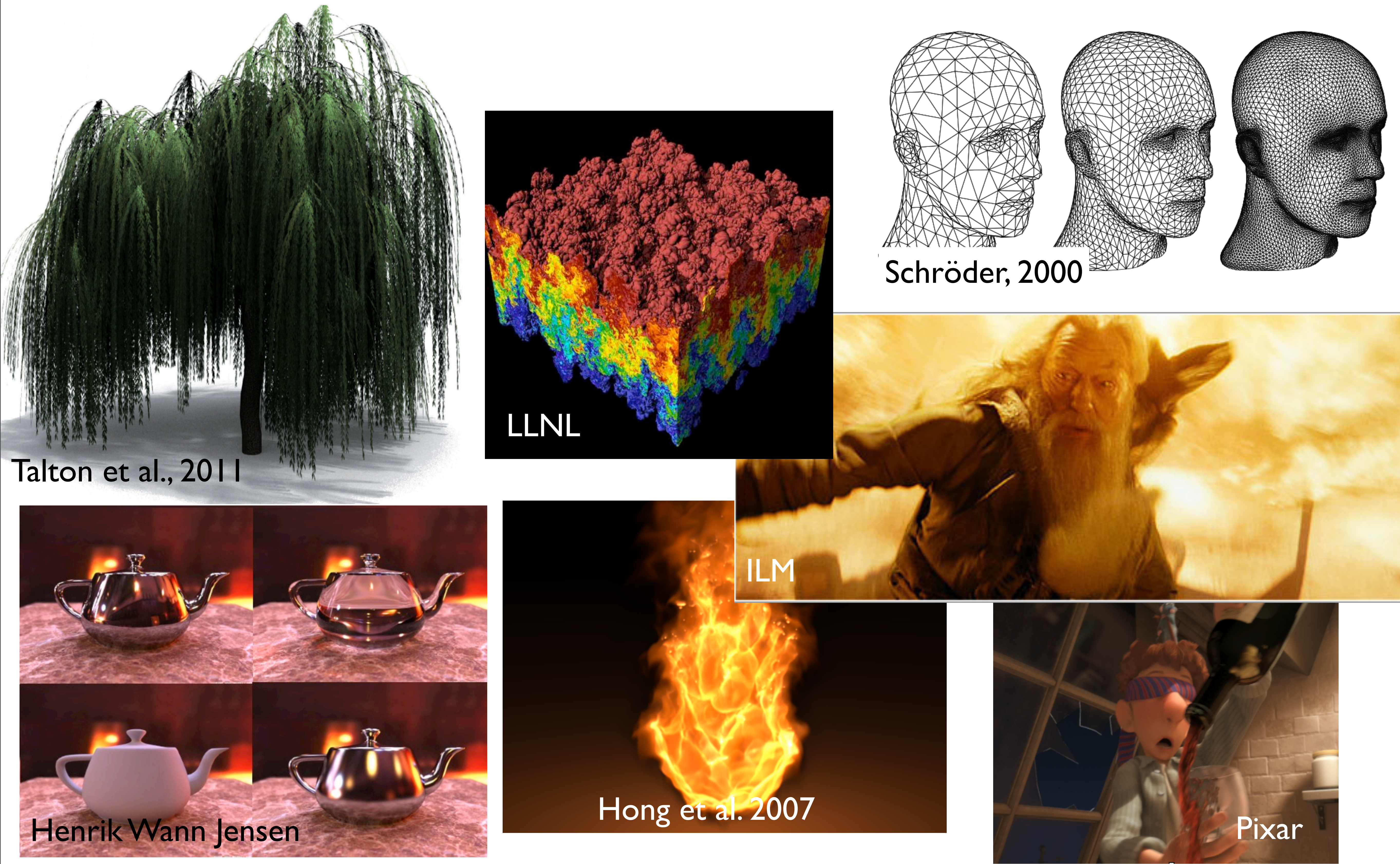
CS130 : Computer Graphics

Fall 2015

Tamar Shinar
Computer Science & Engineering
UC Riverside

Welcome to CS130!



Examples of different works in graphics. Clockwise: procedural modeling, scientific visualization, geometric modeling, live action special effects, animated special effects, physics-based special effects research, rendering

Today's agenda

- Course logistics
- Introduction: graphics areas and applications
- Introduction to OpenGL
- Math review

Course Overview

- Learn fundamental 3D graphics concepts
- Implement graphics algorithms
  - make the concepts concrete
  - expand your abilities and confidence for future work

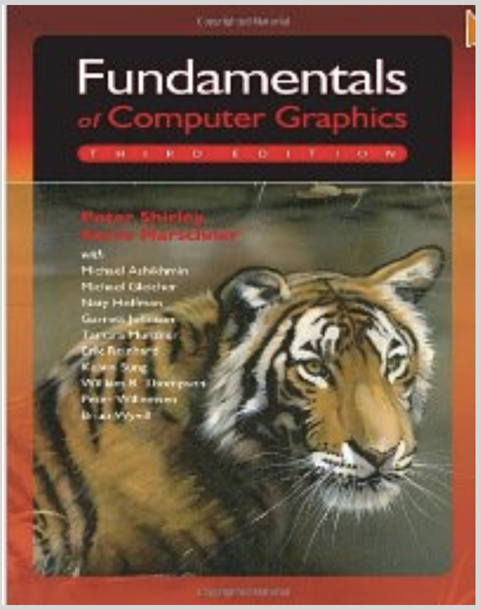
Course Logistics

- Instructor: Tamar Shinar
- TAs: Steve Cook, Wojciech Karas
- Website: http://www.cs.ucr.edu/~shinar/courses/cs130
- Lectures: TuTh 9:40-11am, Chung 143
- Lab: M 8:10-11:00pm, 11:10-2:00pm, Chung 129
- Announcements (assignments, etc.) made in class and through ilearn

Course Logistics

- Grading
  - 10% labs
  - 10% homework
  - 30% assignments (2 assignments, 15% each)
  - 50% tests (2 midterms, 1 final)
- Detailed schedule on class website

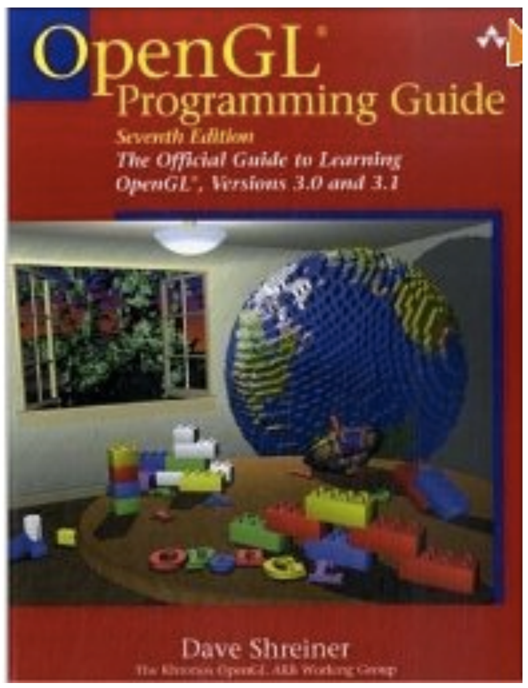
Textbook

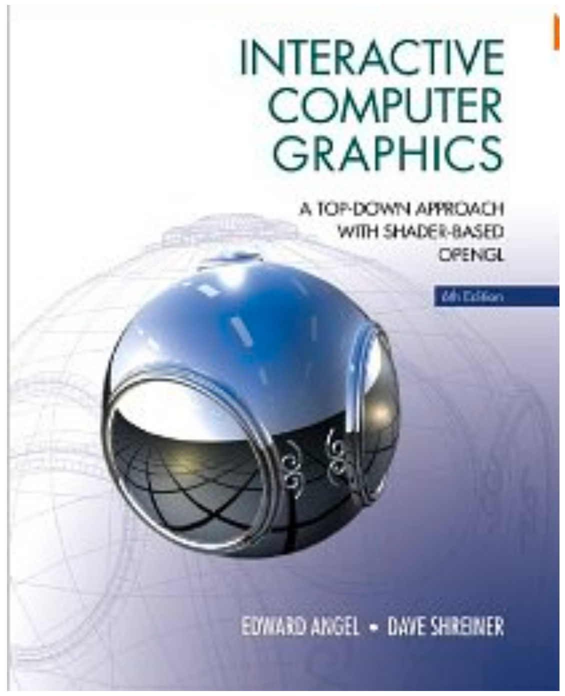


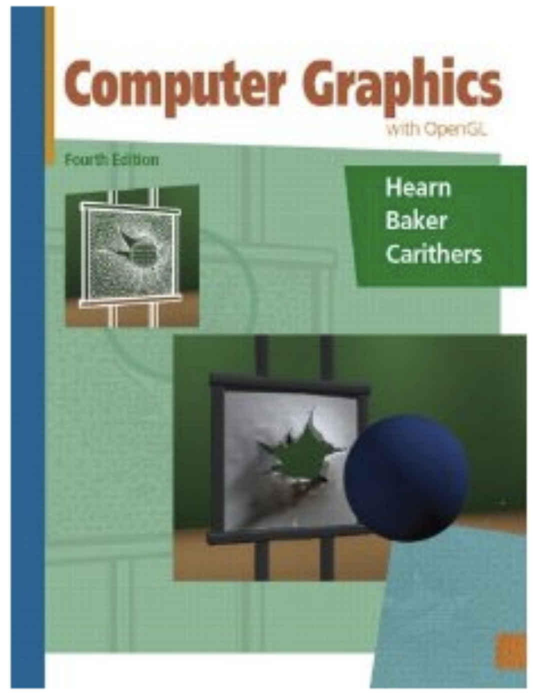
Fundamentals of Computer Graphics

Shirley and Marschner

Additional books







if you like using a book
- red book older version online:http://fly.cc.fer.hr/~unreal/theredbook/

