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

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

CellView: State-of-the-Art Large Scale Molecular Visualization Inside 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, and therefore new visualization methods are needed to handle the large quantity of information to display. Computational biology already offers the way to generate large and static models of cell biology, such as viruses or entire cells on the atomic level. However biologists are beginning to struggle with the viewing of these large scale datasets, and the tools they are usually accustomed to do not suffice anymore. Foreseeing the future increase 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 hundreads millions of atoms on commodity hardware. The visualization of such data in real-time could also serve other purposes than science-driven data exploration, it could also be used to showcase the data to a lay audience in a museum for instance.

In the literature there are already a few state-of-the techniques that allow fast rendering of billions of atoms but to our knowledge those technique are still not accessible to biology researchers because they remained in the prototyping stage. Furthermore, those techniques have only showcased rendering of proteins so far while living organisms also features different types of biochemical elements, such as bi-lipid membranes and DNA/RNA Also the aforementioned technique started to reach the limits of their capabilities when displaying several billions of atoms, demonstrating rather low frame rate which would be critical for interactive showcasing. Finally, despite the simplification schemes offered by those techniques in order to accelerate the rendering speed they still lack the means to perform shape simplification which would be preferred when observing a large structure in its entirety.

In order to reach a maximum number of people we which to implement our tool in the most familiar and user friendly framework available. There is a lot of molecular visualization packages that already exists, each one with their own strengths and weakness, and it is rather hard to settle for the most preferable solution. In computer graphics however, thanks to the democratization of game engines, more generic and universal tools have emerged which allow to easily share, extend and deploy projects. A few projects have already started to set the trend of utilising game engines for interactive visualization of cell biology. Our work follows this trend at the exception that our visualization tool is not the product of the engine but rather integrated in the engine tool set itself, allowing a wide variety of users to utilize our tools and thus democratizing state-of-the art molecular visualizations techniques.

2. Cell Biology Modelling Background

While molecular structure may be obtained via crystallography, in many cases this acquisition method is not applicable and therefore structural data must be generated. This is the case for large bi-lipid membranes, DNA and RNA strands and also large meta-structures, such as viruses or cells. In the case of bi-lipid membranes it is possible to generate large structures thanks to the lipid wrapper tool. The tool is accepts a mesh surface as input as well as a small portion of membrane. In order to cover the entire surface with lipid molecules they project each triangle of the mesh onto the patch and select lipid molecules included in the projected triangle. Since the computation is done off-line there is no real limitation regarding the atom count of the resulting structure other than computation times.

However for very large structures, over a billions of individual atoms, given the lack of storage on GPU memory the real-time visualization might be challenging, especially fn other structures such as proteins and DNA must be displayed too. To overcome these limitation [Bjorn summer] have presented a method that dynamically instantiates a portion of membrane on arbitrary surfaces, this approach exploits the repetition of the membrane structures in order to save memory space. DNA is also a molecular structure which is hard to capture in its entirety because of the overly large length of the strands. For these reason DNA strands are traditionally created using geometry modelling tools [3DNA, Nucleic Acid Builder]. Those tools allow rigorous modeling of DNA stands using curve control points as a input. GraphiteLife-Explorer is a another DNA modelling tool especially designed for user input driven modeling of static DNA strands. Their approach is principally aiming at artistic composition of 3D scene and therefore their method is less accurate but also faster than other methods. Via user input and manual bezier curves editing tools they manage to recreate DNA stands and to export it in standard molecular structure format. The resulting data, e.g the position and rotation of the DNA bases are uploaded on the GPU for fast rendering. The computation of the position and rotations of the individual DNA bases is done on the CPU but the reconstruction of the strand upon control points modifications remain interactive for medium size DNA strands nevertheless. Foreseeing the need to render dynamic DNA strands in large quantities, such approach would be very likely to cause a bottleneck in the pipeline, both in computation, done on the CPU and in transfer times to the GPU. Therefore we introduced a new GPU-based technique which relies on dynamics instancing of DNA bases along a path and which we describe in section X. The presented approach based on our molecule rendering pipeline, rely on the extensive usage of tesselation shader to dynamically generate the double helix structure. The main advantage being that only the DNA path has to be uploaded to the GPU instead, and that the computing of the double helix can benefit a performance boost due to the power of GPU parallel computing. Our technique, similarly to the graphiteliteexplorer offers is aiming at offering a believable depiction of DNA strands rather than a strictly accurate reconstruction. Atomic structures of molecules are usually obtained via crystallography by capturing the 3D position of each individual atom positions. This method is effective for relatively small sized molecules, but is not able to capture very large structures such as cells in their entirety. Because of the challenge of acquiring such data, biologist started to model it instead. A well know modelling tool utilized for dynamic generation of large static structure is CellPACK [x]. Based on real scientific data such as concentration and spatial distribution they managed to generate a whole model model of the HIV virus via a procedural packing method based on collisions constraints. They compare their results with fluorescence microscopy images of real organisms to validate their results.

