Volume Rendering with libMini

Stefan Roettger, April 2007
www.stereofx.org

1. Introduction

For the visualization of volumetric data sets, a variety of algorithms exist which are typically tailored to the underlying data format. Possible data formats are point clouds, regular volumes, hierarchical volumes, rectilinear and curvilinear grids and unstructured grids. Besides the actual representation of the data, the task of volume rendering is the same: for each virtual ray that is thought to originate at the eye point and crosses the volume, the opacity and the color defined at a specific point in the volume must be accumulated on that ray.

Unstructured grids are the most flexible format, since the data definition consists of tetrahedra with the data values being defined at the corner points, the vertices. With the tetrahedra as the building blocks, any grid layout can be realized. Curvilinear grids have the same principal complexity and thus can be regarded as a sub-class of unstructured grids. Regular volumes have an implicit structure which allows to specify algorithms that take advantage of the grid layout. By mapping the regular (or rectilinear) grid to a 3D texture one can use computer graphics hardware to efficiently render this 3D texture. For this purpose, texture slicing [Drebin et al. 1988] and ray casting [Roettger et al. 2003] algorithms are known.

Our topic, however, is not to look at these algorithms in detail. Here we refer to the extensive literature. We rather have a look at the cases where one does not have to cope with a single volume but rather with multiple volumes and also with 4D volumes.

With some exceptions all of the known volume rendering algorithms fail miserably, if two or more volumes overlap or even if the layout of the volumes is only just slightly more complex than a regular tiling pattern. To understand this fact we have to make an excursion to unstructured volume rendering.

2. Unstructured Volume Rendering

Unstructured grids are most flexible but also the most difficult format to handle. This is due to the fact that the grid consists of a loose collection of tetrahedra and these tetrahedra must be processed in a depth ordered fashion to compose

the final rendered image. The depth or visibility sort of the tetrahedra corresponds to a graph ordering problem in theoretical computer science. Point clouds can be transformed to unstructured grids by applying a 3D Delaunay triangulation (or tetrahedrization) to the point cloud. Therefore, the graph ordering problem applies to point clouds as well.

The solvability and runtime of the graph ordering is well analyzed but depends strongly on the well-behavness of the graph structure. For some types of graphs fast sorting algorithms are known (e.g. for Delaunay triangulations) but for others there might no even be a solution (if the graph contains a cycle). Therefore, the general task of sorting an arbitrary tetrahedral grid is a difficult and time consuming task.

As a result, most visualization applications which need to display unstructured grids at high interactive rates typically do not use full volume rendering, but rather extract iso-surfaces (= all the points of equal data value) from the unstructured grid. Still, the extracted triangles which make up the iso-surface need to be depth-sorted if the iso-surface is semi-transparent and not just opaque. However, this sorting task can be accomplished in at least O(n logn) computational time. But even still, the sort is much more difficult if triangles intersect each other. This can happen easily for the case of multiple volumes.

For this particular case, the depth peeling algorithm [Everitt 2001] yields good results at reasonable speed but requires quite some resources on the graphics hardware. The depth peeling approach assumes that the contributions of the surfaces can be limited to a number of n frontmost surfaces. What is beyond this depth complexity is just ignored, because it's contribution is considered to be too small to be visible.

Applied back to the case of volume rendering with multiple 3D textures (each volume can be decomposed into at least 5 tetrahedra), we can conclude that if none of the volumes intersect, we can determine the order in which the volumes need to be processed in O(n log n) time. But if the volumes intersect, the assumption which was made by the depth peeling approach is no longer valid. We can easily construct a case where the topmost n contributions are almost transparent, so that neglecting all what is behind leads to severe rendering artifacts. A generic solution to this problem is beyond the scope of this article. Nevertheless, in the following we will describe an optimized variant of the depth peeling algorithm for multiple semi-transparent iso-surfaces and multiple volumes. This algorithm is implemented in the libMini rendering library (http://www.stereofx.org). The work on this implementation has been sponsored by Makai Ocean Engineering, Inc. in Apr. 2006.

