HHS Public Access
Author manuscript
IEEE Trans Vis Comput Graph. Author manuscript; available in PMC 2019 January 01.

Published in final edited form as:
IEEE Trans Vis Comput Graph. 2018 January ; 24(1): 862–872. doi:10.1109/TVCG.2017.2744258.

Instant Construction and Visualization of Crowded Biological Environments

Tobias Klein,
TU Wien, Austria

Ludovic Autin,
The Scripps Research Institute, California, USA

Barbora Kozlíková,
Masaryk University, Brno, Czech Republic

David S. Goodsell,
The Scripps Research Institute, California, USA

Arthur Olson,
The Scripps Research Institute, California, USA

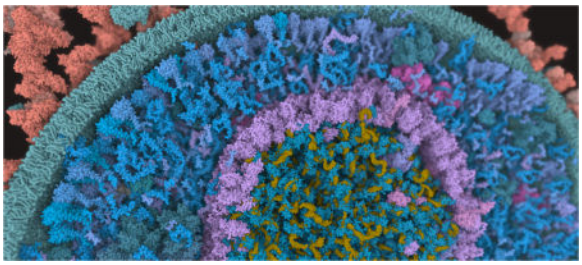
M. Eduard Gröller, and
TU Wien and VRVis Research Center, Austria

Ivan Viola
TU Wien, Austria

Abstract

We present the first approach to integrative structural modeling of the biological mesoscale within an interactive visual environment. These complex models can comprise up to millions of molecules with defined atomic structures, locations, and interactions. Their construction has previously been attempted only within a non-visual and non-interactive environment. Our solution unites the modeling and visualization aspect, enabling interactive construction of atomic resolution mesoscale models of large portions of a cell. We present a novel set of GPU algorithms that build the basis for the rapid construction of complex biological structures. These structures consist of multiple membrane-enclosed compartments including both soluble molecules and fibrous structures. The compartments are defined using volume voxelization of triangulated meshes. For membranes, we present an extension of the Wang Tile concept that populates the bilayer with individual lipids. Soluble molecules are populated within compartments distributed according to a Halton sequence. Fibrous structures, such as RNA or actin filaments, are created by self-avoiding random walks. Resulting overlaps of molecules are resolved by a forced-based system. Our approach opens new possibilities to the world of interactive construction of cellular compartments. We demonstrate its effectiveness by showcasing scenes of different scale and complexity that comprise blood plasma, mycoplasma, and HIV.

Graphical Abstract



Index Terms

Interactive modeling; population; biological data; interactive visualization

1 Introduction

Technological advances in structural biology, proteomics, and biophysics, combined with the rapid advance of computational capabilities, have opened the door to studying increasingly large and complex biological systems. Recently, we have seen a shift from studying individual proteins to modeling and analyzing functional protein assemblies and even larger systems, such as viruses, bacteria, and portions of eukaryotic cells. These are often denoted as biological mesoscale structures, representing an intermediate scale between molecular and cellular biology. On the molecular (nanoscale) level, cells are built of proteins, nucleic acids, lipids, and polysaccharides. The mesoscale level reveals how these molecules are assembled into more complex subcellular environments that orchestrate the processes of life. Given the complexity of these massive mesoscale environments, modeling must be tightly coupled with a proper visual representation. This provides intuitive feedback for their validation, tools for exploration of the models, and visual materials for public dissemination and to communicate findings to peers. In typical scenarios, these visualizations are created by scientific illustrators who traditionally produce images by hand drawing or by using 3D modeling tools. Examples include our illustrations of cellular environments [15], and a detailed 3D model of a synaptic bouton created in Maya [47]. This is a laborious process with limited interaction and exploration capabilities, which also requires non trivial domain knowledge about the depicted biological structures.

A more general solution is to describe the scene on a higher level of abstraction and use this description to automatically create the resulting scene. This approach has recently been implemented in cellPACK [24] to generate and visualize 3D models of complex biological environments. CellPACK integrates data from multiple sources into a ’recipe’ that describes the constitution of the desired environment. The environment is partitioned into membrane-bounded compartments, often arranged hierarchically. The compartments are defined by mesh surfaces that represent the lipid-bilayer membranes. Each compartment, including membranes, can be populated by a different set of ingredients, which include soluble molecules inside the compartment and molecules embedded in the membranes. These ingredients can be fibrous components, such as RNA or carbohydrate chains, soluble components such as proteins and metabolites, or lipids in membranes. CellPACK combines

the knowledge about membranes, compartments, and ingredients, and then assembles a mesoscale model with respect to mutual biologically-relevant interactions between them.

The main drawback of this approach is the computation time for assembling the 3D mesoscale model from the input data and the recipe. Depending on the desired quality of the distribution of ingredients, this stage spans from minutes to hours. Therefore, it is often a lengthy process to, for example, change parameters like the number of ingredients (molarity) or their interactions and observe the impact of the change. Our aim is to overcome this limitation and to instantly provide distributions of various ingredients within biological compartments and on membranes. This will open new possibilities for biologists to gather domain-specific knowledge about mesoscale-level structures, integrate it into a model, and explore the consequences of gaps in knowledge or ranges of experimentally-observed parameters.

Another irreplaceable part of the whole process is the visual representation of the scene and its content. Molecular visualization techniques have been developed in the last decades, which predominantly focus on visual depictions of molecules captured by X-ray crystallography, NMR spectroscopy, and more recently, electron microscopy. Due to the nature of this structural data, the molecules may be visualized at atomic resolution. However, the mesoscale level poses new challenges. Mesoscale models are larger, with millions of atoms, taxing existing visualization hardware and software. They are also more complex, often integrating multiple types of data at different resolutions, and with complex hierarchical relationships that span from individual molecules to entire cells. Visual representations must provide the user with an overview of the structures as well as with means to explore their details. To address this challenge, our state-of-the art tool cellVIEW [36] enables a seamless transition between visual representations of structures on different levels of detail. CellVIEW integrates the latest GPU-based algorithms from computer graphics and visualization to interactively render large biological scenes.

The approach described here incorporates instant modeling into cellVIEW to provide a unified visual framework for modeling and for visualization. The main contributions include:

- New algorithms for instant modeling of different types of compartments and membranes of biological structures on the mesoscale level.
- An interactive visual environment that allows users to change the population parameters and instantly explore the resulting scene on both nanoscale and mesoscale levels.
- Enhancing the process of creating models of mesoscale structures by shifting it from a non-visual and non-interactive environment to a fully interactive and visually-supported one.

We demonstrate the approach with mesoscale models of different scale and complexity, including blood plasma, HIV, and an entire mycoplasma bacterium.

2 Related Work

Experimental methods to determine mesoscale structures are extending to larger and larger subjects, but currently fall short of being able to determine the atomic structure of living cells. For instance, cryo-electron microscopy (EM) can give a detailed structure of the regular envelope of the Zika virus, but the more randomly arranged genome inside is still largely inaccessible to experiments [40]. Similarly, cryo-EM tomography is currently yielding detailed views of complex enveloped viruses, bacteria, and eukaryotic cells, but are typically able to resolve only large molecules such as ribosomes, cytoskeletal elements, and membranes [2].

Many groups are attempting the structural modeling of the mesoscale with atomic detail, using computation to combine information from mesoscale experimental techniques with atomic structural information on the components [22,38]. Much of the work in structural mesoscale modeling has focused on the soluble components of bacterial cytoplasm, using models to explore diffusion in crowded environments [35]. More recently, several groups are building tools for full structural models that include cellular infrastructure, including our own work with cellPACK [24] and LIFEEXPLORER [21,43].

Procedural Modeling

The mesoscale structure of cells is similar to other natural phenomena with high geometric and visual complexity as well as with requirements for explicit control of details. Other examples include the modeling of clouds and meadows covered by grass. The general approach to model these types of phenomena is to first define the overall shape of the environment, then fill it in with detailed objects using a stochastic process. This approach, called procedural modeling, has a long tradition in computer graphics, for example, for on-the-fly generation of infinite cities [16] or forests [7] within rich game environments. Currently the focus is rather on parallelization and scheduling optimizations, so that the procedural content generation is efficiently realized on graphics hardware [4,41]. Wonka et al. [49] presented a method for the automatic modeling of architecture. Recently, a novel technique was proposed for viewpoint-guided generation of high geometric details on facades [30]. These techniques typically use certain plans or grammars, which provide structural rules for the generation of shapes. In our work, we focus rather on the distribution of objects where the space is already constrained by membranes and divided into compartments.

Visualization of Large Biological Scenes

Driven by the wealth of structural information available through sources like the Protein Data Bank [3], highly effective software is currently available for the visualization of biomolecules and assemblies. For recent reviews, see Kozlikova et al. [29] and Johnson and Hertig [27]. These methods typically fail when faced with the large number of mesoscale molecules. Mesoscale structures require specialized, highly-optimized solutions. Currently, there are two publicly-available open-source systems where large numbers of atoms can be rendered efficiently by leveraging graphics hardware capabilities. One system is MegaMol [17], which is useful for various atomic representations, not only biomolecules. The other

one is cellVIEW [36], a tool for illustrative multiscale visualization of large biomolecular datasets. An alternative approach, published by Waltermate et al. [45], introduces an interactive tool combining mesoscopic and molecular scale visualization. They reach this by using a magnifier tool that enables the user to select a region on the cell membrane and map a pre-computed membrane patch with atomistic resolution onto it. This approach is able to populate only a very limited area of the cell surface.

