OpenGL & Glut
Part II: Viewing

COSC 4431/5331
Computer Graphics

Bill Kapralos

Thursday January 22, 2004

Overview (1):

- **Getting Started**
  - Setting up OpenGL/GLUT on Windows/Visual Studio
- **Viewing Overview**
  - Introduction
    - Camera analogy
    - Matrix operations and OpenGL
- **Viewing Transformation Details**
  - Viewing transformation
  - Projection transformation
  - Viewport transformation

Getting Started

Getting Started (1):

- **Installing GLUT**
  - Download GLUT from Nate Robin's web site
    - http://www.xmission.com/~nate/glut.html
    - Download: glut-3.7.6-bin.zip (117 KB)
  - Three files of interest: *glut32.dll, glut32.lib & glut.h*
    - Place them in following directories:

```
glut32.dll to Windows\System,
glut32.lib to \VC98\lib,
glut.h     to \VC98\include\GL.
```

VC98 directory is in the Visual Studio directory:
C:\Program Files\Microsoft Visual Studio\VC98

Getting Started (2):

- **Microsoft Windows (XP) & Visual Studio (C++)**
  - OpenGL included in newer versions of Windows OS
  - Compiling and linking – After creating project
    - From menu bar, go to

      "Project -> Settings -> … Link"
    - Append the following string to the existing "Objects/Library Modules" string

      `"opengl32.lib glu32.lib glut32.lib"`
    - Build & execute program

Getting Started (3):

- **Include Libraries**
  - For all OpenGL applications, include gl.h in every file
  - Almost all OpenGL applications use GLU, so include glu.h as well
  - If using Glut, you also need glut.h
  - OpenGL source file typically begins with

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

  - But glut.h includes the gl.h and glu.h so really only glut.h is needed!

Getting Started (4):

- **Setting Up Project in Visual Studio/C++**

1. Load Visual Studio/C++
2. File->New a dialog box will appear - choose "Win 32 Console Application", give the project a name and press "OK"
3. Another dialog box will then appear: choose "A Simple Application" and click "Finish"
4. Your new project workspace will now be available and on the screen - add the three libraries as previously described
5. All necessary files etc. will be generated in addition to the file containing "main" method – this is entry point

7

Viewing Overview

8

Introduction to Viewing (1):

- **Summary of Transformations**
  - Viewing
    - Position viewing volume in the world
  - Modeling
    - Position models in the real world
  - Projection
    - Determine shape of viewing volume
  - Viewport
    - Draw final "image" to display window

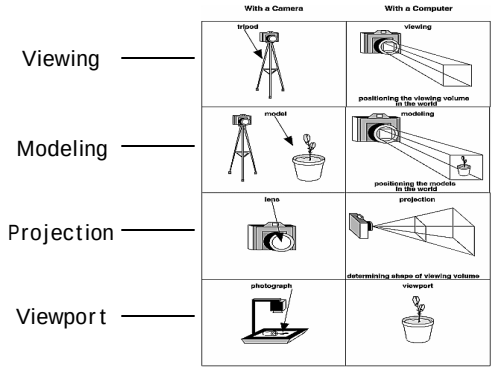
9

Introduction to Viewing (2):

- **Camera Analogy**
  - Viewing
    - Set-up tripod
    - Point camera at the scene
  - Modeling
    - Arrange the scene to be photographed
  - Projection
    - Choose desired camera lens and zoom
  - Viewport
    - Determine how large final photograph will be

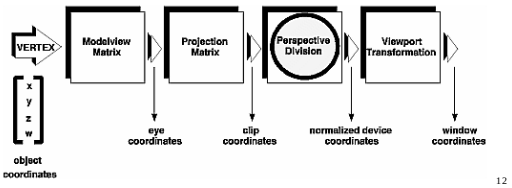
10

Introduction to Viewing (3):



Introduction to Viewing (4):

