COS426 Computer Graphics

Precept 2

Aleksey Boyko

February 18, 2011

Topic

Topics

- Getting started
- Initialization
- Drawing
- Transformations
  - Cameras
  - Animation
- Input
  - Keyboard
  - Mouse
  - Joystick?
- Textures
- Lights
- Programmable pipeline elements (shaders)

OpenGL

Open Graphics Library

- Industry standard
- Open source
- Cross-platform API
- 2D/3D graphics
- OpenGL ES – for mobile devices
- OpenGL 1.x
  - fixed rendering pipeline
- OpenGL 2+
  - Partially programmable pipeline via GLSL (OpenGL Shading Language)
- OpenGL 4.1 – as of July 2010
  - 5 programmable stages of pipeline

GLUT

Open**GL** **U**tility **T**oolkit

What

- Cross-platform system-level IO
  - Window definition
  - Window control
  - Monitoring keyboard/mouse input
- Drawing some primitives
  - Sphere
  - Cube
  - Utah teapot

GLUT

Why?

- OpenGL relies on application to provide platform dependent draw context
- GLUT provides means to make application code rather cross-platform
- Makes learning easier and starting faster

So Let's Start!

- OpenGL comes installed with most modern Operating systems
- Get glut
  - glut.h – to build ->SDKs/include
  - glut32.lib – to link -> SDKs/libs
  - glut32.dll – to run -> Windows/system32
- Other OS and IDE:
  - Install as described in COMPILER.txt in assignment 1
  - OR http://www.videotutorialsrock.com/opengl_tutorial/get_opengl_setup/home.php
  - OR start with your assignment 2 skeleton code and add/remove as necessary

Including headers

```
#if defined(_WIN32) || defined(__CYGWIN__)
   # include <windows.h>
   # include <GL/glut.h>
#elif defined(__APPLE__)
   # include <GLUT/glut.h>
#else
   # include <GL/glut.h>
#endif
```

Tip: can be found in assignment 1 skeleton code file cos426_opengl.h

Main method

```
void main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE |
    GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(320,320);
    glutCreateWindow("GLUT tutorial");

    glEnable(GL_DEPTH_TEST);

    glutDisplayFunc(renderScene);

    glutMainLoop();
}
```

void glutDisplayFunc(void (*func)(void))

Registers a redraw callback function with glut

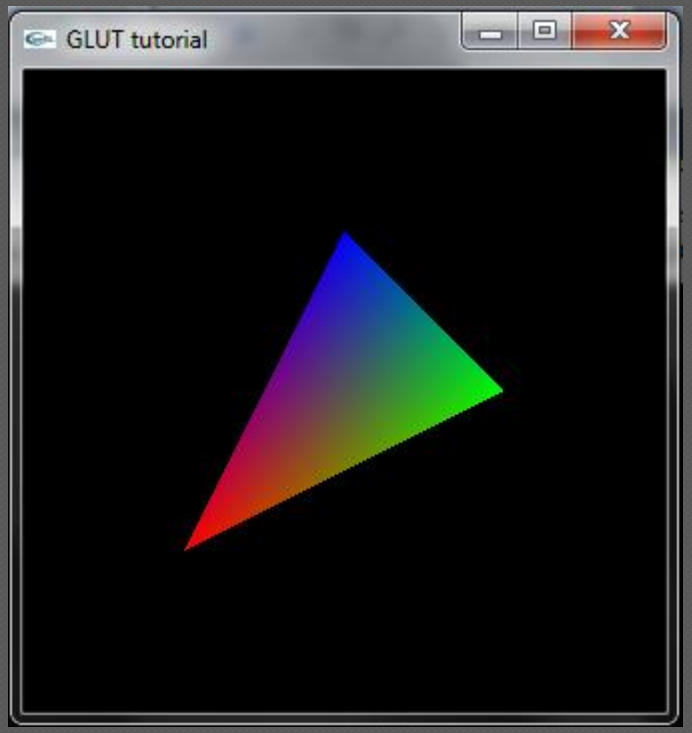
In main:

```
…
 glutDisplayFunc(renderScene);
…
```

