CS230 : Computer Graphics
Winter 2012

Tamar Shinar
Computer Science & Engineering
UC Riverside
Welcome to CS230!

Talton et al., 2011

Schröder, 2000

LLNL

ILM

Hong et al. 2007

Pixar

Henrik Wann Jensen
Today’s agenda

• Course Logistics
• Introduction: graphics areas and applications
• Course schedule
• Introduction to the graphics pipeline and OpenGL
• Math review
Course Logistics

- Instructor: Tamar Shinar
- No TA
- Website: http://www.cs.ucr.edu/~shinar/courses/cs230
- Lectures: MW, 3:40-5pm
- Office hours: after class, and Wed 1-2pm, WCH 419
- announcements (assignments, etc.) made in class and on course website
Course Logistics

- Grading
  - 10% class participation
  - 60% assignments (3 assignments, each ~2 weeks)
  - 30% final project
  - No exams

- Total of 3 late days (72 hours) for the quarter for the assignments only

- final must be submitted on time

- assignments individual; project individual or group of 2

- In-class participation
  - 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

Computer Graphics with OpenGL
Hearn Baker, Carithers

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, art

• Ph.D., 2008, Stanford University on simulation methods for computer graphics

• Started at UCR in the fall

• Work in graphics simulation and biological simulation

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

Talton et al., 2011

Fig. 1: 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
- `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*
Other areas...

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

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

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

tentative; see course website for up-to-date schedule

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Date</th>
<th>Topic</th>
<th>Reading</th>
<th>Assigned</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jan 9</td>
<td>Introduction</td>
<td>HBC, Ch. 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Jan 11</td>
<td>Modeling, graphics primitives</td>
<td>HBC, Ch. 4, 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jan 16</td>
<td><strong>HOLIDAY</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Jan 18</td>
<td>Rasterization</td>
<td>HBC, Ch. 6</td>
<td>Assignment 1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Jan 23</td>
<td>3D transforms and viewing</td>
<td>HBC, Ch. 9, 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Jan 25</td>
<td>3D Modeling</td>
<td>HBC, Ch. 13, 14, 15</td>
<td>Assignment 2</td>
<td>Assignment 1</td>
</tr>
<tr>
<td>6</td>
<td>Jan 30</td>
<td>3D Modeling, cont.</td>
<td>HBC, Ch. 13, 14, 15</td>
<td>Assignment 2</td>
<td>Assignment 1</td>
</tr>
<tr>
<td>7</td>
<td>Feb 1</td>
<td>Visible surface detection</td>
<td>HBC, Ch. 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Feb 6</td>
<td>Lighting and surface shading</td>
<td>HBC, Ch. 17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Feb 8</td>
<td>Texture mapping</td>
<td>HBC, Ch. 18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Feb 13</td>
<td>Global Illumination</td>
<td>HBC, Ch. 21</td>
<td>Assignment 3</td>
<td>Assignment 2</td>
</tr>
<tr>
<td></td>
<td>Feb 15</td>
<td>Programmable Shaders</td>
<td>HBC, Ch. 22</td>
<td>Assignment 3</td>
<td>Assignment 2</td>
</tr>
<tr>
<td></td>
<td>Feb 20</td>
<td><strong>HOLIDAY</strong></td>
<td></td>
<td>Project Proposal</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Feb 22</td>
<td>Animation</td>
<td>HBC, Ch. 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Feb 27</td>
<td>Graphics hardware</td>
<td></td>
<td>Assignment 3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Feb 29</td>
<td>Particle systems</td>
<td></td>
<td>Assignment 3</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Mar 5</td>
<td>Rigid body simulation</td>
<td></td>
<td>Assignment 3</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Mar 7</td>
<td>Deformable body simulation</td>
<td></td>
<td>Assignment 3</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Mar 12</td>
<td>Character animation</td>
<td></td>
<td>Assignment 3</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Mar 14</td>
<td>Fluid simulation</td>
<td></td>
<td>Assignment 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Finals week (TBA)</td>
<td>Project presentations</td>
<td></td>
<td>Final Project</td>
<td></td>
</tr>
</tbody>
</table>
CS230
Intro to OPEN GL
(with some slides courtesy of V. Zordan)
OpenGL: Conceptual Model

Real Object

Real Light

Human Eye
OpenGL: Conceptual Model

Real Light

Real Object

Human Eye

Synthetic Light Source

Synthetic Camera

Synthetic Model

Graphics System

Display Device

Human Eye

Real Object
Introduction to OpenGL

- **Open Graphics Library**, managed by Khronos Group
- A software interface to graphics hardware
- 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](http://www.opengl.org/wiki/Main_Page)

- used to produce interactive 3D graphics
- sits between programmer and 3D accelerators and 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
What can OpenGL do?
Examples from the
OpenGL Programming Guide (“red book”)
- **Wireframe** models
  - shows each object up of polygons
- the *lines are are the edges* and the *faces of the polygons make up the object surface*
supports the atmospheric effect “fog”

