Universitat de Girona

Shading and Illumination

Gonzalo Besuievsky

IMAE - UdG

Universitat de Girona

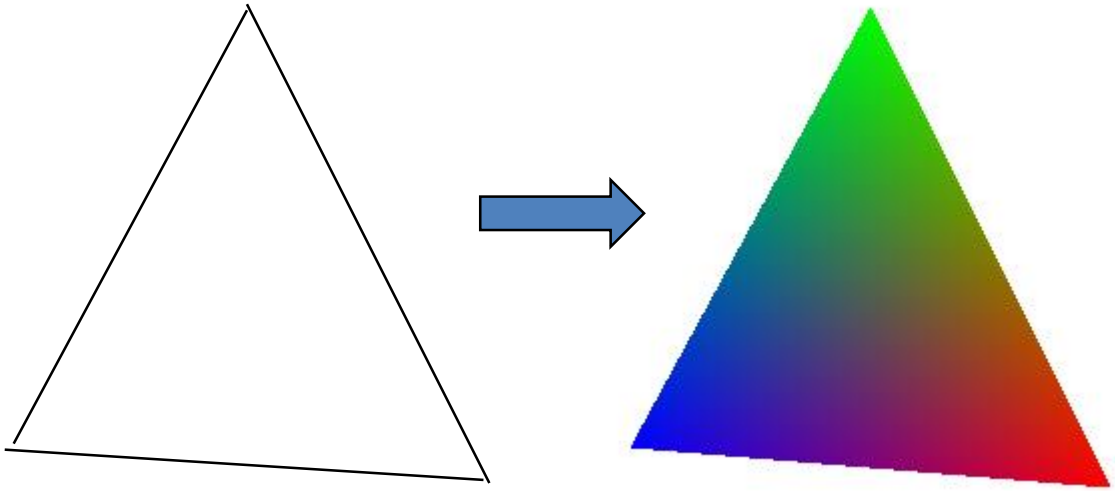
Content

- Introducction: Rendering and Ilumination
- Color
- Ilumination Models
  - Local models
  - Global illumination
- Illumination in WebGL

Universitat de Girona

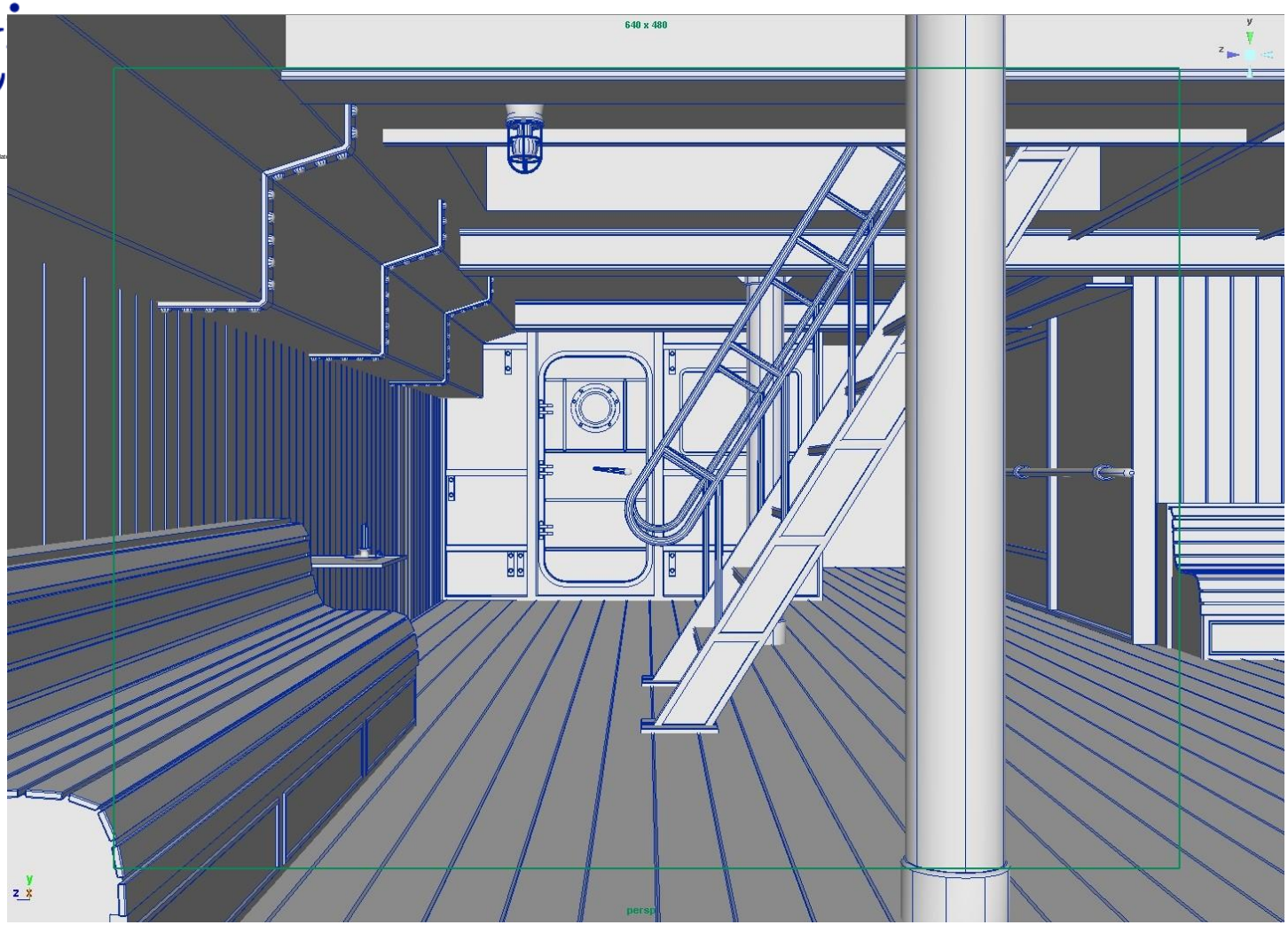
Rendering

- We know how to specify the geometry, but how is the color calculated?



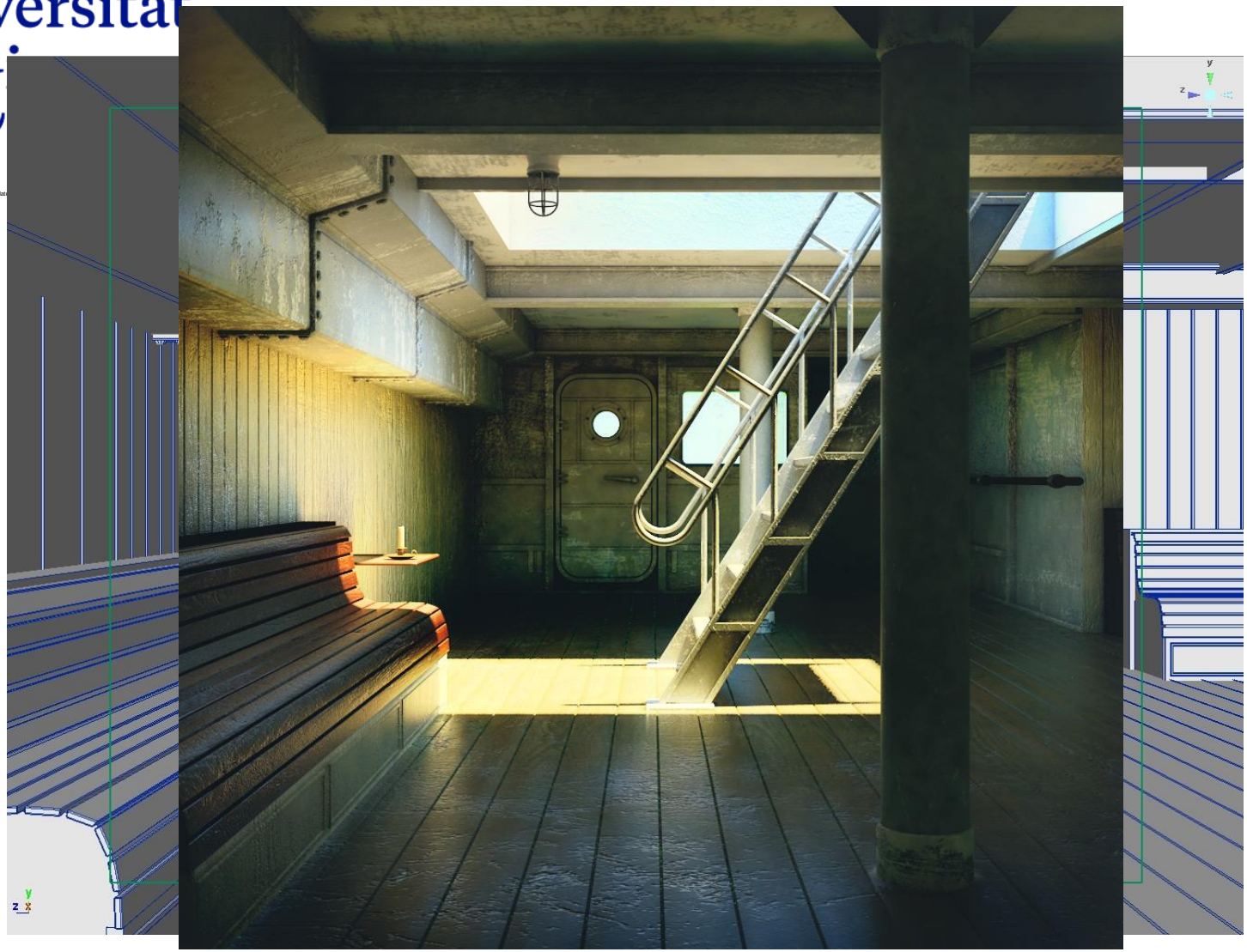
Universitat de G

- We know how to specify the geometry but how is the color calculat



Universitat de Gi

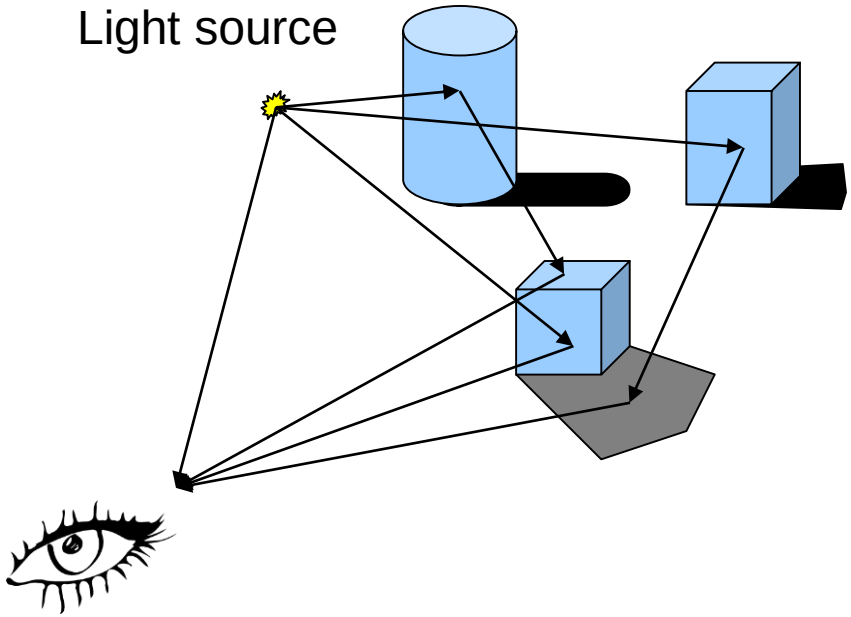
- We know how to specify the geometry but how is the color calculat



