Real-Time Visualization of Large Time-Varying Molecules

Xuejun Hao **Amitabh Varshney** **Sergei Sukharev**
Department of Computer Science and UMIACS **Department of Biology**
University of Maryland at College Park
College Park, MD 20742
{hao, varshney}@cs.umd.edu **ss311@umail.umd.edu**

Keywords: Molecular dynamics visualization, triangle strips, visibility-based culling, multi-resolution

Abstract

Interactive display of molecular datasets is an important tool to better understand the structural and functional properties of biological samples. It remains a challenge to display large molecular datasets, especially time-varying ones, interactively. In this paper, we develop a time- and memory-efficient algorithm to solve this problem. Our approach speeds up the graphics rendering pipeline at several stages by developing and extending various rendering techniques for efficient display of time-varying molecular data, such as run-time triangle strip and triangle fan generation, visibility-based culling, view-dependent precision control, and memory-bandwidth reduction. More importantly, our algorithm requires no pre-processing and little memory overhead. It is linearly scalable in the sizes of the molecular datasets. Overall, we have achieved interactive display of time-varying molecular datasets with several thousands of atoms. Our algorithm is flexible and scalable and our ideas for this problem can also be applied to visualization of other large time-varying datasets. We illustrate the capability of the proposed algorithm by presenting a fast animation of the gating transition of the bacterial mechanosensitive channel MscL.

1 INTRODUCTION

Interactive visualization of large datasets has become one of the major challenges for computer graphics and scientific visualization researchers. Several acceleration algorithms for faster rendering have been developed over the years. They include techniques such as multi-resolution rendering and visibility-based culling that work by reducing the number of graphics primitives to be rendered based on viewing and illumination parameters. Another class of techniques works by improving the memory bandwidth efficiency by better organization of the data (for example triangle strips and triangle fans) or reducing the precision of each graphics primitive (for example variable-precision rendering). These two kinds of techniques complement each other and have been applied successfully to render static scenes.

The above techniques work by a pre-analysis of data with the design of clever data structures for efficient run-time access. They have achieved impressive results on static data. However, it is non-trivial to adapt the above techniques to time-varying datasets and there is little prior art for accelerating the rendering of time-varying datasets. In this paper, we address the problem of interactive rendering and visualization of large time-varying molecular datasets. We solve this driving application by developing new techniques and extending existing ones to deal with time-varying molecular datasets.

Globular proteins are well packed and adopt ordered three-dimensional structures. More importantly, they possess a variety of motions such as bond vibrations, side-chain rotations, segmental motions, and domain movements. It is motion that is required for proteins to work, and it is our inability to fully understand protein dynamics and their role in protein function that restricts our understanding of the mechanisms of protein folding, recognition, allostery, and catalysis. Protein dynamics are extremely complex and difficult to analyze, because a variety of motions take place in the same molecule and at the same time. Being able to simulate and visualize protein motions on a computer is therefore of utmost importance for the understanding of the very complex picture of protein dynamics and for the development of proper theoretical models for the analysis. In this paper we discuss interactive rendering of large molecular dynamics simulations to allow a better and more informed understanding of the relationships between their structure and function.

The main contributions of this paper are:

1. We show how to extend current acceleration techniques, such as multi-resolution rendering and precision control for rendering static scenes to display time-varying molecules.

2. We develop a novel occlusion-culling scheme for rendering time-varying molecules. At each frame we

build a new conservative occlusion map to cull the invisible atoms.

3. We build front-facing triangles into triangle fans and strips at run-time for efficient use of memory bandwidth. We show that our algorithm is linearly scalable.

2 RELATED WORK

Many techniques have been developed for speeding up the visualization of large datasets, especially static scenes. In this section, we give an overview of the relevant previous work, such as level-of-detail hierarchies, triangle-strip generation, occlusion-based culling, and variable-precision rendering.

