Computer Graphics Programming I

- Agenda:
  - Introduce course
  - Introduce OpenGL & SDL
  - Basics of drawing with OpenGL
    - Basic drawing / view state
    - Overview of common drawing operations
  - OpenGL's buffers
    - Color buffer
    - Depth buffer
    - Stencil buffer & buffers that we won't use this term (briefly)

What should you already know?

- C++ and object oriented programming

What should you already know?

- C++ and object oriented programming
  - For most assignments you will need to implement classes that conform to a very specific interface.

What should you already know?

- C++ and object oriented programming
  - For most assignments you will need to implement classes that conform to a very specific interface.
- Graphics terminology and concepts

What should you already know?

- C++ and object oriented programming
  - For most assignments you will need to implement classes that conform to a very specific interface.
- Graphics terminology and concepts
  - Polygon, pixel, texture, infinite light, point light, spot light, etc.

What should you already know?

- C++ and object oriented programming
  - For most assignments you will need to implement classes that conform to a very specific interface.
- Graphics terminology and concepts
  - Polygon, pixel, texture, infinite light, point light, spot light, etc.
- Some knowledge of linear algebra / vector math.

What should you already know?

- C++ and object oriented programming
  - For most assignments you will need to implement classes that conform to a very specific interface.
- Graphics terminology and concepts
  - Polygon, pixel, texture, infinite light, point light, spot light, etc.
- Some knowledge of linear algebra / vector math.
  - Can probably pick most of it up on the way, but be prepared to work a little harder.

What will you learn?

- Create and use a window for OpenGL drawing.

What will you learn?

- Create and use a window for OpenGL drawing.
  - As a *cross-platform* graphics interface, OpenGL has no knowledge of windows, mice, keyboards, etc.

What will you learn?

- Create and use a window for OpenGL drawing.
  - As a *cross-platform* graphics interface, OpenGL has no knowledge of windows, mice, keyboards, etc.
- Draw static and animated models.

What will you learn?

- Create and use a window for OpenGL drawing.
  - As a *cross-platform* graphics interface, OpenGL has no knowledge of windows, mice, keyboards, etc.
- Draw static and animated models.
  - There are several methods available in OpenGL... the advanced methods will wait until next term.

What will you learn?

- Create and use a window for OpenGL drawing.
  - As a *cross-platform* graphics interface, OpenGL has no knowledge of windows, mice, keyboards, etc.
- Draw static and animated models.
  - There are several methods available in OpenGL... the advanced methods will wait until next term.
- Fixed-function lighting and texture combiners.

What will you learn?

- Create and use a window for OpenGL drawing.
  - As a *cross-platform* graphics interface, OpenGL has no knowledge of windows, mice, keyboards, etc.
- Draw static and animated models.
  - There are several methods available in OpenGL... the advanced methods will wait until next term.
- Fixed-function lighting and texture combiners.
  - Most of OpenGL 1.x *except* shadow maps.
  - Programmable shaders will wait until next term too.

How will you be graded?

- Bi-weekly quizzes worth 5 points each.
- A final exam worth 50 points.
- Bi-weekly programming assignments with 10 points each.
- A term project worth 50 points.

How will programs be graded?

- First and foremost, does the program produce the correct output?
- Are appropriate algorithms and data-structures used?
- Is the code readable and clear?

10,000 Foot OpenGL Overview

- Created by SGI due to industry demand for a standard more open than Iris GL.
  - Originally controlled by the OpenGL Architecture Review Board (ARB).
  - Now controlled by the Khronos Group.
- Member companies create and *vote* on additions to the specification.
  - Version 1.0 ratified in 1992
  - Version 2.1 ratified in August 2006.
  - Version 3.0 is coming later this year.

OpenGL Design Principles

- OpenGL is a *low-level*, device independent graphics interface.
- From *The Design of the OpenGL Graphics Interface*, by Mark Segal and Kurt Akeley:

> "An essential goal of OpenGL is to provide device independence while still allowing complete access to hardware functionality. The API therefore provides access to graphics operations at the lowest possible level that still provides device independence."

OpenGL Design Principles (cont.)

- Based on a client-server model.

OpenGL Design Principles (cont.)

- Based on a client-server model.
  - Shows its X-Windows origins. Client (application program) and server (rendering program) were running on different computers.

OpenGL Design Principles (cont.)

- Based on a client-server model.
  - Shows its X-Windows origins. Client (application program) and server (rendering program) were running on different computers.
  - Still works! Client (application program) and server (firmware on the gfx card) *are* different computers.

