CSCI 420 Computer Graphics
Lecture 2

Introduction to OpenGL

OpenGL API
Core and Compatibility Profiles
Colors
[Angel Ch. 2]

Oded Stein
University of Southern California

How do computers even display things?

https://commons.wikimedia.org/wiki/File:Zuse-Z4-Totale_deutsches-museum.jpg

How do computers even display things?

https://commons.wikimedia.org/wiki/File:Printer_dot_matrix_EPSON_VP-500.jpg

How do computers even display things?

https://commons.wikimedia.org/wiki/File:Trinitron_computer-monitor.jpg

How to draw on a screen?

- Even small screens have lots of pixels.
- Refresh rates are very high.
- CPUs are quite slow (especially in the 70s and 80s).
- Specific hardware
  - Drawing simple 2D objects
  - Drawing text
  - Simple effects

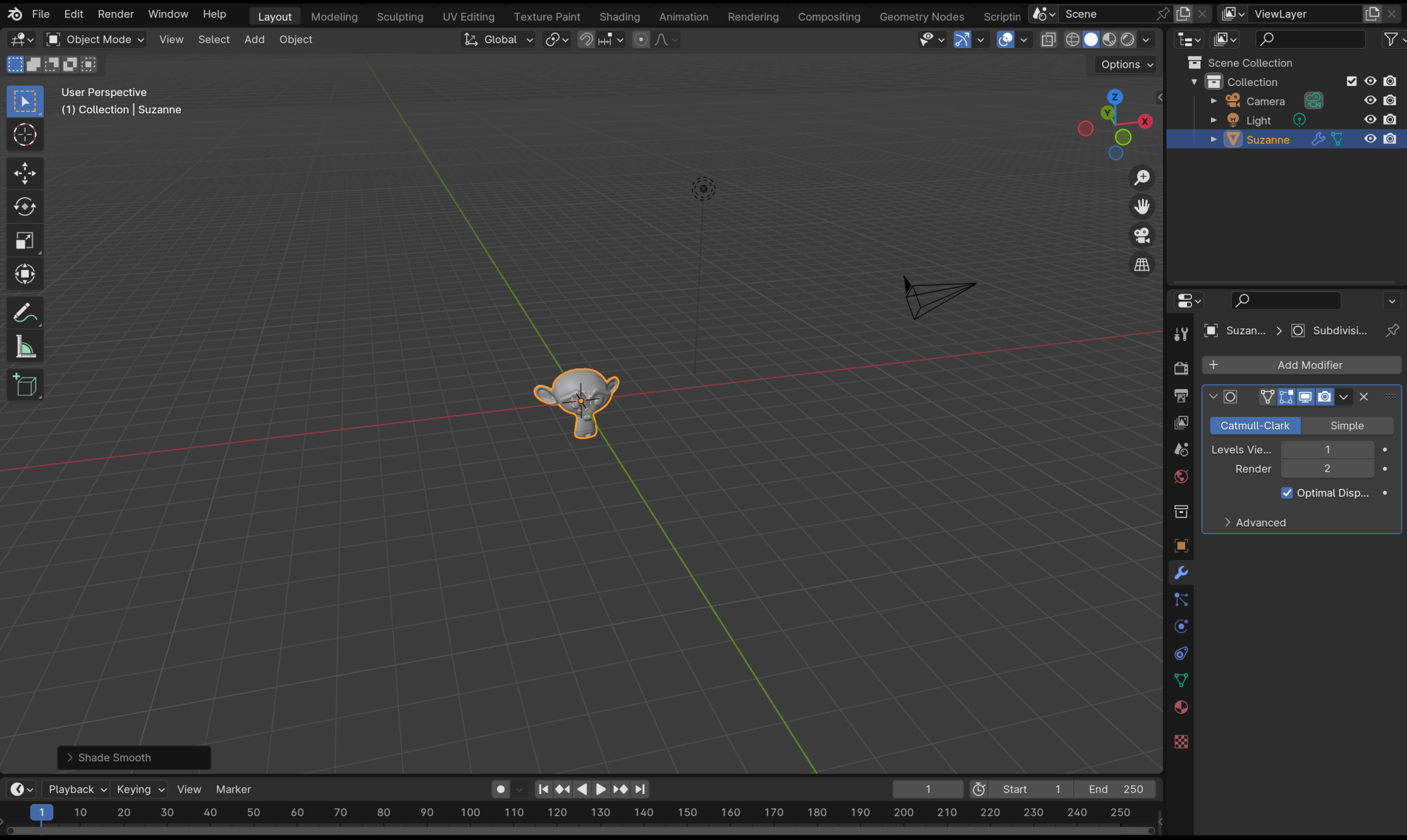
What is OpenGL

- A low-level graphics library (API) for 2D and 3D interactive graphics.
- Descendent of GL (from SGI)
- First version in 1992; now: 4.6 (released July 2017)
- Managed by Khronos Group (non-profit consortium)
- API is governed by Architecture Review Board (part of Khronos)

Where is OpenGL used

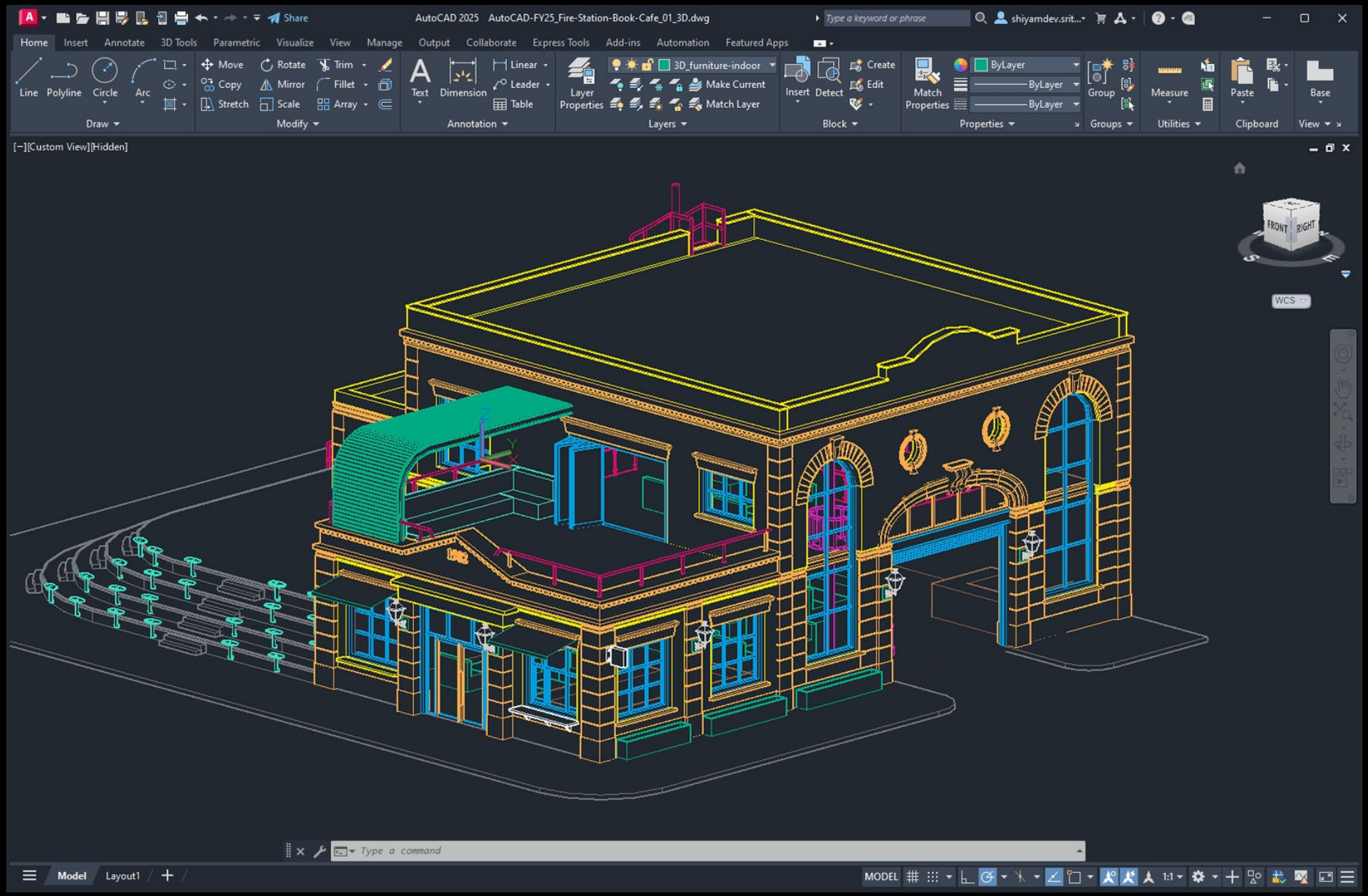
- CAD
- Virtual reality
- Scientific visualization
- Flight simulation
- Video games

