Real-Time Ray Tracing of Complex Molecular Scenes

Lukas Marsalek*, Anna Katharina Dehof‡, Iliyan Georgiev†, Hans-Peter Lenhof†, Philipp Slusallek*†§ and Andreas Hildebrandt†
*Intel Visual Computing Institute, Saarland University, Saarbrücken, Germany
Email: marsalek@cs.uni-saarland.de
†Chair of Computer Graphics, Saarland University, Saarbrücken, Germany
Email: georgiev@cs.uni-saarland.de
‡Center for Bioinformatics, Saarland University, Saarbrücken, Germany
Email: {dehof, lenhof, anhi}@bioinf.uni-sb.de
§DFKI, Saarbrücken, Germany
Email: slusallek@dfki.de

***Abstract*—Molecular visualization is one of the cornerstones in structural bioinformatics and related fields. Today, rasterization is typically used for the interactive display of molecular scenes, while ray tracing aims at generating high-quality images, taking typically minutes to hours to generate and requiring the usage of an external off-line program.**

Recently, real-time ray tracing evolved to combine the interactivity of rasterization-based approaches with the superb image quality of ray tracing techniques. We demonstrate how real-time ray tracing integrated into a molecular modelling and visualization tool allows for better understanding of the structural arrangement of biomolecules and natural creation of publication-quality images in real-time.

However, unlike most approaches, our technique naturaly integrates into the full-featured molecular modelling and visualization tool BALLView, seamlessly extending a standard workflow with interactive high-quality rendering.

***Keywords*-structural bioinformatics; ray tracing; molecular modeling; rendering; visualization; ray casting**

I. Introduction

The comprehension of the three-dimensional geometry of individual molecules, their potential complexes, and their physico-chemical properties is often key for understanding the biomolecular processes. For instance, rational drug design often involves the development of small molecules that are tailored to fit and fill a certain binding pocket and to interact favorably with the target molecule in that position. Thus, visualizing complex molecules and their properties of interest, e.g. electrostatic potentials, has since been one of the cornerstones of molecular biology and related fields.

Significant attention has been paid to create a faithful and intuitive representation of three-dimensional arrangements on a two-dimensional computer screen. Apart from the use of high-end stereoscopic displays to simulate three-dimensional vision, research in molecular visualization has focused on providing the user with visual cues to improve the understanding of structural relationships. These consist, on the one hand, of a variety of different models or visualization modes of biomolecular entities (e.g. cartoon representations, ball-and-stick models, or surface renderings), each one highlighting specific aspects of the structure under consideration. On the other hand, our perception of three-dimensional objects relies on their interaction with light and simulating such an interaction significantly aids the brain in interpreting a two-dimensional picture as a representation of a three-dimensional entity. Obvious examples of such effects are shadows, light attenuation, and reflections.

However, efficient and accurate implementation of such visual effects, is a challenging task. Typically, the demand for real-time display and full interactivity, i.e. for the ability to animate, move, or modify molecular structures and their components, has prevented the routine use of many useful visualization techniques. For instance, most molecular viewers do not render shadows, simulate correct light attenuation and – to the best of our knowledge – none offers reflections.

The state of the art technique for simulating those effects and achieving high rendering quality, is *ray tracing*. Ray tracing, however, has traditionally been deemed too slow for real-time or interactive display of complex scenes. Thus, while ray tracing is widely popular in molecular graphics, its use has so far been restricted to *offline* generation of high-quality publication images, where rendering one image can take up to several hours on a standard PC.

Interactive visualization has instead relied on rasterization, accelerated by dedicated graphics hardware. However, using rasterization, correct and robust implementation of the described visual effects can be both complex and inefficient.

Hence, many important and useful visual cues are neglected when running interactively. In addition, the significant mismatch between high quality image and interactive preview makes preparation of publication images a painful, time consuming, and sometimes even impossible task, especially if movies (like molecular dynamics simulations) are to be prepared.