Mesoscale scenes are also often produced by scientific illustrators using general-purpose 3D modeling tools. To ease their tasks, several extensions specific to molecular modeling have been introduced. For example, our ePMV plugin [25] allows users to run molecular modeling software directly inside of professional 3D animation applications. SketchBio [44] is another example of a 3D interface for molecular modeling and animation. Lv et al. [33] explored the possibility to use game technology in biomolecular visualization, using the Unity3D game engine to develop and prototype a molecular visualization application for subsequent use in research or education.

Texture Synthesis

Our approach to populate large biological membranes with lipids is based on a tiling technique commonly used in seamless 2D texture synthesis. We adapt Wang Tiles [46], which utilize square tiles to generate a plane tiling. The positioning of tiles on the plane is done according to their face-color rules to compose a desired pattern. Recently, several extensions of this basic approach have been published. Most of them were designed for synthesizing 2D textures that are mapped onto 3D surfaces. Fu and Leung [13] applied Wang Tiles to arbitrary topological surfaces. Our case differs in the necessity to synthesize tiles containing 3D objects, such as lipids. Therefore, the detection and handling of overlaps has to take these objects into account. Here we were inspired by the solution provided by Lipid-Wrapper [9]. LipidWrapper solves the collisions for each triangle on the 3D mesh, which makes it unsuitable for large membranes. Neyret and Cani [37] presented another strategy to tiling, using triangles with homogeneous textures to tile surfaces. Culik and Kari [5] introduced Wang Cubes, which extend the Wang Tiles approach and are able to create non-periodic illustrative 3D patterns and textures. Lu et al. [32] presented a framework for volume illustration utilizing Wang Cubes. They extend the original idea and modify it for multipurpose tiling. The Wang Cubes are not applicable in our case. Although we deal with 3D tiles representing the lipid bilayer, the tiles are still defined by a 2D plane. Fleischer et al. [12] presented an approach to texturing surfaces with so called cellular textures. These are textures represented by more complex geometry, which can cover the surface with small-scale features, such as feathers or thorns. This is very close to our situation with lipid bilayers. However, cellular textures cannot be applied in our case because of the regularity of the produced surfaces and the algorithm speed.

Population with Fibrous Structures

Algorithms for self-avoiding walks may be used to populate a compartment with a fibrous structure. Fan et al. [10] published an extension of a Manhattan lattice to generate fibrous structures by going from 2D to 3D space. In our approach we execute a procedural building of fibrous structures, which is similar to the polymerization process addressed by Kolesar et

al. [28]. Their illustrative approach is a fusion of three different modeling techniques, i.e., L-systems, agent-based systems, and systems of densities. Adding new building blocks to the existing polymer is based on attraction forces. Gruenert et al. [18] presented an approach to rule-based spatial modeling of chemical reaction systems. The technique is also suitable for simulating the polymerization process, showcased on the growth of filaments. The growth is driven by parameters, such as torsion and bending, that control the pathways and structures of formed complexes.

The problem of procedurally constructing paths is common in other areas as well. As an example, Galin et al. [14] presented an automatic method for generating roads between two given points, based on a weighted anisotropic shortest path algorithm. The result path minimizes a cost function influenced by different features of the scene, such as terrain shape and obstacles.

3 Overview

When modeling scenes containing complex mesoscale biological structures, the workflow consists of the following basic steps:

1. Scene organization, to define the shape of each membrane-bounded compartment.
2. Recipe definition, to determine the molecular composition of each compartment and membrane.
3. Population of the model, to fill the soluble spaces and membranes with ingredients defined in the recipe.

The organization of the scene is defined by closed 3D meshes that represent the lipid bilayer membranes. The membranes specify the separate compartments of the scene. For example, Figure 1 (a) shows a scene with two concentric closed membranes defining two intracellular environments (labeled 1 and 2) and a surrounding environment (labeled 3). The following step is to compile a recipe that defines the molecular composition of these compartments and membranes (Figure 1 (b)). This recipe includes the structures of the molecules, their concentration (molarity), and constraints related to their position in the membranes, specific interactions with other molecules, etc. The last step takes the recipe and populates the compartments and membranes accordingly (see Figure 1 (c)). This population step is typically stochastic, so it may be repeated to provide an ensemble of similar models that are each consistent with the scene organization and recipe.

With currently available approaches, such as cellPACK [24], the process of transforming the scene definition and recipe to the final model can take from minutes to hours, depending on the complexity of the scene. This makes the modeling process non-interactive as the user cannot immediately see the consequences of parameter changes. We overcome this by applying parallel processing, to improve computational bottlenecks of the workflow.

As can be seen in Figure 1 (b), the individual compartments and membranes can be populated in several ways. We can distinguish three basic cases:

- **Membranes** represent a semi-permeable barrier between two compartments. They often consist of a lipid bilayer with embedded proteins used in the communication and transport of molecules and ions across the membrane. In order to model membranes, we have implemented a solution based on texture tiling.
- **Soluble components** occupy the free space in each compartment. They consist of ingredients (molecules) of different shape and complexity, which influences the difficulty of the population step inside of each predefined space. Our approach for positioning ingredients in compartments consists of two steps. The first step populates the spaces with ingredients without considering overlaps between them. Collisions are resolved in the second step.
- **Fibrous structures** such as DNA, RNA, or actin filaments, are long strands folded inside specific compartments. We use a self-avoiding random walk to construct fibers in the allowable regions of the compartments.

Algorithms for populating spaces with each of these types of ingredients, and the advantages of populating them in a specific order, are described in detail in the following sections.

4 Membrane Population

Biological membranes are built around a continuous fluid sheet containing two opposed layers of lipids. Lipids are small molecules consisting of several dozen atoms, with a characteristic chemical character. They are composed of a head group that is soluble in water, and one or more tails that are insoluble. In water, they spontaneously associate to hide the insoluble tails, forming some of the largest structures in cells. These membranes form the semi-permeable boundary that surrounds cells, as well as the boundary of inner compartments such as the nucleus or mitochondria. Different types of lipids may be distributed unequally to create an asymmetry between the outer and inner surfaces, which is important for the cell function. For each biological structure, such as the HIV virion or the red blood cell, the bilayer has a unique lipid composition. The common property is that lipids are densely packed on the membrane and are oriented roughly perpendicularly to it.

The modeling of lipid bilayers is a complex and laborious task, because of the dense packing and the large number of lipids compared to other surrounding molecules. The main challenge, in terms of modeling, is to incorporate the known structural information and to avoid repetitive patterns on the surface. Several bilayer-modeling programs are available, such as CHARMM-GUI [23], Packmol [34], or cellPACK [24]. These tools position individual lipids one-by-one on the membrane, which leads to unacceptable computational costs. A more sophisticated solution is provided by LipidWrapper [9]. It extracts whole patches from a pre-equilibrated planar membrane model. These patches contain the detailed structural information about the lipids. In order to produce the membrane surface, LipidWrapper uses a triangulated mesh as input, where each triangle patch is randomly cut out from the provided membrane model. However, the edges of adjacent patches on the surface do not fit together and lipids potentially overlap in these regions. LipidWrapper deletes overlapping lipids and fills the resulting holes with new lipids. This is an expensive

process since it must be done for every triangle edge on the surface. Because of this, the computational cost of LipidWrapper is significant and not applicable to instant modeling.

For the interactive population of membranes, our solution uses a tiling approach from texture synthesis to cover the membrane mesh with lipids. In our approach, we require the mesh to be defined as a quad-based surface map. In general, any method can be used that resamples a given mesh into a quad-based surface if the resulting surface map is conformal and has low area distortions. An example can be PolyCubeMaps introduced by Tarini et al. [42]. The mesh generated by this approach is only marginally affected by distortions, which makes it suitable for our tiling approach.

To cover the mesh with lipids, we use only a small number of tiles. These tiles are pre-populated with lipids and here we use the basic principle of LipidWrapper. Our solution is significantly faster because we generate the content for only a limited set of tiles instead of applying it onto the whole mesh geometry. The tiles are subsequently used for covering the membrane. They have to be placed in such a way that their edges fit together and that the tiling will not result in periodic patterns. As the repetition pattern should not be visible, we use Wang Tiles [46].

In the following, we describe individual steps of our approach in detail. We start with the description of the concept and principle of Wang Tiles. Then we present our adaptation of this concept to membrane meshes.

4.1 Wang Tiles and their Extension to Membranes

The Wang tiling concept was introduced in the early ’60s and is well-known in texture synthesis. A Wang Tile is defined as a square with color-encoded edges. A set of Wang Tiles can be used to cover a 2D plane without periodic patterns. The colors of edges restrict how the tiles can be placed during the tiling process. The tiling is valid only if shared edges have the same color. At least four colors are required to tile a plane non-periodically.