3. Depth Peeling

The depth peeling algorithm is a multi-pass rendering technique. It determines the n topmost visible layers by rendering the scene 2*n times. In each pass the Z-buffer is used to peel away the top-most layer (1st pass per layer). After that, the fragments which pass the Z-buffer range test define the color of that layer. This color information is blended into an additional pixel buffer (2nd pass per layer). After the topmost n layers have been peeled away, the final image has been constructed in the pixel buffer in a back to front fashion. Since the blending is performed in a front to back fashion, the pixel buffer must have an alpha channel to store the accumulated opacity. Finally the contents of the pixel buffer are copied into the frame buffer.

The accuracy of the method depends on the number n of rendered depth layers. For typical scenes the depth complexity is between 4 and 8 layers, so that 8 to 16 rendering passes are needed. While 8 passes per scene are acceptable, 16 passes are just too many for the achieved small improvement in image quality. For scenes with a huge amount of triangles even 8 passes might be unacceptable. For iso-surfaces extracted from high-resolution volumes this is the case. Just by extracting 3 semi-transparent iso-surfaces one already has a depth complexity of 6. Therefore, we present an optimized variant of the depth peeling approach which is tailored to large-scale iso-surface visualization.

4. Real-Time Rendering of Multiple Iso-surfaces and Multiple Volumes

The depth peeling algorithm is aimed at the photo-realistic display of computer graphics scenes. For the visualisaztion of volumetric data, however, realism plays a minor role, since parametric data like air pressure or flow vorticity cannot be viewed realistically by definition. For this particular purpose, one has to find an optical model which is emphasizing the important properties of the data set, but this optical model does not necessarily need to be a realistic one. Instead we look for an optical model which helps reducing the number of passes for iso-surface rendering. Basically, there are two possible canditates for this.

The first candidate is derived by neglecting the self-attenuation of the photo-realistic optical model. The latter model assumes that on the viewing ray the ambient background light is attenuated by an optical gaseous medium . Additionally, light can be emitted by the gaseous medium itself, whereby this emission is being again attenuated on its way to the eye point. The latter observed attenuation is called self-attenuation. In order to calculate this term correctly the emissions must be processed in a depth ordered fashion. By neglecting this term the perceived color is the sum of the attenuated background light plus the sum of the emissions on the viewing ray. Both terms

can be calulated order-indpependently. Summing up the emissions is commutative. Multiplying the attenuations is commutative, too. From this perspective, the first proposed multi-pass rendering algorithm consists of 3 passes:

1. render the opaque part of the scene to get the ambient light
   (this includes the opaque background of the scene and all fully opaque iso-surfaces extracted from the volumes)
2. render all semi-transparent iso-surfaces and multiply their attenuations with the previous result
3. render all semi-transparent iso-surfaces and add their emissions to the previous result

The result of this approach is illustrated in Figure 2 with 1 opaque red iso-surface and 2 semi-transparent green and blue iso-surfaces. In contrast to this, Figure 1 shows the result of first rendering the opaque geometry and then blending the semi-transparent geometry. The latter approach is faster because it is requires fewer passes. However, it only produces visual correct results if the depth complexity of the semi-transparent geometry is at most 1. Since the depth complexity is up to 4 in our example (the back and front faces of two semi-transparent iso-surfaces yield depth complexity 4) the 2-pass method produces the discontinuity artifacts as shown in the right image of Figure 1. The 3-pass method does not show these artifacts. While those artifacts don't appear to be visible too much in still images, they clearly pop up if the eye point is moving around the scene. Also, for a 4D visualization, the artifacts also pop up and fade away with the iso-surfaces evolving over time, so that the simple but incorrect 2-pass algorithm is no choice for real-time volume rendering. For this, a minimum of three passes as described above is required.

