EUROGRAPHICS 2014 / B. Lévy and J. Kautz
(Guest Editors)

Volume 33 (2014), Number 2

CellVIEW: An Illustrative And Multiscale Visualization Tool For Large Biomolecular Datasets

1029

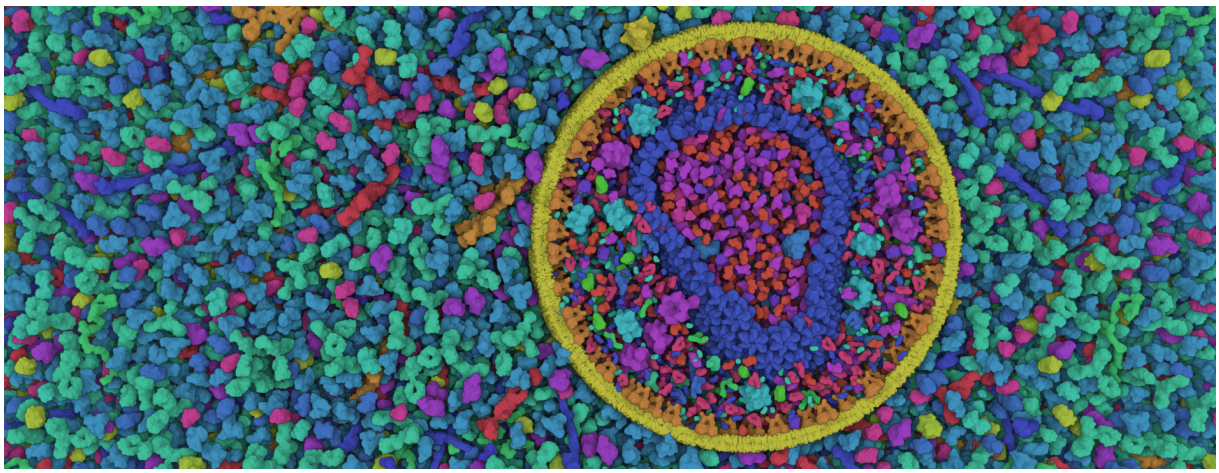

Figure 1: *Real-time screen-shot from our application, an illustrative cross-section of the HIV virus surrounded by blood plasma. Our rendering tool is directly integrated in the Unity3D game engine and is able to render datasets with up to 15 billion atoms smoothly at 60Hz and in high resolution. Because the dataset is rather complex we opted for an illustrative rendering style to improve shape perception and which was inspired by the work of scientific illustrators.*

Abstract

In this article we introduce CellVIEW, a new system to visualize in real-time large biomolecular datasets on the atomic level. Our tool is unique and has been specifically designed to match the ambitions of our domain experts to model and interactively visualize structures comprised of several billions atoms. The CellVIEW system integrates several acceleration techniques to allow for real-time graphics performance of 60 Hz display rate on datasets containing 10^10 *up to* 10^11 *atoms, which corresponds to sizes of small bacterial organisms. Inspired by the work of scientific illustrators we propose a level-of-detail scheme which is two-fold: accelerating the rendering and reducing visual clutter. The main part of our datasets is made out of proteins, but it also comprises of nucleic acids strands which are stored as sets of control points. For that specific case, we extend our rendering method to also support the dynamic generation of DNA strands directly on the GPU, for illustration purposes. It is worth mentioning that our tool has been directly implemented inside a game engine. We chose to rely on third party game engines to reduce software development work-load and to bring bleeding-edge graphics techniques more easily to the end-users. To our knowledge CellVIEW is the only suitable framework for atomistic visualization of large bimolecular landscapes and is also accessible to a large audience due to the use of a free-to-use popular game engine.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Computational biology already offers the means to generate large and static models of cell biology, such as viruses or entire cells on the atomic level [JGA*14] [JAAA*15]. However, biologists struggle with viewing these large scale datasets, because their tools cannot achieve interactive frame-rates for such heavy data loads. Foreseeing the future increase of the size of their datasets, they would need the ability to quickly render billions of atoms, while the current state-of-the-art tools are only able to interactively render up to hundreds of millions of atoms on commodity hardware [GKM*15]. The interactive visualization of such interesting datasets has even greater potential than strictly science-driven data exploration. For instance, interactively showcasing the machinery of life to a lay audience could improve the understanding of basic biology, and thus serve an educational purpose.

CellVIEW is a new tool that provides fast rendering of very large biomolecular scenes and is inspired by state-of-the-art techniques. By introducing new means to efficiently reduce the amount of processed geometries, we obtain a considerable speed-up compared to related works, jumping from 10 fps up to 60 fps for similar sized datasets. As a consequence to such optimisations, the visualization results in an overwhelming number of displayed molecules, which may exhibit strong visual clutter due to the complexity of their shapes. Scientific illustrator David Goodsell has specialized in depictions of complex biomolecular sceneries [Goo12]. The goal of these paintings is to visually communicate machineries of life, and therefore they should provide a maximum degree of understanding and clarity. In most of his painting we observe a high degree of abstraction in the shapes of molecules (see Figure 2). There is an obvious logical reasoning behind this decision which goes beyond simple aesthetics and which is to reduce the level of visual clutter in the scene in order to let the viewer appreciate what is truly shown. Inspired from the work of illustrators we designed a level-of-detail scheme which is twofold: to speed up rendering times and to clarify the scene with simpler shapes, thus procedurally imitating the mindset of scientific illustrators.

Related works have already presented bleeding-edge techniques that can render large datasets with up to billions of atoms at interactive framerates on commodity graphics hardware [LBH12] [FKE13] [LMPSV14]. However, to our knowledge, the tools which implemented these techniques were either not publicly available, or remained in the prototyping stage. Indeed, a very cumbersome task for researchers is releasing a usable version of the source code once the article has been published. Presented techniques are often just a proof-of-concept that would require additional software development to ensure a maximum degree of portability. Unfortunately, this part is often omitted because of a busy research schedule and is simply left in the hand of interested third party developers. Consequently, if this task remains unachieved, end-users are unlikely to use state-of-the-art techniques in their work.

