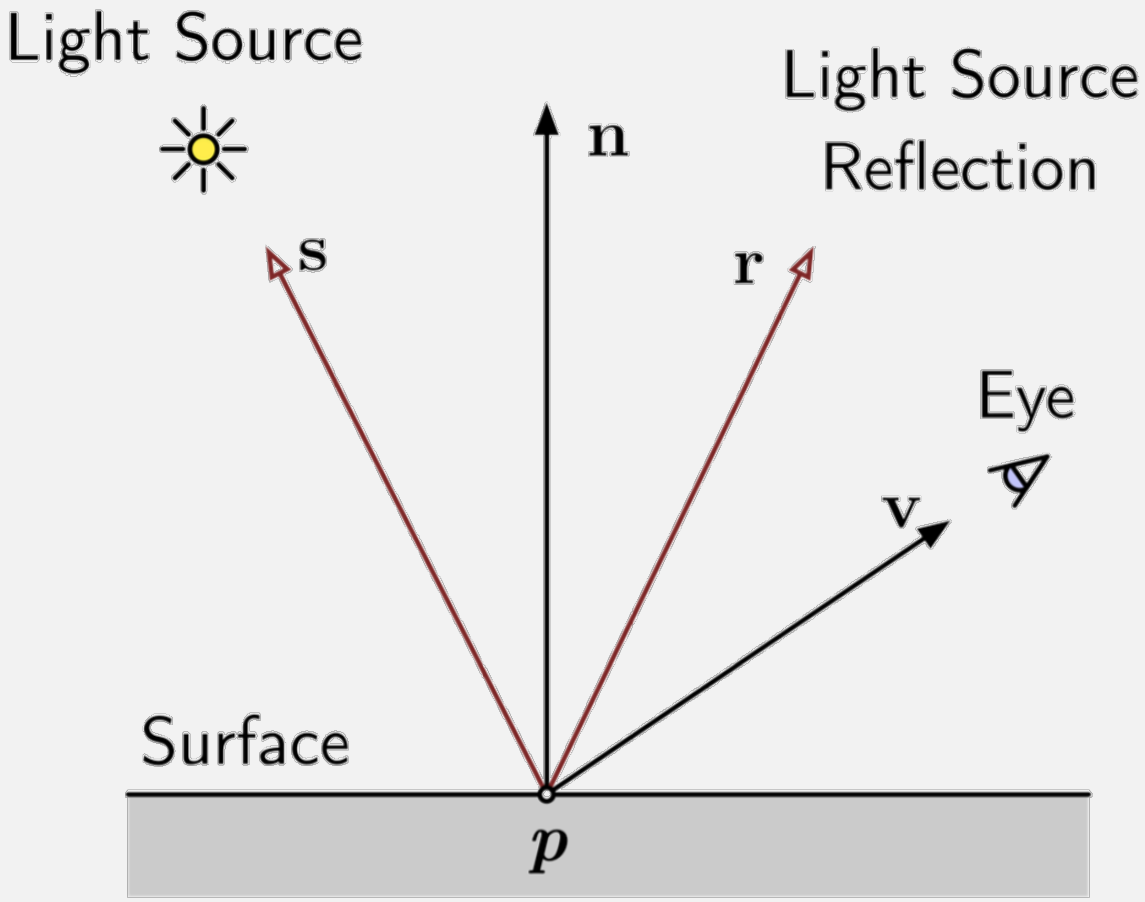
BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Modul: *Real Time Rendering (RTR)* – Phong Shading Refresher –

Master Medieninformatik

Prof. Dr.-Ing. Hartmut Schirmacher

Remember Illumination Calculations ?



Remember Dot Product (*Skalarprodukt*) ?

Algebraic Definition

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} := x_1 x_2 + y_1 y_2 + z_1 z_2$$

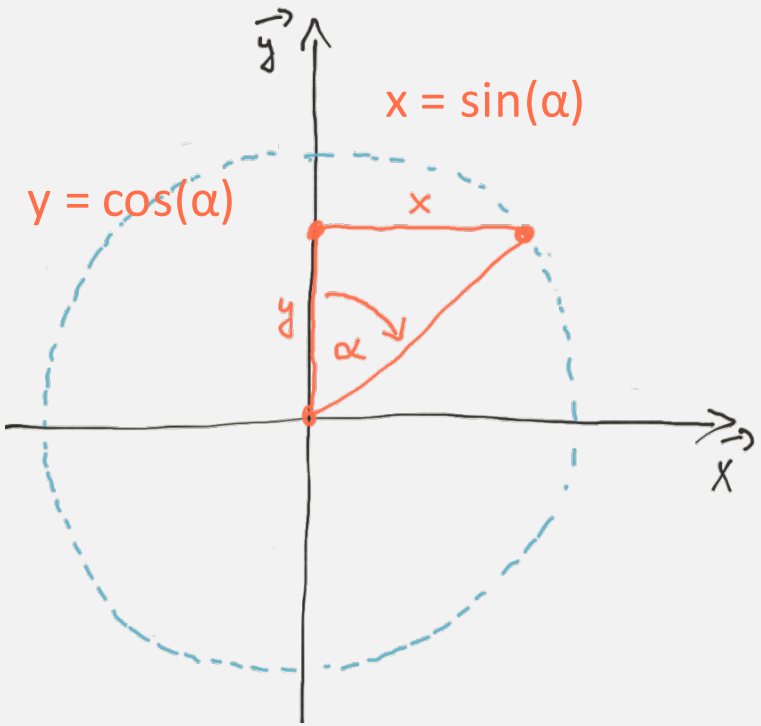
$\vec{x}_1$ $\vec{x}_2$

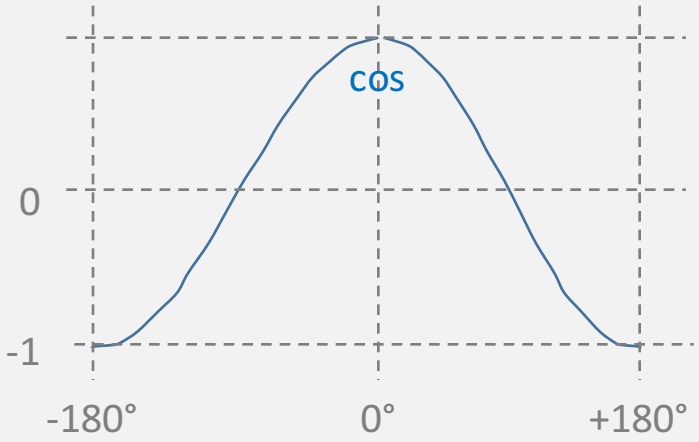
$$\vec{x}_1 \cdot \vec{x}_2 = |\vec{x}_1| \cdot |\vec{x}_2| \cdot \cos \measuredangle(\vec{x}_1, \vec{x}_2)$$

Geometric Meaning:
cosine of enclosed angle

https://en.wikipedia.org/wiki/Dot_product

Remember Cosine ?





