Eurographics Workshop on Visual Computing for Biology and Medicine (2014)
I. Viola, K. Bühler, and T. Ropinski (Editors)

# Membrane Mapping: Combining Mesoscopic and Molecular Cell Visualization

Thomas Waltemate[1], Björn Sommer[2], Mario Botsch[1]

[1]Computer Graphics Group, Bielefeld University
[2]Bioinformatics / Medical Informatics Group, Bielefeld University

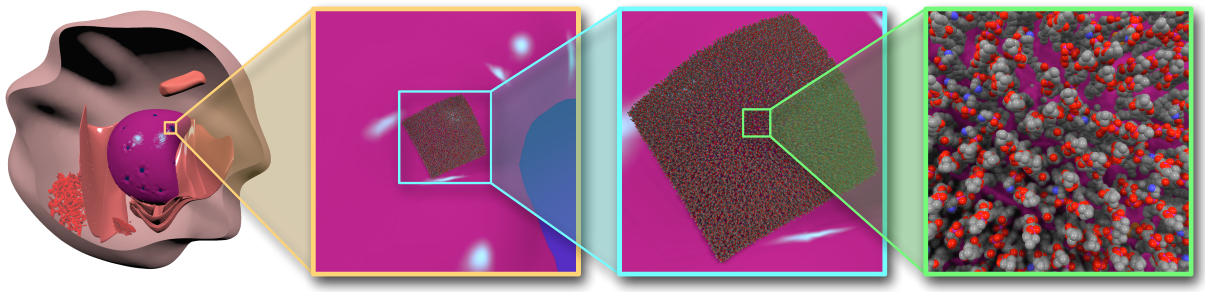

**Figure 1:** *Membrane mapping is an interactive tool that bridges the gap between cell visualization on the mesoscopic (left) and the molecular scale (right). It provides the user with an interactive magnifier tool that maps the pre-computed molecular structure and dynamic behavior of membrane patches to local surface regions of cell components.*

**Abstract**

*Three-dimensional cell visualization is an important topic in today's cytology-affiliated community. Cell illustrations and animations are used for scientific as well as for educational purposes. Unfortunately, there exist only few tools to support the cell modeling process on a molecular level. A major problem is the immense intracellular size variation between relatively large mesoscopic cell components and small molecular membrane patches. This makes both modeling and visualization of whole cells a challenging task.*
*In this paper we propose* Membrane Mapping *as an interactive tool for combining the mesoscopic and molecular level. Based on instantly computed local parameterizations we map patches of molecular membrane structures onto user-selected regions of cell components. By designing an efficient and GPU-friendly mapping technique, our approach allows to visualize and map pre-computed molecular dynamics simulations of membrane patches to mesoscopic structures in real-time. This enables the visualization of whole cells on a mesoscopic level with an interactive magnifier tool for inspecting their molecular structure and dynamic behavior.*

## 1. Introduction

The visualization of biological cells is a common cytological topic which is not only relevant for educational purposes, but also for the analysis and verification of *in silico* experiments. Recently, especially the three-dimensional visualization started to open new perspectives in this field. More generally, two cytological levels can be differentiated: the *mesoscopic* and the *molecular* level.

The mesoscopic level represents cellular structures ranging from micrometer down to nanometer domains. Cell components may be differentiated, but it is not possible to recognize molecular structures. The mesoscopic level is often visualized for educational purposes, e.g., in cell animations, school books, or games. For example, approaches like "Meta!Blast" provide school students a playful access to cytology [WBD*10]. Famous cell animations like "The Inner

DOI: 10.2312/vcbm.20141187

Life of the Cell", commissioned by Harvard University in 2007, transport cytological knowledge in a visually appealing way to a broad audience. Various efforts exist to extend the amount of educational as well as scientific visualization in the cytological area [Lok11]. Scientific visualizations at the mesoscopic level are especially important to explore intracellular relationships, e.g., protein-protein interaction networks [WPL*09], metabolic pathways [SKS*10], or disease-related networks [SKD*13]. Moreover, even simulation environments based on differential equations like VCell provide the integration of three-dimensional cell models for visualization purposes [LS01].

Even more established in the scientific context is the visualization at the molecular level, usually operating at a scale of a few Ångström. Molecular simulation environments like GROMACS [HKvdSL08], AMBER [CCD*05], or CHARMM [BBMJ*09] are employed for simulations of transport processes, membrane behavior, or formation of larger molecular structures like vesicles. These simulation packages usually do not provide visualization tools, they are optimized for the simulation of molecular structures, which usually have to be supported by large CPU or GPU clusters. The visualization of the results is done by external tools, such as VMD [HDS96], PyMOL [DeL02], or QuteMol [TCM06]. These toolchains, however, are not able to simulate or visualize the atomistic behavior of a complete cell at the mesoscopic scale. Established molecular simulation packages like GROMACS cannot simulate large structures containing billions of atoms, because the computation of the physicochemical properties and interactions is extremely complex. Some recent GPU-accelerated approaches achieve whole-cell simulations, but they restrict themselves to coarse-grained simulations [FOE*11] and visualizations [FKE13], where larger molecular groups are kept rigid. All-atom simulations are still restricted to smaller areas of the cell, such as a ribosome, a small vesicle, or membrane patches containing a protein surrounded by a bilayer with a known lipid composition.

