Computer Graphics
- Introduction of OpenGL

Junjie Cao @ DLUT

Spring 2016

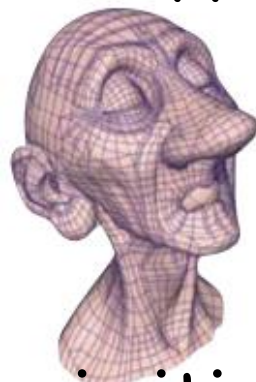http://jjcao.github.io/ComputerGraphics/

Last Time

Story ➡ Computer Graphics ➡ Image

Render [ren-der]

- OpenGL's primary function – Rendering

- Rendering?
  - converting geometric/mathematical object descriptions into frame buffer values

- OpenGL can render:
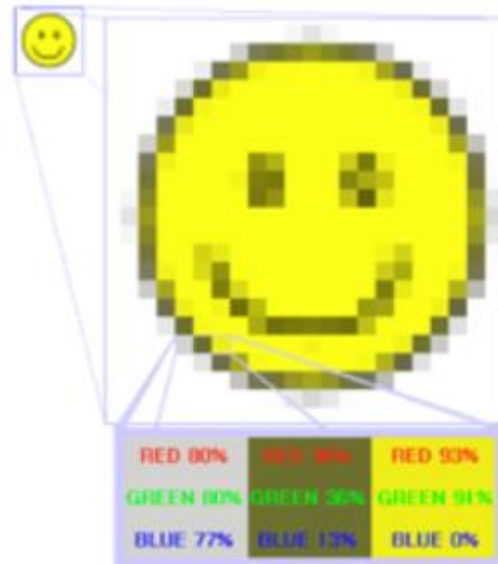 - Geometric primitives
  - Bitmaps and Images (Raster primitives)
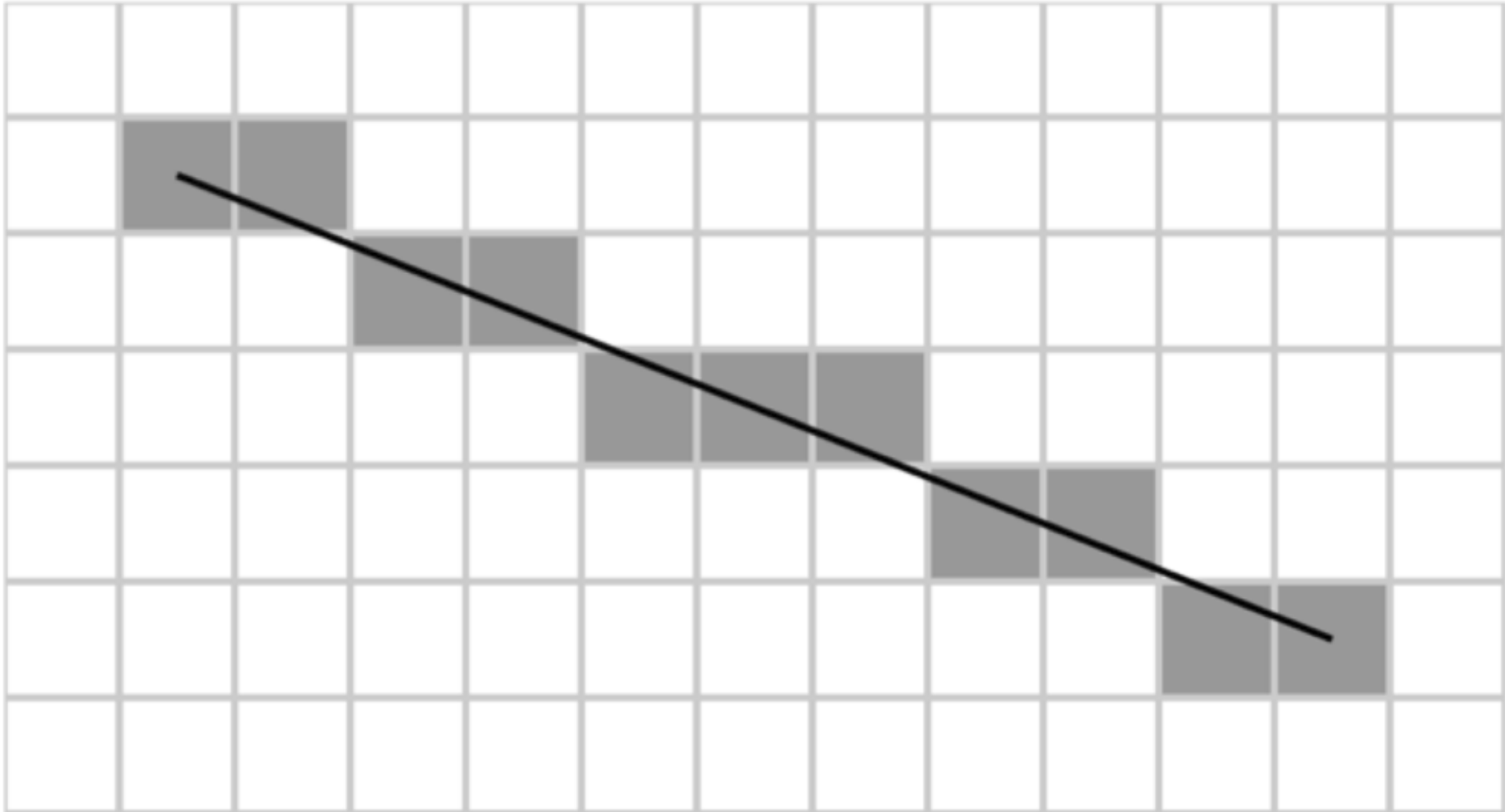
input data

output rendering

Raster Image

- ## 2D array of pixels (**pic**ture **el**ements)
  - – regular grid sampling of arbitrary 2D function
  - – different formats, e.g., bitmaps, grayscale, color
  - – different data types, e.g., boolean, int, float
  - – color/bit depth: #bits/pixel
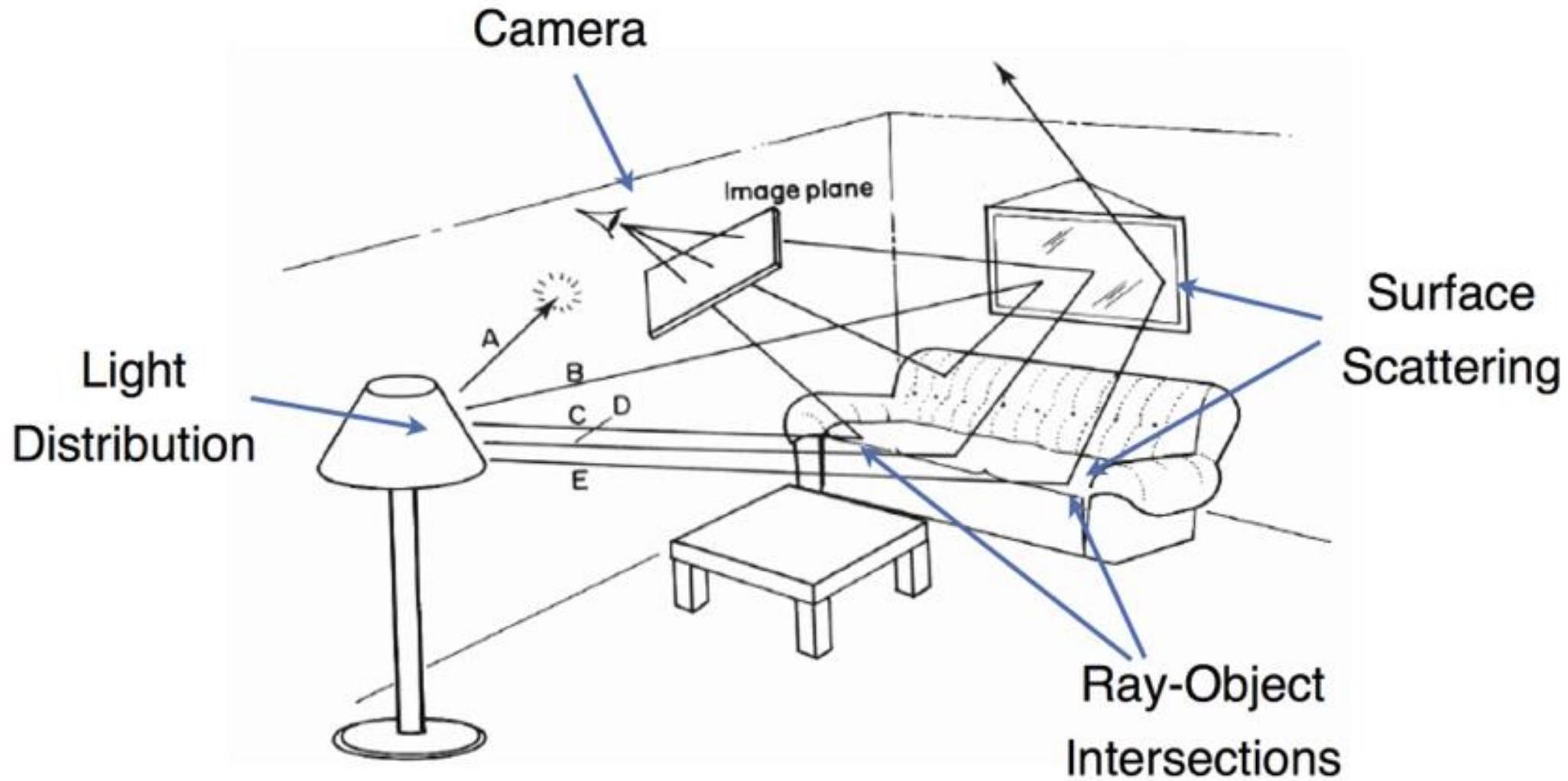 - – transparency handled by alpha channel, e.g., RGBA

