CIS 454 Computer Graphics
Lecture 3, 09/12/2006

Li Shen
Computer and Information Science
UMass Dartmouth

Notes

- Learning portal
  - Homeworks
  - Class examples

Computer Graphics

Basic System
History
Applications

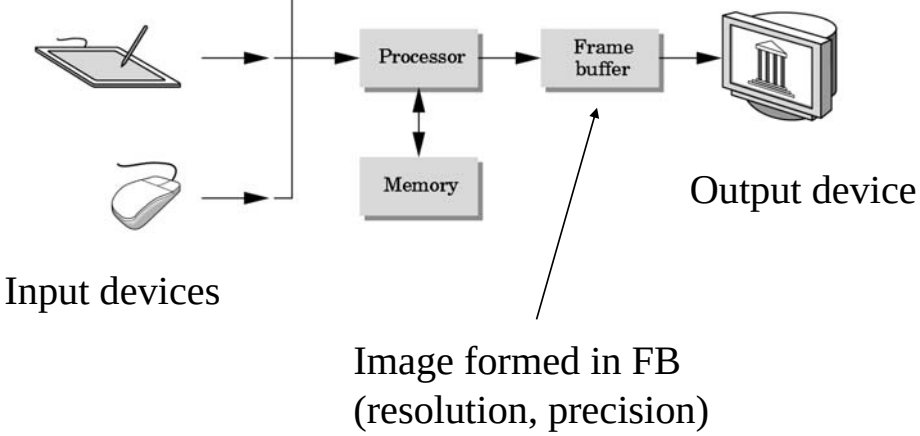
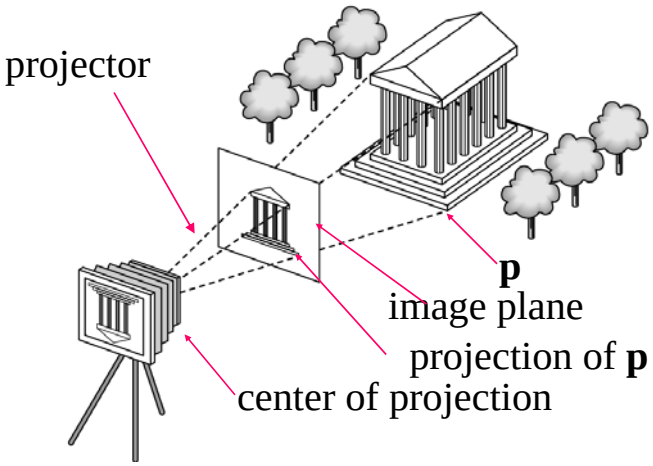


Image Formation

- Fundamental imaging notions
- Physical basis for image formation
  - Light
  - Color
  - Perception
- Synthetic camera model
- Other models

Synthetic Camera Model



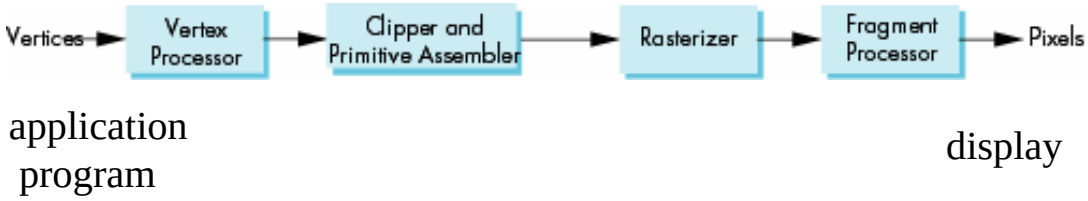
Models and Architectures

Objectives

- Learn the basic design of a graphics system
- Introduce pipeline architecture
- Examine software components for an interactive graphics system

