Computer Graphics

COMP3421/9415

2021 Term 3 Lecture 12

What did we learn last lecture?

Introduction to Lighting

- Real world vs Simulation
- The possibilities of accurate simulation
- Phong Lighting due to processing limitations
- Beginning to look closely at the maths for Ambient and Diffuse lighting

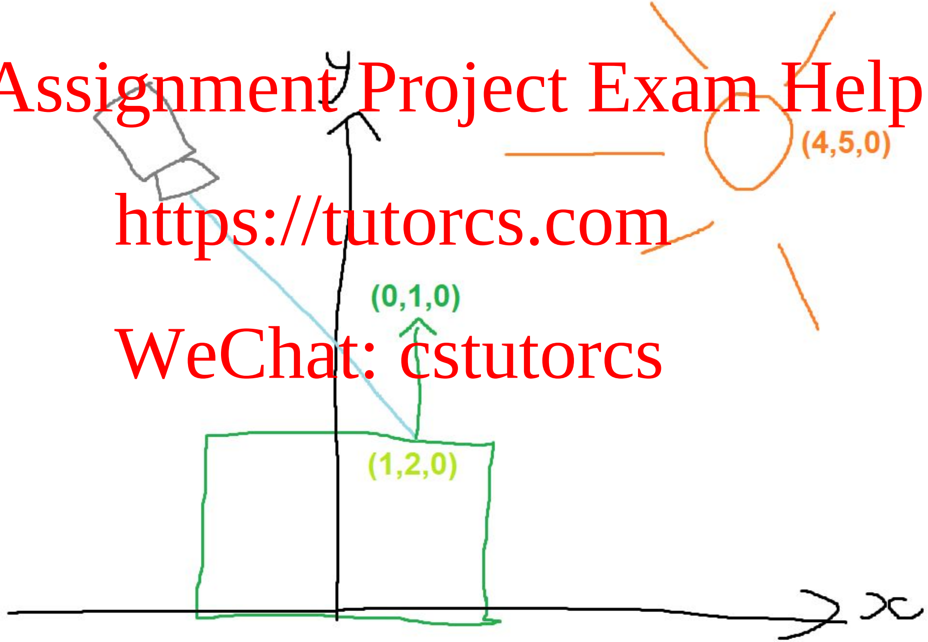
What are we covering today?

Continuing the deep dive into Phong Lighting

- Diffuse Lighting
- Specular Lighting
- Dealing with multiple lights

Diffuse Lighting Walkthrough

A worked example



A worked example

Let's calculate some Ambient and Diffuse Lighting

- A scene with a light at (4,5,0)
  - The light's colours are (0.8, 0.8, 0.5), so a bit yellowish

- Ambient light is (0.1, 0.1, 0.1)
- The current fragment is at (1,2,0)

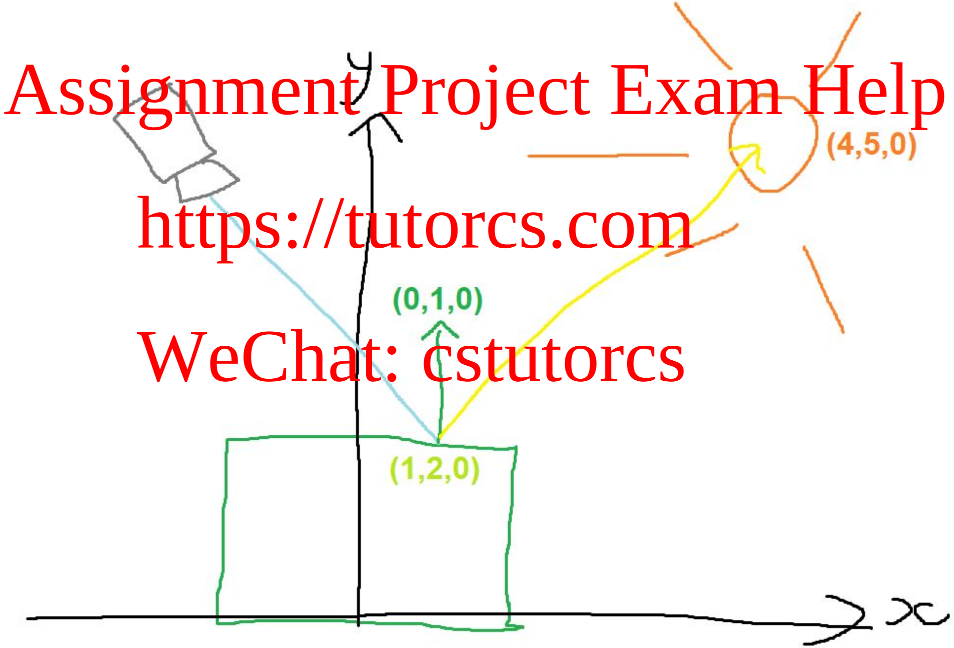
  - The surface normal is (0,1,0)
  - The diffuse reflective colour is (0.1, 0.8, 0.3), mostly green

What Information do we have?