Universitat de Girona

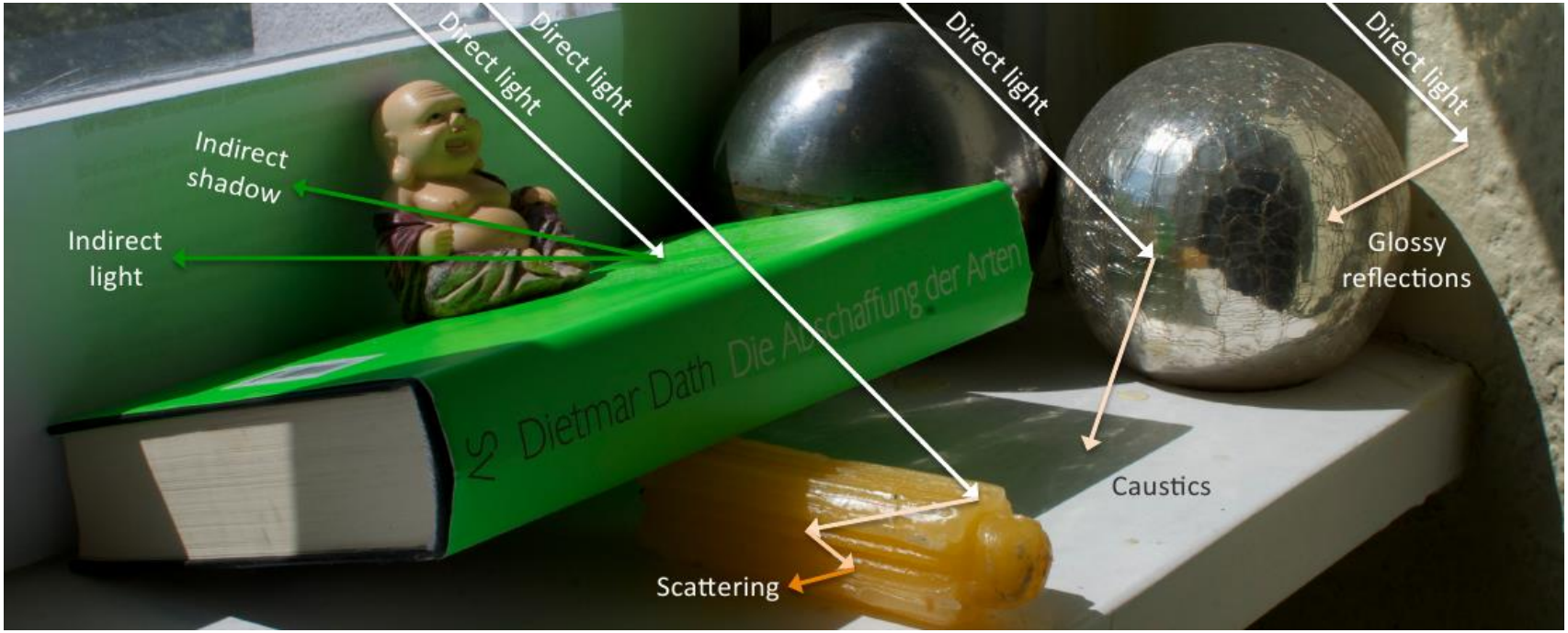
Light-Matter interaction

- What we see is light after interaction with matter



Universitat de Girona

Photograph



Universitat de Girona

Rendering: simulation of light transport

- So, **what** makes up the final color of an object?

Universitat de Girona

Rendering: simulation of light transport

- Examples:
  - Diffuse scattering
    - matte surfaces
  - Specular reflection
    - shiny surfaces
    - highlight
  - Transparency
    - glass, water
    - penetrate the surface

Universitat de Girona

Rendering: simulation of light transport

- **How** do we represent these observations in a mathematical framework?

Universitat de Girona

Rendering: simulation of light transport

- Real time rendering is generally **not** concerned with using a "correct" lighting equation, just a series of hacks to make things look right with as little computational effort as possible
- Offline rendering **does**

Universitat de Girona

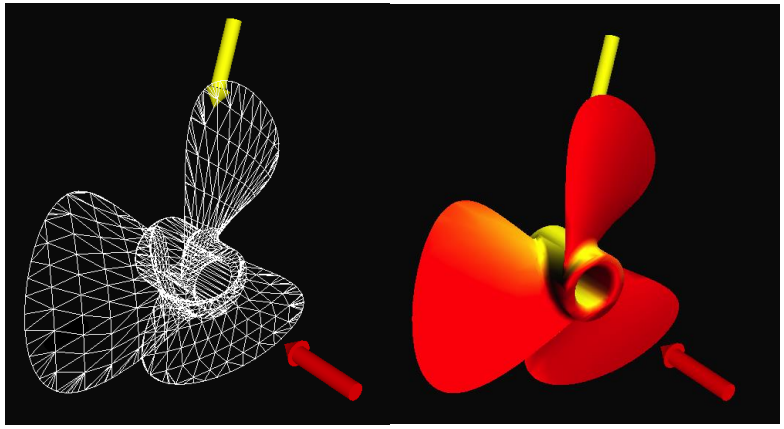
Illumination models

- Local models
  - Graphics hardware
  - Approximation
  - Do not use all the objects to compute illumination
- Ray tracing models
  - Still an approximation
  - Use all the objects to compute illumination
- Global illumination models
  - Simulate light transport

Universitat de Girona

Local illumination

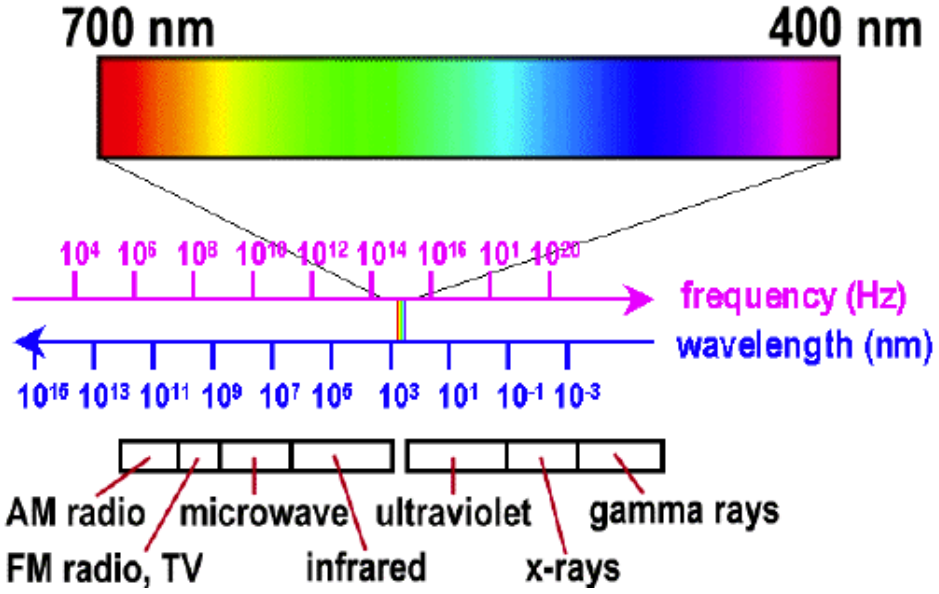
- Input:
  - a 3D object
  - Material and color of the object
  - Position and structure of the light source
  - “Intensity” of the light source
- Output:
  - Color and intensity of points of the given object



Universitat de Girona

Light and Color

- Electromagnetic radiation



Universitat de Girona

Local illumination

- Color is computed from:
  - Camera position
  - Light position
  - Surface properties
- Objects are illuminated by lights
- You can see:
  - Diffuse illumination, glossy spots.
  - No shadows, no mirror, no refraction
- Fast

Universitat de Girona

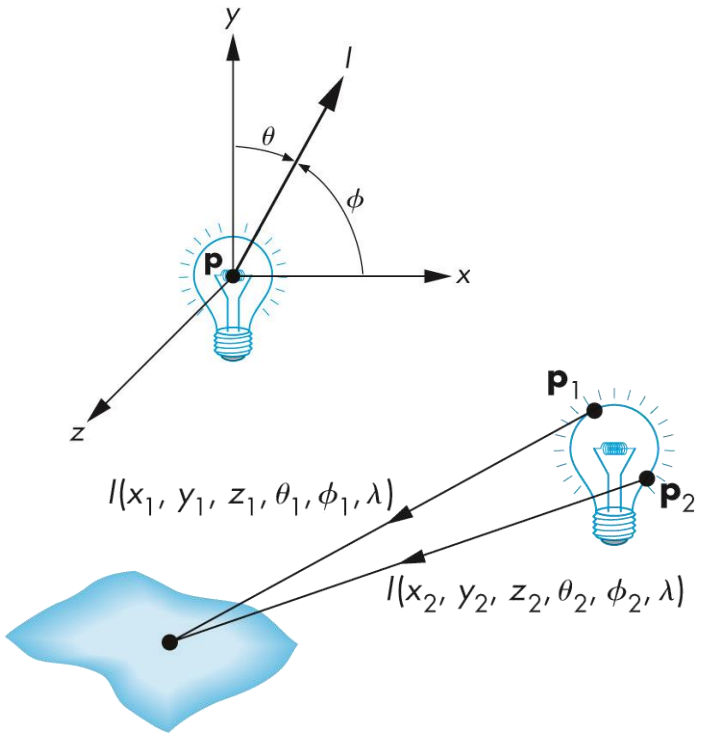
Light sources

- Two basic types of emitting light
  - Incandescence:
    - Sunlight, regular light bulbs (not florescent) and fires
  - Luminescence:
    - Florescent lights, neon light, LED, television screens and computer monitors
- A photon of light is emitted by the atom, at a very specific wavelength

Universitat de Girona

General Light Source Model

- Can be modeled by an illumination function I(x,y,z,θ,ϕ,λ)
- Each frequency considered independently
- Total contribution can be computed by integration
- Directional properties can vary with frequency
- Too complicated to compute analytically



Universitat de Girona

Simplified Light Sources

- Four types: ambient lighting, point sources, spotlights, and distant lights
- Light sources with three components, RGB
  - based on “three-color theory”
- Each component calculated independently
- Intensity or luminance:

$$\mathbf{I} = \begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix}$$

Universitat de Girona

Type #1: Ambient Light

- Models uniform illumination
- Simplified as an intensity that is identical at every point in the scene:

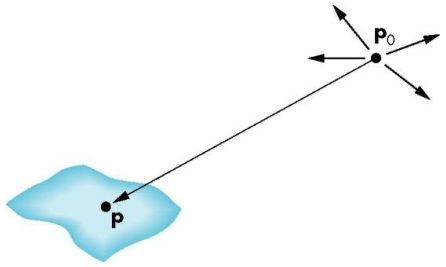
$$\mathbf{I}_a = \begin{bmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{bmatrix}$$

Universitat de Girona

Type #2: Point Sources

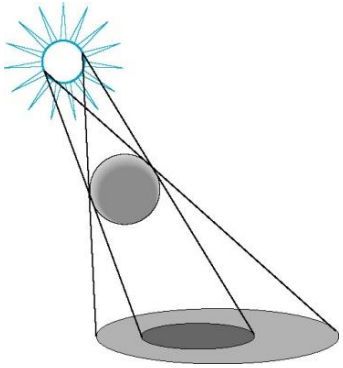
- Located at p0:

$$\mathbf{I}(\mathbf{p_0}) = \begin{bmatrix} I_r(\mathbf{p_0}) \\ I_g(\mathbf{p_0}) \\ I_b(\mathbf{p_0}) \end{bmatrix}$$



- Intensity received at p:
- High contrast than surface light
- Can be made soft using a distance term

$$\mathbf{i}(\mathbf{p}, \mathbf{p_0}) = \frac{1}{|\mathbf{p} - \mathbf{p_0}|^2} \mathbf{I}(\mathbf{p_0})$$

