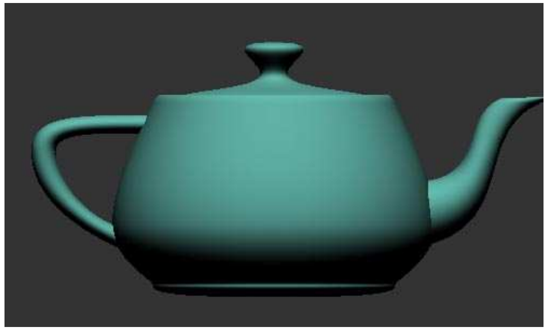
Shader

Lighting

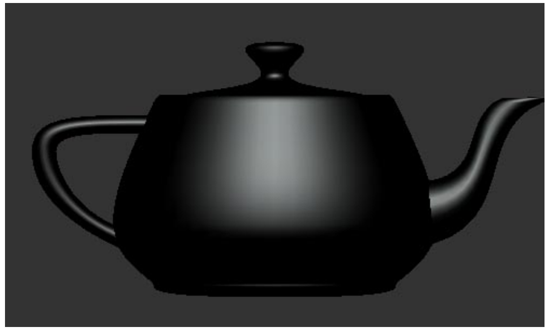
Local lighting model – hack!

- I = ambient + diffuse + specular

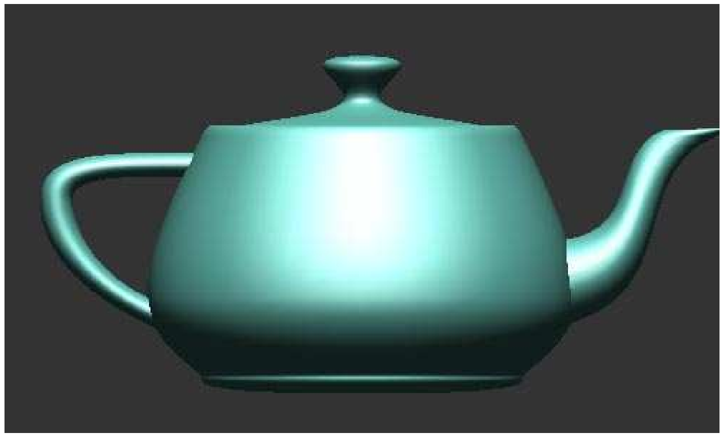
+



+

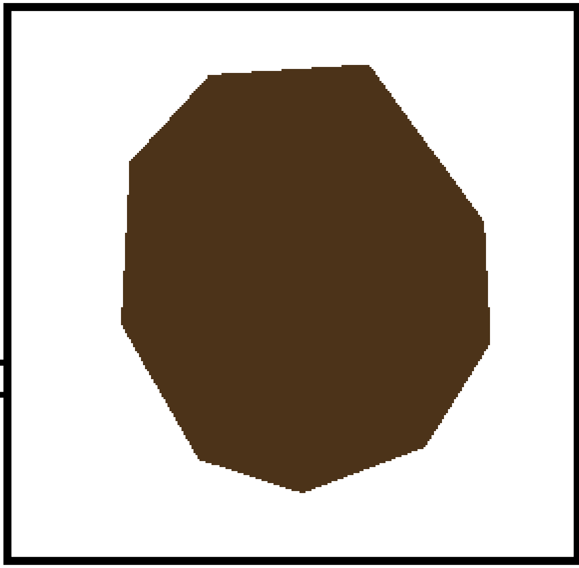


=



Ambient Lighting

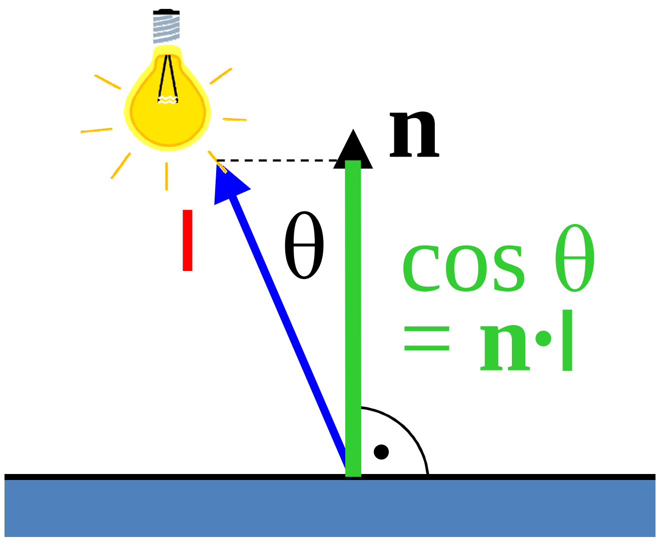
```
// vertex shader
uniform vec3 uAmbient;
varying vec3 vColor;
void main() {
  vColor = uAmbient * color;
  …
```

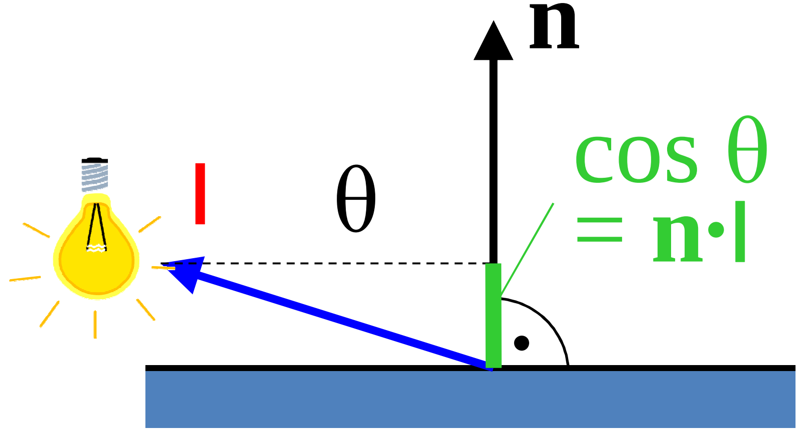


```
// fragment shader
varying vec3 vColor;
void main() {
  gl_FragColor = vec4(vColor, 1.0);
}
```

Lambertian (Diffuse) Reflection

- Lambert's cosine law $L_{diff} = k_d \cdot I \cdot (\mathbf{n} \cdot \mathbf{l})$

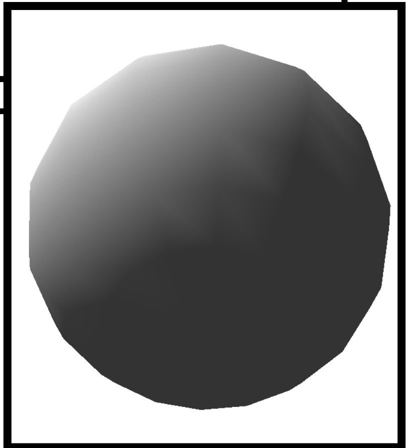




Diffuse lighting (Gouraud)

