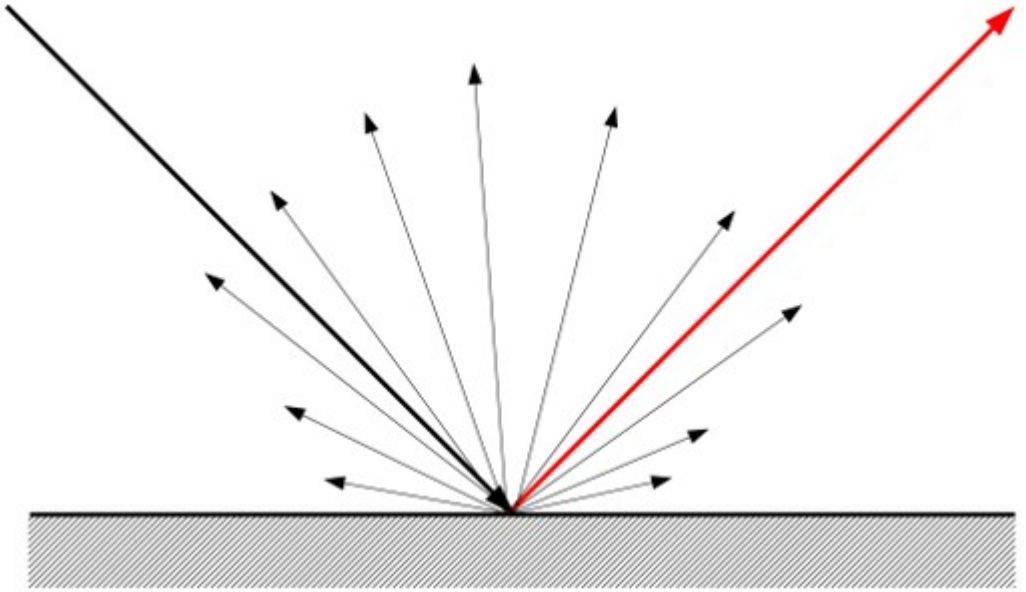
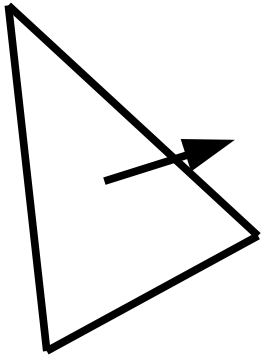
-=Bui Tuong Phong's Lighting=-
University of Utah, 1973
but with shaders

Anton Gerdelan – Trinity College Dublin

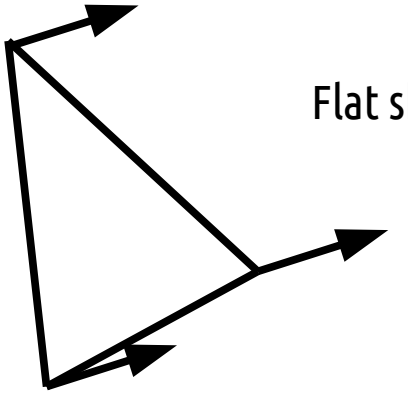


Before we do anything - normals

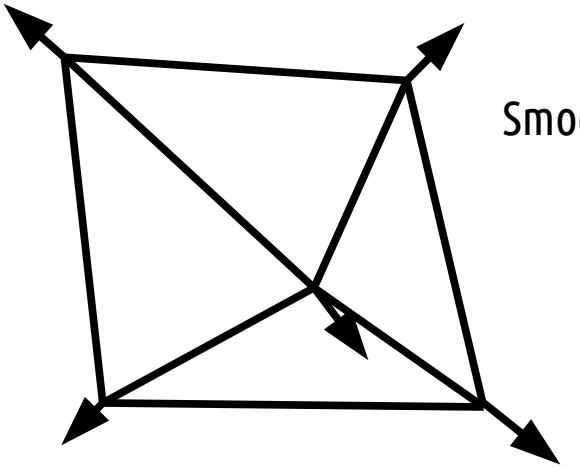
- **Q. What does a normal do?**
- **Q. How do we usually calculate them?**
- A per-vertex unit vector
- Interpolated to per-fragment
- If all per-vertex normals on triangle face the same way
  - Flat face shading
- If each per-vertex normal is average of surrounding faces
  - Smoothed face shading



Usually calculated for a face



Flat shading



Smooth shading

Literature

- **Bui Tuong Phong's** PhD thesis "*Illumination for computer generated pictures*", 1973 – **U.Utah**
  - Also describes a per-pixel shading method "Phong Shading"
- **James Blinn** "*Models of light reflection for computer synthesized pictures*" proc. CGIT, 1977
- Blinn-Phong was the built-in default lighting method in older GL and D3D

