Department of Computer Science, Texas A&M University, Technical Report tamu-cs-tr-2008-4-1

Mesh Colors

Cem Yuksel*
Texas A&M University

John Keyser†
Texas A&M University

Donald H. House‡
Texas A&M University

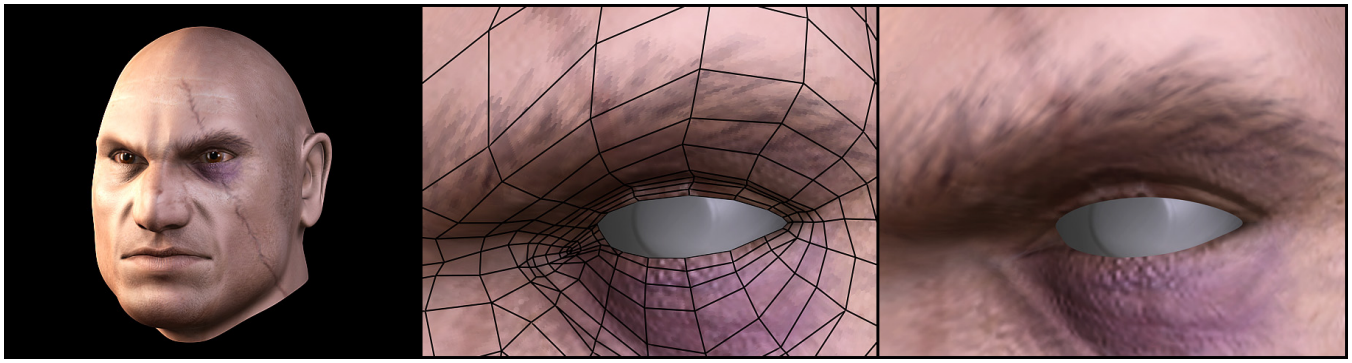


Figure 1: *A head model textured using mesh colors. The image in the middle shows color samples on the low resolution mesh and the image on the right shows the result after final filtering operations. (Modelled and painted by Murat Afsar)*

Abstract

The coloring of three dimensional models using two or three dimensional texture mapping has well known intrinsic problems, such as mapping discontinuities and limitations to model editing after coloring. Workarounds for these problems often require adopting very complex approaches. Here we propose a new technique, called mesh colors, for associating color data directly with a polygonal mesh. The approach eliminates all problems deriving from using a map from texture space to model space. Mesh colors is an extension of vertex colors where, in addition to keeping color values on each vertex, color values are also kept on edges and faces. Like texture mapping, the approach allows higher texture resolution than model resolution, but at the same time it guarantees one-to-one correspondence between the model surface and the color data, and eliminates discontinuities. We show that mesh colors integrate well with the current graphics pipeline and can be used to generate very high quality textures.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: Mesh colors, texture mapping, vertex colors, 3D paint

1 Introduction

Since the early days of computer graphics, the mapping of textures onto surfaces has been the preferred non-procedural way of providing surface detail without adding geometric detail. Notwithstanding its popularity, the need to maintain a map from texture to geometric space, as an intrinsic characteristic, makes texture mapping prone to artifacts and complex to use in situations demanding high visual fidelity. Here, we propose an alternative approach to providing surface detail that is immune to these problems, since it eliminates the need for a map. We call this approach *mesh colors*.

In the mesh colors framework, we associate colors and other surface attributes directly with mesh geometry. This is done in a way that integrates well with existing work-flows and graphics pipelines. Figure 1 shows mesh colors used to texture a head model. The example illustrates the high-detail that can be obtained, the use of level-of-detail filtering to handle far and near views, and the integration of the approach with mesh subdivision approaches for final rendering. Before explaining mesh colors, we offer a critique of the various mapping approaches that are currently in use.

When 2D textures are used, the crucial step is defining a one-to-one mapping (modulo any periodicity) from 2D image space to the 3D surface. Even though there are methods to automate this procedure, in practice it usually requires manual intervention. In a production pipeline, significant amounts of time are required from both modelers and texture painters to create "good" maps. An inherent problem of this mapping is that, for almost all 3D models, mapping discontinuities are inevitable. These discontinuities are especially visible in displacement mapping and in level-of-detail approaches like MIP-mapping. Techniques that aim to hide these discontinuities violate the one-to-one mapping property and often break down beyond the first few detail levels.

Using 3D textures eliminates the 2D to 3D mapping procedure; on the other hand, it creates other challenges to ensure one-to-one mapping. In particular, models with sharp and thin edges or closely adjacent surfaces may require very high texture resolution to avoid color bleeding between separate components. These problems become even more prominent across MIP-map levels. Moreover, even the slightest changes in the textured 3D model may require regeneration of the map, which often degrades the texture quality.

Mesh colors, on the other hand, keep color information directly associated with the 3D surface geometry. This eliminates the problems inherent in 2D and 3D maps, while supporting most of their strong points. The key characteristics of mesh colors are:

*e-mail: cem@cemyuksel.com
†e-mail: keyser@cs.tamu.edu
‡e-mail: house@viz.tamu.edu

- The inherent parametrization of the model is used, so no mapping function is necessary,
- Without mapping, there are no mapping discontinuities,
- An intrinsic one-to-one correspondence is maintained between the 3D surface and the color data,
- MIP-mapping for level-of-detail filtering is supported
- Texture detail can be adjusted locally with no global effects,
- Models can be edited, after coloring, without resampling,
- The procedure is compatible with the current real-time graphics pipeline.

Since mesh colors can be implemented to satisfy all of the above criteria with high performance and low memory use, the approach is ideal for many high-end applications like 3D texture painting, and for storing precomputed data, such as ambient occlusion, light maps, and global illumination, on a 3D surface.

