Octree Textures

David Benson *Joel Davis*

Industrial Light and Magic
San Rafael, CA 94912

Abstract

Texturing using a set of two dimensional image maps is an established and widespread practice. However, it has many limitations. Parameterizing a model in texture space can be very difficult, particularly with representations such as implicit surfaces, subdivision surfaces, and very dense or detailed polygonal meshes. This paper proposes the use of a new kind of texture based on an octree, which needs no parameterization other than the surface itself, and yet has similar storage requirements to 2D maps. In addition, it offers adaptive detail, regular sampling over the surface, and continuity across surface boundaries. The paper addresses texture creation, painting, storage, processing, and rendering with octree textures.

CR Categories and Subject Descriptors: I.3.m [Computer Graphics]: Texturing; **Additional Key Words:** Volume Texture

1 Introduction

Traditionally, texturing a model with image data requires setting up a 2D parameterization of a model. This parameterization is typically set up manually by an artist. This step can be automated with techniques such as the work presented by Maillot [14] and recent work by Piponi [19]. These approaches still require that dicontinuous cuts be introduced in the mapping.

In addition, even after a parameterization is constructed, some texture distortion, or "stretching" may be caused by the parameterization. Techniques such as [21] can help minimize this stretching on polygonal meshes. Another problem arises when there is the need to have a small area of a large model, such as a decal, have much higher detail than the rest. In some cases, this can result in parts of the model needing to be split to accommodate these local area of high texture detail.

Furthermore, these 2D approaches may not apply to geometric representations which are difficult to assign a parameterization to, such as implicit surfaces [3] or Adaptive Distance Fields [8].

This paper was motivated by these issues, and presents a solution in a new form of texture as well as details of integrating these textures into a production environment.

2 Related Work

2.1 Previous Work

Texture mapping was first introduced by Catmull [4]. This concept was extended in 1985 with the introduction of the concept of solid textures, presented independently by Perlin [17] and Peachey [16]. They realized that using a surface's position in 3D space as texture coordinates allowed complex surfaces to be texture mapped. This work was mainly concerned with procedural textures extending throughout a volume enclosing the surfaces being rendered. Peachy's work [16] also suggested the idea of digitized solid textures by scanning slices of a model. He points out that for a regular voxel grid of samples, high resolution textures would require huge amounts of memory. This form of solid texture is a volume texture. Akeley [1] described a hardware implementation. They have also become part of the OpenGL [22] graphics standard. However, their main use has been volume rendering [7].

One piece of related work in volume rendering was presented in 1991 by Laur et al. [12]. In this approach, a pyramid of average colors is formed from a traditional volume texture, and then an octree is constructed that approximates the original data to a given accuracy. This results in a similar data structure to work presented in this paper, but it is used as a volume rendering optimization.

Octree based images have also been considered in the related subject of 3D Image Processing, and work by Tamminen in 1984 [24], introduces this connection.

We have recently become aware of work by DeBry et al [6], who have independently developed a similar approach to texture mapping using octrees.

2.2 Other Approaches

Turk developed a method [25] for using a regular distribution of samples spread across a surface to represent a Reaction-Diffusion texture. He suggested choosing random samples over a surface, which were then relaxed to get a more even distribution. Texture look-up was accomplished by a weighted sum of nearby samples. This contrasts with the more structured texture presented in this paper, in which the sample positions are implicit in its location in the octree. Recent work by Pfister et al. [18], extends this concept to represent the surface itself at the sample points and replaces the random sample points by a more uniform set, sampled on a grid pattern (a layered depth cube representation [13]), which makes it more closely related to the work in this paper, but focuses on using the structure as a representation of the geometry for fast rendering.

Recent work by Frisken [8] presented the concept of Adaptive Distance Fields for modeling surfaces. In this work, an octree of scalar values is used to represent the distance of each sample point from a surface. This structure is similar in some ways to the octree texture presented in this paper, but the structure and interpolation scheme are quite different.

