EUROGRAPHICS '98 / N. Ferreira and M. Göbel
(Guest Editors)

Volume 17, (1998), Number 3

Importance Driven Texture Coordinate Optimization

Peter-Pike J. Sloan, David M. Weinstein, J. Dean Brederson

Department of Computer Science University of Utah
email: {ppsloan,dweinste,jdb}@cs.utah.edu

Abstract

Traditionally, texture coordinates have been generated based solely on the model's geometry, often even before a model's textures have been created. With the arrival of new technologies, such as 3D paint programs, weaknesses of a static optimization pre-process are becoming apparent. These weaknesses arise from constructing a parameterization based solely on the model's geometry, ignoring the fact that detail is not uniformly spaced throughout the texture space. In fact, certain regions of the texture are more important than other regions. In this paper we introduce the notion of the "importance map" and describe how importance values are derived from both intrinsic properties of the texture and user-guided highlights. Furthermore, we describe how importance maps are used to drive the texture coordinate optimization. Finally, we show how this optimization process can be integrated into a 3D painting environment, enabling periodic optimization at any stage of texture design.

Additional Keywords and Phrases: texture mapping, texture map distortion, interaction

1. Introduction

Texture mapping[7] has become a popular technique for representing intricate detail with relatively simple geometry. In constructing texture mapped models, the modeler must define a parameterization of the model. This parameterization, often stored as texture coordinates, maps the three-dimensional model space to a flat, two-dimensional texture space. Most surfaces must undergo a great deal of distortion when they are reparameterized into a plane, often resulting in undesirable artifacts when the model is rendered.

One way to avoid these artifacts is to optimize the parameterization according to the relative detail in the texture. An example of this is shown in Figure 1, where a texture is mapped to a sphere. A standard latitude-longitude parameterization of the sphere is used in the left column, while an optimized parameterization is used in the right column. Since the detail in the mapped texture is close to the equator rather than the poles of the sphere, we can improve the appearance of the mapped texture by predistorting the texture map to effectively increase the texture resolution in areas of detail.

Minimizing the *metric* distortion induced by embedding three-dimensional models in texture space has been the focus of previous texture coordinate optimization research [18, 3, 19, 22, 13]. While these earlier methods do an excellent job of addressing the geometric form of the model, they are intrinsically limited because they reparameterize as a preprocess. In other words, they apply their minimization operations before any "paint" has been applied to the surface. As a result, too much texture space is often devoted to regions that ultimately contain little or no texture content, while regions destined to contain rich detail are starved for adequate memory. Allotting insufficient texture space for regions containing large amounts of detail can result in undersampling artifacts, such as the "jaggies" seen in Figure 1 in the lower left closeup and the blurring seen in Figure 2b.

What these optimization methods have failed to take into account is the information distribution *within the texture itself*: they assume that all areas of the model will have equal texture detail. This shortcoming has recently been highlighted with the advent of 3D painting programs[11, 22, 23]. With these programs, users now have fine control over where they apply detail on the surface of models, and in practice it is quite common for textures to have extremely inhomogeneous distributions of detail.

For example, in industrial design it is common to use a 3D paint program to sparsely sketch various details over the surface of a model. Figure 2a shows such an example of a car

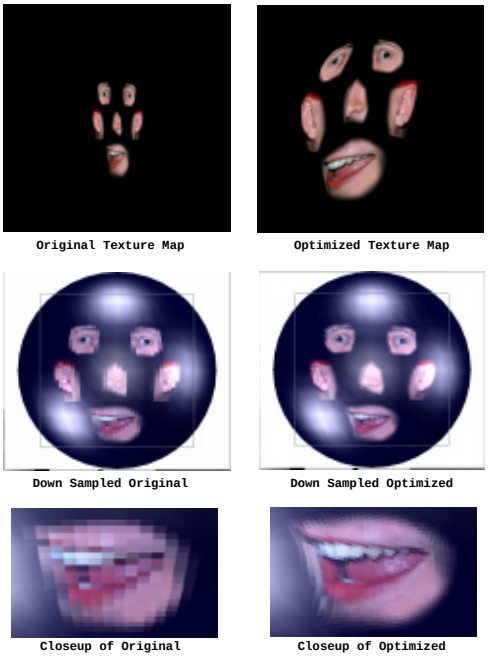

Figure 1: *Example of an unoptimized (left column) and optimized texture (right column) mapped onto a sphere.*

