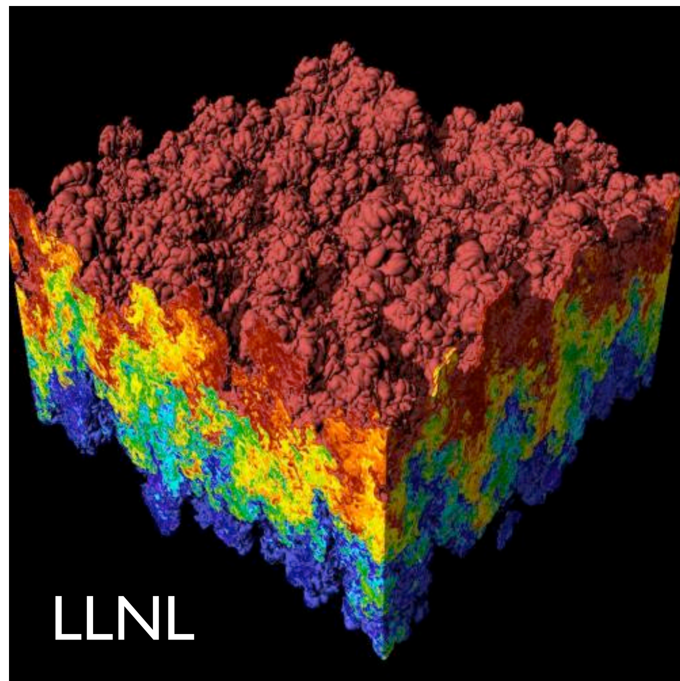
CS230 : Computer Graphics

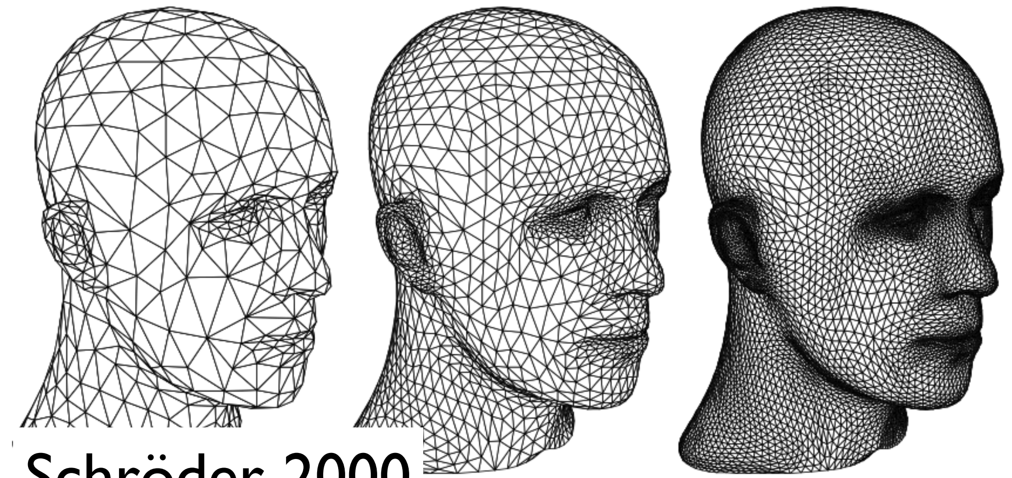
Fall 2014

Tamar Shinar
Computer Science & Engineering
UC Riverside

Welcome to CS230!


Talton et al., 2011


LLNL


Schröder, 2000


ILM


Henrik Wann Jensen


Hong et al. 2007


Pixar

Today's agenda

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

Course Logistics

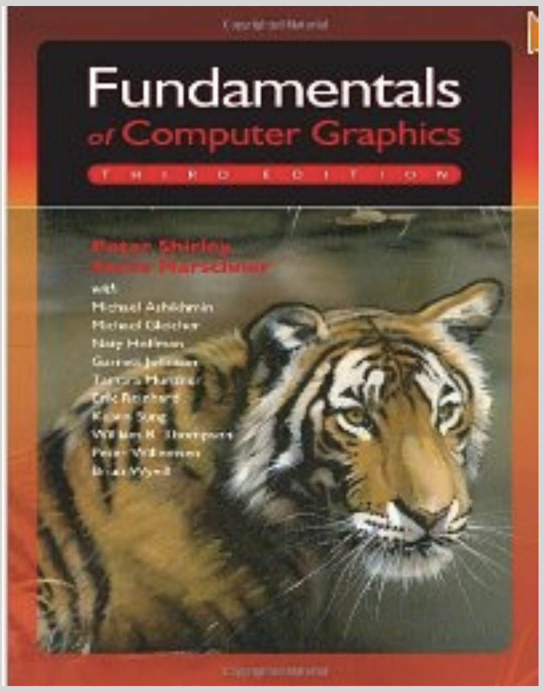
- Instructor: Tamar Shinar
- Website: http://www.cs.ucr.edu/~shinar/courses/cs230
- Lectures: TuTh, 12:40-2pm
- Office hours: Tu, 11am-12pm, WCH 419

Course Logistics

- Grading
  - 15% quizzes and exercises
  - 50% assignments (2 assignments, each ~2-3 weeks)
  - 35% final project
  - No exams
- Total of 2 late days (48 hours) for the quarter for the assignments only
- final project must be submitted on time
- assignments individual; project individual or group of 2

- quizzes and exercises
  - some in class problems - only graded for correctness if we've already covered it
    - otherwise only graded for presence and effort
  - may ask someone to work a problem
- quiz will normally be in the first 5-10 minutes of class -- today we'll have a short one at the end that you will get full credit on -- check your own math skills and give me a sense of class's math skill
- Q. how many people have taken graphics before? MS students? PhD students? Want to go on to work in graphics?
- final project:
  - there will be a proposal due

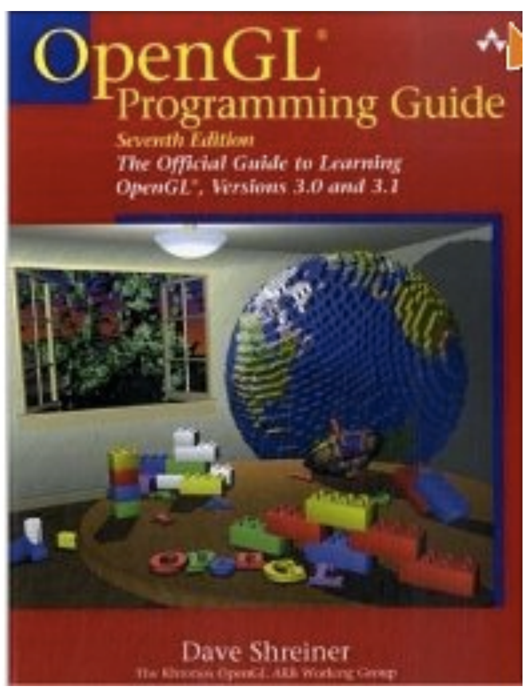
Textbook

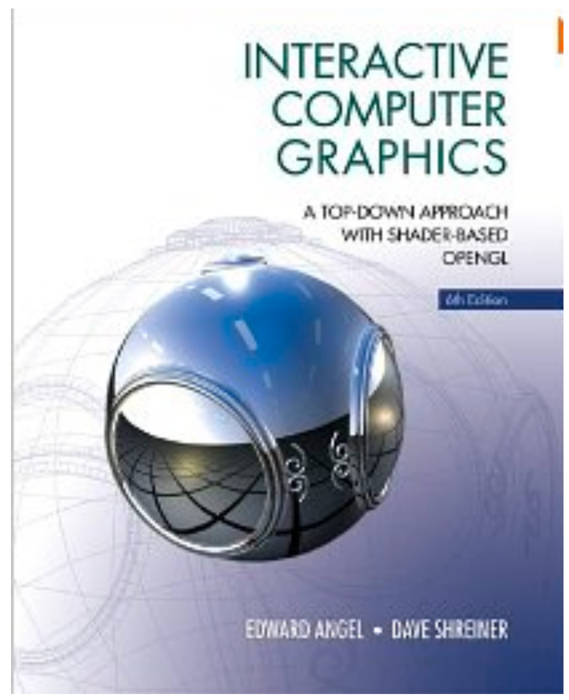


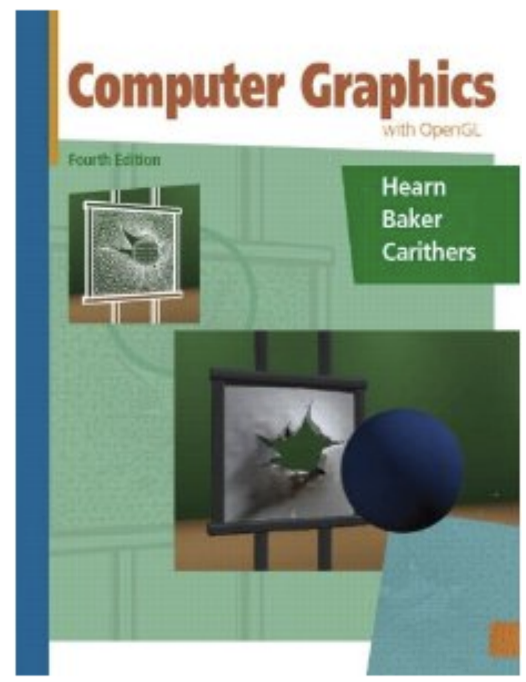
Fundamentals of Computer Graphics

Shirley and Marschner

Additional books







if you like using a book
- red book older version online:http://fly.cc.fer.hr/~unreal/theredbook/
And if you prefer -- all material is online in one form or another -- you don't have to buy a book but it can be useful for a coherent presentation

