



Universidade de Aveiro
Departamento de Electrónica,
Telecomunicações e Informática

Introduction to Computer Graphics

main concepts and methods - III




(Wikipedia)

Topics

- Computer Graphics main tasks
- 2D and 3D visualization
- Geometric transformations
- Projections
- Illumination and shading

CG Main Tasks

- Modeling
 - Construct individual models / objects
 - Assemble them into a 2D or 3D scene (using transformations)
- Rendering
 - Generate final images:
 - How is the scene illuminated? 
 - What are the materials of the objects?
 - Where is the observer? How is he/she looking at the scene?
- Animation
 - Static vs. dynamic scenes
 - Movement and / or deformation

Lights and materials

- Types of light sources
 - Point vs distributed light sources
 - Spot lights
 - Near and far sources
 - Color properties
- Material properties
 - Absorption: color properties
 - Scattering: diffuse and specular
 - Transparency



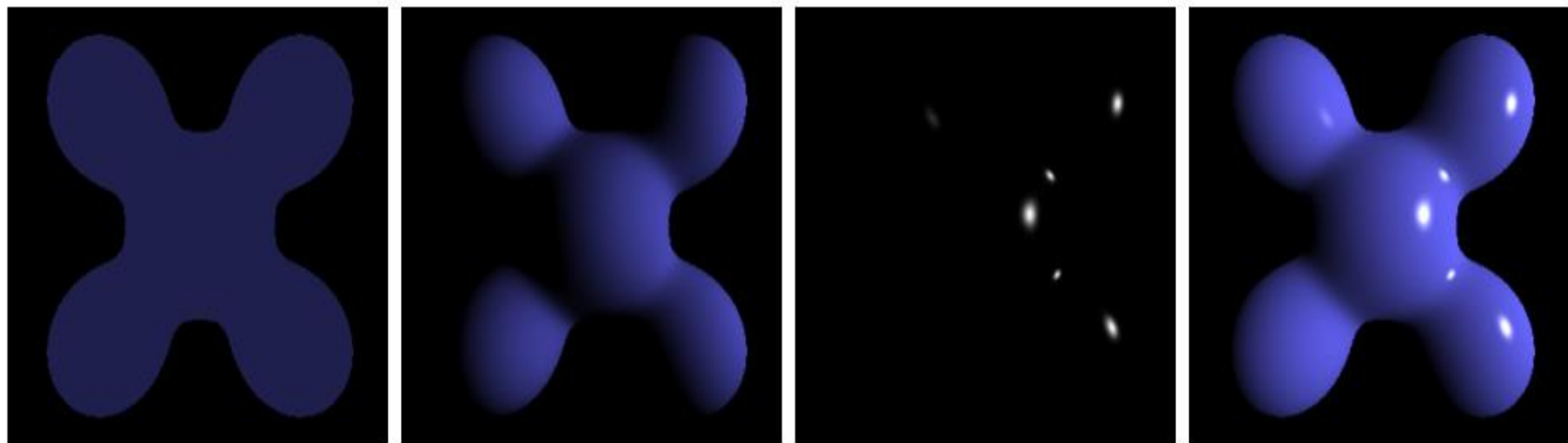
Lighting

- Compute **surface color** based on
 - Type and number of **light sources**
 - **Illumination model**
 - Phong: ambient + diffuse + specular components
 - Reflective surface properties
 - Atmospheric effects
 - Fog, smoke
- **Polygons** making up a model surface **are shaded**
 - Realistic representation

Phong reflection model

Empirical model of the local illumination of points on a surface

It describes the way a surface reflects light as a combination of the **diffuse reflection** of rough surfaces with the **specular reflection** of shiny surfaces and a component of **ambient light**



Ambient

+

Diffuse

+

Specular

=

Phong Reflection

(Wikipedia)

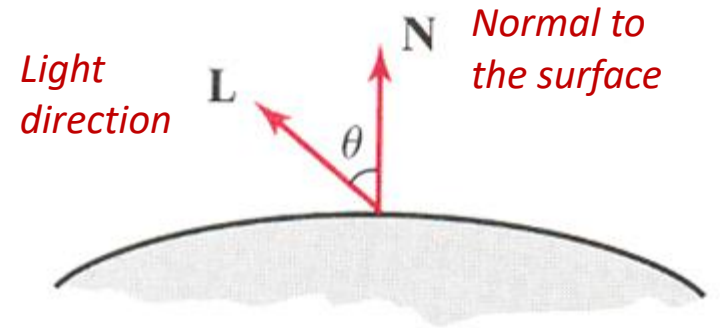
Phong Model – Ambient illumination

- Constant illumination component for each model
- Independent from viewer position or object orientation
- Take only material properties into account



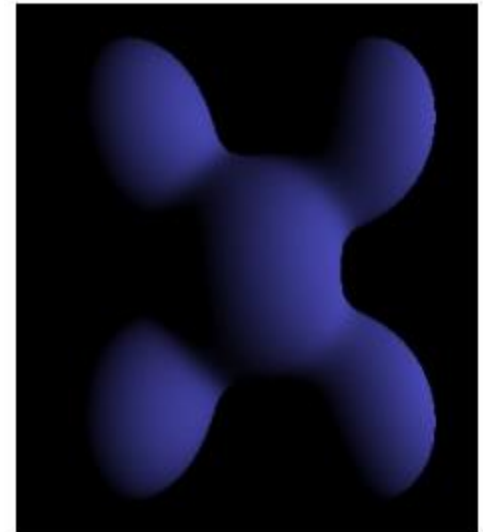
Phong Model – Diffuse reflection

$$I_{l,\text{diff}} = \begin{cases} k_d I_l (\mathbf{N} \cdot \mathbf{L}), & \mathbf{N} \cdot \mathbf{L} > 0 \\ 0.0, & \mathbf{N} \cdot \mathbf{L} \leq 0 \end{cases}$$



(Hearn & Baker, 2004)

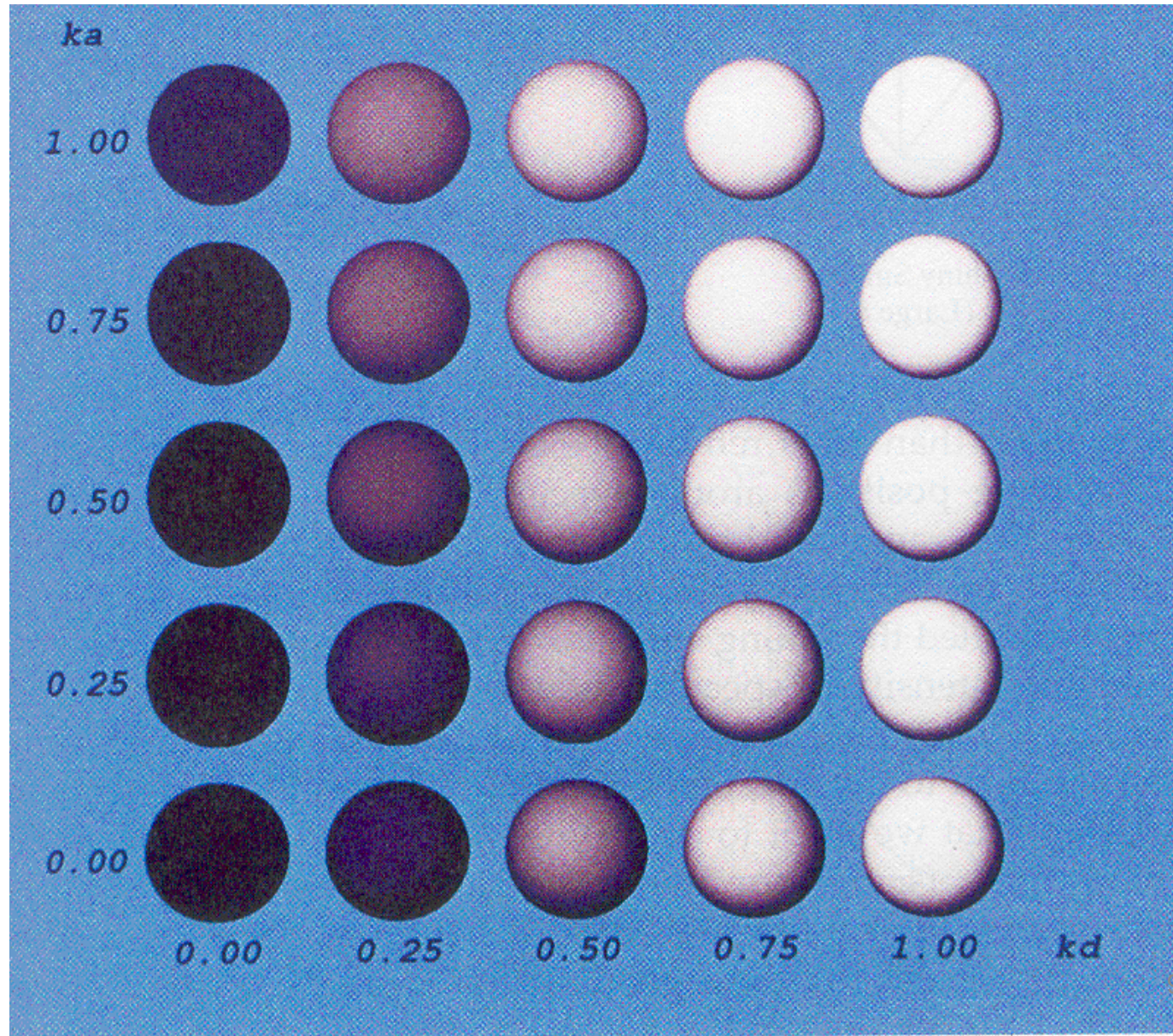
- Model surface is an ideal diffuse (Lambertian) reflector (“matte”)
 - What does that mean ?
- Independence from viewer position (reflected light is the same in all directions)
- Light reflected depends on Θ (angle between the vectors \mathbf{N} and \mathbf{L})



