Adaptive Border Sampling for Hardware Texture-based Volume Visualization

Eric LaMar

Institute for Scientific Research, Fairmont, WV

ABSTRACT

This paper introduces a technique to properly sample volume boundaries in hardware texture-based Volume Visualization. Prior techniques render a volume with a set of uniformly-spaced proxy geometries that sample (and represent) a set of uniform-depth slices. While this is sufficient for the core of a volume, it does not consider a sample's *partial* overlap at the boundaries of a volume, and this failing can lead to significant artifacts at these boundaries. Increasing the sampling rate doesn't solve the problem - but the proper calculation will. While these artifacts might not be easily visible with large datasets, this paper expands on the fundamentals of visualization by presenting a *correct* handling of sampling at boundaries - which is missing from previous literature. Our technique computes the non-unit depth contributions of the volume at the boundaries. We use fragment programs to perform this adaptive border sampling to compute the partial sample contributions and to match sampling-planes at the volume boundaries with the sampling geometry in the core of the volume.

keywords: volume visualization, image accuracy, sampling, boundary-handling

1. INTRODUCTION

Hardware texture-based volume visualization is capable of significantly faster visualization of a dataset than software based techniques. However, hardware texture-based techniques have issues with image accuracy because they do not consider the reduced contribution of samples near the edge of a volume. A sample, represented by the texture applied to some proxy geometry, assumes a finite-length segment that is wholly contained in the volume. Segments associated with samples near the boundary of a volume only partially intersect the volume. The contributions of these samples should be correspondingly reduced, but current techniques do not consider this. Secondly, if the user can see between the layers of the proxy geometry used to sample the volume, the appearance of a continuous volume is destroyed, drawing the user's attention away from the volume and to the proxy geometry.

Figure 1 shows this effect on a 4^3 checker-board dataset. The software version, shown in the left column, was created using a simple ray-caster with very high-frequency sampling.

The middle column of Figure 1 shows a hardware texture-based rendering of this same volume, with a proxy geometry spacing of one-half a voxel. We notice two things: (1) the proxy geometry is quite evident, and one's attention is drawn to it and not the volume. (2) the opacity does not gently fall off towards the edge of the dataset. The geometry is uniformly opaque, then suddenly drops to zero at the boundary - which is not physically proper. Also, increasing the sampling rate does *not* solve the problem. It will make the problem less visible at the cost of dramatically decreased rendering rates (increasing the sampling rate by a factor of just two results in a 50% decrease in performance); in addition, increasing the sampling rate will make the problem of numeric precision much worse (see LaMar[1]).

Finally, the right column of Figure 1 shows that our *adaptive border sampling (ABS)* technique captures the proper opacity extinction at the edges of the volume, and that one cannot see the proxy-geometry. We wrap the volume with simple geometry and render that geometry with our ABS fragment program that calculates partial-voxel contributions. This solution comes at the cost of a small (~8%) drop in performance for a 64^3 voxel brick.

2. RELATED WORK

Cabral *et. al.*[2] discuss how hardware-accelerated texturing can perform volume rendering and how this technique also can be used to reconstruct a volume from tomographic image sets.

Kim *et. al.*[3] and Meißner *et. al.*[4] have compared the imagery and artifacts of different rendering modalities and have noted the differences between techniques but did not comment on how boundary-sampling artifacts degrade image quality.

Weiler *et. al.*[5] discuss issues with multiresolution volume visualization. They present a technique to mitigate the interference pattern that appears in volume visualization imagery from proxy geometry of differing sample rates (i.e., between bricks different levels-of-detail in a multiresolution decomposition). Their technique computes a set of transfer functions and patching proxy geometries to "fill-in" the gaps. While their technique