In this paper we propose an interactive tool for whole cell visualization that efficiently combines rendering on the mesoscopic and the molecular scale. Since atom-level simulation and visualization of whole cells is out of reach, some kind of compromise has to be done. Instead of assuming static data sets or piecewise rigid simulations, we chose a different trade-off: Our method enables the visualization of detailed MD simulations on an atomistic level, however not for the whole cell but only for a small region of interest. Our membrane mapping technique allows the user to zoom in onto certain cell components and to map pre-computed (static and dynamic) molecular representations of membrane patches onto the surface of existing three-dimensional models. In an educational context our method can be seen as an "atom-scale magnifier glass" for *locally* inspecting the molecular structure and its dynamic behavior of certain cell component surfaces, as depicted in Figure 1.

To the best of our knowledge our approach is the first to interactively combine these two highly specific cytological visualization techniques. So for the first time, users are able to combine (i) simulations of a well-established molecular simulation program (GROMACS) with (ii) customized three-dimensional models of mesoscopic cell structures.

## 2. Related Work

While both mesoscopic visualization and molecule rendering have been explored to a larger extent, bridging the gap between mesoscopic and atomistic scale is the target of only a few modeling and visualization approaches.

One possible approach is the simplification of molecular structures with the objective to hide the complex atomistic representation. A number of approaches is following this simplification paradigm. For instance, cellPACK is a plug-in compatible with different 3D modeling tools, which can be applied to the packing of shapes derived from molecular structures in a style known, e.g., from illustrations of David S. Goodsell [JGA*14]. Therefore, although cellPACK's algorithms are not exclusively restricted to shape-based computations, its current visualization approaches usually show simplifications of the original atomic structures.

The BioBlender project tries to simplify the correlation between PDB-based structures and shape-based models [ACZ*12]. It can be used to visualize, e.g., the electrostatic potential of a PDB file. However, the visualization of larger atomic structures is very restricted with BioBlender. The Graphite LifeExplorer project develops different standalone tools with the objective to simplify the cell modeling process [HLLF13]. The CELLmicrocosmos MembraneEditor can be used to generate membrane models for downstream molecular dynamics simulations using GROMACS [SDG*11]. However, this tool also follows the simplification paradigm by using rather abstract shapes to enable the visualization of larger structures like vesicles.

The aforementioned approaches are not intended to visualize large numbers of atoms at a time. There are, however, already several approaches that are able to interactively visualize several millions of atoms using GPU-accelerated splatting of spheres and cylinders. These methods render each atom or connection by generating an enclosing screen-space rectangle and ray casting the respective glyph for each pixel of the bounding rectangle [Gum03, TL04, RE05, SWBG06, TCM06, GRE09]. These approaches mainly differ in how they generate the bounding rectangles, where Grottel et al. [GRE09] provide detailed performance benchmarks of the different techniques.

The above methods, while being highly efficient, are still not capable of visualizing the multi-billion atom representation of a whole cell. Exploiting instancing of larger groups of molecules, pre-computed spatial data structures, and GPU-based ray marching, the approach of Lindow et al. [LBH12] is capable of rendering up to several billion atoms at interactive rates. This method was adapted by Falk et al. [FKE13] for the atomistic visualization of mesoscopic whole-cell simulations. Both methods, however, achieve their impressive rendering performance by restricting to either static datasets or by grouping larger molecules to rigid pieces. As a consequence, they cannot visualize dynamic simulations on the atomic level, as required by our proposed approach. We therefore base our rendering on the splatting method of Sigg et al. [SWBG06], which does not require any precomputation of spatial data structures and hence can render dynamic animations on an atomic level.

Since we map the pre-computed 3D molecular structure of a membrane patch onto a user-selected region of a cell component, our approach is conceptually similar to shell maps [PBFJ05], where a 3D geometry is mapped from a 3D texture domain to the so-called *shell space* enveloping a surface triangle mesh. This shell space can be thought of as a "thick surface" that is bounded by two offset surfaces in positive and negative normal direction, respectively. The shell space is first decomposed into prisms and then tessellated into tetrahedra, which allows a piecewise linear barycentric mapping from texture space to shell space. But this piecewise linear mapping has the drawback of (i) not fitting very well to a GPU implementation and (ii) possibly distorting molecules that cross tetrahedra boundaries.

Our membrane mapping approach, described in the next section, avoids these limitations by using either a rigid transformation for each molecule (e.g., a lipid or protein), or a smooth polynomial mapping for a whole membrane patch.

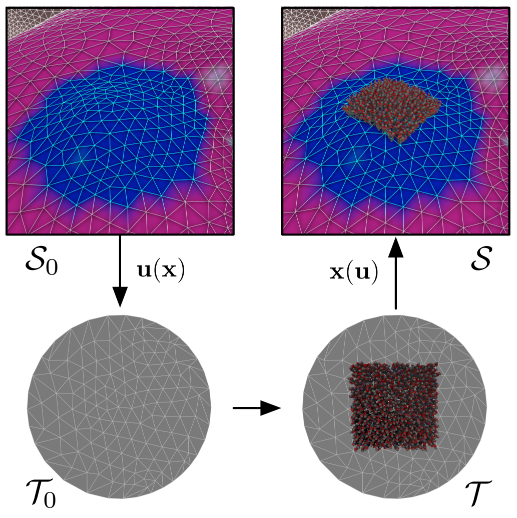

**Figure 2:** *The surface patch $\mathcal{S}_0$ (blue) is mapped to the 2D texture domain $\mathcal{T}_0$, which is extruded to the 3D domain $\mathcal{T}$ into which the membrane patch is embedded. The texture domain $\mathcal{T}$ and thereby the membrane patch are mapped to the shell space $\mathcal{S}$ surrounding the original surface $\mathcal{S}_0$.*

## 3. Membrane Mapping

The interaction metaphor for the membrane mapping approach is as follows: The user zooms in onto a cell component and selects a center point on the component's surface, around which the membrane should be mapped. Starting from this center point, a local geodesic patch of sufficient radius is grown, which is then mapped to a 2D texture domain using a planar parameterization (Section 3.1). The resulting piecewise linear mapping from 2D texture domain to the surface patch is extended to the 3D shell space around the surface using either our molecule instancing for static structures (Section 3.2) or a smooth polynomial space warp for mapping dynamic simulations (Section 3.3).