```
// vertex shader
uniform vec3 uLight;
uniform vec3 uColor;
varying vec3 vColor;
void main() {
  vec3 lightDir = normalize(uLight);
  float diff = max(0.0, dot(normal, lightDir));
  vColor = diff * uColor;
  …
```

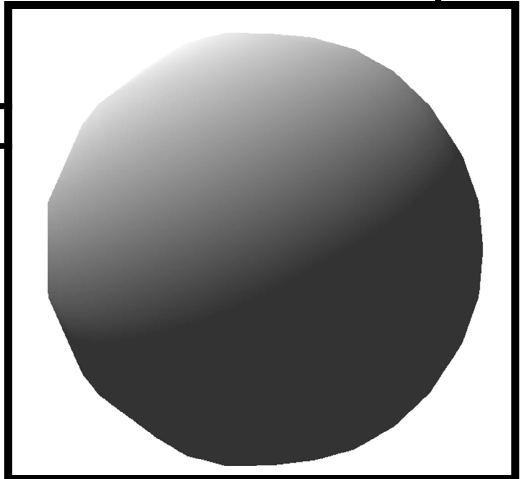
```
// fragment shader
varying vec3 vColor;
void main() {
  gl_FragColor = vec4(vColor, 1.0);
}
```



Diffuse lighting (Phong)

```
// vertex shader
varying vec3 vNormal;
void main() {
  vNormal = normal;
  …
```

```
// fragment shader
varying vec3 vNormal;
uniform vec3 uLight;
uniform vec3 uColor;
void main() {
  vec3 lightDir = normalize(uLight);
  vec3 normal = normalize(vNormal);
  float diff = max(0.0, dot(normal, lightDir));
  ...
```



Spaces

- In what space is
  - `(v)normal`?
  - `uLight`?