3D Modeling software



- Blender

CAD software



- AutoCAD

Video games

no OpenGL

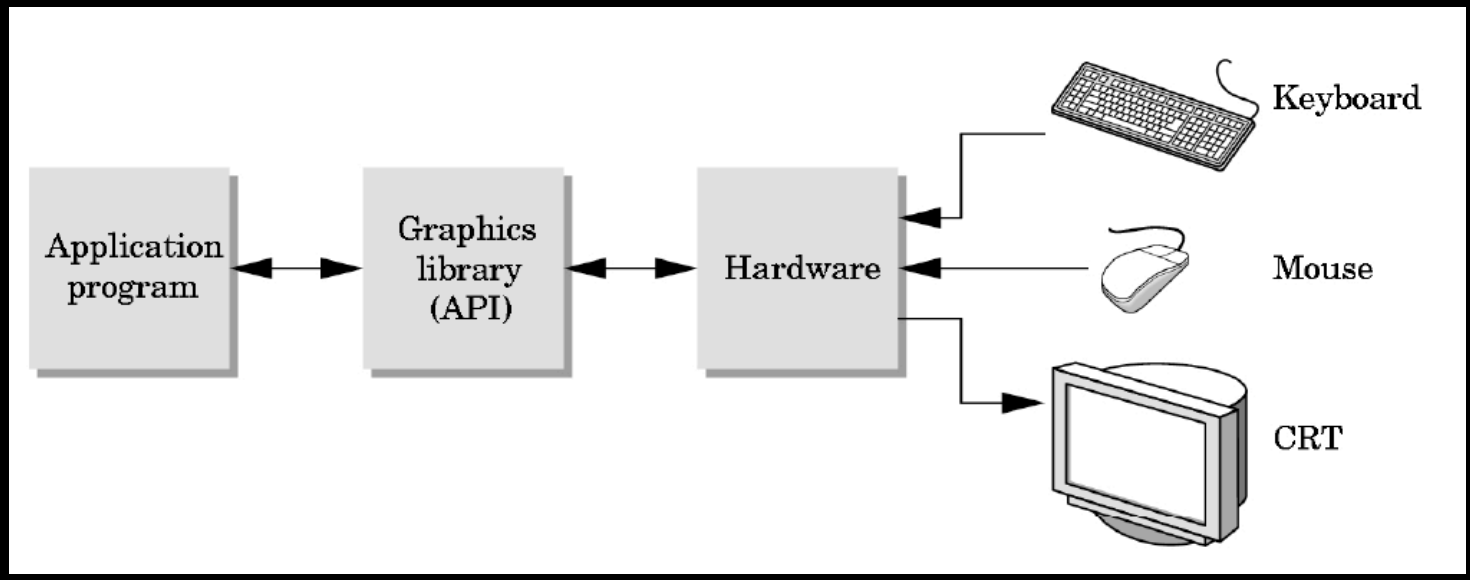
source: eurogamer.net

OpenGL

source: Paste Magazine

Graphics library (API)

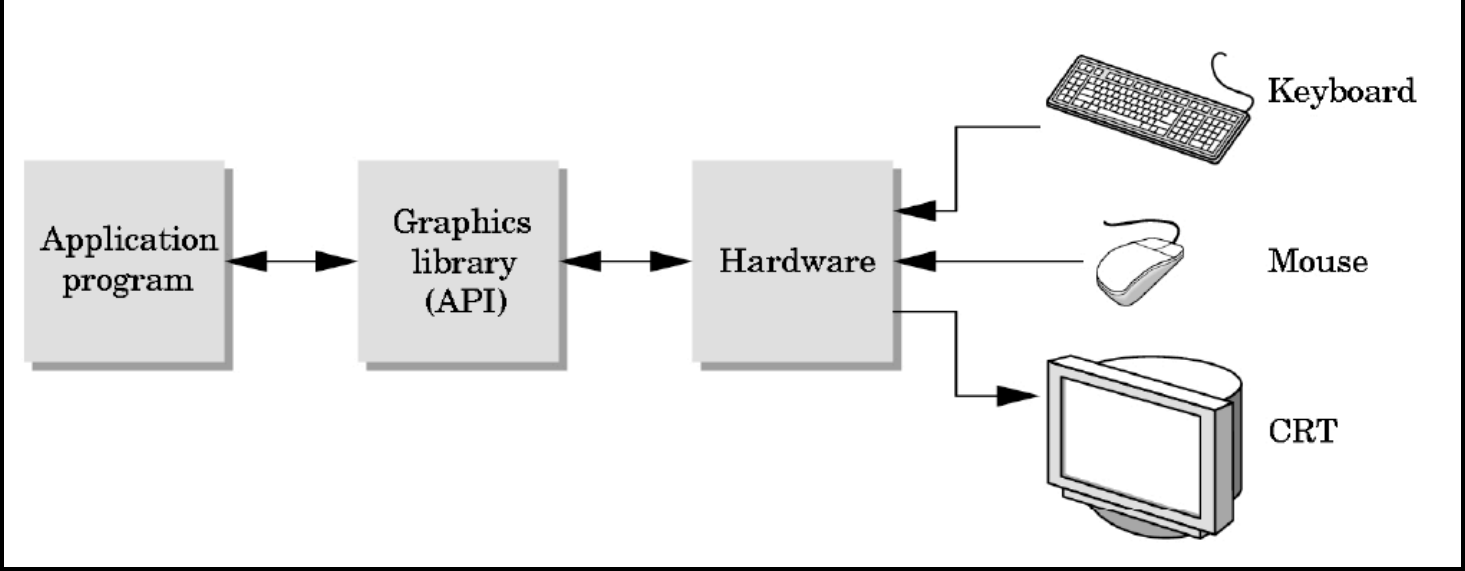
- Intermediary between applications and graphics hardware



- Other popular APIs:
  Direct3D (Microsoft)
  OpenGL ES (embedded devices)
  X3D (successor of VRML)
  Vulkan (more low-level than OpenGL)

Graphics library (API)

- Intermediary between applications and graphics hardware



- Other popular APIs:
  WebGPU
  And many more (ThreeJS etc.)

Before Graphics Engines

- Directly rendering to a file.
- Directly writing to a framebuffer

Before OpenGL

- The first graphics cards become common.
- Graphics card vendors created their own libraries to do common 3D tasks.
- PHIGs (Programmer's Hierarchical Interactive Graphics System)
- Silicon Graphics makes **IRIS GL**

(this is the early 90s... before my time)

Before OpenGL

- The first graphics cards become common.
- Graphics card vendors created their own libraries to do common 3D tasks.
- PHIGs (Programmer's Hierarchical Interactive Graphics System)
- Silicon Graphics makes **IRIS GL**
- Other creators of graphics hardware rush to make their own libraries.
  - Attract developers => sell more hardware
- IRIS GL is open sourced and standardized as the first OpenGL

History of OpenGL

- OpenGL is administered by OpenGL ARB (OpenGL 1, 2)
- The ARB defines a standard API with versions that the different graphics card vendors (nvidia, AMD) implement
- OpenGL becomes very popular: most graphical devices these days support OpenGL in some form.
- OpenGL transitions to Khronos industry group
- New "modern" OpenGL is introduced (3, 4)

Old OpenGL

- You tell the graphics driver all the primitives you want to draw.
- You tell the driver about lights, material parameters, textures, etc...
- The GPU does the rest for you.

```
glBegin(GL_TRIANGLES);
  glColor3f(0.0,0.0,1.0);
  glVertex3f(x1, y1, z1);
  ...
  glVertex3f(xN, yN, zN);
glEnd();
```

Modern OpenGL

- Old way is actually very far from how GPUs like to work internally.
- More modern:
  - Load everything into GPU memory.
  - Tell the GPU where all objects are.
  - Write GPU shader that assigns colors to all your objects.
  - Run GPU program.

