GLUT, Transformations I

Week 2, Wed Jan 13

Redrawing Display
- display only redrawn by explicit request
- glutPostRedisplay() function
- default window resize callback does this
- idle called from main loop when no user input
- good place to request redraw
- will call display next time through event loop
- should return control to main loop quickly
- continues to rotate even when no user action

GLUT Callback Functions
- // you supply these kind of functions
- void reshape(int w, int h);
- void keyboard(unsigned char key, int x, int y);
- void mouse(int btn, int state, int x, int y);
- void idle();
- void display();
- // register them with glut
- glut reshape Func ( reshape ) ;
- glut keyboard Func ( keyboard ) ;
- glut mouse Func ( mouse ) ;
- glut display Func ( display ) ;
- void glut display Func ( void *Func) (void);
- void glut key board Func ( void *Func) (unsigned char key, int x, int y);
- void glut mouse Func ( void *Func) (int btn, int state, int x, int y);
- void glut id le() ;
- void glut display() ;

GLUT Example 1

```c
#include <GLUT/glut.h>

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glVertex3f(1.0, 0.0, -1.0);
    glVertex3f(1.0, 0.0, -1.0);
    glVertex3f(0.0, 1.0, -1.0);
    glEnd();
}
```

GLUT Example 2

```c
#include <GLUT/glut.h>

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-6.0, 6.0, -6.0, 6.0, -10.0, 10.0);
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glEnable(GL_DEPTH_TEST);
    glBegin(GL_TRIANGLES);
    glVertex3f(1.0, 0.0, -1.0);
    glVertex3f(1.0, 0.0, -1.0);
    glVertex3f(0.0, 1.0, -1.0);
    glEnd();
}
```

GLUT Example 3

```c
#include <GLUT/glut.h>

void idle() {
    glRotatef(0.1, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glEnable(GL_DEPTH_TEST);
    glBegin(GL_TRIANGLES);
    glVertex3f(1.0, 0.0, -1.0);
    glVertex3f(1.0, 0.0, -1.0);
    glVertex3f(0.0, 1.0, -1.0);
    glEnd();
    glutSwapBuffers();
}  
```

GLUT Example 4

```c
#include <GLUT/glut.h>

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glEnable(GL_DEPTH_TEST);
    glBegin(GL_TRIANGLES);
    glVertex3f(1.0, 0.0, -1.0);
    glVertex3f(1.0, 0.0, -1.0);
    glVertex3f(0.0, 1.0, -1.0);
    glEnd();
    glutSwapBuffers();
}
```

Keyboard/Mouse Callbacks
- again, do minimal work
- consider keypress that triggers animation
- do not have loop calling display in callback!
- what if user hits another key during animation?
- instead, use shared/global variables to keep track of state
- yes, OK to use globals for this!
- then display function just uses current variable value

Redraw Display
- display only redrawn by explicit request
- glutPostRedisplay() function
- default window resize callback does this
- idle called from main loop when no user input
- good place to request redraw
- will call display next time through event loop
- should return control to main loop quickly
- continues to rotate even when no user action

GLUT: OpenGL Utility Toolkit

- developed by Mark Kilgard (also from SGI)
- simple, portable window manager
- opening windows
- handling input with callbacks
- keyboard, mouse, window reshape events
- timing
- idle processing, idle events
- good place to request redraw
- default window resize callback does this
- track of state
- good place to request redraw
- continued rotation even when no user action

Review: Rendering Pipeline
- Geometry Database
- Model/View Transform.
- Lighting
- Perspective Transform.
- Clipping
- Scan Conversion
- Depth Test
- Texturing
- Blending
- Frame-buffer

Event-Driven Programming
- main loop not under your control
- vs. batch mode where you control the flow
- control flow through event callbacks
- redraw the window now
- key was pressed
- mouse moved
- callback functions called from main loop when events occur
- mouse/keybord state setting vs. redrawing

Review: Graphics State
- set the state once, remains until overwritten
- glColor3f(1.0, 1.0, 0.0) → set color to yellow
- glSetClearColor(0.0, 0.0, 0.0) → dark blue bg
- glEnable(LIGHT0) → turn on light
- glEnable(GL_DEPTH_TEST) → hidden surf.

