Lighting
Part 1

CMP301 Graphics Programming with Shaders

This week

- Lighting recap
  - Ambient light
  - Diffuse light
  - Specular light
- Reminder of dot/cross product
- Per-pixel lighting
- Lots of code

Lighting recap

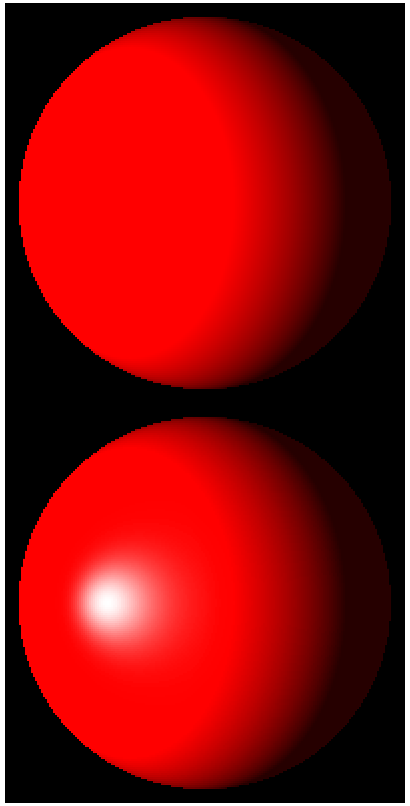
- Ambient light
  - Light bounced off the environment on the surface
  - Comes from everywhere
  - Minimum level of lighting

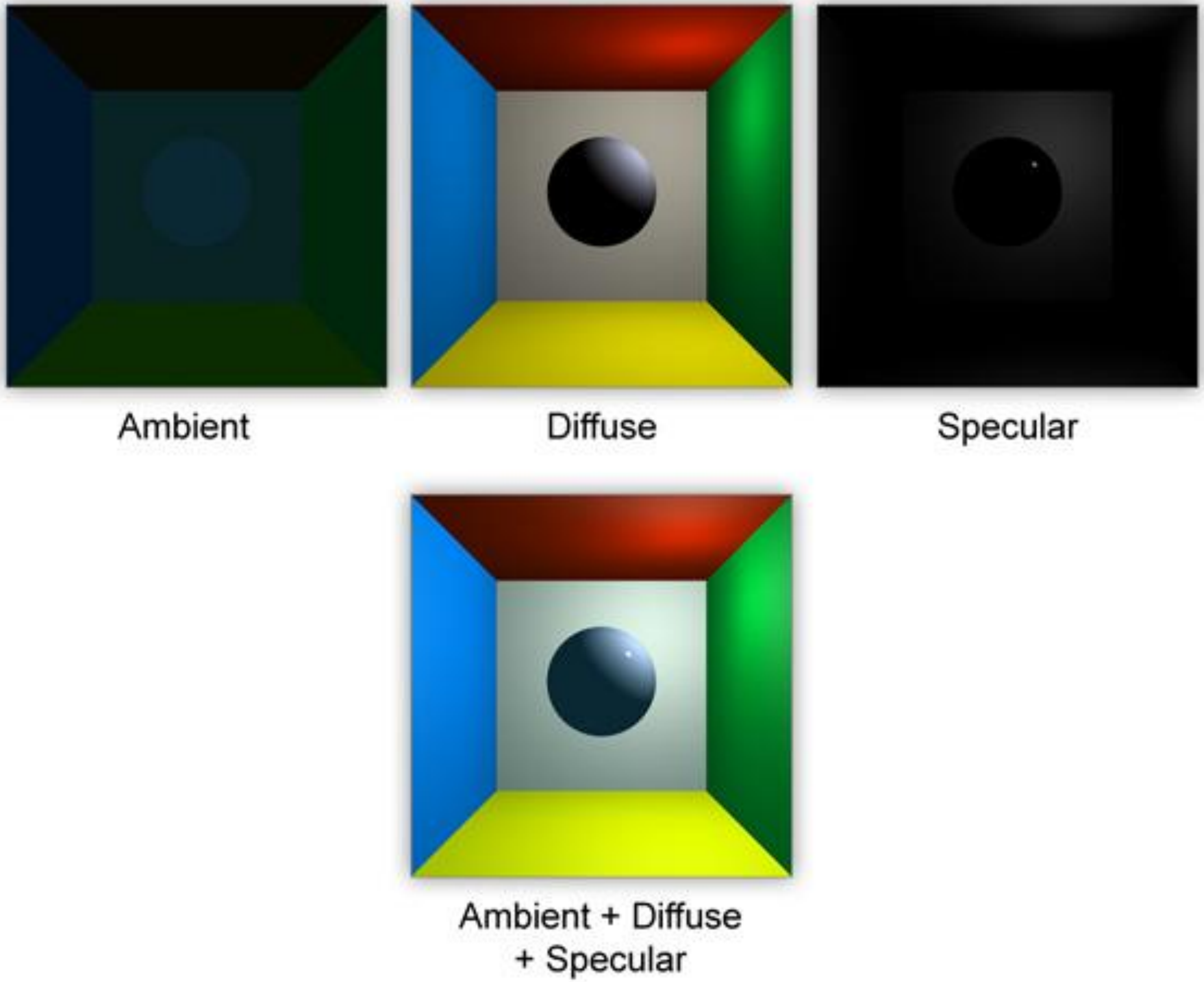
Lighting recap

- Diffuse light
  - Has a definable source
  - Is a light intensity that is modulated with the colour of the surface
  - Light at an angle to a surface should partially light the surface
  - Light behind a surface should not light it at all

Lighting recap

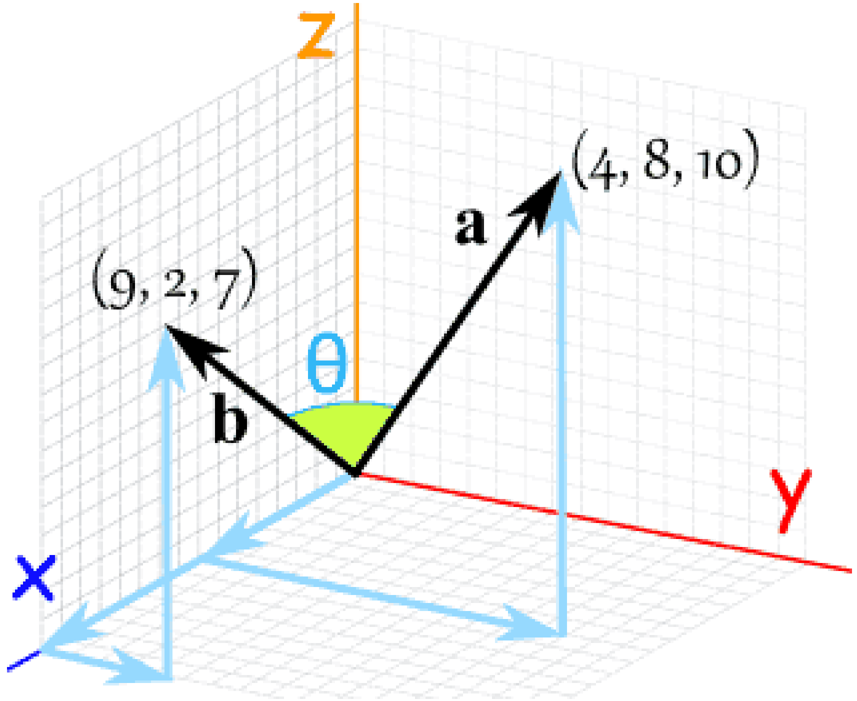
- Specular light
  - Similar to diffuse light
  - Additionally it simulates reflections of light as it bounces off the surface and hits the eyes
  - Shiny objects reflect light more sharply
    - Creating a highlight you see on an object
  - On non-shiny/non-smooth surfaces the specular light will be low or non existent




