Introduction to OpenGL

(Slides courtesy of Tamar Shinar)

OpenGL - Software to Hardware

- Silicon Graphics (SGI) revolutionized the graphics workstation by putting graphics pipeline in hardware (1982)
- To use the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications

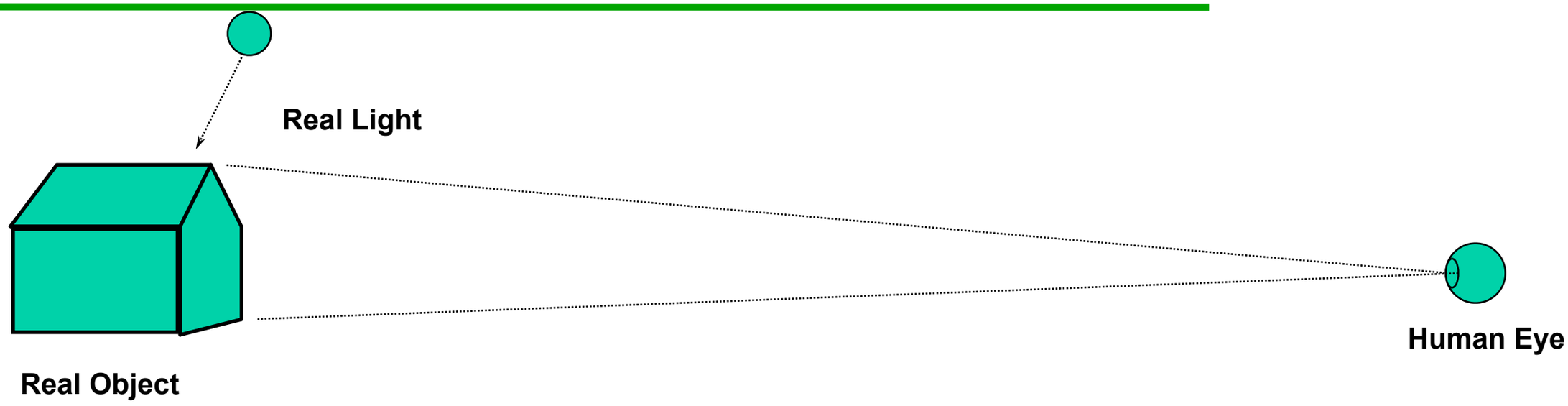
OpenGL

- The success of GL lead to OpenGL (1992), a platform-independent API that was
  - Easy to use
  - Close to the hardware - excellent performance
  - Focus on rendering
  - Omitted windowing and input to avoid window system dependencies