About me

- B.S., University of Illinois in Urbana-Champaign, Mathematics, Computer Science, Fine Art
- Ph.D., 2008, Stanford University on simulation methods for computer graphics
- Started at UCR in the Fall 2011
- Work in graphics simulation and biological simulation

http://www.cs.ucr.edu/~shinar

Introduction

Graphics applications

- 2D drawing
- Drafting, CAD
- Geometric modeling
- Special effects
- Animation
- Virtual Reality
- Games
- Educational tools
- Surgical simulation
- Scientific and information visualization
- Fine art

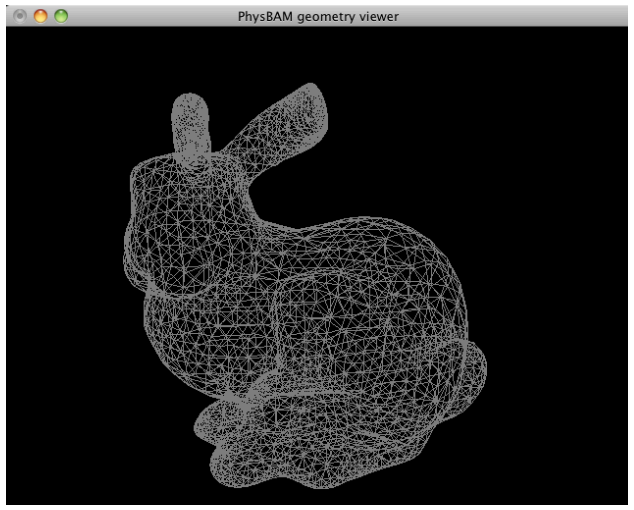
Graphics areas

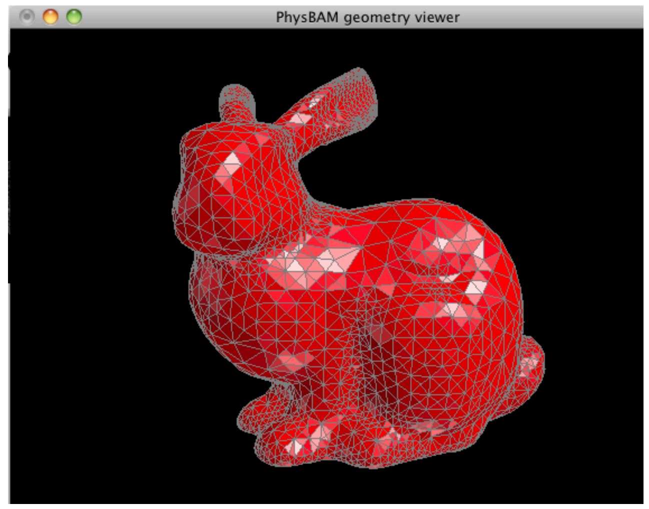
- **Modeling** - mathematical *representations* of physical objects and phenomena
- **Rendering** - creating a *shaded image* from 3D models
- **Animation** - creating motion through a sequence of images
- **Simulation** - physics-based models for modeling dynamic environments

Modeling and **rendering** are separate stage
- first design and position objects -- **modeling**
- then add lights, materials properties, effects -- **rendering**

Modeling

Talton et al., 2011





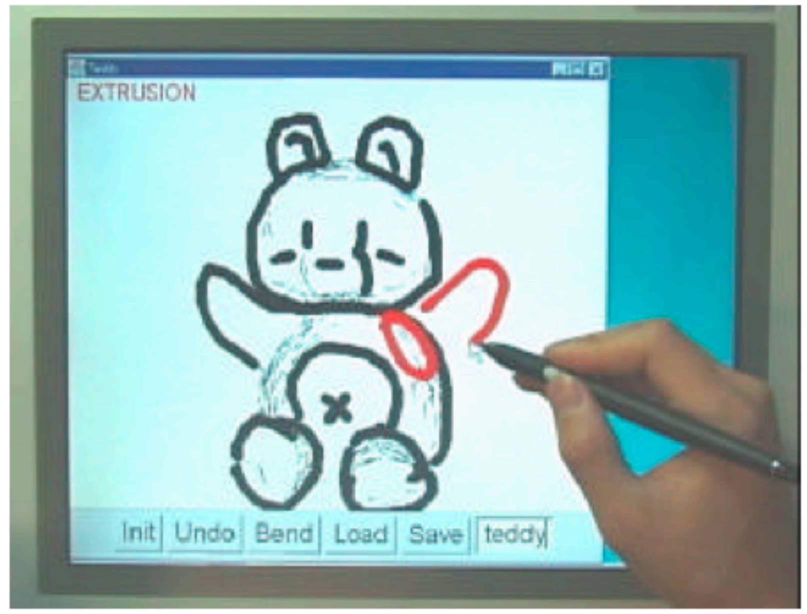
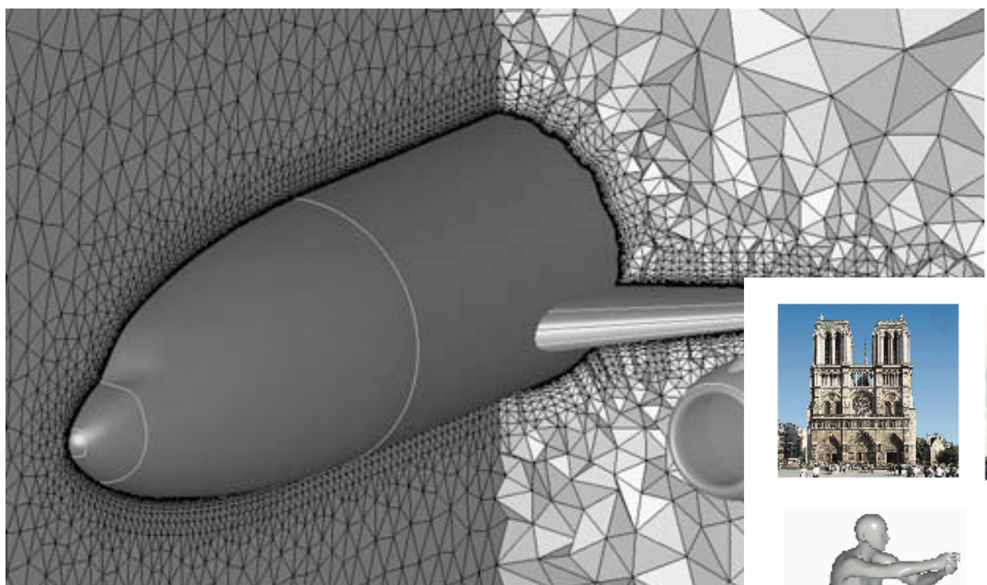


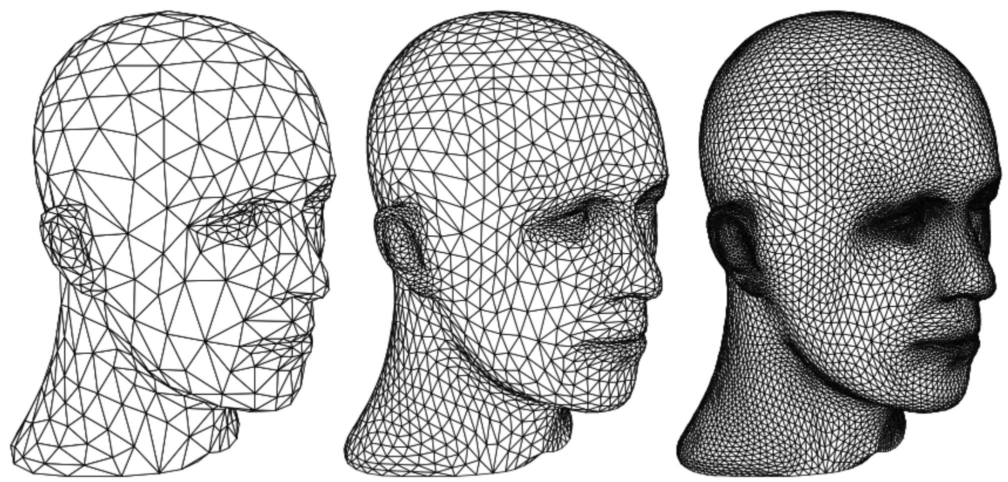
Figure1: Teddy in use on a display-integrated tablet.

