Painting and Rendering Textures on Unparameterized Models

David (grue) DeBry Jonathan Gibbs Devorah DeLeon Petty Nate Robins

Thrown Clear Productions*

Abstract

This paper presents a solution for texture mapping unparameterized models. The quality of a texture on a model is often limited by the model's parameterization into a 2D texture space. For models with complex topologies or complex distributions of structural detail, finding this parameterization can be very difficult and usually must be performed manually through a slow iterative process between the modeler and texture painter. This is especially true of models which carry no natural parameterizations, such as subdivision surfaces or models acquired from 3D scanners. Instead, we remove the 2D parameterization and store the texture in 3D space as a sparse, adaptive octree. Because no parameterization is necessary, textures can be painted on any surface that can be rendered. No mappings between disparate topologies are used, so texture artifacts such as seams and stretching do not exist. Because this method is adaptive, detail is created in the map only where required by the texture painter, conserving memory usage.

Keywords: Texture Mapping, Paint Systems, Rendering Systems, Spatial Data Structures, Level of Detail Algorithms

1 Introduction

With the advent of 3D paint programs, artists were able to paint texture detail directly onto the surface of models. However, 3D paint is limited by the underlying parameterization between the model's geometry and 2D texture space. Poor parameterizations result in unacceptable artifacts such as texture distortion, discontinuities, and singularities.

Geometric detail does not necessarily correspond to texture detail; in fact, the opposite is often true. As a result, it becomes an iterative process to adjust the model's parameterization to the required texture detail, and fix existing texture data to align with new parameterizations. This process can require large amounts of time from both the modeler and texture painter, and is often a bottleneck at CG animation production companies. As a production progresses, camera position and extreme animation poses often reveal areas on the model which require more detail in the texture. This requires the parameterization to be adjusted, the resolution of the map increased, or both.

Our goal is to create a texturing method that fulfills the following criteria:

- No parameterization should be necessary. A model should inherently be paintable.
- Discontinuities should not exist. The topology of the texture should match the topology of the model.
- Increased texture detail should require only a localized increase in map resolution. The texture should contain exactly as much detail as has been painted.
- Memory usage should be similar to existing 2D textures.

After discussing related work, we show that storing texture maps in an octree fulfills these requirements. We then discuss painting textures, overcoming difficulties with some geometry, using these textures in a renderer and memory usage. Finally, we show examples of our method and present ideas for future work.

2 Related Work

Traditional texture mapping applies a 2D image to 3D geometry to represent visual features without requiring an increase in the underlying geometry [Blinn and Newell 1976]. At render time, these texture coordinates are interpolated across the surface and then used to sample the 2D texture.

The most critical part of this process is arriving at a mapping from the 3D geometry to the 2D texture space. Manual wrapping methods include using a set of simple projections or unwrapping the model into a flat sheet (figure 1). Significant work has been done recently on arriving at a mapping automatically [Boada et al. 2001; Haker et al. 2000; Igarshi and Cosgrove 2001; Lévy 2001; Ma and Lin 1988; Malliot et al. 1993; Pedersen 1996; Piponi and Borshukov 2000; Sander et al. 2001]. Optimization methods are somewhat successful [Hunter and Cohen 2000; Maillot et al. 1993, Bennis et al. 1991], but all have significant limitations. Since texture detail has not yet been applied at optimization time, the assumption is made that geometric detail will correspond to textural detail. This issue can be avoided by reparameterizing a model once some texture is applied [Sloan et al. 1998] or by allowing the user to place texture or paint strokes interactively and then derive the appropriate mapping function. Unfortunately, all parameterization techniques introduce discontinuitues, stretching, and other artifacts due to the problems of mapping between disparate topologies.

A 3D volumetric texture can be used to avoid the parameterization problem [Peachey 1985; Perlin 1985]. However, the memory needs for a complete 3D array of texture data can become prohibitive when a large amount of surface detail is required. For a surface texture, most of the space in the 3D texture is unused. Volume textures are almost exclusively generated procedurally at render time because of the high memory requirements.

The uniform resolution of image maps can be avoided by using quadtrees to allocate memory as needed for localized areas of detail [Berman et al. 1994]. Various space partitioning schemes are also used to represent volumes [Frisken et al. 2000; Westermann and Ertl 1997; Wilhelms and Van Gelder 1994]. In particular, octrees can store 3D texture data for volume rendering [Boada et al. 2001] or can be used as an alternate way to represent surfaces, where the geometric detail drives the depth of the tree [Tamminen and Samet 1984]. A 3D texturing system similar to this work but developed independently also uses octrees to store texture data [Benson and Davis 2002].

