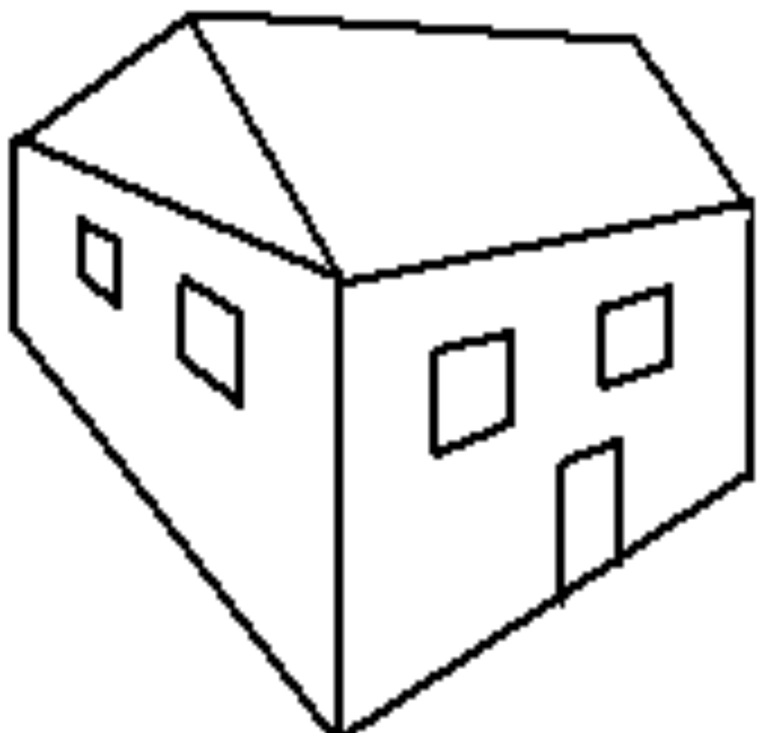
Computer Graphics

COMP H3016

Lecturer: Simon Mcloughlin

Lecture 5

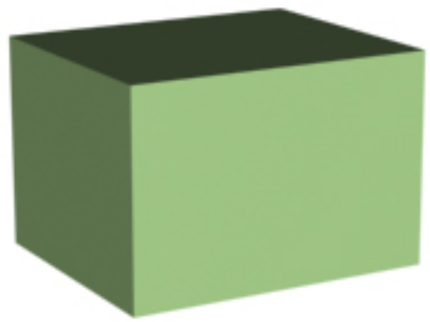


Review and Overview

- Last week we looked at how three dimensional objects are projected onto a 2-d viewing plane using perspective/orthographic (parallel), projection models
- The objects that we were dealing with were simple ones like lines or squares
- In a 3-d scene that we wish to model before projecting, there can exist objects of **many different shapes, colours, textures etc.**
- So not surprisingly there exists many **different methodologies** for describing these objects in a graphics system
- Identifying and utilising a methodology that accurately represents an object in 3-d is called "**three dimensional object representation**"
- We are going to look at some of the more popular (and general) techniques for object representation
- We will also look at how to calculate some properties of the facets that make up these objects (e.g. surface normal).
- We will also see how lighting can be applied to using these properties to give us some more realistic rendering

Polygons

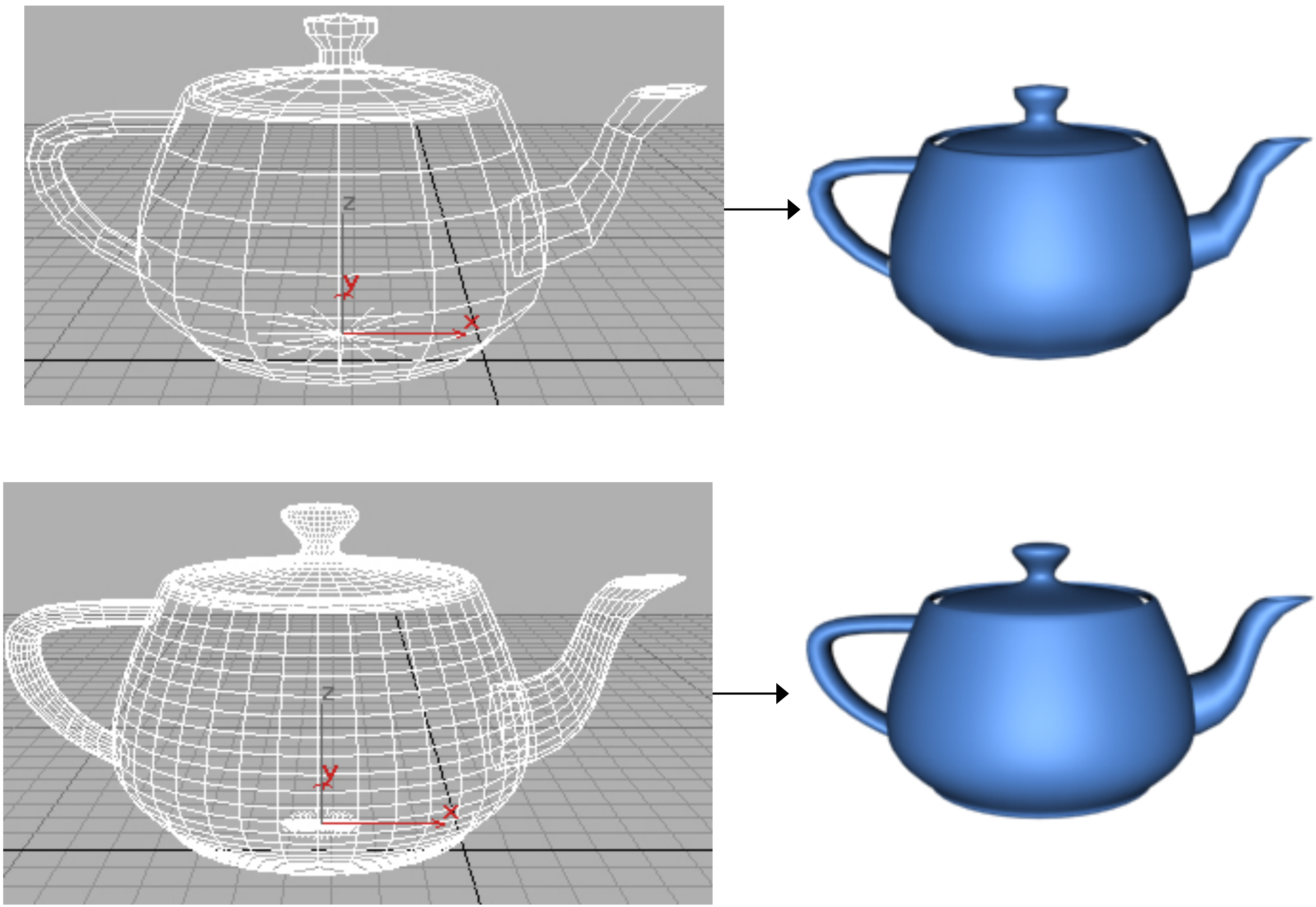
- Polygons are the predominant method for modelling surfaces.
- Objects with flat surfaces can be directly modelled with polygons (e.g. a box).



6 polygons

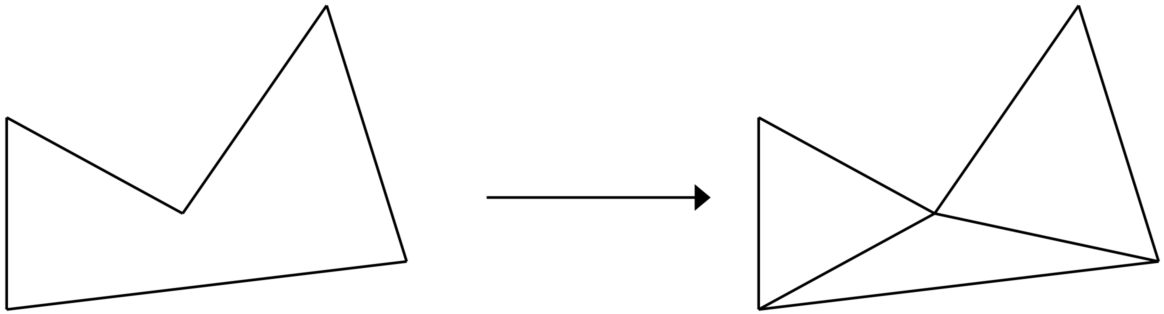
- Objects with curved surfaces are a bit more tricky though.
- A curved surface has to broken up into a **mesh** of joined up polygons, usually triangles or quadrilaterals. This process is called tessellation.
- Smaller polygons => finer polygonal mesh => better result.
- But smaller polygons => more polygons => more memory => slower !

Polygons



Polygon mesh

- The surface polygons are **tesselated or tiled** together to accurately represent the surface
- This type of surface representations is known as a **polygon mesh representation**
- Polygon Meshes give a **wire-frame** representation of the object
- **Realistic rendering** of the surface/object is accomplished by applying **lighting/shading models (later)** across the projected polygon surfaces
- The relative "**fineness**" of the mesh is often called the **tessellation level** (See the teapot on the previous slide for different tesselation levels)
- Every polygon can be represented as one or more **triangles** (triangles are the simplest form of polygon).



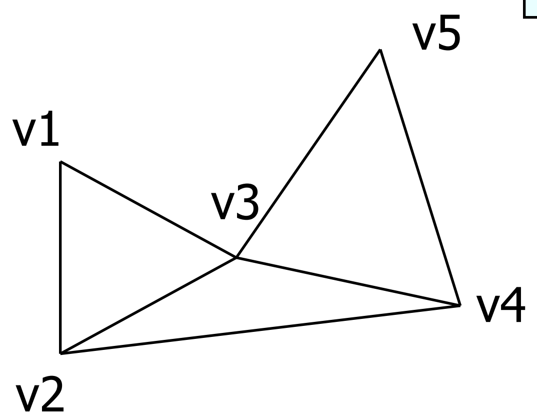
Polygon mesh

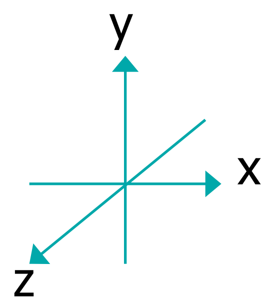
- Polygons can be easily stored in a graphics system as they are simply **lists of vertices** (each vertex is a 3D point). How many vertices define a polygon?
- An internal representation is probably going to store polygons as lists of vertices as follows.

Polygons
Polygon 1: v1, v2, v3
Polygon 2: v2, v4, v3
Polygon 3: v4, v5, v3

Vertices
v1: (0,10,0)
v2: (0,0,0)
v3: (10,4,0)
v4: (20,2,0)
v5: (18,16,0)

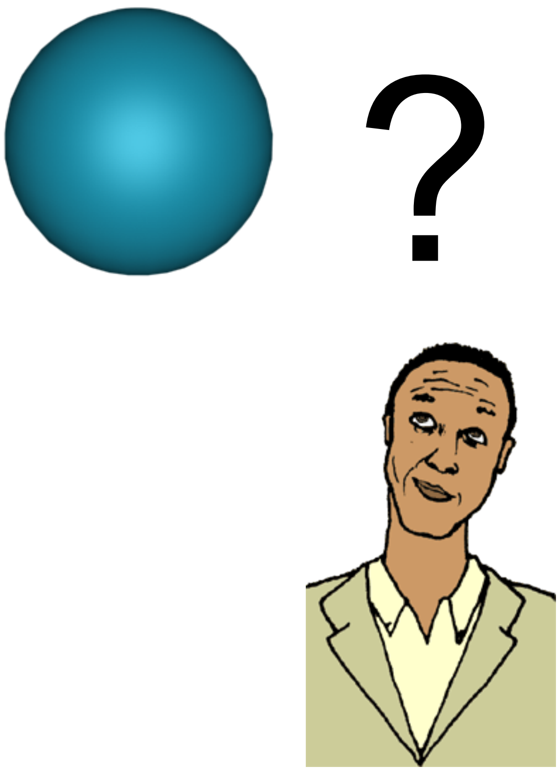
Do we need to store edges? Why not?