Igarashi et al., 2007



CFD Technologies

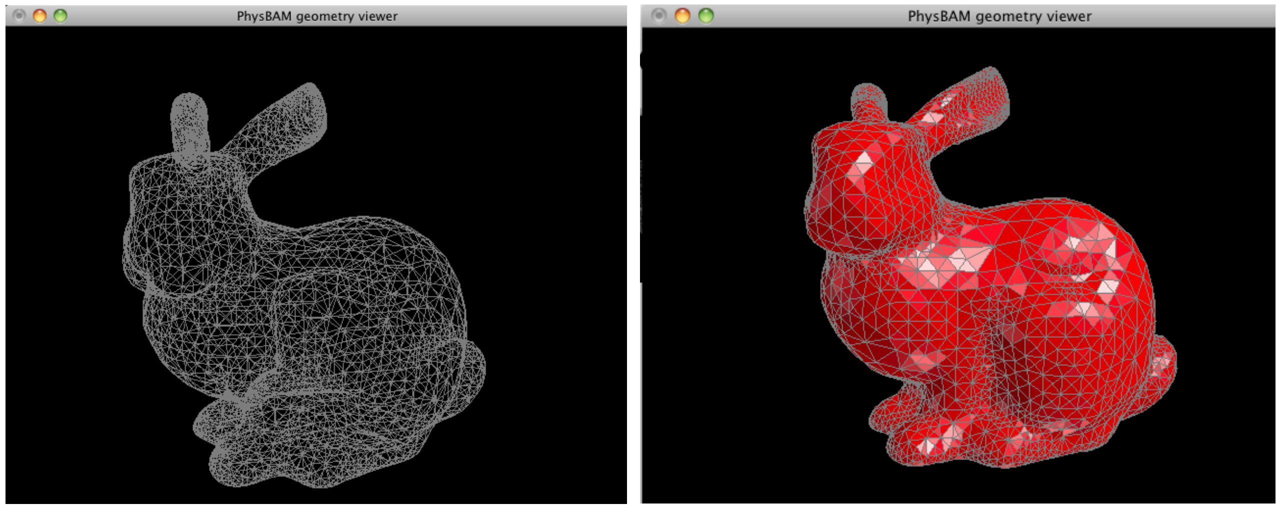
Bronstein et al., 2011



Schröder, 2000

- subdivision surface - Siggraph course notes 2000
- Teddy : sketch based interface for 3D modeling
- Talton et al. -- procedural modeling - for games, virtual worlds, design, etc.
  - combine machine learning and graphics
- Bronstein - reasoning about geometric models for search

Rendering

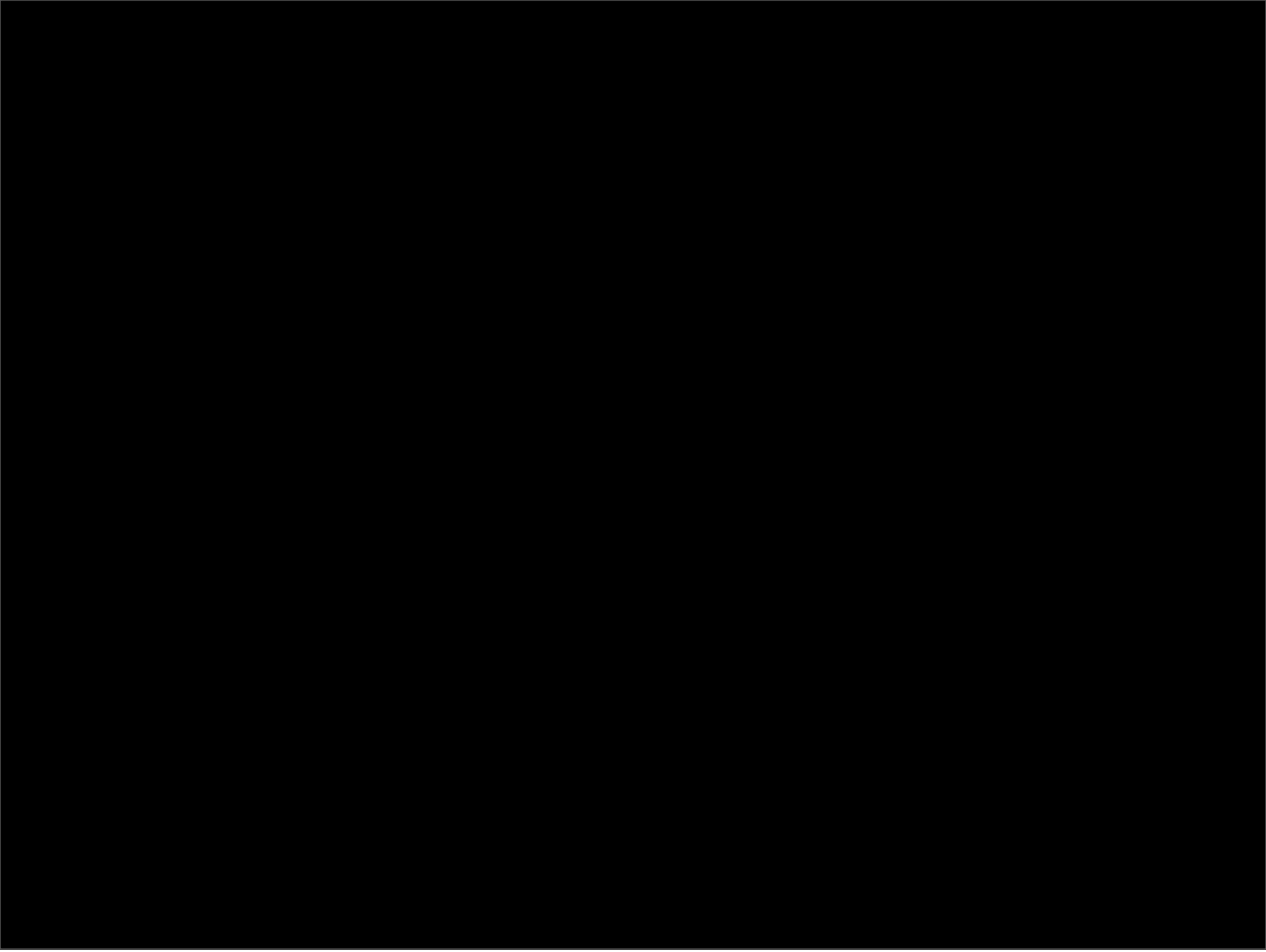


Henrik Wann Jensen

Hong et al. 2007

d’Eon and Irving, 2011

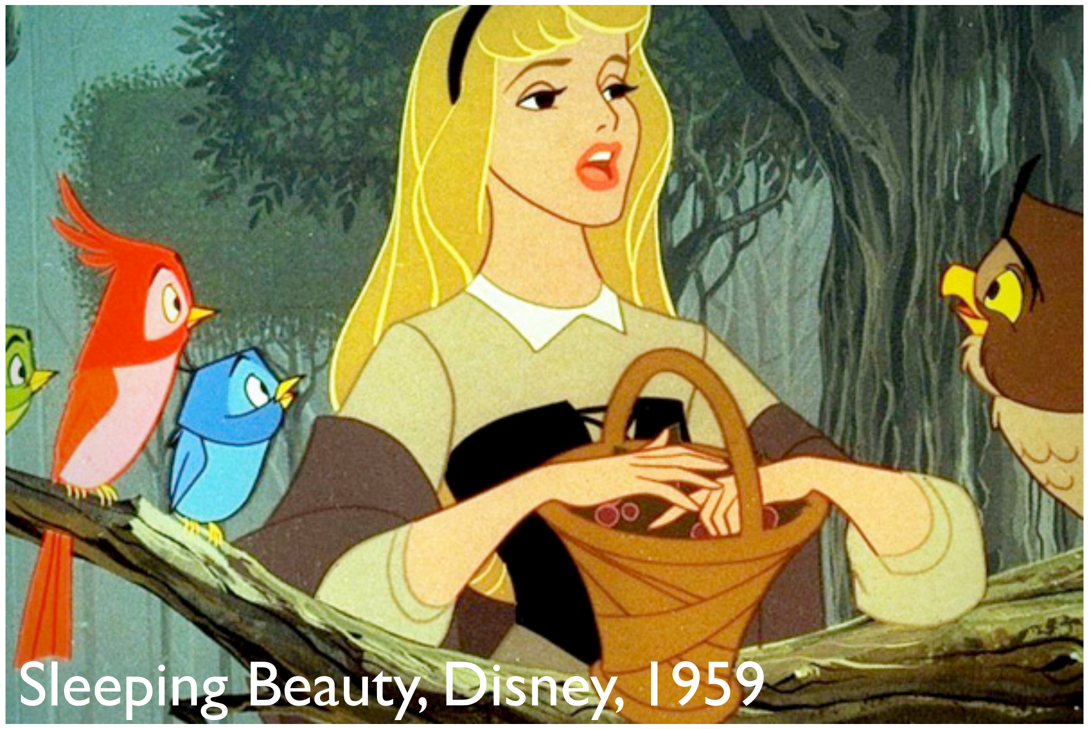
- opengl - 3D graphics (z-buffer) rendering
- **teapot - image-based lighting -** illuminated by a high dynamic range environment - metal, glass, diffuse, and glossy
- **subsurface scattering** - to capture translucent materials such as skin and marble
- rendering a emissive material such as fire - **participating medium** - scattering, absorption
- **local** vs **global** illumination