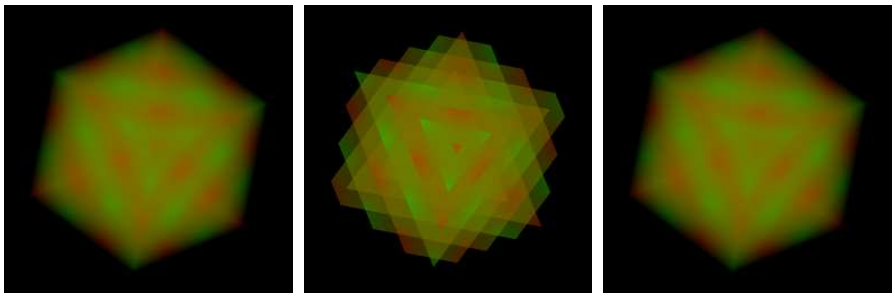

Figure 1. A comparison software, simple hardware-texture based sample, and our *adaptive border sampling* technique on a 4^3 checker-board dataset.

does significantly reduce the interference problem between blocks, it does not address the more general issue of boundary handling and will not work for rendering a single block. Also, the cost of computing patching geometry and downloading new transfer function texture palettes to the hardware occupies a large portion of the overall rendering time.

Rezk-Salama *et. al.*[6] discuss several techniques to improve speed and accuracy (though, not at the same time) through sub- and super-sampling placement of proxy geometry in a volume. While their technique can decrease the boundary-sample error, it does not go away, which can be seen quite easily in Figure 5 of that paper.

Engel *et. al.*[7] introduce pre-integrated volume rendering to solve the problem of sampling high-frequency transfer functions. Their technique does not account for partial volume contributions.

Wylie *et. al.*[8] show how to do rendering of tetrahedra using collapsible geometry and vertex programs, and demonstrated its use on unstructured datasets. However, it requires producing a large number of vertices, most of which are thrown away. This results in an extremely high per-tetrahedra overhead, making the technique a poor choice for large, structured data.

In,[1] we presented two techniques to improve the accuracy of volume visualization on large, highly-transparent datasets. We discussed how limited numeric precision affects imagery and that increasing the sampling rate also increased the errors in the resulting imagery.

Our new, ABS technique differs in that it considers the construction of the volume visualization integral directly and that it allows the composition of a non-ordinal length paths through a volume. We use simple datasets to motivate the problem, and we observe that, while it is difficult to see the problem with large datasets, our technique addresses and improves on the theoretical basis of volume visualization.

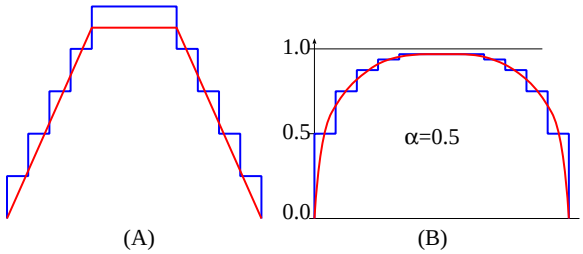

Figure 3. (A) shows the real thickness in red and the computed thickness in blue of the volume from figure 2(B). (B) shows the final opacity for figure 2(B) for a per-voxel opacity of $\alpha = 0.5$.

3. PROBLEM CHARACTERIZATION

The basic assumption of sampling with respect to texture-based volume visualization is that a sample represents and approximates the integrated opacity, emission, and absorption of the volume over a segment of space, and that all segments lie entirely within the confines of the volume. However, not all samples are far enough away from the boundary of the volume so that the associated segment lies entirely within the volume. We see this in Figure 2. Figure 2(A) shows a 4^2 voxel volume, at an angle (the viewer is directly above, looking down), intersected by five sample planes (i.e., proxy geometry), representing five sets of sample points.

