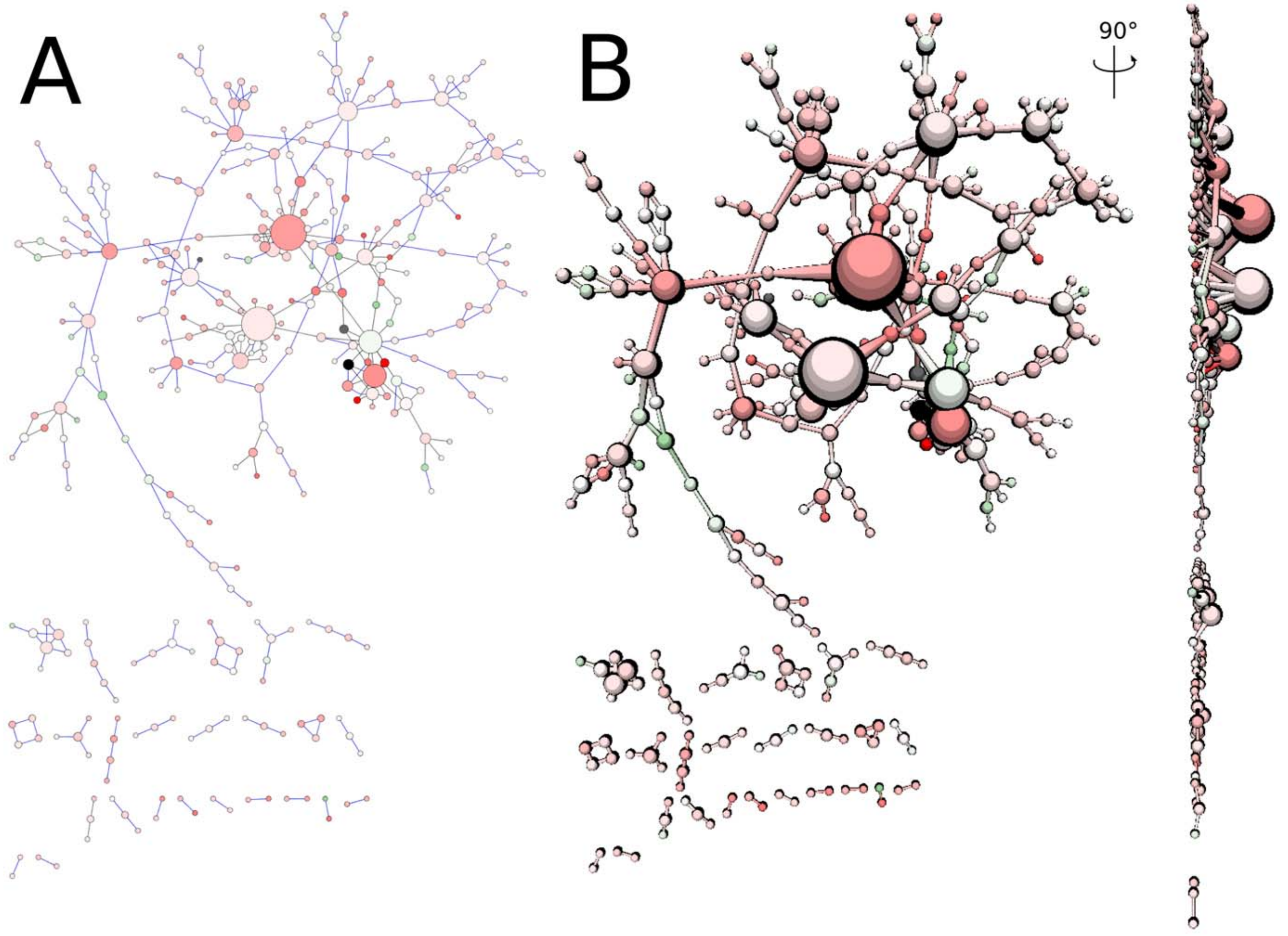


Figure 6. Biological network visualization. A – Original cytoscape visualization used as test case. B - UnityMol visualization with improved visibility by adding depth and contours using a lit sphere.
doi:10.1371/journal.pone.0057990.g006

increasing need of video games for GPU power – to display new visual effects and accelerate interactivity – will help to further improve the visual performances of UnityMol. It should be emphasized that we present here a first version of our approach and we will continue work to develop more efficient algorithms – especially representations using the HyperBalls approach. We hope that our program may interest the community of computer scientists and game designers who may provide valuable help to overcome this type of limitations.

