Journal Name RSCPublishing

ARTICLE

Exploring Genomes with a Game Engine

Cite this: DOI: 10.1039/x0xx00000x

Jeremiah J. Shepherd[a], Lingxi Zhou[a], William Arndt[b], Yan Zhang[a], W. Jim Zheng[c,*] and Jijun Tang[a,d,*]

Received 00th January 2012,
Accepted 00th January 2012

DOI: 10.1039/x0xx00000x

www.rsc.org/

More and more evidence indicates that the 3D conformation of eukaryotic genome is a critical part of genome function. However, due to the lack of accurate and reliable 3D genome structural data, this information is largely ignored and most of these studies have to use information systems that view the DNA in a linear structure. Visualizing genomes in real time 3D can give researchers more insight, but this is fraught with hardware limitations since each element contains vast amounts of information that cannot be processed on the fly. Using a game engine and sophisticated video game visualization techniques enables us to construct a multi-platform real-time 3D genome viewer. The game engine based viewer achieves much better rendering speed and can handle much larger amount of data compared to our previous implementation using OpenGL. Combining this viewer with 3D genome models from experimental data could provide unprecedented opportunities to gain insight into the conformation-function relationships of a genome.

Introduction

A significant fraction of important questions in biology research today require analysis of genomic data sources. Regulatory functions of the cell operate in three dimensions such that currently available software, which displays genetic data as linear sequences, offers insufficient insight into some problems. New experimental techniques are revealing information that impacts the 3D structure of genetic molecules such as nucleosome position distribution [1], histone methylation [2], and transcription factory complexes [3]. As displayed by current software, the 3D context of these information sources is not shown, which potentially obscures important spatial relationships. To address this problem, we have developed a system to visualize genomes in a way that incorporates sources of 3D information while preserving responsive viewing and interaction.

A primary benefit to software that can display genomic data in three dimensions is it facilitates advances in the creation of 3D models. The chromosome conformation capture family of experiments generates data directly relevant to the 3D structure of chromatin. This data can be used to create and display small models of a few hundred thousand base pairs arranged in 3D space such as for the interesting alpha-globin gene domain on human chromosome 16 [4], all the way up to possible full human genome models using information derived from Hi-C experiments [5]. Even though data generated from current techniques are sparse and not yet sufficient to create accurate high-resolution 3D models of eukaryotic genome, it is expected that more believable and detailed models may be derived in the future. A software platform that can visualize such high-resolution model would prepare us for the future needs of studying 3D genome model when available.

A system that successfully displays genetic molecules in three dimensions must have the following abilities: it must display full size mammalian chromosomes on the order of billions of base pairs, display the details of important segments at the scale of individual atoms, and allow real time manipulation of the view at focus of detail. Our software behavior mimics that of the popular Google Earth service. When using Google Earth, a user can zoom out to view the entire planet as a whole but with little detail, or seamlessly zoom in to show a progressively smaller section of the surface but with greater detail. In our case the broadest view would display all chromosomes contained in a nucleus while zooming in would reveal the progression of more detailed features such as chromatin fiber arrangement, histone positions, base pair locations, and individual atoms. Creating such a system is not a trivial task. The volume of data needed to specify an arbitrary mammalian genome in 3D space well exceeds the limit of available memory necessitating careful loading and pruning. Google Earth only has one model to display and thus can take advantage of advanced indexing and caching optimizations, while in contrast, our software must display a wide variety of possible genomes and structures; this removes our ability to construct data structures in advance.

Most modern video games must visualize vast amounts of information at once while being interactive in real time. Most video game developers save resources by using game engines, which are of the shelf systems that implement common display and interaction features. These same engines have been shown to be useful for other applications such as simulation research [6]. Biology is no stranger to using game concepts to assist in new discoveries. Games like Fold It have been developed to help solve difficult questions about protein folding [7]. Our software is the first attempt to use game engine technology for the purpose of displaying chromatin in three dimensions. We show that these techniques developed for the video game industry can be successfully used to make useful and responsive displays of genomes in 3D.

