Decoupling Polygon Rendering from Geometry using Rasterization Hardware

Rüdiger Westermann, Ove Sommer, Thomas Ertl
University of Utah, University of Stuttgart

Abstract. The dramatically increasing size of polygonal models resulting from 3D scanning devices and advanced modeling techniques requires new approaches to reduce the load of geometry transfer and processing. In order to supplement methods like polygon reduction or geometry compression we suggest to exploit the processing power and functionality of the rasterization and texture subsystem of advanced graphics hardware. We demonstrate that 3D-texture maps can be used to render voxelized polygon models of arbitrary complexity at interactive rates by extracting isosurfaces from distance volumes. Therefore, we propose two fundamental algorithms to limit the rasterization load: First, the model is partitioned into a hierarchy of axis-aligned bounding boxes that are voxelized in an error controlled multi-resolution representation. Second, rasterization is restricted to the thin boundary regions around the isosurface representing the voxelized geometry. Furthermore, we suggest and simulate an OpenGL extension enabling advanced per-pixel lighting and shading. Although the presented approach exhibits certain limitations we consider it as a starting point for hybrid solutions balancing load between the geometry and the rasterization stage and we expect some influence on future hardware design.

1 Introduction

Despite the tremendous attention that is currently paid to volume graphics we still find polygonal models dominating the field of interactive computer graphics. Recent hardware advances brought performance rates of about 5 million textured triangles per second to low priced PC graphics adapters and the development does not seem to have reached its peak. Nevertheless are 3D scanners and advanced animation systems the source of models composed of millions of polygons which prohibit real-time rendering on even the most expensive high-end systems. Coping with that level of complexity is still an important field of computer graphics research and the investigated approaches fall into three broad classes. *Polygonal simplification* reduces the number of triangles to be used in level-of-detail representations of small or distant objects. *Visibility processing* eliminates polygons which are not in the viewing frustum or which are occluded by other objects before they are sent down the graphics pipeline. *Texturing* conveys visual detail within a polygon allowing larger and fewer triangles to be used in a scene. While the first two approaches offload vertex processing from the geometry engine to a preprocessing step in the CPU, does the third approach substitute geometry processing by rasterization.

Modern graphics architectures try to support the latter by increasing the performance of the rasterization and texturing subsystem. Silicon Graphics' InfiniteReality system [11] introduced virtual textures and efficient loading and paging of 64 MByte texture memory as well as fill rates of about 800 million pixels per second. 3D PC graphics adapters like Nvidia's Riva TNT claim 180 million pixels per second and pro-

vide fast texture loading through a AGP2X bus and single pass multi-texturing. Our approach for rendering large polygonal models, which we present in this paper, tries to exploit this tremendous processing power in the rasterization hardware. However, instead of working with 2D textures, we follow the ideas of volume graphics. That means, rather than reducing the geometric representation in such a way that the number of available primitives still approximates the original data sufficiently, we aim in developing a volumetric model that provides an alternative approach for the rendering of complex polygonal models.

2 Overview and related work

The basic idea of volume graphics as introduced in [2, 6, 7, 14] is to represent graphics objects on a 3D raster of volume primitives called voxels, thus repeating the 2D transition from vector graphics to raster graphics now in 3D. One of the advantages of this technique is the decoupling of the rendering complexity from the complexity of the geometry. It comes at the additional cost of the 3D scan conversion of the objects and the related aliasing errors as well as the high memory and processing demands of rendering the voxelized representation, which up to now limited the broad success of volume graphics. Interactive rendering of high resolution volumes became practical only recently based on sophisticated algorithms [9], dedicated hardware architectures [8, 12] or 3D-texture slicing available in medium to high-end graphics workstations [1].

The basic idea of texture-based volume rendering is the compositing of planes perpendicular to the viewing direction with the plane pixels trilinearly interpolated from a 3D scalar texture and transformed to RGBα by a lookup-up table. It proved to be a valuable tool for the visualization of medical and scientific data, but the lack of lighting capabilities made it hard to render objects with opaque surfaces. However, using the alpha test, the stencil buffer and a few more OpenGL extensions, it turned out that isosurfaces can be extracted from a volume and rendered with diffuse lighting as fast as traditional texture slicing [15]. It is a slightly modified and extended version of this method that we use in our paper to render shaded polygonal surfaces.

In order to transform the rendering problem into a isosurface extraction task we have to voxelize the polygonal scene in a special way. Gibson [3] has shown that the inaccuracies of surface renderings from intensity-based volumes can be overcome by means of distance maps. The distance-to-closest-surface function varies smoothly across surfaces and can be used to determine the surface as the zero-value isosurface of the distance volume and the normals from its derivatives. While there are many other sophisticated approaches for voxelizing meshes [2, 6] we follow a rather straightforward implementation of distance maps since efficient voxelization is not a main concern of our method.