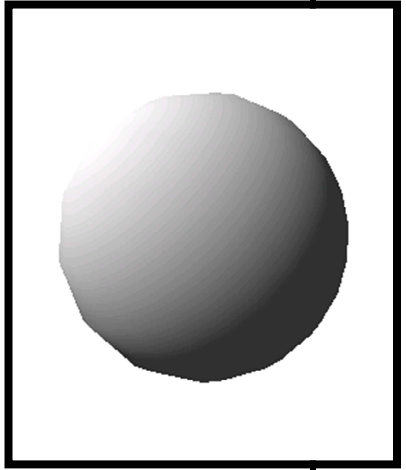
```
// vertex shader
varying vec3 vNormal;
void main() {
  vNormal = normal;
  …
```

```
// fragment shader
varying vec3 vNormal;
uniform vec3 uLight;
uniform vec3 uColor;
void main() {
  vec3 lightDir = normalize(uLight);
  vec3 normal = normalize(vNormal);
  float diff = max(0.0, dot(normal, lightDir));
  ...
```

Diffuse lighting with correct spaces

```
// vertex shader
uniform vec3 uLight;
varying vec3 vNormal, vLight;
void main() {
  vNormal = normalMatrix * normal;
  vLight = (viewMatrix * vec4(uLight, 0.0)).xyz;
  …
```

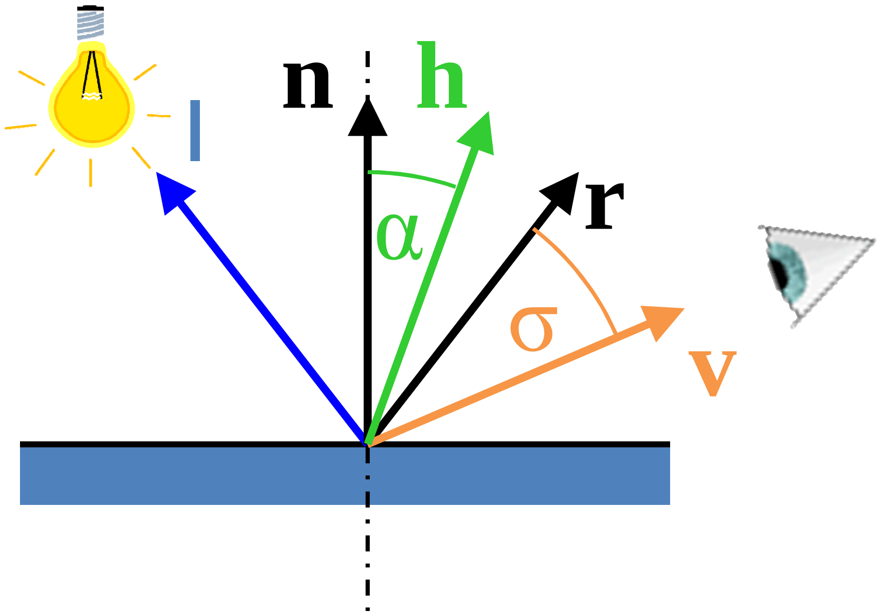
```
// fragment shader
varying vec3 vNormal, vLight;
void main() {
  vec3 n = normalize(vNormal);
  vec3 l = normalize(vLight);
  float diff = max(0.0, dot(n, l));
  ...
```