[wikipedia]

Rasterization

Okay... let's take a step back

In the physical world

Camera

Image plane

Light Distribution
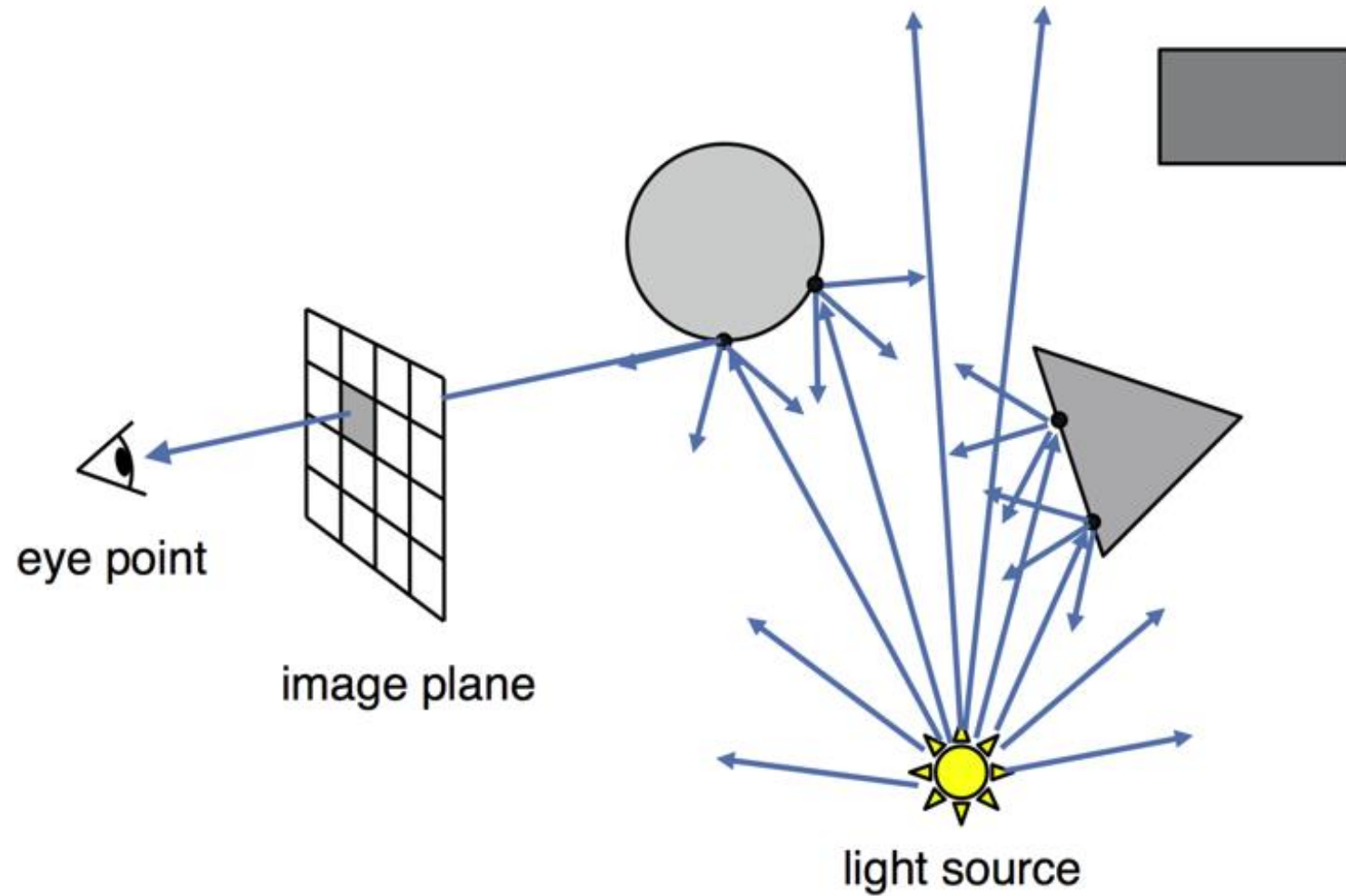
A
B
C D
E

Surface Scattering

Ray-Object Intersections
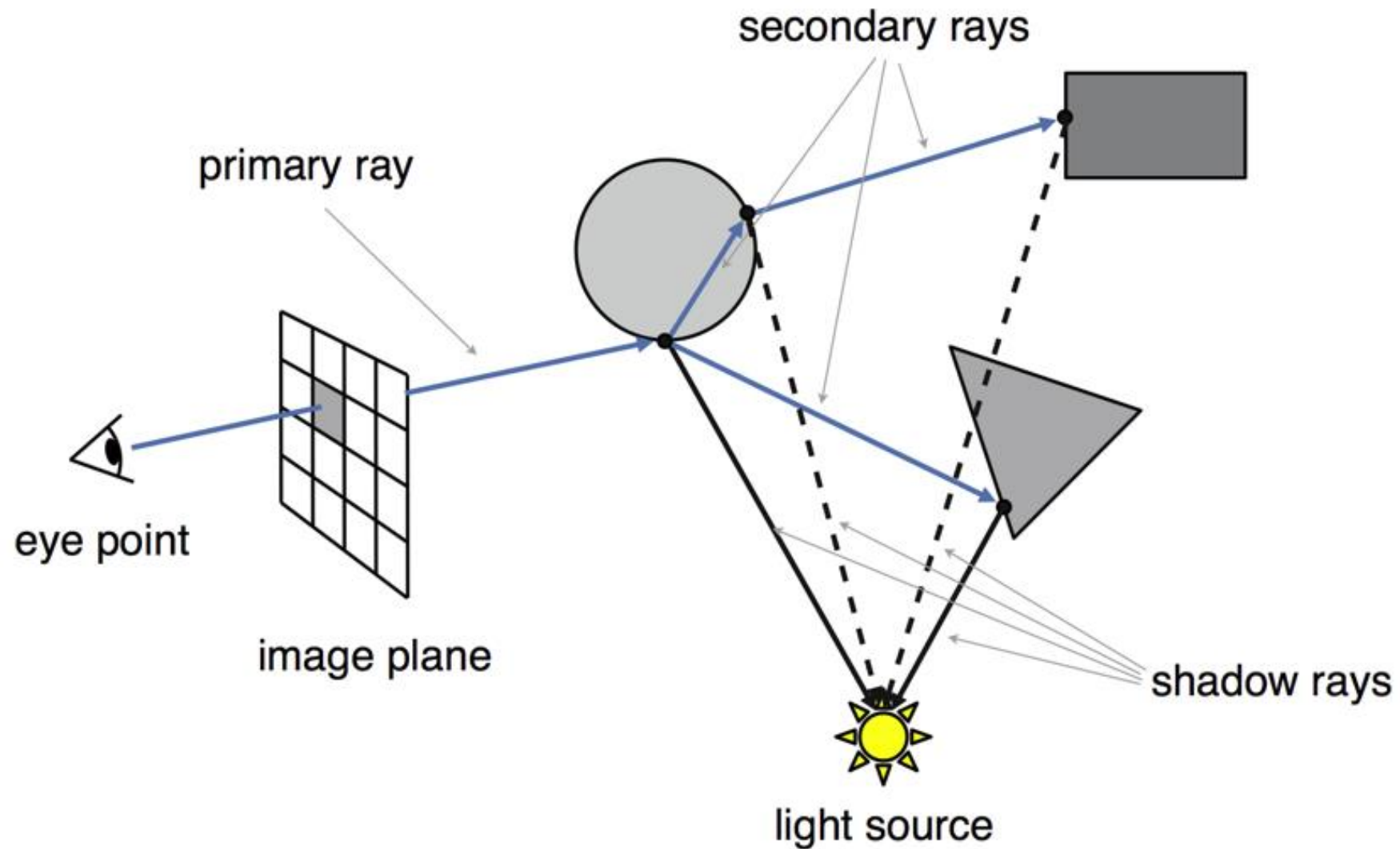
Light Transport