Recent advances in computer graphics research enable ray tracing to achieve real-time performance on complex scenes while retaining high visual quality. Consequently, interest in

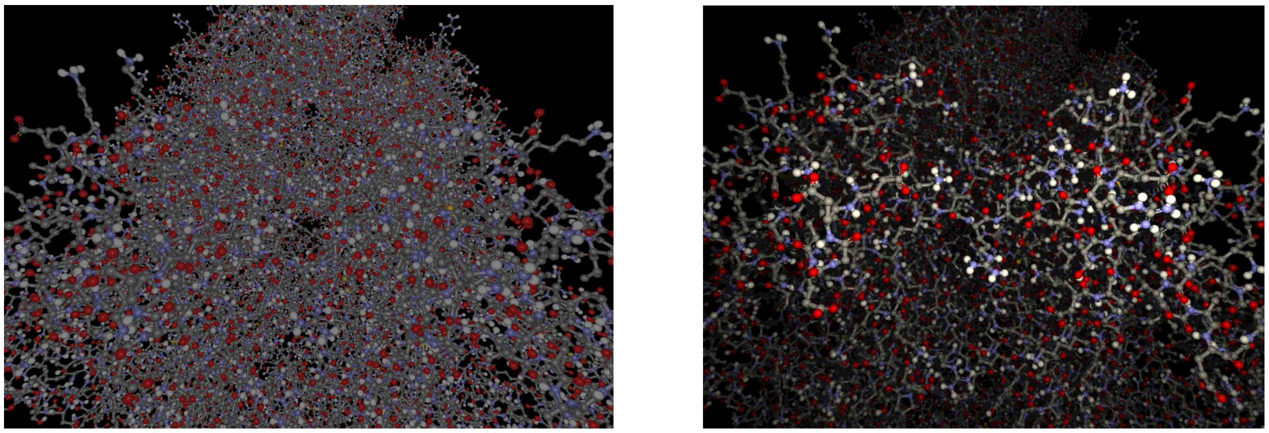


Figure 1. **Light attenuation in depth perception**. These images demonstrate the effect of light attenuation on a spatial perception. On the **left** is an image of *1pma* renderered using only direct illumination without shadows and light attenuation. On the **right** is an interactively ray traced image with shadows and correct light attenuation.

the molecular visualization community in such methods has greatly increased, and indeed real-time ray tracing methods have been reported in the literature [1]–[3].

All previous work, however, has focused on rendering of selected representation(s) only, neglecting the need to combine visualization with the full set of modelling capabilities. In particular, the highly-used cartoon model has been so far completely neglected even though its use especially for complex structures is very common. Also, a standalone visualization-only tool is of little practical importance. To provide a tangible advantage, the high-quality visualization must be tightly integrated into a day-to-day researcher worflow, giving as smooth experience as possible.

In our work we have integrated a general, triangle-based real-time ray tracing framework, the RTfact library [4], into the full-featured molecular viewing and modelling tool BALLView [5], [6]. This allows us to render all the standard representations and their arbitrary combinations, including the cartoon model. The interactive ray tracing is used in the standard workflow, making its use as easy as a classical rasterization while offering the advanced visual effects available all the time. We show how the use of real-time ray tracing allows to boost spatial perception and how it facilitates and simplifies the creation of publication quality images from an intuitive graphical user interface.

II. BALLView

Studying, simulating, and manipulating molecular objects on a computer requires powerful frameworks that on the one hand offer different visualization modes and coloring schemes to graphically present the properties of interest and on the other hand provide all the required editing, modelling, and simulation machinery. But implementing all these functionalities is a complex task. Force fields, for instance, are usually numerically unstable and hard to implement efficiently. Further complications arise when trying to offer all required functionality in an intuitive graphical user interface (GUI) to a largely non-technical user base.

Despite the technical difficulties, a number of successful molecular modeling and viewing applications exist today, such as PyMol [7], VMD [8], Chimera [9], and BALLView [5], [6]. BALLView is a graphical front-end to the Biochemical Algorithms Library (BALL) [10], and as such, it offers direct access to a wide range of modelling functionality, e.g., secondary structure prediction, force field evaluations (Amber, Charmm, MMFF94), structure minimization, or molecular dynamics simulation, to name but a few. These modelling methods are complemented with all standard visualization modes for molecules (lines, stick, ball and stick, Van-der-Waals, SES, SAS, backbone, cartoon, ribbon) and their properties (e.g. HBond-models, force visualization, volume rendering, contouring, etc). BALLView allows visualization models to be easily implemented and added to the system. All these models can be combined with a variety of coloring schemes (e.g. by element, by residue index, or by external properties read from a grid).

BALLView's GUI additionally enables the interactive manipulation and editing of molecules; for instance, small molecules can be easily drawn on the screen and further visualized and simulated. Molecules can be read in from (and exported to) various file formats, or can be directly downloaded from the appropriate databases right from the GUI. Finally, BALL's Python interface is fully embedded into BALLView, rendering it fully scriptable and easily extensible at runtime. Furthermore, BALLView's modern C++ design makes the system an ideally suited target for the integration of real-time ray tracing methods.