There are several approaches that extend the concept of Wang Tiles to 3D space. In our solution, we follow the approach of Fu and Leun [13], which applies the tiling concept of Wang Tiles to arbitrary topological surfaces. They generate each tile from four different diamond-shaped input patches that are positioned west, east, north, and south around the tile center, respectively (Figure 2). These patches are combined to generate a set of tiles with different colors on the edges. We adapt this approach for use in membrane modeling by synthesizing a collection of lipid-textured tiles and then mapping them onto the membrane mesh.

4.1.1 Tile Synthesis for Lipid Bilayer Models—Synthesis of the tiles is the major challenge to generating 3D models of biological membranes. First, we have to extract the content for the tiles from an input texture. For biological membranes, this input texture is a planar representation of the lipid bilayer containing non-overlapping, tightly-packed lipid molecules. Atomic coordinates for the lipid bilayer can be generated, for example, by a molecular dynamics simulation using the Amber force field for lipids [8]. The tiles are generated in several steps. First, four small diamond-shaped patches are extracted from the

large bilayer texture. When extracting these patches (Figure 2 (a)), we include only lipids whose center is inside the bounds of the specified patch, where the center of a lipid is defined as the center of its bounding box. A large diamond-shaped patch is then created by positioning four of these tiles together. Finally, we extract a rectangular tile from the large diamond-shaped patch (see Figure 2 (b)). The different tiles needed for Wang Tiling are generated by different arrangements of the small patches into the large patches. The tile has to be further processed to resolve overlaps and fill gaps.

Detection of Overlapping Lipids: The assembly of a tile from multiple input patches inherently leads to overlapping lipids in areas where the input patches meet. Collisions are evaluated in a two step process to improve performance. First the overlap between bounding boxes is tested, then if necessary individual lipid atoms are checked. The lipid with the highest number of collisions is removed from the tile. If multiple lipids have the same number of collisions, one is chosen randomly for removal. The number of collisions is then updated for all neighbors of the removed lipid, and the procedure is repeated until all collisions are eliminated. This leaves the tile with undesirable holes (Figure 2 (c)), which are filled in the final step.

Hole Filling: The holes in a tile are filled by non-colliding lipids (Figure 2 (d)). Individual lipids are extracted from the same input membrane bilayer that was used to generate patches for tiles. For each side of the bilayer, a plane is defined that intersects the hydrophilic head groups of the lipids in the tile. Trial lipids are randomly chosen from the input texture and placed at regular intervals along the diagonals of the tile, aligning their head groups within the planes. If there are no collisions, the lipid is added to the tile, otherwise, the lipid is rotated around an axis perpendicular to the plane and tested for collisions. If no acceptable orientations are found, the process continues with another lipid from the input texture. Additionally, we reduce the number of possible candidates by rejecting those that already intersect with the lipids from the tile.

The result of the hole filling process depends on three parameters: how many lipids do we have in the testing set, how many rotations of lipids we test, and how many times we repeat the entire hole filling procedure. These parameters can be adjusted according to user needs. As we apply the process only on the tiles and not on every quad of the mesh, the parameter setting is not a critical aspect.

4.2 Tile Mapping

Finally, the generated tiles are mapped onto the membrane mesh consisting of quads. Since we are using data from a bilayer lipid membrane instead of textures, the 3D coordinates are mapped onto the quads instead of 2D texture coordinates. For this reason, we use quadrilateral coordinates to map lipid centers (λ, μ) to the quad with the vertices (v_0, v_1, v_2, v_3) through the following equation, where p is the resulting vertex in 3D space.

$$p=(1-\lambda)(1-\mu)\nu_0+\lambda(1-\mu)\nu_1+\lambda\mu\nu_2+(1-\lambda)\mu\nu_3 \quad (1)$$

At each resulting position, the corresponding lipid is instantiated. The orientation of the lipids on the quad has to be adjusted as well. As the main axis of the lipids is always perpendicular to the plane of the tile, we determine their orientation by computing the rotation angle between the original tile and its position on the quad. This rotation is applied to all lipids of a given tile. Figure 3 shows the result after positioning the tiles with lipids on the membrane mesh.

5 Soluble and Membrane-Bounded Components

Cells are typically crowded with proteins, nucleic acids, and other molecules. Soluble components and assemblies fill the compartments. Membrane-bounded components are embedded in the lipid bilayers. The recipe for a mesoscale scene specifies the quantity of these ingredients, their locations and orientations (if appropriate), and their mutual interactions. The task of population boils down to spatially distribute these ingredients in the corresponding space avoiding overlaps. Currently available techniques are capable of positioning soluble ingredients inside compartments, however, only in a sequential manner. This means that they position ingredients one by one. After positioning each ingredient the surrounding space is updated so it is aware of the distance to the closest ingredient. The next ingredient can only be positioned at locations with a sufficient distance to the nearby ingredients. The computational cost of the sequential approach is not acceptable for interactive environments. Therefore, our solution uses parallel processing to increase the performance.

To make the procedure applicable to GPU implementation, we divide our approach into three consecutive steps (see Figure 4):

1. Compartment space organization using voxelization.
2. Populating the space with the given ingredients.
3. Detecting and resolving collisions between ingredients.

The major advantage of this approach is that the processing of the individual ingredients becomes mostly independent from the processing of other ingredients. In the rest of this section we describe the steps in more detail.

5.1 Step 1 – Space Partitioning using Voxelization

The first step is to partition the scene into a set of compartments, delimited by membrane meshes. Our approach is based on the state-of-the-art method of GPU voxelization by Schwarz and Seidel [39]. The voxelization process starts with the classification of voxels that intersect with the compartments. In one of the grid axes, a scanline algorithm is applied to all voxels to determine if they belong to cellular compartments, membranes, or to the surrounding environment. We assign a negative value to cellular compartment voxels, a positive value to surface voxels, and zero to outside voxels. We repeat this process for all compartments in the scene. The resulting structure, which we call a **compartment grid**, identifies the compartment or membrane associated with each voxel, for the rapid population in the following step. Moreover, we can sort the voxels per compartment and estimate the

volume for each of them. Thus, given a molarity we can calculate the proper count for a specific protein.

In addition to the compartment grid, we define an **occupancy grid** that helps to resolve some of the overlaps between ingredients. The occupancy grid is of the same size and resolution as the compartment grid. It holds negative values for empty voxels or the corresponding ID of the ingredient if the voxel is occupied. This is especially important if the size of ingredients differs significantly, e.g., if surface ingredients protrude from the compartment surface. For this reason, we start the population process by distributing the large surface ingredients and then use the remaining space for the interior ones by employing the updated occupancy grid. This divides the population process into two passes, but prevents small ingredients from being completely overlapped by larger ingredients coming from a different compartment.

5.2 Step 2 – Population

In this step we populate a given compartment with individual ingredients defined in the recipe, using the compartment grid and occupancy grid generated in the previous step. Here we distinguish between two types of ingredients to be populated. The first type are ingredients that occupy the inner part of a compartment but are bounded to the compartment membrane. The second type of ingredients populates the soluble space of the compartment. In order to avoid additional overlaps, we first populate the surface of the compartment and then its inner part.

5.2.1 Population of Compartment Surface—Cell membranes typically include a diverse collection of membrane-bounded proteins, which may interact just on one side of the membrane or extend through the membrane. This poses several challenges: the proteins must be oriented correctly, must face the proper direction, and their membrane-spanning portions must be embedded in the membrane. These proteins often have large portions extending from one or both sides of the membrane, which must be evaluated for collisions with other membrane-bounded proteins and soluble components. In order to spatially distribute ingredients on a given membrane, we need the information about the ingredient type, its molarity, the ID of the membrane, and the principal vector and offset of the ingredients (Figure 5 (a)). Then we update the information stored in the occupancy grid (Figure 5 (b)).

The ingredients should be randomly distributed and should not overlap. To solve this issue, Willmott previously used the property of a Halton sequence to distribute in real time non-overlapping objects in a plane [48]. Similar to his approach, we use a Halton sequence to select positions on the mesh of the membrane to place the ingredients.

In general, a Halton sequence [19] provides point sets with low discrepancy, i.e., it produces well-spaced samples. The sequence is constructed using a prime number that defines the number of divisions of a unit interval into sub-intervals. These are subsequently divided using the same prime number until the desired length of the sequence is reached. The sequence is then ordered in such a way that it produces subsets that evenly cover the entire domain. Halton sequences are often used for numeric methods like Monte Carlo simulations [6]. Another advantage of a Halton sequence is that it never contains the same number twice

so that we do not choose the same position for multiple ingredients. This approach provides an efficient and rapid way of distributing ingredients on the surface of the membrane with sufficient randomness while minimizing potential overlaps.

The orientation of the ingredient is then described as the principal vector defining the orientation of the ingredient with respect to the membrane, and an offset vector that places the ingredient at the proper position relative to the surface of the membrane (Figure 6). This information is usually computed using the OPM webserver [31].