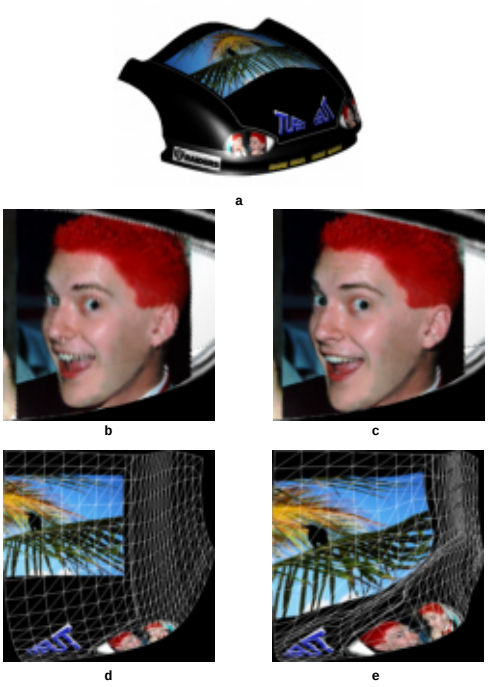

Figure 2: *A comparison of texture mapping using standard (b,d) parameterization versus importance driven (c,e) parameterization. Note the fidelity preserved in a newly painted detail (the head), when added in a region identified as important.*

hood model with a texturing applied. In Figure 2b, we have a zoomed in view of the headlight from this model, showing that an insufficient amount of texture space has been allocated to capture the fine detail of the texture locally. Figure 2d shows the texture map of this model, illustrating how common texture parameterization techniques fail to redistribute texture space based on detail.

In contrast, Figures 2c, 2e show the corresponding texture close-up and texture parameterization, respectively, using our importance-driven technique, demonstrating that by distributing the texture space according to the detail of the texture, we can increase the fidelity of texture-mapping without increasing the texture memory budget. In this paper we discuss how we have extended the previous work of texture map optimization to include a "relative importance" coefficient for each region of the texture. This coefficient combines the information distribution of the texture with a user-defined importance value.

Section 2 relates our work to previous techniques for texture mapping. In Section 3 we show how we derive the importance map of a texture, discussing both the automatic methods used, and the user-driven interface. Section 4 describes how this importance map, in conjunction with a metric distortion minimization term, drives the texture coordinate optimization process. Finally, in Section 5 we summarize the major contributions of our work, and discuss how our method can be incorporated into existing texture mapping methods.

2. Relation to Previous Work

Much research has examined texture mapping and its relation to modeling. We give a brief overview of these earlier contributions in the three general related areas of texture coordinate generation/optimization techniques, 3D painting paradigms, and compressed texture representations.

Two prior papers[18, 19] have used a conjugate gradient algorithm to minimize the metric distortion introduced by the mapping. Maillot[19] presented an attractive interactive environment for creating an initial parameterization over a surface. Krishnamurthy[13] describes how to fit smooth surfaces to polygon meshes, using a relaxation technique to optimize the parameterization of the surfaces (and therefore of the textures). The key distinctions between our method and these techniques are our use of an importance metric and where, and how often, we optimize the parameterization.

Eck *et al.*[10] use harmonic mappings to generate a parameterization over a polyhedral mesh for fitting a multiresolution surface to the model. Again, the key distinctions between

this work and our work are in the lack of an importance notion and in the static nature of the parameterization.

Litwinowicz *et al.*presented a method for interactively placing textures on a surface[17], via a user interface for specificing an importance map. A key distinction is the amount of user interaction required for this method and the fact that their method assumes the textures have already been created. Essentially, they are matching corresponding points between the model and the texture, not optimizing the parametrization based on the notion of where detail is.

