Eurographics Workshop on Visual Computing for Biology and Medicine (2012)
T. Ropinski, A. Ynnerman, C. Botha, and J. B. T. M. Roerdink (Editors)

Atomistic Visualization of Mesoscopic Whole-Cell Simulations

Martin Falk[†], Michael Krone[†], and Thomas Ertl[†]

Visualization Research Center (VISUS), University of Stuttgart, Germany

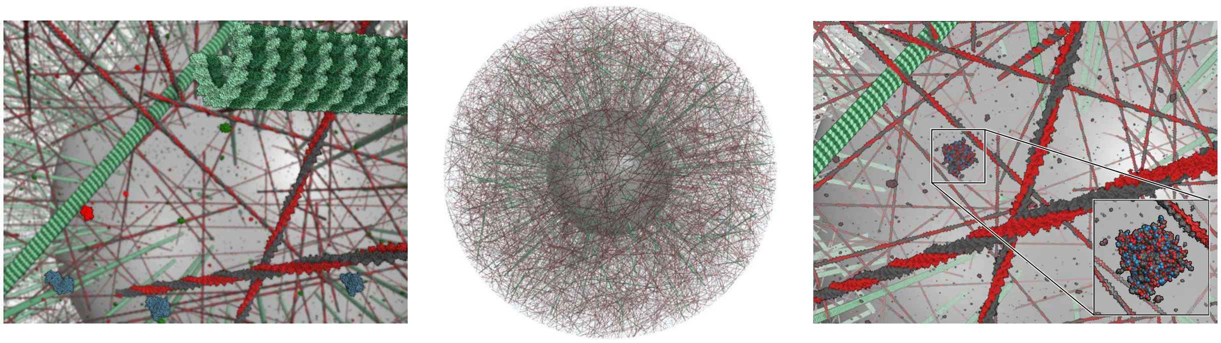

Figure 1: *Simulation of the ERK pathway within a simplified cell model (25 billion atoms, center). With our optimized visualization algorithm, it is possible to render this data set at 3.6 fps. Left: close-up view rendered with depth of field. Right: close-up view showing the cytoskeleton and signal proteins. For molecules near the camera, individual atoms are discernible.*

Abstract

Molecular visualizations are a principal tool for analyzing the results of biochemical simulations. With modern GPU ray casting approaches it is only possible to render several millions of atoms at interactive frame rates unless advanced acceleration methods are employed. But even simplified cell models of whole-cell simulations consist of at least several billion atoms. However, many instances of only a few different proteins occur in the intracellular environment, which is beneficial in order to fit the data into the graphics memory. One model is stored for each protein species and rendered once per instance. The proposed method exploits recent algorithmic advances for particle rendering and the repetitive nature of intracellular proteins to visualize dynamic results from mesoscopic simulations of cellular transport processes. We present two out-of-core optimizations for the interactive visualization of data sets composed of billions of atoms as well as details on the data preparation and the employed rendering techniques. Furthermore, we apply advanced shading methods to improve the image quality including methods to enhance depth and shape perception besides non-photorealistic rendering methods.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing, I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types, J.3 [Computer Applications]: Life and Medical Sciences—Biology and genetics

1. Introduction

The interactive rendering of large particle systems has been an active area of research for many years. One of the most prominent areas of application is the visualization of the results from particle-based simulations. The data sets coming from this domain have been rapidly increasing in size over the last years. This is partly due to the advancements of the simulation software and partly due to the increasing compute capability and the availability of massively parallel hardware like compute clusters and accelerators like GPUs. Exa-scale visualization still seems to be a long way off, but even the data sets available today are pushing the rendering capability of current graphics hardware to its limits. However, it is

[†] {martin.falk|michael.krone|thomas.ertl}@vis.uni-stuttgart.de

DOI: 10.2312/VCBM/VCBM12/123-130

important that the visualization keeps up with the simulations to ensure that the domain scientists can still analyze their data sets. One possible and popular approach is to use compute clusters for visualization. From our perspective, it is important to enable the visualization of large particle data sets on common desktop workstations. This is not only more convenient for the user but also enables a wider range of users to get access to the data. Our main focus lies on the visualization of biomolecular data sets. More precisely, we want to visualize the results from coarse-grained (so-called mesoscopic) particle-based simulations of whole cells. Even though the simulations operate on single particles representing whole molecules, our goal was to visualize the simulation on an atomic scale (cf. Figure 1). This can help scientists to see the detailed function of cellular processes and gain a better understanding of cellular dimensions and complexity. While atom-based simulations typically range up to only tens of millions of atoms, mesoscopic simulations can easily reach tens of billions of atoms if shown on an atomistic level. Our work builds on the recent work of Lindow et al. [LBH12] who visualize molecular structures reconstructed from electron tomography images. As in the work of Lindow et al., our simulation data sets consist of large numbers of instances of only few individual molecules. Apart from showing the underlying data of the simulation and providing new, detailed insights into the simulated system, our work can also be used for artistic reasons and educational purposes, e.g. for showing the crowded internal environment of the cell. Therefore, we also show the applicability of various image enhancement techniques to create renderings which are visually pleasing and, moreover, highlight the structure and spatial relationships in the data.