*{grue,jono,devorah,nate}@thrownclear.com

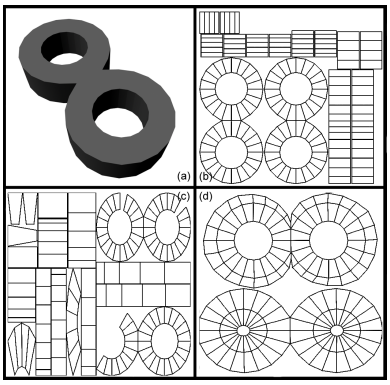

Figure 1: Several methods of parameterizing an object (a). Automatic unwrapping yields (b), iterative packing and relaxing results in (c), and (d) was generated by defining the parameterization by hand.

A 3D paint application allows the user to paint a model using all the standard painting and image manipulation tools. The first 3D paint systems stored the painted color in the vertices of the model [Hanrahan and Haeberli 1990]. Current 3D painting systems put the painted colors directly into the 2D texture map using texture coordinates which already exist in the model [Daily and Kiss 1995].

3 Octex: Octree as a Texture Map

Instead of storing the texture data in a 2D image, we store it in a sparse, adaptive octree [Samat 1990]. An octree is a spatial partitioning tree where each node is divided evenly into eight children. An octree consisting of a single node is constructed around the model before it is transformed or deformed, and a location in 3-space is used as the index into the octree. Thus, the model's texture coordinates are the same as its vertex coordinates in the model's local space. Since there are no discontinuities or singularities in 3-space, this mapping will always be as smooth and continuous as the model itself. With the addition of data described in the following sections, we call this structure an *octexture*, or *octex* for short.

The detail in the tree is driven by the detail in the texture. Of the eight children a given node can have, only those which contain a portion of the model's surface are ever created. These are called the node's *potential children*. Depending on the details in the texture, only some of the potential children will actually be created for a given node.

Each leaf node of the octex contains a color sample for every texture stored in the tree. A typical model painted for high-end production work will have at least three and as many as ten or more textures painted for it. The model will usually need at least a diffuse color texture, a specular color texture, and a bump map or a displacement map. The textures are often painted to a similar level of detail, therefore storing them all in the same tree significantly reduces the total storage overhead. Textures not having some correspondence in detail or coverage (i.e., a decal) can be broken out into their own tree.

A color sample is also stored in parent nodes, but only if the parent node does not have all its potential children. This sample represents the color for all the potential children that have not been created. When all the potential children exist, they completely define the color of the surface in that node and no color sample is stored in the parent node (figure 2a). For texture filtering during rendering, an additional color can be stored in all parent nodes. This color represents the average of all the potential children's colors. It is important to consider the non-existing children as well as the existing children. Because of this filtered color, the octex now contains the 3D equivalent of a traditional MIP map [Williams 1983]. When the renderer provides sample area information to the octex look-up functions, the filtered texture data can be used when the sample area covers several octex nodes. The use of this data is addressed further in section 6.

Consider a model that is initially all white and represented by a single white node. If a small red dot is painted somewhere on the model, the original white node is given one child which corresponds to the part of the model where the dot was painted. Nodes are recursively subdivided until the size of the nodes receiving color approximately matches the size of the red dot. These leaf nodes are colored red. The parents of that node are outside the red dot, so the rest of the model is still white. Figure 2b shows a 2D version of this octex after the red dot is added.

4 Painting

3D painting is a natural interface for creating octextures because detail can easily be added as painting occurs. As the model is painted, we obtain the 3D locations of the paint using the z- buffer and inverse projection matrix. These positions are transformed into the model's local space, and hence the octex space, using the inverse viewing matrix. We then iterate over the leaf nodes affected by the new paint.

