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

Volume 33 (*2014*), *Number 2*

CellVIEW: Multi-Scale Biomolecular Visualization With a Game Engine

007

Abstract

Pork chop doner chicken, tail shankle t-bone tri-tip. Ground round jowl pork beef, short loin tail hamburger shoulder shankle tri-tip. Cow cupim ribeye bacon pancetta landjaeger prosciutto. Pork belly cow tongue corned beef pastrami venison spare ribs frankfurter t-bone. Pig sirloin beef ribs, biltong meatball alcatra short loin shank sausage hamburger tail cow pork loin jowl tri-tip. Pork chop salami pig, tri-tip ground round boudin shankle tail leberkas filet mignon.

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

1. Introduction

In visualization sciences, datasets are constantly increasing in size and complexity, calling for new methods that could handle the large quantity of information to display. 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. However biologists are struggling with the viewing of these large scale datasets because the tools they are usually accustomed to cannot achieve interactive frame-rates for such heavy data loads. Foreseeing the future increase of the size of their dataset they would need the ability to quickly render billions of atoms while the current state of the art tools are only able to render up to hundreds of millions of atoms on commodity hardware.

Previous works have already presented methods that can render large datasets with up to billions of atoms at interactive framerates [LBH12] [FKE13] [LMPSV14]. However to our knowledge, these techniques were either not publicly available or they remained in the prototyping stage. Indeed, a very cumbersome task for researchers is to release of the source code once the article has been published. Presented techniques are often just a proof-of-concept that would require additional work, such as software development or documentation writing to ensure a maximum degree of reproducibility. 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 will unlikely be able to use state-of-the-art techniques in their work.

It exists a large array of tools that are employed by biologists which are extensible and could potentially bring state-of-the-art techniques to end users. Yet it has been highlighted that these solutions usually lack flexibility and access to low-level graphics API and features which is crucial for robust GPU computing [GKM*15]. In the domain of interactive entertainments on the other hand, professional content creation frameworks have emerged over the last years, such as Unity3D or Unreal Engine. These tools provide access to low-level graphics API which is missing in other solutions while remaining flexible and user friendly. They also allow to easily share and deploy projects across multi-disciplinary teams, and have greatly contributed to the democratization of real-time computer graphics.

CellVIEW is a real-time visualization tool which 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 utilize a game engine as development platform we are able to release our tools faster since our projects only consist of a few scripts and shaders. We are therefore 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 researcher too, thus breaking the barriers caused by heterogeneous toolset usage across research departments.

We implement a method for fast rendering of very large biomolecular scenes inspired by the work of Le Muzic et al. [LMPSV14]. By introducing efficient means to reduce the amount of processed geometries we manage to obtain a average speed-up of 600 percent over the previous work jumping from 10 fps up to 60 fps for similar sized datasets. We also showcase our technique on real large-scale scientific data such as entire viruses and which is obtained using specific modelling tools. Additionally we provide the means for accurate shape abstraction of protein data in order to reduce visual clutter and improve the visual appeal of the results. Finally we demonstrate the use of our technique to also render various types of large biomolecular structures such as strands of nucleic acids or bi-lipid membranes.

2. Related Work

We cover the related work in two sections in the first part we will review state of the art techniques that are designed for interactive rendering of large scale particle-based data. In the second section we will cover the use of game engine in the domain of biomolecular visualization and we will discuss how could the community benefit from the use of such tools.

2.1. Particle-based Molecular Visualization

There is a multitude of frameworks to display biomolecular data, however most of them are focused on displaying rather small-scale datasets only. It is possible using efficient ray tracing approaches to interactively render large-scale particle data on super-computers, the main limitation being the size of the memory. However such hardware is unfortunately not accessible to most people for obvious reasons.

On the other hand, commodity graphics hardware packages impressive computing capabilities on one board and which are have been used by Grottel et al. [GRDE10] to display large-scale datasets. The acceleration is based on an efficient two-level occlusion culling scheme and have demonstrated the rendering of hundreds of millions of atoms in real-time. On the first level they cull entire groups of atoms that are hidden by other elements via hardware occlusion queries (HOQ). On the second level they use hierarchical z-buffers (HZB) to cull individual atoms that are hidden within the visible groups of particles. This method is rather designed for generic purpose particle-data, biomolecular data however often exhibits repetitive structures which could be displayed more efficiently via instanced rendering.

