API Background

Prof. George Wolberg
Dept. of Computer Science
City College of New York
Objectives

- Graphics API history
- OpenGL API
- OpenGL function format
- Immediate Mode vs Retained Mode
- Examples
The Programmer’s Interface

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

- We shall use the OpenGL API
- **Benefits:** widespread, portable across all platforms
- **Includes:** functions needed to form an image
  - Objects
  - Viewer
  - Light Source(s)
  - Materials
- **Goal:** provides an abstraction layer between the application and the underlying graphics subsystem (GPU).
Example (old style)

```
glBegin(GL_POLYGON)
glVertex2f(0.0, 0.0);
glVertex2f(0.0, 1.0);
glVertex2f(1.0, 0.0);
glVertex2f(1.0, 0.0);
glEnd();
```

- **type of object**
- **location of vertex**
- **end of object definition**
A Simple Program

Generate a square on a solid background
#include <GL/glut.h>
void mydisplay(){
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_QUAD);
   glVertex2f(-0.5,-0.5);
   glVertex2f(-0.5, 0.5);
   glVertex2f( 0.5, 0.5);
   glVertex2f( 0.5,-0.5);
   glEnd();
}
int main(int argc, char** argv){
   glutCreateWindow("simple");
   glutDisplayFunc(mydisplay);
   glutMainLoop();
}

OpenGL Functions

• Primitives
  - Points
  - Line Segments
  - Triangles
• 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
    - In modern OpenGL (3.1+) most state variables are defined by the application and sent to the shaders
Lack of Object Orientation

- OpenGL is not object oriented so there are multiple functions for a given logical function
  - `glUniform3f`
  - `glUniform2i`
  - `glUniform3dv`
OpenGL Function Format

function name

belongs to GL library

x, y, z are floats

dimensions

p is a pointer to an array (vector)
Modern OpenGL (3.1+)

- Performance achieved by using GPU rather than CPU
- Control GPU through programs called shaders
- Application’s job is to send data to GPU
- GPU does all rendering
Immediate Mode Graphics

• Geometry specified by vertices
  - Locations in space (2 or 3 dimensional)
  - Points, lines, circles, polygons, curves, surfaces

• Immediate mode
  - Each time a vertex is specified in application, its location is sent to the GPU
  - Old style used `glVertex`
  - Creates bottleneck between CPU and GPU
  - Removed from OpenGL 3.1 and OpenGL ES 2.0
Retained Mode Graphics

- Put all vertex attribute data in array
- Send array to GPU to be rendered immediately
- Almost OK but problem is we would have to send array over each time we need another render of it
- Better to send array over and store on GPU for multiple renderings
OpenGL and GLSL

• Shader-based OpenGL is based less on a state machine model than a data flow model
• Most state variables, attributes and related pre 3.1 OpenGL functions have been deprecated
• Action happens in shaders
• Job in application is to get data to GPU
Execution Example (WebGL)
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
  - Will only work with legacy code (pre 3.1)

• 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 sliders, spinboxes, combo boxes, radio buttons, …
    - We will use Qt instead
Evolution of OpenGL

Prof. George Wolberg
Dept. of Computer Science
City College of New York
Objectives

• Evolution of the OpenGL pipeline
• A prototype application in OpenGL
• OpenGL shading language (GLSL)
• Vertex shaders
• Fragment shaders
• Examples
What Is OpenGL?

• OpenGL is a computer graphics rendering application programming interface, (API)
  - With it, you can generate high-quality color images by rendering with geometric and image primitives
  - It forms the basis of many interactive applications that include 3D graphics
  - By using OpenGL, the graphics part of your application can be
    • operating system independent
    • window system independent
Course Ground Rules

• We’ll concentrate on latest versions of OpenGL
• Enforces a new way to program with OpenGL
  - Allows more efficient use of GPU resources
• Modern OpenGL (3.1+) doesn’t support many of the “classic” ways of doing things, such as
  - Fixed-function graphics operations, like vertex lighting and transformations
• All applications must use *shaders* for their graphics processing
# OpenGL Versions

<table>
<thead>
<tr>
<th>Version</th>
<th>Publication Date</th>
<th>Version</th>
<th>Publication Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenGL 1.0</td>
<td>January 1992</td>
<td>OpenGL 3.0</td>
<td>August 2008</td>
</tr>
<tr>
<td>OpenGL 1.1</td>
<td>January 1997</td>
<td>OpenGL 3.1</td>
<td>March 2009</td>
</tr>
<tr>
<td>OpenGL 1.2</td>
<td>March 1998</td>
<td>OpenGL 3.3</td>
<td>March 2010</td>
</tr>
<tr>
<td>OpenGL 1.2.1</td>
<td>October 1998</td>
<td>OpenGL 4.0</td>
<td>March 2010</td>
</tr>
<tr>
<td>OpenGL 1.3</td>
<td>August 2001</td>
<td>OpenGL 4.1</td>
<td>July 2010</td>
</tr>
<tr>
<td>OpenGL 1.4</td>
<td>July 2002</td>
<td>OpenGL 4.2</td>
<td>August 2011</td>
</tr>
<tr>
<td>OpenGL 1.5</td>
<td>July 2003</td>
<td>OpenGL 4.3</td>
<td>August 2012</td>
</tr>
<tr>
<td>OpenGL 2.0</td>
<td>September 2004</td>
<td>OpenGL 4.4</td>
<td>July 2013</td>
</tr>
<tr>
<td>OpenGL 2.1</td>
<td>July 2006</td>
<td>OpenGL 4.5</td>
<td>August 2014</td>
</tr>
</tbody>
</table>
Evolution of the OpenGL Pipeline