3. Related Work

3.1. Large scale atomic visualization

It exists a multitude of frameworks to display molecular data, however most of them are rather focused on displaying small scale particle data. It is possible using the proper ray tracing approach to render large scale particle data on the CPU, the main limitation being the size of the memory. CPU memory limitation may bypassed by using supercomputers, however these are expensive and therefore not quite accessible to most people. On the other hand commodity graphics hardware are packing impressive computing capabilities which have been used, instead of CPU to display a large amount of particles. One the most active framework in terms of state-of-the visual computing research is MegaMol, and can efficiently render large molecular datasets. They are utilising a two level occlusion technique and are able to render up to hundreds of millions of atoms in real-time. The key of their two level occlusion is to reduce the amount of information which is sent to the renderer. On the first level they are trying to cull groups of atoms that are not visible on screen using hardware occlusion queries, comparing the depth of an object with the previous depth buffer. On the second level they are culling individual atoms that are not visible within a visible group of particle from level one. This tool is a generic-purpose particle data viewer which is not aiming at only displaying molecular data. Hence it does not use utilize instancing of repetitive structures to spare memory which limitates the rendering for that specific case, due to the limitation of GPU hardware memory sizes. While this tool does not meet our requirements in terms of capabilities, their two level occlusion technique which we inspire in this work has shown to play an essential role in the rendering performance.

Lindow et al. have introduced a rendering technique that uses repetitive structures to their advantage in order to enable interactive rendering of large-scale atomic data up to several billions of atoms. Rather than storing the entire dataset on the GPU they only store one molecule structure of each type and utilize instancing to repeat these structures in the 3D scene. For each type of protein a 3D grid structure containing the atoms positions is initially filled and stored on the GPU. During the rendering the bounding box of each instance of each atom structures are drawn and individually raycasted similarly to volumetric impostors. Subsequently Falk. et al presented an similar method with improved depth culling and hierarchical ray casting to speed-up the rendering of elements that are located far away and do not require a full grid traversal.

Le Muzic et al, have also showcased a technique they relies of repetitive structures but tehy draw atomic data stored on the GPU through the normal rasterizing pipeline instead. To render large-scale atomic data they utilize the tessellation shader to dynamically inject atoms in a similar way as [Lampe].

The speed of the technique relies heavily on level-of-detail schemes, in order to increase the rendering speed they reduce the number of atoms according to the distance with the camera. This operation is done on-the-fly in the tesselation shader which can dynamically control the number of injected atoms. The molecular structure simplification scheme employed for the level-of-detail is rather trivial in this work. To reduce the number of atoms they omit atoms uniformly along the protein chain, and they increase the radius of the remaining atoms to compensate the loss of volume. This technique, although it does not require pre-computation, offer good results for low number to decimated atoms but exhibits artifacts when many atoms are remove because they is not guaranty that the overall volume of the protein will be preserved.

In our work we present a technique which is similar to the later technique mostly because of its versatility and its ease of implementation and also because it showcased very good real-time performance. We introduce new technical improvements that are greatly improving rendering time over the previous work which we describe in section X, additionally we show how could this technique be use to perform level-of-detail that preserve the original shape and also provide and more comprehensive shape simplification for a better visual communication. Finally we demonstrate the use of this technique to display other types of structures such as bi-lipid membranes and DNA.