Multi-resolution hierarchies for level-of-detail-based rendering have traditionally involved modelling each object at multiple levels of detail. The detail is usually measured in the number of geometric primitives required for representation. A high-detail triangle-mesh object will require a higher number of vertices, edges, and triangles. At run-time, an appropriate level of detail is selected based on viewing parameters for a faithful representation. Even better, level of detail can be applied in a view-dependent manner to take advantage of temporal coherence and adaptively refine or simplify the geometry between adjacent frames [Luebke et al. 2003]. Normally the multi-resolution hierarchies of the geometry are built as a pre-process.

Triangle strips provide a compact representation of triangular meshes and are supported by popular graphics APIs such as OpenGL. The use of triangle strips results in fast rendering and transmission. A triangle strip with n triangles can be rendered with only $n + 2$ instead of $3n$ vertices. Thus substantial savings for memory bandwidth and computation of per-vertex operations such as transformations, lighting, and clipping can be achieved. Triangle strips can be generated as a pre-process and stored with the geometry for later usage [Evans et al. 1996], or can be pre-generated and later updated view-dependently [El-Sana et al. 1999]. It can be costly to generate triangle strips from scratch at run-time [El-Sana et al. 1999]. Triangle strips can also be used for polygonal mesh compression [Gumhold and Straßer 1998; Taubin et al. 1998].

Occlusion culling works by culling away portions of the scene that are not visible from the viewer. Culling can be done in object space through the use of spatial partitions or bounding volume hierarchies [Coorg and Teller 1997; Hudson et al. 1997]. Object-space algorithms have been developed for several specialized environments such as architectural or urban datasets [Airey et al. 1990; Hudson et al. 1997; Wonka et al. 2001]. However such techniques are not very effective on scenes with several small occluders. Culling can also be done in image space using hierarchical Z-buffer [Greene and Kass 1993] or hierarchical occlusion maps [Zhang et al. 1997]. Image-space occlusion culling usually achieves better occluder fusion. Normally occlusion culling is done conservatively [Klosowski and Silva 2001; Yoon et al. 2003]. Non-conservative culling [Klosowski and Silva 2000] can lead to popping artifacts when objects appear and disappear between adjacent frames.

Variable-precision approach [Hao and Varshney 2001] complements the multiresolution techniques as it reduces the precision, instead of the number, of the primitives to be rendered. This method relates the minimum number of bits of accuracy required in the input data to achieve a desired accuracy in the display output. The reduced precision can be used to accelerate the graphics rendering pipeline at various stages. One can speedup the geometry transformations and lighting using SIMD parallelism on modern processors. One can also reduce the memory bandwidth bottleneck between central and graphics processors by sending less precise values with fewer bits. Best of all, all this acceleration comes without sacrificing image quality.

Most of the above techniques rely on a certain level of pre-processing of the scenes and build the appropriate data structures before the rendering phase. Hence they are well-suited for scenes with static geometry. For time-varying scenes, especially molecular dynamics simulations with significant changes, these techniques can not be readily applied. In this paper, we develop techniques for interactive display of large time-varying molecules. Our algorithm requires no pre-processing of the data, has nearly no memory overhead, and is linearly scalable.

3 OUR APPROACH

The space-filling display of time-varying molecules involves rendering each atom of the molecule as a sphere with a van der Waals radius for every time frame. Different atoms are represented by different-sized spheres, with possibly different resolution tessellations. Since viewing individual atoms is not a normal real-life experience, parallel projections are often considered more informative in molecular graphics than foreshortened perspective projection. Here we assume parallel projection.

Figure 1 shows the pipeline of our algorithm for displaying each frame of a time-varying molecular dynamics data. We start by loading the list of atoms with their 3D positions for current time frame, and we sort them according to their distance from the viewer using a quick-sort algorithm. Next we determine the visibility of each atom, by using our visibility-based culling algorithm (detailed in Section 3.1). We use multi-resolution techniques to decide the appropriate number of triangles with which to represent the spherical atoms. We also decide the necessary precision for vertex data from display resolution specification. For the triangles that survive the back-face culling phase we generate triangle strips and compute illuminated color. Finally, we send the tri-