$I_a + I_d = i_a * k_a + k_d * (L.N) * i_d$

- $i_a = (0.1, 0.1, 0.1)$
- $k_a = (0.1, 0.8, 0.3)$ we're using the diffuse colour
- $k_d = (0.1, 0.8, 0.3)$
- L = needs to be calculated
- $N = (0,1,0)$
- $i_d = (0.8, 0.8, 0.5)$

L: Direction to the Light Source



Calculate L

L: Direction Vector aiming at the light

- We have a start and end of the vector
- end - start = vector

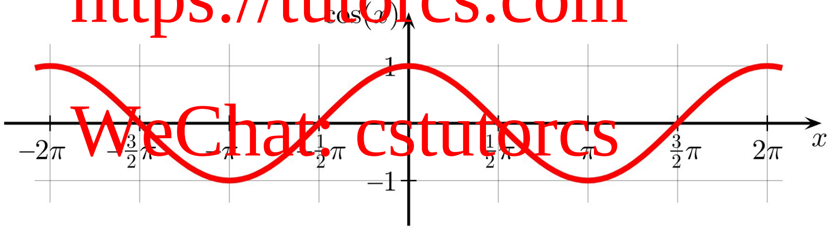
- Remember to Normalise!
- `Normalise((4,5,0) - (1,2,0))`

- `Normalise(3,3,0)` Divide the vector by its own length to normalise it!
- Remember Pythagorean Triangles? $a^2 + b^2 = c^2$

L.N

What does the dot product tell us?

- Dot product of two unit (length 1) vectors
- is the cosine of the angle between them



The complete equation

$I_a + I_d = i_a * k_a + k_d * (L.N) * i_d$

- $I_a = (0.1, 0.1, 0.1) * (0.1, 0.8, 0.3)$
- $I_d = (0.1, 0.8, 0.3) * ((1/\sqrt{2}, 1/\sqrt{2}, 0).(0,1,0)) * (0.8, 0.8, 0.5)$
- $I_a + I_d = ...$
- Try this out yourself!
- Also try changing the light position and see the effect

In the Shaders

Our Graphics card will be doing this maths!

- Vertex Shader Outputs:
    - Fragment Position
    - Surface Normal (either generated from vertices or lerped from normals at vertices)
    - Colour and/or TexCoord
- Fragment Shader Inputs:
    - the outputs above
    - Light Position or Direction
    - Light Colour
- Frag shader will calculate the Light Direction
- Frag shader will complete the algorithm and calculate the frag colour

Specular Lighting

Specular Lighting Equation

$I_s = k_s * (R.V)^a * i_s$

- I_s Final intensity of specular light
- k_s Specular reflectivity of the fragment
- R Direction of reflected light
- V Direction to the viewer
- a Phong Exponent
- i_s Specular intensity of light source

$(R.V)^a$ - What's this part of the Equation?