The original 3D painting paper by Hanrahan[11] did not explicitly use textures, but instead used a dense mesh (which could be later turned into a texture if necessary) to represent the surface. The mesh subdivision was irregular and based on detail, but it was never discussed how this could be mapped to a texture. Agrawala[1] also used a dense regular mesh, but never attempted to generate a parameterization over the surface. It would be difficult to paint fine details on this type of surface.

Pedersen[22, 23] presented a general 3D painting framework. He advocates the use of "patchinos," which are small regions of the surface that have their own local mapping and can be easily dragged around the surface. He uses the same techniques as[19] to continually minimize the distortion in the mapping. A shortcoming to this approach is that it only works on smooth surfaces and fails at discontinuities. This method also requires a non-standard representation for these texture patches.

Two earlier papers referred to compressed textures: one using vector quantization by Beers[2], the other by Torborg which utilized a new architecture[25]. The work presented here is complimentary to both of these techniques. The notion of user-specified importance could not be easily addressed in a typical compressed representation. Also, tying the parameterization of the surface into the compression of the texture could be useful when looking at how to deal with metric distortion. Another key distinction is that our representation is fully supported on current hardware and software rendering engines.

3. Importance Specification

Our use of importance represents the intrinsic content of texture space on a local level. The distribution of information in the texture guides the distribution of texture memory. Areas of a texture are allocated texture space that is proportional to their importance. Thus, texture memory is budgeted such that areas of high detail can be reproduced with high fidelity without requiring a larger texture memory. Importance is determined for each texel and an importance map is defined by the scalar field consisting of the importance values over the domain of texture space. This importance map is then used to drive the optimization of texture coordinates. We have implemented two methods for specifying importance: the first method employs analysis and decomposition techniques to automatically derive importance from textures, while the second method is user-specified.

3.1. Automatic Specification

There has been an extensive amount of research in computer vision[6] and recent work in computer graphics[12] addressing methods of decomposing images to extract features and details. We have looked at several common decomposition techniques including Laplacian pyramids[20], wavelets[8, 4], and steerable pyramids[12]. For our application, we use image decomposition to analyze the texture content, thereby deriving the importance value for each texel in its context. Specifically, we sum the absolute values of the coefficients for the basis functions covering each pixel, where each coefficient is scaled by the appropriate scaling function. This gives a measure of the local information content of texels indicating characteristics such as detail, contrast, and noise. An example of an automatically generated importance map for a texture image is shown in Figure 7b.

3.2. User Specification

Often automatic methods do not faithfully capture the essence of the texture and user intervention is desired. We accommodate this by using existing image creation and manipulation software. The user explicitly tags regions of higher and lower importance with an "importance brush" to define an importance map. User-defined importance may also be combined with automatic importance to leverage the advantages of both methods. Combining the automatic and user-specified importance maps is done either as a direct sum or by modulating the two maps together.

In the 3D painting system incorporating our method, the user paints on the model from a fixed view, with the paint reflecting the lighting of the scene. To push the paint into the texture map, the user has to explicitly "dry" the paint, which generates or modifies the texture map based on the painting done and the parameterization of the model. This "drying" process takes anywhere from a fraction of a second to a few seconds depending on the available hardware.

There are several places during the painting process where the user may choose to specify importance. The user can initially paint importance directly on the model to optimize the parameterization of the model. Once the user "dries" the paint into the texture map, the painted data is resampled according to the parameterization of the model, hence artifacts can occur in the drying process. The user can undo the "dried" paint and modify the importance map to reallocate detail for regions of the texture. The user may also specify importance once painting is finished to highlight important perceptual features. Such an approach would be particularly

useful in entertainment applications, where the final production texture maps may have to be quite small compared to the development texture maps.

