CSC418: Computer Graphics Tutorial 1

Michael Tao

September 20, 2012

Plan for Today (and next Week)

- Event-Driven Programming
  - GLUT
- C++ Quick Introduction
- OpenGL
  - Commands
  - Hierarchical Programming

Event-Driven Programming

- We want to manipulate what we see
  - Traverse Scene
  - Modify Environment
  - Framerate

Event-Driven Programming

- We want to manipulate what we see
  - Traverse Scene
  - Modify Environment
  - Framerate
- Graphics Programs require Graphicical User Interfaces
  - User Input
    - Mouse
    - Keyboard
  - System Input
    - Window Resizing
    - Window Minimization
    - Timers

Simple Event-Driven Program

```
int main()
{
    while(true)
    {
        ...
        if(event.happened())
        {
            doEventCode();
        }
        ...
    }
}
```

Packages To Use

- GLUT
  - Used in this class!
- QT
  - My Favorite!
- GTK
- ...

GLUT

- Set of of slots for various functionalities
  - What to render?
  - What to do when window reshaped?
  - What to do when key pressed?
  - What to do when mouse pressed?
- Called Callback Registration

GLUT

Functional slots defined by Callback Registration

```
int main(int argc, char * argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("Window");
    glutDisplayFunc(renderFunction);
    glutKeyboardFunction(keyboardFunction);
    glutReshapeFunc(reshapeFunc);
    glutMainLoop();

}
```

GLUT: Callback Function Examples

```
void keyboardFunction(unsigned char key, int x, int y)
{
    if(key == ’p’)
    {
        printf("Mouse position: %d %d",x,y);
    }

}
void renderFunction()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER);
    glClearColor(0,0,0,1);

    drawStuff();
    glutSwapBuffers();
}
```

GLUT: Use Sparingly

- Designed for rapid prototyping of small applications
- Lacks a variety of features
  - GLUI
- Very C way to do things (C++ is better (will get to that later...))
- For personal projects use Qt
  - C++ and Object Oriented
  - Super popular (all KDE programs)
  - Signal/Slots are really nice
  - QML
  - We won't be using this in this class

C++ Quick Introduction

- Incredibly Complicated Language
  - Lots of nice features piled ontop of each other
- Definitely not C
  - Object Orientation
  - References
  - const Correctness
  - Templates (Generics)
- Combination of things seen in C and Java
- Do you guys want to hear about this?

C++: Classes

- Basically structs with member functions
    - Difference is default privacy
- Different syntax for accessing depending on context
- Constructors and Destructors

C++: Classes

```
struct Foo() {
    Foo(int a_=0): a(a_) {}
    int f() {return 1;}
    int a;
    static int g(){return 3;}
}
class Bar() {
    public:
        Bar(): myfoo(new Foo(4)) {}
        ~Bar(){delete myfoo;}
        Foo * myfoo;
        int g(){return -1;}
}
Foo foo;
Bar * bar_ptr     = new Bar();
bar_ptr->myfoo.a = foo.f();
foo.a             = bar_ptr->g();
int a             = Foo::g();
```

C++: References and const

- Pointers
- Similar to what you see in most other languages so far
  - Pass by value / pass by reference
  - Except we explicitly declare when to do what
- const provides security over modification

C++: References and const

```
int f(Foo & foo)
{
    return foo.a = 3;
}
int g(const Foo & foo)
{
    h(a);//h has to be h(const Foo &)
    return foo.a;
}
const x = 0;
const Foo;
const * const Foo = &foo;
```