We implement renderScene() before the main():

```
void renderScene(void)
{
     glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glBegin(GL_POLYGON);
            glColor3f(1, 0, 0);
            glVertex3f(-0.5,-0.5,0.0);
            glColor3f(0, 1, 0);
            glVertex3f(0.5,0.0,0.0);
            glColor3f(0, 0, 1);
            glVertex3f(0.0,0.5,0.0);
    glEnd();
    glFlush();
}
```

Result



glClear()

- GL_COLOR_BUFFER_BIT
  - Indicates the color buffer.
  - Color can be set with **glClearColor()**
- GL_DEPTH_BUFFER_BIT
  - Indicates the depth buffer.
  - Value can be set with **glClearDepth()**
- GL_STENCIL_BUFFER_BIT
  - Indicates the stencil buffer
  - Value can be set with **glClearStencil()**

- More here
http://www.khronos.org/opengles/documentation/opengles1_0/html/glClear.html

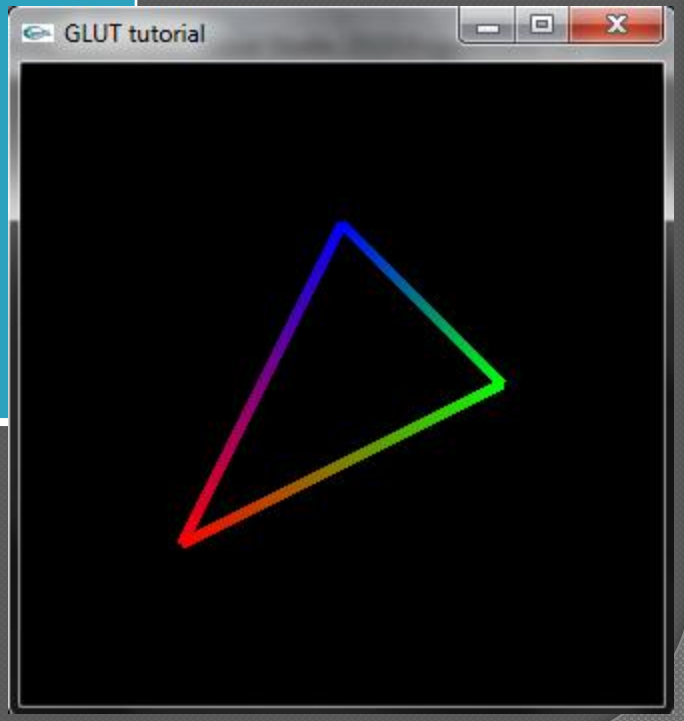
glBegin()

- GL_POLYGON
  - Draw filled convex polygon.
- GL_TRIANGLES
  - Pass *3n* vertices, draws *n* triangles.
- GL_QUADS
  - Pass 4n vertices, draws n quads.
- GL_LINES
  - Pass 2n vertices, draws n line segments.
- GL_LINE_LOOP
  - Draws outline of polygon.
- GL_LINE_STRIP
  - Draw segments (v1,v2), (v2,v3),…, (vn-1, vn).
- GL_POINTS
  - Pass n vertices, draws n points.
- For more, see glBegin man page, e.g. here
  http://www.opengl.org/sdk/docs/man/xhtml/glBegin.xml

