Interactive Graphics
CSCI B481 – Spring 2017

Instructor:
Mitja Hmeljak,
http://pages.iu.edu/~mitja
mitja@indiana.edu
Interactive Graphics
CSCI B481 Spring 2017

Instructor:
Mitja Hmeljak, mitja@indiana.edu

Lectures: 11:15AM-12:30PM on Tuesday and Thursday
in I2 130 (School of Informatics East, 919 E. 10th St)
Office hours: Tuesday and Thursday 2:30PM-3:30PM
(office: LH 425)

Assistant Instructors:
Leif Christiansen, leifchri@iu.edu – OpenGL ES, WebGL
Steve Babcock, wsbabcoc@iu.edu – Android, iOS

Labs: 11:15AM-12:30PM on Friday in LH 030 (Lindley Hall)
Office hours: Leif – Wed 8:30-9:30AM, LH 030
Steve – Mon 1:00-2:00PM, LH 201
Objectives

Broad introduction to Computer Graphics
  Software
  Hardware
  Applications

Shader-Based OpenGL:
  → → OpenGL ES ↔ ↔

( we'll use **OpenGL ES 2.0**, which is compatible with "desktop" OpenGL 3.1 (and later) and "browser" WebGL 1.0 (and later) )
Prerequisites

*Good* **programming skills**
(in language(s) such as C & C-descendants, Python, ...)

*Basic* **Data Structures**, e.g.:
  - Linked lists
  - Arrays

**Geometry**
Simple **Linear Algebra**
(vector and matrix operations)
Assignments

Programming Assignments
Simple Operations
(sometimes demanding to implement!)
Interactive
2D and 3D

Written (non-programming) Homeworks

B481 syllabus on main page at:
https://www.cs.indiana.edu/classes/b481/2017/
References

Textbooks

Required:

by A. Munshi, D. Ginsburg and D. Shreiner
Addison-Wesley Professional 2009

Required:

Required: "Interactive Computer Graphics" **6th (or 7th) Edition**
by Edward Angel and Dave Shreiner - Pearson
[https://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/](https://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/)
Additional Useful (optional) References

http://www.khronos.org/
standard-defining consortium: OpenGL &c.
official documentation and specifications
Khronos OpenGL ES Registry - https://www.khronos.org/registry/gles/#specs32

OpenGL ES Programming Guide for iOS

http://www.opengl.org
used to be the official repository for standards documents
links to man pages, OpenGL wiki, etc.

a little dated: OpenGL Shading Language, 3rd Edition

also of interest: The OpenGL Programmer’s Guide (the Redbook) 7th Edition
The definitive reference for desktop OpenGL
From OpenGL to OpenGL ES

Shader-based

Many computer graphics courses use OpenGL but still use fixed-function pipeline...
...which does not require shaders

Does not make use of the full capabilities of the graphics processing unit (GPU)

OpenGL ES

OpenGL "for Embedded Systems"

runs on the vast majority of mobile devices

makes use of local hardware
# About You (index card)

<table>
<thead>
<tr>
<th>B481</th>
<th>First Name, Last Name</th>
<th>2017-01-09</th>
</tr>
</thead>
</table>

1. Name, Major, Class Year  
2. Why are you taking this course?  
3. Do you have any previous computer graphics programming experience? In what APIs and programming language(s)? When did you take any previous graphics classes?  
4. Do you have any previous experience with *native* mobile programming? If so, was it for Android (Java), iOS (Objective C, Swift), other?  
5. What would you like to be able to do with computer graphics?  
6. What are your concerns about this course (if any) ?  
7. Do you have access to a laptop, and would you be willing to bring it to *lectures and labs*? (if yes, please state operating system type and version version)  
8. Do you have access to a smartphone device running iOS or Android? (if yes, please state operating system type and version version)
What is Computer Graphics?

Computer graphics simulates the physics of the interaction of light with matter.

The computational problem in computer graphics is to carry out that simulation to a level of accuracy consistent with the available resources and the requirements of the problem.
Computer Graphics

Computer graphics deals with all aspects of creating images with a computer

Hardware
Software
Applications
Example

Where could this image be coming from?