One of the advantages of volume rendering as opposed to isosurface extraction in visualization applications is, that potentially every voxel will contribute to the final image and no volumetric information in between surfaces is lost. Therefore it is usually no waste of rasteriziation resources that the 3D texture slicing algorithm renders polygons which cover the entire volume. This is no longer true for volume rendering of voxelized polygonal models because in general there are many empty voxels in the bounding volume of the entire object. We overcome this problem by partitioning the scene into a hierarchy of axis-aligned bounding boxes. The efficient determination of bounding box hierarchies is a widely investigated problem in computer graphics, but since it is not in the main focus of our approach, we follow the ideas suggested in [4]. Having bounding boxes available that provide a tight enclosing of a group of polygons we can dramatically decrease the rasterization load by slicing many small 3D textures

with fewer empty voxels instead of a large one.

The set of bounding boxes serves for a further improvement in our approach since we do not have to voxelize with the same resolution in each box. Instead we start out with a very coarse voxelization and determine the approximation error by estimating the one-sided Hausdorff distance [5] of the isosurface to the original polygonal representation. Only if the error exceeds a certain threshold do we continue with the next finer resolution. Thus, we get coarse 3D textures for very smooth surfaces and a fine resolution for highly curved parts. However, we have to invest some effort to guarantee continuity across neighboring boxes.

Although the use of bounding boxes as described dramatically decreases the number of operations that have to be performed in the rasterization unit we still recognize a considerable waste of resources. Since the voxelized models are opaque the distance values inside do not contribute to the final image but have to be rasterized as well. In order to avoid the processing of structures inside the model we developed a technique particularly designed to render thin boundary regions. We construct two coarse meshes: an outer one which entirely covers the original model and an inner one which is enclosed by it. Then we tessellate the area between the two meshes within each slicing plane and we render only the generated triangles. In this way the rasterization is restricting to the thin region around the isosurface that represents the voxelized geometry.

Subsequent shading is then performed in image space implementing the lighting evaluations in software on a per-pixel basis. Some evaluations are already supported by the OpenGL imaging subset, others could be efficiently accomplished using an proposed OpenGL extension. The outlined extension works independently of our volumetric approach, but it enables high quality lighting and shading evaluations on a per-pixel basis only using gradient textures.

We now describe the organization of the rest of the paper. Section 3 explains the core algorithm we developed to adaptively convert polygonal meshes into multiresolution distance volumes. Section 4 focuses on the details of rendering the models by extracting isosurfaces from the 3D textures. Finally, we present some results and draw some conclusions.

3 A volumetric model for polygonal rendering

Just recently, 3D textures have been established as fundamental rendering primitives in volume visualization [1], but even more importantly they have proved to be an effective tool for the rendering of isosurfaces on a per-pixel basis thereby avoiding any polygonal representation [15]. As a consequence, previous work on the voxelization of polygonal models [2, 7, 14] gains a completely new relevance. No longer is the rendering of large scale volume data too slow to allow for interactive frame rates, which up to now prohibited its break through in practical applications.

However, the use of voxelized models still exhibits certain limitations due to the fact that in general these models contain large parts covered with redundant information, i.e. empty voxels as shown in Figure 1, which demand the rasterization hardware to no purpose. In order to avoid the encoding and thus the processing of empty regions we propose an adaptive voxelization based on axis aligned bounding boxes.

3.1 Axis aligned bounding boxes

In [4] oriented bounding boxes have been introduced for fast and robust collision detection between arbitrary parts of complex scenes. The construction of a bounding box

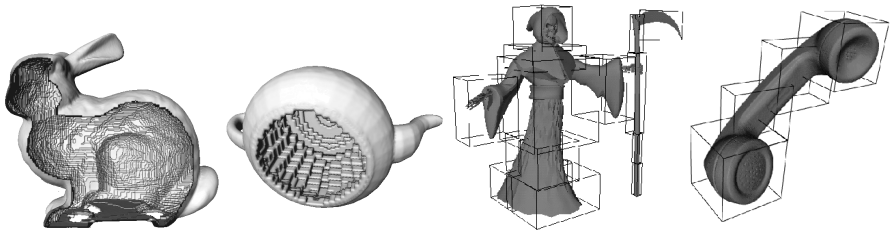

Fig. 1. The left two images demonstrate sparse volumetric representations as they usually result from voxelizing polygonal models. The right two images show axis aligned bounding boxes for different models.

hierarchy allows for efficient determination of interferences by recursively traversing the underlying tree structure. Although oriented bounding boxes provide much tighter bounds for the enclosed objects compared to axis aligned bounding boxes (AABB) we are not sure whether this kind of representation can be effectively integrated into our approach due to reasons described later on.