Without loss of generality, we assume that a sample point lies at the middle of its corresponding segment, and that the sampling planes are placed one voxel apart. If we show the region that a set of sample points on one plane represent, we obtain a rectangle (which we will call the sampling volume), shown in Figure 2(B) in light blue. We observe that portions of the sampling volumes do not

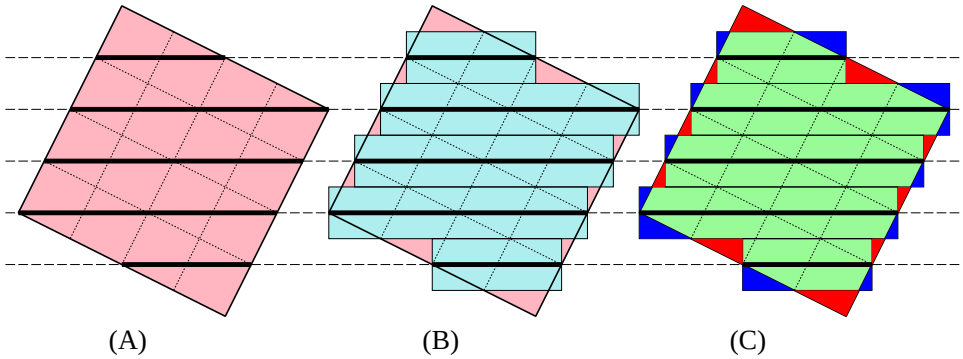

Figure 2. A rotated 4^2 voxel data set, intersected by five sample planes. (A) shows the real volume in red and proxy-geometry/sampling-planes as heavy black lines. (B) shows the effective coverage of the sample regions in light blue. (C) shows the erroneous contribution of non-existent data in blue and the missing contribution of real data in red.

completely cover the data volume in some areas and extend beyond it in other areas. This is shown in Figure 2(C), with non-represented regions shown in red, and incorrectly represented regions in blue. The opacity (and hence, the emission and extinction) for sample points inside the volume whose segments extend outside of the volume should be much smaller because the intersection of the sample's segment and the volume is much shorter than the normal segment length. Figure 3 illustrates the difference between the computed and actual thickness and final opacity of the volume shown in Figure 2.

4. *ADAPTIVE BORDER SAMPLING*

The rendering of a volume with our technique requires breaking the rendering of the volume into three disjoint regions with respect to the view direction, as shown in Figure 4. These regions are the center region (B), the front and back faces (C), and edges (D). The center region is rendered as prior techniques, albeit over a smaller domain, while the face and edge regions provide a transition from the more opaque interior to the boundary. The rendering is ordered back faces, center, edges, and front faces (if compositing back-to-front, or reverse if compositing front-to-back).

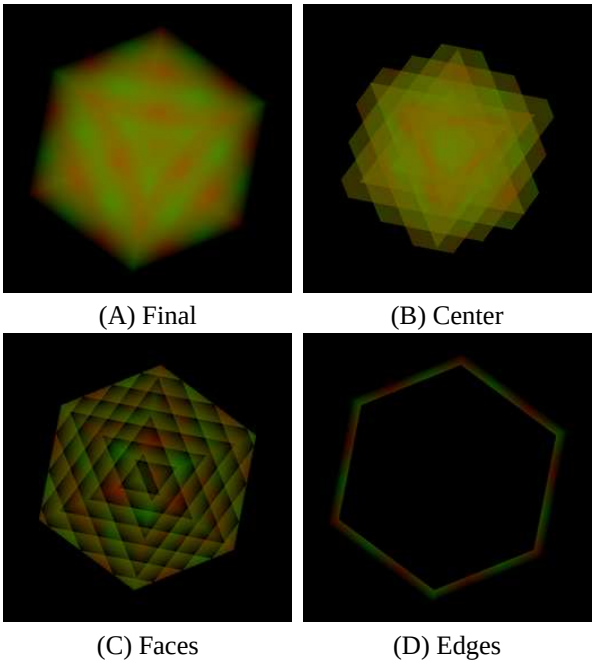

Figure 4. *Adaptive Border Sampling:* Image (A) is the sum of its parts, (B)-(D).

