DEFINITIONS
WHAT IS COMPUTER GRAPHICS?

- **Computer graphics** – creating and manipulating images using computers
  - Data used/manipulated can be:
    - 2D or 3D
    - Synthetic or real-world
AREAS OF COMPUTER GRAPHICS

- Roughly, there are 3 main areas of computer graphics
  - **Modeling**
    - Creating and specifying shape and appearance properties of an object to be rendered
    - Discusses how that model information is stored
  - **Rendering**
    - Creation of shaded images from 3D computer models
    - I.e., getting images from a virtual camera in a virtual 3D world
  - **Animation**
    - Techniques used to create the illusion of motion
    - Deals with change of the model over time
FOCUS OF THIS COURSE

• The focus of this course will be on the underlying mechanics and algorithms of computer graphics (so mostly rendering and animation)
  • We will NOT be discussing how to use tools to create graphical CONTENT (i.e., a computer art course)
  • However, we will discuss different ways to store model information
APPLICATIONS
SO, WHY DO WE NEED IT?

• Practically EVERY field/discipline/application needs to use computer graphics in some way
  • Science
  • Art
  • Engineering
  • Business
  • Industry
  • Medicine
  • Government
  • Entertainment
  • Advertising
  • Education
  • Training
  • ...and more!
APPLICATIONS: GRAPHS, CHARTS, AND DATA VISUALIZATION

• Graphics and charts
  • One of the earliest applications → plotting data using printer

• Data visualization
  • Sciences
    • Show visual representation → see patterns in data
      • E.g., the flow of fluid (LOx) around a tube is described using streamtubes
    • Challenges: large data sets, best way to display data

• Medicine
  • 2D images (CT, MRI scans) → 3D volume rendering
    • E.g., volume rendering and image display from the visible woman dataset

Images from VTK website: http://www.vtk.org/VTK/project/imagegallery.php
APPLICATIONS: CAD/CADD/CAM

• Design and Test
  • CAD = Computer-Aided Design
  • CADD = Computer-Aided Drafting and Design
  • Usually rendered in wireframe
• Manufacturing
  • CAM = Computer-Aided Manufacturing

• Used for:
  • Designing and simulating vehicles, aircraft, mechanical devices, electronic circuits/devices,…
  • Architecture and rendered building designs

• ASIDE: Often need special graphics cards to make absolutely SURE the rendered image is correct (e.g., NVIDIA Quadro vs. a garden-variety GeForce)

APPLICATIONS: COMPUTER ART / MOVIES

• First film to use a scene that was completely computer generated:
  • Tron (1982)
  • Star Trek II: Wrath of Khan (1982)
  • ...(depends on who you talk to)

• First completely computer-generated full-length film:
  • Toy Story (1995)

http://upload.wikimedia.org/wikipedia/en/1/17/Tron_poster.jpg
http://bigcosta.files.wordpress.com/2007/12/genesis1.jpg
http://www.standbyformindcontrol.com/wp-content/uploads/2013/05/khan-poster.jpg
http://s7d2.scene7.com/is/image/Fathead/15-15991X_dis_toy_story_prod?layer=comp&fit=constrain&hei=350&wid=350&fmt=png-alpha&qlt=75,0&op_sharpen=1&resMode=bicub&op_usm=0,0,0,0,0&iccEmbed=0
APPLICATIONS: VIRTUAL-REALITY ENVIRONMENTS / TRAINING SIMULATIONS

- Used for:
  - Training/Education
  - Military applications (e.g., flight simulators)
  - Entertainment
APPLICATIONS: GAMES!

Unreal 4 Engine

Crytek 3 Engine
APPLICATIONS: GAMES!

Unity Engine
(Wasteland 2)

Source Engine
(Portal 2)
APPLICATIONS: GAMES!

Wolfenstein: Then and Now

http://www.dosgamesarchive.com/download/wolfenstein-3d/
http://mashable.com/2014/05/23/wolfenstein-then-and-now/
http://www.gamespot.com/images/1300-2536458
CHARACTERISTICS OF COMPUTER GRAPHICS
CHARACTERISTICS OF COMPUTER GRAPHICS

- Depending on your application, your focus and goals will be different:
  - Real-time vs. Non-real-time
  - Virtual Entities / Environments vs. Visualization / Representation
  - Developing Tools / Algorithms vs. Content Creation
REAL-TIME VS. NON-REAL-TIME

- **Real-time rendering**
  - 15 frames per second (AT BARE MINIMUM – still see skips but it will look more or less animated)
    - 24 fps = video looks smooth (no skips/jumps)
    - 30 – 60 fps is a more common requirement
  - *Examples*: first-person simulations, games, etc.

- **Non-real-time**
  - Could take hours for one frame
  - *Examples*: CG in movies, complex physics simulations, data visualization, etc.

