

The background of the slide is a faded, pixelated image of a Super Mario Bros. level. It features a blue sky with white clouds, a green pipe on the left, a brick block with a question mark in the center, and a brick floor at the bottom. There are also some green hills and a small red flower on the right side.

Intro to Lighting

GAME 300

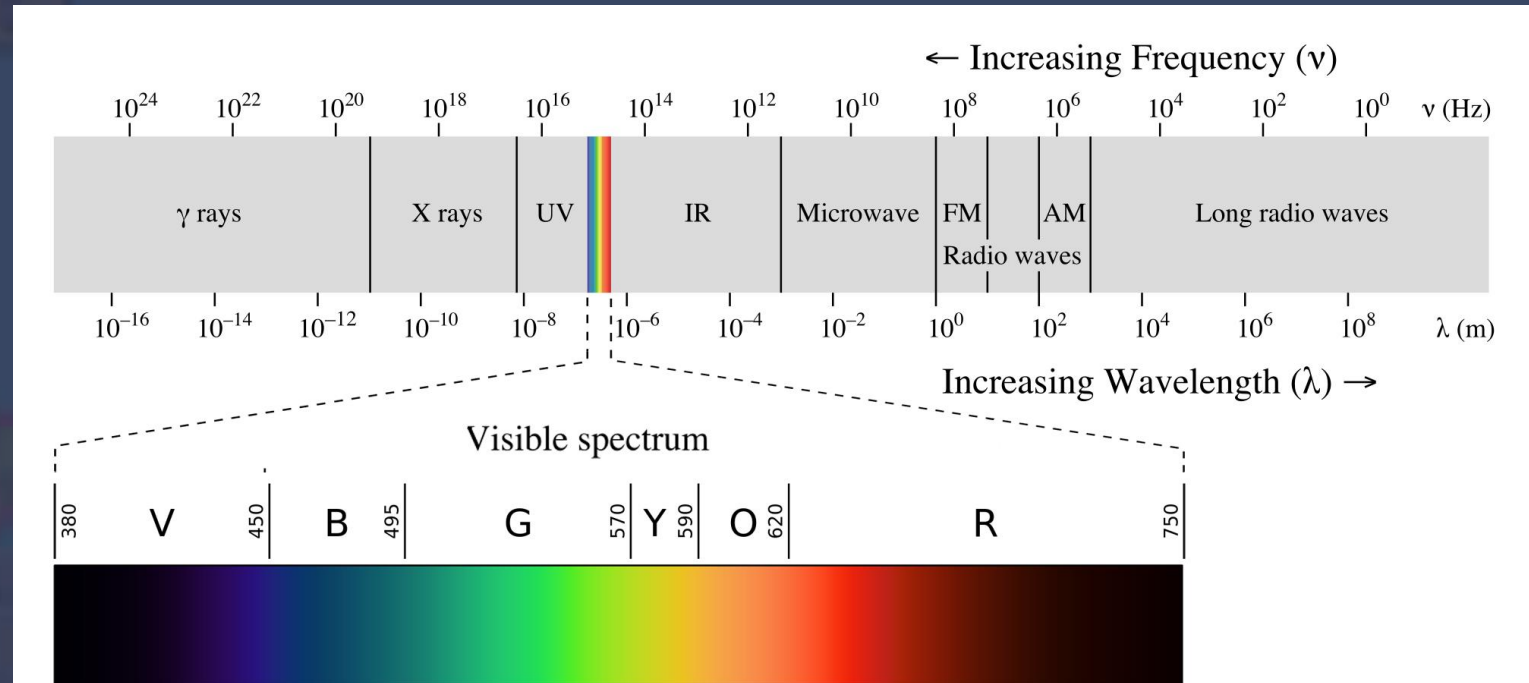
James Dupuis

Objectives

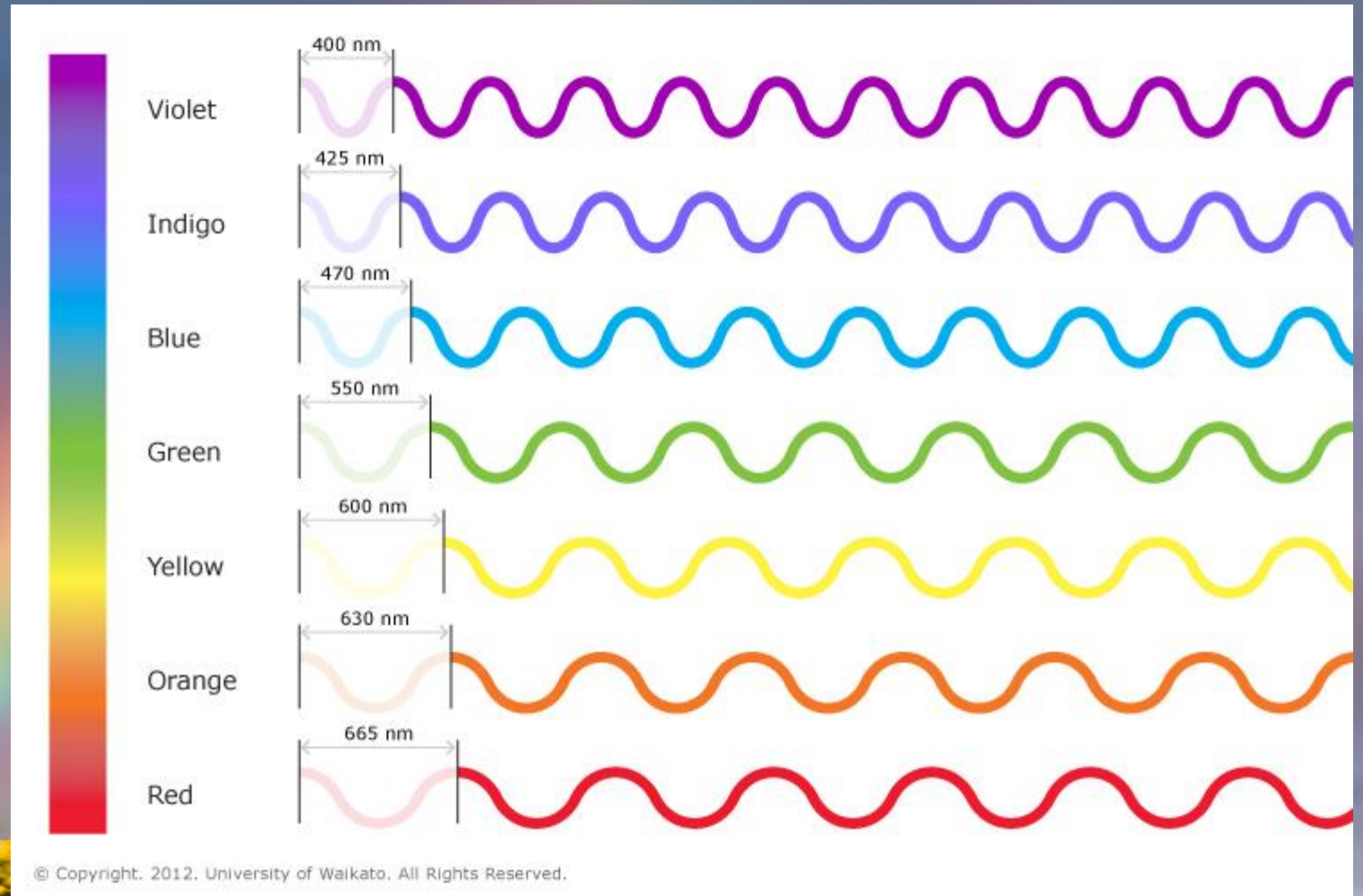
- Evaluate lighting techniques
- Explain the difference between a Material and a texture
- Describe reflection and refraction
- Describe how properties of object effect their interaction with lights.

Basics of Lights

- Lights in real life move if waves similar to sound / radio waves, Microwaves and even our Wifi all around us.