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

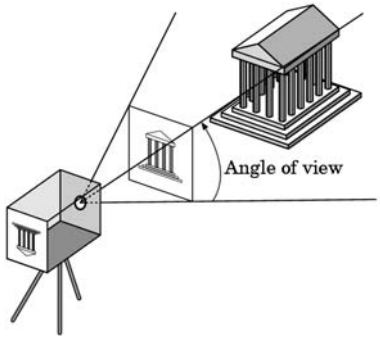
Vertices must be collected into geometric objects before clipping and rasterization can take place

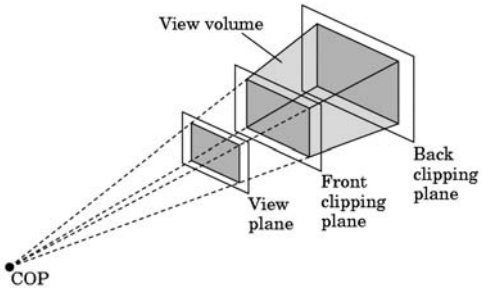
- Line segments
- Polygons
- Curves and surfaces

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





Rasterization

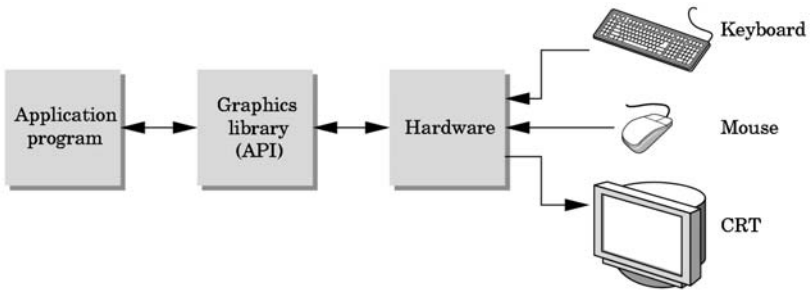
- If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors
- Rasterizer produces a set of fragments for each object
- Fragments are “potential pixels”
  - Have a location in frame buffer
  - Color and depth attributes
- Vertex attributes are interpolated over 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

type of object

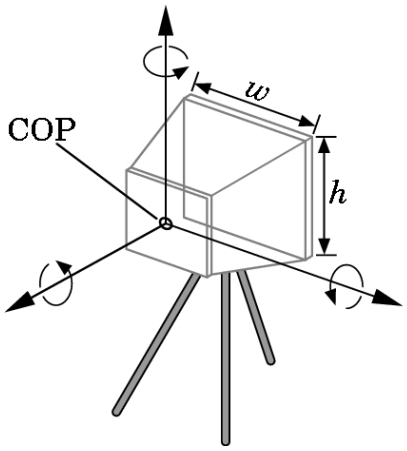
location of vertex

```
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( );
```

end of object definition

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

Programming with OpenGL Part 1: Background

Objectives

- Development of the OpenGL API
- OpenGL Architecture
  - OpenGL as a state machine
- Functions
  - Types
  - Formats
- Simple program

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 IS0 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

OpenGL Evolution

- Controlled by an Architectural Review Board (ARB)
  - Members include SGI, Microsoft, Nvidia, HP, 3DLabs, IBM,…….
  - Relatively stable (present version 2.0)
    - Evolution reflects new hardware capabilities
      - **3D texture mapping and texture objects**
      - **Vertex programs**
  - Allows for platform specific features through extensions

