The Graphics Pipeline and OpenGL

Prof. Vladlen Koltun
Computer Science Department
Stanford University
The synthetic camera model

- Two components of viewing
  - Set of geometric objects that form the content of the scene
  - Viewer through which the scene is imaged
The graphics pipeline

- Primitives
- Geometry processor
- Rasterizer
- Fragment processor
- Frame buffer
The geometry processor

- Transforms primitives to the camera’s coordinate system, prepares them for rasterization
- Culls primitives facing away from the camera or lying outside the view frustum
The rasterizer

- Generates fragments (proto-pixels)
Fragment processor

- Checks if fragments are visible
- Determines color
- All fragments treated identically, irrespective of the original primitive
Frame buffer

- Memory buffer used for the construction of the image.
- Not all data that passes through the frame buffer is displayed. It is like a sandbox in which the image is constructed.
- Used by the window system for display.
Double buffering

- Render into the back buffer while the window system points to the front buffer. When the next frame is assembled, swap.
- Avoids terrible visual artifacts
Double buffering

- Render into the back buffer while the window system points to the front buffer. When the next frame is assembled, swap.
- Avoids terrible visual artifacts
Advantages and disadvantages of pipeline model

- **Great for parallel processing**
  - Primitives processed independently
  - Fragments processed independently

- **Does not support interactions between multiple objects in the scene**
  - Global illumination, shadows, reflection, refraction
The pipeline is evolving
Alternatives to the pipeline?
Global illumination

- Consider indirect illumination that is transmitted by means of other objects
- Primitives are no longer independent
Ray tracing

- Rays are cast from the viewpoint and followed recursively through the scene.
- Whitted ray tracing: Compute direct illumination from light sources at every point hit by traced rays
• Discretize scene into patches. Compute strength of interaction between patches.

• Shoot light from source patches, deposit in other patches. Iterate until light is absorbed.
Photon mapping

- Stage 1: Trace photons from light sources and deposit onto photon map when photons interact with diffuse surfaces.
- Stage 2: Cast rays from viewpoint and estimate radiance.
OpenGL
Why a graphics API?
Why a graphics API?

- Save developers from having to implement standard functionality
- Facilitate portability through abstraction
```c
#include <GL/glut.h>

void display()
{
    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();
}

void init()
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glColor3f(1.0, 1.0, 1.0);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutCreateWindow("simple");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
}
```
Design considerations for OpenGL

• Enable applications to run on a variety of platforms
• Describe rendering operations succinctly
• Accommodate extensions as graphics hardware evolves
• Maximize performance by closely modeling the capabilities and characteristics of graphics hardware
Agnostic to the window system

- OpenGL was designed to work with a variety of window systems
  - Interaction with the window system is handled by GLUT
- No facilities for obtaining user input
- Does not handle the actual display, merely assembles an image
A rendering API

- The OpenGL API supports the display of geometric primitives
  - Takes a specification of geometric objects
  - Computes illumination
  - Images the scene
- Modeling and animation largely left to the application
Separation of content and viewer

- Separates object description from viewer specification
- Two types of functions
  - Describe objects in the world (the input)
  - Specify how the objects should be processed for constructing an image (the state)
OpenGL is a state machine

- State machine with inputs and outputs
  - Input is geometric objects, output is a set of pixels
  - State machine converts a collection of geometric objects in three dimensions to an image. This process is controlled by the state.
  - State specifies how objects are projected onto the image plane, how they are colored, etc.
Figure 1. Block diagram of OpenGL.
The OpenGL® Machine

The OpenGL® graphics system diagram, Version 1.1. Copyright © 1996 Silicon Graphics, Inc. All rights reserved.

Key to OpenGL Operations

Segal and Akeley, The OpenGL Graphics System: A Specification (Version 1.1)
Declarative vs. Imperative programming

• OpenGL is imperative
  - Operations specify how rendering should be conducted (by changing the state)
  - Operations do not execute individual rendering operations

• In declarative APIs (cf. RenderMan), the application commands the graphics system what to do, not just how to do what it does

• The OpenGL abstraction controls what appears to happen, not what actually happens. Leaves significant freedom for optimization by OpenGL implementations.
Central assumptions:

- The client is computationally weak, the server is computationally powerful
- Bandwidth is low
• Examples
  - Color is part of the state, not associated with the primitive.
  - When a new transformation matrix is specified, the default behavior is to multiply, rather than replace
Persisting primitives

• Display lists
• Vertex buffer objects