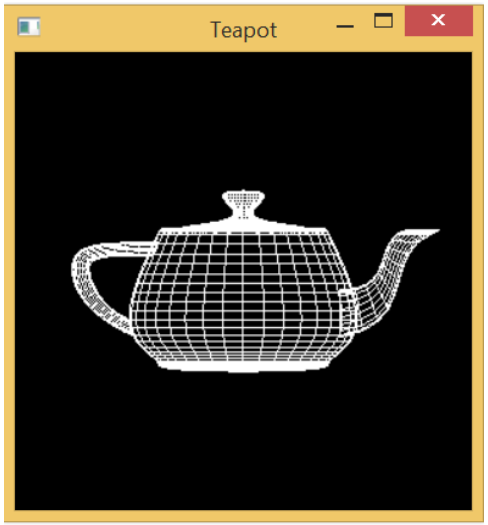
CS 451
Computer Graphics

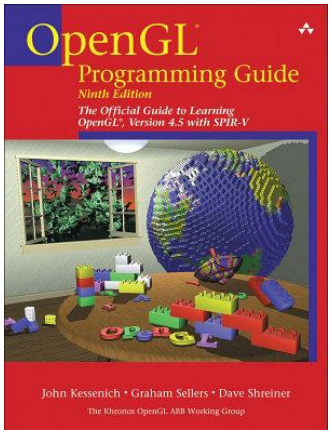
Yu Ji
yji8@gmu.edu

M&W: 12 – 13:15
MTB 1007

Lecture 5: OpenGL Basics

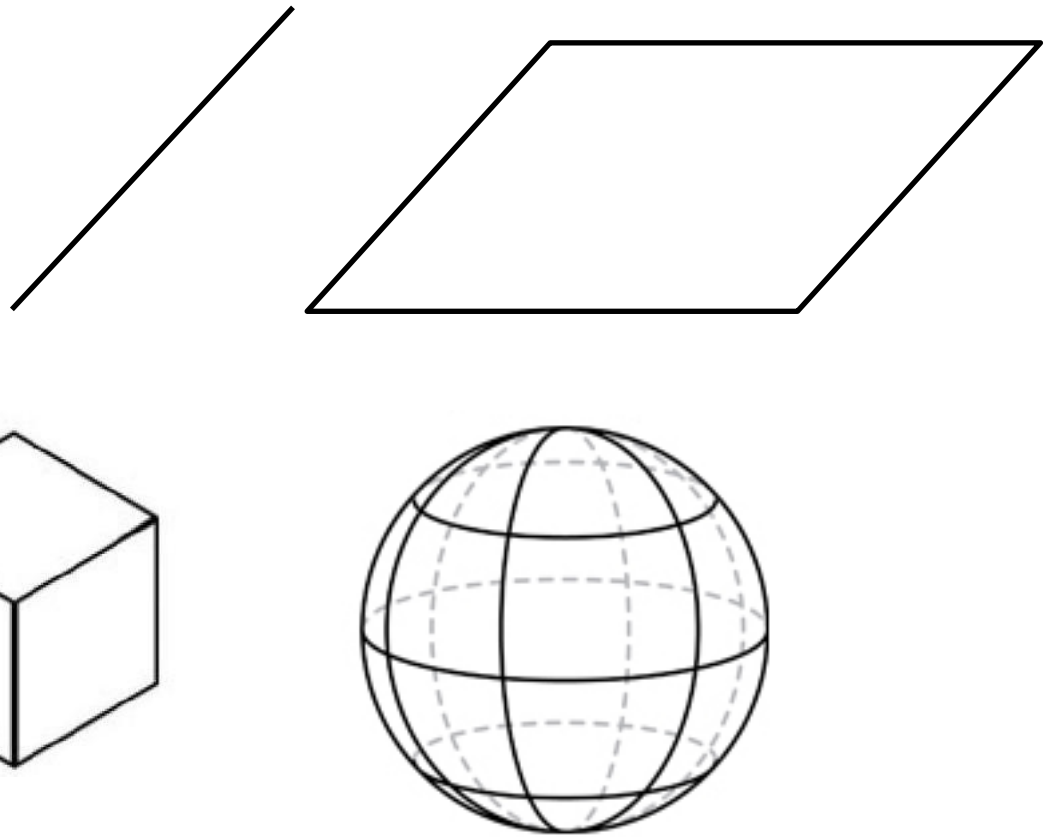
- Today let's do some drawing!
- What is OpenGL?
- How to write an OpenGL program?
- Reading:
  - OpenGL Programming Guide, 6th edition (for OpenGL 2.x)