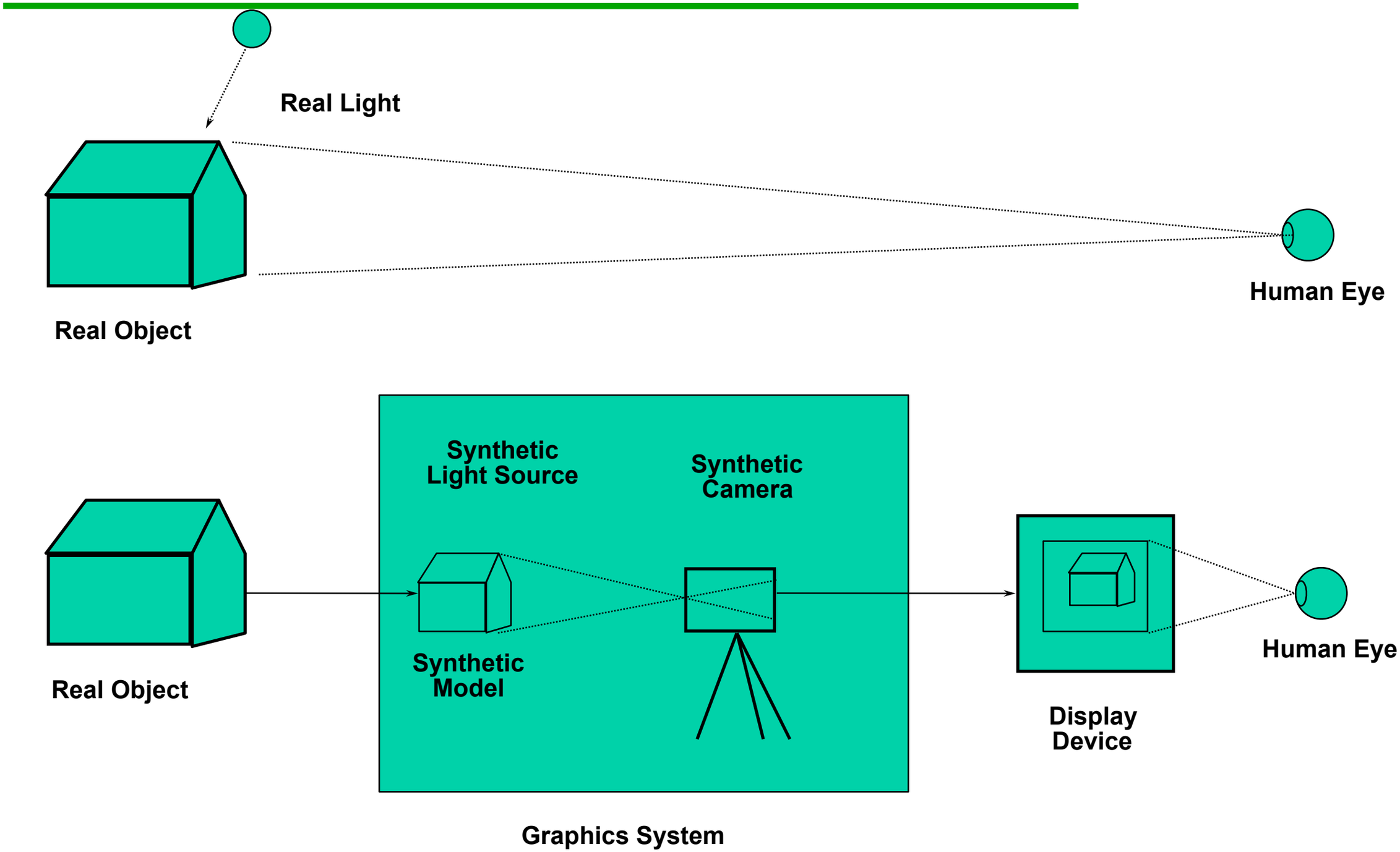
Introduction to OpenGL

- **Open G**raphics **L**ibrary, managed by Khronos Group
- A software interface to graphics hardware (GPU)
- Standard API with support for multiple languages and platforms, open source
- ~250 distinct commands
- Main competitor: Microsoft's Direct3D
- http://www.opengl.org/wiki/Main_Page

