CSC317 Computer Graphics

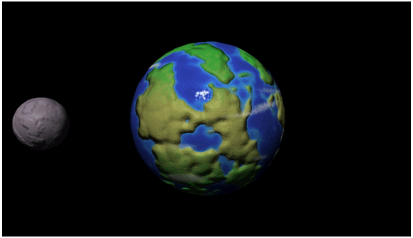
Tutorial 6

Adapted from Wenzhi Guo's tutorial notes

October 23, 2024

Assignment 6: Shader Pipeline

Due date: November 5 at 11:59pm



What are shaders

- Shaders are small programs running on the GPU to control rendering.
- **Vertex Shader**: Processes vertex attributes (like position) and outputs transformed positions and other data.
- **Fragment Shader**: Determines the final color of each pixel (or "fragment").

Why do we use shaders

- Save memory: Only simple geometry or data is passed to the GPU.
- Efficient: Shaders run on many GPU cores, allowing for parallel processing.
- Flexible: Customizable control over rendering behavior (lighting, color, etc.).

How do we implement them?

Using OpenGL Shading Language (GLSL)

- GLSL has a C-like syntax.

Pipeline:

- On CPU: Gather scene data and compile shader code (if needed).
- Send compiled shaders and scene data to the GPU.
- GPU runs the shaders and outputs results to the framebuffer.
- Instruct the GPU to render the framebuffer to the display.

GLSL - file types

Vertex Shaders (unit: per vertex): .vs files

- Transform vertices from model space and pass data to later stages.

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

pass-through.vs

GLSL - Tessellation Control Shader I

Tessellation Control Shaders (unit: per patch): .tcs files

- Set parameters for tessellation (`gl_TessLevelOuter` and `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;
```

GLSL - Tessellation Control Shader II

```
    gl_TessLevelInner[0] = 1;
  }
  pos_es_in[gl_InvocationID] =
   ↪  pos_cs_in[gl_InvocationID];
}
```

pass-through.tcs

GLSL - Tessellation Evaluation Shader

Tessellation Evaluation Shaders (unit: per vertex): .tes files

- Takes tessellated data and computes vertex positions.

```
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;
}
```

pass-through.tes

GLSL - Fragment Shader

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

- **Fragment**: Represents a "potential pixel" after rasterization.
- **Antialiasing/Multi-sampling**: Multiple fragments are blended to smooth jagged edges.
- **Purpose**: Compute final color for each fragment.
- **Input**: Surface data like normals, positions, and textures.
- **Output**: Final pixel color.