Researchers use a variety of tools to place their experimental data in the context of reference genomes, but among these tools the ability to view spatial relationships in three dimensions is missing. Genome browsing tools have been developed by variety of institutions, such as UCSC, Ensemble, and NCBI. These genome browsers not only integrate vast amount of genome information from varied sources, but also allows users to develop custom tracks which easily integrate their own data with existing genome information [8]. Such feature-rich genome browsers have become very popular and played essential roles in several important, large-scale genome projects, including Encyclopedia of DNA Elements (ENCODE) [9] and 1000 Genome Project [10]. Existing viewers display genomes and annotations along a single dimension, the sequence coordinates, making them unsuitable for displaying some kinds of epigenetic and structural information generated by recent technologies such as chromosome conformation capture style experiments.

Genome3D was the first model-view framework developed to work with current genome browsers to address these challenges, and to facilitate multi-scale integration and visualization of large genomic and epigenomic datasets in three dimensions [11]. This model-view framework enabled researchers to infer new knowledge about structure/function of genomes that would have been difficult to accomplish by primary sequence-based browsers. For example, phosphate groups of different base pairs in DNA strand can be either exposed to the outside or confined between the histone proteins and DNA backbone. This information has important functional implications, as exposed phosphate groups can be easily accessible by DNA binding proteins [12]. While new sequencing technologies allow researchers to map individual nucleosomes across the whole genome [13], it is a significant challenge for current genome browsers to display this epigenetic information in an intuitive manner. Visualizing such information using a 3D genome model can facilitate new inferences about potential regulatory behavior such as functional versus non-functional SNP's in non-coding regions.

Unfortunately, in the years since its release a number of limitations with the Genome3D software package were revealed, warranting the development of a new version. Genome3D had no internal throttle to manage the loading or view of very large data sets. A careless user could easily load more chromosomes than are able be displayed in real time which would slow responsiveness to unusable levels. The former software was developed in C++ using libraries that have since become obsolete. These dependencies severely restricted the portability and maintainability of the code base.

Using game engine technologies and some sophisticated visualization techniques we intend to solve the two critical flaws of Genome3D. Game engines are designed to aid in the development of games by providing interaction and complex rendering techniques as standard features. This frees attention for the development of logic and content. Also, because game development is are a large consumer industry whose products need to be highly marketable, most modern engines have multi-platform support built in, which allows the programs developed with one to easily be ported to a number of other devices. Furthermore, game engines have previously been used for large simulation research where they are proven to work well in several different disciplines [14]. For these reasons, we found it most appropriate to use a game engine to display genomes in three dimensions.

The System

The greatest dilemma facing this system was how to both manage the large amount of information while still maintaining real time responsiveness. Since modern games engines provide open world management as a standard feature, this was chosen as a good starting point. We have used the Unity game engine, which has become highly popular and widely used by independent game developers and researchers. While the interaction features are standard to most game engines, our need to visualize data on the order of billions of base pairs is unique enough to still be challenging. The largest portion of our attention has been directed towards efficiently rendering the scene. One of aspect of our approach is the level of detail (LOD) system, which is traditionally used to cut down the amount of processing power needed to generate the elements in the scene. Different parts of the scene are presented at different LODs based on criteria such as the distance between the camera and that object. An object in the far distance can be rendered with a lower LOD in a way that conserves computational resources but is indistinguishable to the viewer. An object very close to the observer will be displayed with a higher LOD and use more resources, but geometry limits the number of objects which can occupy this space and any further objects that have been occluded need not be drawn at all. To best solve the problem of determining placement and distances of the genome we used a data structure commonly used in games, and then further processed multiple data sources in different LODs.

The system runs on multiple platforms with consistent interface as shown in Figure 1:

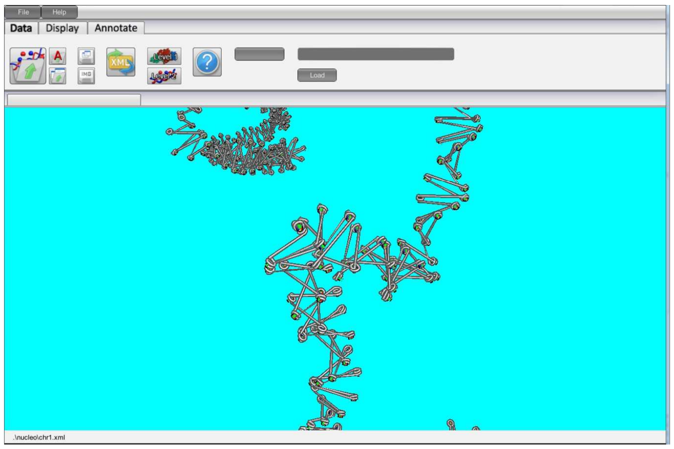

Fig. 1. System interface with a chromosome loaded at the histone level.