3.2. Use of Game Engines in molecular visualization

The visualization of structural information in biology 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, as well as frameworks dedicated to molecular visualization too. However those solution are generally not suitable for intensive GPU computing[cite megamol] that require a lot of flexibility and access to low level-api graphics features, which is why many people tend to implement their own prototyping framework from scratch.

But dealing with the problem by implementing a solution from scratch might not always be the most preferable option either. It offers a lot of flexibility to begin with, but it also requires a lot of work in order to make the tool available to end users usually from another domain and not familiar with such technologies. One must take care of portability issues, reimplementation of a lot of features, and also maintain the framework constantly in order to provide support for the bleeding edge features. Additionally creating a new framework, and thus adding another solution to the list of already existing solutions results in a highly heterogeneous toolset usage across different research departments, which make it cumbersome to share and deploy techniques between researchers and also to end users.

On the other hand, outside of the visualization community, many good quality game engines have emerged over the last years. Their simplicity and low prices have contributed to the democratization of computer graphics development, which has always been a bit out of reach before. They offer a generic and universal tools to create either 2D or 3D real-time graphics program, and are not only used to develop game anymore. Visualization scientists have also started to develop project with such tools, they have been seduced by their ease of use, in both programming and portability, which allows them to focus on doing actual research rather than complex software design.

Although those engine are still far from being the perfect all-in-one solution they are a good contenders to traditional visualization framework nevertheless, and also offer ease of access to GPU features for fast computation, which is critical for high performance computing. In the domain of molecular visualization a few articles are highlighting the fact that they have used a game engine to develop their program. Jeramiah et al. [Exploring Genomes with a Game Engine] have showcased a 3D viewer that showcase genomes in real-time, their visualization is also multi-scale and allow visualize large amount of data thanks to the implementation of a level-of-detail scheme. Various works on illustration/visualization of biological processes also mention the use of a game engine to visualize polymerization[ivo] or membrane crossing [me] in real-time and in 3D. Baaden et al. [unity mol] have also developed a molecular viewer based on a game engine, their primary goal was to find out weather it would be possible to implement a state-of-the art viewer with such technology, and they managed to. A noticeable difference between XXXX and UnitMol is that our tool is not the direct product of a game engine but is integrated in the editor instead. While UnityMol is an application that need to be compiled in order to work. Our tool allows to display large molecular structures in the editor too, thus coexisting with the engine tool set and provide us a very nice set of perks which we use to enhance the quality of our visualization and which we describe in section X. In this paper we are presenting a tool to visualize large scale molecular scene using a game engine too, the noticeable difference with UnityMol is not only that we can display for information, but also that our tool in not the direct product of a game engine.

4. Pipeline Overview

Before proceeding with the description of the pipeline we deemed important to clarify the organisation of the data on the GPU. There is two main type of buffers which we stored on the GPU: On the one hand we have information relative to groups of atoms which may represent either entire proteins 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 separate buffer and which includes position and atom type only. We also store information relative to level of details in similar buffers in the case of proteins data, more detail about this is given in section X. When drawing instanced proteins it is possible to access the atom count and the pointer which indicates the address of the first atom of the protein in the atomic data buffer.

Additionally our pipeline is also capable of rendering large structures that do not exhibit repetitive structures, such as large bi-lipid membranes. In this case we gather atoms in generic groups and store the position of the group on the GPU as well as the atom count and the address pointer of the first atom of the groups. More detail on how we render efficiently large and non repetitive structures is given in section X.

An overview of the pipeline is given in Figure X. As a first step we compute the different levels of the N-Buffers. Then we determine the visibility of the groups in step 2, this includes cross section culling, frustrum culling and occlusion culling. In step 3 visible groups are divided into batches of maximal size 4096 according their number of atoms and the level-of-detail and this information is fed to the batch buffer. Finally in step 4 the rendering pipeline reads the batch buffer and is then able to gather relative information in the group buffer and the atom buffer in order to output the proper amount of atoms given a group of atoms and its level-of-detail.

