



COMP220: Graphics & Simulation

7: Lighting & Post-processing

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** post-processing effects in your application

Agenda

- ▶ Lecture (async):
 - ▶ **Calculate** the colours in a lit scene using the Blinn-Phong model.
 - ▶ **Introduce** a variety of 2D post-processing effects.
- ▶ Workshop (sync):
 - ▶ **Create** a light source and use it to illuminate objects in the scene.
 - ▶ **Implement** a selection of post-processing effects by rendering to a texture.

Vector products

Dot and cross product

$$a \cdot b = |a||b|\cos\theta$$

where θ is the **angle** between a and b

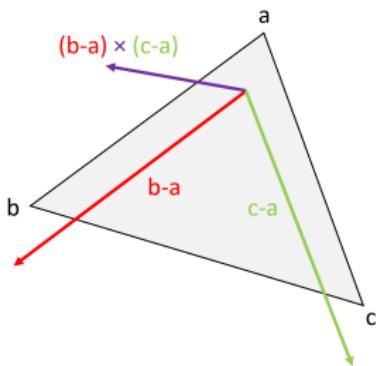
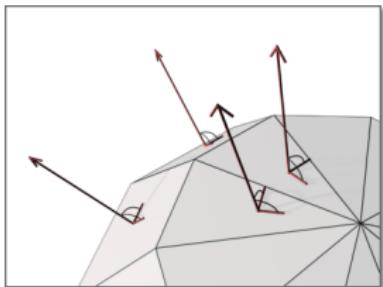
$$a \times b = (|a||b|\sin\theta)n$$

where n is a unit vector **perpendicular** to both a and b
with direction given by the **right-hand rule**

Uses

- ▶ Both dot and cross product are **quick to calculate**
- ▶ Dot product can be used to find the **angle** between vectors
 - ▶ Actually the **cosine** of the angle
 - ▶ If $a \cdot b = 0$ (and a, b are non-zero) then $\cos \theta = 0$, i.e. $\theta = 90^\circ$: a and b are **perpendicular**
 - ▶ If $a \cdot b = 1$ and a, b are unit vectors then $\cos \theta = 1$, i.e. $\theta = 0^\circ$: a and b are **parallel**
- ▶ Cross product can be used to find a vector **perpendicular** to two others

Surface normals



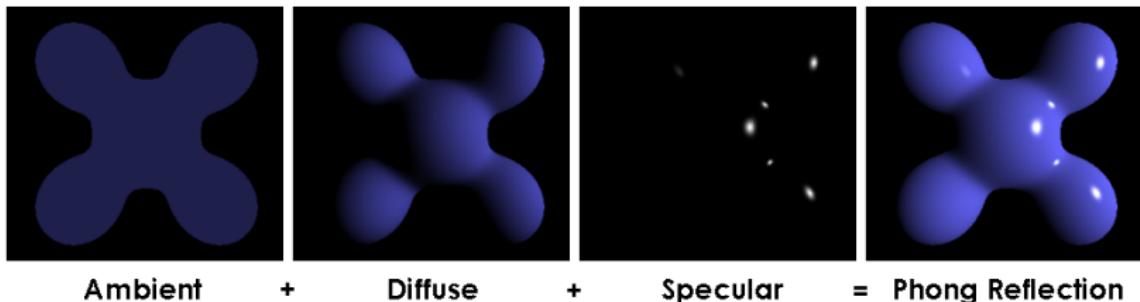
- ▶ The **normal** to a surface is a **unit vector** that is **perpendicular** to the surface
- ▶ If we have two non-parallel vectors that are **tangent** to the surface, we can use the **cross product** to find the normal
- ▶ For a triangle with vertices a, b, c , two such vectors are $b - a$ and $c - a$
- ▶ So the normal is
$$\frac{n}{|n|} \quad \text{where} \quad n = (b - a) \times (c - a)$$

The Blinn-Phong illumination model

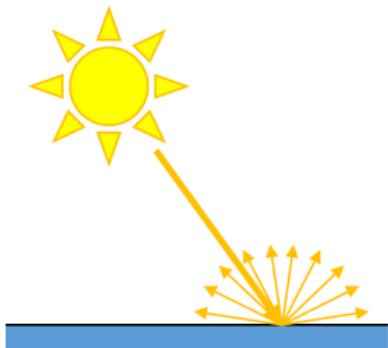
The Blinn-Phong illumination model

Jim Blinn, "Models of light reflection for computer synthesized pictures". *ACM SIGGRAPH Computer Graphics*, 11(2):192–198, 1977.