4.1. Center

The first step is to compute the *unit-segment* region, the region where the segments of all possible sample points are contained entirely within the volume. The orientation of the segments is toward (or away from) the viewer. Given a sampling distance (i.e., segment length) of T, the *unit-segment* volume lies between $\frac{1}{2}T$ behind the faces of the volume oriented toward the viewer and $\frac{1}{2}T$ in front of the faces oriented away from the user. This is shown in Figure 5 for different orientations of the volume. The red vectors' lengths are $\frac{1}{2}T$ and are parallel to the view direction. Note that the size and shape of the *unit-segment* volume is not constant.

The extent of the *unit-segment* volume is calculated as follows (and shown in Figure 6). Given a volume of size N, with spatial extents $A =< 0, 0, 0 >$ and $B =< N, N, N >$, a viewing transformation matrix M, and a unit vector in the direction of the viewpoint V, then the extents of the *unit-segment* volume, C and D, are defined as:

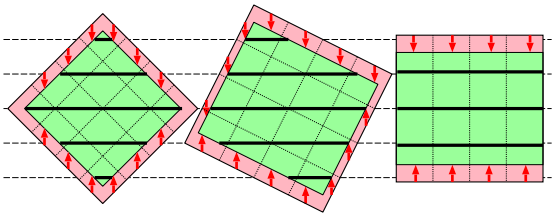

Figure 5. *Unit-segment* volume (in green) vs full volume (in red). Red arrows show "contraction" of full volume to unit-segment volume with respect to view direction.

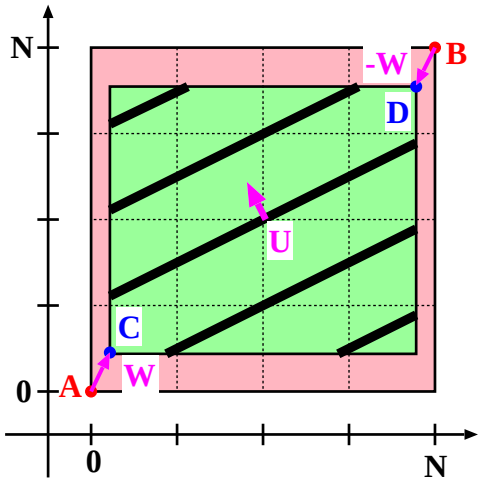

Figure 6. Construction of the *unit-segment* volume from the full volume. A and B are the min/max bounds of the full volume. The view direction vector U gives the contraction vector W, which produces the min/max bounds of the unit-volume, C and D.

$$
\begin{aligned}
\overrightarrow{U} &= M^{-1} \times \overrightarrow{V} \\
\overrightarrow{W} &= \frac{1}{2} abs(\overrightarrow{U}) = \frac{1}{2} < |U_x|, |U_y|, |U_z| > \\
C &= A + \overrightarrow{W} \\
D &= B - \overrightarrow{W}
\end{aligned}
$$

4.2. Faces

The face regions are defined as the regions of the original volume to which the *unit-segment* volume projects. This is shown in Figure 7 in 2D. The blue lines show the front faces (with respect to the viewing direction), and the red lines show the back faces. The thickness of the faces are one-half that of the sampling thickness. The face geometry is just like "normal" proxy geometry in that it is embedded in a volume and is textured with the space it

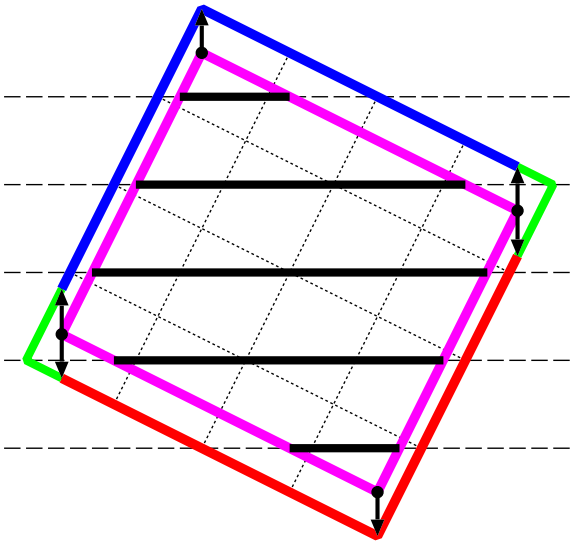

Figure 7. Relationship of *face* (in blue and red) regions with respect to the *unit-segment* (purple) volumes in 2d. The black vectors' length is $\frac{1}{2}$ of a voxel.