- Light travels in **straight lines**

- Light rays **do not interfere** with each other if they cross

- Light travels from the **light sources to the eye** (physics is invariant under path reversal reciprocity)

Light-Oriented (Forward Raytracing)

eye point

image plane

light source

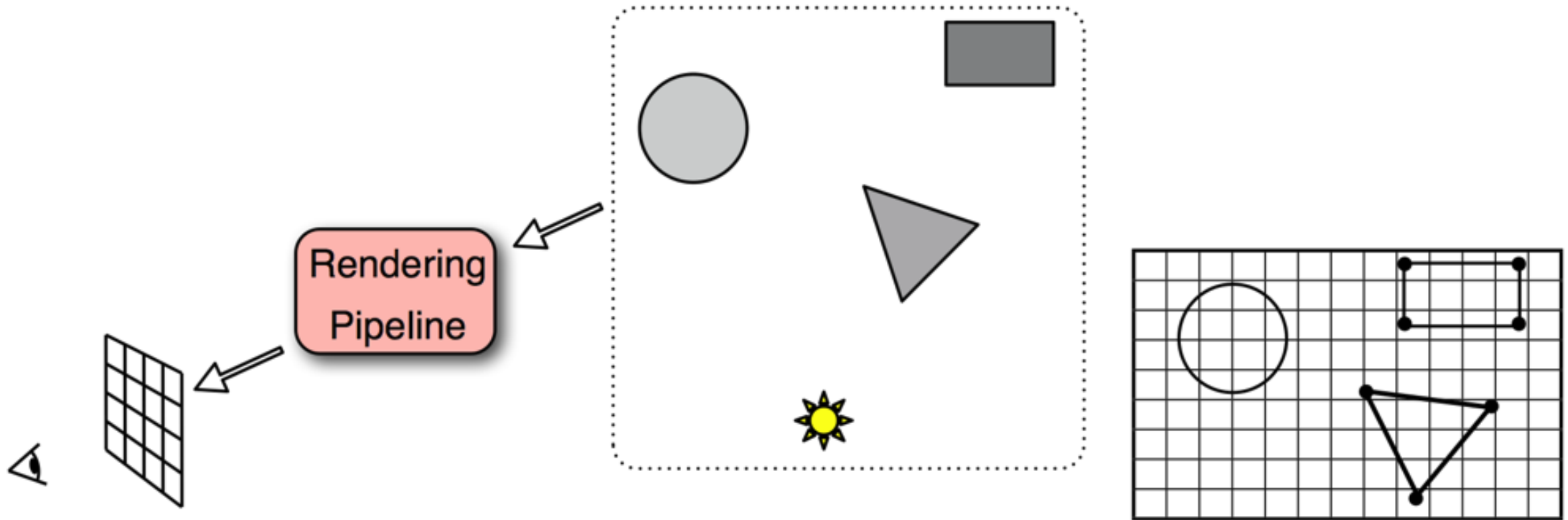Only a `fraction` of light rays reach the image

Eye-Oriented (Backward Raytracing)

or simply "Raytracing"

Object-Oriented (Forward Rendering)

Scene is composed of `geometric structures` with the buiding block of a `triangle`. Each triangle is projected, colored, and painted on the screen

Light vs. Eye vs. Object-Oriented Rendering

- **Light-oriented (Forward Raytracing)**
  - light sources send off photons in all directions and hits camera

- **Eye-oriented (Backward Raytracing or simply Raytracing)**
  - walk through each pixel looking for what object (if any) should be shown there

- **Object-oriented (OpenGL):**
 - walk through objects, transforming and then drawing each one unless the z-buffer says that it's not in front
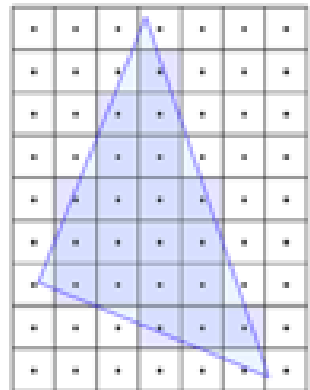
Let's leave rasterization to the GPU

OpenGL Programming Main Steps

- Initialize OpenGL (using GLUT, discussed later)
- Define the geometry (points lines Define the geometry (points, lines, triangles/polygons)
- Define the vertex attributes (color normal etc)
- Transform the geometry (translate, rotate, scale)
- Set up the camera (position direction angle etc)
- Set up lighting (light position/color etc)
- Set up textures
- Draw
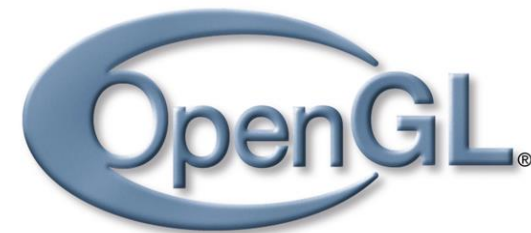
OpenGL
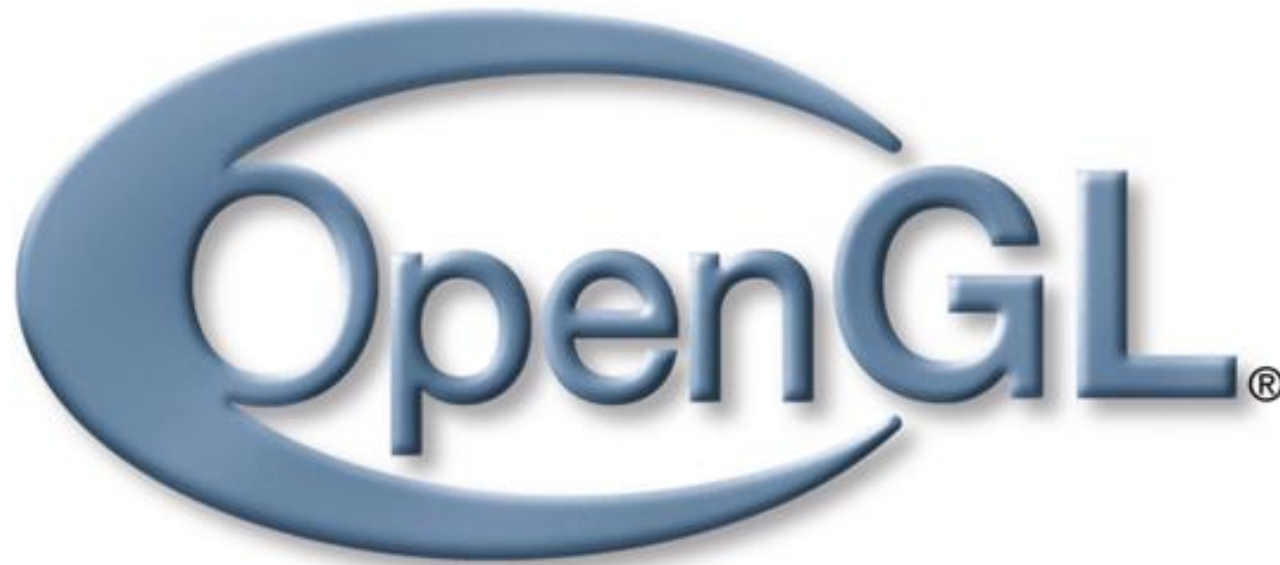
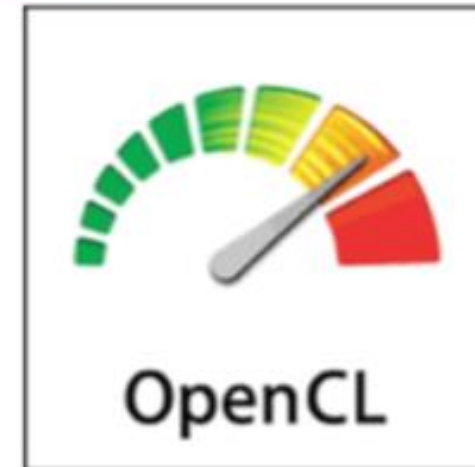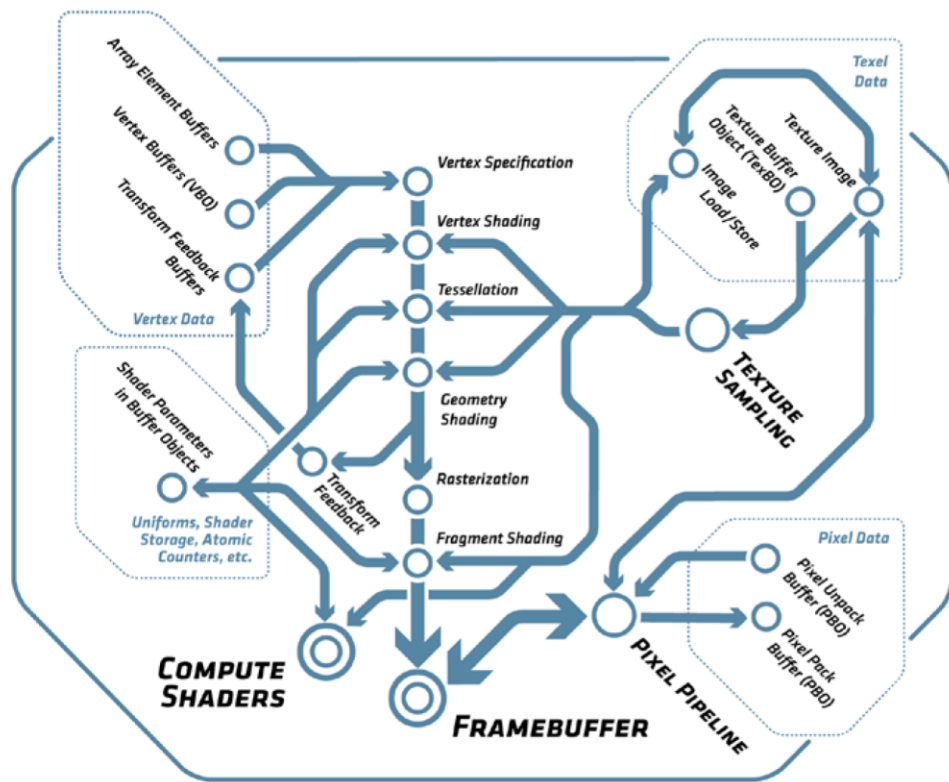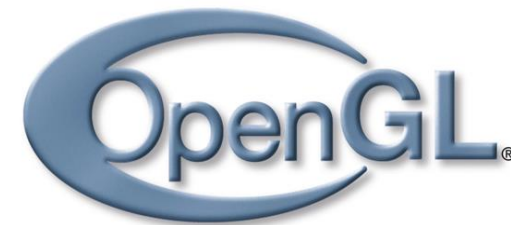- **Industry Standard API for Computer Graphics**