In the next section we present an overview of related previous work. We explain the details of the mesh color structure in Section 3 and filtering the mesh colors in Section 4. Our implementations for both offline and real-time systems are described in Section 5, along with results. We provide a discussion of the advantages and limitations of mesh colors in Section 6, and conclude in Section 7.

2 Related Work

Several techniques have been proposed for automatic planar parametrization of 3D surfaces for mapping 2D textures [Ma and Lin 1988; Bennis et al. 1991; Maillot et al. 1993; Zhang et al. 2005; Lévy and Mallet 1998; Hunter and Cohen 2000; Piponi and Borshukov 2000; Haker et al. 2000; Lévy et al. 2002; Sheffer and Hart 2002; Zhang et al. 2005]. Some methods guarantee one-to-one mapping [Hormann and Greiner 1999; Sheffer and de Sturler 2000; Sander et al. 2001; Floater 2003], and some permit user defined constraints [Lévy 2001; Desbrun et al. 2002; Kraevoy et al. 2003]. The general problem with these methods comes from the fact that arbitrary 3D surfaces cannot be mapped onto a plane without discontinuities, which appear as interpolation artifacts (*seams*) especially in MIP-map levels [Williams 1983]. These seams cannot be avoided without duplicating colors on region borders [Carr and Hart 2002; Purnomo et al. 2004; Carr et al. 2006], which breaks the one-to-one mapping property. Furthermore, even though there are methods that allocate more resolution to regions with finer texture detail [Sloan et al. 1998; Balmelli et al. 2002; Sander et al. 2002] and others that allow dynamic reparametrization as texture detail changes [Carr and Hart 2004; Igarashi and Cosgrove 2001], texture detail cannot be adjusted locally without changing the resolution of other parts of the surface. Moreover, these methods are very sensitive to underlying 3D model topology, and arbitrary model editing with face add/delete operations often requires regeneration of the parametrization and resampling of color values. Floater and Hormann [2005] and Sheffer et. al. [2006] provide detailed overviews.

Problems associated with parametrization can be avoided by eliminating the mapping and using a 3D structure to store the color data. Benson and Davis [2002] and DeBry et. al. [2002] suggested using an octree structure, which reduces the extreme memory requirement of 3D textures and permits different texture resolutions over different parts of the 3D model. However, depending on the shape of the 3D model, octree textures may require too many levels to avoid color bleeding across close surfaces, which may then still appear in MIP-map levels. Furthermore, real-time implementations of octree textures [Kniss et al. 2005; Lefebvre et al. 2005a; Lefohn et al. 2006] require an arbitrary number of dependent texture lookups that severely reduce the performance. An alternative structure is proposed by Lefebvre and Hoppe [2006], which uses a hash function to efficiently store color data. This structure requires only two texture lookups, but it does not allow non-uniform resolution over the 3D surface. In general, 3D data structures are very sensitive to underlying model geometry and even the slightest change may require resampling the color data. Therefore, even subdivision operations, which are often applied right before rendering, are not supported by these structures, and texture colors have to be assigned to the finest resolution of the 3D model.

Recently, many researchers have proposed hybrid methods to address some of these problems. Tarini et. al. [2004] proposed polycube maps, which define colors on the faces of multiple cubes stacked in a way to resemble the general shape of the 3D model, to avoid mapping discontinuities. Polycube maps are not generalized enough to represent any model, and require long fragment programs for real-time implementation. They also have a fixed resolution over the surface and require significant manual intervention for parametrization. Ray et. al. [2006] used a technique that automatically generates a similar structure for global parametrization of 3D models. Lefebvre and Dachsbacher [2007] used tile trees to store colors on square texture fragments that are kept in the leaf nodes of an octree structure. By limiting the octree subdivision level, they reduced the memory requirements of octree textures and improved their performance, but they share the other limitations of octree textures. A different approach proposed by Lefebvre et. al. [2005b] uses texture sprites: multiple small 2D texture elements placed over the 3D surface. This method is useful for creating high resolution textures from repetitive small fragments, but not suitable for applications that require one-to-one mapping.

Our approach resolves these problems by associating colors directly with a surface. Mesh colors behave like a 2D texture on a local area, but without the intrinsic problems of texture mapping.

3 Mesh Color Structure

We describe the mesh color structure assuming the underlying 3D model data is a triangular mesh. However, it is possible to extend the mesh colors approach to more general polygonal models, subdivision surfaces, NURBS, or other parameterized surfaces, as is discussed briefly at the end of this section.

3.1 Mesh Colors on Triangular Faces

The mesh colors method is basically an extension of vertex colors. As in standard vertex colors, each vertex is associated with a color sample. In addition, the mesh color structure has evenly spaced color samples defined along the edges and over the faces. The number of color samples on a face or an edge depends on the desired resolution in the local area on the mesh.

