Computer Graphics
- Introduction of OpenGL

Junjie Cao @ DLUT

Spring 2017

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

Last Time

Last Time

Story → Computer Graphics → Image

What is computer graphics?

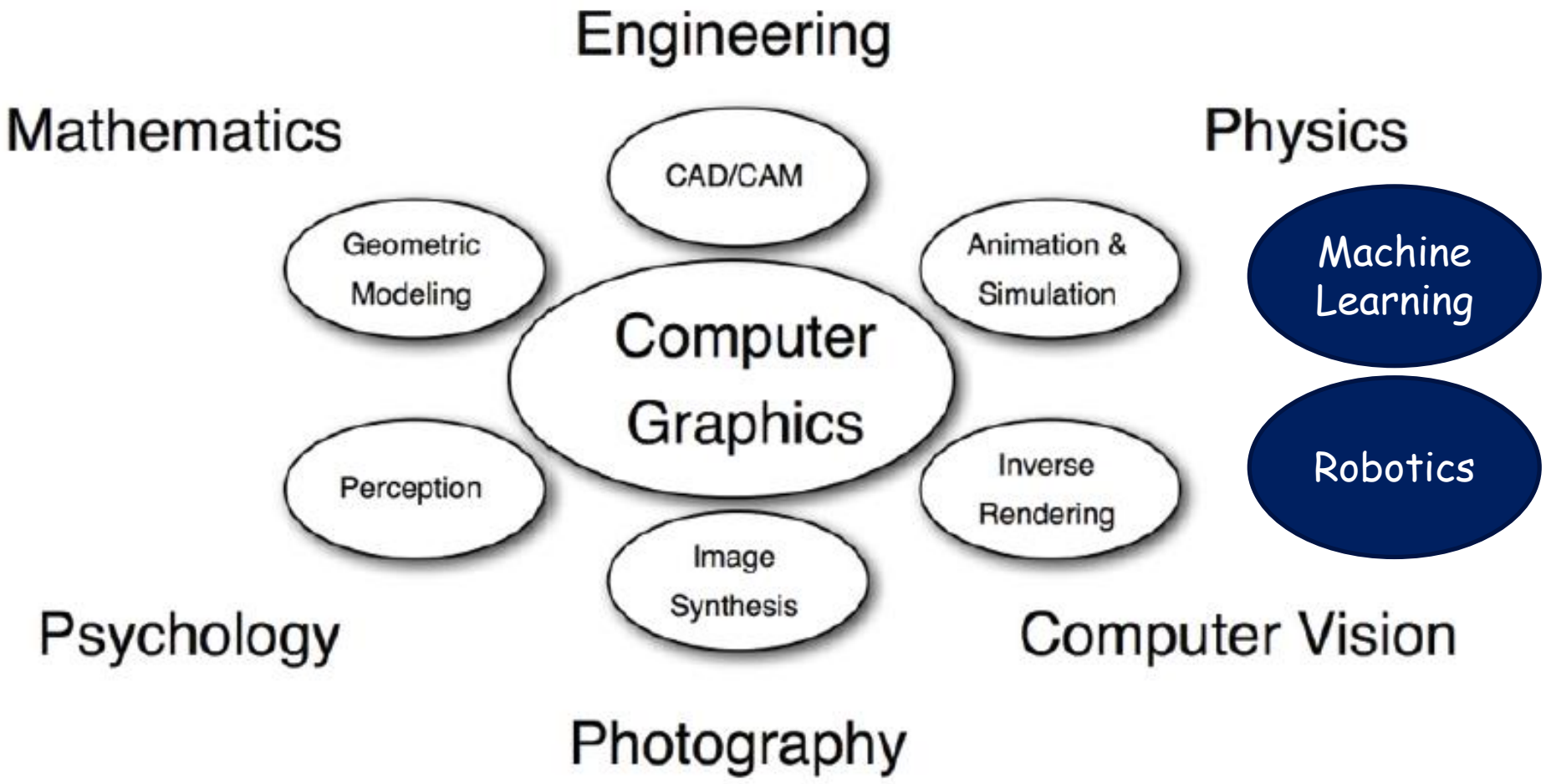
- The use of computers to synthesize and manipulate **visual information**.

- The use of computers to synthesize and manipulate **ser**

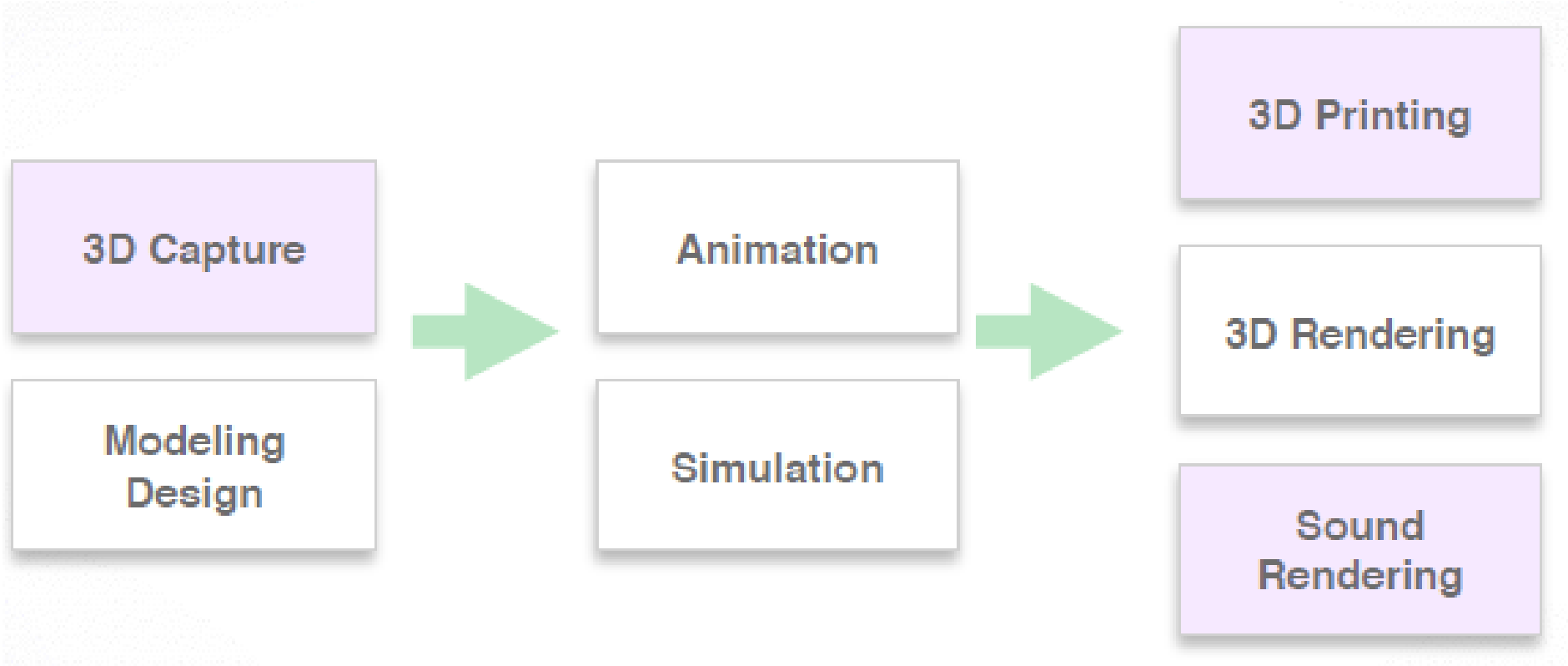
(sound)

(touch)

Related to many Disciplines



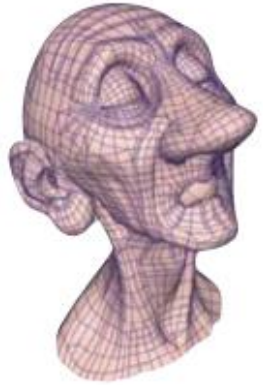
3D Computer Graphics Pipeline



Emerging Fields

Render [ren-der]

- OpenGL's primary function – Rendering
- Rendering?
  - converting geometric/mathematical object descriptions into frame buffer values, i.e. pixel array