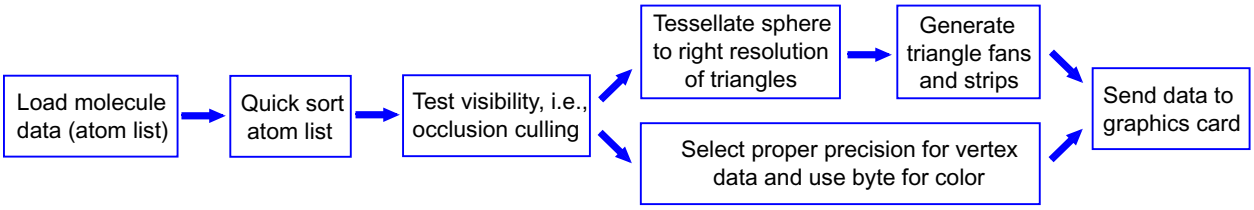


Figure 1: Pipeline of our run-time algorithm

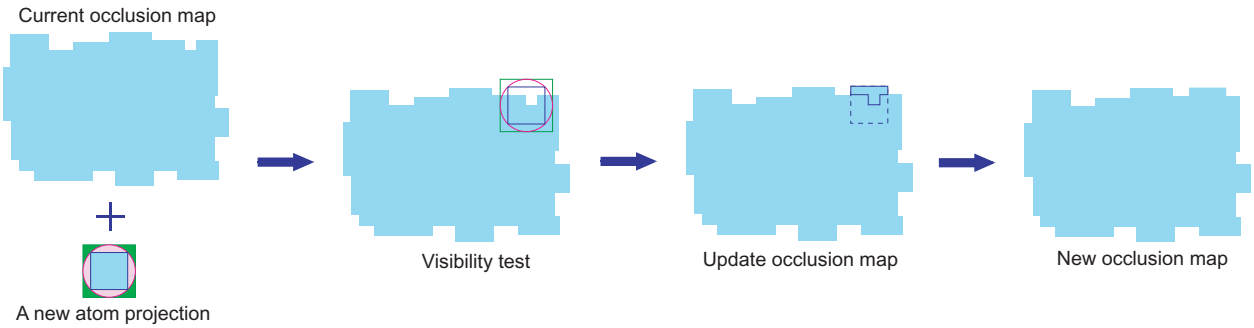


Figure 2: Visibility test of an atom

angle strips and triangle fans with appropriate precision to the graphics card for rasterization and display.

3.1 Determination of the Visible Set of Atoms

Previous approaches for occlusion culling deal with general environments. They achieve efficiency of occlusion test by pre-processing the scene and building a good data structure such as a cluster hierarchy [Yoon et al. 2003] to store the relationships between objects. Run-time efficiency is achieved through temporal coherence, since the viewing parameters seldom change significantly from one frame to the next. Occlusion culling for time-varying molecules is different from the previous situations. First, molecules go through large structure changes, so the occlusion information between adjacent frames may change significantly. This makes use of temporal coherence by pre-processing the scene difficult and less efficient. Second, there are no large occluders in time-varying molecules. Each molecule consists of thousands of atoms whose sizes are of the same order. The relationships among this large number of small potential occluders vary significantly over time. Thus, instead of trying to use pre-processing with temporal coherence, we decided to build per-frame occlusion map on-the-fly to achieve better efficiency. We use the culling algorithm described below to build per-frame occlusion map and estimate the visible set of atoms.

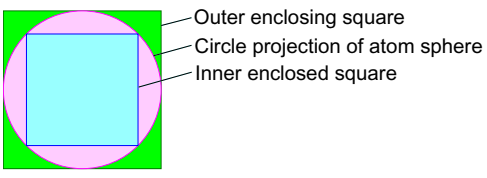


Figure 3: Over- and under-estimation for Occlusion Culling

The list of atoms for each frame is sorted using the quick sort algorithm. We use a conservative culling scheme to determine the potentially visible atoms. Since our algorithm is conservative, it is possible for a few non-visible atoms to be sent to the graphics card for rendering. Figure 3 shows the conservative nature of our culling. Each spherical atom is projected to a circle on the image plane under parallel projection. We project the atoms in a front-to-back order. If all the screen-space pixels of an atom are already occupied by the nearer atoms, then the current atom will not be visible. Since the projection and checking for overlap of circles is hard to implement efficiently, we instead use two different-sized nested squares. As shown in Figure 3, the blue square covered by the projected circle is used to represent the definite occlusion by this atom for the atoms behind it. We use the inner square of each atom to build the occlusion map and the outer green square (in Figure 3) to check if the atom has been blocked by previously rendered atoms. An example is shown in Figure 2. Here the atom is visible since its outer square has not been totally blocked. The occlusion map is then updated using the atom's inner square.