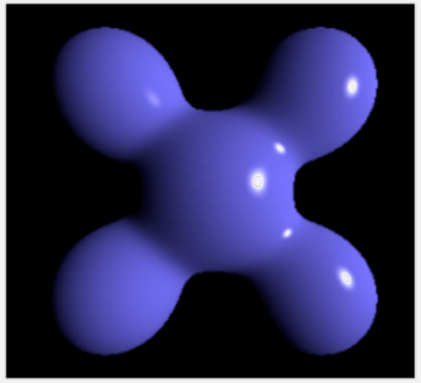
Unit circle

- cos(0°) = 1
- cos(90°) = 0
- cos(-90°) = -1
- cos(180°) = 0

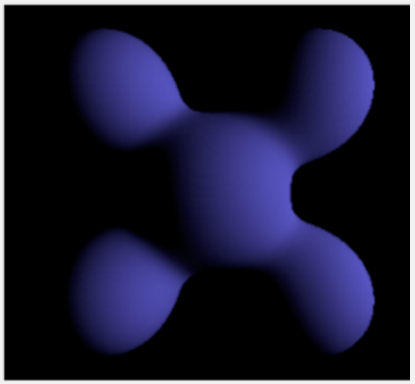
cos is max when vectors point in same direction!

- BTW: cos(45°) = 0,707

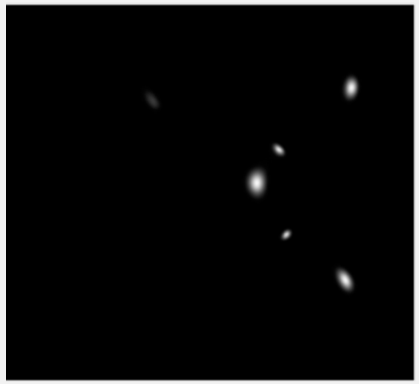
Phong Illumination Model



+



+



$$L_r = k_a L_a + k_d \sum_j L_j(\vec{n} \cdot \vec{s}_j) + k_s \sum_j L_j(\vec{r}_j \cdot \vec{v})^{k_e}$$

ambient term

diffuse term

specular term

https://en.wikipedia.org/wiki/Phong_reflection_model

Phong as GLSL Function

```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, vec3 lightDir)
{

    vec3 ambient = k_ambient * ambientLightColor;

    float ndotv = dot(normalDir,viewDir);
    if(ndotv<0)
        discard;

    float ndotl = dot(normalDir,-lightDir);
    if(ndotl<0.0)
        return ambient;

    vec3 diffuse = k_diffuse * lightColor * ndotl;

    vec3 r = reflect(lightDir,normalDir);
    float rdotv = max( dot(r,viewDir), 0.0);
    vec3 specular = k_specular * lightColor * pow(rdotv, shininess);

    return ambient + diffuse + specular;

}
```

Fragment Shader Code

Phong: Ambient Term

```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, vec3 lightDir)
{

    vec3 ambient = k_ambient * ambientLightColor;

    float ndotv = dot(normalDir,viewDir);
    if(ndotv<0)
        discard;

    float ndotl = dot(normalDir,-li
    if(ndotl<0.0)
        return ambien

    vec3 diffuse = k_diffuse * lightColor * ndotl;

    vec3 r = reflect(lightDir,normalDir);
    float rdotv = max( dot(r,viewDir), 0.0);
    vec3 specular = k_specular * lightColor * pow(rdotv, shininess);

    return ambient + diffuse + specular;

}
```

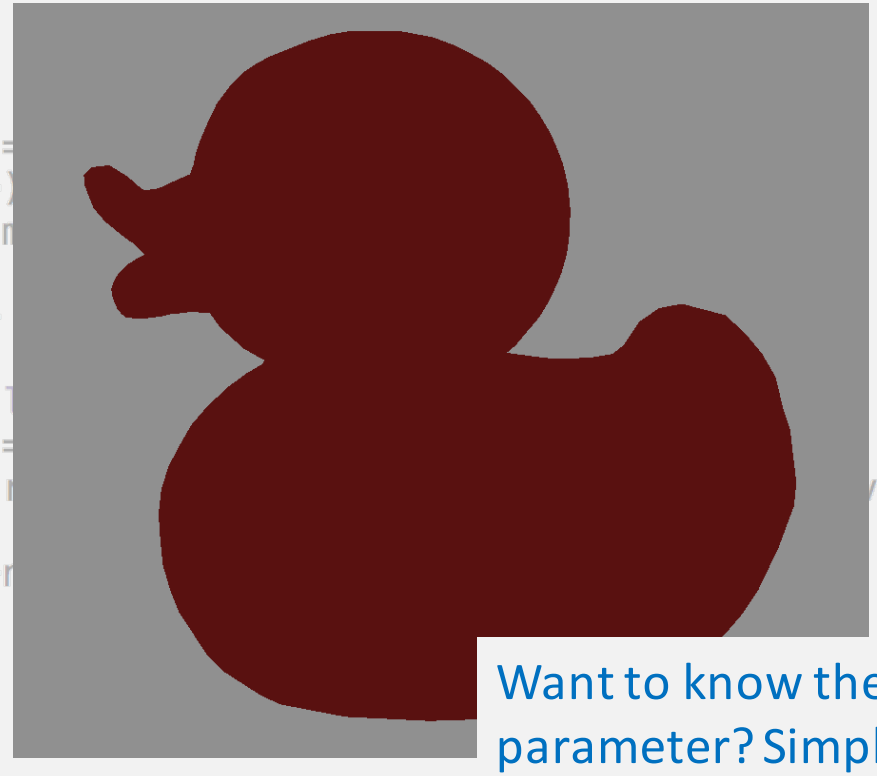
Use light color to change appearance of all objects at once

Use coefficient to change just one shape / object.

Fragment Shader Code

Phong: Ambient Term

```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, vec3 lightDir)
{

    vec3 ambient = k_ambient * ambientLightColor;

    return ambient;

    if(ndotv<0)
        discard;

    float ndotl =
    if(ndotl<0.0)
        return am

    vec3 diffuse

    vec3 r = refl
    float rdotv =
    vec3 specular                                   , shininess);

    return ambien

}
```



Want to know the effect of the ambient parameter? Simply change the return value of your Phong function.

n dot v

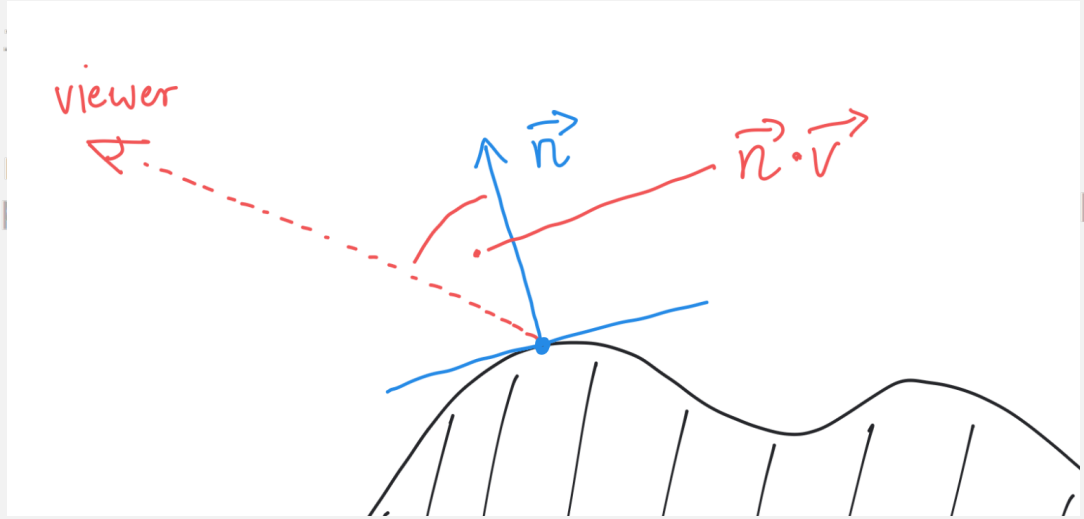
```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, vec3 lightDir)
{

    vec3 ambient = k_ambient * ambientLightColor;

    float ndotv = dot(normalDir,viewDir);
    if(ndotv<0)
        discard;

    float
    if(nd
        re
    vec3

    vec3 r = reflect(lightDir,normalDir);
    float rdotv = max( dot(r,viewDir), 0.0);
    vec3 specular = k_specular * lightColor * pow(rdotv, shininess);

    return ambient + diffuse + specular;

}
```

`discard`: throw this fragment away, instead of rendering it. Special instruction, only available in fragment shader.