Lindow et al. [LBH12] have first introduced a method capable of fast rendering of large-scale atomic data up to several billions of atoms. Rather than transferring the data from CPU to GPU every frame they store the structure of each type of molecule only once and utilize instancing to repeat these structures in the scene. For each type of protein a 3D grid structure containing the atoms positions is created and then stored on the GPU memory. Upon the rendering, the bounding boxes of the instances are drawn and individually raycasted similarly 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, while they could have probably benefited from a culling on the instance level too. With their new improvement they managed to obtain 3.6 fps in full HD resolution for 25 billions of atoms on a NVidia GTX 580 while Lindow et al. managed to get around 3 fps for 10 billions 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 billions of atoms at 10 fps in full HD resolution on a NVidia GTX Titan. They utilize 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 accordingly to the camera depth. To reduce the geometry processing, for each molecule 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 guaranty to preserve the overall shape of molecules for highly simplified structures.

We base our work on the later technique mostly because of its versatility and its ease of implementation. We introduce new set of technique that are improving rendering time over the previous work by a ratio of 600 percent. Additionally we introduce a new level-of-detail scheme that preserves the original shape for highly simplified protein structures and which provides and more accurate depiction of the scene. Finally we demonstrate the use of this technique to display other types of structures such as nucleic acids strands and also large bi-lipid membranes that do not feature repetitive structures.

2.2. Game Engines and Biomolecular Visualization

The visualization of biomolecluar structures often leads to the use of 3D graphics, preferably in real-time for interactive exploration. To facilitate the development of such applications, many generic visualization framework exists [AGL05], as well as frameworks dedicated to biomolecular visualization only [Sch10] [HDS96]. However it has been highlighted that these solution usually lack of flexibility and access to low level-api graphics features which is crucial for robust GPU computing [GKM*15]. Although game engines are not yet the perfect all-in-one solution for 3D graphics applications they are a good contenders to traditional visualization framework nevertheless.

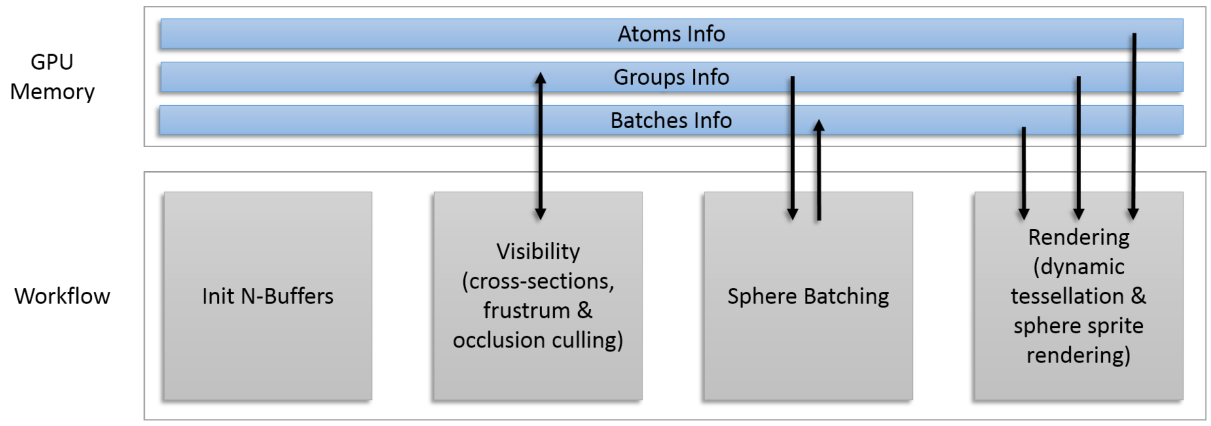
GPU Memory
Atoms Info
Groups Info
Batches Info
Workflow
Init N-Buffers
Visibility (cross-sections, frustrum & occlusion culling)
Sphere Batching
Rendering (dynamic tessellation & sphere sprite rendering)

Figure 1: *Description of the figure goes here*

In the domain of biomolecular visualization a few articles are highlighting the fact that they have used a game engine to develop their applications. Shepherd et al. [SZA*14] have developed a viewer to interactively view 3D genome data. Their visualization is multi-scale 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 crossing [LMWPV15] in 3D and in real-time.