OpenGL: Conceptual Model

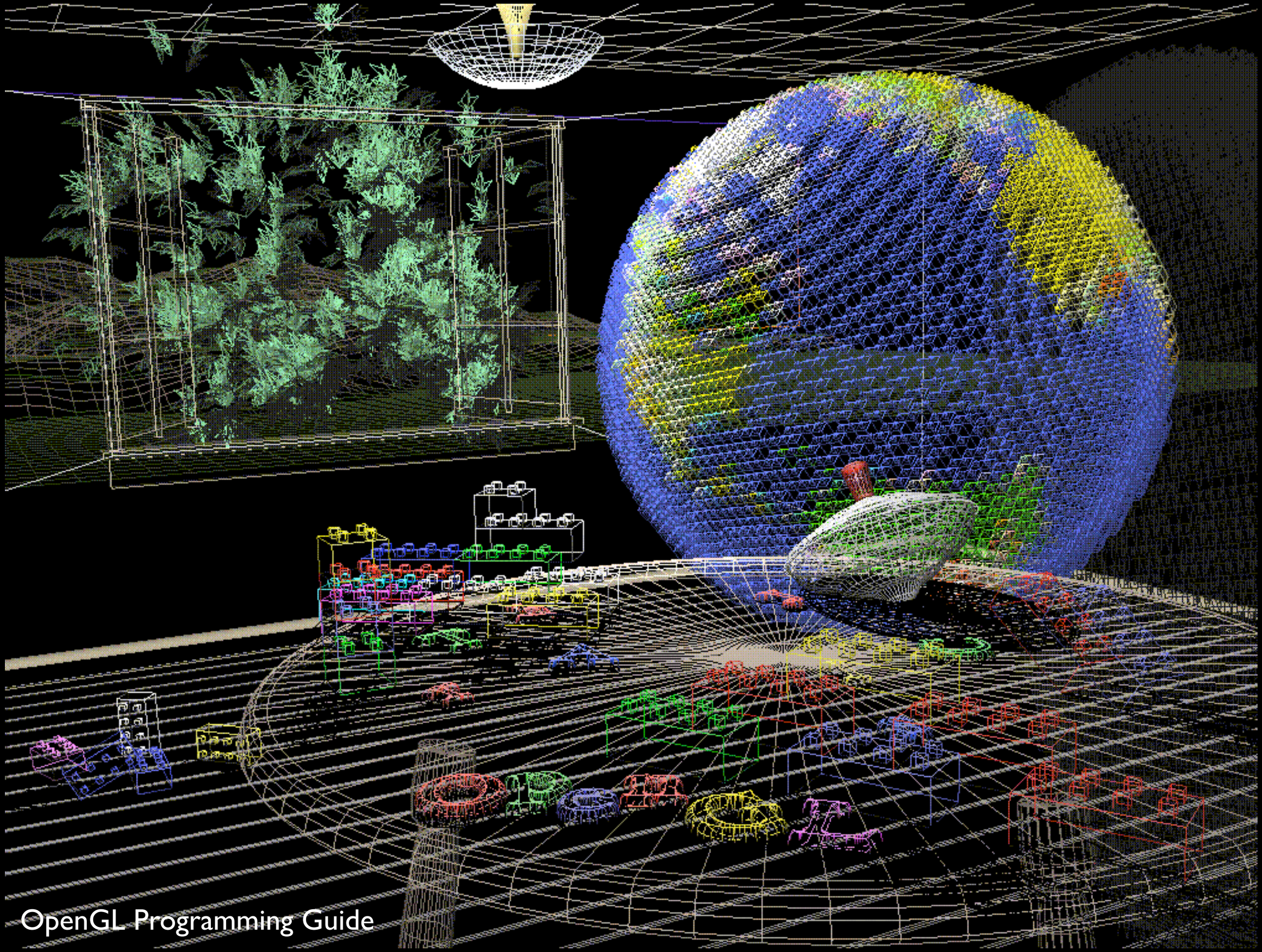


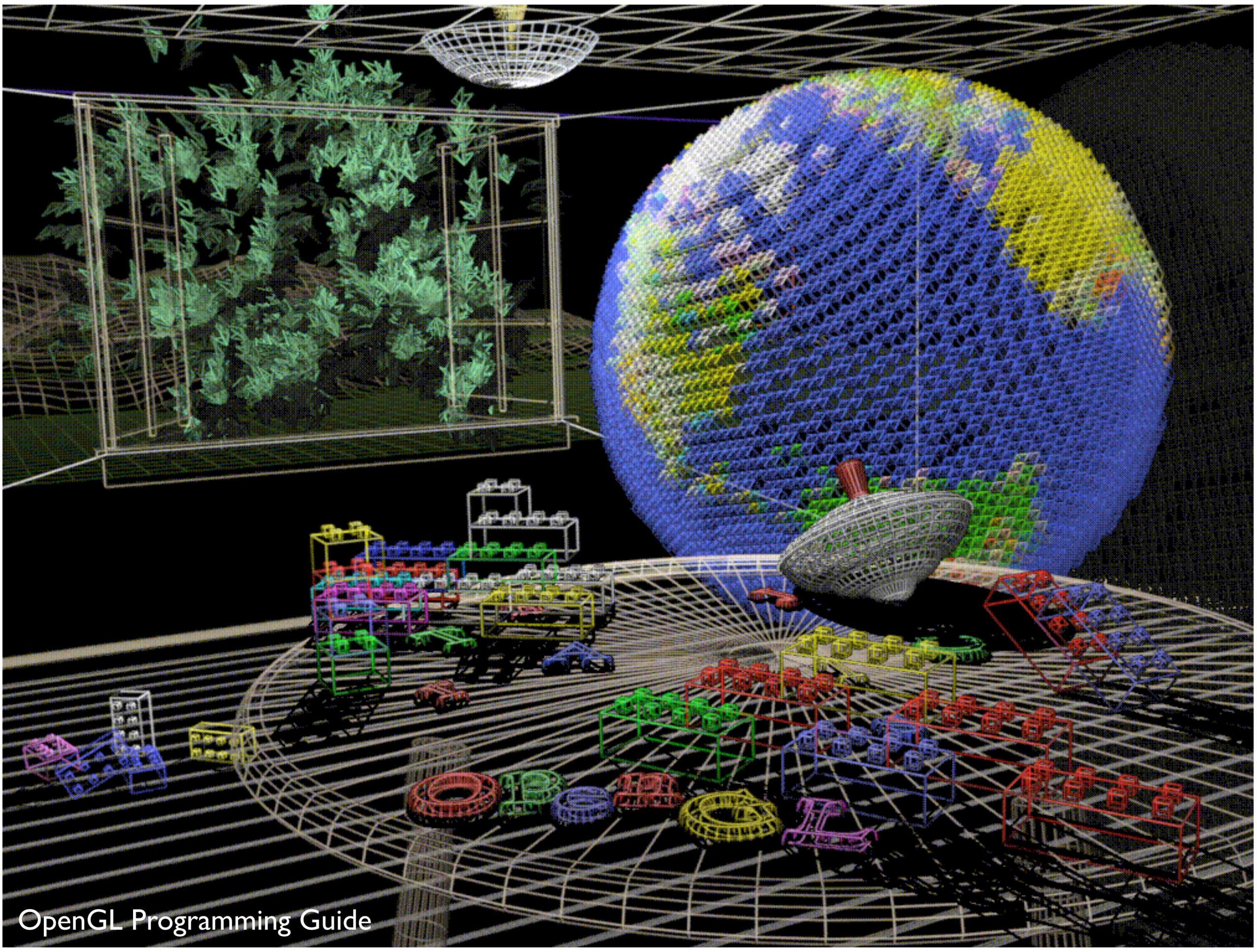
OpenGL: Conceptual Model



What can OpenGL do?

Examples from the OpenGL Programming Guide ("red book")


OpenGL Programming Guide


OpenGL Programming Guide

OpenGL Programming Guide

OpenGL Programming Guide

OpenGL Programming Guide

OpenGL Programming Guide

OpenGL Programming Guide

OpenGL Context

- contains all the information that will be used by OpenGL in executing a rendering command
- OpenGL functions operate on the “current” context
- local to an application
- application may have several OpenGL contexts

OpenGL State

- context contains "state" information
- put OpenGL into various states
  - e.g., current color, current viewing transformation
  - these remain in effect until changed
  - glEnable(), glDisable(), glGet(), glIsEnabled()
  - glPushAttrib(), glPopAttrib() to temporarily modify some state

OpenGL Rendering Pipeline

- sequence of steps taken when user issues a rendering command
- objects (appear to be) rendered in the exact order user provides

OpenGL Shaders

- Some stages of the rendering pipeline are programmable
  - programs are called “Shaders”
- Written in the OpenGL Shading Language

OpenGL command syntax

- commands: **gl**ClearColor();
  - glVertex**3f**()
- constants: **GL**_COLOR_BUFFER_BIT
- types: GLfloat, GLdouble, GLshort, GLint,