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

Course overview

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

Course schedule

see course website for up-to-date schedule

Introduction

Graphics applications

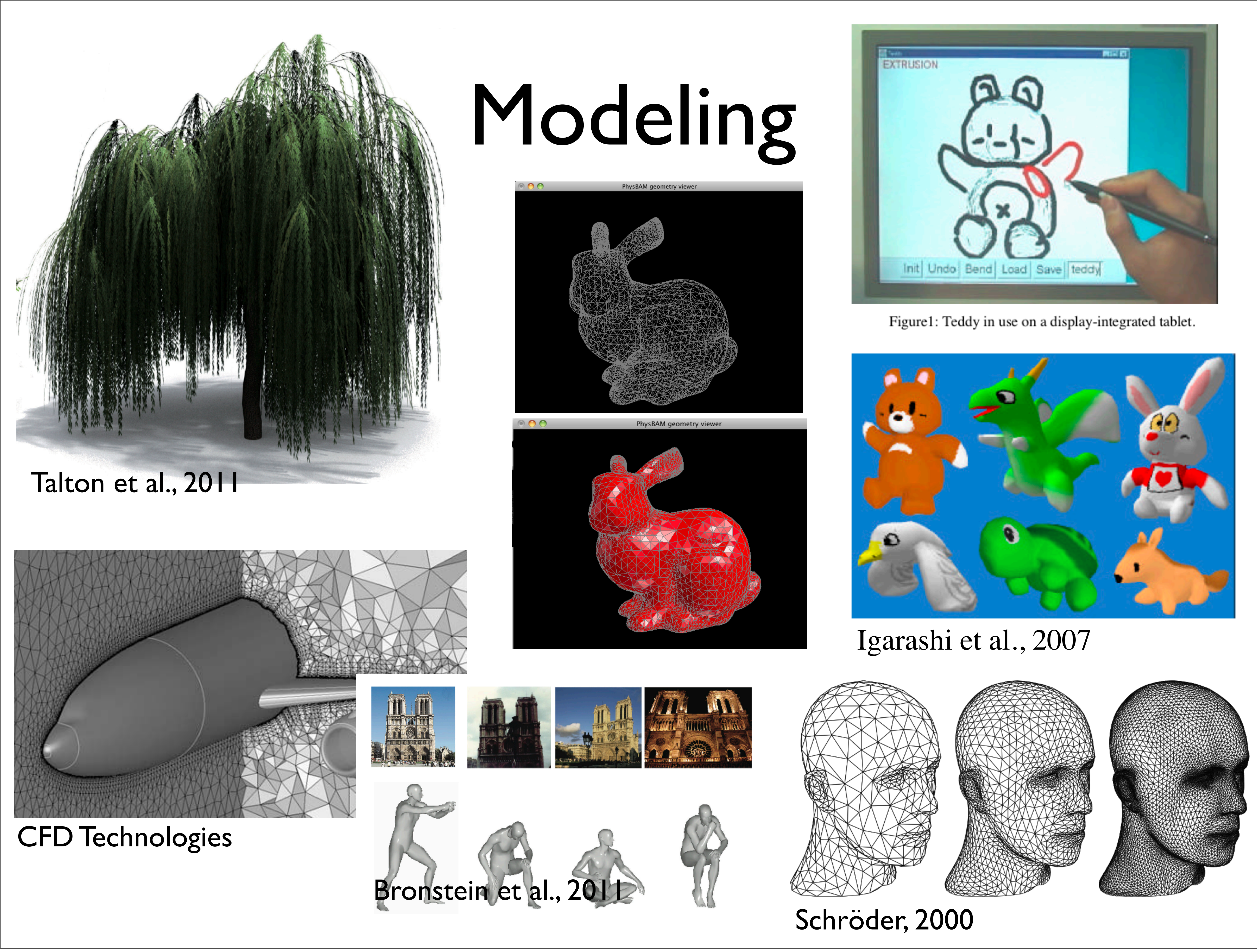
- 2D drawing
- Drafting, CAD
- Geometric modeling
- Special effects
- Animation
- Virtual Reality
- Games
- Educational tools
- Surgical simulation
- Scientific and information visualization
- Fine art

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

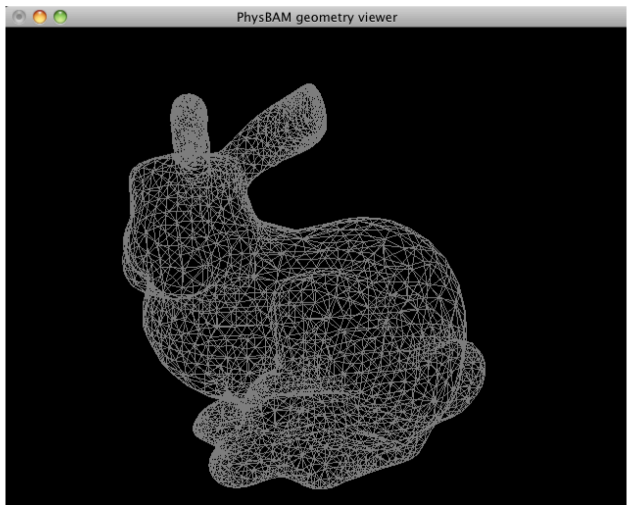
Which area would you like your final project to be in?

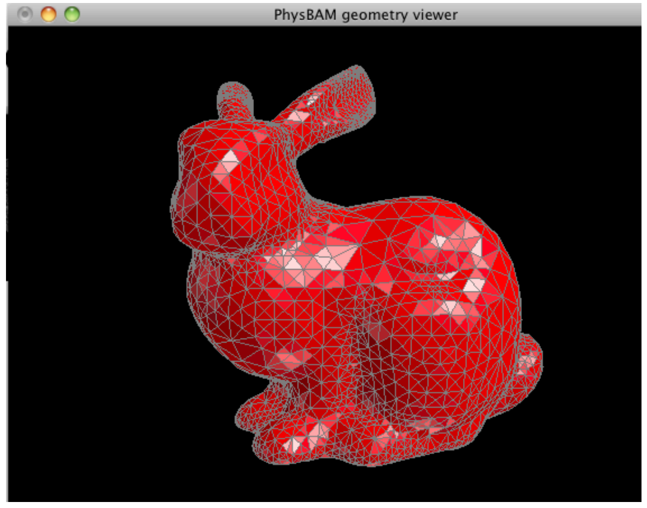
Think about which area interests you, dovetails with your present or future research, or that you want to learn more about
Modeling and **rendering** are separate stage
- first design and position objects -- **modeling**
- then add lights, materials properties, effects -- **rendering**