The main contribution of our work is an application for the interactive visual exploration and visual analysis of mesoscopic whole-cell simulations on an atomistic level of detail. We use an extended and optimized version of [LBH12] to visualize our dynamic data sets resulting from simulations. Furthermore, we show the applicability and feasibility of illustrative methods to enhance the data analysis and image quality. Our visualization is interactive for giga-scale particle data sets on consumer graphics hardware.

2. Previous Work

When dealing with huge particle data sets, parallel visualization approaches are a reasonable and, in most cases, promising approach. Popular visualization frameworks like ParaView [LHA01] support parallel visualization. As stated in the introduction, our work focuses on visualizations using commodity desktop workstations equipped with high-end consumer graphics hardware. In the following, we therefore focus on the previous work from this area.

Simple glyph-based visualization of particle data sets is a flexible and powerful representation for data analysis. Most commonly, particles are rendered as spheres or ellipsoids, where features can be mapped to visual qualities, like the radius or color of a sphere. Gumhold [Gum03] introduced the concept of point-based GPU ray casting to render large numbers of ellipsoids interactively. This method can be seen as the de-facto standard for rendering quadrics nowadays since it is vastly superior to classical triangle-based geometry in terms of speed and image quality. Recently, Grottel et al. [GRE09] showed that it is possible to render millions of spheres interactively on modern GPUs without advanced acceleration techniques. They also evaluated different ways to upload the data to the GPU for rendering, which today is one of the major bottlenecks for visualizing large, dynamic data sets. Subsequently, they extended their method using a two-level occlusion culling and deferred shading [GRDE10]. With this optimized technique, they have been able to render data sets of up to 100 million particles interactively. However, as with all occlusion culling methods, their algorithm is only beneficial for dense data sets with a large amount of occluded geometry. This is not the case for our data sets since the interior of our simulated cell is rather sparsely populated due to the simplified simulation model.

Recently, Lindow et al. [LBH12] presented a method to render biomolecular data sets comprising over a billion atoms. Since they visualize molecular structures reconstructed from electron tomography (ET) images, their data sets consist of many instances of the same molecules. Due to the limited resolution of ET images it is only possible to identify individual molecules or molecular complexes, but not to extract the spatial structure or even individual atoms. Therefore, the same model is used to visualize all molecules of a certain species. Their visualization method exploits this fact to achieve interactive rendering performance. The general idea of their algorithm can be outlined as follows: Each molecular model is uploaded to the GPU memory during the preprocessing stage. During rendering, instances of the bounding boxes of the uploaded models are rendered for each molecule. The individual atom spheres are then rendered using a GPU ray marching approach within each bounding box, similar to GPU volume ray casting [HSS*05]. In Section 5, we give a detailed description of the original algorithm and explain our extensions and optimizations. A similar approach was presented by Lampe et al. [LVRH07] to visualize large proteins. They assemble the proteins on the GPU after transferring only the position and rotation of each amino acid. The individual atoms per amino acid are stored in a texture. The points for the atoms are generated and transformed in the geometry shader and rendered using ray casting in the fragment shader. In contrast to [LBH12], this method is limited by the number of primitives that can be emitted by the geometry shader.

3. Biological Background

In recent years, systems biology became an emerging, multidisciplinary field. Systems biology investigates the characteristics and the complex interactions of all elements in a particular biological system [Pal00]. Methods from systems theory are used to build mathematical models of processes within an organism. These models are tested either with *in-silico* simulations or heuristics. Afterwards, the model results are then compared with the data obtained in experiments. In the context of our work, signaling processes in the cellular environment are of special interest.

We approximate the complex interior of a biological cell for in-silico simulations as described by Falk et al. [FKRE09]. The plasma membrane defines the outer boundaries of the cellular environment and has a spherical shape. The nucleus, which contains the DNA, is located in the interior of the cell. The remaining space inside the plasma membrane, the cytoplasm, is filled with various compartments, proteins, hormones, and water molecules, which are the most abundant species. The number of proteins of the same species in a single cell ranges from only a few to billions. For example, there are about 500 billion actin molecules in a liver cell. The structure of the cell is maintained by the cytoskeleton, a scaffold built of three different filament types. Microtubules exhibit a cylindrical form of thirteen protein columns. Their main functions are the stabilization and movement of the cell and the intracellular transport. Actin filaments, also known as microfilaments, stabilize the cell and withstand both tensile and compressive forces. The intermediate filaments, the third filament type, build a tightly connected network around the nucleus. They serve as fixture for organelles in the cytoplasm. All three types of filaments are represented by elongated cylinders. Proteins in the cytoplasm move by diffusion and through the so-called motorized transport along microtubules and actin filaments with the aid of motor proteins.