 - Light travels in electromagnet waves in a visible frequency to humans between 400 -> 700 nanometers in wavelength.
- That being said, light is also a form of energy.
- Other things which travel on wavelengths:
 - UV Rays
 - X-Rays
 - Gamma Rays



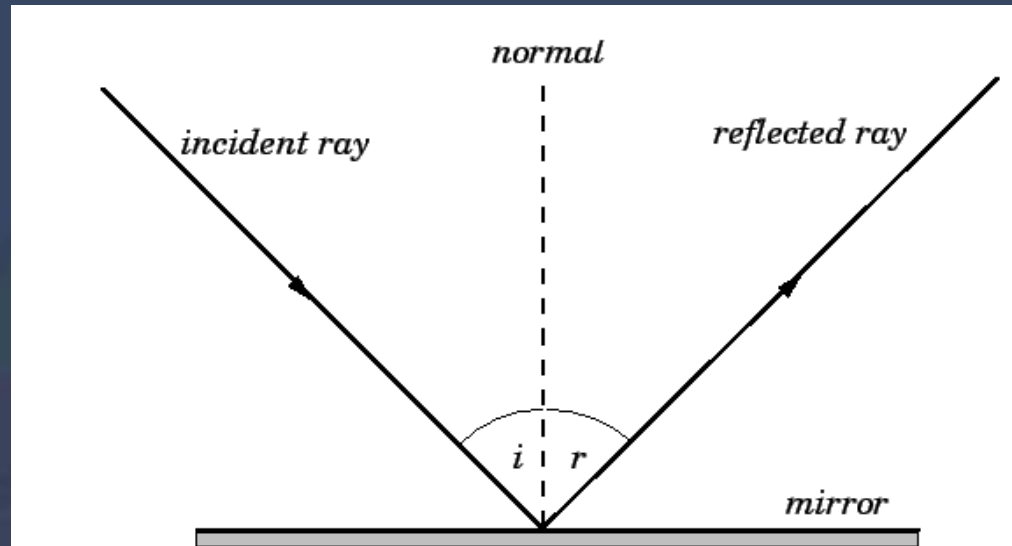
Colour Wavelengths



- <https://www.sciencelearn.org.nz/resources/47-colours-of-light>

REFLECTION

- The three laws of reflection:
 - The **incident** ray, the **reflected** ray and the **normal** to the reflection surface at the point of the incidence lie in the same plane.
 - The angle which the incident ray makes with the normal is **equal** to the angle which the reflected ray makes to the same normal.
 - i & r
 - The reflected ray and the incident ray are on the opposite sides of the normal.

