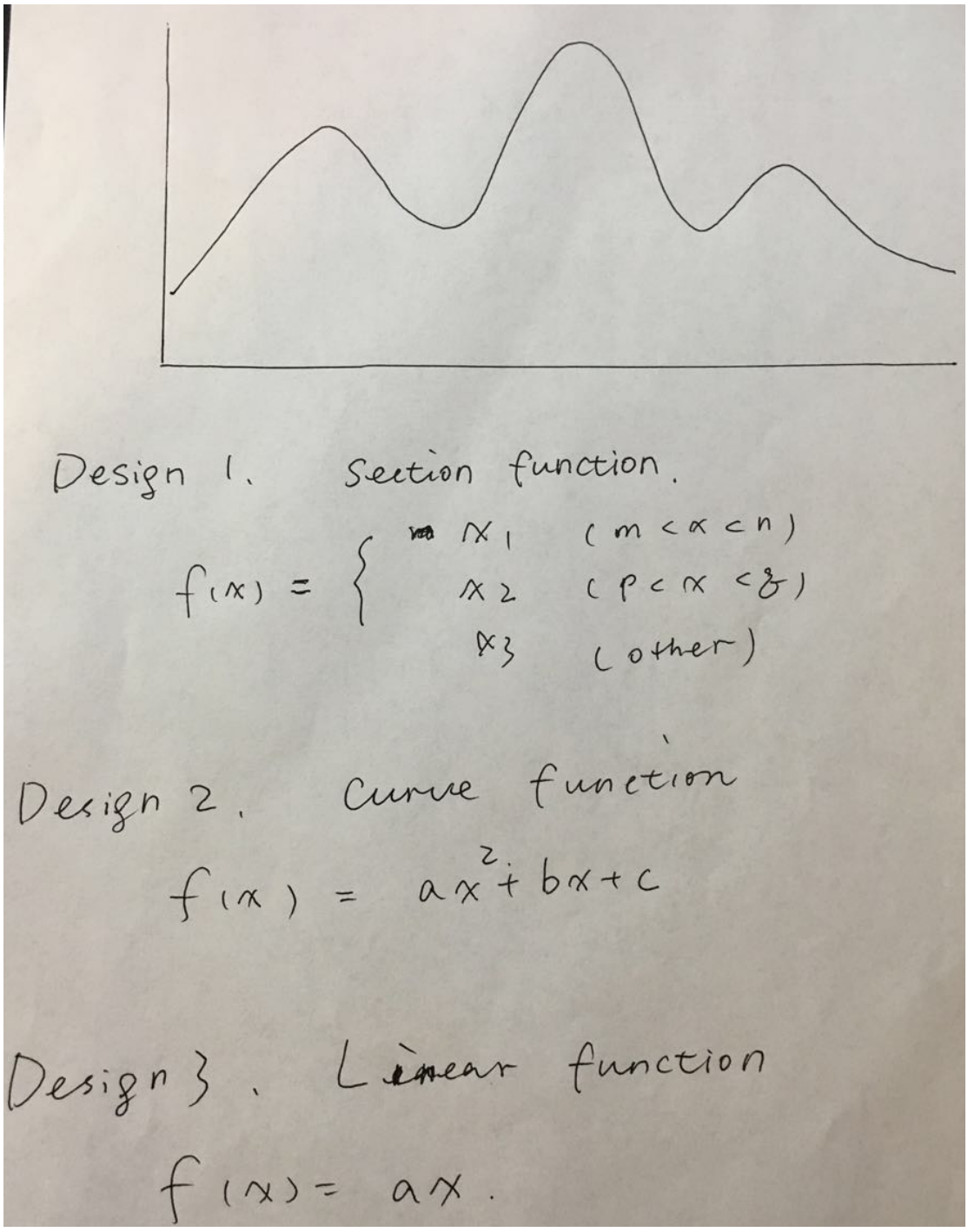
HW6 Design

Design Transfer Function

According to the data file, the range of count (object value) would be divided into three sections. In this way, we can have three designs below.



1. Section Function
   This method can satisfy the color into the three sections.
2. Curve function

This method also can fill the color into coordinate places, but the weakness is that this function should be transformed according to the different input data. For example, whether the vertex should be the highest point or lowest point depends on the dataset.

3. Linear function
   This function also can represent the content of the data, but it is not a good design since it would miss the details. For example, it cannot represent the edge of an object.