The Octree

Many modern games have vast open worlds, which need to be carefully processed in order to keep the action fluid. To reduce the amount of data processed at a given time, most of these applications use a data structure called an octree. This structure subdivides 3D space into eight sections that are represented by nodes in the tree as seen in Figure 2. These subsections can be recursively subdivided making more nodes in the tree with a finer level of detail. When a node is very far away from the camera it is displayed directly with a lower LOD and any children it has are ignored. When the camera is moved closer to a node, then it is examined more carefully such that each of and each of its children are fully processed and displayed.

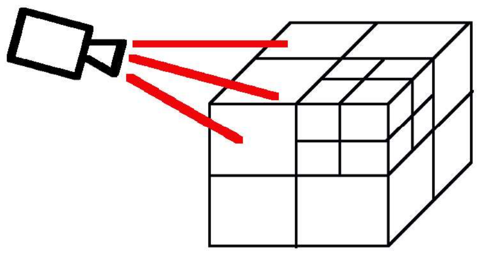
Fig. 2. Octree example. This shows how it will calculate the ranges from each cube to the camera.

Most commercial implementations use static octrees with a set number of subdivisions that is optimized for a single given model. However, our system is distinct because our 3D models must be generated on the fly, and can only be partially built at any one time given the large amounts of data that needs to be processed and stored. This has led us to choose a dynamic octree implementation [15]. The dynamic variation of the data structure automatically subdivides into eight parts when the data contained in a node exceeds a given tolerance. In this way, we avoided having an unnecessarily big tree(s) in memory. Also since we are dealing with a large object it makes sense to stay away from loose octrees [16].

Given the difference between the input, formatted as a spline representing the chromosome fiber, and the internal storage as an octree, faithfully preserving the structure of the data was a key priority. In traditional implementations the mesh for the objects in the scene are provided, and each polygon in the mesh is placed in a node in the tree corresponding to its spatial location. However, in our case, the mesh is generated given a series of files that contain an ordered set of control points on the curve. Instead of dividing the problem by polygons we do so using the spline control points. Wherever the curve intersects an octree node boundary, a new control point is placed at the point of intersection and inserted into both octree nodes. While this increases memory usage slightly, it preserves information content perfectly.

Further Levels of Detail

In addition to the octree, a level of detail system has been implemented to manage display complexity given the context of the camera and model. Information is grouped into three different levels: fiber, nucleosome, and atomic. The fiber level represents the lowest level of detail where each chromosome's 30nm fiber is displayed as seen in Figure 3 and Figure 4. At this level the data is formatted as a series of files, each in XML format specifying an individual chromosome and the control points of the spline which positions it. The system scans the directory provided by the user, and then loads and displays each chromosome. Files are first loaded into memory and then placed into an octree. As the data proceeds, additional octrees are constructed for each chromosome to guarantee they remain segregated from one another.

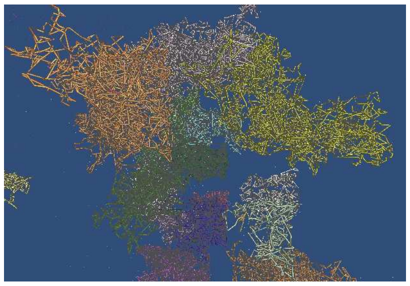
Fig. 3. Fiber Level Far with 22 Chromosomes Loaded

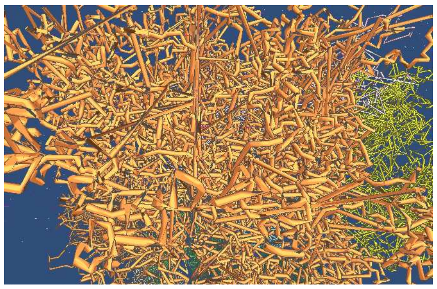
Fig. 4. Fiber level with close up view of chromosome 1.