Ambient
Diffuse
Specular
Ambient + Diffuse
+ Specular

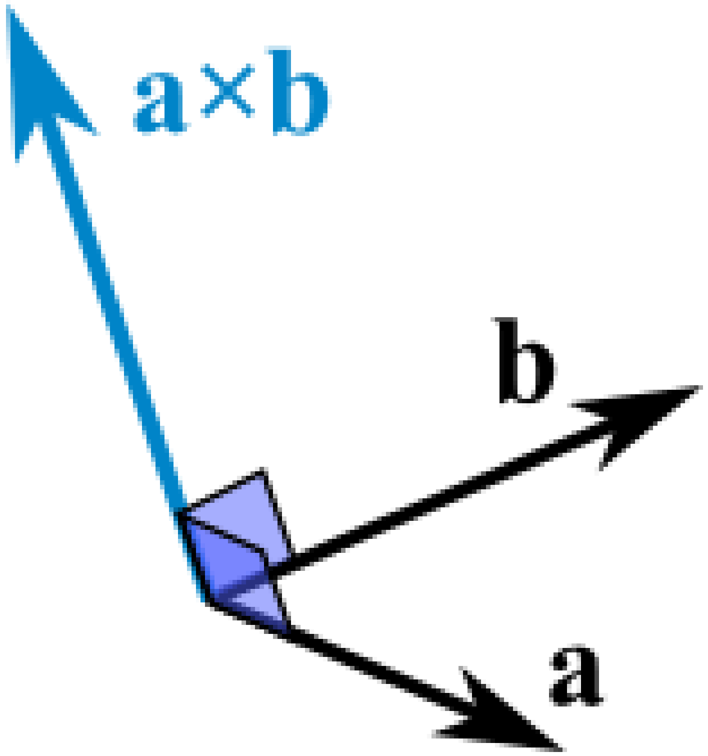
Dot product

- Directional growth of one vector over another
- Result is how much we went along the original path
  - Zero
    - No growth in original direction
    - Parallel to the surface
  - Positive
    - Some growth in original direction
  - Negative
    - Reverse growth



Cross product

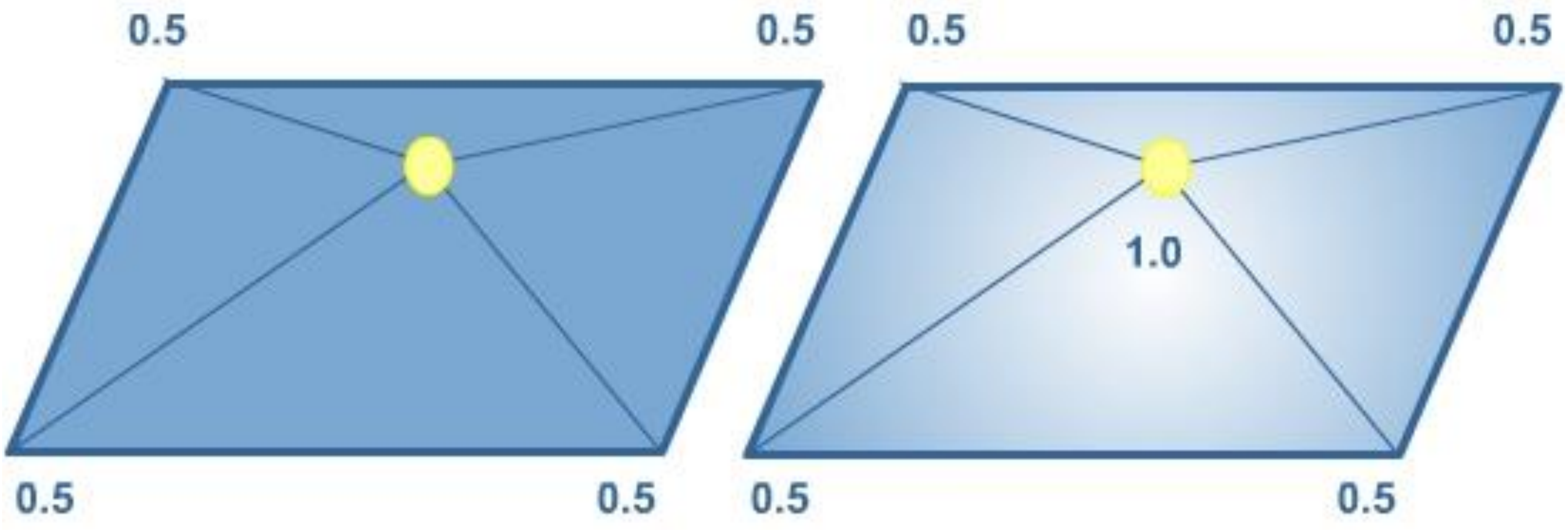
- Two vectors get another vector which is at right angles to both