- Often trade-off between speed and quality (image, accuracy, etc.)
VIRTUAL ENTITIES / ENVIRONMENTS VS. VISUALIZATION / REPRESENTATION

• Virtual Entities / Environments
  • Rendering a person, place, or thing
  • Often realistic rendering, but it doesn’t have to be
  • Examples: simulations (any kind), games, virtual avatars, movies

• Visualization / Representation
  • Rendering data in some meaningful way
  • Examples: graphics/charts, data visualization, (to a lesser extent) graphics user interfaces

• Both
  • Rendering some object / environment, but also highlighting important information
  • Examples: CAD/CAM

Remy from Pixar’s Ratatouille: http://disney.wikia.com/wiki/Remy

Face using Crytek engine: http://store.steampowered.com/app/220980/

Tiny and Big: Grandpa’s Leftovers game: http://www.mobygames.com/game/windows/tiny-and-big-grandpas-leftovers/screenshots/gameShotId,564196/

http://www.vtk.org/VTK/project/imagegallery.php
TOOLS/ALGORITHMS VS. CONTENT CREATION

• Developing Tools / Algorithms
  • It’s... well... developing tools and algorithms for graphical purposes
  • Using computer-graphics application programming interfaces (CG API)
    • Common CG APIs: GL, OpenGL, DirectX, VRML, Java 2D, Java 3D, etc.
    • Interface between programming language and hardware
  • Example: how do I write code that will render fur realistically?

• Content Creation
  • Using pre-made software to create graphical objects
  • Called “special purpose software packages” in Hearn-Baker book
  • Example: how do I create a realistic-looking dog in a 3D modeler program?
In this course, we’ll mostly be focusing on developing tools / algorithms to render in virtual entities / environments (in real-time and non-real-time)
COMPUTER GRAPHICS APIS
DEFINITIONS

• Application Programming Interface (API)
  • Standard collection of functions to perform a set of related operations

• Computer Graphics API
  • Performs set of functions related to drawing/rendering images and objects to windows on screen

• *Two paradigms for CG APIs:*
  • CG API only does drawing → only acts as interface between programming language and graphics hardware
    • *Examples:* GL, OpenGL, DirectX
  • Graphics functionality and user-interface functionality integrated:
    • *Example:* Java 2D and Java 3D

• Let’s briefly go over some CG APIs...
GKS AND PHIGS

• **GKS (Graphical Kernel System) – 1984**
  • International effort to develop standard for computer graphics software
    • Adopted as first graphics software standard by ISO (International Standards Organization) and ANSI (American National Standards Institute)
  • Originally 2D → 3D extension developed later

• **PHIGS (Programmer’s Hierarchical Interactive Graphics Standard)**
  • Extension of GKS
  • Developed in 1980’s → standard by 1989
  • 3D standard
  • Increased capabilities for hierarchical modeling, color specifications, surface rendering, and picture manipulations
  • PHIGS+ → added more advanced 3D surface rendering
GL AND OPENGL

• **GL (Graphics Library)**
  - Developed by Silicon Graphics, Inc. (SGI) for their graphics workstations
  - Became de facto graphics standard
  - Fast, real-time rendering
  - Proprietary system

• **OpenGL**
  - Developed as hardware-independent version of GL in 1990’s
  - Specification
  - Was maintained/updated by *OpenGL Architecture Review Board*; now maintained by the non-profit *Khronos Group*
    - Both are consortiums of representatives from many graphics companies and organizations
  - Designed for efficient 3D rendering, but also handles 2D (just set $z = 0$)
  - Generally stable → new features added as extensions

DIRECTX AND DIRECT3D

• DirectX
  • Developed by Microsoft for Windows 95 in 1996
    • Originally called “Game SDK”
  • Actually combinations of different APIs: Direct3D, DirectSound, DirectInput, etc.
  • Less stable → adopts new features fairly quickly (for better or for worse)
  • Only works on Windows and Xbox
WHY ARE WE USING OPENGL?

• In this course, we will be using OpenGL because:
  • It works with practically every platform/system (Windows, Unix/Linux, Mac, etc.)
  • It’s more stable than DirectX/Direct3D
  • It is NOT because DirectX/Direct3D is a bad system
ON THE HORIZON...

• **Vulkan**
  • Next API on the horizon from the Khronos Group → designed from the ground up
  • Gives graphics programmers more direct control over graphics hardware
    • Graphics drivers are simpler
    • More predictable behavior from applications (doesn’t change as much because of different drivers)
  • Unified API for mobile, desktop, console, and embedded platforms
  • Uses intermediate language SPIR-V for shader code
    • Shader source code doesn’t have to be shipped along with program
    • Shader compilation faster
  • Not yet released (but reputedly soon)

https://www.khronos.org/vulkan
OPENGL, GLFW, GLEW, GLM, AND SOIL
OPENGL CORE LIBRARY (OR BASIC LIBRARY)

• Hardware and platform independent
  • Specification that each platform must implement

• Provides functionality for specifying:
  • Graphics primitives
  • Attributes
  • Geometric and viewing transformation
  • ...and more!