Background

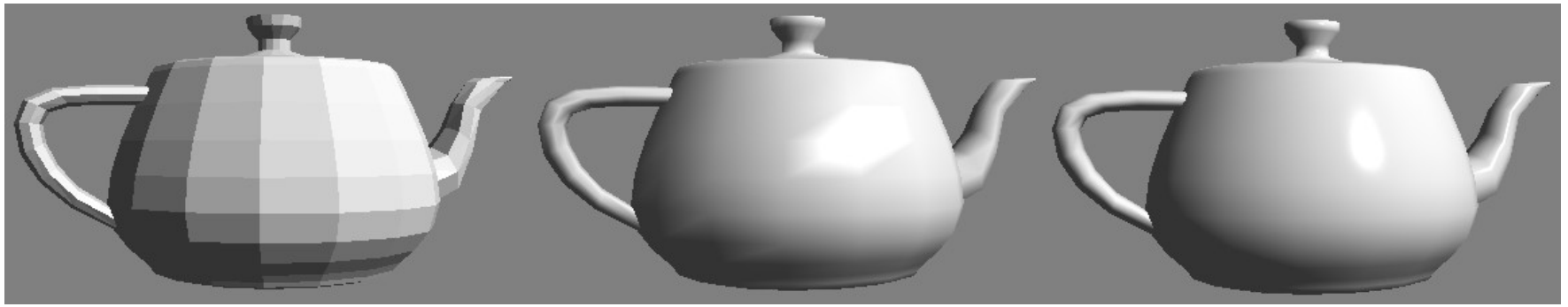
- Realistic lighting models based on Optics (physics of light)
- Radiosity – consider energy absorption by surfaces
- Light rays that reflect/refract on multiple surfaces
- Hence "**global illumination**"
- Expensive to calculate

Background

- Real-time rendering favours quick approximations
- **Local illumination** techniques
  (only consider one surface)
- Diminishing colour with distance from view (Doom)
- Manually pre-setting colour of world areas (DN3D)
  - We still do this with pre-rendered **light maps**
- Mixture of light maps and dynamic lighting (Quake)
- Introduce some GI techniques?

Shading Models

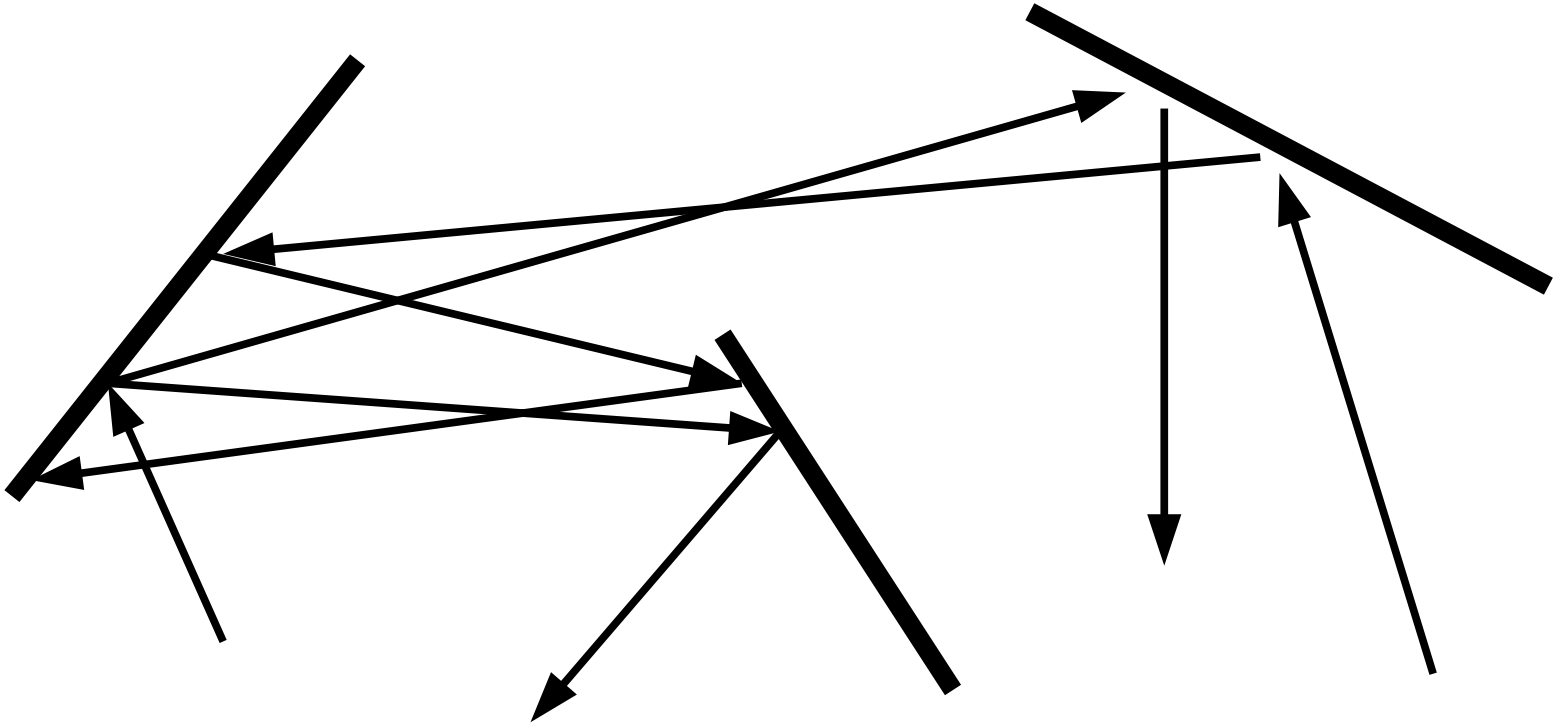
- **Per-Facet**
- **Gouraud** (per-vertex and then interpolated)
  Henri Gouraud "*Continuous shading of curved surfaces*" (1971),
- **Phong** (per-pixel or per-fragment)
- **Q. Where would we code each of these?**



demos

Ambient Lighting

- Approximate accumulation of general background light reflections by a single number



