University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2007

Tamara Munzner

Transformations I

Week 2, Mon Jan 15

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007

News

- Labs start tomorrow

Labs

Week 2 Lab

- labs start Tuesday
- project 0
  - http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007/a0
  - make sure you can compile OpenGL/GLUT
    - very useful to test home computing environment
  - template: spin around obj files
  - todo: change rotation axis
  - do hand in to test configuration, but not graded

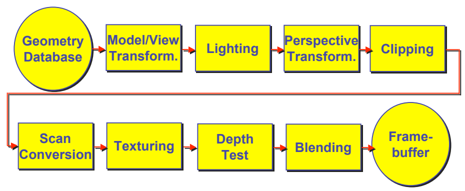
Remote Graphics

- OpenGL does not work well remotely
  - very slow
- only one user can use graphics at a time
  - current X server doesn't give priority to console, just does first come first served
  - problem: FCFS policy = confusion/chaos
- solution: console user gets priority
  - only use graphics remotely if nobody else logged on
    - with 'who' command, ":0" is console person
  - stop using graphics if asked by console user via email
  - or console user can reboot machine out from under you

Readings for Today + Next 3 Lectures

- FCG Chap 6 Transformation Matrices
  - *except* 6.1.6, 6.3.1
- FCG Sect 13.3 Scene Graphs
- RB Chap Viewing
  - Viewing and Modeling Transforms *until* Viewing Transformations
  - Examples of Composing Several Transformations *through* Building an Articulated Robot Arm
- RB Appendix Homogeneous Coordinates and Transformation Matrices
  - *until* Perspective Projection
- RB Chap Display Lists

Review: Rendering Pipeline



Review: OpenGL

- pipeline processing, set state as needed

```
void display()
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 1.0, 0.0);
  glBegin(GL_POLYGON);
    glVertex3f(0.25, 0.25, -0.5);
    glVertex3f(0.75, 0.25, -0.5);
    glVertex3f(0.75, 0.75, -0.5);
    glVertex3f(0.25, 0.75, -0.5);
  glEnd();
  glFlush();
}
```


GLUT

GLUT: OpenGL Utility Toolkit

- developed by Mark Kilgard (also from SGI)
- simple, portable window manager
  - opening windows
    - handling graphics contexts
  - 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

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/keyboard 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 but, int state, int x, int y);
void idle();
void display();

  // register them with glut

glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMouseFunc(mouse);
glutIdleFunc(idle);
glutDisplayFunc(display);
```

```
void glutDisplayFunc (void (*func)(void));
void glutKeyboardFunc (void (*func)(unsigned char key, int x, int y));
void glutIdleFunc (void (*func)());
void glutReshapeFunc (void (*func)(int width, int height));
```


GLUT Example 1

```
#include <GLUT/glut.h>
void display()
{
  glClearColor(0,0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor4f(0,1,0,1);
  glBegin(GL_POLYGON);
  glVertex3f(0.25, 0.25, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.25, 0.75, -0.5);
  glEnd();
  glutSwapBuffers();
}
```

```
int main(int argc,char**argv)
{
  glutInit( &argc, argv );
  glutInitDisplayMode(
   GLUT_RGB|GLUT_DOUBLE);
  glutInitWindowSize(640,480);
  glutCreateWindow("glut1");
  glutDisplayFunc( display );
  glutMainLoop();
  return 0; // never reached
}
```


GLUT Example 2

```
#include <GLUT/glut.h>
void display()
{
  glRotatef(0.1, 0,0,1);

  glClearColor(0,0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor4f(0,1,0,1);
  glBegin(GL_POLYGON);
  glVertex3f(0.25, 0.25, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.25, 0.75, -0.5);
  glEnd();
  glutSwapBuffers();
}
```

```
int main(int argc,char**argv)
{
  glutInit( &argc, argv );
  glutInitDisplayMode(
   GLUT_RGB|GLUT_DOUBLE);
  glutInitWindowSize(640,480);
  glutCreateWindow("glut2");
  glutDisplayFunc( display );
  glutMainLoop();
  return 0; // never reached
}
```


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 Example 3

```
#include <GLUT/glut.h>
void display()
{
  glRotatef(0.1, 0,0,1);

  glClearColor(0,0,0,1);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor4f(0,1,0,1);
  glBegin(GL_POLYGON);
  glVertex3f(0.25, 0.25, -0.5);
  glVertex3f(0.75, 0.25, -0.5);
  glVertex3f(0.75, 0.75, -0.5);
  glVertex3f(0.25, 0.75, -0.5);
  glEnd();
  glutSwapBuffers();
}
```