III. Real-Time Rendering

In computer graphics, rendering is the process of producing a two-dimensional image of a virtual three-dimensi-onal scene from a camera perspective. To this end, one has to

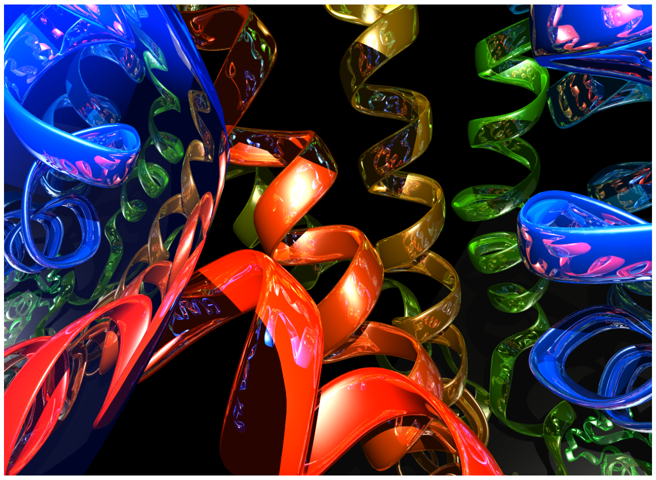


Figure 2. **Reflections in molecular visualization**. Accurate complex multiple interreflections and reflections from curved objects enabled by ray tracing.

compute the radiance of the light falling onto the camera's sensors, or pixels.

In the real world, light starts its journey from light sources, bounces off the particles it encounters, and continues traveling until absorption, eventually reaching the camera. Thus, an accurate computer simulation of the light paths from the light sources to the camera sensors would give a faithful representation of what the camera sees. A formalization of this process is known as Radiative Transfer Theory. In the field of computer graphics this theory is often simplified to account only for light interaction with solid surfaces, neglecting the contribution of participating media. This form has been popularized by the well-known *rendering equation* [11]:

$$L(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega L(h(x,\omega_i),-\omega_i) f_r(\omega_i,x,\omega_o) cos\theta_i \, d\omega_i \quad (1)$$

This model states that the outgoing radiance L from some surface point x in direction ω_o is equal to the self-emitted radiance L_e at the same point and direction plus the radiance that reaches this point from all possible directions ω_i and is reflected in the direction of interest ω_o. The recursive term $L(h(x,\omega_i),-\omega_i)$ yields the incoming radiance from direction ω_i by using the so-called ray tracing operator h, which returns the first visible point from x in direction ω_i. This radiance is weighted by $f_r(\omega_i,x,\omega_o)$, which specifies the reflectance properties of the surface at point x, and $\cos\theta_i$, which accounts for the orientation of the surface with respect to the incoming light direction. Figure 3 left depicts this process.

Most of the existing surface rendering algorithms can be seen as solutions to this equation, the distinguishing feature

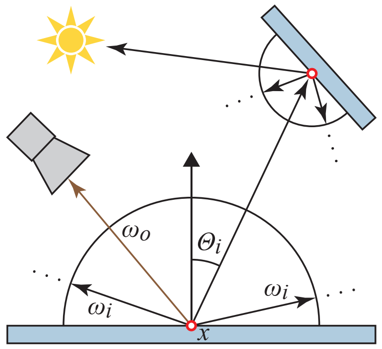


Figure 3. **Principles of ray tracing**. The radiance emitted from a surface point x in direction ω_o equals the self-emitted radiance L_e in that direction plus the radiance from all possible directions ω_i reflected off to ω_o, weighted by the surface orientation and reflectance properties.

being the accuracy of the possible light interactions.

A. *Rasterization*

Rasterization algorithms, like the z-buffer or the painter's algorithm, try to solve the camera visibility problem, i.e. finding the closest visible geometry from the camera perspective. Most such approaches are limited to projecting the geometry onto a planar image. On the other hand, they can robustly handle arbitrary changes in geometry and some algorithms (e.g. z-buffer) have been efficiently implemented in hardware. However, rasterization is not suited for solving the full rendering equation, especially for accurate simulation of secondary lighting effects that require global visibility information. While rasterizing a single primitive, no global knowledge about the scene exists and thus complex effects cannot be computed in a single pass. Over the time, a number of techniques have been proposed to approximate shadows, reflection, and refraction. However, such methods require high graphics expertise to implement, and still trade off correctness for speed.

