Direct Lighting

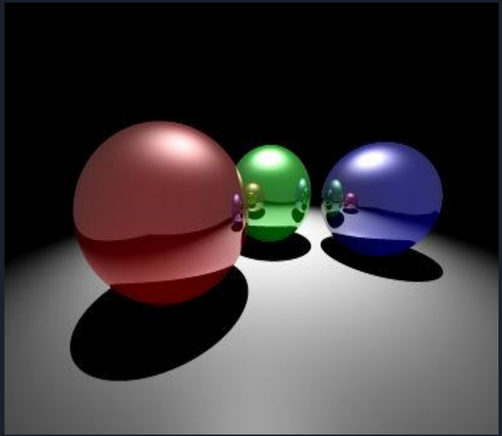
Introduction to 3D lighting

Programming – Computer Graphics

AIE
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Contents

- Lighting in Games vs Real-Life
- Types of Lights
  - Ambient
  - Point
  - Directional
  - Spot
- Normals
- Lighting Models
  - Phong



Lighting in Games

- Lighting plays a pivotal part in modern video games
- Lighting can be used to set the mood of a game, or direct the player down safe paths
- It increases the aesthetic of a scene and makes it more realistic
- Or it can be implemented in a way to make a scene unrealistic!

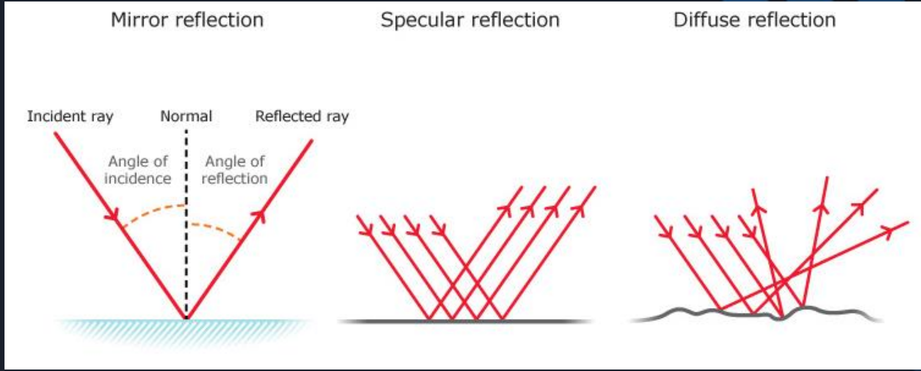
aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Lighting in Real Life

- In nature light is a stream of interacting photons
- A photon has a fixed wavelength and fixed frequency
  - Which determines light colour
- When light hits an object
  - Photons hit electrons in the object
  - Light beam is divided into
    - Photons absorbed - generating energy
    - Photons reflected – generating ray of light
    - Photons refracted - travel through object
  - Light reflections are either Diffuse Reflections or Specular Reflections
- Shadows are the occlusion of light

Light Reflections

- Light rays emit from a source in a direction
  - The light's Incident ray



- Light hits a surface and reflects off it
  - The light's Reflected ray
- The direction the ray reflects is dependent on the surface properties
  - Smooth surfaces cause mirror-like reflections, called Specular Reflections
  - Rough surfaces cause the reflected rays to scatter in many directions, called Diffuse Reflection