- **Stages of Vertex Transformations**
  - 4 x 4 matrix M is used to specify viewing, modeling and projection transformations
  - Transformation is accomplished by multiplying coordinates of each vertex *v* in scene by M ? *v'* = *Mv*



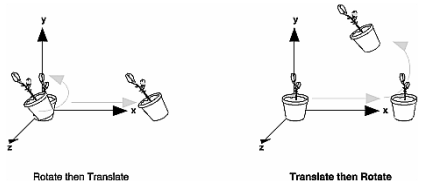
12

Introduction to Viewing (5):

- **Transformation Matrices**
  - Homogenous coordinates e.g. [x, y, z, w]
    - w is typically equal to 1
  - Modelview Matrix
    - Combined viewing and modeling transformations
    - Convert "object" coordinates in world to (viewer) eye coordinates

Introduction to Viewing (6):

- **Transformation Matrices (cont…)**
  - Ordering of Transformations is important
    - Rotation followed by translation is not necessarily equivalent to translation followed by rotation!
    - Matrix multiplication: ML not always equal to LM



Introduction to Viewing (7):

- **Current Matrix**
  - State Variable
  - Single matrix used to perform transformations
    - Modelview, projection & texture transformations
  - Transformations are applied to current matrix
    - Vertices multiplied by current matrix
  - Warning ? Transformations are accumulative
  - Typically need to "reset" current matrix prior to performing transformation

```
glLoadIdentity()
```


Introduction to Viewing (8):

- **Current Matrix (cont…)**
  - Can specify which of the three matrices becomes the current matrix using glMatrixMode()

```
glMatrixMode(matrix)
```

  - *matrix* ? GL_MODELVIEW, GL_PROJECTION,GL_TEXTURE

```
glMatrixMode(GL_MODELVIEW)
glMatrixMode(GL_PROJECTION)
glMatrixMode(GL_TEXTURE)
```

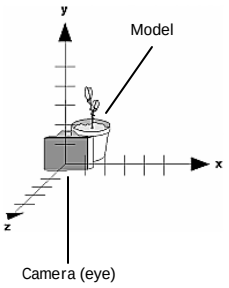

Introduction to Viewing (9):

- **Notes Regarding Transformations**
  - Window Coordinates
    - Obtained after applying the viewport transformation
    - Coordinates relative to display window
- **Transformations Assumptions**
  - Requires some knowledge of linear algebra (matrices)

Viewing Details

Viewing Transformations (1):

- **Viewing Transformation**
  - Changes direction and orientation of viewpoint or *eye*
  - Default location
    - Origin (0,0,0)
    - Pointing (looking) down –z axis



19

Viewing Transformations (2):

- **Viewing Transformation (cont…)**
  - Several ways to change viewing position/direction
    1. Use modeling transformation commands: glRotate() and glTranslate()
    2. GLU routine: gluLookAt()
    3. Create your own "utility routine" which encapsulates rotations and translations

20

Viewing Transformations (3):

- **Transformations in OpenGL**
  - Translation: `glTranslate(x, y, z)`
    - Multiplies current matrix that moves object by the given x,y,z values
  - Rotation: `glRotation(angle, x, y, z)`
    - Multiplies current matrix that rotates object in counter-clockwise direction about ray from origin through x,y,z
  - Scale: `glScale(x, y, z)`
    - Stretches, shrinks or reflects object along axis

21

Viewing Transformations (4):

- **Using Modeling Transformations**
  - Rather than moving camera (or viewer, eye), move the model while leaving camera at default position
    - Same effect – as it's the position of camera relative to model that's of interest
    - e.g. rather than moving camera backwards, 5 units, from objects (model), move objects forward from camera by 5 units

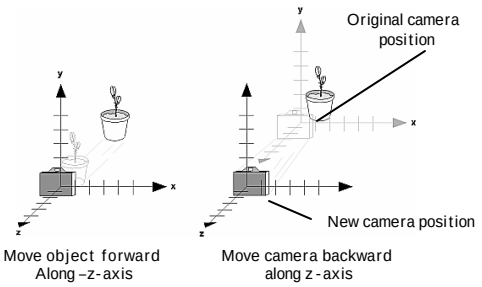
```
glTranslate(0.0, 0.0, -5.0)
```

    - **Remember** ? Forward is down –z axis!