- 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



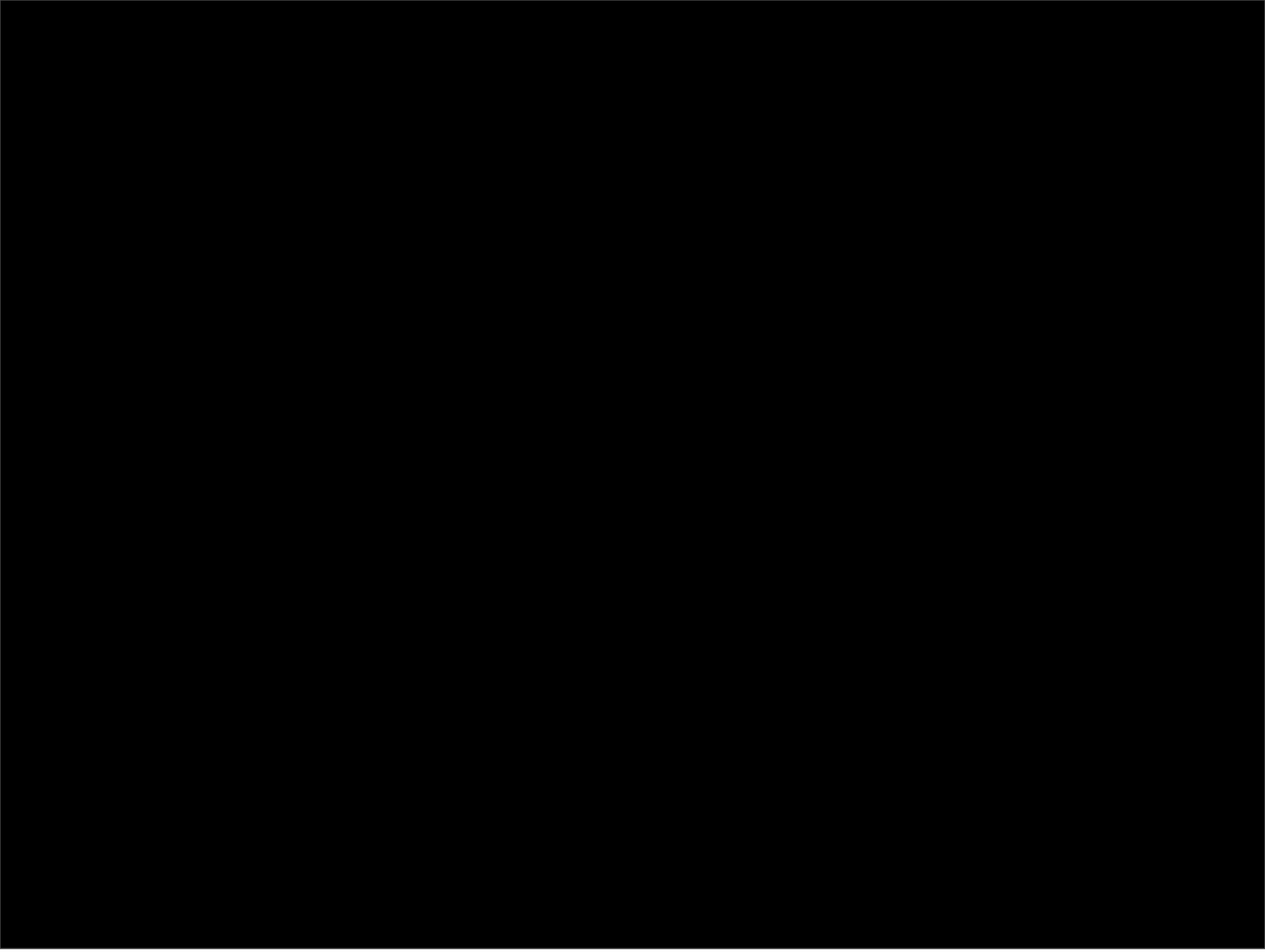


Hong et al. 2007

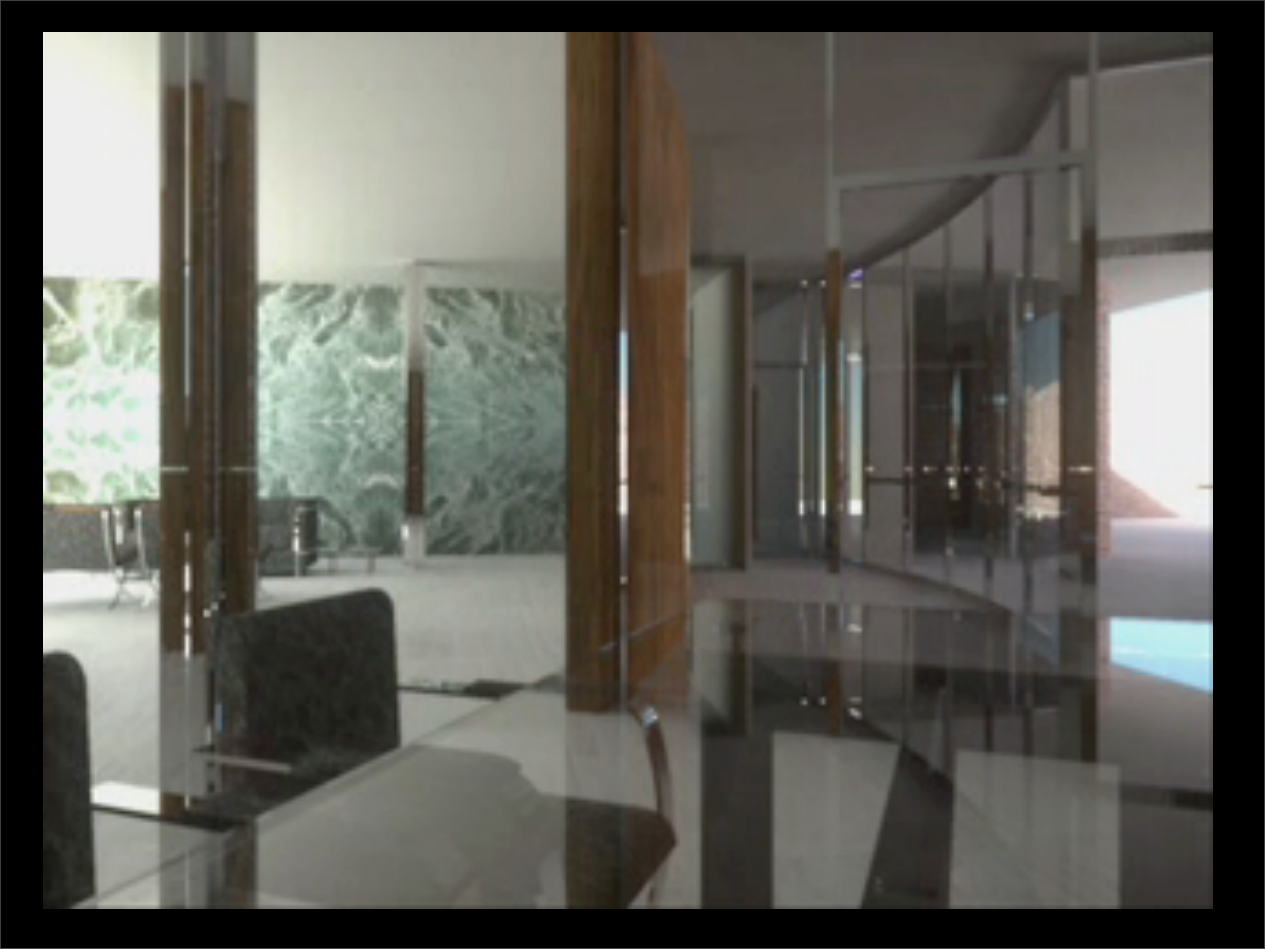
Henrik Wann Jensen

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

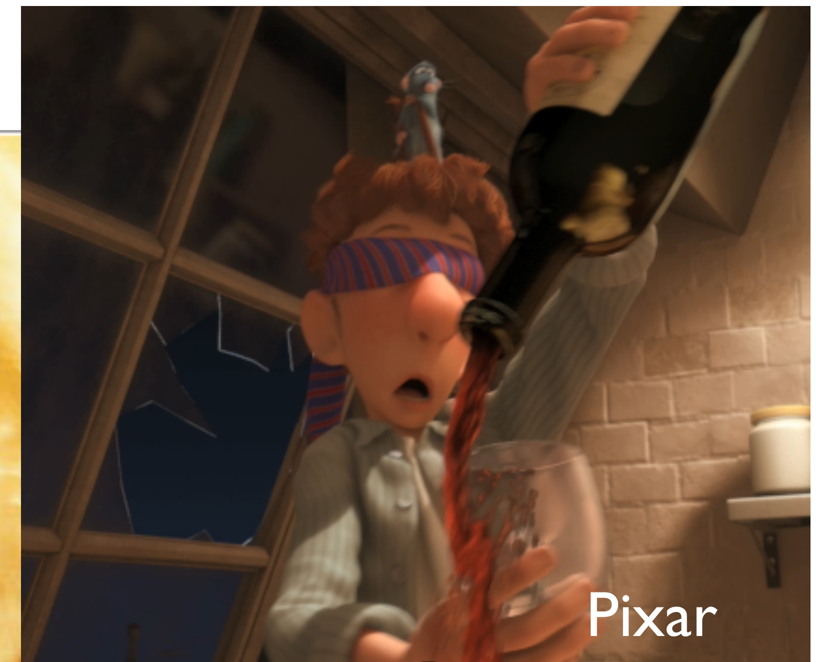
Adventures of Tintin, Weta 2011