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

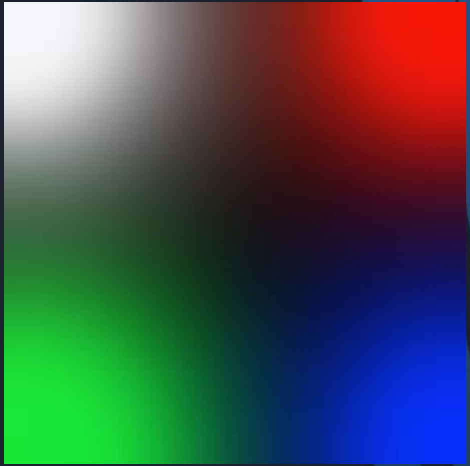
Lighting in Real Life

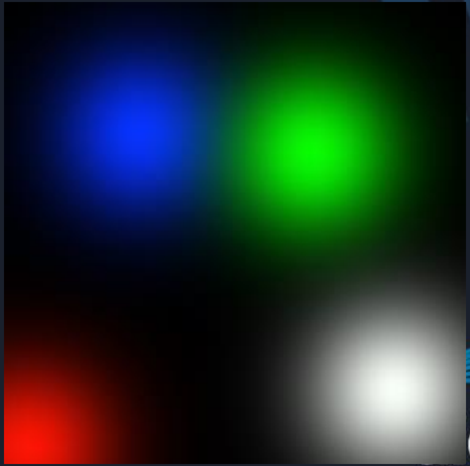
- Impossible to emulate real lighting on a computer in real-time
  - Have to approximate as best as possible
  - Shadows are implemented separate to lighting calculations
- We can also use offline rendering algorithms that are slowly gaining real-time implementations:
  - Ray Tracing
  - Photon Mapping

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

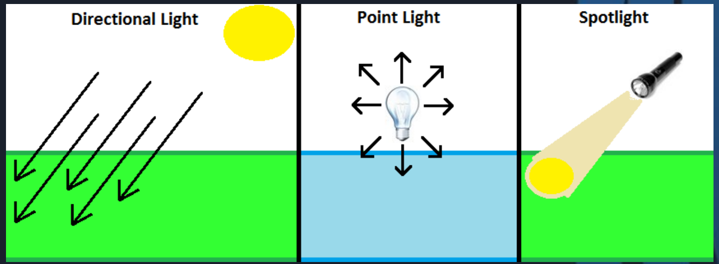
Lighting in Games

- In games we simulate lighting by calculating colours on visible pixels depending on if the light is shining on it
  - The colour can either be calculated per-vertex and interpolated by the Rasteriser across the pixels, or we can calculate it at the per-pixel level
- Per-Vertex Lighting
  - Light calculated in the Vertex Shader
  - Vertices must be illuminated by the light source
    - Light shining on the primitive but not covering any vertices won't be calculated!
- Per-Pixel Lighting
  - Light calculated in the Fragment Shader
  - Each pixel calculates its own lighting rather than receiving interpolated light colour from the Rasteriser
  - Higher resolution lighting





Types of Lights



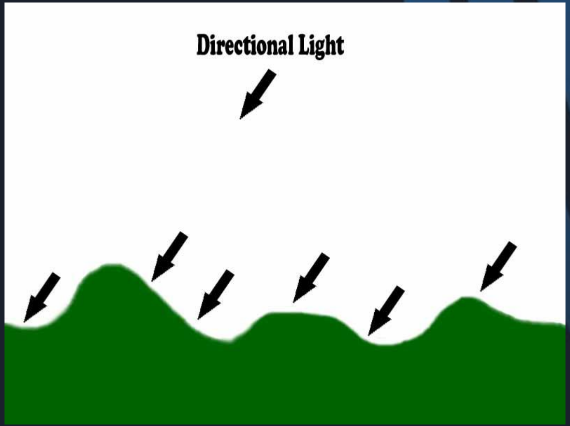
- There are a few different types of lights used in games, the most common being:
- Ambient:
  - Represents the ambient reflected light in a scene when there is no light directly reflecting off a surface
- Directional:
  - Light travels globally in a set direction, with no single originating position
- Point:
  - Light emits from a single position outwards in all directions
  - Usually has a limited range or falloff
- Spot:
  - Light emits from a single position in a limited cone direction

Ambient Light

- Ambient Light is a way of simulating all of the reflected light in a scene that does not directly shine onto surfaces
  - An example could be the ambient sunlight in a house even though the Sun isn't directly shining inside the house
  - Usually the entire scene uses the same ambient light