For the assignment, I implemented the first and the third transfer function. For the linear function, I added the color scaling widget at the top right of the webpage to adjust the different colors, alpha value and steps in different sections.

When users launch the web page, the section transfer function would be used. This part is implemented in script.js.

Then when users adjust the top right panel, it would switch to linear function automatically, since the three values can be modified independently. This part is implemented in VolumeRenderer.js.

Code Design Analysis

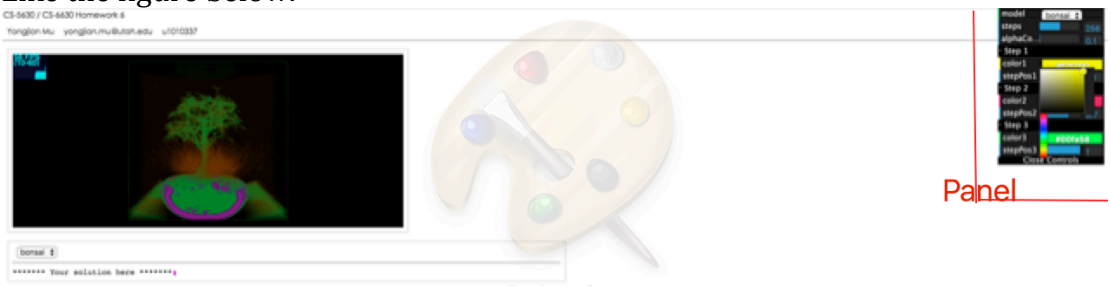
```
var gradient = ctx.createLinearGradient( 0, 0, 4, 25 );
gradient.addColorStop(0, rgb2hex(color));
gradient.addColorStop(1, 'transparent');
ctx.fillStyle = gradient;
ctx.fill();
```

This is the gradient part in script.js

```
if(render_flag == false) {
    var grd = ctx.createLinearGradient(0, 0, canvas.width - 1, canvas.height - 1);
    grd.addColorStop(guiControls.stepPos1, guiControls.color1);
    grd.addColorStop(guiControls.stepPos2, guiControls.color2);
    grd.addColorStop(guiControls.stepPos3, guiControls.color3);
    ctx.fillStyle = grd;
    ctx.fillRect(40,0,canvas.width -1 ,canvas.height -1 );
}

if(render_flag == true) {
    for (i = 0; i < self.steps; i += 1) {
        ctx.fillStyle = colorScale(i/self.steps);
        ctx.fillRect(i, 0, 1, 2);
    }
}
```

```
//Setup transfer function steps.
var step1Folder = gui.addFolder('Step 1');
var controllerColor1 = step1Folder.addColor(guiControls, 'color1');
var controllerStepPos1 = step1Folder.add(guiControls, 'stepPos1', 0.0, 1.0);
controllerColor1.onChange(function(value) { render_flag=false; self.updateTransferFunction(value); } );
controllerStepPos1.onChange(function(value) { render_flag=false; self.updateTransferFunction(value); } );

var step2Folder = gui.addFolder('Step 2');
var controllerColor2 = step2Folder.addColor(guiControls, 'color2');
var controllerStepPos2 = step2Folder.add(guiControls, 'stepPos2', 0.0, 1.0);
controllerColor2.onChange(function(value) { render_flag=false; self.updateTransferFunction(value); } );
controllerStepPos2.onChange(function(value) { render_flag=false; self.updateTransferFunction(value); } );

var step3Folder = gui.addFolder('Step 3');
var controllerColor3 = step3Folder.addColor(guiControls, 'color3');
var controllerStepPos3 = step3Folder.add(guiControls, 'stepPos3', 0.0, 1.0);
controllerColor3.onChange(function(value) { render_flag=false; self.updateTransferFunction(value); } );
controllerStepPos3.onChange(function(value) { render_flag=false; self.updateTransferFunction(value); } );
```

In the source code, I implemented two transfer function. To invoke the different transfer functions (one in script.js and the other is in VolumeRenderer.js), I used the flag to distinguish them.
The transfer function in script.js would appear at first, then when users change the value from the panel, it would use the transfer function in VolumeRenderer.js. Like the figure below.



The advantage of this design is we can make comparison of these two transfer functions and we can also find the perfect value by adjusting the colors independently.
There is a disadvantage that we cannot change back to the first transfer function when we already changed to the second one.