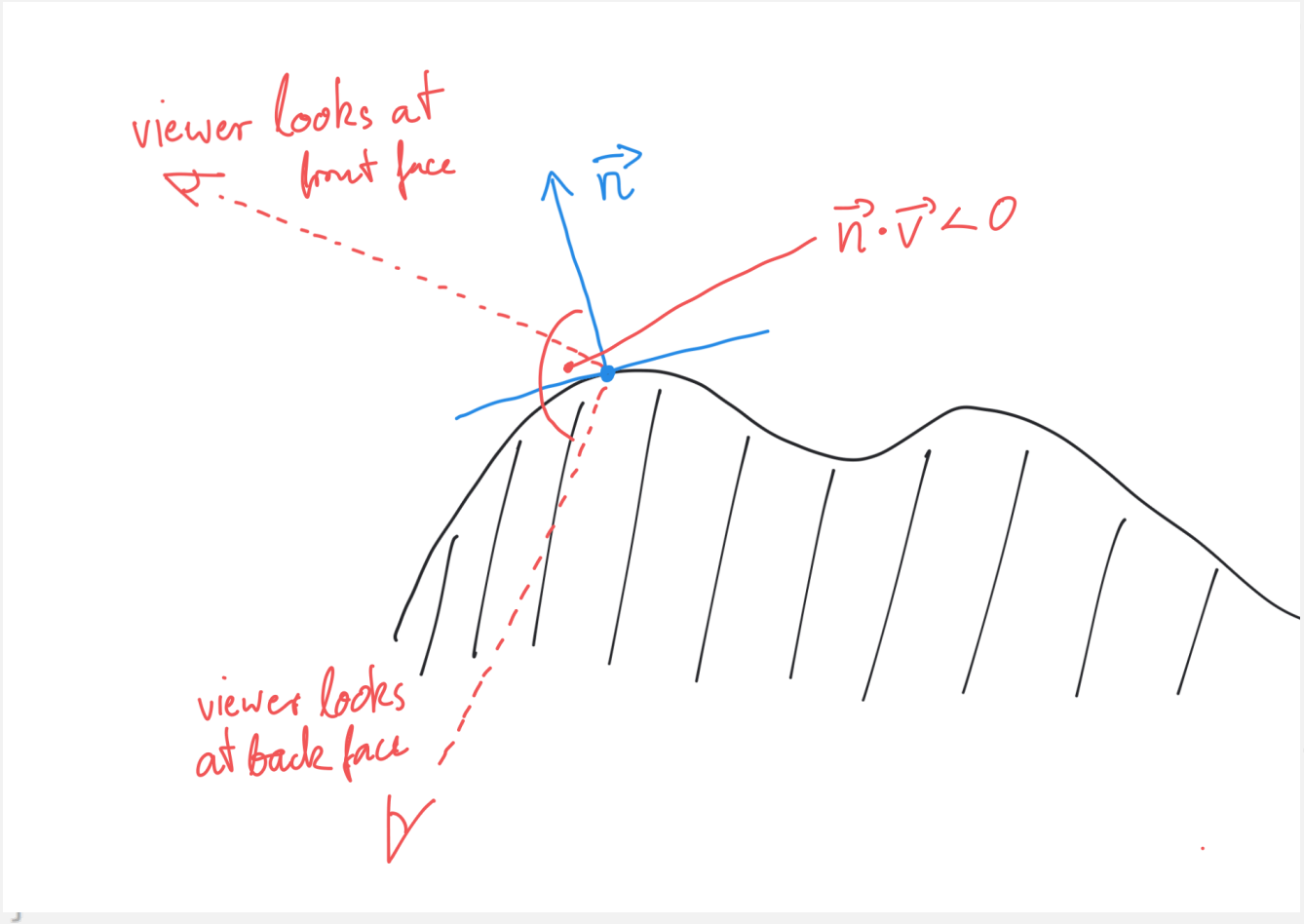
n dot v

```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, vec3 lightDir)
{

    vec3 ambient = k_ambient * ambientLightColor;

    float ndotv = dot(normalDir,viewDir);
    if(ndotv<0)
        discard;

    float
    if(ndo
        re

    vec3 d

    vec3 r
    float
    vec3 sp                                              hininess);

    return

}
```

This is often omitted in Phong shaders (which is OK).
What does this do?



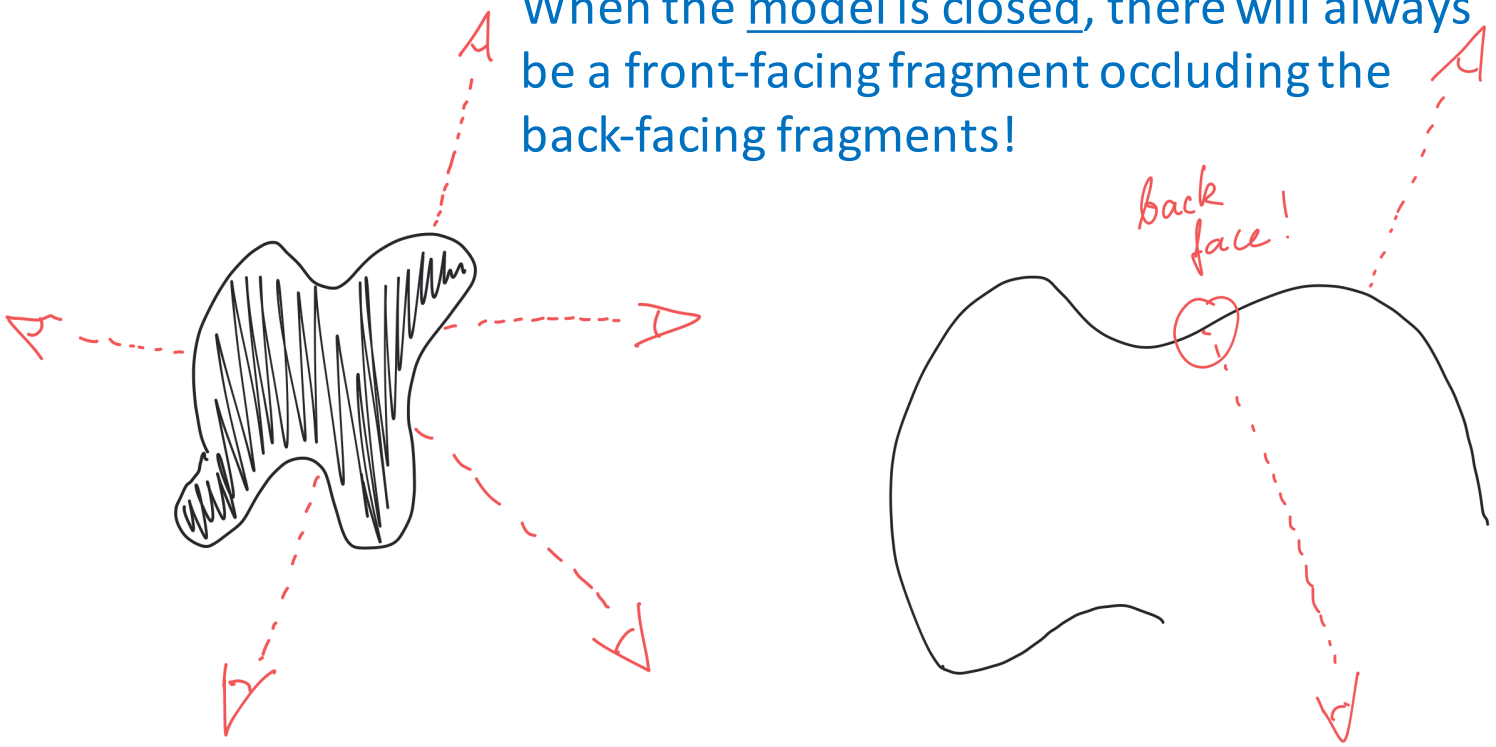
`n dot v < 0` : Primitive is *Back Facing*