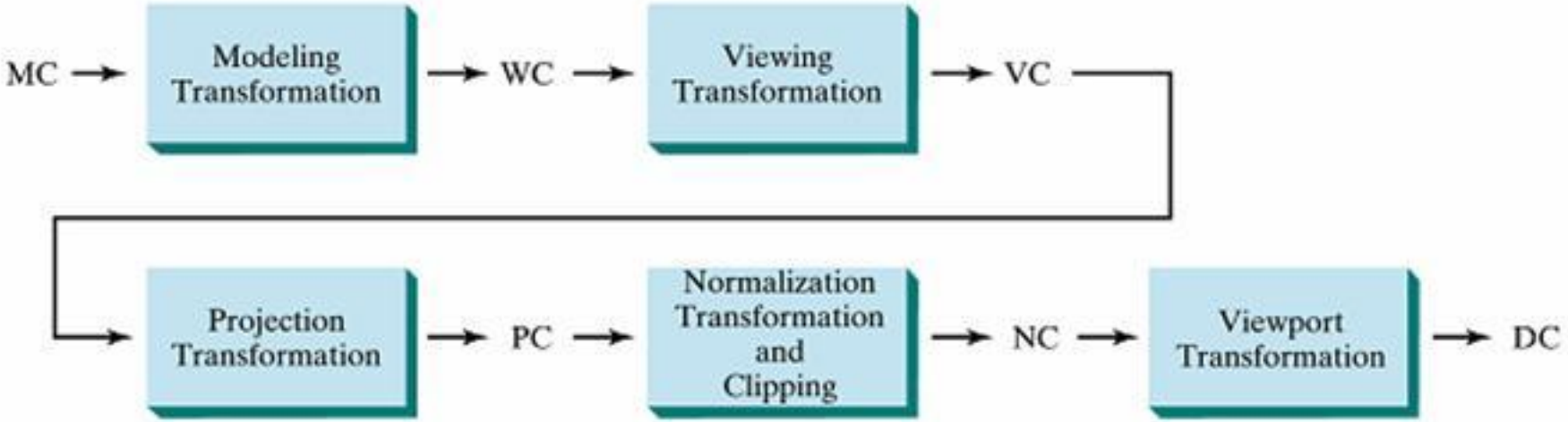
glBegin() example

```
glBegin(GL_POLYGON);
glColor3f(1, 0, 0);
glVertex3f(-0.5,-0.5,0.0);
glColor3f(0, 1, 0);
glVertex3f(0.5,0.0,0.0);
glColor3f(0, 0, 1);
glVertex3f(0.0,0.5,0.0);
glEnd();
```

```
glLineWidth(3);
//optional line width,
//default is 1
glEnable(GL_LINE_SMOOTH);
//enable antialiasing
glBegin(GL_LINE_LOOP);
glColor3f(1, 0, 0);
glVertex3f(-0.5,-0.5,0.0);
glColor3f(0, 1, 0);
glVertex3f(0.5,0.0,0.0);
glColor3f(0, 0, 1);
glVertex3f(0.0,0.5,0.0);
glEnd();
```



Transformation pipeline



General three-dimensional transformation pipeline, from modeling coordinates to world coordinates to viewing coordinates to projection coordinates to normalized coordinates and, ultimately, to device coordinates.

Transformations

- "Camera" – GL_PROJECTION matrix
  - Update the view perspective
    - when resizing the window
    - when user or event changes the "camera" position
- Scene – GL_MODELVIEW matrix
  - Hierarchical scene representation
  - Animation - Update the scene
    - with time
    - upon event
    - upon user input

OpenGL Transformations

- Vertex is transformed by
  - GL_MODELVIEW matrix,
  - then GL_PROJECTION matrix.
- After applying both matrices, vertex is in clip coordinates
- Divide by w component of vector to get normalized device coordinates (NDC).
- NDC: x, y, z values from -1 to 1.
- Initially, both matrices are identity.

OpenGL Transformations

- There is a stack of matrices for each matrix mode
- Current matrix is the matrix on top of the stack for each mode
- Initially stack has 1 element – an identity matrix
- glPushMatrix() – pushes a copy of current matrix on top of the stack
- glPopMatrix() – pops the stack, replacing the top matrix with the one below it

- **WHY?**
  - **Convenient for hierarchical coordinate systems, e.g.**
    - **articulated bodies**
    - **Scene graphs**

General matrix functions

- glMatrixMode()
  - switch active matrix stack
- glLoadIdentity()
  - replaces current matrix with an identity matrix
- glLoadMatrix(float M[16]) –
  - replace current matrix with M (column major)
- glMultMatrix(float M[16])
  - Multiply current matrix with M

Tip: if T is the current matrix and M is matrix we're multiplying by, the result is TM

Simple matrix manipulations

- glRotatef(degrees, x,y,z)
- glTranslatef(x,y,z)
- glScalef(x,y,z)

Scene animation example

- Register a callback – add to main():
  - glutIdleFunc(renderScene);

- Replace the renderScene() body with the following:
  - **Bold** font represents new code

```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

static float angle = 0.;
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glRotatef(angle,0,1,0);

glBegin(GL_POLYGON/*GL_LINE_LOOP*/);
    glColor3f(1, 0, 0);
    glVertex3f(-0.5,-0.5,0.0);
    glColor3f(0, 1, 0);
    glVertex3f(0.5,0.0,0.0);
    glColor3f(0, 0, 1);
    glVertex3f(0.0,0.5,0.0);
glEnd();

glPopMatrix();
glutSwapBuffers();
angle+=.05;
```