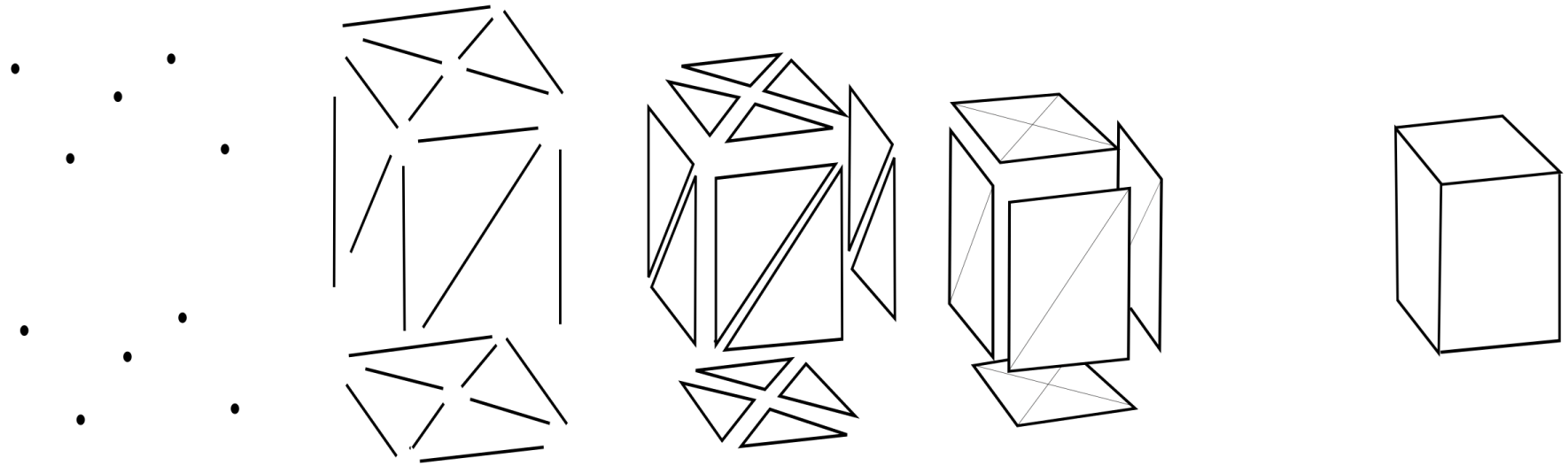


Build A 3D World

- What geometric primitives will you need?
  - Point
  - Line
  - Plane
  - Cube
  - Sphere
  - …