5. 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 molecular visualisation [Grottel]. In the domain of very large scale molecular visualization however, related work such as lindow, falk, lemuzic, do not leverage their rendering fully with state-of-the art occlusion culling methods. Because the amount of processed geometry with these technique is always very critical, we must ensure that all the geometry going through the pipeline is actually visible in order to spare GPU bandwidth. Therefore we propose to improve the technique presented by lemuzic et al. with an efficient occlusion culling scheme which was inspired by Grottel et al. and their two level culling method.

5.1. First Level Culling

In Grottel et al the first level of culling is done on the global level by grouping atoms together via a uniform grid and determining the visibility of the groups. This approach has the advantage of working with all type of particle-based data. When applying this approach to molecular visualization 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 simple uniform grid partitioning.

In their implementation Grottel et al. used hardware occlusion queries to determine the visibility of the groups, the downside of occlusion queries is that each queries are issued from the CPU, causing GPU driver overheads. In their case overhead due to CPU-GPU invocation was very low because of the small number of queries they had to issue. In our case, however, when dealing with several millions of proteins to query this approach would be very likely to cause a severe bottle neck is our pipeline. Instead we implemented an occlusion culling technique named N-Buffers which is a variant implementation of the well known hierarchical z-buffer (HZB) culling. The idea behind HZB culling is to compute different mip levels of a given depth buffer. The mip levels are computed in cascade, for each level only the deepest depth value is retained instead of averaging when down-sampling an image region to the next miplevel. Thus efficient depth test of squared size bounding regions may be performed with only few texture lookups by comparing the depth of a region with the depth stored in the proper mip-level. HZB is certainly usually less accurate than geometry based occlusion culling techniques but it is also much cheaper to compute in comparison which is preferred in our case due to the large number of queries that we must issue. An advantage of the N-Buffers technique over traditional HZB is that the depth mip-level can be generated from arbitrary sized depth buffer, which means that we can exploit temporal coherency and reuse the depth from the previous frame in order to determine occluders and occludees.

Prior to the depth culling test, we compute the N-buffer from the depth buffer of the previous frame, we also cull groups of atoms that are lying outside of the frustrum planes in order to avoid useless occlusion culling tests. Before proceeding to the next step we process all the occlusion queries in a single pass using a compute shader. We simply check the depth of the atom ground bounding volume against the N-Buffers and determine their visibility, a boolean flag stored on the GPU for each group of atoms is updated accordingly.

5.2. Sphere Batching

Upon drawing of the atoms tesselation shaders are unsed in order to determine how many atoms will be rendered for the given group of atoms, may it be a generic group of atoms or an entire protein. Consequently we may dynamically reduce the amount to processes geometry through 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, lemuzic et al, used the geometry shader to inject the rest of the atoms. During out tests we found out that this approach was rather inefficient because of the relatively low performance of geometry shaders compared to tessellation shaders for injecting primitives in the pipeline. Instead we propose to split coarse groups of atoms that a greater than 4096 atoms into smaller batches with a maximal size of 4096 vertices. The batching of spheres is done dynamically on every frame, and is also adaptive to the level of detail. In case the atom count of a molecule is reduced due to level-of-detail geometry simplification the number of batches for this molecule will dynamically adapt to the reduced number of atoms, thus reducing unnecessary geometry processing too. All the batches are streamed into a separate buffer in parallel, using stream output buffers, which offer en efficient way to output data on the GPU memory in a stack without the use of costly atomic operations.

The batch buffer contains all the needed information to render the spheres of a batch and is cleared at the beginning of every frame. Each buffer entry contains the id of the group of atoms, which is needed to fetch important information such as position, rotation or type, it also contains the address pointer the first atom of the batch, and the number of atoms of the batch. Once the batching over, all the batches are drawn in a single draw call, allocating one vertex shader per batch while the tessellation shader is taking care of injecting the sphere atoms. Since prior to the batching we have already determined the visibility of the groups we can easily discard the hidden ones during the process. As a result no batch for hidden molecules will be streamed to the batch buffer. This is a great advantage because without a stream output buffer of dynamic size the hidden groups would have to be discarded in the vertex shader instead which would cause unnecessary processing overhead.