"Too hard – everything gets +(0.1,0.1,0.1)"

Ambient Reflection

i_a = l_a * k_a

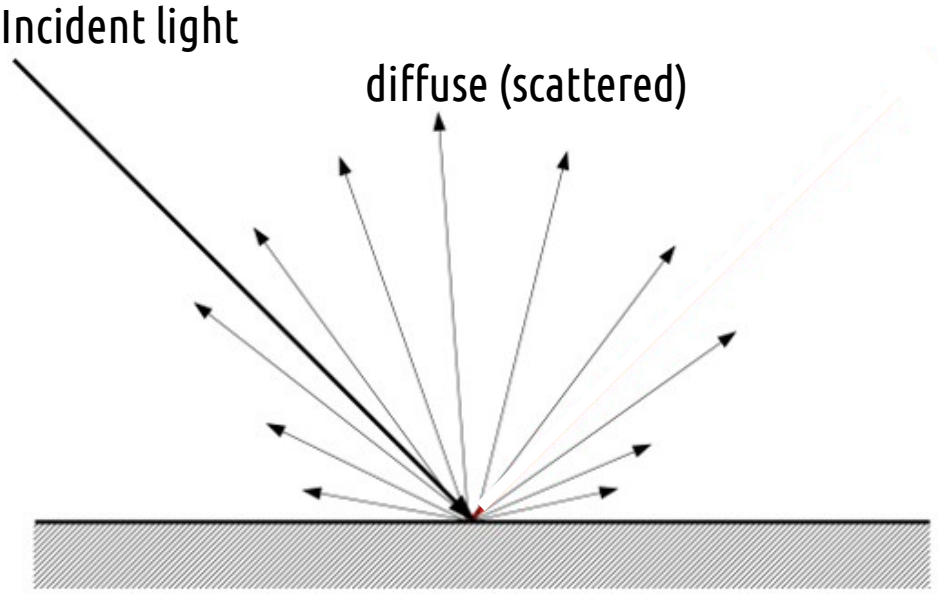
```
vec3 l_a = vec3 (0.2, 0.2, 0.2);
vec3 k_a = vec3 (1.0, 0.0, 0.0);
```

← Light's ambient colour

← Reflection / material colour

Diffuse Reflection (Lambertian Reflectance)

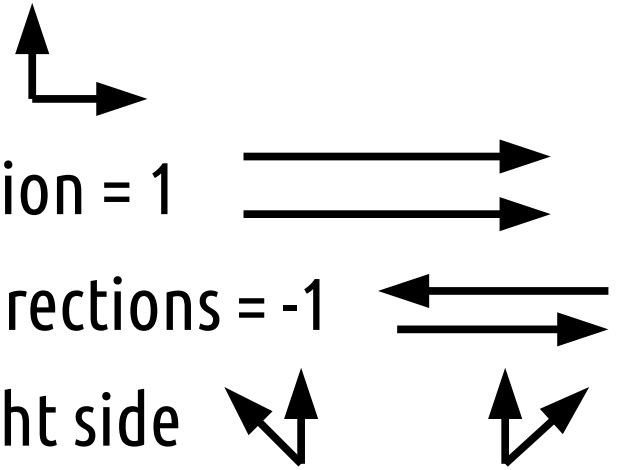
- Approximate light hitting surface and scattering in all directions
- Optics model
  - Johann Heinrich Lambert, "*Photometria*", 1760.
- As surface is perpendicular to light = most reflective
- Parallel to light = not reflective at all
- **Q. How can we calculate this angle/factor idea?**

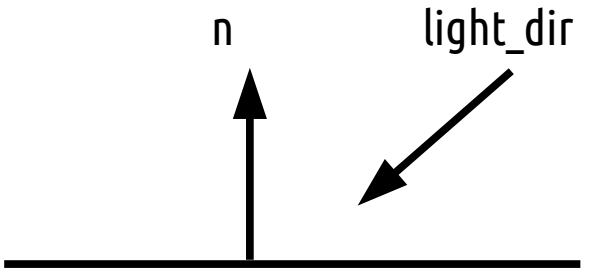


Note: perfect (equal) diffusion is assumed

The Vector Dot-Product

- Gives cosine of angle between 2 vectors
- Perpendicular = 0
- Parallel, same direction = 1
- Parallel, opposing directions = -1
- No notion of left/right side





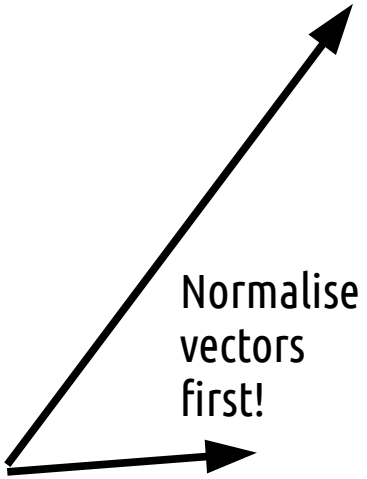
Q. Angle between light and surface?