Molecular dynamics simulations are not suitable to simulate such a cellular environment on the atomic scale because they only cover time scales in the range of picoseconds and nanoseconds. Such timescales are too small to reproduce the effects of cellular signaling mechanisms, which take place in seconds to even hours. To solve this problem, mesoscopic simulations can be employed. When neglecting atomistic effects, the atomic structure of a signaling molecule can be replaced by a sphere with the hydrodynamic radius of the physical molecule. We use a three-dimensional agent-based stochastic simulation employing CUDA for GPU acceleration [FOE*11] where each agent represents one single molecule. This type of simulation also considers spatial effects like asymmetric protein distributions in contrast to models solely based on differential equations, which use average concentrations [PSQH06].

4. Data

The atomic structures of all molecules used in this paper are obtained from the protein data bank (PDB) [PDB]. The data

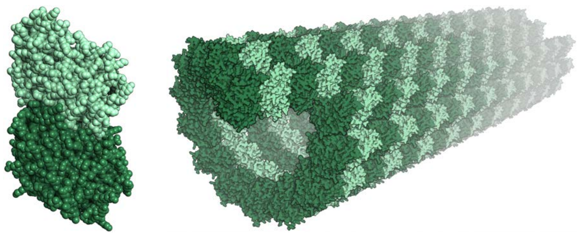
Figure 2: *Structure of microtubules. Left: α-tubulin (dark green) and β-tubulin (light green). Right: thirteen complexes of α- and β-tubulin are arranged in a circle building one turn of a microtubule.*

files provide information on the location and type of atoms as well as conformal information.

Proteins can usually be loaded from a single file. However, in some cases a file might contain only a part of the protein which then has to be combined with other protein parts.

Microtubules consist of α- and β-tubulin, which form dimers (Figure 2). Thirteen dimers (PDB-ID: 1TUB) are arranged in a circle building one row of the microtubule yielding a hollow structure. The dimer itself is about 8 nm high and is continuously shifted along the microtubule axis reaching 12 nm per turn [LBK*07]. The resulting seam on one side of the tubule can be seen in Figure 2, right.

Actin filaments are two intertwined monomers consisting of individual actin molecules (PDB-ID: 3MFP). Figure 3 shows one complete turn of the helical structure built by 28 actin molecules, i.e. 14 molecules per monomer. The length of one turn is 77.3 nm. The helix can be constructed either with the transformation provided in the PDB entry or the model of Holmes et al. [HPGK90].

5. Algorithm

The rendering method we are using relies on the fact that, even though the overall number of particles in the scene may be as high as several billions of atoms, the number of individual molecules is typically quite low. That is, the atom positions of each type of molecule have to be stored only once and can be rendered for all instances of the same molecules. This reduces the number of atoms which have to be stored in main or graphics memory to at most a few millions. In the following, we explain the rendering technique presented by Lindow et al. [LBH12] onto which our work is based on. We also go into detail about our extensions and optimizations to the original work.

Molecule Setup and Instancing. As outlined in Section 3, our primary sources of data are mesoscopic simulations which do not operate on individual atoms but on whole molecules. Therefore, all molecules in the scene are rigid, that is, they are only translated and rotated, but do not undergo any internal deformations. Consequently, we have to

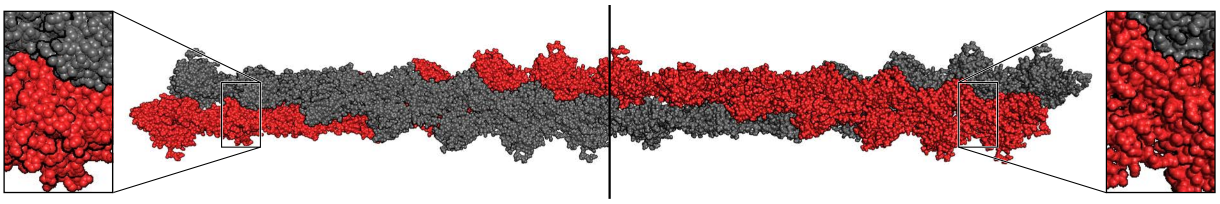

Figure 3: *Actin filaments are built of two intertwined actin monomers and one complete turn of a monomer consists of 14 actin molecules. The normal approximation (right) enhances the global structure compared to the shading with exact normals (left).*

transfer the atomistic molecular models only once per protein type to the GPU memory—just like the original algorithm. During rendering, we can draw multiple instances of the same molecule. This allows keeping the amount of data which has to be transferred for each render pass from the CPU to the GPU very low. For each molecule we render, we only have to transfer the translation and rotation which can either be stored in a single matrix or in a quaternion and an additional translation vector. In the vertex shader, the quaternion rotation must then be converted back into a matrix.