In addition to molecules, we have implemented a 3D visualization of biological networks from Cytoscape [17] where the depth can be adapted to avoid overlap between different nodes. For the best of our knowledge, this is the first time that such a representation is used to display networks. All these depictions can be tuned using the simple graphical user interface to generate high-quality publication-ready images.

An Easily Modifiable and Extensible Molecular Viewer with Editable Content

UnityMol is different from classical pre-compiled molecular viewer executables. In addition to the standalone and webserver versions, it can be run directly within the freely available Unity3D game engine. The game engine environment provides the user with full control over any detail of UnityMol. Because the Unity user interface is very intuitive, many simple things can be changed by the click of a button. It is straightforward to extend the application. If a GUI menu entry is missing, it can be added in minutes with just a few lines, most often by copying existing examples in the code. Moreover, when running UnityMol from within Unity3D, one can switch from game to editor mode and hand-edit the scene. Because the elements in the scene are Unity3D primitives, they can be modified on-screen by the user. For example, when visualizing field lines, the editor mode enables the user to select and delete field lines that are unwanted and hence fine-tune the scene (as in **Fig. 3-C**), despite the fact that we never explicitly implemented such a functionality (see **Fig. S5 in File S1**). This feature can be particularly useful to prototype new 3D molecular representations. Furthermore, instead of the need to recompile programs such as VMD or PyMol to see the results of modifications, using the Unity3D engine helps developers to modify, test and debug their code interactively using the editor window, on-the-fly compilation and the runtime debugger of MonoDevelop, the integrated development environment furnished with Unity3D. These features clearly help to simplify the visualization development due to a rapid and interactive trial-and-error test cycle. Furthermore, Unity3D provides simplified access to low-level graphics hardware functionalities that are

Table 2. Performance comparison between UnityMol using HyperBalls (with Unity 3D Engine Version 3.5 and 4), VMD and PyMol to display molecules with Ball & Stick representation on two different configurations.

MacOSX 10.6 - 2.93 Ghz GTX 285						**Windows 7 - 3.0 Ghz GTX 480**		
PdB id	**Nb Atoms**	**UnityMol V3.5**	**UnityMol V4**	**VMD 1.9 glsl**	**Pymol 1.5**	**UnityMol V4**	**VMD 1.9 glsl**	**Pymol 1.5**
1KX2	1 249	44	52	198	135	35	272	330
1GLQ	3 516	14	16	83	111	12	108	328
3SYJ	7 968	7	8	37	109	5	51	325
2XN2	12 627	4	4	24	70	3	32	321
2XNT	16 550	3	3	18	68	3	24	314
3NOC	25 558	3	3	11	51	3	16	299
3MUY	36 360	3	3	8	44	3	11	278
ZY0S	52 472	3	3	5	34	3	7	239
2B9V	80 710	3	3	4	26	3	4	197
3OAA	99 573	3	3	3	19	3	3	135

doi:10.1371/journal.pone.0057990.t002

usually not exposed in the APIs of existing programs such as VMD or PyMol.

Thus, such a wide range of control by the user could be seen as a new paradigm for scientific software development and presents a clear rupture with current approaches used for molecular visualization tools. At best, common packages provide scripting access to selected functionality to the user, otherwise programming know-how is required to delve directly into the application source code – if it is available. In the context of open computer programs [34], this approach provides users and developers not only with access to the source code but also with a very accessible interface (i.e. the editor) to examine the underlying science and implementation of the program.