$\vec{A} \cdot \vec{B} = \sum_{i=1}^{n} A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_n B_n$

The dot product returns a single **scalar** value. $\theta = arccos(\hat{A} \cdot \hat{B})$

$\theta = arccos(\frac{\vec{A} \cdot \vec{B}}{||\vec{A}||||\vec{B}||})$

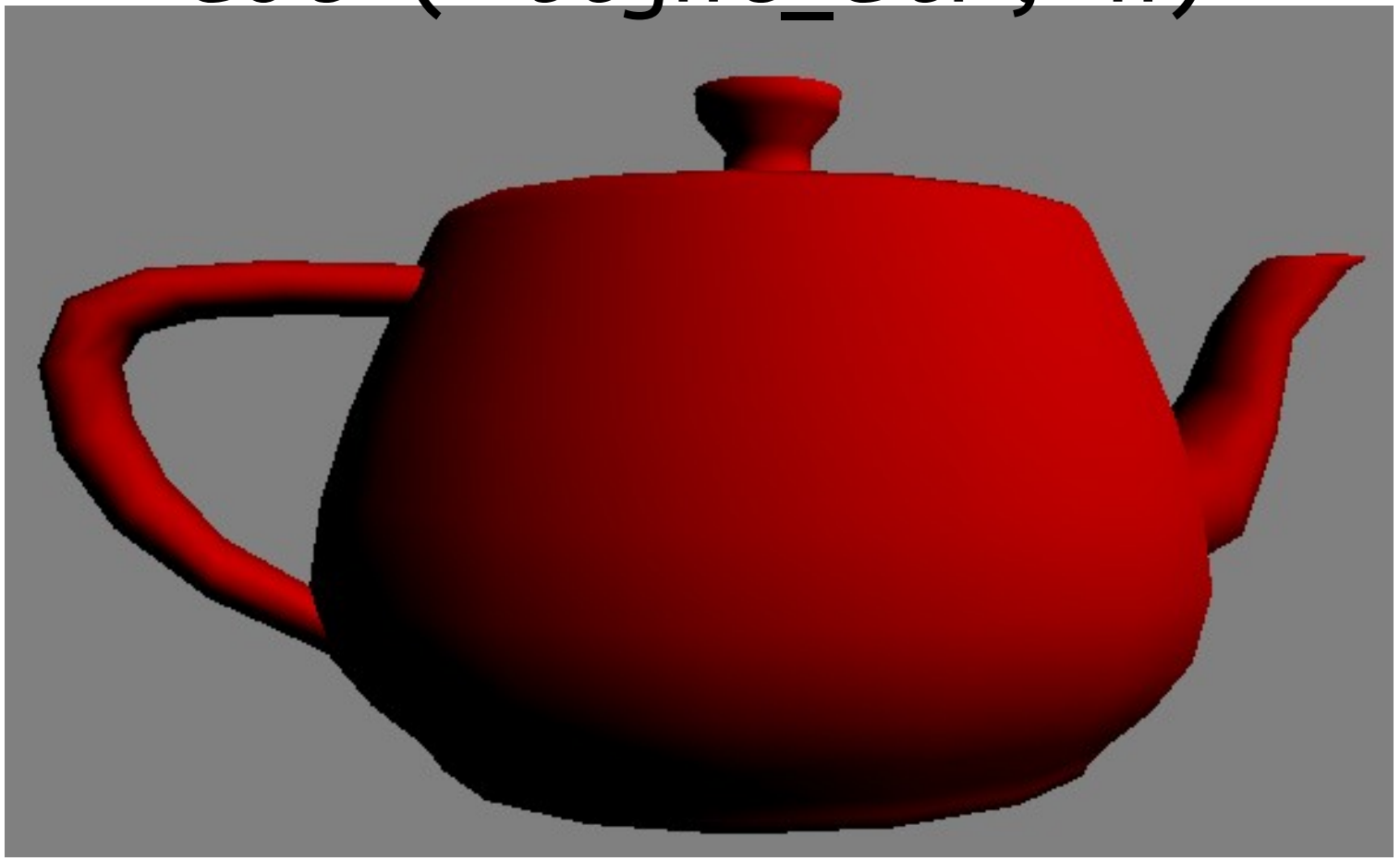
Where $arccos$ is inverse cosine cos^{-1}.



Normalise vectors first!

Diffuse Reflection

```
i_d = l_d * k_d *
dot (-light_dir, n)
```



```
vec3 l_d = vec3 (0.8, 0.8, 0.8);
vec3 k_d = vec3 (1.0, 0.0, 0.0);
```