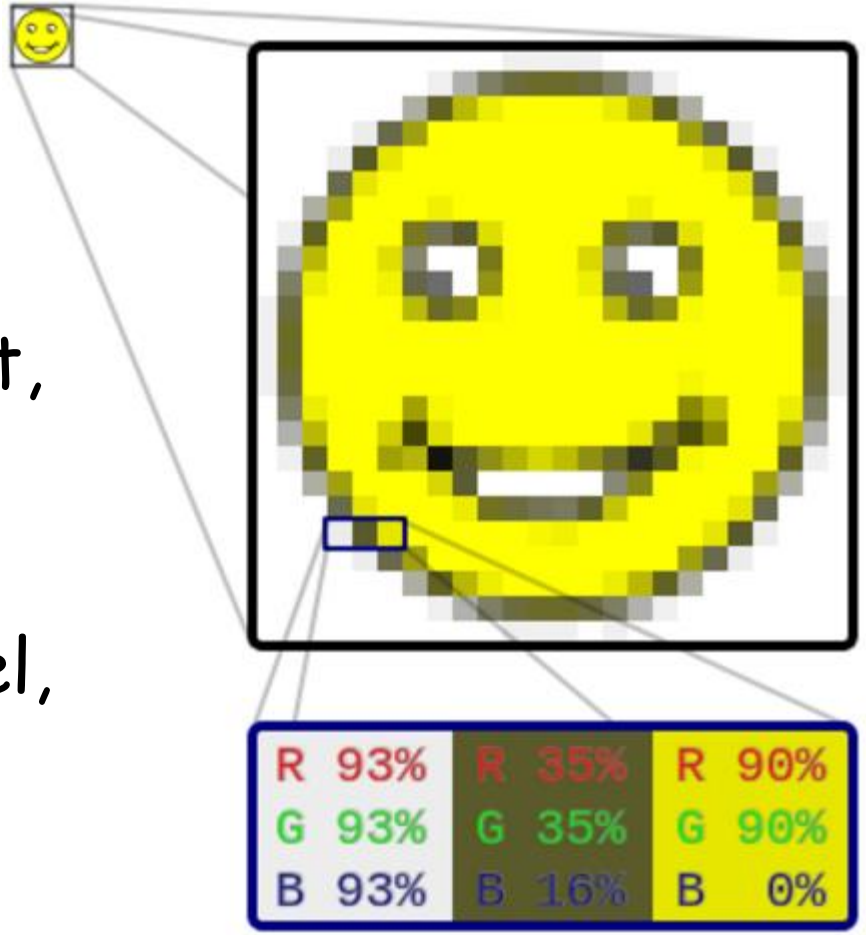
input data

output rendering

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

Output: 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



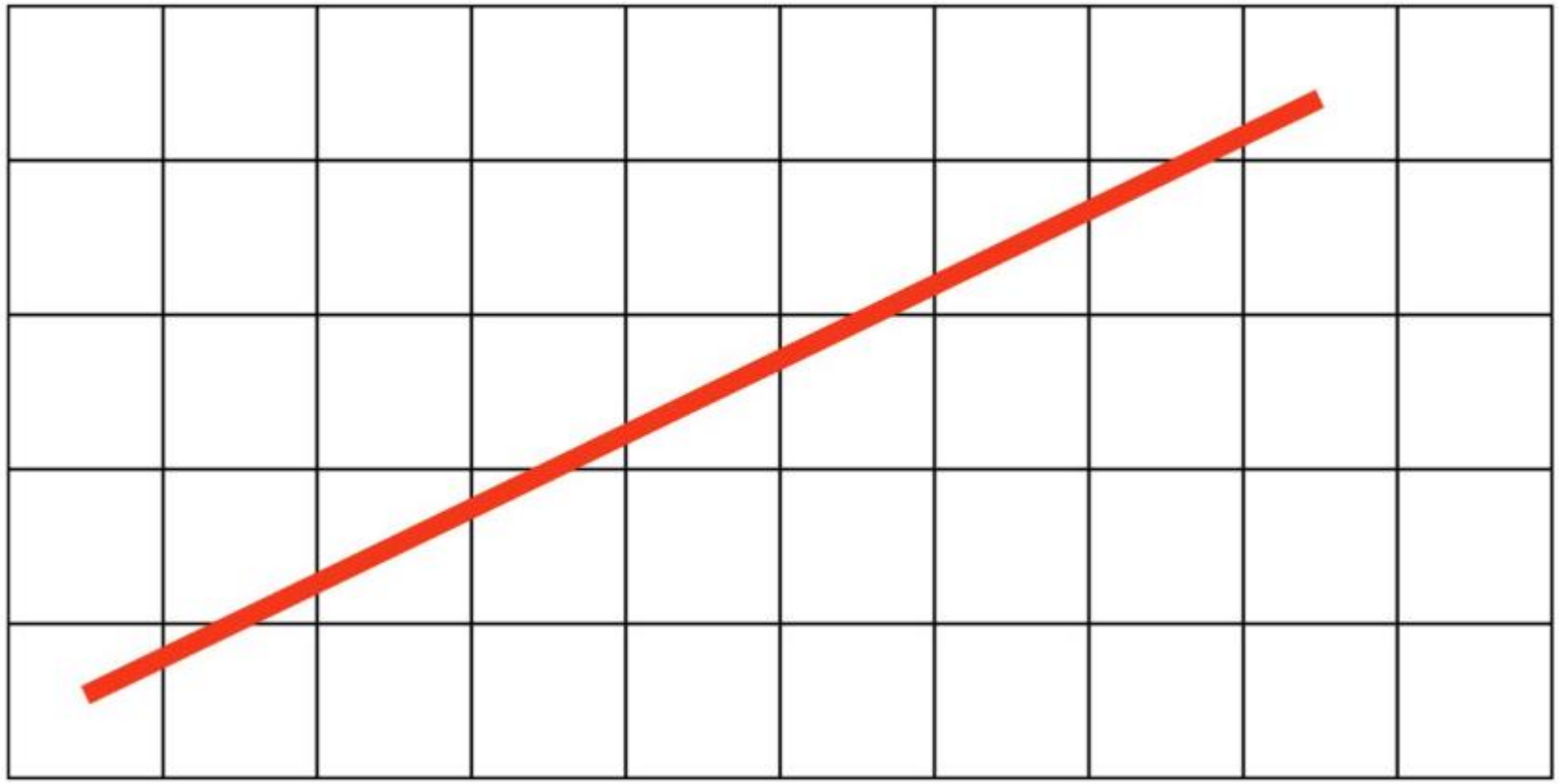
How to make an image?

drawing

photography

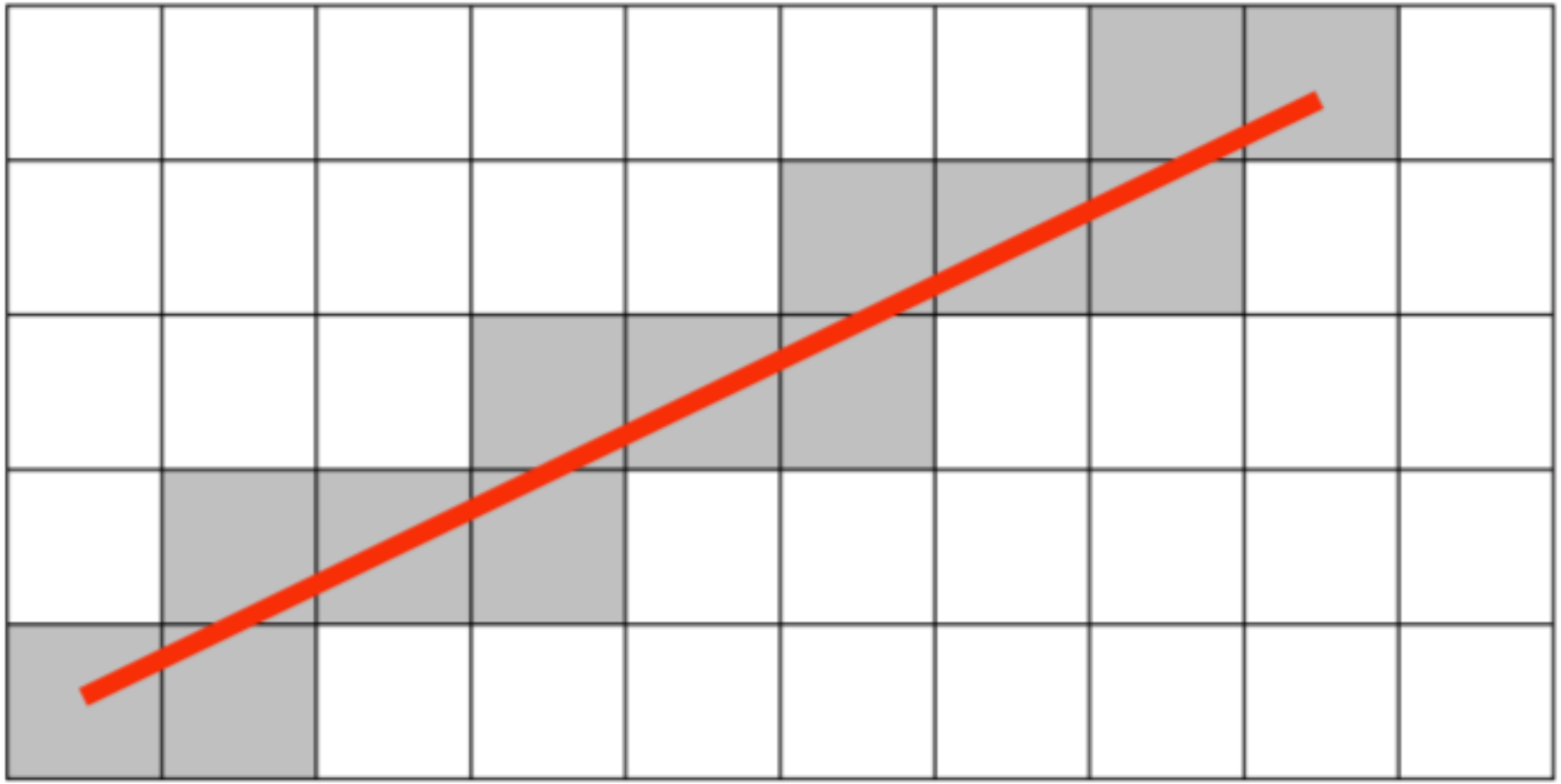
What pixels should we color in to depict a line?

- "Rasterization": process of converting a continuous object to a discrete representation on a raster grid (pixel grid)



What pixels should we color in to depict a line?

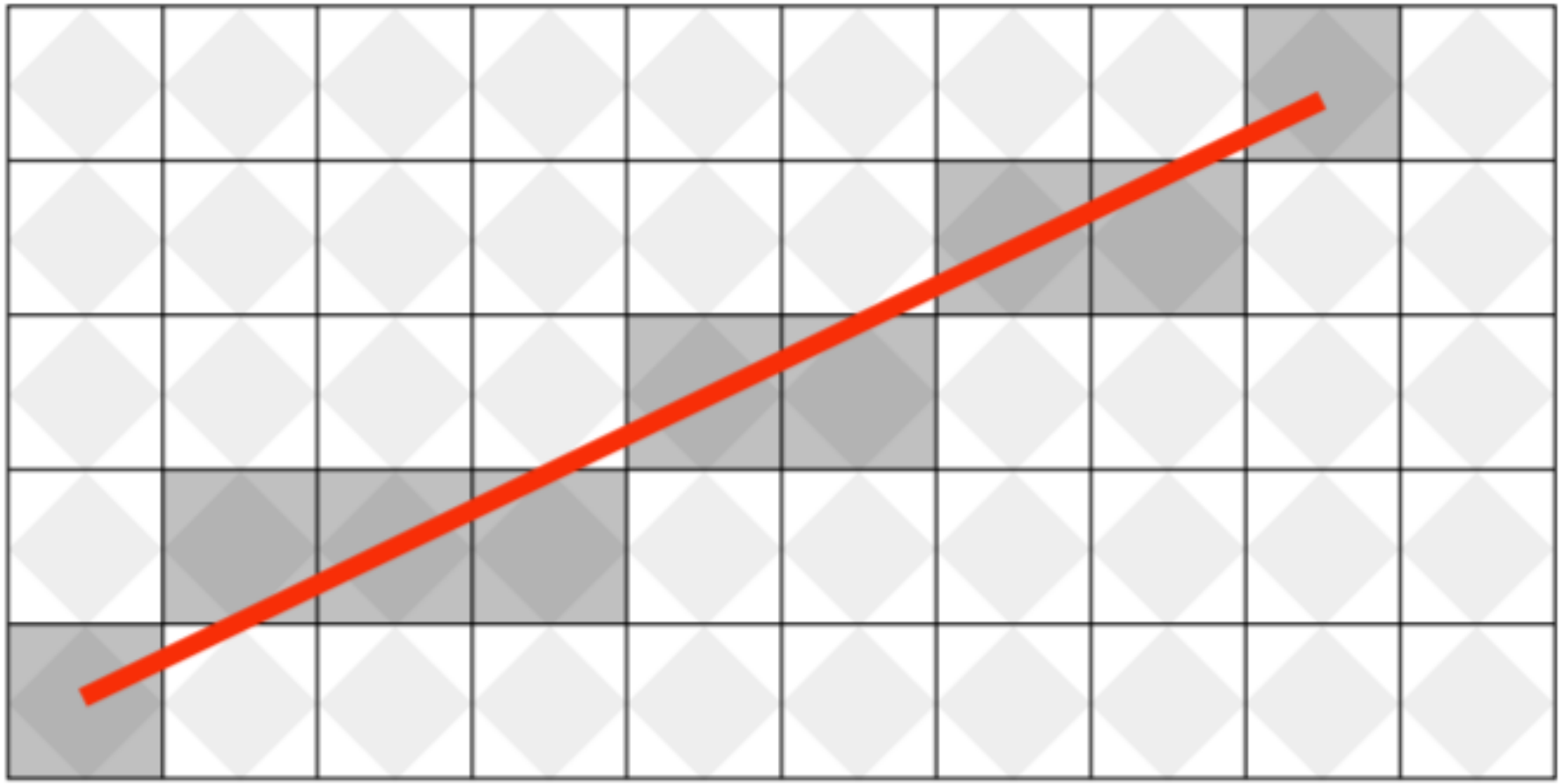
- Light up all pixels intersected by the line?