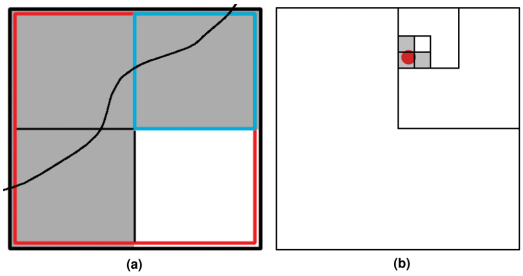

Figure 2: 2D example node (a) which has three potential children (shaded area) due to where the surface passes through the node. Only one of the potential children exists (blue), where paint was applied. The color stored in this node (red) is the color for the two non-existing potential children. Octex subdivision (b) when a small red dot is painted into a coarse white octex. The shaded areas are leaf nodes.

To assure that only the visible surfaces receive paint, in addition to the usual clipping and front facing tests, the surface must be transformed to screen space, and compared against the depth buffer of the rendered model.

When the screen space covered by a given leaf node has an area larger than a pixel, and the new paint in that area is not all the same value, then each of the node's children that contain part of the surface are created. The new children inherit the parent node's color, and are iterated. Finally, the paint color is composited over the color already stored in the leaf node. Because the octex is subdivided to match the resolution of the painting, detail is only created in the map where it is painted (figure 3).

After the paint is applied, nodes are culled if their information is redundant. When a child is a leaf node and contains the same color data as the parent, the child is removed. Thus, the octex can also shrink in size in a similar manner if detail is removed.

The nature of this method allows for as much detail as the painter can create, making it easy for too much detail to be created. Setting a limit on the depth of the octree or the total number of nodes can prevent painters from unwittingly creating an overly large texture. It can be useful to block new nodes from being created altogether so a painter can zoom in to a model to tweak existing paint color without increasing detail.

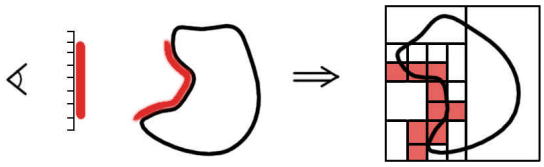

Figure 3: A 2D example of applying paint into an octex. The paint fills a number of pixels from the point of view of the camera. The octex is subdivided until the nodes are approximately the size of a pixel.

The process of culling a child node that is identical to its parent can be broadened to include child nodes that are merely similar to their parents. When the color of the child is within some epsilon of the parent's color, the child can be removed. The epsilon can vary depending on the depth of the node.

A primary benefit of this system is that any type of model which can be rendered can be painted. The geometric operations used during painting are essentially a subset of the operations which would occur during rendering. Checking for an intersection between a cube and a surface is well defined. It is also necessary to clip models to a leaf node. As such, our method works well on a wide variety of surface types: polygons, parametric spline surfaces, subdivision surfaces, and so on.

5 Normals in the Octex

The octex stores the texture volumetrically, but it is only meant to be applied to a surface. It is possible for paint to leak to the opposite side of a very thin model. Consider painting on an airplane wing. As the top of the wing is painted near the sharp edge, some of the nodes may be large enough to also contain the surface on the bottom of the wing (figure 4).

Figure 4: The top of the wing is painted red, the bottom blue. The nodes colored yellow contain two surfaces, so the paint there will depend on the order of application.

To avoid this problem, the normal of the surface receiving paint is written into the leaf node along with the paint color. Later, if another paint sample is applied to the node, the normal of the existing paint and the normal of the incoming paint are compared. When the normals are close enough, then the colors are combined into one color, and the normals are averaged into one normal (figure 5). If they are further apart, then the new paint sample and normal are written into the node in addition to the original data. "Close enough" is determined by controls in the 3D paint application, but generally defaults to 90 degrees.

Just as the deformations of the model during animation must be accounted for by storing the undeformed spatial coordinates as texture coordinates, the undeformed normals must also be stored in the model. While this increases the size of both the model and the octex, this extra overhead only needs to be taken on if the model is thinner than the detail of the paint to be applied.

Since an accurate normal is not needed, the normal in the octexture can be encoded in a small number of bits [Zhang and Hoff 1997]. Even 8 bits per normal can indicate 256 directions which are more than enough for our purposes.

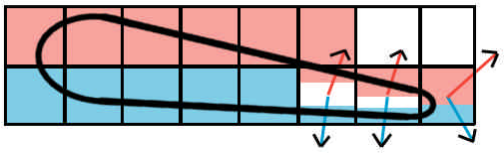

Figure 5: Nodes with two surfaces contain two color samples, with an associated normal.