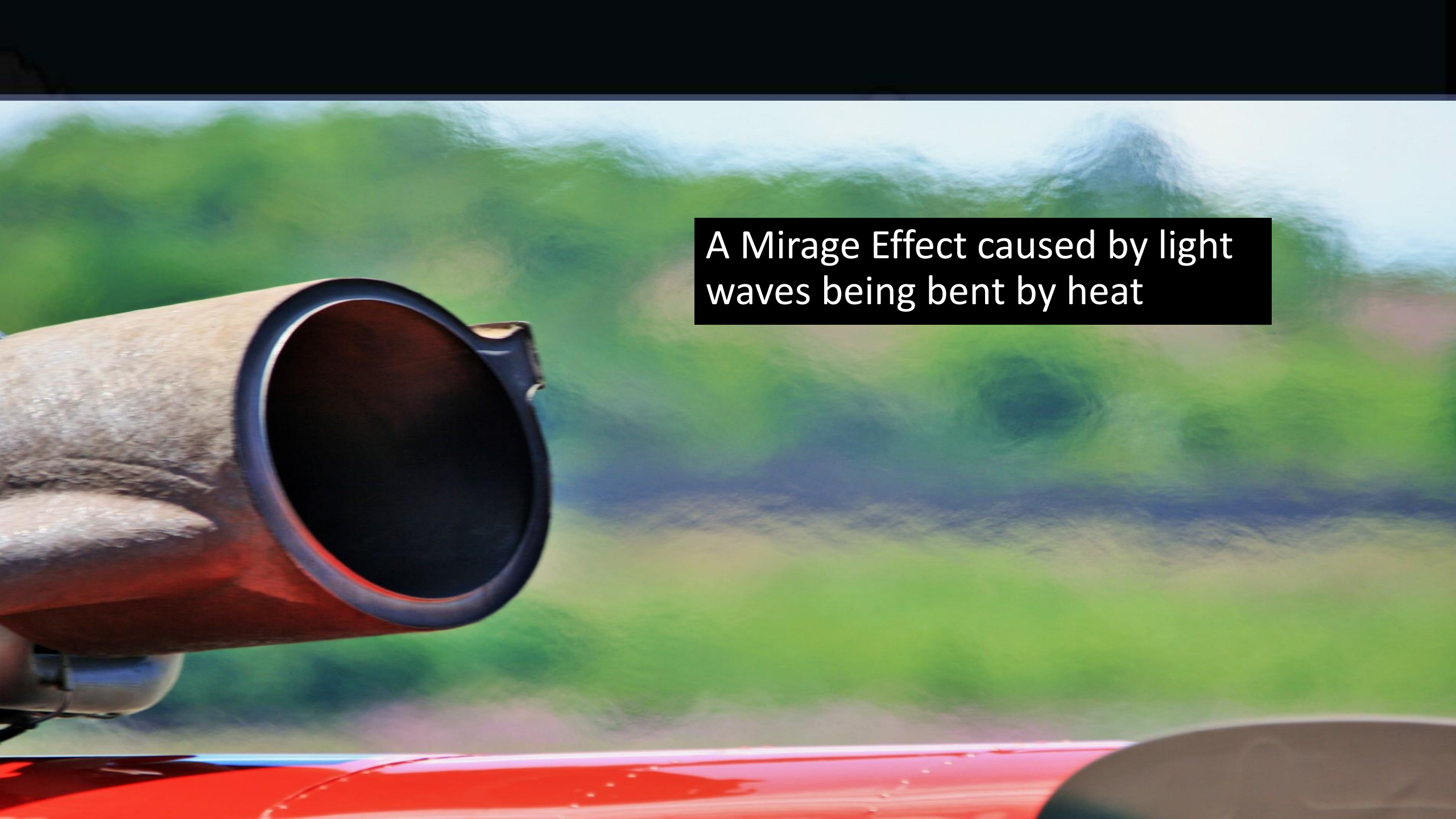


ATTENUATION

- The farther away a light is the less effective it should become.
 - This is known as light attenuation.
- Can sometimes be written as the intensity value of a light.
- Light, like sound, travels through the airwaves... or other materials... and breaks down as it progresses farther and farther.
 - Depending on the density of the material lights impedance changes.
 - In reality colours of light actually have different properties for how far they can reach.

REFRACTION

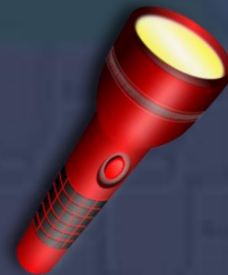
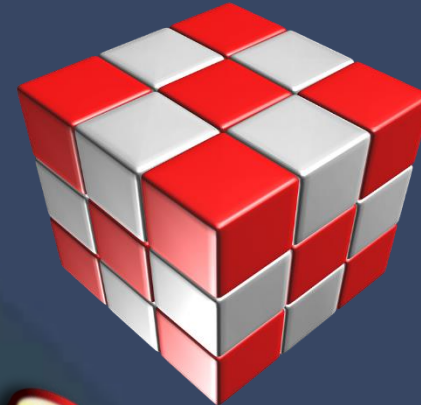
- Refraction is what happens to light or sound when it passes through a material or object.
 - The material itself slows down the speed of light altering the angle at which it advances.
 - This is known as a bend.
 - Different materials have can have a different value for the **Index of Refraction**
 - How much light bends when intersecting it.
- As an example water has a **Refraction Index** of 1.333333.
 - This means that light travels at 75% the typical rate through it pierces the air.
 - A Diamond is at roughly 40% with an index of 2.417
 - <https://www.sciencelearn.org.nz/resources/49-refraction-of-light>