5.2.2 Population of Compartment Inner Area—For populating the soluble space of compartments, we use the information stored in the compartment grid and the occupancy grid to find appropriate positions for the ingredients. The membrane-bounded ingredients positioned in the previous step are considered as non-moving obstacles, i.e., they are handled as static objects when resolving collisions.

The population procedure works as follows. For each instance of the ingredient defined in the recipe, we place it at a voxel marked with the appropriate value in the compartment grid, and with an empty value in the occupancy grid. The occupancy grid is then updated with the identity of the ingredient, and these steps are repeated until all ingredients are positioned (Figure 5 (c), (d)). The occupancy grid is updated even after the positioning of all ingredients is completed, so that the grid may be used in populating this compartment with fibrous structures (see Section 6).

As the size of the ingredients can be substantially larger than the voxel size, this will ultimately result in intersecting ingredients if they are placed in a close proximity. The population step does not necessarily need to take overlap into consideration, although the less overlap is produced, the less computation is required. The following resolves the overlaps. A straightforward approach to avoid overlaps is to uniformly distribute ingredients. However, this leads to an undesired visual appearance since the grid structure becomes clearly visible. In order to introduce a certain degree of randomness and still reduce overlaps, we again utilize a Halton sequence to choose potential grid points for population. In practice a compartment populated using a Halton sequence can still suffer from visible regularities. Therefore, we additionally introduce jittering by a random vector in each grid position and we also apply a random rotation to each ingredient. With the Halton sequence, we can significantly reduce the number of overlapping molecules but it does not completely avoid them. The following step detects and resolves these remaining overlaps.

5.3 Step 3 – Detecting and Resolving Collisions

Ideally, we would want to detect collisions by evaluating contacts between all atoms in the molecules. However, this is not practical in terms of performance, so for every molecule type, we compute a two-level proxy geometry that approximates its shape. The higher-level proxy is simply the bounding sphere of the molecule. The lower-level proxy approximates the molecule with a small number of spheres calculated with a GPU-based K-means clustering algorithm [11]. As preset we use 16 spheres. The clustering provides the centers of the spheres and the radii which correspond to the cluster sizes. This ensures that all atoms are covered by the proxy geometry. The collision-detection process first uses the higher level

proxy geometry to detect potential collisions between two molecules, which are then verified with the lower-level proxy geometries using the finer approximation of the shape. Figure 7 shows the lower-level of the proxy geometries (a) and the scene populated by ingredients (b). The proxy geometry slightly overestimates the actual shape of the molecule to ensure that a certain distance is created between molecules after the collision is resolved. From a perceptual point of view, the molecules are easier to distinguish if their shape boundaries are not in direct contact.

The resolving of collisions is loosely based on standard rigid body dynamics [1]. Usually the exact point of contact between two bodies is detected and then corresponding forces are computed. Since our molecules are already in a colliding state, we simply compute forces that resolve these collisions. We use the information about collisions from the lower-level of the proxy geometry as a basis for the computation. For each intersecting pair of molecules we determine the overlapping sphere pairs and consider each as one collision. Each molecule accumulates a linear and angular force that is updated for every collision. The linear force is defined by a direction and strength. The direction is derived from the vector between the centers of the two spheres. The distance between the two sphere centers determines the strength of the force. As in rigid body dynamics, we compute the angular force by taking into account the vector between the center of the molecule and the collision, as well as the direction vector. After the forces for all collisions are computed, they determine the new position and rotation of molecules, as in a standard physics-based system. This defines one integration step. During the collision-resolving process, a molecule might temporarily leave its compartment. In this case we apply a force to the molecule, that will steer it back into the compartment. This process is repeated for a certain number of integration steps until all collisions are resolved or a stop criterion is reached. The criterion can be set by the user and specifies a certain number of acceptable remaining collisions. This makes the system flexible for less powerful hardware systems as well. Both the collision detection and the subsequent resolving is computed on the GPU.

6 Fibrous Structures

Fibrous ingredients are linear or branched polymers of repeated units, found in many places in cells. They include protein filaments that form infrastructure, polysaccharides that provide protection or store energy, and nucleic acids that encode genetic information. They are usually modeled using a procedural growing algorithm, such as self-avoiding random walks. Angle and length are constrained to maintain the persistent length of the processed fiber. The process of building the fibrous structure is sequential by nature as we have to be aware of the previous state of the fiber. However, this process can be greatly improved by using the compartment and occupancy grid information. For every incremental step of the walk, the corresponding grid element can be checked if it belongs to the compartment (compartment grid) and if it is occupied by some other protein (occupancy grid). This improves the computational cost by restraining the growing fiber to allowed space and minimizing intersections with previously positioned ingredients, including self-intersections. For large systems, such as genomes, the fiber can contain millions of subunits, so the complete fiber-growing process will not be interactive. Here, we provide the possibility for the user to observe the current progress of the fiber growth, to remain interactive.

The self-avoiding random walk in our approach works as follows. In the target compartment, a starting point is randomly chosen and the ensuing protocol is iterated until the required length of the fiber is reached. In each step, a random direction is selected from the hemisphere that is oriented and positioned according to the previous direction, as illustrated in Figure 8 (a). Possible intersections of the random walk with membranes, soluble molecules or itself are tested by using the occupancy and compartment grids. If there are no intersections, the segment is added and the grids are updated. Otherwise, a different random direction is chosen. A dead-end is reached if the walk cannot be continued after a given number of attempts. In our experience, the order in which molecules are distributed strongly influences the success of the population step. We have obtained the best results by placing membrane-bounded proteins first, followed by fibers, and finishing with soluble ingredients. This sequence addresses two significant potential problems: (i) membrane-bounded proteins often protrude in the compartments, so the fiber must avoid them, and (ii) typical soluble components are highly crowded into compartments, lead to few free grid points, and leave no options for fiber growth. Figure 8 (b) shows result of positioning DNA and RNA fibrous structures.

The method provides interactive performance for small fibers, such as the lipopolysaccharides extending from the mycoplasma surface or small RNA molecules in the cytoplasm. Long genomic DNA molecules, however, require several minutes to generate. We provide two methods to allow the user more interaction for these large structures: a premade DNA model may be preloaded, or we grow the walk in real-time to show the progress. This enables the user to stop and restart if the growing model shows problems.

7 Results and Discussion

The approach described here is flexible, allowing the user to model a variety of biological systems. This ability is due, in part, to the relationship of all living organisms on Earth. Similar membranes surround all cells and define their inner organelles, and similar proteins orchestrate the traffic of molecules and information across the membranes. Similar proteins and nucleic acids are used for the many metabolic and genetic tasks required for energy production, biosynthesis, and reproduction, which are packaged in very similar compartments within cells. Since viruses rely on cells for reproduction, they are also built from a subset of this machinery. With a limited set of structural modeling tools, we are able to build a variety of different cellular scenes.

Building mesoscale models is always an iterative process, where the recipe, ultrastructure and other parameters are tuned to reflect on and lend insight to the available experimental results. This is a laborious process with systems such as cellPACK, where single models require from minutes to hours to generate, followed by a significant effort to visualize the results. Our approach tightens this research cycle, allowing researchers to generate models or ensembles of models rapidly in an intrinsically visual environment. This allows the users to interactively optimize recipes, identify and correct any errors or bugs, and incorporate new data rapidly as it becomes available.

We demonstrate the capabilities and advantages of this new approach in three test systems. In a mesoscale model of blood plasma, the efficiency of the current method allows us to create a very large scene containing millions of molecules, which required a prohibitive computational effort with previous methods. With HIV, the interactive capability allows us to generate multiple models from a single recipe, and explore stochastic variations of a structurally complex subject. In a model of an entire mycoplasma bacterium, we are able to tune the recipe and modeling parameters interactively to achieve a particular goal: creating a 3D version of an existing 2D rendering of the cell. In the following sections, we describe these cases as well as the achieved results. The performance was measured using a computer with an Intel Core i7-6700K CPU 4.00 GHz and NVIDIA GeForce GTX 1080 graphics card with 8 GB memory.

Blood Plasma

Blood plasma is largely homogeneous and relatively sparsely occupied by proteins and short fibrous components. These proteins have a variety of functions, including molecules of the immune system, blood clotting factors, transport molecules, and hormones. It is a perfect system for early tests of methods for populating mesoscale scenes, because the proteins have a variety of sizes and shapes, including several with very large aspect ratios. When generating the model using cellPACK, a scene containing nearly 30,000 molecules was created in 116 seconds. With our approach we have tested a scene with blood plasma containing more than 33,000 molecules. The population step takes on average 100 ms and the consecutive overlap resolving takes on average 2000 ms. Our system is also capable of populating significantly larger scenes. As shown in Figure 9, we have populated a scene with a volume of two cubic microns with blood plasma. Given the exact molarity, our approach produces over 4.6 million soluble protein ingredients in 0.6 seconds. In our current implementation, this is beyond the capabilities of our force-based system to resolve potential overlaps. The main bottleneck here is the collision detection. However, this is not a conceptual limitation. Current research [20] has shown that it is already possible to interactively solve the collisions of millions of particles.