For the microtubules we use two instance blocks, one with ten rows consisting of 130 tubulin dimers resulting in 882 k atoms and one with twenty rows (1.76 M atoms). The two blocks of the actin filaments consist of 28 actin molecules (82 k atoms) and 56 molecules (164 k atoms) respectively. The blocks of actin filaments or microtubules are fitted on the filaments of the cytoskeleton from the simulation. Since the elongation of those blocks is small compared to the total length of the filaments, the remaining uncovered part is less than 80 nm and, therefore, negligible.

Grid Traversal and Ray Casting. To allow for fast, high-quality rendering of the atoms, we employ ray casting to compute the intersections between the viewing rays and the atoms. A uniform grid is used for space subdivision to reduce the number of intersection tests. All atoms of a molecule are sorted into this grid. A box-sphere intersection is used to determine all cells of the grid with which an atom intersects. To store the data of the uniform grid we use the grid data structure proposed by Lagae and Dutré [LD08]. Each cell of the grid is mapped onto one voxel of a 3D texture. The atomic data—i.e. position, radius, and color IDs—are stored in two additional 2D textures. The first texture contains the atom position and its radius represented in 32-bit floating point values. The color identifiers, e.g. atom type, chain ID, or strand ID, are stored in the second texture with up to four channels. This additional data is accessed via a 2D index stored in the 3D grid texture. In addition to the 2D index, each voxel also contains the number of atoms in this cell. Upon rendering the atoms, only the grid's bounding box of each instance is drawn. By rendering only the back faces of the bounding box and computing the corresponding position on the front side, the ambiguity between front and back faces is avoided. The fragment shader computes the view ray through each fragment covered by the bounding box and traverses the grid cells front to back. The individual ray-sphere intersections for the atoms are consequently computed per grid cell. As soon as at least one atom is hit, the grid traversal can be stopped. Note that it is important to compute all intersection in one grid cell to obtain the closest intersection because the atoms are not ordered.

Lindow et al. [LBH12] observed that a ray-voxel traversal [AW87] is superior to a ray-layer traversal, where several grid cells have to be considered for each step along the view ray. The only drawback of the former method is that the atoms have to be replicated for each grid cell they intersect. If the size of the grid cells is chosen with respect to the atom size, one can ensure that a single atom is contained in at most eight grid cells. Moreover, the better rendering speed outweighs the additional memory consumption, which is not a problem on current GPUs with 1 GiB or more of graphics memory. Lindow et al. proposed a grid where the number of voxels equals the number of atoms. We chose the size of the grid cells with respect to the atom size, ensuring that a single atom is contained by at most eight grid cells. Hence, a voxel size of 4 Ångström was chosen. In our experience, this also gives the best performance. We observed that in this case, the resulting number of voxels roughly corresponds with the number of atoms, that is, we concur with Lindow et al.

Deferred Shading. Similar to Grottel et al. [GRDE10], Lindow et al. [LBH12] use deferred shading to speed up the rendering. Both works also use a normal correction scheme to smooth out high frequencies between adjacent normals of distant objects. This results in a more continuous lighting which creates a surface-like impression.

Grottel et al. [GRDE10] propose to use deferred shading with different normal calculations depending on the distance from the camera. For near objects, the analytically computed normals are used for shading. As the objects move away from the camera an approximated normal is used. The approximated normal is the normal of the center point of a quadratic Beziér surface over the current point and its neighbors. A smooth transition between analytical and approximate normal is obtained by linear interpolation. Figures 3 & 4 show a comparison between analytical and approximated normals.

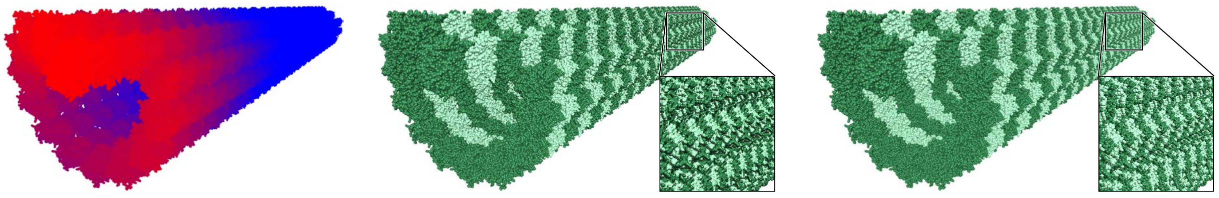

Figure 4: *Normal estimation along a microtubule. Left: interpolation between analytical normal (red) and averaged normal (blue) depending on the camera distance. Center: the roughness of the surface is revealed with normal estimation. Right: without normal estimation. Note, only direct illumination is applied.*