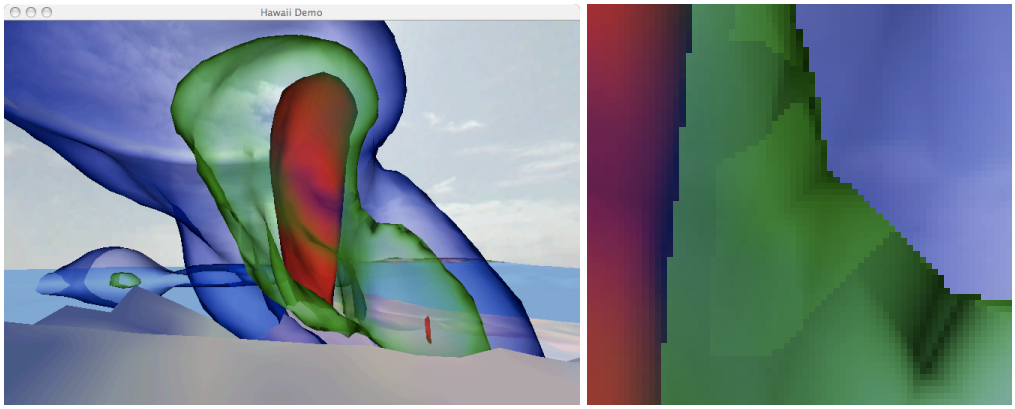

Figure 1: 2-Pass rendering (left) with discontinuity artifacts (right)

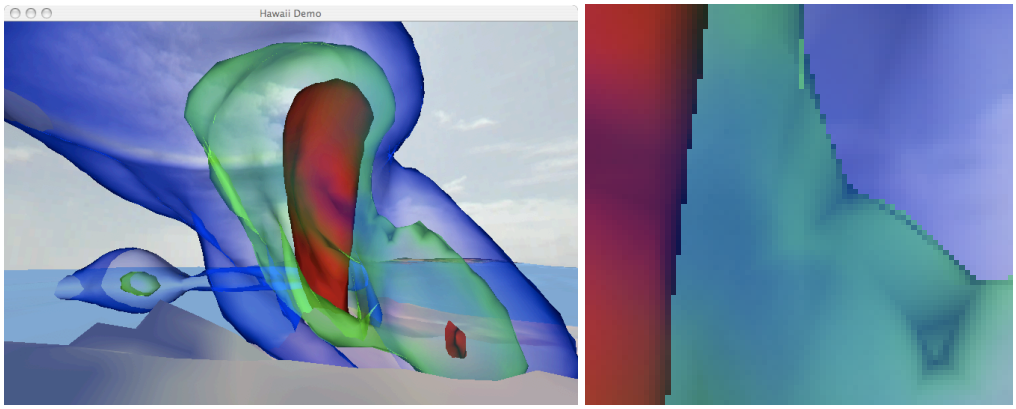

Figure 2: 3-Pass rendering with neglected self-attenuation and without artifacts

A disadvantage of the 3-pass algorithm is that the perception of depth is poor for the semi-transparent iso-surfaces, because they are treated equally by summing up the emission terms. As a result, one cannot judge what iso surface is occluding each other and most important what the frontmost iso-surface would be. This can be seen in the right image of Figure 2 where the intermixed blue and green colors make it impossible to tell which of the two iso-surfaces are in front of each other.

In order to improve the depth perception, we propose the following 4-pass rendering method:

1. render the opaque part of the scene to get the ambient light
   (this includes the opaque background of the scene and all fully opaque iso-surfaces extracted from the volumes)
2. render all semi-transparent iso-surfaces and multiply their attenuations with the previous result
3. render the back-faces of all semi-transparent iso-surfaces and keep the front-most Z-value in the Z-buffer
4. render all semi-transparent iso-surfaces with a lower or equal Z-value than the previously calculated Z-value and add their emissions to the previous result

This approach effectively emphasizes the iso-surface with the front-most back face and ignores all emissions from behind it. Therefore the depth order of the iso-surfaces is easily perceivable as can be seen in Figure 3.

