CSE 167:
Introduction to Computer Graphics
Lecture #2: OpenGL Primer

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Spring Quarter 2015

Announcements

- Project 1 due this Friday at 1pm
- Homework 2 discussion on Monday at 4pm

Lecture Overview

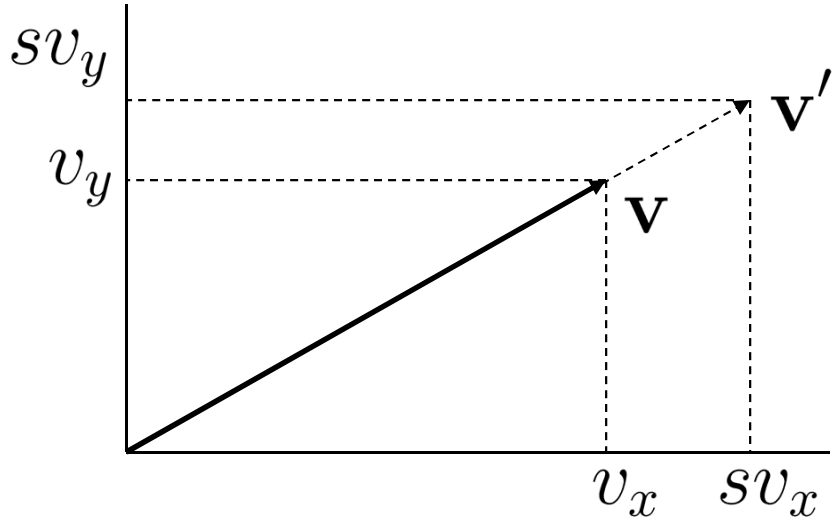
- Affine Transformations
- Homogeneous Coordinates

Affine Transformations

- Most important for graphics:
  - rotation, translation, scaling
- Wolfram MathWorld:
  - An **affine transformation** is any **transformation** that preserves collinearity (i.e., all points lying on a line initially still lie on a line after **transformation**) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after **transformation**).
- Implemented using matrix multiplications

Scaling

- Uniform scaling matrix in 2D

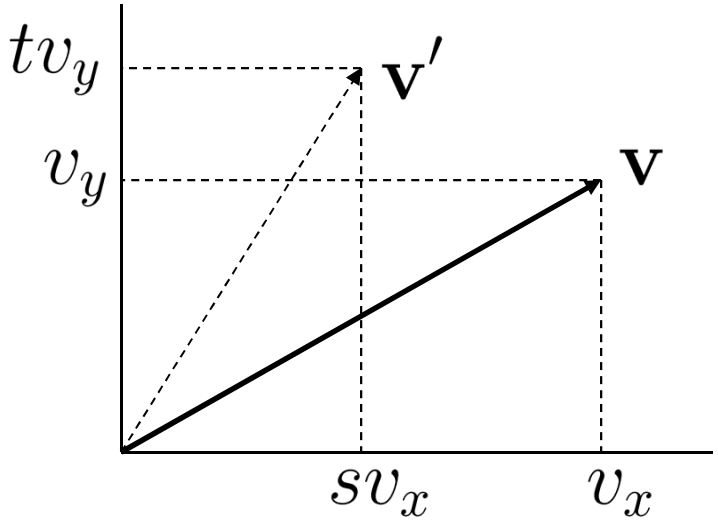


- Analogous in 3D

$$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \mathbf{v} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} v'_x \\ v'_y \end{bmatrix} = \mathbf{v}'$$

Scaling

- Nonuniform scaling matrix in 2D



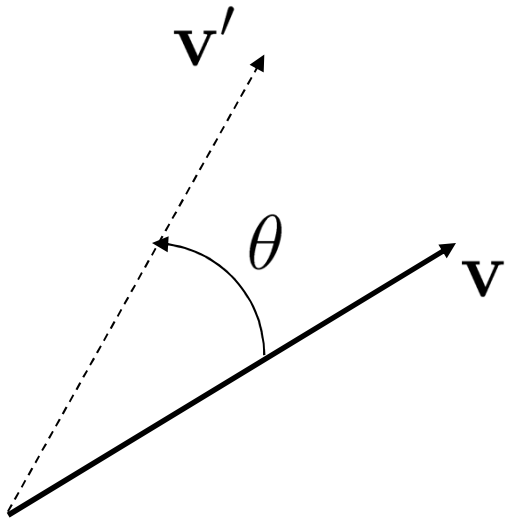
- Analogous in 3D

$$\begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix} \mathbf{v} = \begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} v'_x \\ v'_y \end{bmatrix} = \mathbf{v}'$$

Rotation in 2D

- Convention: positive angle rotates counterclockwise
- Rotation matrix

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$



$$\mathbf{v}' = \mathbf{R}(\theta)\mathbf{v}$$

Rotation in 3D

Rotation around coordinate axes

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation in 3D

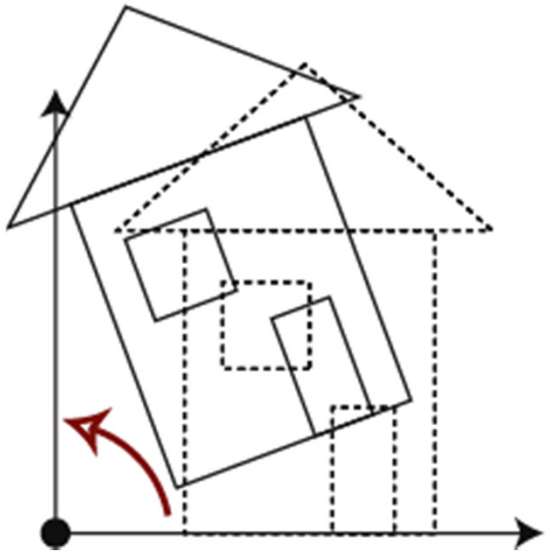
- Concatenation of rotations around x, y, z axes

$$\mathbf{R}_{x,y,z}(\theta_x, \theta_y, \theta_z) = \mathbf{R}_x(\theta_x)\mathbf{R}_y(\theta_y)\mathbf{R}_z(\theta_z)$$

- $\theta_x, \theta_y, \theta_z$ are called Euler angles
- Result depends on matrix order!