Simple OpenGL program

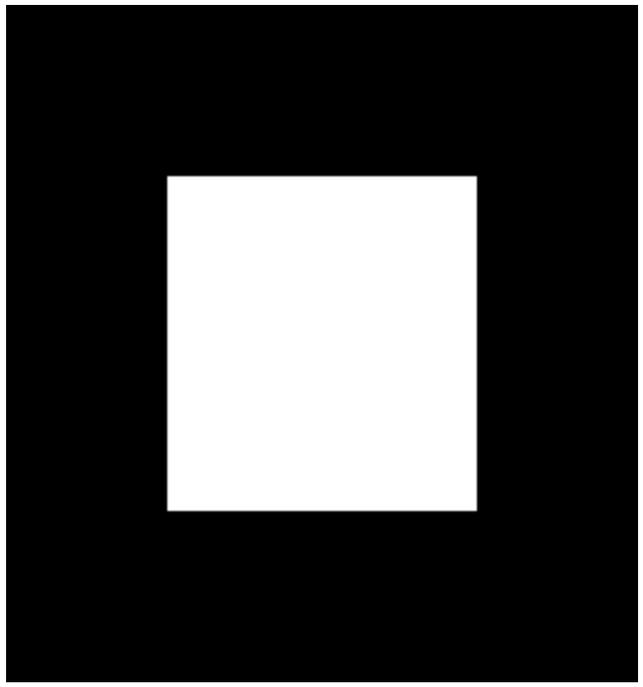
```
#include <whateverYouNeed.h>

main() {

   InitializeAWindowPlease();

   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f(0.25, 0.25, 0.0);
      glVertex3f(0.75, 0.25, 0.0);
      glVertex3f(0.75, 0.75, 0.0);
      glVertex3f(0.25, 0.75, 0.0);
   glEnd();
   glFlush();

   UpdateTheWindowAndCheckForEvents();
}
```

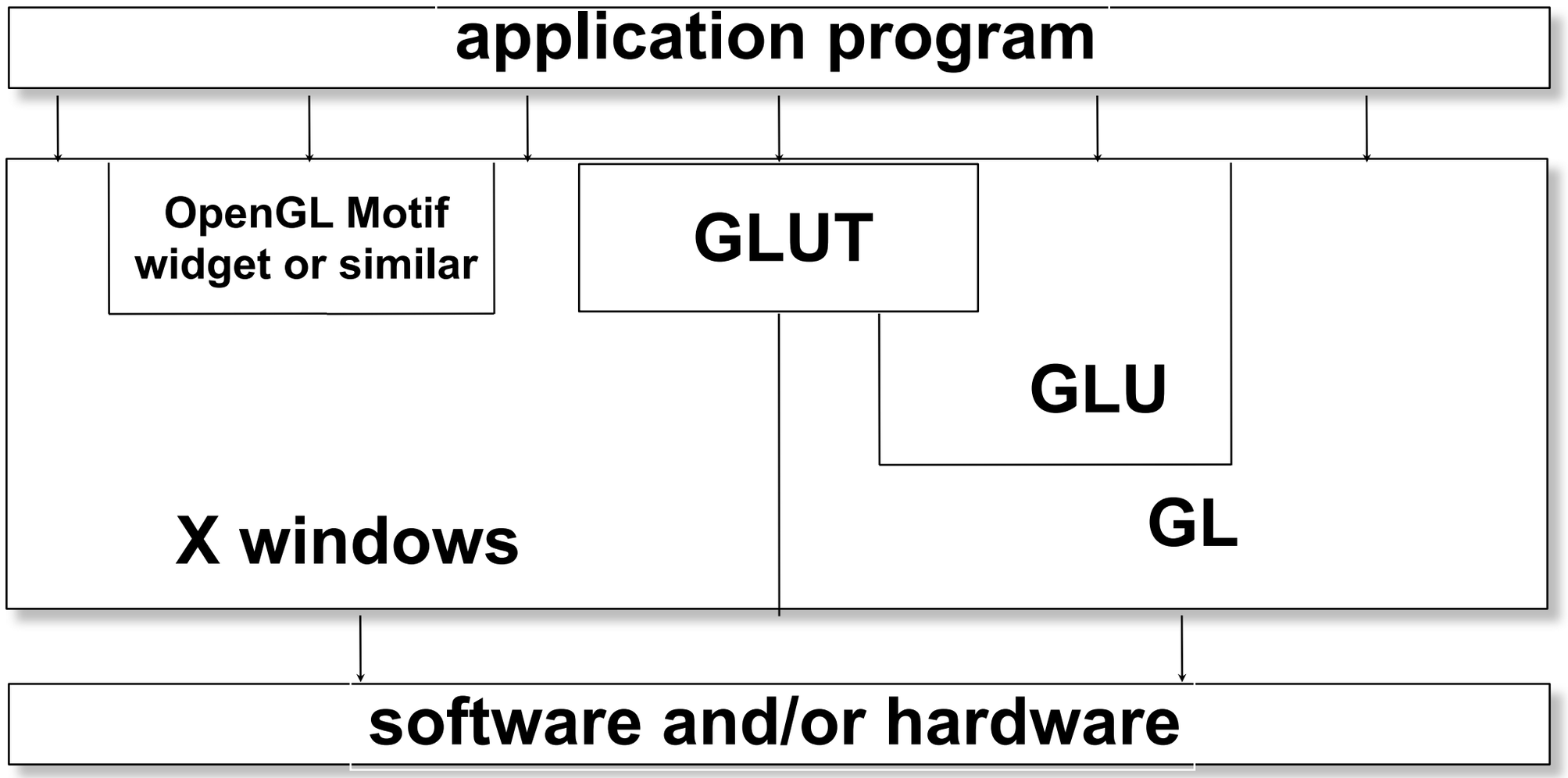


OpenGL Programming Guide, 7th Ed.