Modern OpenGL

```
// Write shader programs that will later assign
// colors to the triangle in vertShaderCode
unsigned int vertShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertShader, 1, & vertShaderCode, NULL);
glCompileShader(vertShader);
glAttachShader(ID, vertex);
glLinkProgram(ID);
float vertices[] = {0.,0., 1.,0., 0.,1.};
GLuint vbo;
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER,
   sizeof(vertices), vertices, GL_STATIC_DRAW);
glDrawArrays(GL_TRIANGLES, 0, 3);
```

After OpenGL

- Future for OpenGL is uncertain.
  - But not for shader-based graphics pipelines.
- Many people use higher-level engines (Unreal, Unity, etc...)
- Big engine developers like access that is more low-level than OpenGL
- Vulkan (also by Khronos)
- OS-Specific walled gardens: Direct3D, Metal
- ...but OpenGL is very easy to get into, and will work on many devices for a long time.

OpenGL is cross-platform

- Same code works with little/no modifications
- Windows: default implementation ships with OS
  Improved OpenGL: Nvidia or AMD drivers
- Linux: Mesa, a freeware implementation
  Improved OpenGL: Nvidia or AMD drivers
- Mac: ships with the OS. Apple announced deprecation in 2018, but OpenGL continues to work.

Choice of Programming Language

- OpenGL lives close to the hardware
- OpenGL is not object-oriented
- OpenGL is not a functional language (as in, ML)
- Use C to expose and exploit low-level details
- Use C++, Java, ... for toolkits
- Support for C in assignments
  - (but why would you)

So, who actually implements OpenGL?

- Graphics Card vendors
  - Graphics cards need to translate OpenGL code to microcode that can run on GPUs.
  - Graphics cards need to translate OpenGL instructions to actual electrical signals on the display port.
  - GPU vendors write drivers for OpenGL.

- Operating Systems
  - Need to provide a window (or fullscreen environment)
  - Perform I/O
  - Runs the actual driver

- OpenGL is not just a thing you can download and install!

OpenGL is cross-platform

Include file (OpenGL Compatibility Profile) :

```
#if defined(WIN32) || defined(linux)
  #include <GL/gl.h>
  #include <GL/glu.h>
  #include <GL/glut.h>
#elif defined(__APPLE__)
  #include <OpenGL/gl.h>
  #include <OpenGL/glu.h>
  #include <GLUT/glut.h>
#endif
```

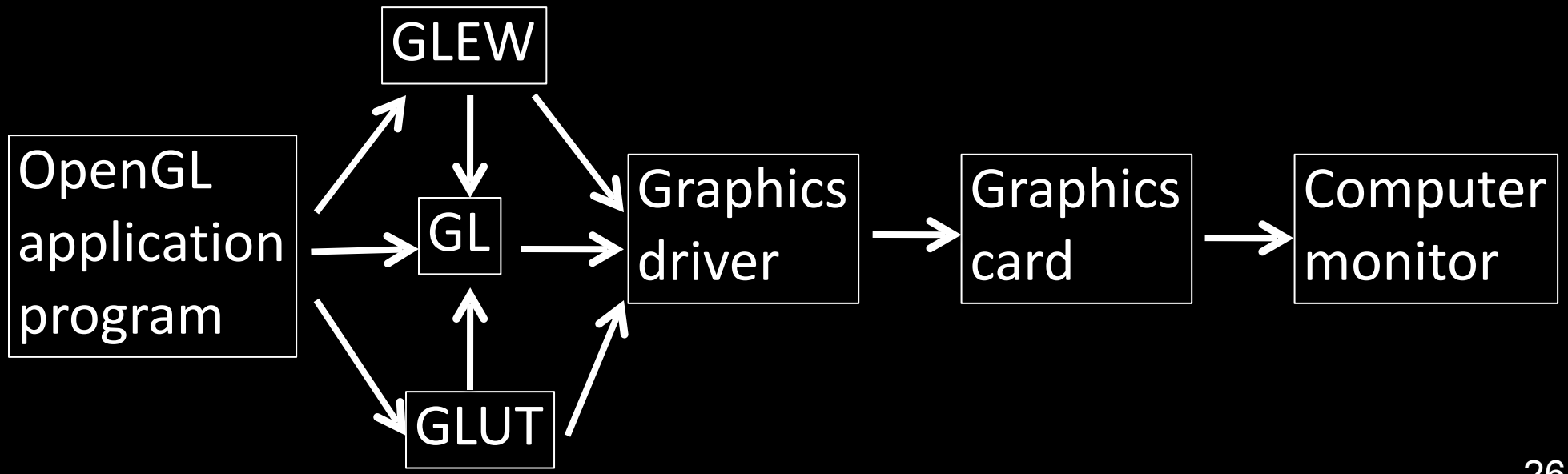
OpenGL is cross-platform

Include file (OpenGL Core Profile) :

```
#if defined(WIN32) || defined(linux)
  #include <GL/glew.h>
  #include <GL/glut.h>
#elif defined(__APPLE__)
  #include <OpenGL/gl3.h>
  #include <OpenGL/gl3ext.h>
  #include <GLUT/glut.h>
#endif
```

OpenGL Library Organization

- **GL** (Graphics Library): core graphics capabilities
- **GLUT** (OpenGL Utility Toolkit): input and windowing
- **GLEW** (Extension Wrangler): removes OS dependencies
- **GLU** (OpenGL Utility Library; compatibility profile only): utilities on top of GL



Core vs Compatibility Profile

- Core Profile:
  - “Modern” OpenGL
  - Introduced in OpenGL 3.2 (August 2009)
  - Optimized in modern graphics drivers
  - Shader-based
  - Used in our homeworks

- Compatibility Profile:
  - “Classic” OpenGL
  - Supports the “old” (pre-3.2) OpenGL API
  - Fixed-function (non-shader) pipeline
  - Not as optimized as Core Profile

What is the difference between classic and modern OpenGL?

- Classic:
  - Issue CPU commands to draw graphics primitives (like vertices, triangles, lines)
  - Supports a "classical" graphics pipeline, with classical graphics transformations.
  - Less performant

- Modern:
  - You directly interact with the GPU memory
  - You write a program which the GPU executes on each primitive (the shader)
  - Allows for more flexibility.
  - More performant

Mixing core and compatibility profiles

- Windows, Linux:
  Can mix core and compatibility profile OpenGL commands
  ➔ can lead to confusion
  (is the specific OpenGL command optimized?)
  ➔ advantage: more flexible (can re-use old code)

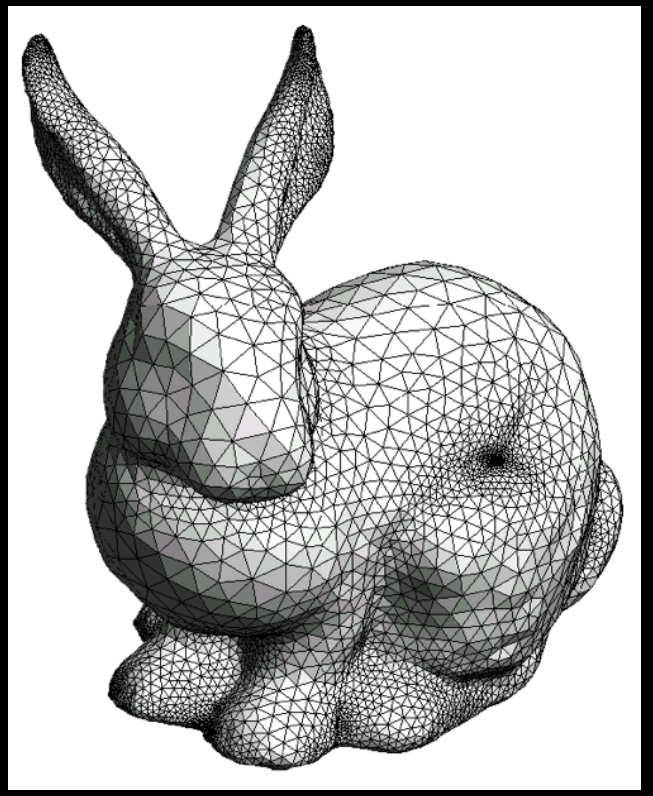
- Mac:
  Can only choose one profile (in each application)

How does OpenGL work

From the programmer's point of view:

1. Specify geometric objects

2. Describe object properties
   - Color
   - How objects reflect light

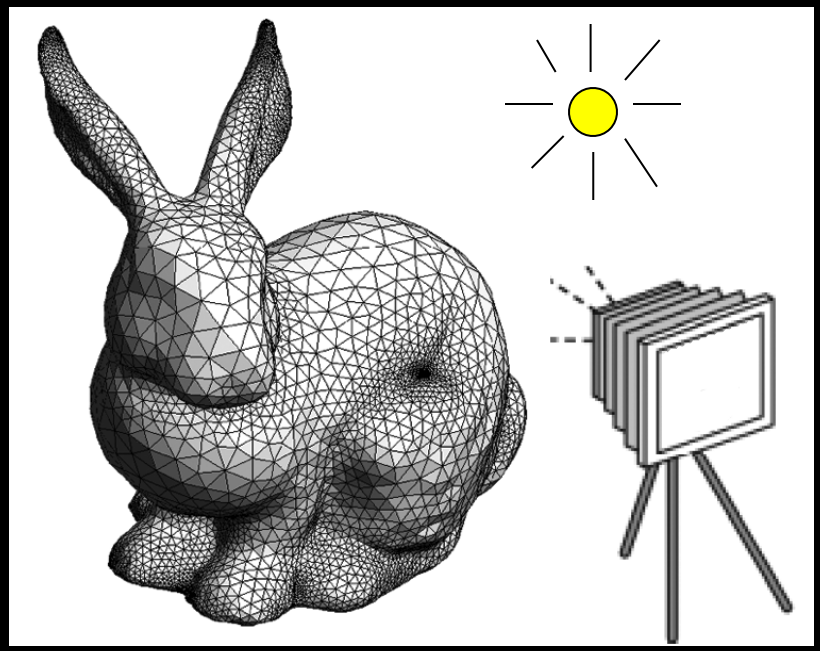


How does OpenGL work (continued)

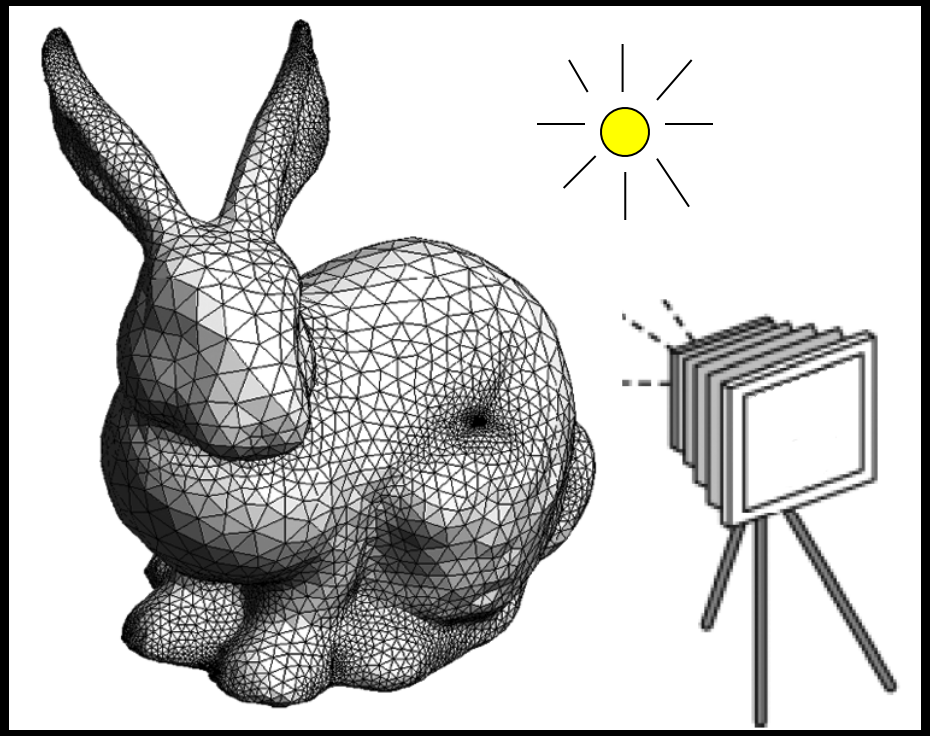
3. Define how objects should be viewed
   - where is the camera
   - what type of camera

4. Specify light sources
   - where, what kind

5. Move camera or objects around for animation



The result



the result

If only it were so easy...

This was easy to say.

Of course, all of this has to be programmed!

Mix of mathematics & GPU coding to make everything happen.

Everything revolves around "the pipeline" (next class)

OpenGL is a state machine

State variables: vertex buffers, camera settings, textures, background color, hidden surface removal settings, the current shader program...

These variables (the *state*) then apply to every subsequent drawing command.

They persist until set to new values by the programmer.

Geometric objects:
vertices and triangles

- OpenGL can draw points, lines, and triangles
  - (but really, probably only triangles)
- We have to specify these objects (or primitives), and load the into GPU memory.
- We then need to tell the GPU where in memory which primitive lies, and to draw each of them.

Geometric objects:
vertices and triangles

$$\Delta = ABC$$
$$A = (0,0)$$
$$B = (1,0)$$
$$B = (0,1)$$

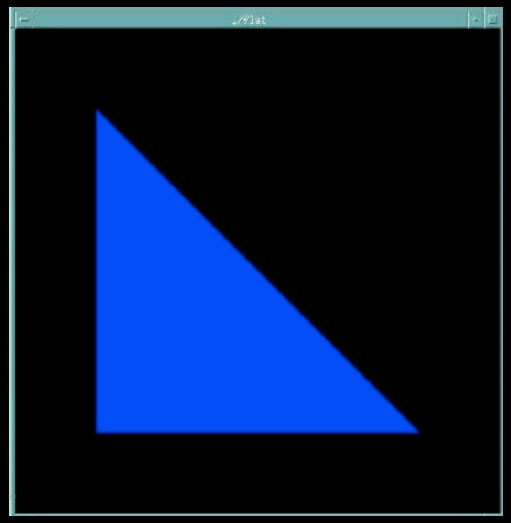
a triangle

```
vec2 A{0., 0.};
vec2 B{1., 0.};
vec2 C{0., 1.};
```

CPU

```
float vertices[] = {0.,0., 1.,0., 0.,1.};
GLuint vbo;
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER,
   sizeof(vertices), vertices,
GL_STATIC_DRAW);
glDrawArrays(GL_TRIANGLES, 0, 3);
```

GPU



screen

Attributes:
color, shading and reflection properties

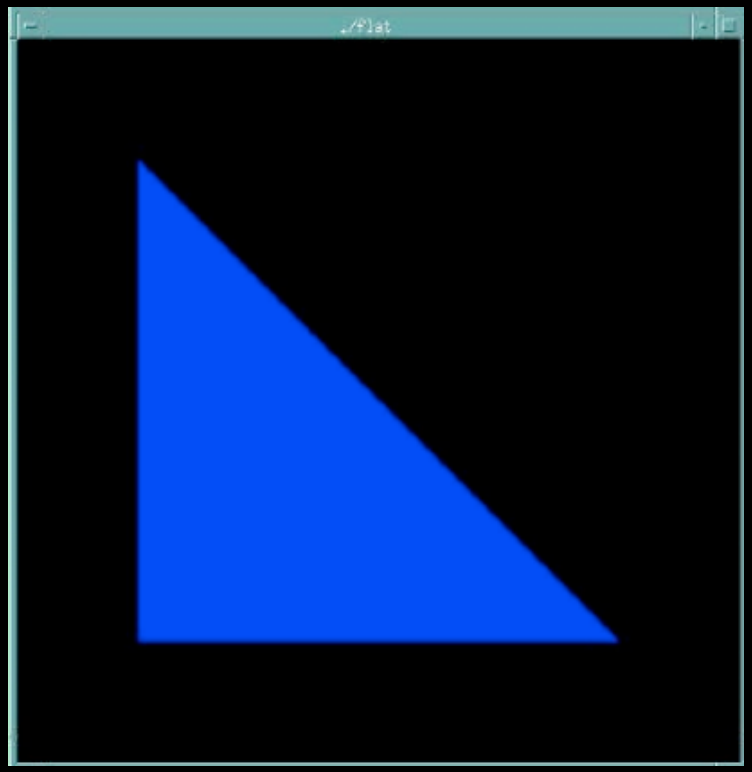
- Set before primitives are drawn
- Remain in effect until changed!
- The GPU will just continuously redraw with our set properties.