OpenGL Design Principles (cont.)

- Based on a client-server model.
  - Shows its X-Windows origins. Client (application program) and server (rendering program) were running on different computers.
  - Still works! Client (application program) and server (firmware on the gfx card) *are* different computers.
- The GL is a *state machine* with a data *push model.*

OpenGL Design Principles (cont.)

- Based on a client-server model.
  - Shows its X-Windows origins. Client (application program) and server (rendering program) were running on different computers.
  - Still works! Client (application program) and server (firmware on the gfx card) *are* different computers.
- The GL is a *state machine* with a data *push model.*
  - Data typically only flows *into* the GL.

OpenGL Design Principles (cont.)

- Based on a client-server model.
  - Shows its X-Windows origins. Client (application program) and server (rendering program) were running on different computers.
  - Still works! Client (application program) and server (firmware on the gfx card) *are* different computers.
- The GL is a *state machine* with a data *push model.*
  - Data typically only flows *into* the GL.
  - Commands change state that affect rendering.

References

http://citeseer.ist.psu.edu/segal94design.html

- Paper is a bit dated, but it's still an interesting read.

http://www.opengl.org/news/permalink/the_opengl_arb_officially_announced_opengl_3/

Break

OpenGL Conventions

- OpenGL has a very specific set of naming conventions.
  - Each function, type, or enumerant must adhere to a set of rules defined in the spec.
  - Some of these conventions make up for the fact that C does not have function overloading.

OpenGL Types

- Each data type name begins with `GL`.
- Each data type has a defined function suffix.
  - More on this later.
- Each data type has a defined bit-size.
  - The bit-size is the same on *all* platforms.
- Integral types may be signed or unsigned.
  - Unsigned types get a u after the `GL`.

OpenGL Type Examples

GL Type Name	Common C Type	Bit-size	Notes
GLbyte	char	8-bits	
GLshort	short	16-bits	
GLint	int	32-bits	May be long
GLubyte	unsigned char	8-bits	
GLushort	unsigned short	16-bits	
GLuint	unsigned int	32-bits	May be unsigned long
GLfloat	float	32-bits	Single precision float
GLdouble	double	64-bits	Double precision float
GLboolean	unsigned char	8-bits	

Table 2.1 on page 44 in the book lists the remaining types.

OpenGL Enumerants

- Each enumerant name begins with `GL_`.
- Names of enumerants are always upper-case.
- When passed as parameters to functions, enumerants have the type `GLenum`.
- Examples:
  - `GL_TRIANGLES`, `GL_PROJECTION`, etc.

OpenGL Functions

- Each function name begins with `gl`.
- Each function name that has multiple forms will end with a description of its parameter types.
- Each function name separates words by alternating upper and lower case.

OpenGL Function Examples

- Single signature functions:
  - `glBegin`, `glEnd`, `glClearColor`, `glShadeModel`, etc.
- Multiple signature functions:
  - `glVertex3f`, `glVertex3fv`, `glVertex4f`, etc.
  - 3 and 4 specify the data count. `f` specifies the data type (`GLfloat`). `v` specifies a pointer (vector).

OpenGL Function Examples

- Single signature functions:
  - `glBegin`, `glEnd`, `glClearColor`, `glShadeModel`, etc.
- Multiple signature functions:
  - `glVertex3f`, `glVertex3fv`, `glVertex4f`, etc.
  - 3 and 4 specify the data count. `f` specifies the data type (`GLfloat`). `v` specifies a pointer (vector).
  - `glVertex3f(GLfloat x, GLfloat y, GLfloat z);`
  - `glVertex3fv(const GLfloat *v);`

What is SDL?

- From the front page of libsdl.org:

  “Simple DirectMedia Layer is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer.”

- What does this mean for us?

What is SDL?

- From the front page of libsdl.org:

  “Simple DirectMedia Layer is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer.”

- What does this mean for us?
  - *Lots* of web sites have OpenGL example code that uses SDL.
  - Since I use Linux, code that I write will be useful to you. :)

What is SDL? (cont.)

- SDL gives us a platform independent way to interact with platform-dependent issues.
  - OpenGL makes the 3D part platform-independent, but that's it.
  - At the very least we need to open a window and process some keyboard input.

Basic SDL Usage

- Every SDL program must initialize the library:

```
if (SDL_Init(SDL_INIT_VIDEO) != 0) {
    exit(1);
}
atexit(SDL_Quit);
```

- This is more a C way. In C++ we could use a singleton instead.
  - After the contructor, call an `init` method that does `SDL_Init`.
  - The destructor calls `SDL_Quit`.