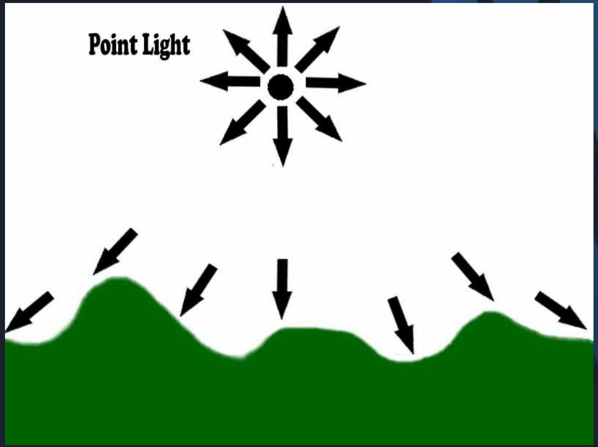
Directional Lights

- A directional light is typically a light that falls globally onto every surface in a set direction
- Most games treat the Sun as a directional light and it is usually the only directional light
- Any surface facing towards the direction the light is coming from can have light reflected from it



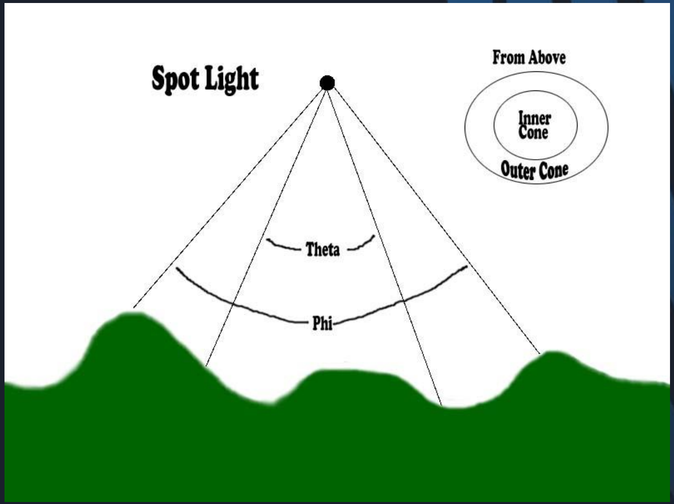
Point Lights

- Point lights are common in games, representing things such as fire, candles, explosions, light bulbs etc
- The direction that the light is traveling is determined by a vector between the surface position and the light position
- Point lights typically have a maximum distance, or falloff, controlled by an attenuation factor that reduces the light intensity based off distance to the surface



Spot Lights

- Spot lights can be used to represent street lamps, flashlights, car headlights etc
- Similar to a point light in that they have a position and a falloff, but also use two angles to determine the spot light's cone of influence
  - Theta and Phi
- Theta represents the inner angle of the light where there is no angle falloff
- Phi represents the outer angle of the spot light's influence
  - The spot light's light falls off between Theta and Phi