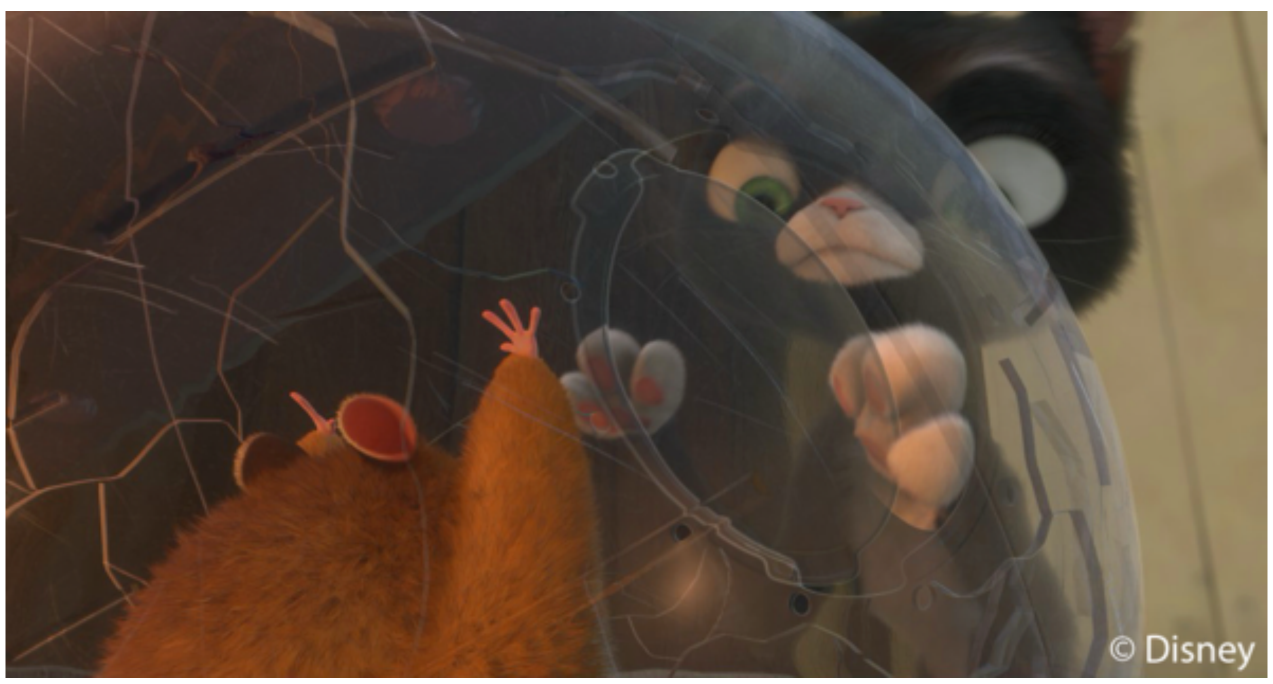
Animation

Sleeping Beauty, Disney, 1959

Adventures of Tintin, Weta 2011

Simulation



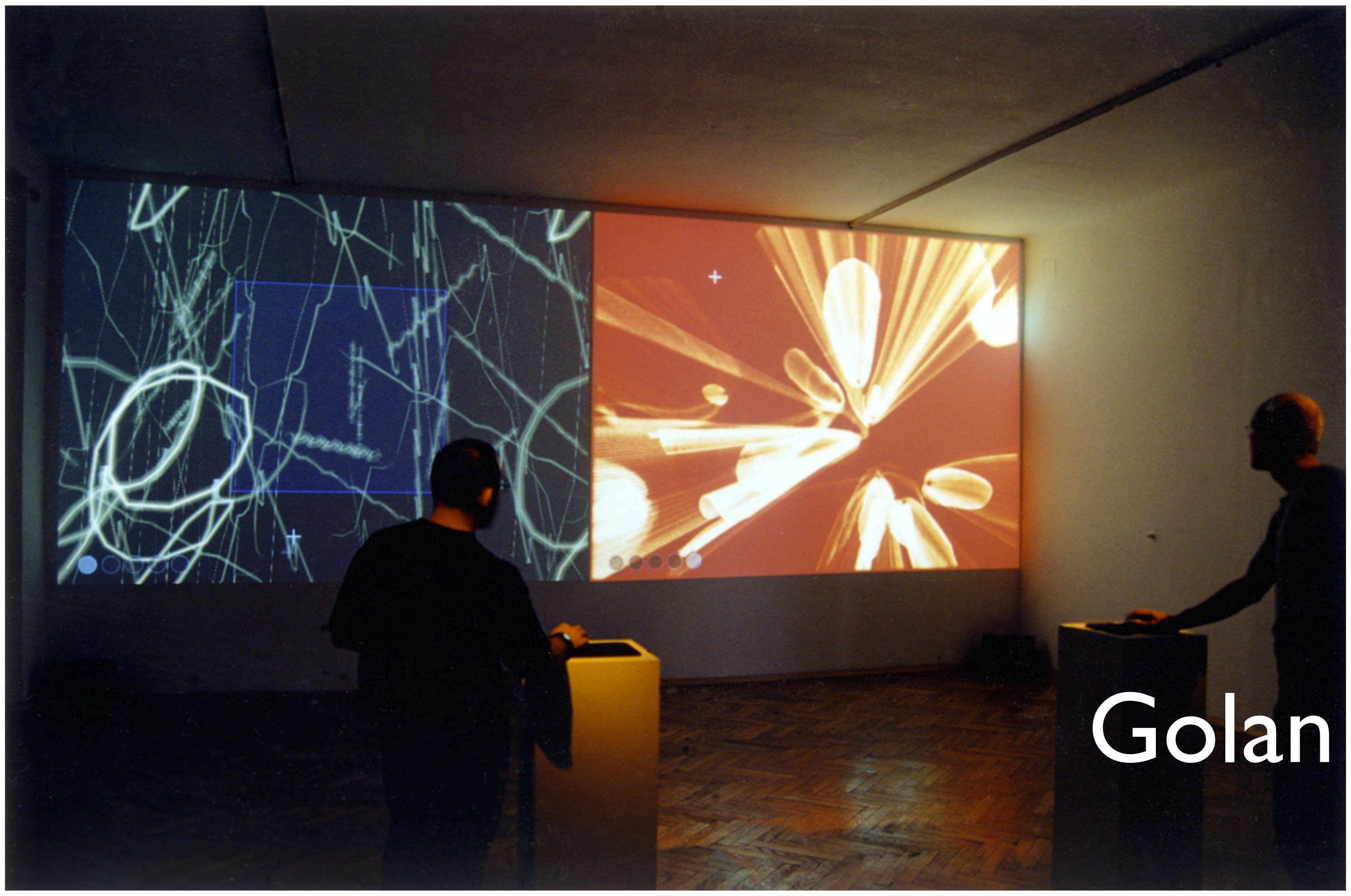


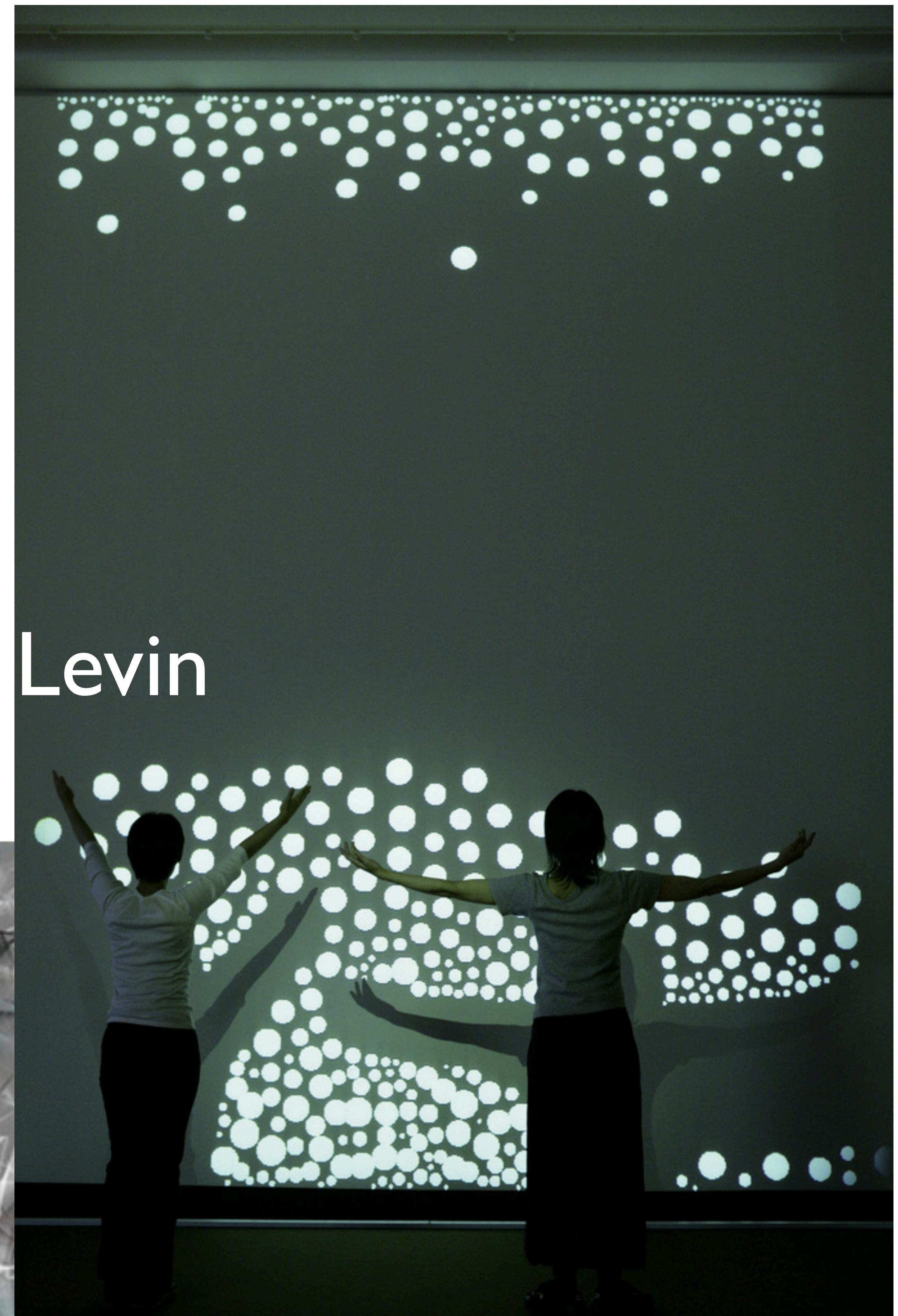
Firestorm
Harry Potter and the Half Blood Prince
Industrial Light + Magic