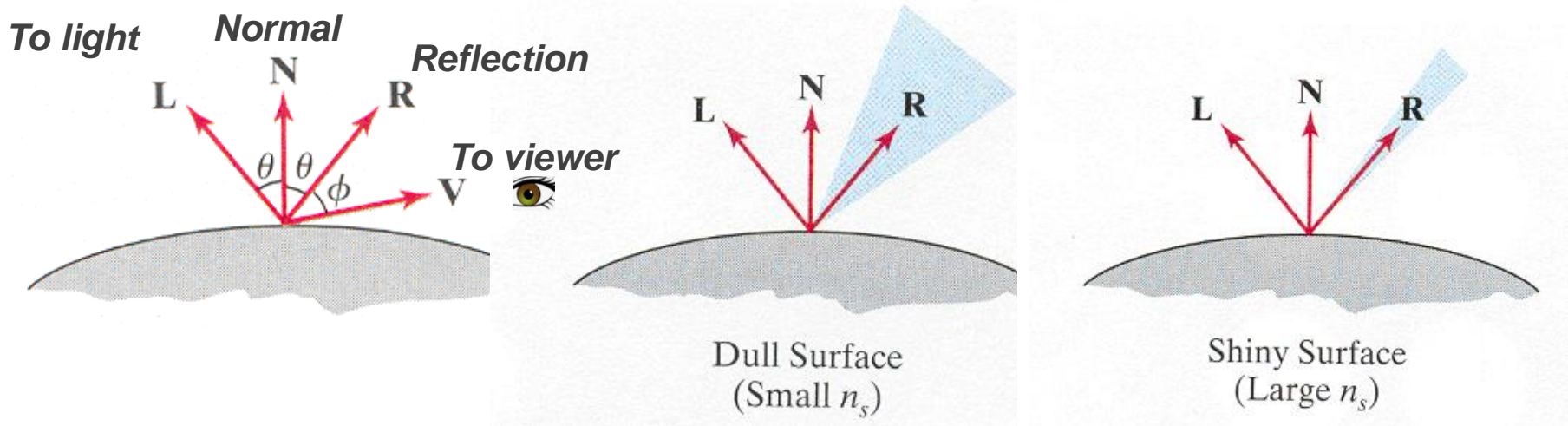
Phong Model

k_a – ambient

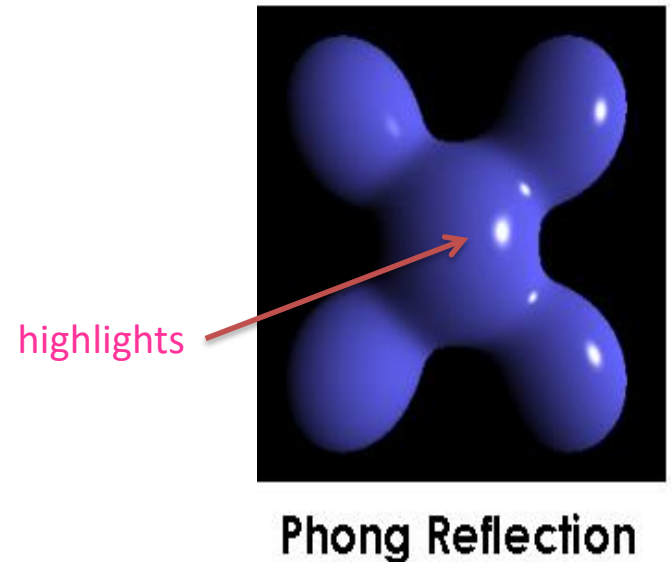
K_d - diffuse

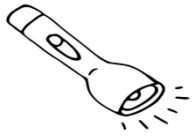


Phong Model – Specular reflection

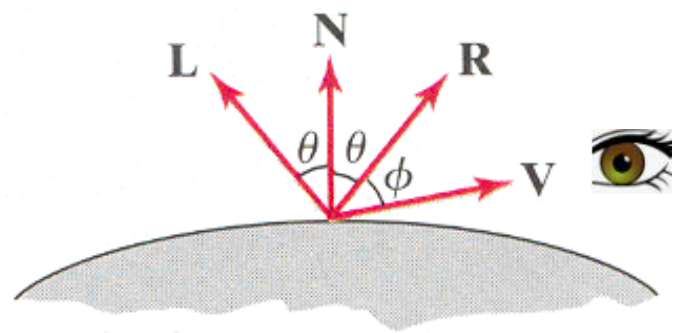


- Important for shiny model surfaces
 - How to model **shininess** ?
- Take into account **viewer position**
- Unit vectors (L, N, R, V)





Phong Model – Specular reflection



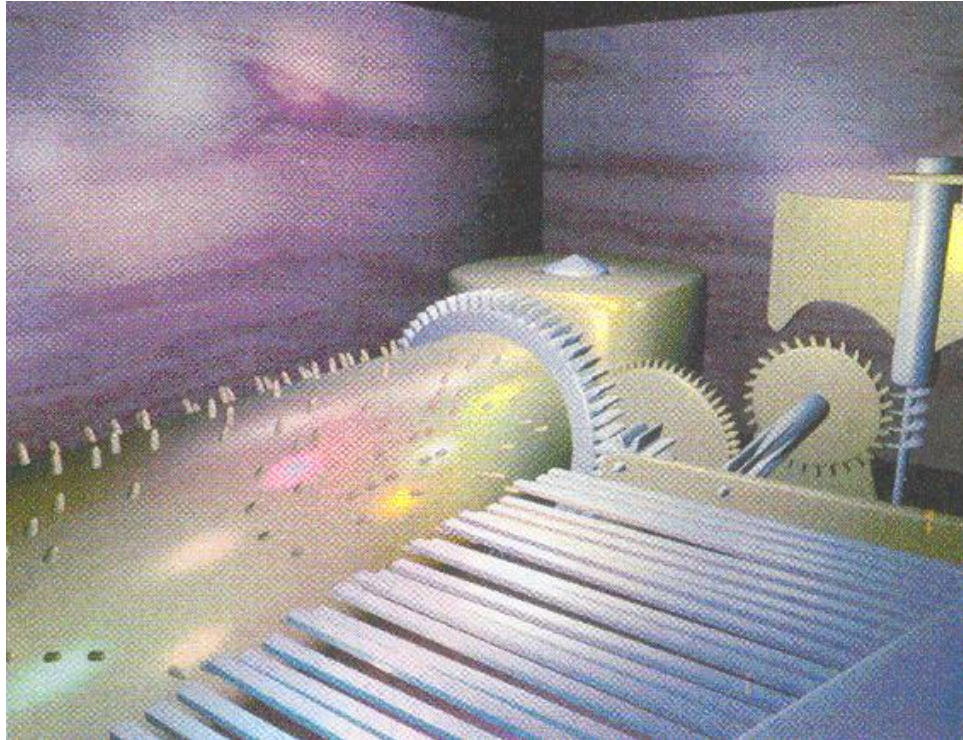
[OpenGL – The Red Book]

$$I_{l,spec} = \begin{cases} k_s I_l (\mathbf{V} \cdot \mathbf{R})^{n_s}, & \text{if } \mathbf{V} \cdot \mathbf{R} > 0 \\ 0.0, & \text{if } \mathbf{V} \cdot \mathbf{R} < 0 \end{cases} \quad \text{and } \mathbf{N} \cdot \mathbf{L} > 0$$

or $\mathbf{N} \cdot \mathbf{L} \leq 0$

n_s – parameter controlling the shininess

More than one light source



For n light sources:

$$I = k_a I_a + \sum_{l=1}^n I_l [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{N} \cdot \mathbf{H})^{n_s}]$$

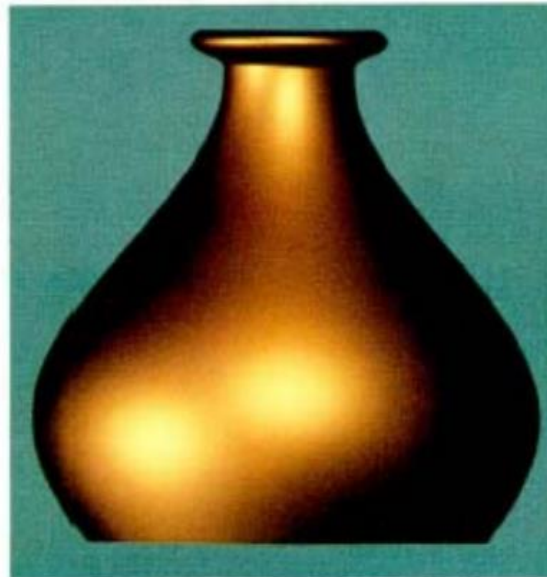
Other illumination models

- The Phong illumination model is much used, but there are other models that produce more realistic results (e.g. Cook Torrance) ...