22

Viewing Transformations (5):

- **Using Modeling Transformations - Illustration**



23

Viewing Transformations (6):

- **gluLookAt()**
  - Three sets of arguments (all of type GLdouble):
    - Eye coordinates (x,y,z)
    - Point to be viewed along line of sight (x,y,z)
    - Orientation vector – which direction is UP (x,y,z)

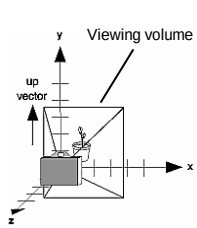
```
gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY,
  centerZ, upX, upY, upZ);
```

  - Default Settings
    - Eye (camera) at origin
    - Looking down –z axis, positive y-axis straight up

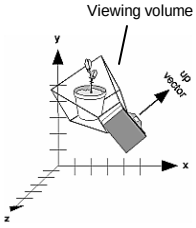
24

Viewing Transformations (7):

- **gluLookAt() Examples**



Default



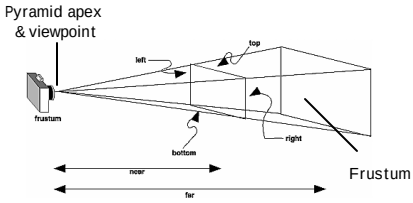
gluLookAt(4,2,1, 2,4,-3, 2,2,-1);

Projection Transformations (1):

- **Projection Transformations**
  - Purpose ? define a viewing volume which:
    1. Determines how object is projected onto screen ? either perspective or orthographic projection
    2. Defines which objects or portions of objects are "clipped" (removed) from final image

Projection Transformations (2):

- **Perspective Projection**
  - Viewing volume is a *frustum*
    - Truncated pyramid with top cut off
    - Six planes: left, right, bottom, top, near, far



Projection Transformations (3):

- **Perspective Projection (cont…)**
  - Objects falling within frustum are projected towards viewpoint (pyramid apex)
  - Objects closer to view point occupy larger amount of viewing volume
    - Foreshortening ? the farther an object from camera, the small it appears in final image
    - Similar to how our eyes work
  - Two ways to set up in OpenGL
    1. gluFrustum()
    2. gluPersepctive()

Projection Transformations (4):

- **glFrustum()**

```
glFrustum(left, right, bottom, top, near, far);
```

  - Viewing volume defined by coordinates:
    - (left, bottom, -near) ? lower left (x,y,z) coordinates
    - (right, top, -far) ? upper right (x,y,z) coordinates

Projection Transformations (5):

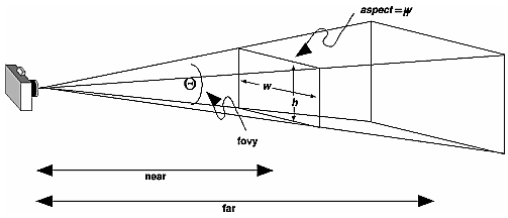
- **gluPerspective()**

```
glFrustum(fovy, aspect, near, far);
```

  - *fovy* ? Field of view angle in the yz plane [0 – 180º]
  - *aspect* ? Aspect ratio (width/height)
  - *near*, *far* ? distance to near and far planes from viewpoint down –z-axis!!