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

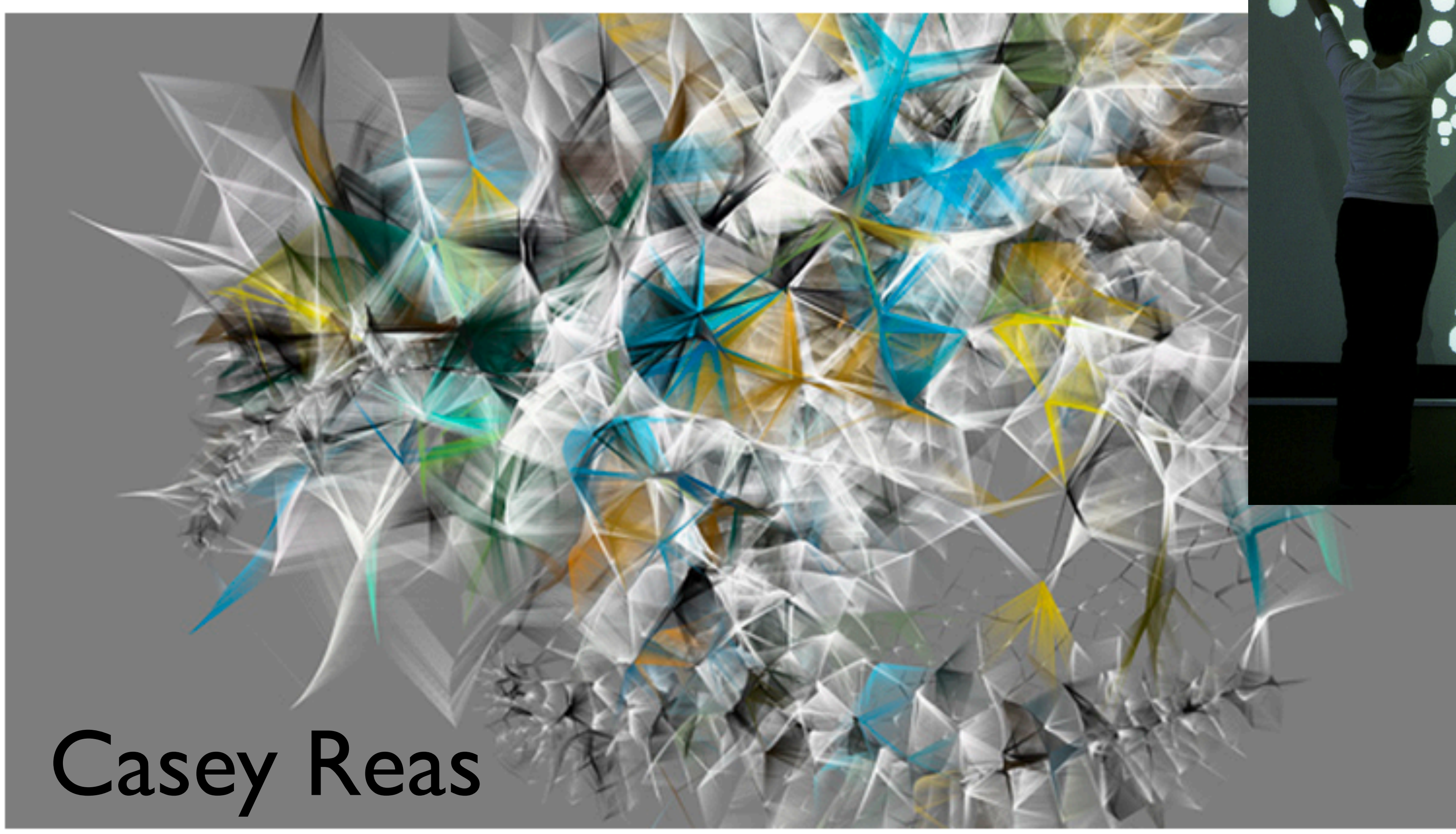
fluid simulation in Pixar's *Ratatouille*

fluid simulation in Pixar's *Ratatouille*





Golan Levin

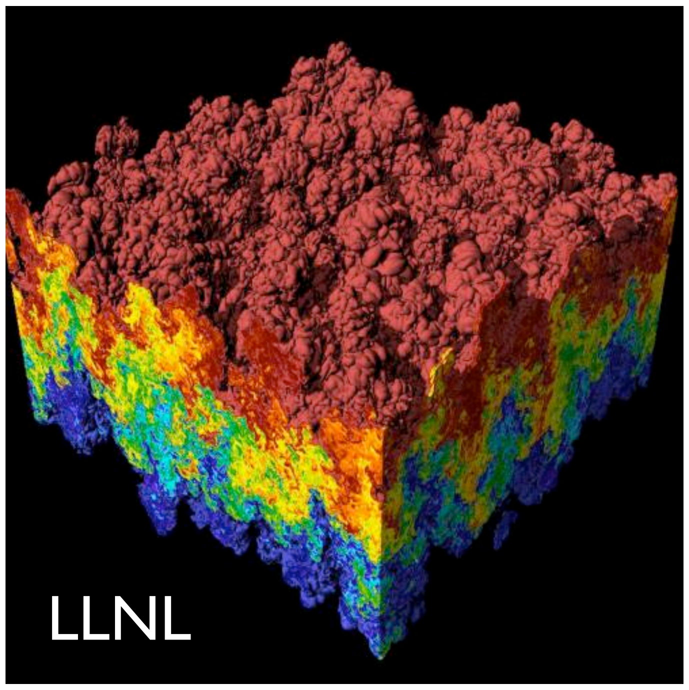


Casey Reas

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

Introduction to OpenGL

- **Open G**raphics **L**ibrary, managed by Khronos Group
- API for drawing 2D and 3D graphics
  - communicates with GPU
    - accelerate graphics rendering
- Standard API with support for multiple languages and platforms, open source
    - functions and named integer constants
    - many language bindings
      - e.g., JavaScript binding WebGL (browser-based)