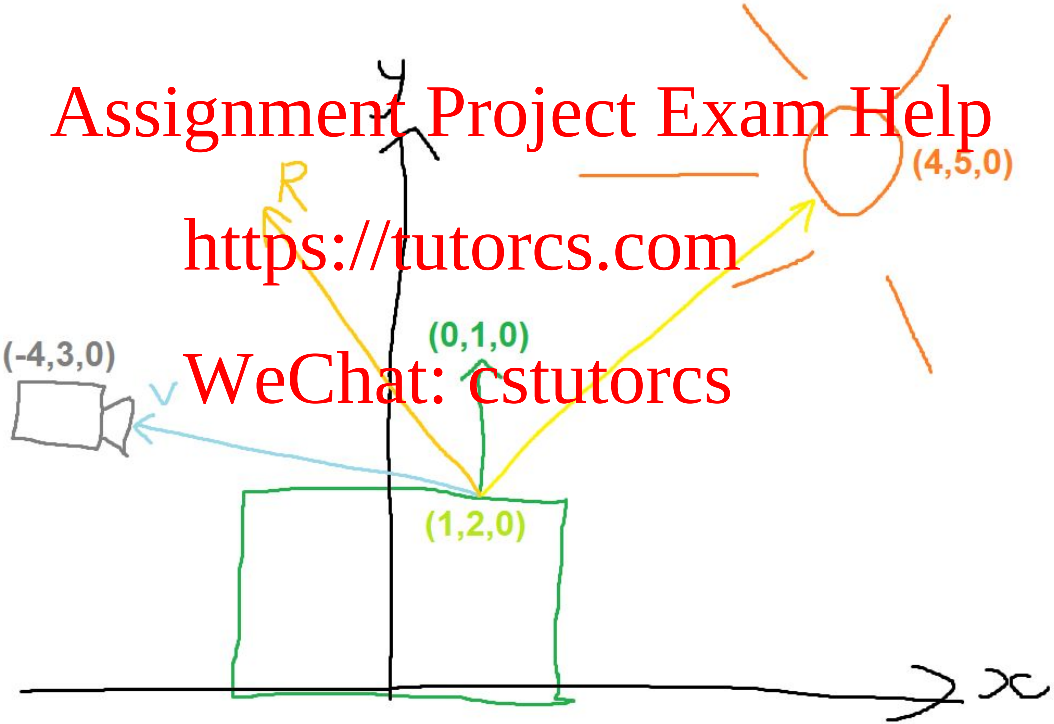
Important directions for reflections

- `R` Direction of reflected light
  - Calculated based on the Light Direction and the Surface Normal

- `V` Direction to the viewer
  - The direction from the Fragment to the Camera
- This dot product is similar to the Diffuse `L.N`

- How close is the reflected light to the camera?
- This means: Is the light reflecting directly into the camera?

Reflected and Viewer Vectors



What is the Phong Exponent?

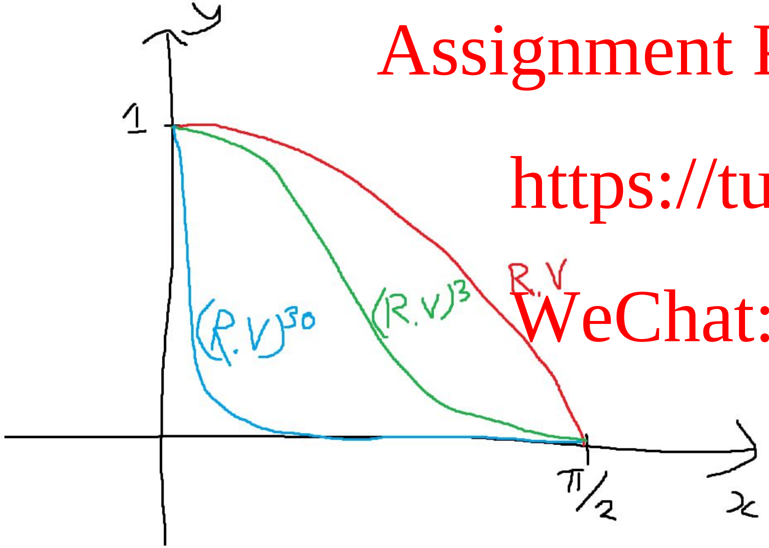
A measure of "shininess"

- An abstract concept, not grounded in reality
- The concept:

  - The shinier something is, the more focused a reflection is
  - Something less shiny still reflects the light, but in a "wider" fashion
- The maths:

  - R.V will be between 0 and 1
  - Any positive power of 0-1 will "narrow" its curve

The Phong Exponent in action



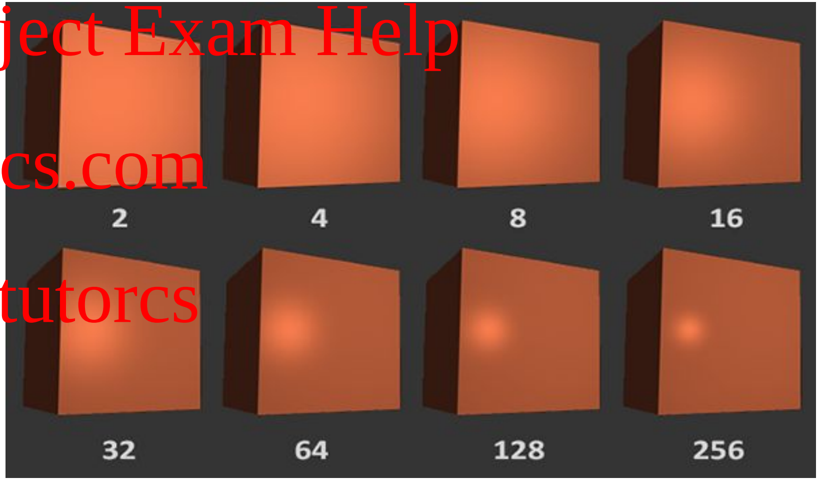
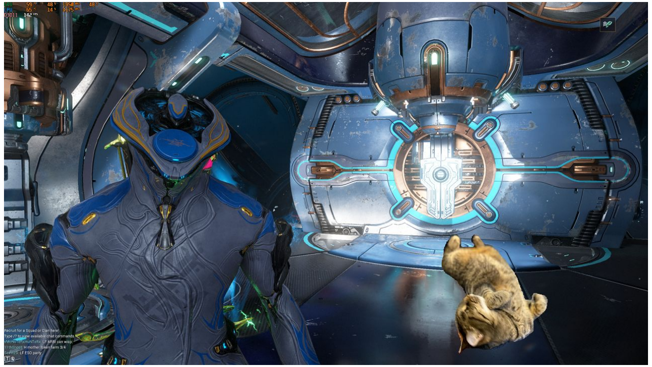