Plastic-looking copper rendered using Phong model



A Copper Vase with a more metallic appearance



<https://www.cs.unc.edu/xcms/courses/comp770-s07/Lecture11.pdf>

Three.js

- Cross-browser JavaScript library/API used to create and display animated 3D computer graphics in a web browser.
- Uses WebGL

three.js ^{r87}

featured projects

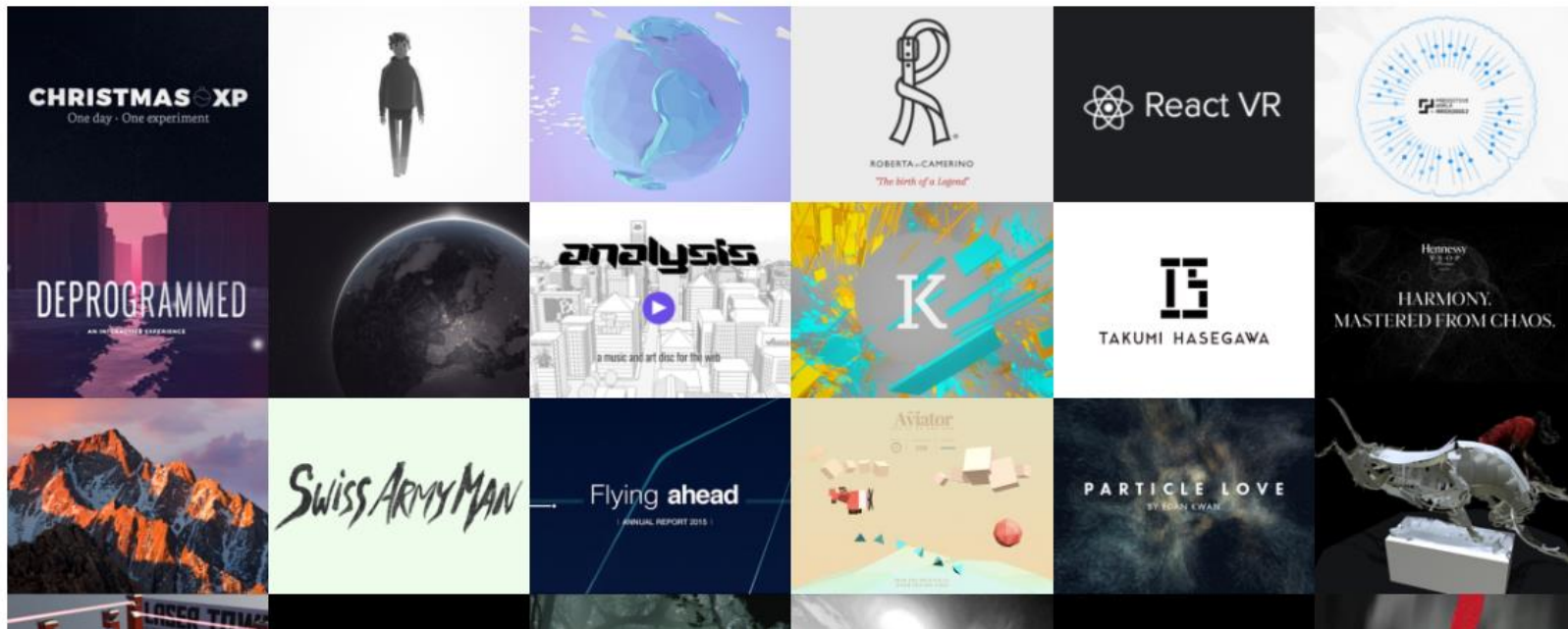
[submit project](#)

[documentation](#)
[examples](#)


[download](#)

[source code](#)
[questions](#)
[forum](#)
[irc](#)
[slack](#)
[google+](#)

[editor](#)



Interactive 3D Graphics
Taught by Eric Haines



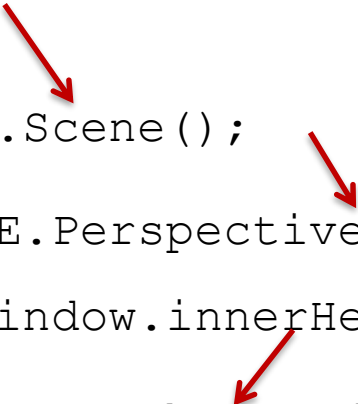
UDACITY

<https://threejs.org/>

Thee.js first example

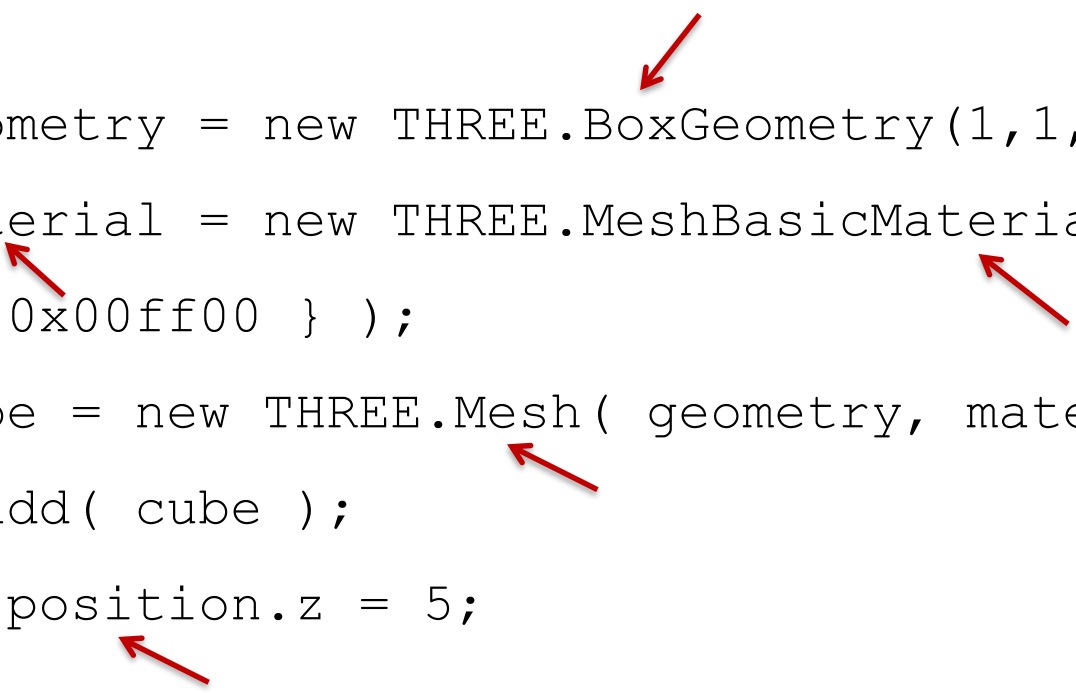
1. Defining the scene, the camera and where the scene is rendered

```
var scene = new THREE.Scene();  
var camera = new THREE.PerspectiveCamera( 75,  
window.innerWidth / window.innerHeight, 0.1, 1000 );  
var renderer = new THREE.WebGLRenderer();  
renderer.setSize( window.innerWidth, window.innerHeight );  
document.body.appendChild( renderer.domElement );
```




2.Creating an object and camera position

```
var geometry = new THREE.BoxGeometry(1,1,1);  
var material = new THREE.MeshBasicMaterial( {  
color: 0x00ff00 } );  
var cube = new THREE.Mesh( geometry, material );  
scene.add( cube );  
camera.position.z = 5;
```




3. Scene rendering

```
function render() {  
  requestAnimationFrame(render);  
  renderer.render(scene, camera);  
}
```



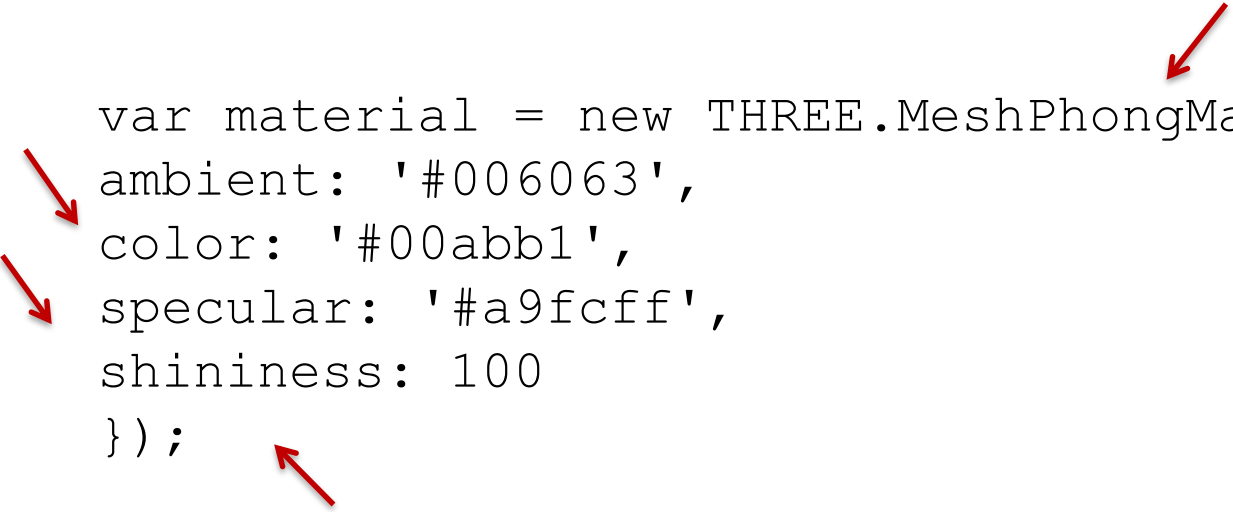
4. Scene animation

```
render();  
cube.rotation.x += 0.1;  
cube.rotation.y += 0.1;
```