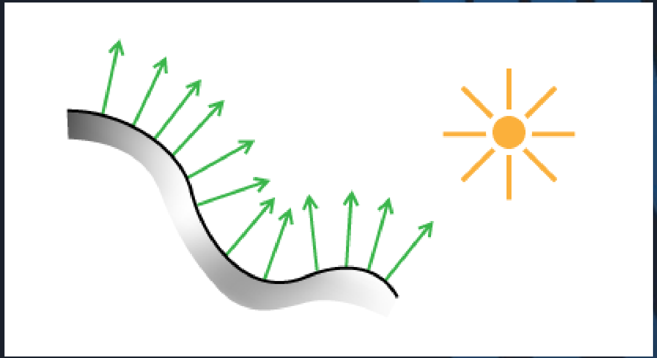


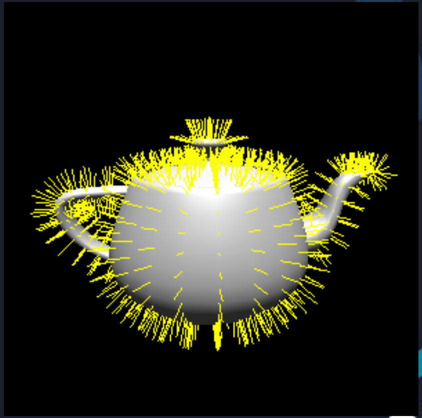
Surface Normals

- To correctly calculate lighting for a surface we need to know the direction the surface is facing so that we can calculate if light is shining on it
  - This direction is called the surface normal
  - All of our vertices in our vertex buffer need to contain additional information about the normal at each point

- This direction is then used to determine if a light is facing towards the surface, and to calculate how the light reflects off it

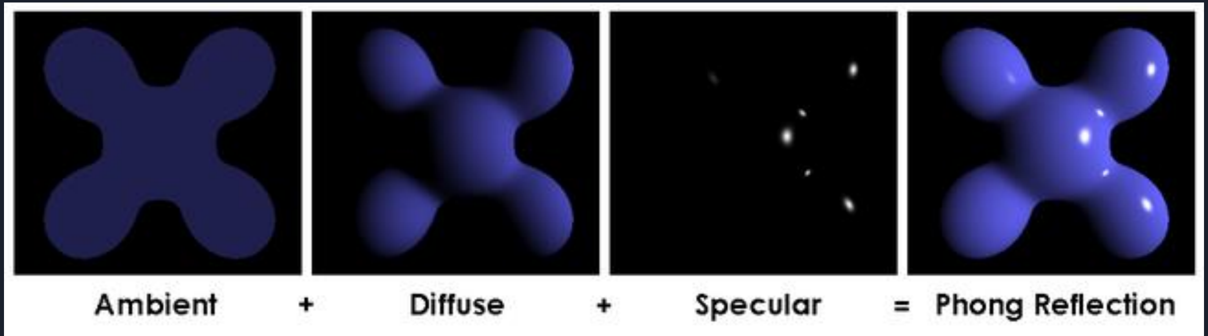
- A normal is typically a normalised unit vector





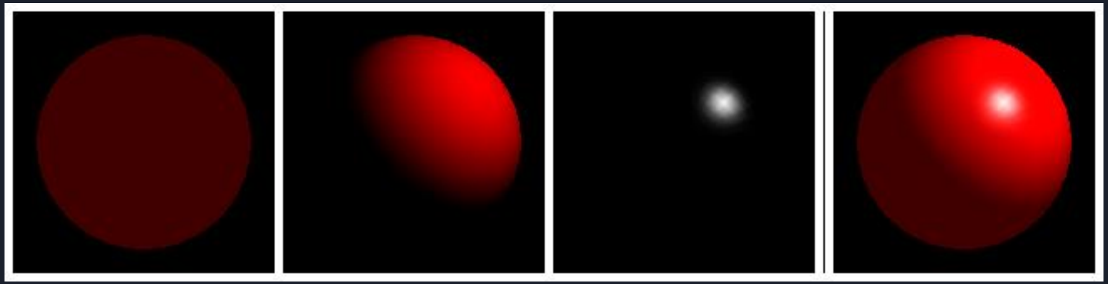
Lighting Models

- Researchers have found many different ways to calculate lighting based on light properties and surfaces
  - Some simplify the equation for speed, not accuracy
  - Others are far more accurate but take a lot of processing
- By far the most common simple lighting model used in computer graphics is Phong Lighting, due to its efficiency and simplicity
  - Phong lighting takes the 3 light properties, Ambient, Diffuse and Specular, and combines them with matching material properties to create a final colour