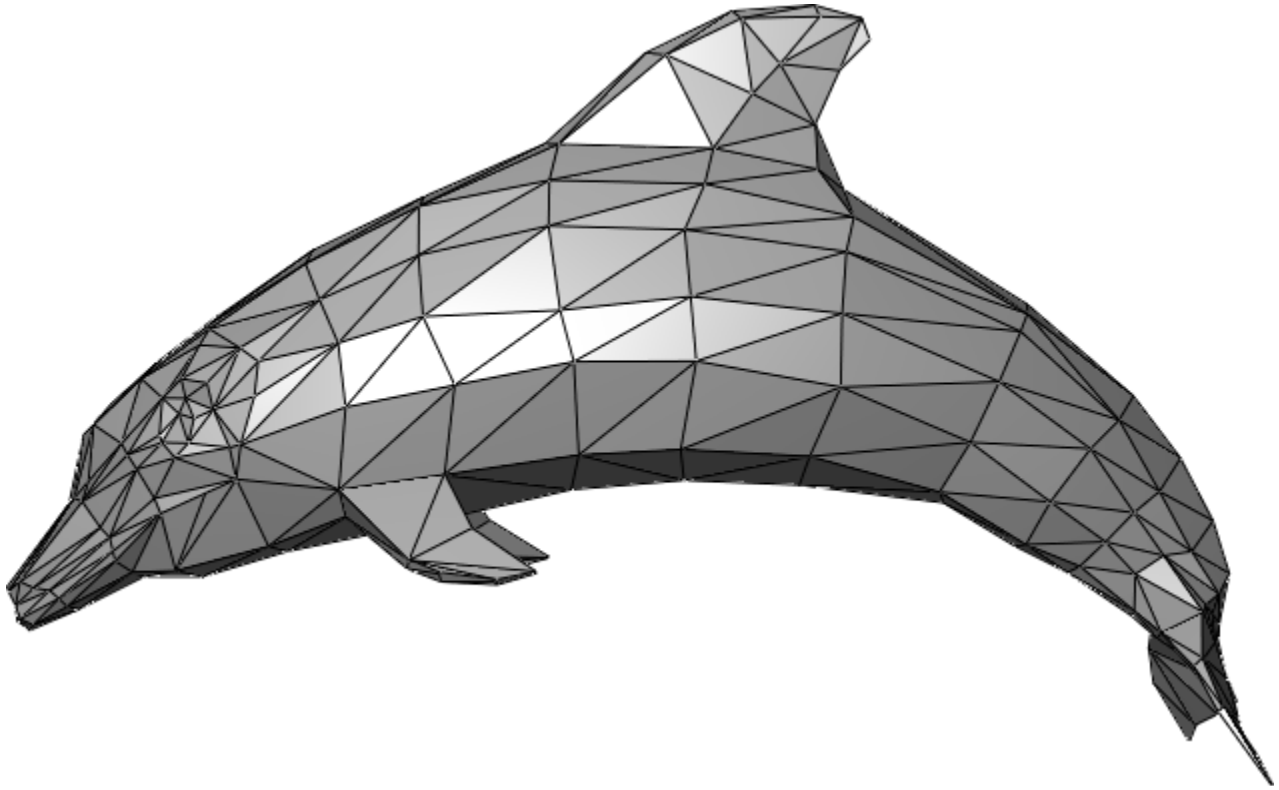


3D World Assembly



vertices edges faces polygons

3D Mesh with Triangles



3D Scene with Triangles

What is OpenGL?

- Simple API for 2D/3D graphics
  - Cross-platform 3D solution
  - Optimized for graphics card
- Libraries
  - GL (Graphics Library): 2D/3D drawing
  - GLU (GL Utilities) : camera setup and higher-level shape description
  - GLUT (GL Utilities Toolkit): utility functions dealing with windows and user interface

Other Useful Libraries

- GLUT replacements
  - Freeglut: open source and extended alternative to GLUT
  - http://freeglut.sourceforge.net/
  - GLFW: cross-platform windowing/UI toolkit
  - http://www.glfw.org/
- We will use Freeglut for our homework

History of OpenGL

- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware: Iris GL (1982)
- First version of OpenGL: an open source alternative to Iris GL (1992)
- Controlled by an Architectural Review Board (ARB)
  - Oversees changes in the language specification
  - Founder: SGI, Intel, IBM, DEC and Microsoft
  - Editor board of the Redbook

KHRONOS GROUP

OpenGL Evolution

- OpenGL is still young (born in 1992)
  - Cross-platform (Windows, Linux, Mac OS, Mobile OS …)
  - Easy to use and focus on rendering
  - Close to the hardware to get excellent performance
- Changes in the language have been slow
  - OpenGL 2.0 (2004)
  - OpenGL 3.0 (2008)
  - OpenGL 4.0 (2010)
  - Evolution reflects new hardware capacities