3-D descriptions

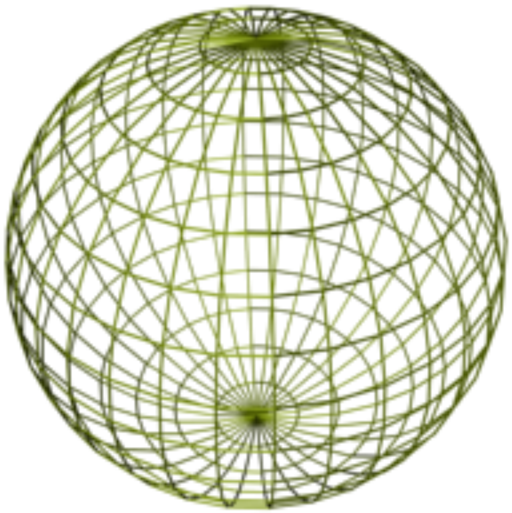
- A polygon representation will lead to a much less compact representation of the box than the obvious one …
- Press on anyway and ask – "can we do the same thing with the sphere?"

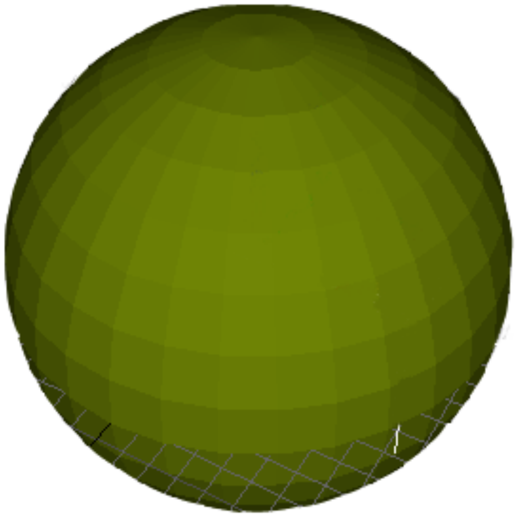


- Here we have a problem because the surfaces of the sphere are not flat?
- In fact there is really only one surface and it's curvy all round …
- The solution is to represent the sphere as a polygonal mesh
- This is an inter-connected mesh of polygons that approximates the surface of the sphere.
- Pay attention to the word approximate!

3-D descriptions

- Here's a polygonal mesh representation of a sphere.



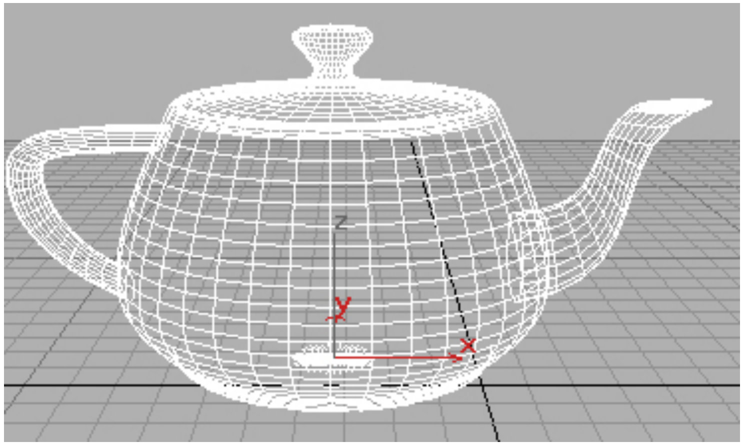


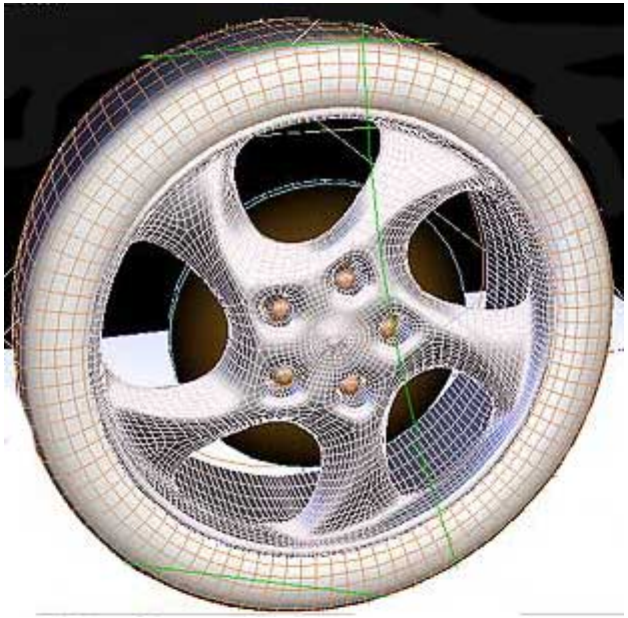
  - It's comprised of lots and lots of quadrilateral polygons.
  - Each of them is a small flat surface with four vertices.
  - The sphere representation now becomes a list of hundreds of polygons ….

3-D descriptions

- We can do the same for all our other primitives (cones etc..).
- Note the following:
  - A polygonal mesh is just an approximation of the original surface!
  - The representation is much less compact than the primitive representation.
- So what's the point of using it?
- The point is that we can use this idea to not just represent any primitive object but any type of object at all.
- The next slide shows polygon meshes of a number of different objects …
- The polygon count for complicated object like the helicopter overleaf can easily run into the thousands …

3-D descriptions





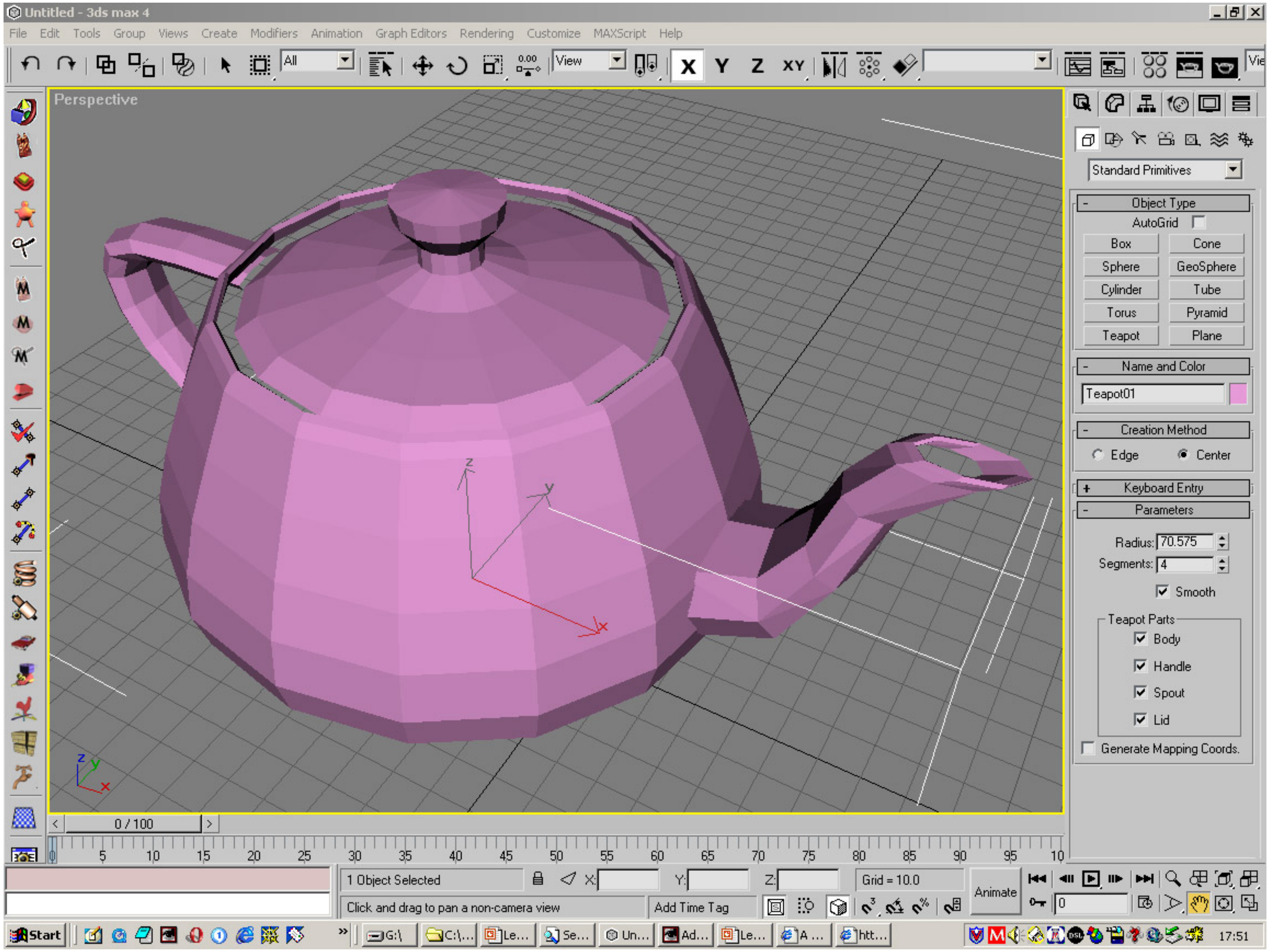
3-D Modeling

So how do we create these 3D models?

- … try and make a polygonal mesh by guessing positions of polygons … it's just not going to work for very complex objects (might do for simple ones)
- So we generally use some form of 3D modeling software. Such software provides an interface for creating and manipulating 3D objects. Examples include:
  - 3D Studio Max
  - SoftImage

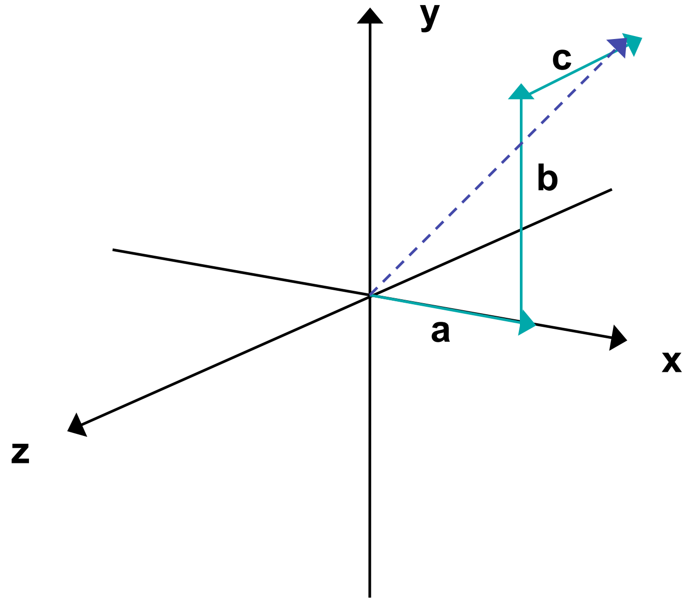
3-D descriptions

- The interface to one of these programs might look like this:



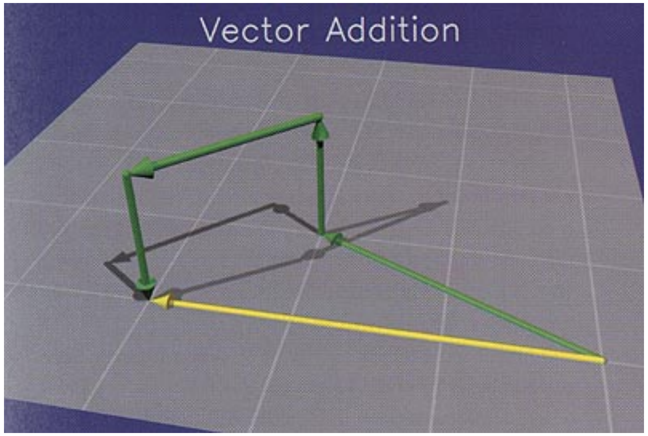
Polygon Surface Normals and 3D vectors

- The use of vectors is crucial to 3D graphics as they are the means used to specify directions – directions of cameras, light sources, orientations of objects and so on.
- We are now going to cover the basic of vectors in 3D space
- As before the vector *(a,b,c)* is like an instruction to move *a* units to the right, *b* units up, and *c* units forwards.