5.1. Optimizations

Depth Culling. In classical polygon-based computer graphics, the OpenGL pipeline can reduce the computational load by the early z test, which takes place after the vertex processing stage: If a fragment which is closer to the camera than the current fragment was already rendered, the computation for the current fragment can be aborted. However, the early z test is automatically disabled if a fragment shader manipulates the depth of the fragment or uses `discard` to reject fragments. We actually need both functions: We need the correct per-fragment depth values for all spheres for molecule-molecule intersections and we must discard fragments where a ray traverses the bounding box of an object but does not write any output because no atom is hit.

We propose to store the depth of the closest intersection and reuse it in the fragment shader to perform an early reject. If the stored depth value is smaller than the depth of the bounding box front, the computation in the fragment shader can safely be discarded. In case the front position is closer the grid traversal takes place and if an intersection is found, the resulting depth is stored into a second texture. The two textures with the depth values are swapped in a ping-pong fashion after n molecule instances have been drawn. The regular depth buffer is still necessary to ensure the correct depth ordering of the n instances of one pass. Since we assume the molecules to be randomly distributed in our cell n can be large and the optimization is still beneficial (see Section 6).

To avoid a second render target, which would slow down the rendering, we combine the depth information with the normal and the color ID resulting from the ray-sphere intersections. By using integer color IDs, we can save the scaled third component of the normal together with the color id and are able to reconstruct the normal in the deferred shading. Therefore, it is possible to store the normal, the color ID, and the closest depth in a single texture. In the subsequent deferred rendering, both result textures are sampled and the texture with the smaller depth value is chosen.

Hierarchical Ray Casting When visualizing our simulation results, most of the molecules will only cover a few pixels on the final image. The resulting artifacts can be remedied by the approximate normal approach described above. But this does not reduce the computation workload for rendering. Hence, we extend the original algorithm of Lindow et al. [LBH12] by a hierarchical ray casting. If, during the grid traversal, a grid cell covers only a single pixel, we can omit the ray-sphere intersection tests, because it is impossible to distinguish individual spheres anyway. In this case, our algorithm only makes one texture lookup to determine whether the grid cell is empty or not. If the grid cell is not empty, this will be considered as a hit and the grid traversal will be stopped. This is only valid because the grid cells have roughly the same size as an atom. Therefore, the probability of an atom being hit by a ray traversing the corresponding grid cell is high. The result is basically a binary voxelization of the data set. The same approach can be used for whole molecules if the bounding box of the molecule covers only one pixel on the image plane. This technique is especially feasible for our simulation data sets where the cytoskeleton is composed of very large molecules and the signal proteins are quite small in comparison. Note that this has no effect on the deferred shading, since we only use approximated normals for distant objects. The approach also prevents molecules from being invisible because they would be smaller than one pixel and not be hit by the view rays.

The size of the pixel is calculated for the depth at the end of the current view ray. With the aid of the intercept theorem the pixel size is obtained from the field of view, the viewport height in pixels, and the corresponding depth. Since no intersection tests are performed when the pixel is larger than a grid cell, a special coloring scheme has to be employed. We propose to use the color ID of the first entry of the cell. This has the advantage, that coloring according to chain, strand, or instance ID is still possible and also visible. Other possible color schemes could comprise of a precomputed averaged color per grid cells or just a single color predefined per molecule instance.

5.2. Shading and Postprocessing

We use deferred shading [ST90] for the final image generation. One of the most important features of deferred shading is that it limits lighting computations only to visible fragments. This it not only advantageous because it reduces the computational load, it also allows us to use various post-processing techniques in image-space. Using the atomistic representation, the user can apply the common coloring

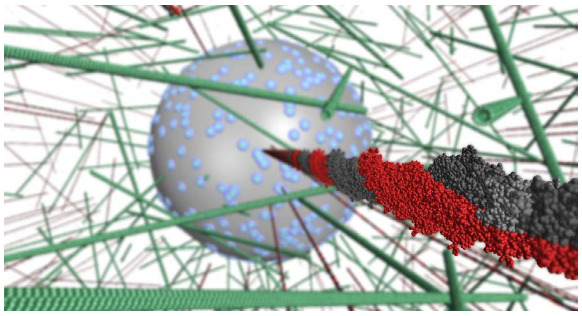

Figure 5: *Depth of field applied to a sparse test scene (10 billion atoms). The red actin filament (right) is in focus.*

schemes for molecular graphics. For the proteins, we implemented coloring by element, by molecular subunit or chain, and by instance ID. For the cytoskeleton, the user can additionally select coloring per strand.