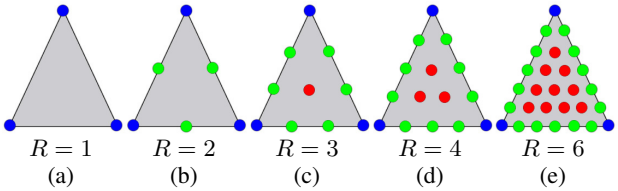


Figure 2: *Color positions on vertices (blue), edges (green), and face (red) for different resolutions (R).*

The lowest resolution mesh color map is a standard vertex color map as shown in Figure 2a. In this case, no color samples are defined on the edges or faces. When we move to one higher resolution, we also define a single color sample in the middle of each edge as in Figure 2b. Color samples on faces appear only in higher

resolutions (Figure 2c-e). As the resolution increases, we add more color samples on edges and faces. If R is the resolution, the numbers of color samples on each vertex, edge, and face are given by

$$\text{colors per vertex} = 1\,, \quad (1)$$

$$\text{colors per edge} = R-1\,, \quad (2)$$

$$\text{colors per face} = \frac{(R-1)(R-2)}{2}\,. \quad (3)$$

Note that color samples on edges and faces are evenly spaced in barycentric coordinates. Therefore, the 3D surface position of any color sample can be computed using barycentric coordinates based on the index of the color sample. For a face with resolution R, the positions $\mathbf{P}_{ij}$ in barycentric coordinates of color sample C_{ij} are

$$\mathbf{P}_{ij} = \left[\, \frac{i}{R},\ \frac{j}{R},\ 1-\frac{i+j}{R} \,\right]\,, \quad (4)$$

where i and j are integers such that $0 \le i \le R$ and $0 \le j \le R-i$. Note that color samples C_{00}, C_{R0}, and C_{0R} are on the vertices (*vertex colors*); color samples C_{0k}, C_{k0}, and $C_{k(R-k)}$ for $0 < k < R$ are on the edges (*edge colors*); and the rest of the color samples are on the face (*face colors*). This structure totally eliminates the need for keeping 3D coordinates and topology data for color samples.

Similarly, since color sample positions are based on their indices, for any given point on the face the indices of the nearest color samples can be computed from the barycentric coordinates of the point. The details of this procedure are given in Section 4.

Depending on the requirements of the implementation, mesh color samples can be embedded in the mesh data, such that each vertex, edge, and face would keep its own color samples. However, for many applications we want mesh colors to be usable in the current graphics pipeline. To do this, we need to decouple mesh and color data, similar to standard texture maps. This can be easily accomplished by storing all mesh colors in an array and keeping an index to the first color sample for each vertex, edge, and face. We explain the details of this implementation in Section 5.

3.2 Non-uniform Face Resolutions

One of the most important features of the mesh color structure is that it allows different faces of a model to have different resolutions. Using this property, color sample density can be concentrated on areas with high detail, achieving maximum texture detail with minimal memory consumption. Figure 3 shows an example of color sample distribution over faces with different resolutions.

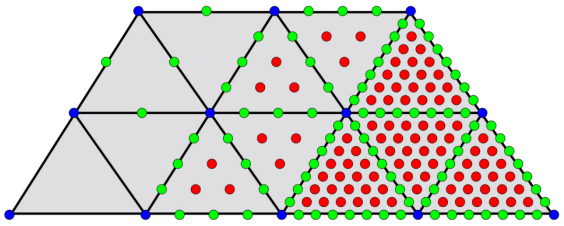


Figure 3: *Faces of a mesh with different resolutions.*

In the mesh color structure, each face keeps its resolution value, which defines the number of color samples on the face. The color samples that belong to the face (i.e. face colors) are not shared among neighboring faces, but vertex colors and edge colors are often used by multiple faces. Sharing vertex colors among a number of faces with different resolutions is easy, because we only keep a single color value per vertex regardless of the resolution. However, if two faces sharing a common edge have different resolutions, the number of color values and positions of these color values on the edge differ depending on which resolution we pick for the edge. It is straightforward to up-sample or down-sample along an edge, so this does not pose a fundamental problem. As a rule of thumb, we set the resolution of an edge to the highest resolution of its faces.

3.3 Editing Mesh Colored Geometry

A fundamental feature of the mesh color structure is that it does not depend on the geometry of the object. Therefore, changing the positions of the vertices has no effect on the mesh color structure.

On the other hand, mesh colors strictly depend on the topology. When operations like remeshing, which generates a whole new topology, is applied to a mesh colored surface, the mesh color structure needs to be regenerated by resampling the previous mesh colors. Note that this limitation is also shared by 2D texture mapping methods. Edge flips may require local resampling.

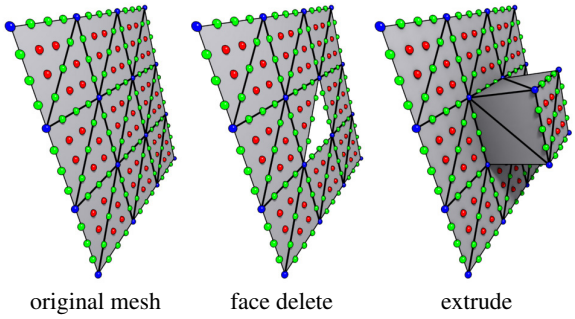


Figure 4: *Mesh colors can easily handle mesh editing operations.*

Mesh colors can easily support most other mesh editing operations without the need for resampling. Inserting or deleting faces can be trivially handled by creating or deleting extra color samples as needed. Extrusion operations are supported in a similar fashion by duplicating color samples along extruded boundaries. More color samples can always be added/removed as desired. Since color samples are directly associated with object primitives, none of these operations has a global effect.

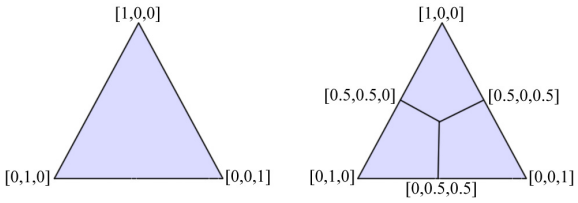
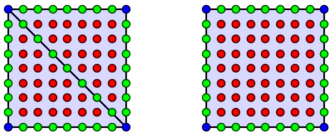


Figure 5: *Barycentric coordinates as texture coordinates before (left) and after (right) a face subdivision operation.*

Operations that divide faces, such as certain subdivisions, can be handled without resampling, if mesh and color data are decoupled. Effectively, a one-to-one mapping between the old and new faces is created. In this case, the new faces share the same color index with the parent face, the subdivided edges keep their previous indices, and new edges or vertices would not have any indices assigned. Moreover, barycentric coordinates should be redefined to match the ones before the subdivision (Figure 5). This is done by keeping barycentric coordinates as texture coordinates.