As a consequence our partitioning strategy is actually restricted to axis aligned bounding boxes, but it slightly differs from the one proposed in [4]. During the top-down construction of the hierarchical data structure we split each box that contains more than a given number of primitives into two smaller boxes each of them enfolding a disjunct subset of the primitives. In order to keep the space covered by adjacent boxes as small as possible we choose a dividing plane that is orthogonal to the main axis along the largest dimension of the AABB (cf. Figure 1) instead of selecting the axis that is orthogonal to the largest eigenvector of the covariance matrix.

In the actual implementation, however, we do not consider the hierarchical tree structure. Although the entire hierarchy is generated only a particular level is used further on.

3.2 Distance volumes

Once the underlying domain has been partitioned into a number of axis aligned bounding boxes we start building our volumetric model. Each partition is converted into a scalar volume consisting of the distance-to-closest-surface function at every grid point. A detailed description of the fundamental algorithms to voxelize surfaces and to encode polygonal descriptions in volume data by means of distance maps can be found in [2, 3, 6, 14] and will not be discussed in this work.

Some of the nice features distance maps offer are the smooth variation of the internal representation across surfaces, the encoding of the original surface model by the zero-values and the low variation of surface gradients calculated from the distance map. 3D distance volumes thus offer an attractive alternative to encode polygonal meshes in a volumetric representation at the same time allowing for the quite optimal reconstruction of the converted models. Putting together distance volumes and axis aligned bounding boxes we are now able to adaptively convert arbitrary polygonal scenes into volumetric models thus considerably reducing the memory requirements and the rasterization operations to be performed.

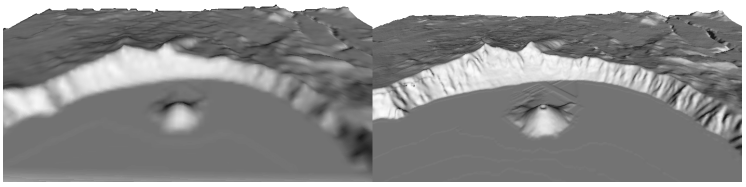

Fig. 2. The terrain model was converted into a signed distance volume considering different error tolerances. For the right model the Hausdorff distance between the reconstructed isosurface and the original one was below 5% of the maximal height of the terrain.

3.3 Error controlled voxelization

In this paragraph we will address the problem of accurately encoding the original polygonal representation in a distance volume. Obviously, the accuracy by which the scene can be reconstructed depends on the geometry, e.g. the curvature of the underlying mesh, and the resolution of the generated distance map.

Although one could think about a theoretical investigation of the relation between the curvature and the voxel size necessary to adequately reconstruct the surface from its voxelized counterpart, this approach seems to be rather cumbersome due to the less intuitive meaning of the curvature and the fact that it is quite difficult to obtain adequate curvature bounds in advance for arbitrary meshes.

A more intuitive and stable algorithm to compute a measure for the maximum deviation of the approximating mesh from the original one is based on the estimation of the one-sided Hausdorff distance between the two meshes. By calculating the maximal distance between vertices on one mesh and triangles of the other one we obtain an intuitive upper bound for the introduced approximation error. Although the calculation of the one-sided Hausdorff distance is numerically intensive it can be easily integrated in our volumetric approach.

At the beginning of the voxelization process we start out with a coarse volume resolution and we compute the signed distance values for each grid point. In order to efficiently perform the calculation of the maximal distance between the original mesh and the newly generated one we exploit the fact that the isosurface passing through the volume can be extracted separately for each cell by a Marching Cubes algorithm [10]. For each vertex on the original mesh all volume cells within the desired maximal deviation are traversed. If the minimal distance between the vertex and the triangles generated by the Marching Cubes algorithm is above the tolerance or if no surface was found the procedure stops and the voxelization is performed on the next finer grid (see Figure 2).

3.4 Multi-resolution representation

In general it might occur that the volume resolution in adjacent boxes will differ because parts of the geometry have been voxelized according to different error tolerances or because regions with high oscillations have been separated from rather smooth ones. In order to guarantee continuous transitions some extra effort has to be made.