• OpenGL 1.0 was released on July 1st, 1994
• Its pipeline was entirely *fixed-function* - the only operations available were fixed by the implementation

• The pipeline evolved - but remained based on fixed-function operation through OpenGL versions 1.1 through 2.0 (Sept. 2004)
Example (old style)

```c
glBegin(GL_POLYGON)
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 1.0);
    glVertex2f(1.0, 0.0);
    glVertex2f(1.0, 0.0);
    glVertex2f(1.0, 0.0);
    glVertex2f(0.0, 0.0);
glEnd();
```
#include <GL/glut.h>

void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_QUAD);
    glVertex2f(-0.5,-0.5);
    glVertex2f(-0.5,0.5);
    glVertex2f(0.5,0.5);
    glVertex2f(0.5,-0.5);
    glEnd();
}

int main(int argc, char** argv){
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}

A Simple Program: It Used to be Easy
Event Loop

• Note that the program specifies 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
Beginnings of The Programmable Pipeline

• OpenGL 2.0 (officially) added programmable shaders
  - *vertex shading* augmented the fixed-function transform and lighting stage
  - *fragment shading* augmented the fragment coloring stage
• However, the fixed-function pipeline was still available
Example (GPU based)

• Put geometric data in an array
  
  ```cpp
  std::vector<vec2> points;
  points.push_back(vec2(0.0, 0.0));
  points.push_back(vec2(0.0, 1.0));
  points.push_back(vec2(1.0, 0.0));
  numPoints = 3;
  ```

• Send array to GPU
  - See next slide

• Tell GPU to render geometry as triangles
  
  ```cpp
  glClear(GL_COLOR_BUFFER_BIT);
  glDrawArrays(GL_TRIANGLES, 0, 3);
  ```
enum {
    ATTRIB_VERTEX,
    ATTRIB_COLOR,
    ATTRIB_TEXTURE_POSITION
};

// create vertex buffer and get handle to it
GLuint vertexBuffer;
glGenBuffers(1, &vertexBuffer);

// bind buffer to the GPU; all future drawing calls gets data from this buffer
glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);

// copy the vertices from CPU to GPU
glBufferData(GL_ARRAY_BUFFER, numPoints*sizeof(vec2), &points[0], GL_STATIC_DRAW);

// enable the assignment of attribute vertex variable
glEnableVertexAttribArray(ATTRIB_VERTEX);

// assign the buffer object to the attribute vertex variable
glVertexAttribPointer(ATTRIB_VERTEX, 2, GL_FLOAT, false, 0, NULL);
An Evolutionary Change

- OpenGL 3.0 introduced the *deprecation model* - the method used to remove features from OpenGL
- Pipeline remained the same until OpenGL 3.1 (released March 24th, 2009)
- Introduced change in how OpenGL contexts are used

<table>
<thead>
<tr>
<th>Context Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>Includes all features (including those marked deprecated) available in the current version of OpenGL</td>
</tr>
<tr>
<td>Forward Compatible</td>
<td>Includes all non-deprecated features (i.e., creates a context that would be similar to the next version of OpenGL)</td>
</tr>
</tbody>
</table>
The Exclusively Programmable Pipeline

- OpenGL 3.1 removed the fixed-function pipeline
  - programs were required to use only shaders
  - no default shaders
  - app must provide both a vertex and a fragment shader

- Additionally, almost all data is GPU-resident
  - all vertex data sent using buffer objects
More Programmability

• OpenGL 3.2 (released August 3rd, 2009) added an additional shading stage: geometry shaders
  - modify geometric primitives within the graphics pipeline
More Evolution: Context Profiles

• OpenGL 3.2 also introduced context profiles
  - profiles control which features are exposed
  - currently two types of profiles: core and compatible
  - Core profile deprecates legacy features (trim version)
  - Compatibility profile maintains backwards compatibility with all versions back to version 1.0.

<table>
<thead>
<tr>
<th>Context Type</th>
<th>Profile</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>core</td>
<td>All features of the current release</td>
</tr>
<tr>
<td></td>
<td>compatible</td>
<td>All features ever in OpenGL</td>
</tr>
<tr>
<td>Forward Compatible</td>
<td>Core</td>
<td>All non-deprecated features</td>
</tr>
<tr>
<td></td>
<td>Compatible</td>
<td>Not supported</td>
</tr>
</tbody>
</table>
The Latest Pipelines

- OpenGL 4.1 (released July 25th, 2010) included additional shading stages: \textit{tessellation-control} and \textit{tessellation-evaluation} shaders
- Latest version is 4.5 (August 2014)
Other Shader-Based Versions

• OpenGL ES
  - Designed for embedded and hand-held devices such as cell phones and tablets
  - Version 1.0 (2003) based on OpenGL 2.1
  - Version 2.0 (2007) based on OpenGL 3.1
  - Version 3.0 (2012) based on OpenGL 4.3
  - Version 3.2 (August 2015)

• WebGL
  - Javascript implementation of ES 2.0
  - Runs on most recent browsers