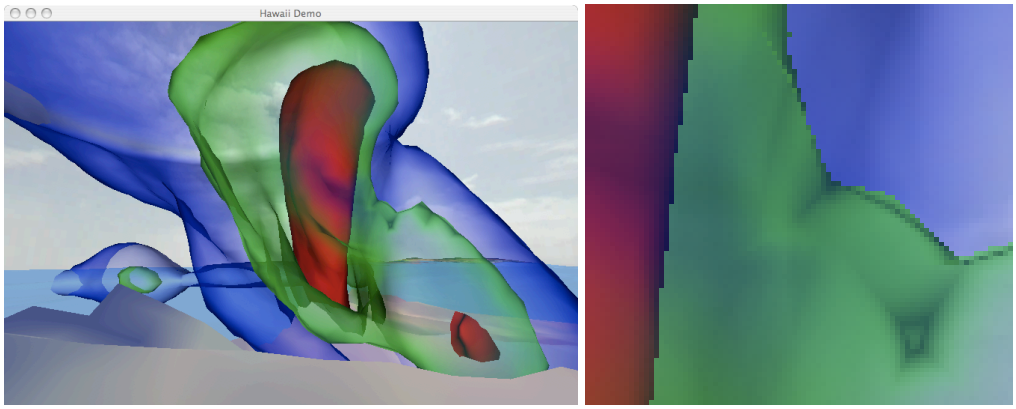

Figure 3: 4-Pass rendering with improved depth perception and without artifacts

With the two proposed methods it is possible to extract multiple semi-transparent iso-surfaces and render them without artifacts. But one can also extract iso-surfaces from multiple volumes and the visualisation still will be artifact-free with respect to the applied optical model. This is also the case if the volumes intersect, because both methods are per-pixel exact. For this to be true, each render pass must be executed for all volumes rather than executing all render passes for each volume. Then the two described methods are an efficient solution for the display of multiple semi-transparent iso-surfaces extracted from possibly overlapping multiple volumes.

Shaded Iso-Surfaces

Typically iso-surfaces are considered to have constant opacity. This is an easy assumption but it has no physical correspondence in nature. Here the opacity of any material comes from the thickness of the material layer but an iso-surface does not have a thickness at all. It is infinitesimally thin. But if we consider a small range of iso-values rather than a single iso-value then the corresponding volume defined by the range of iso-values (which in fact is an iso-spectrum) has a small but non-zero thickness. We can now compute the opacity α of the iso-spectrum by considering the so-defined thickness d of the material layer from the absorption coefficient c of the material:

$$\alpha = \exp(-c\ d)$$

As an example, we assume an iso-spectrum which is bounded by the two spherical iso-surfaces of the iso-value 1 and 0.95. A cut through this iso-spectrum is displayed in Figure 4 where the spectrum is depicted by the green dashed and the pink dotted curve. Assuming that the viewer looks at the iso-spectrum from straight above, the opacity of the spectrum is given by the red line (for c=10). If we map the diffuse cosine term of the OpenGL lighting equation (which is cos(t) = sqrt(1-x*x)) onto the graph we see that we can fit this term to the absorption α of the iso-spectrum by taking the cosine term to the

power of a and by subtracting a constant value b. In the example we set a to exp(-c 0.05) and determine b by performing a non-linear fit agaist the original curve.

This means that we can mimic the look and feel of a volumetrically rendered iso-spectrum by rendering a normal “iso-surface“ but not with a constant opacity but rather with a cosine-modulated alpha value. However, OpenGL applies a cosine lighting term to the RGB channels of each rendered fragment but not to the alpha channel. As a solution for this this problem, we simply use a fragment shader that applies the cosine term as described above to the alpha channel so that it effectively computes the approximated opacity of the corresponding virtual iso-spectrum.