For example, in Figure 3 two cells from different resolution levels are supposed to share one boundary face. Although both cells consist of the same values at the corner

vertices the continuity of the scalar field does not in general guarantee the continuity of the isosurface if extraction is performed on different levels. This can clearly be seen from the fact that the isocurve on the common face is approximated by a straight line from the coarser side while it is a broken line with several segments on the finer side. To

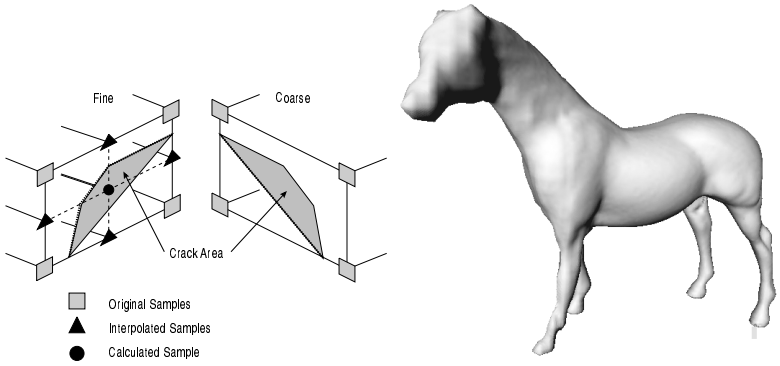

Fig. 3. Left: cracks in the piecewise linear approximation to the isosurface occur at common cell faces where cells from different octree levels meet — even if edge-compatibility is guaranteed. Right: the voxelization process is performed with varying resolution. From the tail to the head the horse has been converted into distance volumes with decreasing resolution. Three different resolution levels were used.

maintain a *continuous* scalar field even if the extraction level changes, the scalar values at those cell faces where a level transition occurs are properly adjusted: Whenever a cell is adjacent to a coarser level cell, the data values on the cell edges are linearly interpolated between the voxel values at the coarser level. The distance function for the midpoint, however, has to be recalculated in order to obtain the same surface on either of both sides. This can easily be achieved by considering the line equation obtained from the intersection with the coarser cell edges and the already interpolated data values.

If we allow for the resolution between adjacent distance volumes to only differ about a factor of two we can always guarantee a continuous transition. A huge amount of memory can be saved by adjusting the volume resolution with respect to the geometry of the data to be voxelized or if a LOD representation is desired (i.e. Figure 3).

4 Rendering surfaces from distance volumes

Once the volumetric model consisting of the distance-to-closest-surface function has been created its use in practical applications still depends on how fast it can be rendered. Since in the voxel data the original mesh is represented by the isosurface at zero-value, for a particular view it is sufficient to re-sample the volume along the lines of sight and to determine the first sample where the isovalue is hit.

4.1 Volume re-sampling via 3D textures

Most efficiently the re-sampling of large scale volume data can be accomplished by exploiting 3D texture mapping hardware that allows for the interpolation of texture samples with about some hundred MOps per second on high-end graphics workstations. Although at this time hardware accelerated 3D texture mapping is only supported on a

few particular architectures we expect the same functionality to be available on low-end architectures like PCs in the near future.

The key idea in volume rendering via 3D texture maps is to re-sample the data on clipping planes that are oriented orthogonal to the viewing plane and to blend the results appropriately [1]. By only slightly modifying this approach it can be employed to render lighted isosurfaces [15]. Each distance volume has to be converted into a RGBα texture, which stores the gradient components and the distance values as the scalar material values in the color channels and the alpha channel, respectively. Note that all components have to be properly scaled and translated to the range (0,1) in order to fit into the internal texture format.

By slicing the texture in front-to-back order and by exploiting the OpenGL alpha-test and the depth-test it is guaranteed that only the first hit with the isosurface is rendered into the frame buffer. Finally, the gradient components and the distance value of the surface points are resident in the pixel values. The diffusely lighted surface points (cf. Figure 4) are obtained by multiplying each pixel value with a $4x4$ matrix. The matrix is initialized in order to transform the gradients back to the range (-1,1), to rotate them with respect to the rotational part of the model view matrix and to perform the calculation of the inner product with the light direction. The matrix multiplication is accomplished by a single per-pixel copy operation with a properly initialized color matrix.

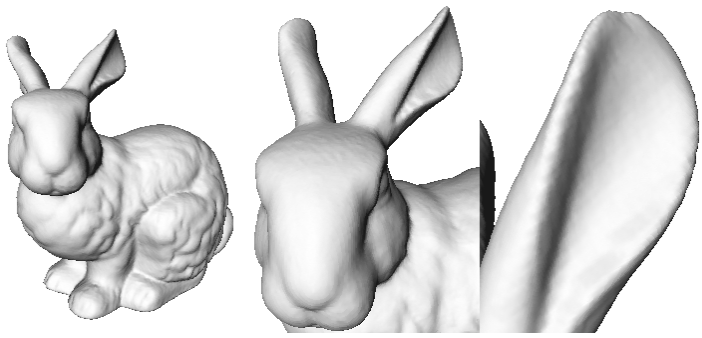

Fig. 4. All images show the isosurface at zero distance rendered with an inter-slice distance of one voxel. Note that the isosurface won't be hit exactly, but that we can arbitrarily decrease the inter-slice distance in order to get more accurate results.