Swap buffers

- OpenGL supports several buffers
  - Front
  - Back
  - Left
  - Right
  - Permutations thereof
  - …
- To use this we did:
  - Added GLUT_DOUBLE as an argument to glutInitDisplayMode()
  - Called glutSwapBuffers() in the end of renderScene()

Camera

Our options are

- Change the entire scene through modifying bottom of the GL_MODELVIEW stack matrix
- Change the current GL_PROJECTION matrix
  - Using the previously shown matrix manipulations
    - glMatrixMode(GL_PROJECTION);
    - Call matrix manipulator functions
  - Helpful utilities from GL/glu.h

Camera with GLU

- #include <GL/glu.h> - just in case
- Describe the camera (either one)
  - gluPerspective()
  - gluOrtho2D()
- Point the camera
  - gluLookAt()

gluPerspective

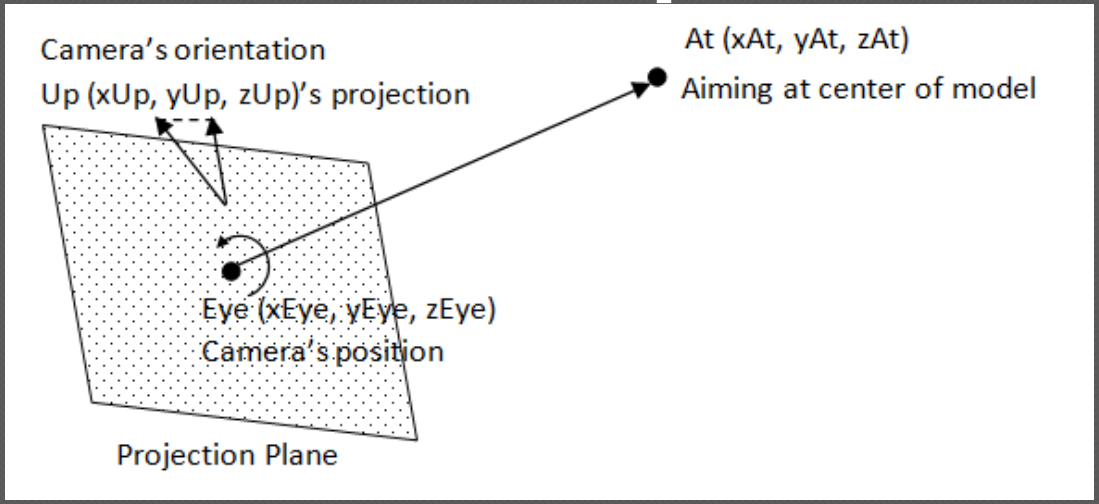
- Sets up a perspective projection matrix
  - fovy
    - field of view angle in camera's y direction
  - aspect
    - aspect ratio (x/y)
  - znear
    - Distance from the camera to the near clipping plane
  - zfar
    - Distance from the camera to the far clipping plane

gluOrtho2D

- Set up a 2D orthographic projection matrix
  - left, right
    - Coordinates of the left and right clipping planes
  - bottom, top
    - Coordinates of the bottom and top clipping planes

