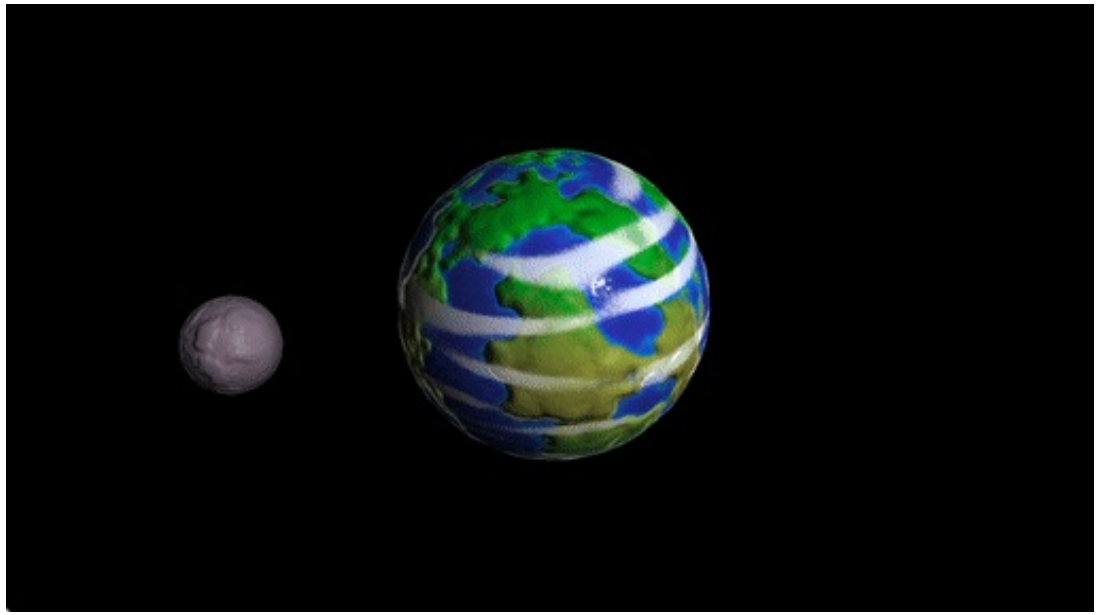
# CSC317 Computer Graphics
# Tutorial 6

October 25, 2023

# Assignment 6: Shader Pipeline

- Due Date: November 1 @ 11:59 pm



# Overview

- A Real-time Rendering pipeline
  - Procedural Rendering: a sequence of steps to display a 3D scene on a 2D screen

- Why?
  - Saves memory/storage: input can be simple geometry
  - Efficient + scalable: same procedures can be used for multiple objects
  - Easy to parallelize: good run time

# GLSL

- GLSL: OpenGL shading language
  - “C” like syntax
  - Shaders: programs that can be run on the GPU

- Pipeline
  - On CPU, gather scene data + compile shader code (if needed)
  - Send compiled shader + scene data to GPU
  - GPU does calculation based on shaders + output to framebuffer
  - Ask GPU to draw output to window

# GLSL – File Types

- Vertex Shaders (unit: per vertex): **.vs files**
  - Transforms the vertex -> geometry of the object
  - Input: vertex position; Output: vertex position

Input

Output

Computation

```
in vec4 pos_vs_in;
out vec4 pos_cs_in;
void main()
{
  pos_cs_in = pos_vs_in;
}
```

Passthrough Filter

# GLSL – File Types

- Tessellation Control Shaders (unit: per face): **.tcs files**
  - Set parameters for tessellation (gl_TessLevelOuter/ gl_TessLevelInner)

```
layout (vertices = 3) out;

in vec4 pos_cs_in[];
out vec4 pos_es_in[];

void main()
{
  // Calculate the tess levels
  if(gl_InvocationID == 0)
  {
    gl_TessLevelOuter[0] = 1;
    gl_TessLevelOuter[1] = 1;
    gl_TessLevelOuter[2] = 1;
    gl_TessLevelInner[0] = 1;
  }
  pos_es_in[gl_InvocationID] = pos_cs_in[gl_InvocationID];
}
```