The major benefit of this approach is that any polygonal representation is completely avoided at run-time. Thus the complexity of the rendering process is raster bound and does not depend on the complexity of the surface to be reconstructed. However, there are still a few problems that have to be addressed.

4.2 Rendering thin boundary regions

Although in the proposed framework we are only interested in re-sampling the texture around the isosurface at zero distance a huge number of operations is performed in regions that do not represent the surface. Bounding boxes as introduced already help to make these regions smaller by defining tight bounds around the object, but the texture is also rendered inside the object although the reconstructed samples consist of redundant information.

Consequently, the re-sampling should be restricted to the thin region around the isosurface at zero-distance. Therefore we need to find two approximating meshes for which the minimal deviation from the original one is above a certain tolerance, but one of them completely covers and the other one is completely enclosed by the original model. Then the region between the two meshes exactly covers the relevant texture samples.

Again, from the volumetric representation by means of distance values these two meshes can be reconstructed in a quite efficient way. Let us consider the original mesh to be scan-converted into a signed distance representation at a very coarse resolution. In general, the original surface *cannot* be reconstructed properly due to the insufficient resolution of the underlying grid. Nevertheless, the approximating meshes can be reconstructing from the distance volume according to different isovalues. If the voxel size is supposed to be one, then the two surfaces encoded by distance values 1 and -1 comply with the requirements. We reconstruct these surfaces by means of the Marching Cubes algorithm but we place vertices on the edge midpoints. Thus the surface will most likely to be consisting of many planar triangles that can be collapsed in order to effectively decimate the meshes (see Figure 5).

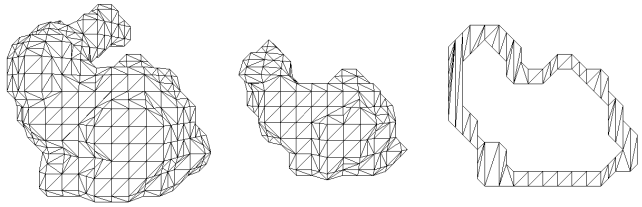

Fig. 5. The illustrations on the left demonstrates the shape of the two approximating meshes for the bunny model. In the middle, for a particular slice the region between the outer mesh and the inner one has been tessellated.

4.3 Clipping and tessellation

Once the two approximating meshes have been constructed the boundary region between them needs to be tessellated. In general, this can be accomplished in two ways: in a preprocessing step by approximating the region by 3D primitives, e.g. cubes or tetrahedra, as in the SGI-Volumizer [13], or in turn during rendering by tessellating the contours that result from clipping both meshes with the slicing planes (see right image in Figure 5).

In our implementation we chose the latter one since we found it attractive to inherit the various techniques already developed for the efficient handling of triangle meshes. We proceed by calculating the sectional polygons with the outer and the inner mesh for each slicing plane. This yield pairs of contours, one inside the other. For the tessellation of the boundary region we utilized the OpenGL tessellation utilities for computing trapezoidal decompositions of concave polygons with multiple holes.

We further exploit an active edge data structure including topological information that allows us to efficiently find those triangles of the approximating meshes that intersect with a certain clipping plane. Initially, once the two meshes have been constructed, to each vertex the appropriate texture coordinate is assigned, which has to be used to texture the newly generated triangles. The intersection procedure thus interpolates vertex positions and texture coordinates.

In case that multiple inner and outer contours are generated or the outer contour is enclosed by the inner one a simple in-out test yields the appropriate pairs to be tessellated. Note that self-intersecting contours cannot occur due to the way they are reconstructed from the distance volume.

With these modifications on hand we are now prepared to considerably decrease the load in the rasterization unit. Only those regions that are close to the isosurface at zero-distance are re-sampled by rendering the appropriate triangles. Concerning the overhead that is spent to generate sectional contours and to tessellate the boundary region we will show in the results section that this doesn't effect the overall performance noticeable. At this point let us just summarize that the use of distance volumes allows us to generate the approximating meshes in a coarse resolution with as few triangles as desired.

4.4 Rendering multiple distance volumes

Although multiple axis aligned distance volumes can be rendered in random order due to the depth comparison of fragments some additional issues have to be considered:
Alignment: When adjacent bounding boxes are not coherently aligned to each other the reconstruction process no longer guarantees continuous transitions. However, if the resolution of each distance volume is constrained to be a multiple of an initial size Δ_{Vox} we can quite easily achieve that cells from adjacent volumes always coincide by just translating and scaling the bounding box appropriately. Now the interpolation method will always yield the same results where bounding boxes overlap or share a common edge. Note that boxes must have an overlap of at least one voxel due to the treatment of texture boundaries by OpenGL. Therefore bounding boxes are appropriately scaled in advance.
Slicing: In order to continuously re-sample the surface at bounding box transitions it is necessary to choose a unique inter-slice distance Δs for the texture based rendering. However, since slices used to re-sample different volumes usually do not coincide we constrain their position in such a way that they are always in a distance $k \cdot \Delta s$, $k \in Z$, from the viewing plane. This strategy allows for the reconstruction of continuous surfaces even if a multi-resolution representation has to be rendered.