HIV

We have used HIV as a convenient test system for mesoscale modeling since the inception of the cellPACK project. It has several advantages: it is relatively small, but still taxes most methods and it is one of the most-studied organisms, so there is abundant information available. However, there are still significant gray areas in the available data, so this leaves room for experimentation in the parameters of the recipe.

The recipe used here was described by Johnson et al. [26], and comprises five types of surface proteins, 19 types of interior proteins and two RNA copies (9,200 bases, roughly 30,000 Ångströms long).

A membrane envelope surrounds the virion, with several copies of the viral envelope protein embedded in the membrane and extending outwards. Inside, a protein-enclosed capsid protects two copies of the viral genome. A variety of proteins, including the viral enzymes and several accessory proteins, are found in the space between the membrane and the capsid.

Figure 10 depicts the different steps of our procedural approach to model HIV. The final model consists of 11,790 protein ingredients. The entire model requires 0.07 seconds to generate without the RNA, and requires ~3 seconds to generate the two copies of the RNA. It takes up to an hour to generate this model using cellPACK.

Mycoplasma Bacterium

The ultimate goal of this work is to create computational models of entire cells, to interpret experimental results, and model structure/function relationships in healthy and diseased states. To address this challenge, we have focused on one of the simplest cells, a mycoplasma bacterium. This bacterium has a very simple structure, with a single cellular membrane and a small genome. The membrane is filled with pumps and channels, and decorated on the outer surface with lipopolysaccharides that form a protective coat around the cell. The detailed organization of molecules inside the cell is only now beginning to be explored by cryo-EM microscopy, given the small size of the cells. We have modeled a random tangle of DNA surrounded by the soluble protein and RNA components.

The draft recipe for the current model (Figure 11) includes the membrane, one strand of DNA (1,211,703 base pairs), 1500 lipopolysaccharides, and a collection of 18 of the most prevalent proteins and nucleic acids. The model required ~100 sec for the generation of all the fibers in the model (DNA, mRNA, peptides, lipopolysaccharides) and 0.1 sec to fill the remaining space with soluble components. Using cellPACK, the creation of the mycoplasma model takes several hours. We employed the interactive capabilities of the method to tune the recipe, based on previous research used to create a hand-drawn illustration of the cell. Figure 11 includes the illustration, a 3D model using the draft recipe, and a 3D model generated after interactively tuning the recipe to better match the illustration.

Performance and Interactive Capabilities

Creation of the presented types of models has previously been a laborious process, forming a bottleneck in their study. For instance, the micron-sized blood plasma scene previously required hours of computation time, and is modeled with the new method in less than a second. Currently, the method is able to populate soluble compartments interactively, allowing the user on-the-fly experimentation with recipes and testing their effects on the resultant scenes. The performance of mod-eling fibrous and membrane components has also been significantly improved when comparing with previous approaches. It takes seconds to generate a cell-sized membrane and about a minute to calculate a genome-sized object.

One of the great advantages of our presented methods is the ability to test hypotheses interactively. Biological research is an ongoing process, and there are many gray areas in the body of knowledge about the structure and function of cells at the mesoscale level. In many cases, it is useful to test the emergent mesoscale consequences of different properties at the molecular level. In the current implementation, many useful parameters are tunable at runtime. The user imports meshes that represent compartments, descriptions of ingredients, and then creates and visualizes the model based on the recipe on the fly. Tunable parameters in the recipe currently include the location, the molarity, the orientation (as defined by

principal vector and offset), and fiber-generation parameters. Once the basic geometry is loaded, everything may be changed interactively.

For example, in Figure 12, we tested two assumptions on DNA persistence length in a model of a small genome packed into a spherical bacterium. A longer persistence length leads to DNA strands that wrap around the periphery of the cell, whereas a shorter persistence length leads to a trajectory that is more reminiscent of a 3D random walk. The reality is probably a combination of these two assumptions, since DNA-binding proteins often form kinks that mimic the short persistence length, while free strands of DNA show a longer persistence length.

8 Conclusion and Future Work

In this work we have presented the first interactive approach to an integrative structural modeling of the biological mesocale. The interactive performance of the method provides several significant advantages in scientific and educational applications. With the possibility to create interactive models, domain experts are now able to tune parameters and explore the mesoscale properties that emerge. In addition it is now possible to create an ensemble of models from a single recipe, allowing researchers to explore the stochastic variations in mesoscale properties and compare them with experiments. In education, interactive capabilities can allow curriculum developers to better tune their models and representations to optimize the clarity of the resultant imagery. A long tradition in scientific illustration has proven the pedagogical utility of introducing small artistic changes to a subject to improve the interpretability of what is shown.

The next major enhancement of the method will be the incorporation of a faster and more intuitive approach to model interactions between the ingredients. For instance, modeling the interaction of ribosomes with messenger RNA to form a polysome, or crosslinking of DNA by DNA-binding proteins, is currently not feasible. We are currently exploring particle simulation methods such as NVidia Flex to add this capability. In addition, we will continue to optimize the generation of fibrous components, which is currently the rate-limiting step of the model generation with the new approach.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

This project has been funded by the Vienna Science and Technology Fund (WWTF) through project VRG11-010 and also supported by EC Marie Curie Career Integration Grant through project PCIG13-GA-2013-618680. The Scripps Research Institute researchers acknowledge support from NIH P41-GM103426 and R01-GM120604, and this manuscript has TSRI reference number 29494.

References

1. Baraff D. An introduction to physically based modeling: rigid body simulation I-unconstrained rigid body dynamics. SIGGRAPH Course Notes. 1997:D31–D68.

2. Beck M, Baumeister W. Cryo-electron tomography: Can it reveal the molecular sociology of cells in atomic detail? Trends in Cell Biology. 2016; 26(11):825–837. DOI: 10.1016/j.tcb.2016.08.006 [PubMed: 27671779] Special Issue: Future of Cell Biology
3. Berman HM, Westbrook J, Feng Z, Gilliland G, Bhat TN, Weissig H, Shindyalov IN, Bourne PE. The Protein Data Bank. Nucleic Acids Research. 2000; 28(1):235–242. [PubMed: 10592235]
4. Boechat P, Dokter M, Kenzel M, Seidel HP, Schmalstieg D, Steinberger M. Representing and scheduling procedural generation using operator graphs. ACM Transactions on Graphics. 2016; 35(6):183:1–183:12. DOI: 10.1145/2980179.2980227
5. Culik K, Kari J. An aperiodic set of Wang Cubes. Journal of Universal Computer Science. 1996; 1(10):675–686.
6. Dalal, IL., Stefan, D., Harwayne-Gidansky, J. Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on. IEEE; 2008. Low discrepancy sequences for monte carlo simulations on reconfigurable platforms; p. 108-113.
7. Decaudin, P., Neyret, F. Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques, EGSR'04. Eurographics Association; 2004. Rendering forest scenes in real-time; p. 93-102.
8. Dickson CJ, Madej BD, Skjevik AA, Betz RM, Teigen K, Gould IR, Walker RC. Lipid14: The Amber lipid force field. Journal of Chemical Theory and Computation. 2014; 10(2):865–879. DOI: 10.1021/ct4010307 [PubMed: 24803855]
9. Durrant JD, Amaro RE. LipidWrapper: an algorithm for generating large-scale membrane models of arbitrary geometry. PLOS Computational Biology. 2014; 10(7):e1003720. [PubMed: 25032790]
10. Fan K, Li XY, Wu DC. Self-avoiding walk on a three-dimensional Manhattan lattice. Journal of Physics A: Mathematical and General. 2000; 33(22):3971.
11. Farivar R, Rebolledo D, Chan E, Campbell RH. A Parallel Implementation of K-Means Clustering on GPUs. PDPTA'08 - The 2008 International Conference on Parallel and Distributed Processing Techniques and Applications. 2008:340–345.
12. Fleischer, KW., Laidlaw, DH., Currin, BL., Barr, AH. Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques. ACM; New York, NY, USA: 1995. Cellular texture generation; p. 239-248.
13. Fu CW, Leung MK. Texture tiling on arbitrary topological surfaces using wang tiles. Rendering Techniques. 2005:99–104.
14. Galin E, Peytavie A, Maréchal N, Guérin E. Procedural Generation of Roads. Computer Graphics Forum (Proceedings of Eurographics). 2010; 29(2):429–438.
15. Goodsell DS. Inside a living cell. Trends in Biochemical Sciences. 1991; 16:203–206. DOI: 10.1016/0968-0004(91)90083-8 [PubMed: 1891800]
16. Greuter, S., Parker, J., Stewart, N., Leach, G. Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, GRAPHITE '03. ACM; New York, NY, USA: 2003. Real-time procedural generation of 'pseudo infinite' cities; p. 87-ff
17. Grottel S, Krone M, Müller C, Reina G, Ertl T. MegaMol – a prototyping framework for particle-based visualization. Visualization and Computer Graphics, IEEE Transactions on. 2015; 21(2): 201–214. DOI: 10.1109/TVCG.2014.2350479
18. Gruenert G, Ibrahim B, Lenser T, Lohel M, Hinze T, Dittrich P. Rule-based spatial modeling with diffusing, geometrically constrained molecules. BMC Bioinformatics. 2010; 11(1):307.doi: 10.1186/1471-2105-11-307 [PubMed: 20529264]
19. Halton J. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. Numerische Mathematik. 1960; 2:84–90.
20. Hoetzlein R. Fast fixed-radius nearest neighbors: interactive million-particle fluids. GPU Technology Conference. 2014:18.
21. Hornus S, Lévy B, Lariviére D, Fourmentin E. Easy DNA modeling and more with GraphiteLifeExplorer. PLoS ONE. Jan; 2013 8(1):1–12. DOI: 10.1371/journal.pone.0053609
22. Im W, Liang J, Olson A, Zhou HX, Vajda S, Vakser IA. Challenges in structural approaches to cell modeling. Journal of Molecular Biology. 2016; 428(15):2943–2964. DOI: 10.1016/j.jmb.2016.05.024 [PubMed: 27255863]