```
void idle() {
    glutPostRedisplay();
}

int main(int argc,char**argv)
{
  glutInit( &argc, argv );
  glutInitDisplayMode(
   GLUT_RGB|GLUT_DOUBLE);
  glutInitWindowSize(640,480);
  glutCreateWindow("glut1");
  glutDisplayFunc( display );
  glutIdleFunc( idle );
  glutMainLoop();
  return 0; // never reached
}
```

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

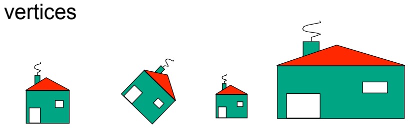
GLUT Example 4

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

bool animToggle = true;
float angle = 0.1;

void display() {
  glRotatef)angle, 0,0,1);
   ...
}
void idle() {
  glutPostRedisplay();
}
int main(int argc,char**argv)
{  ...
  glutKeyboardFunc( doKey );
   ...
}

void doKey(unsigned char key,
           int x, int y) {
  if ('t' == key) {
    animToggle = !animToggle;
    if (!animToggle)
      glutIdleFunc(NULL);
    else
      glutIdleFunc(idle);
  } else if ('r' == key) {
    angle = -angle;
  }
  glutPostRedisplay();
}
```


Transformations

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

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}a & b\\c & d\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} \qquad \begin{aligned}x'&=ax+by\\y'&=cx+dy\end{aligned}$$

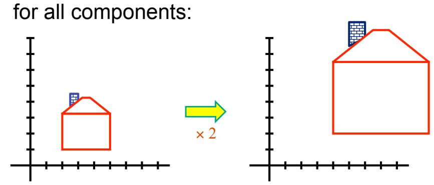
- transformations combined by multiplication

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}a & b\\c & d\end{bmatrix}\begin{bmatrix}d & e\\f & g\end{bmatrix}\begin{bmatrix}h & i\\j & k\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}$$

  - matrices are efficient, convenient way to represent sequence of transformations!

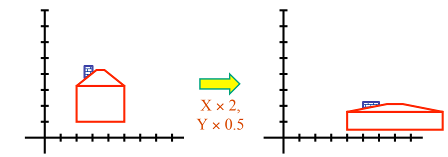
Scaling

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



Scaling

- non-uniform scaling: different scalars per component:

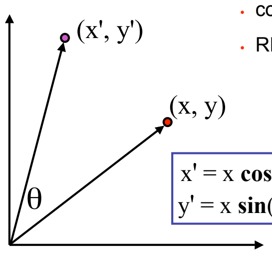


- how can we represent this in matrix form?

Scaling

- scaling operation: $\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}ax\\by\end{bmatrix}$
- or, in matrix form: $\begin{bmatrix}x'\\y'\end{bmatrix}=\underbrace{\begin{bmatrix}a & 0\\0 & b\end{bmatrix}}_{\text{scaling matrix}}\begin{bmatrix}x\\y\end{bmatrix}$

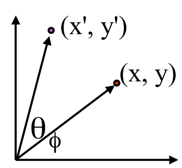
2D Rotation



- counterclockwise
- RHS

$$x' = x\,\mathbf{cos}(\theta) - y\,\mathbf{sin}(\theta)$$
$$y' = x\,\mathbf{sin}(\theta) + y\,\mathbf{cos}(\theta)$$

2D Rotation From Trig Identities



x = r cos (φ)
y = r sin (φ)
x' = r cos (φ + θ)
y' = r sin (φ + θ)

Trig Identity…
x' = r cos(φ) cos(θ) – r sin(φ) sin(θ)
y' = r sin(φ) cos(θ) + r cos(φ) sin(θ)

Substitute…
$x' = x\,\mathbf{cos}(\theta) - y\,\mathbf{sin}(\theta)$
$y' = x\,\mathbf{sin}(\theta) + y\,\mathbf{cos}(\theta)$

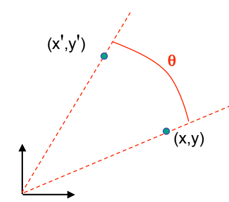
2D Rotation Matrix

- easy to capture in matrix form:

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}\cos(\theta) & -\sin(\theta)\\\sin(\theta) & \cos(\theta)\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}$$

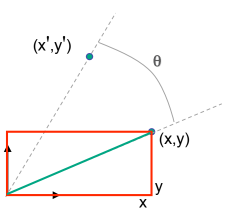
- even though sin(q) and cos(q) are nonlinear functions of q,
  - x' is a linear combination of x and y
  - y' is a linear combination of x and y

2D Rotation: Another Derivation



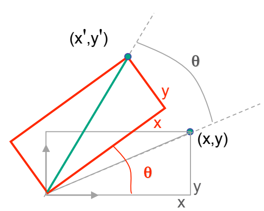
$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