OpenGL Versions

- Classic OpenGL (2.x): Scene-based
  - Specify the objects, camera and lighting
  - Everything else handled for you
  - Supported on all non-mobile platforms
- New OpenGL (3.x and higher): shader program
  - Low-level control (vertex/fragment)
  - Shader programs are the primary focus
  - Allow customized graphics effects

OpenGL Versions

- Classic OpenGL (2.x): Scene-based
  - Specify the c lighting
  - Everything e
  - Supported o forms
- New OpenGL (3.x and higher): shader program
  - Low-level control (vertex/fragment)
  - Shader programs are the primary focus
  - Allow customized graphics effects

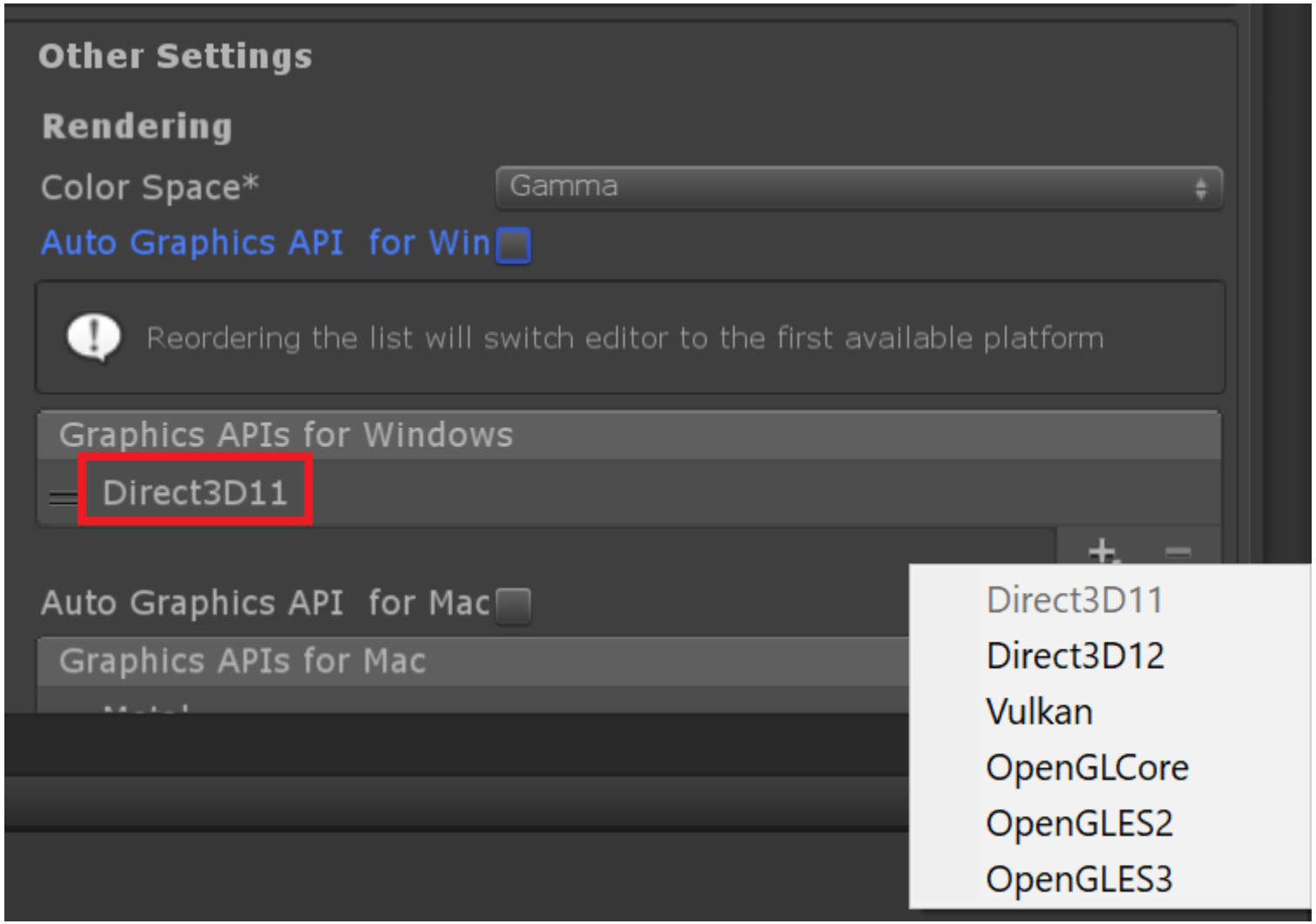
Good starting point for learning OpenGL

Other Modern Low-Level APIs

Initial release: February 2016
Programming language: C
Developer(s): Khronos Group

Initial release: September 1995
Programming language: C
Developer(s): Microsoft

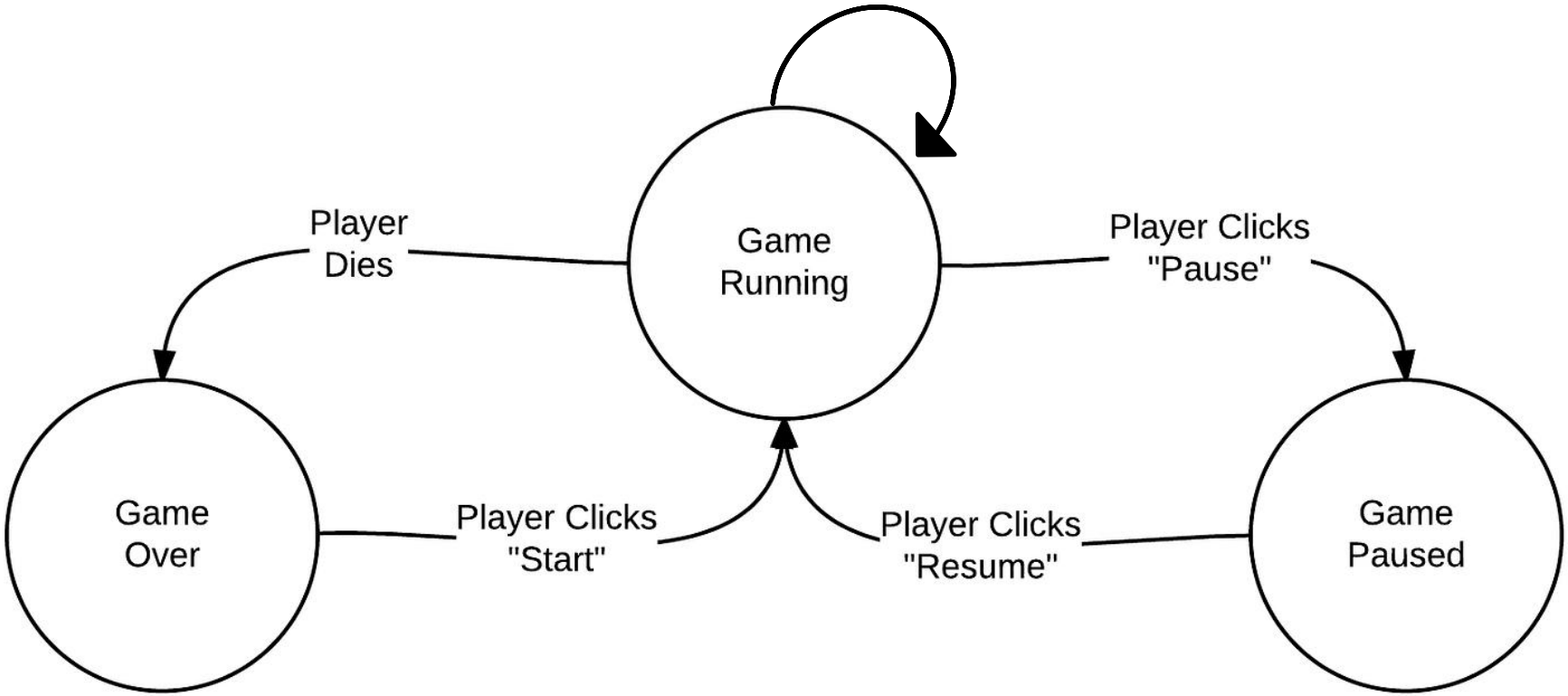
Programming languages: C++, Swift, Objective-C
Initial release date: June 2014
Developer(s): Apple Inc.


