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, “O” is console person
  - stop using graphics if asked by console user via email
  - or console user can reboot machine out from under you

Transformations I

Week 2, Mon Jan 15

Transformations

Readings for Today + Next 3 Lectures

- FCG Chap 6 Transformation Matrices
  - except 6.1.6, 6.3.1
- FCG Chap 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

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

GLUT Callbacks

// you supply these kind of functions
void reshape(int w, int h);
void key(unsigned char key, int x, int y);
void mouse(int button, int state, int x, int y);
void idlet();
void display();

// register them with glut
glutReshapeFunc( reshape );
glutKeyboardFunc( keyboard );
glutMouseFunc( mouse );
glutIdleFunc( idlet );

void glutDisplayFunc( void (*)(void) );
void glutKeyboardFunc( void (*)(unsigned char key, int x, int y) );
void glutMouseFunc( void (*)(int button, int state, int x, int y) );

GLUT Example 1

#include <GLUT/glut.h>
void display()
{
  glClearColor(0.0, 0.0, 1.0, 1.0);
  glClear(GL_COLOR_BUFFER_BIT);
  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, 1.0, 0.0);
  int main(int argc, char** argv)
  {
    glutInitDisplayMode( GLUT_RGB|GLUT_DOUBLE );
    glutInitWindowSize(640,480);
    glutInit( argc, argv );
    glutDisplayMode ( GL_RGB_ GL_DOUBLE );
    glutMainLoop();
    return 0; // never reached
  }
}

GLUT Example 3

#include <GLUT/glut.h>
void display()
{
  glRotatef(0.1, 0.0, 1.0, 0.0);
  int main(int argc, char** argv)
  {
    glutInitDisplayMode( GLUT_RGB|GLUT_DOUBLE );
    glutInitWindowSize(640,480);
    glutCreateWindow("glut3");
    glutDisplayFunc( display );
    glutIdleFunc( idlet );
    glutMainLoop();
    return 0; // never reached
  }
}
2D Rotation
\[ (x', y') = (x \cos(\theta) - y \sin(\theta), x \sin(\theta) + y \cos(\theta)) \]
- counterclockwise
- RHS

Scaling
- scaling a coordinate means multiplying each of its components by a scalar
- uniform scaling means this scalar is the same for all components:
\[ \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \]
- matrices are efficient, convenient way to represent sequence of transformations!

Matrix Representation
- represent 2D transformation with matrix
- multiply matrix by column vector
- transformations combined by multiplication
\[ \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \]
- matrices are efficient, convenient way to represent transformation to point

Transformation
- transforming an object = transforming all its points
- transforming a polygon = transforming its vertices

GLUT Example 4
```c
declare int argc, char** argv)
declare bool animToggle = true;
declare float angle = 0.1;
declare void display() {
    glutPostRedisplay();
    else
        glutIdleFunc(idle);
    else if ('r' == key) {
        angle = -angle;
    } else if ('r' == key) {
        glutPostRedisplay();
    }
}
declare int main(int argc, char**argv) {
    glutKeyboardFunc( glutPostRedisplay();
    return 0;
}
declare glutKeyboardFunc( do key :
```
Homogeneous Coordinates Geometrically
- point in 2D cartesian

Affine Transformations
- affine transforms are combinations of
  - linear transformations
  - translations

Homogeneous Coordinates Geometrically
- homogeneous cartesian
- points 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
- point in 2D cartesian

Affine Transformations
- 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