Adding lights and shading

```
var material = new THREE.MeshPhongMaterial({  
  ambient: '#006063',  
  color: '#00abb1',  
  specular: '#a9fcff',  
  shininess: 100  
});
```

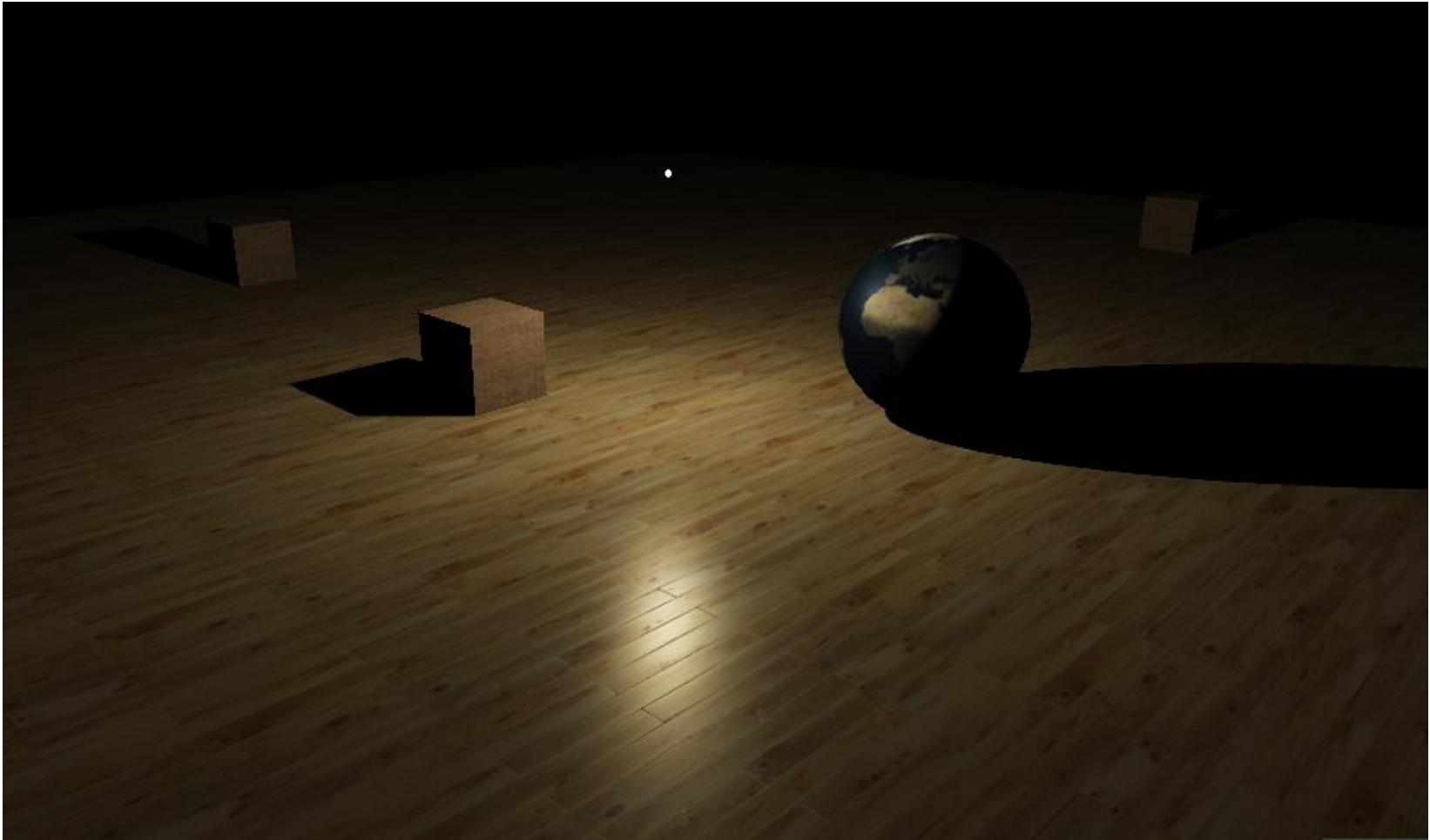


Illumination models - example



https://threejs.org/examples/#webgl_lightprobe

Physically accurate lighting

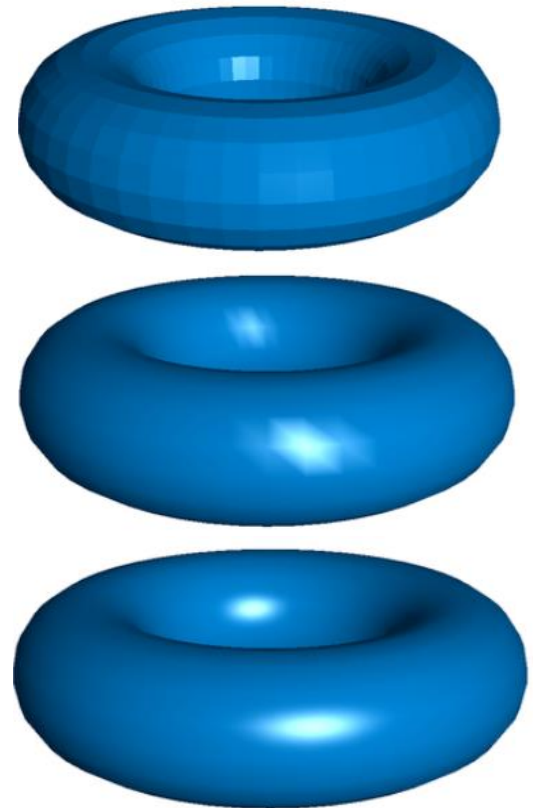


https://threejs.org/examples/#webgl_lights_physical

Illumination and shading

Computing the colour that should be assigned to each pixel based on the geometry of the scene, the materials and light to create the illusion of depth

- Flat-shading
- Gouraud shading
- Phong shading



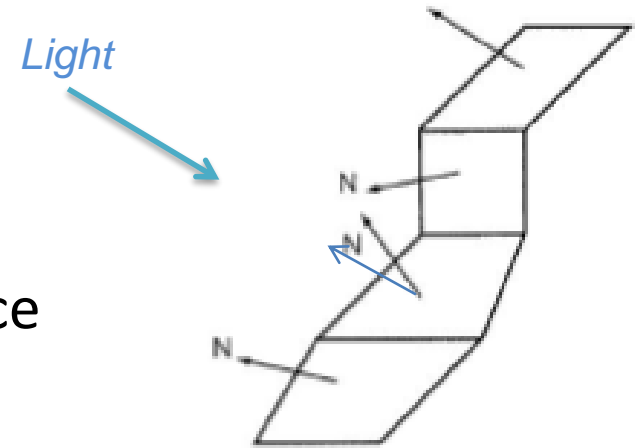
Illumination models - example



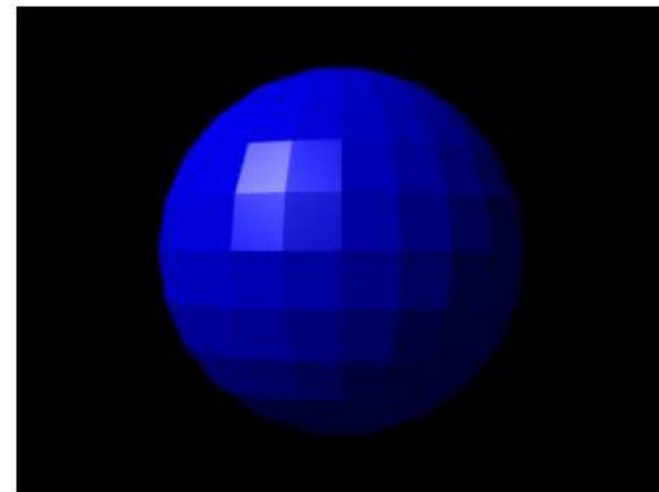
https://threejs.org/examples/#webgl_geometry_teapot

Flat shading

- For each polygon:
- Applies the illumination model just once
- All pixels have the same color
- smooth objects seem “blocky”
- It is fast