C++: Templates

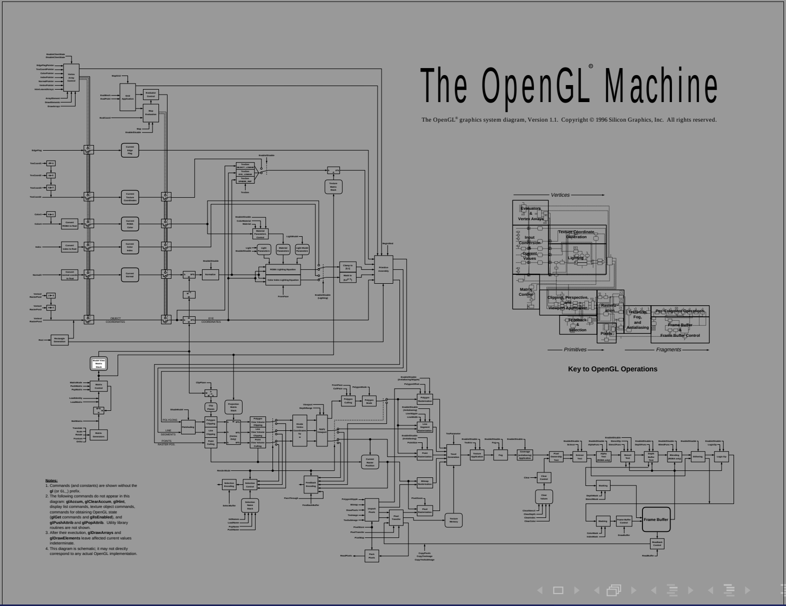
- Allow for generic typing of functions/classes
- Resolved at compilation

```cpp
template <typename T>
T mymax(const T & a, const T & b)
{
    return (a>b)?a:b;
}
int a = mymax(3,4);
float b = mymax(1.0f,2.0f);
double c = mymax(1.0,2.0);
std::string str = mymax(std::string("foo"),std::string("bar"));
```

OpenGL

- OpenGL is how we draw things on the screen
- Push vertex information to graphics card
  - Vertex positions
  - Colors
  - Normals
- Get pretty pictures
- Two main pipelines
  - Fixed Pipeline
  - Programmable Pipeline
    - Shader Programs
    - Rapidly Changing!

OpenGL 1.1 Fixed Pipeline State Machine



The OpenGL® Machine

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

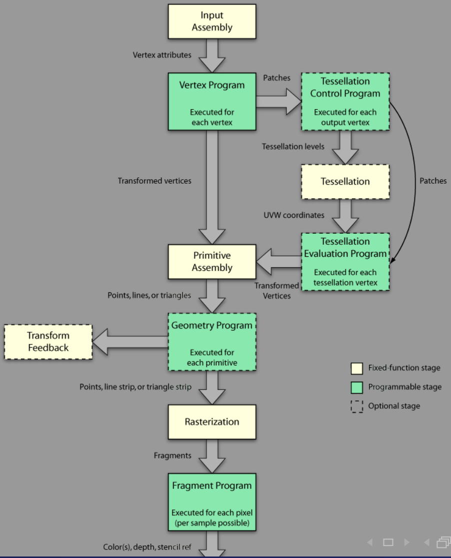
Vertices
Evaluators & Vertex Arrays
Input Conversion & Current Values
Texture Coordinate Generation
Lighting
Matrix Control
Clipping, Perspective, and Viewport Application
Feedback & Selection
Rasterization
Pixels
Texturing, Fog, and Antialiasing
Per-Fragment Operations
Frame Buffer & Frame Buffer Control
Primitives
Fragments
Frame Buffer

Key to OpenGL Operations

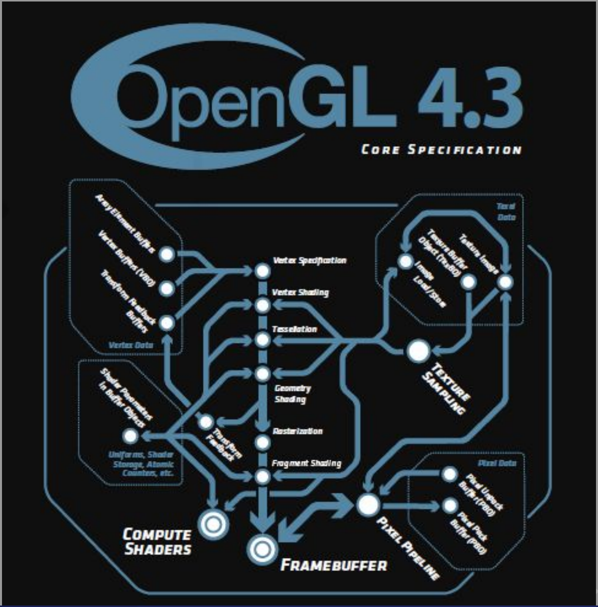
Notes:

1. Commands (and constants) are shown without the **gl** (or GL_) prefix.
2. The following commands do not appear in this diagram: **glAccum, glClearAccum**, **glHint**, display list commands, texture object commands, commands for obtaining OpenGL state (**glGet** commands and **glIsEnabled**), and **glPushAttrib** and **glPopAttrib**. Utility library routines are not shown.
3. After their exectution, **glDrawArrays** and **glDrawElements** leave affected current values indeterminate.
4. This diagram is schematic; it may not directly correspond to any actual OpenGL implementation.

OpenGL 4.0 / Direct3D 11 Programmable Pipeline Diagram



OpenGL 4.3 Programmable Pipeline Diagram



OpenGL is sort of a jumbled mess..

- Too many changes and subtle differences between versions
- We'll be sticking to the fixed pipeline
  - However, feel free to play with the programmable pipeline
- OpenGL comes with Core and Compatibility profiles, where Core removes fixed pipeline stuff
  - Have to manage your own Matrices
  - Graphics Cards are optimized for Compatibility
  - Few people use Core...

Useful OpenGL Tutorials

- Fixed Pipeline OpenGL
  - NeHe Tutorials
- Programmable Pipeline OpenGL
  - Wikibooks OpenGL
  - arcsynthesis.org/gltut
  - Mike Bailey's CS519 handouts and SIGGRAPH 2012 notes
- Both
  - Lighthouse3D
- Note: Tutorials tend to jump between different OpenGL specifications

OpenGL: The Fixed Pipeline

- What can we do?

  Assert State Information

```
glEnable(GL_DEPTH_TEST);
glDisable(GL_DEPTH_TEST);
glBegin(GL_QUADS);
glEnd();
glPushMatrix();
glTransformf(0.0,0.5,0.0);
glPopMatrix();
```

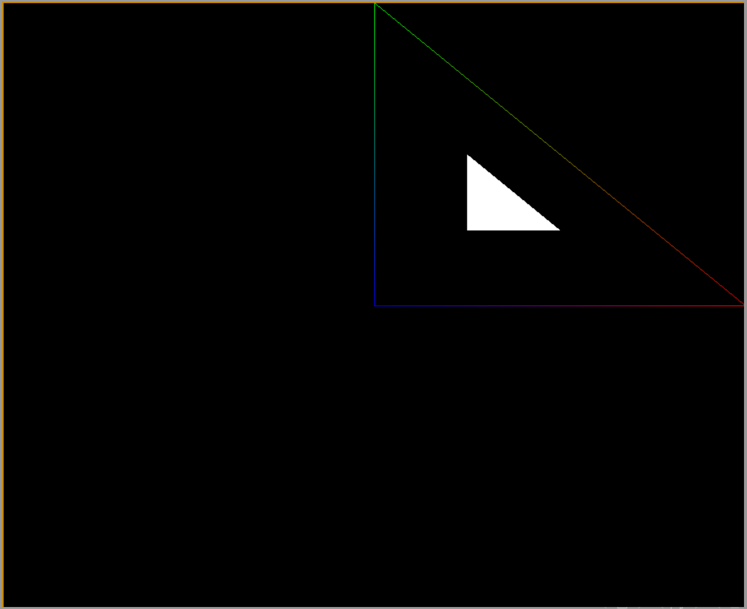
  Assert Vertex Information

```
glNormal3f(0.0f,1.0f,0.0f);
glColor4f(0.0f,0.0f,1.0f,1.0f);
glVertex3d(1.0,0.0,0.0);
```