B. *Ray tracing*

Ray tracing based rendering methods [12] determine the closest visible surface through each pixel by shooting rays from the camera. In contrast to rasterization, visibility computations are not strictly performed with respect to a plane and are image-based rather than object-based. This means that rays can be shot from arbitrary points in arbitrary directions. Moreover, since rays are infinitely thin, ray tracing can be used to *simulate* light transport directly based on the rendering equation, which is a line integral equation. Ray tracing can naturally handle complex lighting phenomena, such as soft shadows, reflections, refractions, as well as full global illumination [13]. Such effects are essential for enhancing the visual perception of the generated image and for providing better understanding of the scene structure. However, ray tracing generally has higher computational

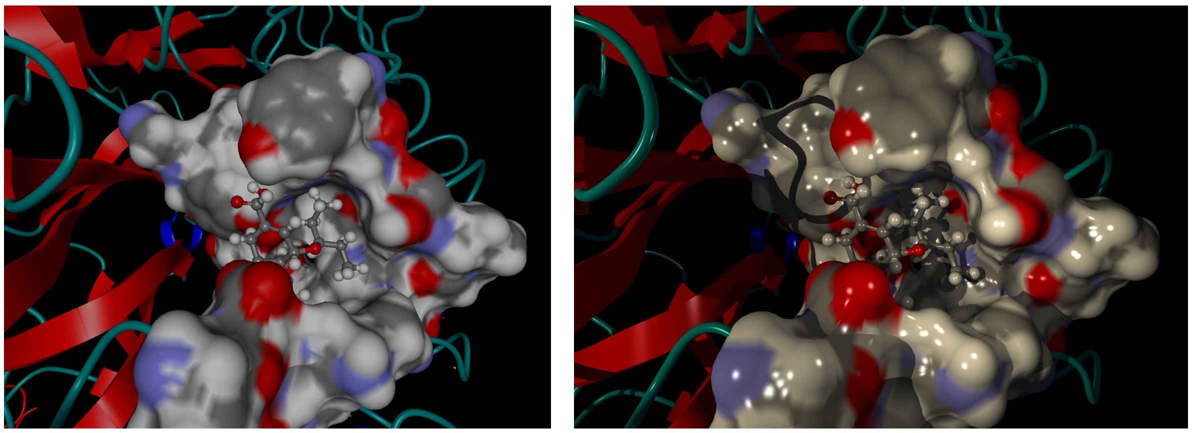


Figure 4. **Shadows in understanding of a structural relationship**. On the **left** is an image of N1 Neuraminidase with oseltamivir/Tamiflu (2HU4.pdb) under direct illumination only. On the **right** the same scene, this time interactively ray traced with realistic shadows. Notice the relative position of the coil and the SES surface, distance of the ball-and-stick model and the depth of the binding site, all of which are not apparent under the direct illumination only.

requirements than rasterization and has thus been used predominantly for offline image generation.

The tremendous technological advances over the past decade have enabled ray tracing to achieve real-time performance on commodity hardware. State-of-the-art ray tracing algorithms exploit the thread- and data-level parallelism provided by modern hardware and utilize highly optimized acceleration structures [14]–[16] to efficiently find ray/object intersections. Real-time ray tracing has for the first time enabled interactive visualization of advanced secondary lighting effects and has thus opened opportunities for unifying interactive and high quality rendering environments.

IV. Integration

We have integrated the RTfact real-time ray tracing engine [4] as a rendering backend in addition to the default OpenGL renderer in BALLView. RTfact is a generic C++ library that utilizes generic programming concepts to deliver both performance and flexibility, and that can robustly handle multiple ray tracing configurations simultaneously. This allows us to use the best suitable algorithms and data structures for a specific application in order to achieve high performance. RTfact also directly exposes the ray tracing functionality to the application, which for example enables object picking or collision testing.

Full-fledged integration of a ray tracer into a molecular modeling tool is a considerable challenge. One reason is that real-time ray tracers are often very rigid, specialized, and hand-optimized programs. Furthermore, molecular modelling tools often deeply integrate the rasterization pipeline. Our aim was to integrate ray tracing into BALLView transparently, so that it would be interchangeable with the default OpenGL renderer.