Alternatives

interactive, but not cross-platform

OpenGL Family

Continuing OpenGL Innovation

Bringing state-of-the-art functionality to cross-platform graphics

OpenGL 4.5

OpenGL 4.4

OpenGL 4.3

OpenGL 4.2

OpenGL 4.1

OpenGL 3.3/4.0

OpenGL 3.2

OpenGL 3.1

OpenGL 2.0   OpenGL 2.1   OpenGL 3.0

| 2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014
DirectX 9.0c		DirectX 10.0		DirectX 10.1		DirectX 11		DirectX 11.1		DirectX 11.2

Descendent of GL (from SGI) since 1992

What is OpenGL

- **Low-level**
  - A software interface to graphics hardware that consists of about 250 distinct functions

- **System-independent**
  - Designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms

- Client-Server abstraction
  - Client is the program which sends commands to the server
  - Server (graphics card) produces pixels on the screen
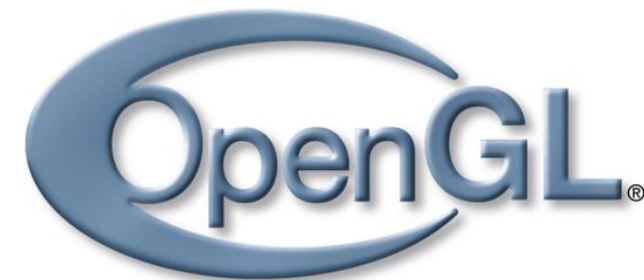
Where is OpenGL used?

- CAD

- VR
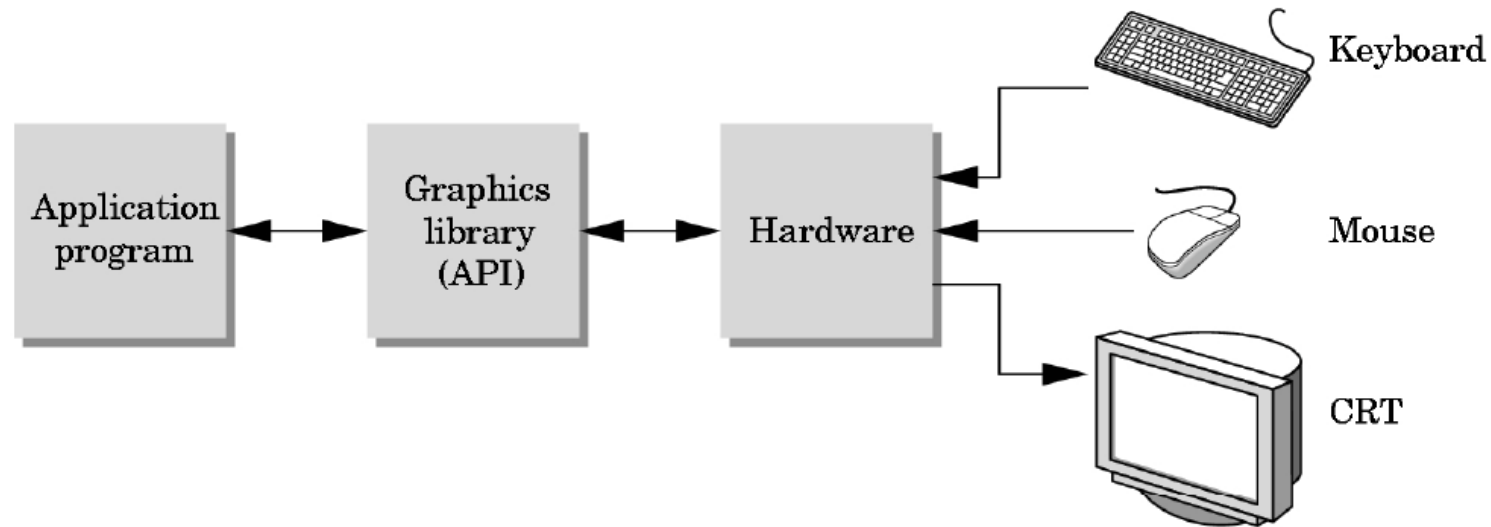
- Scientific Visualization

- Simulators

- Video games
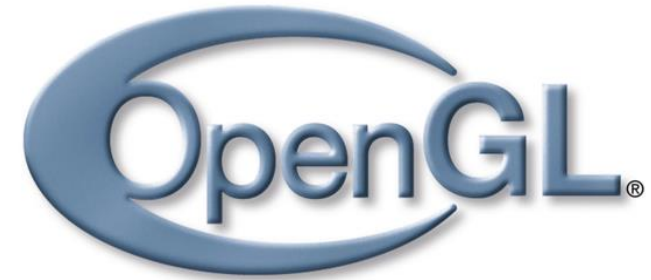
OpenGL®
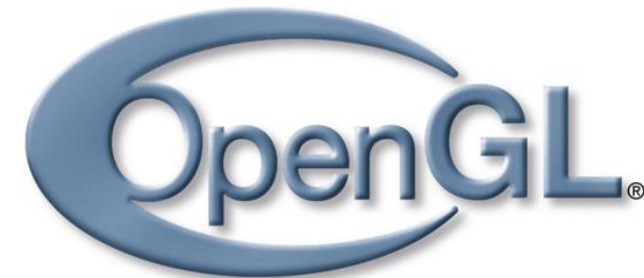
Realtime Graphics Demo