For memory and time efficiency, we use a bit pattern to store the occlusion map and check for atom visibility. Initially, each bit is set to zero to indicate non-occupancy. The bit is set to one when the pixel gets

covered by the inner square of an atom's projection for the first time. The pixel will from then on act as an occluder for the atoms that project on it later. The bit pattern storage using integers improves the memory usage. For a 1024×1024 image, we only need $128K$ bytes for storing the occlusion map. More importantly, storing the bit pattern as packed integers improves efficiency. As an example, if a new atom is visible and we want to adjust the occlusion map according to its inner-projected square, then we just set the occlusion map pixels covered by the square to one. So we can use a bitwise-OR operation of the values of the covered pixels with a all-one pattern to simultaneously cover several pixels in a single operation. Similarly if we want to check for visibility of a new atom, we need only to use a bitwise exclusive OR operation of the values of the covered pixels with an all-one pattern and check if the result is zero.

3.2 Generation of Appropriate Triangle Tesselations of Spheres

Recent multi-resolution literature [Luebke et al. 2003] has discussed the interactivity and visual realism tradeoffs in selecting an appropriate resolution for geometry. The screen-space size of an atom is an important determinant for picking the number of triangles for representing a sphere. We pick the tesselation resolution such that each triangle will have about ten pixels on the image plane.

For flexibility in adjusting the tesselation resolution, we generate points on spheres along circles with same latitude (i.e. same angle to the z–axis) and symmetric over the x–y plane as shown in Figure 4. We then connect the points to form triangles. Points connected to the North or South pole will form a complete triangle fan. One such triangle fan $(0, 1, 2, 3, 4, 5, 6, 7)$ is shown in 5(a), where the i^{th} triangle is described by the 0^{th}, i^{th}, and $(i+1)^{st}$ vertices in the sequence. The points between adjacent circles form a complete triangle strip. One such strip $(0, 1, 2, 3, 4, \mathbf{3}, 5, 6, 7, \mathbf{6}, 8, 9, 10, 11, 12, \mathbf{11}, 13, 14, 15, \mathbf{14}, 16, 17, 18, \mathbf{17}, 19, 1, 0)$ is shown in 5(b). Note that this is generalized triangle strip [Evans et al. 1996] where the repeated vertices are shown in bold.

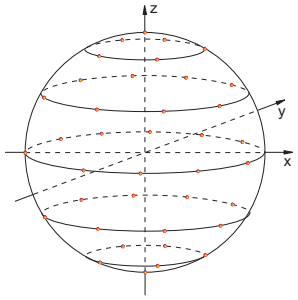
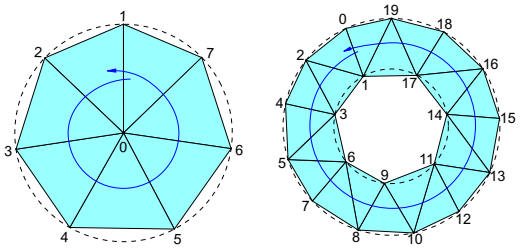


Figure 4: Generating points on a sphere

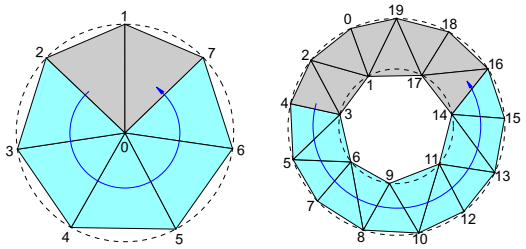


(a) Triangle fan (b) Triangle strip

Figure 5: A complete triangle fan and triangle strip as seen from above the pole of a sphere)

