Beginner

OpenGL ES & GLKit

Hands-On Challenges

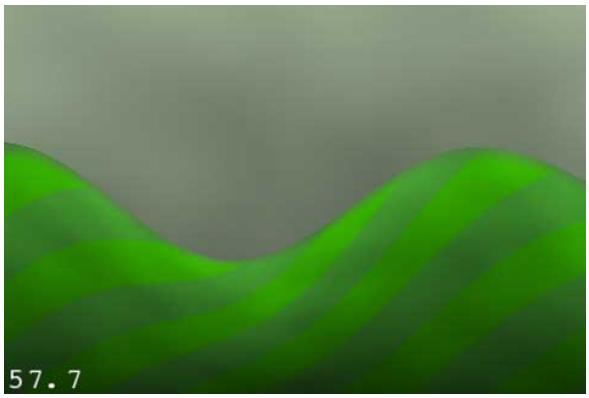
Beginner OpenGL ES & GLKit Hands-On Challenges

Challenge #3: Tiny Hills

So far, you have learned how to use OpenGL to render a triangle, square, and a star. I bet you could imagine extending this into other shapes – such as an octagon if you are a UFC fan!

However, one of the coolest reasons to use OpenGL is that you have the ability to create any shape you can imagine as the game runs. For example, can make geometry as the user drags their finger across the screen or in response to a user's data (like a bar chart).

This effect was used to great success in Tiny Wings, where the hills were generated randomly at runtime, kinda like this:



Your task in this challenge is to experiment with making dynamic terrain shapes to accomplish a similar effect to the hills in Tiny Wings (but much simpler).

Part 1: Drawing the Hilltops

In the resources for this challenge, you will find a starter project. This is a project that is equivalent to where things left off in the lecture, with a square drawn to the screen. Look through the project and make sure you have a good understanding of how it works.

Remember that even though this is a square, it is stretched to fill the entire screen. This is because by default, OpenGL displays geometry within a 2x2 3D cube centered on (0, 0). You'll learn how to fix this in the next lecture.

Next, drag the **Challenge Resources** folder from the resources for this challenge into your project. Make sure that **Copy items into destination group's folder (if needed)** is checked and the **HelloOpenGL** target is checked, and click **Finish**.

Take a look at the file inside this folder – it's just a helper method I got off Stack Overflow that helps you generate colors along the rainbow. You just pass in a value between 0 and 1 and it sends back the appropriate R, G, B color. This will be useful in a bit to make your hills look nice.

In the lecture, you generated the square by defining 4 vertices, and then using an index array to describe two triangles made up by this vertices. In this part of the lab, you're going to use an alternate rendering mode in OpenGL called `GL_LINE_STRIP`. With this, you simply list one vertex after the other, and OpenGL will draw lines between them.

In the next part, your goal is to get some simple test terrain showing up on the screen. Add these constants and import to the top of **RWTViewController.m**:

```
#import "RWTRainbow.h"

#define kNumHillKeyPoints       10
#define kNumHillVertices        (kNumHillKeyPoints)
```

In this section, basically you will be making 10 dots randomly across the scren, and drawing lines between them to connect. Since you're only drawing lines right now, you only need one vertex per dot. In the next section, you'll need more than one vertex per dot (so you can draw triangles to fill the area underneath the top of the hills).

Next add the following private instance variables:

```
CGPoint _hillKeyPoints[kNumHillKeyPoints];
```

```
RWTVertex _hillVertices[kNumHillVertices];
CGRect _rect;
```

This defines the arrays for the hill key points and the vertices, and a rectangle that defines the area in which to render the hills.

Next add the following helper method to the top of the file:

```
#define ARC4RANDOM_MAX      0xFFFFFFFFu
CGFloat RandomFloatRange(CGFloat min, CGFloat max) {
  return ((double)arc4random() / ARC4RANDOM_MAX) * (max - min) +
    min;
}
```

This is a helper function that gives you a random number between two float values.

Then add the following method:

```
- (void) generateHillKeyPoints {
    float x = _rect.origin.x;
    float y = _rect.origin.y;
    _hillKeyPoints[0] = CGPointMake(x, y);
    for(int i = 1; i < kNumHillKeyPoints - 1; ++i) {
        x += _rect.size.width/(kNumHillKeyPoints - 1);
        y = RandomFloatRange(_rect.origin.y,
          _rect.origin.y + _rect.size.height);
        _hillKeyPoints[i] = CGPointMake(x, y);
    }
    _hillKeyPoints[kNumHillKeyPoints-1] = CGPointMake(
      _rect.origin.x + _rect.size.width, _rect.origin.y);
}
```

This generates the random points for the tops of the hills from left to right across the screen. There's nothing OpenGL-specific about this code, it's just straight Objective-C – so take a look through and make sure you understand how it works.

