Lecture 4 of 41

Lab 1a: OpenGL Basics

William H. Hsu
Department of Computing and Information Sciences, KSU

KSOL course pages: http://snipurl.com/1y5gc
Course web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:
Today: Chapter 2 (review), Chapter 16, Eberly 2e – see http://snurl.com/1ye72
Next class: Section 2.3 (esp. 2.3.7), 2.6, 2.7, Eberly 2e
Angel, OpenGL: A Primer, 3e
This week: FVFH slides on Viewing
Lecture Outline

- Reading for Last Class: §2.3 (esp. 2.3.4), Eberly 2e; Foley et al. Slides
- Reading for Today: Chapters 2, 16, Eberly 2e; Foley et al. Slides
- Reading for Next Class: §2.3 (esp. 2.3.7), 2.6, 2.7, Eberly 2e
- Last Time: Matrix Stack for 3-D Viewing Transformation
  - \( \mathbf{N} = \mathbf{D}_{\text{Persp}} \mathbf{S}_{\text{Far}} \mathbf{S}_{xy} \mathbf{M}_{\text{Rot}} \mathbf{T}_{\text{Trans}} \)
  - Perspective: optical principles, terminology
- Today: Highlights from First of Three Tutorials on OpenGL (Three Parts)
  - 1. OpenGL and GL Utility Toolkit (GLUT) – V. Shreiner
  - 2. Basic rendering – V. Shreiner
  - 3. 3-D viewing setup – E. Angel
- Next Class: Scan Conversion (Rasterization) of Lines, Polygons
## Where We Are

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Topic</th>
<th>Primary Source(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Course Overview</td>
<td>Chapter 1, Eberty 2nd</td>
</tr>
<tr>
<td>1</td>
<td>CG Basics: Transformation Matrices; Lab 0</td>
<td>Sections (s) 2.1, 2.2</td>
</tr>
<tr>
<td>2</td>
<td>Viewing 1: Overview, Projections</td>
<td>§ 2.2.3 – 22.4, 2.8</td>
</tr>
<tr>
<td>3</td>
<td>Viewing 2: Viewing Transformation</td>
<td>§ 2.3 esp. 2.3.4; PFH slides</td>
</tr>
<tr>
<td>4</td>
<td>Lab 1A: Flash &amp; OpenGL Basics</td>
<td>Ch. 2.16; Angel Primer</td>
</tr>
<tr>
<td>5</td>
<td>Viewing 3: Graphics Pipeline</td>
<td>§ 2.3 esp. 2.3.7, 2.6, 2.7</td>
</tr>
<tr>
<td>6</td>
<td>Scan Conversion 1: Lines, Midpoint Algorithm</td>
<td>§ 2.5.1, 3.1; PFH slides</td>
</tr>
<tr>
<td>7</td>
<td>Viewing 4: Clipping &amp; Culling; Lab 1b</td>
<td>§ 2.3.5, 2.4, 3.1.3</td>
</tr>
<tr>
<td>8</td>
<td>Scan Conversion 2: Polygons, Clipping Intro</td>
<td>§ 2.4, 2.5 esp. 2.5.4, 3.1.6</td>
</tr>
<tr>
<td>9</td>
<td>Surface Detail 1: Illumination &amp; Shading</td>
<td>§ 2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2</td>
</tr>
<tr>
<td>10</td>
<td>Lab 2A: Direct3D / DirectX Intro</td>
<td>§ 2.7, Direct3D handout</td>
</tr>
<tr>
<td>11</td>
<td>Surface Detail 2: Textures; OpenGL Shading</td>
<td>§ 2.6.3, 20.3 – 20.4, Primer</td>
</tr>
<tr>
<td>12</td>
<td>Surface Detail 3: Mappings; OpenGL Textures</td>
<td>§ 20.5 – 20.13</td>
</tr>
<tr>
<td>13</td>
<td>Surface Detail 4: Pixel/Vertex Shading; Lab 2b</td>
<td>§ 3.1</td>
</tr>
<tr>
<td>14</td>
<td>Surface Detail 5: Direct3D Shading; OGLSL</td>
<td>§ 3.2 – 3.4, Direct3D handout</td>
</tr>
<tr>
<td>15</td>
<td>Demos 1: CGA, Fun; Scene Graphs: State</td>
<td>§ 4.1 – 4.3, CGA handout</td>
</tr>
<tr>
<td>16</td>
<td>Lab 3A: Shading &amp; Transparency</td>
<td>§ 2.6, 20.1, Primer</td>
</tr>
<tr>
<td>17</td>
<td>Animation 1: Basics, Keyframes; HW/Exam</td>
<td>§ 5.1 – 5.2</td>
</tr>
<tr>
<td>18</td>
<td>Exam 1 review: Hour Exam 1 (evening)</td>
<td>Chapters 1 – 4, 20</td>
</tr>
<tr>
<td>19</td>
<td>Scene Graphs: Rendering; Lab 3b: Shader</td>
<td>§ 4.4 – 4.7</td>
</tr>
<tr>
<td>20</td>
<td>Demos 2: SFX: Skinning, Morphing</td>
<td>§ 5.3 – 5.5, CGA handout</td>
</tr>
<tr>
<td>21</td>
<td>Demos 3: Surfaces, B-reps/Volume Graphics</td>
<td>§ 10.4, 12.7, Mesh handout</td>
</tr>
</tbody>
</table>