3.3 Run-time Triangle Strip and Triangle Fan Generation

After we decide the appropriate tesselation of an atom, we know from Section 2 that sending triangle strips or fans to the graphics card is more efficient than sending separate triangles. We can generate the triangle strips and fans easily from our tesselation scheme in Section 3.2 using traditional methods. However, that will not always be the best solution. A pre-generated triangle strip is fixed and will include both visible and non-visible triangles for each viewing direction. Even if an atom is visible, its back-facing triangles will not be visible. Pre-generated triangle strips can be updated at run-time for complex geometry [El-Sana et al. 1999]. However, here we observe that we can take advantage of the spherical atoms to generate the proper triangle strips and fans for their front-facing triangles. Thus we can benefit from the efficiency of triangle strips without the need to send back-facing triangles or to update the triangle strips for every frame.



(a) Front-facing fan (b) Front-facing strip

Figure 6: Triangle fan and triangle strip of front-facing triangles (in blue) as seen from above the pole of a sphere)

We first decide which triangles are front-facing. The front-facing triangle is defined as one with at least one front-facing vertex. The front-facing vertex can be easily determined by a dot product of its normal with viewing direction. Then we generate the triangle fan for

triangles consisting of front-facing vertices connecting to the pole of the sphere, and triangle strips for triangles consisting of front-facing vertices between adjacent circles on the sphere. As an example, Figure 6(a) shows a run-time triangle fan $(0, 2, 3, 4, 5, 6, 7)$ of Figure 5(a), while Figure 6(b) shows a run-time triangle strip $(4, 3, 5, 6, 7, \mathbf{6}, 8, 9, 10, 11, 12, \mathbf{11}, 13, 14, 15, 16)$ of Figure 5(b). The grey triangles in the figure are back-facing triangles.

An alternative to run-time generation of triangle strips and triangle fans is to generate a fixed triangle strips and triangle fan for the visible hemisphere, and rotate spheres at run-time to keep their orientation relative to the viewer fixed.

3.4 Memory Bandwidth Reduction

To further improve the memory and run-time efficiency, we adapt the variable-precision approach [Hao and Varshney 2001] to reduce the precision of the vertex data. As shown in [Hao and Varshney 2001], we need no more than 16 bits of accuracy to represent vertex data for pixel-level positional accuracy in up to $2^{13} \times 2^{13}$ rendering window under parallel and $2^{11} \times 2^{11}$ window under perspective projection. In this work we use 16 bits instead of 32 bits (floating point representation) to reduce the memory bandwidth by half.