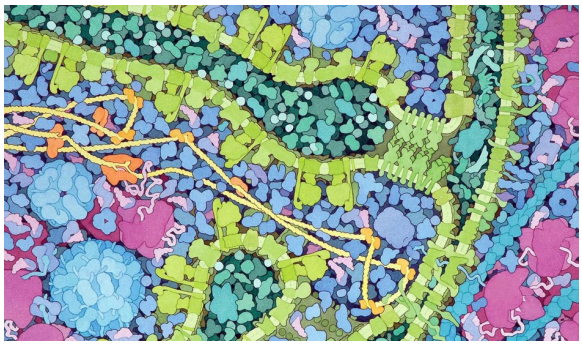

Figure 2: *An illustration of David Goodsell depicting a cross section of a Mitochondrion. Given the complexity of the scene the artist deliberately chose to render molecules with highly abstracted shapes.*

CellVIEW provides a direct connection between state-of-the-art techniques and end users. We implement our tool using a free-to-use game engine, and it is therefore publicly available to anyone. Because we are using a game engine as our development platform, we are able to release our tool quickly and easily. We are also released from the burden of maintaining a complex software solution and from tedious deployment constraints. Additionally, since more people in the community are getting familiar with these engines, we ensure a maximum level of reproducibility among other researchers, thus breaking the barriers caused by heterogeneous toolset usage across research departments.

We showcase our tool with real large-scale scientific data such as the HIV virus and Mycoplasma bacteria, which have been provided to us by domain experts. Their datasets not only contains information relative to protein locations but also to the path of nucleic acids strands, which is stored in the form of control points. We also extend our method to procedurally generate large strands of DNA on-the-fly via GPU tessellation, thus reducing the modelling effort as well as GPU transfer times and memory space.

2. Related Work

Large-scale Molecular Visualization Lindow et al. [LBH12] first introduced a method capable of quickly rendering large-scale atomic data consisting of several billions of atoms on commodity hardware. Rather than transferring the data from CPU to GPU every frame, they store the structure of each type of molecule only once and repeat these structures in the scene via instancing. For each type of protein a 3D grid structure containing the atom positions is created and stored on the GPU memory. Upon rendering, the bounding boxes of the instances are drawn and individually

raycasted, similar to volumetric billboards [DN09]. Subsequently, Falk et al. [FKE13] presented a similar approach with improved depth culling and hierarchical ray casting for impostors that are located far away and do not require a full grid traversal. Although this implementation features depth culling, their method only operates on the fragment level, and could have probably benefited from culling on a coarser instance level. With their new improvement they managed to obtain 3.6 fps in full HD resolution for 25 billion of atoms on a NVidia GTX 580, while Lindow et al. managed to get around 3 fps for 10 billion atoms in HD resolution on a NVIDIA GTX 285. Le Muzic et al [LMPSV14] introduced another technique for fast rendering of large particle-based datasets using the GPU rasterization pipeline instead. They were able to render up to 30 billion of atoms at 10 fps in full HD resolution on a NVidia GTX Titan. They use tessellation shaders to inject atoms on-the-fly in the GPU pipeline similarly to the technique of Lampe et al. [LVRH07]. In order to increase the rendering speed they dynamically reduce the number of injected atoms according to the camera depth. To simplify the molecular structures they discard atoms uniformly along the protein chain and increase the radius of remaining atoms to compensate the volume loss. This level-of-detail scheme offers decent results for low degrees of simplification, but it does not truly guarantee preservation of the overall shape of molecules for highly simplified structures.

Occlusion Culling A key aspect when rendering large and complex scenes is efficient occlusion culling. Grottel et al. presented a method to perform coherent occlusion culling for particle-based datasets, which is closely related to Deferred Splatting [GBP04], and relies on temporal coherency. Their particle data is stored in a uniform grid, and operate occlusion culling on two-levels: on the level of grid cells first, and on the atomic level afterwards. Individual atoms are rendered via 2D depth impostors, because they have a much lower vertex count than sphere meshes for the same results. At the beginning of each frame they render an early depth pass with atoms that were visible during the previous frame. This pass results in an incomplete depth buffer that they utilize to determine the visibility of the remaining particles. When rendering a lot of primitives, the graphics hardware usually takes care of discarding hidden fragments beforehand. However, this feature is disabled if the fragment output depth is modified, which is the case with sphere impostors. Therefore, they only render flat impostors instead to accelerate the process. For the coarse-level culling, they determine the visibility of the grid cells by testing their bounding boxes against the incomplete depth buffer via hardware occlusion queries (HOQ). For the fine-level culling, they test the visibility of individual atoms in the final render using the well known hierarchical z-buffer (HZB) visibility technique [GKM93]. They construct the HZB from the incomplete depth buffer beforehand; during the final render they discard fragment operations from the vertex shader if the visibility test fails, thus compensating for the lack of early fragment rejection with depth impostors.

Illustrative Molecular Visualization When rendering large structures the speed of execution is not the only concern. As the structures increase in size, they also increase in complexity, and it is necessary to display the data in the most suitable way. Ambient occlusion, for example, has shown to play an essential role when rendering large molecular structures, as it provides important depth cues which increase shape perception [GKSE12, ESH13]. But the rendering style is not the only means to define visual encoding, and geometric modification may be applied too. Parulek et al. [PJR*14] demonstrated a continuous level-of-detail scheme for molecular data. Their object-space approach offers detail-on-demand in the focus area while applying gradual shape simplification schemes elsewhere. On the finest level of detail, they showcase SES surface representation and an abstracted molecular shape in the distance. They introduced an interesting molecular structure abstraction approach, based on a union of spheres obtained via clustering methods. Several common clustering methods are evaluated, and those that preserve elegantly the low-level shape details are highlighted. The continuous level-of-detail was only demonstrated on large molecules and has not been extended to large multi-molecular scenes. This work is an attempt to extend their continuous level-of-detail approach to larger scales.