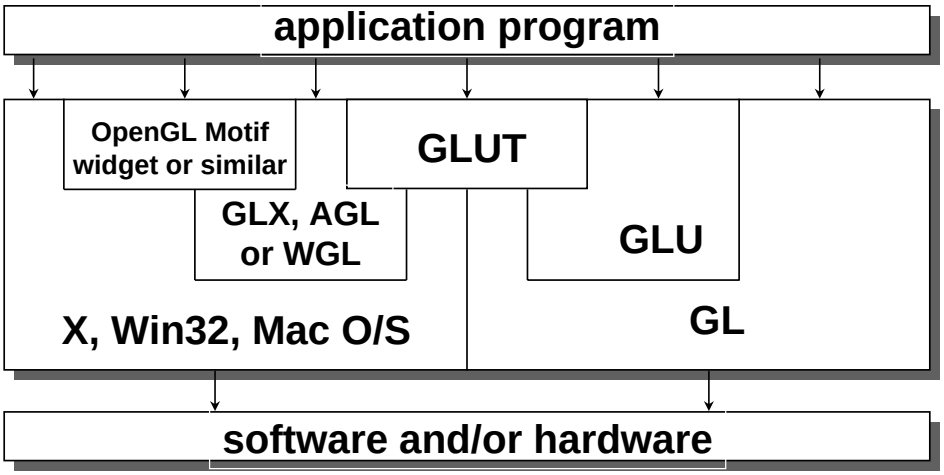
OpenGL Libraries

- OpenGL core library
  - OpenGL32 on Windows
  - GL on most unix/linux systems (libGL.a)
- OpenGL Utility Library (GLU)
  - Provides functionality in OpenGL core but avoids having to rewrite code
- Links with window system
  - GLX for X window systems
  - WGL for Windows
  - AGL for Macintosh

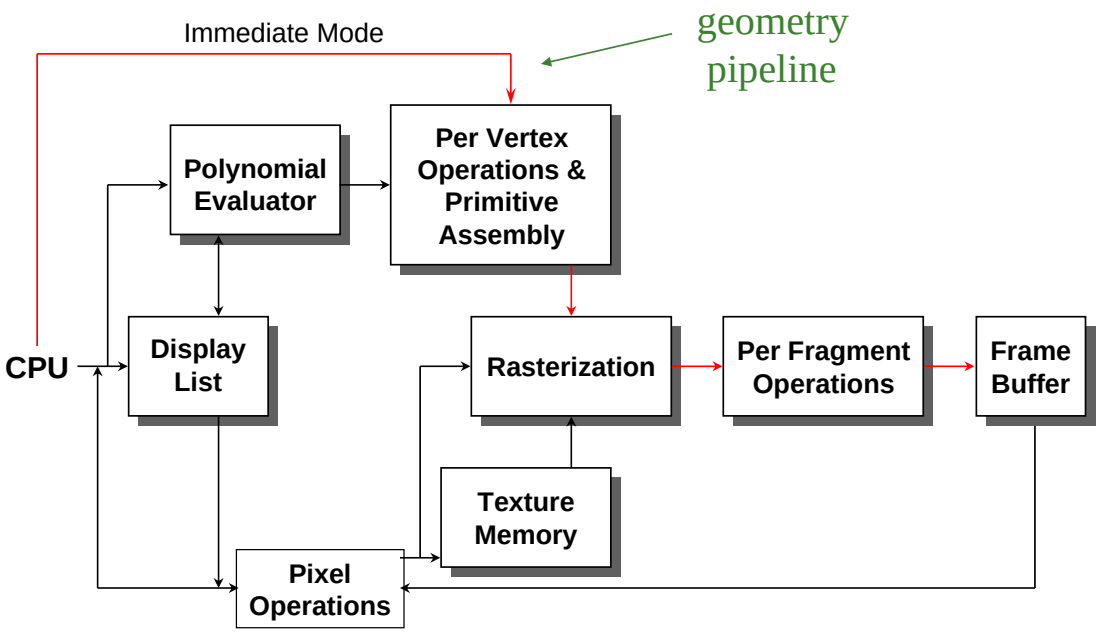
GLUT

- OpenGL Utility Toolkit (GLUT)
  - Provides functionality common to all window systems
    - Open a window
    - Get input from mouse and keyboard
    - Menus
    - Event-driven
  - Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform
    - No slide bars

Software Organization



OpenGL Architecture



OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control (GLUT)
- Input (GLUT)
- Query

