Visualizing Simulated Time-Dependent Atomic Data with OpenGL

Tyler Spears[1], Bidur Bohara[2], Bijaya B. Karki[2]
[1] College of Engineering, McNeese State University, Lake Charles, LA
[2]LSU School of Computer Science and Electrical Engineering, Baton Rouge, LA

Introduction

Computer visualization is the conduit between the hard, quantified data of experimentation and a human's intuitive understanding. Consequently, computer visualization systems are a necessity in displaying not just traditionally collected data, but also data acquired from simulated experiments, due to the incredibly large volume of data acquired in such experiments. The ultimate goal of computer visualization is to have efficient and intuitive representations, not only for the benefit of scientists and their peers, but also for the understanding of the general public.

OpenGL

Due to the flexibility and computational resource requirements of visualizing large amounts of data, a general-purpose, low-level graphical programming framework is often utilized. The operating-environment independent framework OpenGL (Open Graphics Library) is a popular choice for visualization. OpenGL is a graphics specification that is most often implemented in C++, and specifies GLSL (GL Shading Language). GLSL is a C-like language used to program custom shaders (programs that run on the GPU and produce visual output).

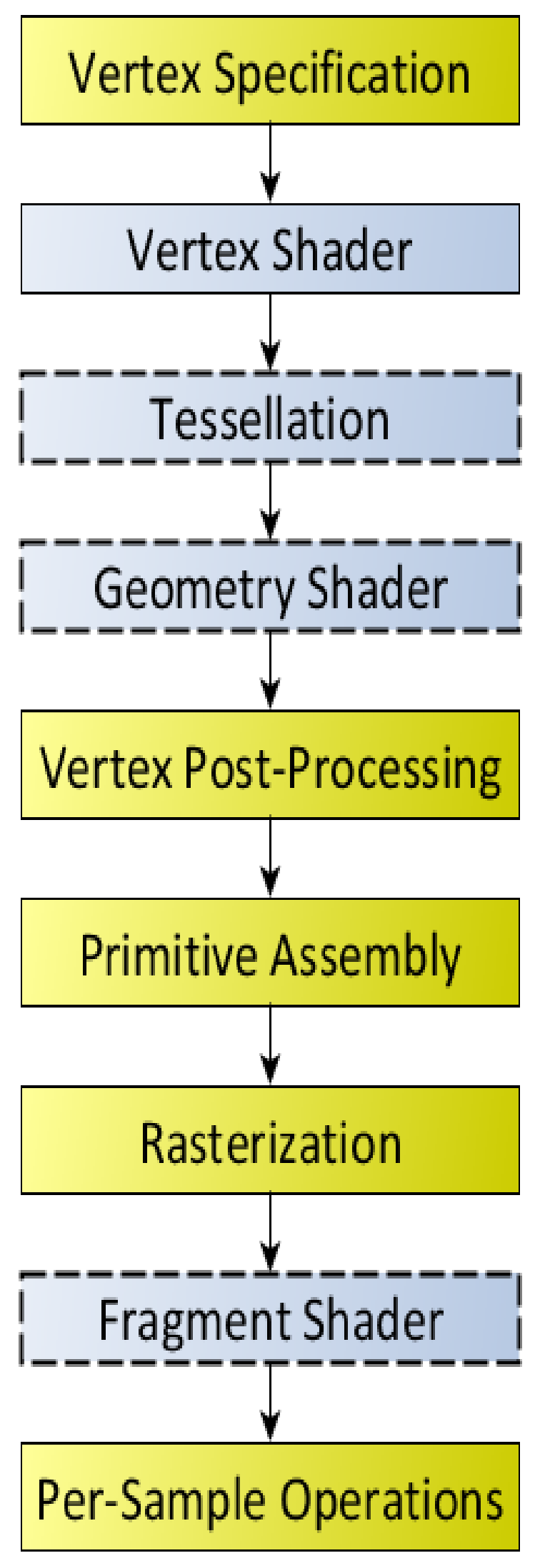

Figure 1: The OpenGL rendering pipeline.

Visualization Techniques

The flexibility of OpenGL allows for several different rendering techniques for visualizing atomic data. The first, and most readily implemented, method is the rendering of 3-D spheres. Through the use of freeGLUT (an open-source GL Utility Toolkit, a C++ library), true 3-D spheres are rendered in 3-D space.