Lightly-shaded entries denote the due date of a written problem set, heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review, and the green-shaded entry, that of the term project.

Green, blue and red letters denote exam review, exam, and exam solution review dates.
Review:
Viewing Transformation

- **Placement** of view volume (visible part of world) specified by camera’s position and orientation
  - Position (a point)
  - Look and Up vectors
- **Shape** of view volume specified by
  - horizontal and vertical view angles
  - front and back clipping planes
- Perspective projection: projectors intersect at Position
- Parallel projection: projectors parallel to Look vector, but never intersect (or intersect at infinity)
- Coordinate Systems
  - **world coordinates** – standard right-handed xyz
  - **camera coordinates** – camera-space right handed coordinate system
    - origin at Position and axes rotated by orientation; used for transforming arbitrary view into canonical view

https://bit.ly/hiSt0f

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
Review:
CTM for "Polygons-to-Pixels" Pipeline

- Entire problem can be reduced to a composite matrix multiplication of vertices, clipping, and a final matrix multiplication to produce screen coordinates.
- Final composite matrix (CTM) is composite of all modeling (instance) transformations (CMTM) accumulated during scene graph traversal from root to leaf, composites with the final composite normalizing transformation $N$ applied to the root/world coordinate system:
  1) $N = D_{persp} S_{for} S_{xy} M_{rot} T_{trans}$
  2) $CTM = N \cdot CMTM$
  3) $P' = CTM \cdot P$ for every vertex $P$ defined in its own coordinate system
  4) $P_{\text{screen}} = 512 \cdot P' + 1$ for all clipped $P'$

Adapted from slides © 1997 – 2010 van Dam et al., Brown University
CIS 636 Interactive Computer Graphics
CIS 736 Computer Graphics

Lab 1a of 7
OpenGL Setup and Basics

The purpose of this lab exercise is to help you get up and running with Mesa (Linux OpenGL) in the CIS Department's Linux environment and over XWindows, and to show you some basic rendering.

This lab assignment is worth a total of 10 points (1%).

Upload an electronic copy of the assignment in PDF form (converted from your word processor, or scanned) to your K-State Online (KSOL) drop box before the due date and time.