intersects. However, if the face geometry is placed coincident with the faces of the full volume, we will not be able to match and blend with the proxy-geometry of the *unit-segment* region. Also, if the geometry is placed coincident with the faces of the *unit-segment* volume, then we would not render the proper spatial extent of the full volume.

Our solution is to place the geometry coincident with the full volume, but *fold* the sampling planes towards the volume center (see Figure 8). When we assign the texture coordinates for a vertex of the sampling geometry, we add a second texture coordinate that encodes the depth of the vertex with respect to the viewing point. Without less of generality, we assume that the sampling thickness is one voxel, or a depth of 1. The placement of the proxy-geometry in the *unit-segment* volume is such that it is placed at the mid-point between ordinal depths (thick, black lines in Figure 8(A) and (B)). This means that the extent of a sample plane is from one ordinal depth to the next ordinal depth.

The remaining contribution of the volume is a series of regularly shaped wedges, whose depth ramps linearly from 0.0 to 1.0. The sample points are *folded* toward the *unit-segment* volume. This is shown in figures 8(A) and 8(B), where the blue and red dotted lines show the placement of the face proxy-geometry, while the thick blue and red lines show the computed location of the sample points. A *fold* vector is computed (per-frame) in the direction of the viewer. The sample point is computed by taking the

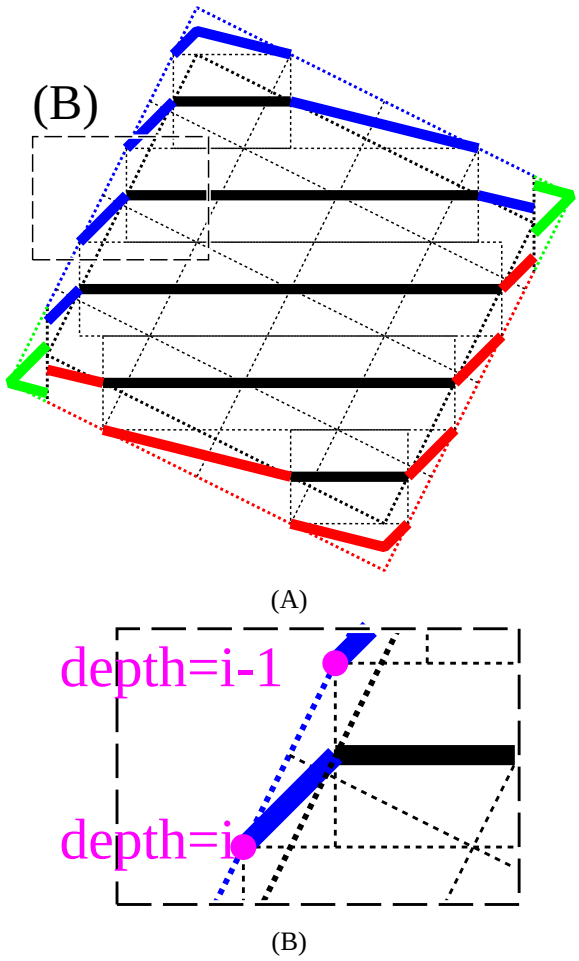

Figure 8. Folding: The face proxy geometry is placed coincident with the faces of the full volume, but the sampled points blend (*fold*) between the full volume faces and the *unit-segment* faces. Figure (B) (inset from (A)) shows details of *folding*.

current texture coordinates and moving it in the *fold* direction by a ratio of one-half the current depth of the ramp.