Passthrough Filter

# GLSL – File Types

- Tessellation Evaluation Shaders (unit: per vertex): **.tes files**
  - Takes the result of the tessellation that the tessellation control shader specifies

Hint!

```
layout(triangles, equal_spacing, ccw) in;
in vec4 pos_es_in[];
out vec4 pos_fs_in;
// expects: interpolate
void main()
{
  pos_fs_in = interpolate(gl_TessCoord,pos_es_in[0], pos_es_in[1], pos_es_in[2]);
  gl_Position = pos_fs_in;
}
```

Passthrough Filter

# GLSL – File Types

- Fragment Shaders (unit: per fragment): **.fs files**
  - Fragment: result from rasterization process
  - Fragment: “a potential pixel”
    - Antialiasing /Multi-sampling: Multiple fragments will be interpolated to form a single pixel
  - Assign color to each fragment
  - Input: surface location vectors; output: color vector

# GLSL – File Types

- Fragment Shaders (unit: per fragment): **.fs files**

```
in vec4 pos_fs_in;
out vec3 color;
void main()
{
  // Set color to screen position to show something
  color = 0.5+0.5*pos_fs_in.xyz;
}
```

Passthrough Filter

- .glsl files – helper functions

# GLSL – Code Sharing

- No “#include” line
- Use .json to collect different files

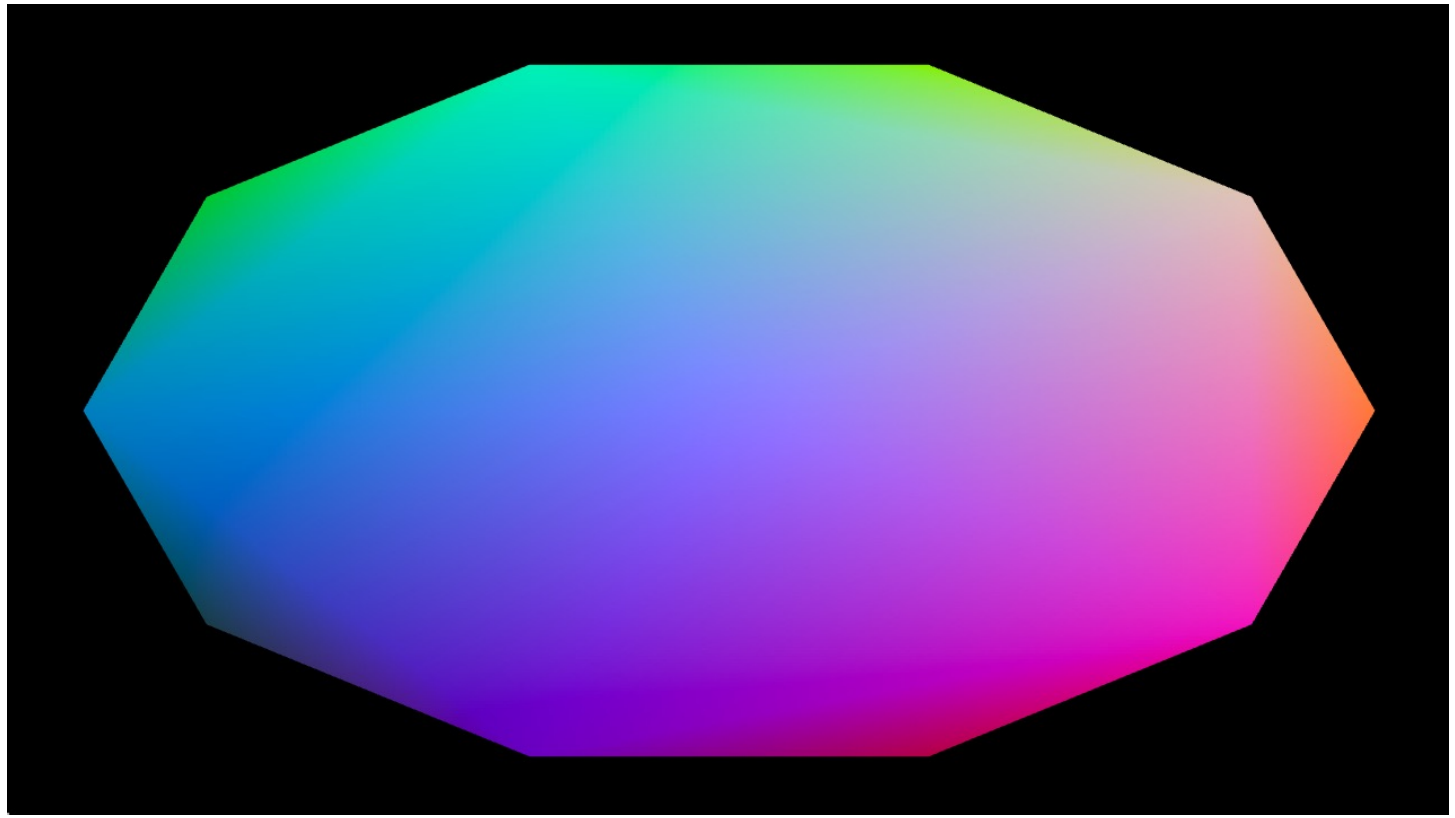
```
{
  "vertex": [ "../src/version410.glsl","../src/pass-through.vs"],
  "tess_control": [ "../src/version410.glsl","../src/pass-through.tcs"],
  "tess_evaluation": [ "../src/version410.glsl","../src/interpolate.glsl","../src/pass-through.tes"],
  "fragment": [ "../src/version410.glsl","../src/pass-through.fs"]
}
```