`n dot v < 0` : Where does this Happen?

```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, vec3 lightDir)
{

    vec3 ambient = k_ambient * ambientLightColor;
```

When the model is closed, there will always* be a front-facing fragment occluding the back-facing fragments!

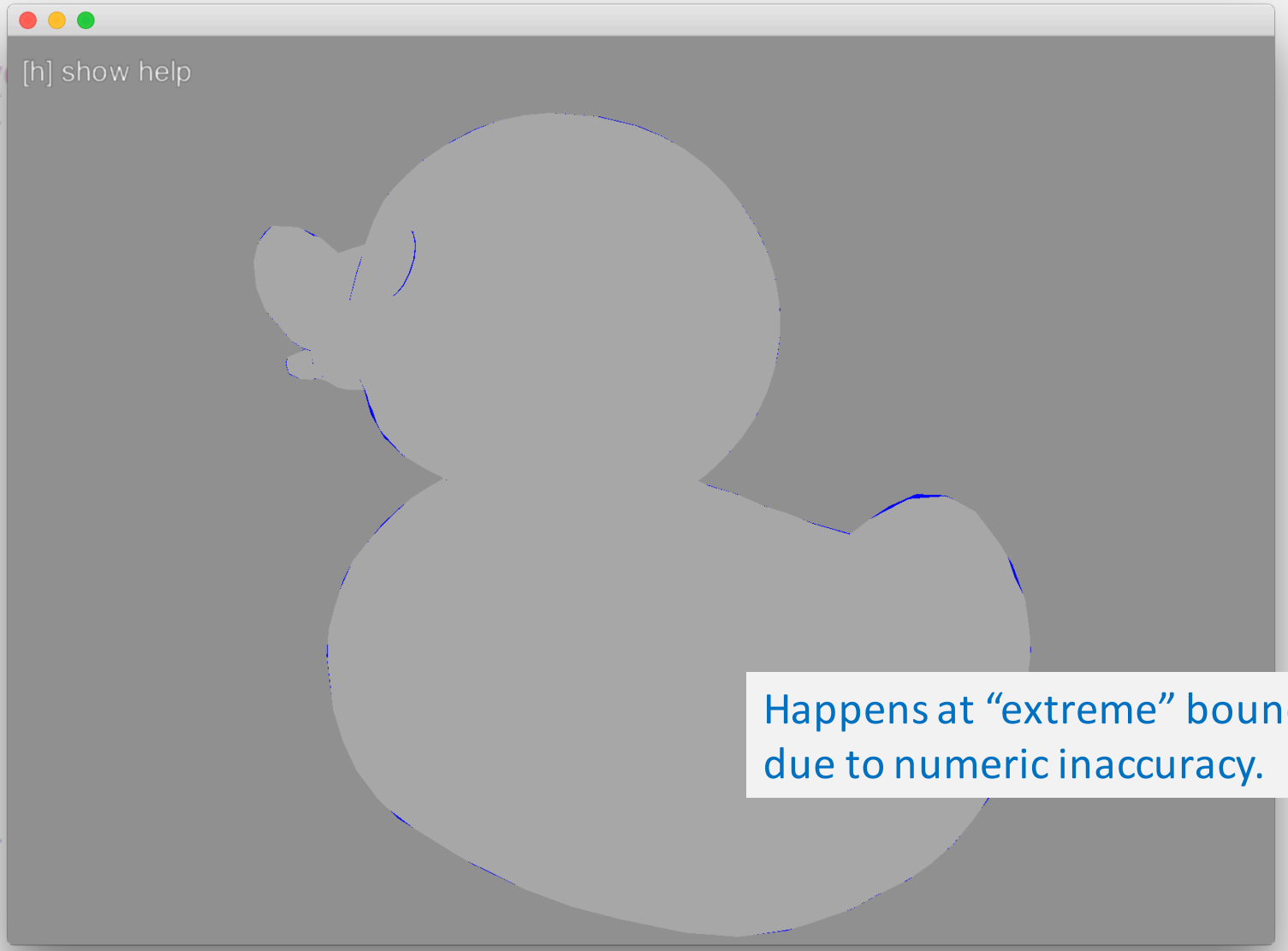


*) except if the camera is inside the model

n dot v < 0 : Let's Check!

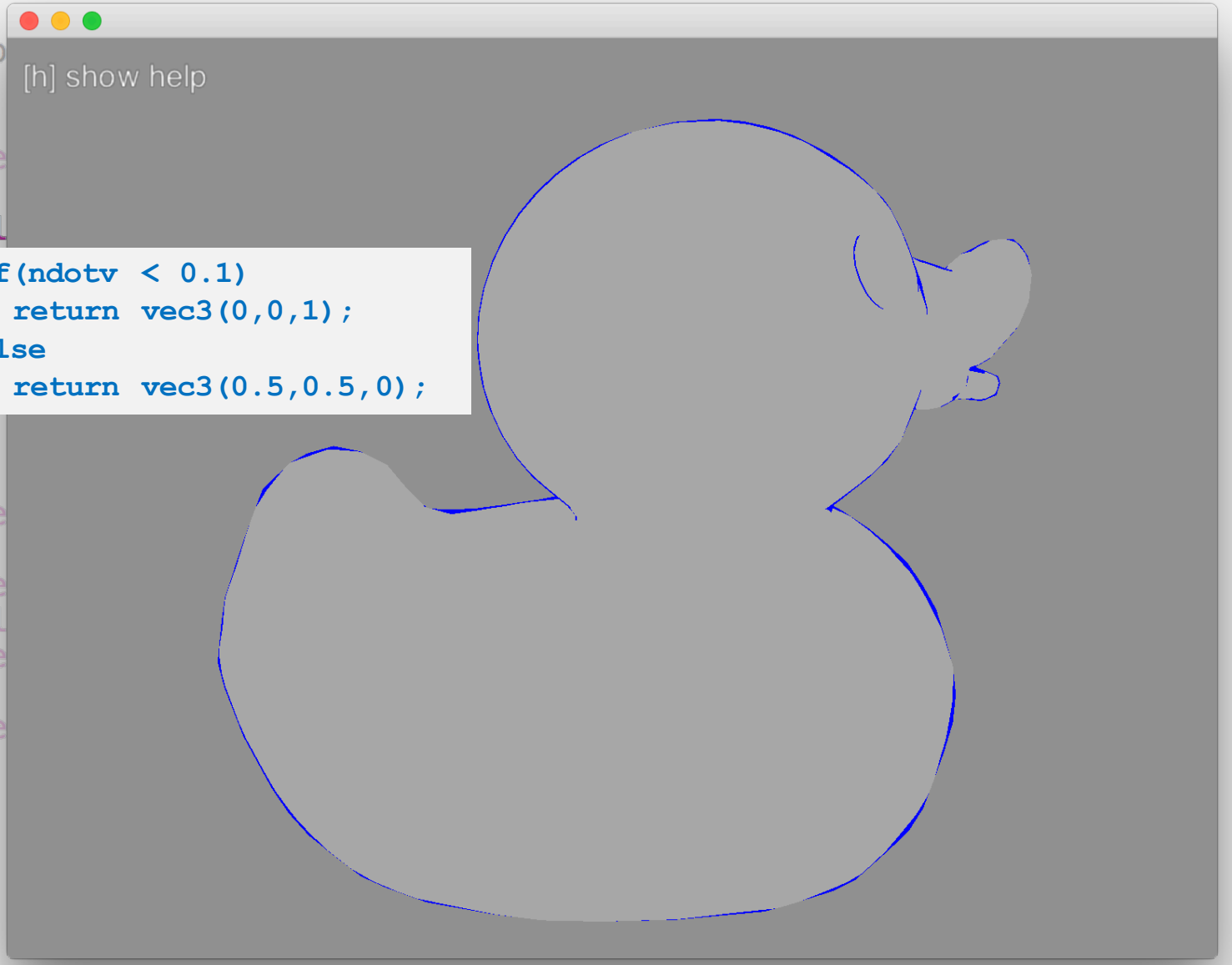
```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, vec3 lightDir)
{

    vec3 ambient = k_ambient * ambientLightColor;

    float ndotv = dot(normalDir,viewDir);
    if(ndotv<0)
       return vec3(0,0,1);
    else
       return vec3(0.5,0.5,0);

    vec3 diffuse = k_diffuse * lightColor * ndotl;

    vec3 r = reflect(lightDir,normalDir);
    float rdotv = max( dot(r,viewDir), 0.0);
    vec3 specular = k_specular * lightColor * pow(rdotv, shininess);

    return ambient + diffuse + specular;

}
```