```
in vec4 pos_fs_in;
out vec3 color;
void main() {
  color = 0.5+0.5*pos_fs_in.xyz;
}
```

pass-through.fs

GLSL - Misc

- Helper functions are stored in .glsl files.
- GLSL lacks `#include` because shaders are compiled at runtime
- We use .json to specify arguments for the program.

GLSL Data Types and Variable Types

Data Types:

- `vec2`, `vec3`, `vec4`: Vectors with 2, 3, or 4 components for positions, colors, etc.
- `mat3`, `mat4`: 3x3 and 4x4 matrices, often used for transformations.
  - **Important:** these matrices are column major matrices
- `float`, `int`, `bool`: Basic scalar types.

Variable Types:

- `in`: Input variables (e.g., vertex attributes).
- `out`: Output variables passed to the next stage.
- `uniform`: Global variables set by the CPU, constant for all vertices or fragments.

GLSL Basic Operations and Functions

Basic Operations (similar to C++):

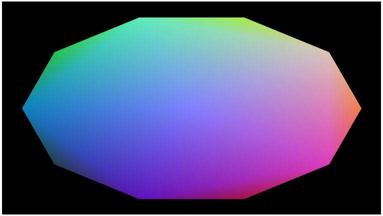
- Arithmetic: +, −, ∗, /, %.
- Component-wise operations for vectors.
- Comparisons: ==, !=, <, >, <=, >=.
- Logical: `&&`, `||`, `!`.

Basic Functions:

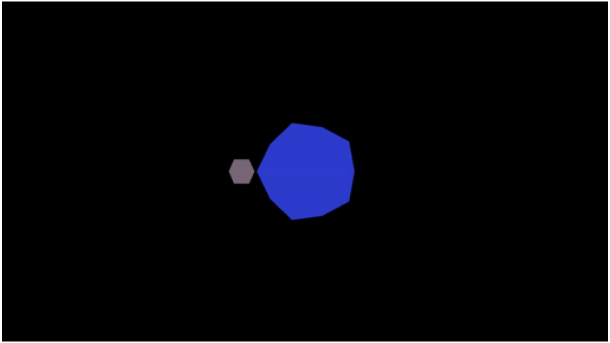
- `dot(a, b)`: Dot product of two vectors.
- `cross(a, b)`: Cross product of two `vec3`.
- `normalize(v)`: Normalize vector `v` to unit length.
- `length(v)`: Returns the magnitude of vector `v`.
- `mix(x, y, t)`: Linearly interpolates between `x` and `y`.
- `clamp(x, min, max)`: Clamps `x` between `min` and `max`.

test-01

Make sure your OpenGL and shader setup is correct. No implementation required.



test-02 I

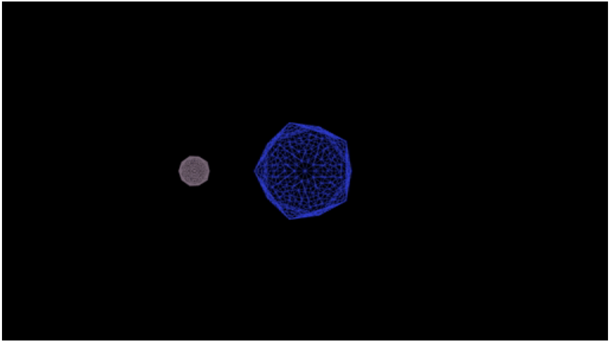


test-02 II

Files to be implemented:

- `src/identity.glsl`
- `src/uniform_scale.glsl`
- `src/rotate_about_y.glsl`
- `src/model.glsl`
  - Transform the moon as per the comments in `model_view_projection.vs`.
- `src/model_view_projection.vs`
- `src/blue_and_gray.fs`
  - Any reasonable choice of blue and gray will be accepted.

test-03 I

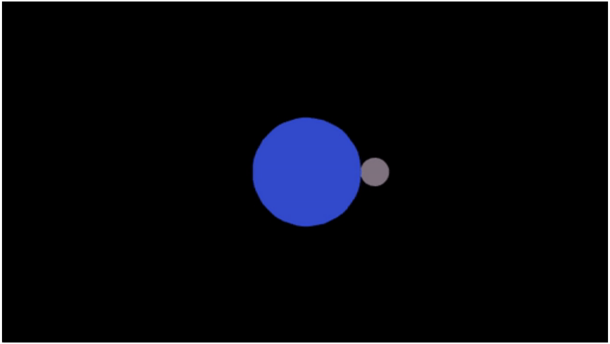


test-03 II

Files to be implemented:

- `src/5.tcs`