# GLSL – Sample Datatypes

- **vec3 *a*:** 3d vector; *a*.x, *a*.y, *a*.z
- **vec4 *b*:** 4d vector; vec4 b = vec4(a,1.0)
- Other types: *bool*, *float*, *mat4*, *scalar*, etc
- Variable types
  - in: input
  - out: output
  - uniform: parameter

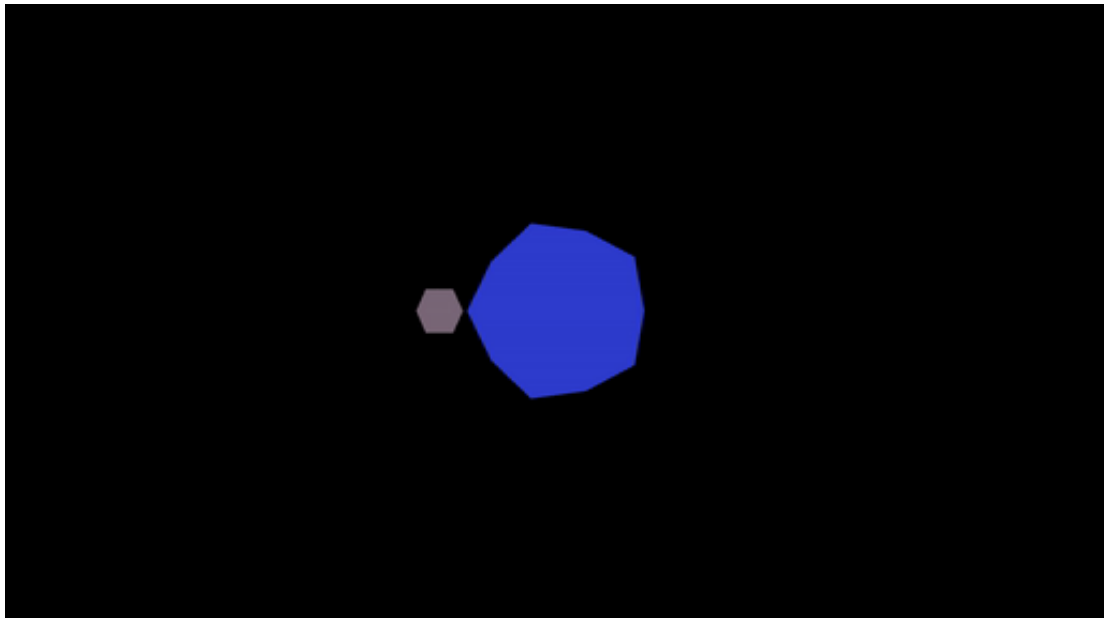
# Assignment 6 – Example Outputs

- Test-01: ./shaderpipeline ../data/test-01.json
-



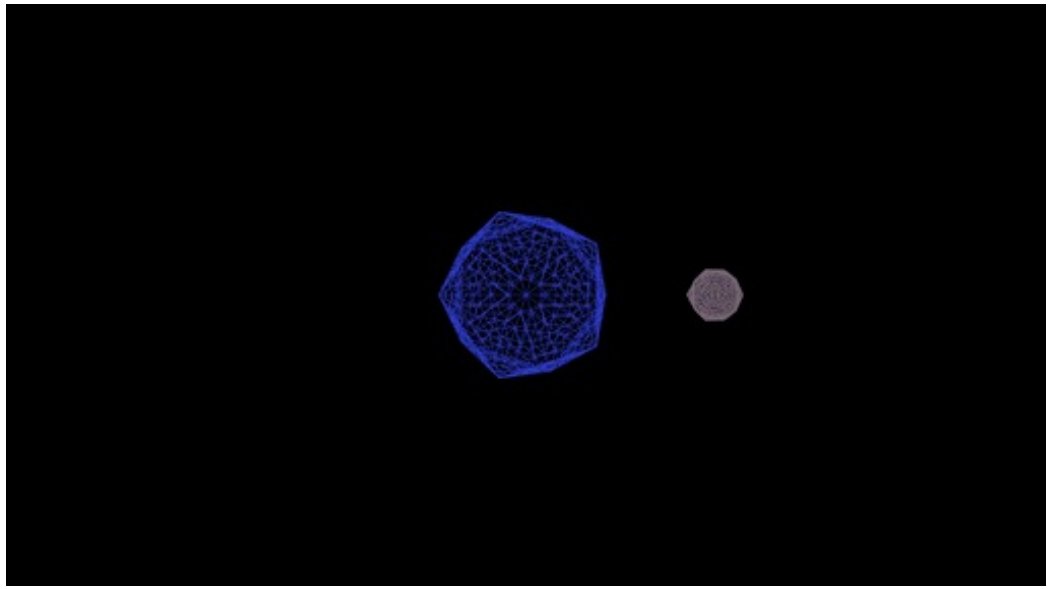
# Assignment 6 – Example Outputs

- Test-02: ./shaderpipeline ../data/test-02.json
  - src/identity.glsl
  - src/uniform_scale.glsl
  - src/translate.glsl
  - src/rotate_about_y.glsl
  - src/model.glsl
  - src/model_view_projection.vs
  - src/blue_and_gray.fs