UnityMol as a Platform for Prototyping Scientific Tools

We have described how to use a game engine such as Unity3D to create a viewer for molecular structures as well as for biological networks. During the development of these tools, it became obvious that a game engine facilitates the development of scientific applications: the editor window provides an interface to position 3D objects and assist with the attribution of functionalities, e.g. implemented by a script, by dragging an icon onto the target object. Thus, the amount of lines of code to be written is rather limited and can in large parts be hidden for non-experts. Development is further facilitated by support from a very active Unity3D user community (see http://unity3d.com/support/community.html), which provides assistance to novice developers (this is also true for many other game engines). In particular the existing internet forums and answer sites are a good source of information with many pieces of code almost ready-to-use. For example, the original lit sphere shader was adapted from the official Unity3D forum. Several screen-space visual effects and the file browser are user contents as well. For non-expert programmers the availability of such resources speeds-up the development cycle and softens the learning curve of the programming environment. Many common visualization techniques are already implemented and provided with Unity3D such as triangulated spheres, lines, cubes and more, as well as a physics engine and built-in functions to manage different types of controllers. While the development tools are only available on Mac and Windows, stand-alone applications can be deployed for the Linux platform as well. There are some specific limitations in the free version of the Unity3D program (such as less visual effects). Buying a professional license provides access to special features or dedicated functions for systems such as game consoles or smartphones. Work is in progress in our group to explore these additional features and deploy UnityMol on new platforms such as mobile devices.

We only scratched the surface of many features that can be further extended to develop efficient programs. For example, the physics engine could be exploited to create spring networks, molecular dynamics and flexibility. Using this physics engine combined with joystick or Wiimote controllers, it would be straightforward to develop simple tools for interactive docking.

Our immediate plans are to extend UnityMol with practically important aspects for the biophysical community such as molecular dynamics trajectory loading and analysis. We started to experiment with the MDDriver library [35] to be able to connect to running or recorded MD simulations. Concerning analysis facilities (including atom selection and scripting) we intend to link UnityMol to existing frameworks such as for example MDAnalysis [36].

And Designing Software for Edutainment

The video game industry is now booming and outperforms the movie industry with revenues reaching $18.6 billion in the U.S. in 2010. Interactivity is a key advantage that can be achieved with this medium. Direct interaction with the game itself as well as the creation of common virtual worlds where players can interact with each other contribute to this success. Such games can be a platform for education [37] as well as a place of study for academic fields such as social, behavioural and economic sciences [38]. Furthermore, scientific projects start to involve the general public in their investigations [39]. The structure of games is well adapted to involve a community to fulfil a particular task and may be used to resolve scientific issues. This approach has recently been employed to predict protein folding [40] and lead to impressive results [41]. There may be an enormous potential in such scientific "games". Using adequate tools such as game engines may facilitate the creation of such games by scientists. Starting from the code base that we provide, it would be easy for a game designer working with Unity3D to create simple games for education or beyond. Furthermore, using dedicated game controllers such as the Wiimote and Kinect, will allow users to interact with molecules in a more natural manner. This may be a good solution to further implicate young people in classroom demo presentations or museum exhibitions. Interactivity is equally important for researchers to evaluate their hypotheses or develop new ones on live computer models by being able to "play" with them. Currently, interacting with models using tools dedicated for the movie industry is a difficult task given that making an animation may require several weeks, months or more [7].

Concluding Remarks: Developing New Visual Paradigms to Deal with Big Data Challenges and Citizen Science

In the context of big data challenges, the development of citizen science – sometimes referred to as crowdsourcing - has been particularly welcomed and has proven its usefulness [39,42,43]. This approach is particularly promising in combining games with research applications in the biology and computer science fields and hence has already lead to major successes [41,44,45]. In this article we have shown that software usually dedicated to game development, so called game engines, can be used for 3D molecular visualization. This approach raises huge potentialities for quick developments which are particularly useful for the creation of small educational programs or simple serious games with a scientific purpose. The possibility to easily develop programs to automatically target major platforms (Linux, MacOS X and Windows) as well as media such as internet or video game consoles and nomad hardware (smartphones and tablets) re-inforces the impact of this approach to touch a broad audience. The easiness of developing new visual methodologies – such as lit spheres presented in this paper – using the editor window can help developers designing new visual paradigms which are nowadays a priority to deal with challenges of data visualization [46]. These new developments can then be more easily included in well known molecular viewers such as VMD, PyMol, etc … Even if this first version is not as powerful as such molecular viewers, it opens a new way for developing visualizations and may pave the first step towards a convergence between game design and molecular visualization research.