3.4 Non-triangular Meshes

The simplest solution for applying the mesh color definition to non-triangular surfaces is to simply triangulate the polygons (Figure 6a); this approach was used for the quadrilaterals in Figure 1. An alternate approach for quadrilaterals would be to define an indexing scheme as shown in Figure 6b. Instead of positioning samples based on barycentric coordinates, bilinear interpolation could be used.



(a) (b)

Figure 6: *Mesh colors on non-triangular geometry. (a) Representing polygons with triangles. (b) Defining mesh colors on quadrilaterals.*

The mesh color structure on NURBS and parametric surfaces could be handled similar to quadrilaterals. Note that this procedure is different from assigning a separate 2D image to each patch; since the colors along connections of neighboring patches are shared, we guarantee continuity across edges.

Subdivision surfaces can be handled similar to polygonal meshes, and resampling the mesh color map for each level is not required as long as the subdivision operation is based on face division rather than a complete remeshing. Note that the Catmull-Clark subdivision scheme and its extensions, which are the most commonly used subdivision techniques, satisfy this condition.

4 Filtering

For evaluating the color value at any point on the surface or an area that corresponds to a screen pixel, we need a reconstruction filter. In this section we describe how standard texture filtering operations can be used with mesh colors.

4.1 Two-dimensional Filtering

The filtering operation begins by finding the closest color samples to a given point on the surface. Since locations of the color samples are based on their indices, for any given point on a face the indices of the nearest color values can be computed from the barycentric coordinate $\mathbf{P}$ of the point by multiplying it with the face resolution R. To find the nearest color samples and the weights with which to linearly combine them, we first compute two values: the integer portion $\mathbf{B} = [i, j, k] = \lfloor R\,\mathbf{P} \rfloor$, and the fractional part $\mathbf{w} = [\mathbf{w}_i, \mathbf{w}_j, \mathbf{w}_k] = R\,\mathbf{P} - \mathbf{B}$.

There are three cases. If $\mathbf{w} = 0$, we are at the sample point $\mathbf{B}$ (and thus that is the color). If $\mathbf{w}_i + \mathbf{w}_j + \mathbf{w}_k = 1$, then the nearest color

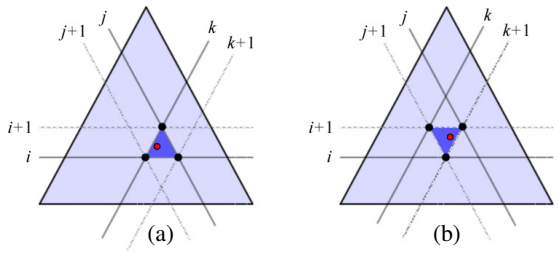
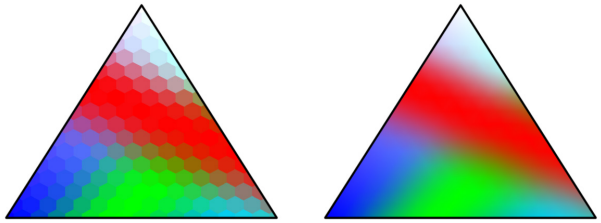


Figure 7: *Nearest color values of the red point.*



(a) (b)

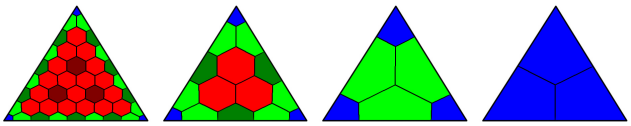
Figure 8: *(a) Nearest and (b) linear filtering.*

values for the point are $C_{(i+1)j}$, $C_{i(j+1)}$, and C_{ij} respectively as in Figure 7a. The weights for those color values are $\mathbf{w}$. In the third case, if $\mathbf{w}_x + \mathbf{w}_y + \mathbf{w}_z = 2$, the nearest color values are $C_{i(j+1)}$, $C_{(i+1)j}$, and $C_{(i+1)(j+1)}$; the weights are $\mathbf{w}' = [1, 1, 1] - \mathbf{w}$ as in Figure 7b.

If we directly use the color of the sample with the highest weight (*nearest color filtering*), face colors form a hexagonal pattern as in Figure 8a. By blending the colors using the weights we achieve a linear filtering (Figure 8b).

4.2 MIP-map Filtering

For integrating the filter over an area, we use MIP-mapping just as in standard texture maps. The only difference is the way that the MIP-map is generated and stored. Since we allow faces to have different resolutions, we generate a separate MIP-map for each face, edge, and vertex (the resolution of an edge or a vertex is the finest resolution of its adjacent faces).



level-3 level-2 level-1 level-0

Figure 9: *MIP-map levels for a face, colored according to the nearest mesh color: vertex color (blue), edge color (green), or face color (red). Darker colors indicate that the color position also appears in the lower level.*