Image credit: learnopengl.com

Break Time

Phong's specular lighting

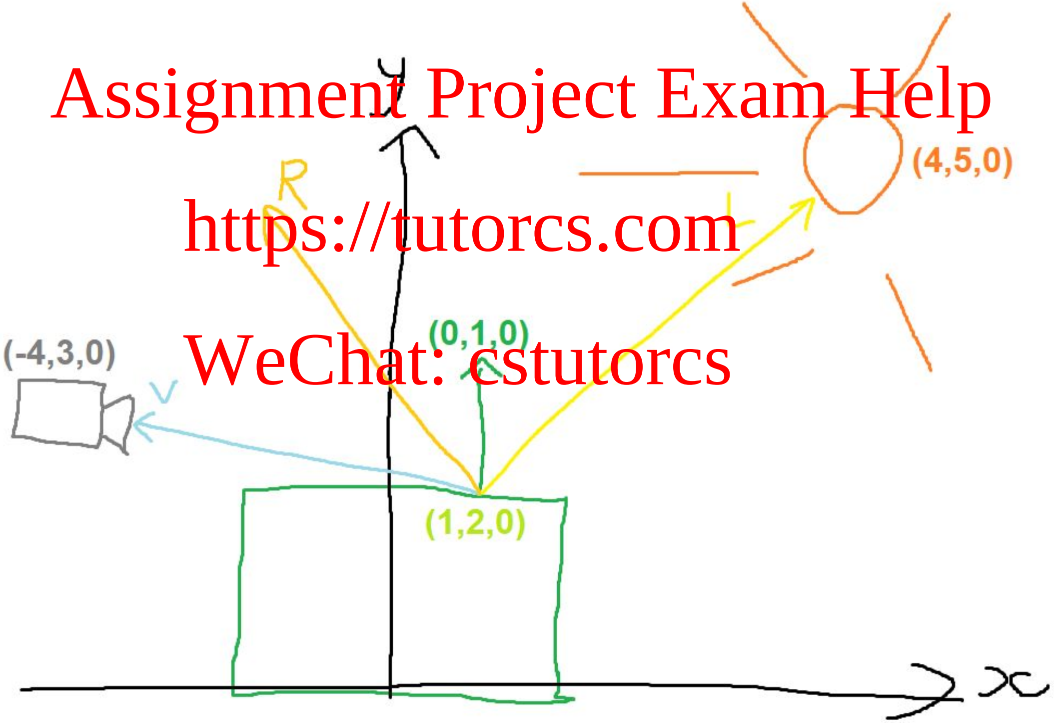
- The "bright spot" is like a reflection of the light source
- But it's all a trick!
- Our eyes are used to not being able to see a bright light reflected clearly
- So we just get a "splodge" of the light's colour
- Like being dazzled by brightness
- Specular highlights make us believe in metal, liquids etc



Warframe,
Image credit: Digital Extremes

Specular Lighting Walkthrough

Another worked example



Another worked example

Let's calculate some Specular Lighting

- A scene with a light at (4,5,0)
  - The light's colours are (0.8, 0.8, 0.5), so a bit yellowish

- Ambient light is (0.1, 0.1, 0.1)
- The current fragment is at (1,2,0)

  - The surface normal is (0,1,0)
  - The specular reflective colour is (1.0, 1.0, 1.0), it's pure reflective
- The camera is at (-4,3,0)
- The Phong Exponent is 20 (this can be experimented with!)