Proceeding deeper, the next level of detail is the nucleosome stage. At this level, nucleosome base-pair positions plus the DNA that wraps around and links to them are displayed as seen in Figure 5. Instead of loading all the chromosomes like the previous level, this view only displays one at a time. The files for this level of detail are significantly different from the previous section as they are much more detailed. Once again, XML file formats are used to hold the structure information for each chromosome. These files not only include the base pair number and positional information but also the orientation of each histone. Linker DNA can either be automatically generated for display or specified in the corresponding data files. An additional feature of this view, because most researchers are not interested in specific interesting regions and not an entire chromosome, we implemented an option to reduce the viewing size to a user specified range as seen in Figure 6.

Finally, the highest level of detail is at the atomic scale. Similar to the nucleosome stage, the user chooses a range from a start base pair to an ending base pair, and the system displays that atomic structure as seen in Figure 7. This level faces an additional challenge, as two sets of information must be merged into one display. Information provided at the nucleosome level determines positional information, which is then combined with a fasta file containing the DNA sequence. The user specified position is matched to both the fasta file and the nucleosome positions, which are read in tandem and used to place the appropriate base pair atoms at their correct positions.

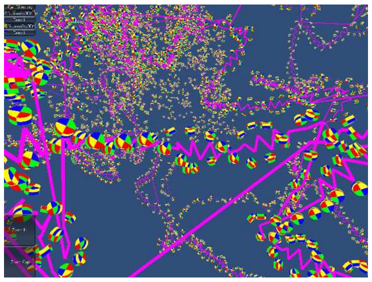

Fig. 5. Histone level of chromosome 1 in mid range. This shows 100,000 histones rendered in real time with procedurally generated linker data which is highlighted in pink.

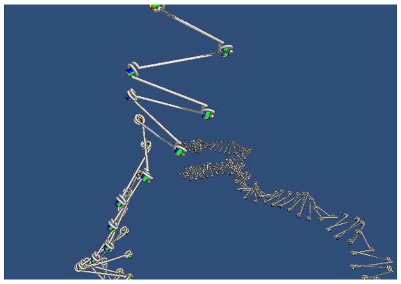

Fig 6. Histone level of chromosome 1 at a reduced scale close up (linkers are not highlighted).

Other Benefits

In addition to the robust interactivity and the superior display management provided by a game engine, there are other benefits to using it for this application. A game with a larger marketplace is more likely to succeed, making portability to multiple platforms a desirable quality. The system we created has been deployed to Windows, Mac, and some mobile devices, including tablets as seen in Figure 8. Each version required some minor changes but overall the system still works the same with minimal added development time. For example, in order to function on mobile devices we must limit the rendering capabilities and read the files from an external server instead of from the device itself. The simplicity of multi-platform support will greatly increasing our potential audience.

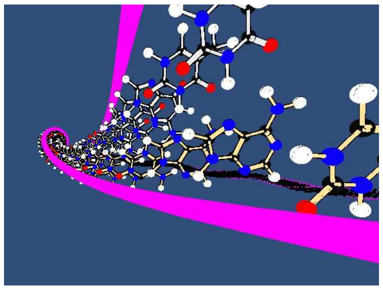

Fig. 7. Atomic level of chromosome 1 from base pair 10400 to 10900

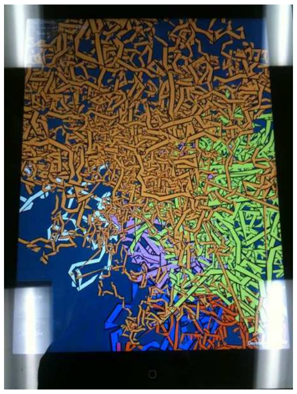

Fig. 8. iPad Example. This is an example demonstrating the octree based LOD working on an iPad.

Results

In order to test the performance and functionality of this application, we compared it to the previous version of Genome3D, which is the only other implementation of this concept [11]. Each test ran on a standard desktop with 4 GB of RAM, an Intel c Xeon c CPU at 3.07 GHz, and running 64-bit Windows 7 Professional. Our tests and results follow.