3D Vectors

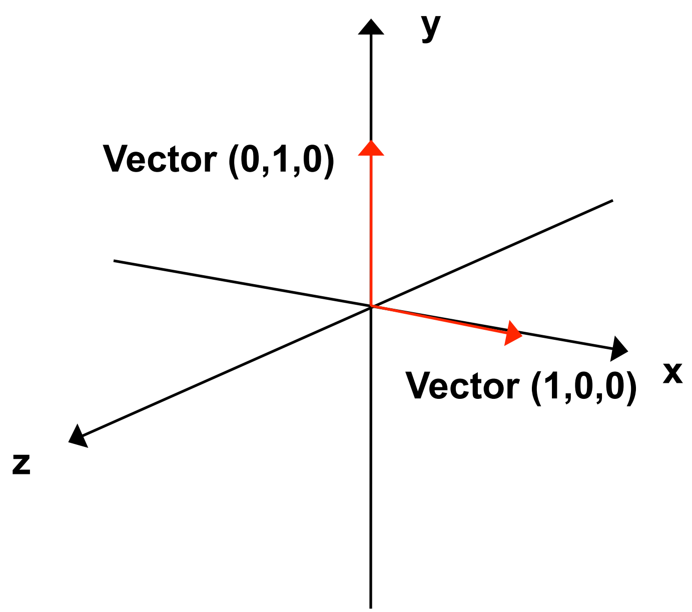
- The vector hence represents a direction in 3D space (represented by the blue dashed arrow in the diagram on the previous slide).
- The vector also has a length calculated by $\sqrt{a^2 + b^2 + c^2}$
- There are a number of mathematical operations we can carry out on vectors.
- The first of these is *addition*. We can add one vector to another by adding its corresponding components. The result is a new vector.
- So adding (a1, b1, c1) to (a2, b2, c2) will give (a1+a2, b1+b2, c1+c2).
- The sum of two or more vectors is simply the sum of their displacements.



- The yellow vector pictured left is the sum of the four green ones.
- The sum of (-2,1,0), (0,0,1),(-1,-1,0) and (0,0,-1) is (-3,0,0).
- Subtraction is done in the same way.

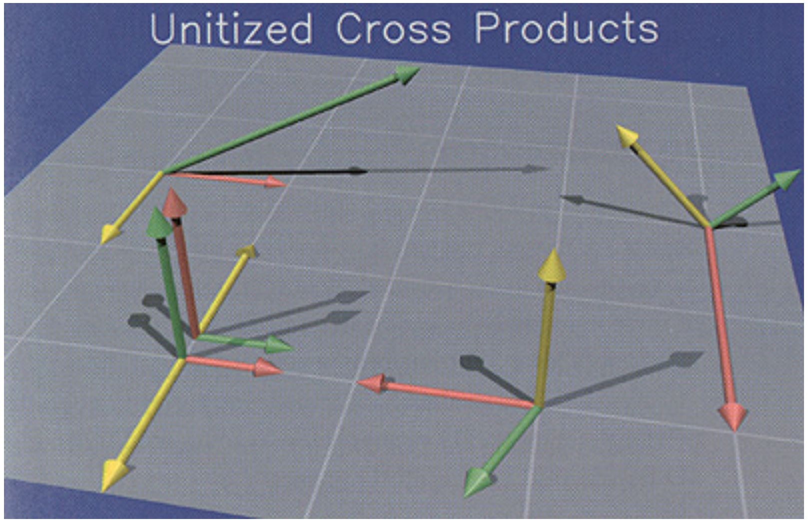
3D Vectors

- The next useful operation is the *dot product*. This returns the cosine of the angle between two vectors.
- The dot product of (a1, b1, c1) and (a2, b2, c2) is a1.a2+b1.b2+c1.c2.
- So the dot product of (1,0,0) and (0,1,0) is 1.0+0.1+0.0 = 0 which implies angle between them is 90 degrees. Correct !



3D Vectors

- Finally we have another operation called the *cross product* that can be carried out on two vectors.
- Unlike the dot product, the cross product of two vectors gives back another vector.
- The handy thing about it is that the vector it returns is perpendicular to *both* of the originals. In the picture below the yellow vectors are all cross products of the red and green.

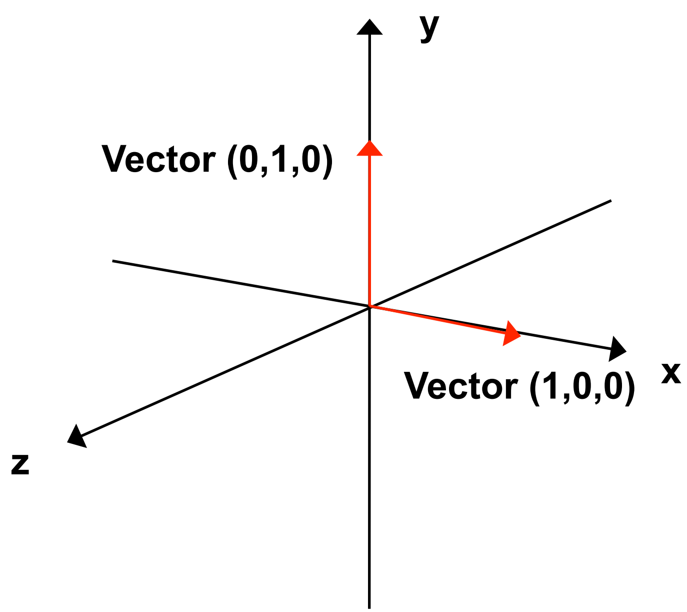


3D Vectors

- Calculate the cross product of two vectors (ax, ay, az) and (bx, by, bz) as follows.
- Cross product is the vector (cx, cy, cz) where

  cx = (ay.bz – az.by)
  cy = (az.bx - ax.bz)
  cz = (ax.by – ay.bx)

- So suppose we have the two vectors (0, 1, 0) and (1, 0, 0)



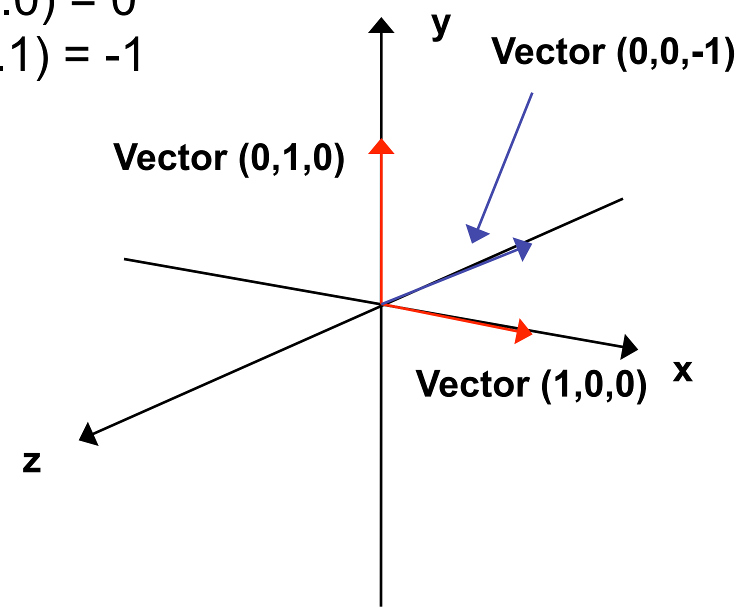
3D Vectors

ax ay az bx by bz

- Calculate the cross product of (0, 1, 0) and (1, 0, 0) as follows.
- Cross product is the vector (cx, cy, cz) where

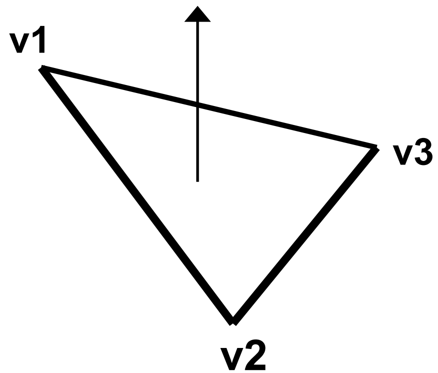
cx = (ay.bz – az.by) = (1.0 – 0.0) = 0
cy = (az.bx - ax.bz) = (0.1 – 0.0) = 0
cz = (ax.by – ay.bx) = (0.0 – 1.1) = -1

Gives us the vector (0,0,-1)

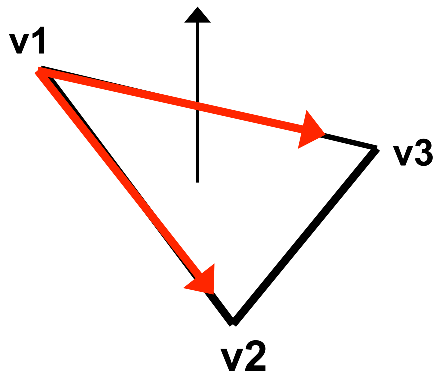


Normal Vectors to Triangles

- We have talked already about normal vectors
- How would we calculate a normal vector to a 3D triangle?
- Luckily there's a simple method … take the following triangle …



- The normal vector is shown pointing upwards and perpendicular to the surface.



- Now imagine a vector pointing from v1 to v2 and another one pointing from v1 to v3 (as shown on the left in red).
- The normal vector is the one that is perpendicular to both of these red vectors.

Normal Vectors to Triangles

- So all we have to do is calculate these red vectors and then get their crossproduct. That's the normal vector.

- How do we calculate them? To get the vector pointing a to b just subtract a from b.

- So the two red vectors are v2-v1 and v3-v1 where v1,v2, and v3 are the vertices of the triangle.

- The vector should be normalised by finding its length and dividing each component by it. Now it's the unit normal.

- So what use is it knowing the normal vector to a polygon anyway?

- Well, for one thing, they allow us to do Hidden-Surface Removal (later)!

Polygon Plane Equations

- The equation of a plane is linear and of the form

$$Ax + By + Cz + D = 0$$

- The (A,B,C) here are the components of the normal vector to the plane, N = (A,B,C). If we have three points (x1,y1,z1), (x2,y2,z2) and (x3,y3,z3) we can find the normal vector as seen previously.
- We can find the D value by substituting any point on the plane (x,y,z) with the normal vector (A,B,C) into the equation.
- Now we have the full plane equation