What pixels should we color in to depict a line?

Diamond rule (used by modern GPUs):

light up pixel if line passes through associated diamond



How do we find the pixels satisfying a chosen rasterization rule?

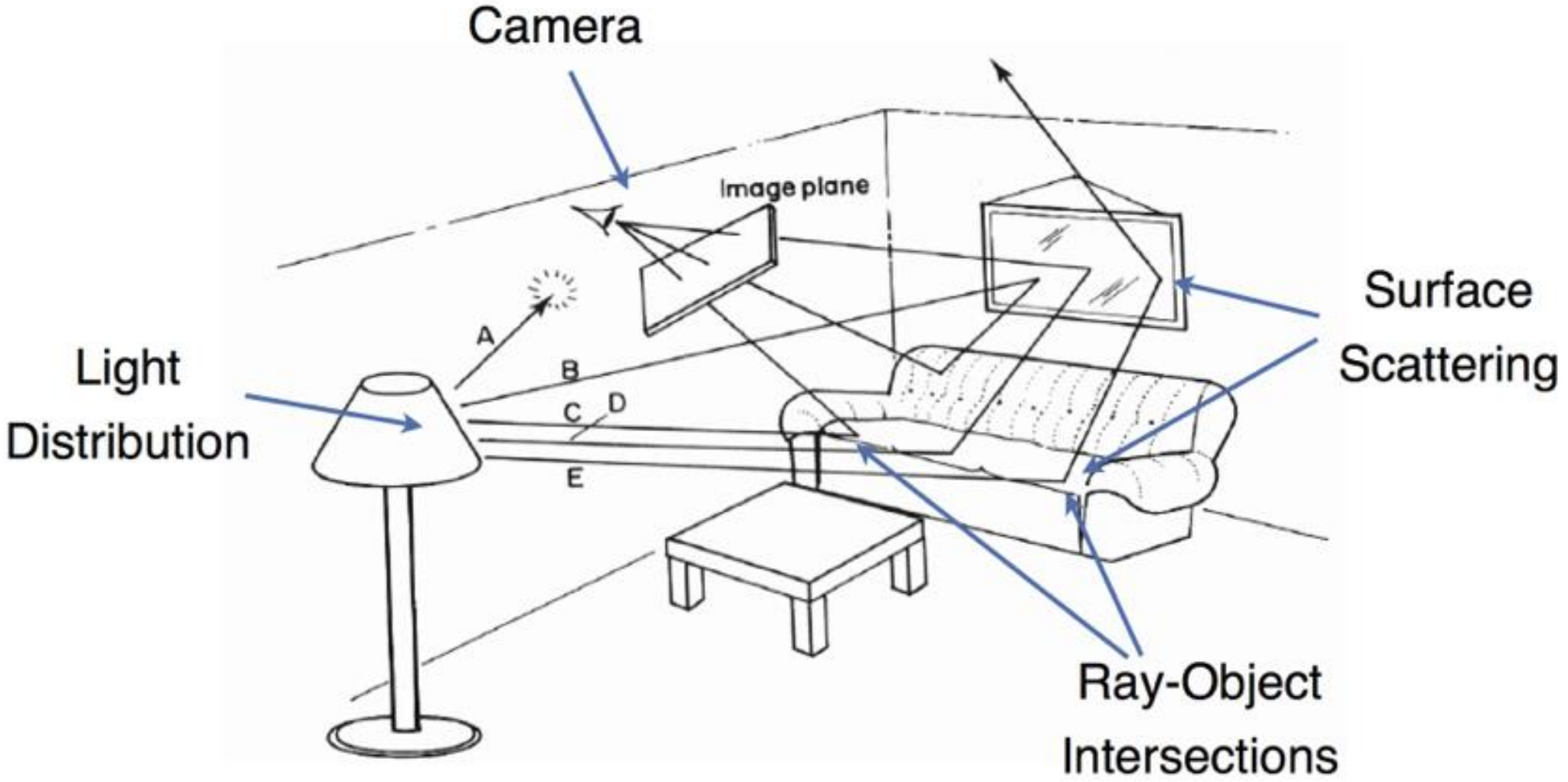
- Could check every single pixel in the image to see if it meets the condition...
  - O(n2) pixels in image vs. at most O(n) "lit up" pixels
  - must be able to do better! (e.g., work proportional to number of pixels in the drawing of the line)
  - Back to this later