23. Jo S, Lim JB, Klauda JB, Im W. CHARMM-GUI membrane builder for mixed bilayers and its application to yeast membranes. Biophysical Journal. 2009; 97(1):50–58. DOI: 10.1016/j.bpj.2009.04.013 [PubMed: 19580743]
24. Johnson GT, Autin L, Al-Alusi M, Goodsell DS, Sanner MF, Olson AJ. cellPACK: a virtual mesoscope to model and visualize structural systems biology. Nature methods. 2015; 12(1):85–91. [PubMed: 25437435]
25. Johnson GT, Autin L, Goodsell DS, Sanner MF, Olson AJ. ePMV embeds molecular modeling into professional animation software environments. Structure. 2011; 19(3):293–303. [PubMed: 21397181]
26. Johnson GT, Goodsell DS, Autin L, Forli S, Sanner MF, Olson AJ. 3D molecular models of whole HIV-1 virions generated with cellPACK. Faraday Discussions. 2014; 169:23–44. [PubMed: 25253262]
27. Johnson GT, Hertig S. A guide to the visual analysis and communication of biomolecular structural data. Nature Reviews Molecular Cell Biology. Oct; 2014 15(10):690–698. [PubMed: 25245078]
28. Kolesár I, Parulek J, Viola I, Bruckner S, Stavrum AK, Hauser H. Interactively illustrating polymerization using three-level model fusion. BMC Bioinformatics. 2014; 15(1):345.doi: 10.1186/1471-2105-15-345 [PubMed: 25315282]
29. Kozlíková B, Krone M, Falk M, Lindow N, Baaden M, Baum D, Viola I, Parulek J, Hege HC. Visualization of biomolecular structures: State of the art revisited. Computer Graphics Forum. 2016; doi: 10.1111/cgf.13072
30. Krecklau L, Born J, Kobbelt L. View-dependent realtime rendering of procedural facades with high geometric detail. Computer Graphics Forum. 2013; 32(2pt4):479–488. DOI: 10.1111/cgf.12068
31. Lomize MA, Lomize AL, Pogozheva ID, Mosberg HI. OPM: orientations of proteins in membranes database. Bioinformatics. 2006; 22(5):623–625. [PubMed: 16397007]
32. Lu A, Ebert DS, Qiao W, Kraus M, Mora B. Volume illustration using Wang Cubes. ACM Transactions on Graphics. 2007; 26(2):11.doi: 10.1145/1243980.1243985
33. Lv Z, Tek A, Da Silva F, Empereur-mot C, Chavent M, Baaden M. Game on, science - how video game technology may help biologists tackle visualization challenges. PLoS ONE. 2013; 8(3):e57990. [PubMed: 23483961]
34. Martínez L, Andrade R, Birgin EG, Martínez JM. PACKMOL: A package for building initial configurations for molecular dynamics simulations. Journal of Computational Chemistry. 2009; 30(13):2157–2164. DOI: 10.1002/jcc.21224 [PubMed: 19229944]
35. McGuffee SR, Elcock AH. Diffusion, crowding & protein stability in a dynamic molecular model of the bacterial cytoplasm. PLOS Computational Biology. 2010; 6(3):1–18. DOI: 10.1371/journal.pcbi.1000694
36. Muzic, ML., Autin, L., Parulek, J., Viola, I. Eurographics Workshop on Visual Computing for Biology and Medicine. EG Digital Library, The Eurographics Association; 2015. cellVIEW: a tool for illustrative and multi-scale rendering of large biomolecular datasets; p. 61-70.
37. Neyret, F., Cani, MP. Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '99. ACM Press/Addison-Wesley Publishing Co.; New York, NY, USA: 1999. Pattern-based texturing revisited; p. 235-242.
38. Reddy T, Sansom MS. Computational virology: From the inside out. Biochimica et Biophysica Acta (BBA) – Biomembranes. 2016; 1858(7, Part B):1610–1618. DOI: 10.1016/j.bbamem.2016.02.007 [PubMed: 26874202]
39. Schwarz M, Seidel HP. Fast parallel surface and solid voxelization on GPUs. ACM Transactions on Graphics. 2010; 29(6):179:1–179:10. DOI: 10.1145/1882261.1866201
40. Sirohi D, Chen Z, Sun L, Klose T, Pierson TC, Rossmann MG, Kuhn RJ. The 3.8 Å resolution cryo-EM structure of Zika virus. Science. 2016; 352(6284):467–470. DOI: 10.1126/science.aaf5316 [PubMed: 27033547]
41. Steinberger M, Kenzel M, Kainz B, Wonka P, Schmalstieg D. On-the-fly generation and rendering of infinite cities on the GPU. Computer Graphics Forum. 2014; 33(2):105–114. DOI: 10.1111/cgf.12315
42. Tarini M, Hormann K, Cignoni P, Montani C. Polycube-maps. ACM Transactions on Graphics. 2004; 23(3):853–860. DOI: 10.1145/1015706.1015810

43. Vendeville A, Lariviére D, Fourmentin E. An inventory of the bacterial macromolecular components and their spatial organization. FEMS Microbiology Reviews. 2011; 35(2):395–414. DOI: 10.1111/j.1574-6976.2010.00254.x [PubMed: 20969605]
44. Waldon SM, Thompson PM, Hahn PJ, Taylor RM. SketchBio: a scientist's 3D interface for molecular modeling and animation. BMC Bioinformatics. 2014; 15(1):334.doi: 10.1186/1471-2105-15-334 [PubMed: 25359079]
45. Waltemate, T., Sommer, B., Botsch, M. Eurographics Workshop on Visual Computing for Biology and Medicine. The Eurographics Association; 2014. Membrane Mapping: Combining Mesoscopic and Molecular Cell Visualization.
46. Wang H. Proving theorems by pattern recognition-II. Bell Labs Technical Journal. 1961; 40(1):1–41.
47. Wilhelm BG, Mandad S, Truckenbrodt S, Kröhnert K, Schäfer C, Rammner B, Koo SJ, Claßen GA, Krauss M, Haucke V, Urlaub H, Rizzoli SO. Composition of isolated synaptic boutons reveals the amounts of vesicle trafficking proteins. Science. 2014; 344(6187):1023–1028. DOI: 10.1126/science.1252884 [PubMed: 24876496]
48. Willmott A. Fast object distribution. In SIGGRAPH sketches. 2007:80.
49. Wonka P, Wimmer M, Sillion F, Ribarsky W. Instant architecture. ACM Transactions on Graphics. 2003; 22(3):669–677. DOI: 10.1145/882262.882324

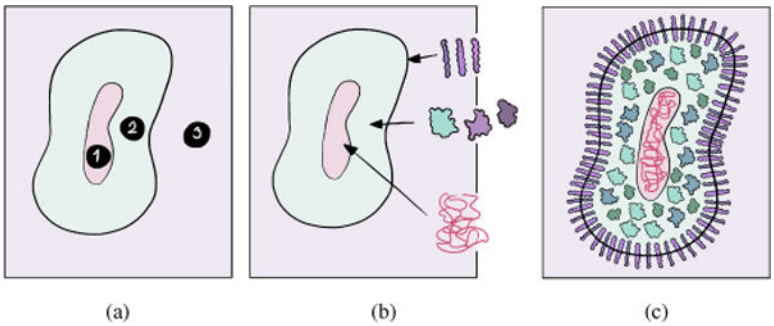


Fig. 1.

(a) The hierarchy is provided by compartments that organize the scene (here marked as 1, 2, and 3). (b) For each individual compartment, their composition is defined including the type and number of ingredients. (c) Our algorithms automatically populate the compartments and their surfaces with the information about the ingredients.