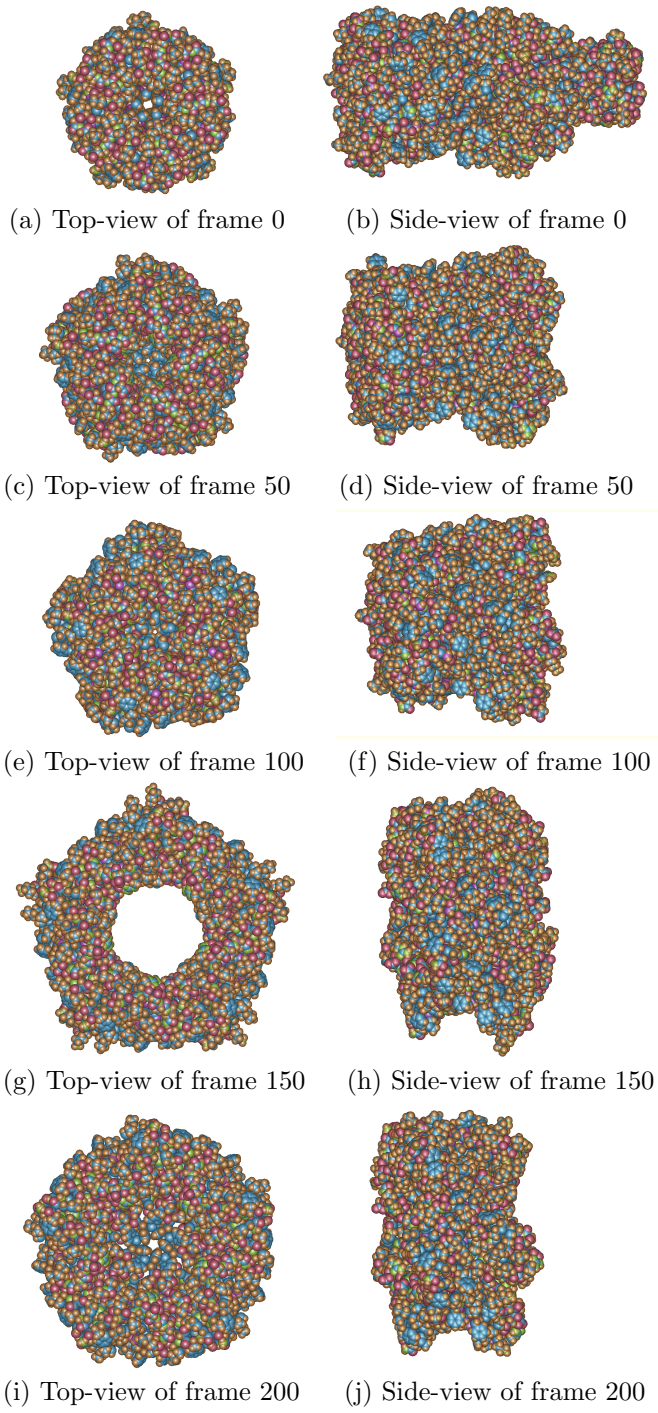
We also save some bandwidth by computing the color of vertices on the CPU and sending only the unsigned byte color (three bytes total for red, green, and blue) to the graphics card, instead of sending a vector vertex normal.

We have also used display list mechanism provided by OpenGL to get better memory coherence to display atoms. At each frame, we generate a new display list for each *type* of atom (carbon, oxygen, nitrogen, hydrogen, etc). Every visible atom is just a differently translated instantiation of the display list containing the triangles for its canonical representation.

4 RESULTS AND DISCUSSION

We have attempted to apply the approach described above to ion channel studies, a dynamic branch of biophysics with multiple applications in neurobiology, pharmaceutical research, and many other branches of biomedical science. Fast and efficient 3D visualization and animation of molecular models helps to present the dynamic nature of structures and facilitates the cross-disciplinary exchange with information. To illustrate the power of the fast-rendering algorithm we present a two hundred-frame animation of the structure of the large-conductance mechanosensitive channel MscL as it transitions from the closed to the open state. The animation was based on five models representing the closed, open and three intermediate conformations. The smooth transition between these modeled states was implemented using a linear interpolation algorithm.

MscL is a ubiquitous part of the osmoregulation system residing in the cytoplasmic membrane of most bacteria, both free-living and pathogenic. *Escherichia*



(a) Top-view of frame 0 (b) Side-view of frame 0

(c) Top-view of frame 50 (d) Side-view of frame 50

(e) Top-view of frame 100 (f) Side-view of frame 100

(g) Top-view of frame 150 (h) Side-view of frame 150

(i) Top-view of frame 200 (j) Side-view of frame 200

Figure 7: Top and side views of five stages of Escherichia coli mechanosensitive channel opening and closing

coli MscL (EcoMscL) is the best understood model mechanosensitive channel gated directly by membrane tension. The atomic-scale model of EcoMscL based on the crystal structure of its homolog from *Mycobacterium tuberculosis* was built in order to relate the structural information to the wealth of experimental data available specifically for the *E. coli* channel. In the closed state the channel core is represented as a tightly packed bundle of ten transmembrane alpha helices (Fig. 7a, b). The opening transition driven by external tension was modeled as an iris-like expansion of the transmembrane bundle accompanied by tilting of alpha-helices. The open conformation is characterized by a large (3 nm) pore capable to pass a large current or a flux of small omolytes (Fig. 7g, h). The last row of images (Fig. 7i, j) represent the intermediate semi-closed conformation.