A Mirage Effect caused by light waves being bent by heat

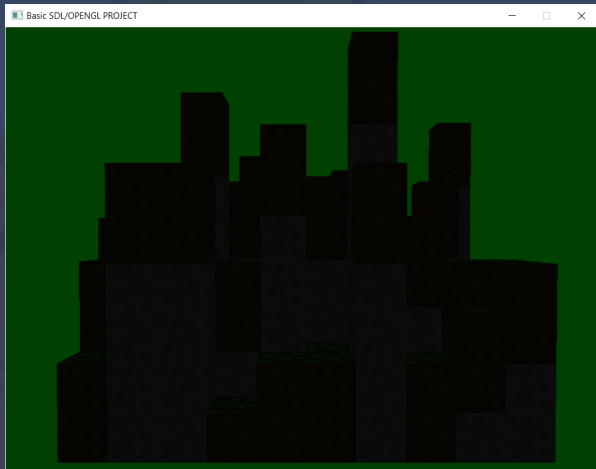
Lighting in Games

- Lighting is the backbone to a shaders capabilities.
- Lights in shaders are designed to emulate the effects of real life light on objects within the world.
 - Instead of treating lights like waves though we typically treat them as direct beams/ rays.
- Lighting typically has three main things to consider when dealing with graphics programming:
 - The light source itself.
 - The object it is hitting.
 - The viewer of the object being hit by light.
- Can you take a guess at which shader will mostly control lighting?



Primitive Lighting

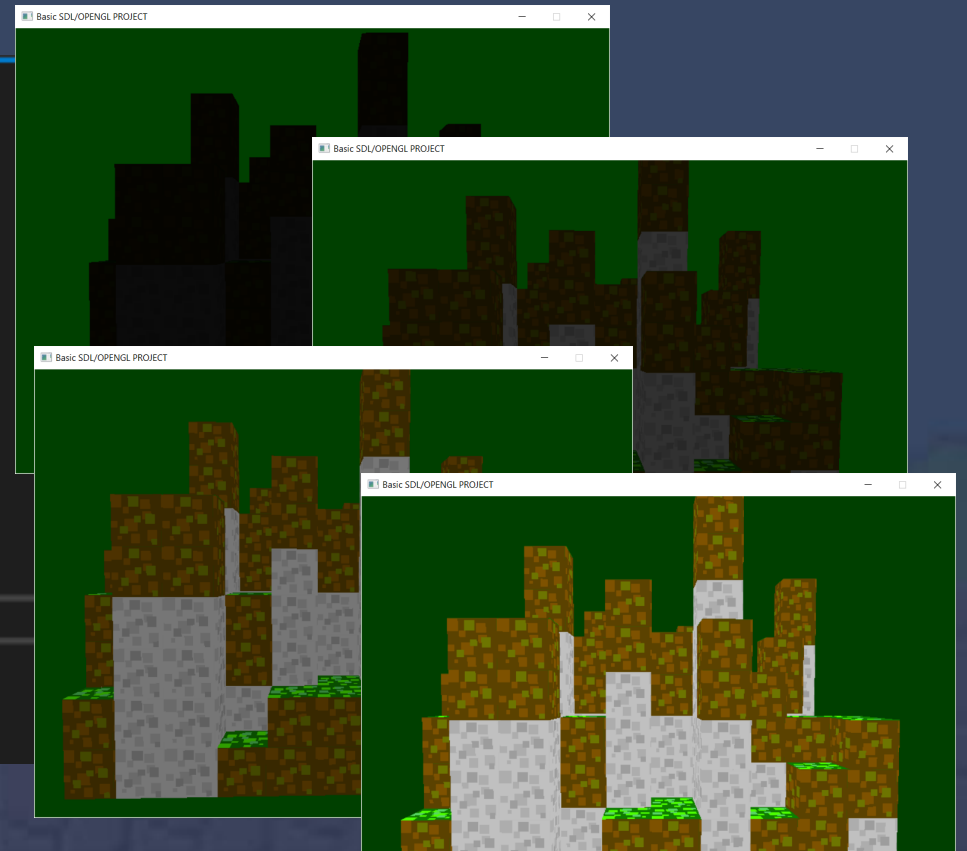
- The simplest form of lighting to add to a scene is our **ambient** light.
 - This is light that exists in the world without a source of power or starting point.
 - Just exists...
 - Typically considered by light that has been dispersed throughout the world through so many reflections and refractions it's original source is undetermined.



Creating Ambient Lighting

- Ambient lighting takes only a single value added to your Fragment Shader as a float.
 - This can be passed in as a uniform variable and applied directly to the output fragment color.

```
1  #version 430 core
2
3  out vec4 color;
4
5  in vec2 UV;
6
7  uniform sampler2D texture0;
8
9  uniform vec3 ambientLight = vec3(0.2,0.2,0.2);
10
11 void main(void)
12 {
13     color = texture( texture0, UV ) * vec4(ambientLight, 1.0f);
14 }
```



Creating Ambient Lighting

- The uniform must then be passed in from the OpenGL C++ code through a `glUniform3fv` function call after establishing a handle using the `glGetUniformLocation` OpenGL API call.

