What is Computer Graphics (CG)?

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

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

- **Photo-realistic**: E.g. ray tracing
  - Highest quality image possible
  - Slow: may take **days** to render

- **Real Time graphics**: E.g. game engine
  - **Milliseconds** to render (30 FPS)
  - Lower image quality

This Class

Not this Class
Uses of Computer Graphics: Entertainment

- **Entertainment**: games

  *Courtesy: Super Mario Galaxy 2*

  *Movies*

  *Courtesy: Spiderman*
Uses of Computer Graphics

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

*Original Image*  
*Sobel Filter*
Uses of Computer Graphics

Simulators

Display math functions
E.g. matlab

Courtesy: Evans and Sutherland
Uses of Computer Graphics

- Scientific analysis and visualization:

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

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

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

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

- Computer Graphics has many aspects
  - **Computer Scientists** create/program graphics tools (e.g. Maya, photoshop)
  - **Artists** use CG tools/packages to create pretty pictures
- Most hobbyists follow artist path. Not much math! E.g. use blender
About This Course

- 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 program computer graphics
  - Underlying mathematics, data structures, algorithms

- This course is a lot of work. Requires:
  - C/C++, shader programming
  - Lots of math, linear algebra, matrices

- We will combine:
  - **Programmer’s view**: Program OpenGL APIs
  - **Under the hood**: Learn OpenGL internals (graphics algorithms, math, implementation)
Course Text


- Buy 6*th* edition (pure OpenGL) ....... NOT 7*th* edition (WebGL)!!!

- Supplementary books available through the WPI library. How?
Syllabus Summary

- 3 Exams (50%), 5 Projects (50%)
- Projects:
  - Develop OpenGL/GLSL code on any platform, must port to Zoolab machine
  - May discuss projects but turn in individual projects
- Class website: http://web.cs.wpi.edu/~emmanuel/courses/cs543/f18/
- Cheating: Immediate ‘F’ in the course
  - Note: Using past projects on Internet, GitHub, bitBucket is cheating!
- Advice:
  - Come to class
  - Read textbook
  - Understand concepts before coding
Elements of 2D Graphics

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

- **Polyline**: vertices (corners) connected by straight lines
- **Attributes**: line thickness, color, etc
Text

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

- Devices have:
  - text mode
  - graphics mode.

- **Graphics mode:** Text is drawn

- **Text mode:** Text produced by character generator, not drawn
Filled Regions

- **Filled region**: shape filled with a color or pattern
- E.g: polygons

Polygons Filled with Color

Polygons Filled with Pattern
Raster Images

- Raster image (picture): 2D matrix of pixels (picture elements), in different colors or grayscale.

Grayscale Image

Color Image
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 cross-platform, device-independent libraries
  - **APIs:** OpenGL, DirectX
  - Working OpenGL program few changes to move from Windows to Linux, etc
Graphics Processing Unit (GPU)

- OpenGL implemented on GPU chip/hardware => FAST!!
- **Programmable**: as shaders
- GPU located either on
  - PC motherboard (Intel) or
  - Separate graphics card (Nvidia or ATI)
OpenGL Basics

- OpenGL’s function is Rendering (drawing)
- Rendering? – Convert geometric/mathematical object descriptions into images
- OpenGL can render (draw):
  - 2D and 3D
  - Geometric primitives (lines, dots, etc)
  - Bitmap images (pictures, .bmp, .jpg, etc)
GL Utility Toolkit (GLUT)

- OpenGL does NOT manage drawing window
- OpenGL
  - Window system independent
  - Concerned only with drawing (2D, 3D, images, etc)
  - No window management (create, resize, etc), very portable
- GLUT:
  - Minimal window management
  - Runs on different windowing systems (e.g. Windows, Linux)
  - Program that uses GLUT easily ported between windowing systems.
GL Utility Toolkit (GLUT)

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

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

- OpenGL programs behave same on different devices, OS
- Maximal portability
  - Display device independent (Monitor type, etc)
  - OS independent (Unix, Windows, etc)
  - Window system independent based (Windows, X, etc)
- E.g. If student writes OpenGL code on Apple Mac at home, it runs well on Zoolab Windows machines
OpenGL Programming Interface

- Programmer view of OpenGL
  - Application Programmer Interface (API)
  - Writes OpenGL application programs. E.g

```c
glDrawArrays(GL_LINE_LOOP, 0, N);
glFlush( );
```
Simplified OpenGL Pipeline

- Vertices input, sequence of rendering steps (vertex processor, clipper, rasterizer, fragment processor) image rendered

- **This class:** learn graphics rendering steps, algorithms, their order
To draw a shape, OpenGL colors a corresponding group of pixels (fragments) called **rasterization**
- E.g. yellow triangle converted to group of pixels to be colored yellow

- **Vertex shader** code manipulates vertices of shapes
- **Fragment shader** code manipulates pixels

[Diagram showing the process from vertices to fragments]
OpenGL Program?

- Usually has 3 files:
  - **.cpp file**: containing OpenGL code, main() function
    - Does initialization, generates/loads geometry to be drawn
  - **Vertex shader**: manipulates vertices (e.g. move vertices)
  - **Fragment shader**: manipulates pixels/fragments (e.g. change pixel/fragment color)
Framebuffer

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

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