– direct vs. global illumination

- direct vs. global illumination

Animation

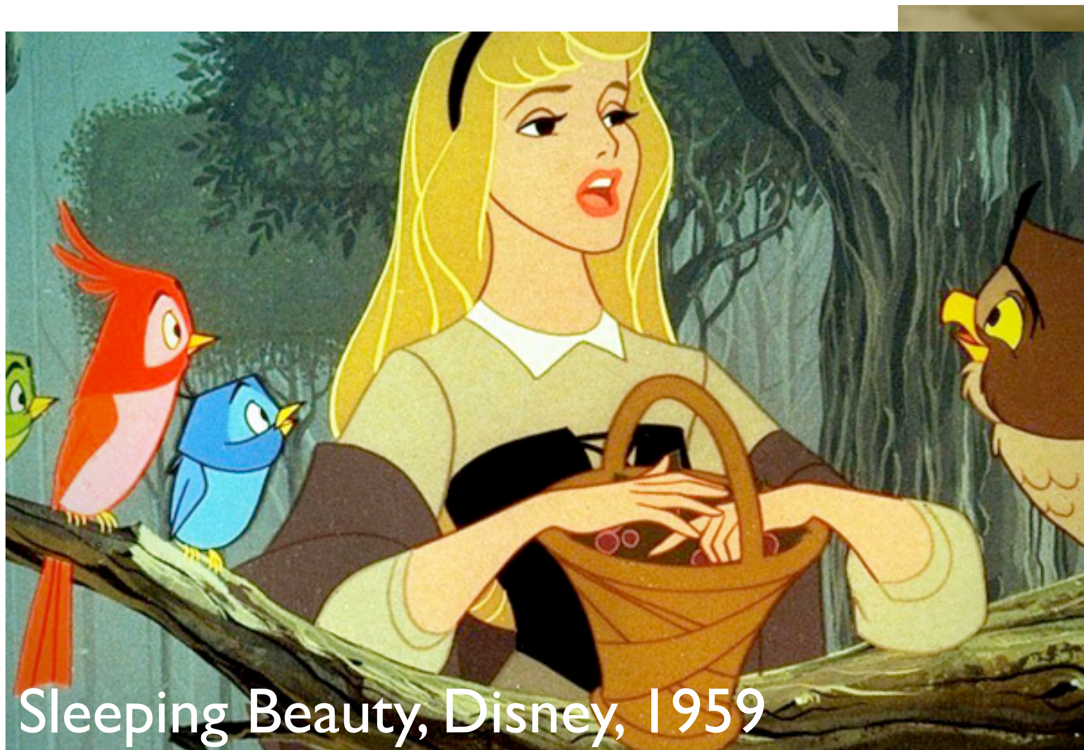

Sleeping Beauty, Disney, 1959


Monsters Inc, Pixar, 2001

Adventures of Tintin, Weta 2011


Life of Pi, 2012

Animation

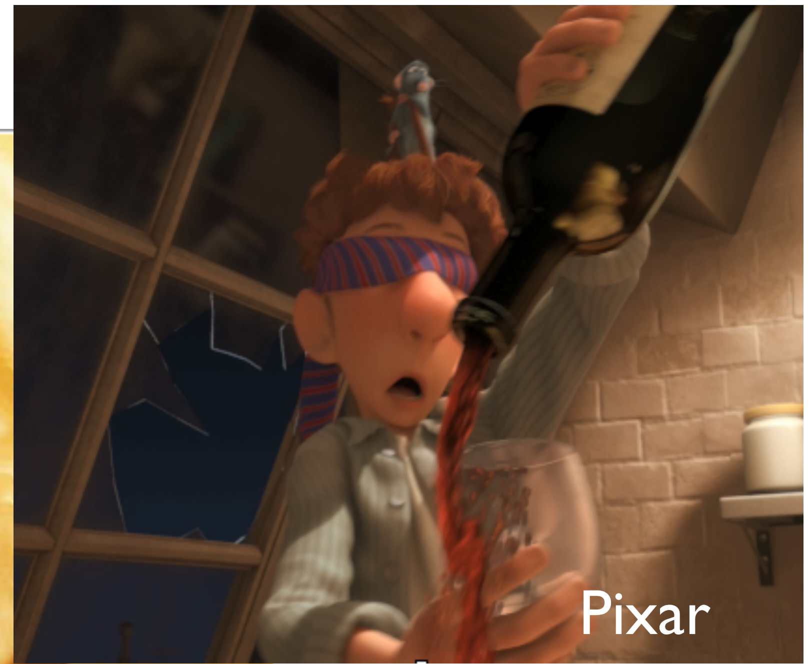

Sleeping Beauty, Disney, 1959


Adventures of Tintin, Weta 2011


Monsters Inc, Pixar, 2001


Life of Pi, 2012

Simulation



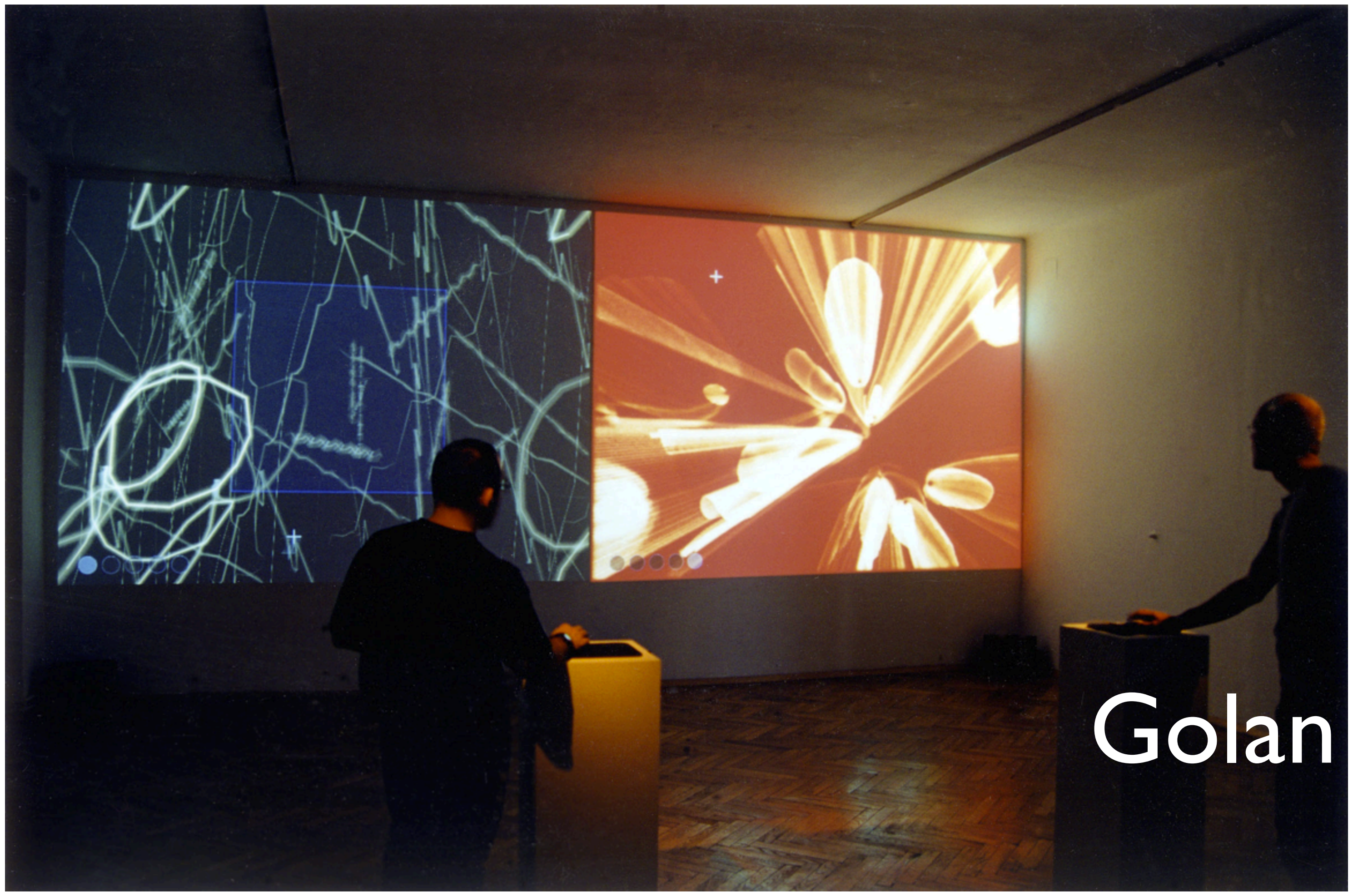
The Perfect Storm, ILM
Firestorm, Harry Potter and the Half-Blood Prince
Lord of the Rings, FOTR River Scene

