Project Topic: Mesh Color Texture

VANNVATTHANA NORNG, Télécom Paris

This report gives an overview and my implementation for the article "Mesh Color Textures" [Yuksel 2017], where the mesh color [C. Yuksel 2010] data are transformed into a format that is suitable for GPU for texture filtering.

Additional Key Words and Phrases: Mesh colors, texture mapping, texture filtering

1 SUMMARY OF THE PAPER

Texture mapping has been a go to way in computer graphics for obtaining the mesh rendering result with high details on the surface. Normally, in order to render a mesh with details, two components is needed : a plain mesh that contains the geometry information of the object, and a texture map. Both of them are created in two different processes, and some mapping techniques are needed so that the final render of the object looks plausible. This is the reason why texture mapping causes high manual labor cost, and it can also cause some artifact around the seams.

. The previous article to the mesh color texure presents a new method called Mesh Color [C. Yuksel 2010], an alternative way to the texture mapping targeting real-time graphics applications. As the name suggest, mesh color stores color data directly on the mesh surface itself (vertices, edges and faces). This eliminates the need for parametrization from texture space to mesh space, and also avoid the the discontinuities along the seams as color values are directly on the edges. After introducing a new way of representing 3D objects with details, the author presents Mesh Color Textures [Yuksel 2017] in order to convert the Mesh Color data into a data that resemble 2D textures, but are able to contain those mesh color values.

1.1 Mesh Color

The color positions of mesh color [C. Yuksel 2010] are evenly distributed along the edges and inside the face of a triangle or a quadrilateral as shown in figure 1. As a result, there is no additional data structure needed to handle the color data. When the model is being edited, the texture painting also happens at the same time because the color values moves according to the face.

. While the advantages of Mesh Color make it suitable for a production pipeline that use 3D painting, it needs a specific shader that only handles its data structure. Moreover, the highest mipmap level of mesh color contains only the vertex colors of the mesh which is a limitation for this method.

This report is submitted as a part of project for Advanced 3D Computer Graphics (IMA904/IG3DA), Telecom Paris.

The original work is introduced by [Culler et al. 2004].

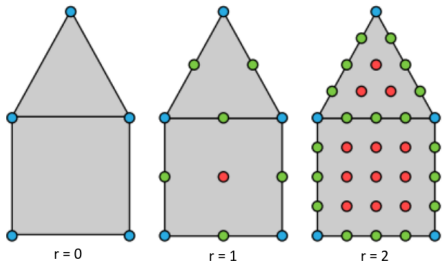

Fig. 1. Mesh Colors for triangular and quad faces with different resolution $R = 2^r - 1$ (R color values are stored in one edge)

1.2 Mesh Color Textures

Mesh Color Texture [Yuksel 2017] provides the solution in order to map the mesh color values into 2D texture-like where it can be used by texture filtering hardware on GPU.

. To generate 2D texture from mesh color values, start by assigning the coordinates of vertices of each face separately on the texture map to form a layout. For the triangular faces, pair every two of them into a rectangle, and leave 2 texel gaps between them so that we can interpolate the color for using in bilinear filtering. If the two triangles share the same edge at diagonal, join them together without leaving any gaps (figure 2 right). Make sure that every edge of each face lies vertically, horizontally and diagonally on the texture layout (figure 2).

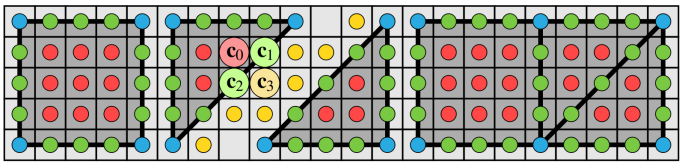

Fig. 2. Texture Layout of mesh colors

1.2.1 Bilinear Filtering. For bilinear filtering, it is enough to map interpolate the texel gap between the rectangle form by two triangles that do not share any edge (figure 2 middle). The texel c_3 can be computed as follow

$$c_3 = c_1 + c_2 - c_0 \tag{1}$$

In case where one of the RGB values of c_3 is greater than 1 or smaller than 0, we just need to update c_1, c_2 and c_0 with

$$c_1 = c_1 + \delta$$
$$c_2 = c_2 + \delta$$
$$c_0 = c_0 - \delta$$

where $\delta = -c_3/3$ if < 0 and $\delta = (1 - c_3)/3$ if > 1. After update them for a few iteration, it will converge.

