CS 450: COMPUTER GRAPHICS

REVIEW: INTRODUCTION TO COMPUTER GRAPHICS

SPRING 2016

DR. MICHAEL J. REALE
• **Computer graphics** – creating and manipulating images using computers
  • Data used/manipulated can be:
    • 2D or 3D
    • Synthetic or real-world
  • **3 Main Areas**:
    • **Modeling** = Creating/specifying/storing shape and appearance properties of object to be rendered
    • **Rendering** = Creation of shaded images from 3D computer models
    • **Animation** = Techniques used to create the illusion of motion → changes of model over time
COMPUTER GRAPHICS APPLICATIONS

- **Examples of Computer Graphics Applications:**
  - Graphics, Charts, and Data Visualization
    - *Challenges*: large data sets, best way to display/represent data
  - CAD/CADD/CAM
    - *Challenges*: need pixel-perfect rendering
  - Computer Art / Movies
  - Virtual-Reality Environments / Training Simulations
  - Games
CHARACTERISTICS OF COMPUTER GRAPHICS

• **Real-time vs. Non-real-time**
  • **Real-time**
    • 15 fps = BARE MINIMUM but still skips
    • 24 fps = minimum without skips
    • 30 – 60 fps = more common requirement
  • **Non-real-time**
    • Emphasis on rendering quality over rendering speed

• **Virtual Entities / Environments vs. Visualization / Representation**
  • Rendering actual person/place/thing
  • Visualizing data (may not correspond to physical visual reality)
  • Combination \(\rightarrow\) e.g., CAD rendering of real object with highlighted parts

• **Developing Tools / Algorithms vs. Content Creation**
  • Tools/code to perform rendering tasks
  • Creating content to be rendered
APIS AND CG APIS

• 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 intergrated:
    • Example: Java 2D and Java 3D
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*
    - Designed for efficient 3D rendering, but also handles 2D (just set z = 0)
    - Generally stable → new features added as extensions
OPENGL AND GLU

• **OpenGL Core**
  
  • Hardware and platform independent
  
  • **Specification** → each platform must implement
  
  • Does NOT provide:
    
    • Input/output functionality (e.g., mouse and keyboard)
    
    • Display windows (depends on operating system / windowing system)

  • Functions prefixed with **gl**

• **OpenGL Utility (GLU)**
  
  • Included with every implementation of OpenGL
  
  • Provides routines for setting up matrices (among other things)
  
  • Uses DEPRECATED (pre-version 3.0) OpenGL calls → should NOT be used!
  
  • Functions prefixed with **glu**
WINDOW CREATION

- **OS-specific libraries**
  - GLX (OpenGL Extension to the X Windows System)
  - AGL (Apple GL)
  - WGL (Windows-to-OpenGL)

- **OS-independent libraries**
  - OpenGL Utility Toolkit (GLUT)
    - OS-independent
    - Allows window creation and input device access
    - Original version getting old
    - Open source version (still maintained) → freeglut
    - Functions prefixed with glut
  - GLFW
  - ...and others
OPENGL VERSIONS

• **OpenGL 1.X**
  • Uses fixed-function pipeline only → data handed to GPU 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 3.X**
  • The fixed function pipeline deprecated (and REMOVED after 3.1 and 3.2)
    • Called legacy OpenGL or **OpenGL compatibility context**
  • **OpenGL core context:**
    • Rendering controlled mostly with shader programs
    • Data to render is copied to buffers on GPU
    • Matrices handled by programmer

• **OpenGL 4.5 → most recent version**