The idea of developing a tool for molecular visualization with a game engine is not novel. Similarly to our work, Baaden et al. [LTDS*13] developed a molecular viewer based on a the Unity3D game engine which offers powerful, artistic and illustrative rendering methods. There primary intention was to democratize biomolecular visualization thanks to the use of a more intuitive and user friendly framework. Their tool, although just a proof-of-concept have managed to prove that game engines are also capable to foster serious visualization projects.

A noticeable difference between CellVIEW and UnitMol, is that our tool is not the direct product of a game engine but is integrated in the engine instead. While UnityMol is an application that need to be compiled in order to work, our tool is also able to display biomolecular scene directly in the "What you see is what you get" (WYSIWYG) editor of the Unity3D engine. Thus coexisting with the engine tool set and providing us a nice set of perks which we can directly use to enhance the quality of our visualization.

3. Pipeline Overview

Although our work is directly embedded in the game engine toolset, we do not rely on the available rendering pipeline because it is mostly designed for mesh data. Therefore we implement a custom pipeline for efficient rendering of large particle data. Before proceeding with the description of the pipeline we deemed important to clarify the way we arrange the data on the GPU. There are several types of buffers which we store on the GPU: On the one hand we have information relative to the groups of atoms, these groups represent either entire proteins, DNA segments or generic group of atoms. This information is stored in large GPU buffer and include the position, rotation and various information such as the type or the culling flags. On the other hand we have the atomic data which is stored in a another large buffer and which includes position and atom types only.

When doing instanced rendering of proteins it is possible to fetch, the address of the first atom of the each protein in the atomic data buffer as well as the number of atoms to draw. Additionally our pipeline is also capable of rendering large elements that do not exhibit a repetitive structure, such as large bi-lipid membranes. In this case we gather atoms in generic groups based on their spatial locality and we store the position of the group on the GPU as well as the atom count and the address of first atom of the group. In-depth explanations on how we render efficiently large and non repetitive structures are given in section 5.2. We also store render commands in a dynamically-sized buffer, dubbed batch buffer, which is filled and cleared every frame.

An overview of the pipeline is shown in Figure 1. As a first step we compute the N-Buffers which we subsequently use for the occlusion culling. Then we determine the visibility of the groups of atoms in step 2, this step includes cross-sections, frustrum and occlusion culling. In step 3 visible groups are divided into batches with a maximal size of 4096 spheres according to their LOD level and this information in then streamed to the batch buffer. Finally in step 4 the rendering pipeline reads the batch buffer and gather relative information from the group and atom buffers in order to output the right number of spheres.

4. Efficient Two Level Occlusion Culling

A key aspect in optimising the rendering of large number of elements is efficient occlusion culling. Many articles in the computer graphics literature have largely covered this topic and it has also been applied to the specific domain of biomolecular visualisation [GRDE10]. State-of-the art techniques in very large scale biomolecular visualization [LBH12] [FKE13] [LMPSV14], however, do not truly leverage their rendering with state-of-the art occlusion culling methods. Because of the high amount of processing which required by these techniques, it is preferable to ensure that the heavy load of most heavy load in only performed for visible elements in order to spare GPU computation. Therefore we propose to improve the technique introduced by Le Muzic et al. [LMPSV14] with an efficient occlusion two-level culling scheme inspired by Grottel et al [GRDE10].

4.1. First Level Culling

First level culling is achieved by Grottel et al. [GRDE10] on the global level by grouping atoms together via a uniform grid and by computing the visibility of the groups. When applying this approach to biomolecules only, groups of atoms can be more intuitively represented by entire proteins, DNA segments or bi-lipid membrane patches. Thus easing the task of grouping atoms together and also providing a tighter fit than uniform grid partitioning.

In their implementation they used hardware occlusion queries (HOQ) to determine the visibility of the groups. In our case, when dealing with several millions of proteins the GPU driver overhead caused by HOQ would very likely cause a severe bottle neck is our pipeline. We implemented an occlusion culling technique named N-Buffers [Déc05] instead and which is a variant of the well known hierarchical z-buffer (HZB) culling method [GKM93]. An great advantage of the N-Buffers technique over traditional HZB is that buffers may be generated from arbitrarily sized depth buffer. Therefore we can exploit temporal coherency and reuse the depth from the previous frame in order to determine occluders and occludees at given frame.