For $n \geq 0$, let $R = 2^n$ be the resolution of a face. This face will have $n+1$ MIP-map levels, such that level-0 is the vertex color level and level-n corresponds to the original resolution. Note that no face colors appear in level-0 and level-1. Figure 9 shows the MIP-map levels of a face with $R = 8$. Color positions that reappear in the lower level are shaded darker. Starting from the highest resolution, each face color in the lower level is set as the color value at the same position of the higher level plus half of the sum of the surrounding six colors, divided by 4 for normalization:

$$C'_{ij} = \frac{1}{4}\left(C_{(2i)(2j)} + \frac{1}{2}\left(\begin{array}{ll} & C_{(2i)(2j-1)} \\ + & C_{(2i)(2j+1)} \\ + & C_{(2i-1)(2j)} \\ + & C_{(2i-1)(2j+1)} \\ + & C_{(2i+1)(2j-1)} \\ + & C_{(2i+1)(2j)} \end{array} \right) \right) . \quad (5)$$

Similarly, an edge color of the lower level is computed from the edge color at the higher level and six adjacent colors: two on the edge and two from each adjacent face. Vertex colors are computed from the vertex color in the higher level and the colors of the edges

(of equal resolution) sharing the vertex. For a vertex $\mathbf{v}$ of valence v, the new vertex color is given by

$$C'_{\mathbf{v}} = \frac{2}{2+v}\left(C_{\mathbf{v}} + \frac{1}{2}\left(\sum_{i=1}^{v} C_{\mathbf{e}_i}\right)\right) , \qquad (6)$$

where $C_{\mathbf{e}_i}$ is the nearest edge color at the higher level for adjacent edge i.

When two neighboring faces have different resolutions, MIP-map generation begins with the face with the higher resolution and the colors on the other face are ignored until the two faces reach the same level. Unlike MIP-maps of 2D textures, mesh colors are guaranteed to generate a continuous coloring function on each level.

One way to store the MIP-map data is to keep each level in separate arrays similar to standard texture maps. However, when the face resolutions are non-uniform, this would require keeping multiple indices for each level of each vertex, edge, and face. We avoid this by storing all layers within the same arrays, such that colors of a vertex for different MIP-map levels appear one after the other from level-0 to the highest level of the vertex. Edge and face colors are stored similarly. In this form, we only keep a single color index, which points to the beginning of the MIP-map data for the vertex, edge, or face. Note that we do not need to store resolutions of edges or vertices, we only need to know the resolution of the face to access the correct colors from the arrays.

4.3 Anisotropic Filtering

As when using standard texture maps, anisotropic filtering is important when computing the color value corresponding to a pixel area. One might expect anisotropic filtering to be more important for mesh colors when the object has triangles with significantly different edge lengths. However, for filtering operations, screen space lengths of edges are more important than the actual edge lengths.

Like standard anisotropic filtering, we combine multiple samples from a single MIP-map level, based on screen-space derivatives of the barycentric coordinates. The only difference is that all MIP-map sample positions of mesh colors are kept within the sampled face area. Because of this, we can only access the colors defined in the MIP-map levels of the face and its edges and vertices. Thus, anisotropic sampling across multiple faces would require finding the neighboring faces from the mesh topology and executing the filtering operations on these faces as well. This situation occurs when a pixel on the screen is covered by multiple faces. In a high quality renderer this should not be much of an issue, since the correct color of the pixel is automatically retrieved by sub-pixel operations.

5 Implementations and Results

Mesh colors are very simple to implement, since no parameter tweaking or precomputation is necessary. The only parameter is the resolution of the faces. Due to this simplicity, the mesh color structure can be easily used in current offline and real-time graphics systems.

5.1 Painting Mesh Colors

3D painting systems are straightforward to implement with mesh colors, in contrast to texturing methods that require mapping. No mapping algorithms need to be implemented and the intrinsic one-to-one correspondence property of mesh colors eliminates book-keeping operations necessary to ensure consistency of duplicated color data and visual continuity across mapping discontinuities.

Figure 10: *Low resolution head model in our painting interface.*

We have implemented such a 3D painting system. In addition to the simplicity of its implementation, our painting system also provides additional features inherently supported by mesh colors. Users can begin painting on any 3D model without going through the trouble of defining appropriate texture coordinates for their task or selecting a final texture resolution. Exploiting the ability of mesh colors to provide non-uniform face resolution, our painting interface allows on-the-fly adjustment of local resolution on any part of the model. A resolution change on one face has no effect on the other faces.

In our system, we separate mesh and color data by storing all mesh colors in an array and keeping an index into this array for each vertex, edge, and face. For edges and faces, these indices point to the first color sample, and the rest of the samples are stored consecutively in the array. This separation permits mesh editing operations even after the object is painted. Note that mesh editing operations that only change the vertex positions are inherently supported by mesh colors, and we have this separation only for supporting topology changes after painting. Each face of the mesh keeps its vertex, edge, and face color indices, and this information is stored as per-face data, which goes through the geometry pipeline. Therefore, even if the vertex, edge or face indices change during mesh editing operations, correct colors indices can be accessed. Using this approach, existing mesh editing tools need not be modified for mesh color support. In addition to editing operations, high resolution geometric detail can be added at render time via subdivision operations or per-pixel displacement mapping.

Figure 10 shows the low resolution head model in Figure 1 in our painting system. The model in the painting interface has far fewer polygons than the final rendered model, which is subdivided after the painting operation. Figure 11 shows an alien model painted using our system. Note that the geometry detail does not correspond to the detail in the texture.

5.2 Offline Rendering

Using mesh colors in offline rendering does not require any modifications to the rendering system other than a special shader. Our offline shader uses the per-face data stored in the mesh structure to access the colors used by the face being shaded. It retrieves the barycentric coordinates from texture coordinates, and applies the specified filtering operation as explained in Section 4.