The main criteria we chose to judge success was the rendering capacity, since that is the best approximation of its functional performance from the perspective of a user. To accomplish this, we loaded a set amount of data, and then progressively add large amounts of data for each individual test, while also keeping track of the running frames per second (FPS). Each of these smaller tests was also run for an isolated LOD: fiber, nucleosome, and atomic levels. The first set of tests was at the fiber level with results shown in Table 1. There is a hardware refresh limit that is 120 Hz in our system. Also game engines typically set an upper limit on FPS to avoid any side effects; this limit is 60 in Unity. In Unity we can set a targeted frame rate, which is the FPS that Unity will try to reach if the system is heavy loaded. As our genome viewer is not fast-paced, we choose 30 FPS as the target. The original genome viewer outperforms the new one with only one chromosome file loaded, but it cannot load additional chromosomes without crashing. Our new system has maximum FPS until 15 chromosomes are loaded; the FPS is still acceptable when all 23 chromosomes of human genomes are loaded.

TABLE 1. Improved performance of game engine based viewer at the chromatin fiber level (the original viewer cannot handle data with more than one chromosome)

Program	1 Chromosome	10 Chromosomes	15 Chromosome	23 Chromosomes
Original Genome Viewer FPS	120	0	0	0
Game Engine Genome Viewer's FPS Range	60	60	60	30

Next, a similar test was conducted at the nucleosome level. In order to gain accurate insight of performance, the test varied the number of histones loaded and rendered while measuring the sustained frames per second. Results appear in Table II. When only a few nucleosomes are loaded the original genome viewer performs faster, but once more data is added the older system becomes unresponsive. In contrast, the performance of the new system did not begin to lose performance noticeably until the number of histones loaded exceeded 10,000.

TABLE II. Improved performance of game engine based viewer at the nucleosome level (the original viewer cannot handle more than 10 nucleosomes)

Program	1 nucleosome	1,000 nucleosomes	5,000 nucleosomes	10,000 nucleosomes
Original Genome Viewer FPS Range	60-120	0	0	0
Game Engine Genome Viewer's FPS Range	60	60	30	30

Finally, the last set of tests was run in a similar vein as the others. Much like the nucleosome scale, the purpose was to test and evaluate each program's performance as an increasing amount of atoms were loaded and rendered, measuring the frames per second. The results, as seen in Table III, show a more comparable trend. Both systems simply rendered the given data without any further LOD data analysis, and it first appears that the original genome viewer out performs the game engine version. However, the original system only seems to allow twenty-five atoms to be rendered at once, so it cannot make a comparison after the initial twenty five. On the contrary, our new system can easily handle 1,000 atoms at once, without noticeable drop of performance, achieves a 40-times increase of information loaded.

TABLE III. Improved performance of game engine based viewer at the atomic level (the original viewer cannot handle more than 25 atoms)

Program	25 Atoms	500 Atoms	1000 Atoms	1500 Atoms
Original Genome Viewer FPS Range	60-120	0	0	0
Game Engine Genome Viewer's FPS Range	60	60	60	30

From these tests, it is obvious to see that using a game engine with sophisticated rendering techniques commonly used in games greatly expands the capacities of the former Genome3D viewer. The key to the performance enhancement was breaking apart and grouping the large amount of data in a more sophisticated way, which was facilitated by using an octree for spatial information. Also, modern game engines typically offload most rendering tasks to GPUs, which are becoming computationally more powerful and can process multiple rendering requests at once. In this way, more data could be loaded, managed, and displayed at any given time yielding significantly improved utility.

Conclusions

In this paper we addressed the problem of visualizing genomes in three-dimensional space, and the potential benefits to such an approach. We solve this problem by applying techniques and tools that are commonly used in video games, which include using a commodity game engine, partitioning data with an octree, and grouping the files to perform multi-tier level of detail analysis (LOD). The game engine has fast rendering systems and also allows for simple and fast multi-platform support. Organizing the data with an octree reduced resource consumption allowing larger data sets to be viewed with visual fidelity and responsive performance. Finally, recognizing the data was given in three different formats provided another level of detail such that the system was able to segregate displayed content based on context. Finally, we showed that using this application compared to the older non-game application out performs in situations where the amount of data being viewed is non-trivial, and is able to load much more data before losing responsiveness. With our application we have provided researchers with a tool that could be used for valuable genomic work.

Acknowledgements

This work is supported by NSF award #1161586 to JJS, LZ, YZ and JT, as well as #1339470 to WJZ. .