5.3. Second Level Culling

For the second level of culling Grottel et al. were computing occlusion culling inside each group of atom in order to discard the hidden ones. 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 fragment in order to imitate the shape of a 3D sphere. Traditionally the rendering pipeline would automatically cull hidden fragment via earl depth rejection 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 get hidden atoms to be culled as they should they used a hierarchical z-buffer based in order to determine each atom visibility. Thanks to the progress in graphics hardware it is now possible to activate early depth rejection when a fragment is modifying the depth value, without any overhead that caused by HZB lookups. This feature is called the conservative depth output and if very useful when drawing a large number of depth sprites.

The only one condition to respect in order for conservative depth output to work is that the fragment must output a depth value which is greater than the depth of the 2D billboard, in this way the GPU is able to tell if a fragment will be visible before even processing it. A description of the depth conservative output sphere impostor is given in Figure X. This does not reduce the load of geometry processing but this also the case with traditional HZB-based culling. During our tests we observed no differences in performances when drawing flat impostors and when drawing depth impostors.

6. Efficient Rendering of Proteins

Proteins are key elements of biological organisms and thus it is important to visualize them in order to understand how those structures work. They are also present in fairly large quantities which represents a challenge for real-time rendering. As we rely on dynamic tessellation for the rendering of atoms the pipeline undergoing a lot of geometric processing which may cause bottlenecks. In order to reduce the vertex bandwidth one can reduce the number of atoms like in le muzic et al. In this previous work atoms were regularly spiked along the backbone of the protein and 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 pre-computation. On the other hand with high degrees of decimation the shape of the protein start to exhibit incoherent shapes, which is rather uneasthetical, 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 their simplistic geometric representation. Since we are using spheres to render individual atoms it appears natural to us to also use this type of primitives to the represent the simplified shapes of a molecules in our level-of-detail schemes. A few of these techniques have already been applied to the domain of molecular visualization [add reference]. Parulek et al had the idea to use clustering algorithms in order to simplify the shape of a protein and also to reduce the number of spheres to render.

Clustering algorithms offer a very good decimation ratio as well as shape abstraction because they tend to remove high frequency details while preserving the low frequency one. Shape simplification, or abstraction is also widely used by scientific illustrator when depicting cell biology. At a certain scale level is not longer necessary to showcase too much detail about the structure of individual proteins. In the illustration of david goodsell for instance (figure) we can see proteins quickly converging to an abstract shape rather than showing high frequency details which could clutter the view.

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 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 the level of details are computed for each molecules and then uploaded to the GPU memory in a dedicated buffer.

An address pointer and the sphere count for each type of protein and each LOD levels is also stored efficently on the GPU. Hence when injecting spheres in the tessellation shader the LOD logic is able to seamlessly fetch the right level of detail form the GPU memory by reading spheres infos form the correct buffer location. We deemed that four levels of details where necessary to obtain satisfying results with our current dataset.

*** List ***

The first level includes 100 percent of the atoms of a protein.

The second level is 15

The third level is 5

The fourth level is 0.5

***

In order to compute the clustering for the second level we simply 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 molecular visualization to accelerate the construction of molecular surfaces [Interactive Visualization of Molecular Surface Dynamics] We simply take into account consecutive atoms in the chain, determine their bounding sphere, and then proceed with the next series of atoms. For the third level we reuse the results obtained for the second level, and proceed similarly for the spheres located in individual protein chains. For the last level we wanted to dramatically reduce the number of spheres, and therefore the previous approach, only based on local proximity only, does not perform well enough for this task. We employ k-mean clustering instead, in the same way as parulek et al. The major issue with clustering algorithms is the high complexity and slow computing speed when dealing with several thousands of spheres. To speed up the clustering we 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 for this level the clustering performs fairly fast. It is worth mentioning that although the LOD levels creation requires pre-computation it is rather light to compute, and could eventually be applied in real time of each frame when dealing with dynamic data with only little overhead.