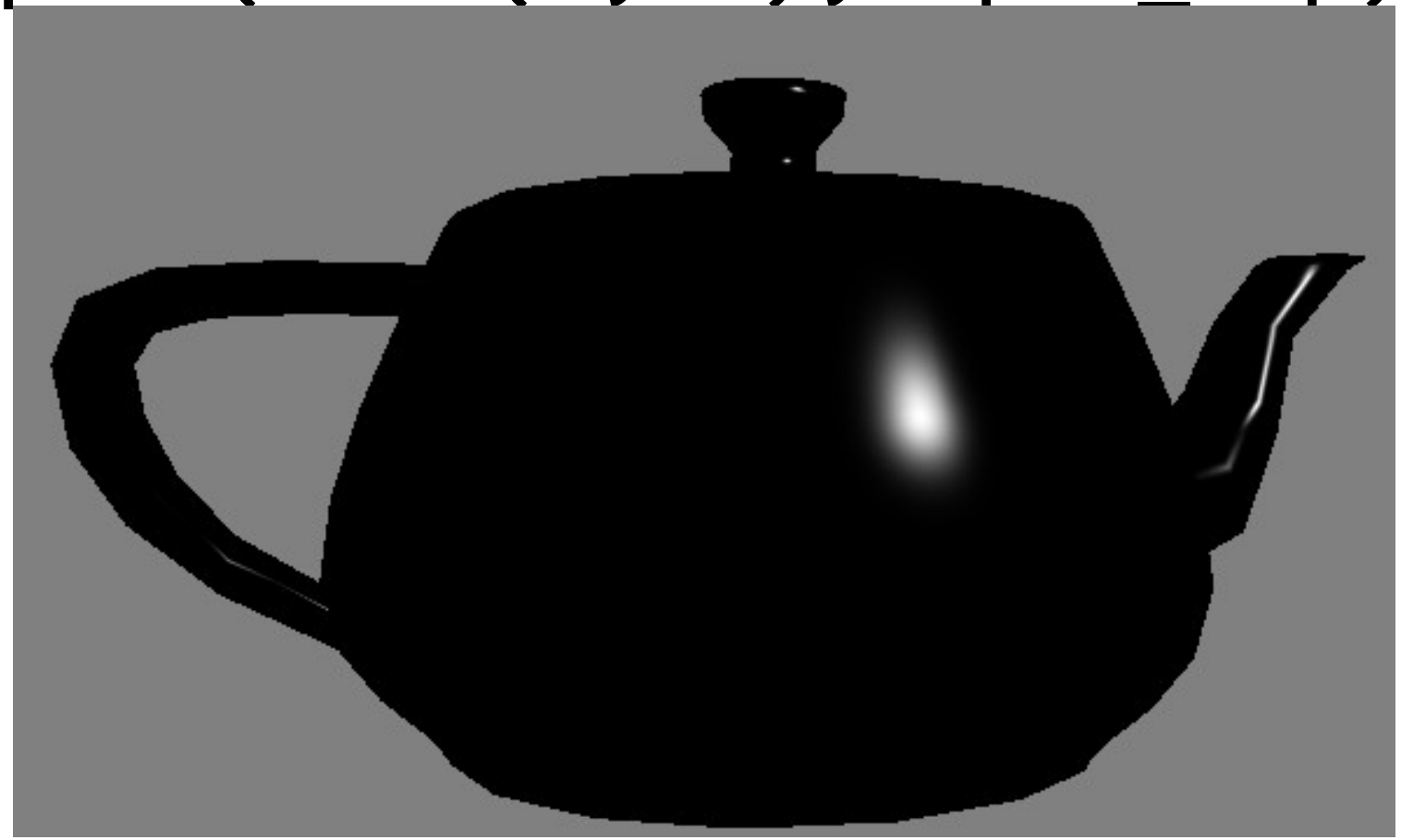
Make sure d.prod >= 0.0

Specular Reflection Model

- Approximate light that hits surface and entirely reflects in one direction, around the normal (like a billiard ball)
- Smoother surfaces = **shinier** = more specular reflection
- Intensity is 1.0 when reflecting directly into eye
- Intensity is 0.0 when reflection is perpendicular to eye

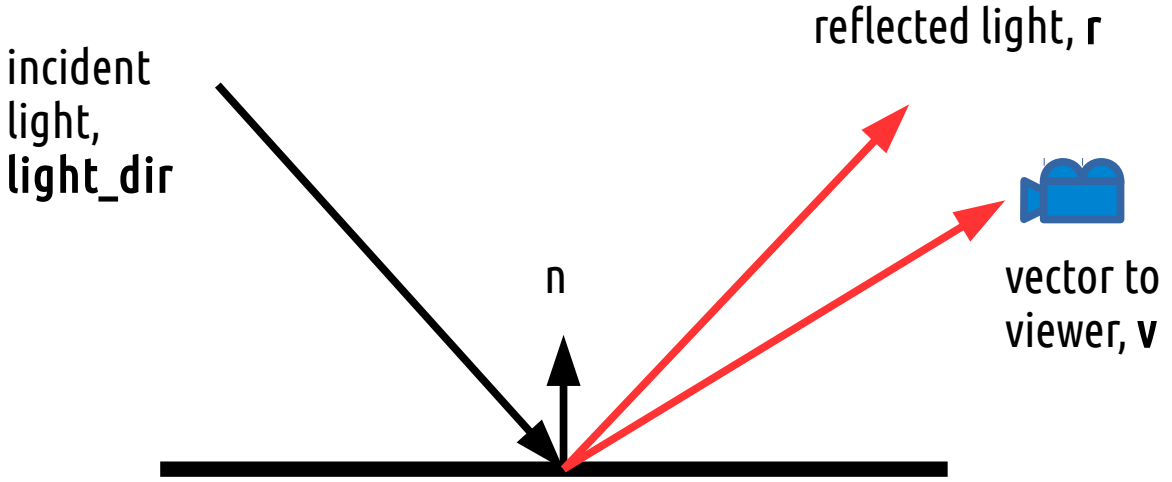
Specular Reflection

```
i_s = l_s * k_s *
pow (dot (r, v), spec_exp)
```