Plate 2. The same scene using fog for depth-cueing (lines further from the eye are dimmer). See Chapter 7.
Plate 3. The same scene with **antialiased lines** that **smooth the jagged edges**. 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.
Plate 9. The scene drawn using atmospheric effects (fog) to simulate a smoke-filled room. See Chapter 7.
OpenGL state machine

- 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 command syntax

- commands: `glClearColor();`
- `glVertex3f()`
- 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
Raster images

Hearn, Baker, Carithers

virtually all graphics systems are raster based, meaning the image we see is a raster of pixels
Here a raster scan device display an image as a set of discrete points across each scanline
Modern graphics system

Interactive Computer Graphics, Angel and Shreiner

the pixels are stored in a location in memory call the **frame buffer**

frame buffer resolution determines the details in the image
  - e.g., 24 bit color “full color”
  - high dynamic range or HDR use 12 or more bits for each color

frame buffer = color buffers + other buffer
Z-buffer Rendering

- Z-buffering is a very common approach, also often accelerated with hardware.
- OpenGL is based on this approach.
Pipelining operations

An arithmetic pipeline that computes \(c + (a \times b)\)

By pipelining the arithmetic operation, the **throughput**, or rate at which data flows through the system, has been **doubled**.

If the pipeline had more boxes, the **latency**, or time it takes one datum to pass through the system, would be higher.

**Throughput and latency must be balanced.**
3D graphics pipeline

**Vertices** → **Vertex processor** → **Clipper and primitive assembler** → **Rasterizer** → **Fragment processor** → **Pixels**

**Geometry**: objects – made of primitives – made of vertices

**Vertex processing**: coordinate transformations and color

**Clipping and primitive assembly**: output is a set of primitives

**Rasterization**: output is a set of fragments for each primitive

**Fragment processing**: update pixels in the frame buffer

---

1. **Geometry**: objects – made of primitives – made of vertices
2. **Vertex processing**: coordinate transformations and color
3. **Clipping and primitive assembly**: use clipping volume. must be primitive by primitive rather than vertex by vertex. therefore vertices must be assembled into primitives before clipping can take place. Output is a set of primitives.
4. **Rasterization**: primitives are still in terms of vertices -- must be converted to pixels. E.g., for a triangle specified by 3 vertices, the rasterizer must figure out which pixels in the frame buffer fill the triangle. Output is a set of **fragments for each primitive**. A fragment is like a **potential pixel**. Fragments can carry depth information used to figure out if they lie behind other fragments for a given pixel.
5. **Fragment processing**: update pixels in the frame buffer. some fragments may not be visible. texture mapping and bump mapping. blending.

**Other rendering approaches include**
Other rendering approaches

- Ray tracing, radiosity, and photon mapping
- more physical but don’t achieve real-time performance
3D graphics pipeline

- optimized for drawing 3D triangles with shared vertices
- map 3D vertex locations to 2D screen locations
- shade triangles and draw them in back to front order using a z-buffer
- speed depends on # of triangles
- most operations on vertices can be represented using a 4D coordinate space - 3D position + homogeneous coordinate for perspective viewing
- 4x4 matrices and 4-vectors

- use varying level of detail - fewer triangles for distant objects
1. construct shapes from primitives - points, lines, polygons, images, bitmaps, (mathematical descriptions of objects) - specify the model
- Assembly-line approach to processing data
- Vertex data: data that is or will be converted to vertices: points, lines, polygons, surfaces
- Pixel data: images, bitmaps, pixels
- vertex data and pixel data follow different paths, but both eventually undergo rasterization and per-fragment operations
- Display lists:
  - data can be save in display lists or processed in immediate mode
  - from display list it can be processed same as in immediate mode
- Evaluators:
  - all geometric data eventually reduced to vertices (evaluators for parametrized curves and surfaces)
- Per-vertex operations: converts the vertices into primitives
  - 3D spatial coordinate transformed to screen coordinate + depth value
  - texture coordinates generated and transformed here
  - lighting calculations using material properties, light information, normals,
  - in newer versions of OpenGL, using a “vertex shader” is mandatory
    - GLSL
  - this replaces all the per-vertex operations
- Primitive assembly:
  - clipping is a major part
  - eliminate parts of geometry outside a half-space
  - remove points or for lines and polygons it can add points
  - results: complete geometric primitives: transformed and clipped vertices with color, depth, and texture-coordinate values
- Pixel operations:
  - pixels take a different route through the OpenGL rendering pipeline
- Texture assembly:
  - apply textures to objects (Ch. 9)
- Rasterization:
open up file to look at this more closely
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 Library - 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

Diagram showing the organization of software and hardware components:

- Application program
  - OpenGL Motif widget or similar
  - GLUT
  - GLU
  - GL
  - X windows
  - Software and/or hardware
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

```c
#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**
END