7. Efficient Rendering of Bi-lipid membrane

Bi-lipid membranes are an inherent part of large molecular biology structure. However they exhibit different properties than proteins must be handled accordingly. Firstly membrane are usually very large and do not exhibit repetitive structures such as a soup of protein for instance. We obtain the static structure of a membrane using the lipid wrapper tool. To compute the membrane the tool applies an small bi-lipid membrane patch over the vertices of an input mesh. Unfortunately it is not possible to use this single patch for instanced rendering, since during the process additional lipid molecules are added in order to fill the void that are present between adjacent vertices. Therefore we most 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 for protein rendering. The main different here being 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 lipid molecule, but given the small size of lipid molecules, between 20 and 50 atoms only, the resulting number of group would be too large for efficient coarse level culling. Although it is not possible to establish a relationship between the lipid molecules present in a triangle and the input patch we may easily group them by vertices since the computing is done per triangle. Thus we are able to determine by comparing two consequent lipid molecules if they belong to the same triangle or not, which facilitate greatly the grouping of the atoms. In comparison the use a clustering method would be much longer demanding and could take up to days to compute while the presented method is very light to compute. For the tested membrane we were able to group atom by group of 2000 atoms average which. 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.

8. Efficient Rendering of DNA/RNA strands

Animating proteins is fairly straightforward, one simply need 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 strands 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 recomputation. While this approach is viable of low to mid sized DNA strands it is very likely to perform poorly for DNA paths with over a million control points. Although the study of DNA structures have reveal many different types of foldings requiring complex algorithm, the most commonly recognizable shape is the B-DNA, which showcases quite a regular structure, namely 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 to 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 and to procedurally generate a double helix structure based of atomic data of nucleic acids. Thus, the GPU transfer time is dramatically reduced, and dynamic level-of-details can be applied too, allowing the rendering of a very large quantity of DNA. However, generating DNA strand procedurally of the GPU introduce a few constraints that are proper to parallel computing. We limitate the number of bases to a fixed and constant number for each segment of the DNA path in order to avoid having to propagate information in serial along the strand. Therefore we must also ensure that the spacing between each control point of the DNA path is constant. We employ cubic splines for smooth interpolation between the control points. We perform uniform sampling along the curve segments to ensure that all the bases of a single segment have the similar spacing. A well know challenge when dealing with 3D splines is to determine curve frames (normal, binormal, and tangent vector) which are constant along the whole strand and do not exhibit any twists. Any twists or abrupt variation of frame orientation would very likely cause artefacts and irregularities in the DNA structure.

Many algorithm have been developed to compute rotation minimizing, but we came up with a simplified version instead which gave us satisfying results. In order to compute a smooth normal vector along the curve, we first determine the normal at the control points. Then for browse each control points in series and rotate the normal around its frame in order to get a minimum variation in orientation according to the normal of the previous control point. The normals with minimal rotations are then uploaded to the GPU along with the control points, and during the instanciation of the nucleic acids, we obtain the normal vector of a nucleic acid by linear interpolation of the two segment normal vectors. The pseudo code for the minimized rotation normal vector determination is given in appendix X.

The DNA rendering pipeline is designed as follows:

Compute control points positions and normals on the CPU and upload info to the GPU. The vertex shader read the control points for each segment of the DNA path (p0, p1, p2, p3). The vertex shader determines the position of each base using uniform sampling along the curves. The position of the bases is then passed on to the tessellation shader which injects point based primitives corresponding to the bases. Geometry shader and fragment shader turns the point primitives into sphere impostors.

During the tessellation we read the atom data of the nucleic acid, we offset the positions and we apply the proper rotation according the to segment normal which is precomputed and rotation caused by the structure of B-DNA, we define the orientation of each pair of bases with the following formula: normal * quat(tan, i * 34.3); In order to connect two segments of the path coherently we ensure all the bases of a segment to perform a 360 degree rotation. As a result the last base of a segment matches elegantly with the first base of the next segment, and this without having to communicate any information between consecutive segments.

9. 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

10. Results and Expert Feedback

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

Mycoplasma DNA

11. Discussion

12. Conclusions and Future Work