1.2.2 Trilinear Filtering with mipmaps. For generating mipmap, the paper assume that the resolutions for mesh color are power of 2 minus one ($R = 2^r - 1$). Here, it provides a solution to be able to generate 2D textures at different mipmap levels using the so-called 4D texture coordinate representation.

. The texture coordinates for mipmap level l, u_l can be computing as follow

$$u_l = u_s/2^l + u_\delta \tag{2}$$

where u_s is a 2D scalable coordinate representing the portion of the texture coordinates that changes for each mipmap level, and u_δ is the constant offset for every mipmap level. Make sure that u_s is always an even multiple of 2^r or $R + 1$

The texture layout of u_s and u_δ can be computed using the algorithm 1 below.

Algorithm 1: Texture layout generation with 4D coordinates.

```
Form rectangles from faces.
Sort rectangles from large to small.
u_s ← [0 0]
u_δ ← [0.5 0.5]
foreach rectangle do
    if reached end of row then
        Find next row
        u_δ ← u_δ + [0 b]
        u_s ← next row starting position −u_δ
    end
    Make sure that u_s is an even multiple of 2^{r_i}.
    Place rectangle using the four texture coordinates:
        u_s, u_δ
        u_s + [0 2^{r_i}], u_δ
        u_s + [2^{r_i} 0], u_δ
        u_s + [2^{r_i} 2^{r_i}], u_δ
    u_{s,x} ← u_{δ,x} + 2^{r_i}
    u_{δ,x} ← u_{s,x} + X_i + b
end
```

Fig. 3. Texture layout for 4D coordinate systems

2 IMPLEMENTATION AND RESULTS

For the implementation, I used the skeleton code from the course *X-INF584 : Image Synthesis* [Boubekeur 2023], since it already provides the necessary component for rendering OFF mesh files. Since OFF only contains the vertex cordinates and face data, I need to assign Mesh color values for my mesh by myself. I planned to write a function for reading the OBJ file format with MTL for its texture, but I was not able to render it properly (the faces were messed up). For the mesh, I used the facelowpolygon.off which has 100 vertices and 192 triangular faces. I assign each face with mesh color of resolution $R = 2^3 - 1 = 7$, with $(R, G, B) = (0, 0, 0.7)$ for vertices, $(R, G, B) = (0, 0.7, 0)$ for edges, and $(R, G, B) = (0.7, 0, 0)$ for the face. Unfortunately, I was only able to integrate the vertex color data into the rendering pipeline, giving the rendered result is all color blue.

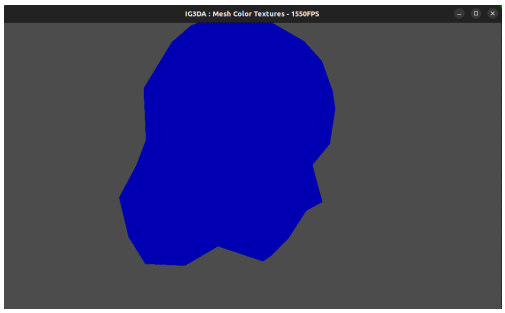

Fig. 4. Face mesh with only vertex colors

Nevertheless, the more important part of the implementation is how I used the mesh color data structure to generate a 2D Mesh color texture layout.

2.1 2D Mesh Color Texture layout

The base code only handles the 3D meshes with triangular faces for rendering, therefore my implementation is also uniquely for handling triangular faces. In the section 3.2 of [Yuksel 2017], the author only mentions that all the edges of each face have to be vertical, horizonal or diagonal without talking about how exactly I should arrange the face positions. To avoid reorganizing the order of the face indices, I paired every two triangles $2f$ and $2f + 1$ to form a rectangle with two texel space between them until I ran out of triangle. For each face, I also store the mesh color data in order so that the edge and vertex colors would be arranged in clockwise order on the texture map (figure 5).

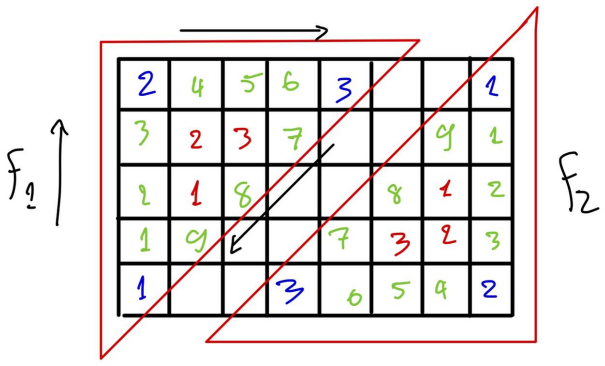

Fig. 5. Order of the mesh color from the data structure to the texture layout (blue: vertex, green: edge, red: face)

. The result is not bad given the limited number of colors is given (figure 6).