We should also mention that for each bounding box the two approximating meshes for the included geometry have to be constructed. In order to apply the proposed tessellation strategy we have to guarantee that a closed surface is maintained at the boundary faces. This is accomplished by clipping the entire mesh with the bounding boxes and by re-tessellating by means of the outlined technique.

4.5 Per-pixel lighting and shading

Hardware accelerated per-pixel shading as proposed in [15] only allows for up to three diffuse light sources. In general, however, in order to simulate realistic lighting and shading effects a more sophisticated lighting model should be provided.

Remember that when the rendering process via 3D textures has been finished the normalized and properly transformed gradients are contained in the pixel components. Instead of performing lighting calculations on a per-vertex or per-fragment basis before fragments are drawn to the frame buffer we propose an OpenGL extension $glShadePixelEXT(x_{min}, y_{min}, x_{max}, y_{max}, mat)$, which enables per-pixel shading thus decoupling the complexity from the scene complexity. A chunk of pixel values is read from the color buffer, then it is multiplied with the matrix *mat* and the resulting values

are interpreted as per-pixel normals for which standard OpenGL lighting is performed. Color values generated in this way are drawn back into the color buffer.

Although this extension is actually not available we simulate it in software. The appropriate pixels are read into main memory. Thereby the color matrix is initialized with *mat* in such a way that the components in main memory represent the correctly rotated gradients in the range (0,1). We now retrieve the current OpenGL state, perform the lighting calculations and write the results back into the color buffer.

Obviously, the software simulation is rather inefficient, but on the other hand we should note that lighting calculations are entirely performed on a per-pixel basis. Thus the complexity does not depend on the complexity of the scene and scales only with the image resolution. The images in Figure 8 outline the results of simulating the Phong lighting model using the proposed algorithm.

5 Results and Analysis

In this section we show results for different models and we analyze some of the main features of our approach. All tests were run on a SGI Onyx2 BaseReality with one R10000, 195 MHz processor, 64 MB texture memory and 256 MB main memory.

Table 1 shows detailed results of the proposed adaptive voxelization technique with respect to memory requirements, rendering complexity and frame rates. The models we compare to each other are illustrated in the top row of Figure 6. The approximation error is given with respect to the longest axis of the models bounding box. The adaptive tessellation technique was not used in these examples.

Table 1: Model characteristics, memory use and timings for different polygonal models

	dino	car	horse
#Triangles	110K	150K	210K
#BBoxes	16	128	16
Preprocessing (min)	1.1	1.6	2.1
Memory use	3MB	6MB	9MB
Hausdorff Distance	$\leq 1\%$	$\leq 2\%$	$\leq 1\%$
Frames per sec	8.8	6.8	5.7
#Rendered triangles	1.0K	1.6K	2.3K

We observe that for all examples the memory use is moderate compared to the amount that is originally occupied by the polygonal representation. For example, storing the horse data set as a triangle mesh without any topological information would require approximately 5 MB. Although in the presented examples our adaptive voxelization strategy roughly needs twice this amount we expect to achieve considerably better results if we further increase the number of bounding boxes. Additionally we have to consider that tighter bounding volumes will obviously result in less operations to be performed in the rasterization unit, but that the CPU will be loaded more heavily in order to clip the slicing planes. In the demonstrated examples, however, this time was negligible compared to the overall rendering times.

Also from Table 1 it can be observed that our approach is still slower than pure polygon rendering. This loss in the rendering performance can be explained by the fact that although we already did considerably reduce the number of operations to be performed in the rasterization unit the capacity of the rasterization hardware still prohibits more optimal frame rates.

The last row of Table 1 should demonstrate the gains of our method in terms of the load in the geometry unit in more detail. Note that only the polygons that result from clipping the slicing planes against the bounding boxes have to rendered. As a consequence only about 1% of the original number of primitives had to be converted into fragments and finally displayed. In this way we effectively decouple the geometry unit from the complexity of the underlying mesh, which we see as one of the most challenging features to be considered in the design of future graphics hardware.

In order to demonstrate the improvements that can be achieved by integrating the proposed adaptive tessellation technique, for the horse data set we constructed the approximating meshes as illustrated in Figure 7. The outer mesh consists of 420 triangles and the inner one of 266 triangles. Rendering was performed using 200 slices. The calculation of intersection points and the tessellation of the sectional contours took approximately 0.1 seconds.