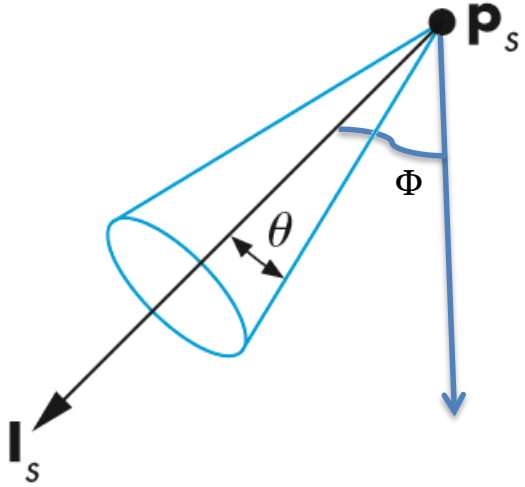


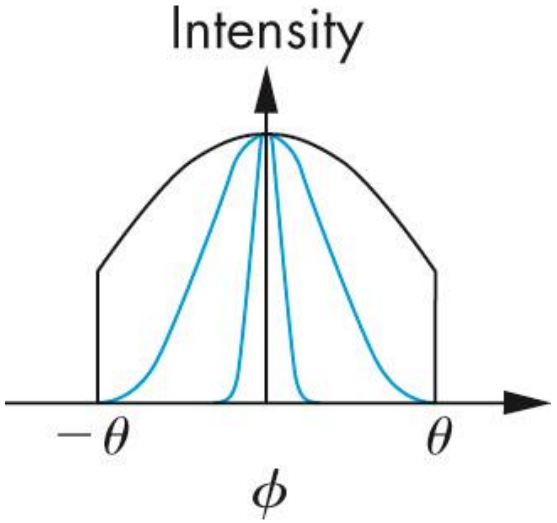
Universitat de Girona

Type #3: Spotlights

- Cone-shaped directional range
- Distribution of the light within the cone usually defined by

$$\cos^e \phi$$

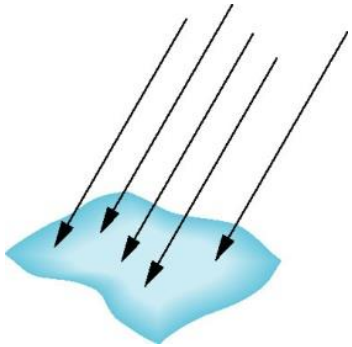




Universitat de Girona

Type #4: Distant Light Sources

- Rays can be assumed parallel
- Direction instead of location:

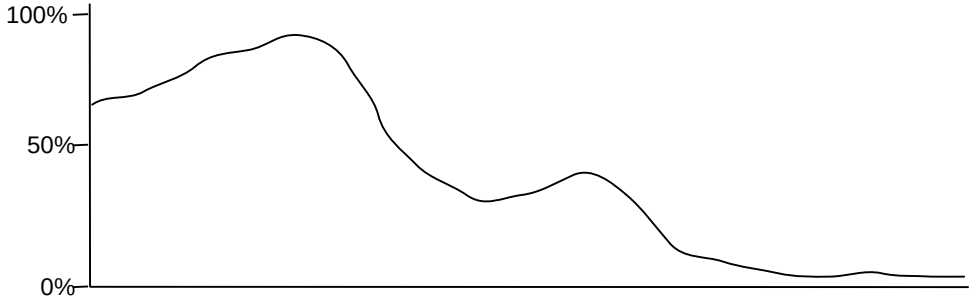


$$\mathbf{p}_0 = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \mathbf{p}_0 = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

Universitat de Girona

Types of light interaction

- Absorption
  - Responsible of “colors” of object



- Scattering
  - Change in direction/wavelength

Universitat de Girona

Scattering: change of direction

- For solids and liquids:
  - Reflection
  - Refraction
- For gas:
  - Volume scattering

Universitat de Girona

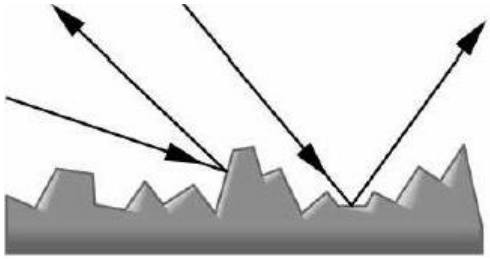
Scattering: change of wavelength

- Fluorescence: emission of light by a substance that has absorbed light or other electromagnetic radiation of a different wavelength
- Phosphorescence: delayed emission of light.

Universitat de Girona

Reflection

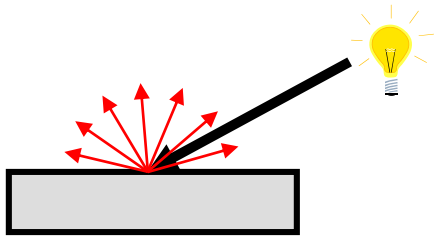
- **Definition**: “change in direction of a wavefront at an interface between two different media so that the wavefront returns into the medium from which it originated”.



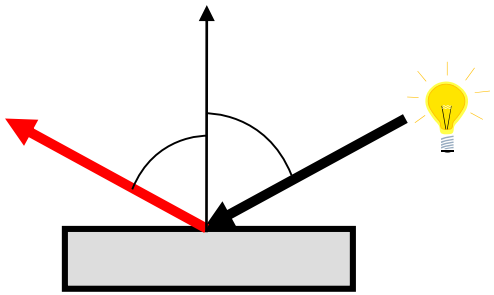
Universitat de Girona

Types of reflection

- Diffuse
- Mirror
- Glossy (specular)



Diffuse

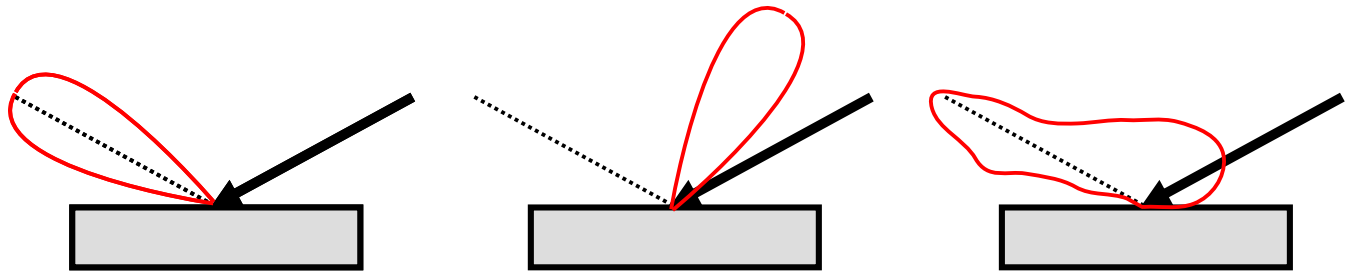


Mirror

Universitat de Girona

Glossy reflection

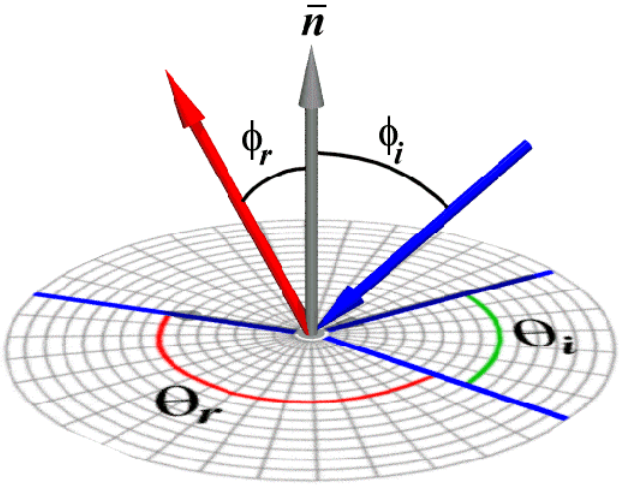
- It is the “real one”
  - There are no perfect diffuse or specular material in nature



Universitat de Girona

Bidirectional Reflectance Distribution Function (BRDF)

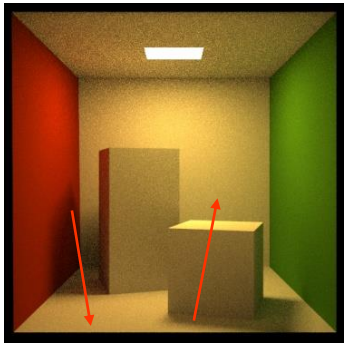
- Defines how light is reflected at a point for a given wavelength on a opaque surface
- *f(N,L,V,λ)*
- *E.g., Phong's formula*

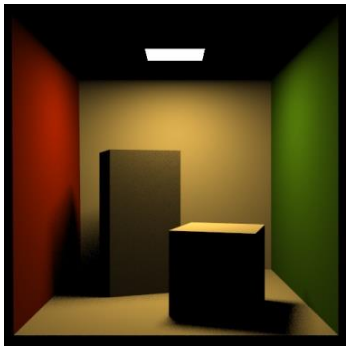


Universitat de Girona

Ambient light

- "Physical rules" are too simplified
- No indirect or global interaction of light





- A hack to overcome the problem: use "ambient light"

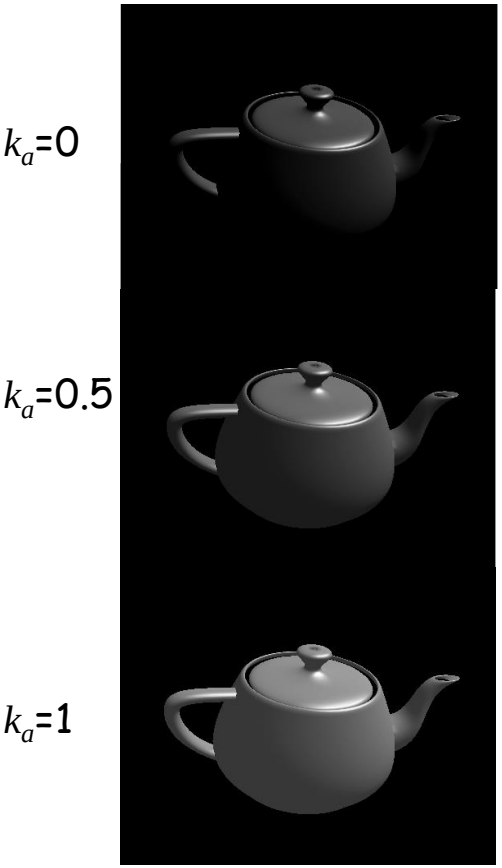
Universitat de Girona

Ambient light specification

- Not situated at any particular point
- Spreads uniformly in all directions

$$I = k_a I_a$$

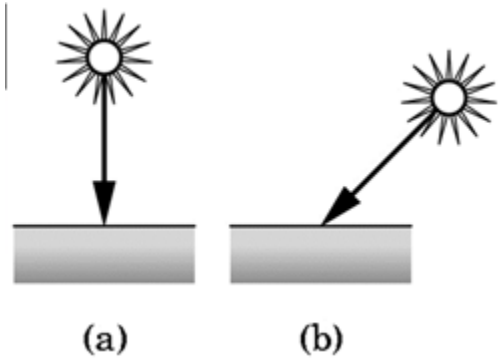
- I_a : intensity of ambient light in the environment
- I : ambient light at a given point
- $0 \leq k_a \leq 1$: coefficient of ambient light reflection

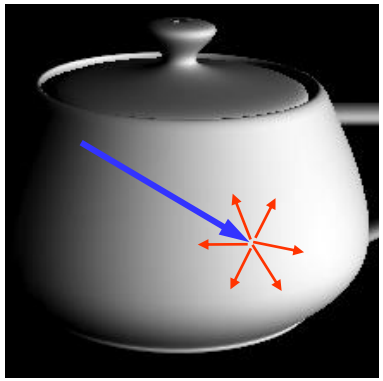


Universitat de Girona

Diffuse reflection

- A perfect diffuse reflector (Lambertian surface) scatters the light equally in all directions
- Same appearance to all viewers
- Depends on:
  - Material of the surface
  - The position of the light

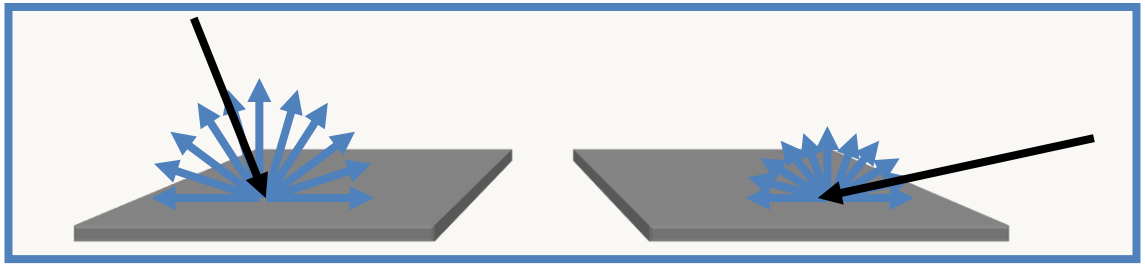




Universitat de Girona

Diffuse Reflection

- Ideal diffuse reflection
  - An **ideal diffuse reflector**, at the microscopic level, is a very rough surface (real-world example: chalk)
  - Because of these microscopic variations, an incoming ray of light is equally likely to be reflected in any direction over the hemisphere