Color

- But... does each triangle have a color?
- Or each vertex of a triangle?
- Or each pixel?

Coloring primitives

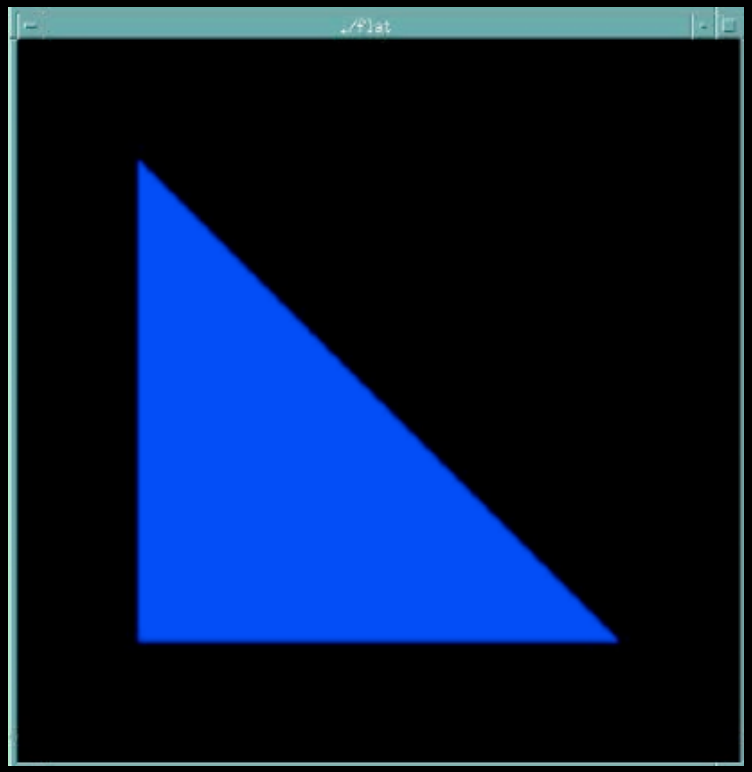
one color per triangle



each vertex separate color
smoothly interpolated

Coloring primitives

one color per triangle



Compatibility profile:
glShadeModel(GL_FLAT)

each vertex separate color
smoothly interpolated

Compatibility profile:
glShadeModel(GL_SMOOTH)

Core profile: use interpolation qualifiers in the fragment shader (do it yourself!)

Coloring primitives

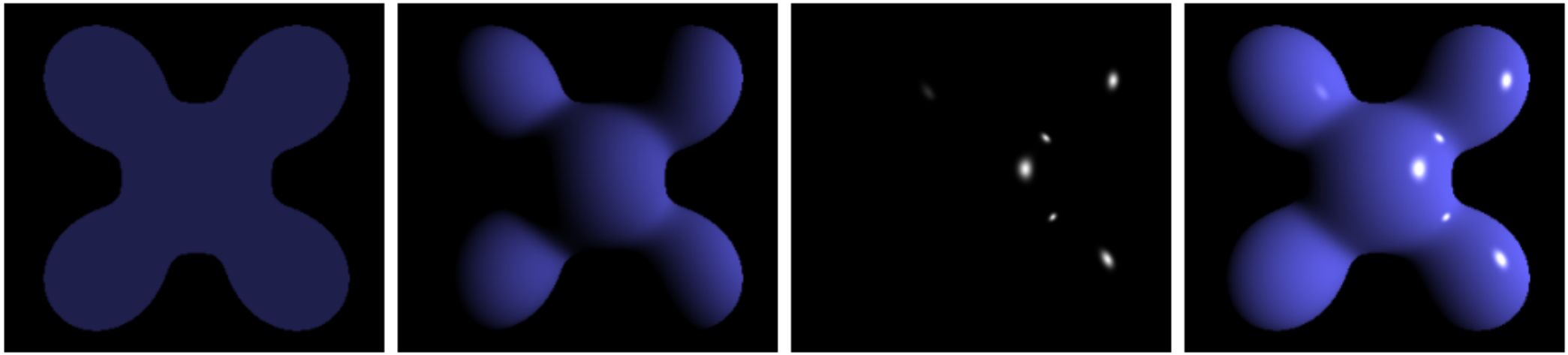
Core profile: use interpolation qualifiers in the fragment shader

- This is nontrivial!
- You will need to write your own shader program to interpolate properties from each vertex onto each point in the triangle.
- Shows the philosophical difference between "classical" and "modern" OpenGL

Flat vs Smooth Shading

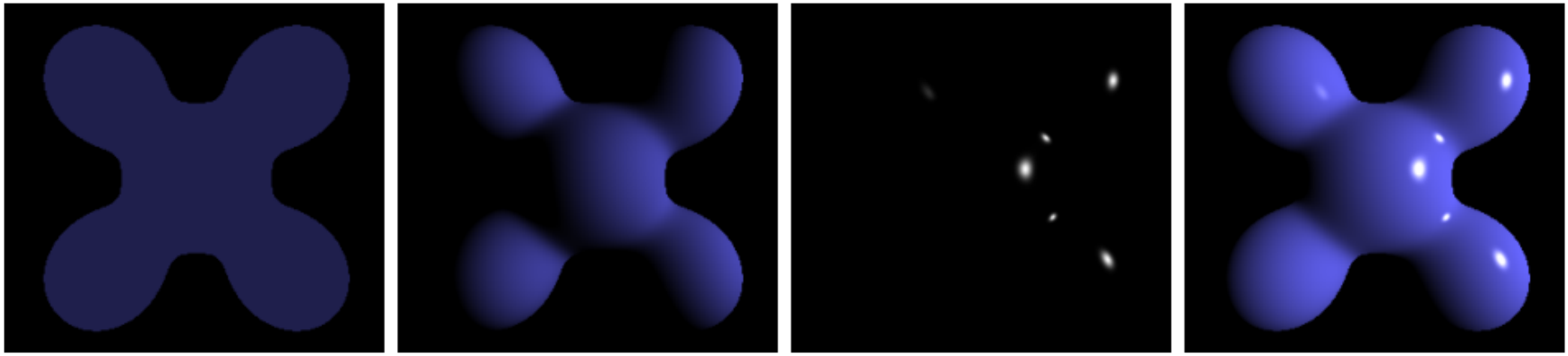
- Colors are not just used to specify the physical color of an object.
- We have to mimic the effect of actual light that hits a camera.
- Physical effects like reflectivity have to be simulated via color.