4.3. Edges

Edges, produced by drawing geometry that links corresponding edges of the *unit-segment* volume and the whole volume, are shown in green in Figures 7 and 8. The thickness of the edges is 0 at the outer boundary of the whole volume and 1.0 at the boundary of the *unit-segment* volume. The thickness at the *unit-segment* edge never varies because it is *defined* to be constant by the construction of the *unit-segment* volume.

4.4. Code

Figure 9 show the fragment program for rendering *face* proxy geometry. Each vertex of the original polygon of

```
void main( in      float4      tex    : TEXCOORD0
         , in      float4      depth  : TEXCOORD1
         , out     float4      color  : COLOR
         , uniform sampler3D   volume
         , uniform samplerRECT trans
         , uniform float3      fold
         , uniform float       thickness
         )
{
  float  fracdp=frac(depth.x);
  float3 smplps=tex.xyz - fold*fracdp;
  float  den   =tex3D( volume, smplps ).a;
  float4 temp  =texRECT( trans,half2(den*256,0));
  color.a      =1-pow(1-temp.a,fracdp*thickness);
  color.rgb    =temp.rgb * color.a;
}
```

Figure 9. The "Face" Cg fragment program compiles to 12 instructions under the FP30/FP40 profiles.

For each fragment, compute the local thickness of the sample by extracting the fractional portion of the fragment's depth (*depth)*. Move the sample point in the direction of the viewpoint by the product of the viewing direction vector (*fold*) and the fraction depth (*fracdp*). Obtain the density value (*den*) from the volume data (*volume*), and apply the transfer function (*tranfunc*). Compute the partial opacity value, assign this to the alpha channel (*a*), and weight the color channels (*rgb*).

```
void main( in      float4      tex    : TEXCOORD0
         , out     half4       color  : COLOR
         , uniform sampler3D   volume : TEX0
         , uniform samplerRECT trans  : TEX1
         )
{
  half val  = tex3D( volume, tex.xyz ).a;
  color     = texRECT( trans, half2(val*256,0) );
  color.rgb = color.rgb * color.a;
}
```

Figure 10. The "Center" Cg fragment program compiles to six instructions under FP30/FP40 profiles - six less than the "Face" program.

For each fragment, obtain the density value (*den*) from the volume data (*volume*), and apply the transfer function (*tranfunc*). Weight the color channels (*rgb*) by the opacity (*a*).

the proxy geometry is assigned two texture coordinate values. The first is the position within the volume, and the second is the distance of the vertex from the viewpoint. We also give the fragment program two invariant values: the direction of the viewpoint (*fold*), and the distance between the sample planes (*thickness)*.

Edge proxy-geometry is rendered with the same fragment program as faces, but with *depth* set to 0.0 at the volume boundary vertices and 0.5at the unit-volume vertices. Two sets of edge geometry are rendered, one set is front facing and the other is rear facing. The *unit-segment* or center region is generated with a simpler fragment pro-

Volume Size	Frame/Sec Old	New	%Drop
32^3	33	27.7	16%
64^3	15.2	13.9	8.6%
128^3	7.02	6.7	4.6%
256^3	3.26	3.19	2.1%

Table 1. The *ABS* technique incurs a very small penalty, shown for volumes 32^3 to 256^3 voxels. The sample interval is $\frac{1}{2}$ of a voxel. Image size is 512^2 pixels.

gram show in Figure 10.

5. RESULTS

The *ABS* technique can capture the appearance of a ray-cast implementation, as shown in Figures 11 and 12. Figure 11 shows proper drop-off at the edges of the volume, and one cannot see the proxy geometry.

Figure 12 demonstrates the ABS technique on a cylinder dataset. The cylinder's density linearly varies from 1.0 on the center. The color transfer function maps all densities to white while the opacity linearly maps density to opacity. The lower set of images show the equalized differences from the software version. Note that the old technique has significant imprinting from the proxy geometry at the top and bottom of the cylinder, and imprinting in the middle from the interference between the two sets of proxy geometry with different spacings.