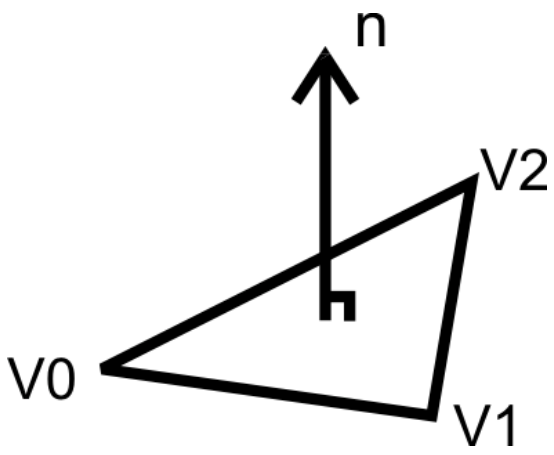
  - *What does the reflected intensity depend on?*

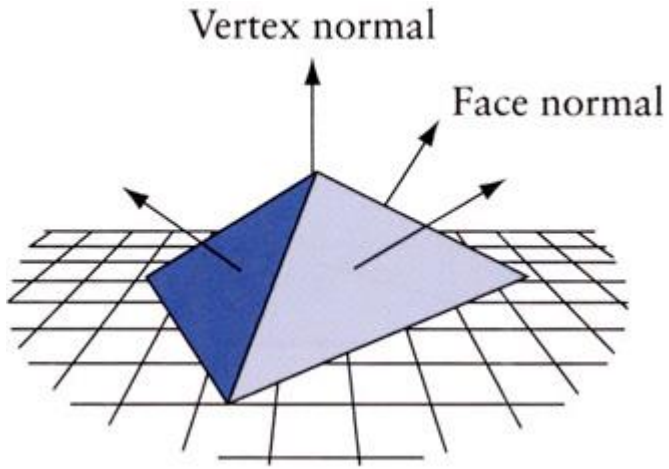
Universitat de Girona

Normals

–For each triangle we can define a normal for the face

–For each vertex we an define a normal by interpolating normals of attached faces

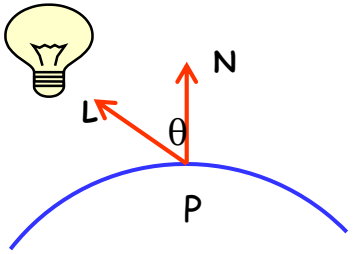




Universitat de Girona

Diffuse: Two important vectors

- To compute the intensity at P, we need
  - The unit normal vector **N**,
  - The unit vector **L**, from P to the light



What is the diffuse color at P?

Universitat de Girona

Computing Diffuse Reflection

- Independent of the angle between **N** and **v**
- Does depend on the direction **L** (Lambertian surface)

Therefore, the diffuse component is:

$$I_{diffuse} = I_{source} k_{diffuse} \cos(q)$$

$$I_{diffuse} = I_{source} k_{diffuse} \frac{L \cdot N}{|L||N|}$$

$$I_{diffuse} = I_{source} k_{diffuse} \max(\frac{L \cdot N}{|L||N|}, 0)$$

Diffuse Reflection Coefficient

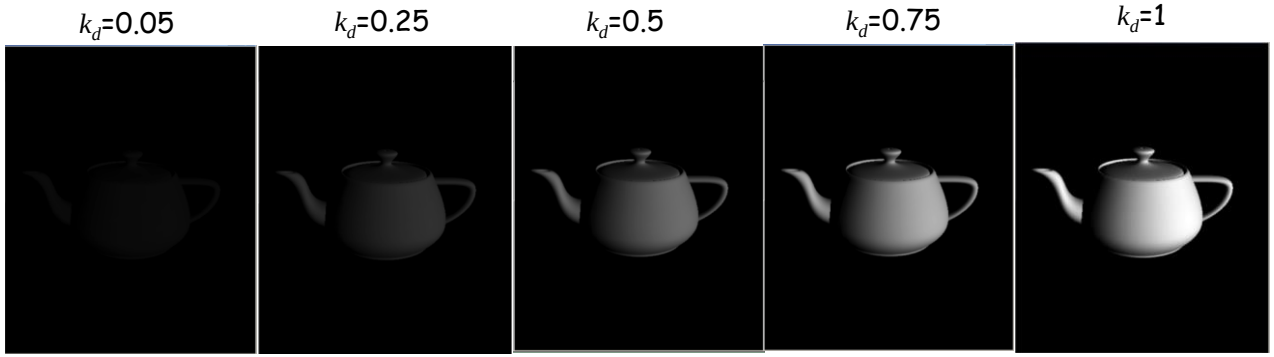
Adjustment for 'inside' face

Universitat de Girona

Coefficient of diffuse reflection

- k_d is usually determined by a trial and error approach
- Examples:

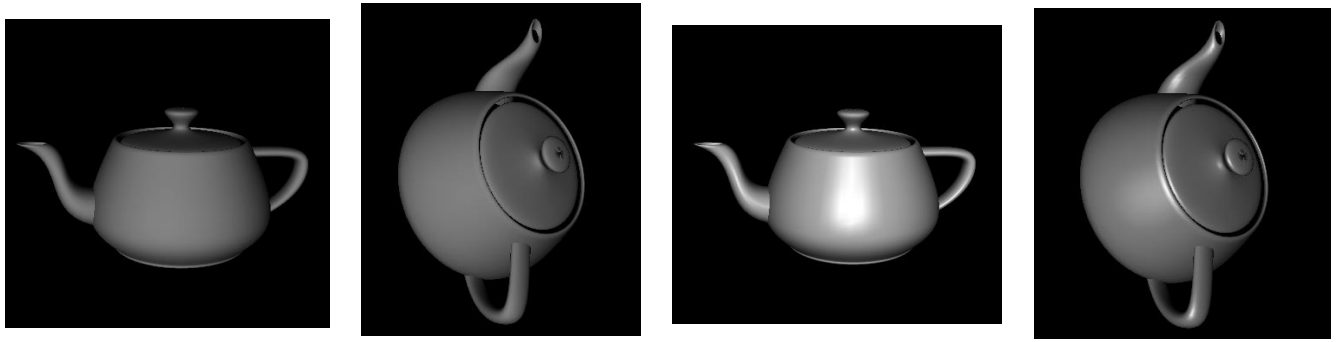
Component	Red	Green	Blue
Gold	0.75	0.01	0.5
Black plastic	0.6	0.01	0.5
Silver	0.22	0.01	0.5



Universitat de Girona

Specular reflection

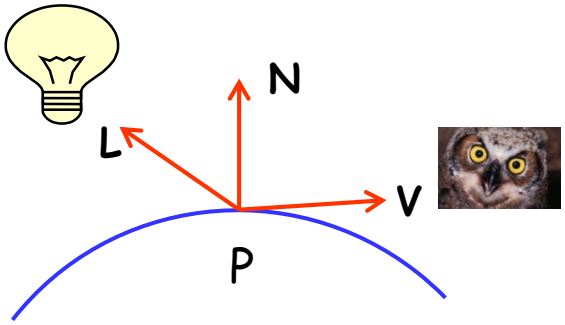
- Diffusive reflection: no highlights, rough surface
- Specular reflection: highlights, shiny and smooth surfaces
- View dependent reflection



Universitat de Girona

Three important vectors

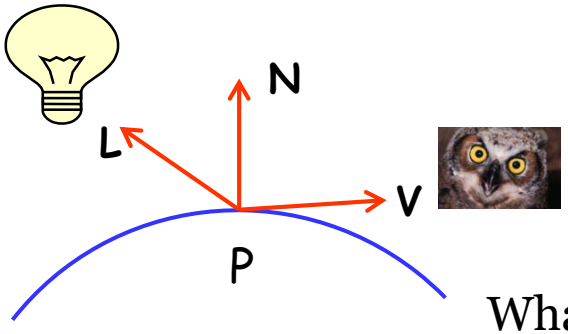
- To compute the intensity at P, we need
  - The unit normal vector **N**,
  - The unit vector **L**, from P to the light
  - The unit vector **V**, from P to the viewer



Universitat de Girona

Three important vectors

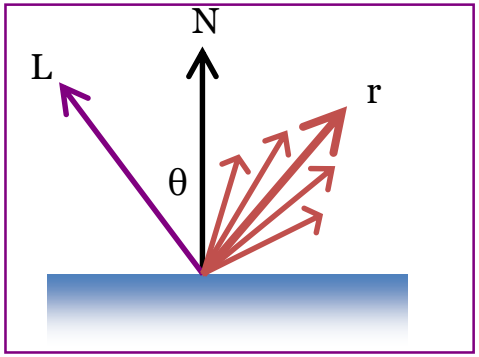
- To compute the intensity at P, we need
  - The unit normal vector **N**,
  - The unit vector **L**, from P to the light
  - The unit vector **V**, from P to the viewer

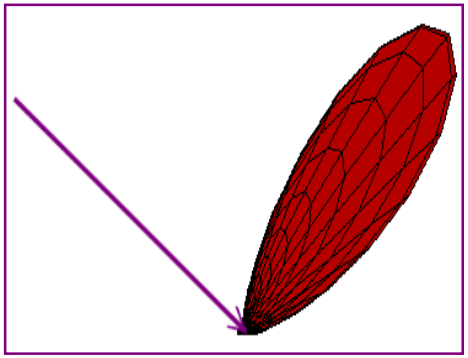


What is the specular illumination at P?

Universitat de Girona

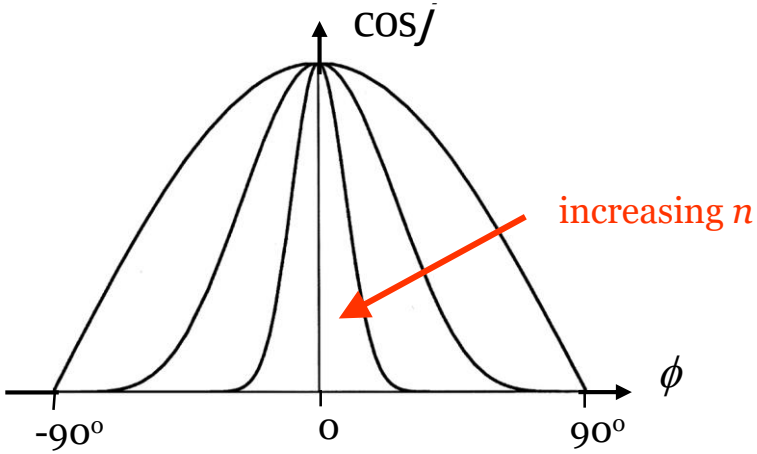
Non-Ideal Specular Reflectance: Phong Model

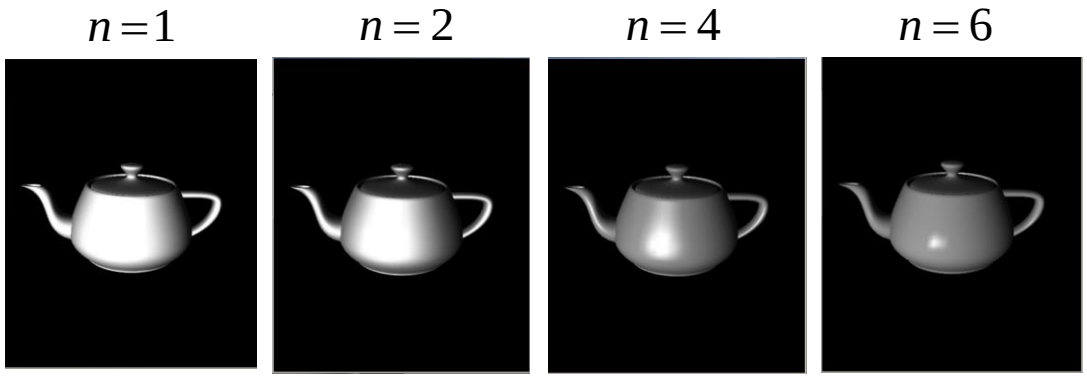




Universitat de Girona

The shininess coefficient





Universitat de Girona

Phong Lighting

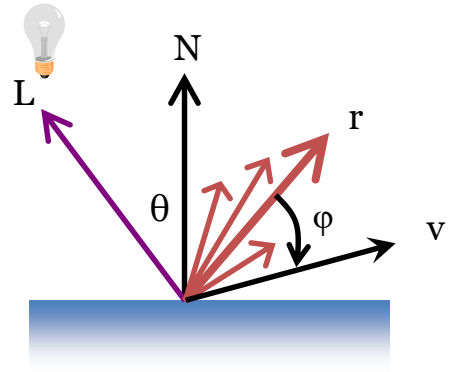
The Specular Intensity, according to Phong model:

Shininess factor

$$I_{specular} = I_{source} k_{specular} \cos^{n}(f)$$

Specular Reflection Coefficient

$$0 \le k_s \le 1$$

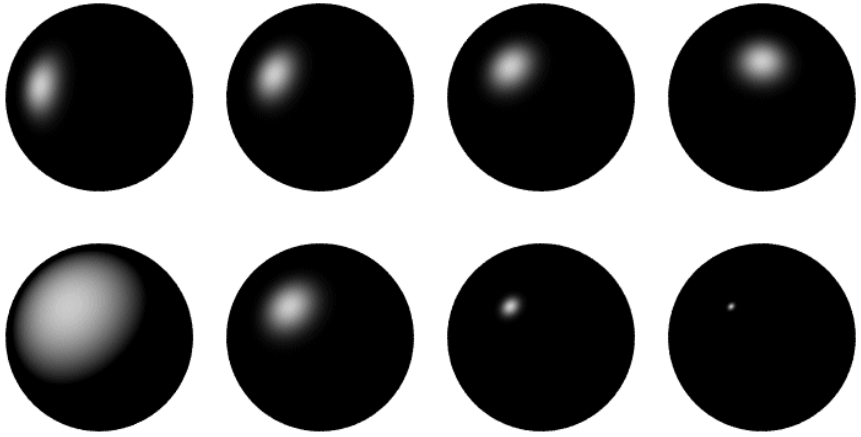


$$I_{specular} = I_{source} k_{specular} \left(\frac{r \cdot v}{|r||v|} \right)^{n}$$

Universitat de Girona

Phong Lighting Examples

- These spheres illustrate the Phong model as *L* and *n* are varied:

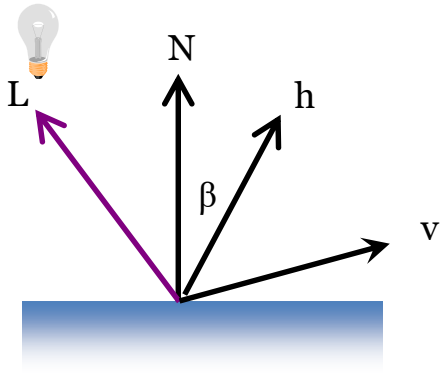


Universitat de Girona

Blinn and Torrance Variation

- In the Phong Model, **r** needs to be found
- Instead, halfway vector **h** = **L** + **V** is used
  - –angle between **N** and **h** measures the falloff of intensity

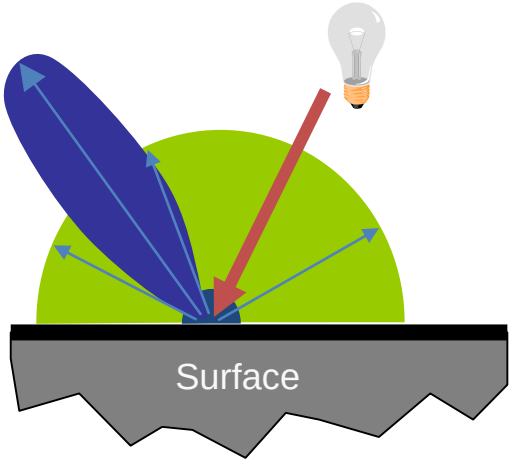
$$I_{specular} = I_{source} k_{specular} \left(\frac{h \cdot N}{|h||N|} \right)^{n}$$



Universitat de Girona

Combining Everything

- Simple analytic model:
  - Ambient +
  - diffuse reflection +
  - specular reflection



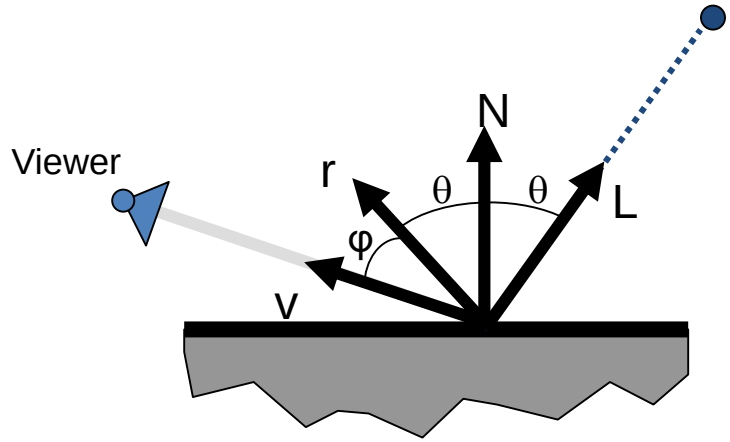
Universitat de Girona

The Final Combined Equation

- Single light source:

$$lambert = \max\left(0, \frac{L \cdot N}{|L||N|}\right)$$

$$phong = \max\left(0, \frac{r \cdot N}{|r||N|}\right)$$



$$I = I_a k_a + I_d k_d \cdot lambert + I_s k_s \cdot (phong)^n$$

Universitat de Girona

Adding Color

- Consider R, G, B components individually
- Add the components to get the final color of reflected light

$$I = I_{ar} k_{ar} + I_{dr} k_{dr} \times lambert + I_{sr} k_{sr} \times (phong)^n$$

$$I = I_{ar} k_{ar} + I_{dr} k_{dr} \times lambert + I_{sr} k_{sr} \times (phong)^n$$

$$I = I_{ar} k_{ar} + I_{dr} k_{dr} \times lambert + I_{sr} k_{sr} \times (phong)^n$$

Universitat de Girona

Multiple Light Sources

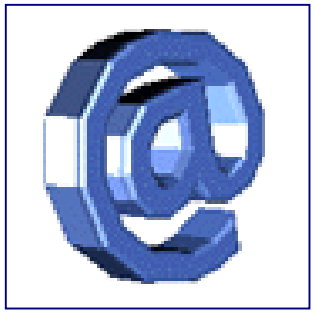
$$I = I_a k_a + \sum_i (I_{i,d} k_d \cdot lambert_i + I_{i,s} k_s \cdot (phong_i)^n)$$

Universitat de Girona

Polygon Shading

Types of Shading Model

Flat Shading



Smooth Shading

Gouraud Shading

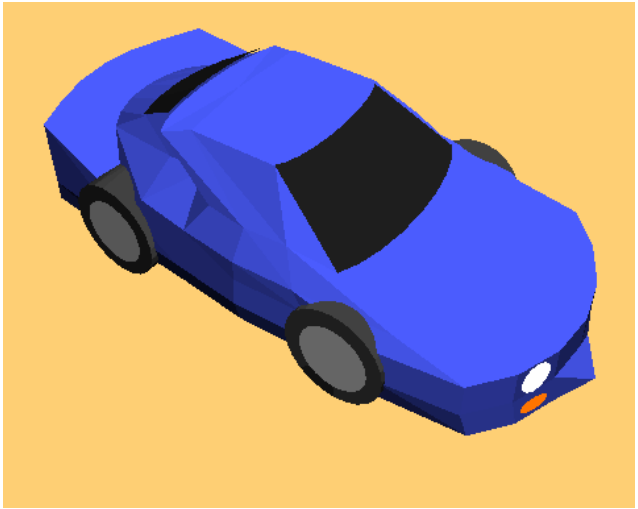
Phong Shading

Universitat de Girona

Flat Shading

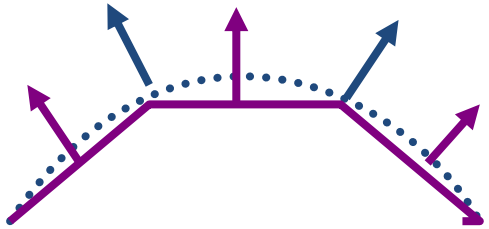
- For each polygon
  - Determines a single intensity value
  - Uses that value to shade the entire polygon

- Assumptions
  - Light source at infinity
  - Viewer at infinity
  - The polygon represents the actual surface being modeled



Universitat de Girona

Smooth Shading



- Introduce vertex normals at each vertex
  - –Usually different from facet normal
  - –Used only for shading
  - –Think of as a better approximation of the real surface that the polygons approximate
- Two types
  - –Gouraud Shading
  - –Phong Shading (do not confuse with Phong Lighting Model)

Universitat de Girona

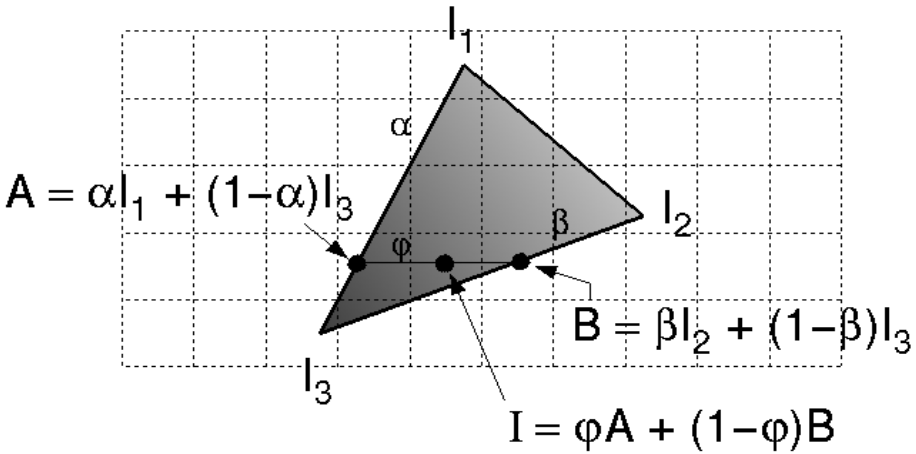
Gouraud Shading

- This is the most common approach
  - Perform Phong lighting at the vertices
  - Linearly interpolate the resulting colors over faces
    - Along edges
    - Along scan lines