Blinn-Phong

- Halfway vector $\mathbf{h}$

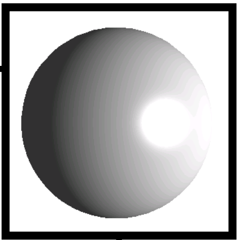
$$L_{spec} = k_s \cdot I \cdot (\mathbf{v} \cdot \mathbf{r})^p \quad \rightarrow \quad L_{spec} = k_s \cdot I \cdot (\mathbf{n} \cdot \mathbf{h})^p$$



$$\mathbf{h} = \frac{\mathbf{l} + \mathbf{v}}{\| \mathbf{l} + \mathbf{v} \|}$$

Specular lighting

- Now using point light (position in view space)
- Need vector to light and vector to viewer

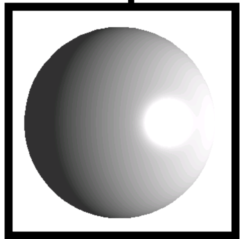


```
// vertex shader
uniform vec3 uLightPosView;
varying vec3 vNormal, vLight, vViewDirection;
void main() {
  vec4 posV = modelViewMatrix
    * vec4(position, 1.0);
  vLight = normalize(uLightPosView - vec3(posV));
  vViewDirection = - vec3(posV);
  …
```

Specular lighting (Blinn-Phong)

- Half angle calculation and specular power

```
// fragment shader
uniform vec3 uDiffCol, uSpecCol;
varying vec3 vNormal, vLight, vViewDirection;
void main() {
  … // diffuse
  vec3 v = normalize(vViewDirection);
  vec3 h = normalize(l + v);
  float NdH = max(0.0, dot(n, h));
  float spec = pow(NdH, 64.0);
  vec3 color = diff * uDiffCol + spec * uSpecCol;
  …
```



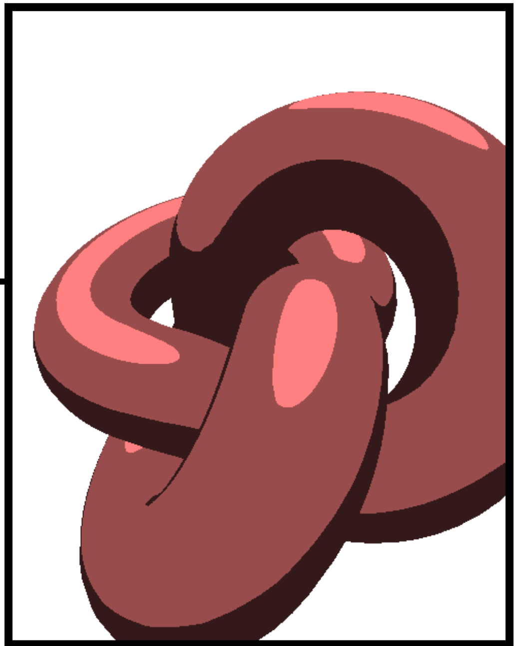
Gooch

- Blend between a cool and a warm color

```
// fragment shader
varying vec3 vNormal, vLight, vViewDirection;
void main() {
  … // diffuse + specular
   vec3 cool = vec3(0.88, 0.81, 0.49);
   vec3 warm = vec3(0.58, 0.10, 0.76);
   vec3 gooch = mix(warm, cool, diff);
  vec3 color = gooch + spec * uSpecColor;
  …
```

Toon shading

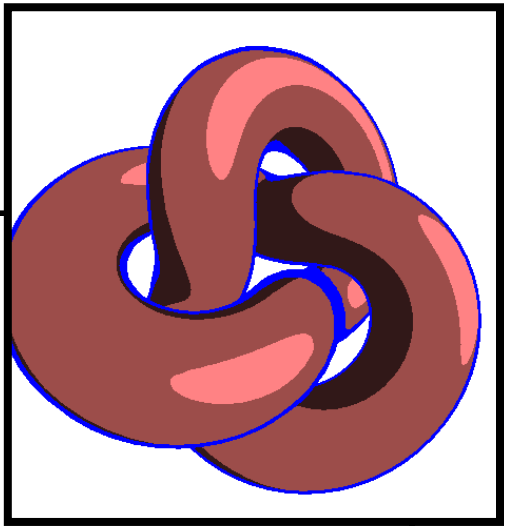
- Discrete color steps for diffuse



```
// fragment shader
…
void main() {
  … // diffuse
  vec3 color = (diff > 0.95) ?
    vec3(1.0, 0.5, 0.5) :
      (diff > 0.5) ? vec3(0.6, 0.3, 0.3) :
        vec3(0.2, 0.1, 0.1);
  …
```