What hardware / software do we need to produce it?
Preliminary Answer

**Application:** The object is an artist’s rendition of the sun for an animation to be shown in a domed environment (planetarium)

**Software:** Maya, Blender, etc. for modeling and rendering but those are built on top of OpenGL

**Hardware:** PC/Mac/mobile device with sufficient graphics capabilities for modeling and rendering
Basic Graphics System

Input devices

Processor (CPU) → Graphics processor → Frame buffer → Output device

Image formed in frame buffer

CPU Memory

GPU Memory
what **we used to have** as output: a CRT

Can be used either as a line-drawing device (calligraphic) or to display contents of frame buffer (raster mode)

Computer graphics goes back to the earliest days of computing
Strip charts
Pen plotters
Simple displays using A/D converters to go from computer to calligraphic CRT
Cost of refresh for CRT too high
Computers slow, expensive, unreliable
Wireframe graphics
Draw only lines
Sketchpad
Display Processors
Storage tube

wireframe representation of object
Sketchpad

Ivan Sutherland’s PhD thesis at MIT

Recognized the potential of man-machine interaction

Loop

  Display something

  User moves light pen

  Computer generates new display

Sutherland also created many of the now common algorithms for computer graphics
Display Processor

Rather than have the host computer try to refresh display use a special purpose computer called a display processor (DPU)

Graphics stored in display list (display file) on display processor
Host *compiles* display list and sends to DPU
Direct View Storage Tube

Created by Tektronix

Did not require constant refresh

Standard interface to computers

Allowed for standard software

Plot3D in Fortran

Relatively inexpensive

Opened door to use of computer graphics for CAD community

Raster Graphics
Beginning of graphics standards
IFIPS
  GKS: European effort
    Becomes ISO 2D standard
Core: North American effort
  3D but fails to become ISO standard

Workstations and PCs
Raster Graphics

Image produced as an array (the raster) of picture elements (pixels) in the frame buffer
Allows us to go from lines and wire frame images to filled polygons
PCS and Workstations

Although we no longer make the distinction between workstations and PCs, historically they evolved from different roots

Early workstations characterized by
- Networked connection: client-server model
- High-level of interactivity

Early PCs included frame buffer as part of user memory
- Easy to change contents and create images
Realism comes to computer graphics

smooth shading  environment mapping  bump mapping

Special purpose hardware
   Silicon Graphics geometry engine
      VLSI implementation of graphics pipeline

Industry-based standards
   PHIGS
   RenderMan

Networked graphics: X Window System

Human-Computer Interface (HCI)

OpenGL API

Completely computer-generated feature-length movies (Toy Story) are successful

New hardware capabilities

Texture mapping

Blending

Accumulation, stencil buffers
Photorealism

Graphics cards for PCs dominate market
Nvidia, ATI

Game boxes and game players determine direction of market

Computer graphics routine in movie industry: Maya, Lightwave

Programmable pipelines
Computer Graphics and Image Formation

Computer graphics systems focus on "simulating the physics of the interaction of light with matter".

The computational problem that Computer Graphics aims to solve is to "carry out that simulation to a level of accuracy consistent with the available resources and the requirements of the problem".

We also refer to this simulation as "Image Formation".
Elements of Image Formation

Objects
Viewer
Light source(s)

Attributes that govern how light interacts with the materials in the scene
Note the independence of the objects, the viewer, and the light source(s)
Light

Light is the part of the electromagnetic spectrum that causes a reaction in our visual systems. Generally these are wavelengths in the range of about 350-750 nm (nanometers). Long wavelengths appear as reds and short wavelengths as blues.
Ray Tracing and Geometric Optics

One way to form an image is to follow rays of light from a point source finding which rays enter the lens of the camera. However, each ray of light may have multiple interactions with objects before being absorbed or going to infinity.
Luminance and Color Images

Luminance Image
- Monochromatic
- Values are gray levels
- Analogous to working with black and white film or television

Color Image
- Has perceptual attributes of hue, saturation, and lightness
- Do we have to match every frequency in visible spectrum? No!
Three-Color Theory

Human visual system has two types of sensors

Rods: monochromatic, night vision

Cones

  Color sensitive
  Three types of cones
  Only three values (the *tristimulus* values) are sent to the brain

Need only match these three values

Need only three *primary* colors
Old-style: Shadow Mask CRT
Additive and Subtractive Color

Additive color
Form a color by adding amounts of three primaries
CRTs, projection systems, positive film
Primaries are Red (R), Green (G), Blue (B)

Subtractive color
Form a color by filtering white light with cyan (C),
Magenta (M), and Yellow (Y) filters
Light-material interactions
Printing
Negative film
Pinhole Camera

Use trigonometry to find projection of point at \((x,y,z)\)

\[
\begin{align*}
    x_p &= -\frac{x}{z/d} \\
    y_p &= -\frac{y}{z/d} \\
    z_p &= d
\end{align*}
\]

These are equations of simple perspective
Models and Architectures

Learn the basic design of a graphics system
Introduce pipeline architecture
Examine software components for an interactive graphics system
So... how do we go about Image Formation?