Universitat de Girona

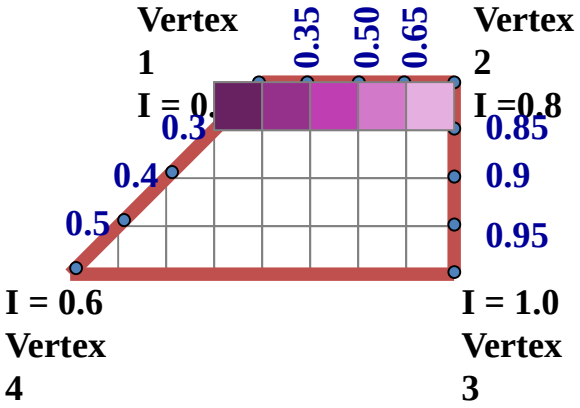
Shadowing: color interpolation

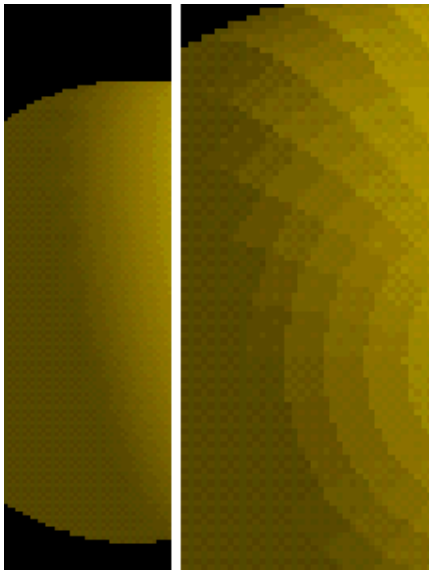
For every horizontal polygon section



Universitat de Girona

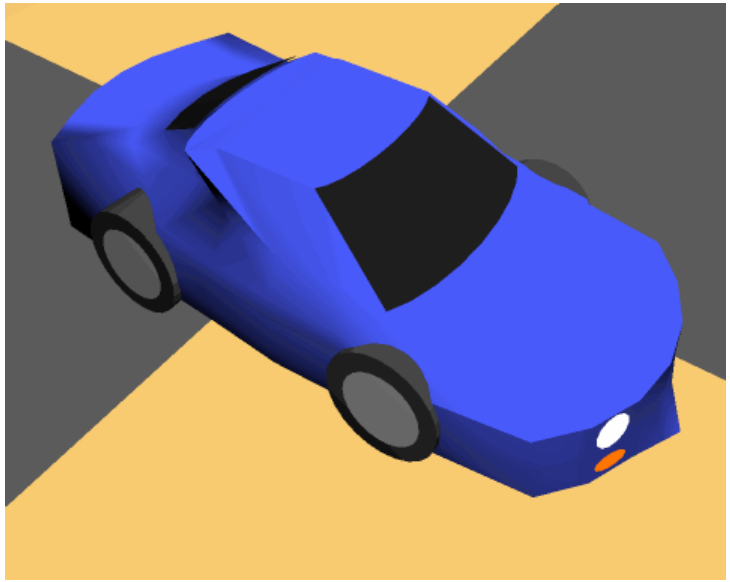
Gouraud Shading





Universitat de Girona

Gouraud Shading

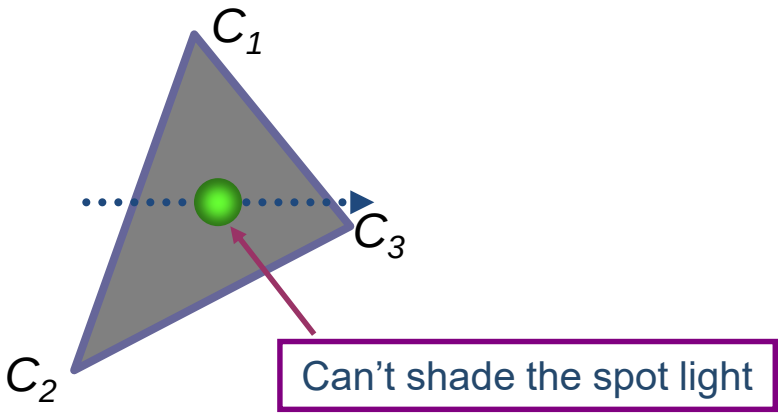


Gouraud Shading

Universitat de Girona

Gouraud Shading

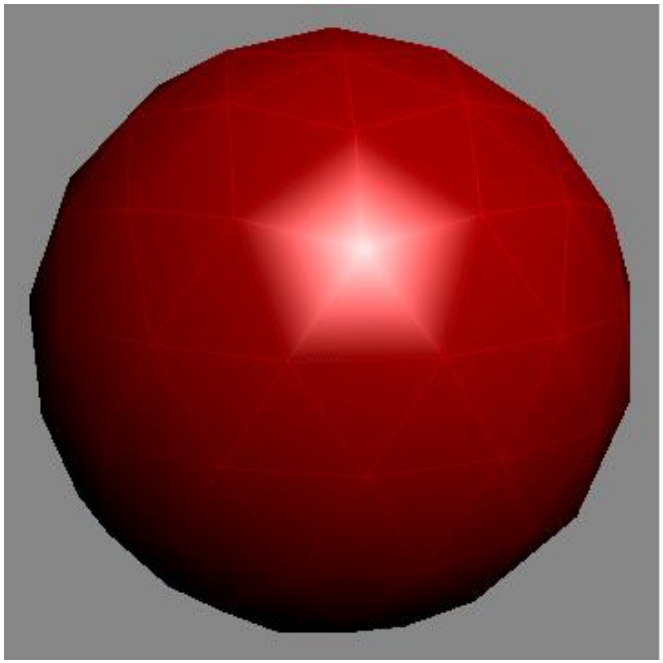
- Artifacts
  - –Often appears dull
  - –Lacks accurate specular component
    - If included, will be averaged over entire polygon

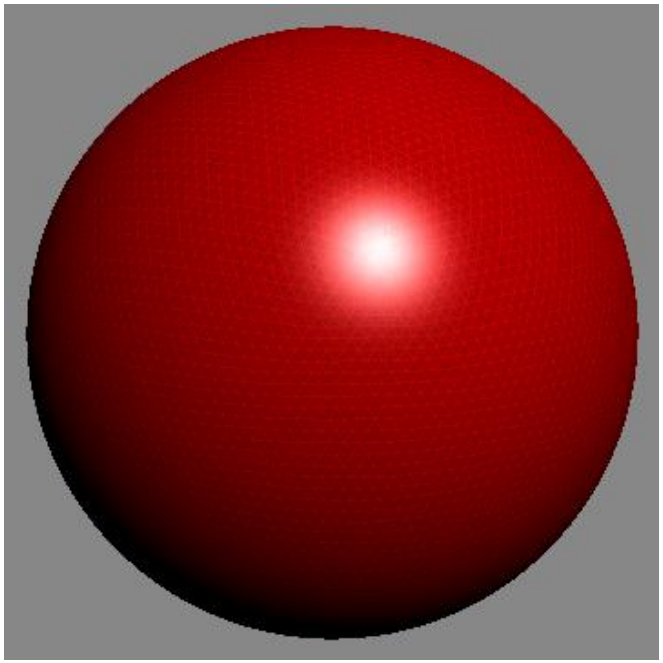


Universitat de Girona

Gouraud Shading

- Specular highlight quality tied to detail of mesh
- Specular highlights can even be missed





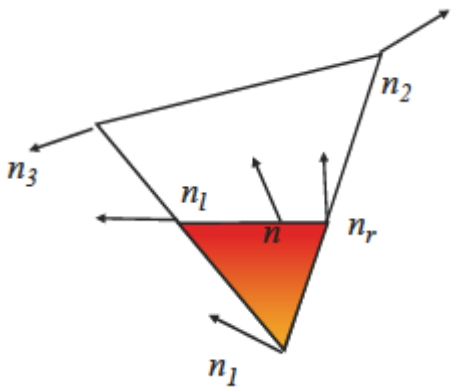
Universitat de Girona

Phong Shading: Clarification

- **Phong reflection** model or phong lighting refers to

$$I = K_d L_d (l.n) + K_s L_s (r.v)^{\alpha} + K_a L_a$$

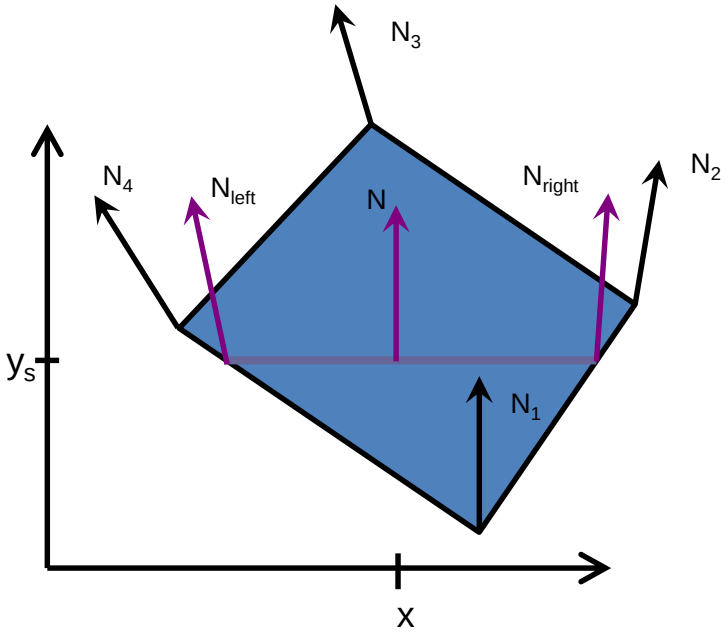
- **Phong Shading** refers to filling a triangle by interpolating the **normal** and calculating the color at each point



Universitat de Girona

Phong Shading

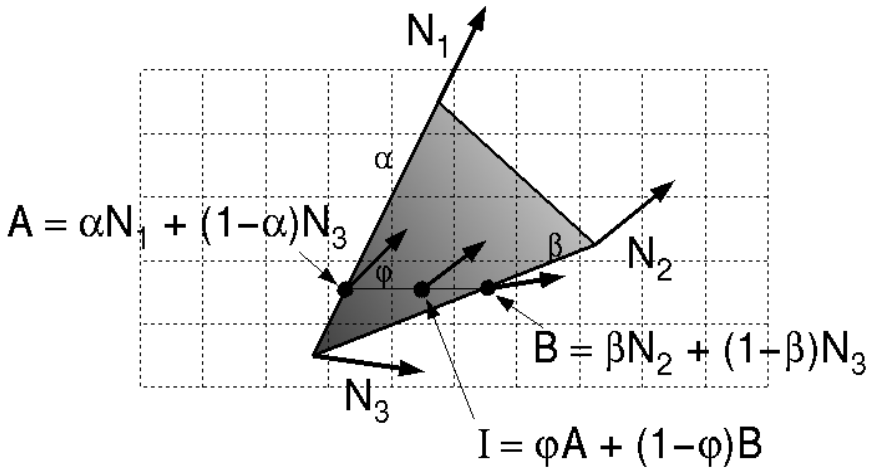
Interpolate normal vectors at each pixel



Universitat de Girona

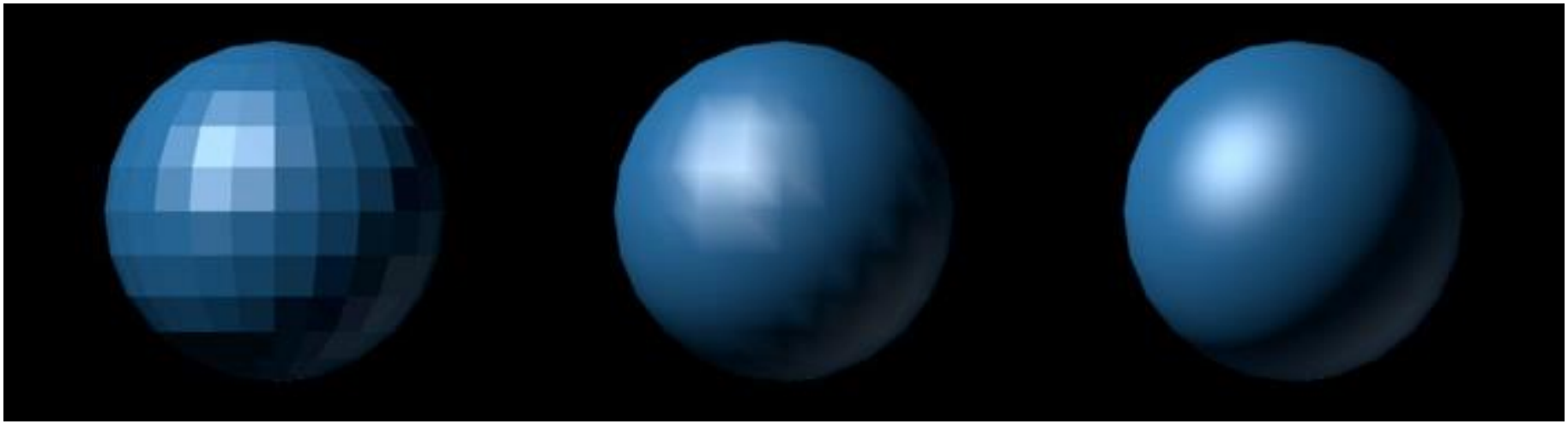
Shadowing: Phong Shading
(Normal interpolation)