- 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

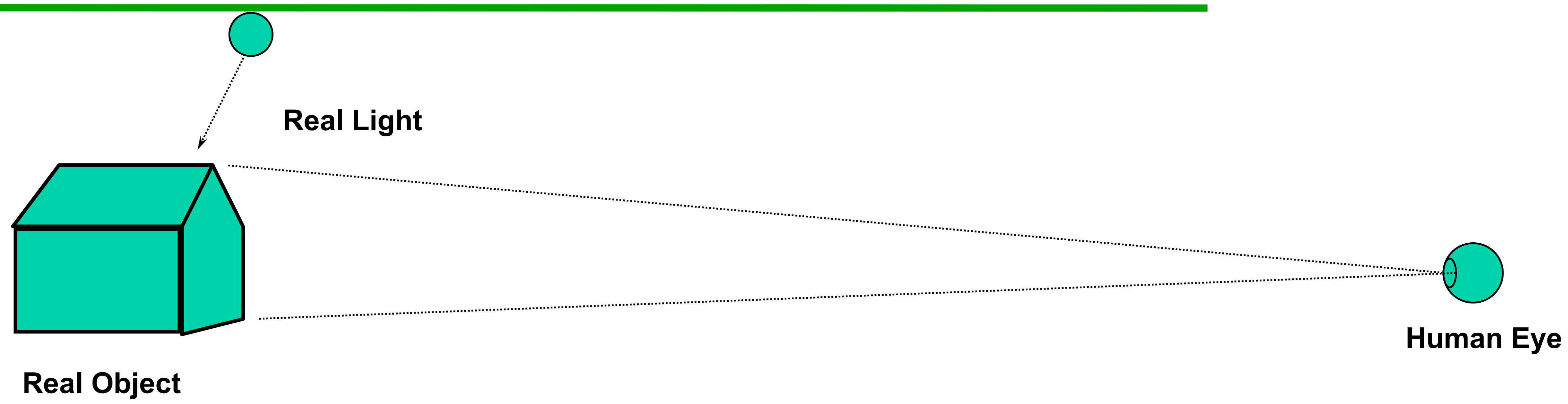
OpenGL - Software to Hardware

- 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

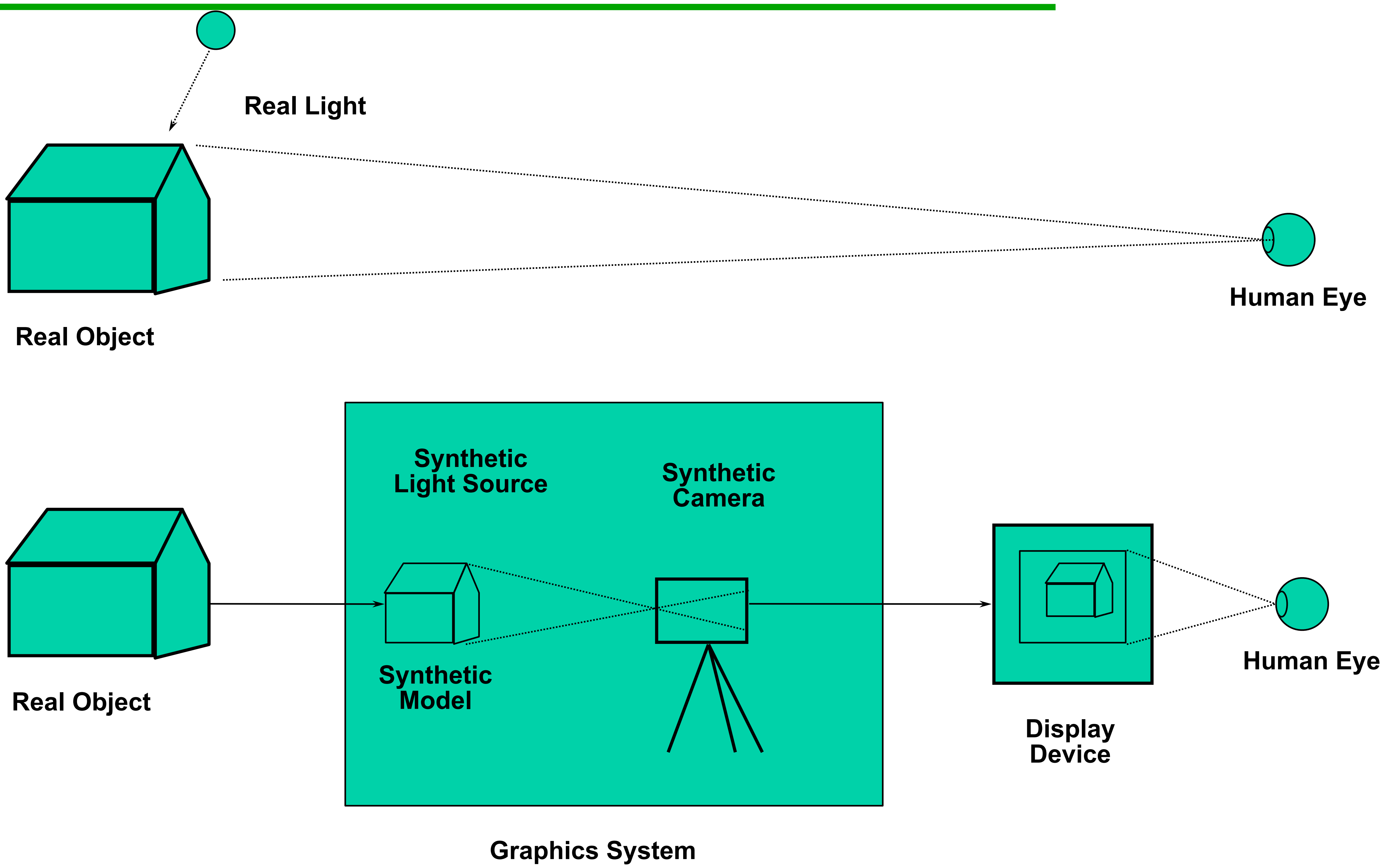
OpenGL

- 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

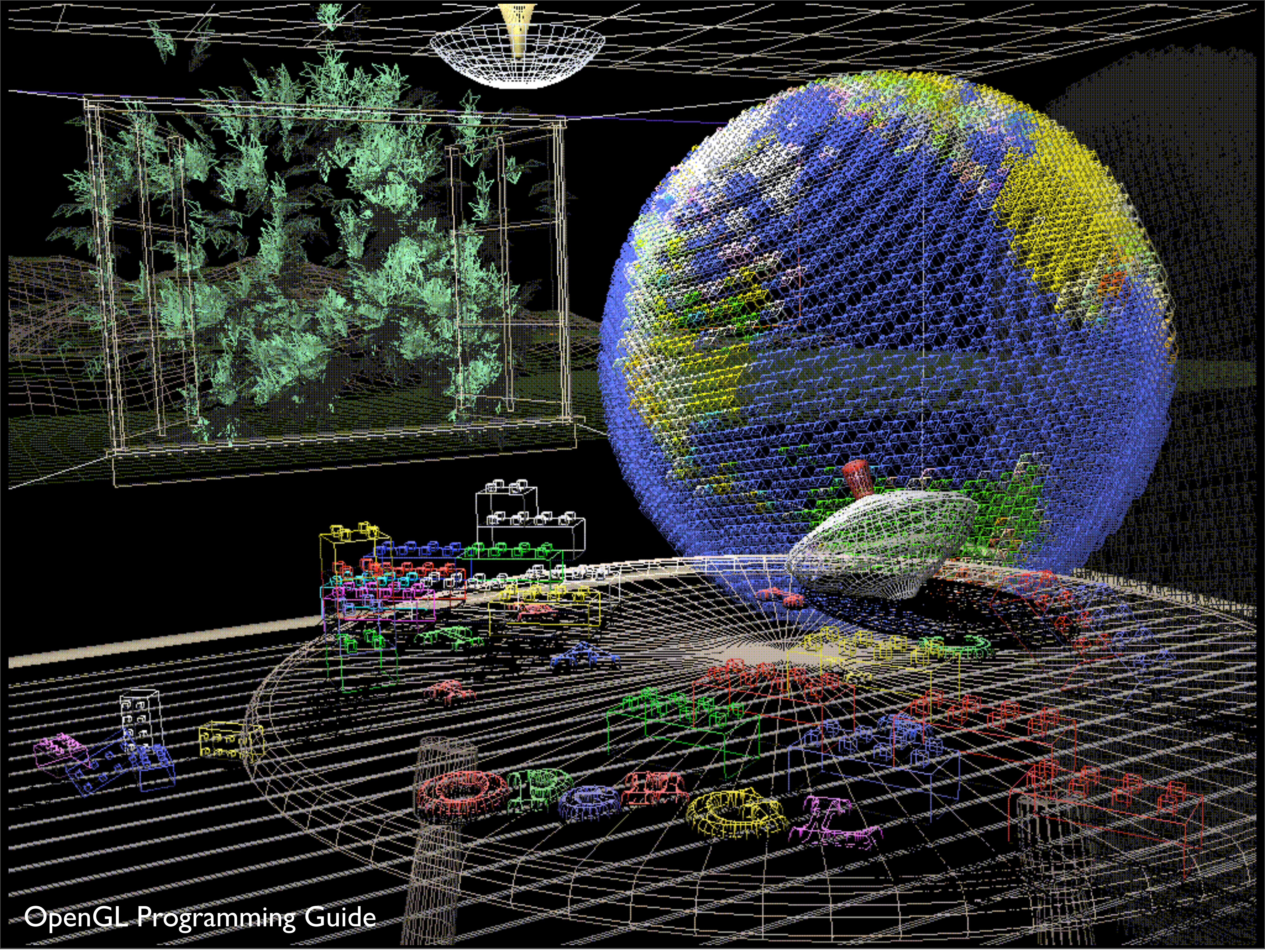
OpenGL: Conceptual Model



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**

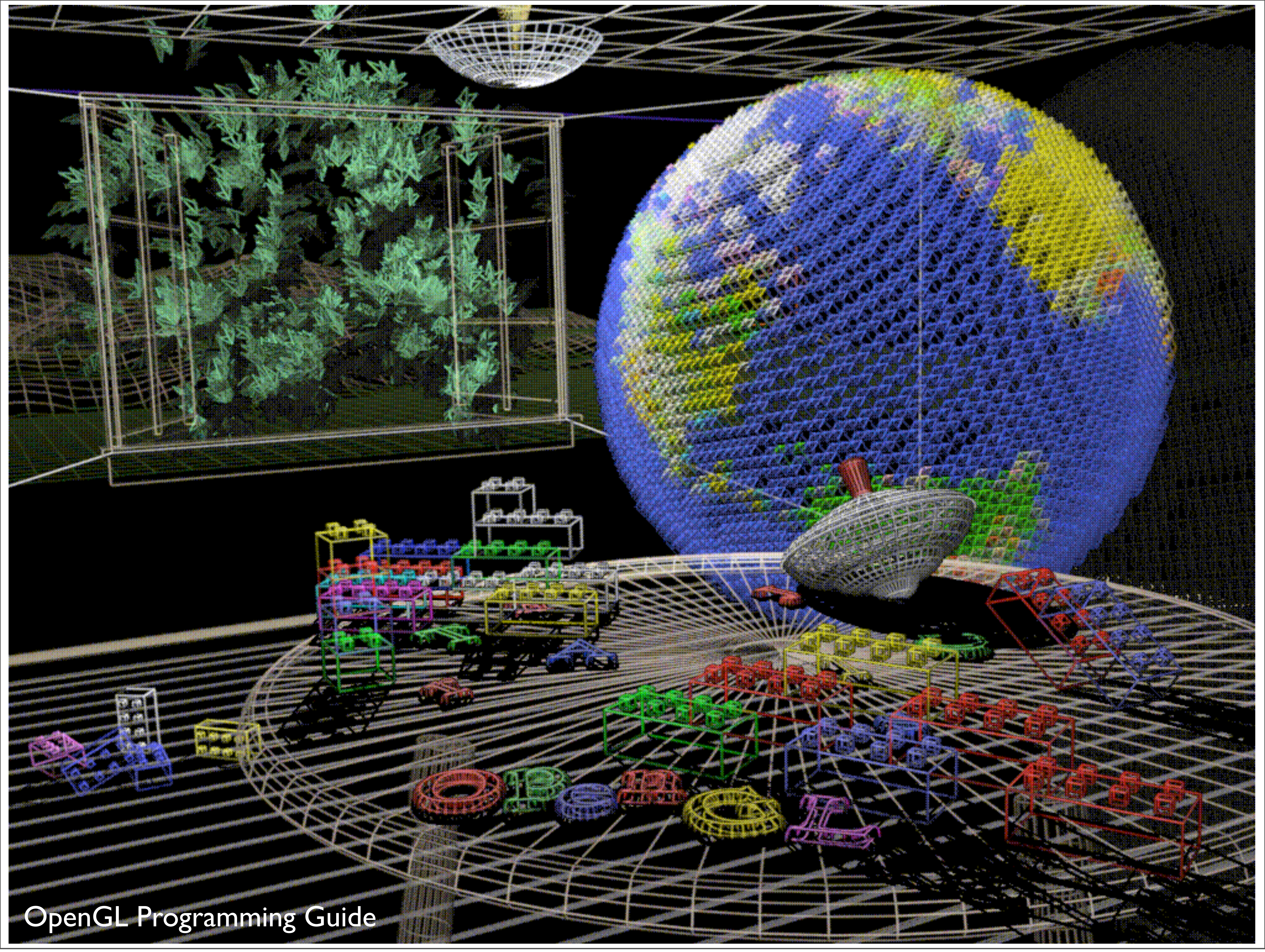


Plate 3. The same scene with **antialiased lines** that **smooth the jagged edge**s. See Chapter 7 .