GLUT Example 3

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

void display() {
    glRotatef(angle, 0,0,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(4.0);
    glBegin(GL_LINES);
    glVertex3f(-1.0, 0.0,  -1.0);
    glVertex3f(1.0, 0.0,  -1.0);
    glEnd();
    glutSwapBuffers();
}  
```

GLUT Example 4

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

void display() {
    glRotatef(angle, 0,0,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(4.0);
    glBegin(GL_LINES);
    glVertex3f(-1.0, 0.0,  -1.0);
    glVertex3f(1.0, 0.0,  -1.0);
    glEnd();
    glutSwapBuffers();
}  
```

GLUT: OpenGL Utility Toolkit

- simple, portable window manager
- opening windows
- handling input with callbacks
- keyboard, mouse, window reshape events
- timing
- idle processing, idle events
- designed for small/medium size applications
- distributed as binaries
- free, but not open source

Review: Rendering Pipeline
- Geometry Database
- Model/View Transform.
- Lighting
- Perspective Transform.
- Clipping
- Scan Conversion
- Texturing
- Blending
- Frame-buffer

Event-Driven Programming
- main loop not under your control
- vs. batch mode where you control the flow
- control flow through event callbacks
- redraw the window now
- key was pressed
- mouse moved
- callback functions called from main loop when events occur
- mouse/keybord state setting vs. redrawing

GLUT Callback Functions
- // you supply these kind of functions
- void reshape(int w, int h);
- void keyboard(unsigned char key, int x, int y);
- void mouse(int btn, int state, int x, int y);
- void idle();
- void display();
- // register them with glut
- glut reshape Func ( reshape ) ;
- glut keyboard Func ( keyboard ) ;
- glut mouse Func ( mouse ) ;
- glut display Func ( display ) ;
- void glut display Func ( void *Func) (void);
- void glut key board Func ( void *Func) (unsigned char key, int x, int y);
- void glut mouse Func ( void *Func) (int btn, int state, int x, int y);
- void glut id le() ;
- void glut display() ;

GLUT Example 1

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

void display() {
    glRotatef(angle, 0,0,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(4.0);
    glBegin(GL_LINES);
    glVertex3f(-1.0, 0.0,  -1.0);
    glVertex3f(1.0, 0.0,  -1.0);
    glEnd();
    glutSwapBuffers();
}  
```

GLUT Example 2

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

void display() {
    glRotatef(angle, 0,0,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(4.0);
    glBegin(GL_LINES);
    glVertex3f(-1.0, 0.0,  -1.0);
    glVertex3f(1.0, 0.0,  -1.0);
    glEnd();
    glutSwapBuffers();
}  
```

GLUT Example 3

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

void display() {
    glRotatef(angle, 0,0,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(4.0);
    glBegin(GL_LINES);
    glVertex3f(-1.0, 0.0,  -1.0);
    glVertex3f(1.0, 0.0,  -1.0);
    glEnd();
    glutSwapBuffers();
}  
```

GLUT Example 4

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

void display() {
    glRotatef(angle, 0,0,1);
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(4.0);
    glBegin(GL_LINES);
    glVertex3f(-1.0, 0.0,  -1.0);
    glVertex3f(1.0, 0.0,  -1.0);
    glEnd();
    glutSwapBuffers();
}  
```
Transformations
- transforming an object = transforming all its points
- transforming a polygon = transforming its vertices

Matrix Representation
- represent 2D transformation with matrix
  - multiply matrix by column vector
  - apply transformation to point

Scaling
- scaling a coordinate means multiplying each of its components by a scalar
- uniform scaling means this scalar is the same for all components:

2D Rotation: Another Derivation
- counter-clockwise
- RHS
  \[
  \begin{align*}
  x' &= x \cos(0) - y \sin(0) \\
  y' &= x \sin(0) + y \cos(0)
  \end{align*}
  \]
  - trig identity
    - \[ x = r \cos(\phi) \]
    - \[ y = r \sin(\phi) \]
    - \[ x' = r \cos(\phi + \theta) \]
    - \[ y' = r \sin(\phi + \theta) \]
    - substitute
      - \[ x' = r \cos(0) \cos(\theta) - r \sin(0) \sin(\theta) \]
      - \[ y' = r \sin(0) \cos(\theta) + r \cos(0) \sin(\theta) \]

2D Rotation: Another Derivation
- easy to capture in matrix form:
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  \cos(\theta) & -\sin(\theta) \\
  \sin(\theta) & \cos(\theta)
  \end{pmatrix} \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  \]
- even though \( \sin(q) \) and \( \cos(q) \) are nonlinear functions of \( q \):
  - \( x' \) is a linear combination of \( x \) and \( y \)
  - \( y' \) is a linear combination of \( x \) and \( y \)

Scaling
- non-uniform scaling: different scalars per component:
- how can we represent this in matrix form?