Importance cannot be simply treated as paint by the application, however: when data is dried, it undergoes an area distortion due to the parameterization and the importance needs to reflect this. Think of a sphere with a standard latitude/longitude parameterization. The regions near the poles have a higher texture space to object space ratio, effectively spreading out the importance in the texture. Compensating for this is fairly straightforward, you simply need to multiply the importance by the object space to texture space distortion. This map can be easily computed in texture space using the graphics hardware:

```
set view matrix to map tex coords to screen

foreach triangle
  set color (AreaR-MinR)/(MaxR-MinR)
  draw using texture coordinates as vertices
```

AreaR is the ratio of the areas of the triangle in object space and texture space, and MinR and MaxR are the minimum and maximum values respectively. This transforms them into [0,1] which is required by OpenGL. When the frame buffer is read back, this transform needs to be inverted. If this map is used as the importance map, the optimization methods will try and make this ratio constant. Other geometric aspects could be drawn in a similar fashion–the error functional in[19] for example– however, this method is the most suited for the optimization techniques here, which work mostly with areas. An example illustrating a combination of automatic and user-defined importance is shown in Figure 7.

4. Texture Coordinate Optimization

Considering the importance map for a texture as a height field, we have implemented the following new optimization for texture mapping: *regions covering equal volume in the importance map should be allocated equal area in texture space.* This heuristic maximizes the potential rendering fidelity for a fixed texture memory size by ranking the representational needs of the texels. The scalar values can be thought of as representing the spatial frequencies that individual texels have in a local neighborhood. Regions that have high importance want more texture area, and regions with low importance want less.

While optimizing the use of texture space, we still need to minimize distortion between texture space and the model surface. We approach the problem in two phases: first we warp the texture coordinates via a deformation that optimizes according to the importance map, and second we relax the texture mesh and add a term to minimize metric distortion.

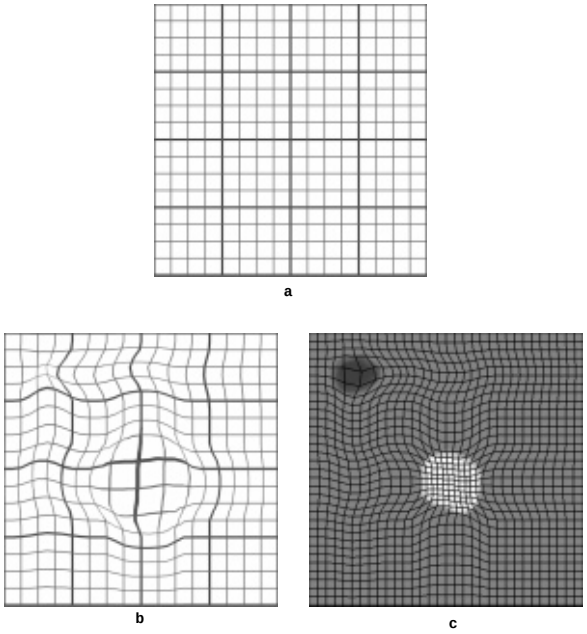

Figure 3: *A simple initial grid (a), and the grid distorted (b) based on a simple importance function (c). The light and dark areas in (c) specify increased and decreased importance, respectively. Note that corresponding areas between grid lines in (b) are larger and smaller relative to their specified importance in (c).*

4.1. Constructing the Deformation

While there are many techniques to perform this part of the optimization (from both the morphing/warping literature[14, 27] and thin plate spline techniques[15]), we chose a technique similar to a multilevel free-form deformation[14]. Using this approach, the deformation is constructed in a coarse-to-fine manner. We begin by computing prefix sum tables for each row and column in the importance map. Entries in these tables are the sum of the value of the current element and all of the preceding elements — a one-dimensional version of a summed area table[9]. Isocontours are then computed separably in each dimension at evenly spaced contour values.

We want to define the deformation as a linear mapping from the canonical texture space to the intersection points of the computed isocontours. However, the isocontours must be filtered to ensure that the deformation will be one-to-one so that the texture space will not fold in upon itself. Filtering is performed on each isocontour until there is no more than a single pixel deviation between successive rows and columns. Once the initial isocontours have been generated and filtered, successively finer levels are computed by sampling between the filtered isocontours. The intersections of the finest level of isocontours then define the vertices of the deformed mesh.