We allow the user to choose between a variety of illustrative methods to enhance the final image. All methods have been chosen to facilitate the visual analysis of the renderings as well as for artistic reasons. Artistic renderings, which resemble a hand-drawn illustration, are often easier on the human perception. A prominent example for this are the Molecule of the Month renderings by the renowned illustrator David Goodsell [Goo] and the work on enhancing real-time molecular graphics by Tarini et al. [TCM06]. Likewise, we added the option to apply depth-dependent silhouettes [ST90]. Silhouettes are a simple, yet effective cue to emphasize spatial differences and to separate objects.

Depth of field is an effect from photography which separates foreground and background since only objects in the focal plane are sharp whereas everything out of focus is blurred (cf. Figures 1 & 5). In visualization, this can be used to draw the attention of the user to a specific region. The field in focus gets shallower as the focused object gets closer to the camera. Consequently, far away focal points lead to a broad field in focus. To obtain the depth of field effect, we create a mipmap chain of the color and depth buffer of the scene. The circle of confusion, which depends on the depth of the fragment, is then used to sample the mipmap chain.

The commonly used local lighting can be confusing and lead to artifacts when rendering a huge number of small spheres. Toon shading is a non-photorealistic rendering technique which shows discreet steps in the coloring instead of a smooth gradient. We allow the user to switch between local lighting, toon shading, and flat shading. Additionally, the user can use screen space ambient occlusion (SSAO) [Kaj09] to enhance depth perception. Due to the lack of lighting, flat shading offers no shape cues for the rendered spheres. However, in combination with the depth-dependent silhouettes and a toon-shaded variant of SSAO, flat shading results in a very clear and useful visual appearance (cf. Figure 6).

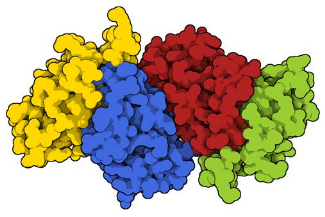

Figure 6: *Non-photorealistic rendering of a protein using flat shading, silhouettes, and toonified screen space ambient occlusion, colored according to the chain (PDB-ID: 3PPJ).*

6. Results

We integrated the atomistic rendering method in our visualization program which is used by our project partners to analyze the results of the mesoscopic simulations. A screenshot of the application is shown in Figure 7. The user can load the simulation data sets into the program and select different views to analyze the results. The main view in Figure 7 shows the atomistic visualization with the cytoskeleton, proteins, and the nucleus. The user can also select graph views like the one bottom right which show for example the radial concentration of a certain kind of protein. The application was designed in collaboration with the users in order to maximize the usability and user-friendliness and the serviceability for the intended visual analysis.

We measured the performance of our visualization using data sets from our mesoscopic simulation. The simulation setup and the resulting data set are described in section 6.1. The results of our measurements are given in section 6.2.

6.1. ERK Pathway

The mitogen-activated extracellular-signal-regulated kinase (ERK) pathway is responsible for cell growth, proliferation, and differentiation. Growth factors initiate the signal cascade from Raf over MEK to ERK. The ERK pathway is modeled in our system as follows. The upstream part of the pathway, B-Raf (PDB-ID: 3PPJ), is located near the plasma membrane and always active. MEK1 (PDB-ID: 1S9J) on the second tier is activated by B-Raf via double-phosphorylation, i.e. adding two phosphate groups. The activated form of MEK1 activates the final protein, which is ERK1 (PDB-ID: 1ERK). Activated proteins might be deactivated by phosphatases, here MKP3 (PDB-ID: 1MKP), by removing the phosphate groups.

The diameter of the plasma membrane was set to $10\,\mu$m and the diameter of the nucleus in the center to $4\,\mu$m. Table 1 lists the components of our simulated cell. About 250,000 proteins are present in the beginning and the total amount keeps almost constant over the course of time. The simulation of 7 s with a step width of $\Delta t = 50\,\mu$s took about two hours on an NVIDIA GeForce GTX 560. The mesoscopic simulation approximates molecules by spheres. Hence, ro-

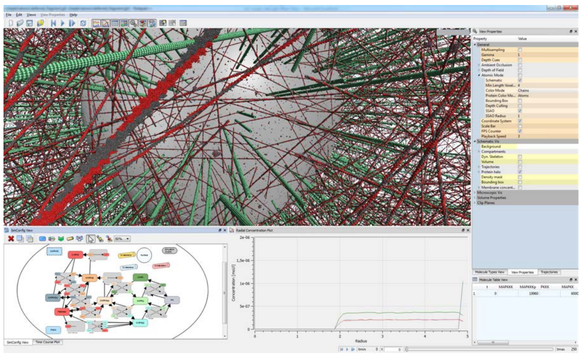
Figure 7: *The visualization application used for our work. The framework supports different data views which can be freely arranged. The top view shows the atomistic visualization of the simulation data, the lower left view shows the cellular simulation setup, and the bottom right is a graph view. The parameters are listed in the panel to the right. Users can create additional views to show other data values or visualization styles.*

tational diffusion is not modeled explicitly, but instead incorporated into the reaction rates. To account for the random rotation of proteins in the visualization, we mimic rotational diffusion by applying random rotations onto each molecule when the respective time step is loaded. We account for the rotational diffusion by biasing the rotation to the direction of movement. That is, we apply the random rotation together with a rotation that aligns the protein to the direction of movement. The molecule position and rotation between two time steps is computed using linear interpolation. Figure 1 shows the ERK pathway simulation data set.