2D Rotation: Another Derivation
- shear along \( x \) axis
  - push points right to right in proportion to height
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  x - A \\
  y
  \end{pmatrix}
  \]
  where
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  x - A \\
  y
  \end{pmatrix}
  \]
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  x - A \\
  y
  \end{pmatrix}
  \]
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  x - A \\
  y
  \end{pmatrix}
  \]

2D Rotation: Another Derivation
- shear along \( y \) axis
  - push points up to up in proportion to height
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  x \\
  y - B
  \end{pmatrix}
  \]
  where
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  x \\
  y - B
  \end{pmatrix}
  \]
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  x \\
  y - B
  \end{pmatrix}
  \]
  \[
  \begin{pmatrix}
  x' \\
  y'
  \end{pmatrix} = \begin{pmatrix}
  x \\
  y - B
  \end{pmatrix}
  \]
Shear
• shear along x axis
  • push points to right in proportion to height
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix}
  =
  \begin{bmatrix}
  1 & s_b & 0 \\
  0 & 1 & 0
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

Reflection
• reflect across x axis
  • mirror
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix}
  =
  \begin{bmatrix}
  1 & 0 & 0 \\
  0 & -1 & 0
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

Linear Transformations
• linear transformations are combinations of
  • shear
  • scale
  • rotate
  • reflect

properties of linear transformations
• satisfies \( T(ax+by) = a T(x) + b T(y) \)
• origin maps to origin
• lines map to lines
• parallel lines remain parallel
• ratios are preserved
• closed under composition

2D Translation
• 2D translation multiplication matrix
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix}
  =
  \begin{bmatrix}
  a & b & 0 \\
  c & d & 0
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

Reflection
• reflect across x axis
  • mirror
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix}
  =
  \begin{bmatrix}
  1 & 0 & 0 \\
  0 & -1 & 0
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

Homogeneous Coordinates
• homogenize to convert homogeneous 3D point to cartesian 2D point:
  - divide by w to get \((x/w, y/w, 1)\)
  - projects line to point onto \(w=1\) plane
  - rotates around \(x\) and \(y\) axes
• homogeneous coordinates trick
  - represent 2D coordinates \((x, y)\) with 3-vector \((x, y, 1)\)

Challenge
• matrix multiplication
  • for everything except translation
  • how to do everything with multiplication?
  • then just do composition, no special cases
  • homogeneous coordinates trick
  • represent 2D coordinates \((x, y)\) with 3-vector \((x, y, 1)\)

Homogeneous Coordinates Geometrically
• our 2D transformation matrices are now 3x3:

Translation:
\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

Challenge
• matrix multiplication
  • for everything except translation
  • how to do everything with multiplication?
  • then just do composition, no special cases
  • homogeneous coordinates trick
  • represent 2D coordinates \((x, y)\) with 3-vector \((x, y, 1)\)

Homogeneous Coordinates Geometrically
• our 2D transformation matrices are now 3x3:

Translation:
\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

• point in 2D cartesian:
  \[
  \begin{bmatrix}
  x \\
  y \\
  w
  \end{bmatrix}
  \]

Affine Transformations
• affine transforms are combinations of
  • linear transformations
  • translations
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  w'
  \end{bmatrix}
  =
  \begin{bmatrix}
  a & b & c \\
  d & e & f \\
  0 & 0 & 1
  \end{bmatrix}
  \begin{bmatrix}
  x \\
  y \\
  w
  \end{bmatrix}
  \]

• properties of affine transformations
  • origin does not necessarily map to origin
  • lines map to lines
  • parallel lines remain parallel
  • ratios are preserved
  • closed under composition

Homogeneous Coordinates Summary
• may seem unintuitive, but they make graphics operations much easier
• allow all affine transformations to be expressed through matrix multiplication
  • we’ll see even more later...
• use 3x3 matrices for 2D transformations
• use 4x4 matrices for 3D transformations