Can we mimic the synthetic camera model to design graphics hardware software?

Application Programmer Interface (API)
Need only specify
  Objects
  Materials
  Viewer
  Lights

But how is the API implemented?
Physical Approaches

**Ray tracing**: follow rays of light from center of projection until they either are absorbed by objects or go off to infinity
- Can handle global effects
  - Multiple reflections
  - Translucent objects
- Slow
- Must have whole data base available at all times

**Radiosity**: Energy based approach
- Very slow
Practical Approach

Process objects one at a time in the order they are generated by the application
Can consider only local lighting

Pipeline architecture

All steps can be implemented in hardware on the graphics card
Vertex Processing

Much of the work in the pipeline is in converting object representations from one coordinate system to another:
- Object coordinates
- Camera (eye) coordinates
- Screen coordinates

Every change of coordinates is equivalent to a matrix transformation.

Vertex processor also computes vertex colors.
Projection

*Projection* is the process that combines the 3D viewer with the 3D objects to produce the 2D image.

Perspective projections: all projectors meet at the center of projection.

Parallel projection: projectors are parallel, center of projection is replaced by a direction of projection.
Primitive Assembly

Vertices must be collected into geometric objects before clipping and rasterization can take place.

- Line segments
- Polygons
- Curves and surfaces
Clipping

Just as a real camera cannot “see” the whole world, the virtual camera can only see part of the world or object space.

Objects that are not within this volume are said to be clipped out of the scene.
If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors.

Rasterizer produces a set of fragments for each object.

Fragments are “potential pixels”

- Have a location in frame buffer
- Color and depth attributes

Vertex attributes are interpolated over objects by the rasterizer.
Fragment Processing

Fragments are processed to determine the color of the corresponding pixel in the frame buffer. Colors can be determined by texture mapping or interpolation of vertex colors. Fragments may be blocked by other fragments closer to the camera. Hidden-surface removal.
The Programmer’s Interface

Programmer sees the graphics system through a software interface: the Application Programmer Interface (API)
API Contents

Functions that specify what we need to form an image
  Objects
  Viewer
  Light Source(s)
  Materials

Other information
  Input from devices such as mouse and keyboard
  Capabilities of system
Object Specification

Most APIs support a limited set of primitives including:

- Points (0D object)
- Line segments (1D objects)
- Polygons (2D objects)
- Some curves and surfaces
  - Quadrics
  - Parametric polynomials

All are defined through locations in space or vertices
Example (old style)

```c
glBegin(GL_POLYGON)
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(0.0, 1.0, 0.0);
  glVertex3f(0.0, 0.0, 1.0);
glEnd();
```

- **type of object**
- **location of vertex**
- **end of object definition**
Example (GPU based)

Put geometric data in an array

```javascript
var points = [
    vec3(0.0, 0.0, 0.0),
    vec3(0.0, 1.0, 0.0),
    vec3(0.0, 0.0, 1.0),
];
```

Send array to GPU

Tell GPU to render as triangle
Camera Specification

Six degrees of freedom
  Position of center of lens
  Orientation
  Lens
  Film size
  Orientation of film plane
Lights and Materials

Types of lights
  Point sources vs distributed sources
  Spot lights
  Near and far sources

Color properties

Material properties
  Absorption: color properties

Scattering
  Diffuse
  Specular
Early History of APIs

IFIPS (1973) formed two committees to come up with a standard graphics API

Graphical Kernel System (GKS)
  2D but contained good workstation model

Core
  Both 2D and 3D

GKS adopted as ISO and later ANSI standard (1980s)

GKS not easily extended to 3D (GKS-3D)

Far behind hardware development
PHIGS and X

Programmers Hierarchical Graphics System (PHIGS)
Arose from CAD community
Database model with retained graphics (structures)

X Window System
DEC/MIT effort
Client-server architecture with graphics
PEX combined the two
Not easy to use (all the defects of each)
SGI and GL

Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)

To access 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 enough to the hardware to get excellent performance
Focus on rendering
Omitted windowing and input to avoid window system dependencies
Reading Assignment for Week 1:

- Edward Angel and Dave Shreiner, Chapter 1 sections 1.2, 1.3, 1.4