Supporting Information

File S1 Describing 5 supporting figures, 3 supporting texts, 1 supporting table and 1 supporting movie.
(PDF)

Acknowledgments

We are indebted to Prof. Ge Chen for his kind assistance. M.C. would like to thank Dr J.P. Nominé for fruitful discussions.

Author Contributions

Conceived and designed the experiments: MC MB. Performed the experiments: ZL AT FDS CE MC MB. Analyzed the data: ZL AT FDS CE MC MB. Contributed reagents/materials/analysis tools: ZL AT FDS CE MC MB. Wrote the paper: ZL AT MC MB.

References

1. O'Donoghue SI, Goodsell DS, Frangakis AS, Jossinet F, Laskowski RA, et al. (2010) Visualization of macromolecular structures. Nat Methods 7: S42–S55. doi:10.1038/nmeth.1427.
2. Gehlenborg N, O'Donoghue SI, Baliga NS, Goesmann A, Hibbs MA, et al. (2010) Visualization of omics data for systems biology. Nat Methods 7: S56–S68. doi:10.1038/nmeth.1436.
3. Walter T, Shattuck DW, Baldock R, Bastin ME, Carpenter AE, et al. (2010) Visualization of image data from cells to organisms. Nat Methods 7: S26–S41. doi:10.1038/nmeth.1431.
4. Johnson GT, Autin L, Goodsell DS, Sanner MF, Olson AJ (2011) ePMV embeds molecular modeling into professional animation software environments. Structure 19: 293–303. doi:10.1016/j.str.2010.12.023.
5. McGill G (2008) Molecular movies… coming to a lecture near you. Cell 133: 1127–1132. doi:10.1016/j.cell.2008.06.013.
6. Zoppè M, Andrei R, Cianchetta S, Zini M, Loni T, et al. (2009) Blender for Biology: The making of Protein Expressions - Study N2 Amsterdam. Proceeding of the Blender Conference, Amsterdam 2009. Available: www.scivis.ifc.cnr.it/images/stories/blenderconf09paper.pdf.
7. Iwasa JH (2010) Animating the model figure. Trends Cell Biol 20: 699–704. doi:10.1016/j.tcb.2010.08.005.
8. Wong H, Victor J-M, Mozziconacci J (2007) An all-atom model of the chromatin fiber containing linker histones reveals a versatile structure tuned by the nucleosomal repeat length. PLoS ONE 2: e877. doi:10.1371/journal.-pone.0000877.
9. Lewis M, Jacobson J (2002) Game Engines in Scientific Research. Communications of the ACM 45: 27–31.
10. Chavent M, Lévy B, Krone M, Bidmon K, Nominé J-P, et al. (2011) GPU-powered tools boost molecular visualization. Brief Bioinformatics 12: 689–701. doi:10.1093/bib/bbq089.
11. Fernando R, Kilgard M (2003) The Cg Tutorial: The definitive Guide to Programmable Real-Time Graphics. Addison-Wesley Longman Publishing.
12. Rost R (2006) OpenGL(R) Shading Language (2nd edition). Addison-Wesley Professional.
13. Chavent M, Vanel A, Tek A, Lévy B, Robert S, et al. (2011) GPU-accelerated atom and dynamic bond visualization using hyperballs: a unified algorithm for balls, sticks, and hyperboloids. J Comput Chem 32: 2924–2935. doi:10.1002/jcc.21861.
14. Sloan P, Martin W, Gooch A (2001) The lit sphere: A model for capturing npr shading from art. Graphics interface.
15. Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, et al. (2000) The Protein Data Bank. Nucleic Acids Res 28: 235–242.