  - This is similar to the pass-through shader discussed earlier.

test-04 I

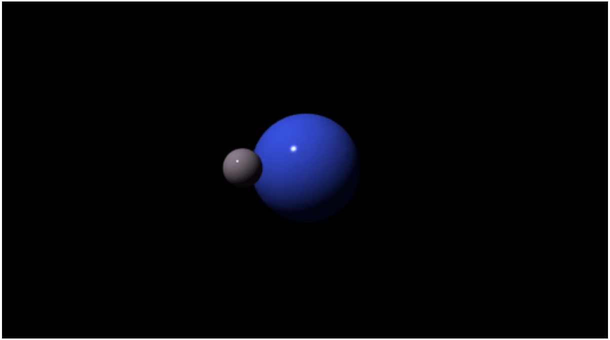


test-04 II

Files to be implemented:

- `src/snap_to_sphere.tes`
  - Find the coordinate of the closest point on the sphere and its normal using `interpolate` and its barycentric coordinates.
  - Apply appropriate transformations to the position and the normal.

test-05 I

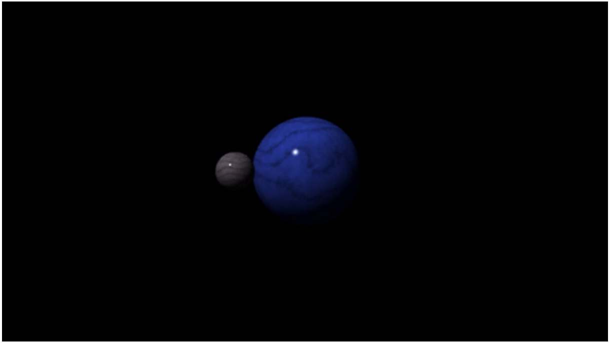


test-05 II

Files to be implemented:

- `src/blinn_phong.glsl`
  - Choose a reasonably low ambient intensity.
- `src/lit.fs`
  - Select reasonable light frequency, direction, and specular exponent (`p`).

test-06 I

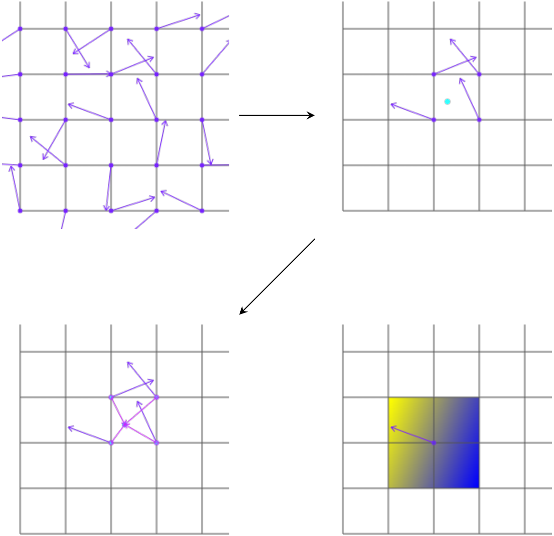


test-06 II

Files to be implemented:

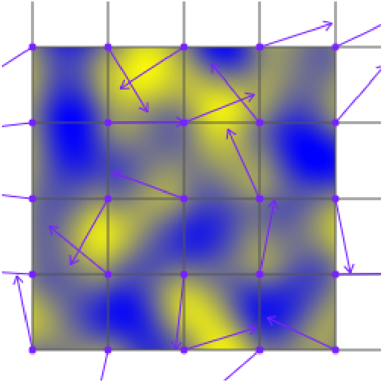
- `src/random_direction.glsl`
  - Hint: Consider sampling uniformly on a sphere using `random2`
- `src/smooth_step.glsl`
- `src/perlin_noise.glsl`
- `src/procedural_color.glsl`
  - Be creative! Output doesn't have to match the example.

test-06 III



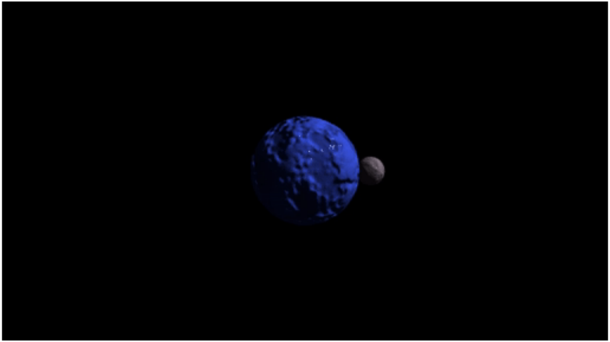
1

test-06 IV



[1]Credit: `https://www.huttar.net/lars-kathy/graphics/perlin-noise/perlin-noise.html`

test-07 I

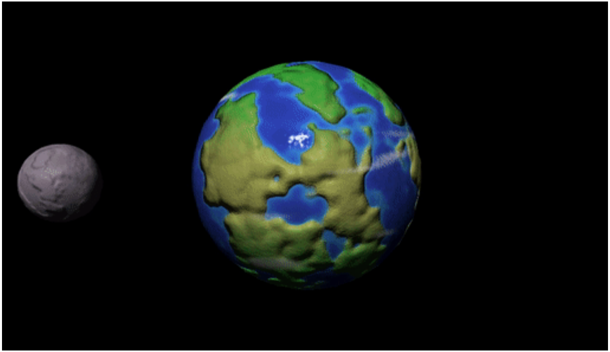


test-07 II

Files to be implemented:

- `src/improved_smooth_step.glsl`
- `src/improved_perlin_noise.glsl`
  - Replace the `smooth_step` call with `improved_smooth_step`. There is no need to change the pseudorandom gradient vector calculation
- `src/bump_height.glsl`
  - You can be creative with the improved Perlin noise here. Make the result visually interesting—it doesn't need to match the example exactly.
- `src/bump_position.glsl`
- `src/tangent.glsl`
- `src/bump.fs`
  - Be creative! Output doesn't have to match the example.

test-08



Files to be implemented:

- `src/planet.fs`
  - Be creative! Output doesn't have to match the example.

Useful Resources

Take a look at the following links for more information

- Learn OpenGL
- OpenGL Wiki
- The Book of Shaders
- Perlin Noise (Wikipedia)
- Understanding Perlin Noise