The pentameric EcoMscL complex consists of 10585 atoms. We display the ion channel transition process as a two hundred frame animation. We have used a 2GHz Pentium 4 PC running Windows 2000 with a nVIDIA GeForce3 graphics card. We have achieved more than 32 frames per second (fps) rendering speed on this time-varying dataset, each frame of which consists of 1.3 million triangles. Our approaches are about four times faster than VMD (version 1.8.2) [Humphrey et al. 1996] with the same image quality.

5 CONCLUSIONS

In this paper we show that large time-varying molecular datasets can be displayed interactively using our time and memory efficient algorithm. Various techniques have been developed or extended to accelerate the graphics rendering pipeline. Our algorithm has several properties which make it very attractive. It has no memory overhead, requires no pre-processing, and is linear scalable. Though the techniques are designed especially for time-varying molecular datasets display, the concepts are general enough to benefit interactive display of large time-varying datasets in other application domain as well.

6 ACKNOWLEDGEMENTS

The authors acknowledge the critical contribution from Dr. H.R. Guy (NIH, Bethesda) in building models of MscL in different conformations [Sukharev et al. 2001]. We would also like to acknowledge the reviewers for their detailed and constructive comments which have led to a much better presentation of our results. This work has been supported in part by the NSF grants: IIS-00-81847, and ACR-98-12572/02-96148.

References

Airey, J. M., Rohlf, J. H., and Brooks, Jr., F. P. 1990. Towards image realism with interactive update rates in complex virtual building environments. In *Computer Graphics*, vol. 24, No. 2, 41–50.

Coorg, S., and Teller, S. 1997. Real-time occlusion culling for models with large occluders. In *Proc. of ACM Symposium on Interactive 3D Graphics*, 83–90.

El-Sana, J., Azanli, E., and Varshney, A. 1999. Skip strips: Maintaining triangle strips for view dependent rendering. In *IEEE Visualization '99 Proceedings*, ACM/SIGGRAPH Press, 131 – 138.

Evans, F., Skiena, S., and Varshney, A. 1996. Optimizing triangle strips for fast rendering. In *IEEE Visualization '96 Proceedings*, ACM/SIGGRAPH Press, 319 – 326.

Greene, N., and Kass, M. 1993. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, 231–240.

Gumhold, S., and Strasser, W. 1998. Real time compression of triangle mesh connectivity. In *SIGGRAPH 98 Conference proceedings*, Annual Conference Series, ACM SIGGRAPH, 133–140.

Hao, X., and Varshney, A. 2001. Variable-Precision rendering. In *2001 ACM Symposiom on Interactive 3D Graphics*, 149–158.

Hudson, T., Manocha, D., Cohen, J., Lin, M., Hoff, K., and Zhang., H. 1997. Accelerated occlusion culling using shadow frustra. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1–10.

Humphrey, W., Dalke, A., and Schulten, K. 1996. VMD–visual molecular dynamics. *J. Mol. Graphics 14*, 33–38.

Klosowski, J. T., and Silva, C. T. 2000. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics 6 (2)*, 108–123.

Klosowski, J. T., and Silva, C. T. 2001. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics 7(4)*, 365–379.

Luebke, D., Reddy, M., Cohen, J., Varshney, A., Watson, B., and Huebner, R. 2003. *Level of Detail for 3D Graphics*. Morgan-Kaufmann, Inc.

Sukharev, S., Durell, S. R., and Guy, H. R. 2001. Structural models of the mscl gating mechanism. *J. Biophys 81(2)*, 917–936.

Taubin, G., Guéziec, A., Horn, W., and Lazarus, F. 1998. Progressive forest split compression. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, ACM SIGGRAPH, 123–132.

Wonka, P., Wimmer, M., and Sillion, F. 2001. Instant visibility. *Computer Graphics Forum 20(3) (EG 2001 Proceedings)*, 411–421.

Yoon, S., Salomon, B., and Manocha, D. 2003. Interactive view-dependent rendering with conservative occlusion culling in complex environments. In *IEEE Visualization '03*, to appear.

Zhang, H., Manocha, D., Hudon, T., and Hoff, K. 1997. Visibility culling using hierarchical occlusion maps. *Proceedings of SIGGRAPH'97*, 77–88.