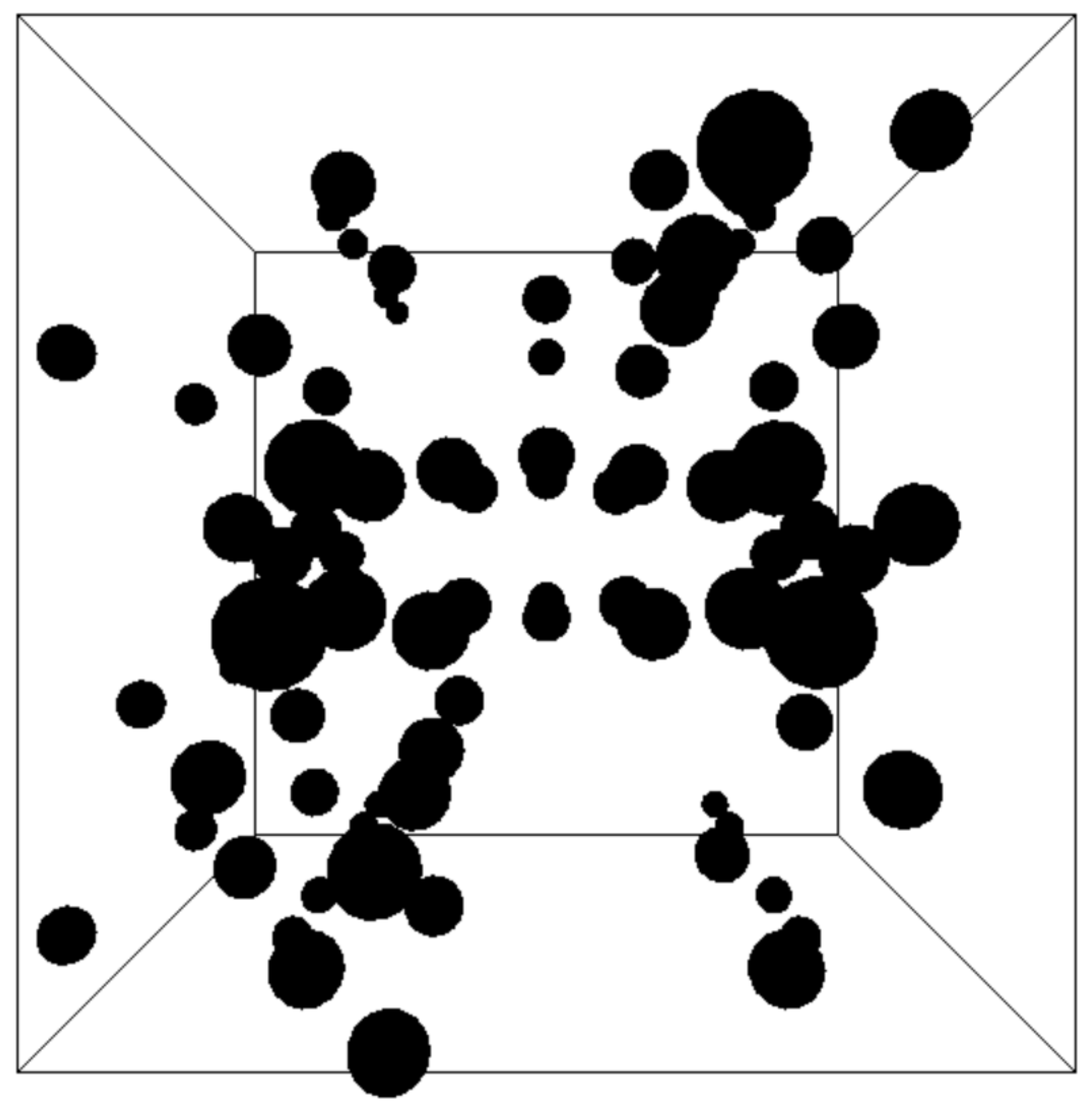

Figure 2: Rendering atomic species and distance traveled. This representation displays the atom's in their initial position, as indicated by their dark colors, and their type, indicated by size.