Okay… let's take a step back

2d Vector image => Bitmap / raster image

? => vector image

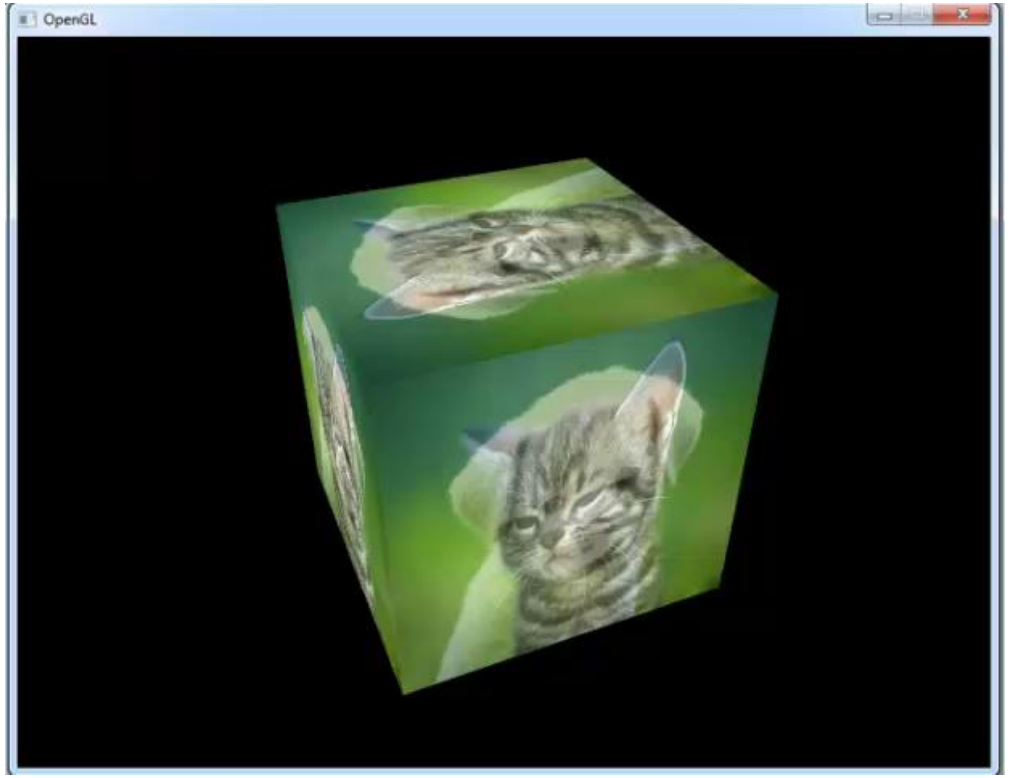
In the physical world



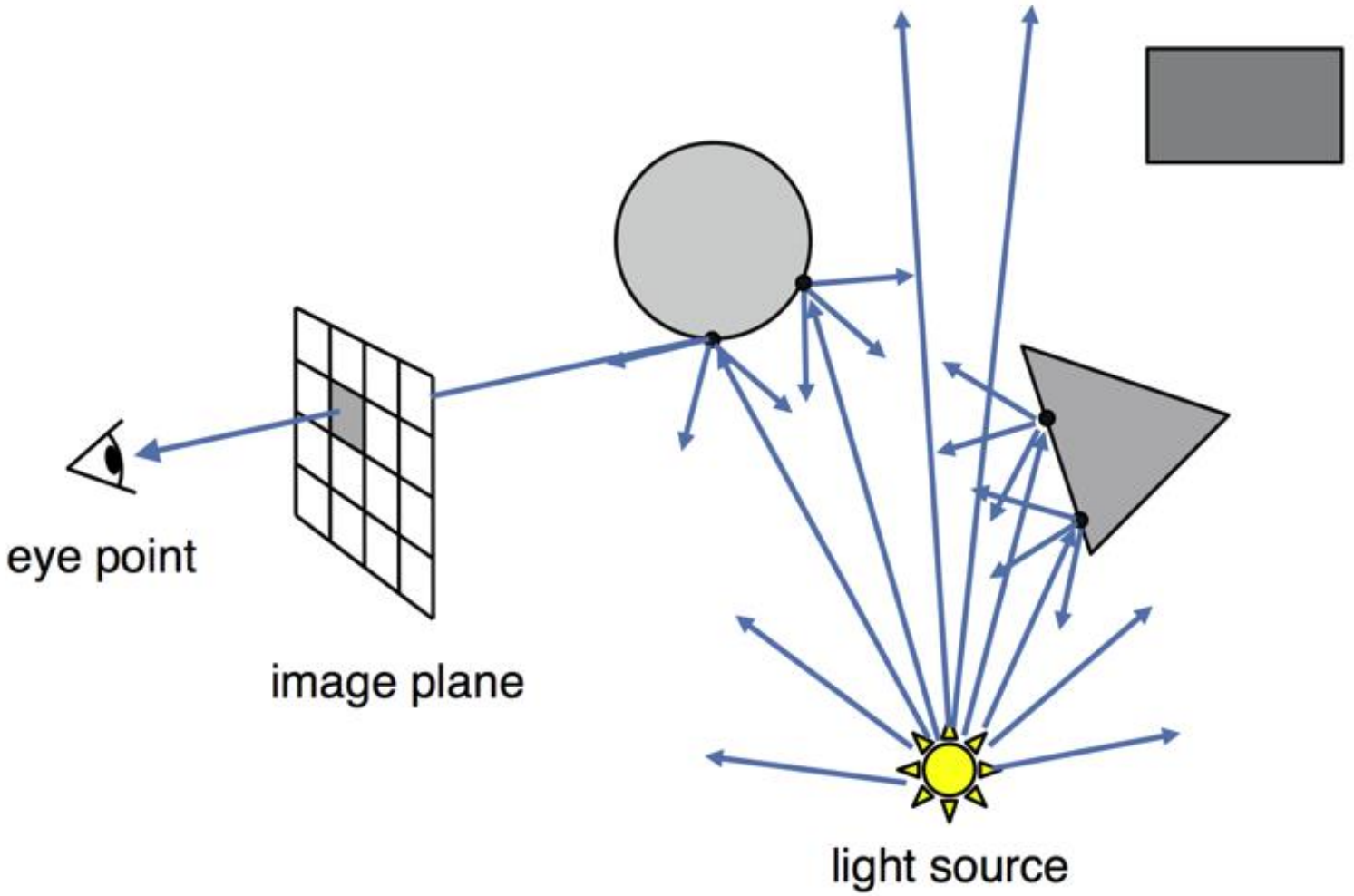
Light Transport

- 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**

Without & with Hidden Surface Removal

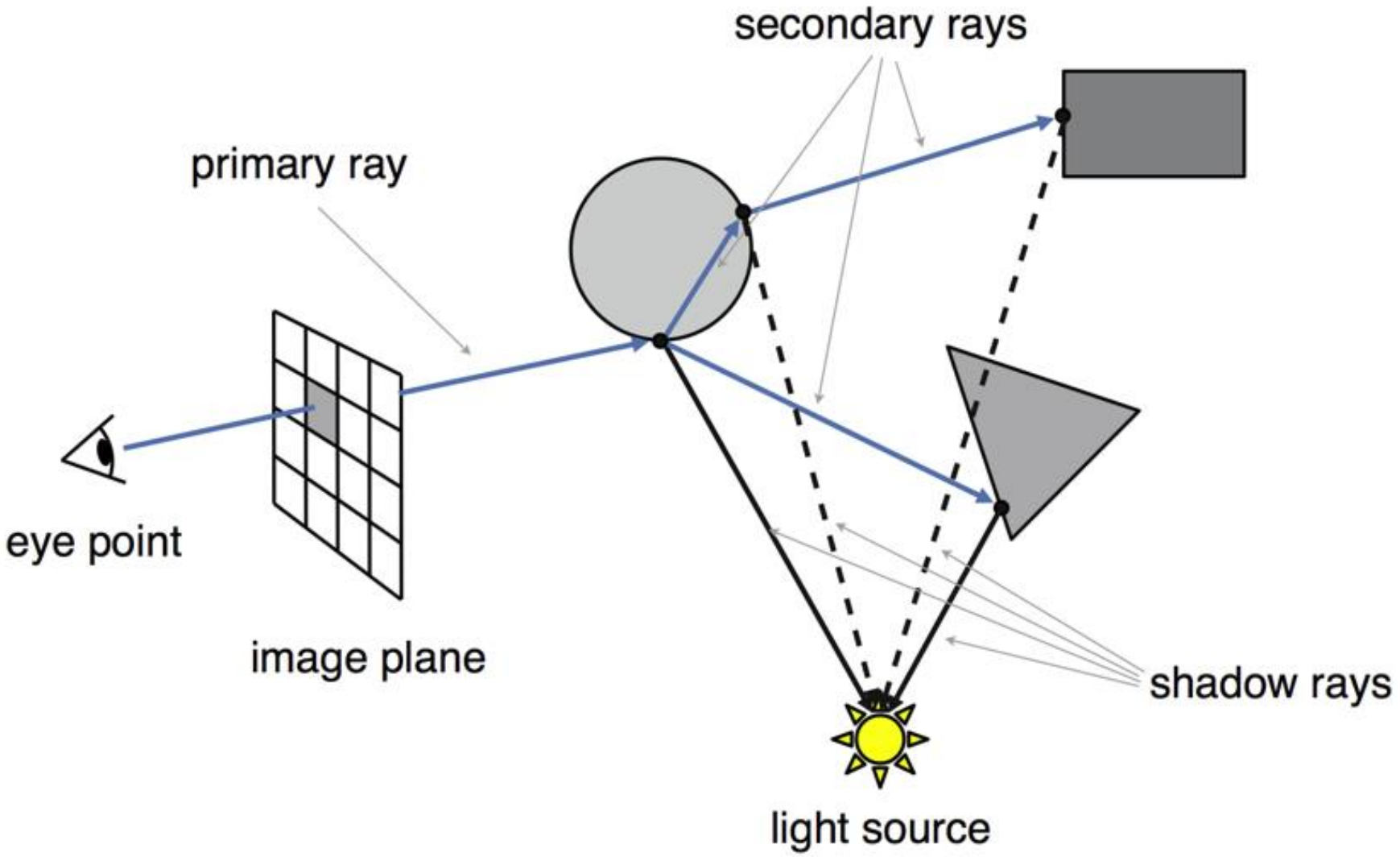


Light-Oriented (Forward Raytracing)



Only a `fraction` of light rays reach the image

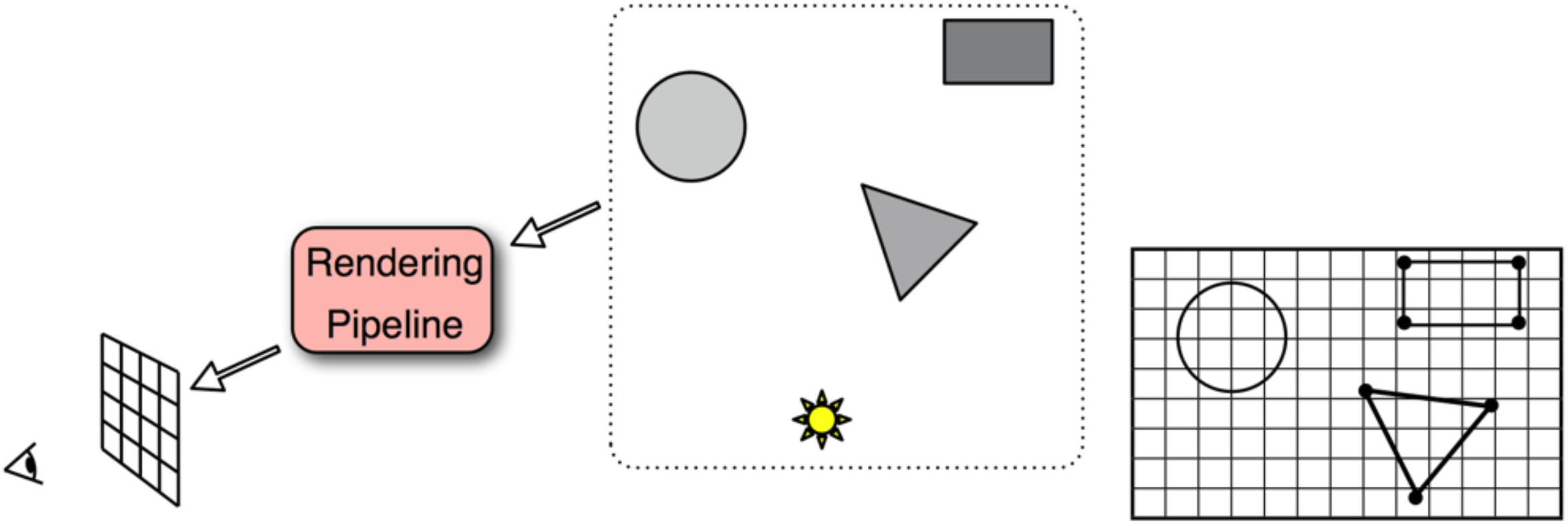
Eye-Oriented (Backward Raytracing)



or simply “Raytracing”

Object-Oriented (Forward Rendering)

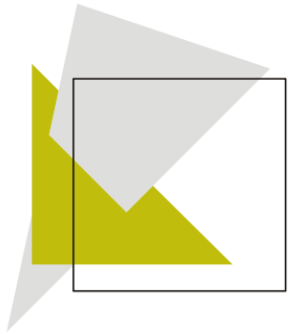
Back to this later

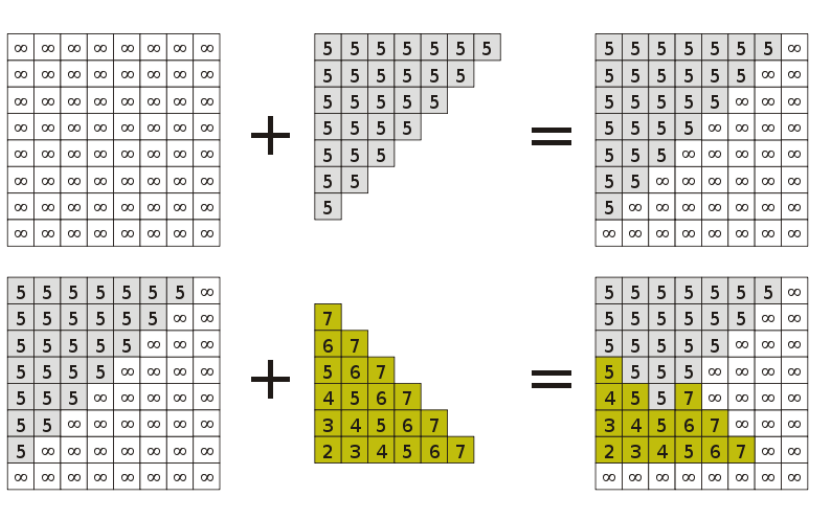


Scene is composed of `geometric structures` with the building block of `a triangle`. Each triangle is projected, colored, and painted on the screen

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):** Back to this later
  - walk through objects, transforming and then drawing each one unless the z-buffer says that it's not in front