Blinn-Phong



https://commons.wikimedia.org/wiki/File:Phong_components_version_4.png

Blinn-Phong



https://commons.wikimedia.org/wiki/File:Phong_components_version_4.png

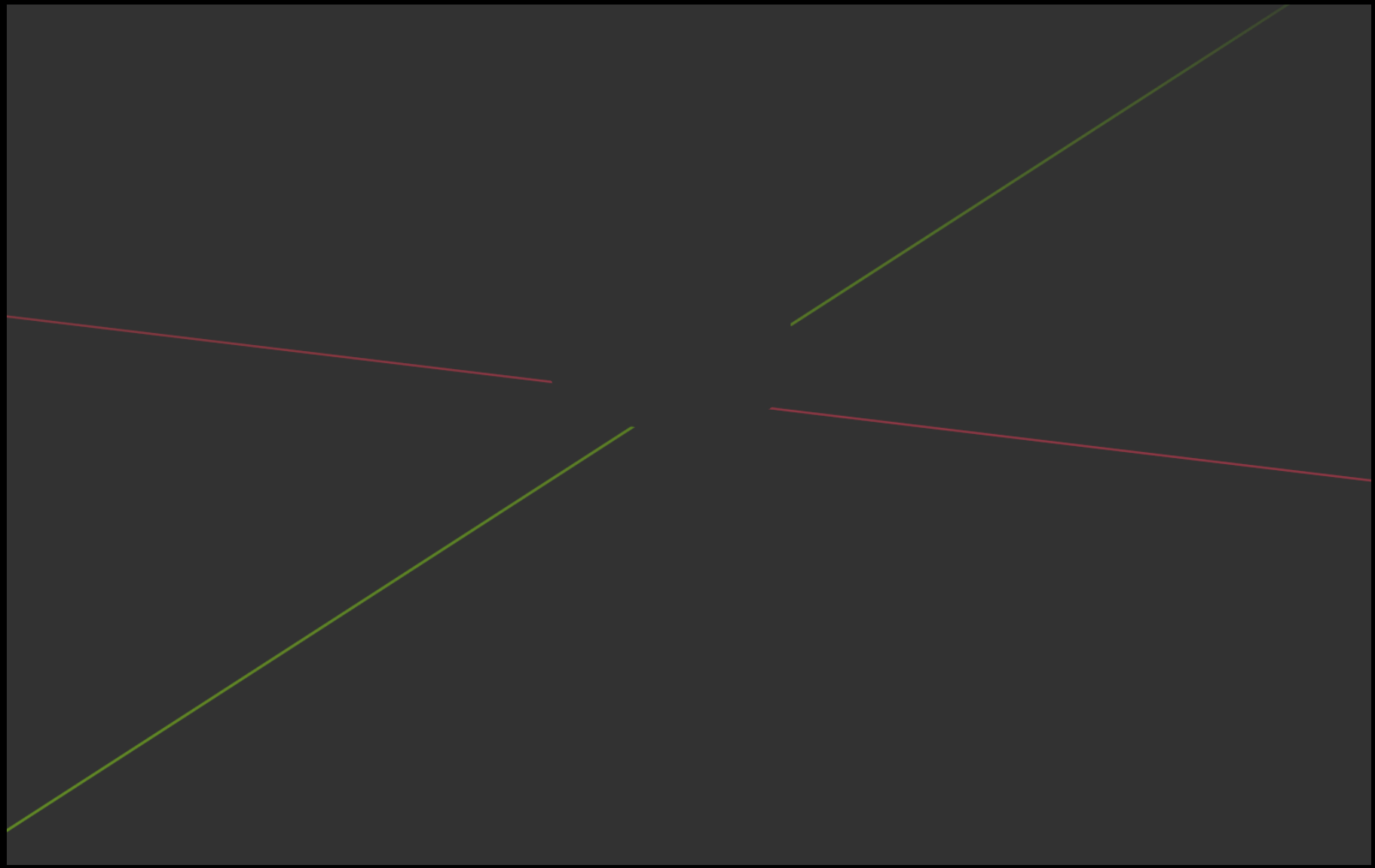
Reflection based on angle of object with respect to light source and camera