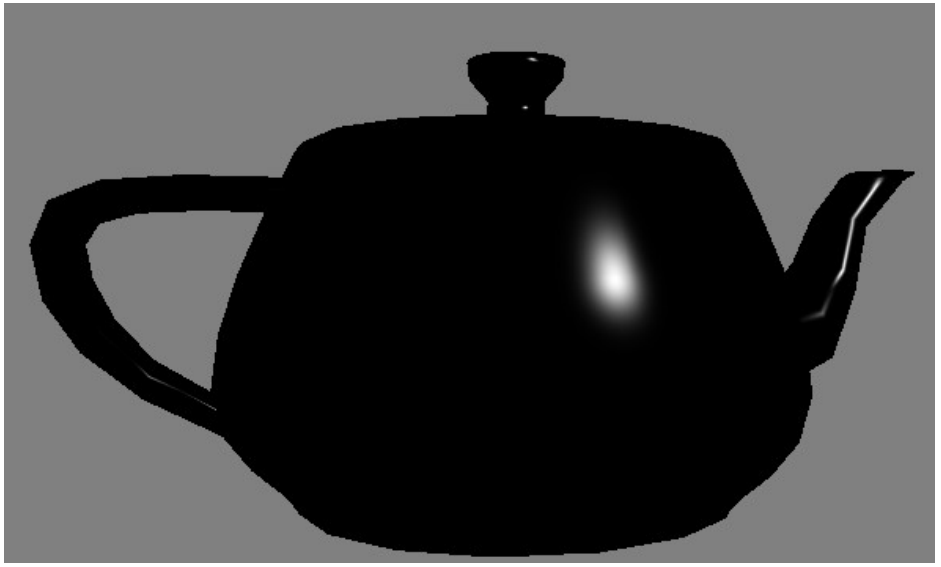
```
vec3 l_s = vec3 (1.0, 1.0, 1.0);
vec3 k_s = vec3 (1.0, 1.0, 1.0);
```

Specular Reflection

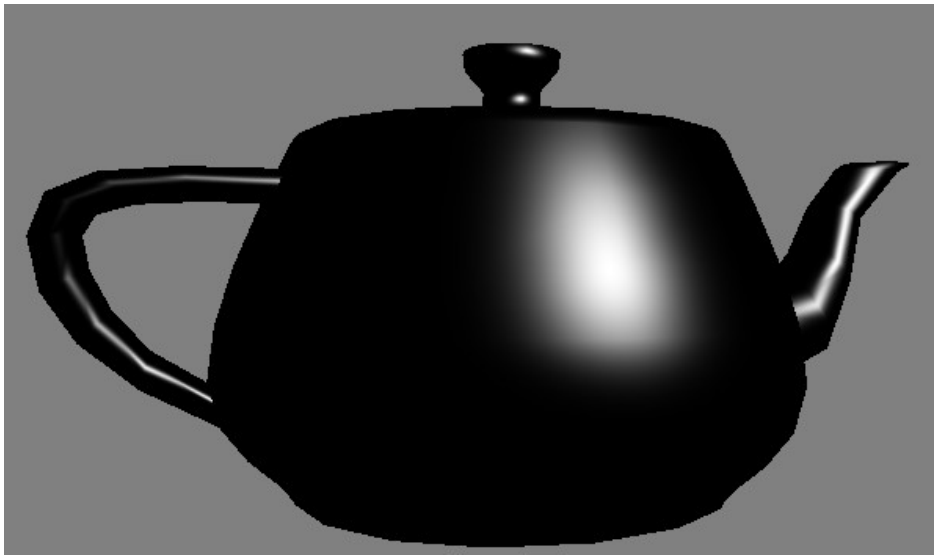


- Work out **r** and **v**, dot product of them to get factor
- When reflected light points right into eye = full specular
- There is also an exponent which we can adjust