References
NeonHelium tutorials: http://nehe.gamedev.net
Mesa home page: http://www.mesa3d.org
OpenGL FAQ: http://www.opengl.org/resources/faq/
OpenGL viewing docs: http://www.opengl.org/resources/faq/technical/viewing.htm

Problems 4-7 of this lab are adapted from:

4. (10%) Step 1: Perspective Viewing with OpenGL
   
   ```
   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity();
   ```

5. (10%) Step 2: Making things move in Perspective
   
   ```
   glFrustum( -width/2.0, width/2.0, -height/2.0, height/2.0, -1.0, -20.0 );
   gluPerspective( field_of_view, aspect, near, far );
   ```

6. (10%) Step 3: Specifying the Viewing matrix
   
   ```
   gluLookAt( eyeX, eyeY, eyeZ,
              centerX, centerY, centerZ,
              upX, upY, upZ );
   ```

7. (10%) Step 4: Instancing Objects
OpenGL Architecture

CPU

Polynomial Evaluator

Display List

Per Vertex Operations & Primitive Assembly

Rasterization

Per Fragment Operations

Frame Buffer

Texture Memory

Pixel Operations

© 2000 Shreiner, D., Angel, E., Shreiner, V.
• Geometric primitives
  • points, lines and polygons

• Image Primitives
  • images and bitmaps
  • separate pipeline for images and geometry
  ▷ linked through texture mapping

• Rendering depends on state
  • colors, materials, light sources, etc.
- AGL, GLX, WGL
  * glue between OpenGL and windowing systems
- GLU (OpenGL Utility Library)
  * part of OpenGL
  * NURBS, tessellators, quadric shapes, etc.
- GLUT (OpenGL Utility Toolkit)
  * portable windowing API
  * not officially part of OpenGL

Adapted from slides © 2000 Shreiner, D., Angel, E., Shreiner, V.
Preliminaries

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

- Libraries

- Enumerated Types
  - OpenGL defines numerous types for compatibility
    - `GLfloat`
    - `GLint`
    - `Glenum`
    - `etc.`
GLUT Callback Functions

- Application Structure
  - Configure and open window
  - Initialize OpenGL state
  - Register input callback functions
    - render
    - resize
    - input: keyboard, mouse, etc.
  - Enter event processing loop
Sample Program

```c
void main( int argc, char** argv )
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutCreateWindow( argv[0] );
    init();
    glutDisplayFunc( display );
    glutReshapeFunc( resize );
    glutKeyboardFunc( key );
    glutIdleFunc( idle );
    glutMainLoop();
}
```
Set up whatever state you’re going to use

```c
void init( void )
{
    glClearColor( 0.0, 0.0, 0.0, 1.0 );
    glClearDepth( 1.0 );

    glEnable( GL_LIGHT0 );
    glEnable( GL_LIGHTING );
    glEnable( GL_DEPTH_TEST );
}
```
GLUT Callback Functions

- Routine to call when something happens
  - window resize or redraw
  - user input
  - animation
- “Register” callbacks with GLUT
  glutDisplayFunc( display );
  glutIdleFunc( idle );
  glutKeyboardFunc( keyboard );

© 2000 Shreiner, D., Angel, E., Shreiner, V.
Rendering Callback

- Do all of your drawing here
  
  ```c
  glutDisplayFunc( display );
  
  void display( void )
  {
    glClear( GL_COLOR_BUFFER_BIT );
    glBegin( GL_TRIANGLE_STRIP );
    glVertex3fv( v[0] );
    glVertex3fv( v[1] );
    glVertex3fv( v[2] );
    glVertex3fv( v[3] );
    glEnd();
    glutSwapBuffers();
  }
  ```
Elementary Rendering

- Geometric Primitives
- Managing OpenGL State
- OpenGL Buffers
OpenGL Geometric Primitives

- All geometric primitives are specified by vertices

GL_LINES
GL_POINTS
GL_LINE_STRIP
GL_LINE_LOOP
GL_TRIANGLES
GL_QUADS
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
GL_QUAD_STRIP
GL_POLYGON
void drawRhombus( GLfloat color[] )
{
    glBegin( GL_QUADS );
    glColor3fv( color );
    glVertex2f( 0.0, 0.0 );
    glVertex2f( 1.0, 0.0 );
    glVertex2f( 1.5, 1.118 );
    glVertex2f( 0.5, 1.118 );
    glEnd();
}
OpenGL Command Formats

```
glVertex3fv( v )
```