Another approach to texture mapping polygons is used extensively

in research on mesh simplification [5], [15], [23], [20]. In this approach a texture tile is created for each face of a simplified mesh. These tiles need to be padded with boundary information from their neighbors to ensure continuity. The tiles are then packed into regular textures.

The drawbacks in this form of texture mapping are that the sampling is once again very uneven, and the discontinuous structure of resulting texture maps prevents them from being turned into mipmaps. In addition these textures cannot be completely continuous, as the technique of padding the boundaries only approximates the real neighboring values.

Gortler et al. [9] presented a method for reconstructing an image from one dimensional line samples. This method consisted of constructing a lower resolution image, and then up-sampling and combining with higher resolution samples to fill in detail. This process has some similarities to the reconstruction phase presented here.

3 Octree Texture Structure and Usage

3.1 Conceptual Overview

Solid texture mapping has been around for some time, and relies on the rest position of a model as the texture coordinates. These coordinates can be created automatically, in contrast to the lengthy manual process of assigning two dimensional coordinates to surfaces which are rarely easy to parameterize. While solid textures are usually procedural, they can also come from a traditional volume texture.

This paper proposes the use of octree textures, a variation on the volume texture idea in which only the subset of the volume that actually intersects the surface of a model is stored. The textures are sparse because only octree cells which intersect an object's surface are stored. The octree structure provides an efficient way to store and look up these textures.

Traditional volume textures are a regular 3D lattice of color samples. Their memory usage grows with the cube of the resolution. This has made them impractical for general use and confined their use to primarily medical and scientific imaging [10].

However, solid textures have other nice properties. For example, at any particular resolution their samples are uniformly distributed across space in the region the surface occupies. In addition, these coordinates are continuous across the boundaries between different geometry types, and independent of the actual texture assignments.

In the visual effects industry, we are mainly concerned with colors on these surfaces, which represent only a two dimensional subset of the volume. These textures concentrate all their detail at the surface itself, and at the limit, behave like two dimensional textures. In other words, at the limit, their size grows with the square of the resolution. This means that they are capable of achieving the high resolutions required in the visual effects industry without becoming unmanageably large.

3.2 Storage

These textures may be potentially very large, so the representation must be compact. We chose to represent a node by a set of flags indicating which of the node's children are present, and which are leaves. Then, we store an array of pointers to child nodes, and another array containing the values of the child nodes that are leaves (Figure 1).

Since most nodes of the tree will typically be leaf nodes, and since near the surface an average of half of a node's children will be leaves and contain a color sample, this greatly cuts down on storage required by pointers to child nodes.

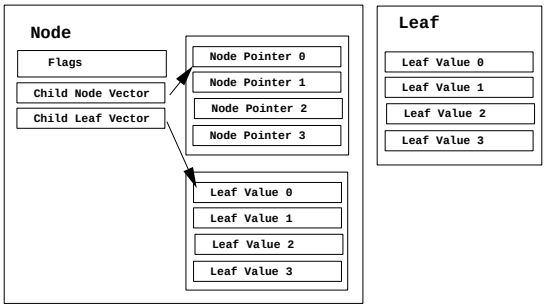

Figure 1 A compact representation of the texture octree. Child node pointers and leaf data are stored in two contiguous arrays to minimize pointer overhead.

3.3 Texture Coordinates

In most cases, the texture coordinates will simply be the surface position. But in some cases, it is convienent to use another parameterization. For example, when using subdivision surfaces, the coordinates on the control hull may be used rather than coordinates on the limit surface. In this way, a polygonal approximation of the limit surface will be textured consistently at any level of subdivision. This introduces a small amount of texture stretching.

For a deforming animated model, the coordinates of the model's rest pose may be used as texture coordinates to attach the texture to the animating model.

3.4 Interpolation

To look up a sample and maintain a continous field of color, trilinear or tri-cubic interpolation is used.

With an octree texture, the resolution varies across the texture, so we have the additional problem of how to interpolate between regions at different levels of detail. Our approach is based on the ideas developed for generating smooth textures from quadtree images presented by Kobbelt et al [11]. Their approach uses a subdivision strategy which could be repeatedly applied to lower resolution areas until the whole image was at the same high resolution and could then be interpolated in the usual way.