We use 16-bit floating point values in the fragment programs instead of 32-bit floats because available GPUs only support 16-bit blending in hardware - and blending of 32-bit floats is only supported in software - and is about three orders of magnitude faster.

Our technique requires some additional computational effort, though, as Table 1 shows, this is not very large. For a brick of 64^3 voxels, projected to a 512^2 pixel screen, our technique is only 8.6% slower. For four datasets of 32^3 to 256^3 voxels, if we turn off rendering of the *unit-segment* region, setup the volumes to projects to the same number of pixels (i.e., to just fill a 512^2 pixel window) and just draw the face and edge regions, we observe a constant frame rate of 135 fps. This demonstrates that the effort required is proportional to the pixels rendered, and not to the size of the volume.

6. CONCLUSIONS

We have presented a technique for properly handling and rendering volume and volume boundaries. We have solved the problem with handling boundaries and, by extension, solved the brick-interface proxy-geometry interference problem with multi-block and multiresolution hardware-texture volume visualization systems. This technique is important in that it solves a problem present in volume data sampling with respect to an oriented ray-segment. We observe that splitting the rendering process into *face* and *center* fragment programs (and rendering phases) avoids wasting a large amount of complex, border-handling fragment processing on the *center* (unit-segment) volume.

REFERENCES

1. E. C. LaMar, "On Issues of Precision for Hardware Texture-based Volume Visualization," in *Visual Data Exploration and Analysis*, R. F. Erbacher, P. C. Chen, J. C. Roberts, and C. M. Wittenbrink, eds., **5295**, pp. 246–255, Jan. 19–23, 2004.
2. B. Cabral, N. Cam, and J. Foran, "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware," in *Volume Visualization*, A. Kaufmann and W. Krueger, eds., pp. 91–98, Oct. 1994.
3. K. Kim, C. M. Wittenbrink, and A. Pang., "Extended Specifications and Test Data Sets for Data Level Comparisons of Direct Volume Rendering Algorithms," *IEEE Transactions on Visualization and Computer Graphics* **7**, pp. 299–317, Oct. 2001.
4. M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis, "A Practical Evaluation of Popular Volume Rendering Algorithms," in *Volume visualization*, pp. 81–90, Oct. 8–13, 2000.
5. M. Weiler, R. Westermann, C. Hansen, K. Zimmermann, and T. Ertl, "Level-Of-Detail Volume Rendering Via 3D Textures," in *Volume Visualization*, pp. 7–13, ACM Press, Oct. 8–13, 2000.
6. C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl, "Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization," in *Graphics Hardware*, B.-O. Schneider and W. Strasser, eds., pp. 109–118, Aug. 21–22, 2000.
7. K. Engel, M. Kraus, and T. Ertl, "High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on on Graphics hardware*, pp. 9–16, Aug. 12–13, 2001.
8. B. Wylie, K. Moreland, L. A. Fisk, and P. Crossno, "Tetrahedral Projection using Vertex Shaders," in *Volume Visualization*, pp. 7–12, Oct. 27–Nov. 1 2002.