Our approach expects the following input data:

- The mesoscopic level is represented by *cell component models*. These models are three-dimensional triangle meshes directly or indirectly derived from microscopic images or image stacks. To allow a local parameterization, the meshes should be (locally) two-manifold and of sufficient element quality.
- The molecular level is represented by *membrane models*. These models are three-dimensional, rectangular membrane patches, which are assemblies of molecules (lipids and/or proteins).
- Dynamic *molecular simulations* of these membrane models are represented by different atom positions for each frame of the simulation.

The membrane models represent a "thick layer", where we assume the $x$- and $y$-coordinates to correspond to the 2D texture coordinates, and the $z$-coordinate to represent the height or offset from the center of the membrane bilayer, which should be centered at the $z = 0$ plane. Consequently, the lipids are approximately aligned with the $z$-axis.

In the following we will denote the surface patch (onto which the membrane is to be mapped) by $\mathcal{S}_0$ and the corresponding 2D texture domain by $\mathcal{T}_0$. The two-manifold / two-dimensional surfaces $\mathcal{S}_0$ and $\mathcal{T}_0$ represent the membrane's center surface in 3D world coordinates and 2D texture coordinates. When extended to thick surfaces in normal direction / height-direction they are denoted as shell space $\mathcal{S}$ and texture space $\mathcal{T}$. We will write shell-space coordinates by $\mathbf{x} = (x,y,z)^\mathsf{T}$ and texture space coordinates by $\mathbf{u} = (u,v,w)^\mathsf{T}$. See Figure 2 for a depiction of these spaces and mappings.

### 3.1. Local Parameterization

The user input for the membrane mapping process is a center point $\mathbf{c} \in \mathcal{S}_0$ around which to center the membrane and the desired patch size $d \times d$, which we translate into a $k \times k$ tiling of the pre-computed membrane.

Using the method of Kimmel and Sethian [KS98] we start a breadth-first Dijkstra-like traversal from $\mathbf{c}$ up to a geodesic radius $r$. This radius is chosen large enough such that the geodesic disk covers the surface region to be mapped onto, which is $r = 2d$ in our case. The resulting (triangulated) geodesic disk is the surface patch $\mathcal{S}_0$.

We naturally choose a 2D disk with radius $r$ as the texture domain $\mathcal{T}_0$. The mapping $\mathbf{u}\colon \mathcal{S}_0 \to \mathcal{T}_0$ is computed by a discrete harmonic parameterization, which amounts to mapping the boundary vertices/edges of $\mathcal{S}_0$ homeomorphically to the 2D circle $\partial\mathcal{T}_0$ and then solving a linear system to minimize the Dirichlet energy [BKP*10]. We employ the mean value weights, which in combination with the convexity of $\mathcal{T}_0$ guarantees a bijective mapping.

Since the surface patch $\mathcal{S}_0$ is a geodesic disk (up to mesh discretization), which is mapped to a planar disk, the mapping $\mathbf{u}(\mathbf{x})$ can be expected to contain only a small amount of distortion. This distortion is typically concentrated at the boundary of the domain, which is why the radius $r$ is chosen slightly larger than necessary, since this further reduces distortion for the region containing the membrane patch. Note that Figure 2 does not include this additional scaling in order to provide a clearer visualization.

### 3.2. Molecule Instancing

Similar to displacement maps the parameterization of the center surface $\mathcal{S}_0$ can be extended in normal direction, allowing to map each single atom position $\mathbf{u} \in \mathcal{T}$ from texture space to shell space $\mathcal{S}$. To this end we project $\mathbf{u} = (u,v,w)^{\mathsf{T}}$ to the membrane's center surface $\mathcal{T}_0$ (i.e., the $w = 0$ plane), which is the base point $\mathbf{u}_0 = (u,v,0)^{\mathsf{T}}$. This point can be mapped to the center surface $\mathcal{S}_0$ by the inverse parameterization $\mathbf{u}^{-1}$, i.e., by simple barycentric mapping: Let $\mathbf{u}_1,\mathbf{u}_2,\mathbf{u}_3$ be the vertices of the triangle in $\mathcal{T}_0$ containing the base point $\mathbf{u}_0$, and let $\alpha_1,\alpha_2,\alpha_3$ be its barycentric coordinates, i.e., $\mathbf{u}_0 = \alpha_1\mathbf{u}_1 + \alpha_2\mathbf{u}_2 + \alpha_3\mathbf{u}_3$. Then the corresponding point $\mathbf{x}_0$ on the surface $\mathcal{S}_0$ is

$$\mathbf{x}_0 = \alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \alpha_3\mathbf{x}_3,$$

with $\mathbf{x}_i = \mathbf{u}^{-1}(\mathbf{u}_i)$. Its normal vector $\mathbf{n}(\mathbf{x}_0)$ is obtained by the (normalized) barycentric interpolation of the $\mathbf{x}_i$'s normal vectors $\mathbf{n}_i$. The final mapped atom position $\mathbf{x} \in \mathcal{S}$ is then

$$\mathbf{x} = \mathbf{x}_0 + w \cdot \mathbf{n}(\mathbf{x}_0),$$

where $w$ is the offset/height of the atom center from the membrane center surface in texture space.