- **Realtime rendering:** Unreal Kite Demo
- **Smoke simulation:** Efficient Smoke Simulation on Curvilinear Grids-HD

Graphics Library (API)

- **Interface** between Application and Graphics Hardware

- Other popular APIs:
  - Direct3D (Microsoft) → XBox
  - OpenGL ES (embedded Devices)
 - X3D (successor of VRML)

OpenGL is cross-platform

- **Same code** works with little/no modifications
- **Implementations:**
  - Mac, Linux, Windows: ships with the OS
  - Linux: Mesa, freeware implementation

```
#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
```
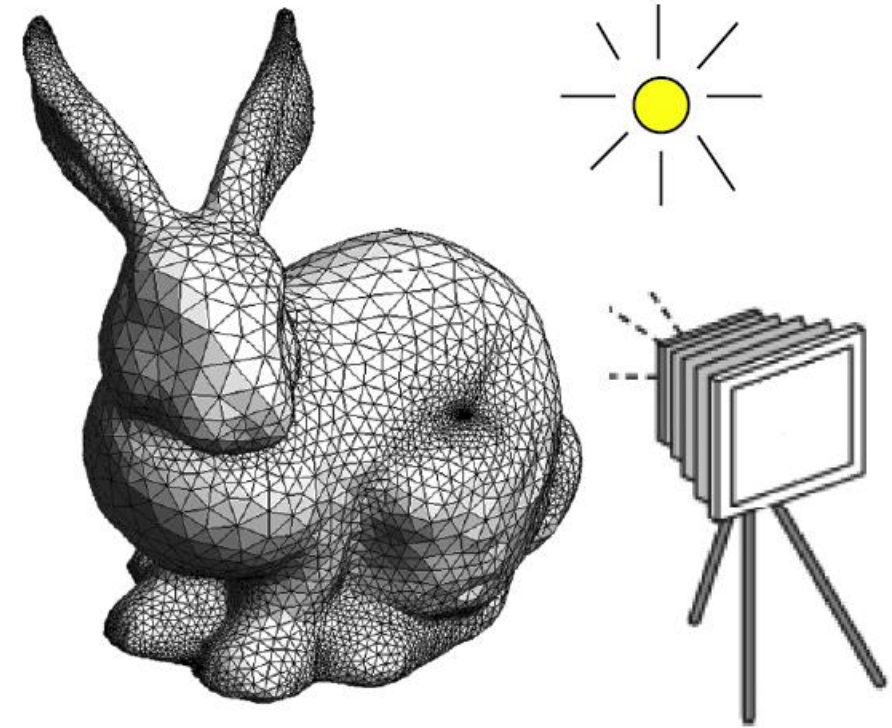
How does OpenGL work

- **From the programmer's point of view:**
  - Specify **geometric objects**
  - Describe **object properties**
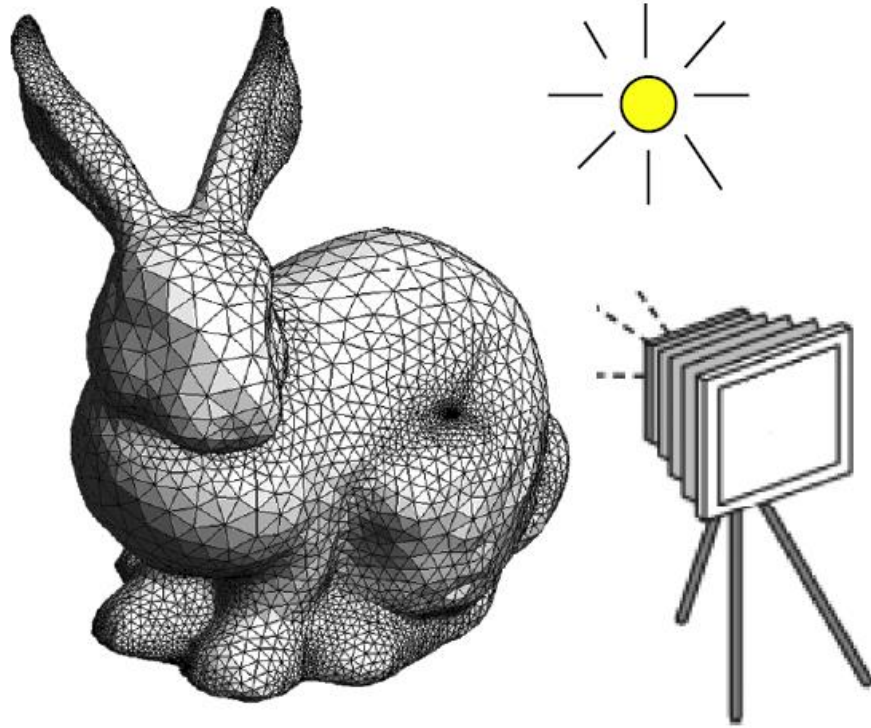   - Color
    - How objects reflect light
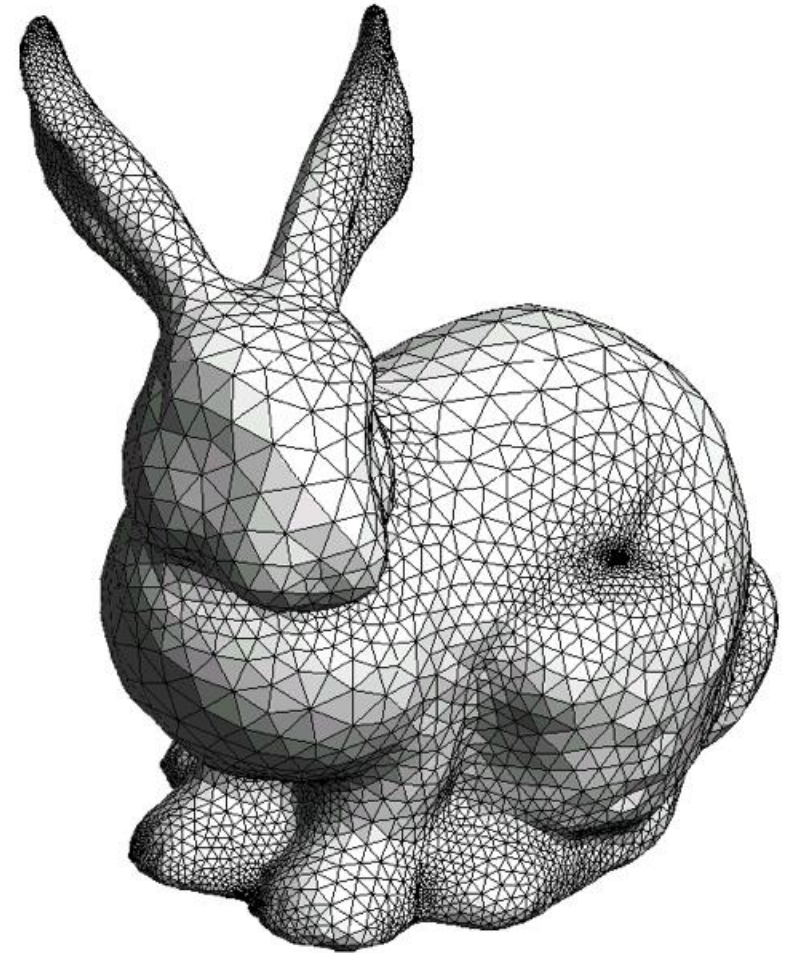
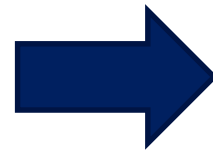# How does OpenGL work (continued)

- **Define how objects should be viewed**
  - where is the camera?
  - what type of camera?
- **Specify light sources**
  - where, what kind?
- **Move camera or objects around for animation**

# The result

the scene

the result

OpenGL is a state machine

- State variables:
  - color, camera position, light position, material properties, model transformation, …