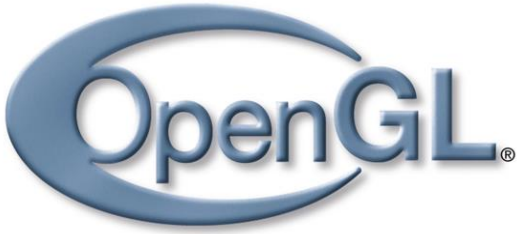


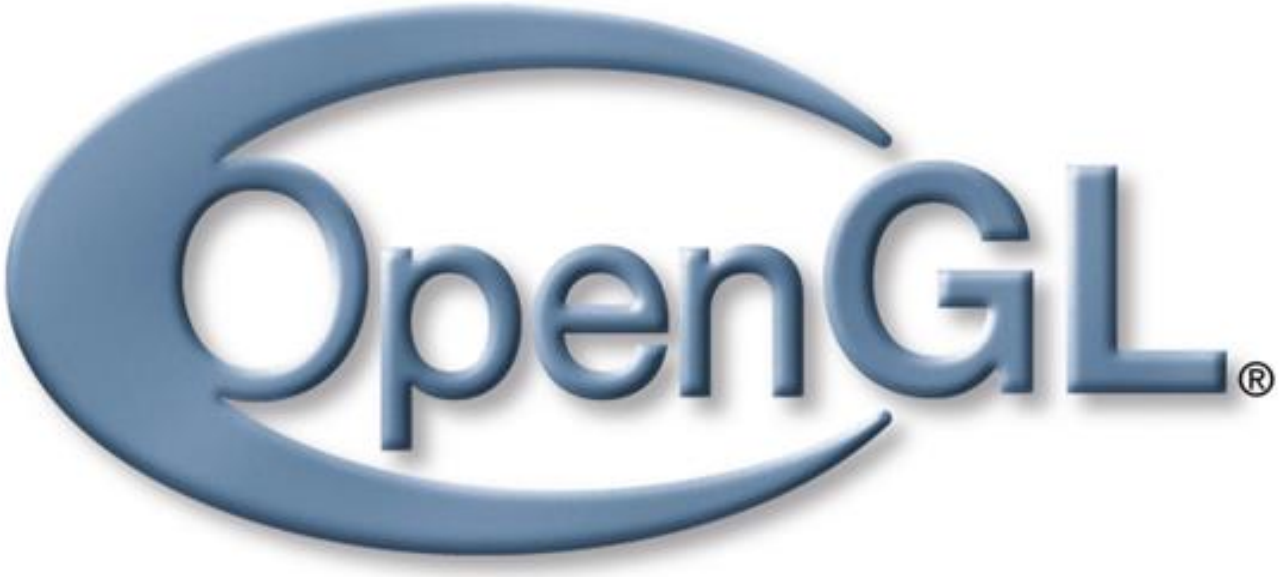


Let's leave rasterization to the GPU

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





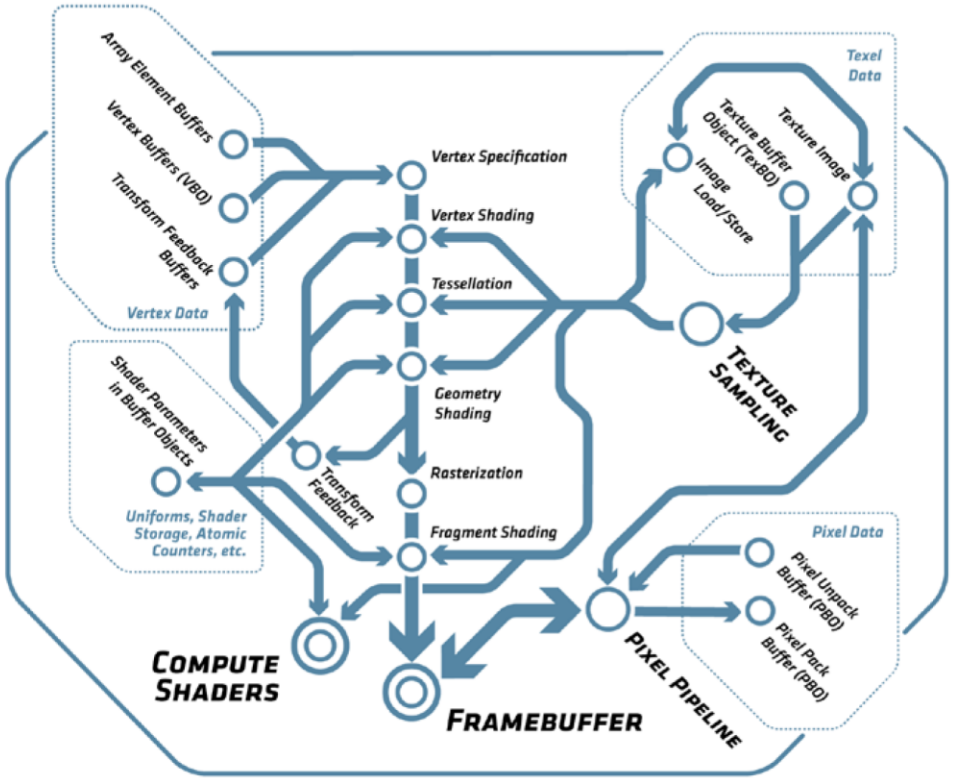
- **Industry Standard API for Computer Graphics**

Alternatives

interactive, but not cross-platform

OpenGL Family

Continuing OpenGL Innovation



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

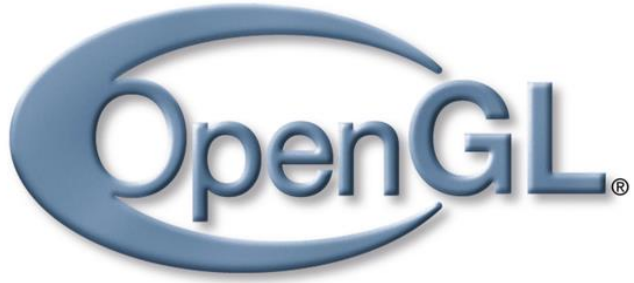
OpenGL 4.5
OpenGL 4.4
OpenGL 4.3
OpenGL 4.2
OpenGL 4.1
OpenGL 3.3/4.0
OpenGL 3.2
OpenGL 3.1
OpenGL 2.0
OpenGL 2.1
OpenGL 3.0

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?

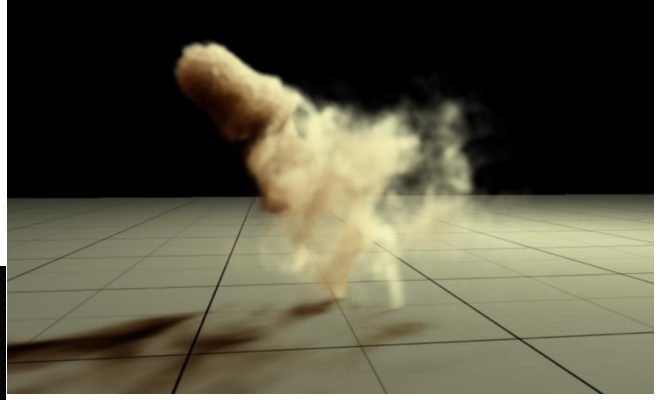
Realtime Graphics Demo

- Realtime rendering: Unreal Kite Demo

Unreal Engine Kite Demo (Epic Games 2015)

Realtime Graphics Demo

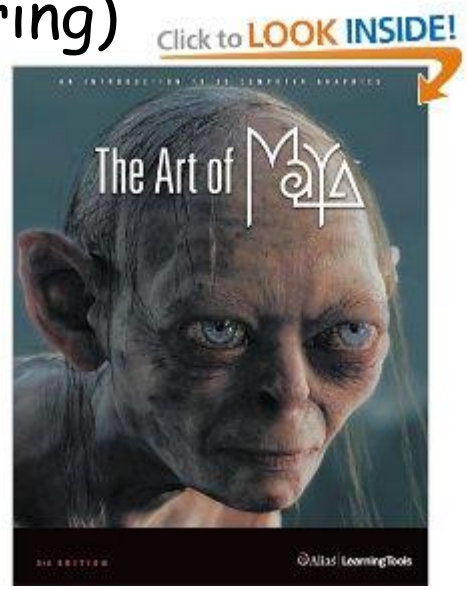
- Smoke simulation:



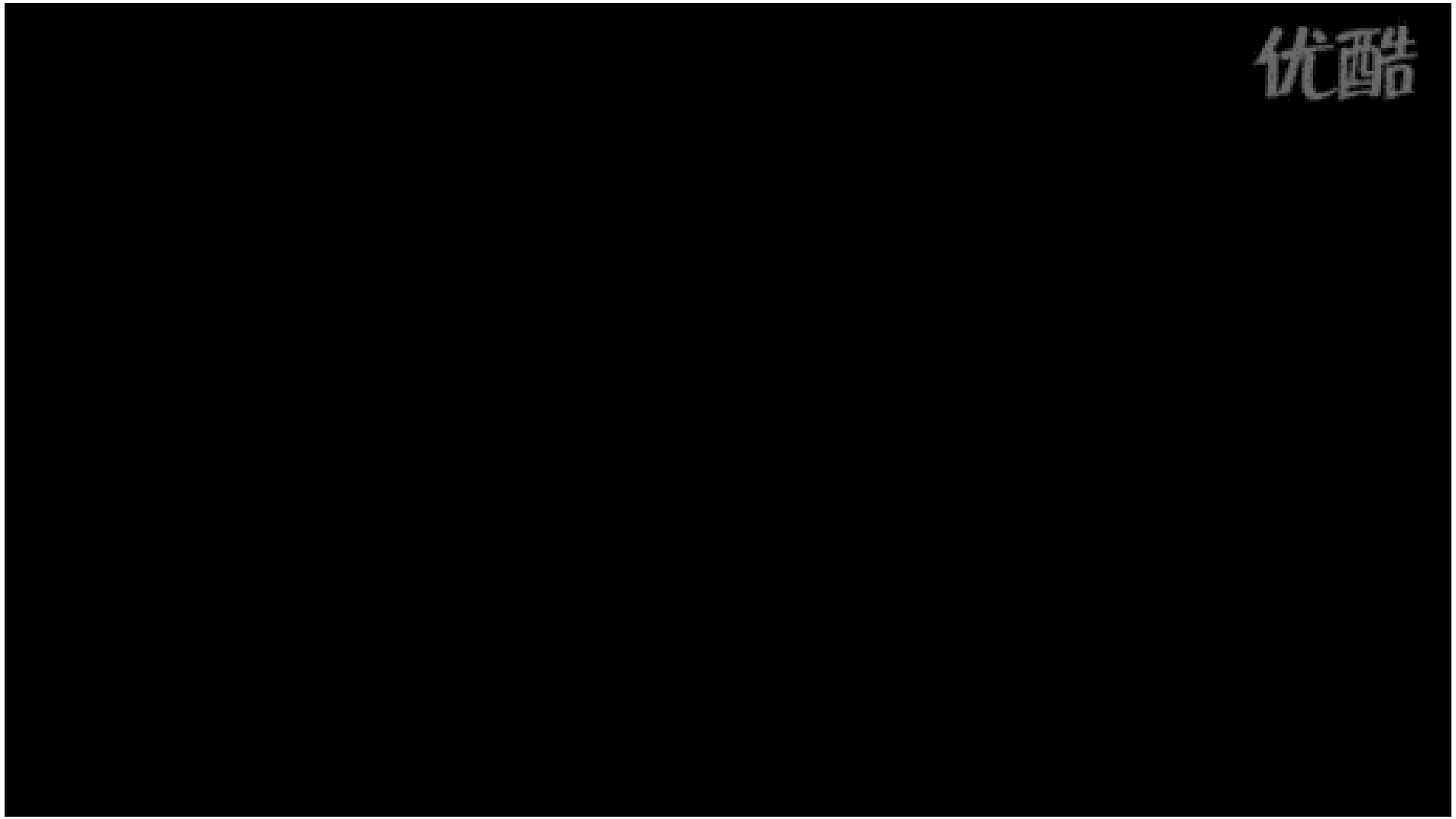
Modelling & Creative Content Creation

A list of Modelling, Animation, Video & Creative Content Creation applications that use OpenGL can be found in:

- **3ds max** (Professional 3D modelling, animation and rendering)
- ImageModeler (Automatic creation of 3D models with textures, from still pictures)
- Lightwave 3D (3D modelling, animation, rendering)
- Cinema 4D (Modelling, ray tracing & animation)
- **Maya** (Character animation, modelling, F/X, rendering)
- **Z-Brush**