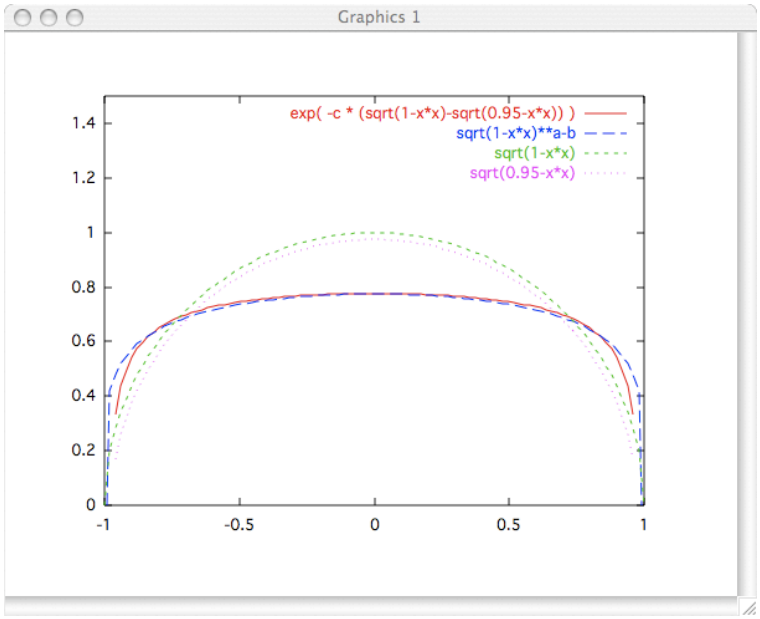

Figure 4: an iso-spectrum between the iso-values 1 and 0.95

4D Visualization

Besides the two described rendering modes the libMini rendering library also supports full 4D interpolation. This means that the volumetric data is also interpoalted in the time domain. This is important if the number of frames for the playback is much higher than the number of key frames provided in the original data. In the example shown in Figure 5 a thunder storm evolves in a series of 30 time steps. The display frame rate is 25Hz and the time over which the storm evolves during playback is about one minute. So the 30 original time steps correspond to roughly 1200 displayed frames or 40 frames per time step. With out 4D interpolation there would be a jump every 40 frames which is not acceptable. With full 4D interpolation the Storm is smoothly evolving over the entire minute of display, since the iso-surface is continuously updated from the 4D interpolated data.

Because of the high number of floating-point interpolations the the actual interpolation and the iso-surface extraction is decoupled from rendering by performing the update of the extracted triangle mesh in a back-ground thread. This efficiently utilizes dual core processors and improves the rendering performance.