`n dot v < 0` : Let's Check!



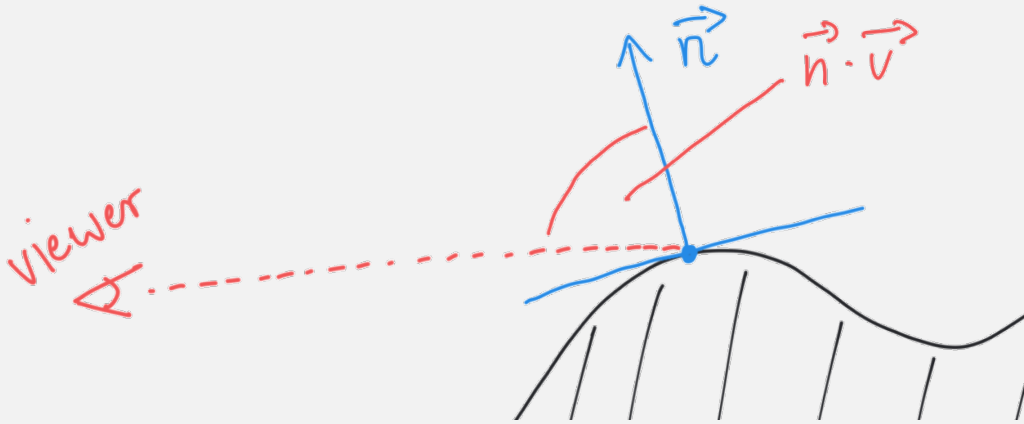
Happens at "extreme" boundaries, due to numeric inaccuracy.

n dot v < 0.1 : Grazing Angles



`n dot v < 0.1` : What does 0.1 Mean?

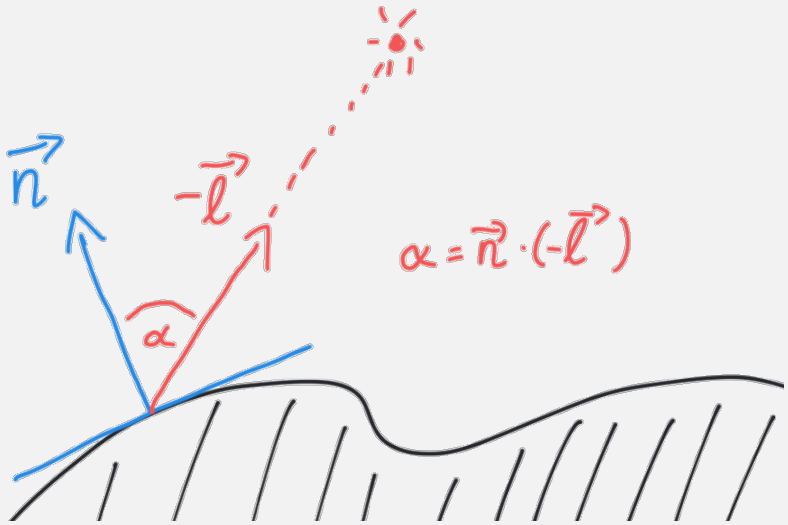
- `dot(n,v) == 0.1`
  - n and v are unit length
  - dot(n,v) = cosine of the angle spanned by n and v



- `acos(0.1) == 1,47...`
  - This does not mean 1,5 °
  - trigonometric functions in GLSL use unit radians (Bogenmaß)

- `degrees(acos(0,1)) == 84,26 °`
  - So this grazing angle is at 84 degrees, measured from the normal (!)

Phong: Diffuse Term

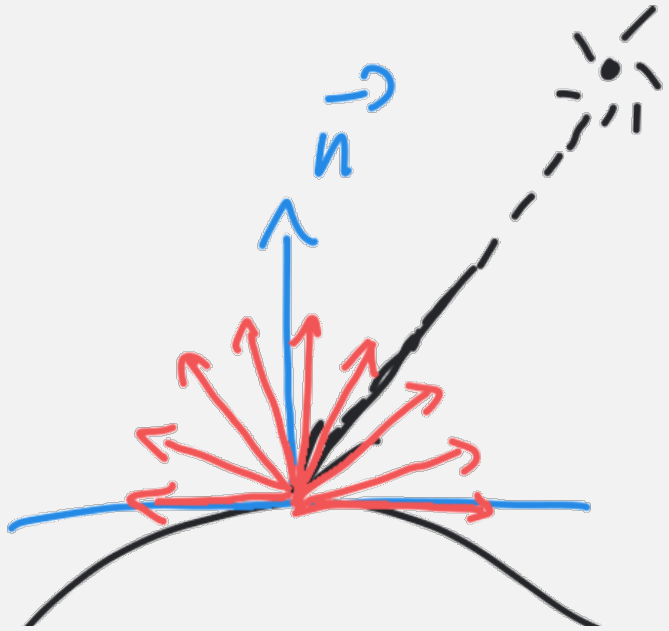



```
vec3 phongIllum(vec3 normalDir, vec
{

    vec3 ambient = k_ambient * ambi
    if(ndotv<0)
        discard;

    float ndotl = dot(normalDir,-lightDir);
    if(ndotl<0.0)
        return ambient;

    vec3 diffuse = k_diffuse * lightColor * ndotl;

    vec3 r = reflect(lightDir,normalDir);
    float rdotv = max( dot(r,viewDir), 0.0);
    vec3 specular = k_specular * lightColor * pow(rdotv, shininess);

    return ambient + diffuse + specular;

}
```