Modelling of Nucleic Acids Chains DNA plays a key role in cell biology and is an important part of our datasets as well. Therefore, as with protein data, we must also provide efficient rendering methods for this type of structure. There are several scientific modeling tools [MC98,LO08,HLLF13] available that are able to generate DNA strands from a simple set of control points. All of these approaches are performed on the CPU, which means that the bases position and rotation must be uploaded on the GPU prior to the rendering. Because of the cost of transferring data from CPU to GPU, however, this approach would likely perform poorly when rendering and animating large DNA strands. Therefore, we introduce a new GPU-based approach which relies on dynamic instancing of DNA bases along a curve. This approach is similar to the work of Lampe et al. [LVRH07], who use geometry shaders to dynamically instantiate residues along the protein backbone. The major difference between their work and ours is the introduction of procedural building rules based on scientific data.

Game Engines and Biomolecular Visualization Game engines are becoming increasingly popular in the visualization community. Researchers have been seduced by their universal dimension and their ease of use, both in programming and deployment. In the domain of biomolecular visualization a few articles highlight the fact that they have used a game engine to develop their applications. Shepherd et al. [SZA*14] have developed an interactive application

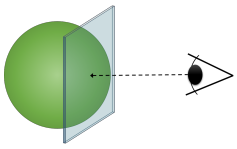

Figure 3: *Depth conservative sphere impostors. In order to benefit from early depth culling for depth impostors, we must guarantee that the output depth will be greater than the depth of the billboard.*

to showcase 3D genome data. Their visualization is multiscale and is able to render a large amount of data thanks to the implementation of a level-of-detail scheme. Various works on interactive illustration of biological processes have also reported using game engines to visualize polymerization [KPV*14] and membrane crossings [LMWPV15] in 3D and in real-time. Similarly to our work, Baaden et al. [LTDS*13] developed a molecular viewer which offers artistic and illustrative rendering methods based on the Unity3D game engine. Their primary intention was to democratize biomolecular visualization thanks to the use of a more intuitive and user friendly framework. Their tool has managed to prove that game engines are also useful in serious visualization projects. One noticeable technical difference between CellVIEW and UnityMol is that our tool is fully integrated in the "What you see is what you get" (WYSIWYG) editor of the Unity3D engine. Thus our tool coexists with the engine toolset which provides a rich set of functionalities that we can directly use to enhance the quality of our visualization. Moreover, unityMol address the problem of visualizing one molecule at a time, while CellView addresses the problem of visualizing multiscale scenes with atom level complexity.

3. Efficient Occlusion Culling

The overwhelming size of the datasets (e.g. from millions to billions of atoms) calls for efficient methods of culling for when aiming at interactive render speeds. The presented method is closely related to the two-level occlusion culling introduced by Grottel et al. [GRDE10]. We have revisited this method to provide efficient occlusion culling for biomolecular datasets that are several orders of magnitude larger than the datasets used with their method. It is worth mentioning that their technique was designed for general-purpose particle data, which introduced more constraints, while our method is purely designed for biomolecular data only. Our rendering pipeline is based on the work of Le Muzic et al. [LMPSV14] which is designed for the rendering of large number of molecules. This method relies on the tessellation shaders to dynamically inject sphere primitives into the pipeline for each molecule.

The heaviest part of the computation for this rendering method is the geometry processing, which is performed during the dynamic tessellation. In order to lighten the processing load, Le Muzic et al. [LBH12] reduced the number of injected atoms according to the camera depth. However, without proper occlusion culling this computation, although reduced, would still be performed, even if a molecule is completely occluded. We use temporal coherency to determine hidden molecules in a given frame. To ensure that only visible geometries are sent to the renderer, we use a dynamic-sized render commands buffer that matches the number of visible molecules and is recomputed before each rendering operation. The steps of our method are layed down as follow:

1. Draw visible molecules at frame t - 1
2. Generate N-Buffers
3. Compute visibility for remaining molecules
4. Draw remaining visible molecules
5. Read visibility from item buffer
6. Fetch colors from item buffer

3.1. Render Commands

Our pipeline is designed so that the visibility of a molecule is already known prior to rendering. A naive approach to discarding hidden molecules would be to do it in the early stages of the rendering in the vertex shader, prior to dynamic tessellation. However, discarding elements in the vertex shader would cause unnecessary processing overhead. We use render commands instead to ensure that only visible molecules will be sent to the renderer. The render commands are recomputed before every render operation. For every visible molecule we issue a render command entry to the buffer via stream output. The resulting size of the commands buffer therefore matches the number of visible molecules. Each buffer entry simply contains the id of the corresponding molecule, which serves during the rendering as an index to fetch needed information, such as position, rotation, type and also to fetch corresponding atoms for the dynamic tessellation.

3.2. Temporal coherency

The key to our culling method is the use of temporal coherency to determine hidden molecules. The rendering generates a texture, dubbed item buffer, whose pixels contain the id of the visible molecules instead of the color. We process this texture at the end of each frame in order to determine the visible molecules. These molecules are then drawn at the beginning of the next frame by issuing one render command per visible molecule.

The render results are thus incomplete and only contains molecules that were previously visible. To query the visibility of the remaining molecules we generate a HZB buffer from the incomplete depth buffer which we previously rendered. We subsequently determine the occlusion of the remaining molecules and generate render commands for the

visible ones. The remaining molecules are finally rendered, thus completing the depth and id buffer, which are then used to determine the visible elements for the next frame.

