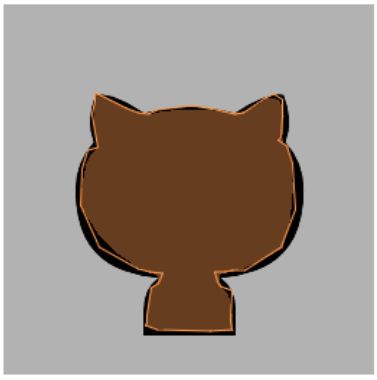
楊宗翰
101062142
2015/4/13

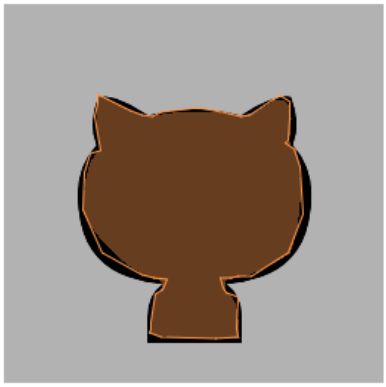
Introduction to Multimedia Homework 2 Report

• Part 1 - Bézier curve

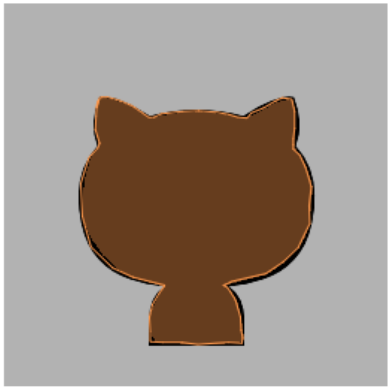
First, I dump the **ctrlPointList** to a **.mat** file so that I can access these points without having to click again. And I also refactor the points to **3n+1**, for the ease of my implementation. I dispatch 4 points in a group and then use blending function in the handout. The result is as follow:



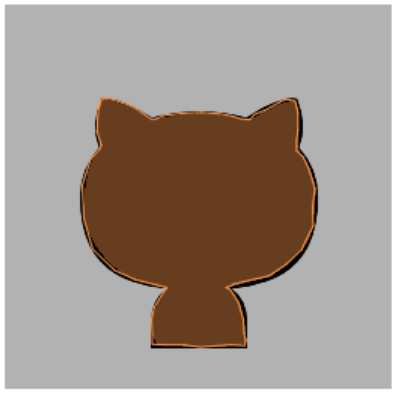
Sample: 46 LoD: 5



Sample: 46 LoD:



Sample: 73 LoD: 5

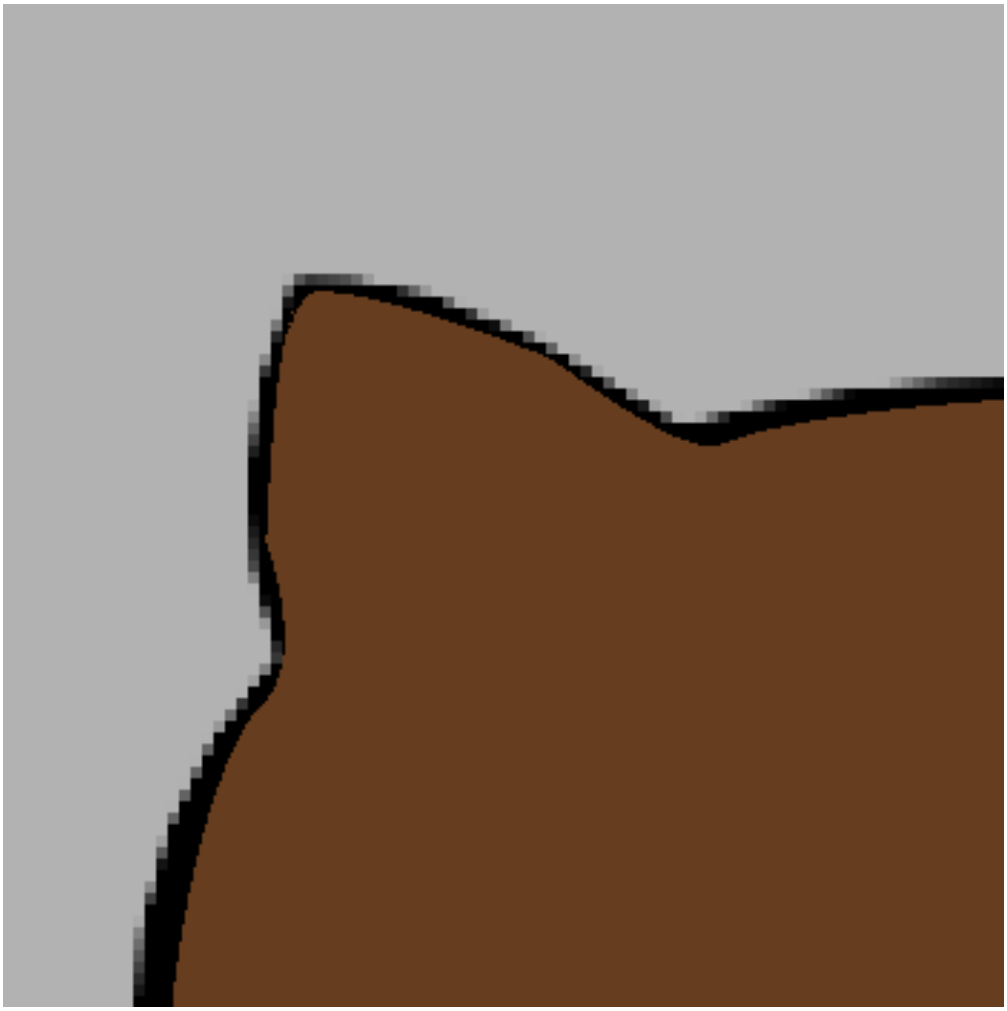


Sample: 73 LoD:

Not far from our exceptions, high sample rate make it more like the original image. As for LoD, lower LoD will make the restored image less smooth.

The enlarged image is as follow. Because we use NN interpolation, it means that the on cell in the original image will became 16 cells in the enlarged one. The curve, however, has vector's good property, which can be precisely enlarged.

The artifact on the original image can be seen clearly. And the restored image has only some zigzags.



I also noticed that on the place which has higher curvature, we should use more points to approximate the outline. This is because Bézier curve does not guarantee to pass all points of that group. Thus, some place would be weird.

- Part 2 - 3D Models

(a) RGB Cube

This section requires us to trace and modify code of **makeRGBCube.m** and construct an **.obj** file that shows a RGB cube. Base on the tutorial, I understand that we can assign color to each vertices and many vertices can become a face.

Then, the missing part of the code provided by TAs becomes clear: we need to assign color to each vertices and add the missing triangles.

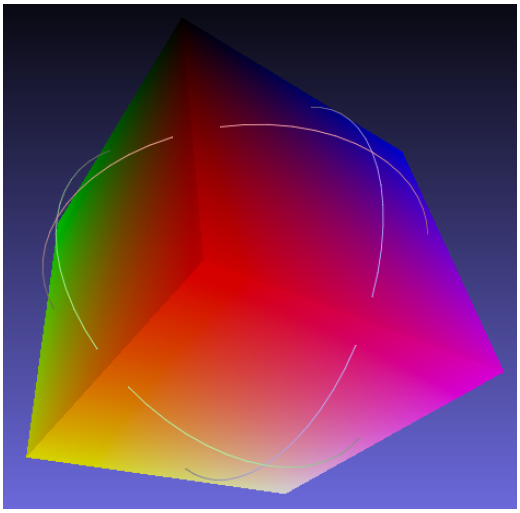
The color of each vertices on RGB cube is opposite to it's diagonal vertex. Since the position of each vertices is composed of 0 and 1. Just assign it's position to each vertices will do the trick! The missing triangles are also easy to find, just add an extra vertex and construct a new triangle will solve this problem.

(b) HSV Cylinder

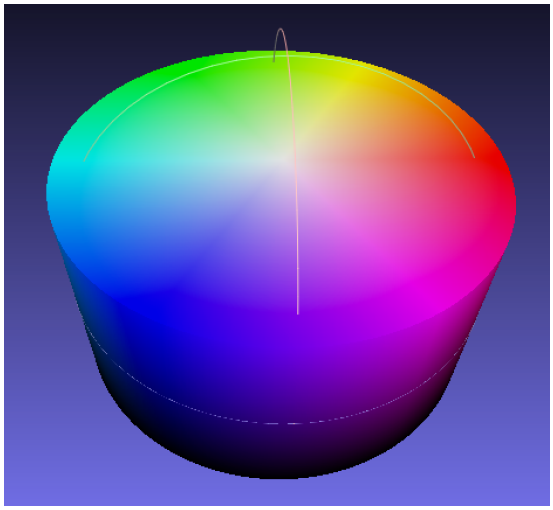
Having the experience of constructing RGB cube, HSV cylinder is just few lines away.

Here, we have to use polyhedron to approximate cylinder. My cylinder consists 2*n+2 vertices. I use n vertices to approximate top and bottom circle and 2 center of circle. We can construct the cylinder using triangles as TAs hint say.

The color part would be easy with the assistance of **hsv2rgb**; also, setting the height of the cylinder and radius of the circles to be 1 will save a lot of effort because the parameters in **hsv2rgb** are from 0~1. Here, I assign h(hue) to be **i/numOfVert**, then, each vertices will have it's corresponding hue. As for S(saturation), I assign that value according to how far it's from the center of circle. V(value) is the height of that vertex, here, I use z-index to do that trick.



Result of (a)



Result of (b)