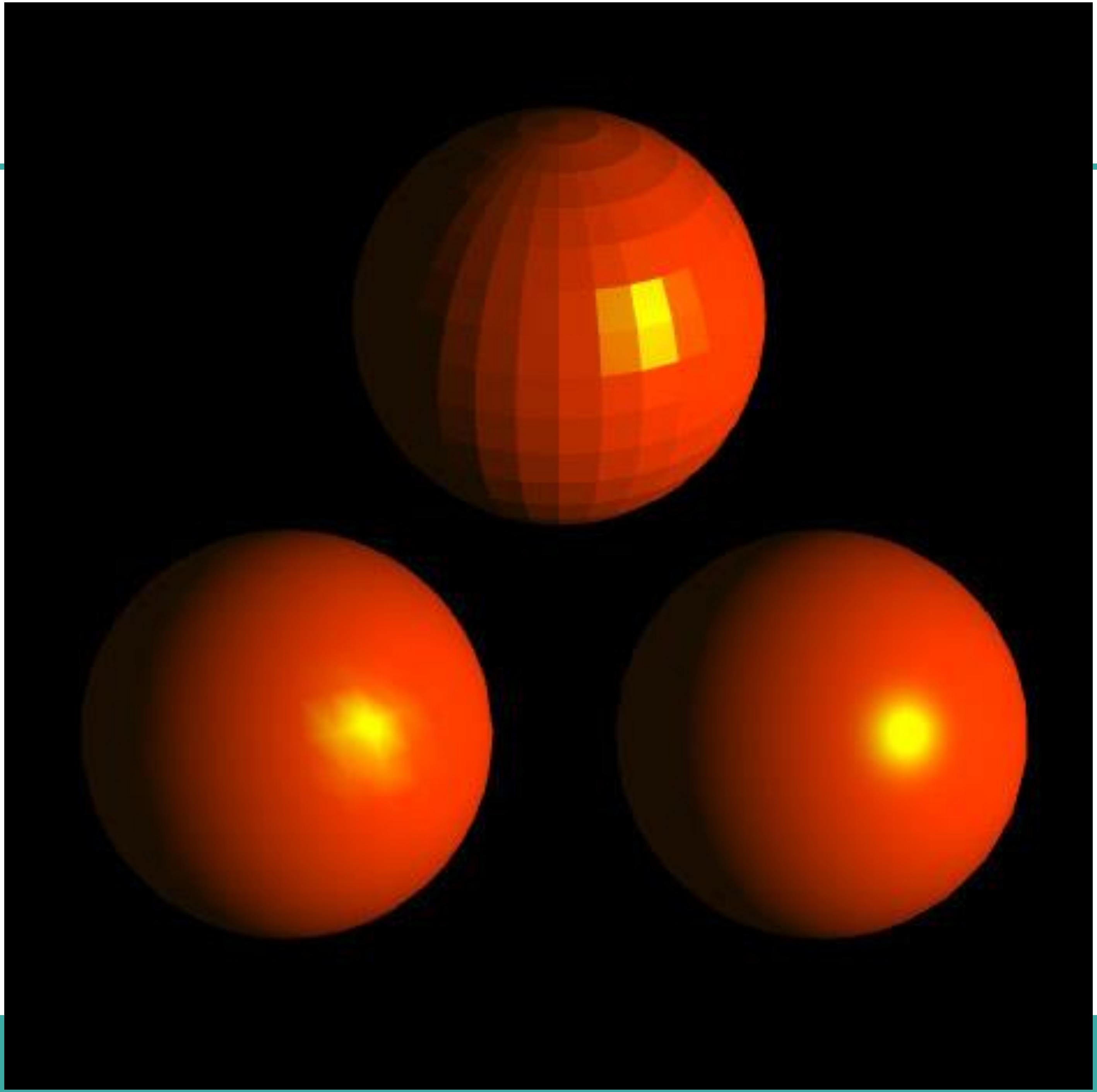


Per pixel lighting

- Last year we did
  - Per face lighting
  - Per vertex lighting
- Per pixel lighting
  - Calculates the illumination for each pixel
  - Can be combined with other techniques
    - Normal/Bump maps
    - Specular maps
    - etc

Per pixel lighting





Examples

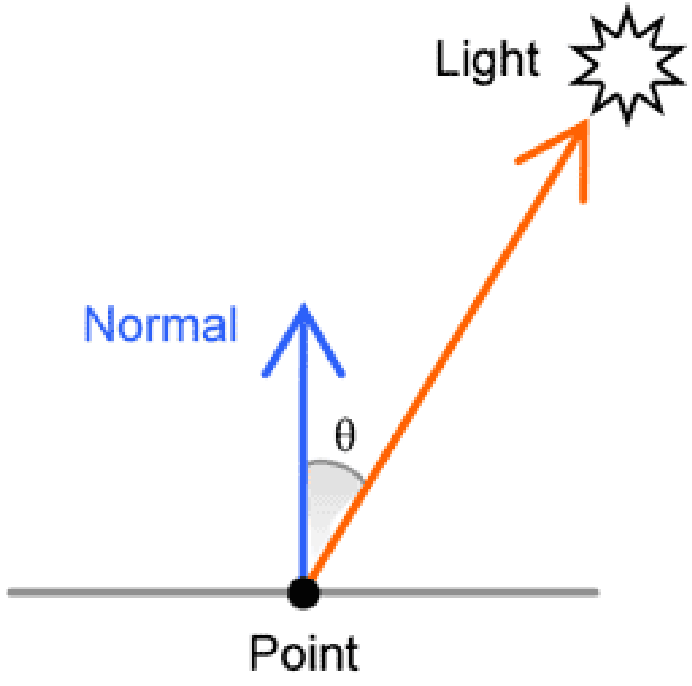
- Single directional light
- Add ambient
- Add specular component
  - Requires additional data sent to shaders
  - We need to know the camera position

Light class

- A class that will represent a light
- Storing and providing access to
  - Ambient colour
  - Diffuse colour
  - Direction
  - Specular colour
  - Specular power
  - Position
- Getters and setters for all of these
- Already in the framework

How - Diffuse

- Create a directional light
  - Lighting that emulates how the Sun illuminates the Earth
  - Light that travels in a single direction
    - Represented by a single vector
  - Lighting anything it touches
  - Light intensity based on angle of face to direction of light
    - Calculated as the dot product of the normal vector and the light direction



How - Diffuse

- Calculate light intensity
  - Dot product of normal and (inverse) light direction
    - Returns a value between -1 and 1
    - Represents our light intensity
  - Clamp this value between 0 and 1
  - Combine light intensity with diffuse colour
    - Diffuse colour * light intensity
    - Clamp this value to between 0 and 1

How - diffuse

- Send additional data to shaders
  - Need to define a struct in C++ to define the data
  - A matching struct must be defined in the shader/HLSL
  - We have to create a buffer to pass the data
  - We have to pass the data
- More messy than it sounds

C++ struct

```
struct LightBufferType
{
    XMFLOAT4 diffuse;
    XMFLOAT3 direction;
    float padding;
};
```

Shader struct

```
cbuffer LightBuffer : register(cb0)
{
    float4 diffuseColour;
    float3 lightDirection;
    float padding;
};
```

Initialise buffer()

```
// Setup the description of the light dynamic constant buffer that is in the pixel shader.
lightBufferDesc.Usage = D3D11_USAGE_DYNAMIC;
lightBufferDesc.ByteWidth = sizeof(LightBufferType);
lightBufferDesc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
lightBufferDesc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
lightBufferDesc.MiscFlags = 0;
lightBufferDesc.StructureByteStride = 0;


// Create the constant buffer pointer so we can access the vertex shader constant buffer
from within this class.
device->CreateBuffer(&lightBufferDesc, NULL, &lightBuffer);
```

Set shader parameters()

```
// Send light data to pixel shader
LightBufferType* lightPtr;

// Lock the light constant buffer so it can be written to.
deviceContext->Map(m_lightBuffer, 0, D3D11_MAP_WRITE_DISCARD, 0, &mappedResource);
// Get a pointer to the data in the constant buffer
lightPtr = (LightBufferType*)mappedResource.pData;

// Copy light data into the buffer
lightPtr->diffuse = light->getDiffuseColour();
lightPtr->direction = light->getDirection();
lightPtr->padding = 0.0f;

// Unlock the buffer
deviceContext->Unmap(m_lightBuffer, 0);
// Set buffer number (register value)
bufferNumber = 0;
// Set the constant buffer in the pixel shader
deviceContext->PSSetConstantBuffers(bufferNumber, 1, &lightBuffer);
```

Something to keep in mind

- Constant buffers work in chunks of 16 bytes
- Or 4 floats
- If not CreateBuffer() will fail