### Number of components
- 2 - \((x, y)\)
- 3 - \((x, y, z)\)
- 4 - \((x, y, z, w)\)

### Data Type
- b - byte
- ub - unsigned byte
- s - short
- us - unsigned short
- i - int
- ui - unsigned int
- f - float
- d - double

### Vector
- omit "v" for scalar form
- \(glVertex2f( x, y )\)

© 2000 Shreiner, D., Angel, E., Shreiner, V.
Specifying Geometric Primitives

- Primitives are specified using

\[
\begin{align*}
glBegin( \text{primType} ); \\
glEnd();
\end{align*}
\]

* \textit{primType} determines how vertices are combined

```c
GLfloat red, green, blue;
GLfloat coords[3];
glBegin( \text{primType} );
for ( i = 0; i < nVerts; ++i ) {
  glColor3f( red, green, blue );
  glVertex3fv( coords );
}
glEnd();
```
Transformations in OpenGL

- Modeling
- Viewing
  - orient camera
  - projection
- Animation
- Map to screen
3D is just like taking a photograph (lots of photographs!)
Camera Analogy & Transformations

- Projection transformations
  - adjust the lens of the camera
- Viewing transformations
  - tripod—define position and orientation of the viewing volume in the world
- Modeling transformations
  - moving the model
- Viewport transformations
  - enlarge or reduce the physical photograph
Steps in Forming an Image
- specify geometry (world coordinates)
- specify camera (camera coordinates)
- project (window coordinates)
- map to viewport (screen coordinates)

Each step uses transformations
Every transformation is equivalent to a change in coordinate systems (frames)
## Preview: Fixed Function Pipeline, Spaces & Matrices

(See Eberly 2e § 2.3.2 – 2.3.7, pp. 48-66, especially p. 58)

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>model coordinates / object coordinates</td>
</tr>
<tr>
<td>2.</td>
<td>world coordinates / scene coordinates</td>
</tr>
<tr>
<td>3.</td>
<td>camera coordinates / eye coordinates</td>
</tr>
<tr>
<td>4.</td>
<td>(optional) view coordinates / clip coordinates</td>
</tr>
<tr>
<td>5.</td>
<td>normalized device coordinates (NDC)</td>
</tr>
<tr>
<td>6.</td>
<td>screen coordinates</td>
</tr>
</tbody>
</table>

\[ H_{\text{world}}: \text{modelview transformation} \]

Normalizing transformation: \( X_{\text{world}} \rightarrow X_{\text{ndc}} \)

\[ H_{\text{window}}: \text{window matrix (aka viewport transformation)} \]

\[ X_{\text{model}} \rightarrow (H_{\text{world}}) \]
\[ X_{\text{world}} \rightarrow (H_{\text{view}}) \]
\[ X_{\text{view}} \rightarrow (H_{\text{proj}}) \]
\[ X_{\text{clip}} \rightarrow \text{(perspective division)} \]
\[ X_{\text{ndc}} \rightarrow (H_{\text{window}}) \]
\[ X_{\text{window}} \]

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( H_{\text{view}} )</td>
<td>“view matrix” (really NT!)</td>
</tr>
<tr>
<td>( H_{\text{proj}} )</td>
<td>projection matrix</td>
</tr>
<tr>
<td>( /W )</td>
<td>perspective division</td>
</tr>
</tbody>
</table>
Matrix Operations

- Specify Current Matrix Stack
  \texttt{glMatrixMode( \texttt{GL\_MODELVIEW} or \texttt{GL\_PROJECTION} )}

- Other Matrix or Stack Operations
  \texttt{glLoadIdentity() \quad glPushMatrix() \quad glPopMatrix()}

- Viewport
  * usually same as window size
  * viewport aspect ratio should be same as projection transformation
    or resulting image may be distorted
  \texttt{glViewport( x, y, width, height )}
Projection Transformation

- Shape of viewing frustum
- Perspective projection
  - `gluPerspective( fovy, aspect, zNear, zFar )`
  - `gluFrustum( left, right, bottom, top, zNear, zFar )`
- Orthographic parallel projection
  - `glOrtho( left, right, bottom, top, zNear, zFar )`
  - `gluOrtho2D( left, right, bottom, top )`