6.2. Rendering Performance

We measured the performance of our method using various data sets from real molecular dynamics simulations. Our test system was an Intel Core i7 920 (4×2.6 GHz) with 6 GiB RAM and a NVIDIA GeForce GTX 580 (1.5 GiB RAM). We used a grid cell size of 4 Å. For the depth culling, the number of proteins per iteration was set to 4,096, which turned out to give the best results. The view resolution was 1920×1200. We adjusted the camera so that most of the screen was covered by the cell and zoomed in so that the proteins in the foreground were not rejected by hierarchical ray casting. As with all ray casting methods, the rendering performance is highly depending on the camera adjustment and the number and size of objects in the scene. Therefore, we will not conduct a formal comparison with the performance values given by Lindow et al. [LBH12] but rather compare our optimizations to the original algorithm.

For the huge data set containing more than 25 billion atoms, we measured 1.8 fps without any optimizations. With our custom depth culling enabled, the frame rate was 2.2 fps which is about 20 % faster. With the hierarchical ray casting and no depth culling, we measured 3.3 fps (about 80 % faster). With hierarchical ray casting and depth culling, the frame rate was at 3.6 fps, which is twice as high as the non-optimized version. Zooming further into the scene similar to Figure 1 (right) led to an increase in the rendering performance to about 8 fps. The rendering of the whole cell, as depicted in the center of Figure 1, was possible with 3.6 fps at a height of 1200 pixels when enabling all optimizations. The data set was rendered at 10.3 fps when covering only half the viewport height and 21.8 fps at a third of the height. We executed the same test using various other simulation data sets which were different in size but used the same building blocks as the one described in section 6.1. In all our measurements, we observed similar speedups for our optimizations.

Table 1: *Parameters of the components of the ERK model. The letter* B *denotes billions* (10^9).

Type	# Elems.	Length [μm]	# Instances long	short	# Atoms
Actin	7,500	9,873	62,051	3,622	14.34 B
Tubules	800	1,303	7,954	378	10.44 B
ERK	246,000	—	—	—	0.59 B
Total			25,421,804,076 = 25.42 B		

7. Future Work

So far, our simulations do not include information about the rotation of the proteins. However, we apply a biased rotation for rendering which is modeling the rotational diffusion as explained in section 6.1. In the future, we want to add a more refined stochastic model of the rotational diffusion to the underlying simulation which is developed in collaboration with our project partners.

The rendering method we used for our work is restricted to static molecules. As future work, we want to investigate the feasibility of this algorithm for rendering partly dynamic models which are composed rigid molecules. One application could be myosin filaments, where only the head domain is moving, or partly rigid MD simulations of viral envelopes, where the capsid proteins are internally rigid but moving.

Another promising direction for future developments would be the usage of tight-fitting bounding geometry. Especially for the filaments and tubules, the currently used bounding box contains many empty cells. Using cylindrical bounding boxes could lead to higher performance while requiring an only slightly more elaborate grid traversal strategy. Other possible bounding geometries would, for example, include spheres, object-aligned bounding boxes or roughly approximated convex hulls.

GPU ray casting can not only useful for rendering implicit surfaces but also applicable for rendering large numbers of triangulated objects [GRZ*10]. In the future, we would like to investigate how the method described in this paper could be combined with mesh-based geometry. This would enable rendering not only simple sphere-based molecular models but also arbitrary objects like complex, precomputed molecular surfaces [CG07, KSES12].

8. Summary and Conclusion

We presented a method to visualize mesoscopic whole-cell simulations in atomic detail. Our rendering method is an extension of the recent work by Lindow et al. [LBH12]. With our optimized algorithm, it is possible to render the intracellular environment with several billions of atoms at interactive frame rates on current graphics hardware. We modeled the simulated interior of the cell using readily available standard data sets from the Protein Data Bank [PDB].

Our application allows the user to apply several methods for image enhancement methods like screen space ambient occlusion [Kaj09] and illustrative, non-photorealistic rendering techniques. These methods can be used to facilitate shape and depth perception. They allow to interactively create visually pleasing, artistic images similar to [Goo], which can be used for instance in publications or as educational resources.