- If the padding is poorly placed the shader will interpret the data incorrectly

How – diffuse example

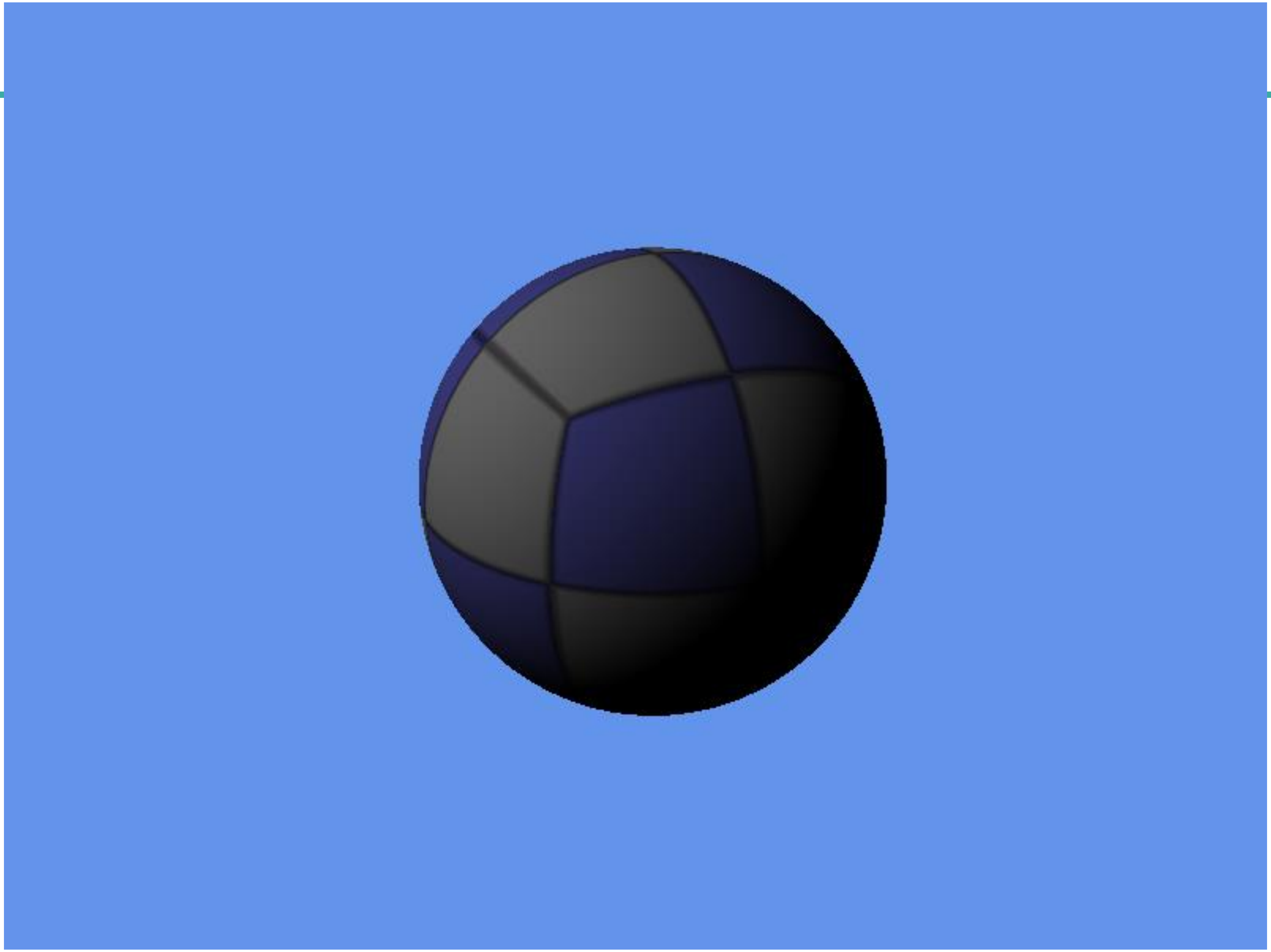
- Diffuse directional lighting
- Vertex shader
  - Same as last week
  - Transform vertex by matrices
  - Transform normal
  - Pass on texture coordinates
- Pixel shader
  - Receives lighting data (in addition to textures from last week)
  - Does lighting calculation

Pixel shader

```
Texture2D shaderTexture : register(t0);
SamplerState SampleType : register(s0);

cbuffer LightBuffer : register(cb0)
{
    float4 diffuseColour;
    float3 lightDirection;
    float padding;
};

struct InputType
{
    float4 position : SV_POSITION;
    float2 tex : TEXCOORD0;
    float3 normal : NORMAL;
};
```

Pixel shader

```
float4 main(InputType input) : SV_TARGET
{
    float4 textureColour;
    float3 lightDir;
    float lightIntensity;
    float4 colour;

    // Sample the pixel colour from the texture
    textureColour = shaderTexture.Sample(SampleType, input.tex);
    // Invert the light direction for calculations.
    lightDir = -lightDirection;
    // Calculate the amount of light on this pixel.
    lightIntensity = saturate(dot(input.normal, lightDir));
    // Determine the final amount of diffuse colour based on the diffuse colour and light intensity.
    colour = saturate(diffuseColour * lightIntensity);
    // Multiply the texture pixel and the final diffuse colour to get the final pixel colour result.
    colour = colour * textureColour;

    return colour;
}
```