To look up the texture at a sample point, a small neighborhood is followed down from the octree root to the leaves. Our implementation uses a 3x3x3 neighborhood for tri-linear interpolation. A 5x5x5 neighborhood could be used for tri-cubic interpolation.

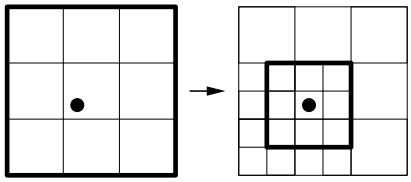

Figure 2 A fixed neighborhood around a sample point is subdivided to the next higher resolution and filled or interpolated from the octree values.

Figure 2 demonstrates how this process works. On the left is the neighborhood at depth n, and on the right the neighborhood (outlined in black) at depth (n+1). The black dot represents that sample

point, and at each stage it lies at the center of the neighborhood. The higher resolution neighborhood (right) is constructed from the lower resolution one (left) by subdividing the cells closest to the sample point. Then the sub-cells adjacent to the sample point are gathered together. As none of the cells in the new neighborhood can be touching the boundary of the old one, the old neighborhood contains all the information necessary to create the new one.

The following steps give the rule for subdividing a cell:

1. If the cell contains empty space, then all children are empty.
2. Otherwise, if the cell contains a non-leaf node, then the children are the node's children.
3. Otherwise, if the cell contains a leaf or a subdivided leaf, then form the children from a weighted sum of the neighboring cell's colors. Extrapolate for node neighbors which do not have a color.

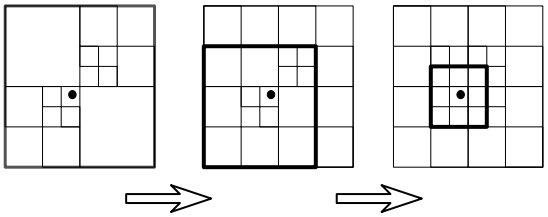

Figure 3 Texture interpolation between the different resolutions of a sample point's surrounding cells is achieved by following a 3x3x3 neighborhood from the octree root.

The neighborhood is initially set to be empty, except for the central cell, which points to the entire octree. As we progress down the tree, the region of the octree represented by the neighborhood converges towards the sample point as shown in Figure 3. This process is continued until no cell in the neighborhood contains an octree node, i.e. we stop when every cell contains a leaf, a subdivided leaf, or empty space. The neighboring colors are then tri-linearly interpolated to give the result.

This process results in a continuous interpolation of the sample points across regions with varying level of detail. However, this algorithm is a point sampling of the texture, and it may be necessary to average a number of these samples to obtain a filtered texture look-up.

3.5 Texture Filtering

When a textured object is drawn at a distance, some kind of minification filter is desired to prevent aliasing. The ideal filter is to take the average of many visible texels contained in the area that is being sampled.

With traditional 2D textures, it is common to use mipmaps [27] or summed area tables to effectively do some of this filtering ahead of time as a preprocessing step. Mipmaps also help reduce the amount of the texture that needs to be accessed when the surface is far from the camera, and small in the view, because only the lower resolution version of the texture need be accessed.

Traditionally this average is simply made over a rectangle in a 2D texture's parameter space. Making the average in the texture's space may include areas of the texture that are not visible from a distance, and may even include regions that are not assigned to any part of the model.

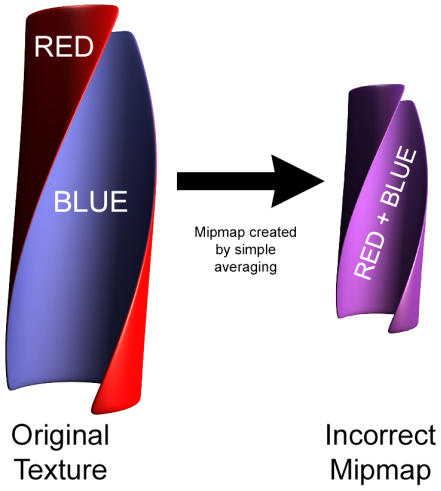

Figure 4 On a model with two surfaces of different color nearby but facing different directions–such as a thin sheet with contrasting colors on each side–simply averaging the texture values can fail and cause colors to bleed or mix.