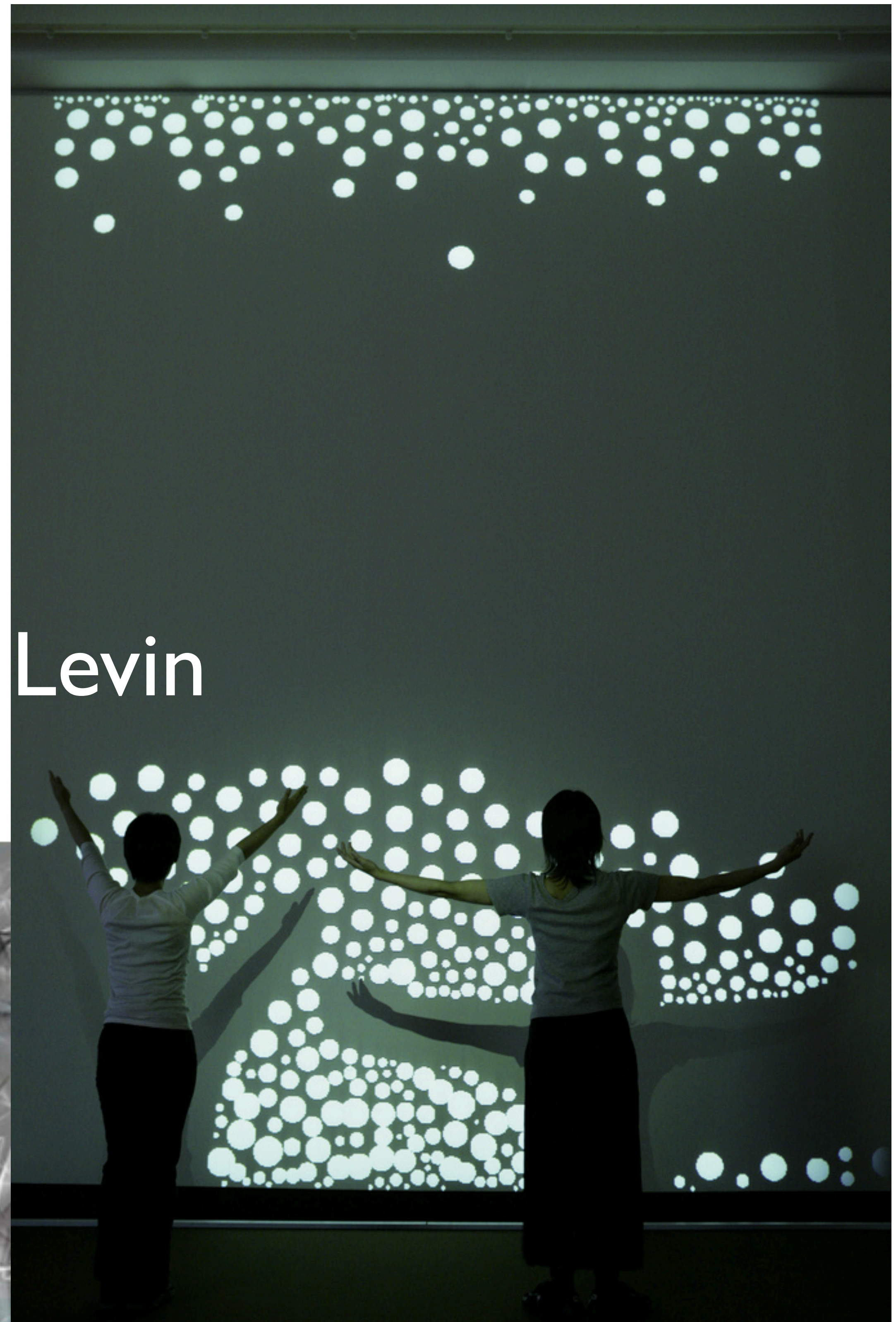
Firestorm
Harry Potter and the Half Blood Prince
Industrial Light + Magic

Firestorm
Harry Potter and the Half Blood Prince
Industrial Light + Magic

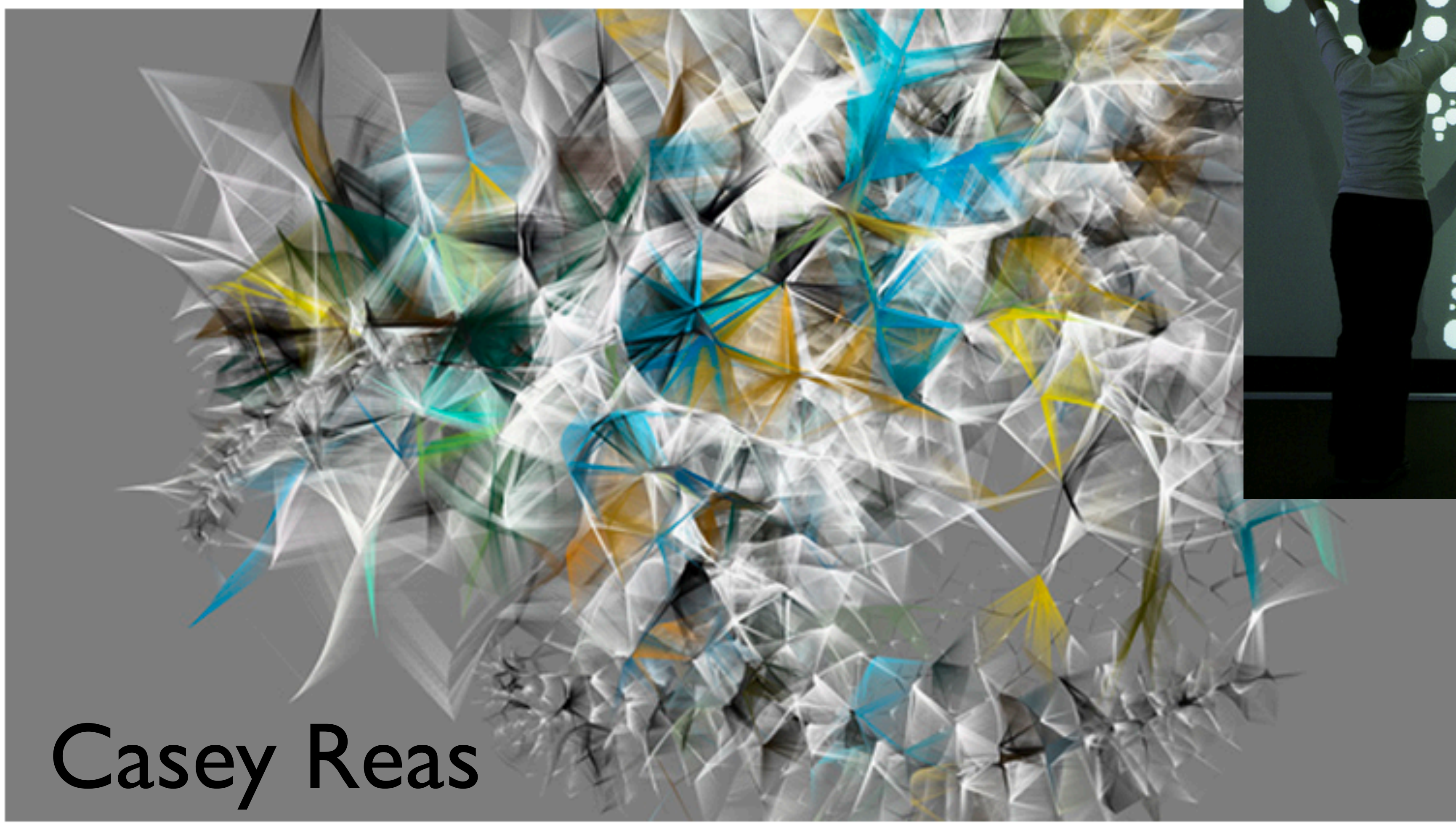
fluid simulation in Pixar's *Ratatouille*

fluid simulation in Pixar's *Ratatouille*





Golan Levin

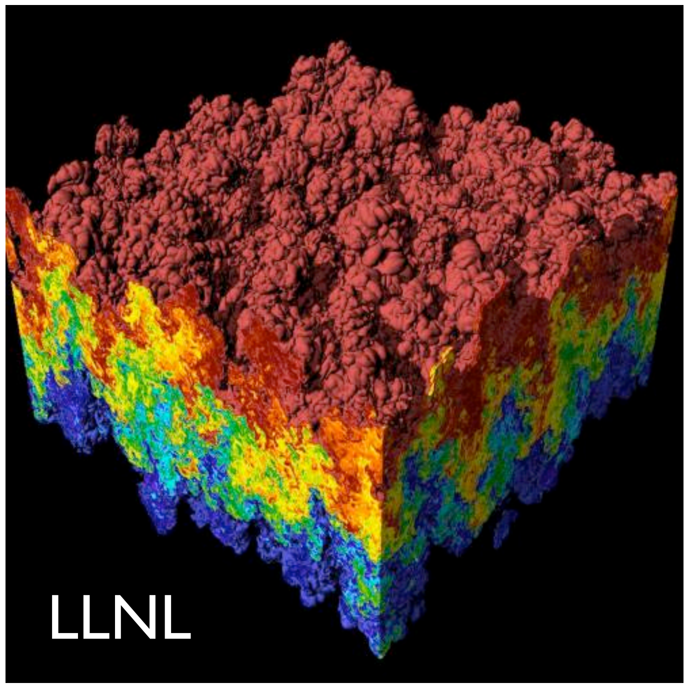


Casey Reas

Other areas...

- Interactivity (HCI)
- Image processing
- Visualization
- Computational photography