OpenGL Libraries

- OpenGL core library (gl.h)
  - OpenGL32 on Windows
  - GL on most unix/linux systems
- OpenGL Utility Library -GLU (glu.h)
  - avoids having to rewrite code
- OpenGL Utility Toolkit -GLUT (glut.h)
  - Provides functionality such as:
    - Open a window
    - Get input from mouse and keyboard
    - Menus

Software Organization



Simple OpenGL program

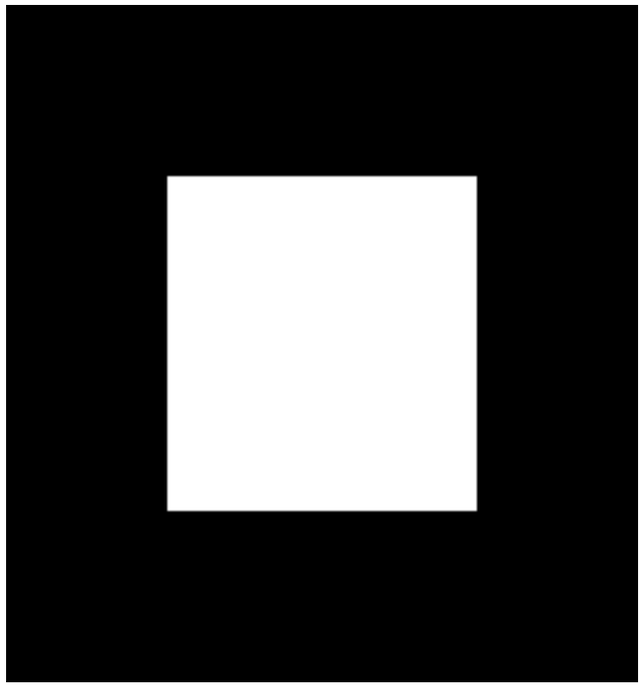
```
#include <whateverYouNeed.h>

main() {

   InitializeAWindowPlease();

   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f(0.25, 0.25, 0.0);
      glVertex3f(0.75, 0.25, 0.0);
      glVertex3f(0.75, 0.75, 0.0);
      glVertex3f(0.25, 0.75, 0.0);
   glEnd();
   glFlush();

   UpdateTheWindowAndCheckForEvents();
}
```



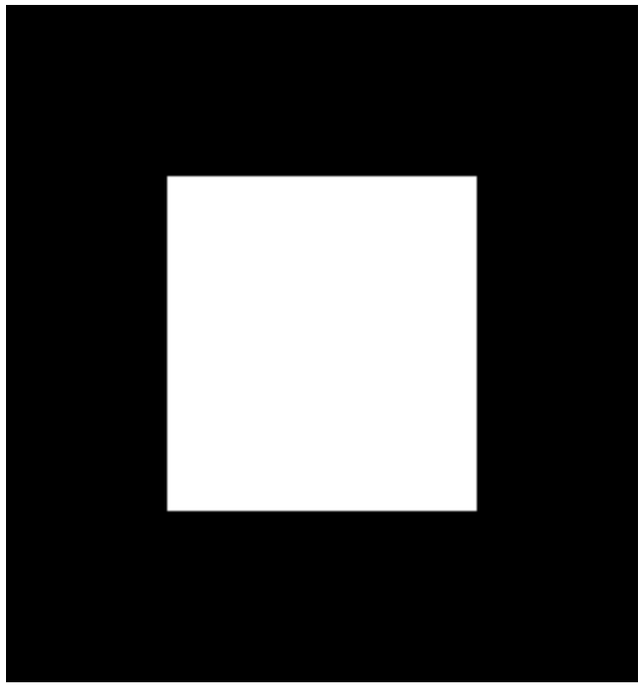
OpenGL Programming Guide, 7th Ed.

Simple OpenGL program

```
#include<GL/glut.h>

void init() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
}

void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f(0.25, 0.25, 0.0);
      glVertex3f(0.75, 0.25, 0.0);
      glVertex3f(0.75, 0.75, 0.0);
      glVertex3f(0.25, 0.75, 0.0);
   glEnd();
   glFlush();
}
main() {
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (FB_WIDTH, FB_HEIGHT);
    glutCreateWindow ("Test OpenGL Program");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
}
```



Choice of primitives

- Which primitives should an API contain?
  - small set - supported by hardware, *or*
  - lots of primitives - convenient for user

Choice of primitives

- Which primitives should an API contain?