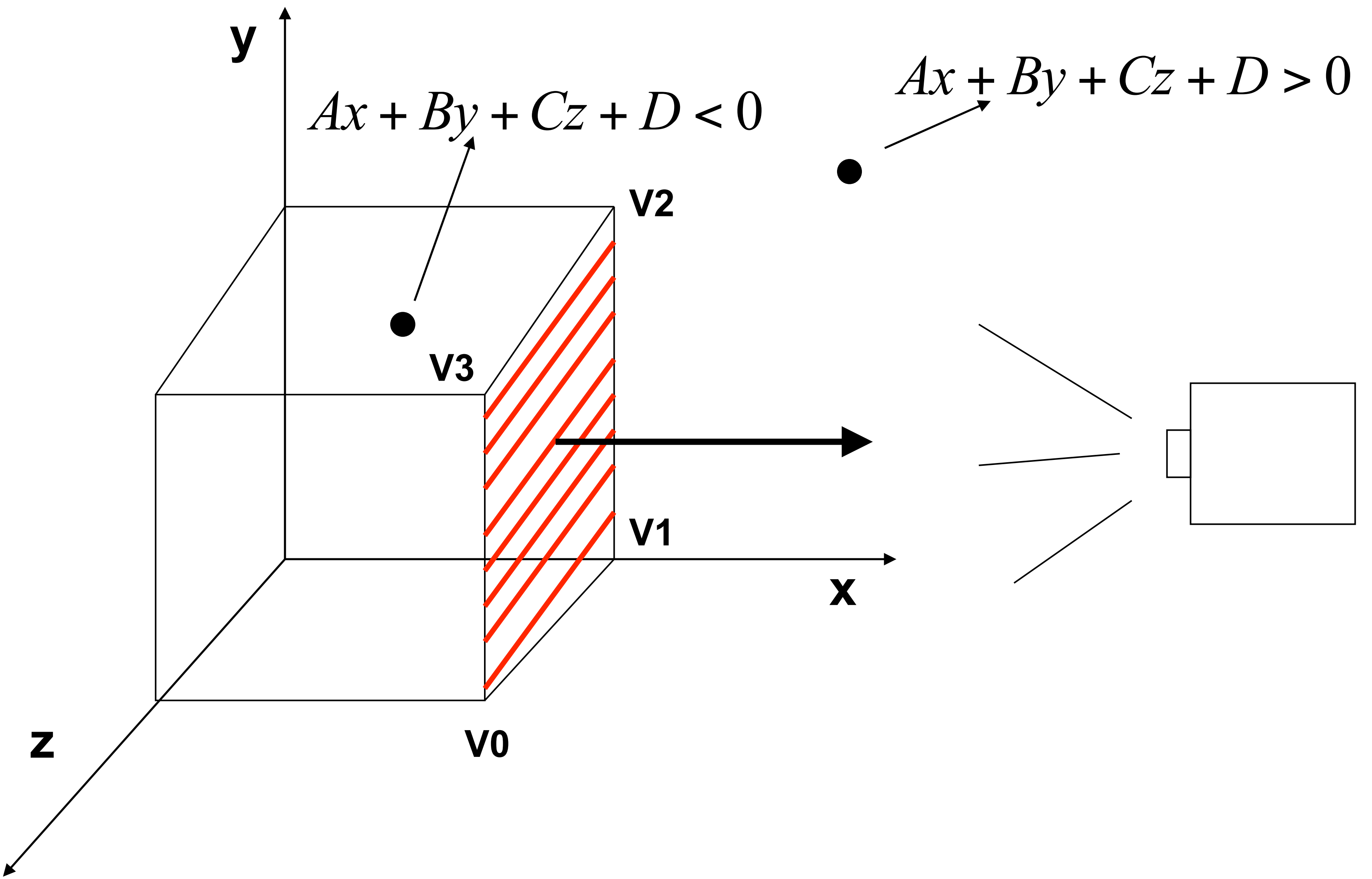
Polygon Plane Equations

- if $Ax + By + Cz + D < 0$ the point (x, y, z) is inside the surface
- if $Ax + By + Cz + D > 0$ the point (x, y, z) is outside the surface
- The inside of a polygon surface is **toward the object interior**
- The outside of a polygon surface is **toward the object exterior**
- If we specify polygon vertex coordinates in a **counter-clockwise direction** in a **right hand coordinate system** when viewing the outer side of the of the plane these equations will be satisfied
- The orientation of the of the polygon surface is given by the surface normal (N), that is a vector perpendicular to the surface

$$\mathbf{N} = (A, B, C)$$

- The surface normal direction will be from inside to out if the vertex direction/ viewing direction/coordinate system direction assumptions made above are satisfied

Polygon surface normal



Why plane equations?

- To produce an input display of a 3-d object we must process the representation of that object through **several steps**, **transformation of world coordinates** to **viewing coordinates**, then to **device coordinates**; **identification of visible surfaces** and the application of **surface rendering techniques**

- Knowing the **orientation of the polygonal planes in space** gives us an insight into how the **illumination model** will affect the surface (later)

- Secondly knowing whether points lie inside or outside a surface allows us to make various intersection tests with other structures in the scene and allows us to determine if other **structures are in front of/behind a polygon (later)**

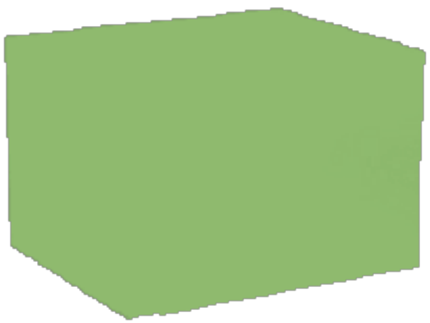
Review and Overview

- A major consideration in the generation of realistic graphics displays of 3-d objects is **determining which parts/components of these objects are visible** for a particular viewing direction
- Visible Surface Detection methodologies exist for finding **backfaces** (back-face detection based on surface normals) and **surfaces obscured** by other surfaces (using the depth buffer) (later!)
- This week we will look at creating realistic rendering of the visible surfaces
- This is achieved using **lighting** and **shading models**
- A lighting model is a mathematical/computational technique for computing the colour value at a single point on a 3D surface.
- A shading model is a mathematical/computational technique for colouring an entire surface.
- In particular we will look at the **Phong lighting model** and the **Gouraud shading algorithm**

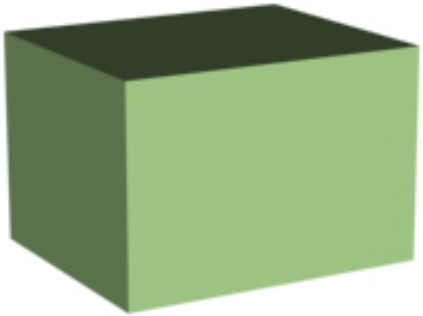
Introduction to Lighting

- So we need to choose a **colour/intensity** for each polygon. How do we do this?
- Well one option is that each object is **assigned some basic RGB** colour and then when the object is rendered we colour in its polygons with that colour.
- That’s not going to work too well because every polygon will be the **same colour** and hence we won’t be able to distinguish them from each other.
- Objects will appear to have **no solidity** or form.
- Our brains get many subtle shape cues from how the **brightness** of an object **varies across its surface**.
- The brightness varies across a surface depending on how much it **faces toward the predominant light**.
- Your brain interprets these brightness variations into **three-dimensional shapes**.

Introduction to Lighting



NO!



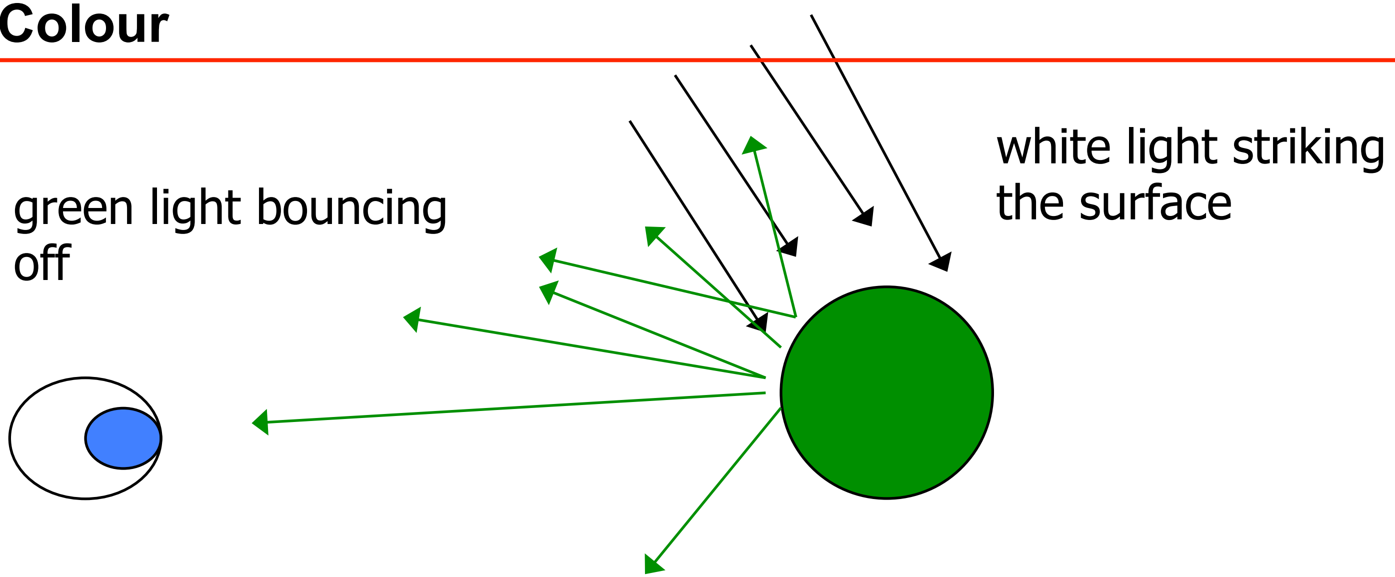
YES!

- What we have to do is to model light sources in our scene as well as objects and then **compute colour/intensity values** for polygons **based on their position with respect to these light sources**.
- Only by doing so will we be able to simulate lighting as it occurs in the **real world.**

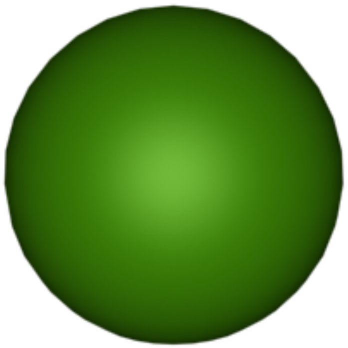
Light and Colour

- A light emitter emits light radiation (rays). When these rays strike an object some of them are **absorbed** into the object, some are **reflected off**, and some are **transmitted through**.
- If we are dealing with just **opaque objects** (not transparent) then the reflected rays dictate how that object appears to the eye.
- Objects are different colours because they reflect and absorb different rays across the spectrum.
- So a **green object** tends to **reflect the green parts of the light** that hits it and absorb the rest giving us the situation on the next slide.

Light and Colour

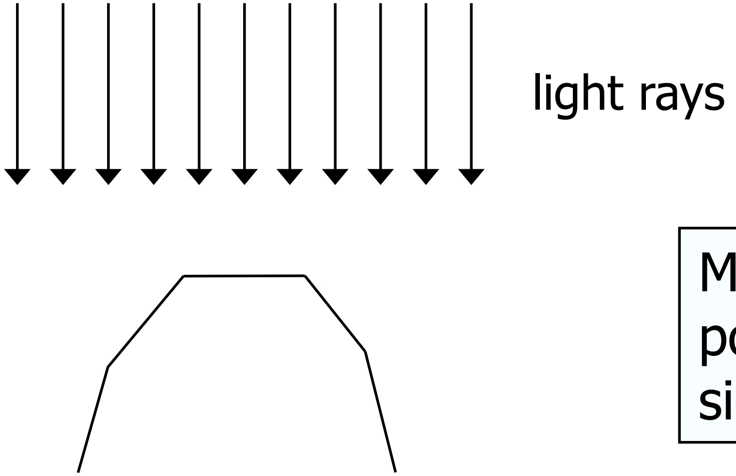


- So it appears green to the eye.
- If we **increase the intensity** of the light source then more light rays will be striking it and hence more will be reflected so the result will be a **brighter shade** of green.
- The sides of the sphere appear darker. Why?



Light and Colour

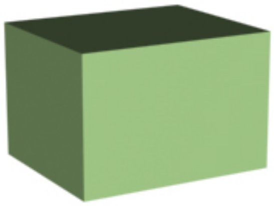
- The main reason for this is surface orientation.
- Suppose we have a sphere made up of equally sized polygons.
- Suppose also we are lighting it directly from above.



More hit the top polygon than the side ones!

Light Sources

- In computer graphics we often use the concept of a **point source**.
- This is a light source that is a point in space and emits light **equally in all directions**.
- Often we assume that point sources don't get dimmer with distance. This is completely **physically incorrect** and its just a convenience to make the computations easier.
- The **key factor** when lighting a polygon is to consider the **angle** that it makes with the **direction of the light striking** it.
- If the light is striking it from a **perpendicular direction** the polygon should be lit a brighter shade of the base colour.
- If the light is striking it at an **oblique angle** it should be lit a **darker shade**.
- Where's the light source here?