1. Establish a location in the shader to pass the uniform float data to:

```
AmbientUniformHandle = glGetUniformLocation(programObj, "ambientLight");
```

2. Pass data into the shader for the light intensity, here AmbientLight is a local array modified in the C++ code.

```
float LightVals[3] = { 0.2f, 0.2f, 0.2f };  
glUniform3fv(AmbientUniformHandle, 1, &LightVals[0]);
```

Creating Ambient Lighting

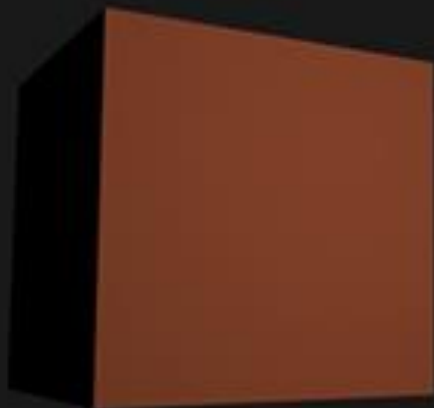


PHONG LIGHTING PROPERTIES

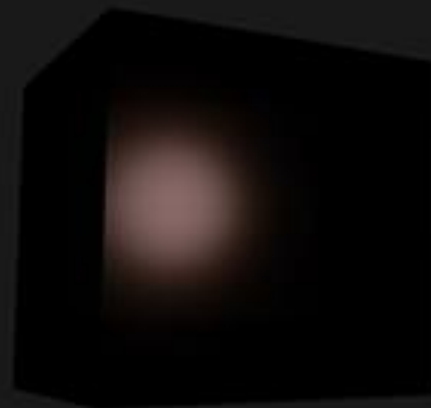
- Phong Lighting attempts to bring more real world lighting effects in a simplified way to graphics Programming.
- In addition to Ambient Light, it adds in two new properties of Diffuse and Specular Highlights



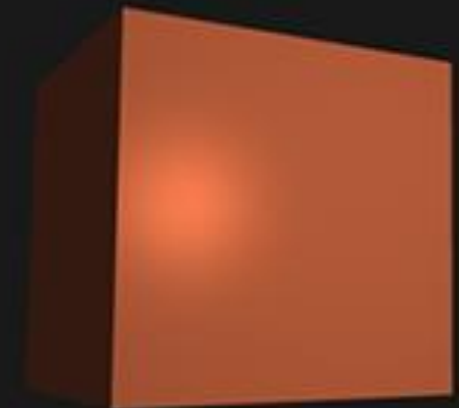
ambient



diffuse



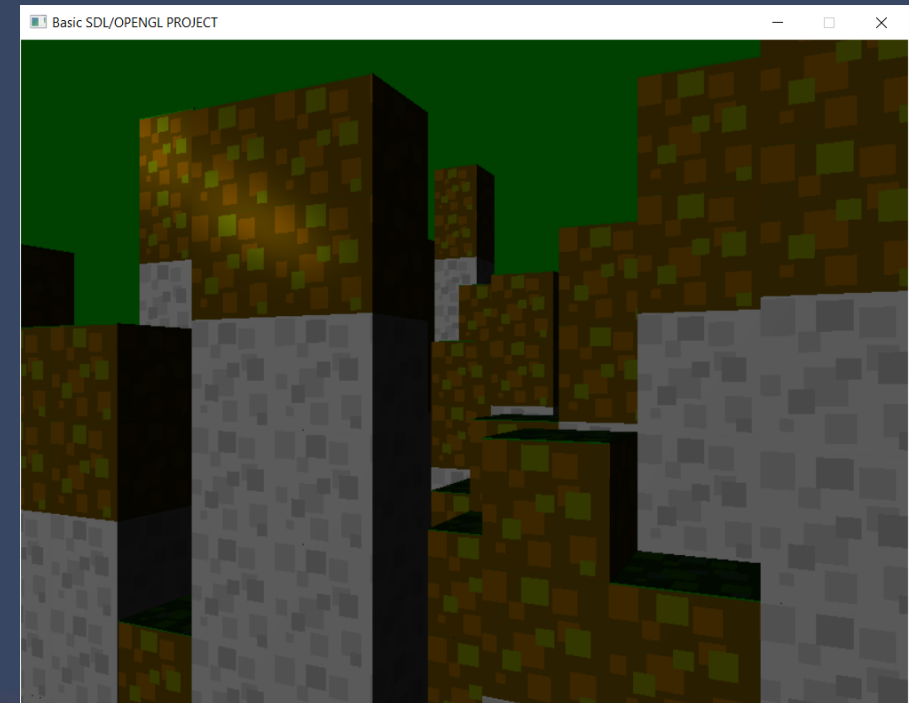
specular



combined (Phong)

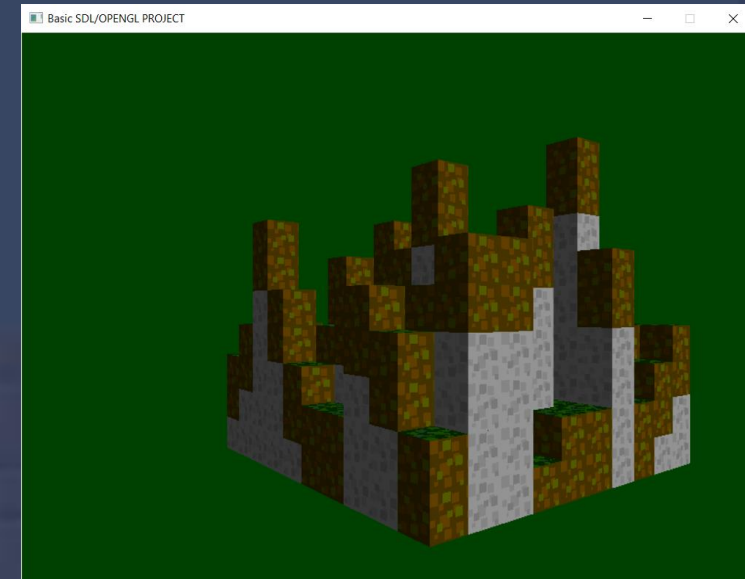
PHONG LIGHTING PROPERTIES

- Diffuse:
 - Property of a light is used to determine how focused the light is on a particular direction/ it's center of power.
 - The diffuse value will also determine how much of that power of light energy is dispersed upon contact with an object.
 - Multiplied by the value of the material to get the diffuse.
- Specular:
 - Specular light handles the reflective capabilities of the light itself.
 - Laser beam would have the maximum amount of specularity as it doesn't disperse or die off upon reflection.