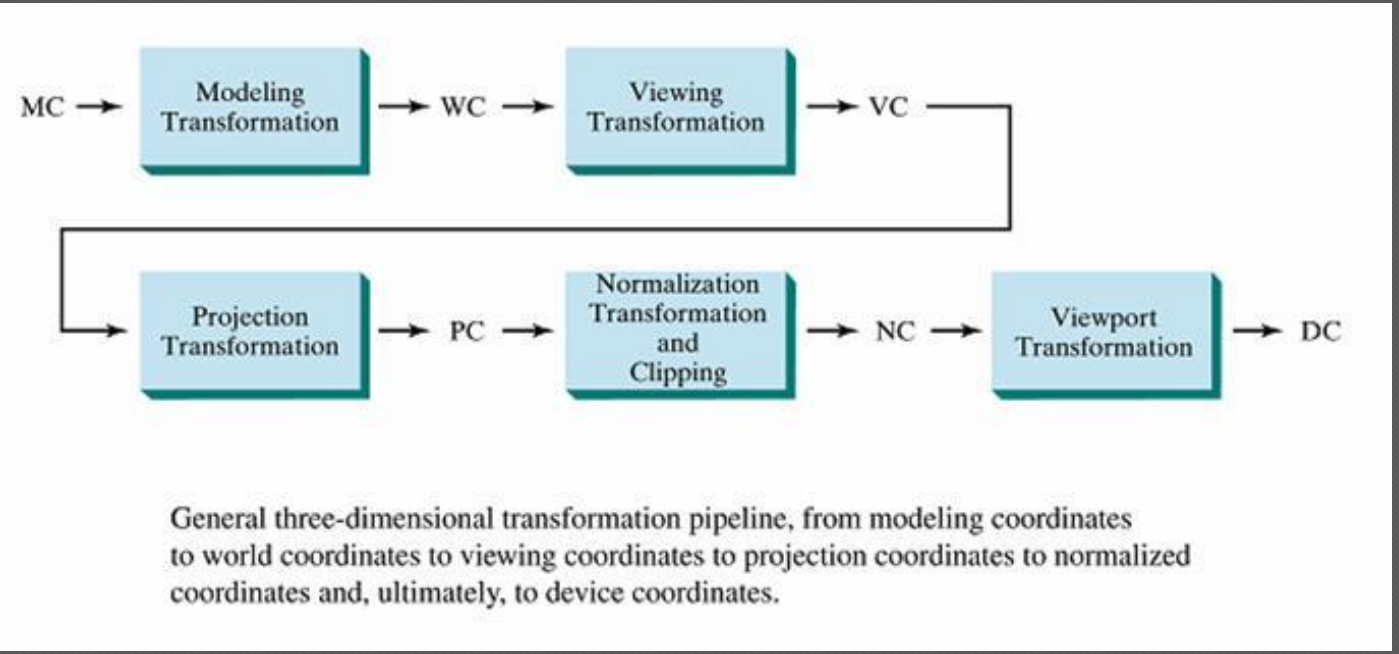
gluLookAt

- Defines a viewing transformation (points the camera)
  - eyeX, eyeY, eyeZ
    - Specifies the position of the "eye" point
  - centerX, centerY, centerZ
    - Specifies the position of the reference point
  - upX, upY, upZ
    - Specifies the direction of the "up" vector



glViewport

- Defines the viewport transformation
  - Specifies transformation NDC to window coordinates: glViewport()



General three-dimensional transformation pipeline, from modeling coordinates to world coordinates to viewing coordinates to projection coordinates to normalized coordinates and, ultimately, to device coordinates.

Resize window example

- Register a callback -add to main():
  - glutReshapeFunc(changeSize);
- Add above the main():

```
void changeSize(int w, int h)
{
    // Prevent a divide by zero, when window is too short
    // (you cant make a window of zero width).
    if(h == 0) h = 1;

    float ratio = 1.0* w / h;

    // Reset the coordinate system before modifying
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Set the viewport to be the entire window
    glViewport(0, 0, w, h);

    // Set the correct perspective.
    gluPerspective(45,ratio,1,1000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0,0.0,5.0, 0.0,0.0,-1.0, 0.0f,1.0f,0.0f);
}
```

Drawing in 3D - Depth test

- To avoid drawing in order (painter's algorithm) we do:
  - Add GLUT_DEPTH to arguments in the glutInitDisplayMode call
    - Creates a z-buffer
  - glEnable(GL_DEPTH_TEST);
    - Enables depth comparison and update of z-buffer

Drawing in 3D – Primitive Shapes

- GLU:
  - gluSphere()
  - gluCylinder()
  - gluDisk()
  - …
- GLUT:
  - Solids
    - glutSolidTorus()
    - glutSolidTeapot()
    - …
  - Wireframes
    - glutWireIcosahedron()
    - glutWireTeapot()

Topics for future discussions

- Getting started
- Initialization
- Drawing
- Transformations
  - Cameras
  - Animation
- Input
  - Keyboard
  - Mouse
  - Joystick?
- Textures
- Lights
- Programmable pipeline elements (shaders)

References

Code from this precept:
http://www.cs.princeton.edu/courses/archive/spr11/cos426/precepts/GlutTest.zip

More tutorials (partly used in the presentation):
http://www.lighthouse3d.com/opengl/glut
http://nehe.gamedev.net/
http://www.videotutorialsrock.com/

OpenGL quick reference:
http://www.khronos.org/files/opengl4-quick-reference-card.pdf