3.5.1 Problems With Extending 2D Mipmapping

The octree texture has a structure similar to a mipmap and it is natural to consider following the same procedure to build a 3D mipmap. This idea was explored for volume rendering in [12], but those results are not directly applicable to surface rendering. Replacing a neighborhood of voxels in a texture by a single larger voxel is in some cases a poor approximation to the ideal average mentioned above, because this average may contain colors from neighboring surfaces which are not visible at the higher resolution level (Figure 4). Because of this, colors can bleed from nearby. For example, if the back and front of a thin model are different colors, the resulting average color represents a color that should never be seen. Color may also bleed between unconnected, but neighboring, close surfaces.

3.5.2 Normal Flags

In most cases, these problems can be solved manually by assigning the conflicting parts of the model to separate octree textures. However, in order to handle a broader range of situations automatically, the octree nodes can be extended by adding normal flags for rendering.

Our goal is to preserve the behavior that we get interpolating the highest levels of the texture. We maintian a set of six normal flags with each node, one for each direction of each axis. A flag is set for any direction of a node which contains a surface with a normal closer to the flag's direction than to any other. If any axis has both direction flags set then the minification is considered no longer valid (Figure 5), and a look-up is forced to continue to a higher resolution node. As an optimization, this rule may be ignored if the samples below the node are within some user-specified threshold of each other, and can thus be averaged without producing any visible color bleeding.

3.6 File Format Considerations for Rendering

In order to allow quick access to a lower resolution version of the texture, the tree must be stored in breadth first ordering, not depth

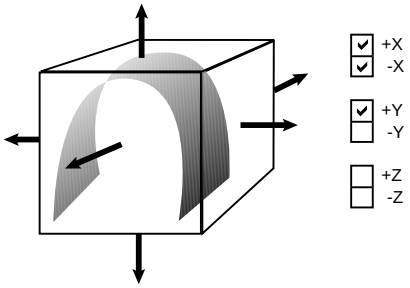

Figure 5 For rendering, every octree node contains a normal flag for each direction of each axis. If the normal flags for both directions of any axis are set, mipmap look-up is forced to continue to the node's children.

first. Unfortunately this requires one or two pointers to be stored for each node, pointing to the child leaf and child node positions in the file.

Fortunately, for any section of nodes on the same level of the hierarchy the pointers will be consecutive, and can be recreated from the node's flags given a starting position for the children and leaves. This means that they can be stored in a file in a compressed form with few pointers, and expanded back up to the two pointer form when reading. In our implementation, the octree was also tiled to allow only a subset of the actual texture to be held in memory.

4 Painting

An octree texture may be generated procedurally, but in this case, one can just use the procedural model directly to texture the object. In most cases, an artist will be asked to paint the details onto the models, or apply scanned photographs to the model. We must be able to apply this technique to an octree texture.

4.1 Overview

The simplest form of 3D paint programs involve painting on a projection of a model, and then projecting the paint back onto the model. In the two dimensional case, the surface geometry is subdivided to the resolution of the 2D textures and each of the resulting micropolygons projected back onto the painted image to get a color for the corresponding texel [2]. This concept extends naturally to three dimensions for octree textures. In this case we need to rasterize the rest geometry into the octree, again keeping the real surface position as a texture coordinate for the look up from the painted image.

If the octree is kept at a fixed depth, painting is greatly simplified, as the hierarchy of the new paint and the existing paint will be identical. However, for an adaptive octree, the hierarchy can differ. In practice, it is convenient to first create a "new paint" octree and then merge that with the existing octree, rather than try to combine the paint and rearrange the octree in one step.