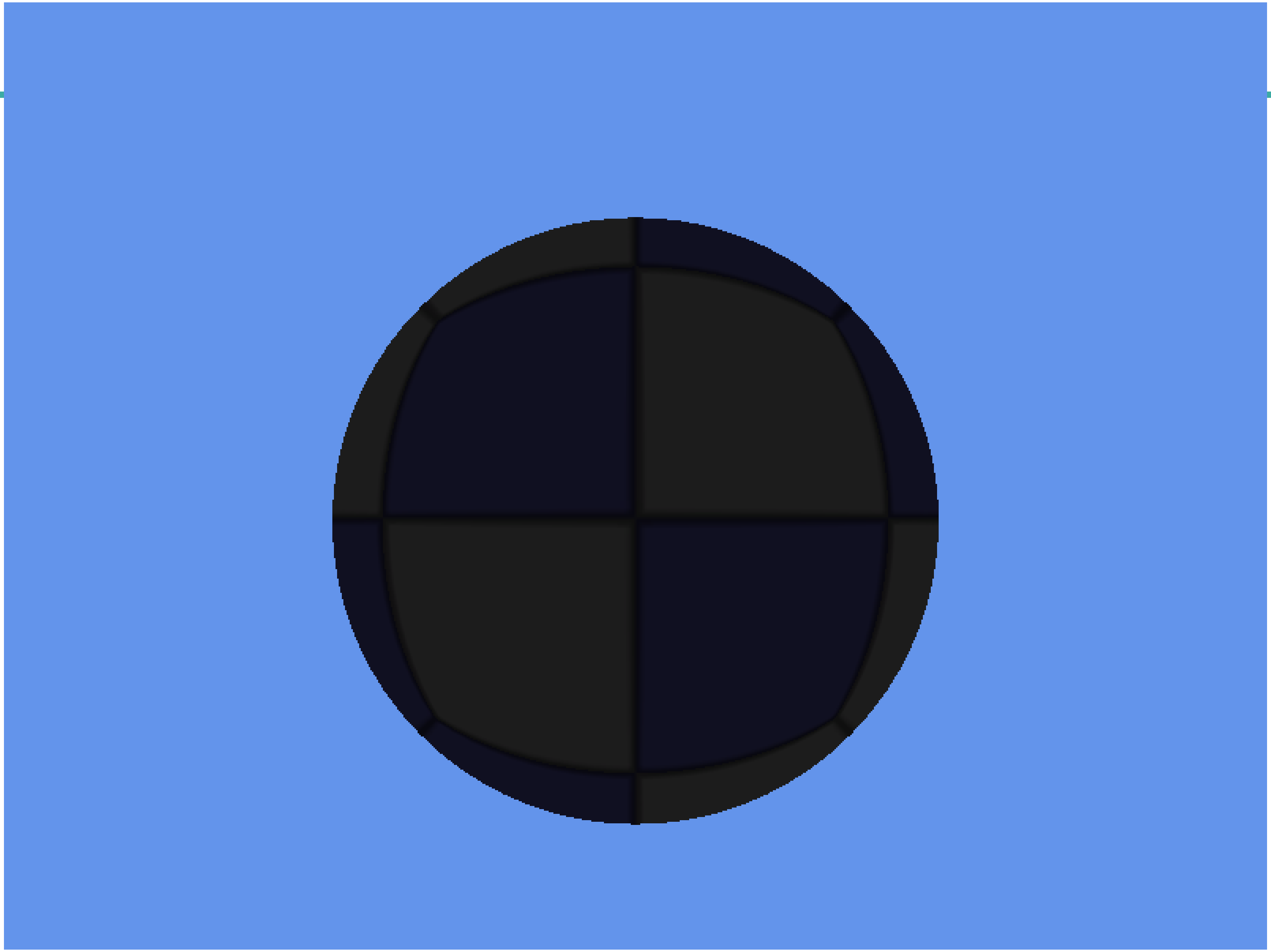


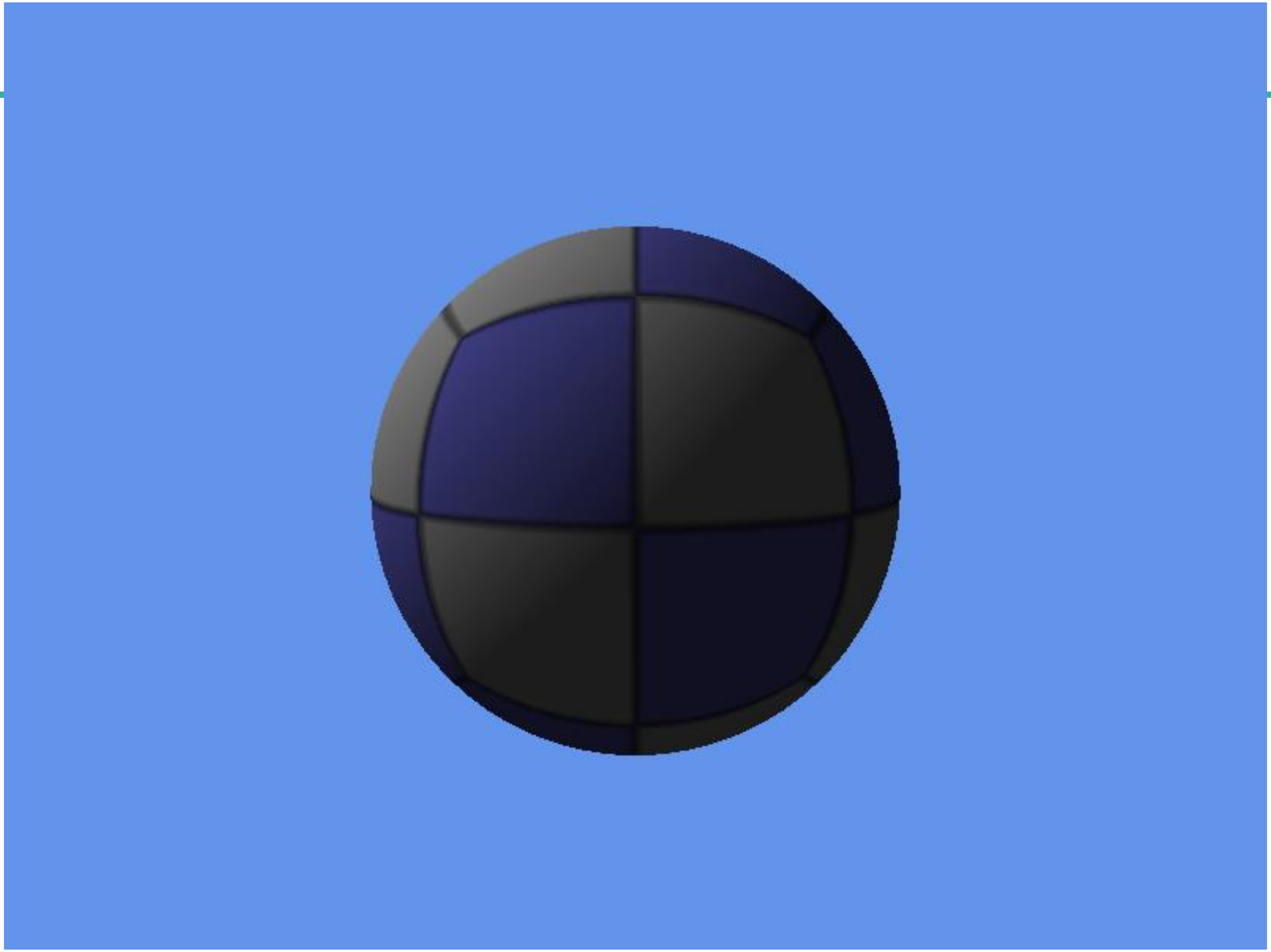
How - Ambient lighting

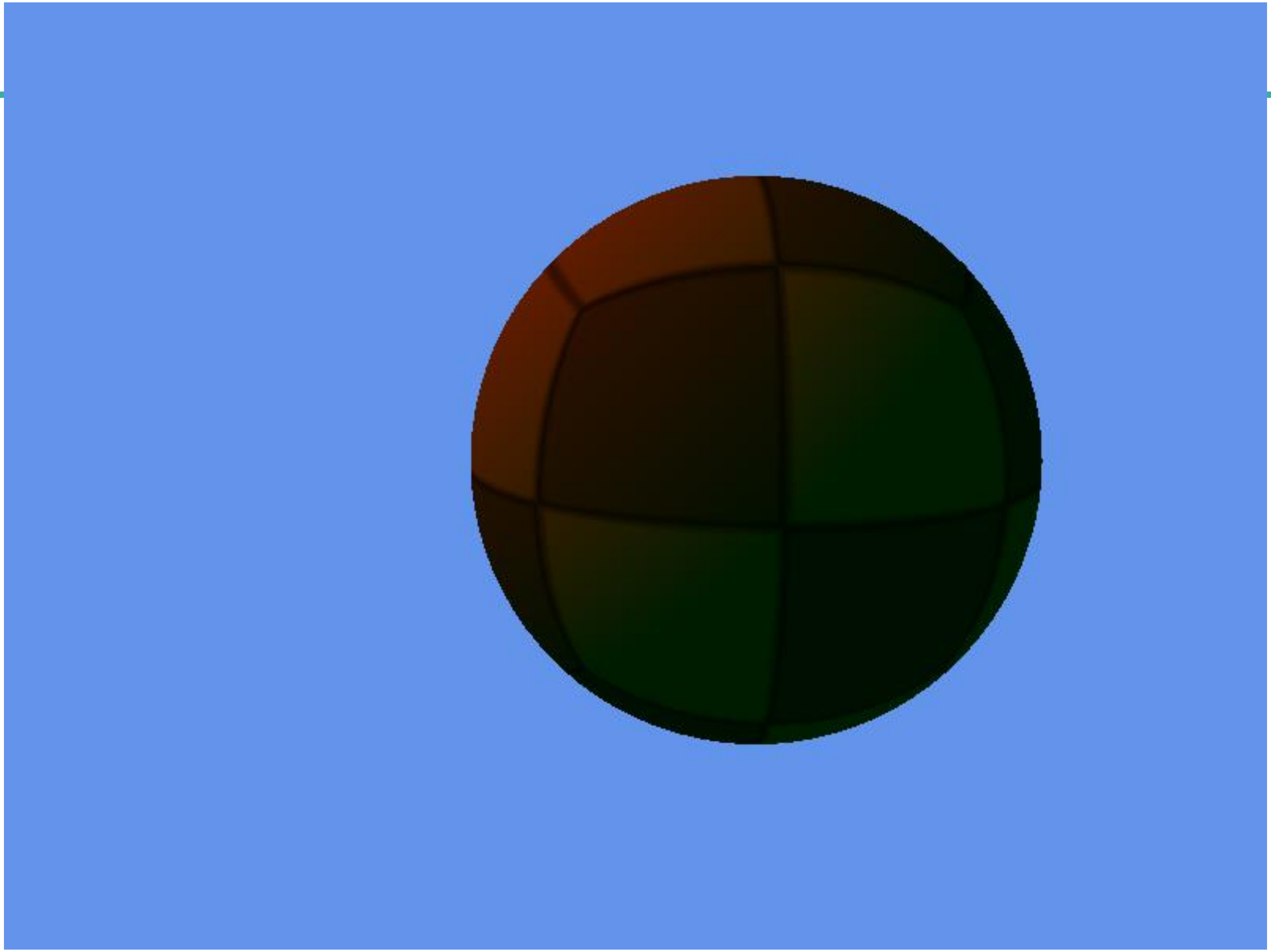
- Building on our diffuse example
- Pass additional lighting information
  - Float4 ambientColour;
- Set a default amount of colour
  - Colour = ambientColour;
- Calculate diffuse light intensity
- Check light intensity value

