Hardware Accelerated Mesh Colors

Cem Yuksel
University of Utah

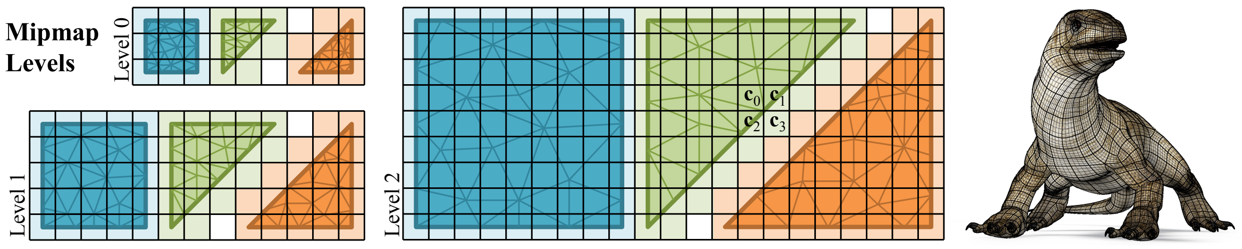

Figure 1: *2D texture layout of mesh colors, and mesh colors applied on lower-resolution tessellation of a high-resolution model.*

Concepts: •**Computing methodologies** → *Texturing;*

Texture mapping arbitrary polygonal meshes has been an important problem in computer graphics. In particular, when one-to-one mapping is desired, such that the same texture pattern is not repeated over a surface, seams are unavoidable. The process of hiding filtering artifacts due to these seams involves additional manual effort onto already labor intensive process of specifying texture coordinates. Furthermore, these seams also substantially limit the number of mipmap levels that can be used without visible filtering artifacts.

Mesh colors [Yuksel et al. 2010] was developed for completely eliminating the fundamental problems of texture mapping. Using the existing topology of a polygonal mesh, mesh colors allow generating detailed textures on arbitrary polygonal meshes without having to specify mapping coordinates. Also, mesh colors totally avoid filtering artifacts of seams by placing color samples exactly on the vertices and the edges of the polygonal mesh. Moreover, mipmap levels generated from mesh colors are guaranteed to be correct and they allow pre-filtering all the way down to vertex colors.

On the other hand, the existing texture filtering hardware we have on commercial GPUs are not designed to perform texture filtering operations of mesh colors. While it is possible to implement mesh color filtering using GPU shaders, this introduces additional code complexity. More importantly, it is known that software texture filtering operations can be up to an order of magnitude slower than hardware implementations.

In this work we use the existing texture filtering hardware available on commercial GPUs for performing a portion of the filtering operations needed for mesh colors. We also identify a few relatively minor modifications on texture filtering hardware that would allow fully hardware accelerated mesh color filtering.

Since the existing texture filtering hardware is designed for 2D (as well as 1D and 3D) textures, we need to convert the mesh color data into a 2D texture. This is extremely easy to do by defining texture coordinates such that vertices are mapped to the centers of some texels, as shown in Figure 1. This way, we can achieve bilinear texture filtering on hardware by copying mesh colors onto a 2D texture, where vertex and edge colors are duplicated as needed. The number of duplicated colors can be minimized by defining mesh colors using a lower-resolution version of the mesh than the tessellated mesh used for rendering, as explained in Yuksel et al. [2010].

The difficulty, however, is trilinear filtering using mipmap levels. A naïve mapping, as explained above, would require specifying different texture coordinates for each mipmap level, which is infeasible. We solve this problem by splitting non-normalized texture coordinates $\mathbf{u} \in [0, S_{0,1}]^2$, where S_0 and S_1 are the texture resolutions (number of pixels) in each dimension, into two components as $\mathbf{u} = \mathbf{u}_0 + \mathbf{u}_\delta$. Let ℓ be the mipmap level, such that $\ell = 0$ is the lowest resolution mipmap level. The texture coordinate for level ℓ is computed as $\mathbf{u}_\ell = 2^\ell \mathbf{u}_0 + \mathbf{u}_\delta$. This way, we only need to store a 4D texture coordinate ($\mathbf{u}_0$ and $\mathbf{u}_\delta$) per vertex. For trilinear filtering, we perform two hardware accelerated bilinear filtering operations using $\mathbf{u}_\ell$ and $\mathbf{u}_{\ell+1}$, and linearly interpolate the result.

While generating the 2D textures, care must be taken for triangle edges that are placed diagonally on the texture map. Bilinear filtering near a diagonal edge would use two texels $\mathbf{c}_1$ and $\mathbf{c}_2$ that correspond to the two colors along the edge, a texel inside the triangle c_0, and a texel outside of the triangle $\mathbf{c}_3$ (Figure 1). The color $\mathbf{c}_3$ must be set as $\mathbf{c}_3 = \mathbf{c}_1 + \mathbf{c}_2 - \mathbf{c}_0$ to make sure that the bilinearly interpolated color along the edge would be a function of $\mathbf{c}_1$ and $\mathbf{c}_2$ only for avoiding potentially visible seams. As a result, when using unsigned color channels, we must enforce $\mathbf{c}_0 \leq \mathbf{c}_1 + \mathbf{c}_2$.

Our trilinear operation only uses two texture lookups and it can also handle anisotropic filtering. However, anisotropic filtering across seams can produce incorrect results. For producing correct filtering results, the texture filtering hardware must avoid barycentric extrapolation of the texture coordinates and only sample the part of the texture that corresponds to the shaded triangle.

For handling trilinear filtering with a single texture lookup, two relatively minor hardware modifications are necessary. The first one is a 2D texture lookup operation using 4D coordinates to compute a different 2D texture coordinate per mipmap level, as explained above. Secondly, the resolution of our mipmap level ℓ is more than half of the resolution of mipmap level $\ell + 1$ in each dimension, which is not supported by current hardware. Hardware extensions for these two relatively minor features would permit hardware accelerated trilinear filtering of mesh colors with only a single texture lookup, thereby eliminating any performance penalty for using mesh colors as opposed to standard 2D textures.

References

YUKSEL, C., KEYSER, J., AND HOUSE, D. H. 2010. Mesh colors. *ACM Transactions on Graphics 29*, 2, 15:1–15:11.

I3D 2016 Posters, February 27–28 2016, Redmond, WA, USA
ISBN: 978-1-4503-4043-4/16/02
DOI: http://dx.doi.org/10.1145/2856400.2876017