L-1 Identity Solns


LLNL

Lytro

Microsoft Kinect

– Lytro demo: http://www.lytro.com/living-pictures/2325

Introduction to OpenGL

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

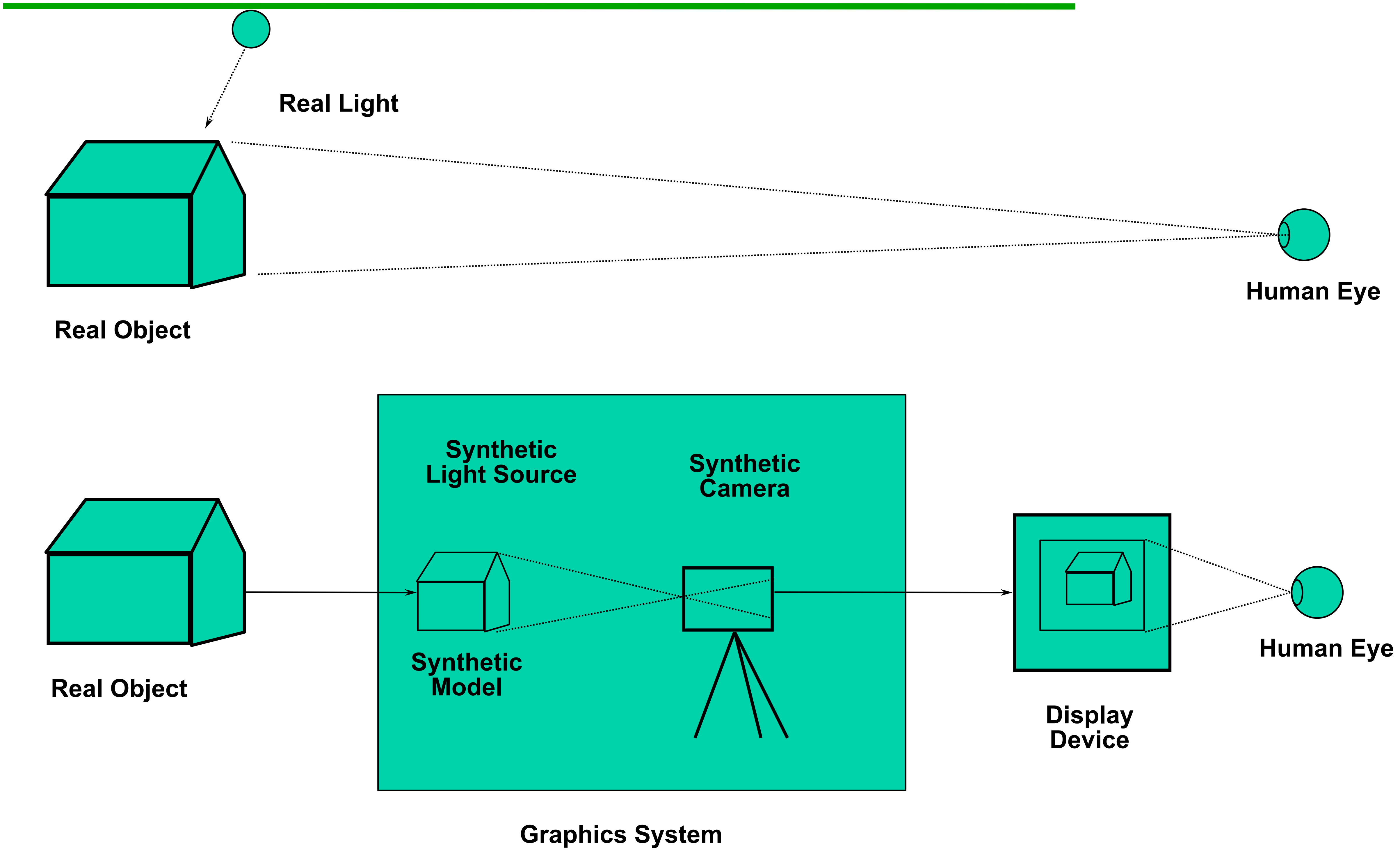
Introduction to OpenGL

- **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**

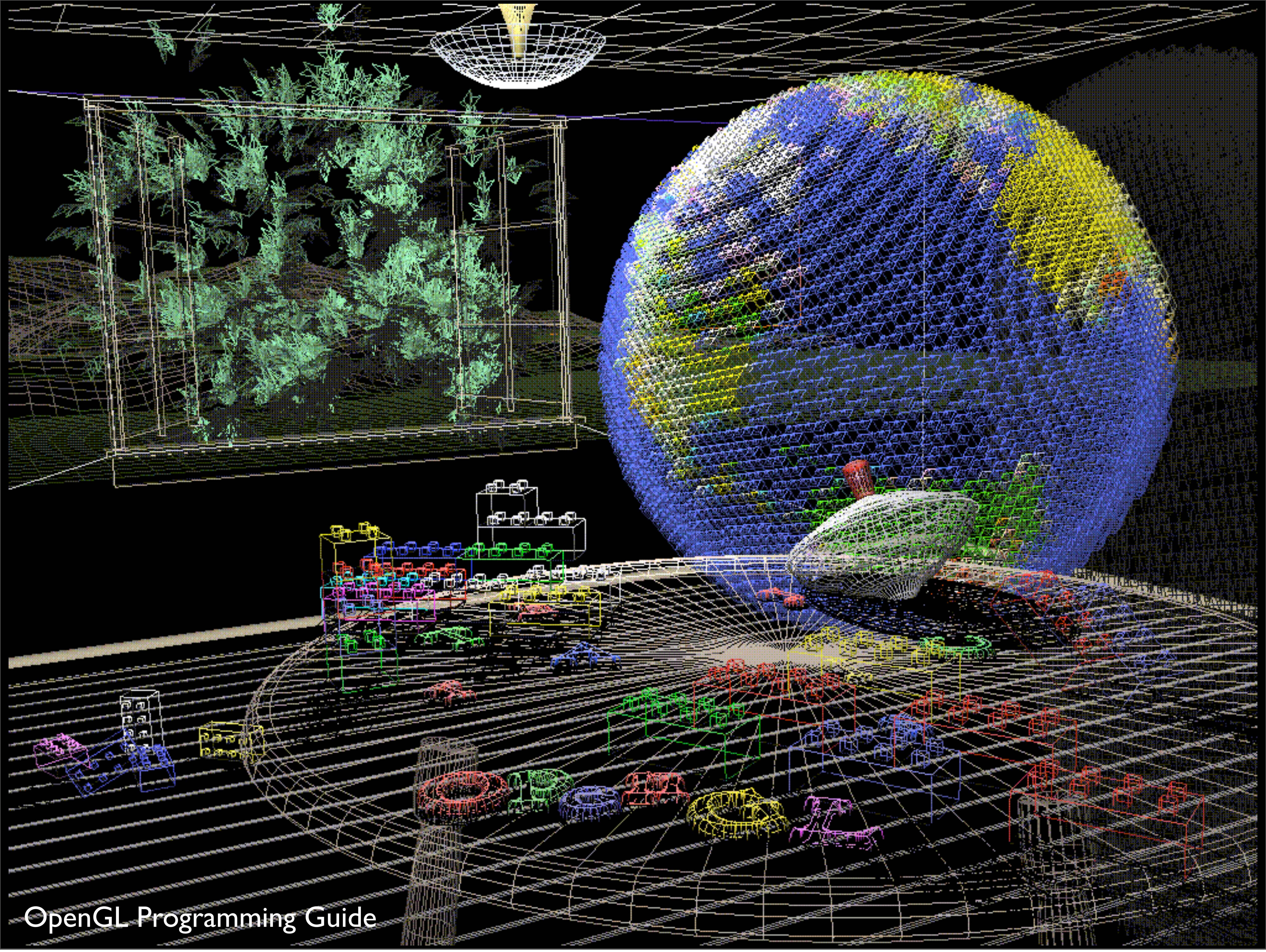
- used to produce interactive 3D graphics
- sits between programmer and 3D accelerators in hardware
- **standard** requires support for feature set for all implementations
- Both OpenGL and Direct3D support feature sets -- they take advantage of hardware acceleration or use software emulation when a feature is unavailable in hardware
- Direct3D is proprietary
- OpenGl and Direct3D both implemented in the display driver

OpenGL: Conceptual Model

OpenGL: Conceptual Model



What can OpenGL do?
Examples from the
OpenGL Programming Guide ("red book")



- **Wireframe** models
  - shows each object made up of polygons
- the **lines are are the edges** and the **faces of the polygons make up the object surface**

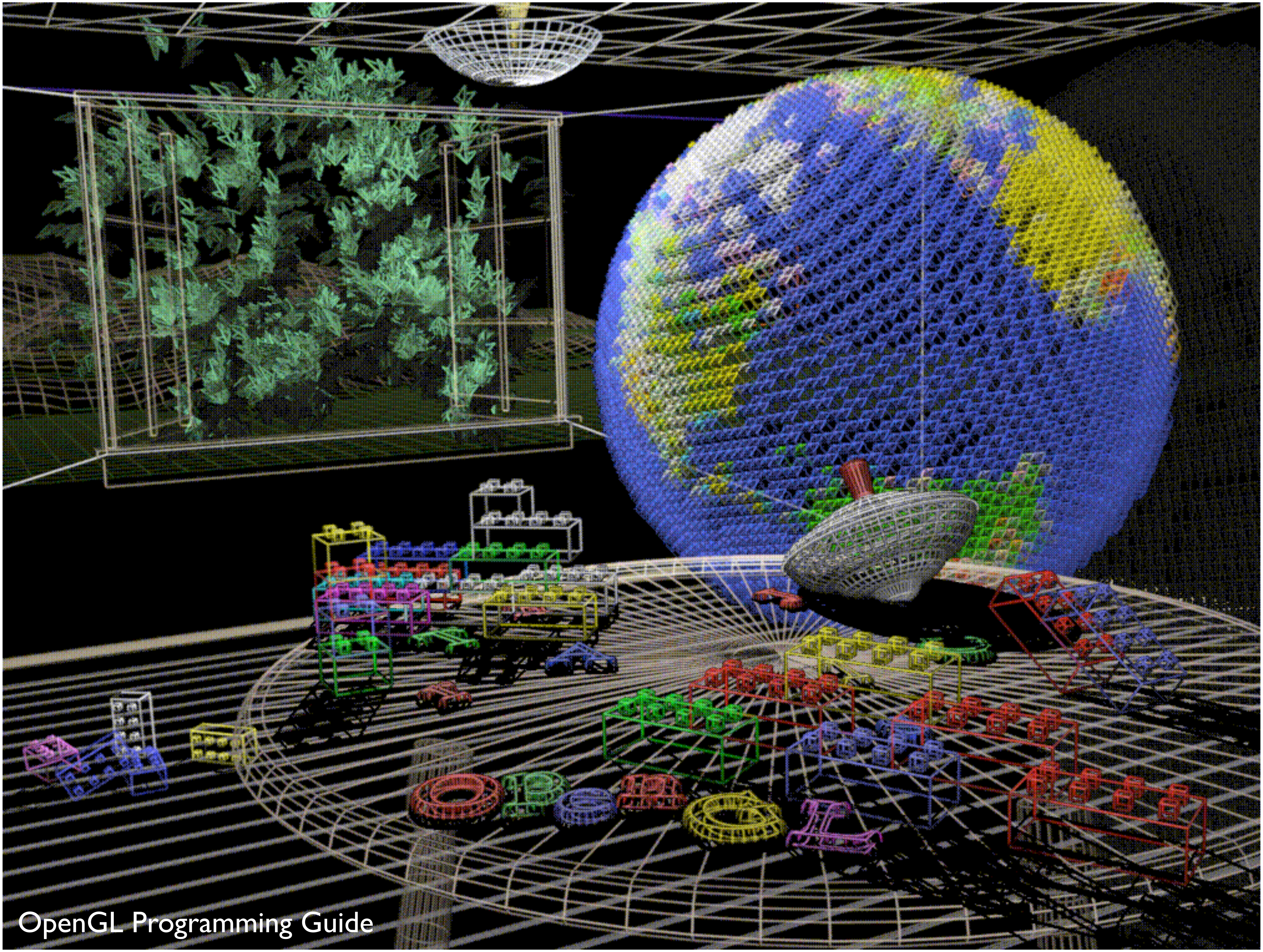


Plate 3. The same scene with **antialiased lines** that **smooth the jagged edge**s. See Chapter 7 .

when you approximate smooth edges using pixels, this leads to jagged lines
especially with near vertical and near horizontal lines

Plate 4. The scene drawn with **flat-shaded polygons** (a **single color for each filled polygon**). See Chapter 5 .