Ambient, Diffuse and Specular Properties

- There are 3 common colour properties of lighting, all of which have Red, Green and Blue (RGB):
  - Ambient : colour of the indirect light in an area
  - Diffuse : colour of the reflected light reflected in such a way that the light is reflected at many angles
  - Specular : colour of the reflected light reflected as a single ray off the objects surface
- Typically both lights and surface materials have these properties
  - A material may have a blue diffuse colour but when light shines at a certain angle it could have a red specular colour
  - Similarly light may be yellow for diffuse, but could have an odd green specular colour



aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Phong Lighting

- The mathematical model for Phong Lighting is:

$$I_p = k_a i_a + \sum_{m \in lights} (k_d(L_m \cdot N) i_d + k_s(R_m \cdot V)^a i_s)$$

  - That looks complex! But in practice it isn't…

  - ***k*** refers to the surface's material property colours (***a***mbient, ***d***iffuse, ***s***pecular)
  - ***i*** refers to the light properties (***a***mbient, ***d***iffuse, ***s***pecular)
  - ***N*** is the surface normal vector
  - ***Lm*** is the light direction, the Incident ray
  - ***Rm*** is the light's Reflected ray
  - ***V*** is a view direction that represents a ray from the surface to the camera
  - ***a*** is a specular power used to control the sharpness of specular reflection

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

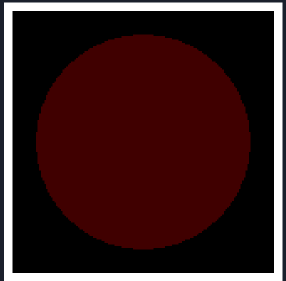
Phong Ambient and Diffuse Equation

- First we will discuss how to implement the Ambient and Diffuse portions of the equation:

$$I_p = \underbrace{k_a i_a}_{\text{Ambient}} + \sum_{m \in lights} (\underbrace{k_d(L_m \cdot N)i_d}_{\text{Diffuse}} + \underbrace{k_s(R_m \cdot V)^a i_s}_{\text{Specular}})$$

- The Ambient portion is simple
  - We simply multiply the surface's ambient colour against the environment's ambient light colour
  - In many cases the surface's ambient colour is the same as it's diffuse colour
  - Note that the light ambient isn't per-light!

$$\text{Ambient} = k_a i_a$$



Example grey ambient light with red ball

Phong Diffuse Equations

- Diffuse is slightly more complex

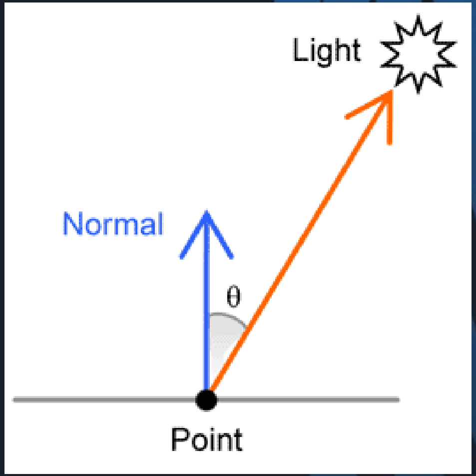
$$I_p = k_a i_a + \sum_{m \in lights} (\underbrace{k_d (L_m \cdot N) i_d}_{\text{Diffuse}} + k_s (R_m \cdot V)^a i_s)$$

- We calculate it for every light and break the equation into 2 parts
  - Calculate the diffuse colour by combining the light and material diffuse properties
  - Calculate Lambert's Cosine Law to determine how much light is actually reflecting off the surface in to the viewer, i.e. the camera
- Calculating the diffuse colour is as easy as the Ambient
  - Multiply the surface's diffuse colour with the light's diffuse colour