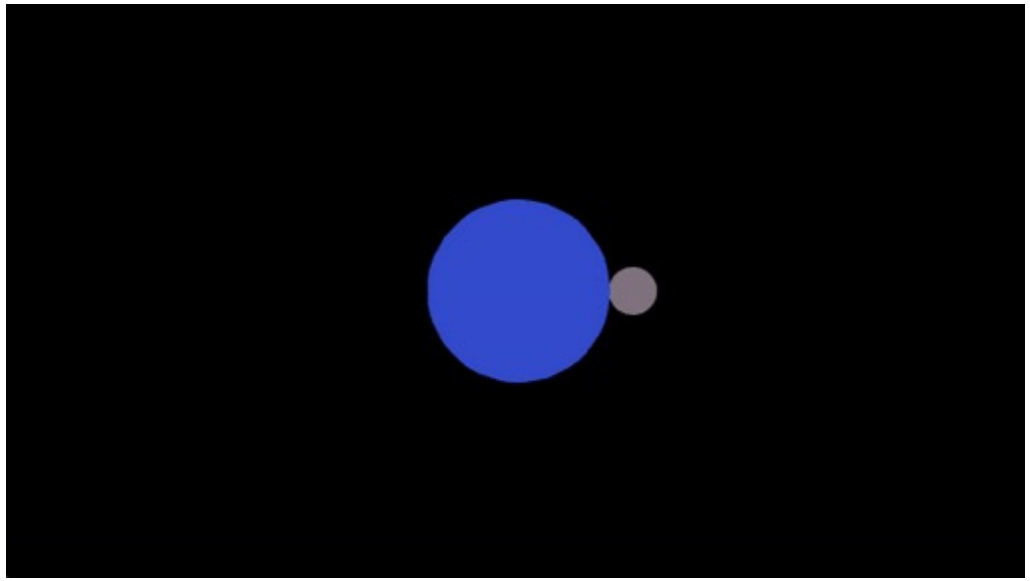
# Assignment 6 – Example Outputs

- Test-03: ./shaderpipeline ../data/test-03.json
  - src/5.tcs



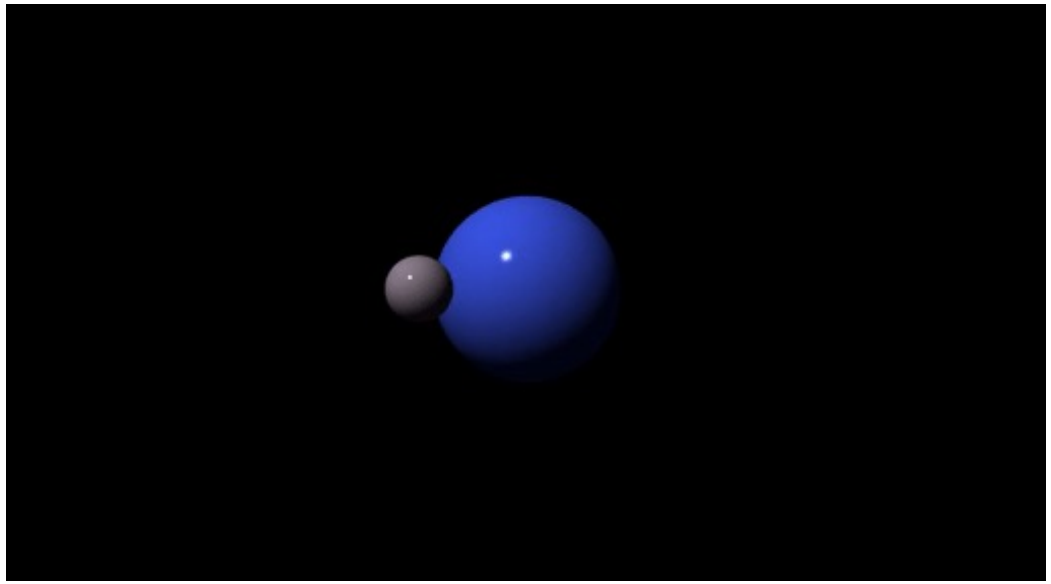
# Assignment 6 – Example Outputs

- Test-04: ./shaderpipeline ../data/test-04.json
  - snap_to_sphere.tes



# Assignment 6 – Example Outputs

- Test-05: ./shaderpipeline ../data/test-05.json
  - blinn_phong.glsl
  - lit.fs

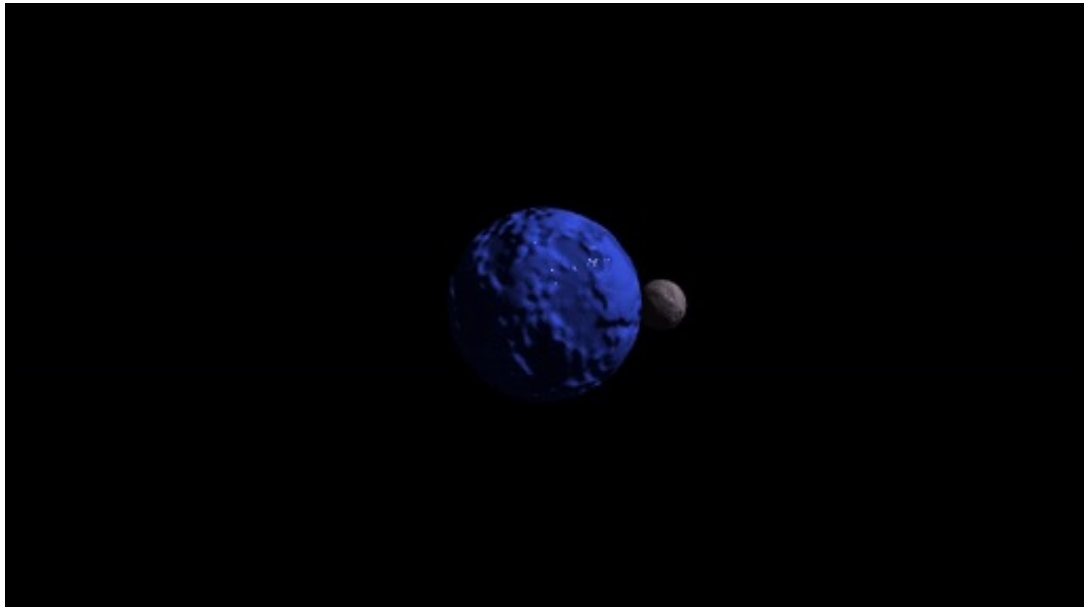


# Assignment 6 – Example Outputs

- Test-06: ./shaderpipeline ../data/test-06.json