If this mesh is degenerate or folds over, a small value is added to every pixel in the importance map and the process is repeated until it converges. Adding this value lifts the entire height field, decreasing the relative changes in volume and causes the isocontours to straighten out. The tiles of this mesh should each be allocated equal area in texture space according to our optimization heuristic. A simple example of this is in Figure 3, where a simple importance map is defined that has one region of increased importance (the center of Figure 3c) and one region of decreased importance (the upper left corner of Figure 3c). The mesh overlayd on Figure 3c is the grid which is mapped back to a regular grid, applying this mapping to the grid results in Figure 3b, where one can see that the "important" region has been streched out, and the unimportant region has been compressed.

4.2. Applying the Deformation

The deformation ζ maps the canonical texture space into the deformed texture space, while the inverse deformation ζ^{-1} maps the deformed texture space back to the canonical texture space (see Figure 4). For a polyhedral model, vertices of the deformed mesh are mapped to the domain of the canonical texture mesh. This mapping is well-defined because the deformation is guaranteed by construction to be one-to-one. We triangulate the deformed mesh and use a piecewise linear mapping between the canonical and deformed texture spaces. A higher order mapping could easily be used provided the additional constraints presented in[14] were used to guarantee that the deformation was one-to-one.

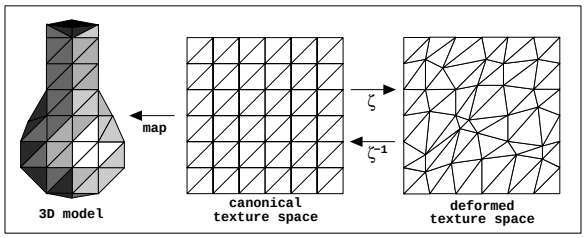

Figure 4: *Figure depicting the relations between model, and the canonical and deformed texture spaces. Note that ζ and ζ^{-1} map texture coordinates in and out of the deformed space.*

4.3. Atlas Based Optimization

We have done some preliminary research into using a spatial data structure (quad tree in this case) to subdivide the domain into tiles of equal importance. The goal is to create a quad tree where the leaf nodes will all be allocated tiles of equal area in the new domain[21]. We use the following method:

```
PQueue.Insert(Root)
numDone=0
while (numDone < numDesired) {
  cur = PQueue.pop()          // top of Q
  if (cur.level > maxLevel) {
    cur.retire                // is final
  } else {
    insert children           // split
    numDone += 3
  }
}
```

PQueue is a priority queue of leaf nodes, sorted based on the integral of the importance function, which is efficiently computed using a summed area table[9]. maxLevel is the deepest level the tree is allowed to reach, and numDesired clearly has to be less than the number of leaves in a full tree.

Any leaf node whose integral is below a preset threshold is analyzed to see if the data is constant (assuming there is already data in the texture). If it is, the tile can be stored as a single texel. We are left with a tiling problem, since all of the leaf nodes have to be packed into a texture map. We can store a border explicitly around each tile that is resampled from the neighbors in the original domain, which will lessen the artifacts resulting from the change in texture detail between neighboring tiles that have different texture densities (with respect to object space). Alternately, the contents of individual tiles could be blurred based on the texture density of its neighbors, causing an even more gradual transition if desired.

This technique does not suffer from the anisotropic distortion that can occur in the warping technique. Figure 5 shows a simple billboard texture that has been optimized by both techniques and down sampled to stress the artifacts. The upper right shows the deformation technique, with a zoomed-in view of some of the thin lines below. These were severely stretched because of the strong important regions below and to its upper right. The upper left shows the results using an atlas. There are some clear artifacts from the change in spatial resolution, but not the smearing that is present using the deformation method. The bottom of the figure shows the texture atlas and the quad tree (the leaf nodes are identified by the black triangles.)

Pushing existing geometry through the atlas transformation is much more involved than using the deformation technique, since polygons cannot cross over between two tiles. There are several possible solutions to this problem. One is to just clip the geometry at the tile boundaries, but this could drastically increase the geometric complexity of the model. Another technique would be to cause either the tile boundaries to conform to the geometry, or the geometry to conform to the tile boundaries. In our simple example, we created the geometry from the quad tree itself.