Projection Transformations (6):

- **gluPerspective() Graphical Illustration**



Projection Transformations (7):

- **Orthographic Projection**
  - Viewing volume is a rectangular parallelepid (a box)
    - Size of viewing volume doesn't change from one end to other
    - Distance from camera is irrelevant to size of projected object
  - Used for blueprint drawings, CAD and applications where object dimensions are important
  - Two ways to set up in OpenGL
    1. glOrtho()
    2. gluOrtho2D()

Projection Transformations (8):

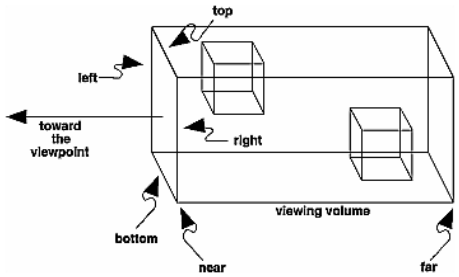
- **glOrtho()**

```
glOrtho(left, right, bottom, top, near, far);
```

  - Viewing volume defined by coordinates:
    - (left, bottom, -near) & (right, top, -near) ? are on near clipping plane & mapped to lower left and upper right viewport window respectively
    - (left, bottom, -far) & (right, top, -far) ? are on far clipping plane & also mapped to lower left and upper right viewport window respectively

Projection Transformations (9):

- **glOrtho() Graphical Illustration**



Projection Transformations (10):

- **gluOrtho2D()**

```
glOrtho(left, right, bottom, top);
```

  - Use with 2D scene onto a 2D screen only!
  - Coordinates of rectangular clipping region
    - Left, right, bottom, top

Viewport Transformations (1):

- **Chooses "Size" of Final Image on Screen**
  - Defines the rectangle in the window which final image is placed

```
glViewport(x, y, width, height);
```

  - Integer argument types (Glint)
    - *x, y* ? lower left corner of viewport
    - width, height ? size of the viewport rectangle
  - Default:
    - (0, 0, winWidth, winHeight) e.g. entire window

Viewport Transformations (2):

- **Some Notes**
  - Aspect ratio of viewport should be equal to aspect ratio of viewing volume otherwise final image will be distorted!
  - Remember ? window re-sizing may require re-setting of viewport!

37

Viewport Transformations (3):

- **Distortion Example**
  - Normal Viewing (square window 400 x 400)

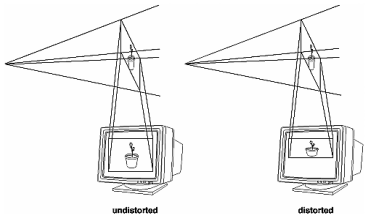
```
gluPerspective(fovy, 1.0, near, far);
glViewport(0, 0, 400, 400);
```

  - Distorted View:
    - Resized window to a non-equilateral rectangular viewport, 400 x 200
    - Projection remains the same (un-changed)

```
gluPerspective(fovy, 1.0, near, far);
glViewport(0, 0, 400, 200);
```

38

Viewport Transformations (4):



- Corrected View:
  - Modify aspect ratio to match viewport

```
gluPerspective(fovy, 2.0, near, far);
glViewport (0, 0, 400, 200);
```

39

Putting it All Together (1):

- **Notes Regarding Viewing Transformations**
  - Modelview transformations typically specified when drawing/re-drawing the scene
    - Display callback function! For example:

```
void display(void) {
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (1.0, 1.0, 1.0);
  glLoadIdentity (); /* clear the matrix */
  /* viewing transformation */
  gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  glScalef (1.0, 2.0, 1.0); /* modeling transformation */
  glutWireCube (1.0);
  glFlush ();
}
```

40

Putting it All Together (2):

- **Notes Regarding Viewing Transformations**
  - Projection and viewport typically specified when window is initially created/re-seized etc…
    - Window re-shape callback function! For example:

```
void reshape (int w, int h) {
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
  glMatrixMode (GL_MODELVIEW);
}
```

41

Putting it All Together (3):

- **Final Notes**
  - Many problems you encounter are probably due to incorrect viewing set-up
  - Ensure objects (model) are within viewing volume – remember, near/far planes are down –z axis
  - For example: if near and far are 1 and 3 respectively, make sure objects are within 1 and 3 as well
    - Try temporarily setting near and far planes to 0.0001 and 100000

42