(cosine of) incident light angle

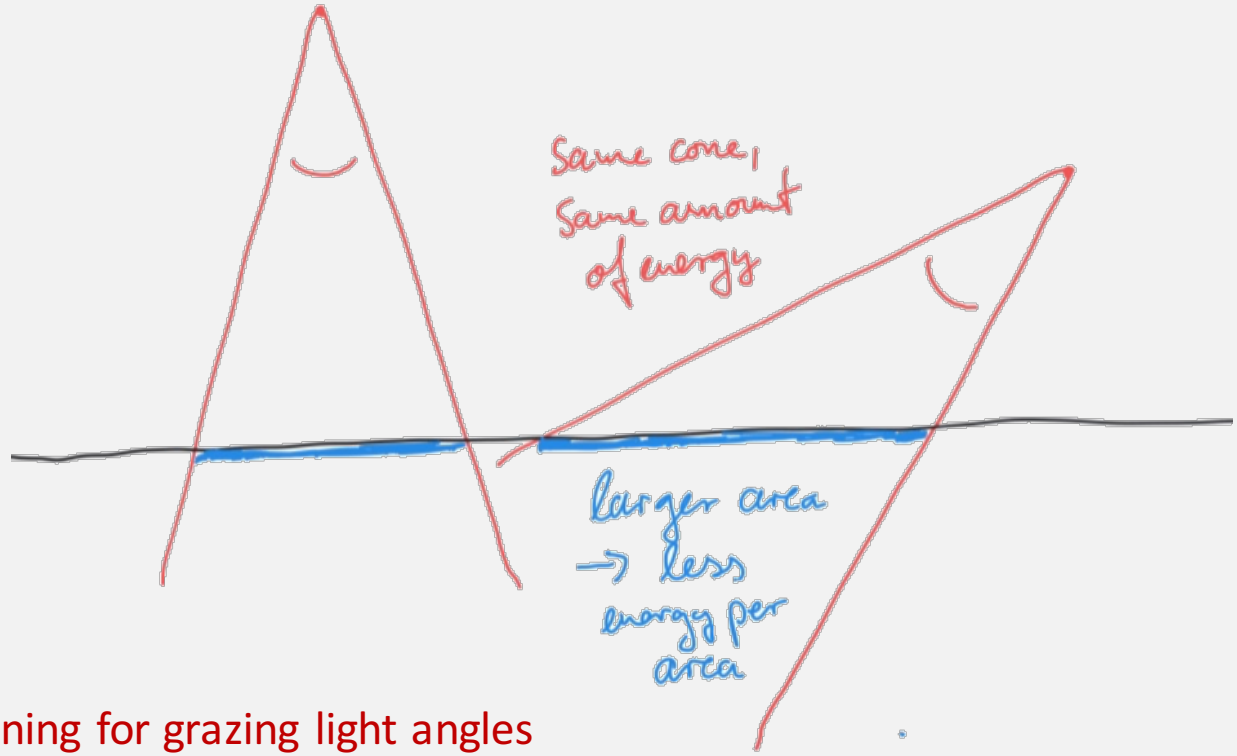
Diffuse Reflection (Lambert)

- The incident light is reflected <u>evenly in all directions</u> (perfectly diffuse surface)



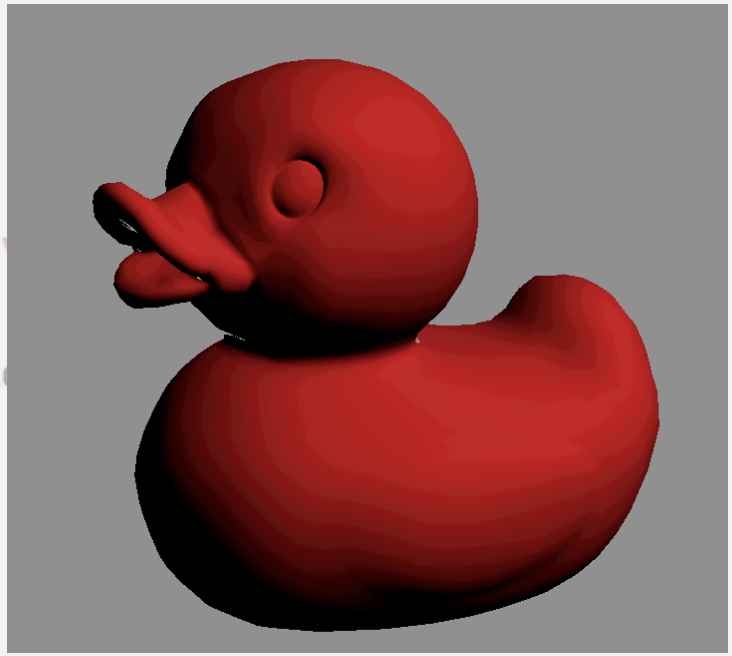
Diffuse Reflection (Lambert)

However, the amount of incident light (power per area) goes down when the angle between n and l goes up.



This darkening for grazing light angles is modeled by the cosine term `n dot l`.

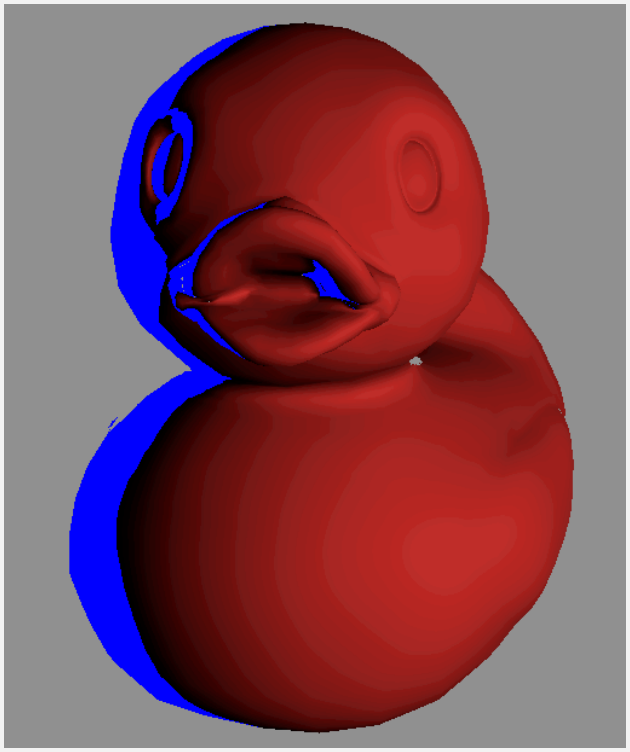
Diffuse Term Only: Results



```
vec3 phongIllum(vec3 normalDir, vec3 viewDir,
{

    vec3 ambient = k_ambient * ambientLightCol

    float ndotv = dot(normalDir,viewDir);
    if(ndotv<0)
        discard;

    float ndotl = dot(normalDir,-lightDir);
    if(ndotl<0.0)
        return vec3(0,0,0);

    vec3 diffuse = k_diffuse * lightColor * ndotl;

    return diffuse;
                                    normalDir);
    float rdotv = max( dot(r,viewDir), 0.0);
    vec3 specular = k_specular * lightColor * pow(rdotv, shininess);

    return ambient + diffuse + specular;

}
```

n dot l < 0 ?