6 Rendering

Rendering using an octex is very similar to existing techniques using volume textures, making it trivial to add this feature to any existing renderer that supports 3D texture coordinates by using an alternate texture-lookup function. The model's untransformed vertex positions are stored into the 3D texture coordinates and interpolated as usual. For deforming models, the undeformed positions from the reference model are used. For renderers that support custom shaders, all the octex sampling can be done inside the shader. Typical hardware renderers handle the texture lookup directly making support of octextures difficult.

As described in section 3, the filtered texture data within a node is a 3D extension of a MIP map. The simplest way to compute this filtered color is to average the colors for all the potential children, even those which do not exist. This is analogous to a simple box filter. Instead of weighting the child nodes equally, we weight the samples using the area of the surface inside the node, preventing portions of the model that just barely intersect an otherwise empty node from being weighted improperly. Other filters could be used by looking into neighboring nodes, but we have not explored using them.

In a traditional texture, the MIP map level is chosen using either the area of the sample in texture space or the screen-space partials of the texture coordinates. This presents problems since the texture coordinates can have discontinuities and singularities. The octex has no such artifacts since the index is simply the model's original vertex locations in 3-space.

When a node is encountered during rendering which has multiple texture samples and associated normals, the normal of the deformed surface needs to be transformed back to model space to know which texture sample to use. If the model is deformed, the original untransformed surface normals will need to be stored in the model in the same way as the untransformed vertices.

7 Memory

Memory requirements of the octex will vary in comparison to a 2D texture of similar detail. The storage size of the leaf nodes can be exactly the same as the storage size of a pixel in a 2D texture map. However, the size of the parent nodes can be quite a bit larger depending on the implementation details. A simple implementation will require up to eight pointers in the parent nodes. At four bytes each, these nodes are significantly larger than the leaf nodes. The overhead of the pointers can be reduced by using a pointerless octree [Samat 1991]. These representations use less memory than the traditional pointer-based tree. In addition, storing extra data in a leaf node (such as normal information) increases the size of that node.

The size of an octex will be larger than the best 2D texture in cases where there is a good 2D mapping and when the texture is of uniform detail. A simple square is very easy to texture using a 2D image. When the texture contains uniform detail and the smallest

detail is 0.1% of the area of the square, the 2D texture map will require 1 million pixels, approximately 3MB of memory. For the same simple object, the octex will be larger. There will still be 1 million leaf nodes using the same 3MB, but there will be some cost for the parents. In this case there will be about 1/3 as many parent nodes as leaf nodes since each parent node will only have 4 potential children. Since all potential children will exist, those parents will not have any colors. Using the simple pointer-based octree, they will require 4 pointers to their children which cost an additional 5.3MB.

The overhead cost of the tree is variable, depending on the uniformity of texture detail. For uniform textures, all potential children will be created. The number of parent nodes in a tree of depth d is:

$$\sum_{k=0}^{d-1} n^k = \frac{n^d - 1}{n-1}$$

where n is the branching factor. When n is the maximum of 8, the octex has an overhead of 14% in terms of the number of nodes. In practice, we have found that a typical surface mode is locally flat. This means on average there are only about four potential children in most of the nodes. In this case, if the tree were built down to a uniform depth, the tree overhead would be about 33%.

Figure 6: 2D texture map used in figure 9a. Note the wasted space, distortion and discontinuities.

A texture is rarely uniform in its detail across the surface. Unlike the octex, the traditional 2D texture wastes quite a bit of space to account for the high-detail regions, and space will also be lost to areas not used by the parameterization (figure 6). The overhead of the octex can be amortized across multiple textures by storing them in the same tree. When there are N textures in the same octex, the overhead cost per texture shrinks by a factor of N.

Suppose that the 1 million pixel texture above is all white except for one pixel. The 2D texture will still cost 3MB in memory. However, the octex will be very small, just one leaf node and 10 parent nodes which can be less than 50 bytes! Real-world textures are not so extreme, but highly non-uniform texture detail is very common.

In order for the cost of the octex to be totally negated, and the octex to actually have fewer nodes than the corresponding 2D texture has pixels, we only need a small portion of the model to have one level less detail than the most detailed region. Assuming all nodes are of equal size, the break-even point is when:

$$\sum_{k=0}^{d-1} n^k + x * n^d = n^d$$

where x is the percentage of leaf nodes which are at the full depth of the octex, and n is the branching factor. This turns out to be just:

$$x = \frac{n^{-d} + n - 2}{n-1}$$

For any reasonable depth, and a typical branching rate of 4, x will be two-thirds. For the octex to have fewer nodes than the 2D texture has pixels, just one third of the leaf nodes need to be at one level less than the maximum.