This two step process also ensures that blending the new paint into the existing texture occurs only once for each voxel when the new paint has an alpha channel.

The steps involved in mapping paint back onto an octree texture are as follows:

1. Rasterize surfaces into the paint octree, using the surface's rest coordinates as its location and its projected position as its texture coordinate to look-up a color from the painted image.
2. Where the existing octree resolution is higher than the new paint's and the paint's alpha is between zero and one, increase the paint octree's resolution to match the texture.
3. Form a boundary layer of empty voxels around the paint, at the same resolution as the neighboring paint. The intensities of the boundary don't matter, only its effect on the next step.
4. Increase the resolution in the main octree anywhere where the paint octree's resolution is higher.
5. Merge the paint octree into the main octree, ignoring the temporary boundary layer voxels.
6. Delete the paint octree.
7. Cull any voxels in the main octree that are not intersected by any surface.

4.2 Merging Existing Paint

The approach presented by [2] for multiresolution painting can be applied to 3D. If an artist paints a transparent wash over some existing detail, their intention is clearly to modify the color of the region. If we simply merged in the wash (which is probably at a lower resolution), the detail would be lost in the process. In order to preserve it, the resolution of the wash must be increased to match the existing detail.

Another situation occurs when an artist paints some fine detail over an area which was previously covered by some smoothly interpolated low resolution paint. In this case, the resolution of the existing low resolution paint must be increased to match the new detail.

In order to increase the resolution of an octree texture, we need to take into account the interpolation scheme being used. If we simply split each leaf into children with the same value, the boundaries of the original voxel will become visible as discontinuities in the resulting texture. This is analogous to performing the corresponding operation on a 2D texture, where the new pixel values would be formed by interpolating the lower resolution values. In addition, the effect of an edit on neighboring voxels must also be taken into account. Ideally, when the resolution of a voxel is increased, the neighboring regions should remain unchanged. This can be achieved by keeping the old low resolution leaf around, and using it during the lower resolution steps of the reconstruction process. These resolution increases are calculated and applied locally, and not to the texture as a whole.

4.3 Considerations for Painting

4.3.1 Color Bleeding

If the model contains intersecting surfaces, they must be separated out into their own octree textures, or a special version of the rest model needs to be constructed in which those parts have been moved apart. If these steps are not taken, the intersecting regions will share a common paint color. This is mainly a problem if these regions are animating, and these intersection regions become exposed.

4.3.2 File Format for Painting

The sparse octree can be stored very efficiently in a consistent depth first order. This method is outlined in work by M. Samet [24], and the authors show that even further compression is possible.

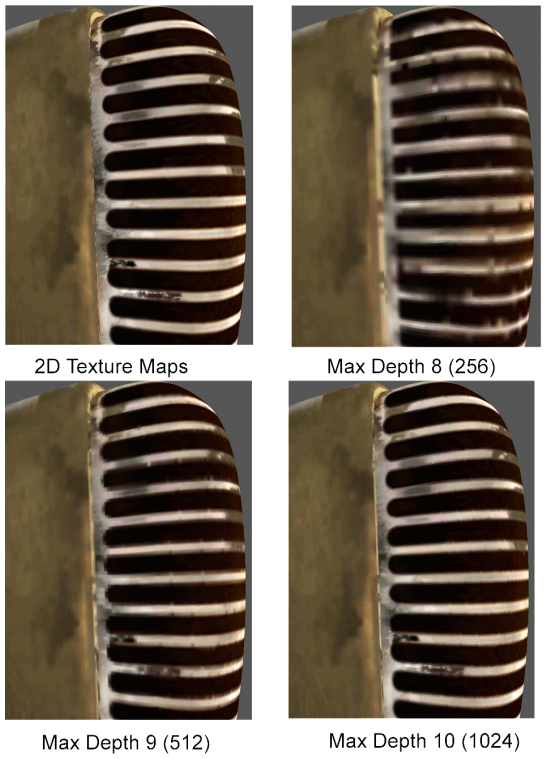

Figure 6 Comparision of octree textures at several resolutions to a set of 2D texture maps.