The Equation

$I_s = k_s * (R.V)^a * i_s$

- k_s = (1.0, 1.0, 1.0)
- L = Normalise((4, 5, 0) - (1, 2, 0))
- N = (0, 1, 0)
- R = needs to be calculated from L and N
- V = Normalise((-4, 3, 0) - (1, 2, 0))
- a = 20
- i_s = (0.8, 0.8, 0.5)

The Reflected Vector

Deciding the direction of a reflection

- We have a vector to the light `L`
- and a surface normal `N`

- A formula for reflection: `R = 2 * (N.L) * N - L`
- The maths behind this formula is interesting if you want to look it up

- `R` will be a direction vector that is an exact reflection of `L`
- `R = 2 * 1/√2 * (0,1,0) - (1/√2, 1/√2, 0)`
- `R = (0, 2/√2, 0) - (1/√2, 1/√2, 0)`
- `R = (-1/√2, 1/√2, 0)`

The complete equation

$I_s = k_s * (R.V)^a * i_s$

- $I_s = (1.0, 1.0, 1.0) *$

  $((-1/\sqrt{2}, 1/\sqrt{2}, 0).(-5/\sqrt{26}, 1/\sqrt{26}, 0))^{20} * (0.8, 0.8, 0.5)$

- $I_s = ...$
- Try moving the camera upwards to see the intensity change

In the Shaders

Again, our Graphics card will be doing the maths!

- Shaders will have similar inputs/outputs to earlier with Diffuse
- New Fragment Shader Input:
  - Camera/Viewer Position

- Frag shader will calculate the Reflected Direction (GLSL `reflect()`)
- Frag shader will complete the algorithm and calculate the frag colour
  - Adds together Ambient, Diffuse and Specular

Multiple Different Lights

Multiple Lights and their different types

Different Types of Lights

- Directional Lights
- Point Lights
- Spot Lights

Handling Multiple Lights in Phong Lighting

- Looping through multiple lights

Directional Lights

Lights so far away, they don't have a position

- Represent distant lights like the Sun
- Represented by a direction vector
- Mathematically easy!
- We no longer calculate the vector to the light, we just use the light's vector

Point Lights

We've been using these in our equations already!

- Lights with a location in the scene
- Represent smaller lights like lamps etc

- Use attenuation to make smaller lights more realistic
- Attenuation is the lowering of intensity based on distance

Spot Lights

Modified Point Lights

- Represent objects like electric torches, vehicle headlights etc
- Adds an aim direction and a cutoff angle
- Some extra calculation needed to see if a fragment is inside the cutoff angle

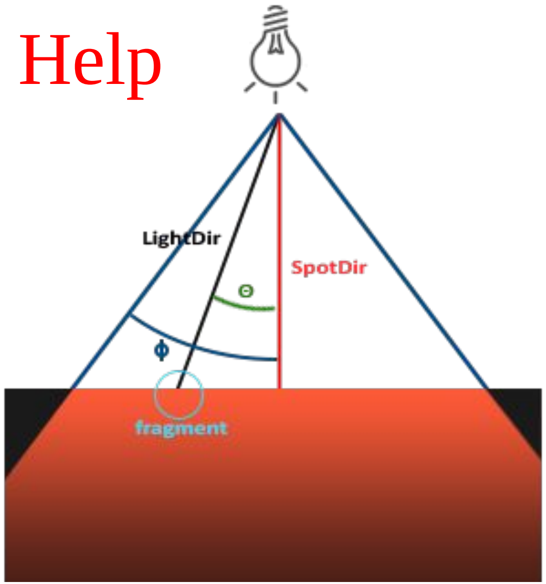


Image credit: learnopengl.com

Multiple Lights

How do we process multiple lights?

- The fragment shader will only run once per fragment
- Each fragment will have one colour, regardless of the number of lights!
- Different code for different lights!
- Calculate Ambient Light
- Loop through all Directional Light(s) diffuse and specular
- Loop through all Point Lights diffuse and specular
- Loop through all Spot Lights diffuse and specular

Phong Lighting

A discussion of Phong Lighting

Pros

- Computes fast
- A good approximation of real light
- Gives us directional lighting
- Gives a simple model for different materials
  - Just alter ambient/diffuse/specular reflectivity
- Can handle a few different light types
- Easy to modify to add capability

A discussion of Phong Lighting

Cons

- Doesn't always look realistic (you can tell it's not real light)
- Specular highlights can get a bit beyond real

- A few genuine issues with the maths
  - Colours can overflow their RGB values
  - Reflected vectors are limited to 0-90 degrees

- Scales by fragments * lights
  - The more lights and objects in a scene multiplies the amount of work for the frag shader

What did we learn today?

Completion of Phong Lighting

- Diffuse walkthrough
- Specular Lighting and walkthrough
- Working with different light types
- and multiple lights
- Some discussion of the pros and cons of Phong Lighting
- We're going to work on some of these issues next week!