The use of the incomplete depth buffer for occlusion culling often results in an overly large number of hidden molecules to pass the test, which can reduce performance. Alternatively we propose a variant of this method which uses the depth buffer from the previous frame to generate the HZB and to determine the remaining visible molecules. As a result much less hidden molecules are sent to the render, since the buffer is not incomplete, thus increasing performance greatly. However, abrupt changes of point of view would result in short but noticeable artefacts due to incoherency between consecutive frames. We suggest using the second variant when dealing with very large datasets.

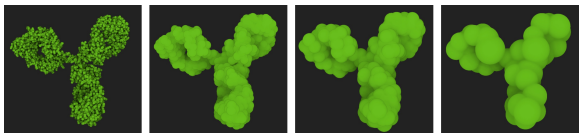

Figure 4: *Our level-of-detail scheme allows a great reduction of the number of sphere primitives. In the upper-left corner the protein is shown with full-atomic detail, in the upper-right corner the molecule is drawn with only 15 percent of the overall sphere count, lower-right is 5 percent and lower-left 0.5 percent reducing the number of spheres from 10182 to 50 while preserving the overall shape of the protein. The shape of individual spheres is more highlighted in this figure compared to our application for the purpose of explanation.*

3.3. Accelerating Texture Writes

When drawing atoms of a single molecule many atoms are actually occluded by atoms of the same molecule and other surrounding atoms. These atoms will be drawn nevertheless, as a well known limitation of graphics hardware, so far, was the lack of early depth fragment rejection for depth impostors. Thanks to advances in graphics hardware however, it is now possible to activate early depth rejection when a fragment is modifying the output depth value. This feature is called conservative depth output. Once activated, in order for conservative depth output to work, we must output a depth which is greater than the depth of the 2D billboard. This way the GPU is able to tell if a fragment will be occluded beforehand by querying the visibility internally. A description of the depth conservative output sphere impostor is given in Figure 3. Additionally, to limit the number of texture writes we only output the id of the molecules to the render texture upon rendering. The colors are fetched afterwards in post-processing by reading the molecules properties from the id. We have witnessed a considerable improvement in performance when writing depth conservative output + id, compared to standard depth output + color + id.

4. Twofold Level-of-Detail

Proteins are key elements of biological organisms and thus it is important to visualize them in order to understand how those machineries work. They are also present in fairly large quantities, which is challenging to render interactively without proper level-of-detail schemes. Additionally, their complex shapes might cause a high degree of visual clutter, which may render overly complex images.

We propose a twofold level-of-detail scheme which provides rendering acceleration and offers clearer depiction of the scene using smoothly abstracted shapes. Our technique also offers a seamless continuum between the different levels of abstractions. We employ clustering methods similarly to the technique of Parulek et al. [PJR*14] to simplify the shape of the molecules and reduce the amount of primitives rendered. Clustering offers a very good decimation ratio as well as accurate shape abstraction, because it tends to preserve low-frequency details. With higher shape accuracy, we are also able to switch to simpler LOD proxies closer to the camera, thus gaining render speed without compromising image quality. The clustering results in a set of spheres which are computed on the CPU and then stored in large GPU buffers similarly to atom spheres. We empirically determined that four levels of detail where sufficient with our current dataset. With these settings we also paid special attention not to compromise image quality. All the proteins share the same level properties: the first level includes 100 percent of the atoms, the second level has a decimation ratio of around 15 percent, the third level 5 percent and the last level down to 0.5 percent.

The first two levels were computed with a simple method based on spatial coherency of atoms along the protein chain, also known as coarse grain molecular simplification. This method has previously been used in biomolecular visualization to simplify the generation of molecular surfaces [KBE09]. We use this approach because it is faster to compute than standard clustering method for similar results. For the last level we wanted to dramatically reduce the number of spheres, and therefore the previous approach would not perform well enough for this task, so we use k-means clustering instead. A major issue common to many clustering algorithms is the slow computing speed when dealing with several thousands of spheres. We use the results of the third clustering level as input for the k-means clustering instead of the complete set of atoms, thus dramatically decreasing computation times. The resulting levels are shown on Figure 4.

When observing the large-scale biomolecular depictions of David Goodsell (see Figure 2), we noticed that molecules tend to increase in volume with distance. This operation is probably done to compensate the relatively small size of proteins when viewed in their natural embedding. We sub-

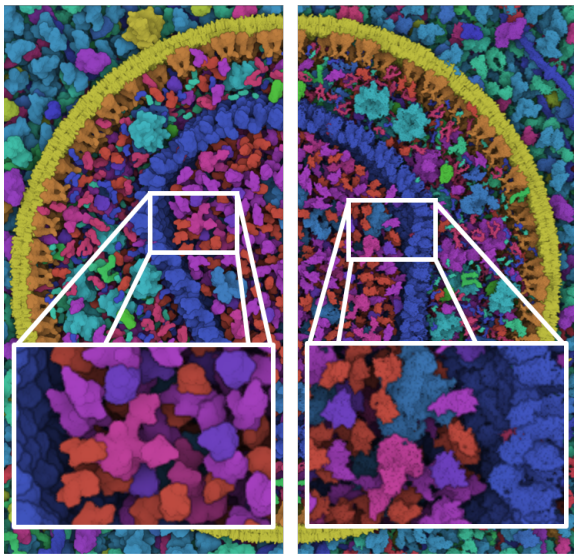

Figure 5: *Side-by-side comparison of our illustrative level-of-detail compared with full atomic details. Our illustrative level-of-detail provide smoother and elegant shapes, while also reducing the processing load.*

jectively found this artistic choice to be relevant and visually pleasant. Therefore, we also include a scaling factor to our illustrative level-of-detail. Each LOD level has a proper range in terms of camera depth which is common to every type of molecule, i.e., level 0 is defined between distance 0 and 50Å, level 1 between 50 and 100Å, etc. We define min and max radius for each range, and we smoothly interpolate between these values. A side-by-side comparison between our illustrative level-of-detail and full atomic representation is provided in Figure 5.