Other Settings
Rendering
Color Space*
Gamma
Auto Graphics API for Win
Reordering the list will switch editor to the first available platform
Graphics APIs for Windows
Direct3D11
Auto Graphics API for Mac
Graphics APIs for Mac
Direct3D11
Direct3D12
Vulkan
OpenGLCore
OpenGLES2
OpenGLES3

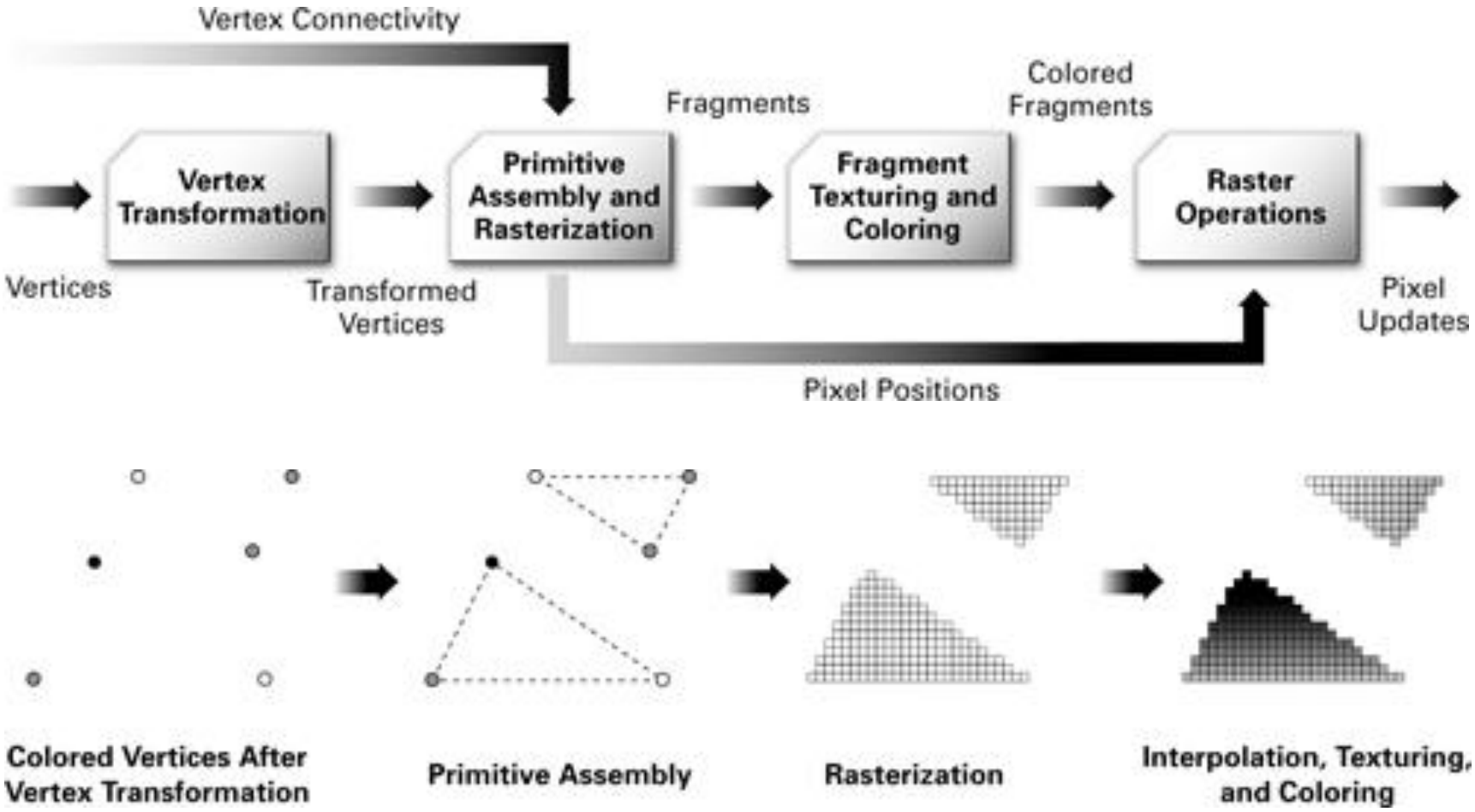
OpenGL: A State Machine

- State Machine: Remain in the current state until you change it
  - For example: color
- Each state has a default value
- Most Functions manipulate global state
  - Modify global state
  - Query global state
  - Cause something to be rendered

A State Machine Example (in Games)



Graphics Rendering pipeline



Images from Nvdia CG tutorial: http://developer.download.nvidia.com/CgTutorial/

Getting Started with OpenGL

- Windows OS: Visual Studio C++
  - https://www.visualstudio.com/vs/community/
- OpenGL is included in the graphics card driver (no installation required)
  - Only handles graphics rendering
- Need to install Freeglut
  - https://www.transmissionzero.co.uk/software/freeglut-devel/
  - Handles windowing, events, and UI

Getting Started with OpenGL

- Mac OS: Xcode
- Add OpenGL & GLUT frameworks
- No need to install Freeglut

Create Project

- Create a “Console Application”
- Create source file
  - Add New Item
  - Choose C++ file
- Configure Project Properties
  - Include additional header files and lib files

Simple_OpenGL.c

```
#include <GL/glut.h>        % use #include <GLUT/glut.h> if using Mac Xcode

void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT);

    glColor3f (1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();

glFlush ();
}

void init (void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

OpenGL Function Conventions

- Many functions have multiple forms:
  - glVertex3f,glVertex2i, glColor3f, glColor3ub, etc.
- Number indicates number of arguments
- Letters indicate type
  - f:float, d:double, i:integer, ub:unsigned byte
- “v” (if present) indicates a pointer argument
  - For example: glVertex3f (x,y,z)
    glVertex3fv(pointer)

Program Structure

- Classic OpenGL programs have the following structure:
  - `main()`
    - Opens one or more windows with the required properties