  - calls `glOrtho` with z values near zero

© 2000 Shreiner, D., Angel, E., Shreiner, V.
Applying Projection Transformations

- Typical use (orthographic projection)
  
  ```c
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho( left, right, bottom, top, zNear, zFar );
  ```
Position the camera/eye in the scene
  * place the tripod down; aim camera

To “fly through” a scene
  * change viewing transformation and redraw scene

\texttt{gluLookAt( } \texttt{eye}_x, \texttt{eye}_y, \texttt{eye}_z, \texttt{aim}_x, \texttt{aim}_y, \texttt{aim}_z, \texttt{up}_x, \texttt{up}_y, \texttt{up}_z \texttt{)}

  * up vector determines unique orientation
  * careful of degenerate positions
Projection Tutorial

```cpp
fovy aspect zNear zFar

gluPerspective( 60.0 , 1.00 , 1.0 , 10.0 );
gluLookAt( 0.00 , 0.00 , 2.00 , <= eye
0.00 , 0.00 , 0.00 , <= center
0.00 , 1.00 , 0.00 ); <= up

Click on the arguments and move the mouse to modify values.
```
Modeling Transformations

- Move object
  \[ \text{glTranslate}(x, y, z) \]
- Rotate object around arbitrary axis
  \[ \text{glRotate}(\text{angle}, x, y, z) \]
  \* angle is in degrees
- Dilate (stretch or shrink) or mirror object
  \[ \text{glScale}(x, y, z) \]
Transformation Tutorial

```plaintext
glTranslatef( 0.00, 0.00, 0.00 );
glRotatef( -52.0, 0.00, 1.00, 0.00 );
glScalef( 1.00, 1.00, 1.00 );
glBegin( ... );
...
```

Click on the arguments and move the mouse to modify values.
Moving camera is equivalent to moving every object in the world towards a stationary camera.

Viewing transformations are equivalent to several modeling transformations:
- `gluLookAt()` has its own command
- You can make your own polar view or pilot view.
Projection transformations ($\text{gluPerspective, glOrtho}$) are left handed

- think of $z_{\text{Near}}$ and $z_{\text{Far}}$ as distance from view point

- Everything else is right handed, including the vertices to be rendered
void resize( int w, int h )
{
    glVertex2d( 0, 0, (GLsizei) w, (GLsizei) h );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluPerspective( 65.0, (GLfloat) w / h, 1.0, 100.0 );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    gluLookAt( 0.0, 0.0, 5.0,
               0.0, 0.0, 0.0,
               0.0, 1.0, 0.0 );
}
void resize( int width, int height )
{
    GLdouble aspect = (GLdouble) width / height;
    GLdouble left = -2.5, right = 2.5;
    GLdouble bottom = -2.5, top = 2.5;
    glViewport( 0, 0, (GLsizei) w, (GLsizei) h );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    ... continued ...

if ( aspect < 1.0 ) {
    left /= aspect;
    right /= aspect;
} else {
    bottom *= aspect;
    top *= aspect;
}
glOrtho( left, right, bottom, top, near, far );
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
}
Summary

- Three Tutorials from SIGGRAPH 2000
  - Overall architecture
  - Initialization
  - Viewport management
- Part 2: Basic Rendering – Vicki Shreiner
- Part 3: 3-D Viewing – Edward Angel
  - Math background (see CG Basics 1)
  - Viewing and normalization transformations (see CG Basics 4)
  - More on viewing in CG Basics 4
  - View volume specification
  - Automated part: clipping
Terminology

- **OpenGL and GL Utility Toolkit (GLUT)**
  - State machine
  - Using GLUT
  - Specifying perspective, parallel projections

- **Transformations**
  - Fixed function pipeline: modelview, normalizing, viewing
  - Rigid body: preserves distance (e.g., translation, rotation)
  - Linear
    - Preserves vector addition, scalar multiplication
    - e.g., rotation, scaling
  - Affine: linear transformation followed by translation
  - Non-affine: all others (e.g., perspective-to-parallel transformation)