2D Rotation: Another Derivation



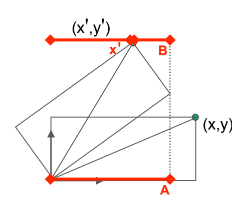
$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

2D Rotation: Another Derivation



$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

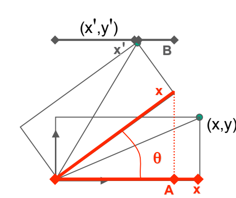
2D Rotation: Another Derivation



$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$x' = A - B$$

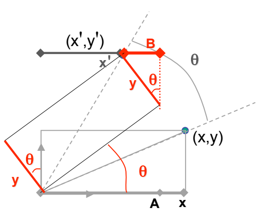
2D Rotation: Another Derivation



$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$x' = A - B$$
$$A = x\cos\theta$$

2D Rotation: Another Derivation



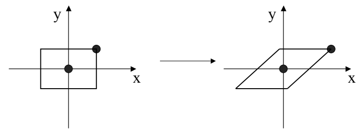
$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$x' = A - B$$
$$A = x\cos\theta$$
$$B = y\sin\theta$$

Shear

- shear along x axis
  - push points to right in proportion to height

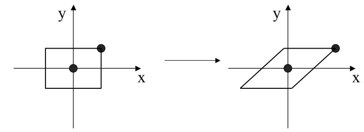
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$$



Shear

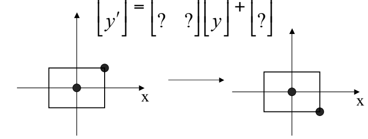
- shear along x axis
  - push points to right in proportion to height

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



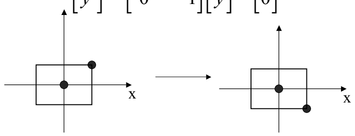
Reflection

- reflect across x axis
  - mirror $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} ? \\ ? \end{bmatrix}$

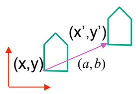


Reflection

- reflect across x axis
  - mirror $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

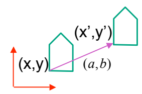


2D Translation



$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

2D Translation



$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

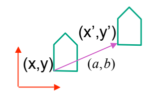
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

scaling matrix

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

rotation matrix

2D Translation



vector addition

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

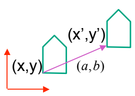
scaling matrix

matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

rotation matrix

2D Translation



vector addition

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

scaling matrix

matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

rotation matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

translation multiplication matrix??

Linear Transformations

- linear transformations are combinations of
  - shear
  - scale
  - rotate
  - reflect

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \begin{aligned} x' &= ax + by \\ y' &= cx + dy \end{aligned}$$

- properties of linear transformations
  - satisifes T(***s*x+*t*y**) = *s* T(**x**) + *t* T(**y**)
  - origin maps to origin
  - lines map to lines
  - parallel lines remain parallel
  - ratios are preserved
  - closed under composition

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

- our 2D transformation matrices are now 3x3:

$$\mathbf{R}otation = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{S}cale = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

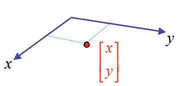
$$\mathbf{T}ranslation = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

- use rightmost column

$$\begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x*1+a*1 \\ y*1+b*1 \\ 1 \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \\ 1 \end{bmatrix}$$

Homogeneous Coordinates Geometrically

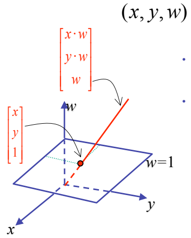
- point in 2D cartesian



Homogeneous Coordinates Geometrically

homogeneous **cartesian**

$$(x, y, w) \xrightarrow{/w} \left(\frac{x}{w}, \frac{y}{w}\right)$$

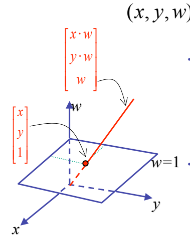


- point in 2D cartesian + weight w = point P in 3D homog. coords
- multiples of (x,y,w)
  - form a line L in 3D
  - all homogeneous points on L represent same 2D cartesian point
  - example: (2,2,1) = (4,4,2) = (1,1,0.5)

Homogeneous Coordinates Geometrically

homogeneous **cartesian**

$$(x, y, w) \xrightarrow{/w} \left(\frac{x}{w}, \frac{y}{w}\right)$$



- homogenize to convert homog. 3D point to cartesian 2D point:
  - divide by w to get (x/w, y/w, 1)
  - projects line to point onto w=1 plane
  - like normalizing, one dimension up
- when w=0, consider it as direction
  - points at infinity
  - these points cannot be homogenized
  - lies on x-y plane
- (0,0,0) is undefined

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