“unlit scene”

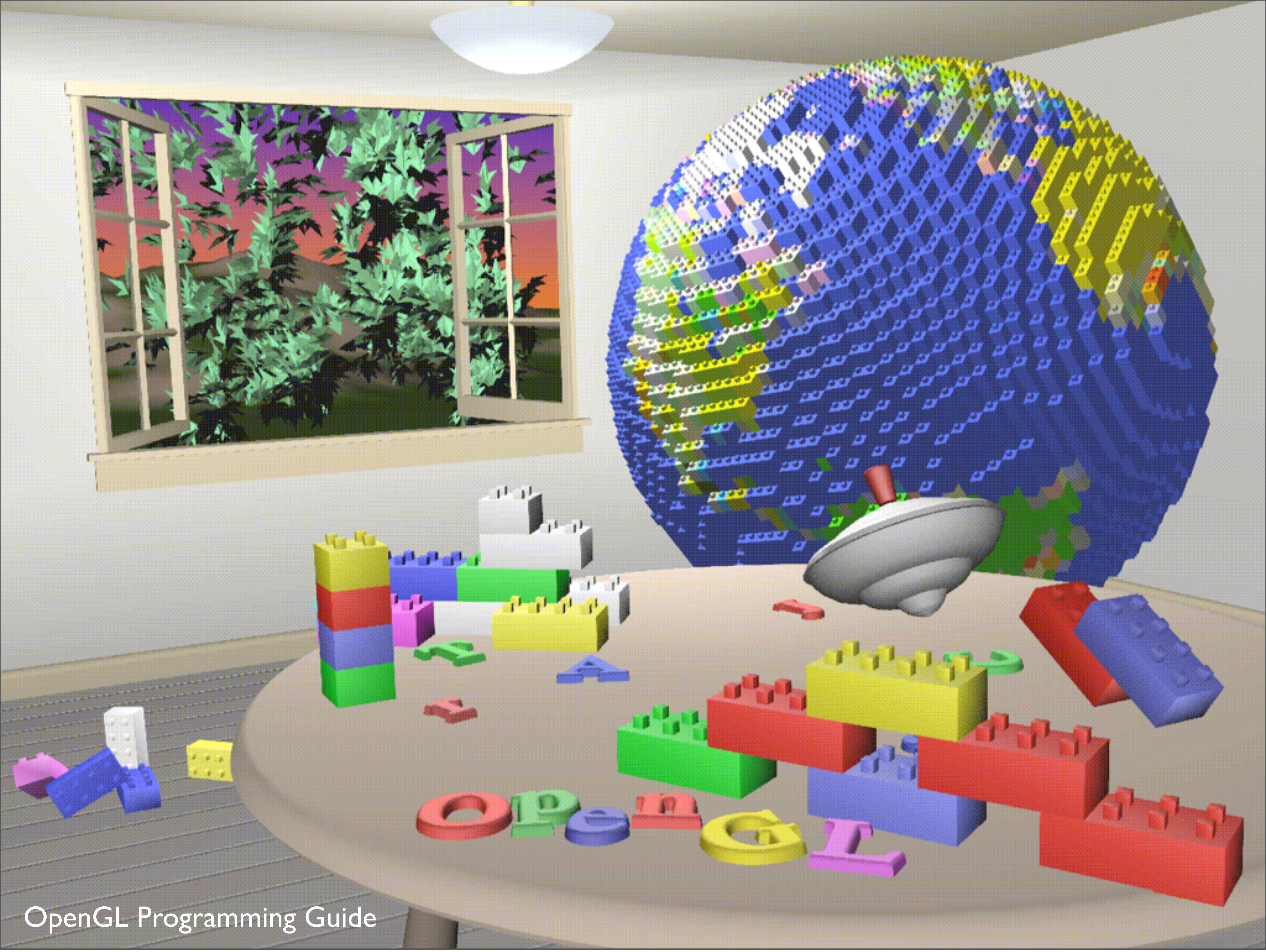


Plate 5. The scene rendered with **lighting** and **smooth-shaded polygons.** See Chapter 5 and Chapter 6 .

Plate 6. The scene with **texture maps and shadows added.** See Chapter 9 and Chapter 13 .

Plate 7. The scene drawn with one of the objects **motion-blurred**. The **accumulation buffer** is used **to compose the sequence of images** needed to blur the moving object. See Chapter 10 .

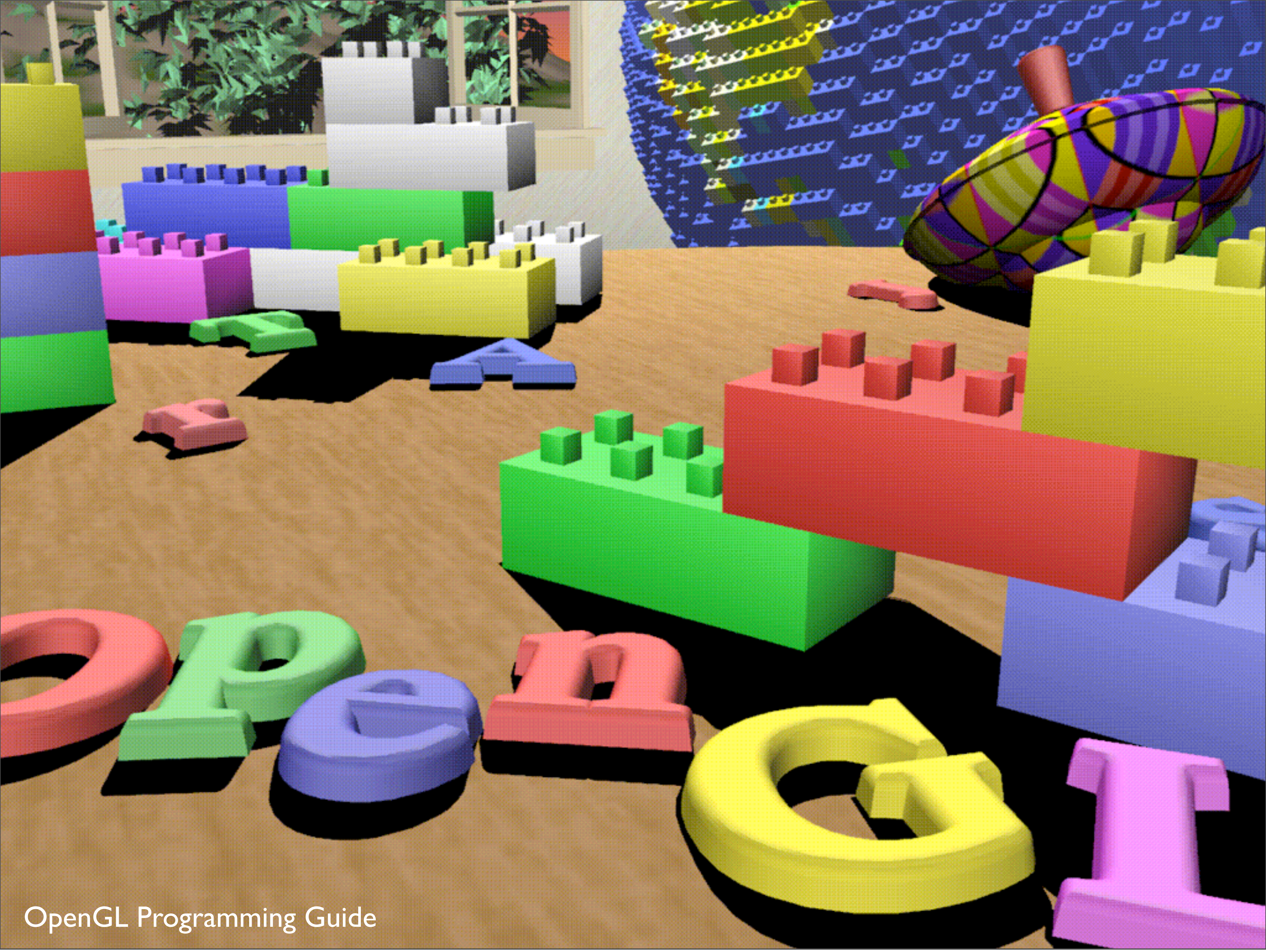


Plate 8. A close-up shot - the scene is **rendered from a new viewpoint.** See Chapter 3 .