5. Dynamic DNA Generation

Animating individual molecules is fairly straightforward because modifying the atomic structure may not required. In the case of DNA, the positions of the control points of the DNA path highly influence its structure, namely the positions and rotations of the individual nucleic acids. As a result, each modification of the control points of the DNA path would require a new computation of the strand. Current approaches are only performed on the CPU, [HLLF13, LO08, MC98] which means that the whole nucleic acids chain has to be transferred to the GPU upon re-computation. While this approach is viable for low to mid sized DNA strands, it is very likely to perform poorly for large and dynamic DNA paths featuring a large number of control points.

We propose using dynamic tessellation to leverage the generation of nucleic acid strands. So far we only used tessellation to instantiate data priorly stored on the GPU memory. However it is possible to include building rules characteristic to DNA modeling to procedurally generate a double helix structure simply based on control points. Thus, the data transfers to the GPU would be dramatically reduced as well as GPU memory space.

Similarly to GraphiteLifeExplorer [HLLF13] our goal is more illustrative than strict biomolecular modeling. Therefore we privilege rendering performance over accuracy, and we provide only a limited array of folding types. Although the study of DNA structures has revealed many different types of foldings requiring complex modeling algorithms, the most commonly recognizable shape is the B-DNA, which exhibits a regular structure and is simple to model: a spacing of 3.4Å and a rotation of 34.3° between each base. Based on these rules we are able to procedurally generate B-DNA strands based on path control points via GPU dynamic tessellation. The workflow which we employ is described as follow:

1. Resample control points (on the CPU).
2. Compute smooth control point normals (on the CPU).
3. Upload control point data to the GPU
4. Draw all the path segments in one pass, one vertex shader per segment
5. Read the control points and adjacent points needed for smooth cubic interpolation. (In vertex shader, for each segment)
6. Do uniform sampling along the cubic curve segment to determine the positions of the bases. (In vertex shader, for each segment)
7. Pass the position of the bases to the tessellation shader. (In vertex shader, for each segment)
8. Compute normal vector of each base using linear interpolation between the control points normals (In tessellation shader, for each base)
9. Inject atom, then translate and rotate accordingly (In tessellation shader, for each atom of each base)
10. Render sphere impostor from injected atom (In geometry & fragment shader, for each atom of each base)

5.1. Smooth Normals Computation

A well known challenge when dealing with 3D splines is to determine smooth and continuous frames along the whole curve. Any twists or abrupt variation in frame orientation can cause visible artefacts due to irregularities in the DNA structure, which we should avoid at all costs. We perform the computation of the smooth and continuous normals primarily on the CPU. We first determine the normal direction for every control point of the path. Then we sequentially browse the control points and rotate the normal direction vector around the tangent vector in order to minimize the variation in orientation compared to the previous control point normal. The recalculated normals are then uploaded to the GPU along with the control point positions. , and during the instantiation of the nucleic acids we obtain the normal

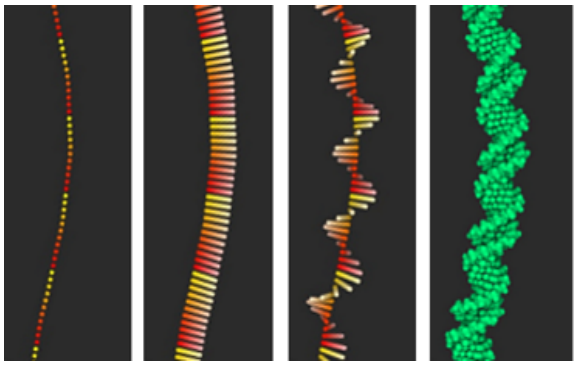

Figure 6: *Procedural generation of B-DNA structures via GPU dynamic tessellation. In the first image we can see the position of the individual bases. The color gradient highlight the individual segments. In the second image we draw the smooth normals along the curve, the color desaturation shows the direction of the vector. The following image shows the rotation offset of the normal vector along the tangent, and the last image shows the final result.*

vector of a nucleic acid by linear interpolation of the two segment normal vectors.

5.2. Double Helix Instancing

When instancing individual pairs of nucleic acids in the tessellation shader, we first fetch the nucleic acid atoms, position them along the curve, orient them toward the normal direction and then rotate then around the tangent vector in order to generate the double helix. We always orient the first base of a segment according to the normal direction only, while the subsequent bases are all oriented towards the normal direction first and then rotated with an increasing angular offset of 34.3° around the tangent of the curve. The angular offset of a given base is defined as follows: $\alpha = i \times 34.3$, where i corresponds to the index of the base inside a segment. The last base of a segment must therefore always perform an offset rotation of $(360 - 34.3)^\circ$ around the tangent vector. This way it connects smoothly to the first base of the next segment, which is oriented towards the normal vector only. The result of the procedural generation of B-DNA is given in Figure 6 as well as a visual explanation of the different steps.

5.3. Control Points Resampling

Given that the bases of a segment must perform a revolution to connect smoothly to the next segment, it is trivial to determine the number of bases per segment as follows: $n = 360 \div 34,3$. From the number of bases per segment we can easily deduce the required size of a segment: $s = n \times 3.4$Å, which results in a segment length of 35Å approximatively. This constraint implies all the control points to be spaced uniformly with a distance of 35Å. However, it may be the case that control points obtained via modelling software have arbitrary spacing. Therefore, we must resample the control points along the curve to ensure a uniform spacing before uploading it to the GPU. Although we resample the control points accordingly to the B-DNA build rules, the length of the interpolated curve segments will always be slightly greater because of the curvature. We did not find this to be visually disturbing, probably because consecutive segments in our dataset did not showcase critically acute angles, so the overall curvature of the path remained rather low.