16. Baker NA (2001) Electrostatics of nanosystems: Application to microtubules and the ribosome. Proceedings of the National Academy of Sciences 98: 10037–10041. doi:10.1073/pnas.181342398.
17. Smoot ME, Ono K, Ruscheinski J, Wang P-L, Ideker T (2011) Cytoscape 2.8: new features for data integration and network visualization. Bioinformatics 27: 431–432. doi:10.1093/bioinformatics/btq675.
18. Andrei RM, Callieri M, Zini MF, Loni T, Maraziti G, et al. (2012) Intuitive representation of surface properties of biomolecules using BioBlender. BMC Bioinformatics 13 Suppl 4: S16. doi:10.1186/1471-2105-13-S4-S16.
19. Sanner MF, Olson AJ, Spehner J-C (1996) AID-BIP4>3.0.CO;2-Y.
20. DeLano W (2002) The PyMOL molecular graphics system. San Carlos, CA: DeLano Scientific LLC; 2002.
21. Pettersen EF, Goddard TD, Huang CC, Couch GS, Greenblatt DM, et al. (2004) UCSF Chimera–a visualization system for exploratory research and analysis. J Comput Chem 25: 1605–1612. doi:10.1002/jcc.20084.
22. Humphrey W, Dalke A, Schulten K (1996) VMD: visual molecular dynamics. J Mol Graph 14: 33–38, 27–28.
23. Sanner MF (1999) Python: a programming language for software integration and development. J Mol Graph Model 17: 57–61.
24. Santavy M, Labute P (n.d.) Electrostatic Fields and Surfaces in MOE. Journal of the Chemical Computing Group. Available: http://wwwchemcompcom/journal/gridhtm. Accessed 2012 Aug 23.
25. Krieger E (2003) YASARA. Available: www.yasara.org. Accessed 2012 Jan 13.
26. Krone M, Stone JE, Ertl T, Schulten K (2012) Fast visualization of Gaussian density surfaces for molecular dynamics and particle system trajectories. EuroVis-Short Papers: 67–71.
27. Lorensen WE, Cline HE (1987) Marching cubes: A high resolution 3D surface construction algorithm. ACM Siggraph Computer Graphics 21: 163–169. doi:10.1145/37402.37422.
28. Chavent M, Levy B, Maigret M (2008) MetaMol: High-quality visualization of molecular skin surface. J Mol Graph Model: 209–216.
29. Krone M, Bidmon K, Ertl T (2009) Interactive visualization of molecular surface dynamics. IEEE Trans Vis Comput Graph 15: 1391–1398. doi:10.1109/TVCG.2009.157.
30. Lindow N, Baum D, Prohaska S, Hege HC (2010) Accelerated visualization of dynamic molecular surfaces. 29: 943–952. doi:10.1111/j.1467-8659.2009.01693.x.
31. Bartalesi I, Bertini I, Hajieva P, Rosato A, Vasos PR (2002) Solution structure of a monoheme ferrocytochrome c from Shewanella putrefaciens and structural analysis of sequence-similar proteins: functional implications. Biochemistry 41: 5112–5119.
32. Weber JR (2009) ProteinShader: illustrative rendering of macromolecules. BMC Struct Biol 9: 19. doi:10.1186/1472-6807-9-19.
33. Bruckner S, Gröller ME (2007) Style Transfer Functions for Illustrative Volume Rendering. Computer Graphics Forum 26: 715–724. doi:10.1111/j.1467-8659.2007.01095.x.
34. Ince DC, Hatton L, Graham-Cumming J (2012) The case for open computer programs. Nature 482: 485–488. doi:10.1038/nature10836.