Computes the normal at each point by **interpolating** the ones at the vertices



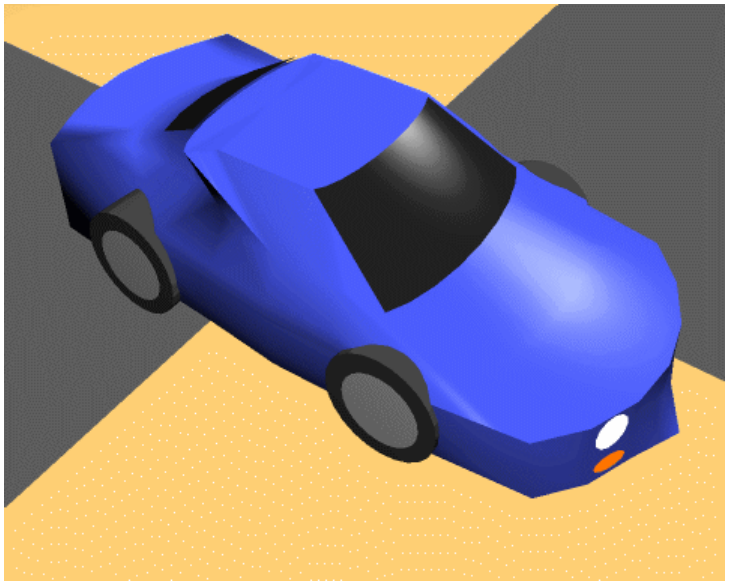
Universitat de Girona

Flat vs Gouraud vs Phong



Universitat de Girona

Phong Shading

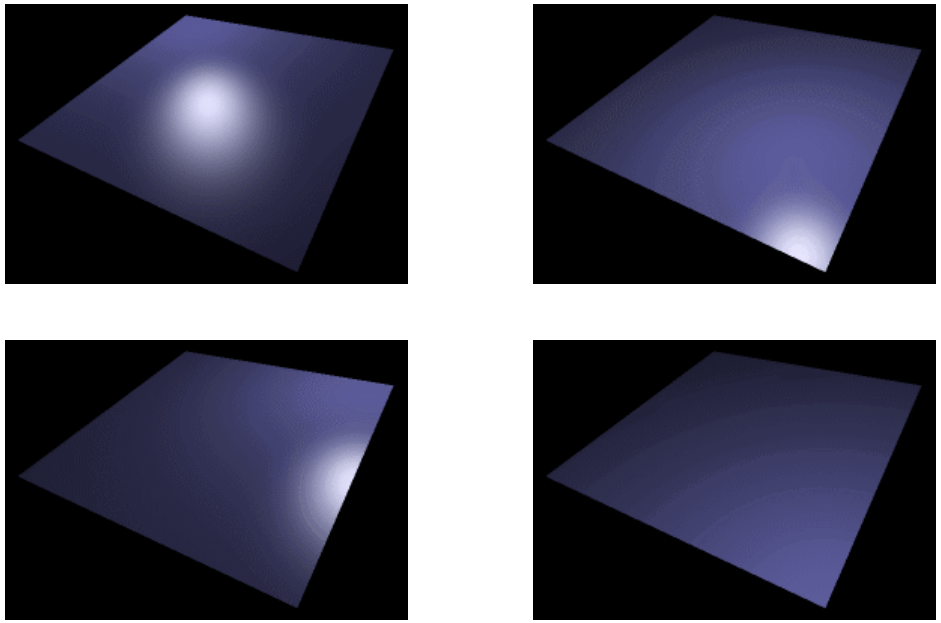


Phong Shading

Universitat de Girona

Phong vs Gouraud Shading

If a highlight does not fall on a vertex
Gouraud shading may miss it completely,
but Phong shading does not.



Universitat de Girona

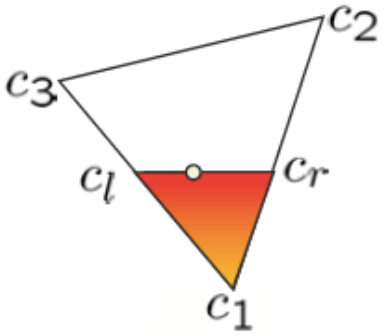
Shading and texturing...

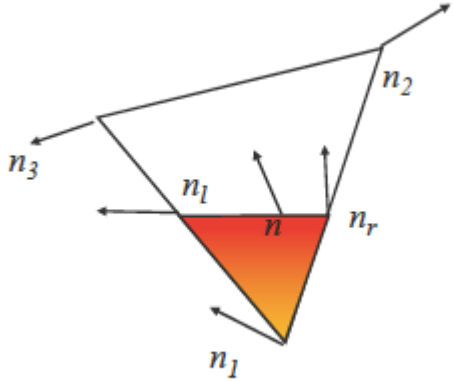
- How to handle ?

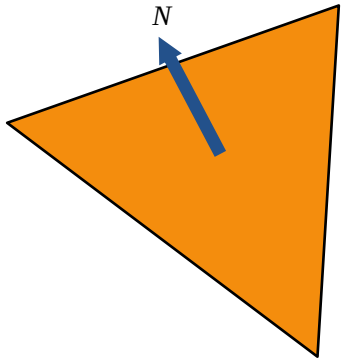
Universitat de Girona

Shading Models: Summary

Gouraud	Phong	Flat
Normal Per Vertex	Interpolated normal for each point across face	Normal per face
Color per vertex, interpolated across face	Color calculated per pixel	Color calculated per face
Faster then phong Bad specular	Costly (not really any more)	Fast, good for distant objects







Universitat de Girona

ILLUMINATION IN WEBGL

Universitat de Girona

Implementing a lighting model

- WebGL has no *built-in* support for lighting
  - WebGL leaves you to do everything yourself
  - Old OpenGL use to have
    - glNormal, glMaterial, glLight …

- Where to implement a lighting model?
- Three chooses:
  - In the Application
  - As a Vertex shader
  - As a Fragment shader

Universitat de Girona

Application model

- Compute all the geometry and colors on CPU
  - Then send color to shader
- Ok for a static scene
- But inefficient for dynamic
  - If a cube rotate
  - Change lighting vectors, thus change lighting components

Universitat de Girona

Shaders: Per-vertex vs. Per-fragment

- Working out the lighting on a **per-vertex** basis
  - Light on the pixels between vertices by linear interpolation
    - For planar Surface (like a cube) it is OK
  - For curved surfaces, where you want to calculate the effects of lighting on every pixel independently
    - use ***per-fragment*** *(or per-pixel)* ***lighting***, which gives much better effects.

Universitat de Girona

Phong Model in shaders

- Implementing a function in GLSL

```
// Ka, Kd, Ks, alpha, La, Ld y Ls son variables uniformes
// N, L y V se asumen normalizados
vec4 phong (vec3 N, vec3 L, vec3 V) {

  vec4  ambient  = Ka * La;
  vec4  diffuse  = vec4(0.0);
  vec4  specular = vec4(0.0);
  float NdotL    = dot(N,L);
  if (NdotL > 0.0)
  {
    vec3  R       = reflect(-L,N);
    float RdotV_n = pow(max(0.0, dot(R,V)), alpha);
    diffuse  = NdotL   * (Ld * Kd);
    specular = RdotV_n * (Ls * Ks);
  }
  return (ambient + diffuse + specular);
}
```

Universitat de Girona

Gouraud Shading

```
#version 300 es // Vertex Shader ----------------------------------------
uniform mat4  projectionMatrix, modelViewMatrix;
uniform mat3  normalMatrix;
uniform vec4  Ka, Kd, Ks;          // material
uniform float alfa;
uniform vec4  Lp, La, Ld, Ls;      // fuente de luz
in      vec3  vertexPosition, vertexNormal;
out     vec4  myColor;

void main()
{
  vec4 ecPosition = modelViewMatrix * vec4(vertexPosition,1.0);
  vec3 N          = normalize(normalMatrix * vertexNormal);
  vec3 L          = normalize(vec3(Lp - ecPosition));
  vec3 V          = normalize(vec3(-ecPosition));

  myColor = phong(N, L, V);
  gl_Position = projectionMatrix * ecPosition;
}

#version 300 es // Fragment Shader --------------------------------------
precision mediump float;
in  vec4 myColor;
out vec4 fragmentColor;

void main()
{
  fragmentColor = myColor;
}
```

Universitat de Girona

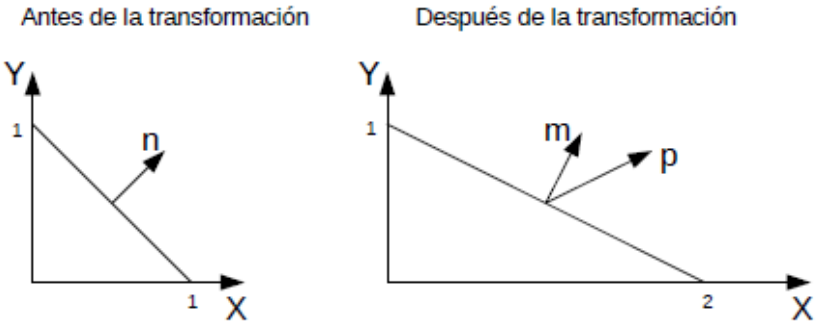
Phong shading

```
#version 300 es // Vertex Shader ----------------------------------------
uniform mat4  projectionMatrix, modelViewMatrix;
uniform mat3  normalMatrix;
in      vec3  vertexPosition, vertexNormal;
out     vec3  Position, N;

void main()
{
  vec4 ecPosition = modelViewMatrix * vec4(vertexPosition,1.0);
  Position = vec3(ecPosition);
  N        = normalize (normalMatrix * vertexNormal);

  gl_Position = projectionMatrix * ecPosition;
}

#version 300 es // Fragment Shader --------------------------------------
uniform vec4  Ka, Kd, Ks;        // material
uniform float alfa;
uniform vec4  La, Ld, Ls, Lp;    // fuente de luz
in      vec3  Position, N;
out     vec4 fragmentColor;

void main() {
  n = normalize(N);
  L = vec3 (normalize(Lp - Position));
  V = normalize (vec3(-Position));
  fragmentColor = phong (n, L, V);
}
```

Universitat de Girona

Normal Transformation

- Non-uniform scale do not preserve Normals



- Solution:

$$N = (M^{-1})^T$$

Universitat de Girona

Normals

```
normalFromMat4(mat3.create(), modelViewMatrix);
```

–Calculates a 3x3 normal matrix (transpose inverse) from the 4x4 matrix

```
var exampleCube = {  // 24 vértices, 12 triángulos

 "vertices" : [-0.5, -0.5,  0.5,   0.0,  0.0,  1.0,
               0.5, -0.5,  0.5,   0.0,  0.0,  1.0,
               0.5,  0.5,  0.5,   0.0,  0.0,  1.0,
              -0.5,  0.5,  0.5,   0.0,  0.0,  1.0,
               0.5, -0.5,  0.5,   1.0,  0.0,  0.0,
               0.5, -0.5, -0.5,   1.0,  0.0,  0.0,
               0.5,  0.5, -0.5,   1.0,  0.0,  0.0,
               0.5,  0.5,  0.5,   1.0,  0.0,  0.0,
               0.5, -0.5, -0.5,   0.0,  0.0, -1.0,
              -0.5, -0.5, -0.5,   0.0,  0.0, -1.0,
              -0.5,  0.5, -0.5,   0.0,  0.0, -1.0,
               0.5,  0.5, -0.5,   0.0,  0.0, -1.0,
              -0.5, -0.5, -0.5,  -1.0,  0.0,  0.0,
              -0.5, -0.5,  0.5,  -1.0,  0.0,  0.0,
              -0.5,  0.5,  0.5,  -1.0,  0.0,  0.0,
              -0.5,  0.5, -0.5,  -1.0,  0.0,  0.0,
              -0.5,  0.5,  0.5,   0.0,  1.0,  0.0,
               0.5,  0.5,  0.5,   0.0,  1.0,  0.0,
               0.5,  0.5, -0.5,   0.0,  1.0,  0.0,
              -0.5,  0.5, -0.5,   0.0,  1.0,  0.0,
              -0.5, -0.5, -0.5,   0.0, -1.0,  0.0,
               0.5, -0.5, -0.5,   0.0, -1.0,  0.0,
               0.5, -0.5,  0.5,   0.0, -1.0,  0.0,
              -0.5, -0.5,  0.5,   0.0, -1.0,  0.0],

 "indices" :  [  0,  1,  2,  0,  2,  3,
                 4,  5,  6,  4,  6,  7,
                 8,  9, 10,  8, 10, 11,
                12, 13, 14, 12, 14, 15,
                16, 17, 18, 16, 18, 19,
                20, 21, 22, 20, 22, 23]

};
```

Universitat de Girona

Materials

```
var Gold = {
  "mat_ambient" : [ 0.24725, 0.1995, 0.0745 ],
  "mat_diffuse" : [ 0.75164, 0.60648, 0.22648 ],
  "mat_specular": [ 0.628281, 0.555802, 0.366065 ],
  "alpha"       : [ 51.2 ]
};
```