This method creates a discontinuous set of local subdomains, meaning that mip-mapping cannot be used for texture filtering. However, traditional mip-mapping tends to do

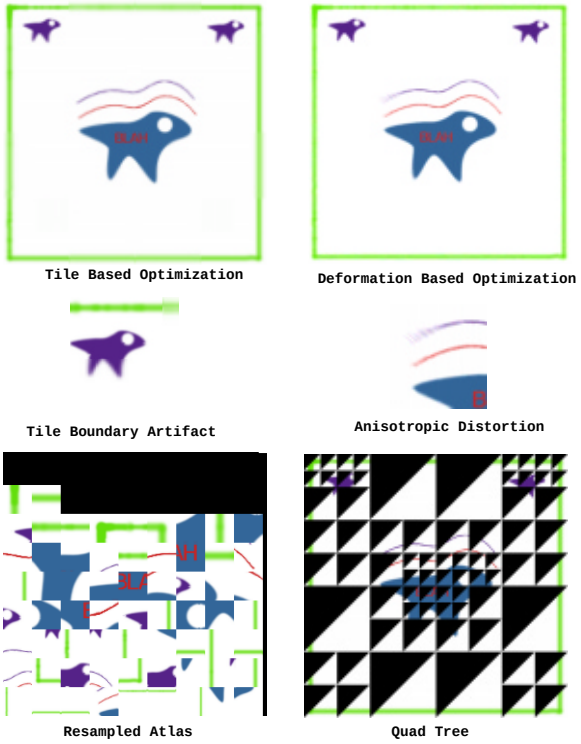

Figure 5: *Figure showing a texture atlas, and highlighting the artifacts from the different methods.*

a poor job where there is any parametric distortion, which almost always occurs with a global parameterization[26]. Anisotropic texture filtering techniques [25], or more clever evaluation of mip-maps[5], would be beneficial for both of these methods. An interesting application for this method is for creating texture maps for the Nintendo 64 game platform. That particular platform has a limited, 4 kilobyte, texture cache which must be explicitly maintained by the user. Textured primitives can only be drawn if there corresponding texture regions is already in the cache. This is an excellent application for this technique.

4.4. Relaxation

The relaxation phase of the optimization operates on the embedding of the model into the deformed texture space in order to minimize distortion. To accomplish this, we induce forces on each vertex of the deformed mesh. These forces are similar to those presented by Krishnamurthy et al. in[13], but our method maintains relative importance and minimizes metric distortion.

There are two forces for each vertex of every triangle, one which maintains importance and another which minimizes distortion. Magnitudes are determined based on the difference between the desired area (the relative amount of the importance map covered) and the actual area of the triangle in the deformed mesh. The force vectors at each vertex are orthogonal to its opposite edge, directed away from that edge to "grow" the triangle or towards that edge to "shrink" the triangle. There are also force vectors at every vertex that minimize metric distortion inherent to the embedding of the model in texture space. The distortion force maintains similarity for corresponding triangles of the deformed mesh and the model mesh. By nudging vertices, the distortion force works to preserve the aspect ratios of corresponding triangles. For each vertex in the mesh, the two resultant force vectors are computed with the magnitudes clamped in order to prohibit the mesh from folding over upon itself. Finally, the importance and distortion forces are alpha-blended according to a user- specified weighting parameter. This relaxation process is iterated until the system converges for a specified tolerance. Figure 6 shows before and after images of a reparameterized space. This solution is fairly inelegant, and often this phase isn't even necessary. A more robust integration of the different metrics is needed.

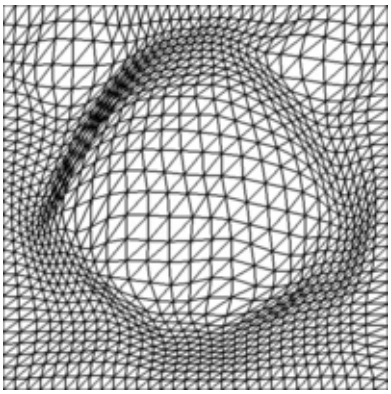

Figure 6: *The warped mesh used to construct Figure 5 with the deformation technique. Notice the distortion around the center of the texture.*

4.5. Application to Parametric Surfaces

If the model is a B-spline or Bezier surface, the deformation mesh can be used to reparameterize the surface, as in[24]. This reparameterization is only used for determining texture coordinates over the surface. The relaxation can be done directly on the deformation mesh, and metric distortion can be explicitly computed and optimized using this mesh as well. Thus, our method can be applied to both polyhedral and parametric models.