Simple Example

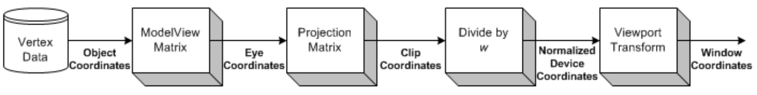
```
glBegin(GL_TRIANGLES);
    glColor4f(1.0f,1.0f,1.0f,1.0f);
    glVertex3d(.5,.25,0.0);
    glVertex3d(.25,.5,0.0);
    glVertex3d(.25,.25,0.0);
glEnd();
glBegin(GL_LINES);
    glColor3f(1.0f,0.0f,0.0f); glVertex3d(1.0,0.0,0.0);
    glColor3f(0.0f,1.0f,0.0f); glVertex3d(0.0,1.0,0.0);

    glColor3f(0.0f,1.0f,0.0f); glVertex3d(0.0,1.0,0.0);
    glColor3f(0.0f,0.0f,1.0f); glVertex3d(0.0,0.0,0.0);

    glColor3f(0.0f,0.0f,1.0f); glVertex3d(0.0,0.0,0.0);
    glColor3f(1.0f,0.0f,0.0f); glVertex3d(1.0,0.0,0.0);
glEnd();
```

Simple Example



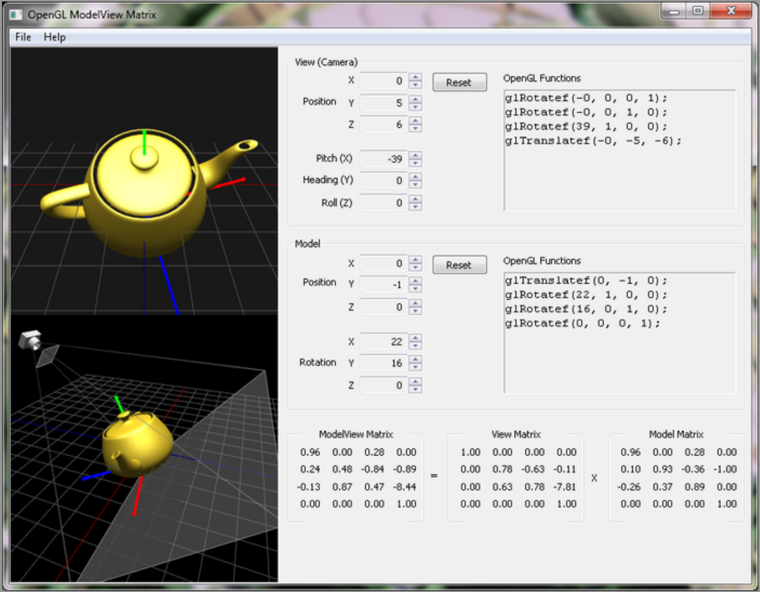
Transformations

- We want to traverse the scene and move scene objects around
- Use linear transformations in homogeneous coordinates:



- Fixed Pipeline maintains two matrices for you
  - Switch between modifying them with `glMatrixMode`
    - `GL_MODELVIEW`
    - `GL_PROJECTION`
- `http://www.songho.ca/opengl/gl_transform.html`

Transformations



View (Camera)

Position X 0
Position Y 5
Position Z 6

Pitch (X) -39
Heading (Y) 0
Roll (Z) 0

OpenGL Functions

```
glRotatef(-0, 0, 0, 1);
glRotatef(-0, 0, 1, 0);
glRotatef(39, 1, 0, 0);
glTranslatef(-0, -5, -6);
```