- ▶ The Blinn-Phong model builds on the Phong shading model
- ▶ It breaks lighting down into three parts: **ambient**, **diffuse**, and **specular**.

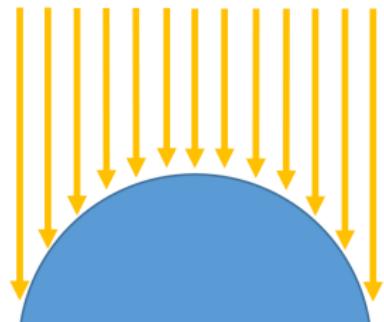


Diffuse lighting

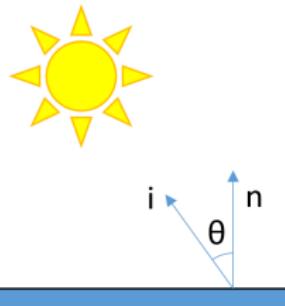


When light hits a “rough” surface, it is
scattered equally in all directions

The amount of light hitting the surface
depends on the **angle** between the
surface and the light source



Diffuse lighting formula

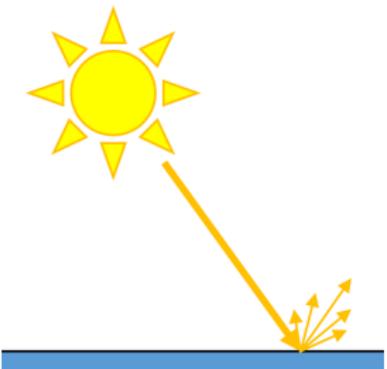


- ▶ Light intensity is proportional to the **cosine** of the angle between the **light direction** and the **surface normal**
- ▶ Let n be the normal, and i be a unit vector pointing towards the light source
- ▶ Light intensity is proportional to $\cos \theta = n \cdot i$
- ▶ If the surface is **pointing away** from the light source, we get $\theta > \frac{\pi}{2}$ so $\cos \theta < 0$ — in this case we **clamp** the answer to 0

Light direction and intensity

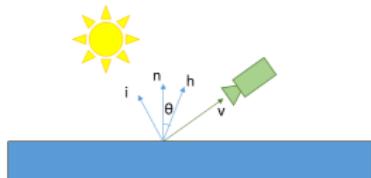
- ▶ For a distant light source (e.g. the sun), direction and intensity are **constant**
 - ▶ These are known as **directional** lights
- ▶ For a **point** light source (e.g. a lightbulb):
 - ▶ Direction is calculated by subtracting the fragment position from the light position
 - ▶ Intensity is usually **attenuated**, e.g. using an **inverse square law**: if the distance between the fragment and the light source is d , then the light intensity is proportional to $\frac{1}{d^2}$

Specular lighting



- ▶ When light hits a “smooth” surface, it is **reflected** across a narrow range of angles
- ▶ This creates a **specular highlight**, with intensity dependent on the **view direction** as well as the direction to the light source.

Specular lighting formula



- ▶ Let h be the “halfway vector”, calculated by $i + v$
- ▶ The closer h is to n , the higher the specular contribution.
- ▶ Specular light intensity is proportional to

$$\text{clamp}(n \cdot h)^s$$

where s is a “shininess” parameter, and $\text{clamp}(x)$ clamps its argument between 0 and 1

Ambient lighting

- ▶ Currently, surfaces pointing away from the light are completely **black** (light intensity = 0)
- ▶ In the real world, light scattered from one surface illuminates others
- ▶ In the Phong model, we cheat and add a little **ambient** intensity to the lighting, which is just the light colour applied to the object colour with a constant “ambient factor”.
- ▶ Another option would be to add more light sources...

Normal mapping



- ▶ A **normal map** is a texture which is used to slightly alter the normal across a surface
 - ▶ Each pixel in the normal map represents a 3D vector, with xyz mapped to RGB
- ▶ Can be used to add detail to flat, low-poly surfaces
- ▶ Can use textures to change other lighting parameters across a surface, e.g. **specular mapping**

Post-processing

What is Post-Processing?