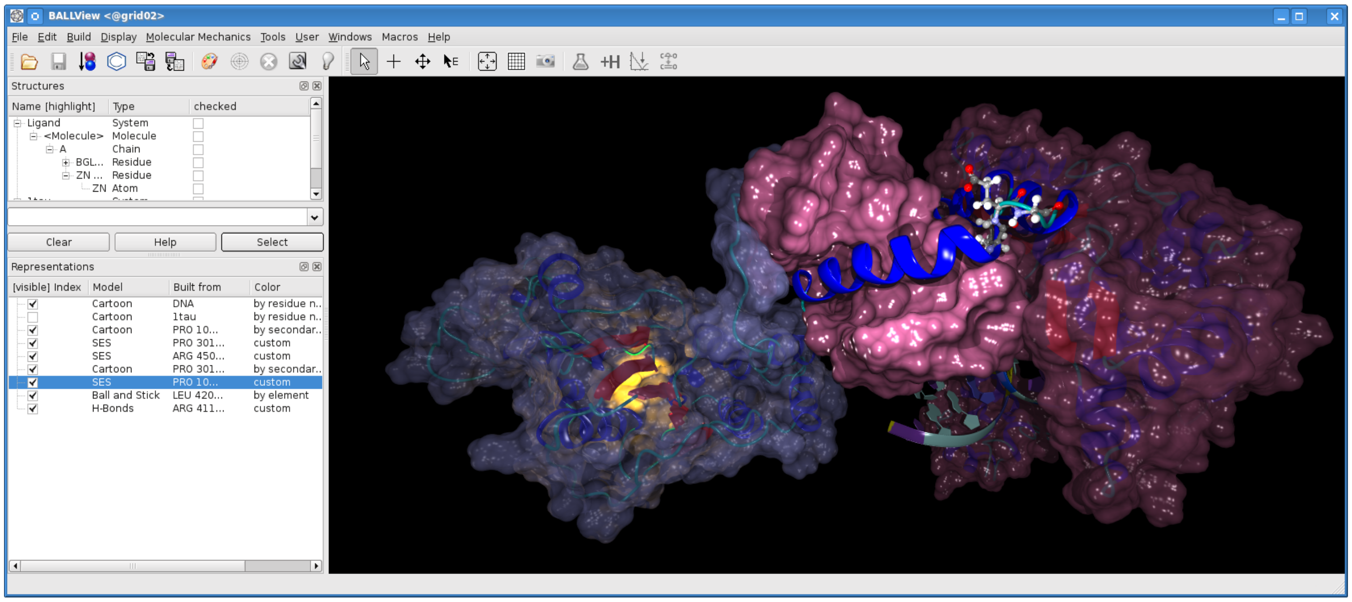
We created an interface in BALLView which is agnostic of the render type and handles both the representations and image display. One reason for this was that ray tracing renderers require a preprocessing phase, where all the objects to be rendered are already known and ray tracing acceleration structures are built. This problem was solved by separating the object handling into two phases. First, a *prepare* stage is executed, where all the representations are gathered and passed on to the renderers. Then, a *render* phase only tells the renderers to display whatever representations they have bufferred. Additionally, such renderers render the image into a memory buffer, whereas the OpenGL renderer directly runs on the graphics card. The image produced by the ray tracer is texture-mapped to a full-screen quad, which is then rasterized. This both enables unified handling of *render targets* and enables leveraging the GPU for post-processing the final image, such as linear or higher order interpolation (see Section V-C). In this way, we abstract the rendering completely from the computational core of BALLView, allowing all of its visualization methods, coloring schemes, and simulation techniques to be used in combination with both the ray tracing and the OpenGL renderers.

V. Results and Discussion

We demonstrate the key advantages and implications to a standard workflow that are achieved by the combination of general real-time ray tracing and a full-featured molecular modelling tool as described above. Specifically, we address the degree of image quality, structural, and depth perception a user can expect in interactive viewing by switching to the ray tracing paradigm, the level of integration achieved, basic performance characteristics, and ease of use and publication image generation.

A. *Structure and Depth Perception*

Ray tracing naturally simulates accurate light attenuation, shadows, and reflections. These effects require almost no


BALLView <@grid02>
File Edit Build Display Molecular Mechanics Tools User Windows Macros Help
Structures
Clear
Help
Select
Representations

Name [highlight]	Type	checked
Ligand	System	☐
<Molecule>	Molecule	☐
A	Chain	☐
BGL...	Residue	☐
ZN ...	Residue	☐
ZN	Atom	☐

[visible] Index	Model	Built from	Color
✓	Cartoon	DNA	by residue n..
☐	Cartoon	1tau	by residue n..
✓	Cartoon	PRO 10...	by secondar..
✓	SES	PRO 301...	custom
✓	SES	ARG 450...	custom
✓	Cartoon	PRO 301...	by secondar..
✓	SES	PRO 10...	custom
✓	Ball and Stick	LEU 420...	by element
✓	H-Bonds	ARG 411...	custom

Figure 5. **Combination of representations.** Integration of the ray tracer is done tightly and transparently, allowing the advanced effects reflection, shadows, light attenuation or transparency to work seamlessly with any combination of available representations, including the cartoon models.

additional effort to implement and are guaranteed to be precise, unlike similar techniques in rasterization.