Figure 12 provides a qualitative comparison of different filtering techniques. Staircase artifacts of nearest-filtering can be hidden by linear filtering. The under-sampling artifacts of linear filtering near the silhouette of the object are resolved by MIP-map filtering, and the excessive blurring of MIP-map filtering at the same areas is overcome by anisotropic filtering.

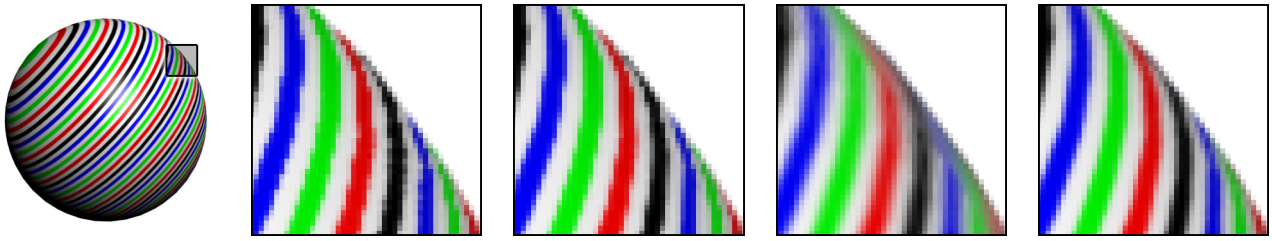


Figure 12: *Enlarged fragments of the sphere image on the left for qualitative comparison of different mesh color filtering techniques.*

Figure 11: *An alien model with over 200 thousand polygons painted using mesh colors. Local texture detail does not correspond to model detail. (Modeled and painted by Murat Afsar)*

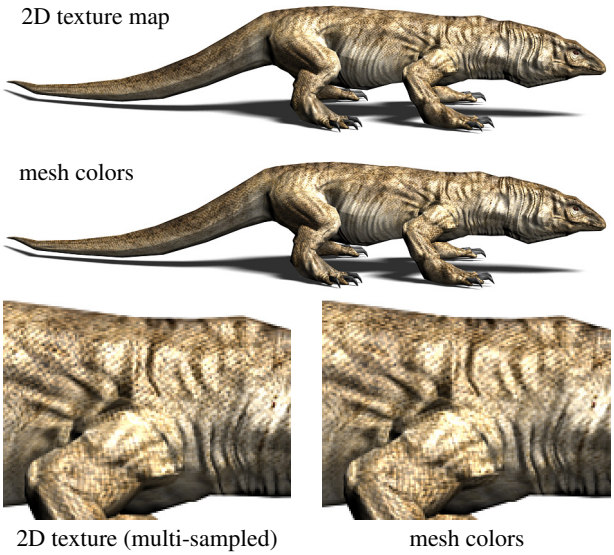


Figure 13: *A low resolution lizard model with normal maps. Images rendered with a 1024x1024 2D texture using sub-pixel sampling, and with mesh colors (about 911x910 samples) are of equal quality. Mesh colors were converted from the 2D texture image. (Modeled and painted by Murat Afsar)*

Figure 13 shows renderings of a lizard model, allowing direct qualitative comparison between the model rendered with texture mapping, using sub-pixel multi-sampling, and mesh colors (converted from the texture map) rendered with anisotropic filtering. In spite of the conversion, mesh colors maintain high quality. Furthermore, mesh colors require fewer color samples (approximately 21% fewer for the example) than the standard texture map, since samples are not "wasted" on buffering around seams.

5.3 Real-time Rendering

For high quality real-time rendering, we store all mesh colors along with the precomputed MIP-map values in a 2D texture. In fact, a 1D texture would be more appropriate for mesh colors, but the current graphics hardware limits 1D texture size to 4096 colors. We convert mesh color indices of the vertices, edges, and faces to 2D texture coordinates for this mesh color texture.

We wrote a custom fragment shader, which shares the same shader code with the offline shader, for mesh color texture lookups. The only difference from the offline version is the way we send per-face data to the fragment shader. While drawing a face, we send texture coordinates of its vertices, edges, and the face along with the native resolution of the face, which limits the maximum MIP-map lookup level. Since we cannot send per-face data, we send the same data for all vertices of the face.

In Table 1, we compare our mesh colors method directly to hardware optimized 2D textures. Based on the results shown, we see that the performance of the mesh color shader is comparable to traditional (and heavily hardware-accelerated) 2D texture mapping, both for high and low resolution models. As would be expected, without direct hardware support, more complex filtering calculations do degrade performance.

In our experience with the mesh color fragment shader, we found two areas where minor changes in future graphics hardware could significantly improve the performance of mesh colors. First, arbitrarily large sizes for 1D textures would allow far simpler implementation. Second, the amount of information passed would be greatly reduced by sending per-face data (instead of interpolated per-vertex data), which permits 32-bit integers, directly to the fragment shader without interpolation.

6 Discussion

Mesh colors can be considered as an extension of vertex colors, but unlike standard vertex colors, mesh colors allow arbitrarily high resolution textures. This is an extremely important feature, since texture mapping is used not only to specify the colors on a 3D surface, but also to provide the illusion of detail where geometric detail is not sufficient.

Perhaps the most important advantage of mesh colors is that each color sample is associated with a well-defined region of the surface.