5. Conclusion and Future Directions

We have introduced the idea of an importance map, and shown how such a metric can be used to better drive the optimization of texture coordinates. By accounting for the

inhomogeneous content of the texture, we are able to rescale the parameterization, partitioning greater space for regions of greater importance. In addition, we have shown how to generate automatic importance maps from the intrinsic content of a texture, as well as how to incorporate or even override that map with user-guided importance values.

Clearly, not all textures can benefit from these techniques. Textures need to have spatially localized detail. This is often the case in industrial design, where users are simply painting local details on objects (it was seeing how a commercial 3D painting system was used in ID that motivated this research. The user was painting detail lines and headlights on a car, around 90% of the texture space was wasted.), and we believe this could also be the case for some entertainment applications. Textures that have uniform spatial frequencies (brick, sand, etc.) receive no benefit from these methods.

There are several interesting areas of future research. More work needs to be done integrating the more conventional metric distortion terms into the optimization process. Ultimately, we would like to further integrate these methods with a 3D painting environment. Such a coupling would more clearly illuminate one of the strengths of our method, namely the ability to optimize the texture parameterization at any of three primary stages of the modeling/painting cycle. The first method is purely as a preprocess, in which the user could paint either on the surface of the model or in the space of the texture where they want more or less detail. The second method is as a postprocess. The user could paint on larger or multi-resolution texture maps, and then optimize when the textures were rescaled to a more practical size. This method would be useful in entertainment applications or when the user has to work with machines that have limited texture memory. The final method is to periodically perform the optimization as the user creates the texture map. This process could be based on brush history, texture characteristics, explicit requests by the user, or any combination thereof. This tight level of integration would give the user more freedom when designing textures for models, and would be a convenient way to work around the metric distortion problem.

We are also currently extending the tiled optimization technique to work with 3D textures used in volume rendering. In the context of volume rendering it is also desireable to have the tiling be a multi-resolution one, allowing the user to dynamically control the detail based on regions of interest. Dynamically evaluating the level of detail of the texture over space could also be applied to rendering surfaces, and fits in nicely with some of the recent Talisman papers from Microsoft[16].

6. Acknowledgments

This work was supported in part by the National Science Foundation. The authors would like to thank Chris Johnson, Peter Shirley, and Brian Smits for their helpful comments and suggestions. We would also like to thank Marty Cole at Parametric Technology for the figure of the car hood, Dave Debry (aka Grue) at PDI for his likeness, and Todd Green for critical system life support. Furthermore, we appreciate access to facilities that are part of the NSF STC for Computer Graphics and Scientific Visualization.

References

1. AGRAWALA, M., BEERS, A. C., AND LEVOY, M. 3D painting on scanned surfaces. In *1995 Symposium on Interactive 3D Graphics* (Apr. 1995), P. Hanrahan and J. Winget, Eds., pp. 145–150.

2. BEERS, A. C., AGRAWALA, M., AND CHADDHA, N. Rendering from compressed textures. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), H. Rushmeier, Ed., pp. 373–378.

3. BENNIS, C., VÉZIEN, J.-M., IGLÉSIAS, G., AND GAGALOWICZ, A. Piecewise surface flattening for non-distorted texture mapping. In *SIGGRAPH 91 Conference Proceedings* (July 1991), T. W. Sederberg, Ed., pp. 237–246.

4. BERMAN, D. F., BARTELL, J. T., AND SALESIN, D. H. Multiresolution painting and compositing. In *SIGGRAPH 94 Conference Proceedings* (July 1994), A. Glassner, Ed., pp. 85–90.