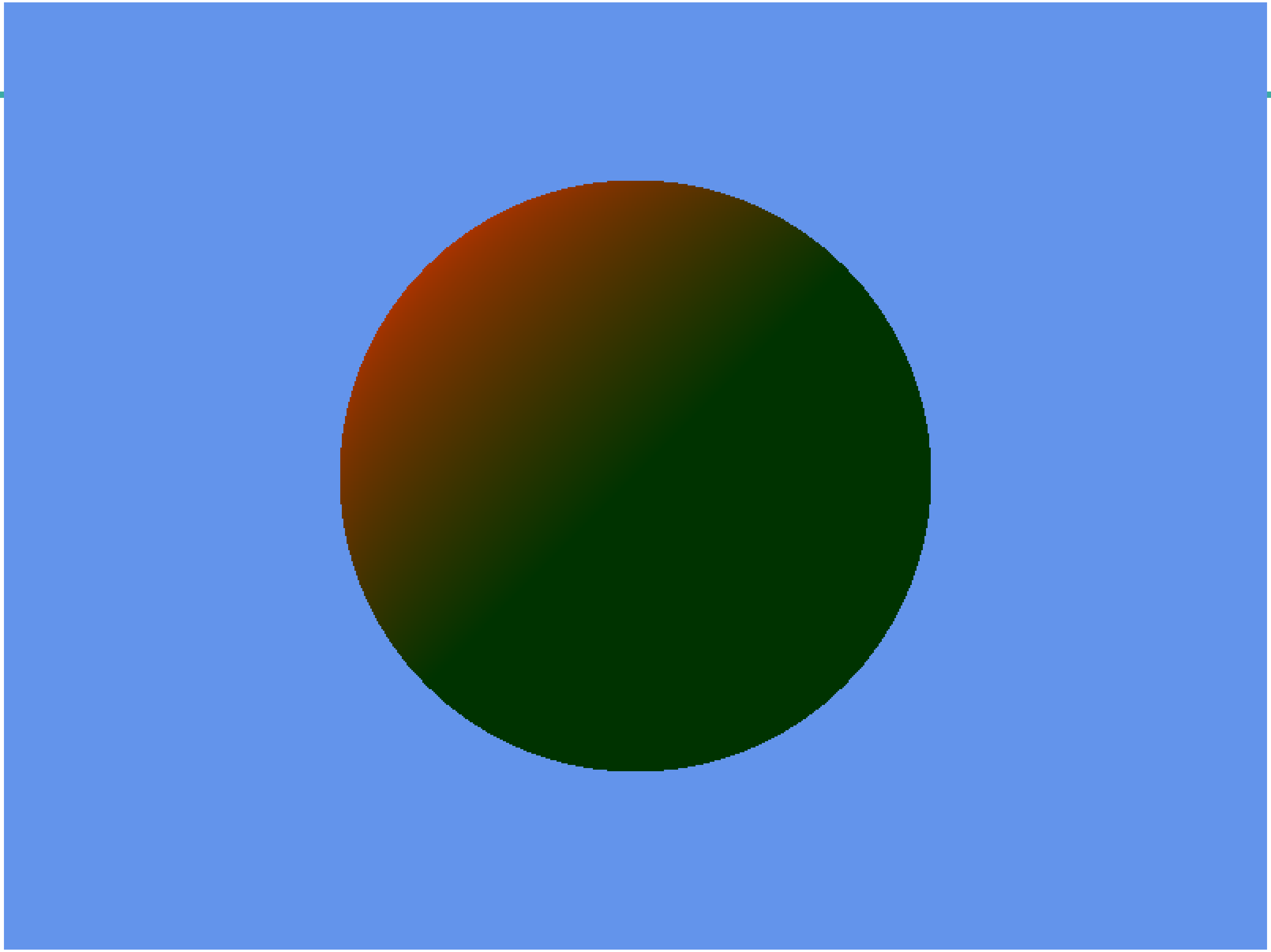
How – Ambient lighting

- If light intensity is greater than zero
  - Combine diffuse colour and light intensity as before
  - diffuse colour * light intensity
- Light intensity could be negative and we don’t want to “add” that to the ambient colour
- Add diffuse component to ambient colour
- Clamp the final colour
- Combine with texture colour