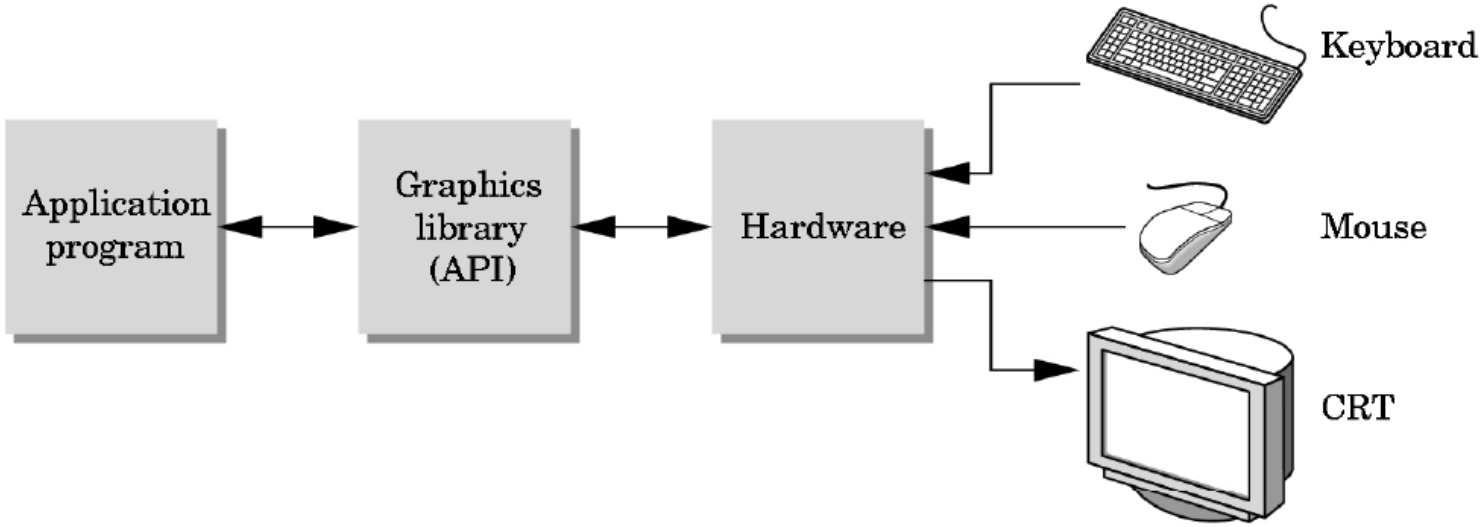


Z-Brush

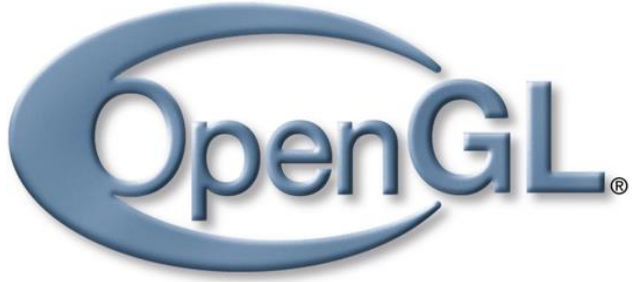


Graphics Library (API)