Opening a window with SDL

- After initializing the library, we have to tell it what kind of window we want.
  - Window size, color depth, etc.
  - `SDL_GL_SetAttribute` does this.

```
/* Request at least 8-bits of red. */
SDL_GL_SetAttribute(SDL_GL_RED_SIZE, 8);

/* Request at least 8-bits of alpha. */
SDL_GL_SetAttribute(SDL_GL_ALPHA_SIZE, 8);

/* Request at least 4-bits of stencil buffer. */
SDL_GL_SetAttribute(SDL_GL_STENCIL_SIZE, 4);
```

Opening a window with SDL (cont.)

- After describing the window we want, we have to open the window.
  - Specify a few more window attributes.
  - `SDL_GL_SetVideoMode` does this.

```
/* Open a double-buffered 640x480 window.  Use
 * the default color depth (set previously).
 */
SDL_GL_SetVideoMode(640, 480, 0,
                    (SDL_DOUBLEBUF|SDL_OPENGL));
```

Getting input with SDL

- SDL provides input as a series of *events*.
  - `SDL_WaitEvent` blocks until an event is received.
  - `SDL_PollEvent` always returns immediately.
- Each event has a *type.*
  - A key press event has type `SDL_KEYDOWN`.
  - If no real event is available, the event type returned by `SDL_PollEvent` is `SDL_NOEVENT`.
- Events may have a data payload depending on the type.

Getting input with SDL

```
SDL_PollEvent(&e);
switch (e.type) {
case SDL_KEYDOWN: {
    switch (e.key.keysym.sym) {
    case 'q':
        exit(0);
    }
    break;
}
```

SDL + OpenGL “Hello, world!”

Two kinds of operations in OpenGL

- State management

Two kinds of operations in OpenGL

- State management
  - Enabling lights
  - Configuring textures
  - Setting alpha blending modes
  - etc.

Two kinds of operations in OpenGL

- State management
  - Enabling lights
  - Configuring textures
  - Setting alpha blending modes
  - etc.
- Drawing

Two kinds of operations in OpenGL

- State management
  - Enabling lights
  - Configuring textures
  - Setting alpha blending modes
  - etc.
- Drawing

Two kinds of operations in OpenGL

- State management
  - Enabling lights
  - Configuring textures
  - Setting alpha blending modes
  - etc.
- Drawing
  - Clearing the screen
  - Drawing 2D images (fonts, HUDs, etc.)
  - Drawing 3D polygons

Required State

- Before drawing anything, some state must be set
  - Set the viewport
  - Set the viewing volume
  - Set the camera
- These *must* also be reset each time the window is resized
  - The SDL drawing surface also has to be recreated on a window resize

Resize Routine

```
void handle_resize(int w, int h)
{
    my_surf = SDL_SetVideoMode(w, h, 0, (SDL_RESIZABLE | SDL_OPENGL));

    // Set the viewport and the view volume.
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (width <= height) {
        const float aspect = float(h) / float(w);
        glOrtho(-range, range, -range * aspect,
                range * aspect, -range, range);
    } else {
        const float aspect = float(w) / float(h);
        glOrtho(-range * aspect, range * aspect,
                -range, range, -range, range);
    }

    // Identity puts camera at (0, 0, 0) looking down -Z axis.
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

Vertex Data

- Vertex data specified inside a begin / end pair.

```
glBegin(GL_LINES);
glVertex3f(1.0, 1.0, 0.0);
glVertex3f(2.0, 3.0, 0.0);
glEnd();
```

- Multiple data elements per vertex:
  - Color, normal, texture coordinate, etc.
- State changes are *not* allowed between begin / end.
- There are other forms of drawing, and they are *all* described in terms of begin / end.

Vertex Data (cont.)

- The `glVertex` call “provokes” the vertex.
  - Conceptually, this is when all the data for the vertex gets sent to the hardware.

Vertex Data (cont.)

- The `glVertex` call “provokes” the vertex.
  - Conceptually, this is when all the data for the vertex gets sent to the hardware.
- What color will each point be?