For example, when texturing a creature, more detail would probably be painted on its face than on its feet, even though both areas are likely to have similar geometric complexity. For an octex to be smaller in memory than traditional 2D textures, some of the texture needs to have less detail than the most detailed regions to negate the overhead cost of the octex. The data structures for parent nodes are typically larger than those for a single pixel, thus the octree will usually require more memory than the visually equivalent 2D texture. The octex does grow at the same rate as the 2D texture as details are added, therefore doesn't have the excessive growth normally associated with volume textures.

8 Examples and Results

We have implemented these techniques in a 3D paint program and a very simple software scan-line renderer. The only octex-specific part of the renderer is a shader that loads and samples the octex files. All images use four samples per pixel. All speed measurements were taken on a 1 GHz Pentium III.

The 3D paint program allows the user to load in any model and immediately begin painting. To rapidly preview the model, the octex is sampled at the vertices of the model and is drawn using interpolated vertex colors. When painting begins, a high quality image is generated using our scan-line renderer. The user can then paint on the model, and the paint is applied to the octex.

Figure 7 shows a model with a complex topology along with a visualization of its octex at several points during the painting process. Initially the texture was roughed in, resulting in a coarse octex. As the texture was refined, the nodes in the tree become smaller and the structure of the model became evident. The final octex has 947,650 nodes, 715,763 of which are leaf nodes. The average number of children per parent node is 4.08.

Figure 8 shows an animating model. This is a simple cylinder that is bending to the left and wrinkling a bit as it bends. The texture sticks to the model appropriately.

Figure 9 shows a creature textured with a 2D map, and the same creature textured with an octex. This creature took about 6 hours to model. Preparing the parameterization for the standard 2D texture maps initially took 3.5 hours, plus about 2 hours to repeatedly go back and adjust them. The mapping was made from an edge cut along the middle of the underside of the creature and down each leg, attempting to hide the seam. The texture coordinates were edited by hand to get the best mapping. Due to their length, the legs were not getting a large enough portion of the texture map, so additional cuts and edits were needed. Sometimes multiple 2D maps are used to avoid wasted space and to better fit the geometry at a consistent resolution, but introduce many more discontinuities. All told, the mapping process took nearly as long as modeling the geometry. Even in the final mapping (figure 6), there are still artifacts due to the parameterization.

The creature's octex has approximately 3 million nodes, 2.24 million of which are leaf nodes. There are roughly twice as many leaf nodes in the octex as there are pixels in the 2D texture map. The average number of children per parent node is 3.92. Not all potential children have been created, indicating that the texture detail is not uniform. The images were rendered at a resolution of 2048x1200. The traditionally mapped images took 1 minute 37 seconds to render, while the octex images required 2 minutes 43

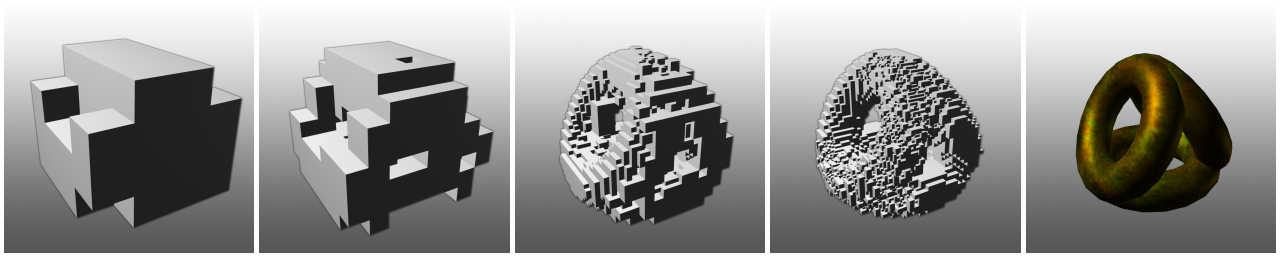

Figure 7: Model with complex topology and its octex at various stages in the painting process. Note how the octex becomes more detailed as more detail is added to the texture. Holes in the refined surface represent potential children that have been unnecessary to create, as only leaf nodes are shown. The final image in the sequence is the actual model with the painted texture.