- These variables (state) then apply to every subsequent drawing command

- Function calls
  - No data structures

- They persist until set to new values by the programmer
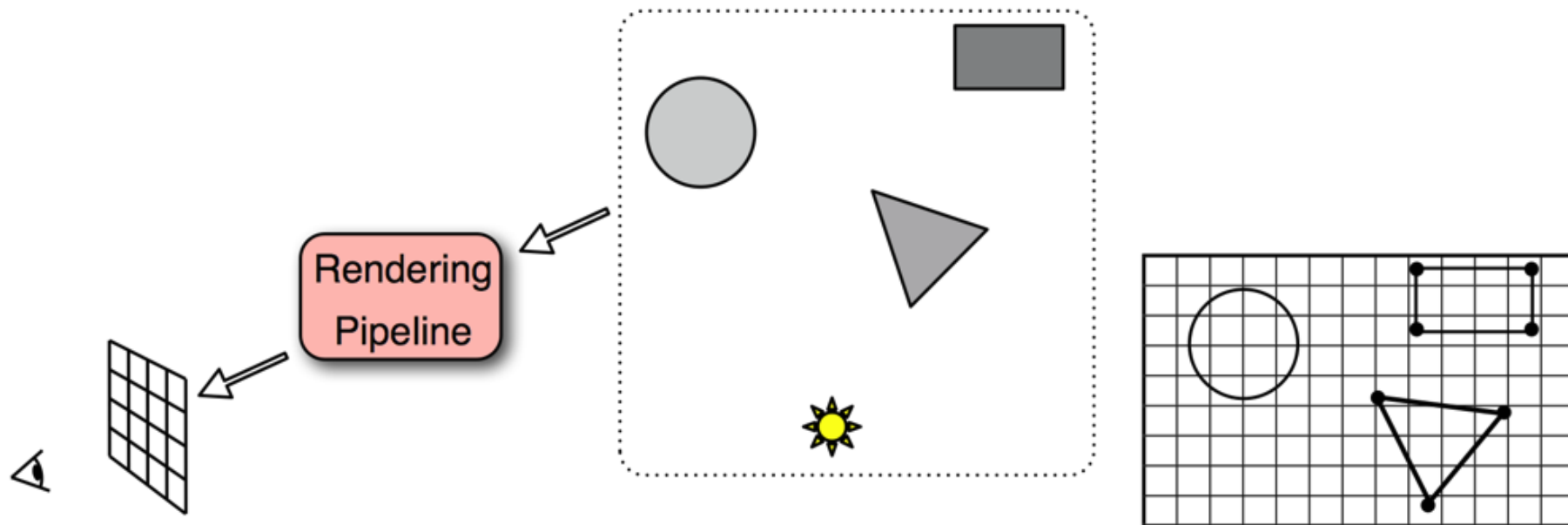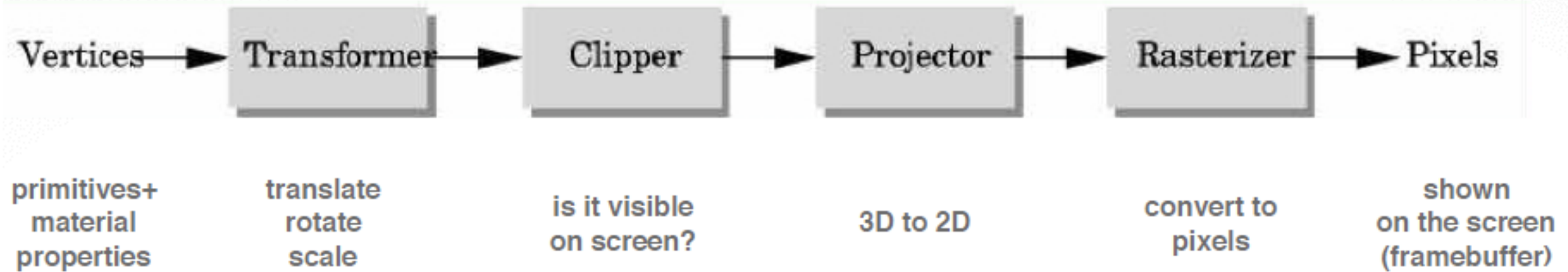
OpenGL Library Organization

- **GL (Graphics Library):** core graphics capabilities
- **GLU (OpenGL Utility Library):** utilities on top of GL
- **GLUT (OpenGL Utility Toolkit):** input and windowing wrapper
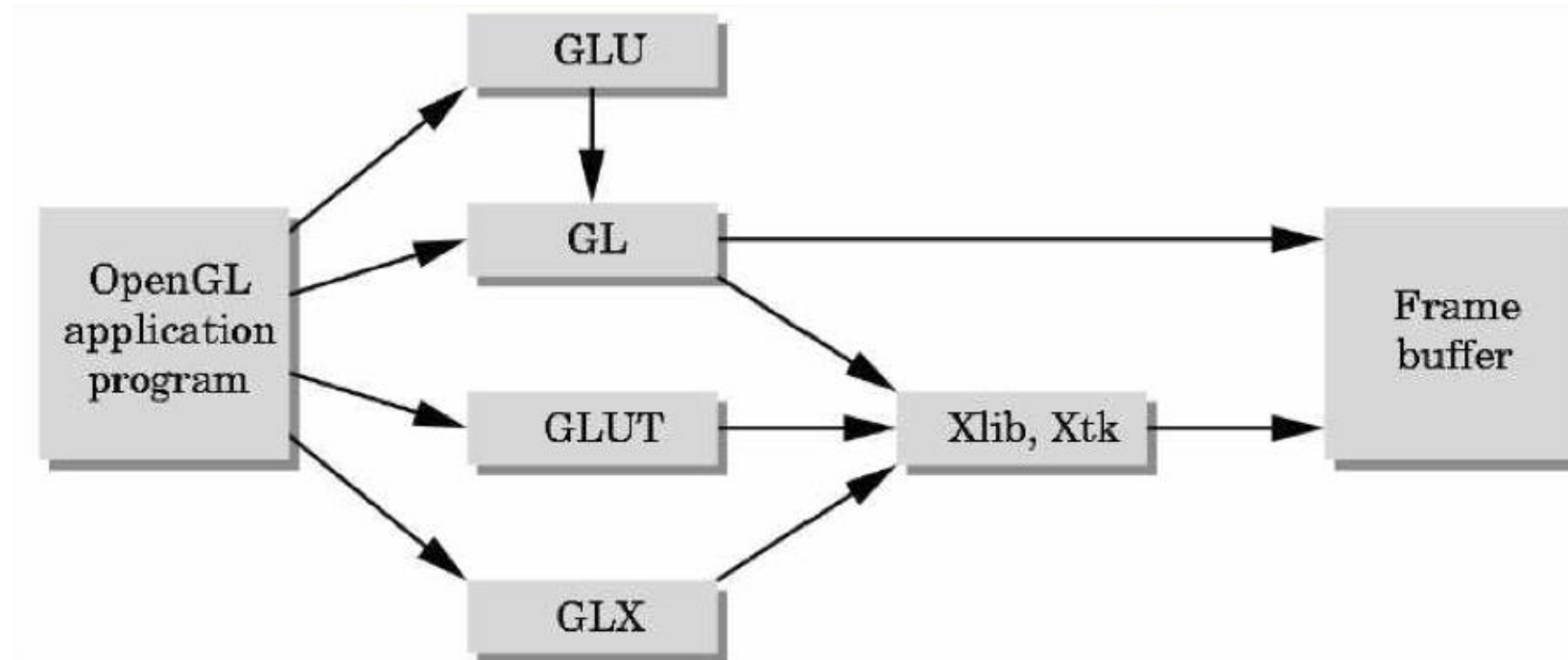
OpenGL Command Syntax

- Constants begin with GL_ and are in capital letters
  - GL_LIGHTING,
  - GL_SMOOTH, etc

- Commands have prefix gl and initial capital letters for each word
  - glEnable(),
  - glDisable(), etc