Specular Highlights



Exponent = 100



Exponent = 10

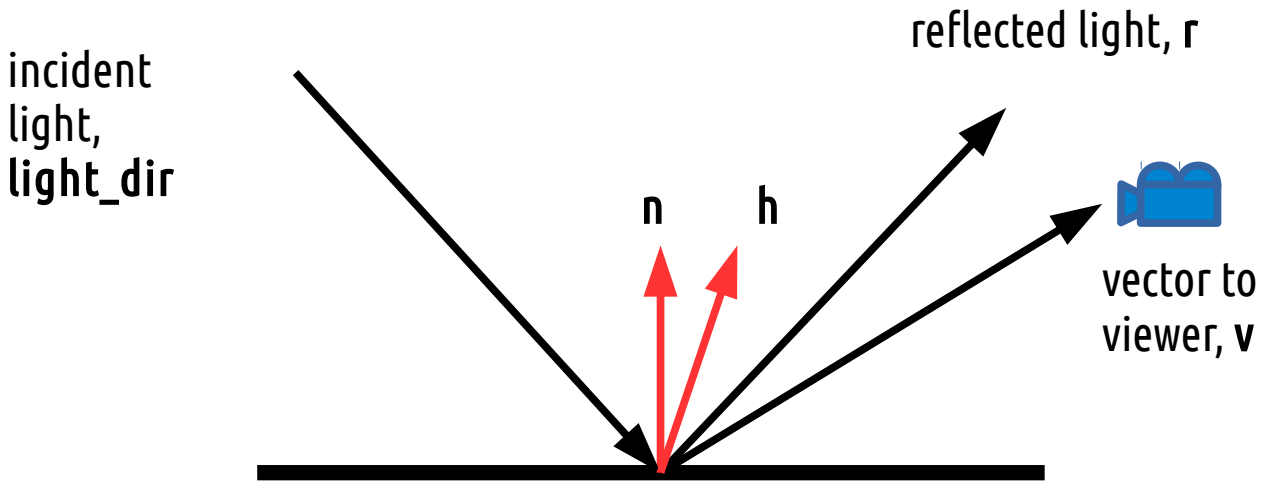
Phong Lighting is the Sum

$$i = i_a + i_d + i_s$$

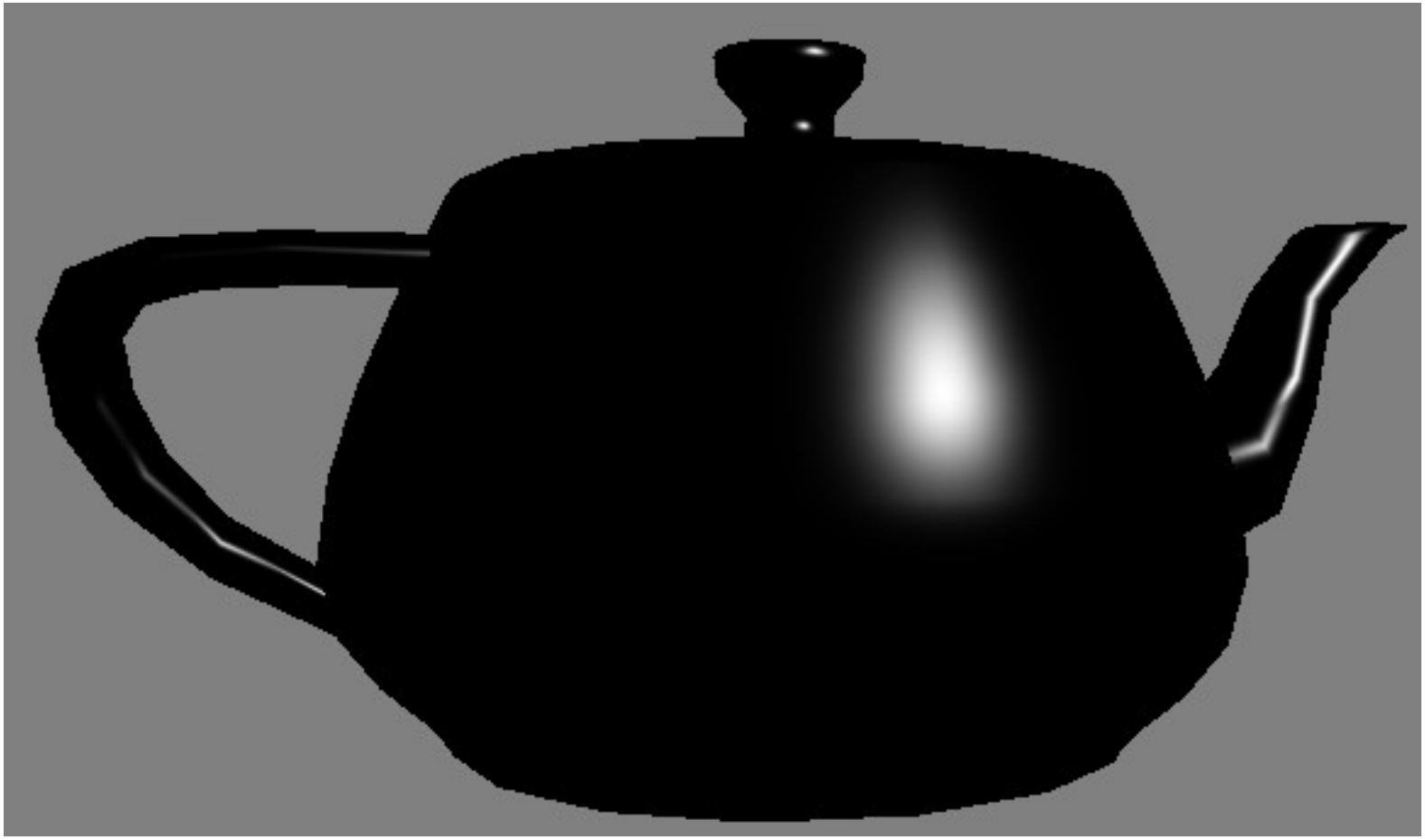
Blinn-Phong

- Lose a small amount of accuracy in specular equation
- Little bit cheaper to calculate
- Replace reflect() with a **half-way vector**:

```
vec3 h = normalize (v - light_dir);
vec3 I_s = l_s * k_s * pow (dot (h, n), spec_exp);
```



Blinn-Phong



- Reduces specular power by about half –> double the exponent

[Somewhat rhetorical] Questions

- Q. How can we model a non-shiny surface?
- Q. Is any real surface completely matte?
- Q. What is physically inaccurate about Phong lighting?
- Q. Do any real surfaces have a non-white specular colour?
- Q. What is missing from this lighting model to make it convincing?