- **Interface** between Application and Graphics Hardware



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



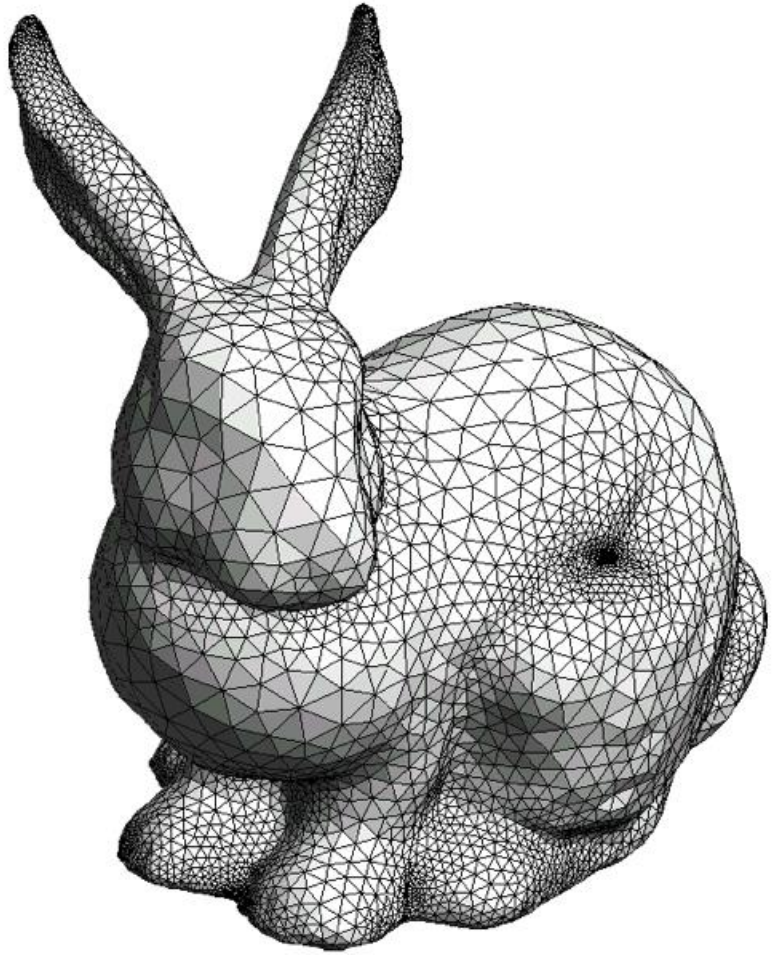
OpenGL is cross-platform

- **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
```

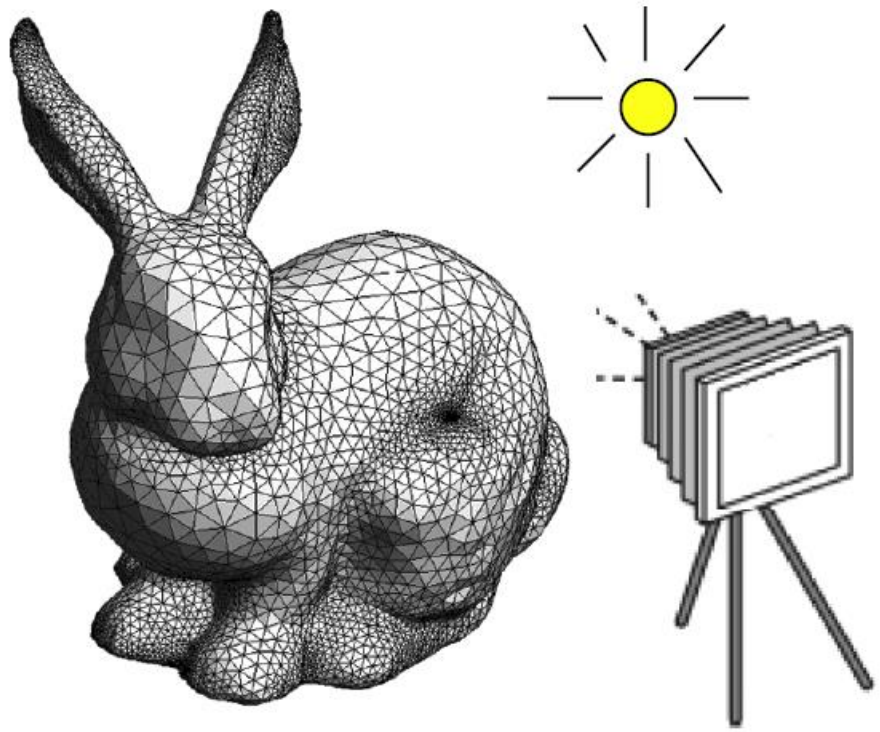
How does OpenGL work

- **From the programmer's point of view:**
  - Specify **geometric objects**
  - Describe **object properties**
    - Color
    - How objects reflect light

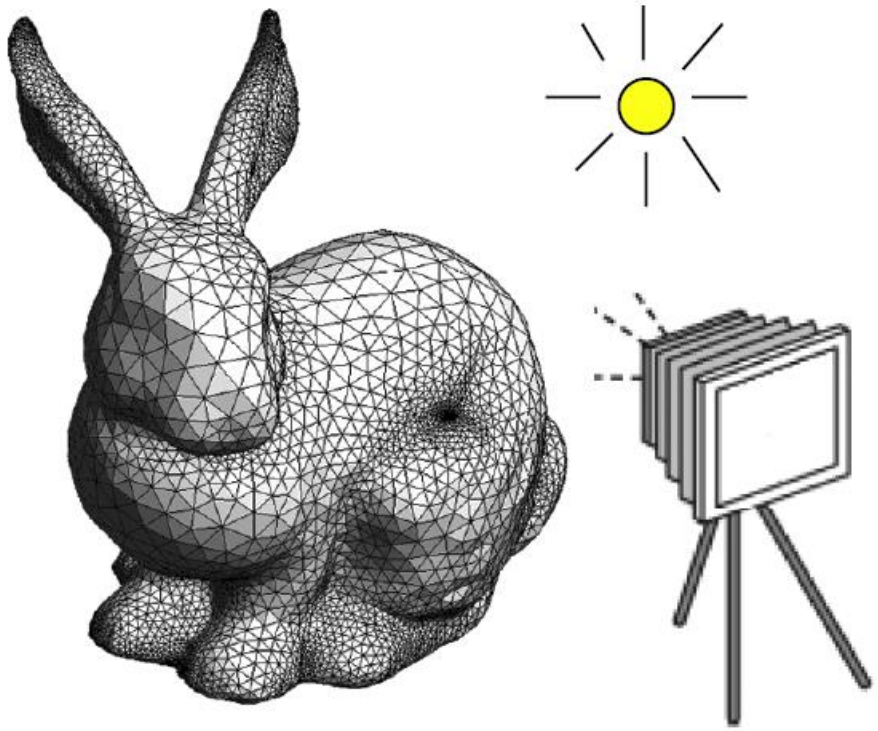


How does OpenGL work (continued)

- **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

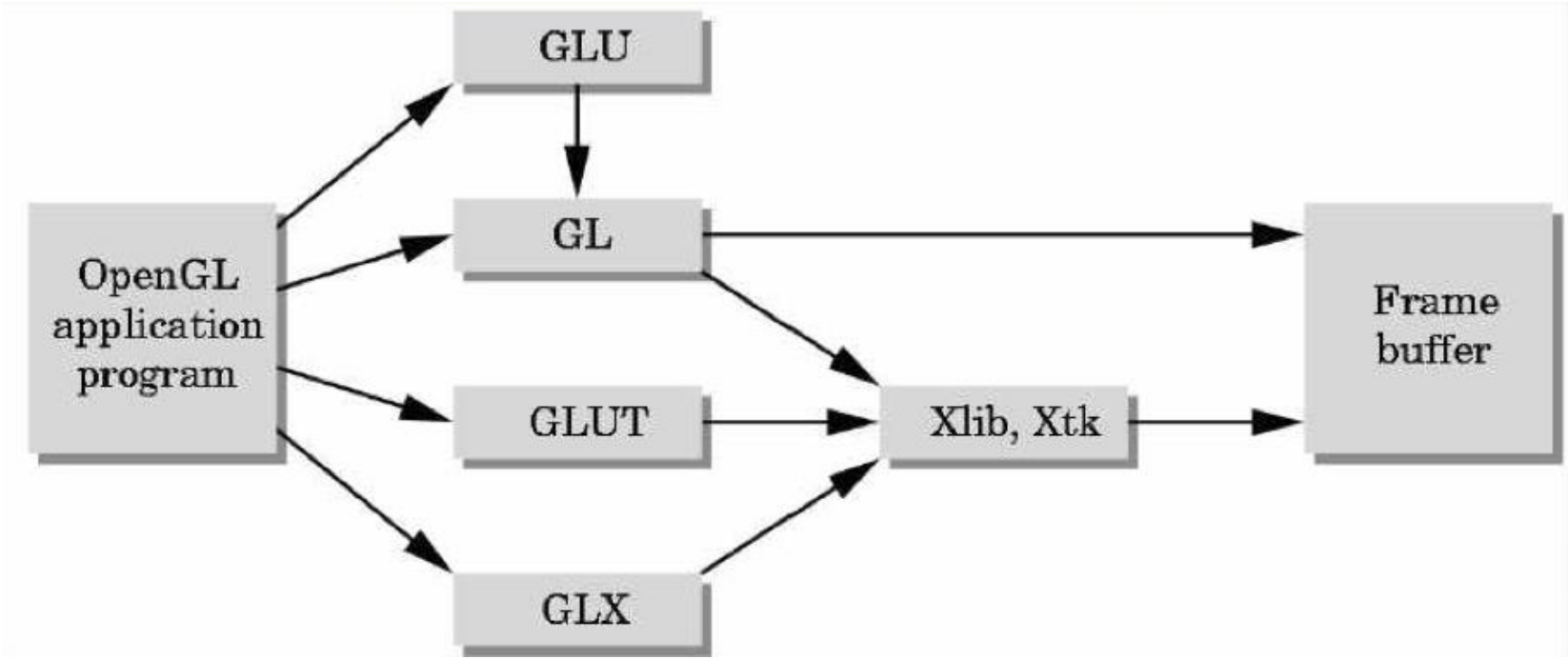
the result

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



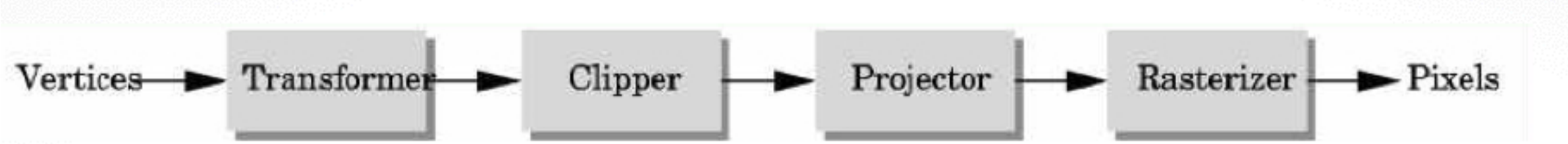
OpenGL Command Syntax

- 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



primitives+ material properties

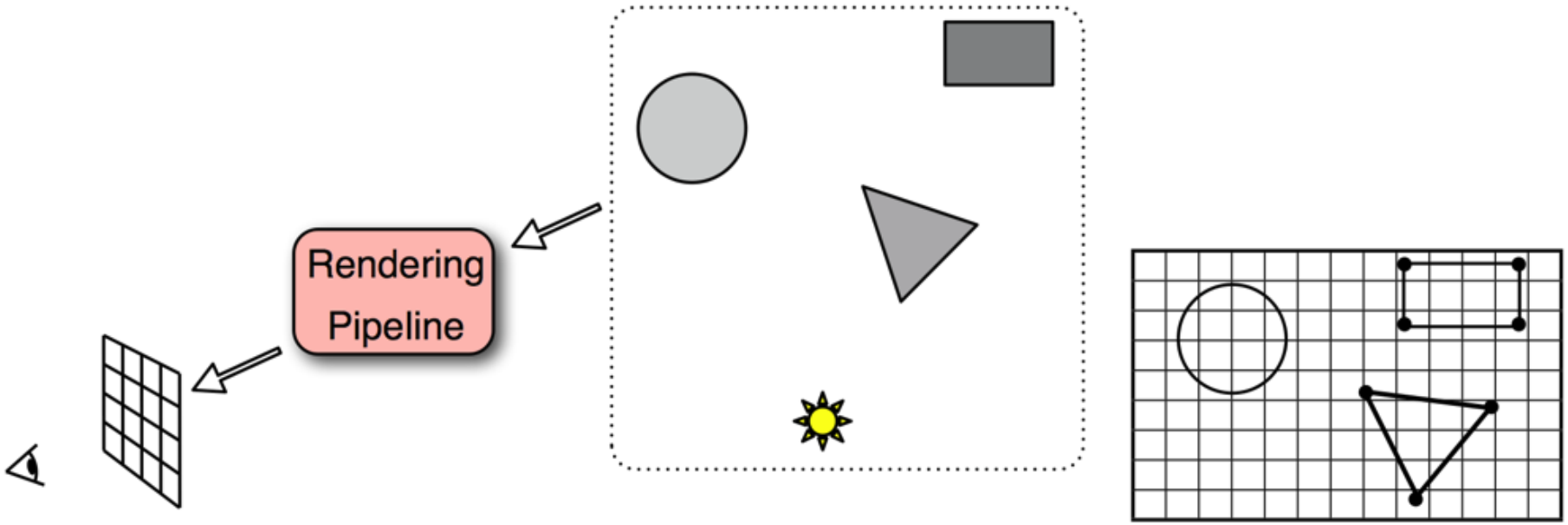
translate rotate scale

is it visible on screen?

3D to 2D

convert to pixels

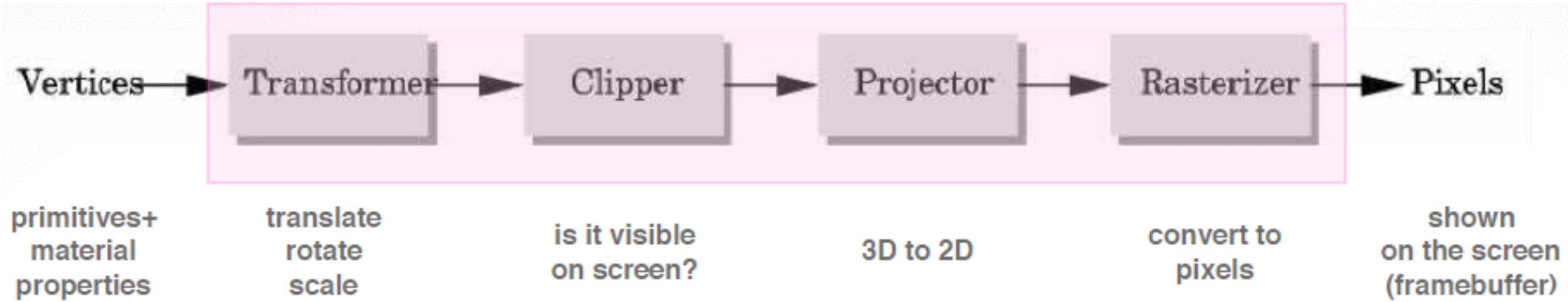
shown on the screen (framebuffer)



OpenGL uses immediate-mode rendering

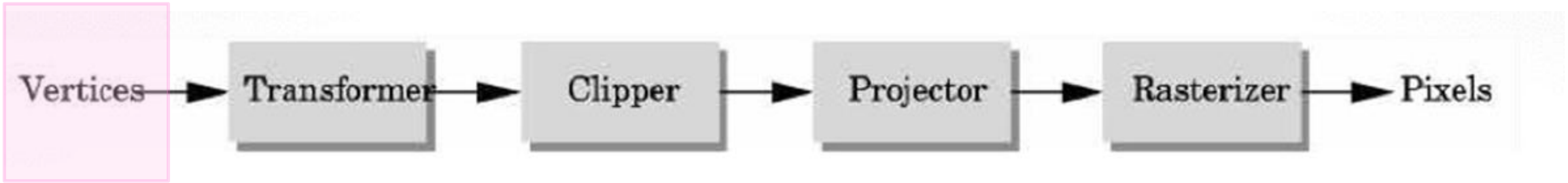
- Application generates **stream of geometric primitives** (polygons, lines)
- System **draws** each one into the **frame buffer**
- Entire scene is **redrawn** for every frame
- **Compare to:** offline rendering (e.g., Pixar Renderman, ray tracers…)

OpenGL Graphics Pipeline



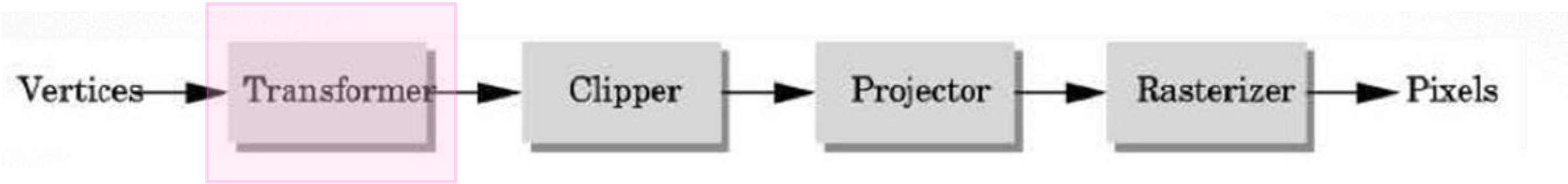