Notes and references

[a] Department of Computer Science and Engineering, University of South Carolina, SC 29205, USA.

[b] Howard Hughes Medical Institute, Janelia Farm Research Campus, Ashburn, VA 20147, USA.

[c] School of Biomedical Informatics, U. Texas Health Science Centre at Houston, TX 77030, USA.

[d] Key Laboratory of Systems Bioengineering of the Ministry of Education, Tianjin University, Tianjin, 300072, PR China.

* Corresponding authors

† Footnotes should appear here.

1 R. Schopflin, V. B. Teif, O. Muller, C. Weinberg, K. Rippe, and G. Wedemann, "Modeling nucleosome position distributions from experimental nucleosome positioning maps," Bioinformatics, 2013.

2 A. Barski, S. Cuddapah, K. Cui, T.-Y. Roh, D. E. Schones, Z. Wang, G. Wei, I. Chepelev, and K. Zhao, "High-resolution profiling of histone methylations in the human genome," Cell, vol. 129, no. 4, pp. 823–837, 2007.

3 P. Unneberg and J.-M. Claverie, "Tentative mapping of transcription-induced interchromosomal interaction using chimeric EST and mRNA data," PLoS One, vol. 2, no. 2, p. e254, 2007.

4 D. Baù, A. Sanyal, B. R. Lajoie, E. Capriotti, M. Byron, J. B. Lawrence, J. Dekker, and M. A. Marti-Renom, "The three-dimensional folding of the α-globin gene domain reveals formation of chromatin globules," Nature Structure & Molecular Biology, vol. 18, no. 1, pp. 107-114, 2011.

5 E. Lieberman, N. L. van Berkum, L. Williams, M. Imakaev, T. Ragoczy, A. Telling, I. Amit, B. R. Lajoie, P. J. Sabo, M. O. Dorschner, R. Sandstrom, B. Bernstein, M. A. Bender, M. Groudine, A. Gnirke, J. Stamatoyannopoulos, L. A. Mirny, E. S. Lander, J. Dekker, "Comprehensive mapping of long-range interactions reveals folding principles of the human genome," Science, vol. 326, pp. 289-293, 2009.

6 J. Shepherd, R. Dougal, and J. Tang, "Visualization and simulation potential of Microsoft's XNA," Grand Challenge in Simulation and Modeling (GCSM), pp. 306–310, July 2010.

7 F. Khatib, F. Dimaio, S. Cooper, M. Kazmierczyk, M. Gilski, and et al, "Crystal structure of a monomeric retroviral protease solved by

protein folding game players," Nature Structural & Molecular Biology, vol. 18, no. 10, p. 1175, November 2011.
8 W. Kent, C. Sugnet, T. Furey, K. Roskin, and T. Pringle, "The human genome browser," Genome Res 12, vol. 12, pp. 996–1006, November 2002.
9 "The Encode (Encyclopedia of DNA Elements) project," Science, vol. 306, pp. 636–640.
10 N. Siva, "1000 genomes project," Nature Biotechnology, vol. 26, p. 256.
11 T. Asbury, M. Mitman, J. Tang, and W. Zheng, "Genome3D: a viewer- model framework for integrating and visualizing multi-scale epigenomic information within a three-dimensional genome," BMC Bioinformatics, vol. 11, p. 444.
12 K. Eisfeld, R. Candau, M. Truss, and M. Beato, "Binding of NF1 to the MMTV promoter in nucleosomes: influence of rotational phasing, translational positioning and histone H1," Nucleic Acids Res, vol. 25, pp. 3733–3742, 1997.
13 D. Schones, K. Cui, T. Roh, and A. Barski, "Dynamic regulation of nucleosome positioning in the human genome," Cell, vol. 132, pp. 887–898, 2008.
14 T. Zhang, J. Shepherd, J. Tang, and R. A. Dougal, "The simulation tool for mission-optimized system design," in Proceedings of the 2011, Grand Challenges on Modeling and Simulation Conference. Society for Modeling & Simulation International, 2011, pp. 348–355.
15 D. Ginsburg, "Octree construction," Game Programming Gems, pp. 439–443, August 2000.
16 T. Ulrich, "Loose octrees," Game Programming Gems, pp. 444–453, August