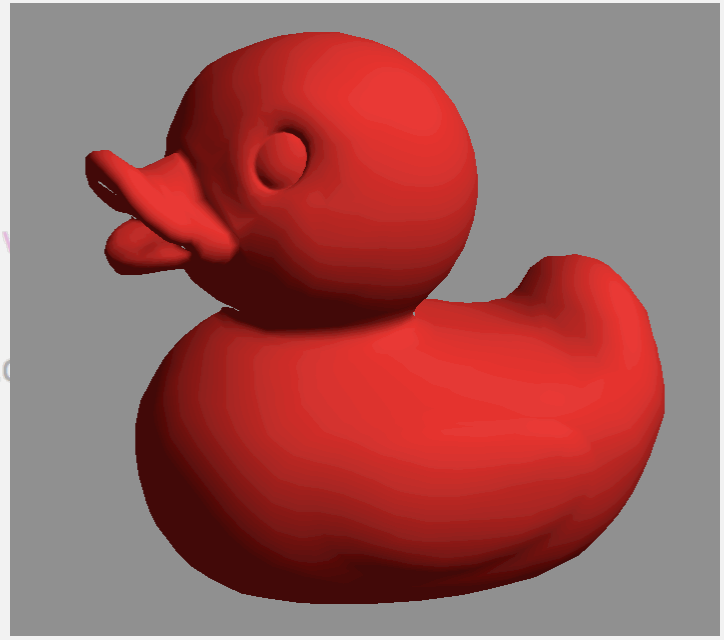
```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, vec3
{

    vec3 ambient = k_ambient * ambientLightColor;

    float ndotv = dot(normalDir,viewDir);
    if(ndotv<0)
        discard;

    float ndotl = dot(normalDir,-lightDir);
    if(ndotl<0.0)
        return vec3(0,0,1);

    vec3 diffuse = k_diffuse * lightColor * ndotl;

    return diffuse;

    normalDir);
    float rdotv = max( dot(r,viewDir), 0.0);
    vec3 specular = k_specular * lightColor * pow(rdotv, shininess);
```



The dot product falls below 0 where the light hits the surface from behind. So these are the shadow regions of the object.

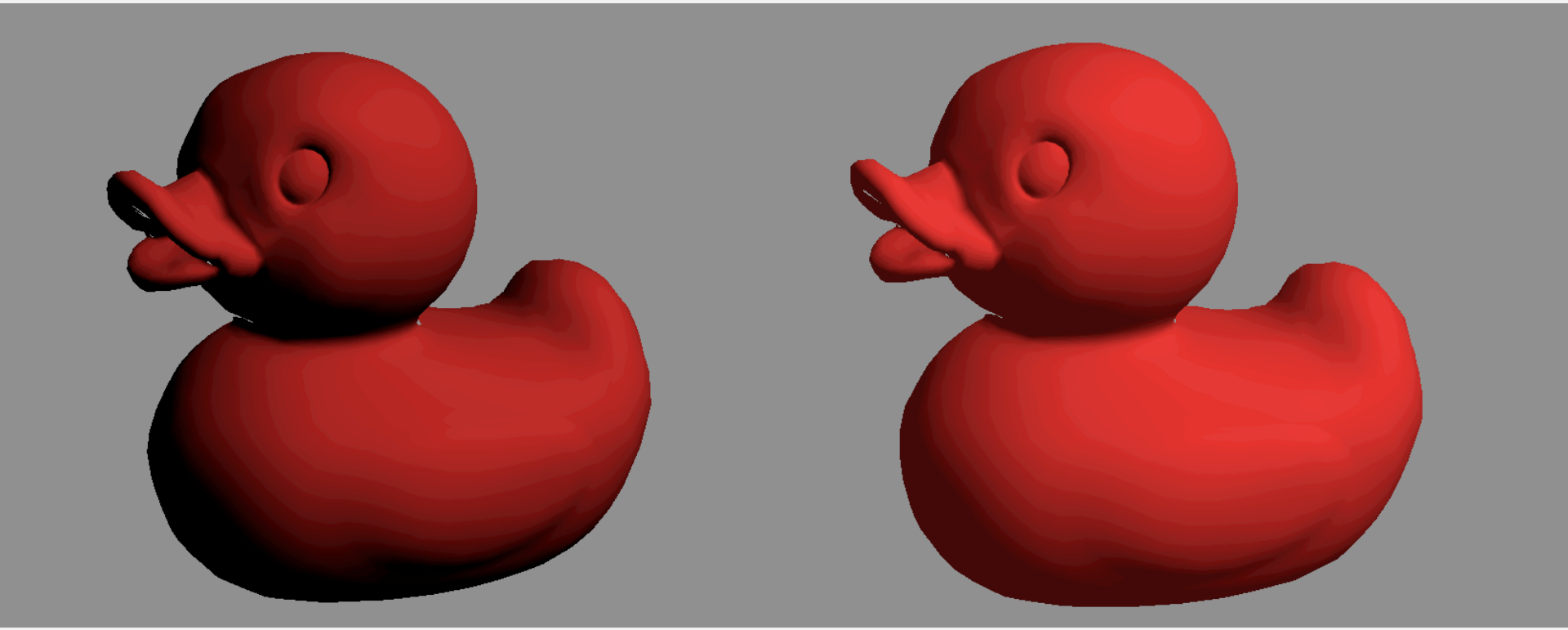
Normally these regions are shaded using the ambient term.

Ambient + Diffuse



```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, v
{

    vec3 ambient = k_ambient * ambientLightColo

    float ndotv = dot(normalDir,viewDir);
    if(ndotv<0)
        discard;

    float ndotl = dot(normalDir,-lightDir);
    if(ndotl<0.0)
        return ambient;

    vec3 diffuse = k_diffuse * lightColor * ndotl;

    return ambient + diffuse;

    float rdotv = max( dot(r,viewDir), 0.0);
    vec3 specular = k_specular * lightColor * pow(rdotv, shininess);

    return ambient + diffuse + specular;

}
```

Comparison



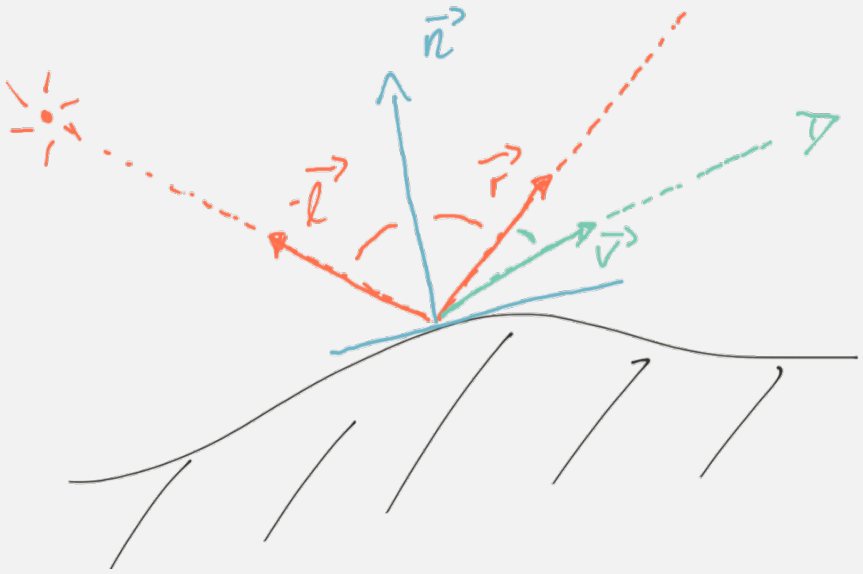
The ambient term basically makes the surface brighter,
also in the shadowy parts.