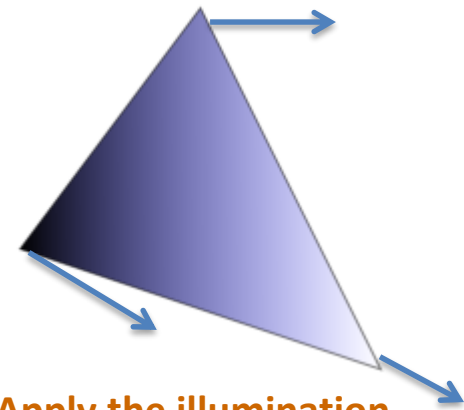
Apply the illumination model once per face



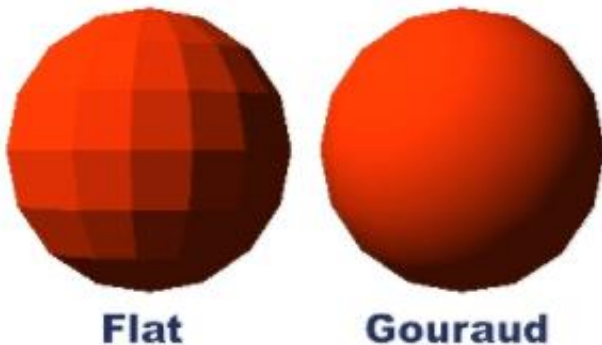
FLAT SHADING

Gouraud shading

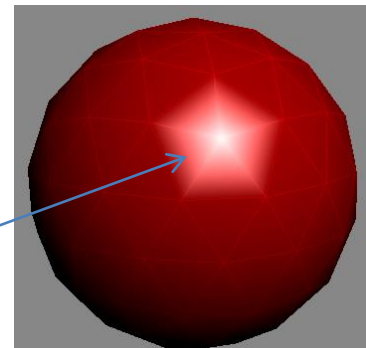
- For each triangle:
 - Applies the illumination model at each vertex
 - Interpolates color to shade each pixel
-
- It provides better results than flat shading
 - But highlights are not rendered correctly



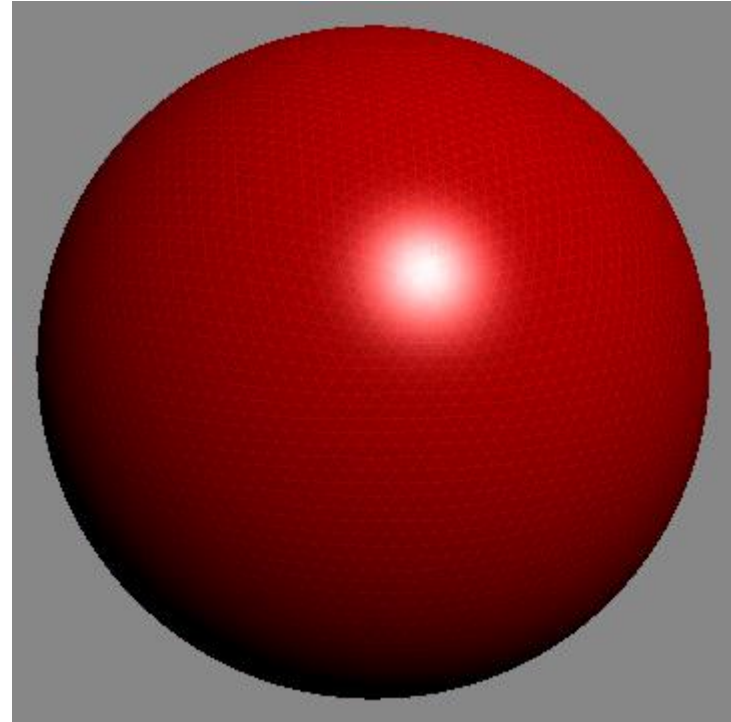
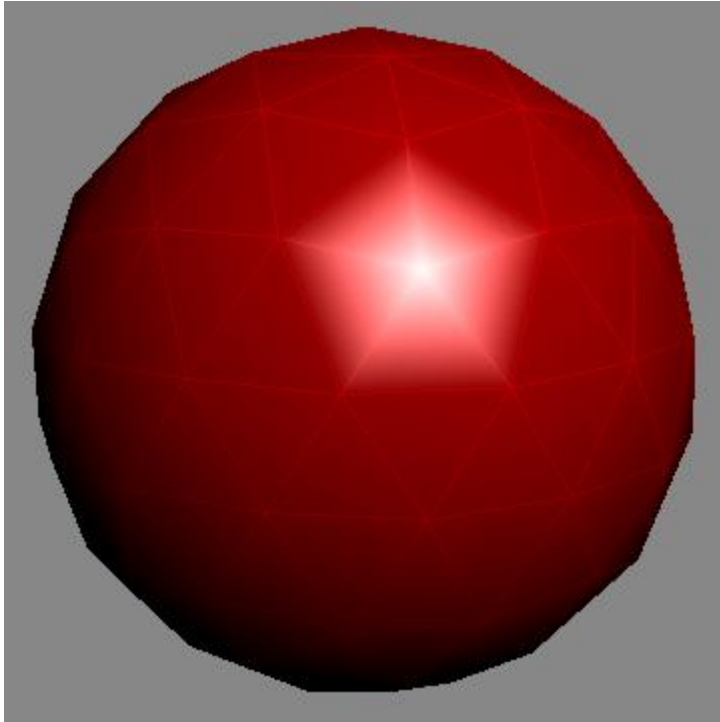
Apply the illumination model at vertices and interpolates color



highlight



Gouraud shading

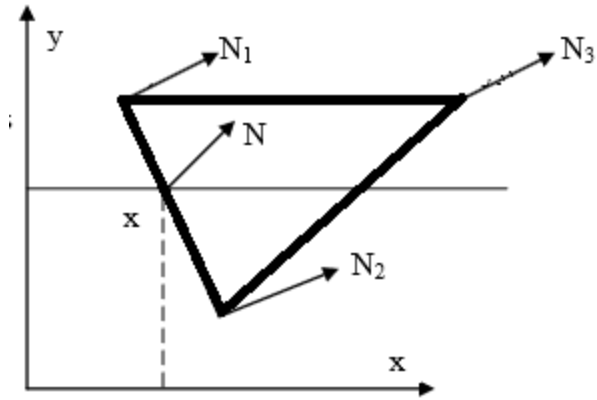


(Wikipedia)

Same object with a much higher n. of faces
(it is not an efficient solution to improve realism)

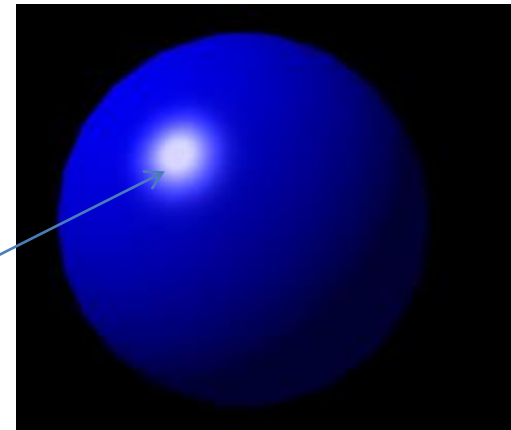
Phong shading

- Interpolates normal vectors across rasterized polygons
- Applies illumination model and computes pixel colors based on the interpolated normals
- It provides better results than Gouraud shading
- But is more time consuming

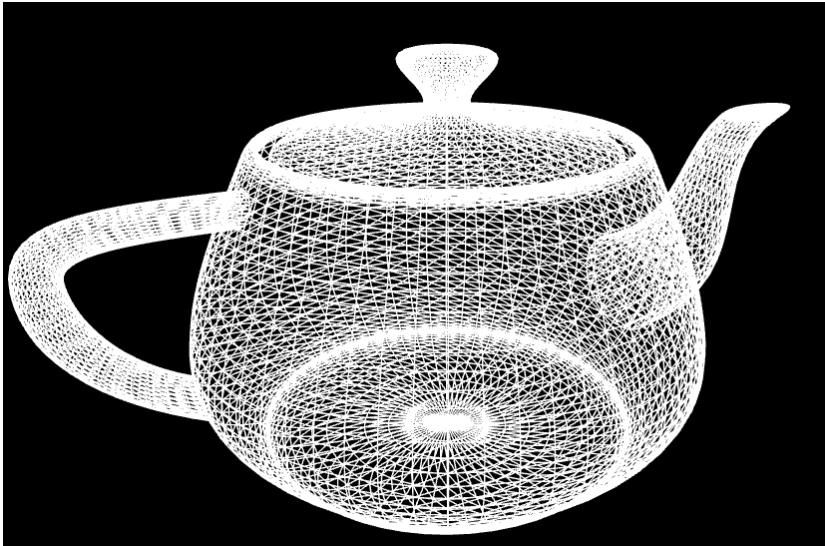


Interpolates normals and apply illumination model

highlight
(Wikipedia)