Cell shading

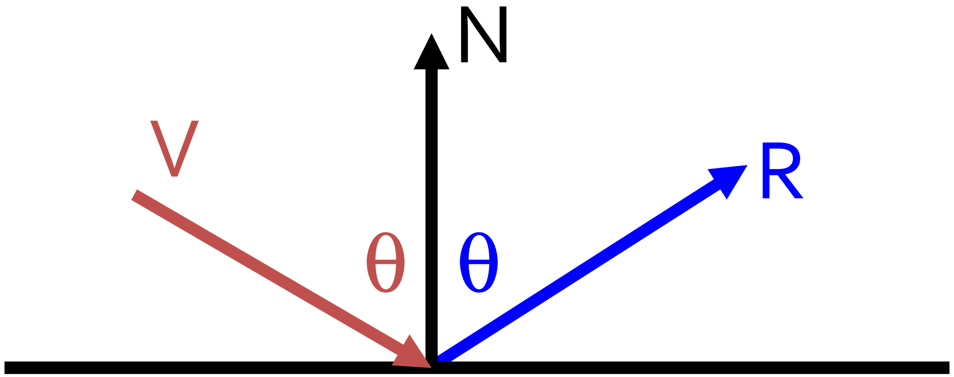
- Detect edges and color them



```
// fragment shader
…
void main() {
  vec3 n = normalize(vNormal);
  vec3 v = normalize(vViewDirection);
  if(abs(dot(n, v)) < 0.3) {
    gl_FragColor = vec4(vec3(0.0), 1.0);
    return;
  }
  …
```

Reflective Environment Mapping

- Angle of incidence = angle of reflection



R = V - 2 (N dot V) N
= `reflect(V,N)`

V and N normalized!

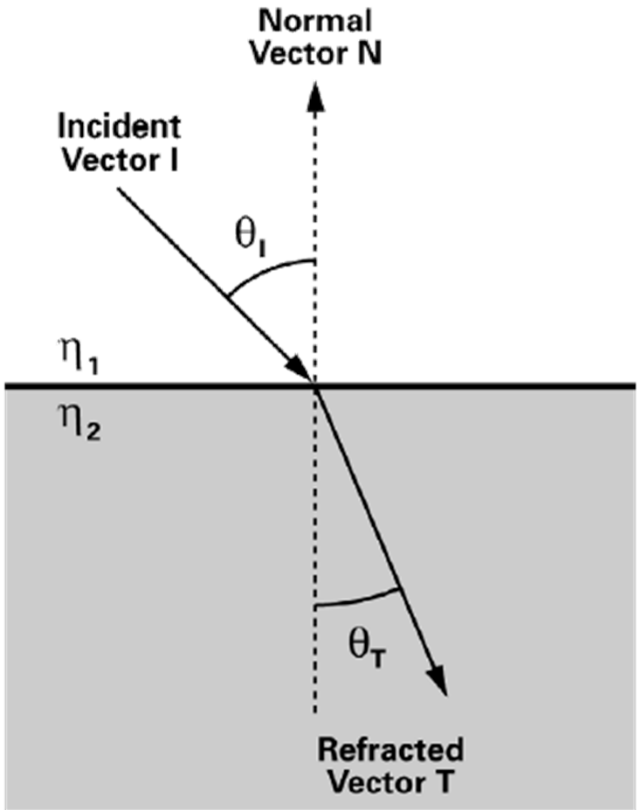
V is incident vector!

- Cube map needs reflection vector in coordinates (where map was created)

Refractive Environment Mapping

- Use refracted vector for lookup:
  - Snells law:

$$\eta_1 \sin\theta_I = \eta_2 \sin\theta_T$$



Specular Environment Mapping

- We can pre-filter the environment map
  - Equals specular integration over the hemisphere
  - Phong lobe (cos^n) as filter kernel
  - **`textureLod`**; **`level`** according to glossiness
  - R as lookup