Figure 8: Several frames from an animated sequence of a bending cylinder. The octex behaves correctly when applied to a deforming model.

seconds. The octex images render 68% slower than the traditional texture maps. This represents the increase in time to sample an octex as opposed to a simple 2D array. The 2D texture takes 1.85MB losslessly compressed, and the octex is 7.4MB, also compressed without data loss.

9 Conclusions and Future Work

We have presented a texture storage, painting, and rendering technique that is free from the issues of parameterization between disparate topologies. We have shown how our method allows for the texture to be exactly as detailed as the paint that created it. While the structure of the texture data is more expensive than a traditional 2D map, it is significantly cheaper than a full 3D volume texture. We consider the expense incurred from the size increase to be well worth the time saved by eliminating the solution of iterative parameterization adjustments.

There are several areas of expansion and improvement open to this method:

- Time repainting a model changed during production could be reduced if data within the octex were able to shift to match the areas where the model had changed.
- Filtering and sampling could both be improved. Since the octex represents a 3D extension of MIP mapping, an improvement on 2D MIP mapping [McCormack et al. 1999] may be applicable. The use of more sophisticated filter kernels will also be required.
- Allow paint to be stored not only at the surface position, but also at a location displaced along the surface normal. For example, a creature's skin texture would be painted normally, but the texture of muscles and fat under the skin could be painted *beneath* a model's surface. This sub-dermal color could be blended with the skin color during rendering.

10 Acknowledgments

Thanks to Lawrence Kesteloot, Drew Olbrich, Dave McAllister, Peter-Pike Sloan, Thouis Jones, Daniel Wexler, and most especially to Mark Edwards. Thanks also to Gail Currey, Phil Peterson, and Andy Hendrickson at ILM, and Aron Warner and Ed Leonard at PDI/DreamWorks for their support, and John Hughes, JP Lewis, Vicki Caulfield, and our referee for all the time and help.

References

BENNIS, C., VÉZIEN, J.-M., IGLÉSIAS, G., AND GAGALOWICZ, A. 1991. Piecewise surface flattening for non-distorted texture mapping. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, vol. 25, 237–246.

BENSON, D., AND DAVIS, J. 2002. Octree textures. In *Proceedings of ACM SIGGRAPH 2002*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 101–106.

BERMAN, D. F., BARTELL, J. T., AND SALESIN, D. H. 1994. Multiresolution painting and compositing. In *Proceedings of SIGGRAPH 94*, ACM SIGGRAPH / ACM Press, Orlando, Florida, Computer Graphics Proceedings, Annual Conference Series, 85–90.

BLINN, J. F., AND NEWELL, M. E. 1976. Texture and reflection in computer generated images. *Communications of the ACM 19*, 10, 542–547.

BOADA, I., NAVAZO, I., AND SCOPIGNO, R. 2001. Multiresolution volume visualization with a texture-based octree. *The Visual Computer 17*, 3, 185–197.

DAILY, J., AND KISS, K. 1995. 3d painting: paradigms for painting in a new dimension. In *Proceedings of CHI'95*, ACM Press.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 249–254.