DETAILED LIGHTS IN OPENGL

- Detailed positional lighting is calculated by formulas within the GLSL shader code written by the programmers.
- Typically this happens per fragment so most of the calculations transpire in the fragment shader code.
- Each individual fragment must be passed data for three separate things in regards to calculating each fragment within world lighting.
 1. The location and information about the light.
 2. The location and information about the object being lit.
 - do they absorb light?
 - do they reflect light?
 - does light pass through them?
 - These considerations are known as the material of the object.
 3. Lastly, shaders need to be aware of the location of the viewer in respect to the object and the light.
 - The Cameras position



OTHER CONSIDERATIONS

- Considerations need to be made per object in regards to lights:
 - How many light sources?
 - Where are each of them positioned?
 - What is each lights value for the following?
 - Ambient
 - Diffuse
 - Specular
- Most of lighting calculations are about creating triangles of reflection between the following:
 - Viewer (eye)
 - The Light
 - The normal (surface the light is bouncing off of) *Next Slide

NORMALS

- Direction of the face of a polygon/ triangle is facing.
 - Required for lighting calculations
 - Vector3
 - Full Length of the vector is 1 (unit length)
 - Typically assigned per vertex
 - Each vertex of a tri will all have the same normal
 - Vertices can be assigned more than 1 normal though.
- Can be calculated using the cross product of two vectors(**edges not vertices**) retrieved from the face of the triangle.
 - each triangle has 3 vertices.
 - (edge1) $A = V2 - V1$
 - (edge2) $B = V3 - V1$
 - Cross product of AB = Normals
- NormalViewer.exe

MATERIALS

- Materials have the same 3 properties of Lights
 - Ambience
 - Diffuse
 - Specular
 - + Emission.
 - Emission is used when an object itself emits a bit of light, regardless of the incoming light
- A newer concept for shaders utilizes what is known as PBR or Physically Based Rendering.
 - This instead looks at normal properties of an object analyzing it for:
 - Color
 - Roughness (matte value or diffuse)
 - Metallic (reflectivity)
 - Specular (shininess)

MATERIAL - DIFFUSE

- In regards to the material of an object, this value determines how much of the light is consumed by the object which the light hits.
 - A high diffuse value will diffuse the light meaning less light will bounce off the object
 - A lower diffuse value means that more light will bounce off the object through reflection.
- This generally determines how rough the objects surface may seem.
 - The more rough a surface is, the more directions light is reflected in causing it's power to seem diminished.

MATERIAL - SPECULAR

- This is the amount of light that is reflected off of the surface of an object.
- The smoother the object the higher specular value it will hold.
 - Plastic and Metals would have a high specular value to produce that glossy reflective look.
- Specular values of objects materials is often paired with it's shininess level.
- Defines the bright spot on objects where the shine is shown.

TYPES OF LIGHTS

- Directional
 - light comes from a specific direction / location
 - Cheap to calculate for the GPU
 - no attenuation (infinitely following direction specified)
 - Examples: sunlight
 - Light properties:
 - diffuse power
 - Ambient power
 - Specular Value
 - Direction (Position)
 - Formula:
 - $\text{Ambient} = \text{MatAmb} * \text{LightAmb}$
 - $\text{Diffuse} = \text{MatDiff} * \text{LightDiff} * \text{Normalize}(\text{dot}(\text{Normals}, \text{LightPos}))$
 - $\text{Specular} = \text{MatSpec} * \text{LightSpec} * \text{Normalize}(\text{dot}(\text{Normals}, \text{LightPos}))$

