SPH (Smoothed-Particle Hydrodynamics) Plugin for OpenCOVER

Jennifer Chandler

Introduction

The SPH plugin uses COVISE's virtual reality software OpenCOVER to view data from SPH simulations of galaxy and star formation. This project is a continuation of a previous MURPA project by Andrew Paterson who built many of the features of the plugin. New additions to the plugin include a column density mode to view the data as an integral through the line of sight and the ability to adjust the number of particles displayed to achieve higher frame rates. In addition the method used to display the particles as spheres has been changed to improve performance and allow greater flexibility and reuse of code.

Background

The purpose of the SPH plugin is to provide real-time viewing of SPH simulations in multiple different viewing modes. Currently the SPLASH software written by Daniel Price can be used to view SPH simulations and generate figures for papers; however, it does not allow real-time interaction with the data. The SPH plugin for OpenCOVER uses the capabilities of the graphics card to improve performance and allow users to interact with the data in real-time by rotating, scaling, and moving the data. Most of the viewing modes of the SPH plugin have reasonably fast frame rates. Because of the limitations of displaying the simulations in real-time some of the modes compromise precision to maintain performance.

Billboarded Circles

The sphere mode, point cloud mode, and column density mode all use billboarded circles as the framework to maintain as close to real-time performance as possible. Billboarding is a technique

in which a two dimensional object is used to represent a three dimensional object by always rotating the object to face the viewer. The two dimensional object generally has a texture applied so that it looks like the three dimensional object it represents. The billboarding in the SPH plugin uses a single equilateral triangle as opposed to a square which is most often used for billboarding. This cuts down on the number of vertices that must be processed every frame. Texture coordinates are assigned to each vertex to be used by the fragment shader to determine which parts of the triangle make up the inner circle. The actual billboarding takes place in the vertex shader rather than generating geometry that faces the viewer which allows open scene graph to use display lists to render the triangles instead of specifying the vertices every frame. The triangles are rotated to face the viewer by first translating each vertex by the negative of the distance from the center of the triangle's inner circle to the origin and then multiplying the vertex by the inverse of the rotation matrix and then translating the vertex back the same distance before applying the normal viewing and projection transformations. To simplify this process the center of the circle is passed into the shader as the position of the vertex, and the offset of the vertex from the center is passed into the shader as the vertex normal since the normal is unused in the shading calculations. The equation then becomes ModelViewProjectionMatrix * (position + (inverseRotation * normal)) which saves the initial step of translating the vertex.

Column Density Display Mode

Column density is a method of viewing three dimensional SPH simulations in two dimensions by integrating the density of the particles through the line of sight. This integration can be done by calculating the contribution from each particle separately as done in the SPLASH software. The contribution from each particle is represented as a circle facing the screen with a radius proportional to its density. A 3D cubic spline curve gradient is applied to the circle so that the

color goes to zero at the edge. This gradient is multiplied by the smoothing length of the particle and then the contributions from all particles are added up to get a 2D representation of the column density. This can be done efficiently by allowing the GPU to perform the summation through the line of sight simply by rendering all the circles. To perform this summation it is necessary to turn off the depth test and enable blending with blend factors of one for the source and destination colors. Since colors are clamped to the zero to one range the color of each fragment must be scaled before assigning it. Ideally the scale factor would be equal to the highest column density value so that all final values would be in the range of zero to one before clamping; however, it is not possible to know this value before performing the summation so an arbitrary scale factor is set by the user which can be adjusted at runtime. After the summation a color scale can be applied so that the data does not have to be viewed in monochrome. The color scale is applied in a post processing step which is done by rendering the summation to a texture instead of to the color buffer. The texture is then applied to a full screen quadrilateral, and the color value in the texture is used to look up the final color in a color scale. The color scale is implemented through a second one dimensional texture with linear interpolation of the texels. The method of summing the density values as colors leads to a loss of precision because the summation happens independently in each color channel so the maximum precision is the size of a single color channel. In a standard RGBA texture each color channel is 8 bits which leads to a significant loss of precision. This results in areas of very high density with many particles each giving a small contribution to the column density appearing as low density regions because the contribution from each particle is so small that it becomes zero. Using a 32 bit floating point texture where each color channel is stored as a 32 bit float gives much greater accuracy but also results in slower frame rates. To achieve greater precision, especially when using a smaller