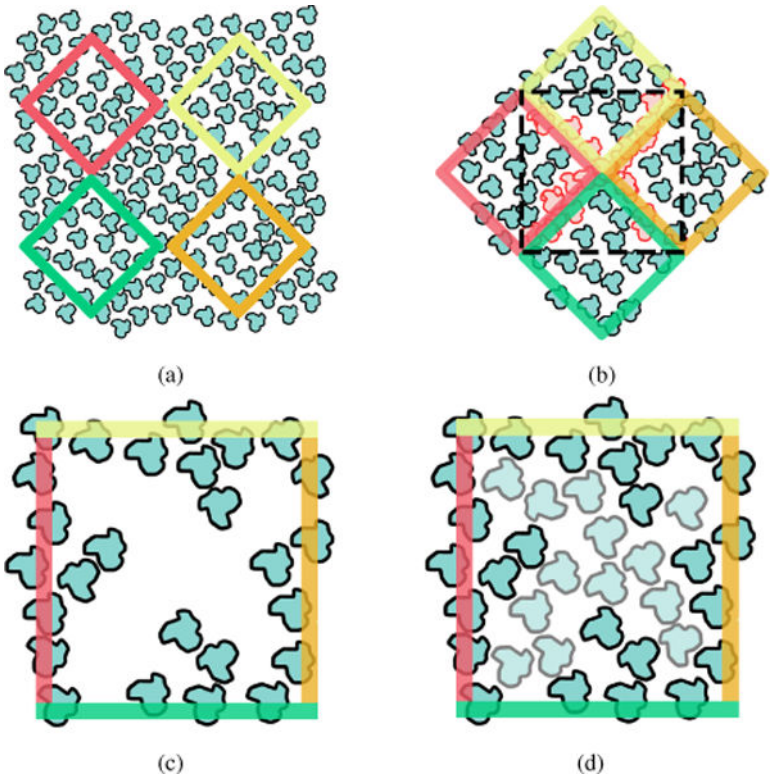


Fig. 2.

Overview of the Wang Tile synthesis. (a) Four small patches are extracted from the input texture. (b) A large patch is assembled from the four small patches, and then the tile is extracted from the center of the patch. (c) Bordering lipids with collisions are removed, leaving holes in the tile. (d) Holes are filled by positioning new lipids.

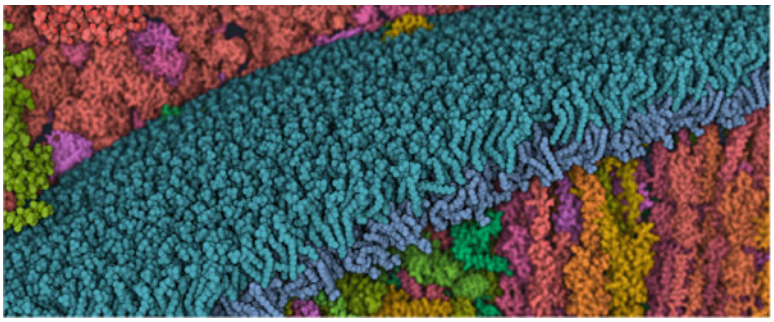


Fig. 3.
Example of lipid-bilayer membrane populated by lipid tiles. This cross-sectional view is visualized using a cutting plane.

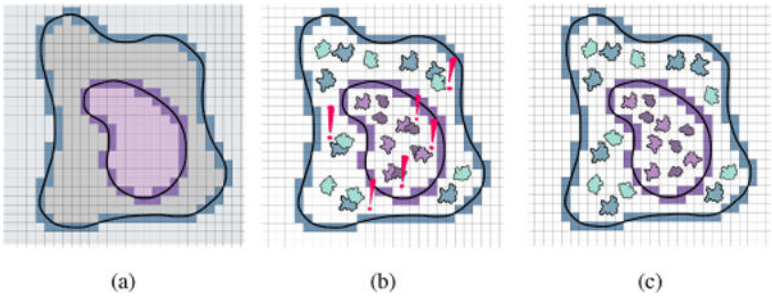


Fig. 4.

The three steps of our approach to populate compartments with soluble components. (a) Voxelization of the space and definition of inner grids (purple and grey), surface grids (dark purple and dark blue), and outer grid (light blue). (b) Positioning of individual ingredients inside the inner compartments. Collisions between ingredients are highlighted. (c) Scene after resolving collisions using a force-based approach.

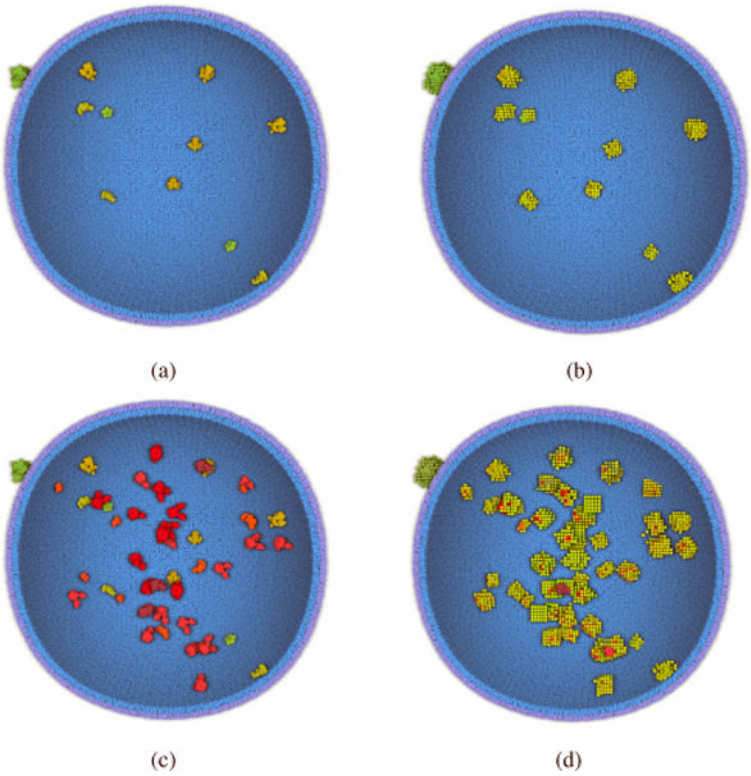


Fig. 5.

(a) Ingredients distributed on the surface of a compartment. (b) Occupancy grid containing information about the voxels occupied by these ingredients (green for outside, yellow for inside). (c) Positioning ingredients inside the compartment (new ingredients have red and orange color). (d) Occupancy grid update.

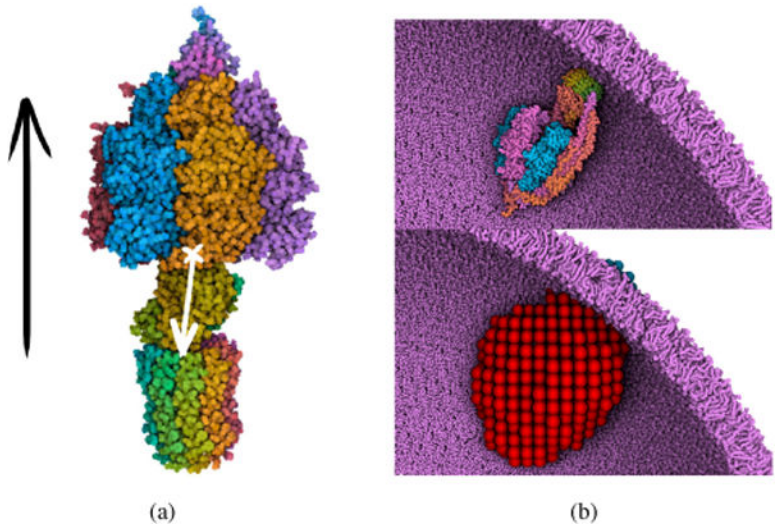


Fig. 6.

(a) Example of surface ingredient with proper principal vector (black), and offset vector (white), pointing from the center of mass to the anchor position at the surface. (b) Positioning of the ingredient on the surface using the correct orientation and offset aligned to the surface normal (top). Grid occupancy update (bottom).

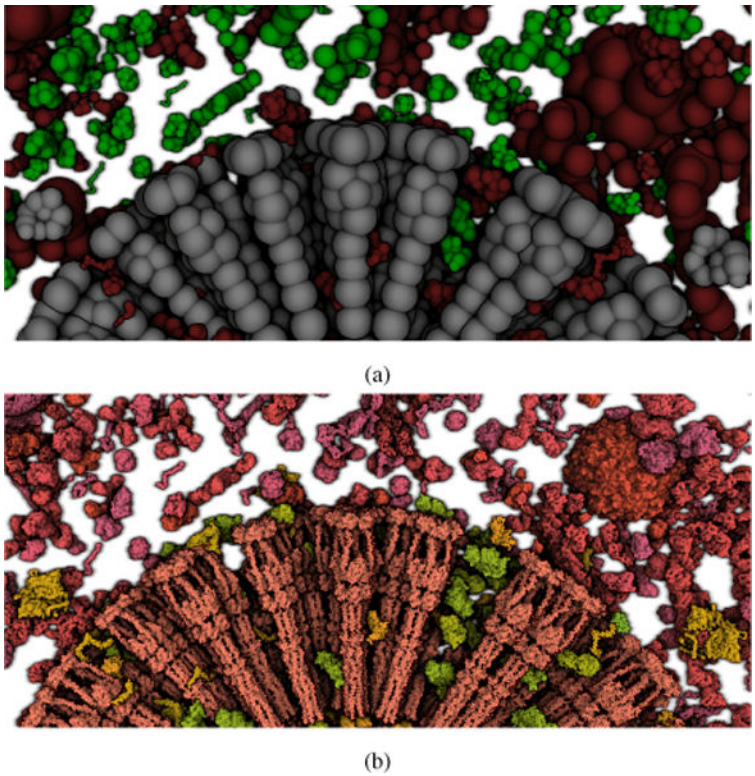


Fig. 7.
(a): Proxy geometry of ingredients. Overlapping geometries are marked in red, non-overlapping ones in green, and static ones in grey. (b) The same scene populated by ingredients.