6. Perks and Limitations of a Game Engine

Although game engines may not yet be the perfect all-in-one solution, they are good contenders to generic visualization frameworks. Above all, they offer ease of access to GPU features for fast computation, while remaining user friendly. CellVIEW is implemented in Unity3D, one of the most popular game creation framework. The engine uses C# as a scripting language and also supports a large array of platforms from desktop PCs to mobile devices. It is worth mentioning that the Unity3D engine is free of charge for the personal edition and thus is accessible to anyone. In this section we briefly review the perks and limitations of the engine when used as a development platform for scientific visualization.

6.1. What We Liked

Scripting The language which is used for coding with Unity3D is C#, while the engine is internally coded in C++. The simplicity of C# makes it a great language for quick prototyping over C++. Also, C# is more accessible to enthusiast programmers than C++, which makes our tool more accessible. Deployment is also easier since our project only consists of a few scripts while the heavy software part is actually inside the engine itself. **Editor** The game engine features a WYSIWYG editor, which is a great tool for artistic setup because it offers a real-time preview of the application. Additionally the editor features a large array of features which we may use out-the-box in our visualization such as high quality image effects for instance (SSAO, Depth-of-field, etc.). The editor also features UI extension capabilities, which we use to design the custom UI of CellVIEW. **Mesh-based Rendering** While our datasets are made out of particles, cell biology offers different types of data acquisitions, which may result in surface representations (e.g. Cryo-electron microscopy, Tomography). Therefore, we designed our rendering pipeline to coexists with the legacy mesh pipeline of the engine. Hence, we may use meshes that are either artist-designed or obtained via data acquisition and render them together with our own data in the same view. **High-End GPU Capabilities** With Unity3D most of the low-level graphics API is exposed to the scripting world. Therefore, we encountered no difficulty to implement our custom pipeline which

relies on advanced graphics features such as tesselation or compute shaders. **Maintenance** Unity3D is a very popular framework which generate large revenues in the game industry. Therefore, teams of professional developers are constantly working on improving the tool and supporting bleeding edge features. **Community** Unity3D is a universal tool that is widely used by many independent developers and professional multi-disciplinary teams. Therefore, the community is very large, and in addition to a very dense online documentation it is very easy to seek help online when facing problems with the engine. **Deployment** Although the editor is already a very good tool to work with, we may still want to export the program in a standalone application for deployment. From our experience, the building process and deployment is always a very cumbersome task. With Unity3D, the process of compiling a program to an actual executable file is only a matter of a few clicks.

6.2. What We Did Not Like

CPU Performance CPU performance might be strongly affected with the use of a C# scripting engine compared to native C++. While modern C#/.NET framework makes considerable steps towards multi-platform and close to native performance, Unity3D still features a scripting engine which is largely out of date. This is why we would not recommend it for heavy CPU computation, although alternative exists such as the use of ad-hoc native libraries for instance. **GPU Performance and Portability** While Unity3D has made cross-platform compilation one its major advantages, to this day, high-end graphics capabilities are only available with DirectX11. Thus, we may only deploy CellVIEW on machines that supports DirectX11 which excludes tablets, Linux, and OSX machines. However, Unity3D recently announced that it will soon extend high-end graphics capabilities to support modern OpenGL, and will even provide support for compute and tessellation to mobile devices. **Source Code Availability** Other game engine such as Unreal Engine have recently released their source code to the public, in order to encourage third party and community-based extensions of the engine. In the case of Unity3D the source code of the engine is not publicly available, which means that if a critical feature is missing in the core of the engine, there will be no other option than requesting it rather than simply coding the feature ourselves.

7. Results

We have tested our techniques with different datasets from the same modeling software. The data was obtained with CellPACK [JAAA*15], a modeling tool for procedural generation of large biomolecular structures. CellPACK summarizes and incorporates the most recent knowledge obtained from structural biology and system biology to generate comprehensive mesoscale models. For instance, based on experimentally obtained data (e.g. data such as proteins structure, concentration and spatial distribution), they managed to generate a whole model of the HIV virus via a packing method based on collision constraints [JGA*14]. Although CellPACK is developed and used mostly by our domain experts, it is publicly available and offers to anyone the means for experimenting and creating their own models.

Our program reads the files that are generated by the tool and is able to reconstruct the scene in a multiscale approach. The generated files comprise of a list of elements, namely the name, position, rotation and PDB identifier [SLJ*98]. The structural data is directly fetched online from the Protein Database. In case an entry is not present, we load the protein information from a dedicated repository provided by the domain experts. The generated files also include control points for the linear type of structures such as DNA, RNA, unfold peptide, lypoglycane, etc...

7.1. Use cases

HIV Virus + Blood Plasma The first dataset we showcase is a combination of two datasets: the HIV virus [JGA*14] surrounded by blood plasma [blo]. The dataset comprises of a total of 70 million atoms and 40 different types of molecules. The rendering test was performed on an Intel Core i7-3930 CPU 3.20 GHz machine coupled with a GeForce GTX Titan X graphics card with 12GB of video RAM. For the purpose of rendering benchmarks, we periodically repeat this dataset to reach an overall number of 15 billion atoms.

It is rather hard to precisely evaluate the performance of a system which is based on level-of-detail as the parameters are arbitrarily chosen to balance image quality and performance. In our tests we were able to render the entire dataset above 60 fps from every viewing angle, from very far zoomed-out to very near close-ups. It is worth mentioning that it would always be possible to use a more aggressive level-of-detail to improve render speeds, in our tests we paid a special attention not to compromise image quality when setting up LOD parameters. Above 15 billion atoms we can hardly guaranty a smooth framerate above 60 fps from any viewing angle, however it would still be possible to render even larger structures at decent interactive framerates. However we start to question the utility of this approach to display datasets that would be one or several orders of magnitude higher. Indeed, when viewing large datasets as a whole, the view starts to exhibit graining aliasing artefacts due to the very small size of individual molecules. While image quality could be improved with anti-aliasing techniques, the issue would still persist for larger zooming scale. What would be truly needed at this scale is a new type of representation for this data that would reduce the amount of displayed geometries and also provide a clearer view of the scene.