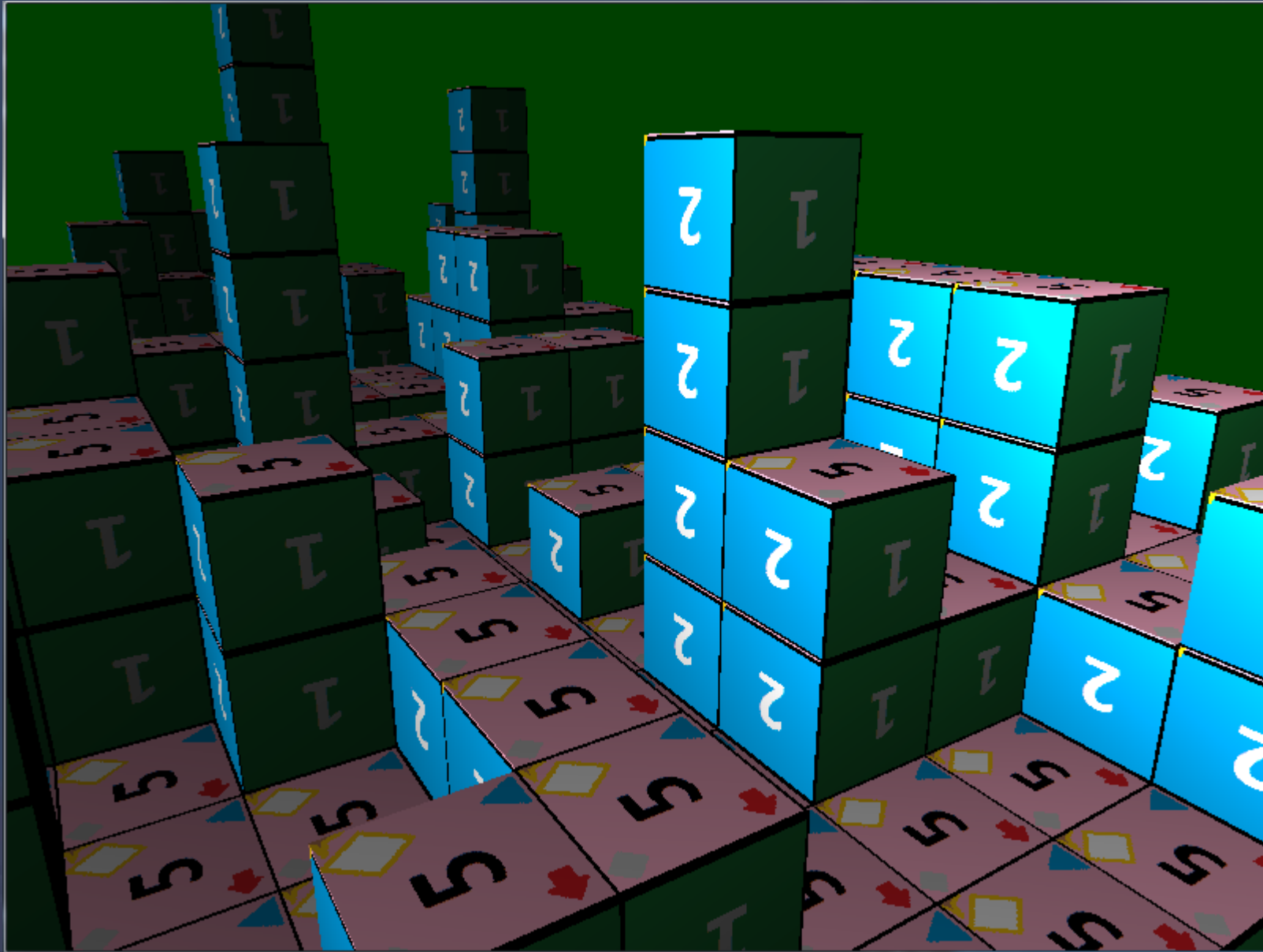
TYPES OF LIGHTS

- Point Light
 - Light is positioned at a location within the world.
 - Projects in a radius surrounding the position specified.
 - has an intensity value to determine its radius and attenuation over distance.
 - Example:
 - Light Properties:
 - diffuse power
 - Ambient power
 - Specular Value
 - Position
 - intensity (attenuation)

TYPES OF LIGHTS

- Spot Light
 - Similar to a cross between a point light and a directional light.
 - Has a location and a intensity
 - Has a directional cone which defines the area being lit.
 - Example: similar to a flashlight
 - Light Properties:
 - diffuse power
 - Ambient power
 - Specular Value
 - Position
 - intensity (attenuation)
 - radius

Time Lapse Shading Example



APPLYING NORMALS

- Using the same concept as Textures UV data, we use buffers to update our GLSL code with our large amount of data for an object.

```
glGenBuffers(1, &uv_buffer);  
glBindBuffer(GL_ARRAY_BUFFER, uv_buffer);  
glBufferData(GL_ARRAY_BUFFER,  
             sizeof(uv_buffer_data),  
             uv_buffer_data,  
             GL_STATIC_DRAW);  
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, NULL);  
glEnableVertexAttribArray(1);
```

- Normal Data can then be represented in our shader (vertex) using the following:

```
layout (location = 2) in vec3 normal;
```

Shader Variables for lighting

- Inside our Vertex Shader we need 3 things:

- A position for our light:

```
vec3 light_pos = vec3(100.0, 100.0, 100.0);
```

- The normals of our object to draw

```
layout (location = 2) in vec3 normal;
```

- Camera and Model Matrices:

```
uniform mat4 mv_matrix;  
uniform mat4 proj_matrix;
```

- The rest is all calculated through our math...

Lighting Calculations

- Each **vec3** is defined above as **out** variables passed into the fragment shader

```
out vec3 normalVec;  
out vec3 lightVec;
```

- Inside our Vertex Shaders main we calculate a few things:

```
// Calculate view-space coordinate
```

```
vec4 P = proj_matrix * mv_matrix * position;
```

```
// Calculate normal in view-space
```

```
normalVec = mat3(mv_matrix) * normal;
```


Lighting Calculations

- Each **vec3** is defined above as **in** variables passed in from the fragment shader:

```
// Input from vertex shader
in vec3 normalVec;
in vec3 lightVec;
```

- Inside our fragment Shaders main we need to normalize these values:

```
// Normalize the incoming N, L and V vectors
vec3 N = normalize(normalVec);
vec3 L = normalize(lightVec);
```