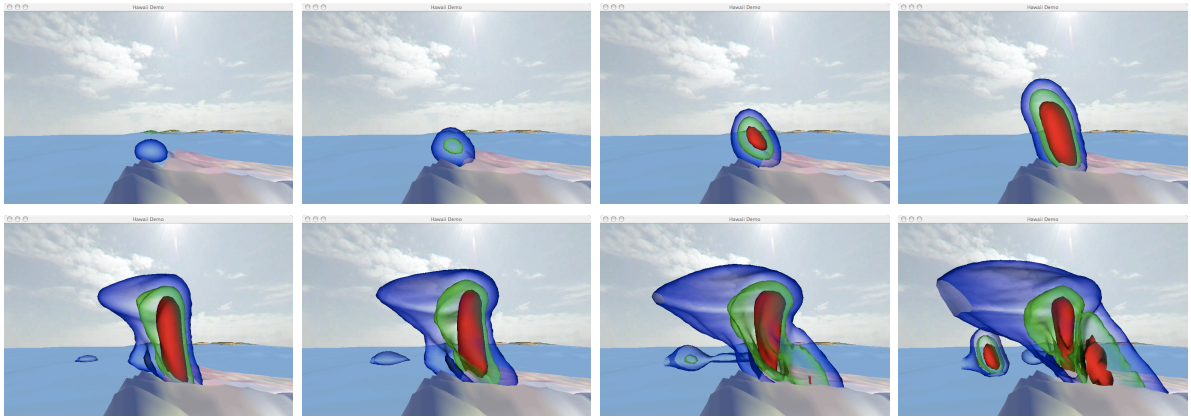

Figure 5: 4D time series of an evolving thunder storm

The rendering performance is also imncreased by reducing the number of extracted traingles in a view-dependent fashion. For this purpose, the so-called C-LOD scheme is used for iso-surface extraction. This means that the level of details and number of extracted triangles is dependent on the distance to the point of view. Details that are far away can be represented with less trianles than those which are near. As a side effect the triangulation changes with a changing point of view which leads to the so-called popping artifacts. To prevent these popping effects the iso-surfaces are geomorphed, that is interpolated between the two nearest levels of detail. This adds another dimension of interpolation, so that libMini actually performs full 5D interpolation at real-time.

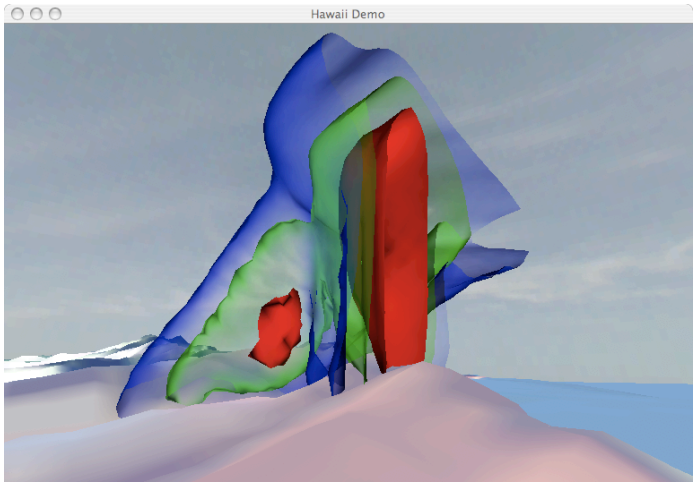

Figure 6: Visualization with an additional clipping plane

In order to get more insight into a volume, libMini can also use clipping planes to cut away occluding parts of the volume. This is depicted in Figure 6 where the right half of the thunder storm has been clipped away.

Performance

The performance of the thunder storm visualization is ca. 25 frames per second on an Apple MacBook Pro laptop with lo-end graphics. This includes rendering the terrain [Roettger et al. 1998], the ocean and the sky dome. For a larger volume the performance drops to 5-10 frames per seconds depending on the point of view and the actual size of the volume. Since the rendering algorithm is not yet fully optimized we expect a speed improvement of 2-3 times and a minimum rendering speed of 25 fps with a hi-end NVIDIA GeForce FX8800 GTX graphics accelerator.

Bibliography

[Drebin et al.1988] R. A. Drebin, L. Carpenter and P. Hanrahan. **Volume Rendering**. In *Computer Graphics* issue 22(4), pages 65-74, 1988.

[Everitt 2001] Cass Everitt. **Interactive Order-Independent Transparency**. published as *NVIDIA white paper*, 2001.

[Roettger et al. 2003] S. Roettger, S. Guthe, D. Weiskopf and T. Ertl. **Smart Hardware-Accelerated Volume Rendering**. In *Proc. Visualization Symposium '03*, IEEE Computer Society Press, pages 231-238, 2003.

[Roettger et al. 1998] S. Roettger, W. Heidrich, P. Slusallek and H.-P. Seidel. **Real-Time Generation of Continuous Levels of Detail for Height Fields**. In *Proc. WSCG '98*, EG/IFIP, pages 315-322, 1998.