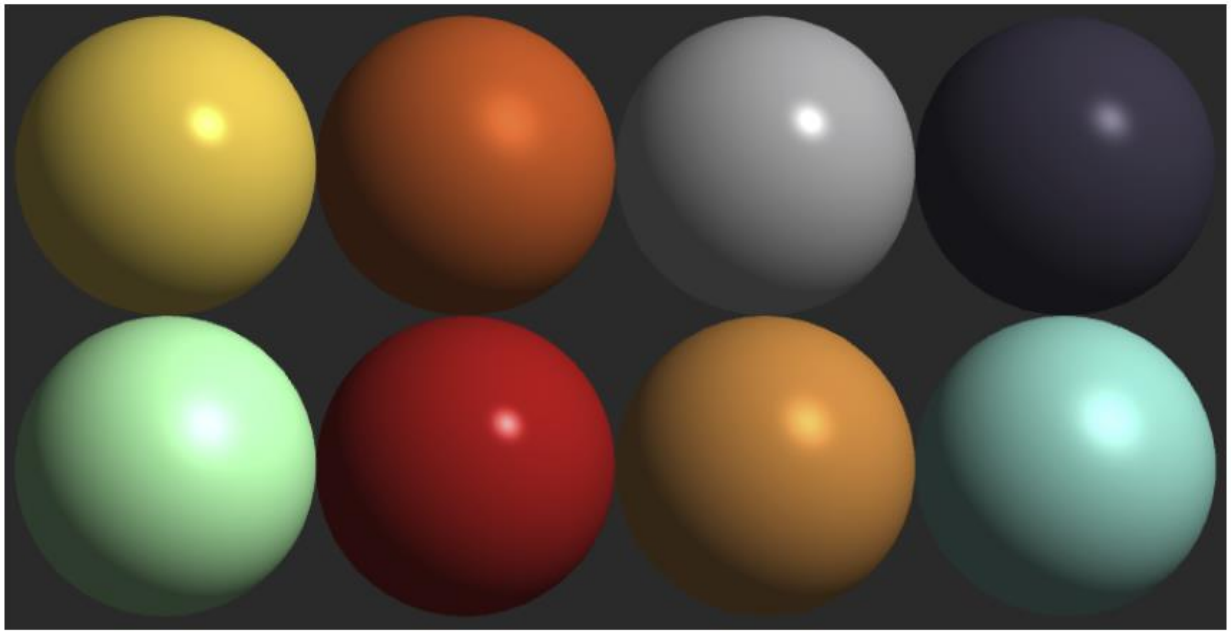


Figura: Ejemplos de materiales, de izquierda a derecha y de arriba a abajo: Oro, Cobre, Plata, Obsidiana, Jade, Rubí, Bronce, Turquesa.

Universitat de Girona

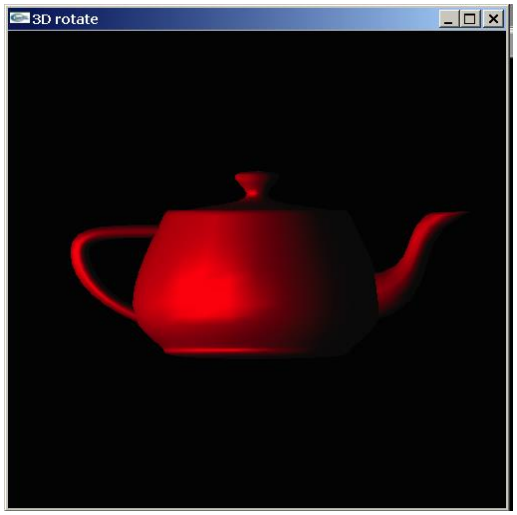
Examples

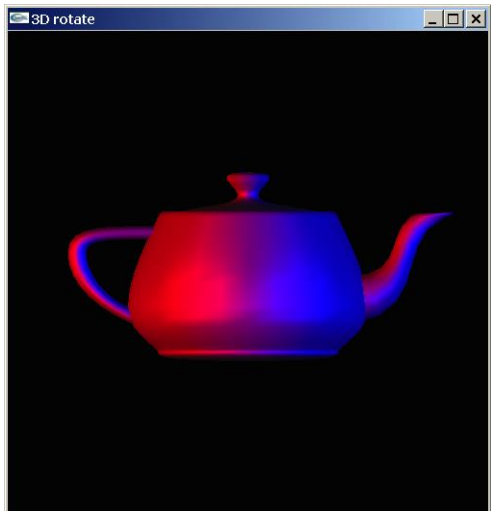
- Phong with Gouraud
- Phong with Phong
- Including OBJ with illumination

Universitat de Girona

Multiple light sources

- The total reflection at p is the
  sum of all contributed intensities from all sources
- WebGL allows us to define several light sources (as uniforms)





Universitat de Girona

Non-photorealistic shading: Toon Shading

- Use shaders for cartoon-like effects
- Examples:
  - Paint with only two colors
  - Emphazise the edge of an object
- Paint with red and yellow
  - Define

```
vec4 color1 = vec4(1.0, 1.0, 0.0, 1.0); // yellow
vec4 color2 = vec4(1.0, 0.0, 0.0, 1.0); // red
```

  - Switch between the colors based on the magnitude of the diffuse color. Using the light and normal vectors, we could assign colors as:

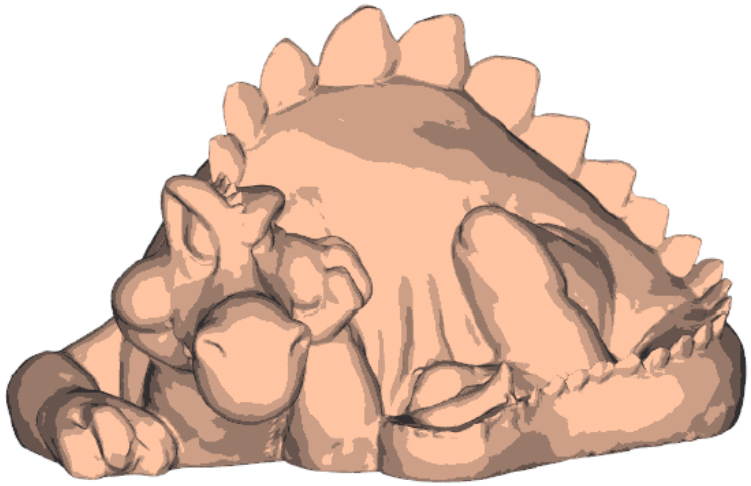
```
gl_FragColor = (dot(lightv, norm)) > 0.5 ? color1 : color2);
```

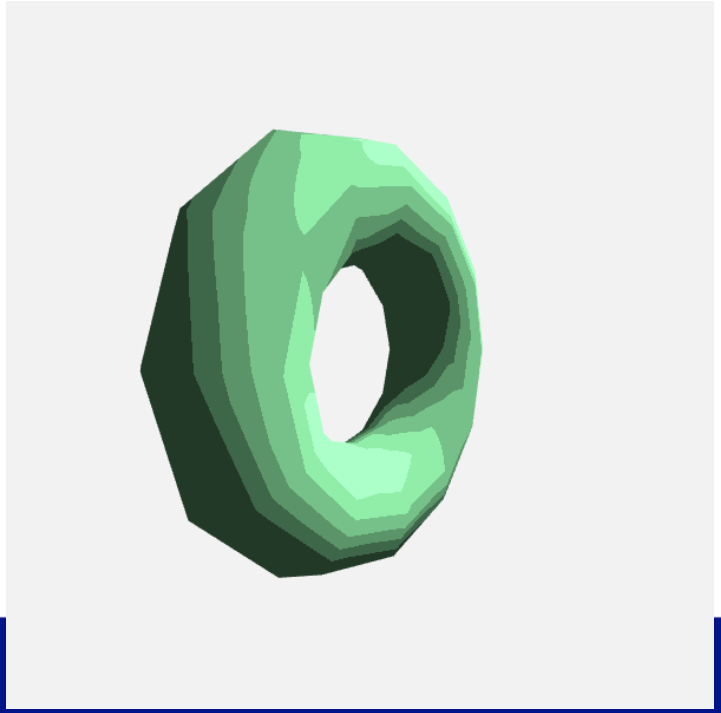
Universitat de Girona

Toon shading II

- Another algorithm:
  - Reduce the difuse component to only a discrete number of levels

```
diffuse = (ceil (NdotL * levels) / levels) * (Light.Ld * Material.Kd);
```



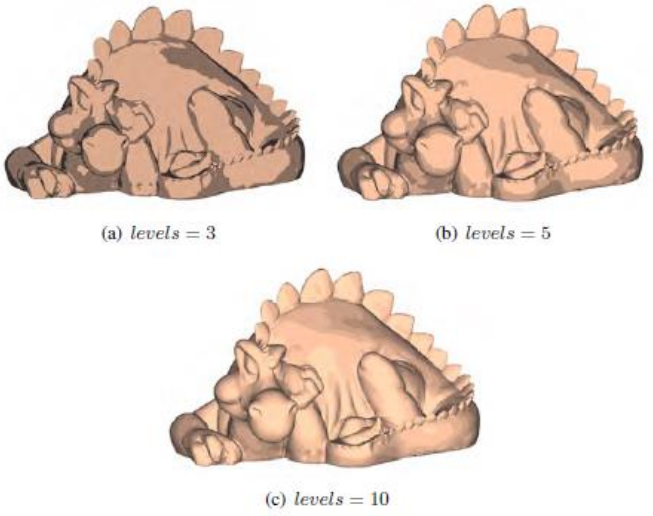


Universitat de Girona

Shader funtion

- Example toonShading

```
vec3 toonShading (vec3 N, vec3 L) {

    vec3  ambient     = Material.Ka * Light.La;
    float NdotL       = max(0.0, dot (N,L));
    float levels      = 3.0;
    float scaleFactor = 1.0 / levels;

    vec3 diffuse  = ceil(NdotL * levels) * scaleFactor *
                    (Light.Ld * Material.Kd);

    return (ambient + diffuse);

  }

void main() {

    vec3 n = normalize(N);
    vec3 L = normalize(Light.Position - ec);

    fragmentColor = vec4(toonShading(n,L), 1.0);

}
```



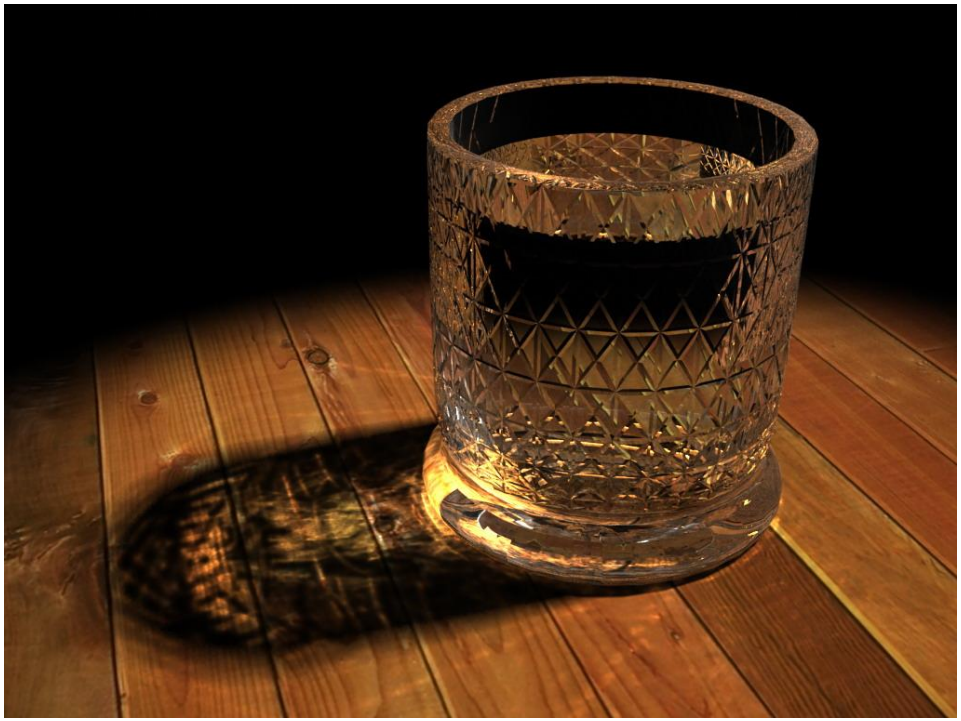
(a) *levels* = 3 (b) *levels* = 5

(c) *levels* = 10

Universitat de Girona

More advanced rendering

Universitat de Girona



Universitat de Girona

Next

- Shadows
- Textures