5. BLEAK, J. N., GRANGE, R. L., AND GARDINER, H. D. Overcoming the limitations of todays image generators. http://www.es.com/Products/Sim/harmony/documents.html.

6. BURT, P., AND ADELSON, E. H. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics 2* (1983), 217–236.

7. CATMULL, E. E. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of CS, U. of Utah, Dec. 1974.

8. CHUI, C. K., Ed. *Wavelets: A Tutorial in Theory and Applications*. Academic Press, 1992.

9. CROW, F. C. Summed-area tables for texture mapping. In *SIGGRAPH 84 Conference Proceedings* (July 1984), H. Christiansen, Ed., pp. 207–212.

10. ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH 95 Conference Proceedings* (Aug. 1995), R. Cook, Ed., pp. 173–182.

11. HANRAHAN, P., AND HAEBERLI, P. E. Direct WYSIWYG painting and texturing on 3D shapes. In *SIGGRAPH 90 Conference Proceedings* (Aug. 1990), F. Baskett, Ed., pp. 215–223.

12. HEEGER, D. J., AND BERGEN, J. R. Pyramid-Based texture analysis/synthesis. In *SIGGRAPH 95 Conference Proceedings* (Aug. 1995), R. Cook, Ed., pp. 229–238.

13. KRISHNAMURTHY, V., AND LEVOY, M. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), H. Rushmeier, Ed., pp. 313–324.

14. LEE, S., CHWA, K., SHIN, S. Y., AND WOLBERG, G. Image metamorphosis using snakes and free-form deformations. In *SIGGRAPH 95 Conference Proceedings* (Aug. 1995), R. Cook, Ed., pp. 439–448.

15. LEE, S.-Y., CHWA, K.-Y., HAHN, J., AND SHIN, S.-Y. Image morphing using deformation techniques. *Journal of Visualization and Computer Animation 6*, 3 (1995).

16. LENGYEL, J., AND SNYDER, J. Rendering with coherent layers. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), T. Whitted, Ed., pp. 233–242.

17. LITWINOWICZ, P., AND MILLER, G. Efficient techniques for interactive texture placement. In *SIGGRAPH 94 Conference Proceedings* (July 1994), A. Glassner, Ed., pp. 119–122.

18. MA, S. D., AND LIN, H. Optimal texture mapping. In *Eurographics '88* (Sept. 1988), D. A. Duce and P. Jancene, Eds., North-Holland, pp. 421–428.

19. MAILLOT, J., YAHIA, H., AND VERROUST, A. Interactive texture mapping. In *SIGGRAPH 93 Conference Proceedings* (Aug. 1993), J. T. Kajiya, Ed., pp. 27–34.

20. OGDEN, J. M., ADELSON, E., BERGEN, J., AND BURT, P. Pyramid-based computer graphics. *RCA Engineer 30* (1985), 4–15.

21. PEDERSEN, H. K. Displacement mapping using flow fields. In *SIGGRAPH 94 Conference Proceedings* (July 1994), A. Glassner, Ed., pp. 279–286.

22. PEDERSEN, H. K. Decorating implicit surfaces. In *SIGGRAPH 95 Conference Proceedings* (Aug. 1995), R. Cook, Ed., pp. 291–300.

23. PEDERSEN, H. K. A framework for interactive texturing on curved surfaces. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), H. Rushmeier, Ed., pp. 295–302.

24. SHIRMAN, L., AND KAMEN, Y. Fast and accurate texture placement. *IEEE Computer Graphics and Applications 17*, 1 (Jan. 1997), 60–66.

25. TORBORG, J., AND KAJIYA, J. T. Talisman: Commodity realtime 3d graphics for the pc. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), H. Rushmeier, Ed., pp. 353–363.

26. WILLIAMS, L. Pyramidal parametrics. In *SIGGRAPH 83 Conference Proceedings* (July 1983), P. Tanner, Ed., pp. 1–11.

27. WOLBERG, G., AND BOULT, T. E. Separable image warping with spatial lookup tables. In *SIGGRAPH 89 Conference Proceedings* (July 1989), J. Lane, Ed., pp. 369–378.