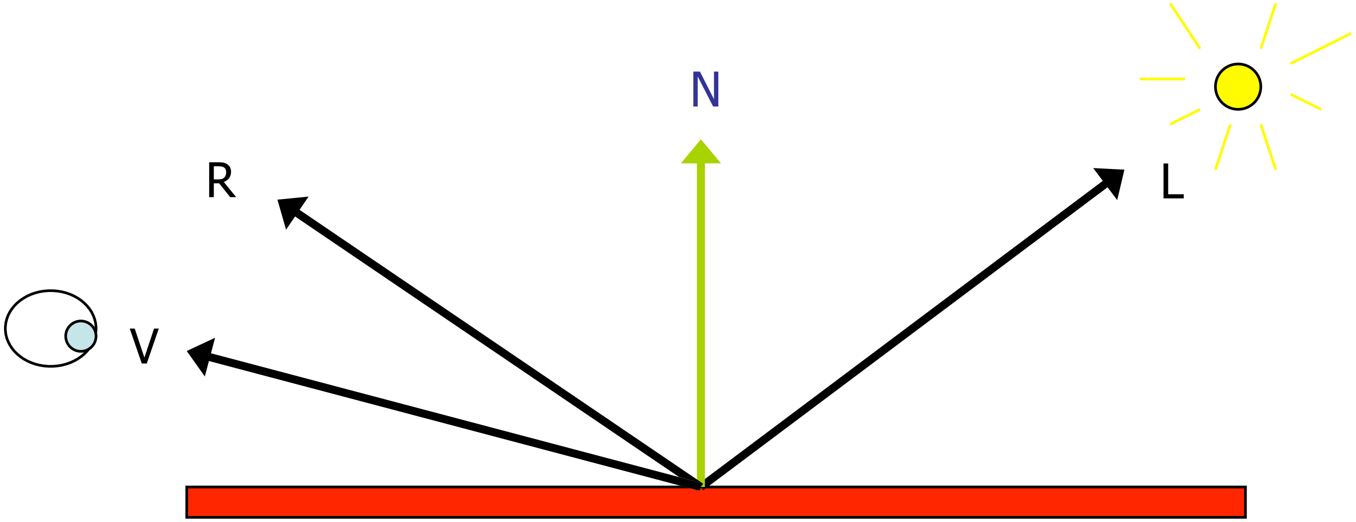
Lighting Models

- It’s all very well being able to specify light sources in a scene.
- How do we go about **computing lighting values**?
- Before we start talking about this we should make a distinction between **lighting models** and **shading models**. We use the term model in its mathematical/scientific sense …
- A lighting model is a mathematical/computational technique for computing the **colour value** at a **single point** on a 3D surface.
- A shading model is a mathematical/computational technique for **colouring** an **entire surface**.
- We will start by looking at the famous **Phong Lighting Model**.

The Phong Model

- The Phong lighting model allows us to compute the **colour/intensity** of an object at a **particular point** when viewed from a **particular direction**.
- The exact physics of how this works is extremely complicated but fortunately in computer graphics we can get reasonable results by making some **assumptions and approximations**.
- So the Phong model is **not a direct implementation** of the physics.
- To compute the lighting for a point according to the Phong model we need to know the following:
  - which way the surface is facing (i.e. the normal vector **N**)
  - where the light is coming from (a vector **L** towards the light source)
  - based on L and N we can compute a reflection vector **R**.
  - the direction we are looking from (the view vector **V**)
  - various surface properties (later)
- The diagram overleaf shows these vectors.

The Phong Model

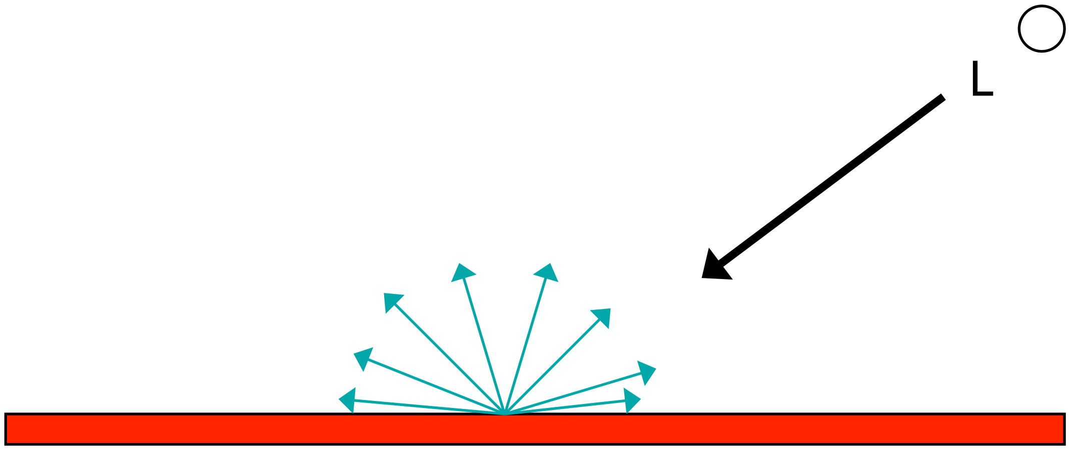


All of these vectors should be unit vectors!

The Phong Model

- So what are the **surface properties** that have to be defined?
- These refer to the **way that light will reflect** off an object.
- A simple lighting model would be too assign an **RGB value to an object** and then do some sort of computation to **return brighter or darker values** of this RGB value **based on the position** of the point in question.
- Things are not, however quite as simple as this.
- For the purposes of the Phong model we define two different ways in which light can reflect off an object's surface – **diffuse reflection** and **specular reflection.**
- **Diffuse reflection** means that when light strikes the surface of an object, the reflected light is **scattered equally in all directions**.
- The **diffuse colour** that results is what we think of as the **normal colour** of the object.

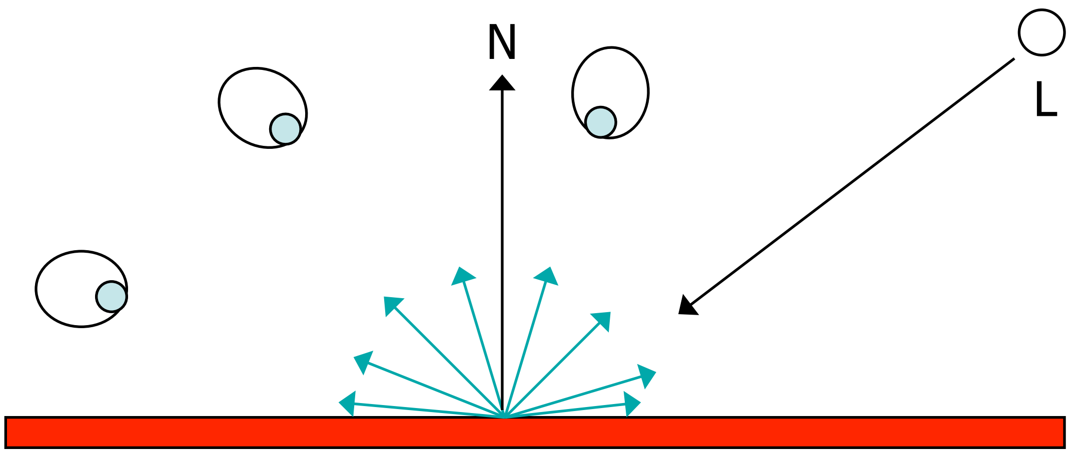
Diffuse Reflection with Phong



Diffuse Reflection – Light reflected equally in all directions

Diffuse Reflection with Phong

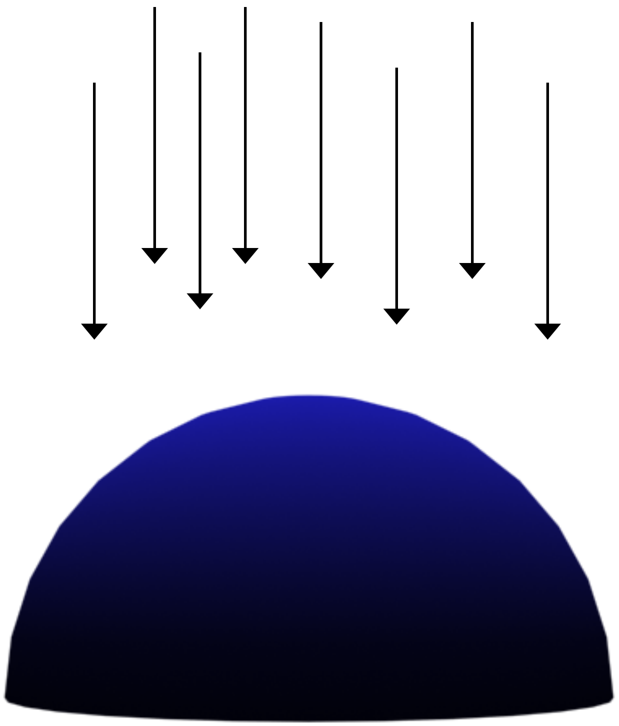
- The light that is reflected is the **base colour** of the object. So if the object is red then the **diffuse reflection** causes **red light** to be reflected off the surface of the object.
- **Crucial Point ! – The same light is reflected in all directions**. This means that if an object just exhibits diffuse reflection then it will **look the same from all viewing angles**.



Diffuse Reflection with Phong

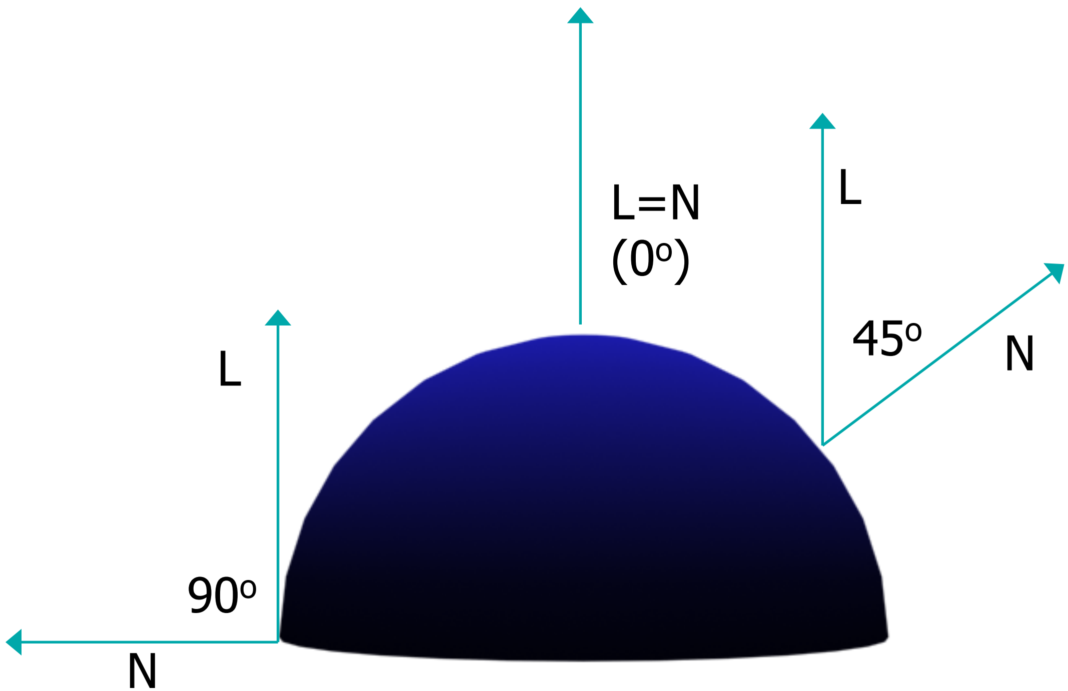
- **Dull matte** (non-shiny) objects are diffuse reflectors.
- So the viewing angle is not significant with diffuse reflection but the **angle** that the **light strikes the surface** most definitely is.
- If the light is **shining flat** onto the surface, it illuminates more than if it is **grazing the surface**.

- Light shining directly down on the top of a sphere.
- **Polygons at the top** are **brighter** because the **rays** are hitting directly **perpendicular**.
- Another way of looking at this is that the **angle between** the **surface normal** (N) and the direction to the **light source** (L) is close to **0**.
- See overleaf.



Diffuse Reflection with Phong

- For the **top polygon** the direction to the **light source** (L) is the **same** as the **surface normal** (N) so the angle is **0^o** => maximum diffuse reflection.
- For the **right polygon L** makes an angle of **45^o** with the **surface normal**.
- For the one on the **left the angle is 90^o**. => **minimum diffuse reflection** (i.e. none)



Diffuse Reflection with Phong

- So in other words the **amount of diffuse reflection** is directly **proportional** to the **angle** between the surface normal (**N**) and the light direction (**L**).
- We use the following formula to calculate it:

$$Id_r = I_s . kd_r . \cos\theta$$

$$Id_g = I_s . kd_g . \cos\theta$$

$$Id_b = I_s . kd_b . \cos\theta$$

- What do all these terms mean ….
- Idr is the **intensity** (strength) of the **diffusely reflected red light** from the surface. In practice it would usually be a number between 0 and 255.
- Similarly Idg is the green and Idb is the blue.