Wire frame



Flat shading



Gouraud shading



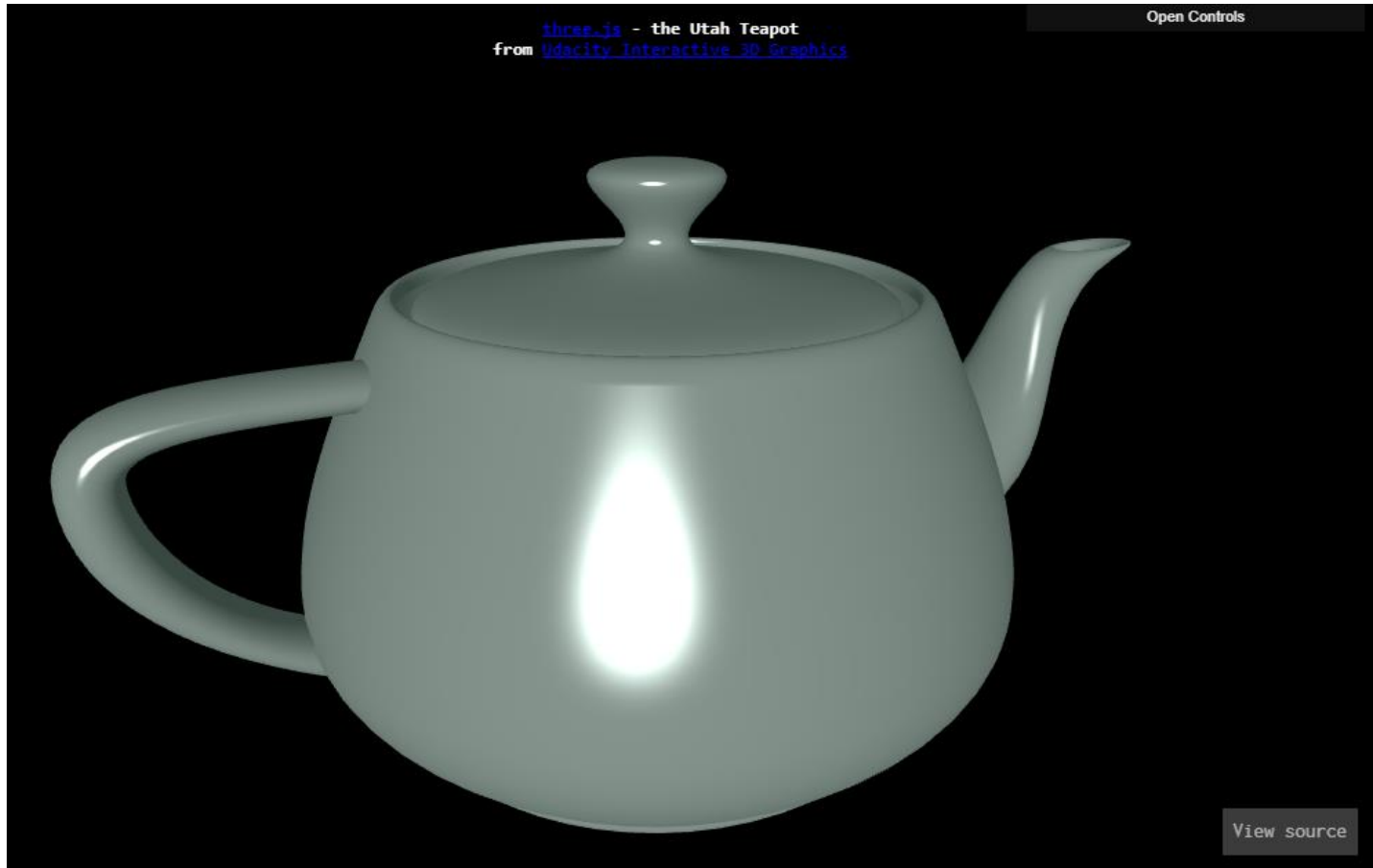
Phong shading



https://threejs.org/examples/#webgl_geometry_teapot

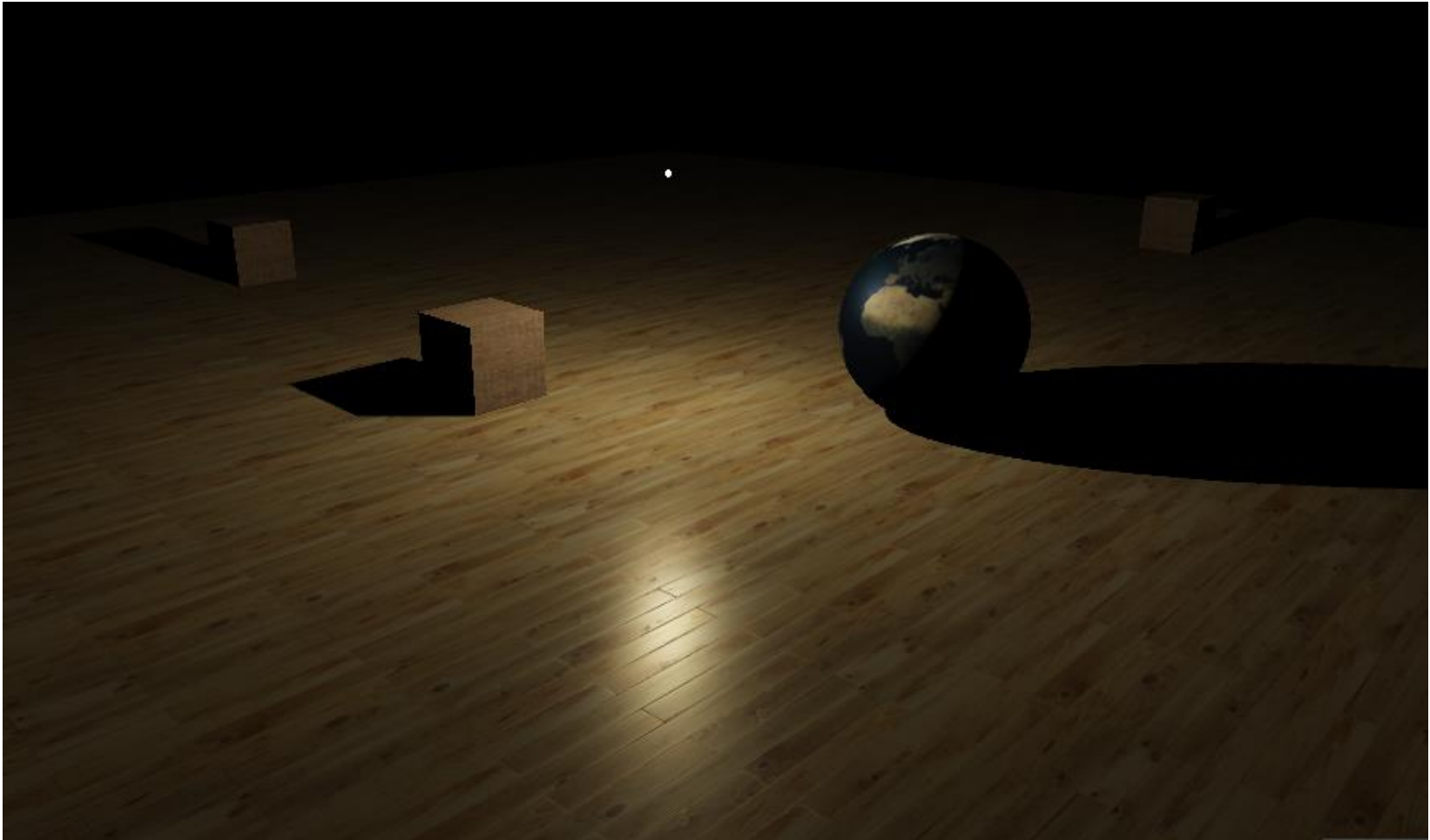
Test these examples in three.js:

Material and light properties



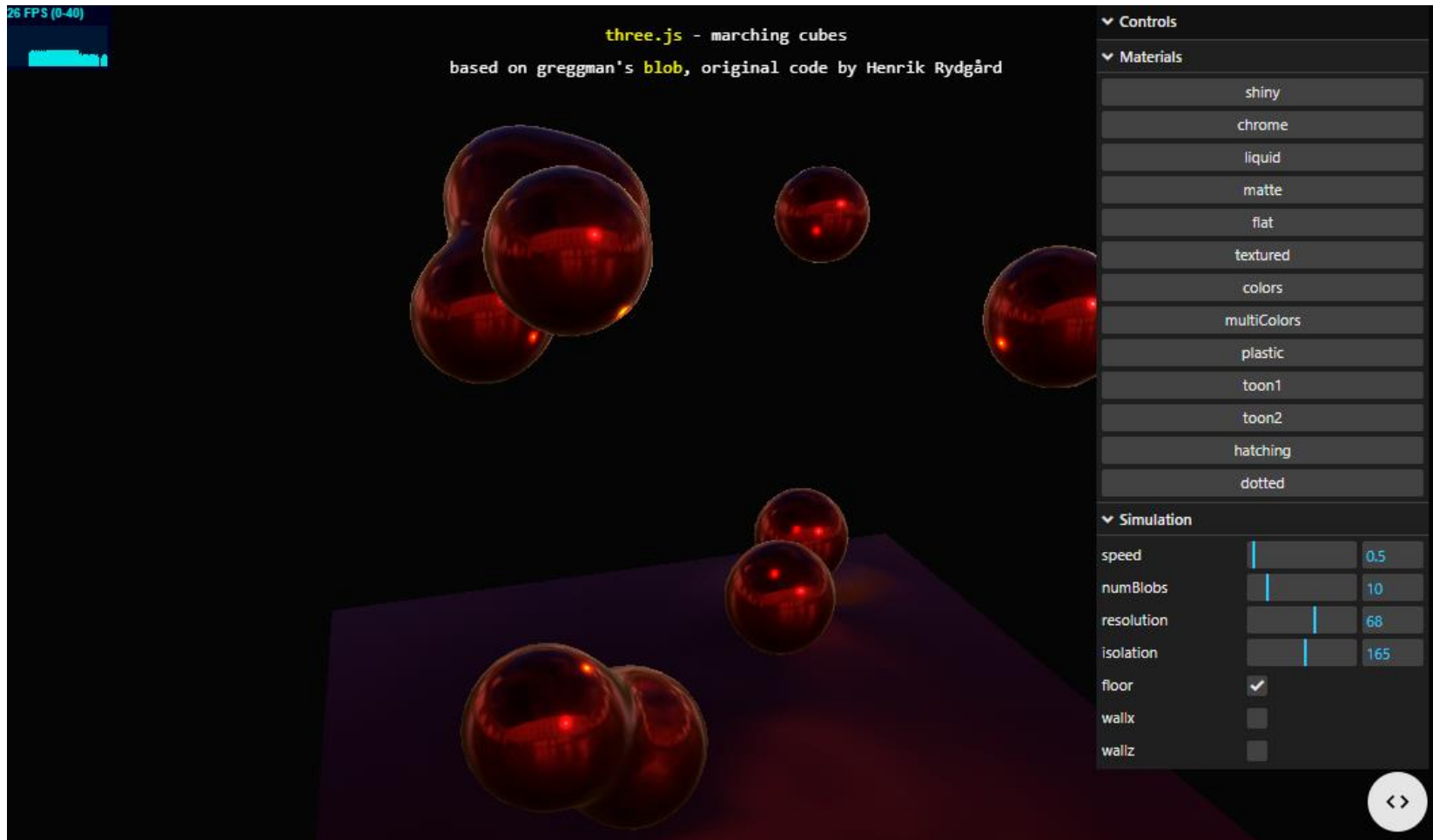
https://threejs.org/examples/#webgl_geometry_teapot

Physically accurate lighting



https://threejs.org/examples/#webgl_lights_physical

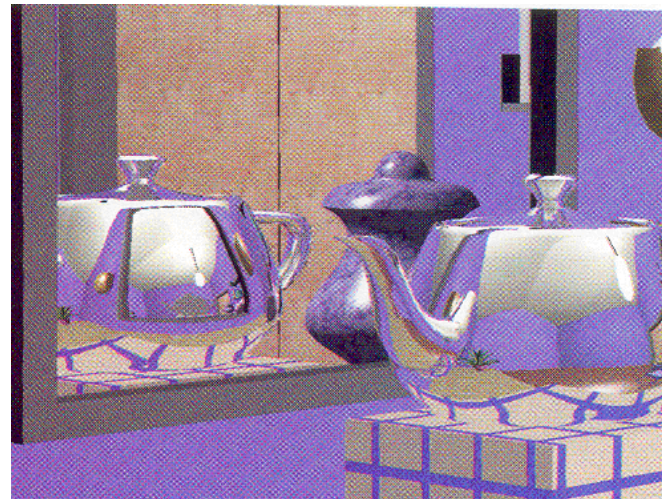
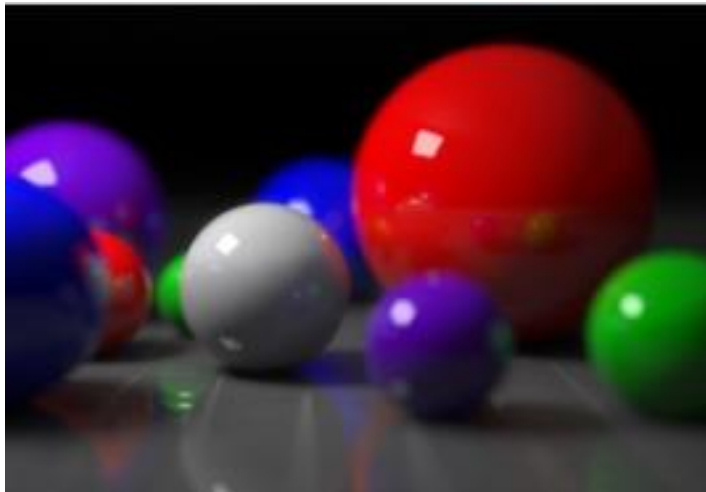
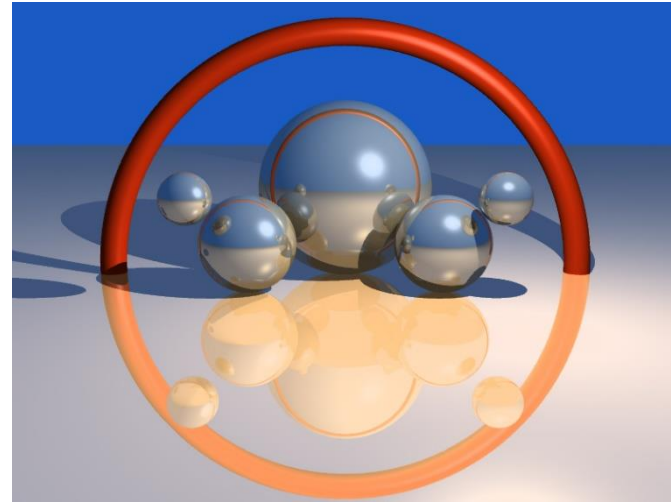
Materials



https://threejs.org/examples/#webgl_marchingcubes

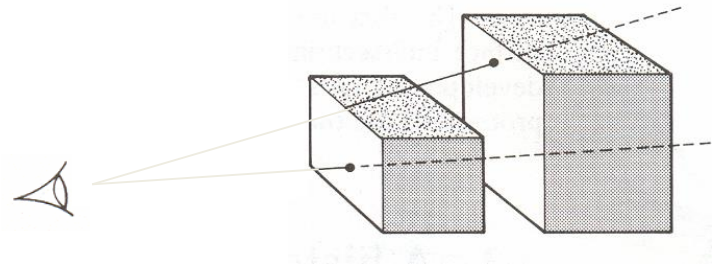
Global illumination methods

Producing more realistic images than scanline rendering



Global illumination

- If you consider the line of sight from a pixel on the viewing plane to the scene, it is possible to determine that objects are intersected



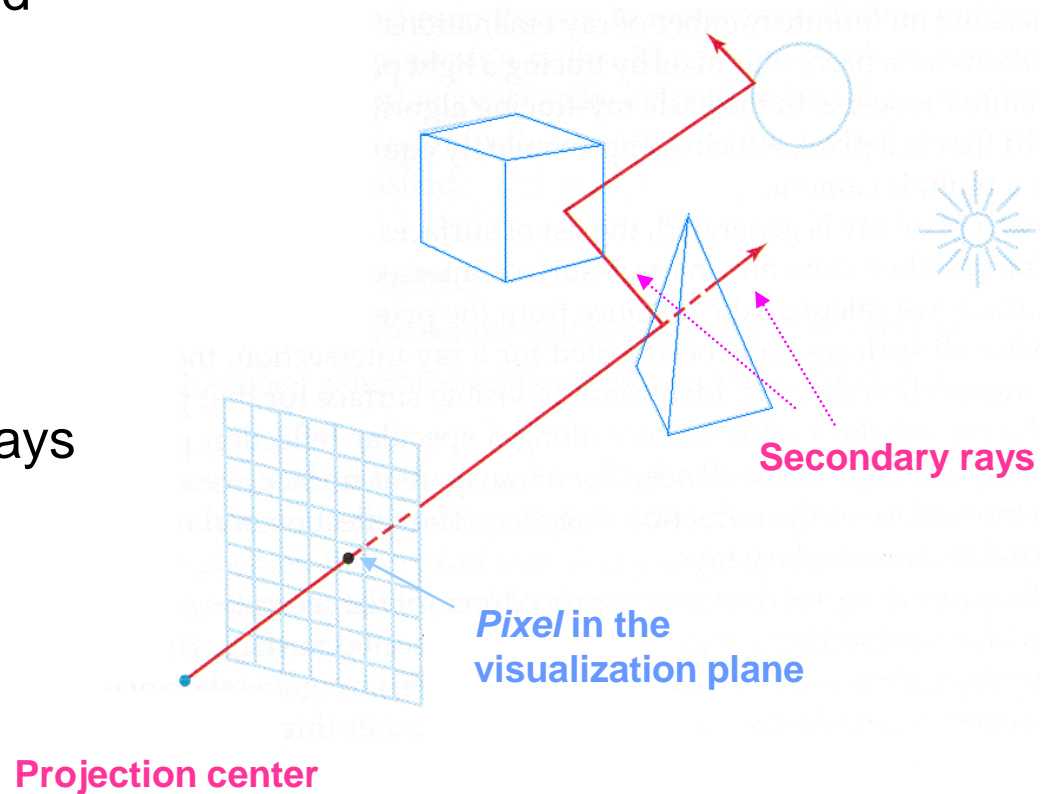
- It is *ray casting*
- It is based on the geometrical optics that determine the light rays paths
- When using perspective projection, rays diverge from the projection center, passing through a pixel and continue to the scene