OpenGL State

- OpenGL is a state machine
- OpenGL functions are of two types
  - Primitive generating
    - Can cause output if primitive is visible
    - How vertices are processed and appearance of primitive are controlled by the state
  - State changing
    - Transformation functions
    - Attribute functions

Lack of Object Orientation

- OpenGL is not object oriented so that there are multiple functions for a given logical function
  - **`glVertex3f`**
  - **`glVertex2i`**
  - **`glVertex3dv`**
- Underlying storage mode is the same
- Easy to create overloaded functions in C++ but issue is efficiency

OpenGL function format

function name

dimensions

```
glVertex3f(x,y,z)
```

belongs to GL library

`x,y,z` are `floats`

```
glVertex3fv(p)
```

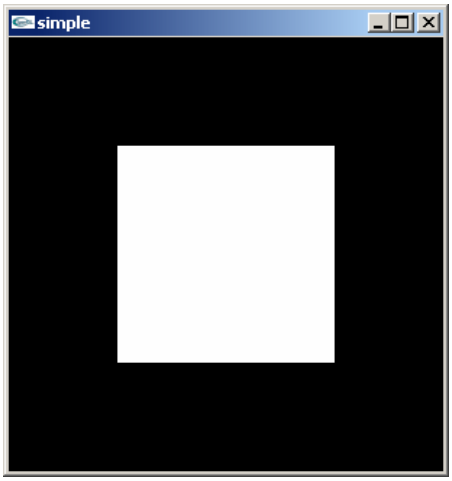
`p` is a pointer to an array

OpenGL #defines

- Most constants are defined in the include files **`gl.h`**, **`glu.h`** and **`glut.h`**
  - Note **`#include <GL/glut.h>`** should automatically include the others
  - Examples
  - **`glBegin(GL_POLYGON)`**
  - **`glClear(GL_COLOR_BUFFER_BIT)`**
- include files also define OpenGL data types: **`GLfloat`**, **`GLdouble`**,….

A Simple Program

Generate a square on a solid background



simple.c

```
#include <GL/glut.h>
void mydisplay(){
      glClear(GL_COLOR_BUFFER_BIT);
      glBegin(GL_POLYGON);
            glVertex2f(-0.5, -0.5);
            glVertex2f(-0.5, 0.5);
            glVertex2f(0.5, 0.5);
            glVertex2f(0.5, -0.5);
      glEnd();
      glFlush();
}
int main(int argc, char** argv){
      glutCreateWindow("simple");
      glutDisplayFunc(mydisplay);
      glutMainLoop();
}
```

The OpenGL Reference Manual
http://www.rush3d.com/reference/opengl-bluebook-1.0/

GLUT
http://www.opengl.org/resources/libraries/glut/spec3/spec3.html

Event Loop

- Note that the program defines a *display callback* function named **`mydisplay`**
  - Every glut program must have a display callback
  - The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened
  - The **`main`** function ends with the program entering an event loop

Defaults

- **`simple.c`** is too simple
- Makes heavy use of state variable default values for
  - Viewing
  - Colors
  - Window parameters
- Next version will make the defaults more explicit

Notes on compilation

- See website and ftp for examples
- Unix/linux
  - Include files usually in …/include/GL
  - Compile with –lglut –lglu –lgl loader flags
  - May have to add –L flag for X libraries
  - Mesa implementation included with most linux distributions
  - Check web for latest versions of Mesa and glut

Compilation on Windows

- Visual C++
  - Get glut.h, glut32.lib and glut32.dll from web
  - Create a console application
  - Add opengl32.lib, glu32.lib, glut32.lib to project settings (under link tab)
- Borland C similar
- Cygwin (linux under Windows)
  - Can use gcc and similar makefile to linux
  - Use –lopengl32 –lglu32 –lglut32 flags

After Class

- Read Chapter 2
- Run class examples
  - Files available in the learning portal