COMP220: Graphics & Simulation

7: Lighting

Worksheet Schedule

Worksheet	Start	Formative deadline
1: Framework	Week 2	Mon **15th Feb** 4pm (Week 4)
2: Basic scene	Week 4	Mon **1st Mar** 4pm (Week 6)
3: Plan/prototype	Week 6	Mon **15th Mar** 4pm (Week 8)
4: Final iteration	Week 8	Mon **12th Apr** 4pm (Week 10)

Learning outcomes

By the end of this week, you should be able to:

- **Explain** the Blinn-Phong illumination model
- **Describe** how effects such as normal mapping can be used to enhance appearance
- **Implement** basic lighting effects in your scene

Agenda

Agenda

- Lecture (async):
  - **Calculate** the colours in a lit scene using the Blinn-Phong model.

Agenda

- Lecture (async):
  - **Calculate** the colours in a lit scene using the Blinn-Phong model.
- Workshop (sync):
  - **Create** a light source and use it to illuminate objects in the scene.
  - **Recap** and **extend** the use of uniforms to pass data to shaders.

Schedule

16:00-16:10	Arrival, sign-in & overview
16:10-16:40	Demo & Exercise: Let There be (a) Light
16:40-17:00	More on Uniforms
17:00-18:00	Exercise: Experimenting with lights

FALMOUTH
UNIVERSITY

More on uniforms

Uniform structs

In GLSL:

```
struct Light {
    vec3 dir;
    vec3 colour;
};
uniform Light light;

// ... //
vec3 lightNorm = normalize(light.dir);  // etc.
```

Uniform structs

In C++:

```
unsigned int lightDirLoc =
        glGetUniformLocation(programID, "light.dir");
unsigned int lightColourLoc =
        glGetUniformLocation(programID, "light.colour");

// ... //
glUniform3f(lightDirLoc, 0.0f, 1.0f, 0.0f);
glUniform3f(lightColourLoc, 1.0f, 1.0f, 1.0f);
```

Uniform blocks

Uniform blocks

- Allow **uniform sharing** between shaders.

Uniform blocks

- Allow **uniform sharing** between shaders.
- Allow setting multiple values at once.

Uniform blocks

- Allow **uniform sharing** between shaders.
- Allow setting multiple values at once.
- Can specify a **storage mode** to determine how the memory is laid out (default is implementation-dependent).

Uniform blocks

- Allow **uniform sharing** between shaders.
- Allow setting multiple values at once.
- Can specify a **storage mode** to determine how the memory is laid out (default is implementation-dependent).
  - Either **shared** or **packed** to remove unused variables (not shareable).

Uniform blocks

- Allow **uniform sharing** between shaders.
- Allow setting multiple values at once.
- Can specify a **storage mode** to determine how the memory is laid out (default is implementation-dependent).
  - Either **shared** or **packed** to remove unused variables (not shareable).
  - May require **padding** to ensure alignment.

Uniform blocks

- Allow **uniform sharing** between shaders.
- Allow setting multiple values at once.
- Can specify a **storage mode** to determine how the memory is laid out (default is implementation-dependent).
  - Either **shared** or **packed** to remove unused variables (not shareable).
  - May require **padding** to ensure alignment.
- Connected to OpenGL buffers via **binding points** that link the block and buffer indices.

Uniform blocks

In GLSL:

```
uniform LightBlock {
    vec3 lightDir;
    vec3 lightColour;
};

// ... //
vec3 lightNorm = normalize(lightDir);    // etc.
```

Uniform blocks

In C++:

```
float lightValues[] = {
                -0.5, 0.7, 1.0, // lightDir
                0.0f,           // padding for alignment
                0.2, 0.6, 0.0   // lightColour
                };

GLuint bindingPoint = 1, uniformBuffer, blockIndex;
blockIndex = glGetUniformBlockIndex(programID, "LightBlock");
glUniformBlockBinding(programID, blockIndex, bindingPoint);

glGenBuffers(1, &uniformBuffer);
glBindBuffer(GL_UNIFORM_BUFFER, uniformBuffer);
glBufferData(GL_UNIFORM_BUFFER, sizeof(lightValues),
             lightValues, GL_STATIC_DRAW);
glBindBufferBase(GL_UNIFORM_BUFFER, bindingPoint,
             uniformBuffer);
```