- ▶ A series of techniques that are carried out **after** a scene has been rendered
- ▶ Typically mimic effects caused by camera (or eye) hardware, e.g. lens flare, bloom, motion blur
- ▶ Traditionally this could only be done as an offline process
- ▶ With the advent of faster GPUs and programmable shaders, we can implement some of the effects in real time

Post-Processing Stages

- ▶ The key to the effect is to switch from drawing to the back buffer to a **texture**
- ▶ This texture will contain the current view of the scene
- ▶ We then use a shader to implement a post-processing effect
- ▶ We then map the processed texture onto a full screen quad, which is rendered to the backbuffer

The Frame Buffer

- ▶ We use several types of **screen buffers** when rendering with OpenGL:
 - ▶ The **colour buffer** stores the pixel output from the fragment shaders.
 - ▶ The **depth buffer** (or z-buffer) stores information for depth-testing.
 - ▶ A **stencil buffer** can be used to mask out certain fragments.
- ▶ A **framebuffer** is just a combination of different buffers.
- ▶ A default framebuffer is created for you; you can create additional ones to render to instead.

COMP120 Refresher

Image Processing Techniques

- ▶ Some of the algorithms you learned in COMP120 can be applied in GLSL:
 - ▶ Colour Replacement
 - ▶ Luminance calculation
 - ▶ Colour Correction
 - ▶ Black & White and Sepia Tone
 - ▶ Edge Detection
- ▶ Key difference: we access the data via **texture coordinates**, rather than array indices.

Modifying Textures

- ▶ A Texture Lookup uses the texture coordinates passed to the fragment shader
- ▶ Additionally, the fragment shader only processes a single fragment at a time
- ▶ If we want to access a pixel adjacent to the current one, we can offset the current texture coordinates - this is especially useful for edge detection
- ▶ GLSL has lots of inbuilt functions (e.g distance) which can aid in creating post-processing effects

Other Post Processing Techniques

Colour Correction

- ▶ Process of taking an image, convert each colour in the image to a some other colour
- ▶ Mimic a specific film stock, provide coherent look or provide a mood



Original Shot



Day-for-Night Color Corrected shot

Colour Correction Again

- ▶ Depending on the technique this could involve manipulating the colours using a Filter Kernel
- ▶ Or we could use a colour palette (see Colour Grading)



Blur

- ▶ This often used as a basis for other techniques such as Depth of Field
- ▶ To achieve blurring we apply a filter to the texture lookup of the texture render target
- ▶ We can apply many different filters, one of the simplest is a Box Filter
- ▶ With a box filter we sample 4 points around the point we are interested in and take an average

Motion Blur

- ▶ If an object moves very fast through your Field of view it can appear that the object leaves a slight ghost of itself
- ▶ There are several ways of implementing this, we can use the actual speed of the object to determine the amount of blur



Motion Blur

- Or we can simply take the results of the last render update and blend them with current render results and blend based on a lerp



Depth of Field

- ▶ DOF attempts to simulate the effect where objects that are too close or too far away appear out of focus
- ▶ We first have to consider in our scene what range of depth values will be considered in focus. We then need to blur everything that is outside this range



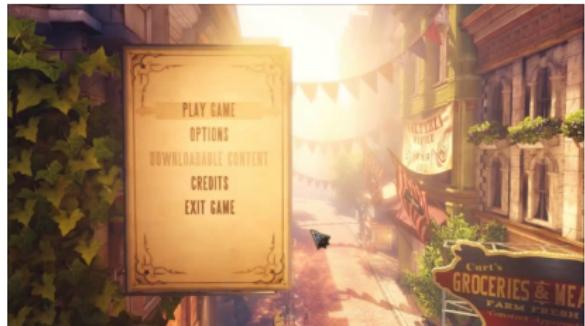
Depth of Field

- ▶ Once we have determined this we render a blurred scene to a render texture



Bloom

- ▶ This attempts to simulate the overloading of the optic nerve if extremely bright areas of a surface are visible
- ▶ First we render to a smaller render target, usually 1/4 size



Bloom

- ▶ We then apply a filter (usually Gauss) to this reduced render target, if we apply this filter in multiple passes then we achieve a smoother blur
- ▶ We then blend the blurred image onto the screen with the original scene



Post Processing Stack

- ▶ In theory we can chain post-processing effects together
- ▶ Essentially we keep rendering to a texture using different fragment shaders each time
- ▶ These could be stored in a data structure such as a stack
- ▶ Once we have completed the last effect on the stack, we then switch to normal rendering and display the result

Next steps

- ▶ **Review** the additional asynchronous material for more background on the lighting equation and post-processing effects.
- ▶ **Attend** the workshop to practice creating a lit scene and modifying textures generated from the framebuffer.