- **Ray tracing** - Whenever there are secondary rays:
 - reflected
 - transmitted

The number of secondary rays may be controlled

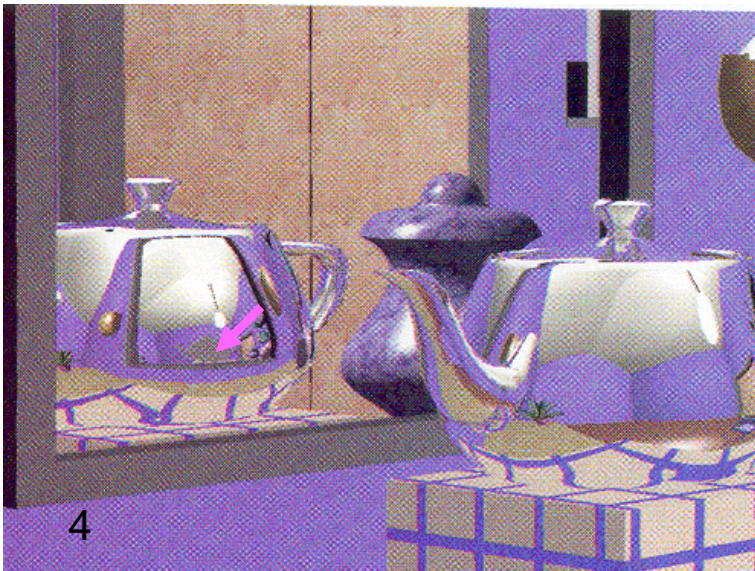
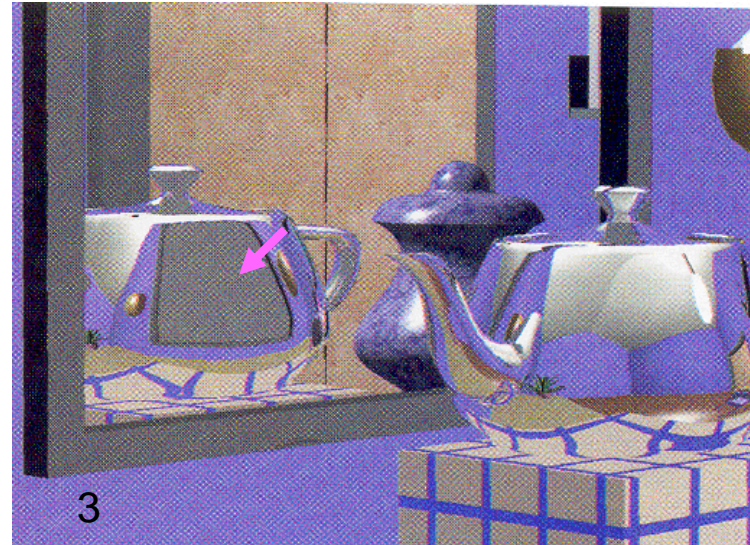
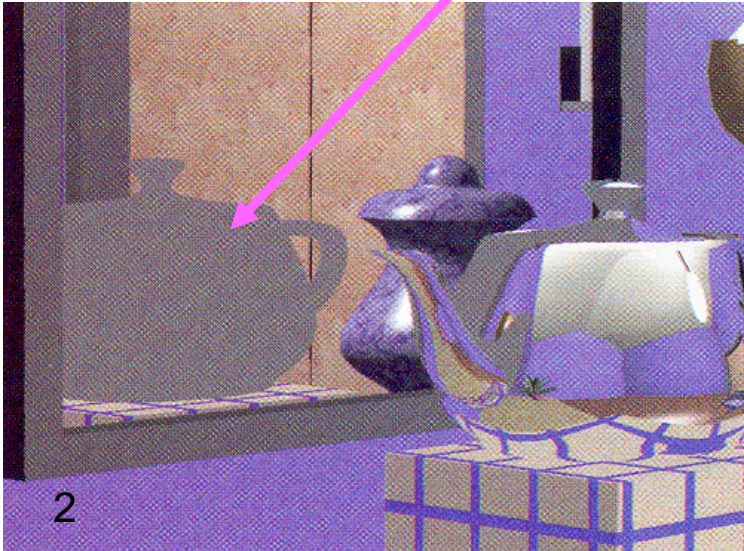


(n=16 in this example)



Disadvantage: it is time consuming!

Pixels without any assigned value (gray)



Ray tracing with different depth :
2, 3 and 4 secondary rays

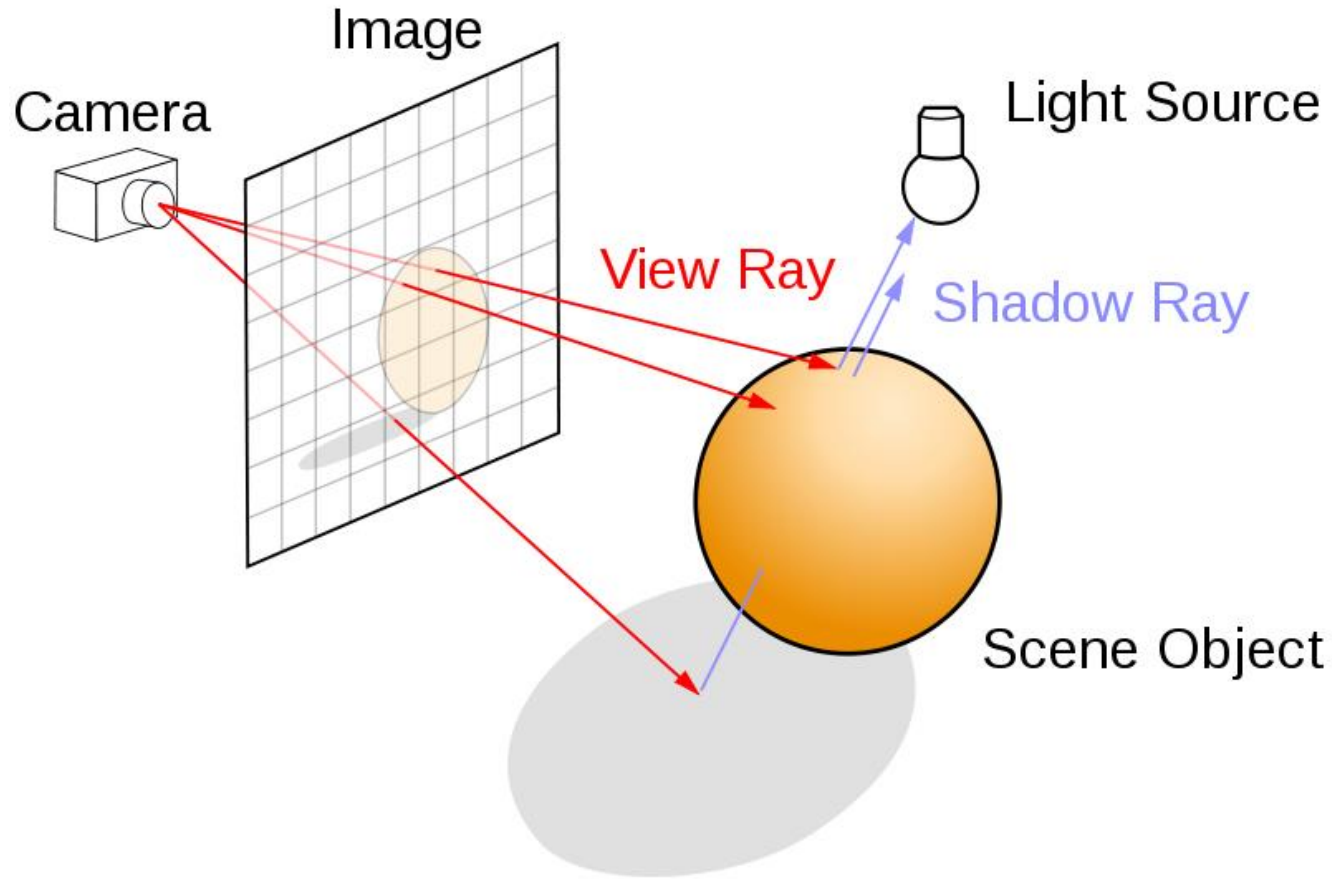


Real time ray tracing NVIDIA Marbles at Night RTX Demo

https://www.youtube.com/watch?v=NgcYLvlp_k



Ray tracing also computes shadows



[https://en.wikipedia.org/wiki/Ray_tracing_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics))

Some reference books

- S. Marschner, P. Shirley, *Fundamentals of Computer Graphics*, 5th ed., A K Peters/CRC Press, 2021
[Fundamentals of Computer Graphics, 5th Edition \(oreilly.com\)](#)
- D. Hearn and M. P. Baker, *Computer Graphics with OpenGL*, 3rd Ed., Addison-Wesley, 2004
- E. Angel and D. Shreiner, *Introduction to Computer Graphics*, 6th Ed., Pearson Education, 2012
- Hughes, J., A. Van Dam, et al., *Computer Graphics, Principles and Practice*, 3rd Ed., Addison Wesley, 2013
[Hughes/Computer Graphics, 3/E \(oreilly.com\)](#)