CS450/550

Pipeline Architecture

Adapted From: Angel and Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

Objectives

Learn the basic components of a graphics system
Introduce the OpenGL pipeline architecture
Introduce the structure of OpenGL Libraries
Image Formation Revisited

Can we mimic the synthetic camera model to design graphics hardware software?

Application Programmer Interface (API)

Need only specify
- Objects
- Materials
- Viewer
- Lights

But how is the API implemented?

Physical Approaches

Ray tracing: follow rays of light from center of projection until they either are absorbed by objects or go off to infinity

Can handle global effects
- Multiple reflections
- Translucent objects

Slow
Must have entire DB available

Photon Mapping: Similar approach to deposit light in the scene
Practical Approach: Pipeline Architecture

Process objects one at a time in the order they are generated by the application
   Can consider only local lighting

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
   World Coordinates
   Camera (eye) coordinates
   Screen coordinates

Vertex processor also computes vertex colors
Projection (3D to 2D) & 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 *framebuffer*.

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

**Full Pipeline Diagram**
The Programmers Interface to the Pipeline: API

Programmer sees the graphics system through a software interface: the Application Programmer Interface (API)

Example (old style) Immediate Mode

```c
    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();
```

Retained Mode Graphics

for some number of points
generate a point
store it in a data structure on client
display_all_points() /* One call */
/* still sends all pts*/

Modern GPU Approach

for some number of points
generate a point
store it in a data structure on client
send_all_points_to_gpu()
display_all_points_on_GPU( AS_TRIANGLES)
/* One call to draw them */
/* Tell server how to interpret the data */
Modern OpenGL API

Performance is achieved by using GPU rather than CPU
Control GPU through programs called shaders
Applications job is to send data to GPU
GPU does all rendering

API History: 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

Originally controlled by an Architectural Review Board (ARB)

- Members included SGI, Microsoft, Nvidia, HP, 3DLabs, IBM, …
- Now Khronos Group (www.khronos.org/opengl)
- Was relatively stable (through version 2.5)
  - Backward compatible
- Evolution reflects new hardware capabilities
- Allows platform specific features through extensions
OpenGL 3.1

Totally shader-based
   No default shaders
   Each application must provide both a vertex and a fragment shader
No immediate mode
Few state variables
Most 2.5 functions deprecated
*Backward compatibility not required*
We'll use version 3.2 for this class


OpenGL Versions

OpenGL ES
   Embedded systems
   Version 1.0 simplified OpenGL 2.1
   Version 2.0 simplified OpenGL 3.1
   Shader based

WebGL
   Javascript implementation of ES 2.0
   Supported on newer browsers
OpenGL 4.1 and 4.2 (up to 4.5 Now)
   Add geometry shaders and tessellator

OpenGL Libraries

OpenGL core library
- OpenGL32 on Windows (opengl32.lib)
- GL on most unix/linux systems
- OpenGL Framework on Mac

OpenGL Utility Library (GLU) (glu32.lib)
- Provides functionality in OpenGL core but avoids having to rewrite code (ie. spheres)

Links with window system
- GLX for X window systems
- WGL for Windows (windows.h)
- AGL for Macintosh
- Platform Independent Options (glut, fltk, etc.)

GLUT

OpenGL Utility Library (GLUT) (glut32.lib)
- 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
Software Organization

application program

OpenGL Motif widget or similar

GLU

GLX, AGL or WGL

GL

X, Win32, Mac O/S

software and/or hardware

Angel: Interactive Computer Graphics 3E © Addison-Wesley 2002