➡**small set - supported by hardware**

- lots of primitives - convenient for user

Choice of primitives

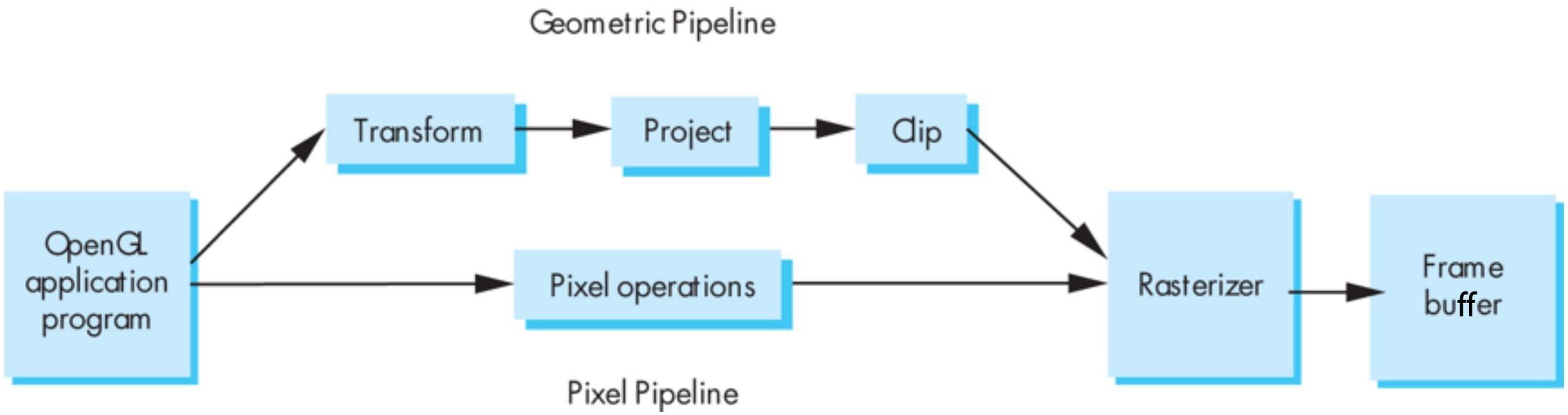
- Which primitives should an API contain?

➡**small set - supported by hardware**

- lots of primitives - convenient for user

GPUs are optimized for
points, **lines**, and **triangles**

Two classes of primitives

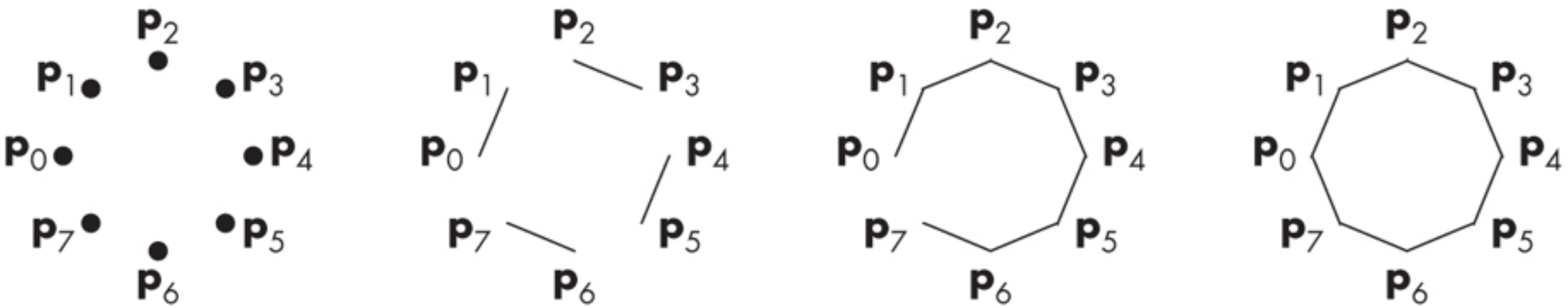


[Angel and Shreiner]