```
glColor3fv(red);
glBegin(GL_POINTS);
glVertex3fv(point[0]);
glColor3fv(blue);
glVertex3fv(point[1]);
glVertex3fv(point[2]);
glColor3fv(green);
glVertex3fv(point[3]);
glColor3fv(purple);
glEnd();
```

Primitive Types

- The type of the primitive to be drawn is specified as a parameter to `glBegin`.
  - Point, line, triangle, quadrilateral, and arbitrary polygon primitives are available
  - Primitives can be grouped in strips (triangles & quads) or fans (triangles)
- `GL_TRIANGLES` and `GL_TRIANGLE_STRIP` are by far the most common.

What the heck is a strip or a fan?

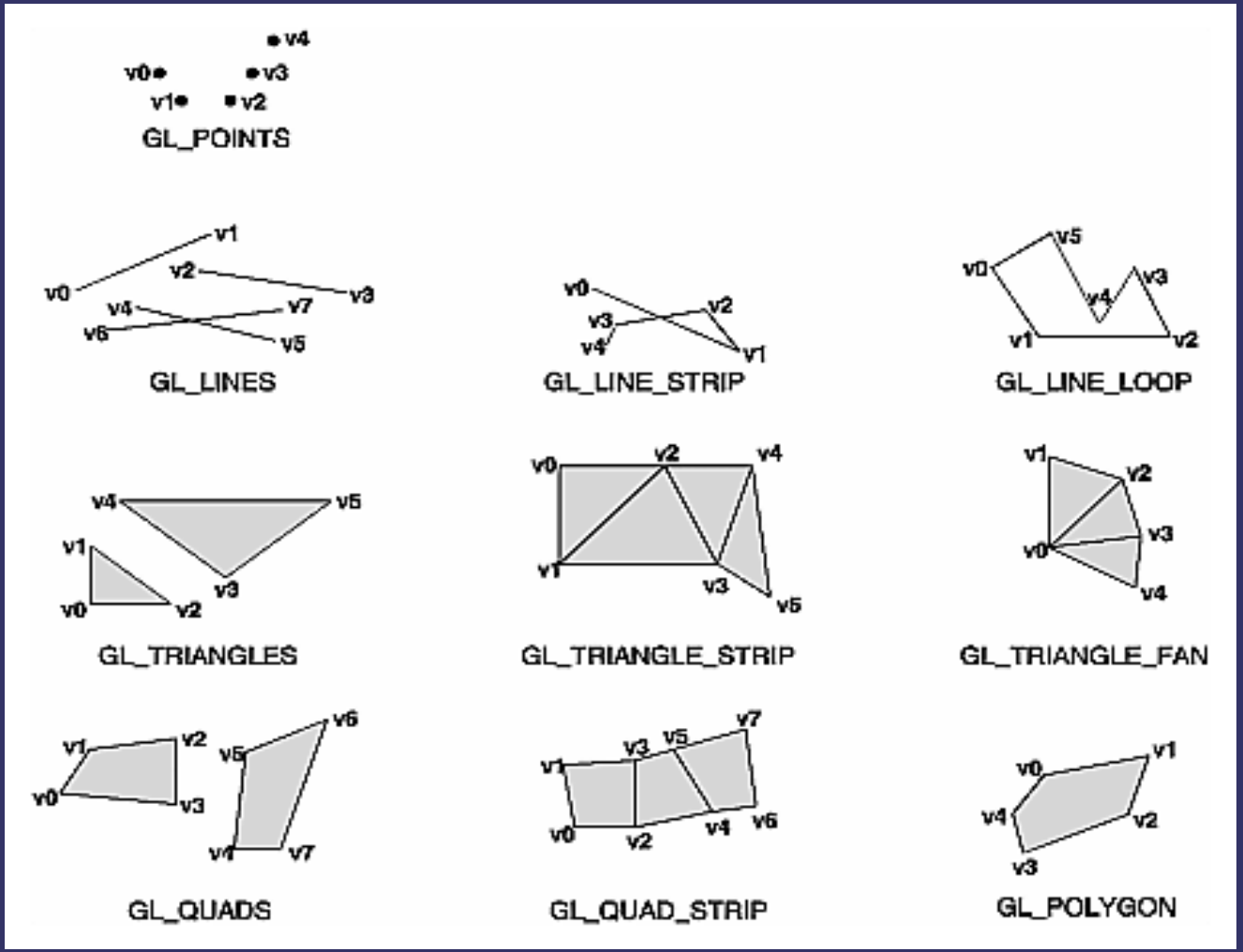

Image borrowed from “OpenGL Programming Guide”.

Hidden Surface Removal

- Two ways built into OpenGL for HSR.
  - Z-buffering
  - Back-face culling
- Nearly every OpenGL program uses both.
  - Z-buffering gives pixel-perfect results.
  - Back-face culling eliminates polygons before they're drawn.

Culling State

- Back-face culling is enabled with `glEnable(GL_CULL_FACE)`.
- Front-facing orientation is selected with `glFrontFace`.
  - `glFrontFace(GL_CW)` makes clockwise ordered faces front-facing.
  - `glFrontFace(GL_CCW)` makes counter-clockwise ordered faces front-facing.

Polygon “winding”

- Several methods exist to do back-face culling. OpenGL uses the “clockwise vs. counter-clockwise method.”
  - When a polygon faces towards the viewer, it's points are viewed in clockwise order.
  - When a polygon faces away from the viewer, it's points are viewed in a counter-clockwise order.
    - Try this with a clock.

Depth Buffer

- Depth buffer (or z-buffer) compares the depth value of each fragment of a polygon with the depth value stored at each pixel.
  - If the test passes, the fragment gets drawn.
  - If the test fails, the fragment is discarded.
- To use the depth buffer, SDL has to create the buffer:

```
SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE,
  15);
```

Additional Depth State

- Depth test has an enable:

  ```
  glEnable(GL_DEPTH_TEST);
  ```

- Also select the comparison mode.
  - `glDepthFunc(GLenum mode)`
  - `GL_LESS`, `GL_LEQUAL`, `GL_GREATER`, `GL_GEQUAL`, `GL_EQUAL`, `GL_NOTEQUAL`, `GL_NEVER`, `GL_ALWAYS`
- Also need to clear the depth buffer.
  - Or `GL_DEPTH_BUFFER_BIT` with the existing `glClear` mask.

Stencil Buffer

- Extra per-pixel buffer containing integer values.
  - Values in the stencil buffer can control drawing.
- Stencil buffer is often stored interleaved with depth buffer
  - 8-bit stencil with 24-bit depth is most common, but 1-bit stencil with 15-bit depth is sometimes available
- To use the stencil buffer, SDL has to create it:

```
SDL_GL_SetAttribute(SDL_GL_STENCIL_SIZE,
  1);
```

What can you do with it?

- Write values to it! Several operations available:
  - `GL_KEEP` – leave the value alone
  - `GL_ZERO` – clear value to zero
  - `GL_REPLACE` – replace value with preset value
  - `GL_INCR` – increment value, clamp to max value
    - `GL_INCR_WRAP` increments but wraps to zero
  - `GL_DECR` – decrement value, clamp to zero
    - `GL_DECR_WRAP` decrements but wraps to max value
  - `GL_INVERT` – bitwise inversion of value

Writing values to the stencil buffer

- A different operation can be set for pixels that pass the Z test, fail the Z test, or fail the stencil test (see next slide)
  - `glStencilOp` sets all three operations
  - Several extensions and OpenGL 2.1 add the ability to perform a *different* set of operations for front facing and back facing polygons
    - We'll talk about this functionality later (probably next term).

Miscellaneous stencil functions

- `glClearStencil` clears the stencil buffer to some value
- `glStencilMask` controls which bits can be written by stencil operations

Stencil testing

- `glStencilFunc` sets the operation, reference value, and a mask
  - The usual depth test values are available: `GL_NEVER`, `GL_LESS`, `GL_LEQUAL`, `GL_GREATER`, `GL_GEQUAL`, `GL_EQUAL`, `GL_NOTEQUAL`, and `GL_ALWAYS`
- Per-pixel, `(ref & mask) op (stencil & mask)` is used *before* the depth test to determine whether or not to write to the color buffer

Example

```
glClearStencil(0);
glEnable(GL_STENCIL_TEST);

/* Write 1 to stencil where polygon is drawn.
 */
glStencilFunc(GL_ALWAYS, 1, ~0);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
draw_some_polygon();

/* Draw scene only where stencil buffer is 1.
 */
glStencilFunc(GL_EQUAL, 1, ~0);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
draw_scene();
```

Other Buffers

- Some advanced OpenGL modes allow calculation of multiple colors at a time.
  - These extra values are written to *auxiliary buffers.*
  - We probably won't cover these in this sequence.
- Selection of the target buffer is made with `glDrawBuffer` (or `glDrawBuffers`).
  - In double buffer mode we can draw to `GL_FRONT` or `GL_BACK`.
  - In stereo mode these become `GL_FRONT_LEFT`, `GL_FRONT_RIGHT`, `GL_BACK_LEFT`, and `GL_BACK_RIGHT`.

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Khronos and OpenGL ES are trademarks of the Khronos Group.
- Other company, product, and service names may be trademarks or service marks of others.