Pause

- **If this is the last slide:**
- Do a Phong tutorial
- Read any of the textbook chaps.
  - Phong lighting
  - Shading models
  - More advanced / general lighting
  - Global versus local illumination

Materials and Textures

- Q. How can we combine textures with Phong lighting?

Warning: Corrupted Normals

- **Q. Why should we never apply a un-equal scaling to a normal?**
- **Q. How can we avoid this?**
  - Create a separate model matrix with just the rotations “normal matrix” **or**
  - Take `inverse (transpose (model_matrix)` instead **or**
  - Don't do lighting on things with uneven scaling **or**
  - Don't ever do uneven scaling

Warning: Negative dot products

- Sometimes a dot-product produces a negative number
- **Q. When does this happen?**
- **Q. What unwanted visual effect would a negative dot-product give us?**
- To avoid this:
  - `float result = max (0.0, dot (a, b));`

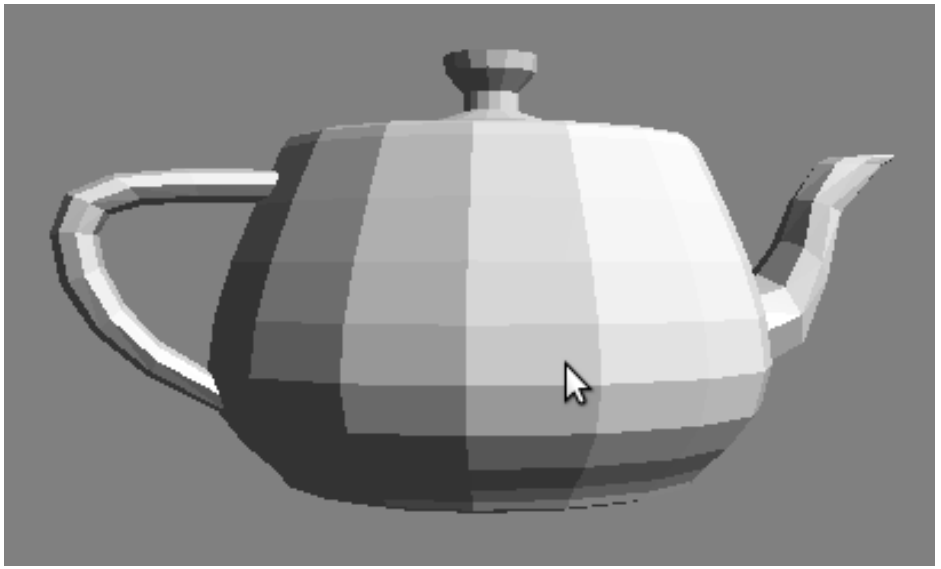
Point / Directional / Spot Lights

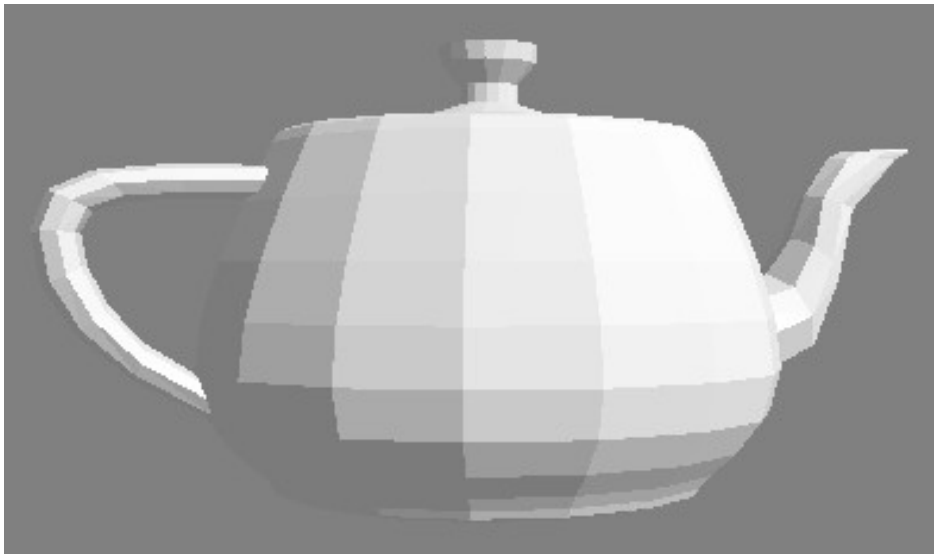
- Were built-in
- Spotlight (Webb?)

Gamma Correction

- Colours and voltages
- Don't correct textures
- sRGB colour palette

Gamma Correction





```
colour = pow (colour, vec3 (2.2, 2.2, 2.2));
```

Gives you the full range of colour intensities on your display

Questions

- Q. More than one light? How?
- Q. Problems with that?

Further Reading

- **Do a Phong lighting tutorial first**
- **Challenge:** Can you figure out how to roll-off the light with distance?
- Deferred lighting and deferred shading, G-buffers
- BDRF
- Radiosity