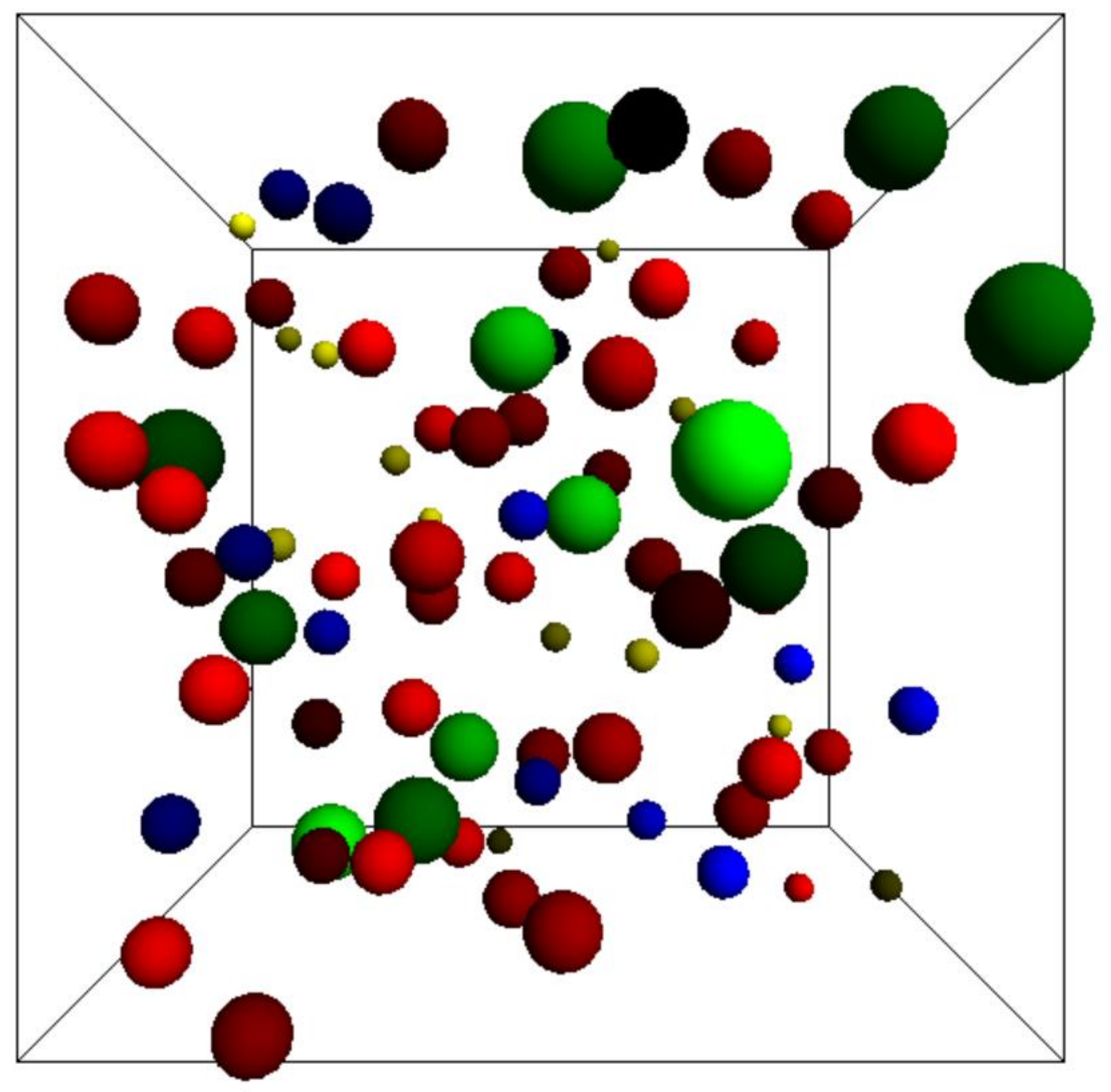

Figure 3: Atomic data representation after a set amount of time. The more visible colors indicate a longer distance traveled from the atom's respective origin point.

The second method utilizes the writing of custom shaders written in GLSL. Through the use of point-sprites (2-D points positioned in 3-D space, usually represented as a single pixel in screen space). The coefficients of the quadratic equation of the sphere are then encoded into a 4x4 matrix, and normalized and diagonalized to form the parameter space. From there, the Variance, Model, View, and Projection Matrices are used to compute the dimensions of a bounding box, a 2-D rectangle that determines the dimensions of the point sprite. A ray-surface intersectional quadratic equation, calculated in the fragment shader, is then used to determine which fragments are to be culled, resulting in a circular representation. Using the Phong shading model, the spheres are given lighting, forming the illusion of being 3-D spheres.

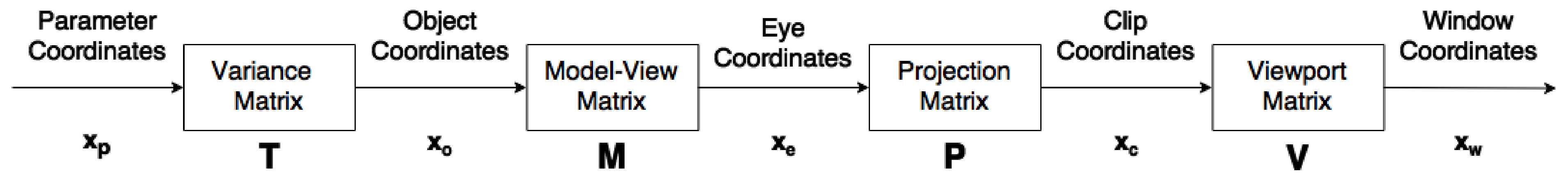

Figure 4: The sequence of coordinate spaces used in OpenGL, with an additional Parameter Coordinates space useful for rendering quadratic surfaces.

Conclusion

The visualization program displays the atomic data in an accurate and aesthetically pleasing manner. The program also encourages basic user interaction.

Out of the two methods discussed, rendering 2-D pseudo-spheres is generally preferred. Solving quadratic equations and culling fragments is more computationally efficient than rendering 3-D spheres since individual triangles are not needed. It also creates cleaner visuals, as modeling is done on a pixel-by-pixel basis.

Future work includes making the visualization more intuitive and interactive for the user. For example, giving shadows to each particle to aid in depth perception, and implementing outlines on each particle for easy differentiation.

References

1) "Rendering Pipeline Overview." *Opengl.org*. N.p., 10 May 2015. Web. 29 July 2015.
2) Sigg, Christian, et. al. "GPU-Based Ray-Casting of Quadratic Surfaces". *Eurographics Symposium on Point-Based Graphics, Boston, Massachusetts, July 29-30, 2006.* Ed. M. Botsch, B. Chen. 2006.

Acknowledgments

This material is based upon work supported by the National Science Foundation under award OCI-1263236 with additional support from the Center for Computation & Technology at Louisiana State University.