Impact of these light effects on depth perception is demonstrated in Figure 1. On the left, an image without any light attenuation or shadows is rendered, while on the right an interactively ray traced image is given. We can see that without the light attenuation and shadows, any sense of depth is completely lost.

Similar situation is depicted on Figure 4. Here, we see how the presence of shadows helps to clarify the relationship of the coils and SES surfaces, as well as position of the ligand in the binding pocket.

Finally, Figure 2 demonstrates the use of reflections in molecular visualization.

Stereoscopic imaging is often used in molecular modeling to enhance depth and spatial perception. Real-time ray tracing techniques are fully compatible with stereoscopy and we have implemented stereo support in BALLView.

B. *Integration of different representations*

The tight integration of modeling and rendering and the choice of triangle-based ray tracing, brings the renderer entirely *into* the molecular scene, making it aware of all the representations and their hierarchy. This enables the ray tracer to treat all the triangle-based representations in the same manner. An important consequence of this is that all the advanced effects presented in the previous section apply automatically and consistently to all representations: for example, the cartoon model can cast shadows on the SES surface, which in turn shadows a ball-and-stick model. Figure 5 shows such a situation. This is in strong contrast to previous applications of ray tracing to molecular visualization, where only selected representation(s) were ray traced. Specifically, the very popular cartoon model was the one most often neglected.

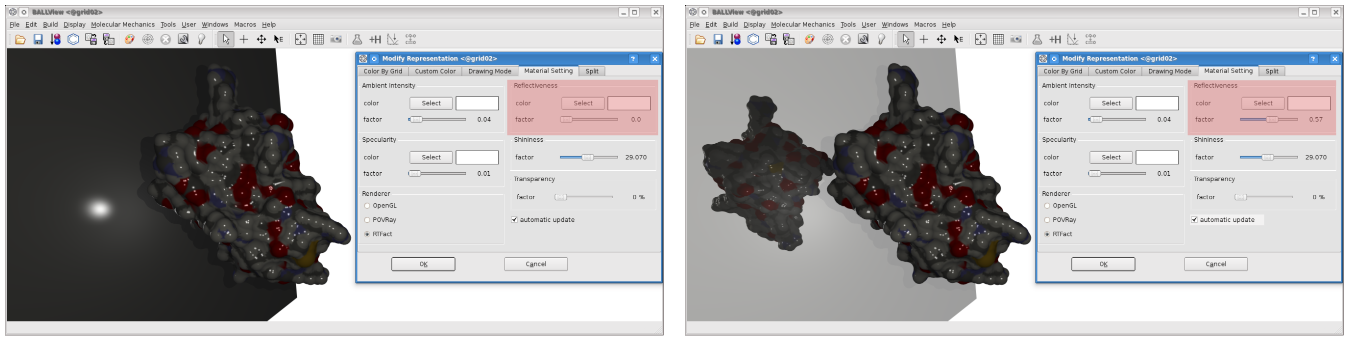
C. *Performance*

In this section we would like to stress out that all of the above mentioned effects are rendered *interactively* in our system and provide an analysis of the performance with respect to the scene complexity and number of processors.

Our ray tracer is a highly multi-threaded application, which can take advantage of all CPU cores available in the system, requiring no user intervention. This means that the ray tracer automatically speeds up when moved to a machine with higher number of cores. If desired, however, the user can specify the number of threads at runtime, e.g. to save shared computing resources. In contrast to rasterization, our method is also independent on the graphics card used. This is advantageous, e.g., in server-based rendering, as CPU clusters are still more reachable than GPU clusters.

Figure 7 on the right, shows how the ray tracing performance scales with the number of cores in the system. We can see that the scaling is linear at the beginning and slightly declines with increasing number of cores. In the end it provides about 5x speed-up for 8x times the amount of cores. This makes good use of all the available resources and provides very good performance/price ratio.