When rendering, the texture should be converted to a file format like the one suggested in Section 3.6 to perform look-ups efficiently.

5 Results

Our initial work involved converting a model that was already painted with a set of traditional textures. The textures were converted into octree texture without any loss of visual quality or artifacts. The octree texture took less disk space than the large set of 2D images it was generated from.

For comparison, a part of a traditionally textured version of the droid model in Figure 8 is compared to an octree textured version of the same model. This part of the model was painted with five maps: two of size 256x1024, one 1024x512 and one 128x1024. The assignments were typical of the way maps are assigned.

Max Depth	Uncompressed Size
8 (256)	190,189
9 (512)	740,613
10 (1024)	2,901,789
2D Version	3,534,944

The maps were then converted to octree texture at different fixed maximum depths. In this case, the octree texture were not adaptive, so areas of similar color were sampled at the same rate as detailed areas.

When the octree texture is at the same resolution as the largest dimension of the texture maps, the resulting file can be smaller than the 2D texture maps. The octree texture stores samples only where geometry is present, whereas the traditionally mapped version must oversample some areas in order to get other areas sampled at the desired resolution.

Of course, careful assignment of texture coordinates, or methods to relax the the texture coordinates such as presented by [14] can offset this small difference. Additionally, image compression techniques are much more well-studied for 2D images. What is important to note is that octree textures have similar storage characteristics to sets of 2D textures as the texture resolution increases.

To integrate the new textures into our production pipeline, we first tested it on a small prop (Figure 7).

Currently, octree textures are used on characters and props that need to be set up quickly, or are difficult to assign texture coordinates to. Models that use a mix of surface representations also are more straightforward to texture using octree textures. The droid model shown in Figure 8 is an example of a model which was painted using octree textures for these reasons.

Octree textures have also allowed us to experiment with other types of surface representations, such as Adaptive Distance Fields and Subdivision Surfaces without worrying about how or even if they can be parameterized for 2D texture maps.

6 Known Problems and Future Work

6.1 Painting in 2D

The most immediate drawback to using octree textures is that one loses the ability to paint on the model in texture space, that is, to paint directly on the maps. An approach similar to that which Wei and Levoy [26] used to generate a local parameterization for texture synthesis could be used to locally unwrap a region of the texture to be painted on.

6.2 Very Close or Intersecting Surfaces

Octree textures do not handle surfaces in the rest model that are intersecting or very close, such as skin and clothing. The normal flags mentioned in section 3.5.2 help to prevent the minification artifacts that this can cause, but care must be taken when setting up a model to either assign different octree textures to intersecting or very close surfaces, or to create a special version of the rest model where these surfaces are moved away from each other.

Figure 7 The octree textures were first tested on a minor prop (the character's necklace) to ensure that they would work within our production pipeline. (Fig. Joshua LeBeau and Scott Bonnenfant)

Figure 8 Several versions of this droid model were painted using octree textures. (Fig. Bridget Goodman)

6.3 Interactive Viewing

Currently, no hardware supports 3D textures as anything other than a regular 3D grid. Until this, or some similar compressed form of texture can be supported in hardware, the texture must be approximated by a set of automatically generated 2D textures if it is to be viewed interactively.

Also, better techniques to automatically generate such sets of textures from an octree texture efficiently, and at high quality, would help make it easier to manipulate octree textures with existing tools.

6.4 Adaptive Resolution Limits

With the ability to add detail at any scale, an octree texture can grow to be much larger than is needed to represent a particular set of texture. Our solution is to simply allow the artist to adjust the maximum resolution they wish to paint with at any given time. However, image processing techniques could be used to adaptively discover the optimal resolution of any new paint without manual intervention.

7 Summary and Conclusion

We have described a new approach to storing textures for geometric models by using a sparse octree to store texture information, typically color. The approach is useful independent of the surface representation and does not need any parameterization other than the surface coordinates themselves.