While this straightforward method is easy to implement, it has two important drawbacks. First, as soon as $\mathcal{S}_0$ is moderately curved, the normal vectors on $\mathcal{S}$ spread apart, which distorts the mapped molecules by changing their intra-molecule atom distances. Second, when mapping $k \times k$ membrane patches, this method has to store all $k^2$ mapped positions for each atom in the membrane model.

We address both problems by grouping together smaller parts of the membrane, namely lipids and proteins, because they usually are the smallest connected entities of a membrane. We refer to these small groups of atoms as *molecules*. Note that, in contrast to [LBH12] and [FKE13], these molecules are rather small, containing about 30–50 atoms only. We circumvent the distortion of intra-molecule distances by mapping each molecule by an individual rigid motion. To this end, we map the center of mass $\bar{\mathbf{u}} \in \mathcal{T}$ of each molecule using the technique described above, resulting in the mapped center $\bar{\mathbf{x}} \in \mathcal{S}$ and thereby defining the translation component $\mathbf{t}$ of the rigid motion. The rotation $\mathbf{R}$ is derived such that it aligns the $w$-axis in texture space with the normal vector $\mathbf{n}(\bar{\mathbf{x}}_0)$ of the mapped base point $\bar{\mathbf{x}}_0$ of the molecule center. This results in a homogeneous transformation

$$\mathbf{M}(\mathbf{x}) = \mathbf{R}\mathbf{x} + \mathbf{t} \tag{1}$$

to be stored for each mapped instance of each molecule.

This mapping approach concentrates all distortion to the empty space between individual molecules, i.e., it trades zero intra-molecule distortion for higher inter-molecule distortion, which visually is more pleasing. It also requires less memory compared to mapping individual atoms. However, when mapping dynamic molecular simulations, this approach requires to store a transformation matrix for each instance of each molecule and each time step of the simulation, because besides new atom positions each time step must have its own set of instance matrices. As a consequence, the molecule instancing technique is not suitable for mapping complex simulations and large membrane patches. This is addressed in the following section.

### 3.3. Polynomial Mapping

In our particular setting, where relatively small membrane patches are mapped onto relatively large cell components, the size of a membrane patch typically is small compared to the local radius of curvature of the target surface patch $\mathcal{S}_0$. Geometrically, this means that the surface patch $\mathcal{S}_0$ can locally be represented at a sufficient accuracy using a local second-order Taylor approximation, or a constant-curvature osculating quadric. As a consequence, it can be expected that the mapping $\mathbf{x}\colon \mathcal{T} \to \mathcal{S}$ can be approximated sufficiently well by a quadratic polynomial, which has the form

$$\begin{aligned}\mathbf{x}(u,v,w) = {} & \mathbf{c}_0 + \mathbf{c}_1 u + \mathbf{c}_2 v + \mathbf{c}_3 w \\ & + \mathbf{c}_4 u^2 + \mathbf{c}_5 v^2 + \mathbf{c}_6 w^2 + \mathbf{c}_7 uv + \mathbf{c}_8 uw + \mathbf{c}_9 vw,\end{aligned} \tag{2}$$

where the $\mathbf{c}_j \in \mathbb{R}^3$ are the degrees of freedom to be determined. To this end we generate a set of mapping constraints $\mathbf{u}_i \mapsto \mathbf{x}_i$, $i = 1,\ldots,m$, and fit the quadratic polynomial to satisfy them in the least squares sense.

The mapping constraints are generated by sampling $m$ points $\mathbf{u}_i$ that are distributed uniformly in the bounding box of the membrane patch in texture space $\mathcal{T}$ as illustrated in

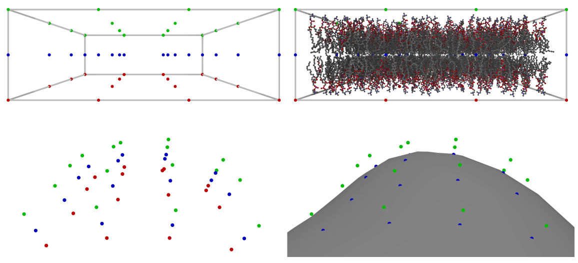

**Figure 3:** *Top: Sampled constraint points inside a membrane patch with and without showing the membrane structure. Bottom: Constraints mapped to shell space, with and without showing the surface of a cell component. Green/blue/red depict sample points at $w = +\varepsilon/0/-\varepsilon$.*

Figure 3, top. We use three layers of $6 \times 6$ sample points: One at the center of the bilayer ($w = 0$) and two offset layers in positive ($w = +\varepsilon$) and negative ($w = -\varepsilon$) $w$-direction. The target positions $\mathbf{x}_i$ in shell space $\mathcal{S}$ are computed using the atom displacement mapping described in the previous subsection (Figure 3, bottom).