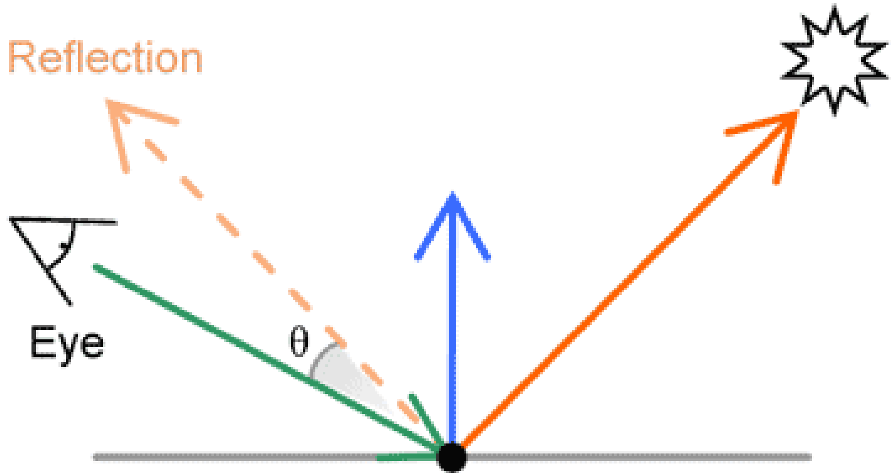






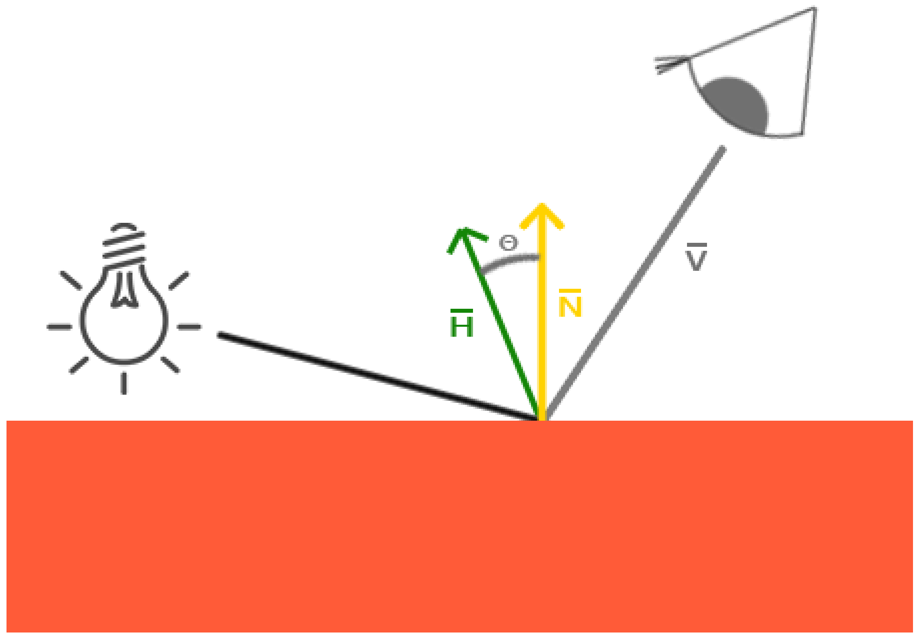
Specular lighting

- Specular light is view dependent
  - The calculation requires the camera position
- Specular equation
  - The specular light value is the specular light colour multiply (viewing direction dot reflection vector) power of specular reflection power
- Works best on a curved surface
- Many methods
  - This is Phong shading

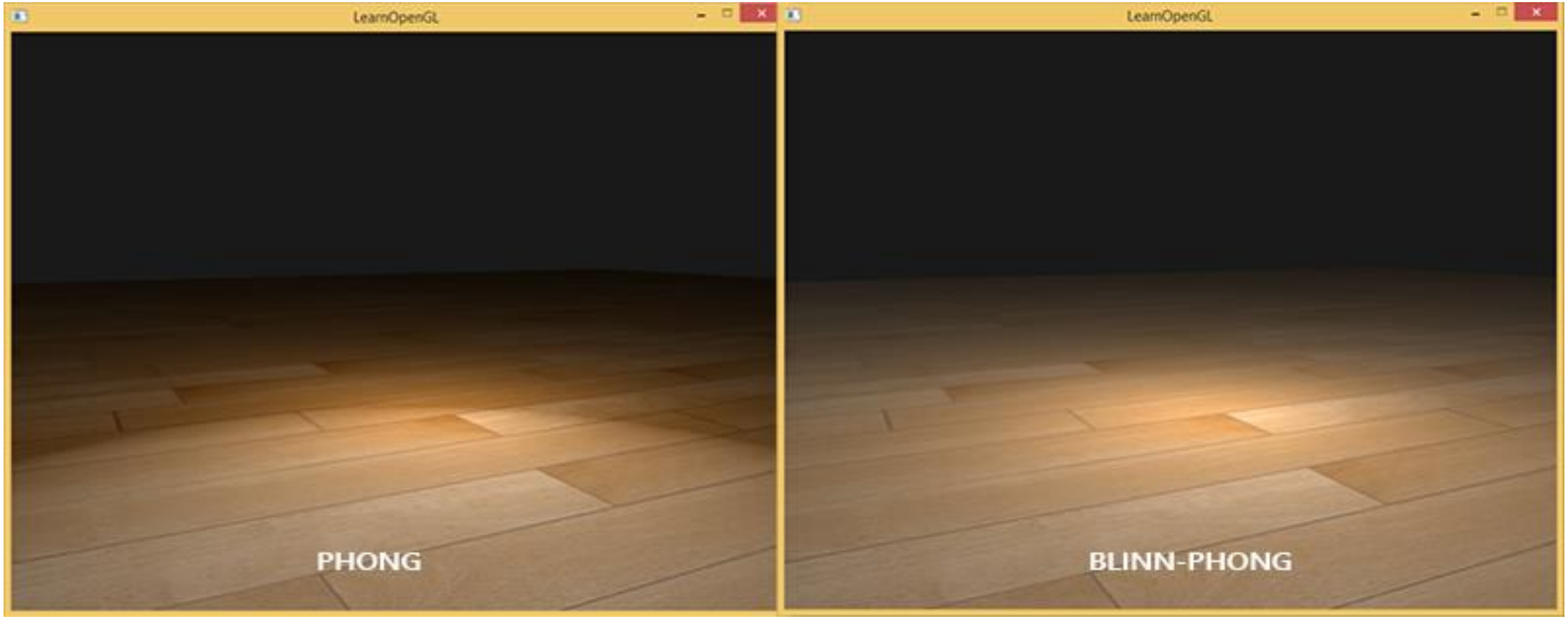


Blinn-Phong

- Phong has “issues” with over 90 degree problems (for low specular component)
- Blinn-Phong uses a half vector to solve the problem
- Halfvector = normalize(view vector + light direction)
- Specular = pow(dot(normal, halfvector), shininess)