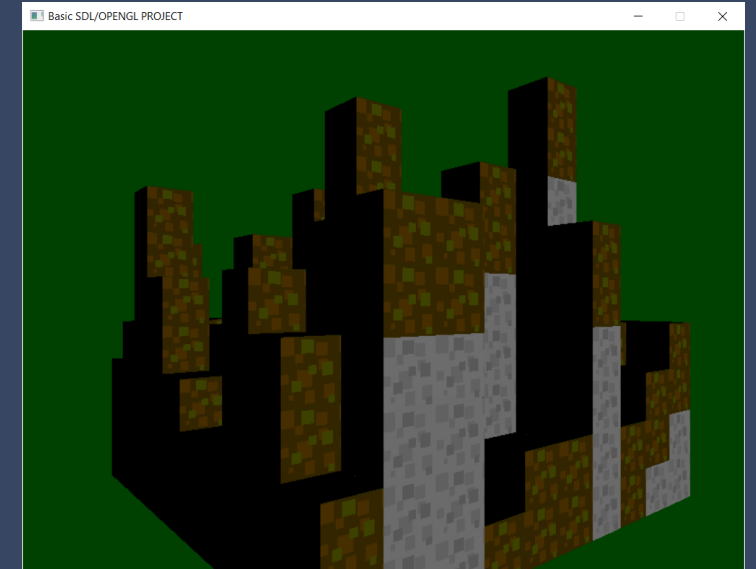
- `normalize()` is a built in function of the fragment shader

Applying the lighting

- Last step is to calculate the final dot product using a new variable as a multiplier:

```
// Material properties  
uniform vec3 diffuse_albedo = vec3(1.0, 1.0, 1.0);
```

- The `diffuse_albedo` is a property of the object being hit known as a material.
- We apply materials in combination with the dot product of the Normal and light to find the diffuse value of the lighting per fragment



```
// Compute the diffuse and specular components for each fragment  
vec3 diffuse = max(dot(N, L), 0.0) * diffuse_albedo;  
  
color = texture( texture0, UV ) * vec4(diffuse + ambientLight, 1.0f);
```

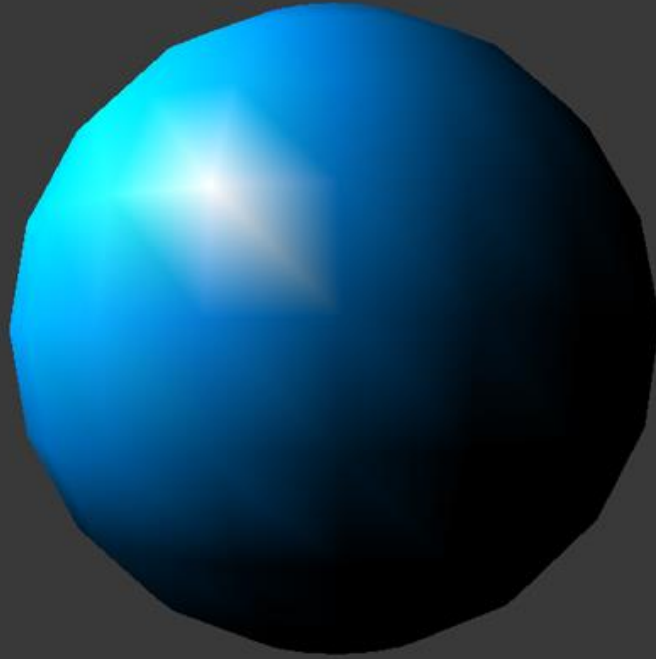
GLSL

- Which shader should we calculate lighting inside of?
- Gouraud shading
 - Handling the processing of the shading of light calculations on a per vertex level rather than in the fragment shader
 - This saves processing time as the vertex shader is only called once per vertex whereas the fragment shader is per fragment
 - Results are interpolated between the 3 vertices of the triangle by OpenGL when forwarded saving time within the individual fragment shader instances.
 - + saves time processing
 - - creates unrealistic linear shading (looks blockier instead of smooth)
- Phong Shading:
 - Uses a mixed approach to calculating light shading which results in a smoother effect
 - Some of the calculations and data retrieval is done in the vertex shader
 - Final calculations are done inside the fragment shader to perform a per fragment light shade value
 - + more realistic looking light and models appear smoother and less like their original polygons
 - - more strenuous on calculations
- phonglighting.exe V key

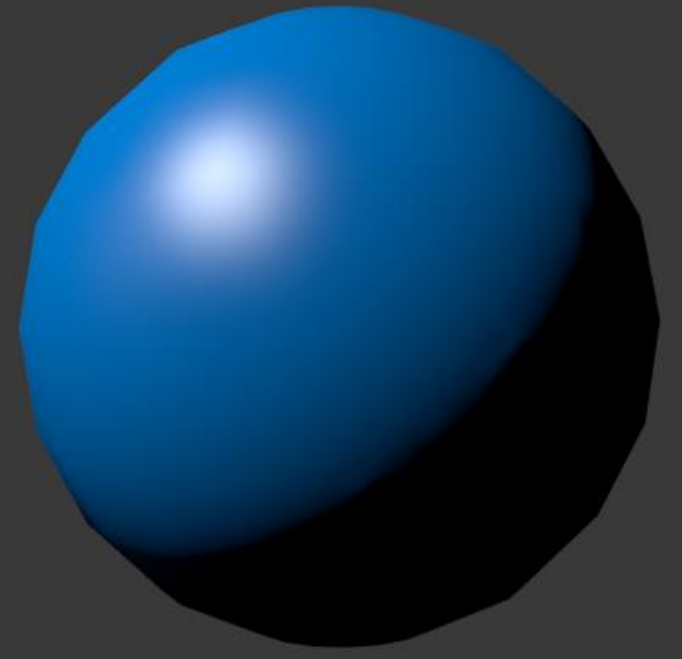
GLSL



FLAT SHADING



GOURAUD SHADING



PHONG SHADING