The visualization is integrated into a visualization application that is used by our collaborators from the field of systems biology. We presented our results to these project partners and got positive feedback. Amongst other things, they had the impression that this detailed visualization creates a better understanding of cellular dimensions and complexity than simplified representations. In the future, we would like to conduct a formal user study to confirm these anecdotal, preliminary statements.

Acknowledgments

This work was partially funded by the German Research Foundation (DFG) within the Cluster of Excellence in Simulation Technology (EXC 310/1) and as part of the Collaborative Research Centre SFB 716 at the University of Stuttgart.

References

[AW87] Amanatides J., Woo A.: A fast voxel traversal algorithm for ray tracing. In *EuroGraphics* (1987), pp. 3–10. 4

[CG07] Cipriano G., Gleicher M.: Molecular surface abstraction. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1608–1615. 8

[FKRE09] Falk M., Klann M., Reuss M., Ertl T.: Visualization of signal transduction processes in the crowded environment of the cell. In *IEEE Pacific Visualization Symposium (PacificVis '09)* (2009), pp. 169–176. 3

[FOE*11] Falk M., Ott M., Ertl T., Klann M., Koeppl H.: Parallelized agent-based simulation on CPU and graphics hardware for spatial and stochastic models in biology. In *International Conference on Computational Methods in Systems Biology (CMSB 2011)* (2011), pp. 73–82. 3

[Goo] Goodsell D.: Molecule of the month. `http://www.rcsb.org/pdb/motm.do` (Online, Aug. 2012). 6, 8

[GRDE10] Grottel S., Reina G., Dachsbacher C., Ertl T.: Coherent culling and shading for large molecular dynamics visualization. *Computer Graphics Forum 29* (2010), 953–962. 2, 4

[GRE09] Grottel S., Reina G., Ertl T.: Optimized data transfer for time-dependent, GPU-based glyphs. In *IEEE Pacific Visualization Symposium* (2009), pp. 65–72. 2

[GRZ*10] Grottel S., Reina G., Zauner T., Hilfer R., Ertl T.: Particle-based rendering for porous media. In *Annual SIGRAD Conference* (2010), pp. 45–51. 8

[Gum03] Gumhold S.: Splatting illuminated ellipsoids with depth correction. In *International Workshop on Vision, Modeling, and Visualization* (2003), pp. 245–252. 2

[HPGK90] Holmes K. C., Popp D., Gebhard W., Kabsch W.: Atomic model of the actin filament. *Nature 347*, 6288 (1990), 44–49. 3

[HSS*05] Hadwiger M., Sigg C., Scharsach H., Bühler K., Gross M. H.: Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum* (2005), 303–312. 2

[Kaj09] Kajalin V.: Screen-space ambient occlusion. In *ShaderX^7*. Charles River Media, 2009, pp. 413–424. 6, 8

[KSES12] Krone M., Stone J. E., Ertl T., Schulten K.: Fast visualization of gaussian density surfaces for molecular dynamics and particle system trajectories. In *EuroVis Short Papers* (2012), vol. 1, pp. 67–71. 8

[LBH12] Lindow N., Baum D., Hege H.-C.: Interactive rendering of materials and biological structures on atomic and nanoscopic scale. *Computer Graphics Forum 31*, 3 (2012), 1325–1334. 2, 3, 4, 5, 7, 8

[LBK*07] Lodish H., Berk A., Kaiser C. A., Krieger M., Scott M. P., Bretscher A., Ploegh H., Matsudaira P.: *Molecular Cell Biology*, sixth ed. W. H. Freeman, 2007. 3

[LD08] Lagae A., Dutré P.: Compact, fast and robust grids for ray tracing. *CFG 27*, 4 (2008), 1235–1244. 4

[LHA01] Law C. C., Henderson A., Ahrens J.: An application architecture for large data visualization: A case study. In *IEEE Symposium on Parallel and Large-data Visualization and Graphics* (2001), pp. 125–128. 2

[LVRH07] Lampe O. D., Viola I., Reuter N., Hauser H.: Two-level approach to efficient visualization of protein dynamics. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1616–1623. 2

[Pal00] Palsson B.: The challenges of in silico biology. *Nature Biotechnology 18*, 11 (2000), 1147–1150. 3

[PDB] Protein Data Bank. `http://www.pdb.org`. 3, 8

[PSQH06] Pogson M., Smallwood R., Qwarnstrom E., Holcombe M.: Formal agent-based modelling of intracellular chemical interactions. *Biosystems 85*, 1 (2006), 37–45. 3

[ST90] Saito T., Takahashi T.: Comprehensible rendering of 3-d shapes. *Computer Graphics (Proceedings of SIGGRAPH 1990) 24*, 4 (1990), 197–206. 5, 6

[TCM06] Tarini M., Cignoni P., Montani C.: Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1237–1244. 6