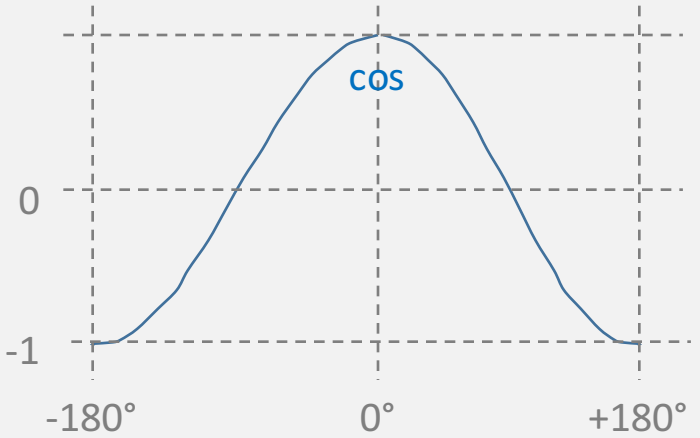
Specular Term

```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, vec3 lightDir)
{

    vec3 ambient = k_ambient * ambientLightColor;

    float ndotv = dot(normalDir,viewDir);
    if(ndotv<0)
        discard;

    float ndotl = dot(normalDir,-lightDir);

    vec3 diffuse = k_diffuse * lightColor * ndotl;

    vec3 r = reflect(lightDir,normalDir);
    float rdotv = max( dot(r,viewDir), 0.0);
    vec3 specular = k_specular * lightColor * pow(rdotv, shininess);

    return ambient + diffuse + specular;

}
```

r = ideal reflection direction

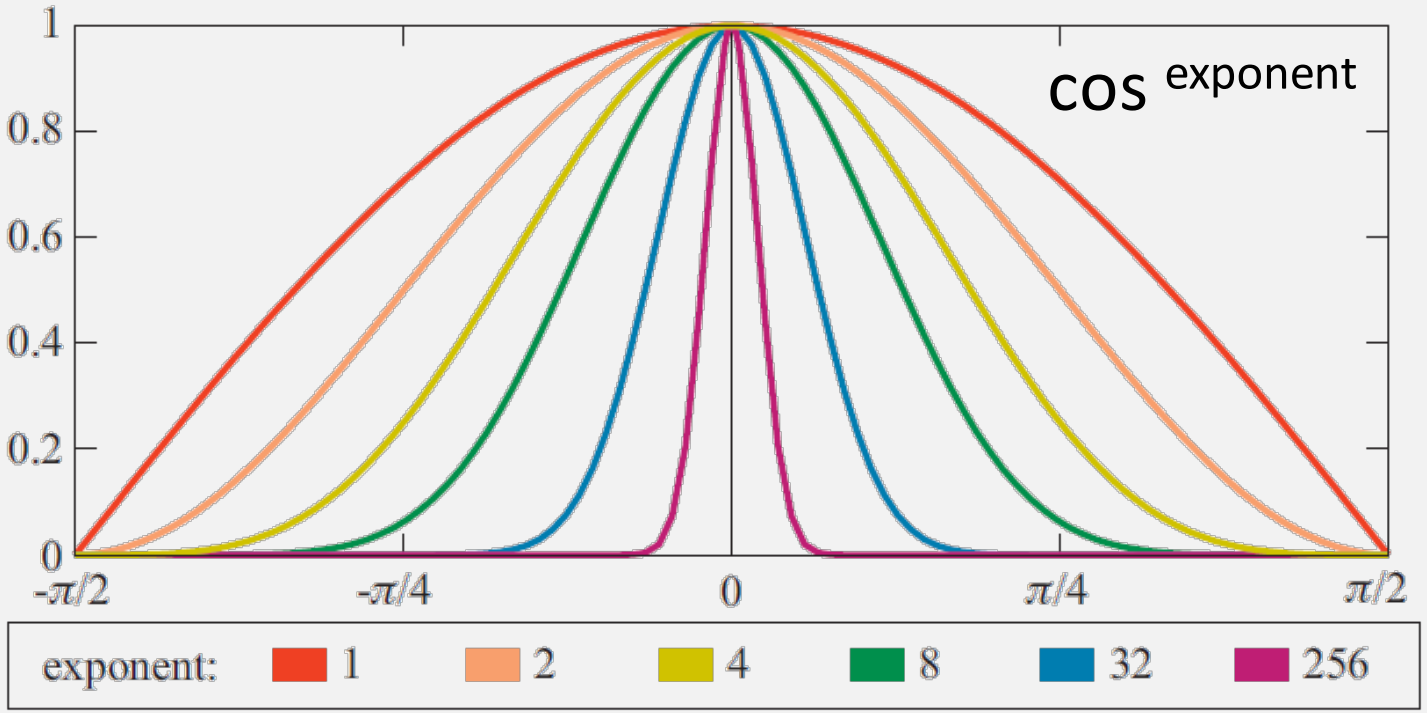
Specular Reflection





- if the surface was a mirror, a point light would be reflected only in direction `r` (ideal reflection direction)
- The further the viewer deviates from that direction, the less light will be reflected in that direction.
- This is modeled using (R dot V), resulting in a cosine distribution

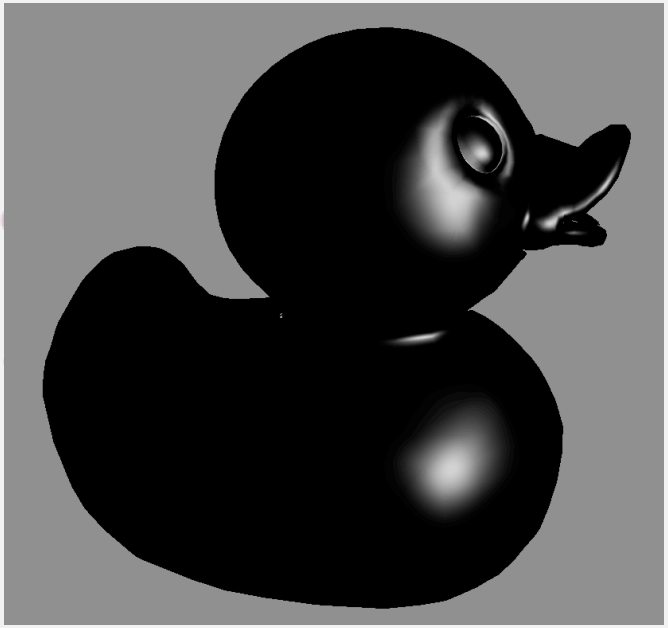
Specular Exponent ("shininess")



- Specular term contains (R dot V)^{exponent}
- If exponent > 1, brightness will decrease more quickly and steeply
- Larger exponents render smaller highlights

Image: Akenine-Möller et al., *Real-Time Rending*, 3rd Edition, AK Peters

Specular Term: Let's See!



```
vec3 phongIllum(vec3 normalDir, vec3 viewDir, ve
{

    vec3 ambient = k_ambient * ambientLightColor

    float ndotv = dot(normalDir,viewDir);
    if(ndotv<0)
        discard;

    float ndotl = dot(normalDir,-lightDir);
    if(ndotl<0.0)
        return vec3(0,0,0);
    vec3 diffuse = k_diffuse * lightColor * ndotl;

    vec3 r = reflect(lightDir,normalDir);
    float rdotv = max( dot(r,viewDir), 0.0);
    vec3 specular = k_specular * lightColor * pow(rdotv, shininess);
    return specular;
                                        specular;

}
```

The specular term is >0 only where there are highlights, with the brightest spot in the center of the highlight.

Putting it All Together