Prior to the occlusion test, we cull groups of atoms that are lying outside of the cross-sections and frustrum planes in order to avoid unnecessary queries. All the queries are then processed efficiently in a single pass via a compute shader. For each group of atoms we check the visibility of the bounding box against the N-Buffers and the corresponding culling flag stored on the GPU is updated accordingly.

4.2. Sphere Batching

Tessellation shaders are used to dynamically inject geometries in the pipeline. The maximum number of output vertices of a single tesselation shader is currently limited to 4096 on high-end graphics hardware. In order to handle proteins and groups of atoms that are larger that this, Le Muzic et al. [LMPSV14] used the geometry shaders to inject the remaining atoms. During out tests we found out that this approach is rather inefficient because of the relatively low performance of geometry shaders compared to tessellation shaders. Instead, we propose to split molecules and groups of atoms greater than 4096 atoms into smaller batches with a maximal size of 4096 vertices.

Batches are analogous to render commands, each buffer entry stores the number of spheres to render and the address of the first sphere to render. Entries also contain the id of the corresponding group which is needed to fetch important information such as position or rotation of the group. All the batches are streamed into a separate buffer and the number of batch per group depends on the number of atoms, the current LOD level and the visibility. Groups of that have priorly been culled during the first level culling will not output any batch thus reducing processing overhead. Once the batching over, all the render command are issued in a single call, allocating one vertex shader per batch. Subsequently the tessellation shader will dynamically injects the corresponding spheres for the given batches.

4.3. Second Level Culling

For the second level of culling Grottel et al. were computing occlusion atom visibility inside each group order to discard the hidden atoms. When rendering a large quantity of atoms a common method is to render camera facing 2D sphere impostors rather then meshes because they only require a few vertices for the same results [reference to sphere impostors]. This method require to modify the output depth value of the fragments in order to imitate the shape of a 3D sphere. Traditionally the rendering pipeline would automatically cull hidden fragment via early depth test before they are even processed.

Unfortunately fragments that are modifying the depth value cannot normally be subject to early depth culling since the GPU is not able to predict in advance whether the fragment will be visible or not. In order to cull hidden atom Grottel et al. used a HZB-based culling instead. Thanks to the progress in graphics hardware it is now possible to activate early depth rejection when a fragment is modifying the depth value. This feature is called the conservative depth output and if very useful when drawing a large number of depth sprites. As a result we do not have any overhead that would be caused by HZB culling and we also benefit from a greater precision culling. It is also worth mentioning that in order to benefit from the two-level occlusion culling Grottel et al. had to generate an additional depth pass which is no longer required in our pipeline.

Once activated, in order for conservative depth output to work the fragment must output a depth value which is greater than the depth of the 2D billboard. This way the GPU is able to tell if a fragment will be visible before even processing it,

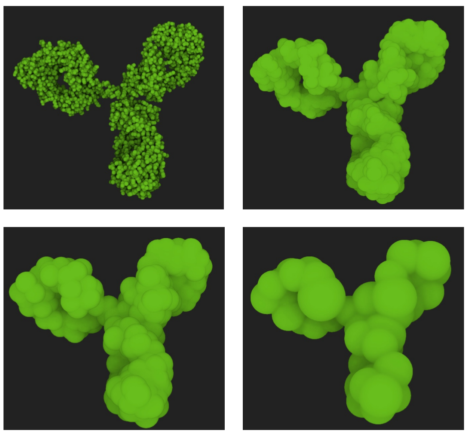

Figure 2:

by querying the visibility of the fragment internally. A description of the depth conservative output sphere impostor is given in Figure X. This feature does not reduce the load of geometry processing but this was also the case with HZB-based culling use by Grottel et al. During our tests we observed no noticeable difference in performances when drawing flat impostors and when drawing depth impostors

5. Efficient Rendering of Biomolecular Data

Biomolecular data comprise many different types of structures, such a proteins, bi-lipid membranes or DNA strands. Although these structures are all made out of atomic particles they aslo exhibits different properties and must be handled differently. In this section we explain how to display efficiently these structures in order to provide the most accurate depiction of cell biology.