• However, because hardware/platform independent, it does NOT provide:
  • Input/output functionality (e.g., mouse and keyboard)
  • Display windows (depends on operating system / windowing system)
OPENGL UTILITY (GLU)

• Every implementation of OpenGL also includes **OpenGL Utility (GLU)**
  • Separate library that provides routines for:
    • Setting up viewing and projection matrices
    • Describing complex objects with line and polygon approximations
    • Displaying quadrics and B-splines (using linear approximations)
    • Surface rendering operations
    • Etc.
  • Also hardware/platform independent
• HOWEVER, GLU calls many of the deprecated OpenGL functionality (pre-version 3.0) and should NOT be used in general.
Arguably the most important thing OpenGL leaves out is creating a window. Fortunately, there are many libraries that handle this for us:

- GLX (OpenGL Extension to the X Windows System)
- AGL (Apple GL)
- WGL (Windows-to-OpenGL)

You’ll notice, however, these are still tied to their respective window systems and/or operating systems...
OPENGL UTILITY TOOLKIT (GLUT)

- Developed by Mark Kilgard while he was working at SGI
  - Built originally to work with GLX
  - Ported to WGL (Windows) by Nate Robins
  - Also version for Apple as well
- Designed to allow programmers to create windows, access input devices, etc., without having to know the specific window system and/or OS
- Still copyrighted by Kilgard
- Unfortunately, it is getting rather old
  - Last Windows port was updated in 2001 → official download site is no longer active
  - Open source version (this is still maintained) → freeglut
TELLING THEM APART

- Functions, constants, and data types will have a different prefix, depending on which library it comes from
  - OpenGL → gl
    - Examples: glBegin, glClear, glCopyPixels, GL_2D, GL_RGB, GLint, GLfloat, etc.
  - GLU → glu
    - Examples: gluLookAt, gluPerspective, etc.
  - GLUT → glut
    - Examples: glutCreateWindow, glutDisplayFunc, etc.
ALTERNATIVES TO GLUT

• Alternatives to GLUT can be found here: https://www.opengl.org/resources/libraries/windowtoolkits/

• For this class, we will be using **GLFW**:  
  • Actively supported  
  • Compatible with Windows, Linux, and Mac  
  • Fairly straightforward
OPENGL VERSIONS

- OpenGL has gone through some significant changes over the years
  - For the full narrative, see: [https://www.opengl.org/wiki/History_of_OpenGL](https://www.opengl.org/wiki/History_of_OpenGL)

- **OpenGL 1.X**
  - Uses fixed-function pipeline only
  - Basically you handed in the rendering data in a piece at a time ("streaming" it in)
  - Still used in the CAM/CAD programs

- **OpenGL 2.X**
  - Introduced GLSL (OpenGL Shader Language)
  - Still used OpenGL 1.X code/functions
OPENGL VERSIONS

• OpenGL 3.X
  • The fixed function pipeline and most of the associated functions were declared deprecated
    • glBegin()/glEnd(), glVertex*, the matrix stacks, etc.
    • Referred to as legacy OpenGL now
    • Many functions outright REMOVED after 3.1 and 3.2
  • Everything pretty much done with vertex and pixel shaders
    • Copy data you need to render to buffers on GPU
    • Have to manage your own matrices
  • Now have notions of core and compatibility OpenGL contexts
    • Implementations do not HAVE to support compatibility contexts, although NVIDIA and ATI are still doing so in Windows and Linux
OPENGL VERSIONS

- OpenGL 4.5
  - Most recent version at time of this slide deck
WHAT CAN I SUPPORT?

• If you downloaded GLEW, there’s a program they have called `glewinfo.exe`
  • Run it → this will create a text file called `glewinfo.txt`
  • Somewhere near the top it will tell you what OpenGL version your system supports:
    • Example: OpenGL version 4.4.0 NVIDIA 344.75 is supported
GLEW

- GLEW
  - OpenGL Extension Wrangler Library
  - Allows us to easily see and access whatever OpenGL extensions are supported on our system
    - You CAN do without GLEW, but it makes life A LOT easier
GLM

• In newer OpenGL versions, you need to manage matrices yourself

• GLM
  • Header-only library
  • Gives you classes for vectors and matrices
  • Also gives you functionality similar to GLU
Not surprisingly, OpenGL (or any of the previous libraries discussed) do not have any code for loading/saving images

- This is a bit awkward when loading textures

SOIL

- Simple OpenGL Image Library
- VERY simple C library for loading images into textures
  - Also allows you to save screenshots