Table 1: *Mesh colors vs. standard 2D texture performance*

	head-low	head-high	alien
face count	3,106	50,576	218,806
color count	530,498	530,498	9,303,241
2D texture	3938 fps	2597 fps	337 fps
Nearest	2567 fps	1147 fps	273 fps
Linear	2076 fps	862 fps	247 fps
MIP-map	991 fps	376 fps	180 fps
Anisotropic	452 fps	152 fps	109 fps

Configuration: 2.14 GHz Core 2 Duo with GeForce 8800 graphics card

This one-to-one correspondence between the texture space and the model eliminates the extra memory (and resulting possible inconsistencies) required to hide discontinuities in most traditional maps. Techniques that map 3D calculations directly to the surface, such as radiosity [Goral et al. 1984], can attain high resolution results without the need for tessellation. Furthermore, mesh colors should easily support new image processing algorithms that are aware of the 3D model geometry for texture synthesis and editing.

Color sample positions of mesh colors are determined by a simple operation, which resembles a tessellation procedure that individually subdivides the faces. In fact, using vertex colors on a mesh that is tessellated a number of times such that the geometric detail matches the texture detail may give the same fine resolution as mesh colors. However, vertex colors on this tesselated version of the model cannot be used with MIP-mapping. More importantly, the mesh color structure is not a real subdivision and it only stores color information, where a tessellated model would also keep unnecessary vertex positions and extra topology data. In other words, a tesselated model would have $4^n f$ faces, where f is the number of faces before tessellation and n is the number of tessellation operations. Therefore, after only two tessellation operations, about 94% of the faces would be wasted to this overhead. Thus, using only vertex colors on a highly tesselated model severely degrades the performance and cannot be used for real-time applications.

Even though mesh colors introduce a new paradigm of keeping high resolution color data directly on 3D surfaces, the approach fits within the current graphics production pipeline. Mesh colors can be used with many different existing 3D data structures. Furthermore, by separating mesh and color data, existing mesh editing and subdivision operations can be used without modification.

One limitation of using MIP-maps with mesh colors is that the resolution of the mesh color structure cannot be lower than that of the vertex colors, whereas standard 2D textures can have MIP-map levels defined down to a single pixel. For the following reasons, we would argue that this is not a serious limitation of mesh colors compared to traditional texture maps. First, most traditional one-to-one mappings are only good for a few MIP-map levels, since the mappings must be artificially padded to prevent color mixing across discontinuities in the map, and this padding must be limited in extent. Second, MIP-map levels below vertex colors are only called for when the size of a face projected into screen space is smaller than a pixel. Offline rendering systems are very good at sub-pixel operations and these possible inaccuracies due to the vertex color limitation would not be perceivable in the final image. In real-time systems, this condition means that the geometric resolution is already too high to be reconstructed correctly in screen space, which presents problems beyond just texture filtering. Third, a common source of such small polygons is subdivision, but the limiting factor of the mesh color structure is the polygons of the *original* (non-subdivided) model. Thus, MIP-mapping actually can occur effectively across multiple polygons that are derived from a single base polygon.

One difficulty associated with mesh colors occurs when two neighboring faces have different resolutions and detail is desired near the edge of the higher resolution polygon. This is a general sampling problem for *all* texturing and coloring methods. In the case of mesh colors with non-uniform face resolutions, we assume that the user would pick a resolution for the desired coloring task that is high enough to ensure that there will not be any visible discontinuities between faces with different resolutions after filtering. Formally speaking, no discontinuities appear on an edge connecting two faces with different resolutions when the edge colors of the higher resolution can be accurately represented by linearly interpolating the edge colors of the lower resolution.

As long as anisotropic filtering is used, faces with significantly different edge lengths do not cause any visual artifacts. On the other hand, since each face of a mesh color structure has a single resolution, the resolution to achieve the desired color sample density along a longer edge of the face may cause a shorter edge to have more color samples than necessary. Therefore, over-sampling may occur along short edges of elongated faces, possibly wasting memory.

Mesh colors are not a universal texturing solution. Periodic texture tiling is not supported. Because our method guarantees continuity across edges, it is not very suitable for applications where discontinuities in the texture maps are desired. In such cases, objects need to duplicate edges (i.e. topologically cut the model) where such discontinuities are desired.

7 Conclusion

We have introduced the concept of mesh colors, a simple structure for keeping color (texture) information of arbitrary polygonal models. The approach presents an attractive alternative to traditional texture mapping, eliminating the intrinsic problems created by the need for a map from texture to model space. The color sample positions of mesh colors are directly defined on the 3D surface, so there is no need for a mapping or parametrization. Because a direct association between colors and geometry is used, mesh colors guarantee a one-to-one correspondence between the color data and the 3D surface without any discontinuities. Furthermore, the mesh colors method allows local adjustment of texture resolution while defining colors, and permits 3D model editing even after coloring. Mesh colors can be used with MIP-maps for fast and high quality texture filtering, and they are compatible with the current graphics pipeline, achieving high real-time performance with low memory requirements on consumer graphics hardware. With all these properties, mesh colors are the ideal choice for applications such as texture painting and storing of precomputed data (e.g. ambient occlusion) to be used in both real-time and offline environments.

Acknowledgements

We would especially like to thank Murat Afsar for his models and mesh color paintings as well as valuable feedback for enhancing our painting system.

References

BALMELLI, L., TAUBIN, G., AND BERNARDINI, F. 2002. Space-optimized texture maps. *Computer Graphics Forum (Proceedings of Eurographics 2006) 21*, 3, 411–420.

BENNIS, C., VÉZIEN, J.-M., AND IGLÉSIAS, G. 1991. Piecewise surface flattening for non-distorted texture mapping. *Proc. SIGGRAPH '91, ACM SIGGRAPH Comp. Graph. 25*, 4, 237–246.

BENSON, D., AND DAVIS, J. 2002. Octree textures. *Proc. SIGGRAPH '02, ACM Transactions on Graphics 21*, 3, 785–790.

CARR, N. A., AND HART, J. C. 2002. Meshed atlases for real-time procedural solid texturing. *ACM Trans. Graph. 21*, 2, 106–131.