5.1. Proteins

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 represents a challenge for real-time rendering. Since we rely on the GPU rasterization pipeline to render large quantities of particles, the graphics hardware undergoes heavy geometric processing which may cause bottlenecks. In order to reduce the computation one could simplify the molecular structure according to the camera distance and thus diminishing the amount of processed geometry. To simplify molecular structures lemuzic et al, were discarding atom uniformly along the backbone of the protein while the radius of remaining atoms was increased in order to compensate the voids. Although naive, this approach has the benefit to be fully dynamic and to work without any precomputation. On the other hand with high degrees of atomic decimation proteins start to exhibit incoherent shapes, and does not truly provide an accurate view of the scene.

Representing shapes using a set of sphere efficiently has been a subject of interest since the early ages of computer graphics [missing reference], mostly because of their simplicity. Since we are using spheres to render individual atoms it appears logical to us to also use this type of primitives to the represent simplified shapes of molecules in our level-of-detail scheme. Sphere-based shape simplification has already been applied to molecules for the purpose of accelerating molecular dynamics simulations [x]. In the visualization domain, Parulek et al have introduced the use of clustering algorithms to simplify the shape of molecule and also to reduce the number of primitives to render.

Clustering algorithms offer a very good decimation ratio as well as visually pleasing shape abstraction because they tend to remove high frequency details while preserving the low frequency ones. Shape simplification, or abstraction is also widely used by scientific illustrators when depicting cell biology. Indeed, at a certain scale is not longer necessary to showcase full atomic details as these may not be perceived from the distance and contribute to visual clutter. In the illustration of David Goodsell for instance (see figure) we can see proteins quickly converging to an abstract shape rather than showing full atomic details.

Our level-of-detail is therefore twofold, not only it accelerates the rendering of a large number of proteins, but also provides a way to clarify the scene with simpler shapes, thus procedurally imitating the mindset of scientific illustrators. Although we wish to simplify the view in the large scale we still do not wish to omit important structure information when the viewer is closely looking at a protein. Therefore we implemented our level of detail in order to provide a smooth transition between the different levels from full atoms representation to very abstracted representation. Prior to the rendering cluster spheres are computed for each molecule and then uploaded to the GPU memory in dedicated buffer.

An address pointer as well as the number of clusters spheres of each type of protein and for each LOD level is also stored on the GPU. Hence when injecting spheres in the tessellation shader the LOD logic is able to seamlessly fetch the right set of clusters spheres by reading the GPU memory form the correct buffer location accordingly. We empirically found that four levels of details where sufficient with our current dataset, and that the degree of atom atomic decimation we used offer a good compromise between visual quality and performance. The first level includes 100 percent of the atoms of a protein, 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.

In order to compute atom clustering for the second level we do not rely yet on heavy clustering algorithms yet, instead we rely on the spatial proximity of the atoms inside single ligands. This simplification is also referred as coarse grain and has already been use in biomolecular visualization to accelerate the construction of molecular surfaces [Interactive Visualization of Molecular Surface Dynamics]. To group atoms in clusters we take into account neighbouring atoms in a single ligand, determine their bounding sphere, and then proceed with the next ligand. For the third level we reuse the results obtained for the second level, and proceed similarly but on the level of protein chains instead. For the last level we wanted to dramatically reduce the number of spheres and therefore the previous approach, based on local proximity only, would not perform well enough for this task. We employ k-mean clustering instead, in a similar way as Parulek et al. The major issue with such clustering algorithm is the slow computing speed when dealing with several thousands of spheres. To speed up the clustering we therefore reuse the results of the third level as input. Also since the number of seeds, i.e the desired number of clusters for the k-means algorithm, is quite low–0.5 percent of the total atom count–the clustering performs fairly fast. It is worth mentioning that although the LOD levels computation is done off-line it is rather light to compute, and could eventually be applied in real time when dealing with dynamic data with only little overhead.