- implemented by **OpenGL, graphics driver, graphics hardware**
- OpenGL programmer does not need to implement the pipeline, but can **reconfigure it through shaders**

Vertices



- 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

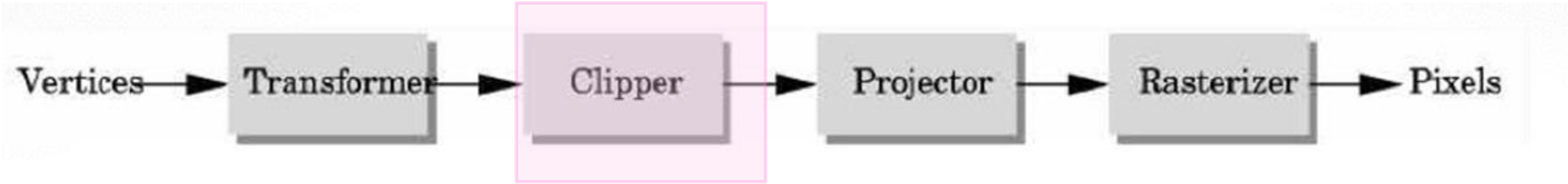


- Transformer in **world coordinates**

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

- Complex [Angel Ch. 3]

Clipper



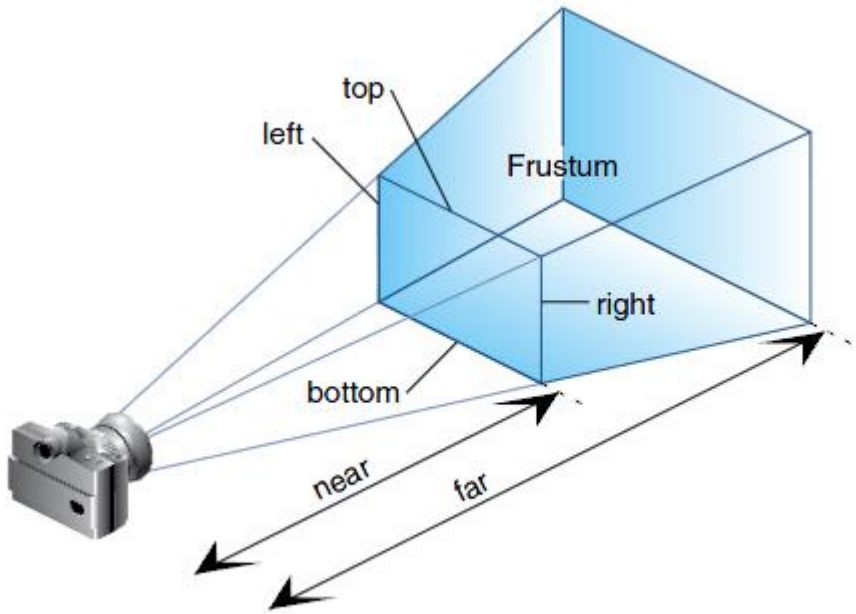
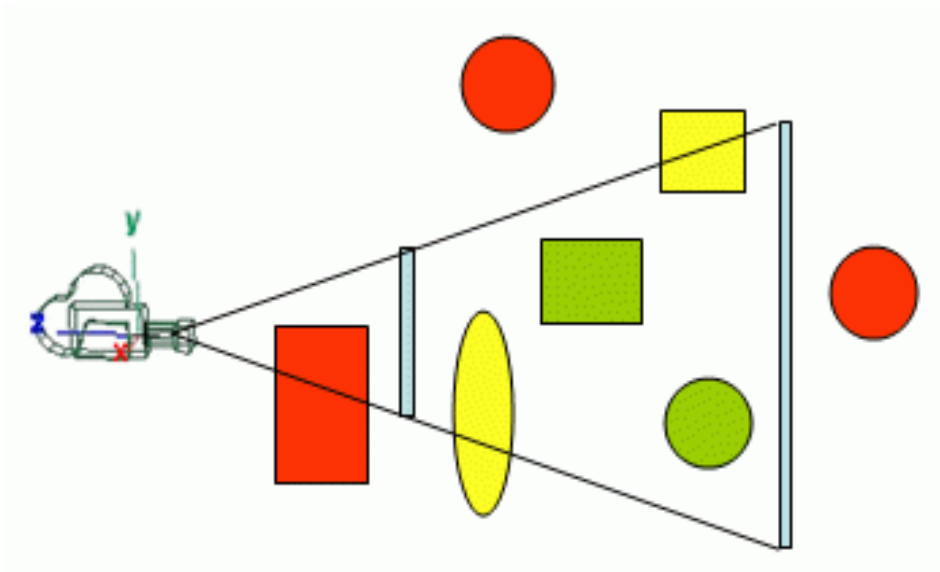
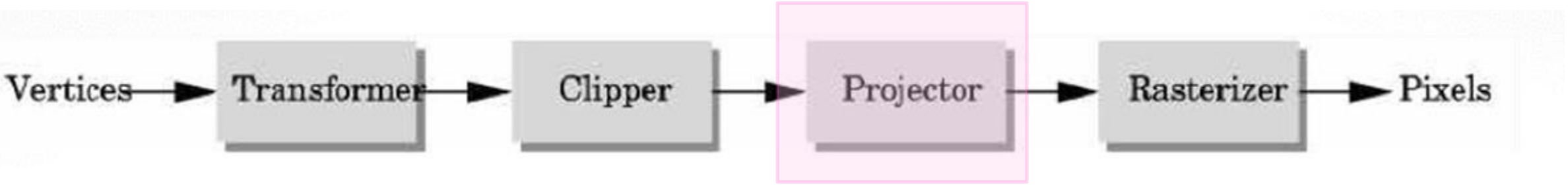


Figure 3-13 Perspective Viewing Volume Specified by glFrustum()

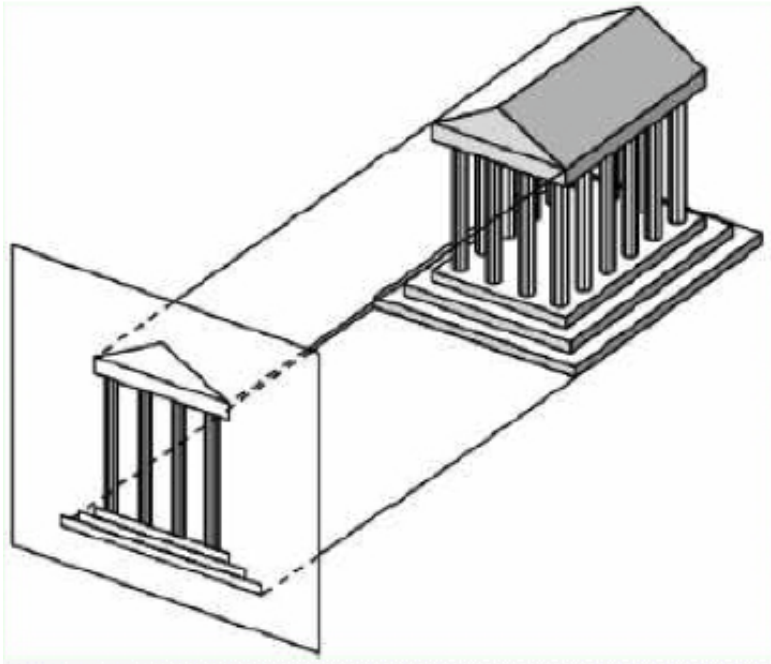


Projector

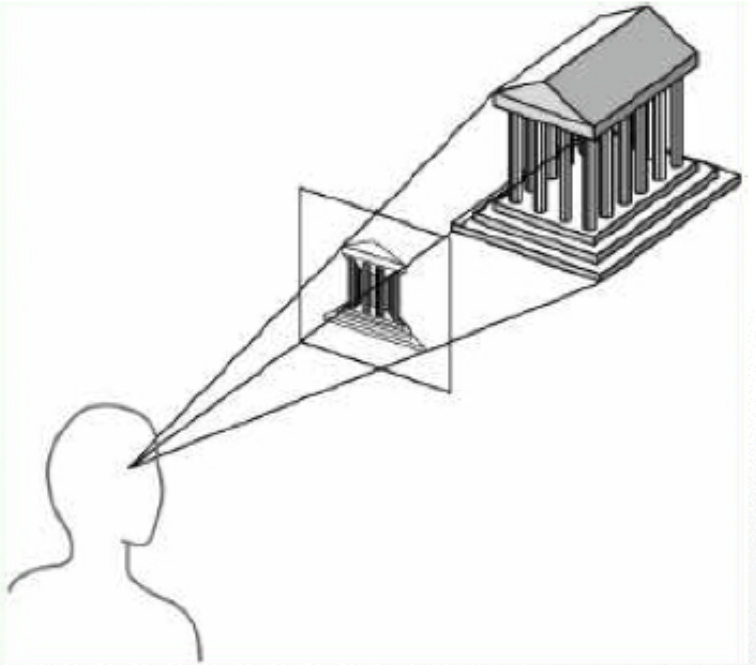


- Complex transformation [Angel Ch. 4]

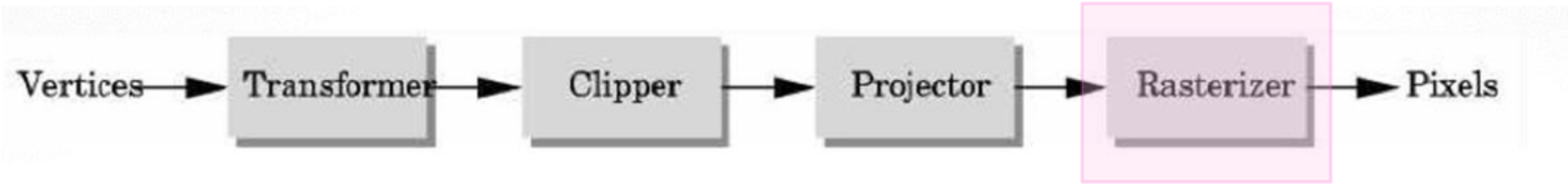
orthographic



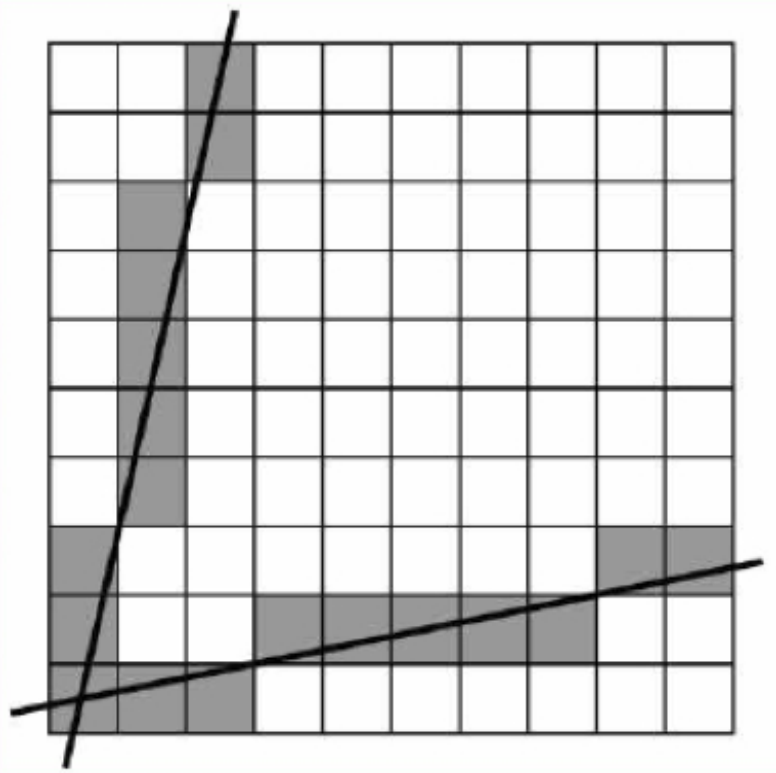
perspective



Rasterizer



- Interesting algorithms [Angel Ch. 6]
- **To window coordinates**
- Antialiasing



More Resources

- Official OpenGL Documentation
  - https://www.opengl.org/wiki/OpenGL_Reference
  - Or "man glVertex" on Linux/Mac

- Legacy OpenGL Tutorials
  - NeHe (http://nehe.gamedev.net/tutorial/lessons_01__05/22004/)
  - Programming Techniques GLUT Tutorial (http://www.programmingtechniques.com/2011/12/glut-tutorial-drawing-basic-shapes.html)

- Modern OpenGL Tutorials
  - OpenGL-Tutorial (http://www.opengl-tutorial.org/)
  - OpenGL-Introduction (https://open.gl/)

Thanks