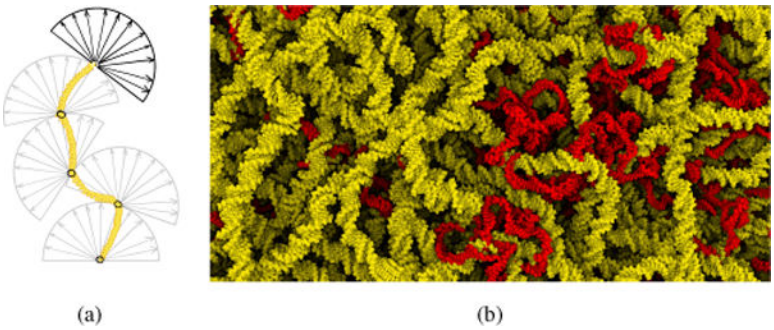


Fig. 8.

Fibrous structures are modeled using a self-avoiding random walk. (a) The walk starts with a random point inside of a certain compartment. A random direction in the forward-hemisphere at this point is then chosen. If there is no intersection with existing structures or compartment surfaces, the direction is selected and the segment is added to the fiber. This process is repeated until the desired length of the fiber is reached. (b) A very long DNA (yellow) with long persistence length is combined with shorter RNA (red) with short persistence length.

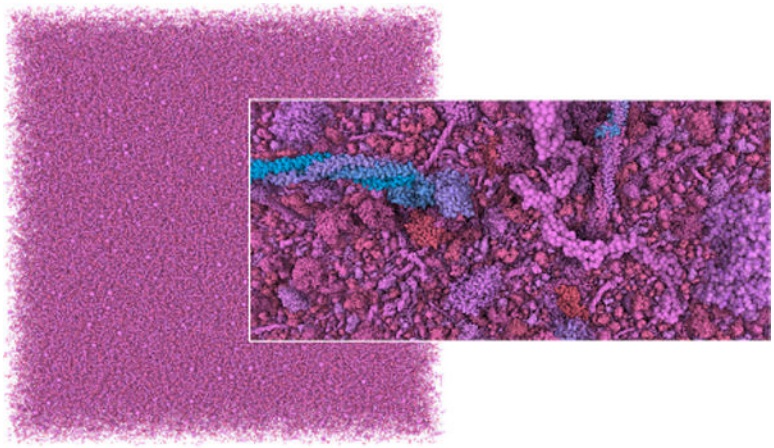


Fig. 9.

A cubic space, two cubic microns wide, filled with blood plasma, with over 4.6 million soluble protein ingredients, populated in ~0.6 sec.

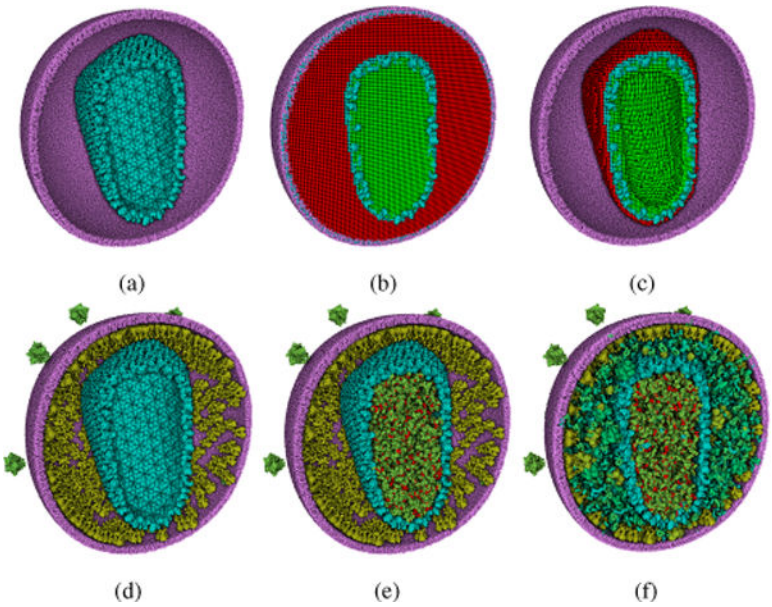


Fig. 10.

Overview of the interactive modeling of the mature HIV virion based on the cellPACK recipe. (a) Building the membrane (magenta) based on a spherical mesh, and preloading the capsid (blue) from a cryo-EM structure (PDB entry 3J3Q). (b) Solid voxelization of compartments. (c) Updating the occupancy grid. (d) Populating the surface ingredients (envelope protein in green, matrix protein in yellow). (e) Building two copies of the RNA (red) using the fiber growth method and attaching the HIV NC protein at all control points. (f) Resulting model.

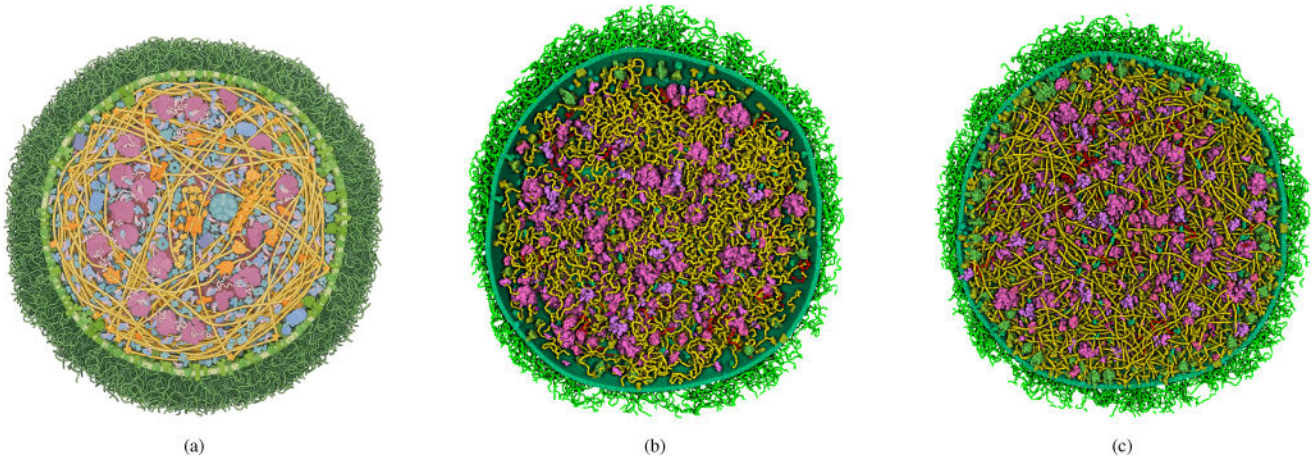


Fig. 11.

Cross-sections of Mycoplasma mycoides. (a) A hand-drawn illustration. (b) 3D model generated with the draft recipe. (c) 3D model generated by interactively tuning the recipe to match general characteristics of the illustration. In the 3D models, the membrane is in green, with membrane-embedded proteins in orange and lipopolysaccharides in green. Inside, DNA is in yellow, soluble proteins in blue and magenta, and RNA in red.

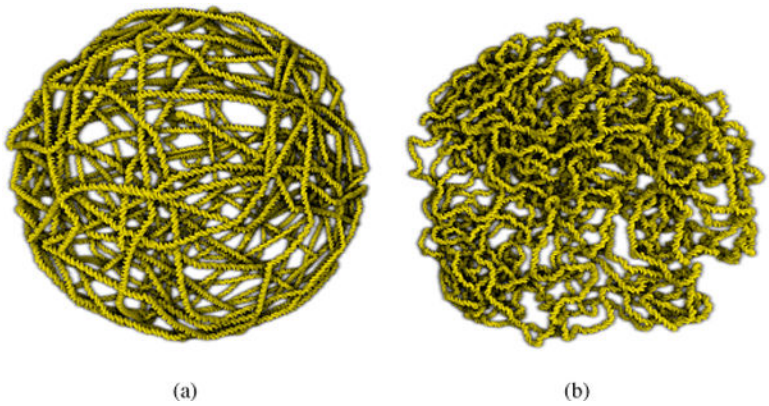


Fig. 12.

Two models of DNA in a bacterial cell, using (a) a long persistence length and (b) a short persistence length (7 sec to generate).