5.2. Bi-lipid Membranes

Bi-lipid membranes are an inherent part of cell biology, however they exhibit different properties than proteins must be handled accordingly. A previous work have already introduced dynamic patching of repetitive membrane portions on arbitrary surfaces[x]. Such technique could greatly facilitate the rendering in our case because of the use of repetitive structures. However we wanted to showcase that our technique is also applicable to non repetitive structures too. One advantage of our technique over ray casting is that it does require any supporting grid structure. Therefore, if we were to render very large non-repetitive structures via ray casting like in [][] the entire dataset would have to be stored in 3d grids on the GPU which would consume a lot of GPU memory. In our case only the raw data is needed plus a little extra information about the groups of atoms, thus we can use the GPU memory more efficiently and display much bigger datasets that exhibits non repetitive structures.

We compute the static structure of bi-lipid membranes using the lipid wrapper tool[x]. The resulting structures we obtain with this tool can be very large and do not exhibit repetitive structures which we could reuse for instanced rendering. Therefore we most store the entire membrane structure on the GPU. In order to benefit from efficient occlusion culling, however we may still group the atoms together just like in the case of protein rendering. The main different here is the fact that each group must have its own address pointer to the first atom of the group.

It is possible to group the atoms by molecules, but given the small size of lipid molecules, between 20 and 50 atoms only, the resulting number of groups would be too large and the size of the groups too small for efficient coarse level culling. Because the atomic data we obtained from the lippid wrapper tool is already sorted based on spatial locality it is rather easy to cluster atoms in coherent and larger groups. For the tested membrane we were able to gather lipid molecules by groups of 2000 atoms on average. Although the data is not instanced as with proteins the rendering is done similarly and follow the exact same pipeline steps. The only different being that we do not pre-compute LOD of details for bi-lipid membrane, instead we employ dynamic atom reduce as used in lemuzic et al. Since the shape of a membrane is much less complex than proteins, the shape approximation does not suffer too much from the use of naive atom reduction.

5.3. DNA Strands

Animating proteins is fairly straightforward, one simply needs to update the position and the rotation the instances which can be either done in parallel directly on the GPU or by computing it on the CPU and transferring the data to the GPU afterwards. In both cases it is not necessary to change the atomic structure of the protein unless of course when viewing the results of MD simulations. In the case of DNA the positions of the control points of the DNA path are highly influencing 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 performing on the CPU[][][], requiring the whole nucleic acids data to be transferred to the GPU after each 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 over a million control bases. Although the study of DNA structures have reveal many different types of foldings requiring complex modeling algorithms, the most commonly recognizable shape is the B-DNA and exhibits a regular structure which is simple to model: a spacing of 3.4 A and a rotation of 34.3 degrees between each base.

Since we are privileging render speeds and artistic composition over accuracy we do not provide support for other realistic types of folding, similarly to the graphite life explorer[]. Once again we leverage the power of the tessellation shader to speed up the rendering. So far we only used tessellation to efficiently instantiate data priorly stored on the GPU memory. However it would be rather trivial to include the rules that are proper to B-DNA modeling to procedurally generate a double helix structure based on control points and atomic data of nucleic acids. Thus, the data transferring operations to the GPU are dramatically reduced, and dynamic level-of-details can be applied too, allowing the rendering of a very large quantity of DNA.

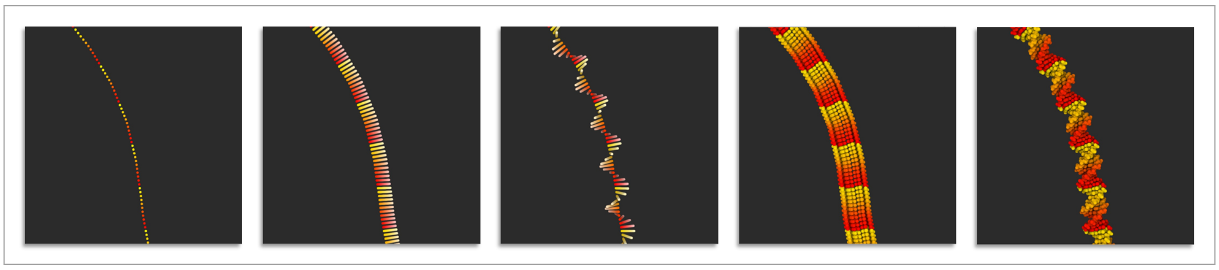

Figure 3:

The workflow which we employ is depicted in Figure X and is defined as follow.