35. Delalande O, Férey N, Grasseau G, Baaden M (2009) Complex molecular assemblies at hand via interactive simulations. J Comput Chem 30: 2375–2387. doi:10.1002/jcc.21235.
36. Michaud-Agrawal N, Denning EJ, Woolf TB, Beckstein O (2011) MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. J Comput Chem. doi:10.1002/jcc.21787.
37. Mayo MJ (2009) Video games: a route to large-scale STEM education? Science 323: 79–82. doi:10.1126/science.1166900.
38. Bainbridge WS (2007) The scientific research potential of virtual worlds. Science 317: 472–476. doi:10.1126/science.1146930.
39. Bonetta L (2009) New citizens for the life sciences. Cell 138: 1043–1045. doi:10.1016/j.cell.2009.09.007.
40. Cooper S, Khatib F, Treuille A, Barbero J, Lee J, et al. (2010) Predicting protein structures with a multiplayer online game. Nature 466: 756–760. doi:10.1038/nature09304.
41. Khatib F, DiMaio F, Foldit Contenders Group, Foldit Void Crushers Group, Cooper S, et al. (2011) Crystal structure of a monomeric retroviral protease solved by protein folding game players. Nat Struct Mol Biol 18: 1175–1177. doi:10.1038/nsmb.2119.
42. Lintott C, Schawinski K, Bamford S, Slosar A, Land K, et al. (2010) Galaxy Zoo 1: data release of morphological classifications for nearly 900 000 galaxies★. Monthly Notices of the Royal Astronomical Society.
43. Sansom C (2011) The power of many. Nat Biotechnol 29: 201–203. doi:10.1038/nbt.1792.
44. Kawrykow A, Roumanis G, Kam A, Kwak D, Leung C, et al. (2012) Phylo: a citizen science approach for improving multiple sequence alignment. PLoS ONE 7: e31362. doi:10.1371/journal.pone.0031362.
45. Eiben CB, Siegel JB, Bale JB, Cooper S, Khatib F, et al. (2012) Increased Diels-Alderase activity through backbone remodeling guided by Foldit players. Nat Biotechnol 30: 190–192. doi:10.1038/nbt.2109.
46. Rinaldi A (2012) More than meets the eye. Modern experimental techniques require increasingly sophisticated approaches to data visualization. EMBO Rep 13: 895–899. doi:10.1038/embor.2012.135.
47. Garcia-Saez I, P rraga A, Phillips MF, Mantle TJ, Coll M (1994) Molecular Structure at 1.8 A of Mouse Liver Class Pi Glutathione S-transferase Complexed with S-(p-Nitrobenzyl)glutathione and Other Inhibitors. J Mol Biol 237: 298–314.
48. Meng G, Spahich N, Kenjale R, Waksman G, St Geme JW (2011) Crystal structure of the Haemophilus influenzae Hap adhesin reveals an intercellular oligomerization mechanism for bacterial aggregation. EMBO J 30: 3864–3874. doi:10.1038/emboj.2011.279.
49. Fredslund F, Hachem MA, Larsen RJ, Sørensen PG, Coutinho PM, et al. (2011) Crystal structure of α-galactosidase from Lactobacillus acidophilus NCFM: insight into tetramer formation and substrate binding. J Mol Biol 412: 466–480. doi:10.1016/j.jmb.2011.07.057.
50. Akdemir A, Rucktooa P, Jongejan A, Elk RV, Bertrand S, et al. (2011) Acetylcholine binding protein (AChBP) as template for hierarchical in silico screening procedures to identify structurally novel ligands for the nicotinic receptors. Bioorg Med Chem 19: 6107–6119. doi:10.1016/j.bmc.2011.08.028.
51. Monroe N, Sennhauser G, Seeger MA, Briand C, Grütter MG (2011) Designed ankyrin repeat protein binders for the crystallization of AcrB: plasticity of the dominant interface. J Struct Biol 174: 269–281. doi:10.1016/j.jsb.2011.01.014.
52. Dugdale ML, Vance ML, Wheatley RW, Driedger MR, Nibber A, et al. (2010) Importance of Arg-599 of beta-galactosidase (Escherichia coli) as an anchor for the open conformations of Phe-601 and the active-site loop. Biochem Cell Biol 88: 969–979. doi:10.1139/O10-144.
53. Wojtas M, Peralta B, Ondiviela M, Mogni M, Bell SD, et al. (2011) Archaeal RNA polymerase: the influence of the protruding stalk in crystal packing and preliminary biophysical analysis of the Rpo13 subunit. Biochem Soc Trans 39: 25–30. doi:10.1042/BST0390025.
54. Barends TRM, Polderman-Tijmes JJ, Jekel PA, Williams C, Wybenga G, et al. (2006) Acetobacter turbidans alpha-amino acid ester hydrolase: how a single mutation improves an antibiotic-producing enzyme. J Biol Chem 281: 5804–5810. doi:10.1074/jbc.M511187200.
55. Cingolani G, Duncan TM (2011) Structure of the ATP synthase catalytic complex (F1) from Escherichia coli in an autoinhibited conformation. Nat Struct Mol Biol 18: 701–707. doi:10.1038/nsmb.2058.