CSC 317

Tutorial: Shader Pipeline

file extensions

vs : vertex shader

fs : fragment shader (pixel shader)

tcs: tesselation control shader

tes: tesselation evaluation shader

identity, uniform_scale, translate, rotate.glsl:

* glsl matrices are column major, ie each entry goes down a column before going down a row

```
return mat4(
0,0,0,0,
0,0,0,0,
0,0,0,0,
0,0,0,0);
```

model.glsl:

* also expects scale

* rotate one orbit every 4s

* want it to not do anything (identity I) if the object is not the moon

* and do the transformation T if the object is the moon

* I + is_moon* (T - I)

model_view_projection.vs:

* spec for how the moon should behave

* to be clear, the transforms on the moon should be defined in model.glsl

* first, convert pos_vs_in to a 4d homogeneous coordinate

* next, do the model transform

* next, do worldspace to cameraspace transform

* next, do perspective projection

blue_and_gray.fs:

* look at whitelist for what function you can use to do this w/o branching

* you basically want (is_moon*blue) + (!is_moon*gray)

5.tcs:

* https://erkaman.github.io/posts/tess_opt.html

snap_to_sphere.tes:

* be mindful of what spaces your vectors belong to
* you can easily snap a point to the unit sphere with normalize
* for normals, consider what each transformation does to the normal
    * ie, what do rotations, translations, scales do to n
* there are a class of matrices where M^T = M^-1 (orthonormal)
* the perspective projection matrix is not orthonormal, but what about the upper left 3x3 block?
* what do we know about the homogeneous representation of vectors?

blinn_phong.glsl:

* careful about the signs
* we want this function to EXPECT the right directions
    * ie: n, v, l point OUTWARD from the surface
    * notice that this function expects normalized n, v, l
* remember the ambient term
* ambient + diffuse + specular

lit.fs:

* to match the example, rotate the light 1 orbit per 8 s
    * pure white light specular response
    * 1000 phong
* get n, v from the fragment shader inputs [normal, view]_fs_in
* careful about signs here, need to match what is in blinn_phong.glsl
* l hardcodes where the point light is coming from
* make sure to normalize n, v, l

random_direction.glsl:

* this is not the trivial random point in R3 case

* https://mathworld.wolfram.com/SpherePointPicking.html

* need to make sure the points are uniformly spaced on the unit sphere

smooth_step.glsl:

* use desmos

* cubic polynomial (ie, can’t use logistic curve, atan)

* f(0) = 0, f(1) = 1

* f’(0) = 0, f’(1) = 0

* 3x^2 - 2x^3 (in slides)

perlin_noise.glsl:

* notice that position st is the seed
* whitelist floor, fract
* need to mix (ie lerp) across all three dimensions
* ie mix(contrib(x), contrib(x+1), x step)
* so:
    * mix (direction a * fraction a , direction b * fraction b, smoothstep(fraction a)) in one dimension
* https://dl.acm.org/doi/pdf/10.1145/325165.325247
* https://mrl.cs.nyu.edu/~perlin/doc/oscar.html#noise

procedural_color.fs:

* creative, though make sure both the planet and the moon has a texture
* make sure you use perlin noise
* also make sure the texture rotates with the moon
    * ie, the same face of the moon always points towards the planet
    * need to make sure view, model transforms do not warp the texture
* start with lit.fs, modify the colours being fed into blinn_phong

improved_smooth_step.glsl:

*

http://www.heathershrewsbury.com/dreu2010/wp-content/uploads/2010/07/ImprovingNoise.pdf

* 6*x^5 - 15*x^4 + 10*x^3

improved_perlin_noise.glsl

* same as perlin_noise.glsl but using improved_smooth_step.glsl

bump_position.glsl

* expects s to be normalized

* adjust the position of a point s by the given amount of bump_height

bump_height.glsl

* creative
* make sure the elevation change is reasonable
    * [-0.1, 0.1]
* s is position and seed
* both moon and planet should be bumpy (can change the bumpiness of either)
* again make sure model and view do not warp the displacement map
* smooth_heaviside.glsl is available

planet.fs

* creative

* can use t as a parameter for animations (example used it to add water and clouds)

* use position as an interesting parameter (example used the latitude to change the colour of land)

* change the material properties (example had the water have specular reflection)

* (example had specular reflection had a bump map)