Diffuse Reflection with Phong

- So these three formulae in combination give us an **RGB colour** for a particular point.
- Is is the **strength of the light source**. I am assuming that our light sources emit **white light**. Therefore the amount of red, green, and blue in the light is equal.
- So this term is basically a number. It has **no direct physical justification** in terms of Watts or anything like that. Its really just something we can turn up and turn down until “**things look right**”.

Diffuse Reflection with Phong

$$Id_r = I_s.kd_r.\cos\theta$$

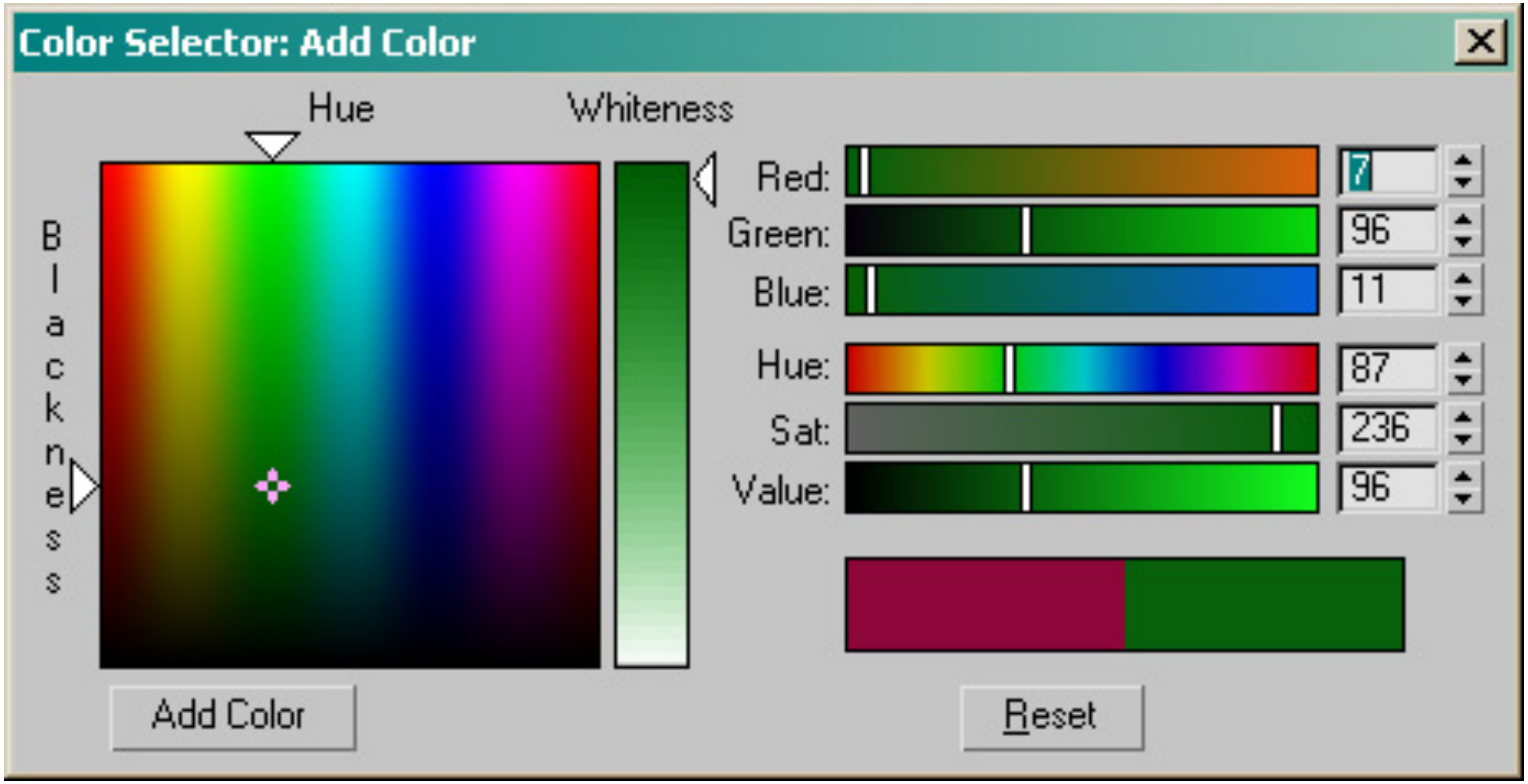
$$Id_g = I_s.kd_g.\cos\theta$$

$$Id_b = I_s.kd_b.\cos\theta$$

- Other terms in the formula.
- kdr is the **red diffuse reflection co-efficient** for the surface.
- Effectively this means the **proportion of the red light** that strikes the surface that is **diffusely reflected**. It’s specified as a number between 0 and 1.
- So, if a surface is **very red** it will have a **kdr close to 1**, otherwise close to **0**.
- kdg and kdb are the green and blue diffuse co-efficients.

Diffuse Reflection with Phong

- This governs the base colour of the object. So suppose we want an object that has the following colour (7,96,11):



Diffuse Reflection with Phong

- The modeling software will have to **scale things** so that the red component is a **number between 0 and 1** rather than a number between 0 and 255. Ditto for green and blue i.e..

$$kd_r = 7/255 = .035$$

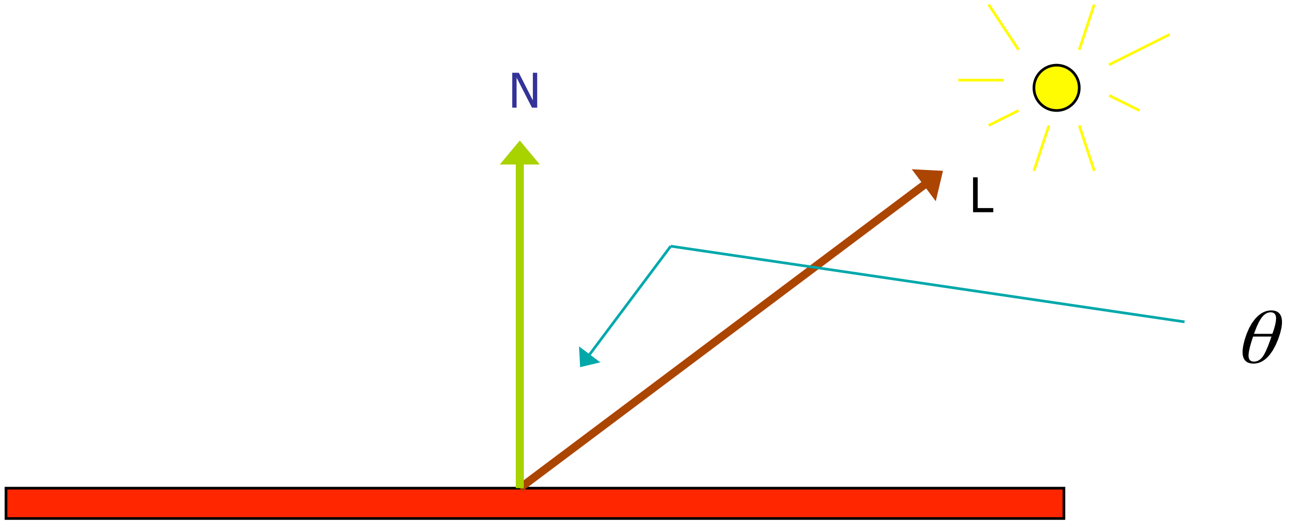
$$kd_g = 96/255 = .376$$

$$kd_b = 11/255 = .043$$

- If we wanted a **surface which is pure** white (255,255,255) the diffuse co-efficients would be set to (255/255,255/255,255/255)=(1,1,1).
- Similarly black would be (0,0,0).

Diffuse Reflection with Phong

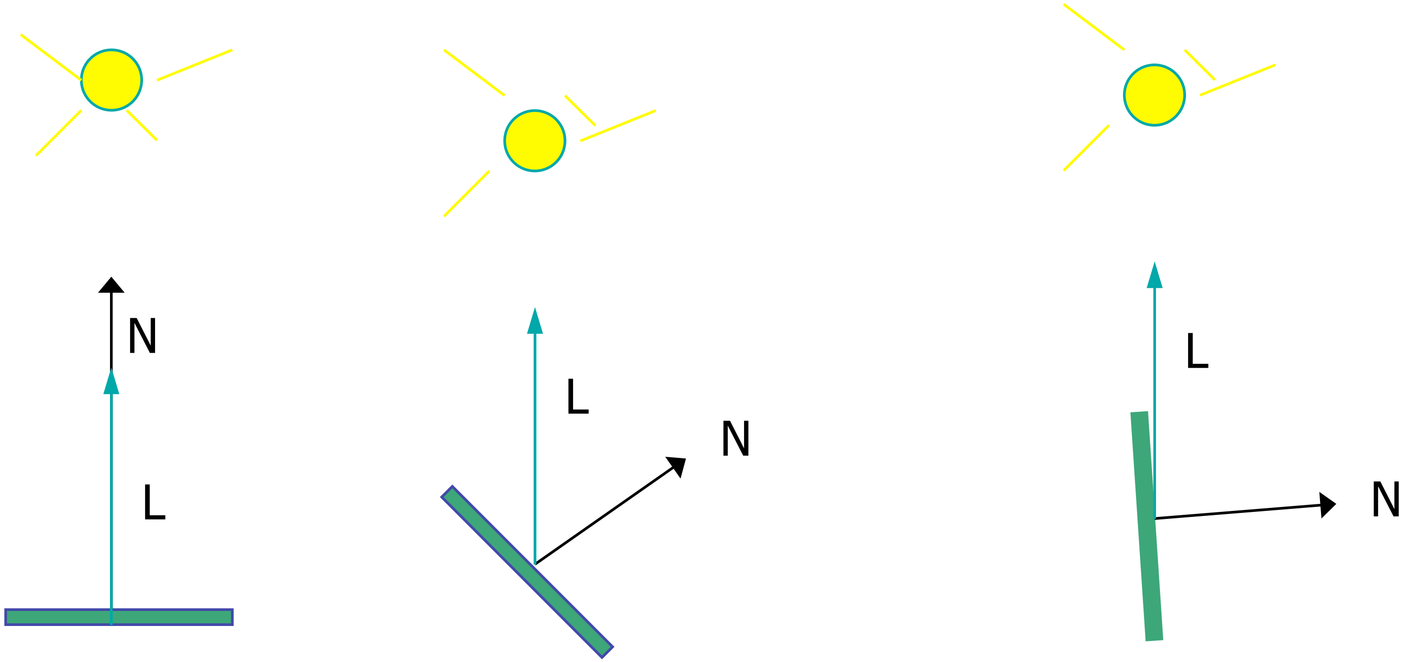
- Finally the cosine term … $\cos\theta$
- This is the cos of the **angle between** the normal vector **N** and the light **direction L**. Calculate as the **dot product** of N and L.



Diffuse Reflection with Phong

- Lets look at a few examples: $\cos\theta$
- Suppose the diffuse co-efficients are as we discussed already (0.035, .376, .043) and suppose Is is 510.

- Now let's look at three different cases.



$\theta = 0^o$ $\theta = 45^o$ $\theta = 89^o$

Diffuse Reflection with Phong

- The first case we get:

$$Id_r = (510)(.035)(\cos 0) = (17.85)(1) = 18$$

$$Id_g = (510)(.376)(\cos 0) = (191.76)(1) = 192$$

$$Id_b = (510)(.043)(\cos 0) = (21.93)(1) = 22$$

- So the colour we see is (18,192,22) which is

Diffuse Reflection with Phong

- The second case we get:

$$Id_r = (510)(.035)(\cos 45) = (17.85)(.707) = 13$$

$$Id_g = (510)(.376)(\cos 45) = (191.76)(.707) = 136$$

$$Id_b = (510)(.043)(\cos 45) = (21.93)(.707) = 16$$

- So the colour we see is (13,136,16) which is a darker shade of the same colour.