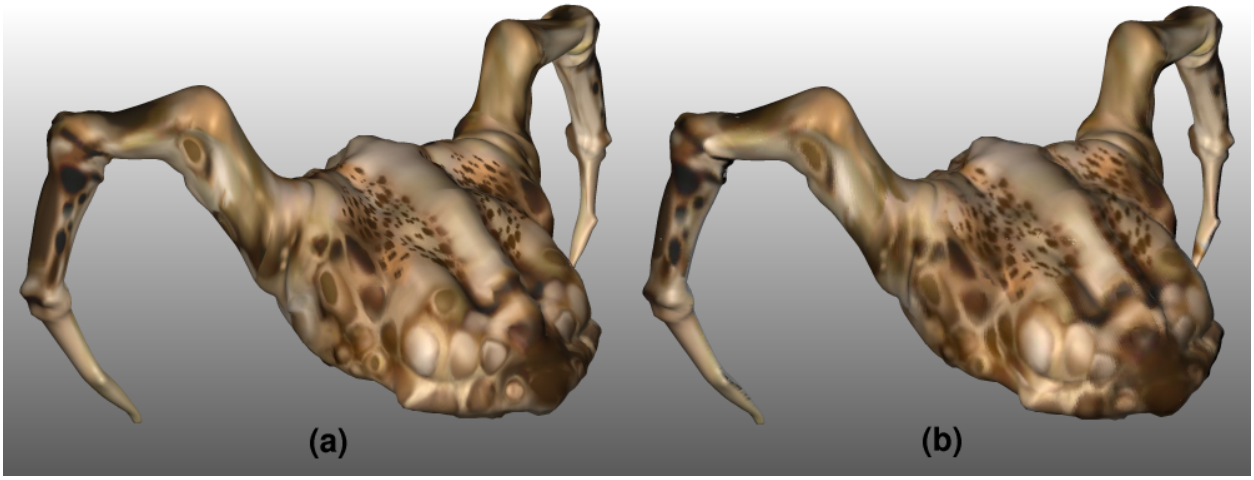

Figure 9: Model with (a) a 2D texture and with (b) an octexture.

HAKER, S., ANGENET, S., TANNENBAUM, A., KIKINIS, R., AND SAPRIO, M. H. 2000. Conformal surface parameterization for texture mapping. *Transactions of Visualization and Computer Graphics 6*, 2, 181–189.

HANRAHAN, P., AND HAEBERLI, P. E. 1990. Direct wysiwyg painting and texturing on 3d shapes. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, 215–223.

HUNTER, A., AND COHEN, J. D. 2000. Uniform frequency images: adding geometry to images to produce space-efficient textures. In *IEEE Visualization 2000*, 243–250.

IGARASHI, T., AND COSGROVE, D. 2001. Adaptive unwrapping for interactive texture painting. In *2001 ACM Symposium on Interactive 3D Graphics*, 209–216.

LÉVY, B. 2001. Constrained texture mapping for polygonal meshes. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 417–424.

MA, S. D., AND LIN, H. 1988. Optimal texture mapping. In *Eurographics '88*, 421–428.

MAILLOT, J., YAHIA, H., AND VERROUST, A. 1993. Interactive texture mapping. In *Proceedings of SIGGRAPH 93*, Computer Graphics Proceedings, Annual Conference Series, 27–34.

MCCORMACK, J., PERRY, R., FARKAS, K. I., AND JOUPPI, N. P. 1999. Feline: Fast elliptical lines for anisotropic texture mapping. In *Proceedings of SIGGRAPH 99*, ACM SIGGRAPH / Addison Wesley Longman, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 243–250.

PEACHEY, D. R. 1985. Solid texturing of complex surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, vol. 19, 279–286.

PERLIN, K. 1985. An image synthesizer. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, vol. 19, 287–296.

PIPONI, D., AND BORSHUKOV, G. D. 2000. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 471–478.

SAMAT, H. 1990. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 409–416.

SLOAN, P.-P. J., WEINSTEIN, D. M., AND BREDERSON, J. D. 1998. Importance driven texture coordinate optimization. *Computer Graphics Forum 17*, 3, 97–104.

TAMMINEN, M., AND SAMET, H. 1984. Efficient octree conversion by connectivity labeling. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, vol. 18, 43–51.

WESTERMANN, R., AND ERTL, T. 1997. A multiscale approach to integrated volume segmentation and rendering. *Computer Graphics Forum 16*, 3 (August), 117–128.

WILHELMS, J., AND GELDER, A. V. 1994. Multi-dimensional trees for controlled volume rendering and compression. 27–34.

WILLIAMS, L. 1983. Pyramidal parametrics. In *Computer Graphics (Proceedings of SIGGRAPH 83)*, vol. 17, 1–11.

ZHANG, H., AND HOFF, K. 1997. Fast backface culling using normal masks. In *Symposium on Interactive 3D Graphics*, ACM Press, 103–106.

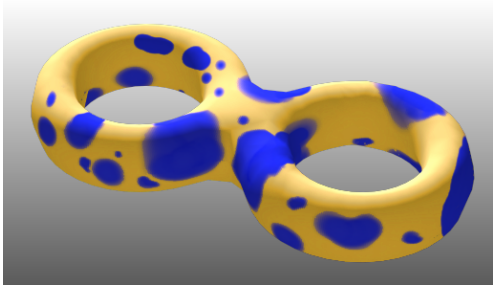

Figure 10: Simple figure eight textured with an octexture.