- Some commands contain extra letters which indicate the number and type of variables
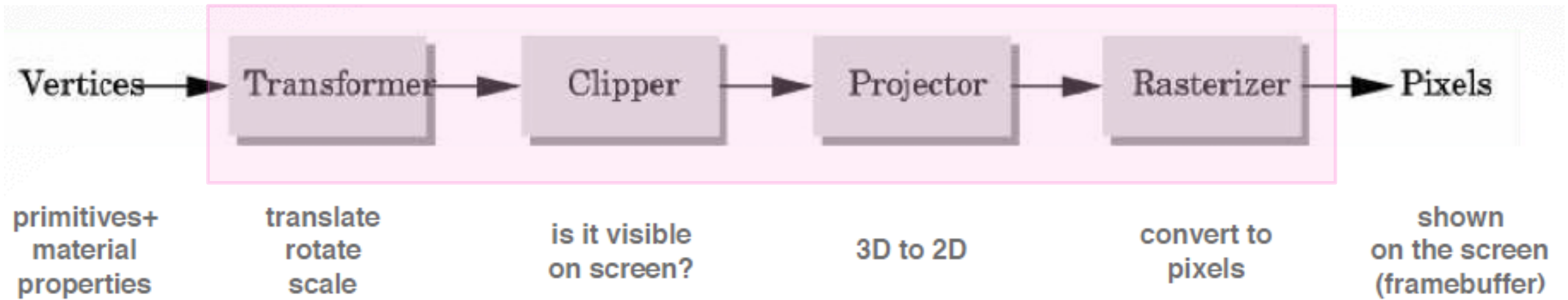 - glColor3b(), glColor3i(), glColor3f(), etc

OpenGL Graphics Pipeline

Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels

| primitives+ material properties | translate rotate scale | is it visible on screen? | 3D to 2D | convert to pixels | shown on the screen (framebuffer) |

OpenGL uses immediate-mode rendering

- Application generates **stream of geometric primitives** (polygons, lines)

- System **draws** each one into the **frame buffer**

- Entire scene is **redrawn** for every frame

- **Compare to**: offline rendering (e.g., Pixar Renderman, ray tracers…)

OpenGL **Graphics Pipeline**

Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels

| primitives+ material properties | translate rotate scale | is it visible on screen? | 3D to 2D | convert to pixels | shown on the screen (framebuffer) |

- implemented by **OpenGL, graphics driver, graphics hardware**

- OpenGL programmer does not need to implement the pipeline, but can **reconfigure it through shaders**

Vertices

Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels
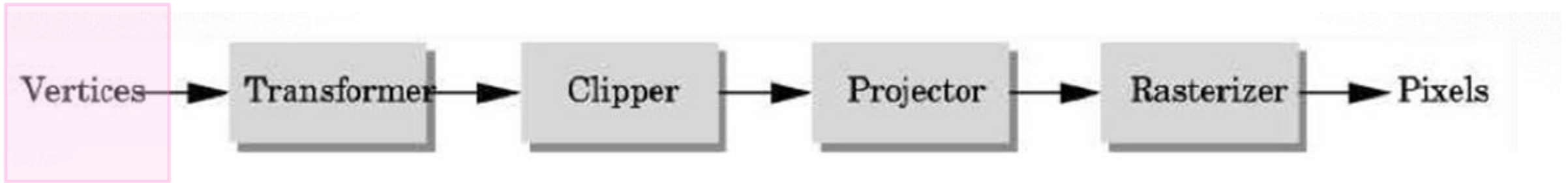
- Vertices in **world coordinates**

- **void glVertex3f(GLfloat x, GLfloat y, GLfloat z)**
  - Vertex(x,y,z) is sent down the pipeline.
  - Function call then returns

- Use **GLtype (e.g., GLfloat)** for portability and consistency

- glVertex{234}{sfid}(TYPE coords)

Transformer

Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels
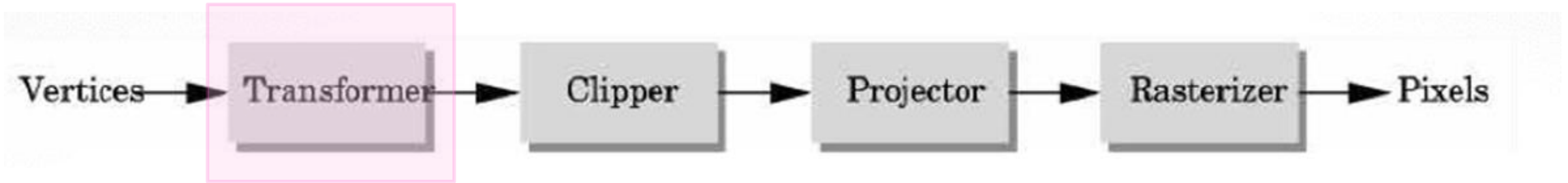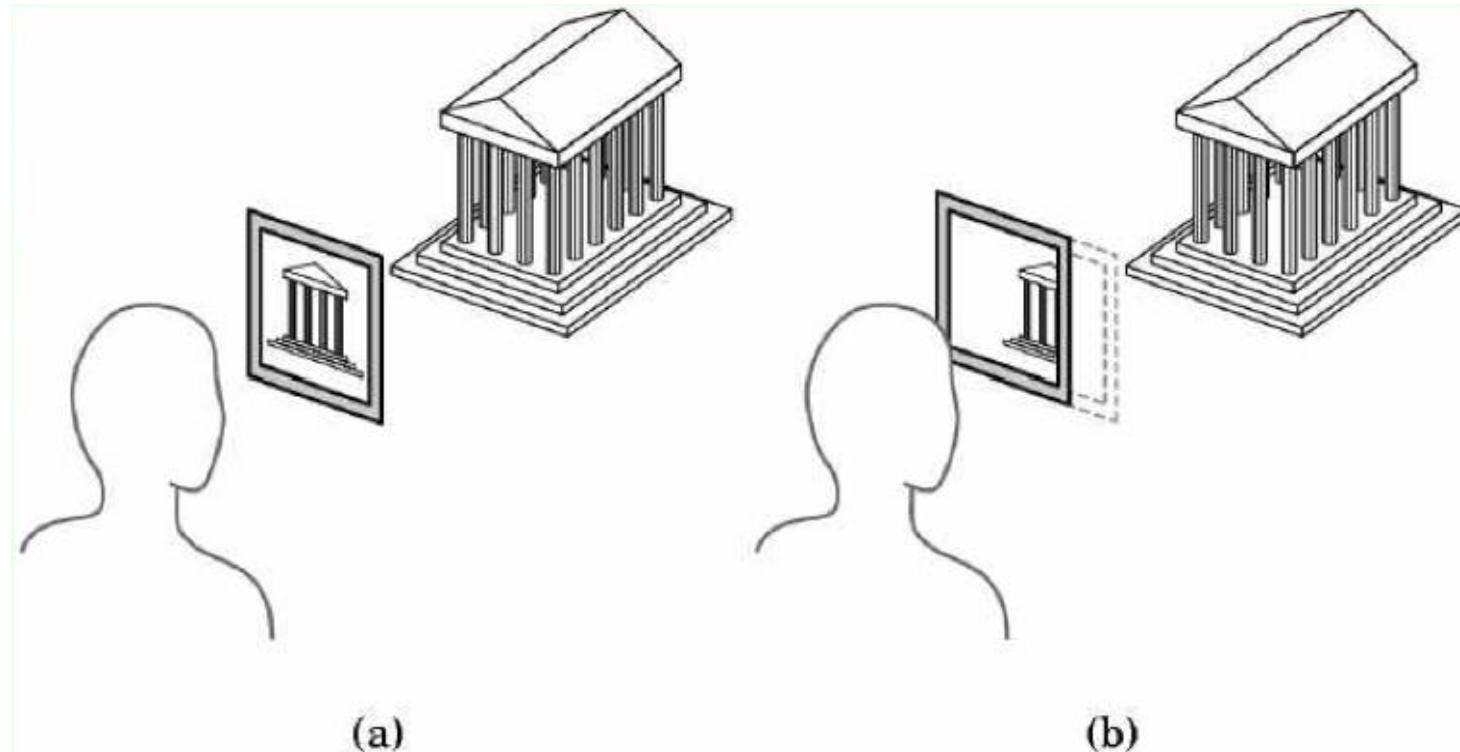
- Transformer in **world coordinates**

- **Must be set before object is drawn!**
  - glRotate (45.0, 0.0, 0.0, -1.0);
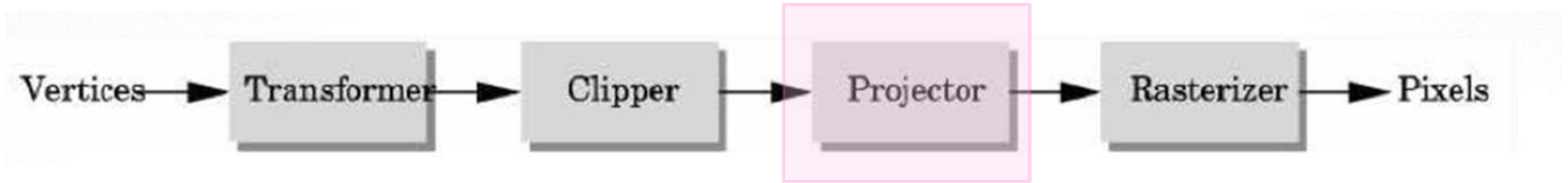 - glVertex2f(1.0, 0.0);