Geometric : points, lines, polygons

Image : arrays of pixels

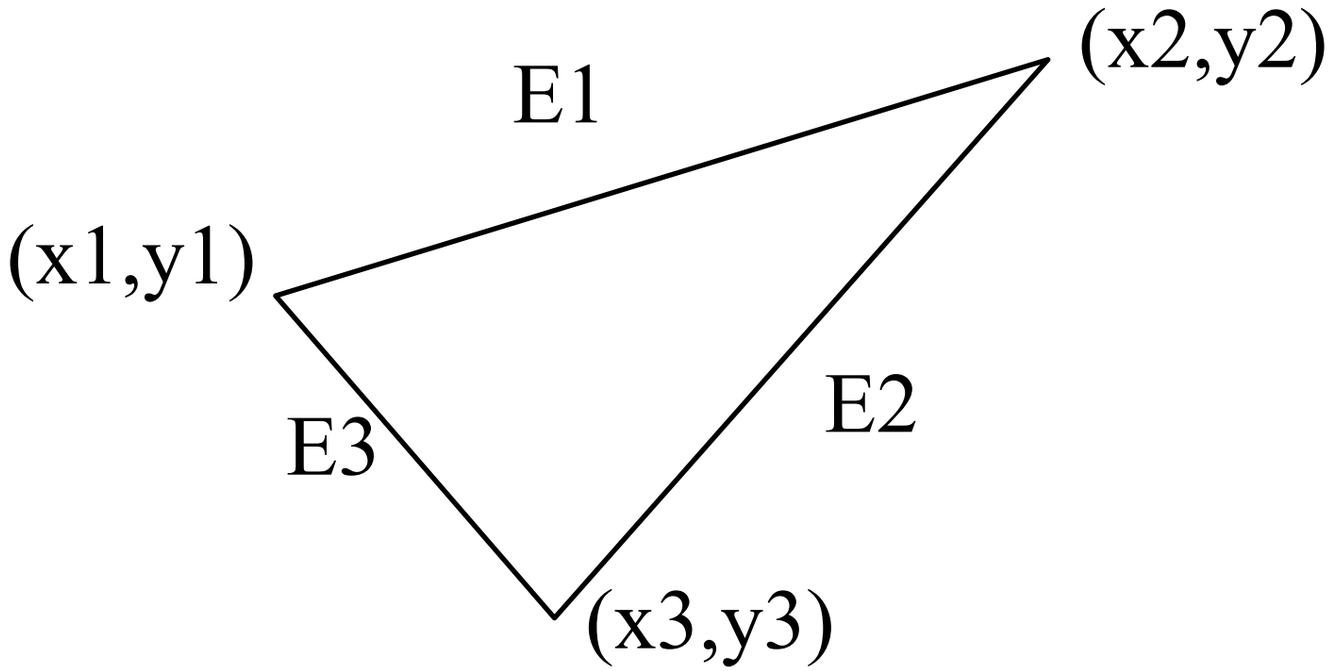
Point and line segment types



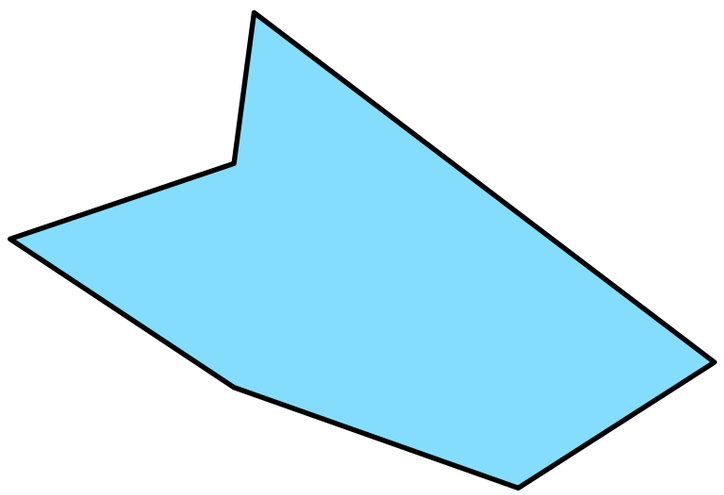
[Angel and Shreiner]

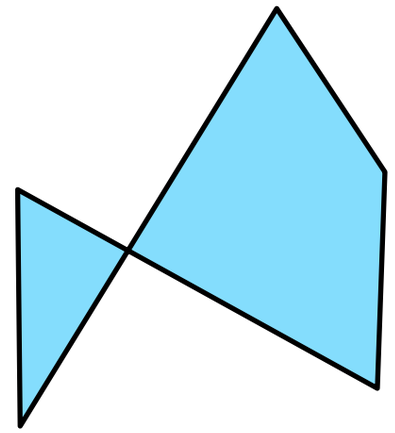
Polygons

- Multi-sided planar element composed of edges and vertices.
- Vertices (singular vertex) are represented by points
- Edges connect vertices as line segments



