What is Computer Graphics (CG)?

- Computer graphics: algorithms, mathematics, data structures ..... that **computer uses to generate PRETTY PICTURES**
- Techniques (e.g. draw a line, polygon) evolved over years
- Built into programmable libraries

Computer-Generated!
Not a picture!
Photorealistic Vs Real-Time Graphics

- **Photo-realistic**: E.g ray tracing slow: may take **days** to render

- **Real Time graphics**: **Milliseconds** to render (30 FPS)
  But lower image quality

Not this Class

This Class
Uses of Computer Graphics

- **Entertainment:** games

*Courtesy: Final Fantasy XIV*

*Courtesy: Super Mario Galaxy 2*
Uses of Computer Graphics

- movies, TV, books, magazines

*Courtesy: Shrek*

*Courtesy: Spiderman*
Uses of Computer Graphics

- Image processing:
  - alter images, remove noise, super-impose images

Original Image

Sobel Filter
Uses of Computer Graphics

- Process monitoring:
  - Layout of large systems or plants

Courtesy: Dataviews.de
Uses of Computer Graphics

- Display simulations:
  - flight simulators, virtual worlds

*Courtesy: Evans and Sutherland*
Uses of Computer Graphics

- Computer-aided design:
  - architecture, electric circuit design

Courtesy: cadalog.com
Uses of Computer Graphics

- Displaying Mathematical Functions
  - E.g., Mathematica®
Uses of Computer Graphics

- Scientific analysis and visualization:
  - molecular biology, weather, matlab, Mandelbrot set

 Courtesy:
Human Brain Project,
Denmark
2D Vs. 3D

- **2-Dimensional**
  - Flat
  - Only (x,y) color values on screen
  - Objects no notion of distance from viewer

- **3-Dimensional**
  - (x,y,z) values on screen
  - Objects have distances from viewer

- This class covers both 2D & 3D!
- Also interaction: Clicking, dragging
About This Course

- Computer Graphics has many aspects
  - **Computer Scientists** create graphics tools (e.g. Maya, photoshop)
  - **Artists** use CG tools/packages to create pretty pictures
  - Most hobbyists follow artist path. Not much math!
- **This Course: Computer Graphics for computer scientists!!!**
- Teaches concepts, uses OpenGL as concrete example
- Course is **NOT**
  - just about programming OpenGL
  - a comprehensive course in OpenGL. (Only parts of OpenGL covered)
  - about using packages like Maya, Photoshop
About This Course

- Class is concerned with:
  - How to build graphics tools
  - Underlying mathematics
  - Underlying data structures
  - Underlying algorithms

- This course is a lot of work. Requires:
  - Lots of coding in C/C++
  - Much more emphasis on shader programming than in past offerings
  - Lots of math, linear algebra, matrices

- We shall combine:
  - **Programmer’s view**: Program OpenGL
  - **Under the hood**: Learn OpenGL internals (graphics algorithms, math, implementation)
Syllabus Summary

- 2 Exams (50%), 4 Projects (50%)
- Projects:
  - Develop OpenGL/GLSL code on any platform, must port to Zoolab machine
  - May discuss projects, turn in individual projects
- Class website: http://web.cs.wpi.edu/~emmanuel/courses/cs4731/C13/
- Text:
- Cheating: Immediate ‘F’ in the course
- Advice:
  - Come to class
  - Read the text
  - Understand concepts before coding
Elements of 2D Graphics

- Polylines
- Text
- Filled regions
- Raster images (pictures)
Elements of 2D Graphics

- **Polyline**: connected sequence of straight lines
- Straight lines connect **vertices** (corners)
Polyline Attributes

- Color
- Thickness
- Stippling of edges (dash pattern)
**Text**

- Devices have:
  - text mode
  - graphics mode.

- **Graphics mode**: Text is drawn

- **Text mode**: Text not drawn uses character generator

- **Text attributes**: Font, color, size, spacing, and orientation

- Big Text
- Little Text
- Shadow Text
- Distorted Text
- Rotated Text
- Outlined text
- SMALLCAPS
Filled Regions

- **Filled region**: shape filled with some color or pattern
- Example: polygons

![Filled regions examples](image)
Raster Images

- Raster image (picture) is made up of many small cells (pixels, for “picture elements”), in different colors or grayscale.

(Right: magnified image showing pixels.)
Computer Graphics Tools

- **Require** hardware and software tools
- Hardware tools
  - **Output devices**: Video monitors, printers
  - **Input devices**: Mouse/trackball, pen/drawing tablet, keyboard
  - Graphics cards/accelerators (GPUs)
- Software tools (low level)
  - Operating system
  - Editor
  - Compiler
  - Debugger
  - Graphics Library (OpenGL)
Graphics Processing Unit (GPU)

- OpenGL implemented in hardware => FAST!!
- **Programmable**: in last 10 years (now as shaders)
- Located either on PC motherboard (Intel) or Separate graphics card (Nvidia or ATI)

On PC motherboard | On separate PCI express card
Computer Graphics Libraries

- Functions to draw line, circle, image, etc
- Previously device-dependent
  - Different OS => different graphics library
  - Tedious! Difficult to port (e.g. move program Windows to Linux)
  - Error Prone
- Now device-independent libraries
  - **APIs**: OpenGL, DirectX
  - Working OpenGL program easily moved from Windows to Linux, etc
OpenGL Basics

- OpenGL’s function – Rendering (or drawing)
- Rendering? – Convert geometric/mathematical object descriptions into images
- OpenGL can render:
  - 2D and 3D
  - Geometric primitives (lines, dots, etc)
  - Bitmap images (pictures, .bmp, .jpg, etc)
- OpenGL does NOT manage drawing window
GL Utility Toolkit (GLUT)

- OpenGL
  - Window system independent
  - Concerned only with drawing (2D, 3D, images, etc)
  - No window management (create, resize, etc), very portable

- GLUT:
  - Minimal window management
  - Interfaces with different windowing systems
  - Easy porting between windowing systems. Fast prototyping
GL Utility Toolkit (GLUT)

- No bells and whistles
  - No sliders
  - No dialog boxes
  - No elaborate menus, etc

- To add bells and whistles, use system’s API or GLUI:
  - X window system
  - Apple: AGL
  - Microsoft: WGL, etc
OpenGL Basics

- Low-level graphics rendering API
- Maximal portability
  - Display device independent (Monitor type, etc)
  - Window system independent based (Windows, X, etc)
  - Operating system independent (Unix, Windows, etc)
- OpenGL programs behave same on different devices, OS
Simplified OpenGL Pipeline

- Vertices go in, sequence of steps (vertex processor, clipper, rasterizer, fragment processor) image rendered
OpenGL Programming Interface

- Programmer view of OpenGL?
  - Application Programmer Interface (API)
  - Writes OpenGL Application programs
Framebuffer

- Dedicated memory location:
  - Draw in framebuffer => shows up on screen
  - Located either on CPU (software) or GPU (hardware)
References

- Angel and Shreiner, Interactive Computer Graphics (6th edition), Chapter 1