Finding the coefficients $\mathbf{c}_j$ such that the polynomial best fits the constraints $\mathbf{u}_i \mapsto \mathbf{x}_i$ amounts to setting up an overdetermined $m \times 10$ linear system $\mathbf{AC} = \mathbf{X}$ with

$$\mathbf{A} = \begin{bmatrix} 1 & u_1 & v_1 & w_1 & u_1^2 & v_1^2 & w_1^2 & u_1 v_1 & u_1 w_1 & v_1 w_1 \\ \vdots & & & \vdots & & & \vdots & & & \vdots \\ 1 & u_m & v_m & w_m & u_m^2 & v_m^2 & w_m^2 & u_m v_m & u_m w_m & v_m w_m \end{bmatrix},$$

$\mathbf{C} = (\mathbf{c}_1, \ldots, \mathbf{c}_{10})^\mathsf{T} \in \mathbb{R}^{10\times 3}$ and $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)^\mathsf{T} \in \mathbb{R}^{m\times 3}$. This system is then solved for the polynomial's coefficients $\mathbf{C}$ in the least squares sense based on Cholesky factorization of its normal equations [GL89]

$$\mathbf{A}^\mathsf{T}\mathbf{A}\,\mathbf{C} \;=\; \mathbf{A}^\mathsf{T}\mathbf{X}. \tag{3}$$

Computing the local parameterization, generating the constraint samples, and fitting the polynomial can be computed instantly (see the accompanying video). The atom mapping from texture space $\mathcal{T}$ to shell space $\mathcal{S}$ can easily be computed in a vertex shader by evaluating Equation (2). Note that the polynomial is constant for a whole membrane patch (in contrast to the per-molecule mapping in the previous section). Furthermore, since this mapping does not depend on individual atom positions, it can be used to map all frames of a dynamic simulation. As a consequence, this method has a very small memory footprint, allows for whole-membrane render batches, and works seamlessly for static or dynamic membranes.

When mapping larger tilings of $k \times k$ patches we simply use one quadratic polynomial for each membrane patch. When computing these local polynomial maps we apply the above fitting procedure to each patch individually. While this does not guarantee the patches to match smoothly at their boundaries, in practice this is usually not a problem, since even for larger tilings the respective surface regions are rather smooth and moderately curved. An alternative would be to employ higher order trivariante spline mappings, which we leave for future work.

## 4. Rendering Techniques

Our rendering framework visualizes the mesoscopic level using cell components represented as triangle meshes, where we use standard OpenGL Core Profile functionality for rendering. The individual triangle meshes are stored in static vertex buffer objects, lighting is performed using standard Phong shading.

On the atomistic scale we cannot use a voxelized ray marching technique as in [LBH12] or [FKE13], since these have to pre-compute local grids for their static/rigid groups of atoms. Our atomistic visualization of dynamic molecular simulations requires to animate individual atom position for each frame. We therefore employ a splatting-based technique [SWBG06] for local ray casting of spheres and cylinders, which allows to visualize space filling as well as ball-and-stick models.

When molecule instancing is used as mapping technique, the vertex shader transforms the atom position from texture space to shell space using a per-molecule transformation matrix $\mathbf{M}$ (see (1)). For polynomial mapping, the vertex shader simply evaluates the quadratic polynomial $\mathbf{x}(\mathbf{u})$ (see (2)) to transform atom positions or cylinder endpoints.

Our MD simulation data typically consists of updated atom-coordinates for every frame. Since color and radius are constant values over all frames, only the atom coordinates have to be updated each frame. Our splatting-based rendering requires only one vertex per atom and two vertices per atom-connection. This leads to a relatively small amount of data that has to be transferred from main memory to GPU memory for each frame.

The results of a molecular dynamics simulation often have a low temporal resolution, such that the movements of single atoms are hard to track. To reduce this effect, we refine the temporal sampling by applying univariate subdivision to each atom's trajectory. Linear subdivision/interpolation yields easier-to-track atom movements, but the missing smoothness of the resulting piecewise linear trajectories is rather disturbing. We instead *smoothly* interpolate the individual atom positions by applying a few steps of 4-point subdivision [DLG87]: The position of atom $i$ at time $t+\frac{1}{2}$, to be inserted between frames $t$ and $t+1$, is computed as

$$\mathbf{x}_{i,t+\frac{1}{2}} \;=\; \frac{1}{16}\left(-\mathbf{x}_{i,t-1} + 9\mathbf{x}_{i,t} + 9\mathbf{x}_{i,t+1} - \mathbf{x}_{i,t+2}\right).$$

The resulting trajectory is a smooth high-resolution curve that interpolates the initial key-frames of the MD simulation, as shown in the accompanying video.