Resample control points positions and compute smooth normals on the CPU and then upload info to the GPU. Segments are then drawn in a single pass allocating on vertex shader per segment. The vertex shader reads the control points for a given segment of the DNA path plus the adjacent control points needed for smooth cubic interpolation. (p0, p1, p2, p3). The vertex shader determines the position of each base using uniform sampling along the cubic curve segment. The vertex shader passes the position of the bases to the tessellation shader. The tessellation shader computes the normal vector of each base using linear interpolation between the control points normals. The tessellation inject the atomic data corresponding to the pair of nucleic acid then position and rotates accordingly. Finally the geometry and fragment shader turns the vertex primitives into sphere impostors.

A well know challenge when dealing with 3D splines is to determine a smooth and continuous frames along the whole curve. Any twists or abrupt variation in frame orientation would cause visible artefacts due to irregularities in the DNA structure which we must avoid as much a possible. In order to compute a smooth normal vector along the curve, we first determine the normal direction for every control points. Then we browse each control point in serial on the CPU and we rotate the normal direction vector around the tangent vector in order in order to minimize the variation angle compared to the previous control point normal. The pseudo code for computing the minimized rotation normal vectors is given in appendix X. The recalculated normals are then uploaded to the GPU along with the control points positions, and during the instantiation of the nucleic acids we obtain the normal vector of a nucleic acid by linear interpolation of the two segment normal vectors.

When instancing individual pairs of nucleic acids in the tessellation shader, we first align the atoms along the tangent vector and then rotate around the tangent vector in order the generate the double helix strcture of the B-DNA. We always orientate the first base of a segment according to the normal direction only, while the subsequent bases are all oriented to the normal direction first and then rotated with an increasing angular offset of 34.3 degree according to the tangent. The formula to compute the rotated normal vector of a given base is defined as follow:

The last base of a segment must therefore always perform a rotation of $360 - 34.3$ degrees around the tangent so that it connects smoothly to the first base of the next segment which is only rated by the normal vector. This constraint requires the number of bases per segment to be fixed, therefore we resample the control points to ensure that the length of segments along the curve is uniform before uploading it to the GPU.

The result of the procedural generation of B-DNA is given in figure X as well as a visual explanation of how we manage to compute the double helix in parallel.

Given the fact that all the bases of a segment must perform a full revolution to connect to the next segment it is trivial to determine the number of bases per segments as follow:

Given the number of bases per segments and their spacing it is also trivial to determine the length of a segment:

Although we space the all the control points with a uniform distance which correspond to the proper length needed for generatic B-DNA structure, the length of the interpolated cubic curve for that segment will be greater than this depending of the curvature, thus increasing the spacing between the bases.

However did not find this to be truly visible as it did not cause any major disturbing artefacts, probably because consecutive segments in our dataset did not showcase critically acute angles so the overall curvature of the path remained rather smooth.

6. Perks of a Game Engine

Ambient occlusion Talk about POINT AO

High Quality Depth of Field Image-based antialiasing, Mesh based pipeline for free Scripting

7. Results

7.1. Use cases

HIV + blood serum (20 billions) 30/60 fps Mycoplasma DNA

7.2. Expert feedback

7.3. Discussion

8. Conclusions and Future Work

References

[AGL05] AHRENS J., GEVECI B., LAW C.: 36 paraview: An end-user tool for large-data visualization. *The Visualization Handbook* (2005), 717. 2

[Déc05] DÉCORET X.: N-buffers for efficient depth map query. In *Computer Graphics Forum* (2005), vol. 24, Wiley Online Library, pp. 393–400. 4

[DN09] DECAUDIN P., NEYRET F.: Volumetric billboards. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 2079–2089. 2

[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. 1, 2, 4

[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. 4

[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. 1, 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. 2, 4

[HDS96] HUMPHREY W., DALKE A., SCHULTEN K.: VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics 14* (1996), 33–38. 2

[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). 3

[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. 1, 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. 1, 2, 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). 3

[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. 3

[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. 2

[Sch10] SCHRÖDINGER, LLC: The PyMOL molecular graphics system, version 1.3r1. August 2010. 2

[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