Since re-sampling is effectively restricted to the thin boundary region around the surface we saved about 85% of the rasterization operations. The frame rates were increased accordingly, from 5.7 fps up to 8.8 fps. The additional number of triangles introduced by tessellating the boundary region was 9.3K. Still, this amount is considerably smaller than the original number of triangles to be rendered, but at the same time the rasterization load is dramatically reduced.

Finally, the images in Figure 8 illustrate the results of per-pixel shading simulated in software. For a 500x500 viewport it took roughly 0.3 seconds to read the pixel values, to perform the lighting calculations and to write the shaded pixels back into the color buffer. We should note that even without constructing a bounding box representation the rasterization load for the rendering of the dragon data sets was reduced of about 76% using the tessellation technique.

6 Conclusion

We have presented a general approach that demonstrates how large polygonal models can be rendered from a volumetric representation with significantly reduced geometry transfer and processing. Although there are some deficits of our approach:

- Colored models can only be rendered by a two-pass algorithm, the rendering of surface texture is not yet worked out but may benefit from multi-texture functionality that will be available soon.
- Realistic lighting effect have to be simulated in software including expensive frame buffer access.
- The approach is restricted to static scenes since it requires a great deal of preprocessing.
- Sharp edges can only be modeled by dramatically increasing the resolution of the volumetric representation.

we are convinced that the ideas we presented could be influential for future graphics algorithm and hardware developments:

- We have demonstrated that there are new challenges and applications in which 3D textures can be used effectively. So far, the use of volume textures was more or less exclusively limited to volume rendering applications.
- The proposed adaptive tessellation technique has great influence on volume visualization algorithms since it allows one to reduce rasterization load by focusing on the relevant parts.

- We believe that some of the described limitations can be overcome by even faster access to textures and advanced features like single-pass multi-texturing.
- With dedicated volume graphics boards being announced we expect new relevance for all voxel algorithms and we are investigating ways of performing hardware accelerated voxelization.
- With dedicated graphics boards enabling hardware supported per-pixel shading we believe that our approach might streamline an important future direction leading to graphics architectures that render normals as color values issued on a per-vertex basis and perform the lighting and shading calculations in image space.
- We will consider hybrid approaches where the load is balanced in between the geometry stage and the rasterization stage by rendering LOD representations either as polygonal or as volumetric models. For example, in mesh decimation high resolutional parts with low curvature could be rendered from the volumetric representation while high curvature parts are still rendered as triangles.

References

1. B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *ACM Symposium on Volume Visualization '94*, pages 91–98, 1994.
2. S. Cohen and A. Kaufman. Scan-Conversion Algorithms for Linear and Quadratic Objects. In *1990 Symposium on Volume Visualization*, pages 280–301. IEEE, 1990.
3. S. Gibson. Using Distance Maps for Accurate Surface Reconstruction in Sampled Volumes. In *1998 Symposium on Volume Visualization*, pages 23–30. ACM, 1998.
4. S. Gottschalk, M.C. Lin, and D. Manocha. OBBTree: A Hierarchical Structure for Rapid Interference Detection. *ACM Computer Graphics, Proc. SIGGRAPH '96*, 1996.
5. H. Hoppe. View-Dependant Refinement of Progressive Meshes. *ACM Computer Graphics, Proc. SIGGRAPH '97*, 1997.
6. J. Huang, R. Yagel, V. Filippov, and Y. Kurzion. An Accurate Method for Voxelizing Polygon Meshes. In *1998 Symposium on Volume Visualization*, pages 119–126. ACM, 1998.
7. A. Kaufman. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces and volumes. *ACM Computer Graphics, Proc. SIGGRAPH '87*, 1987.
8. Knittel, G and Straßer, W. A Compact Volume Rendering Accelerator. In Kaufman, A. and Krüger, W., editor, *1994 Symposium on Volume Visualization*, pages 67–74. ACM SIGGRAPH, 1994.
9. P. Lacroute and M Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transform. *Computer Graphics, Proc. SIGGRAPH '94*, 28(4):451–458, 1994.
10. W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *ACM Computer Graphics, Proc. SIGGRAPH '87*, 21(4):163–169, 1987.
11. J. Montrym, D. Baum, D. Dignam, and C. Migdal. Infinite Reality: A Real-Time Graphics System. *Computer Graphics, Proc. SIGGRAPH '97*, pages 293–303, July 1997.
12. H. Pfister and A. Kaufman. Cube-4 - A Scalable Architecture for Real-Time Volume Rendering. In R. Crawfis and Ch. Hansen, editors, *1996 Symposium on Volume Visualization*, pages 47–54. ACM SIGGRAPH, 1996.
13. IRIS InSight Silicon Graphics Publication. *OpenGL Volumizer Programmer's Guide*. 1997.
14. S. Wang and A. Kaufman. Volume sampled voxelization of geometric primitives. In *Visualization 1993*, pages 78–84. IEEE, 1993.
15. R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. *ACM Computer Graphics, Proc. SIGGRAPH '98*, 1998.