The final issue discussed here is the scaling of the rendering system with the scene complexity. The optimal goal is to achieve a better than linear dependency of the rendering times on the amount of geometry we are displaying. Imple-


BALLView <@grid02>
File Edit Build Display Molecular Mechanics Tools User Windows Macros Help
Modify Representation <@grid02>
Color By Grid
Custom Color
Drawing Mode
Material Setting
Split
Select
OK
Cancel
BALLView <@grid02>
File Edit Build Display Molecular Mechanics Tools User Windows Macros Help
Modify Representation <@grid02>
Color By Grid
Custom Color
Drawing Mode
Material Setting
Split
Select
OK
Cancel

Ambient Intensity
color
factor 0.04
Reflectiveness
color
factor 0.0
Specularity
color
factor 0.01
Shininess
factor 29.070
Transparency
factor 0 %
Renderer
OpenGL
POVRay
[✓ RTFact]
[✓ automatic update]

Ambient Intensity
color
factor 0.04
Reflectiveness
color
factor 0.57
Specularity
color
factor 0.01
Shininess
factor 29.070
Transparency
factor 0 %
Renderer
OpenGL
POVRay
[✓ RTFact]
[✓ automatic update]

Figure 6. **Easy to use user interface**. Adding the advanced ray tracing effects like reflections, for example, is as simple as dragging a material reflectivity slider between 0 and 1, giving a smooth transition between diffuse material and perfect mirror.

menting this functionality in rasterization-based algorithms is, similarly to high-order effects, quite tedious, error prone, and requires high expertise. On the other hand, real-time ray tracing naturally relies on hierarchical tree representations to achieve high-speed rendering. These structures automatically give the desired scaling behaviour, as each ray is checked for intersection only with objects in its vicinity.

The results are summarized in Figure 7 on the left. It demonstrates that our implementation scales logarithmically with the amount of geometry being displayed. For further details please refer to the caption of Figure 7.

D. *Ease of use*

One important drawback of previous ray tracing solutions in molecular visualization was the lack of integration with day-to-day workflow and/or difficult parameter setup. Especially off-line image or movie generation using external ray tracers is very tedious, both due to high rendering times and the complexity of the scene preparation.

In our system, the user will only notice several new options, while receiving automatically all the advanced effects mentioned above *in standard online rendering viewport*. See for example Figure 5. All parameters, such as light sources, colors, or materials, can be changed on the fly and the results of the changes are directly reflected on the screen (Figure 6) In this way, producing publication-quality pictures becomes nearly trivial and does not require more skill or experience than using a molecular viewer in the first place. Indeed, all the images shown in this paper have been generated by either taking a screenshot of a running BALLView instance, or by having BALLView exporting its current display into a file.

VI. Conclusion

One of the most important aims in molecular visualization is a faithful and easily interpretable depiction of the structural relationships in biomolecular systems. In this work, we present a tight integration of general real-time ray tracing functionality into a fully-featured molecular viewing and modelling application.

We have shown that the classical paradigm in molecular visualization – rasterization for interactive display, ray tracing for offline creation of high-quality images with full visual cues for spatial perception – can indeed be changed to the 'best of both worlds', where we have both interactive and high-quality visualization with all the advanced effects at hand for day-to-day routine work. The added information due to the correct shadows, advanced lighting, and even cleverly used reflections can hardly be overemphasized, especially when used in an interactive setup. Moreover, this functionality is directly integrated into the standard workflow, making its use nearly trivial and natural.

However, perhaps the most exciting fact is that with the real-time ray tracing we have only started to explore the whole new world of opportunities and we strongly believe that a new class of interaction and visualization paradigms can be designed, including, e.g., triangle-less rendering of specific representations, volume visualization, or even better use of global illumination.

Acknowledgements

This work was supported by International Max Planck Research School for Computer Science Saarbruecken, Center for Bioinformatics, MMCI, Saarland University, Charles University in Prague, and Grant Agency of Charles University through GAUK Grant no. 88008

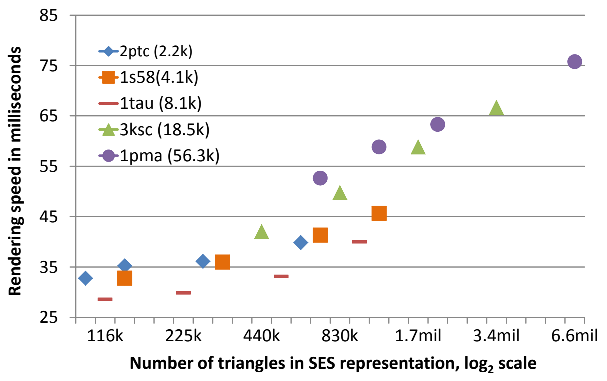
References

[1] M. Chavent, "High quality visualisation: from small molecules to macromolecular systems," *Poster, EMBO Workshop on Visualizing Biological Data*, 2010.