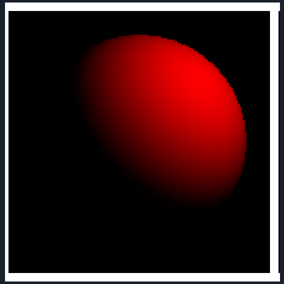
$$\text{Diffuse} = k_d i_d$$

Lambert's Cosine Law

- Lambert's Cosine Law isn't as scary as it sounds!
- To calculate the cosine (commonly called the Diffuse Term or Lambertian Term) you just:
  - Perform a dot product between the surface's normal vector and a vector in the direction the light is coming from
    - This is the inverse of the light's Incident ray
  - Both vectors must be unit length
  - The value is then clamped between 0 and 1
    - This value then represents the percentage of reflected diffuse light from a surface
- The final diffuse part of the equation is just this diffuse term multiplied against the combined diffuse colour
  - This effect shades the surface, lighting the part facing the light



$$\text{DiffuseTerm} = L_m \cdot N$$



Ambient and Diffuse Shader Form

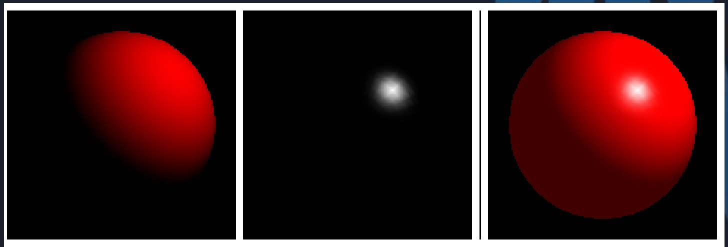
- Implementing the equation for a single light in a shader, such as GLSL, is as follows:

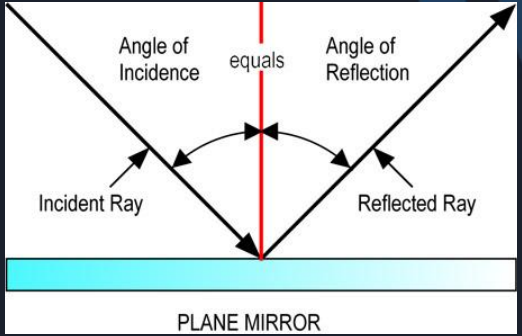
```
uniform vec3 kA = vec3(1,0,0);   // red ambient material colour
uniform vec3 kD = vec3(1,0,0);   // red diffuse material colour

// grey environment ambient light and white diffuse light
uniform vec3 iA = vec3(0.25f,0.25f,0.25f);
uniform vec3 iD = vec3(1,1,1);

in vec3 N;         // normalised surface normal from mesh
uniform vec3 L;  // normalised light direction from light

void main()
{
    vec3 Ambient = kA * iA;   // ambient light

    // Lambert term, with L reversed to face towards the light
    float NdL = max( 0.0f, dot( N, -L ) );

    vec3 Diffuse = kD * iD * NdL;   // diffuse light for one light

    gl_FragColor = vec4(Ambient + Diffuse, 1);
}
```

$$I_p = \underbrace{k_a i_a}_{\text{Ambient}} + \sum_{m \in lights} (\underbrace{k_d(L_m \cdot N) i_d}_{\text{Diffuse}} + k_s(R_m \cdot V)^a i_s)$$

Specular Highlights

- Specular highlights are the bright spots of light that appear on shiny objects when the light reflects right into your eye
  - Think of all those times as a kid you tried to reflect light into your teacher's eyes!
- This light is confined by the law of reflection, in that the Reflected light ray makes the same angle with the surface as the incoming light ray, the Incident
  - If the reflected ray enters the viewer's sight then the specular highlight is added to the final calculated colour at that point