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”

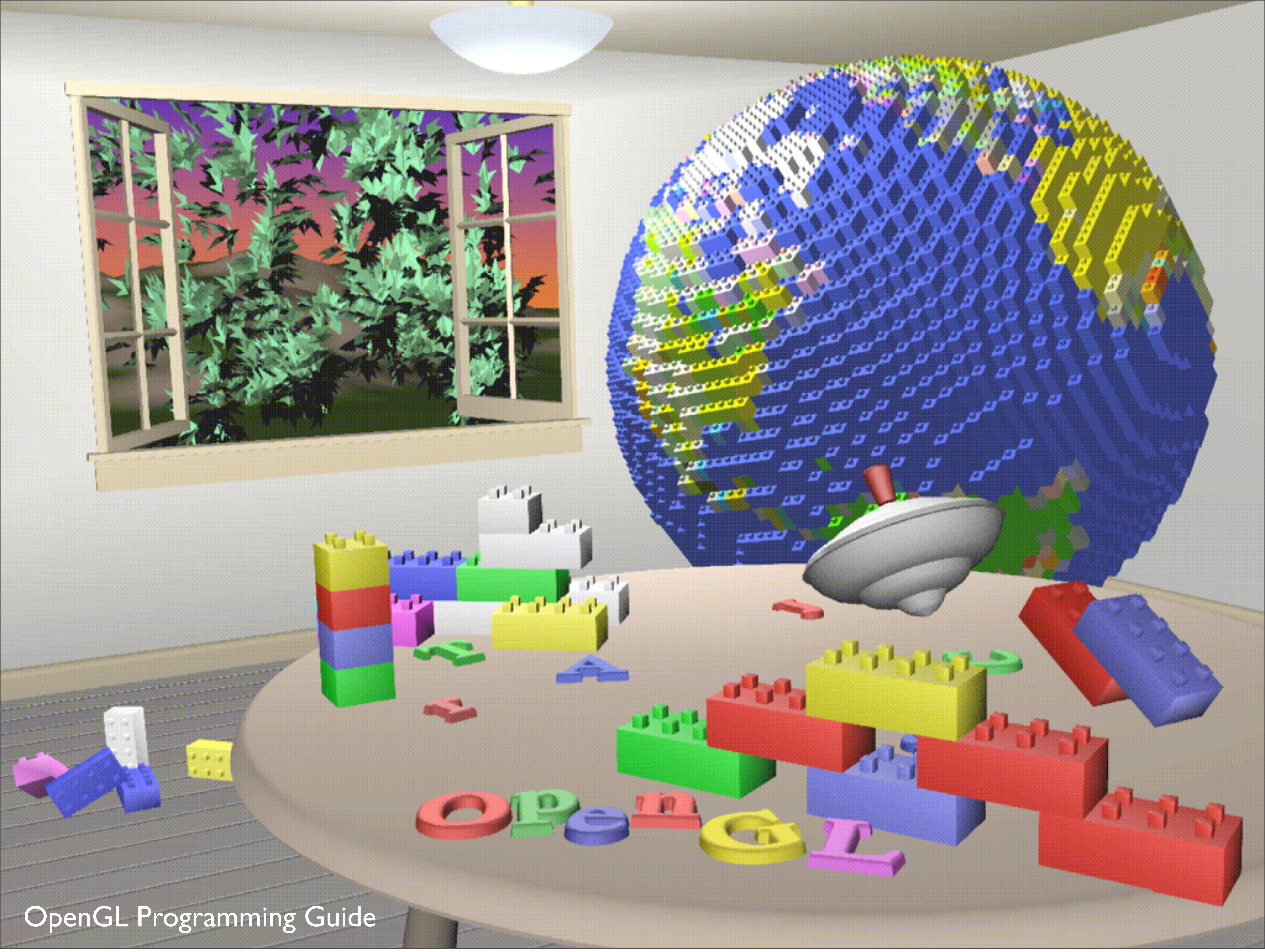


Plate 5. The scene rendered with **lighting** and **smooth-shaded polygons.** See Chapter 5 and Chapter 6 .

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 .

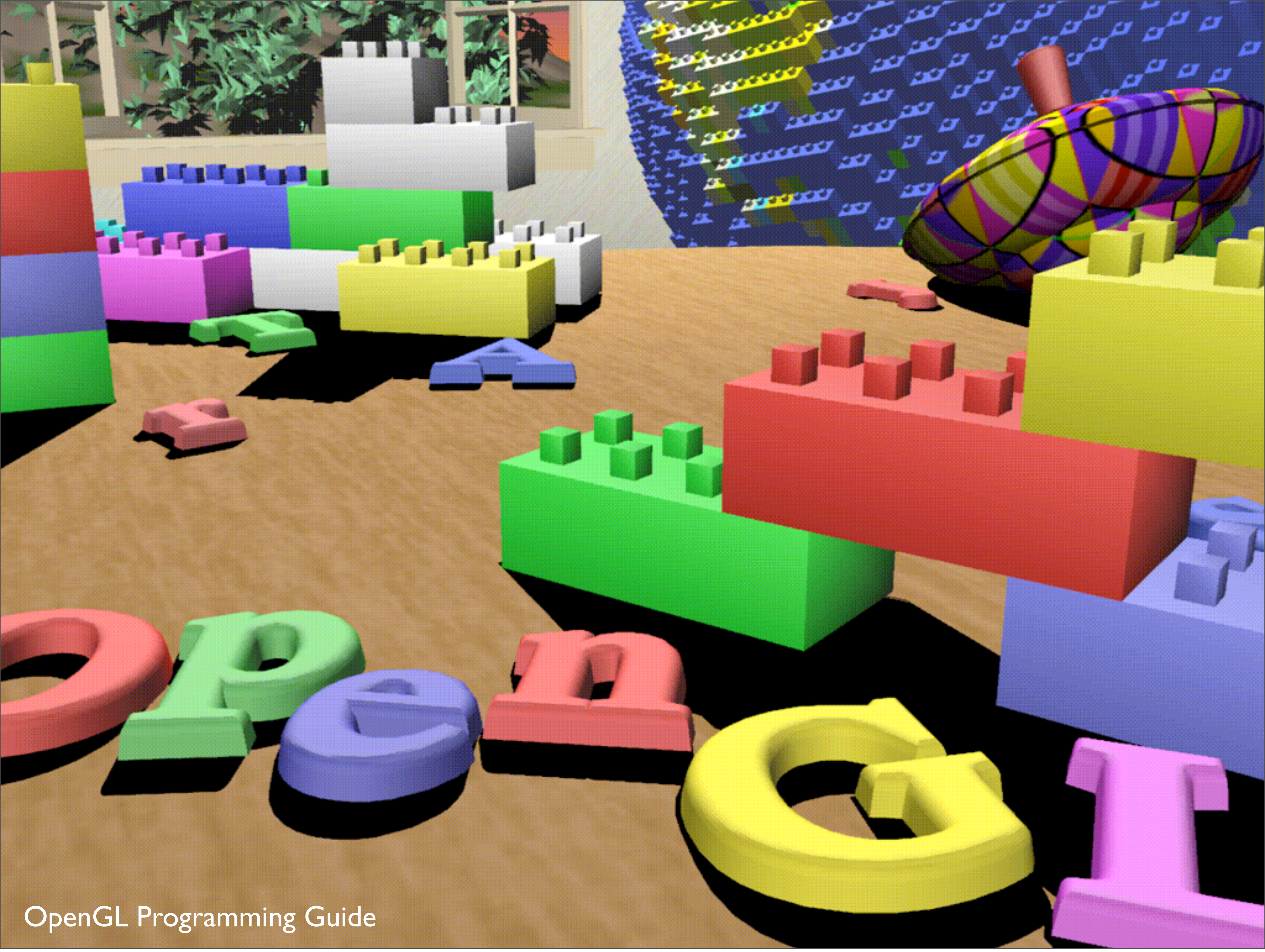


Plate 8. A close-up shot - the scene is **rendered from a new viewpoint.** See Chapter 3 .