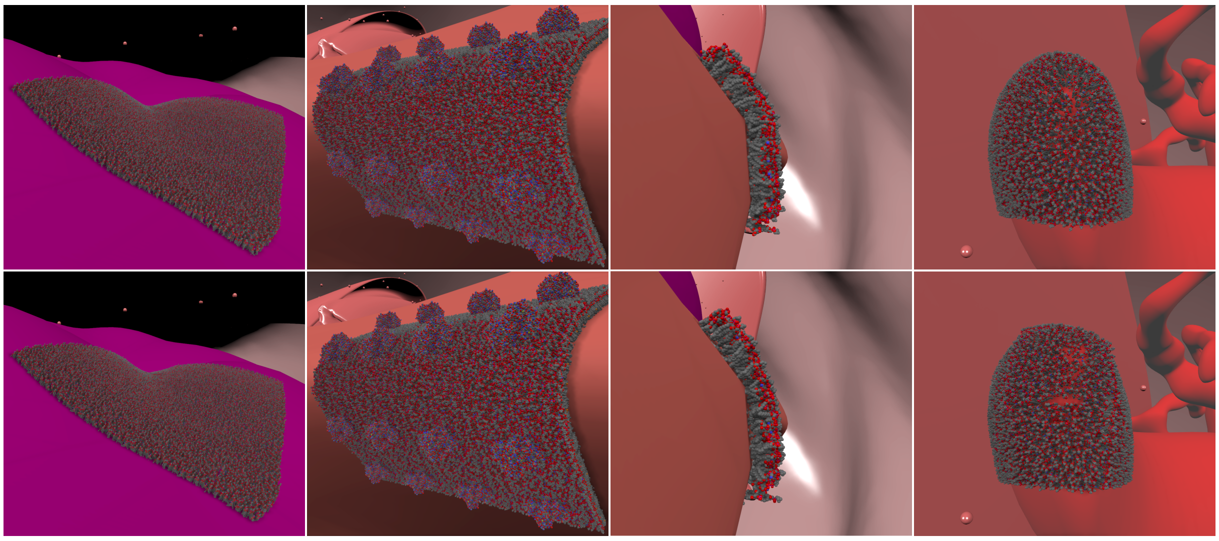

**Figure 4:** *Membrane Mapping examples using the Molecule Instancing method (top row) and the Polynomial Mapping technique (bottom). From left to right: Membrane models 1, 4, 2, and 3.*

In terms of lighting and shading, it is well known that simple Phong lighting of mesoscopic cell components and atomistic membranes does not properly reveal the 3D structure, leading to a rather confusing visualization, where different parts of a molecule are often hard to distinguish. Shadows naturally enhance the depth perception of a rendered scene. Hence, we add two shadowing techniques to our rendering framework: Shadow Mapping for direct shadows and Screen Space Ambient Occlusion (SSAO) [Kaj09] as an approximation to global illumination. Especially SSAO enhances the depth perception very well, while not requiring object space pre-computation. As a consequence it is very well suited for our real-time visualizations of molecular simulations.

## 5. Data

This section describes the data used for the visualization experiments shown in the paper and the accompanying video.

**Cell Model:** The cell component models were constructed based on light microscopic and electron microscopic images of mammalian liver hepatocyte cells. For the modeling process, 3ds max® and Blender were used. The coloring scheme follows the Haematoxylin and Eosin staining. Generated cell components are amongst others: nucleus, rough and smooth endoplasmic reticulum, golgi apparatus, mitochondrion, vesicles, and cell membrane. The models are represented as surface triangle meshes and have been optimized using adaptive isotropic remeshing [DVBB13] and Loop subdivision [Loo87].

**Membrane Model 1, 2 and 3:** Rat Liver Bilayer Models. Several rat hepatocyte-related membranes were constructed using the above methods, representing the normalized lipid composition of different cell components based on [Jai88]. Here, membrane compositions for the nucleus (1), mitochondrion (2) and the endoplasmic reticulum (3) were used.

**Membrane Model 4:** Rat Liver Bilayer with Protein. A bilayer containing different phospholipids and cholesterol was modeled with the MembraneEditor [SDG*11]. The model has a size of 20nm $\times$ 20nm and contains the semi-automatically aligned protein 1O5W.

**Membrane Model 5:** Lipid Membrane Simulation. A DPPC bilayer membrane with a size of 18 nm $\times$ 18 nm containing 1024 lipids was constructed. Then, a molecular dynamics simulation was performed with GROMACS [HKvdSL08]. A standard protocol was applied; the system was solvated in about 3400 water molecules and the Gromos96 force field (ffG45a3) was used. The simulation was run for 1 ns, resulting in 102 frames. During importing the membrane simulation into the software discussed here, water molecules are completely removed. Please see the accompanying video for results.

## 6. Results

In this section we compare the two different mapping techniques—Molecule Instancing and Polynomial Mapping—both in terms of visual quality (Figure 4), memory consumption, and rendering performance (Table 1). The accompanying video demonstrates the complete user interaction and shows the time refinement of simulation data through subdivision of atom trajectories.

| Input Model | | | | Molecule Instancing (fps) | | | | Polynomial Mapping (fps) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| type | patch size | #atoms | #connections | SF | SF+AO | BS | BS+AO | SF | SF+AO | BS | BS+AO |
| | 1×1 | 46k | 46k | 122 | 60 | 139 | 64 | 133 | 63 | 149 | 66 |
| | 2×2 | 184k | 183k | 111 | 58 | 116 | 59 | 111 | 57 | 118 | 59 |
| | 3×3 | 414k | 413k | 91 | 51 | 90 | 51 | 99 | 54 | 95 | 53 |
| Static | 4×4 | 736k | 734k | 82 | 48 | 72 | 45 | 94 | 52 | 86 | 50 |
| | 5×5 | 1M | 1M | 78 | 47 | 47 | 42 | 85 | 49 | 73 | 45 |
| | 10×10 | 4.6M | 4.6M | 23 | 23 | 12 | 12 | 46 | 35 | 33 | 25 |
| | 15×15 | 10.4M | 10.3M | 10 | 10 | 5 | 5 | 28 | 23 | 18 | 15 |
| | 1×1 | 51k | 50k | 124 | 61 | 142 | 64 | 111 | 58 | 132 | 62 |
| | 2×2 | 205k | 200k | 104 | 56 | 111 | 57 | 103 | 56 | 112 | 57 |
| | 3×3 | 461k | 452k | 92 | 52 | 90 | 51 | 97 | 53 | 96 | 53 |
| Animated | 4×4 | 819k | 803k | 86 | 50 | 64 | 46 | 87 | 50 | 81 | 48 |
| | 5×5 | 1.3M | 1.3M | 74 | 46 | 41 | 40 | 78 | 47 | 69 | 43 |
| | 10×10 | 5.1M | 5M | 22 | 22 | 11 | 11 | 41 | 32 | 29 | 23 |
| | 15×15 | 11.5M | 11.3M | 10 | 10 | 5 | 5 | 22 | 20 | 14 | 13 |