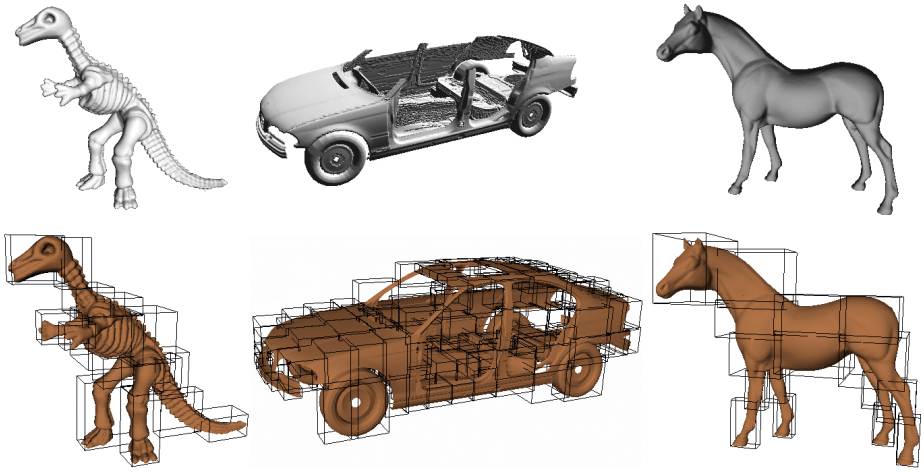

Fig. 6. Different polygonal models have been converted into sets of distance volumes (bottom row) and rendered via 3D textures (top row).

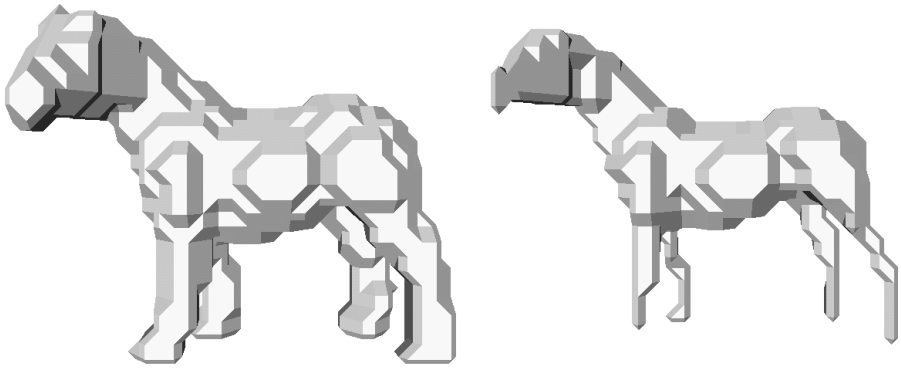

Fig. 7. The outer and the inner mesh used to render the thin boundary region around the surface of the voxelized horse data set.

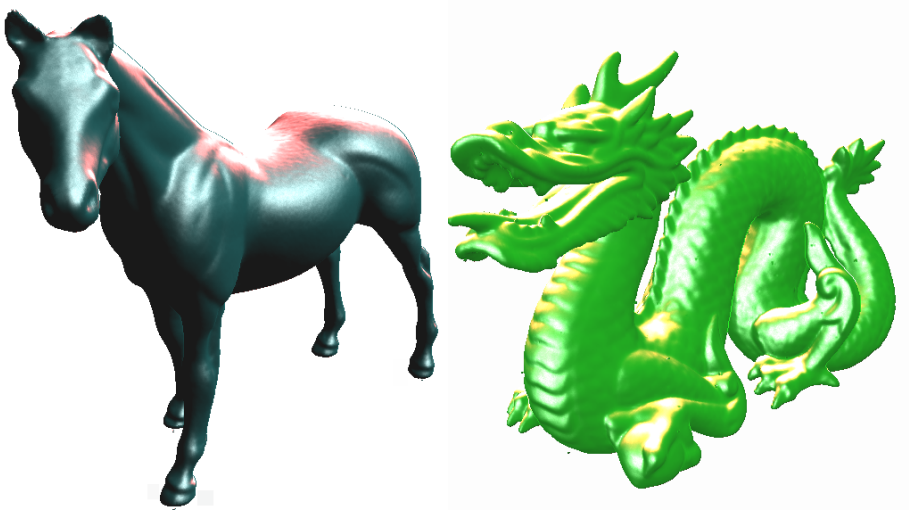

Fig. 8. The shading of voxelized models on a per-pixel basis using Phong's illumination model.