Mycoplasma HIV is a retrovirus and thus only contains RNA, which features more complex modeling rules, than DNA that forbids dynamic procedural generation. For this

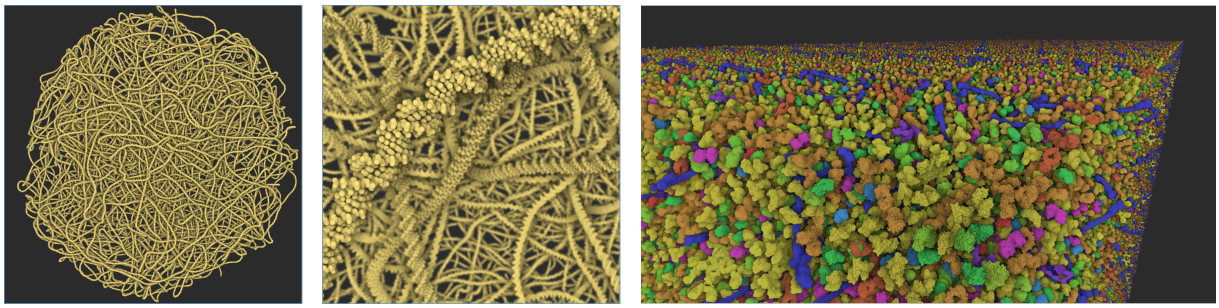

Figure 7: *The results of our rendering test, showing DNA from Mycoplasma on the left and HIV in blood plasma on the right. The first dataset has approximatively 11 million atoms and the second one approximatively 15 billion*

specific case, the atomic structure of RNA would have to be modelled ad-hoc with a third party tool before being loaded in CellVIEW. To fully demonstrate the use of our dynamic structure generation for DNA, we use the data from Mycoplasma mycoide, one of the smallest bacteria with a genome of 1,211,703 base pairs. Mycoplasma has been widely studied by biologists and was the first organism to be fully synthetized. For this dataset we only showcase a preliminary model, build with CellPACK, containing only a quarter of the total genome. The DNA comprises a set of control points as well as the PDB reference to the pairs of nucleic acids and is loaded directly from the files generated by CellPACK. The data we showcase was made out of 9617 control points for an overall number of 11,619,195 atoms, and the results can be seen in Figure 7. We were able to render and procedurally generate the entire strands at 70 fps without any LOD schemes. With this test we simply wanted to show the computation times to proof the efficiency of our technique. Naturally, when using LOD and culling schemes like with protein data, the performance would considerably increase and would not impact the rest of the computation. The results of the two datasets are shown in Figure 7, and a preliminary render of the Mycoplasma is show in Figure 8.

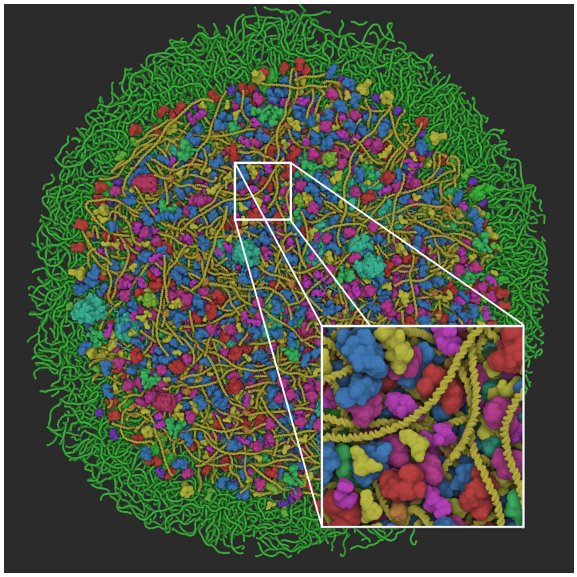

Figure 8: *Preliminary results of the Mycoplasma model. The model additionally features proteins, RNA and lycosomes.*

7.2. Discussion

CellVIEW was very well received by domain experts, although they first regretted that it is not directly embedded in their homemade toolset. However, since they had previous experience working with Unity3D, the transition was rather smooth. So far they could not iteratively display large datasets, such as Mycoplasma for instance, therefore it was difficult for them to visually analyse and debug their models. It is evident that CellVIEW will help the domain experts to scale up and go further in the study of multi-scale modelling of larger cells. A feature that has been frequently requested was the ability to render the scene imitating fluorescent microscopy images. This type of representation is very insightful because it allows them to compare and validate their model with real microscopy images. So far the task of creating such images was very cumbersome for them. In response to their feedback we implemented the option to render the scene with microscopy settings. Thanks to this small add-on, they are now able to generate such microscopy images in real-time side-by-side with 3D multi-scale representation.

8. Conclusions and Future Work

We have introduced CellVIEW, a tool for real-time multi-scale visualization of large molecular landscapes. Our tool is able to load files generated by CellPACK, a powerful modeling tool for designing entire organisms on the atomic level. CellVIEW was engineered to work seamlessly inside the Unity3D game engine, which allows us to prototype and deploy faster without compromising GPU performance. The

method which we presented also features considerable improvements over previous works in order to anticipate the future growth in size of the models. We provide the mean for efficient occlusion culling, which is crucial when dealing with such large scale datasets. We also designed a level-of-detail which allows both acceleration of rendering times and provides a clear and accurate depiction of the scene. Finally, we demonstrated the use of dynamic tessellation to generate biomolecular structures on-the-fly based on scientific modeling rules.