**Table 1:** *Performance comparison for different models and rendering techniques and static/dynamic membranes. BS: ball-and-stick model; SF: space-filling model; AO: ambient occlusion*

The visual quality of our membrane mapping approach is shown in Figures 1 and 4, where the latter compares the two mappings on several examples. Molecule Instancing allows for a more precise mapping and offers a better alignment of individual molecules to the membrane's center surface. Moreover, since each molecule is rigidly mapped as an individual entity, there is no intra-molecular distortion. However, this method suffers from a large memory consumption for storing the per-molecule instance matrices. This is particularly problematic for simulated membranes with many frames and the mapping of larger tilings, as each frame and each patch requires a separate set of instance matrices.

As can be observed in Figure 4, Polynomial Mapping provides visually similar results to Molecule Instancing in most cases. Only in surface regions with extreme local curvature the local per-patch polynomial mapping yields slightly inaccurate mappings and discontinuities at patch boundaries (Figure 4, right). These situations, however, are rather rare, such that in practice the piecewise polynomial mapping works sufficiently well. Since this method requires only one quadratic polynomial per membrane patch (both for static and dynamic membranes), it has a significantly smaller memory footprint. On the downside, the smooth polynomial mapping inevitably leads to a slight intra-molecular distortion, which was not disturbing in our experiments though.

Table 1 lists performance benchmarks for different patch sizes, different visualization techniques, as well as for static and animated membranes. All experiments were performed on a system with a NVIDIA GeForce GTX 770 GPU, an Intel Core i5 Quad-Core CPU, and 8 GB of RAM. The cell model used in the benchmarks consists of about 2M triangles. The performance numbers for Molecule Instancing and Polynomial Mapping are similar for patch tilings up to $4 \times 4$. After that the frame rate for Molecule Instancing is significantly lower (especially for ball-and-stick rendering), which is mainly due to the large number of OpenGL draw calls (one per molecule) and the costly memory transfer (one instance matrix per molecule). The Polynomial Mapping can make use of Uniformbuffers and requires just one draw call for all patches, therefore it does not show this performance drop. It consequently is the more efficient method, even though the computational cost of the vertex shader is slightly higher. Moreover, rendering animated membranes comes at almost no performance penalty compared to the static case.

The comparisons show that Molecule Instancing is the better option if intra-molecular distortion has to be avoided, whereas Polynomial Mapping is the preferred method for visualizing animated membranes or huge patch tilings.

## 7. Conclusion

This work introduced a new methodology for interactively combining the modeling and visualization of the mesoscopic and molecular level of a biological whole-cell model. Our Membrane Mapping combines a membrane patch based on molecular structures with the three-dimensional shape of a cell component. The main difference to previous approaches—in particular to Falk et al. [FKE13]—is that the Membrane Mapping technique is not restricted to the visualization of coarse-grained mesoscopic simulations. Instead, all-atom or united-atom membrane models are used, providing high resolution in both time and space.

In the future, our visualization could be improved by using smoothed normal fields for deferred shading [LBH12] and semi-transparent rendering of cell component surfaces. A spline-based mapping could avoid discontinuities across patch boundaries even in high curvature regions. In addition, a user study evaluating our visualization metaphor would be highly interesting. We also plan to integrate the Membrane Mapping method into the CELLmicrocosmos software.

**Acknowledgments**

This work was supported by the DFG Center of Excellence "Cognitive Interaction Technology" (CoE 277: CITEC).

## References

[ACZ*12] ANDREI R. M., CALLIERI M., ZINI M. F., LONI T., MARAZITI G., PAN M. C., ZOPPÈ M.: Intuitive representation of surface properties of biomolecules using BioBlender. *BMC Bioinformatics 13*, Suppl 4 (2012), S16.

[BBMJ*09] BROOKS B. R., BROOKS C. L., MACKERELL JR A. D., NILSSON L., PETRELLA R. J., ROUX B., WON Y., ARCHONTIS G., BARTELS C., BORESCH S.: CHARMM: the biomolecular simulation program. *Journal of Computational Chemistry 30*, 10 (2009), 1545–1614.

[BKP*10] BOTSCH M., KOBBELT L., PAULY M., ALLIEZ P., LÉVY B.: *Polygon Mesh Processing*. AK Peters, 2010.

[CCD*05] CASE D. A., CHEATHAM T. E., DARDEN T., GOHLKE H., LUO R., MERZ K. M., ONUFRIEV A., SIMMERLING C., WANG B., WOODS R. J.: The amber biomolecular simulation programs. *Journal of Computational Chemistry 26*, 16 (2005), 1668–1688.

[DeL02] DELANO W. L.: The PyMOL molecular graphics system. http://www.pymol.org, 2002.

[DLG87] DYN N., LEVIN D., GREGORY J. A.: A 4-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design 4*, 4 (1987), 257–268.

[DVBB13] DUNYACH M., VANDERHAEGHE D., BARTHE L., BOTSCH M.: Adaptive remeshing for real-time mesh deformation. In *Proceedings of Eurographics Short Papers* (2013).