Furthermore, we present some approaches to using these textures in a production environment and extend traditional 2D sampling techniques to render these textures with fewer visual artifacts. We also provide a technique to create the textures by painting onto projections of geometry.

Our experience using these types of textures in a production environment shows them to be fast and robust, and similar or smaller in disk space to a set of traditional 2D textures at the same resolution. In addition, there can be a tremendous savings in artist time because texture coordinates do not need to be assigned manually for each model.

References

[1] K. Akeley. Realityengine graphics. *Proceedings of SIGGRAPH 93*, 1993.

[2] D. Berman, J. Bartell, and D. Salesin. Multiresolution painting and compositing. *Proceedings of SIGGRAPH 94*, 1994.

[3] J. Bloomenthal. Polygonization of implicit surfaces. *Proceedings of SIGGRAPH 88*, 1988.

[4] E. E. Catmull. A subdivision algorithm for computer display of curved surfaces. Master's thesis, University of Utah, December 1974.

[5] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. A general method for preserving attribute values on simplified meshes. *IEEE Visualization 1998 Proceedings*, pages 59–66, 1998.

[6] D. DeBry, J. Gibbs, D. Petty, and N. Robins. Painting and rendering textures on unparameterized models. *Proceedings of SIGGRAPH 02*, 2002.

[7] R. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Proceedings of SIGGRAPH 88*, 1988.

[8] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. *Proceedings of SIGGRAPH 00*, 2000.

[9] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumigraph. *Proceedings of SIGGRAPH 96*, pages 43–54, 1996.

[10] J. T. Kajiya and B. P. Von Herzen. Ray tracing volume densities. *Proceedings of SIGGRAPH 84*, 1984.

[11] L. Kobbelt, M. Stamminger, and H. P. Seidel. Using subdivision on hierarchical data to reconstruct radiosity distribution. *Proceedings of EUROGRAPHICS 97*, 1997.

[12] D. Laur and P. Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. *Proceedings of SIGGRAPH 91*, 1991.

[13] D. Lischinski and A. Rappoport. Image-based rendering for non-diffuse synthetic scenes. *Rendering Techniques 98*, pages 165–174, 1998.

[14] J Maillot, Y. Hussein, and A. Verroust. Interactive texture mapping. *Proceedings of SIGGRAPH 93*, 1993.

[15] M. Maruya. Generating texture map from object-surface texture data. *Proceedings of EUROGRAPHICS 95*, pages 397–405, 1995.

[16] D. R. Peachy. Solid texturing of complex surfaces. *Proceedings of SIGGRAPH 85*, 1985.

[17] K. Perlin. An image synthesizer. *Proceedings of SIGGRAPH 85*, 1985.

[18] H. Pfister, M. Zwicker, J. VanBaar, and M. Gross. Surfels:surface elements as rendering primitives. *Proceedings of SIGGRAPH 00*, 2000.

[19] Dan Piponi and George Borshukov. Texture mapping of subdivision surfaces by model pelting and texture blending. *Proceedings of SIGGRAPH 00*, 2000.

[20] P. Sander, X. Gu, S. Gortler, and H. Hoppe. Silhouette clipping. *Proceedings of SIGGRAPH 00*, pages 327–334, 2000.

[21] P. Sander, J. Snyder, S. Gortler, and H. Hoppe. Texturing progressive meshes. *Proceedings of SIGGRAPH 01*, 2001.

[22] M. Segal and K. Akeley. The opengl graphics system: A specification. *http://www.opengl.org/Documentations/Specs.html*, 1997.

[23] M. Soucy, G. Godin, and M. Rioux. A texture mapping approach for the compression of colored 3d triangulations. *The Visual Computer*, pages 503–514, 1986.

[24] M. Tamminen and H. Samet. Efficient octree conversion by connectivity labeling. *Proceedings of SIGGRAPH 84*, 1984.

[25] G. Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *Proceedings of SIGGRAPH 91*, 1991.

[26] Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. *Proceedings of SIGGRAPH 01*, 2001.

[27] L. Williams. Pyramidal parametrics. *Proceedings of SIGGRAPH 83*, 1983.