2.2 Bilinear Filtering

In order to use the texture from figure 6 for bilinear filtering, I needed to interpolate the gap texels between each rectangle using the equation 1. After 5 iteration, I got the following result (figure 8)

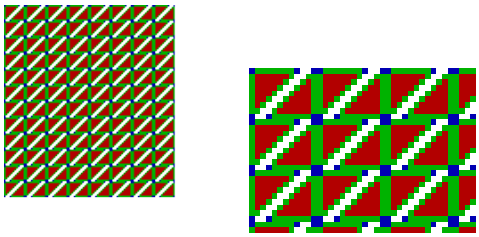

Fig. 6. 2D Layout of Face mesh (left: full texture, right: cropped)

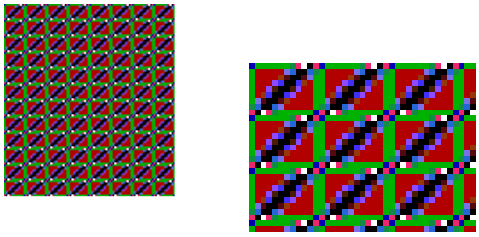

Fig. 7. 2D Layout after interpolating for c_3 (left: full texture, right: cropped)

2.3 Generating Mipmaps for trilinear filtering

Since the resolution that I used for my mesh color data was $R = 7$, so it would generate 4 level of mipmaps for trilinear filtering until it reached the mipmap level of vertex color.

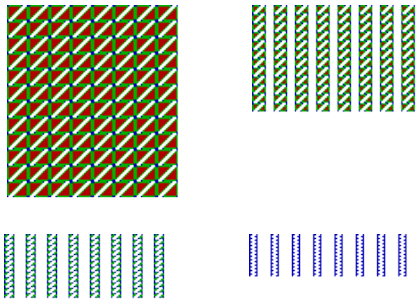

Fig. 8. mipmap level 0 to 3 for resolution $R = 7$ (from left to right, up to down)

The pixel data for each rectangle for all mipmaps looks pretty good in terms of the layout. However, for mipmap level 1 to 3, each column of rectangles keep the distance from each other instead of clamping together. Unfortunately, I had not found the bugs that caused the distancing yet.

2.4 Discussion

As I mentioned previously, I implemented the mesh color texture only to handle meshes with trianglular face uniquely, and without optimization to find the faces that share the same edge together and pair them up. Therefore, I ended up duplicate the vertex and edge color values in the texture layout, and had to interpolate the gap texels (c_3) of each rectangle, and the generated textures waste a lot of storage space as a result. On the other hand, it gives the simplicity of not needing to rearrange the face indices order to pair the every two triangles that share an edge.

. The solutions for generating mipmap level beyond vertex color and for nonuniform mesh colors (example: 1 edge of a face has $R = 3$ while another has $R = 7$) are also presented in the paper, but I did not implement them in my code.

3 CONCLUSIONS

In conclusion, I implemented Mesh Color Textures based on the method proposed by Cem Yuksel. I successfully implemented the generation of 2D Texture Layout for mipmap levels up to vertex color that can be used in for Bilinear and Trilinear Filtering. These two points covers most of the cases when using mesh color for 3D painting.

. For the improvement, I would like to start from the coding of shaders so that I can properly render my 3D meshes with vertex, edge and face colors all together and not only the vertex colors. Secondly, I can improve on handling the paring of every two triangles so that most of them share the same edge in order to avoid making the texel gap for interpolation.

REFERENCES

T. Boubekeur. 2023. Course X-INF584: Image Synthesis. (2023).
D. H.House. C. Yuksel, J. Keyser. 2010. Mesh Color. (2010). http://www.cemyuksel.com/research/meshcolors/meshcolors_techreport.pdf
D. Culler, D. Estrin, and M. Srivastava. 2004. Overview of Sensor Networks. *IEEE Comput.* 37, 8 (Special Issue on Sensor Networks) (2004), 41–49.
C. Yuksel. 2017. Mesh Color Textures. (2017). http://www.cemyuksel.com/research/meshcolors/mesh_color_textures.pdf