Model

Position X 0
Position Y -1
Position Z 0

Rotation X 22
Rotation Y 16
Rotation Z 0

OpenGL Functions

```
glTranslatef(0, -1, 0);
glRotatef(22, 1, 0, 0);
glRotatef(16, 0, 1, 0);
glRotatef(0, 0, 0, 1);
```

```
      ModelView Matrix                   View Matrix                       Model Matrix
 0.96   0.00   0.28   0.00       1.00   0.00   0.00   0.00        0.96   0.00   0.28   0.00
 0.24   0.48  -0.84  -0.89   =   0.00   0.78  -0.63  -0.11   x    0.10   0.93  -0.36  -1.00
-0.13   0.87   0.47  -8.44       0.00   0.63   0.78  -7.81       -0.26   0.37   0.89   0.00
 0.00   0.00   0.00   1.00       0.00   0.00   0.00   1.00        0.00   0.00   0.00   1.00
```

glMatrixMode s

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(left,right,bottom,top,nearVal,farVal);//option 1
glOrtho(left,right,bottom,top,nearVal,farVal);//option 2
gluPerspective(fovy, aspect, zNear, zFar);//option 3

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eyeX,eyeY,eyeZ,centerX,centerY,centerZ,upX,upY,upZ);
glRotatef(angle,x,y,z);
glTranslate(x,y,z);
glScale(x,y,z);
```

Hierarchical Matrix Stacks

- Fixed Pipeline stores a stack for both matrices
- This allows for rendering objects in a hierarchy to keep spacial coherency
- `glPushMatrix()`
- `glPopMatrix()`

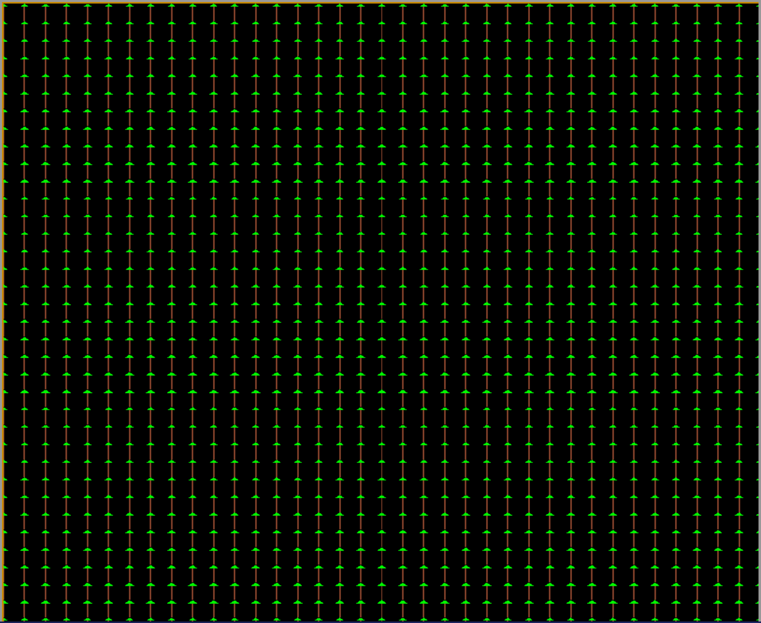
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glPushMatrix(); worldToHouseSpace();//house space
glPushMatrix(); houseToDoorSpace();//door space
glPushMatrix(); doorToDoornobSpace();//doornob space
renderDoornob();//doornob space
glPopMatrix();//Door space
renderFrame();//Door space
glPopMatrix()//house space
...render rest of house
```

Hierarchical Matrix Stacks

another example

```
void renderForest()
{
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    for(int i = 0; i<100; ++i)
    {
        glPushMatrix();
        glTranslatef(i,0.0,0.0);//Push ourselves to row i
        for(int j = 0; j<100; ++j)
        {
            glPushMatrix();
            glTranslatef(0.0,j,0.0);//Push ourselves to row j
            renderTree();//draw tree at position i,j
            glPopMatrix();
        }
        glPopMatrix();
    }
}
```

Forest



Questions?

Questions?