In future work we would like to tighten the collaboration with domain experts and achieve interactive viewing of more complex organisms and bacteria such as E.Coli. As the scale increases the view exhibits highly grained results due to the very small size of molecules. In the future, we would like to focus on better representation for this case, and perhaps find new semantics that could be integrated in our level-of-detail continuum. We also would like to use our rendering to experiment with in-situ simulations as a visual exploration tool for scientists, but also as an educational tool to showcase the machinery of life to a lay audience.

References

[blo] Blood plasma cellpack recipe. URL: http://www.autopack.org/cellpack-recipes/blood-plasma. 8

[DN09] DECAUDIN P., NEYRET F.: Volumetric billboards. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 2079–2089. 3

[ESH13] EICHELBAUM S., SCHEUERMANN G., HLAWITSCHKA M.: PointaoâĂŤimproved ambient occlusion for point-based visualization. 3

[FKE13] FALK M., KRONE M., ERTL T.: Atomistic visualization of mesoscopic whole-cell simulations using ray-casted instancing. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 195–206. 2, 3

[GBP04] GUENNEBAUD G., BARTHE L., PAULIN M.: Deferred splatting. In *Computer Graphics Forum* (2004), vol. 23, Wiley Online Library, pp. 653–660. 3

[GKM93] GREENE N., KASS M., MILLER G.: Hierarchical z-buffer visibility. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM, pp. 231–238. 3

[GKM*15] GROTTEL S., KRONE M., MULLER C., REINA G., ERTL T.: MegamolâĂŤa prototyping framework for particle-based visualization. *Visualization and Computer Graphics, IEEE Transactions on 21*, 2 (2015), 201–214. 2

[GKSE12] GROTTEL S., KRONE M., SCHARNOWSKI K., ERTL T.: Object-space ambient occlusion for molecular dynamics. In *Pacific Visualization Symposium (PacificVis), 2012 IEEE* (2012), IEEE, pp. 209–216. 3

[Goo12] GOODSELL D. S.: 3d molecular models of whole hiv-1 virions generated with cellpackputting proteins in context: Scientific illustrations bring together information from diverse sources to provide an integrative view of the molecular biology of cells. *Bioessays. 34* (2012), 718–720. 2

[GRDE10] GROTTEL S., REINA G., DACHSBACHER C., ERTL T.: Coherent culling and shading for large molecular dynamics visualization. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 953–962. 4

[HLLF13] HORNUS S., LÉVY B., LARIVIÈRE D., FOURMENTIN E.: Easy dna modeling and more with graphitelifeexplorer. *PloS one 8*, 1 (2013), 53609. 3, 6

[JAAA*15] JOHNSON G. T., AUTIN L., AL-ALUSI M., GOODSELL D. S., SANNER M. F., OLSON A. J.: cellpack: a virtual mesoscope to model and visualize structural systems biology. *Nature methods 12*, 1 (2015), 85–91. 2, 8

[JGA*14] JOHNSON G. T., GOODSELL D. S., AUTIN L., FORLI S., SANNER M. F., OLSON A. J.: 3d molecular models of whole hiv-1 virions generated with cellpack. *Faraday Discuss. 169* (2014), 23–44. 2, 8

[KBE09] KRONE M., BIDMON K., ERTL T.: Interactive visualization of molecular surface dynamics. *Visualization and Computer Graphics, IEEE Transactions on 15*, 6 (2009), 1391–1398. 5

[KPV*14] KOLESAR I., PARULEK J., VIOLA I., BRUCKNER S., STAVRUM A.-K., HAUSER H.: Illustrating polymerization using three-level model fusion. *arXiv preprint arXiv:1407.3757* (2014). 4

[LBH12] LINDOW N., BAUM D., HEGE H.-C.: Interactive rendering of materials and biological structures on atomic and nanoscopic scale. In *Computer Graphics Forum* (2012), vol. 31, Wiley Online Library, pp. 1325–1334. 2, 4

[LMPSV14] LE MUZIC M., PARULEK J., STAVRUM A.-K., VIOLA I.: Illustrative visualization of molecular reactions using omniscient intelligence and passive agents. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 141–150. 2, 3, 4

[LMWPV15] LE MUZIC M., WALDNER M., PARULEK J., VIOLA I.: Illustrative timelapse: A technique for illustrative visualization of particle-based simulations. In *IEEE Pacific Visualization Symposium (PacificVis 2015)* (2015). 4

[LO08] LU X.-J., OLSON W. K.: 3dna: a versatile, integrated software system for the analysis, rebuilding and visualization of three-dimensional nucleic-acid structures. *Nature protocols 3*, 7 (2008), 1213–1227. 3, 6

[LTDS*13] 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 8*, 3 (2013), 57990. 4

[LVRH07] LAMPE O. D., VIOLA I., REUTER N., HAUSER H.: Two-level approach to efficient visualization of protein dynamics. *Visualization and Computer Graphics, IEEE Transactions on 13*, 6 (2007), 1616–1623. 3

[MC98] MACKE T. J., CASE D. A.: Modeling unusual nucleic acid structures. 3, 6

[PJR*14] PARULEK J., JÖNSSON D., ROPINSKI T., BRUCKNER S., YNNERMAN A., VIOLA I.: Continuous levels-of-detail and visual abstraction for seamless molecular visualization. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 276–287. 3, 5

[SLJ*98] SUSSMAN J. L., LIN D., JIANG J., MANNING N. O., PRILUSKY J., RITTER O., ABOLA E.: Protein data bank (pdb): database of three-dimensional structural information of biological macromolecules. *Acta Crystallographica Section D: Biological Crystallography 54*, 6 (1998), 1078–1084. 8

[SZA*14] SHEPHERD J. J., ZHOU L., ARNDT W., ZHANG Y., ZHENG W. J., TANG J.: Exploring genomes with a game engine. *Faraday discussions 169* (2014), 443–453. 3