Matthew Conover
CSCE 412
Prof. Wolff
20 May 2016

Normal Mapping

Let me start at the beginning of the process: procedurally generating a height-map. To do this, I simply made use of the simplex noise function provided by GLM, summing the outputs for multiple intervals at each x,y position in the height-map, keeping track of the maximum and minimum heights encountered. I then iterate though the list of heights (stored as floats), and normalize them to a value between zero and one based on the min value, and range. Once normalized, multiplying by 255 yields a value which can be stored in a byte.

Something I noticed, was that when generating 2D simplex noise of a high frequency (greater than a fourth the change in x and y), it started to form some grove-like formations. I am not sure why, but for this reason, I end up adding a 3D simplex value as what I consider the most important value (and I also multiply it by the greatest coefficient). The benefit I found to 3D simplex over 2D, is that instead of going between such hard pure -1 and pure 1 values, it has more middle values that it goes towards; this seems to be a result of taking out a slice of the three dimensional noise, as 4D noise has even more of the middle ranges.

Now with that height map, I generate a normal map by iterating though every pixel, and then calculate the normal for the eight imaginary triangles that could be drawn with the center coordinate being the current pixel, and then each of the eight surrounding pixels acting as additional x, y, z coordinates (z being the height). I manually define them in a way which is similar to going counter-clockwise around a circle from 0 to 2π, touching {1,0}, {1,1}, {0,1}, {-1,1}, …, {1,0}. The normalized sum of these triangles' face normals, is the normal I use for the given pixel, then converting x ⟶ red, y ⟶ green, and z ⟶ blue; yielding the expected, bluish-tinted image.

It is worth making clear that the method I chose for normal-map calculation is one of many, and that there does not seem to be a single preferred way of doing it. The GIMP plugin alone has nine different options; I believe the one equivalent to my implementation is what they refer to as 3x3.

Once I have the normal map, I set it up as a texture, following the examples provided in class for setting up the VAO with the texture coordinates, and then sending the Sampler2D object to the shader. Something I decided on very quickly was to use a global ST/UV space rather than having each part of the road go from zero to one, this way, narrow quads—like those on turns—would not look more scrunched than the rest nor larger quads look more expanded. Basically, I did was set the texture coordinates to the x, z coordinates of the vertex, multiplying by a repeat factor—at the time of writing that repeat factor was 0.25, which means that a 4x4 area of world coordinates is the area of the texture, and it repeats beyond that. This also means that from one quad to the next they will line up perfectly, and in normal generation, I wrapped off the edges to the other side which further helps them tile nicely.

At this point, the normal map was sampleable from the fragment shader, and I only needed to convert the normal to the camera coordinate system from the tangent coordinates. Turns out this was a little harder than I thought as I had been looking at guide which actually put everything into tangent space rather than putting the normal into camera space, and I had not realized that was what they were doing for a little while. Ultimately, it is just another frame transformation to go from

tangent space to camera space, which can be constructed given a tangent to the surface. Normally this tangent would be calculated and initialization based on the direction of the ST/UV coordinates and then stored in the VAO, but since I am only working with a flat object, I arbitrarily set it to a line perpendicular to the y axis. Although unnecessary, I calculate the tangent in the vertex and have it interpolated to the fragment shared, mostly because that is how I would likely do it if I were to expand for a tangent stored in the VAO.

The ending result, is a new normal I am able to used to replace the original in the shading model.

At this point, I decided to add some specular to the road to help highlight the difference, and this also seems appropriate based on the actual appearance of asphalt. Ultimately, the thing I think is the most hard to get right, is the procedural generation of the texture. I put at least an hour or two into tweaking the values, but I would not consider my output to be so much that of asphalt, but it does do a good job of demonstrating the concepts, and the procedural generation could always be swapped out for something that an artist cooked up.