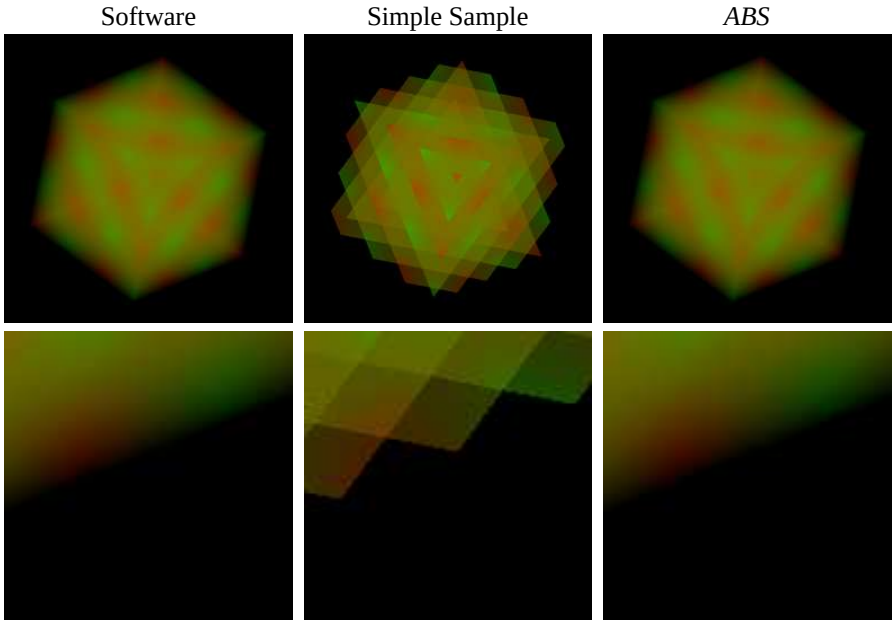

Figure 11. Results of the *ABS* technique on a 4^3 checker-board dataset. The close-ups of the lower-right edge of the volume show the success of the *ABS* technique.

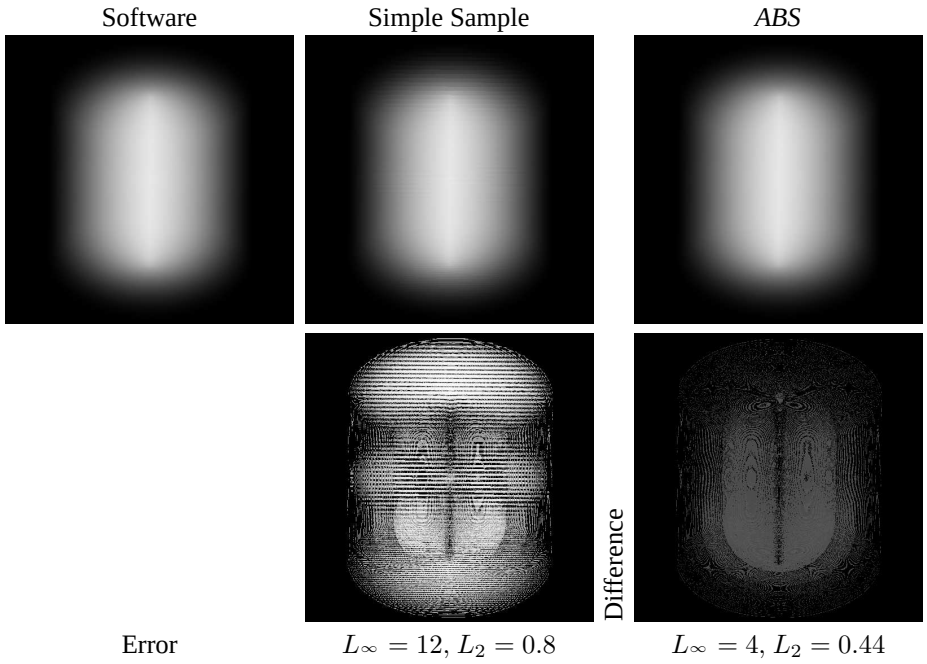

Figure 12. Results of the *ABS* technique on a multi-resolution cylinder dataset: 32^2x16 on the top and 64^2x32 on the bottom. The difference image for Simple Sample shows the errors at the top and bottom, and the interference pattern at the interface caused by the two different resolutions. The difference images are normalized to show the location and nature of the artifacts (the ABS difference image is normalized to the Simple Sample difference image, so is appropriately less bright). The ABS technique is free of artifacts due to proper border sampling; the artifacts present are due to the use of limited precision, 16-bit fbating-point pixel-buffers.