Flat vs Smooth Shading

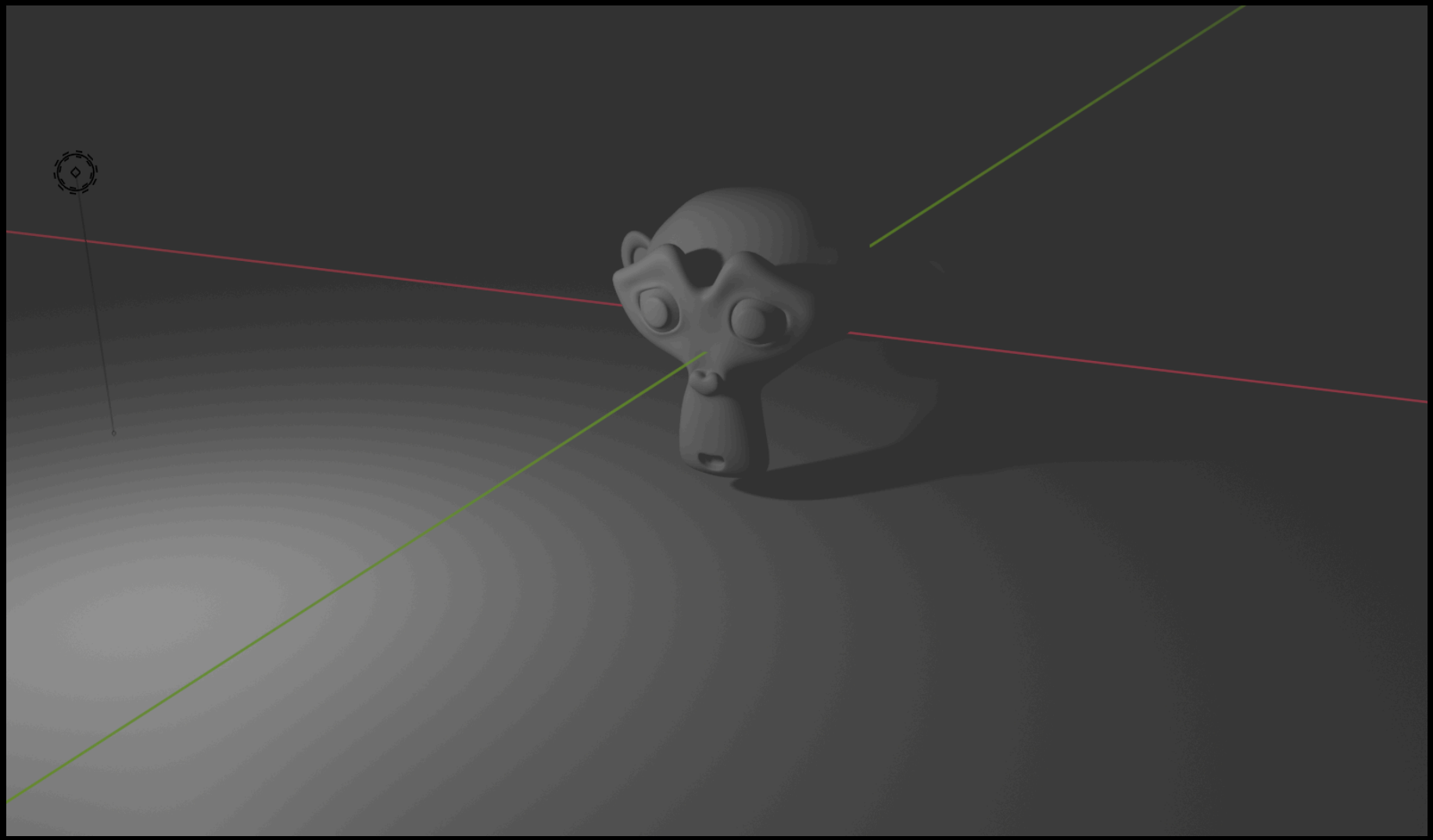
Flat Shading

Smooth Shading

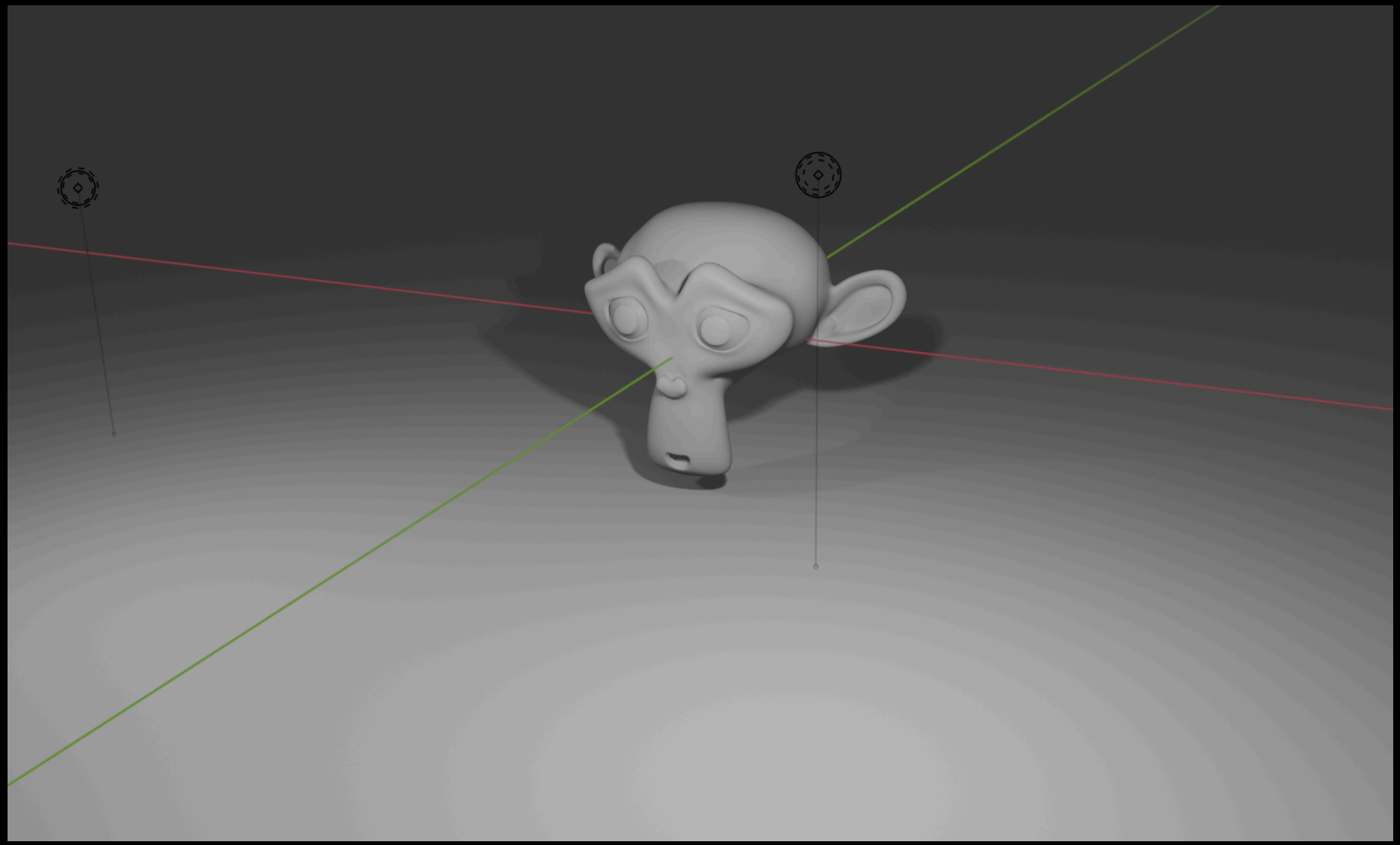
Lights



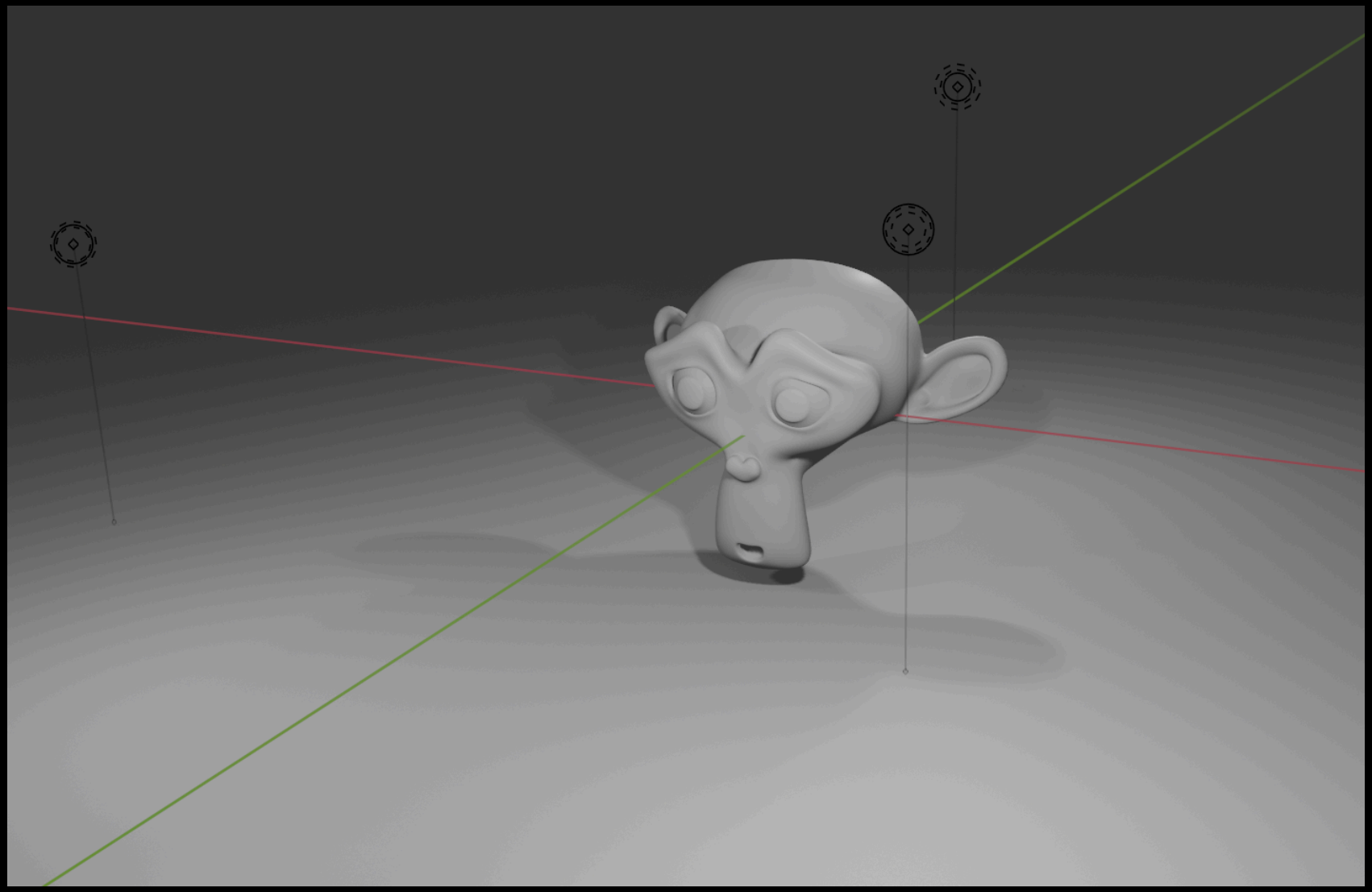
Lights



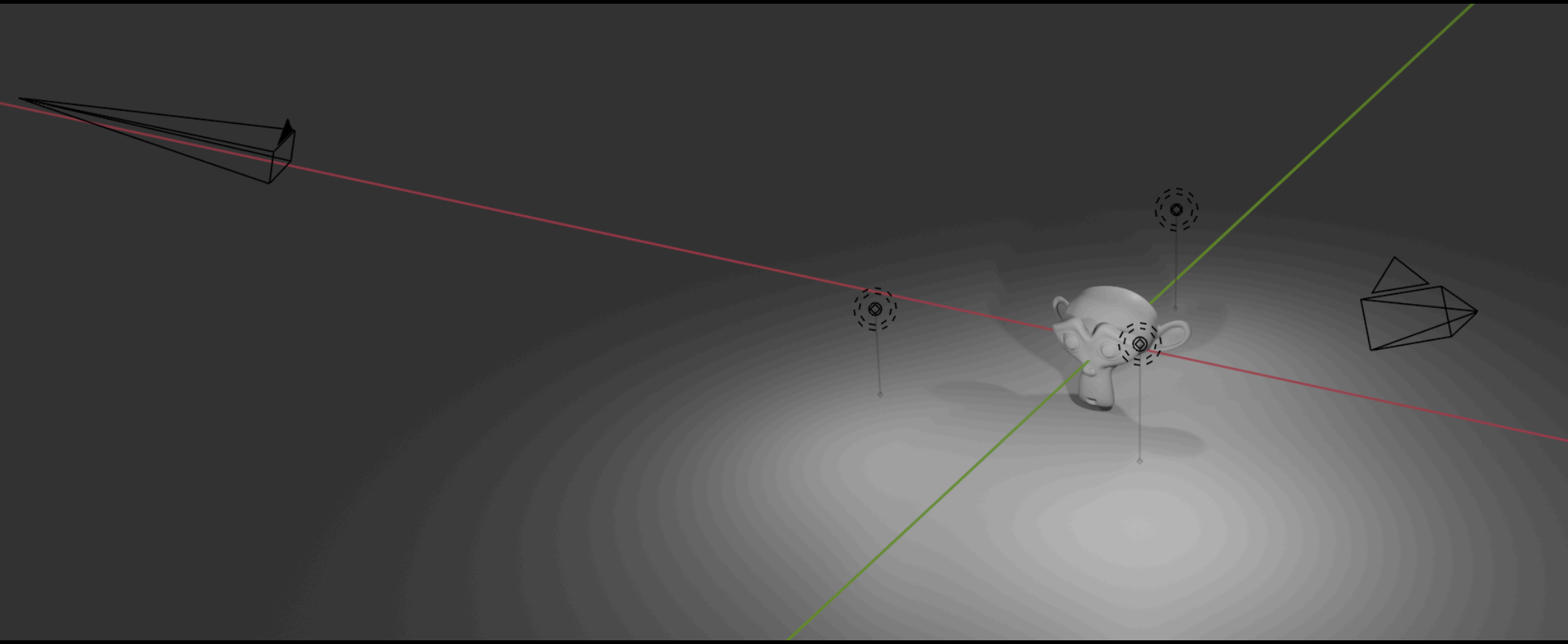
Lights



Lights