texture to store the summation, a different scaling factor can be used for the summation in each color channel. Then in the post processing step the best representation of the depth value is selected by choosing the value with the smallest scaling factor that has a scaled depth of less than one. If the depth value is equal to one then it is likely inaccurate because it has probably been clamped. Using this method gives smoother visual results in the lower density areas; however, in areas that transition rapidly from high to low density there are noticeable visual artifacts. Interpolating the chosen value with the next scaled value removes these artifacts and gives a smoother overall result. Another improvement in performance comes from storing the column density in a one channel texture rather than using a standard RGBA texture. This greatly decreases the size of the stored texture and results in significant increases in the frame rate. In most of the tests I ran using a one channel texture was two to three times faster than using the same resolution for an RGBA texture. This has a great impact on the project as a whole because now high resolution textures can be used in normal interaction with the program because there is no longer a great decrease in performance. However, using a one channel texture means that the interpolation method described above cannot be used. This does not present any problems though because the interpolation method only had observable improvements in the 8 bit per color channel textures.

Future Work

In order to correctly display partially transparent objects in openGL the geometry needs to be sorted from back to front to enable correct blending of the partially transparent colors. In the SPH sphere mode, which assigns transparency to the spheres based on their density, sorting the possibly millions of spheres every frame would result in such low frame rates that it would no longer be able to display in real-time and would lose all interactivity. Currently the sphere mode

sorts the spheres into bins which are rendered back to front; however, this does not give an accurate image because within each bin the spheres are not sorted and can be rendered in an incorrect order. Various methods of order-independent transparency exist which could possibly be used in the SPH plugin to achieve more accurate results while still maintaining real-time interaction. One such method of order-independent transparency called depth peeling developed by Cass Everitt uses the depth buffer to sort transparent geometry in the scene. This is a multi-pass algorithm which renders the scene multiple times, each time rendering the next closest objects and then blending all of the rendering passes at the end. This could be used in the SPH plugin to render a close to accurate image by using fewer passes than necessary to render all of the geometry. Because there are so many spheres in a simulation, the spheres furthest away from the viewer do not have a significant contribution to the image so it would not make much of a difference to the final result to include all of the passes necessary to render every sphere. Using fewer passes would allow this algorithm to perform in real-time for SPH simulations.

Conclusion

The SPH plugin can be a very useful tool to scientists who need to view SPH simulations in real-time. This can be a more efficient way to analyze data and to interact with the data with other users as well. One of the barriers to make this plugin available to scientists who would like to use it is the necessity of having the OpenCOVER software installed. This software is available for free to academic users; however, it can be very difficult to configure for use in a 3D environment.

References

Price, 2007, Publ. Astron. Soc. Aust., 24, 159-173

Delta3D Tutorial Render to Texture

http://sourceforge.net/apps/mediawiki/delta3d/index.php?title=Tutorial_Render_To_Texture

OpenSceneGraph example osgdistortion

http://www.openscenegraph.org/projects/osg/browser/OpenSceneGraph/trunk/examples/osgdistortion/osgdistortion.cpp

Mike Weiblen, GLSL Shading with OpenSceneGraph, 2005

http://mew.cx/osg_glsl_july2005.pdf

Dave Shreiner, OpenGL Programming Guide http://www.glprogramming.com/red/

Lighthouse3D, GLSL Tutorial: The gl_NormalMatrix

http://www.lighthouse3d.com/opengl/glsl/index.php?normalmatrix

Cass Everitt, Interactive Order-Independent Transparency

http://developer.nvidia.com/object/Interactive_Order_Transparency.html