Phong vs Blinn-Phong



Specular calculation

- Currently we set the ambient colour
- Calculate the diffuse intensity
  - If greater than zero
  - Calculate the diffuse component
  - Add that to the ambient colour
  - Calculate specular intensity
  - Combine specular intensity with specular colour
- Combine colour with texture colour
- Add specular component to colour

Additions

- Vertex shader
  - Needs to receive the camera position
  - Build a vector from the camera to the mesh
  - Pass that to the pixel shader
- Pixel shader
  - Receives more lighting information
    - Specular colour and power (light class already stores these)
    - Add the specular highlight calculation
    - Add to colour **after** texture

Vertex shader additions

```
cbuffer CameraBuffer : register(cb1)
{
    float3 cameraPosition;
    float padding;
};

struct OutputType
{
    float4 position : SV_POSITION;
    float2 tex : TEXCOORD0;
    float3 normal : NORMAL;
    float3 viewDirection : TEXCOORD1;
};
```

Vertex shader additions

```
// Calculate the position of the vertex in the world.
worldPosition = mul(input.position, worldMatrix);

// Determine the viewing direction based on the position of the camera and the
position of the vertex in the world.
output.viewDirection = cameraPosition.xyz - worldPosition.xyz;

// Normalize the viewing direction vector.
output.viewDirection = normalize(output.viewDirection);
```

Pixel shader

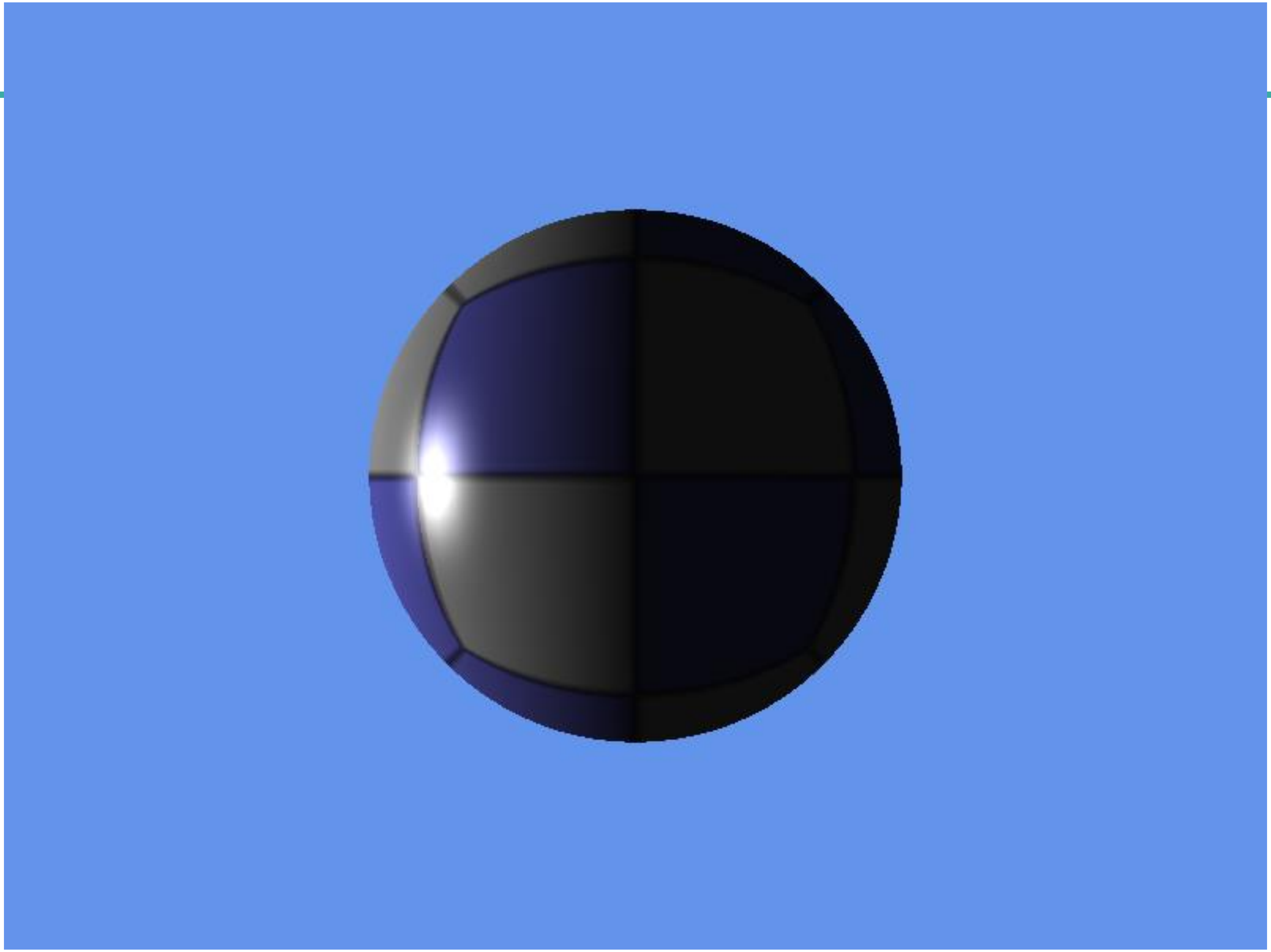
```
Texture2D shaderTexture : register(t0);
SamplerState SampleType : register(s0);

cbuffer LightBuffer : register(cb0)
{
   float4 ambientColour;
   float4 diffuseColour;
   float3 direction;
   float specularPower;
   float4 specularColour;
};

struct InputType
{
    float4 position : SV_POSITION;
    float2 tex : TEXCOORD0;
    float3 normal : NORMAL;
    float3 viewDirection : TEXCOORD1;
};
```

Pixel shader

```
float4 main(InputType input) : SV_TARGET
{
    float4 textureColour;
    float3 lightDir;
    float lightIntensity;
    float4 colour;
    float spec;
    float4 finalSpec;
    float3 halfway;

    // Sample the pixel colour from the texture.
    textureColour = shaderTexture.Sample(SampleType, input.tex);

    // Set the default output colour to the ambient light value for all pixels.
    colour = ambientColour;

    // Calculate the amount of light on this pixel.
    lightIntensity = saturate(dot(input.normal, -direction));
```

Pixel shader

```
if(lightIntensity > 0.0f)
{
    colour += (diffuseColour * lightIntensity);
    colour = saturate(colour);

    // blinn pong
    halfway = normalize(lightDir + input.viewVector);
    spec = pow(max(dot(input.normal, halfway), 0.0), specularPower);
    finalSpecular = saturate(specularColour * spec);
}

colour = colour * textureColour;

// Add the specular component last to the output colour.
colour = saturate(colour + finalSpec);

return colour;
}
```



Blinn-phong point light

In the labs

- I will provide a basic light shader + shaders)
  - We will extend it in the lab
    - Add ambient
  - Make a copy for the specular light
    - This requires more data and therefore a new shader class
- There is a sphere mesh, it is recommended you render this