Specular Lighting

- Specular lighting is calculated for each light just as diffuse was

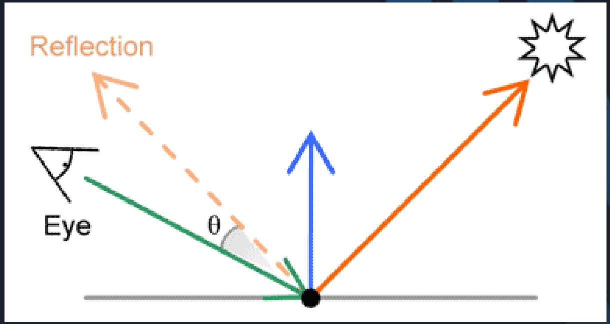
$$I_p = k_a i_a + \sum_{m \in lights} (\underbrace{k_d (L_m \cdot N) i_d}_{\text{Diffuse}} + \underbrace{k_s (R_m \cdot V)^a i_s}_{\text{Specular}})$$

- However it has a few extra bits in the equation
  - ***Rm*** is the light vector reflected about the surface normal, the Reflected ray
  - ***V*** is a vector from the surface to the viewer / camera position
  - The rays are dot product against each other like the diffuse term, but the result is also raised to a power ***a***
  - $\boldsymbol{k_s}$ and $\boldsymbol{i_s}$ just refer to the surface material's specular colour and the light's specular colour

Specular Lighting

- To calculate the reflected light ray we simply reflect it around the surface normal

- We then perform a dot product between the light's reflected ray and a vector from the surface to the viewer
  - This value is the specular term
  - The specular term is clamped between 0 and 1 much like the diffuse term was



$$SpecularTerm = R_m \cdot V$$

Specular Lighting

- We also raise the specular term to a specular power
  - This helps control the intensity of the reflection

$$Specular = k_s \underbrace{(R_m \cdot V)^a}_{SpecularTerm^a} i_s$$

Complete Phong Equation

- The specular term is then multiplied with the specular colour
  - Defined as the light's specular colour multiplied with the surface's specular colour

$$SpecularColour = k_s i_s$$

- Finally the calculated specular colour is added, along with all other light's specular, to the final pixel colour

- Specular lighting helps define shiny surfaces and can mimic glass, plastic, skin, water, etc
  - Also helps highlight shape and texture of a surface

Full Phong Shader Form

```
#version 410
uniform vec3 kA = vec3(1,0,0);   // red ambient material colour
uniform vec3 kD = vec3(1,0,0);   // red diffuse material colour
uniform vec3 kS = vec3(1,0,0);   // red specular material colour

uniform vec3 iA = vec3(0.25f,0.25f,0.25f);
uniform vec3 iD = vec3(1,1,1);
uniform vec3 iS = vec3(1,1,1);
uniform float iSpecPower = 32.0f;   // specular power

in vec3 N;       // normalised surface normal from mesh
in vec3 P;       // world-space surface position from mesh

uniform vec3 camPos; // world-space camera position
uniform vec3 L;      // normalised light direction from light

void main() {
    vec3 Ambient = kA * iA;   // ambient light

    float NdL = max( 0.0f, dot( N, -L ) );  // Lambert term
    vec3 Diffuse = kD * iD * NdL;           // diffuse light for one light

    vec3 R = reflect( L, N );                // reflected light vector
    vec3 E = normalize( camPos – P );  // surface-to-eye vector

    float specTerm = pow( min( 0.0f, dot( R, E ) ), iSpecPower );  // Specular term
    vec3 Specular = kS * iS * specTerm;   // specular light for one light

    gl_FragColor = vec4(Ambient + Diffuse + Specular, 1);
}
```

Summary

- There are many models that implement lighting
  - Phong Lighting being common within games
- There are four common light types
  - Ambient, Directional, Point and Spot
- There are three common light and material properties
  - Ambient, Diffuse and Specular

Further Reading

- Akenine-Möller, T, Haines, E, 2008, *Real-Time Rendering*, 3rd Edition, A.K. Peters
- Wolff, D, 2013, *OpenGL 4 Shading Language Cookbook*, 2nd Edition, PACKT Publishing
- Lengyel, E, 2011, *Mathematics for 3D Game Programming and Computer Graphics*, 3rd Edition, Cengage Learning