Diffuse Reflection with Phong

- The third case we get:

$$Id_r = (510)(.035)(\cos 89) = (17.85)(.017) = .311 = 0$$

$$Id_g = (510)(.376)(\cos 89) = (191.76)(.017) = 3.25 = 3$$

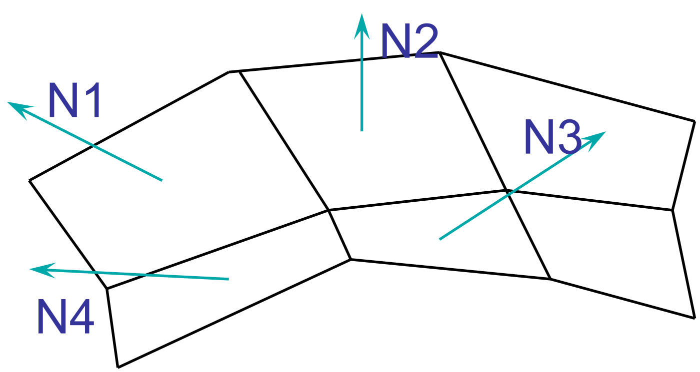
$$Id_b = (510)(.043)(\cos 89) = (21.93)(.017) = .37 = 0$$

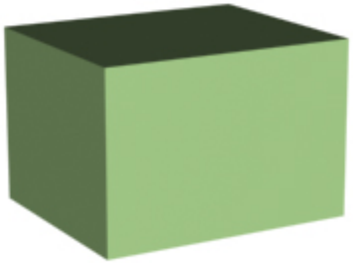
- So the colour we see is (0,3,0) which is almost black

Constant Shading

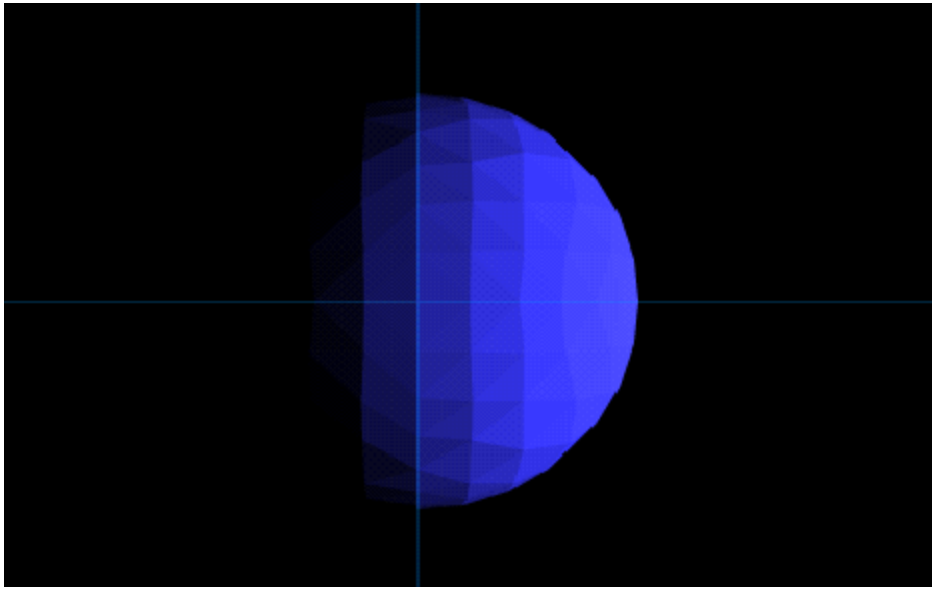
- We saw how to compute a lighting value (based on diffuse reflection) for a **point on an polygon surface**.
- We saw how to represent an **object as a mesh of polygons**.
- We can then render our model using a technique known as **constant shading**. This techniques proceeds as follows:

  - Pick **a point** on the surface of the polygon
  - **Apply the lighting model to get a colour value for this point** (do this based on the normal vector for the polygon)
  - **Project the polygon** and then **fill the projected polygon with this colour**.





Constant Shading



- This example doesn't look too good.
- The difference with this object is that the mesh is an approximation of a curved surface.

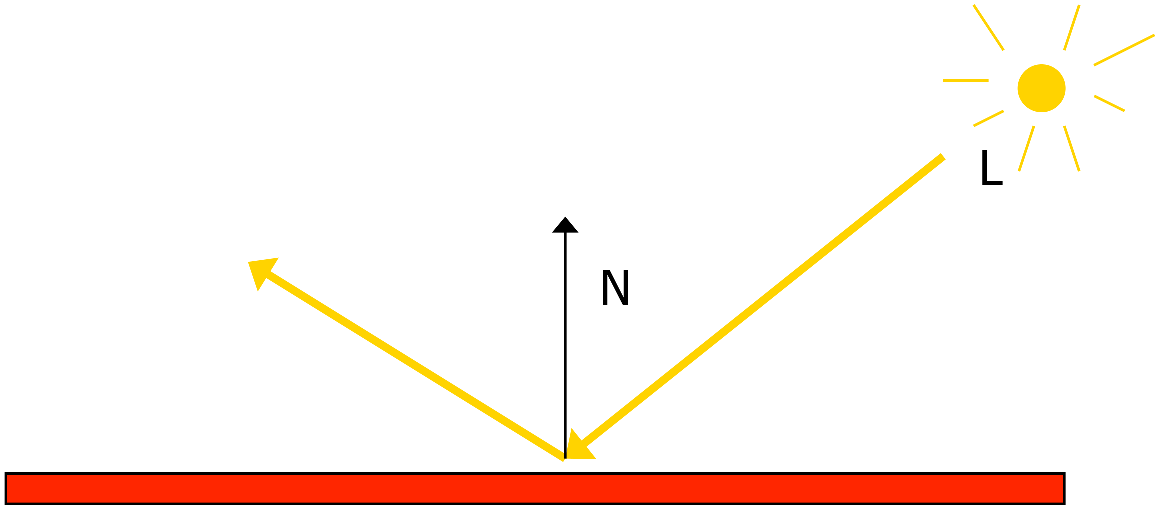
Light Sources

- We also commonly use a concept called **Ambient Light**. This is a convenient hack implemented in most graphics packages and is intended to model the light bouncing around the scene and being reflected from object to object.
- It is some **constant light intensity** that is assumed to strike all points on all surfaces.
- You can think of it as the **all surfaces emitting rays of light in all directions**, therefore where ever the view point is they will be visible.
- Any part of a scene **that is not directly lit** (e.g. under a desk) will **appear black** unless we add in this ambient light factor.
- Because it is constant it **does nothing for shape perception** but prevents **indirectly lit areas** of the scene from **appearing completely black**.
- Overleaf is a scene lit with **Ambient light only**.
- It simulates a constant level of illumination throughout the scene.

Light Sources

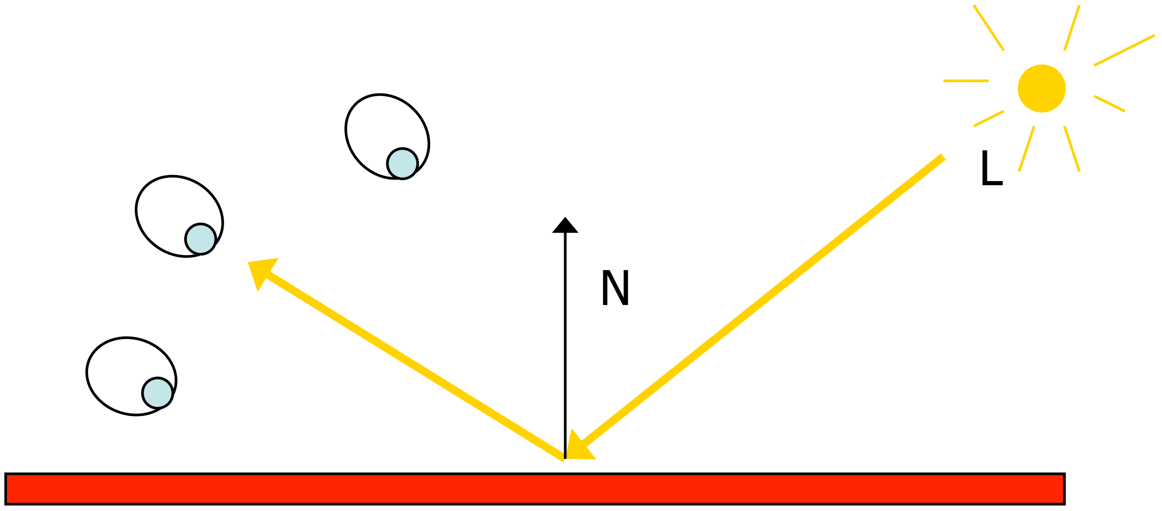
Specular Reflection with Phong

- Diffuse reflection handles dull matte surfaces but what about things that are **shiny**?
- Things get a bit trickier in this case and is handled by the concept of **specular reflection**.
- Specular reflection occurs when light reflects off a surface **in one principal direction**.



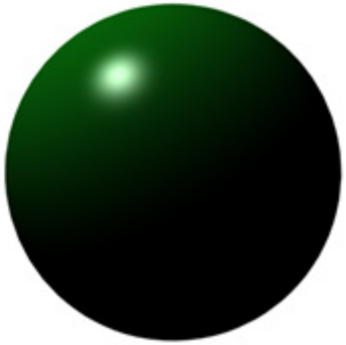
Specular Reflection with Phong

- It reflects in the direction of the **Reflection vector** which can be calculated using the incoming **light direction** and the **surface normal**.
- **Shiny surfaces exhibit specular reflection**.
- In this case the angle at which we are looking at the point has a **major effect on what we see**.



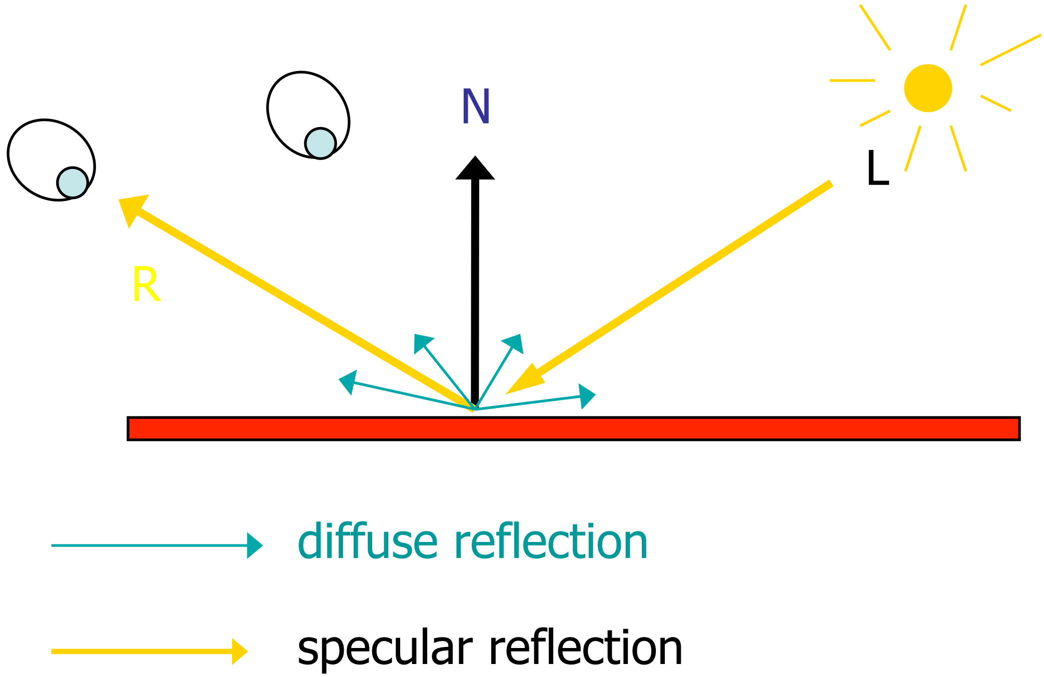
Specular Reflection with Phong

- In fact, on the diagram on the previous slide, only **one of the viewpoints** will **receive any reflected light** at all!
- If the **eye moved away** from this viewing angle it **wouldn't perceive any reflected light**.
- We can see this effect in practice by **looking at a shiny object**, fixing our **gaze on a point**, and then moving our viewpoint around.
- Specular reflection causes a **phenomenon** called **specular highlights**. The classic example of this is the ball of light we perceive on the surface of a **pool ball**.