CARR, N. A., AND HART, J. C. 2004. Painting detail. *Proc. SIGGRAPH '04, ACM Trans. on Graphics 23*, 3, 845–852.

CARR, N. A., HOBEROCK, J., CRANE, K., AND HART, J. C. 2006. Rectangular multi-chart geometry images. In *SGP '06: Proc. Eurographics Symp. on Geometry Processing*, 181–190.

DEBRY, D., GIBBS, J., PETTY, D. D., AND ROBINS, N. 2002. Painting and rendering textures on unparameterized models. *Proc. SIGGRAPH '02, ACM Trans. on Graphics 21*, 3, 763–768.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum (Proceedings of Eurographics 2002) 21*, 209–218.

FLOATER, M. S., AND HORMANN, K. 2005. Surface parameterization: a tutorial and survey. In *Advances in multiresolution for geometric modelling*, 157–186.

FLOATER, M. S. 2003. Mean value coordinates. *Computer Aided Geometric Design 20*, 1, 19–27.

GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. 1984. Modeling the interaction of light between diffuse surfaces. *Proceedings of SIGGRAPH '84, ACM SIGGRAPH Computer Graphics 18*, 3, 213–222.

HAKER, S., ANGENENT, S., TANNENBAUM, A., KIKINIS, R., SAPIRO, G., AND HALLE, M. 2000. Conformal surface parameterization for texture mapping. *IEEE TVCG 6*, 2, 181–189.

HORMANN, K., AND GREINER, G. 1999. MIPS: An efficient global parametrization method. In *Curve and Surface Design Conference Proceedings 1999*. 153–162.

HUNTER, A., AND COHEN, J. D. 2000. Uniform frequency images: adding geometry to images to produce space-efficient textures. In *Proc. IEEE Visualization '00*, 243–250.

IGARASHI, T., AND COSGROVE, D. 2001. Adaptive unwrapping for interactive texture painting. In *I3D '01: Proceedings of the 2001 Symposium on Interactive 3D Graphics*, 209–216.

KNISS, J., LEFOHN, A., STRZODKA, R., SENGUPTA, S., AND OWENS, J. D. 2005. Octree textures on graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, 16.

KRAEVOY, V., SHEFFER, A., AND GOTSMAN, C. 2003. Matchmaker: constructing constrained texture maps. *Proc. SIGGRAPH '03, ACM Transactions on Graphics 22*, 3, 326–333.

LEFEBVRE, S., AND DACHSBACHER, C. 2007. Tiletrees. In *I3D '07*, ACM, New York, NY, USA, 25–31.

LEFEBVRE, S., AND HOPPE, H. 2006. Perfect spatial hashing. *Proc. SIGGRAPH '06, ACM Trans. on Graphics 25*, 3, 579–588.

LEFEBVRE, S., HORNUS, S., AND NEYRET, F. 2005. *GPU Gems 2*. Addison Wesley, ch. Octree Textures on the GPU, 595–613.

LEFEBVRE, S., HORNUS, S., AND NEYRET, F. 2005. Texture sprites: Texture elements splatted on surfaces. In *I3D '05: Proc. Symposium on Interactive 3D Graphics and Games*, 163–170.

LEFOHN, A. E., SENGUPTA, S., KNISS, J., STRZODKA, R., AND OWENS, J. D. 2006. Glift: Generic, efficient, random-access gpu data structures. *ACM Trans. on Graphics 25*, 1, 60–99.

LÉVY, B., AND MALLET, J.-L. 1998. Non-distorted texture mapping for sheared triangulated meshes. In *Proceedings of SIGGRAPH '98*, 343–352.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *Proc. SIGGRAPH '02, ACM TOG 21*, 3, 362–371.

LÉVY, B. 2001. Constrained texture mapping for polygonal meshes. In *Proceedings of SIGGRAPH '01*, 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*, 27–34.

PIPONI, D., AND BORSHUKOV, G. 2000. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *Proceedings of SIGGRAPH '00*, 471–478.

PURNOMO, B., COHEN, J. D., AND KUMAR, S. 2004. Seamless texture atlases. In *SGP '04: Proc. Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 65–74.

RAY, N., LI, W. C., LÉVY, B., SHEFFER, A., AND ALLIEZ, P. 2006. Periodic global parameterization. *ACM Transactions on Graphics (TOG) 25*, 4, 1460–1485.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *Proceedings of SIGGRAPH '01*, 409–416.

SANDER, P. V., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2002. Signal-specialized parametrization. In *EGRW '02: Proc. Eurographics Workshop on Rendering*, 87–98.

SHEFFER, A., AND DE STURLER, E. 2000. Surface parameterization for meshing by triangulation flattening. In *Proceedings of the 9th International Meshing Roundtable*, 161–172.

SHEFFER, A., AND HART, J. C. 2002. Seamster: Inconspicuous low-distortion texture seam layout. In *IEEE VIS '02*, 291–298.

SHEFFER, A., PRAUN, E., AND ROSE, K. 2006. Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision 2*, 2, 105–171.

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

TARINI, M., HORMANN, K., CIGNONI, P., AND MONTANI, C. 2004. Polycube-maps. *Proc. SIGGRAPH '04, ACM Transactions on Graphics (TOG) 23*, 3, 853–860.

WILLIAMS, L. 1983. Pyramidal parametrics. *Proc. SIGGRAPH '83, ACM SIGGRAPH Computer Graphics 17*, 3, 1–11.

ZHANG, E., MISCHAIKOW, K., AND TURK, G. 2005. Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics (TOG) 24*, 1, 1–27.