  - random_direction.glsl
  - smooth_step.glsl
  - perlin_noise.glsl
  - procedural_color.glsl

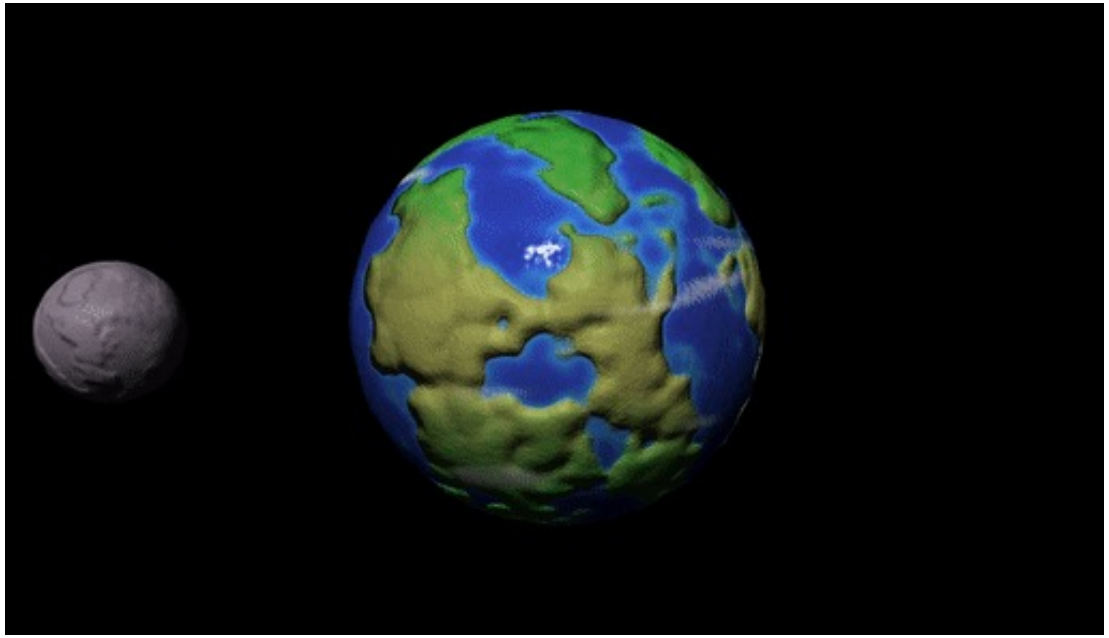
# Assignment 6 – Example Outputs

- Test-07: ./shaderpipeline ../data/test-07.json
  - improved_smooth_step.glsl
  - improved_perlin_noise.glsl
  - bump_height.glsl
  - bump_position.glsl
  - tangent.glsl
  - bump.fs



# Assignment 6 – Example Outputs

- Test-08: ./shaderpipeline ../data/test-08.json
  - planet.fs



# Assignment 6 – Hints

- Set screen colors to debug (see A6 Github)
- .json file provides information on what shaders are used
- Read comments on input/output at the top of the file carefully
- Shader change should be reflected immediately
- Don’t need to get exact output as A6 write-up – Be creative!
- Take a look at the “white list” functions
- Be careful with “row major” vs. “column major” matrices

# Assignment 6 – Resources

- https://adrianb.io/2014/08/09/perlinnoise.html
- https://en.wikipedia.org/wiki/Perlin_noise