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

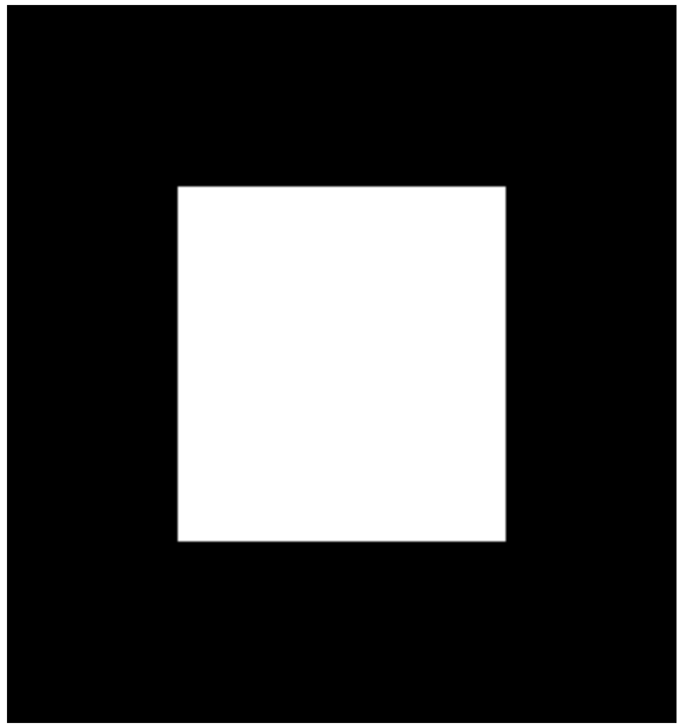
```
    #include <whateverYouNeed.h>

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.

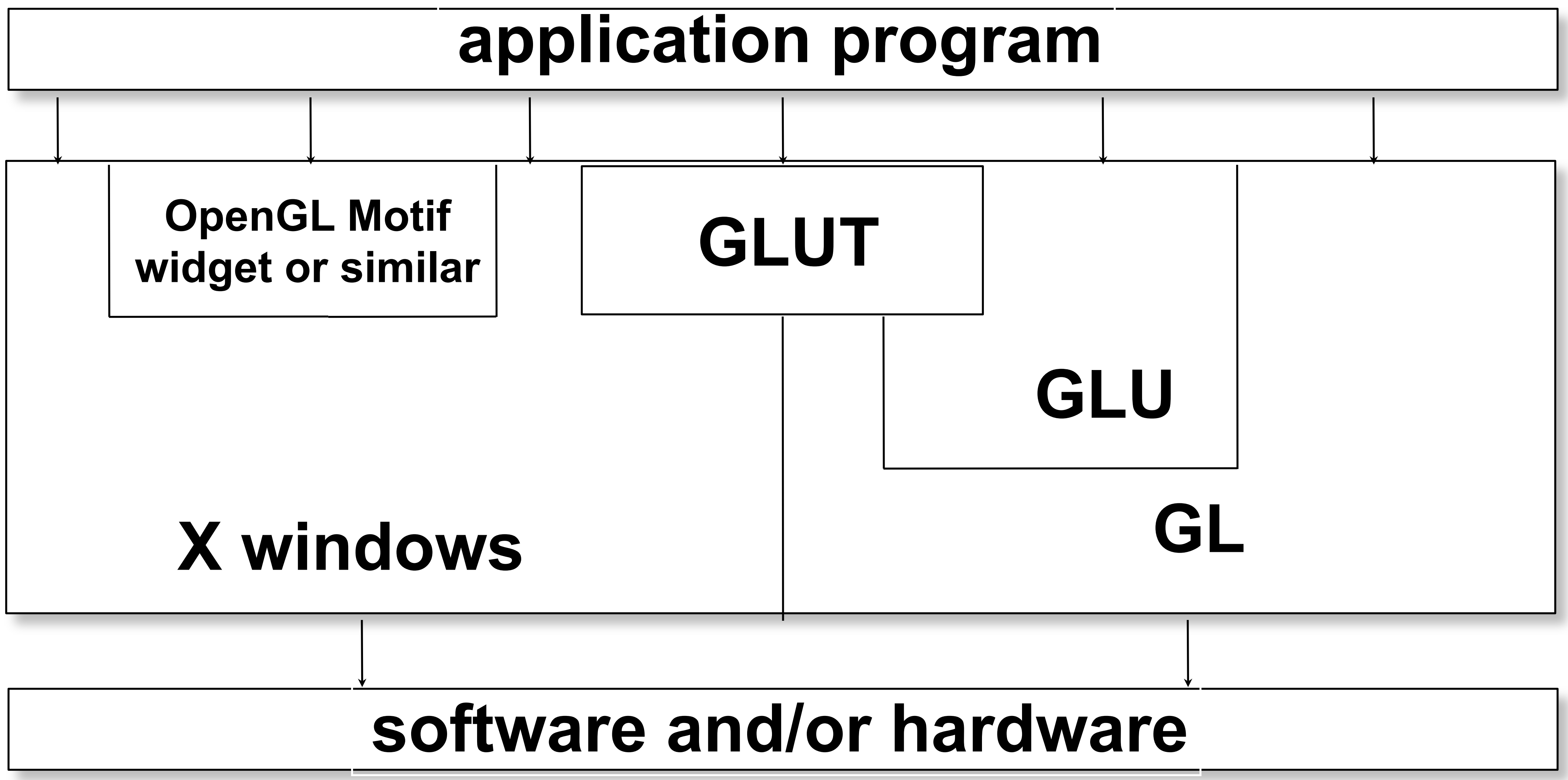
- blue are placeholders for windowing system commands
- clear color, actual clear
- Ortho – the coordinate system
- flush executes the commands

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

- GL
  - no windowing commands
  - no commands for higher-level geometry - you build these using primitives (points, lines, polygons)
- GLU - standard in every implementation
- OpenGL Utility library provides modeling support
  - quadratic surfaces, NURBS curves and surfaces

Software Organization



Simple OpenGL program

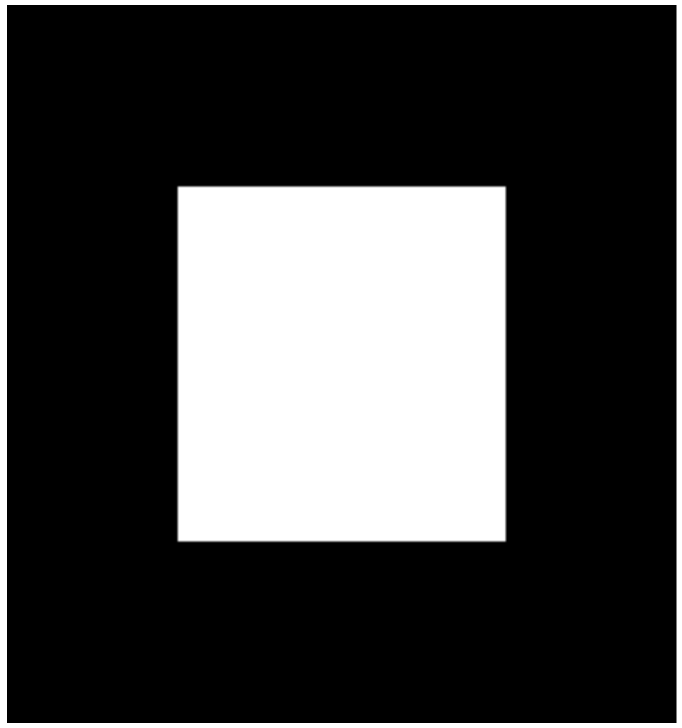
```
   #include <whateverYouNeed.h>

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.

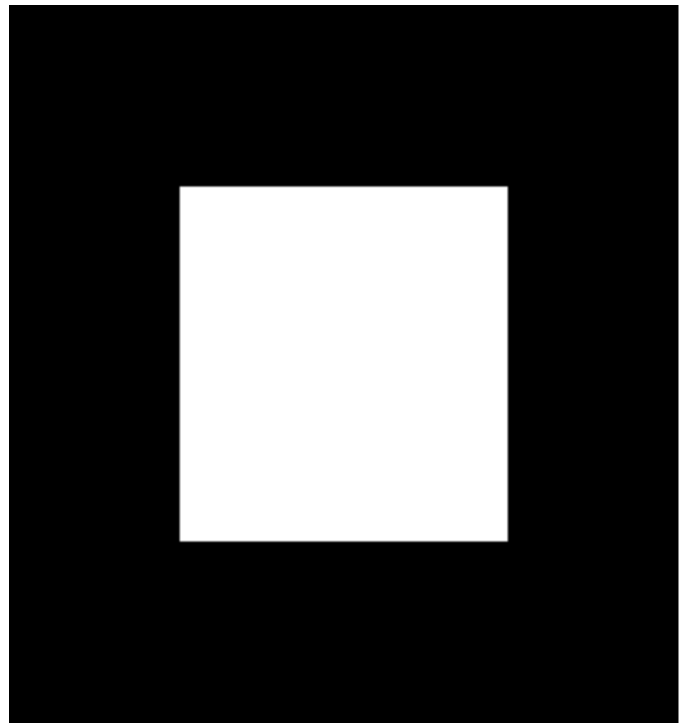
- blue are placeholders for windowing system commands
-can replace blue code with calls to **glut**

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();
}
```



- blue are placeholders for windowing system commands
-can replace blue code with calls to **glut**

Math Review
<whiteboard>