Next add the following two helper methods:

```
- (void)setHillVertexAtIndex:(int)i x:(float)x y:(float)y
  r:(float)r g:(float)g b:(float)b a:(float)a {
    _hillVertices[i].Position[0] = x;
    _hillVertices[i].Position[1] = y;
    _hillVertices[i].Position[2] = 0;
    _hillVertices[i].Color[0] = r;
    _hillVertices[i].Color[1] = g;
    _hillVertices[i].Color[2] = b;
    _hillVertices[1].Color[3] = a;
```

```
    NSLog(@"Point %d: %0.2f %0.2f", i, x, y);
}

- (void)generateHillVertices {
  for (int i = 0; i < kNumHillKeyPoints; ++i) {
    float r, g, b;
    getRainbowColor((float)(kNumHillKeyPoints-i) /
      (float)kNumHillKeyPoints, &r, &g, &b);

    [self setHillVertexAtIndex:i x:_hillKeyPoints[i].x
      y:_hillKeyPoints[i].y r:r g:g b:b a:1];
  }
}
```

This loops through all of the hill key points and creates a vertex for each. Note that it uses the `getRainbowColor` helper method to generate a color that looks nice for each vertex.

Again – nothing very OpenGL sepecific here except it's filling a vertex structure (we created this ourselves in the lecture – you can see it in **vertex.h**) that you will be passing to OpenGL later.

Next replace `setupVertexBuffer` with the following:

```
- (void)setupVertexBuffer {

  _rect = CGRectMake(-1, -1, 2, 2);
  [self generateHillKeyPoints];
  [self generateHillVertices];

  // Create vertex buffer
  glGenBuffers(1, &_vertexBuffer);
  glBindBuffer(GL_ARRAY_BUFFER, _vertexBuffer);
  glBufferData(GL_ARRAY_BUFFER, kNumHillVertices * sizeof(RWTVertex),
    _hillVertices, GL_STATIC_DRAW);

}
```

First you define the rectangle in which to draw the hills. Remember by default OpenGL displays vertices in a 2x2 cube centered on (0, 0), so here you set up the bounds to be equal to those default visible coordinates.

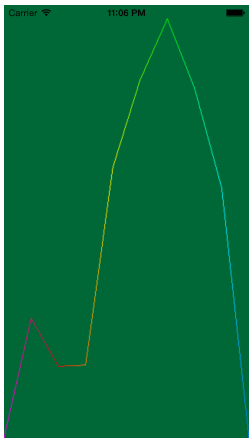
You then call the methods you wrote earlier to set up the hill key points and vertices, and finally you create a vertex buffer with that data so the GPU can access it.

As a final step, inside `glkView:drawInRect:`, replace the `glDrawElements` line with the following:

```
glDrawArrays(GL_LINE_STRIP, 0, kNumHillVertices);
```

This draws your vertices in line strip mode, which simply draws a line from one vertex to the next from your vertex buffer.

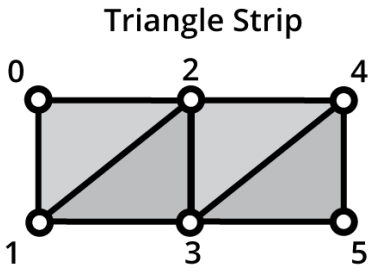
Build and run and you should see the following:



Part 2: Filling the Hills

You're finally on to the fun part – filling in the hills!

To do this you are going to use a new handy rendering mode called `GL_TRIANGLE_STRIP`. Remember from the lecture that a triangle strip will automatically create triangles from each vertex and the previous two:



This is perfect for the hills, because you can make the top of the strip the top of the hills, and the bottom of the strip the bottom of the screen.

Now you will need two vertices per key point (the keypoint itself, and the area right underneath the keypoint), so update `kNumHillVertices` to the following:

```
#define kNumHillVertices        (kNumHillKeyPoints*2)
```

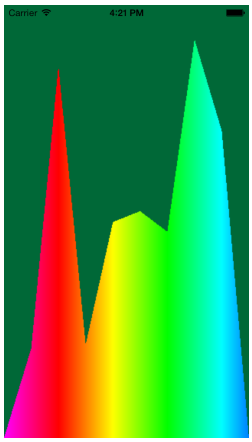
Also update the `glDrawArrays` line in `glkView:drawInRect:` to the following:

```
glDrawArrays(GL_TRIANGLE_STRIP, 0, kNumHillVertices);
```

Finally for your challenge – to modify generateHillVertices to generate 2 vertices per key point: one for the keypoint itself, and one for the area at the bottom of the rectangle underneath.

To get this working, you just need to modify one line of code, and add one.

When you're done, build and run and you should see the following:

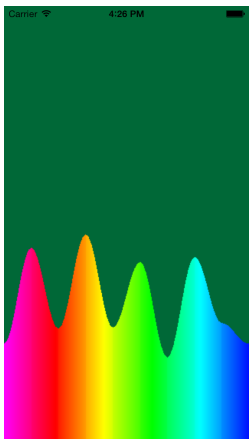


Uber Haxx0r Challenge: Beautiful Terrain

Right now the hills look a bit jaggedy, and a bit too random for my liking. The uber haxx0r challenge is to:

- Smooth them out a bit
- Make the hills begin a bit taller than the bottom of the screen
- Make the hills have less potential random variance between the hills
- Make the hills vary always between down or up
- Anything else you'd like to make the hills look nicer

By the time you're done, your hills should look like this:


Carrier
4:26 PM