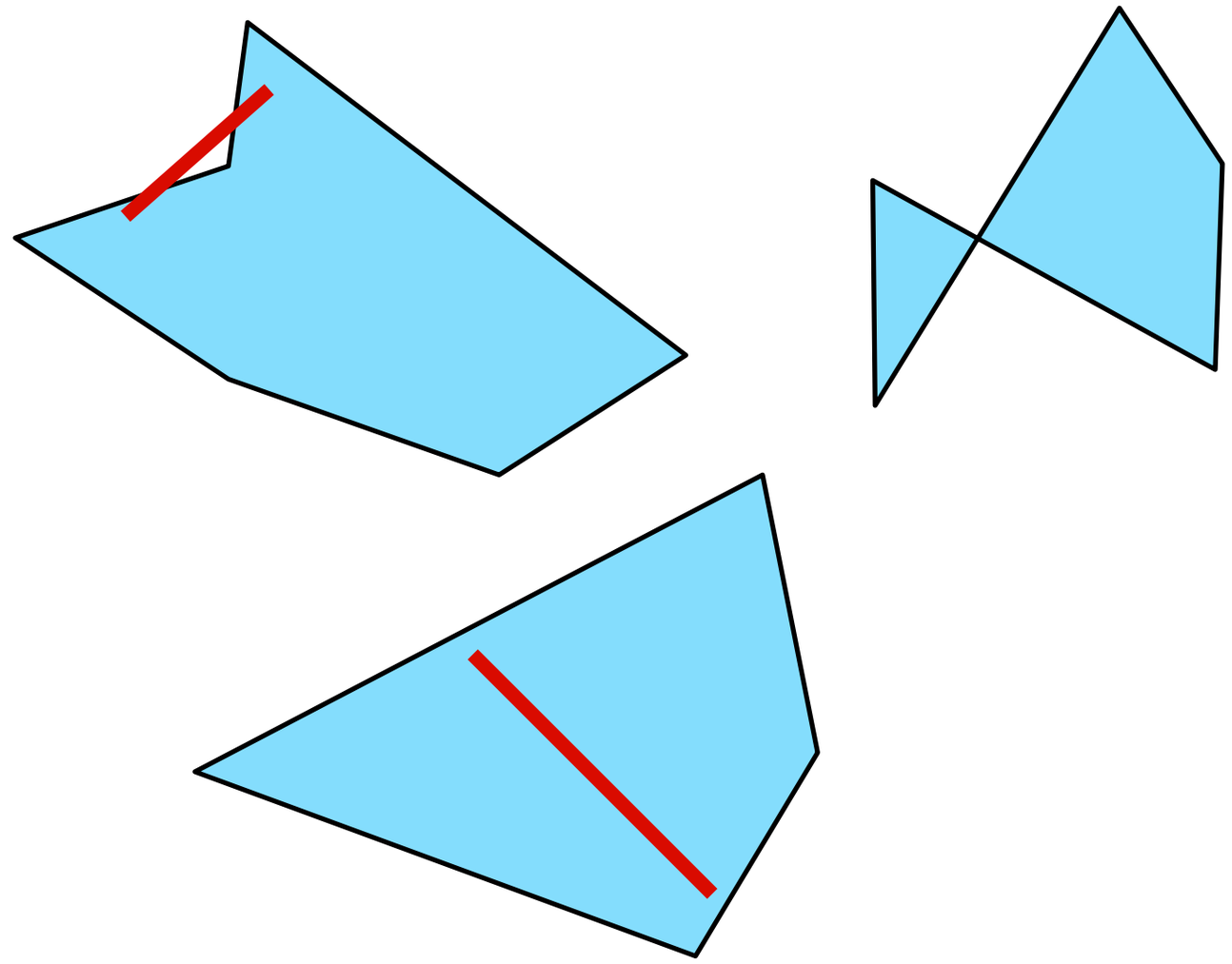
Valid polygons





- Simple
- Convex
- Flat

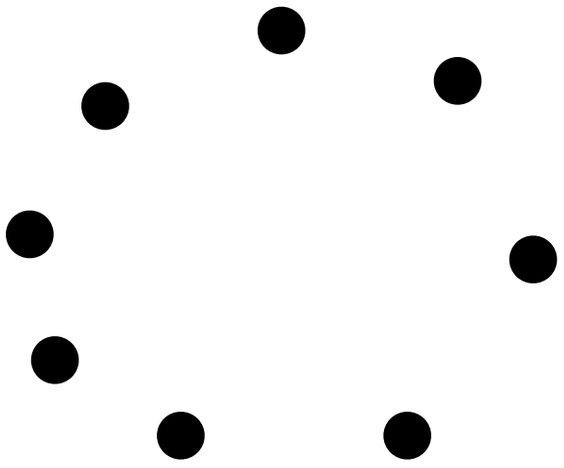
Valid polygons



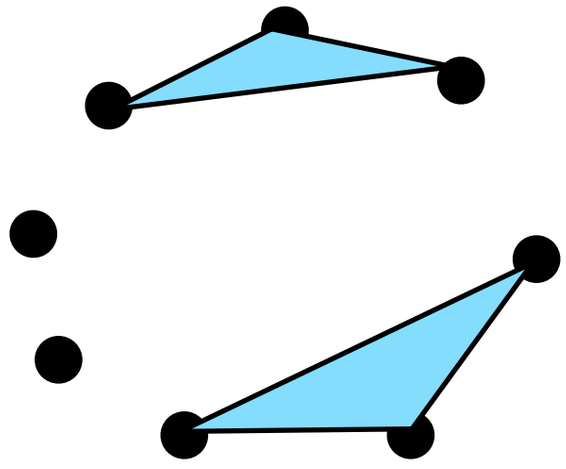
- Simple
- Convex
- Flat

OpenGL polygons

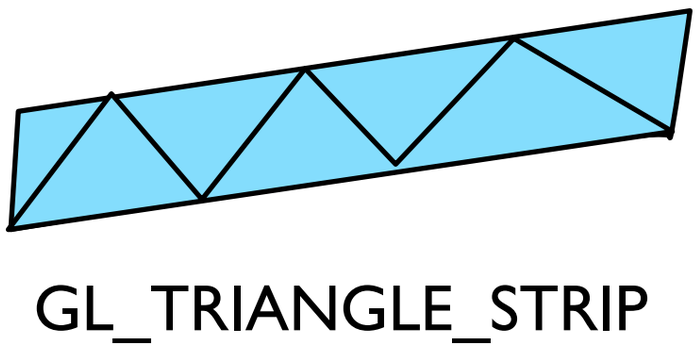
- Only triangles are supported (in latest versions)



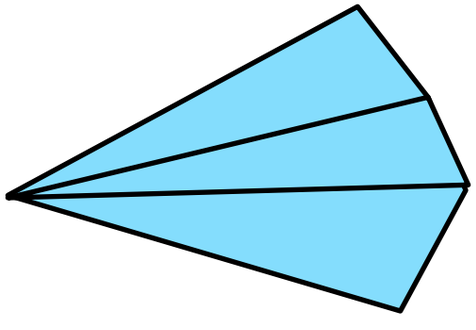
GL_POINTS



GL_TRIANGLES



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

Other polygons