Specular Reflection with Phong

- In computer graphics when we use the Phong model we assume that objects display a **mixture of diffuse and specular reflection** so the situation is as below.



Specular Reflection with Phong

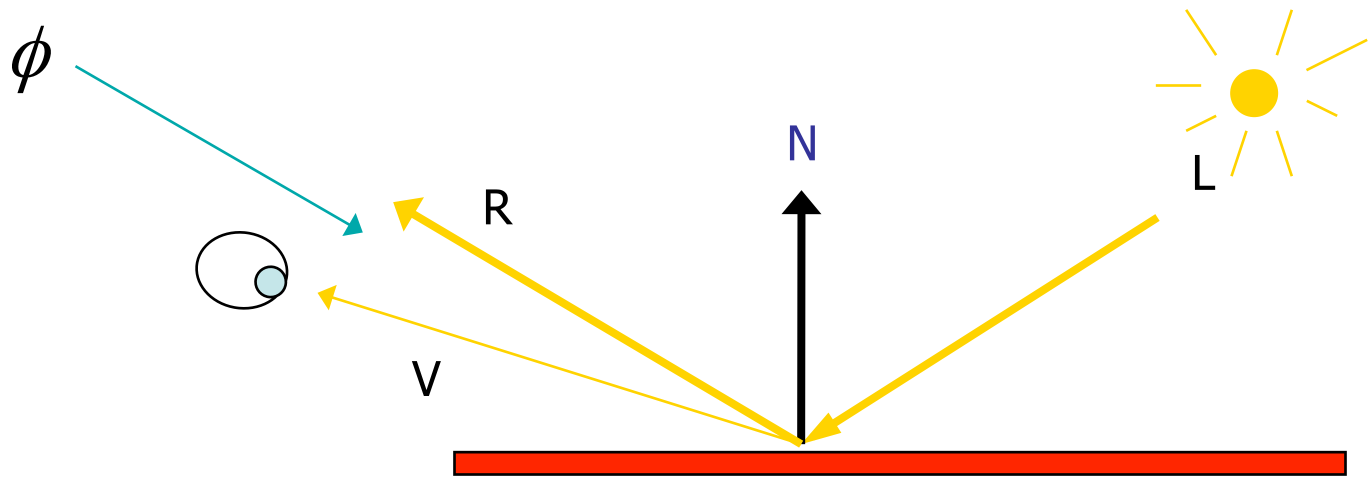
- Furthermore we assume that the **specular reflection is white**. In other words it does **not split incoming light** into colour components.
- If you look at the diagram on the previous page. The leftmost viewpoint is **right in the path of the specular reflection** so it will perceive a **mixture** of red diffuse (assuming the surface is red) and white specular i.e. a highlight!
- The other viewpoint will receive **no specular** and just **red diffuse**.
- The formula to calculate the specular part is:

$$Isp = I_s . ks . (\cos\phi)^n$$

- So Isp is the **intensity of the specularly reflected light**. Is is the strength of the light source as before.
- ks is the **specular reflection co-efficient** associated with the surface.

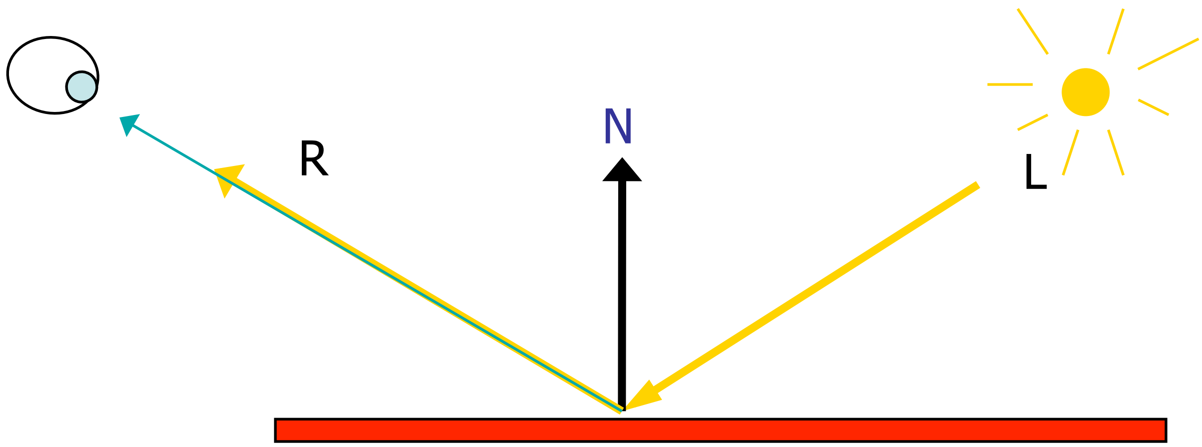
Specular Reflection with Phong

- Intuitively **ks is the proportion of the incoming light** that is **reflected** in a **specular manner**. Again it's a number between 0 and 1.
- **Dull matte** surfaces would have **ks of 0**, meaning none of the light is specularly reflected.
- The angle ϕ is the angle between the reflection direction (R) and the viewing direction (V).



Specular Reflection with Phong

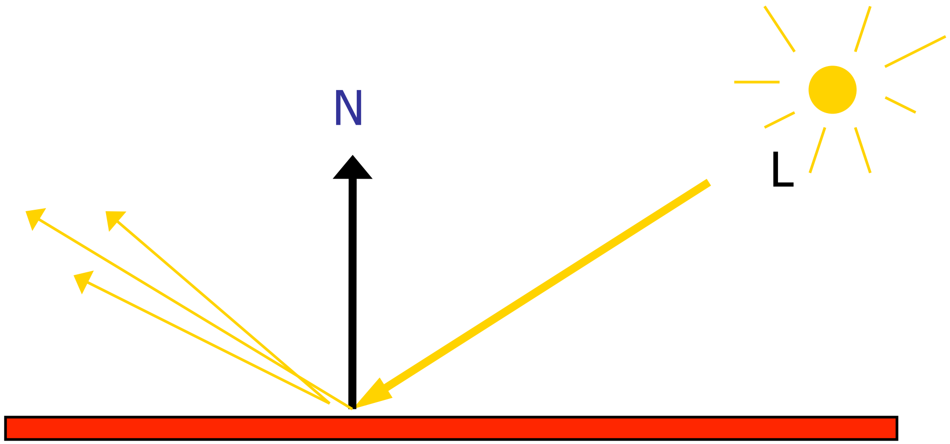
- Suppose that **angle is zero**….. Then we have the situation below.



- A cosine of zero gives 1, a cosine of 45 gives .707, a cosine of 90 gives 0.
- So in other words, because of the cosine term we get **maximum specular reflection** when the **angle is zero** or when we are looking directly down the specular reflection angle.

Specular Reflection with Phong

- What about n ?
- The **cosine term is raised to the power of n** which is the **specular reflection parameter**.
- In reality **light is not specularly reflected along a single reflection direction**. It tends to be **reflected over a cone** of directions as shown below.

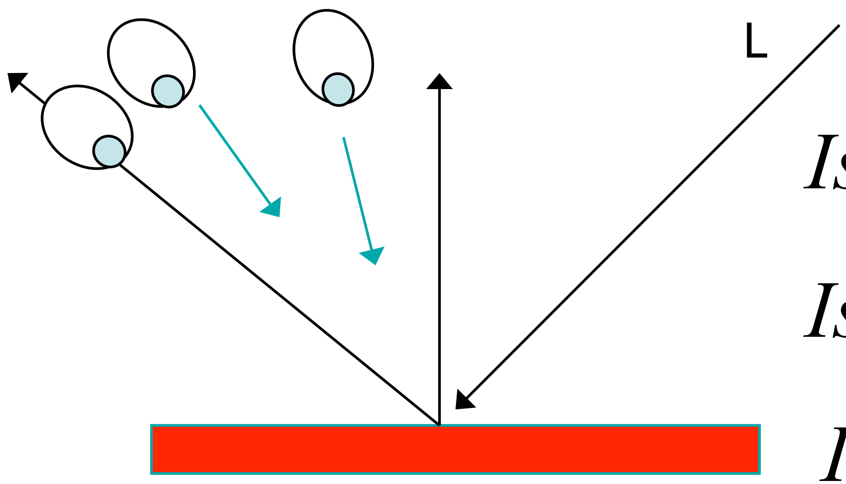


Specular Reflection with Phong

- **Different** types of **surfaces** have different **width cones**.
- For example **extremely shiny surfaces** like polished silver are almost mirror-like and would **reflect over a very narrow cone**. A mirror is a perfect specular reflector and reflects along a single line.
- Other shiny surfaces like **plastic** or **wood** reflect over a **wider cone** of directions.
- The **specular reflection parameter** controls the **width of the cone over** which the surface reflects specularly.
- **Extremely shiny** surfaces would have parameters of **100** or more.
- Let’s take an **example** where we have two surfaces – one with **n of 2** and one of **n of 200** to see how this works.
- Suppose **Is** , the light strength is **510** and ks, the specular reflection co-efficient of the surface is **.5**.

Specular Reflection with Phong

- We'll measure the specular reflection at three different viewing angles (0, 10, 45) degrees for each surface.



$$Isp1 = (510)(.5)(\cos 0)^2 = 255$$

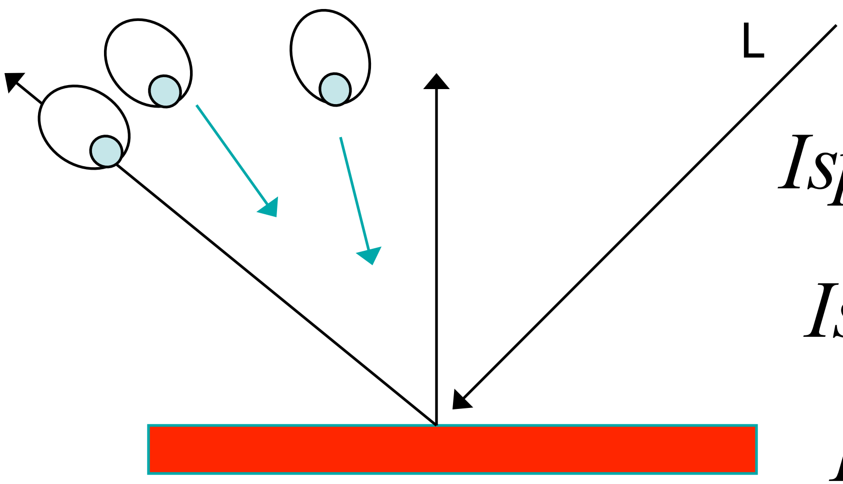
$$Isp2 = (510)(.5)(\cos 10)^2 = 247$$

$$Isp3 = (510)(.5)(\cos 45)^2 = 127$$

Even at a **viewing angle of 45 degrees** we are still seeing a fair amount of specular reflection. So a parameter of **2 means a very wide cone**

Specular Reflection with Phong

- Now let’s change the parameter to 200.



$$Isp1 = (510)(.5)(\cos 0)^{200} = 255$$

$$Isp2 = (510)(.5)(\cos 10)^{200} = 12$$

$$Isp3 = (510)(.5)(\cos 45)^{200} = 0$$

With a paramter of 200 **we get the same at 0**, much less at 10, and **none at all at 45 degrees.** So the cone is much tighter and we get a much shinier surface.

Specular Reflection with Phong

ks=1,n=30,kd=0

Wide cone, no colour because of lack of Diffuse component (kd)

ks=1,n=75,kd=0

Narrower cone because of larger n.

ks=1,n=30,kd="blue"

Wide cone, background colour because we have added diffuse colour.

ks=1,n=75,kd="blue"

Narrower specular cone

Specular Reflection with Phong

ks=.5,n=30,kd="blue"

Wide cone, less specular Reflection (ks)

ks=.5,n=75, ="blue"

Narrower cone because of larger n.

- The above examples are the same as the second pair on the last slide except there is half as much specular reflection in each (ks is .5). Results are more natural looking?