[2] B. M. M. Chavent, B. Levy, "MetaMol: High-quality visualization of molecular skin surface," *Journal of Molecular Graphics and Modelling*, vol. 27, no. 2, 2008.

[3] M. B. M. G. Ch. Sigg, T. Weyrich, "GPU-based ray-casting of quadratic surfaces," *Eurographics Symposium on Point-Based Graphics*, 2006.

[4] I. Georgiev and P. Slusallek, "RTfact: Generic Concepts for Flexible and High Performance Ray Tracing," *IEEE Symposium on Interactive Ray Tracing*, 2008.



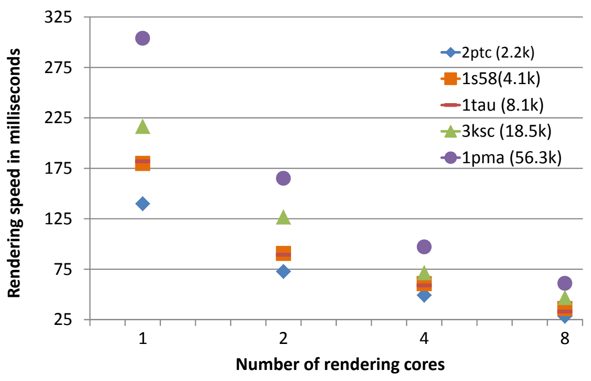


Figure 7. **Ray tracing performance. Left:** Rendering speed in milliseconds with respect to the amount of geometry in the scene. We have measured the speed on 5 different structures (PDB IDs 2ptc, 1s58, 1tau, 3ksc and 1pma) of different sizes (number of atoms in brackets in the legend). Each structure was separately measured with 4 different levels of tesselation of SES. We can see that for each of the larger structures (3ksc, 1pma) separately, the speed maintains logarithmical behaviour with respect to the number of triangles. The smaller structures (2ptc, 1s58, 1tau), while still close to logarithm have some tendency towards linear behaviour. However, across all the different structures, general trend remains logarithmic. **Right:** Dependence of the rendering speed of SES representation on the number of cores available for the rendering. The results were measured on the same set of molecules as above. We can see that with smaller amount of cores the rendering speed exhibits almost linear scaling, while later on the gains shrink. Overall, we obtain 5x speed-up with 8x the amount of cores. Also note, that the scaling behaviour is consistent across the structures.

[5] A. Moll, A. Hildebrandt, H.-P. Lenhof, and O. Kohlbacher, “BALLView: a tool for research and education in molecular modeling.” *Bioinformatics*, vol. 22, no. 3, pp. 365–366, 2006.

[6] ——, “BALLView: an object-oriented molecular visualization and modeling framework.” *J Comput Aided Mol Des*, vol. 19, no. 11, pp. 791–800, 2005.

[7] W. DeLano, “The PyMOL molecular graphics system,” *DeLano Scientific LLC*, 2008.

[8] W. Humphrey, A. Dalke, and K. Schulten, “VMD: visual molecular dynamics.” *J Mol Graph*, vol. 14, no. 1, pp. 27–38, 27–8, Feb 1996.

[9] E. Pettersen, T. Goddard, C. Huang, G. Couch, D. Greenblatt, E. Meng, and T. Ferrin, “UCSF chimera–a visualization system for exploratory research and analysis.” *J Comput Chem*, vol. 25, no. 13, pp. 1605–1612, Oct 2004.

[10] O. Kohlbacher and H. P. Lenhof, “BALL–rapid software prototyping in computational molecular biology. biochemicals algorithms library.” *Bioinformatics*, vol. 16, no. 9, pp. 815–824, Sep 2000.

[11] J. Kajiya, “The rendering equation,” *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 143–150, 1986.

[12] A. S. Glassner, *An Introduction to Ray Tracing*. Morgan Kaufmann, 1989.

[13] M. Pharr and G. Humphreys, “Physically Based Rendering: From Theory to Implementation,” *Elsevier Science and Technology Books*, 1990.

[14] I. Wald, “Realtime Ray Tracing and Interactive Global Illumination,” Ph.D. dissertation, Saarland Univeristy, 2004.

[15] A. Reshetov, A. Soupikov, and J. Hurley, “Multi-Level Ray Tracing Algorithm,” *ACM Transactions on Graphics*, 2005.

[16] I. Wald, S. Boulos, and P. Shirley, “Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies,” *ACM Transactions on Graphics*, 2007.