- Complex [Angel Ch. 4]

Clipper

Vertices → Transformer → **Clipper** → Projector → Rasterizer → Pixels
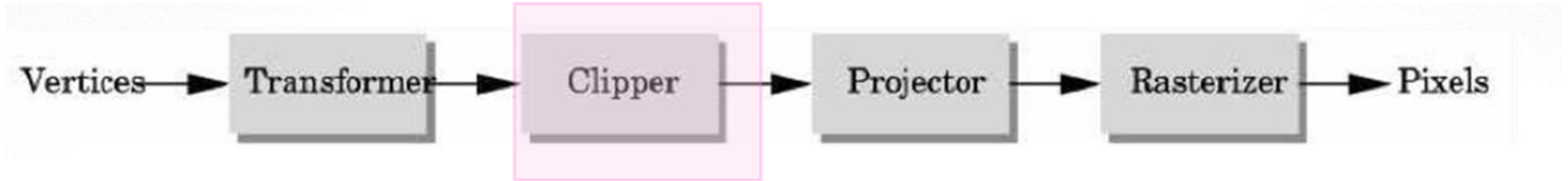
- Mostly automatic (**must set viewport**)
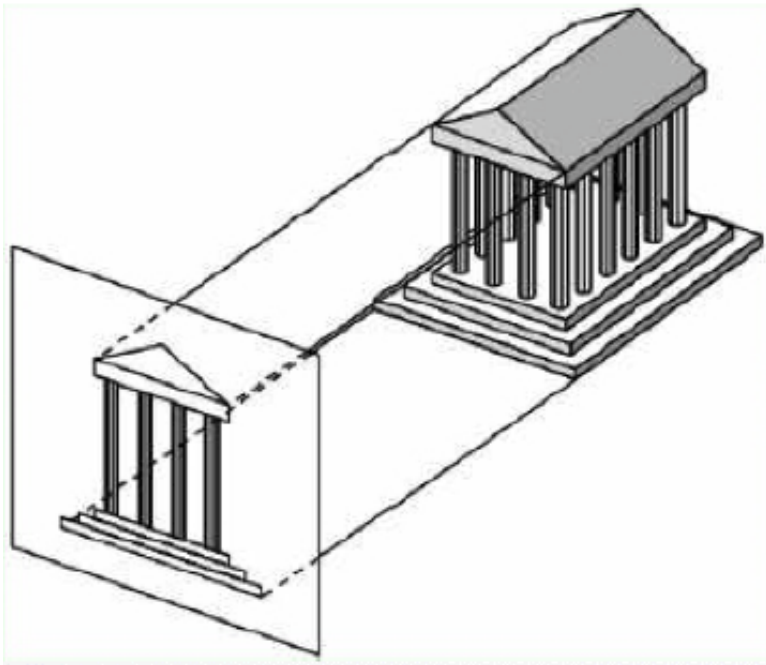

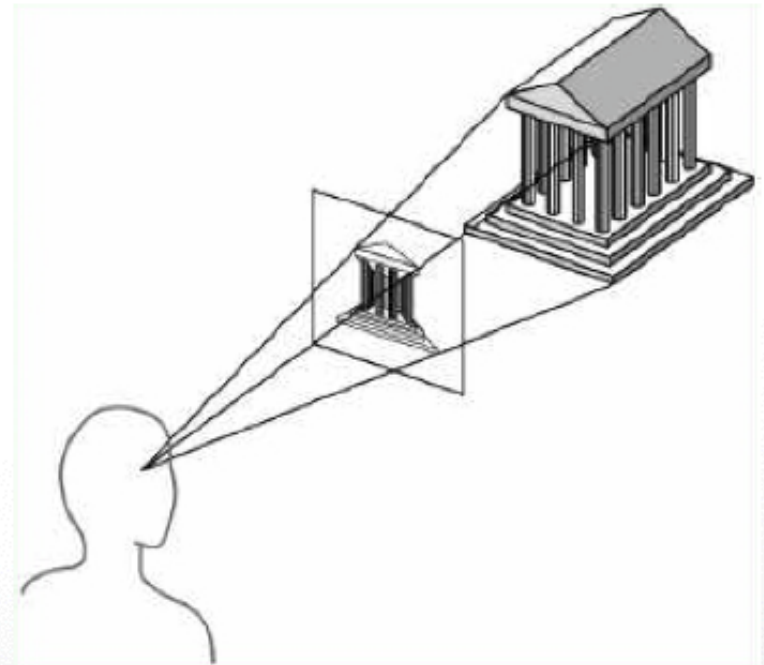(a)   (b)

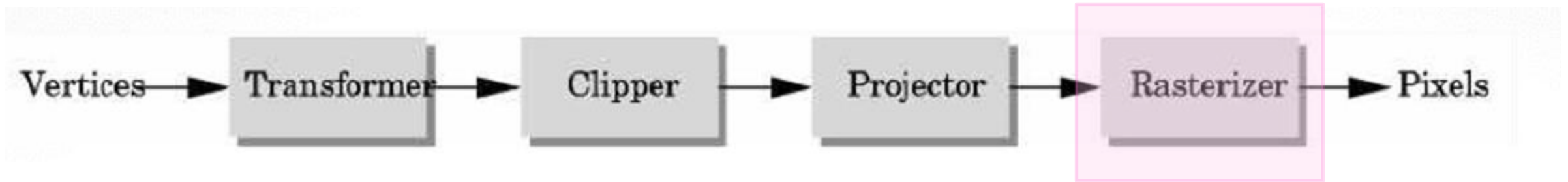# Projector

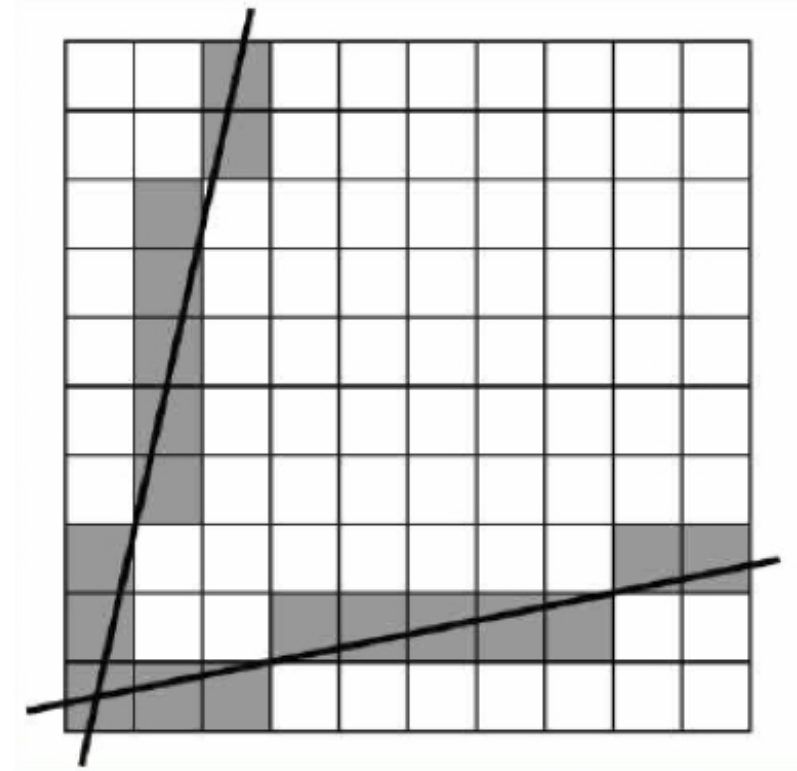- Complex transformation [Angel Ch. 5]

orthographic · perspective

Rasterizer

- Interesting algorithms [Angel Ch. 7]

- **To window coordinates**

- Antialiasing

- Thank you

Reference

- **_Learning Modern 3D Graphics Programming_**
- http://www.songho.ca/opengl/index.html

Fotis Liarokapis   Building Virtual Environments with OpenGL
[Nate Robins]   http://www.morrowland.com/apron/tut_gl.php
[Nehe]   http://nehe.gamedev.net/
[Red book]   OpenGL Programming Guide: The Official Guide to Learning OpenGL, Latest Version