    - Register the callback functions
    - Enters event loop
  - `init()`
    - Sets the state variables (viewing, attributes)
  - callback functions
    - Input and display functions
    - For example: the display function

```
void display(void)
```

Simple_OpenGL.c

```
#include <GL/glut.h>
```

Include GLUT header glut.h (in Mac should be GLUT/glut.h)

```
void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT);

    glColor3f (1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();

glFlush ();
}

void init (void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Simple_OpenGL.c

```
#include <GL/glut.h>

void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT);

    glColor3f (1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();

glFlush ();
}

void init (void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Display function

Simple_OpenGL.c

```
#include <GL/glut.h>

void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT);

    glColor3f (1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();

glFlush ();
}

void init (void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Windows initialization

Simple_OpenGL.c

```
#include <GL/glut.h>

void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT);

    glColor3f (1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();

glFlush ();
}

void init (void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Main function

Main Function

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");

    init ();

    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}
```

Define window properties

Call init() to initialize OpenGL states

Display callback

Enter event loop

GLUT Functions

- `glutInit` allows application to get command line arguments and initializes system
- `glutInitDisplayMode` requests properties for the window (rendering context)
  - RGB color
  - Single Buffering
- `glutWindowSize` specifies size of window in pixels
- `glutWindowPosition` specifies position of the window from top-left corner of the display
- `glutCreateWindow` creates window with OpenGL context
- `glutDisplayFunc` display function callback
- `glutMainLoop` enters infinite event loop