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

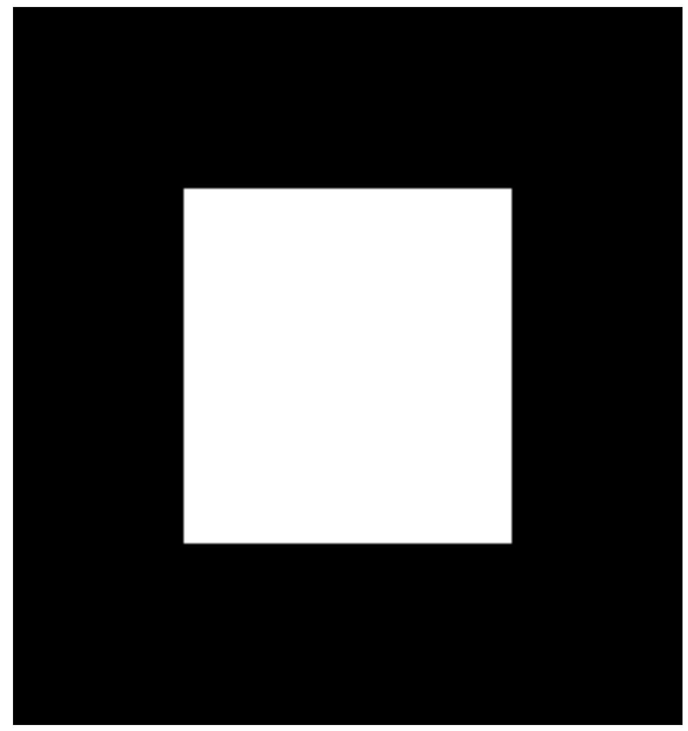
```
#include <whateverYouNeed.h>

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.

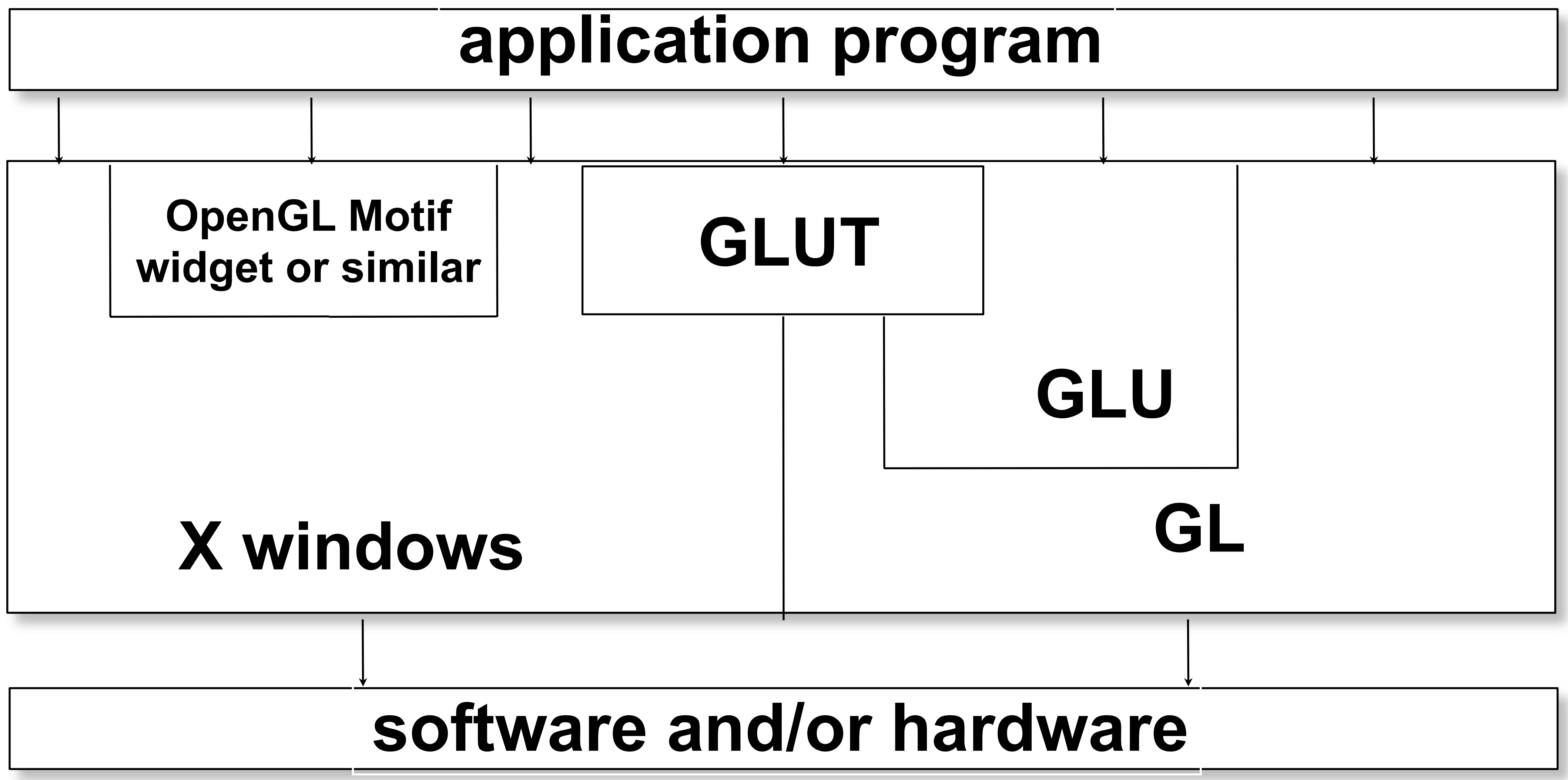
- blue are placeholders for windowing system commands
- clear color, actual clear
- Ortho – the coordinate system
- flush executes the commands

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

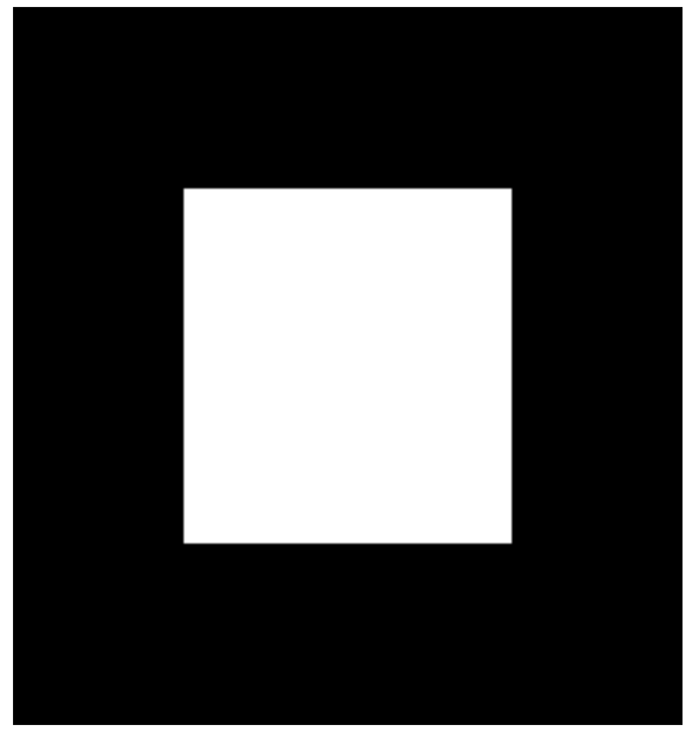
```
   #include <whateverYouNeed.h>

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.

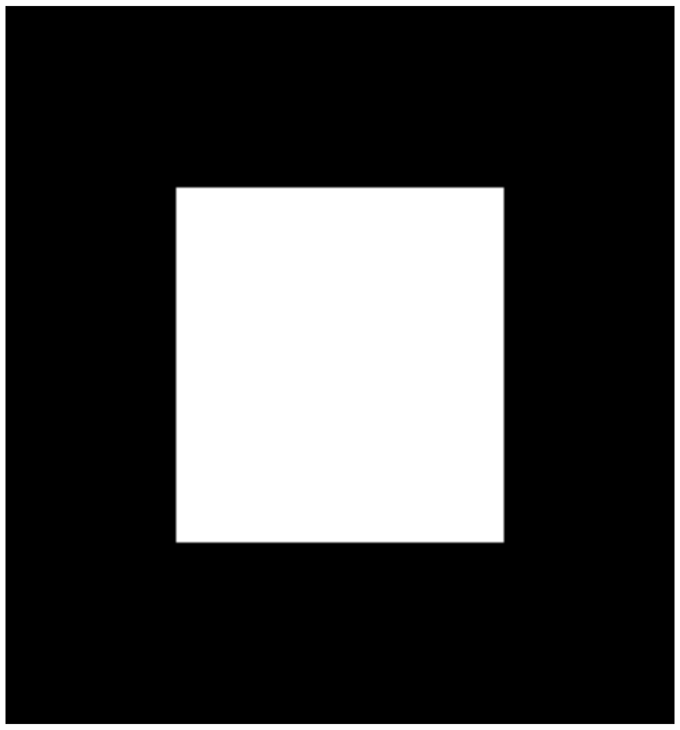
- blue are placeholders for windowing system commands
-can replace blue code with calls to **glut**

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>