[FKE13] FALK M., KRONE M., ERTL T.: Atomistic visualization of mesoscopic whole-cell simulations using ray-casted instancing. *Computer Graphics Forum 32*, 8 (2013), 195–206.

[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 *Proc. of Computational Methods in Systems Biology* (2011), pp. 73–82.

[GL89] GOLUB G. H., LOAN C. F. V.: *Matrix Computations*. Johns Hopkins University Press, Baltimore, 1989.

[GRE09] GROTTEL S., REINA G., ERTL T.: Optimized data transfer for time-dependent, GPU-based glyphs. In *Proceedings of IEEE Pacific Visualization Symposium* (2009), pp. 65–72.

[Gum03] GUMHOLD S.: Splatting illuminated ellipsoids with depth correction. In *Proceedings of Vision, Modeling and Visualization (VMV)* (2003), pp. 245–252.

[HDS96] HUMPHREY W., DALKE A., SCHULTEN K.: VMD: visual molecular dynamics. *Journal of Molecular Graphics 14*, 1 (1996), 33–38.

[HKvdSL08] HESS B., KUTZNER C., VAN DER SPOEL D., LINDAHL E.: Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *J. Chem. Theory Comput 4*, 3 (2008), 435–447.

[HLLF13] HORNUS S., LÉVY B., LARIVIÈRE D., FOURMENTIN E.: Easy DNA modeling and more with GraphiteLife-Explorer. *PloS one 8*, 1 (2013), e53609.

[Jai88] JAIN M. K.: *Introduction to biological membranes*, 2nd ed. Wiley-Interscience, New York, USA, 1988.

[JGA*14] JOHNSON G., GOODSELL D., AUTIN L., FORLI S., SANNER M., OLSON A.: 3D molecular models of whole HIV-1 virions generated with cellPACK. *Faraday Discussions 169* (2014), 1–21.

[Kaj09] KAJALIN V.: Screen space ambient occlusion. In *Shader X7*. Charles River Media, 2009, pp. 413–424.

[KS98] KIMMEL R., SETHIAN J. A.: Computing geodesic paths on manifolds. *Proc. Natl. Acad. Sci. USA 95* (1998), 8431–8435.

[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.

[Lok11] LOK C.: Biomedical illustration: From monsters to molecules. *Nature 477*, 7364 (2011), 359–361.

[Loo87] LOOP C.: *Smooth Subdivision Surfaces Based on Triangles*. Master's thesis, Department of Mathematics, University of Utah, 1987.

[LS01] LOEW L. M., SCHAFF J. C.: The virtual cell: a software environment for computational cell biology. *TRENDS in Biotechnology 19*, 10 (2001), 401–406.

[PBFJ05] PORUMBESCU S. D., BUDGE B., FENG L., JOY K. I.: Shell maps. *ACM Transactions on Graphics 24*, 3 (2005), 626–633.

[RE05] REINA G., ERTL T.: Hardware-accelerated glyphs for mono- and dipoles in molecular dynamics visualization. In *Proceedings of Eurographics/IEEE VGTC Conference on Visualization* (2005), pp. 177–182.

[SDG*11] SOMMER B., DINGERSEN T., GAMROTH C., SCHNEIDER S. E., RUBERT S., KRÜGER J., DIETZ K. J.: CELLmicrocosmos 2.2 MembraneEditor: a modular interactive shape-based software approach to solve heterogeneous membrane packing problems. *Journal of Chemical Information and Modeling 5*, 51 (2011), 1165–1182.

[SKD*13] SOMMER B., KORMEIER B., DEMENKOV P. S., ARRIGO P., HIPPE K., KOCHETOV A. V., IVANISENKO V. A., KOLCHANOV N. A., HOFESTÄDT R.: Subcellular localization charts: A new visual methodology for the semi-automatic localization of protein-related data sets. *Journal of Bioinformatics and Computational Biology 11*, 1 (2013), 1340005.

[SKS*10] SOMMER B., KÜNSEMÖLLER J., SAND N., HUSEMANN A., RUMMING M., KORMEIER B.: CELLmicrocosmos 4.1: an interactive approach to integrating spatially localized metabolic networks into a virtual 3D cell environment. In *Proc. of International Conference on Bioinformatics* (2010), pp. 90–95.

[SWBG06] SIGG C., WEYRICH T., BOTSCH M., GROSS M.: GPU-based ray-casting of quadratic surfaces. In *Eurographics Symposium on Point-Based Graphics* (2006), pp. 59–65.

[TCM06] TARINI M., CIGNONI P., MONTANI C.: Ambient occlusion and edge cueing to enhance real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 6 (2006), 1237–1244.

[TL04] TOLEDO R., LEVY B.: *Extending the graphic pipeline with new GPU-accelerated primitives*. Tech. rep., INRIA Lorraine, 2004.

[WBD*10] WURTELE E. S., BASSHAM D. C., DICKERSON J., KABALA D. J., SCHNELLER W., STENERSON M., VASANTH A.: Meta!Blast: a serious game to explore the complexities of structural and metabolic cell biology. In *Proceedings of the ASME 2010 World Conference on Innovative Virtual Reality* (2010), ASME, pp. 237–240.

[WPL*09] WIDJAJA Y. Y., PANG C. N. I., LI S. S., WILKINS M. R., LAMBERT T. D.: The interactorium: Visualising proteins, complexes and interaction networks in a virtual 3D cell. *Proteomics 9*, 23 (2009), 5309–5315.