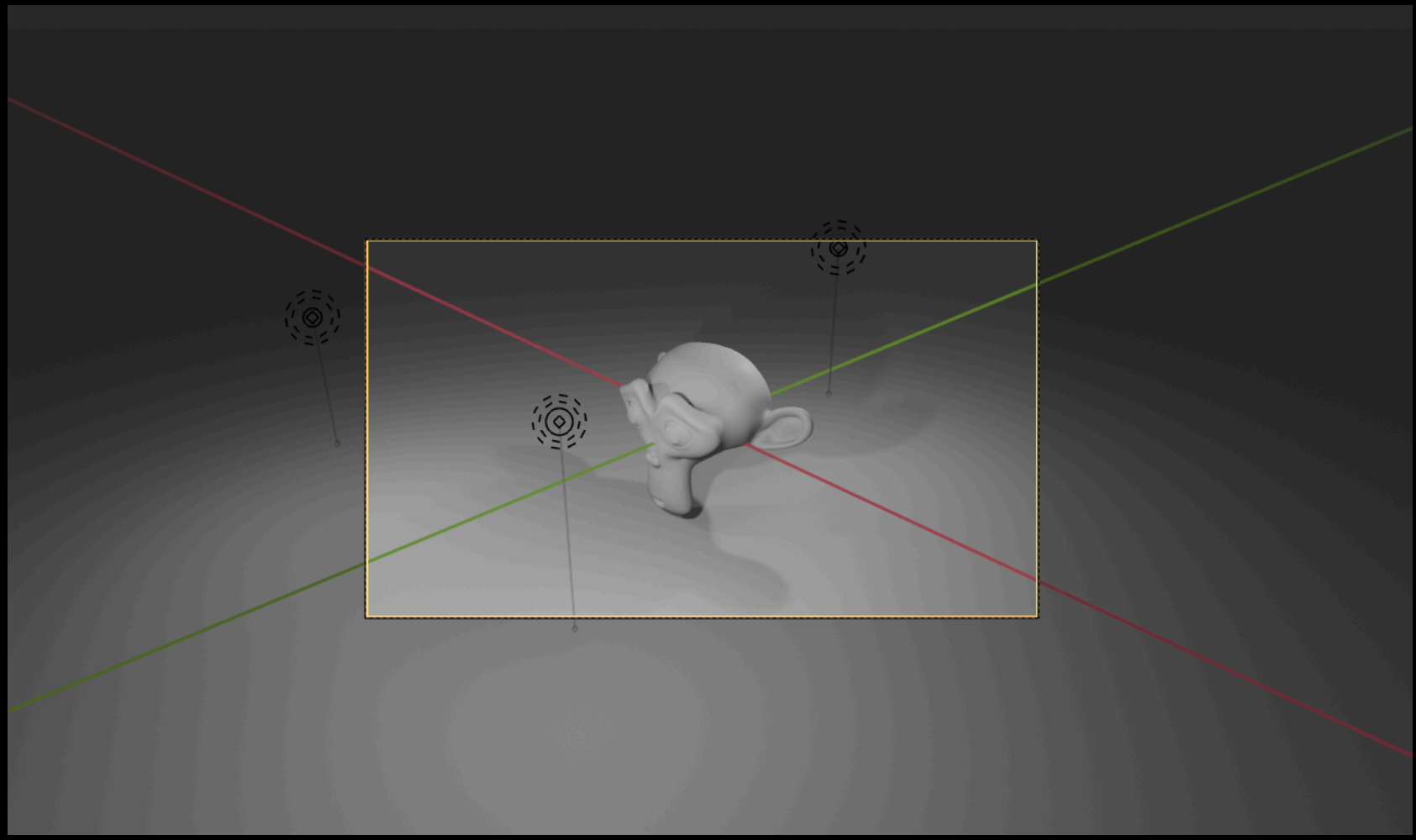


Camera



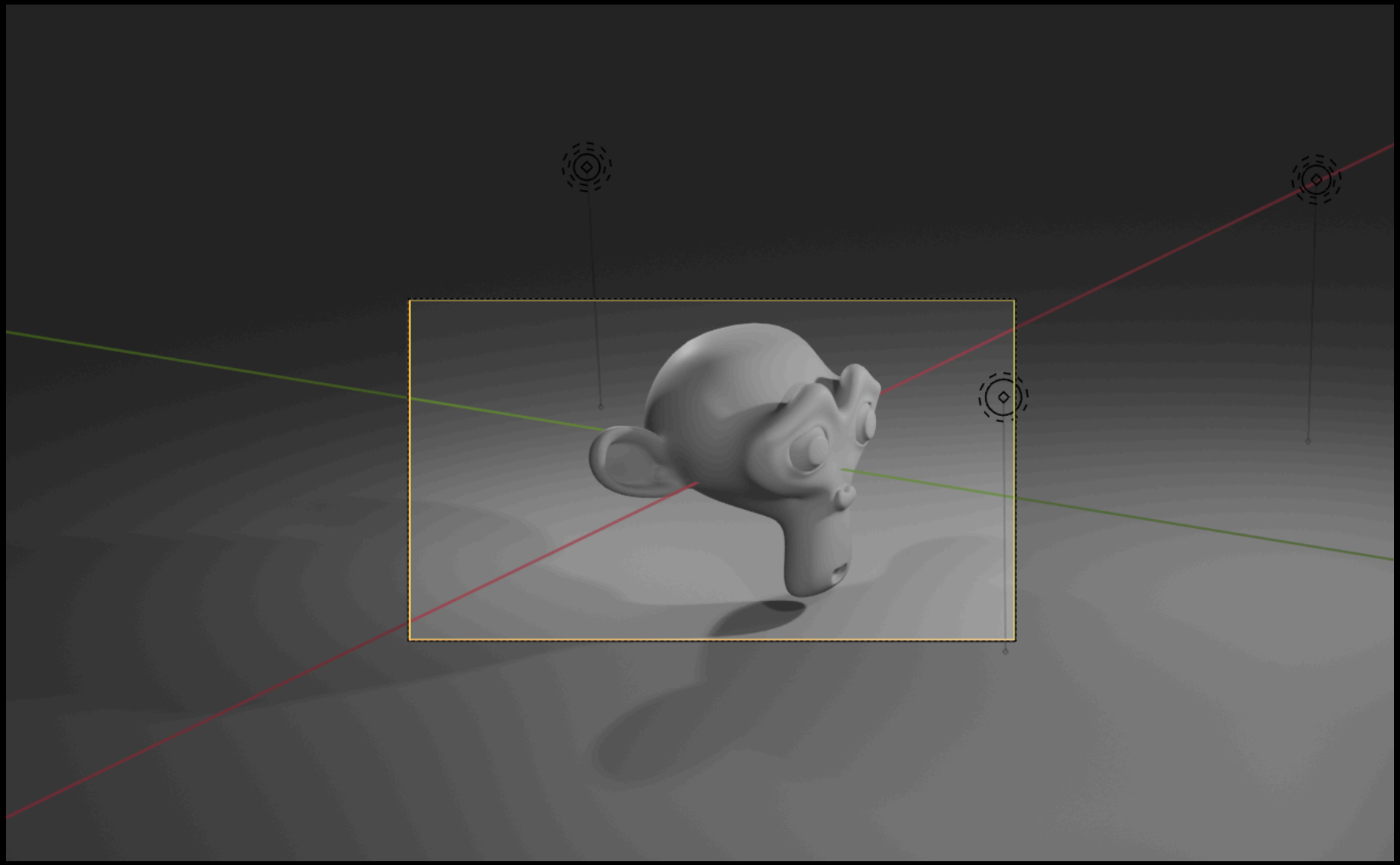
The world

Camera



Through camera 1

Camera



Through camera 2

Summary

1. OpenGL API
2. Core and compatibility profiles
3. Colors
4. Cameras
5. Lights