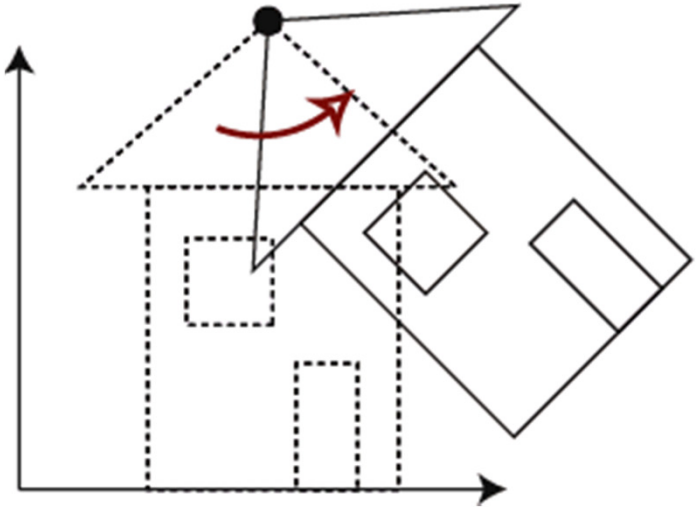
$$\mathbf{R}_x(\theta_x)\mathbf{R}_y(\theta_y)\mathbf{R}_z(\theta_z) \neq \mathbf{R}_z(\theta_z)\mathbf{R}_y(\theta_y)\mathbf{R}_x(\theta_x)$$

How to rotate around a Pivot Point?



Rotation around origin:

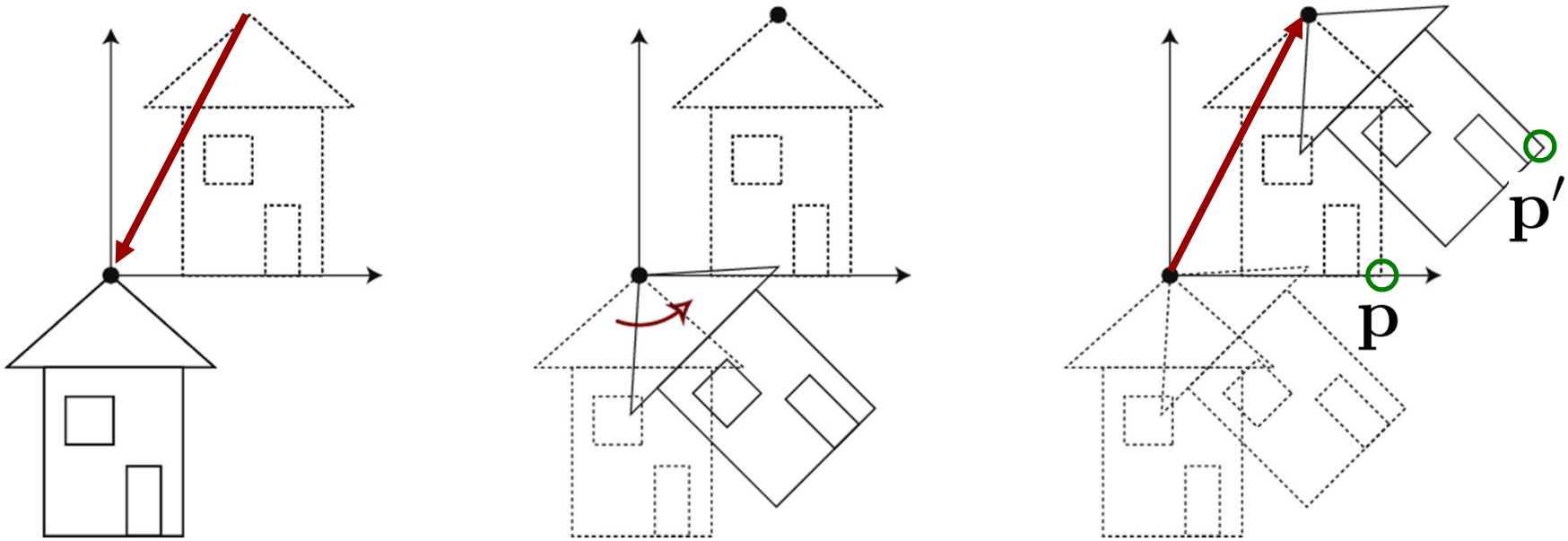
p’ = R p



Rotation around pivot point:

p’ = ?

Rotating point p around a pivot point



1. Translation T 2. Rotation R 3. Translation T^{-1}

$$p' = T^{-1} R T p$$

Concatenating transformations

- Given a sequence of transformations $\mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$

$$\mathbf{p}' = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1\mathbf{p}$$

$$\mathbf{M}_{total} = \mathbf{M}_3\mathbf{M}_2\mathbf{M}_1$$

$$\mathbf{p}' = \mathbf{M}_{total}\mathbf{p}$$

- Note: associativity applies:

$$\mathbf{M}_{total} = (\mathbf{M}_3\mathbf{M}_2)\mathbf{M}_1 = \mathbf{M}_3(\mathbf{M}_2\mathbf{M}_1)$$

Lecture Overview

- Affine Transformations
- Homogeneous Coordinates

Homogeneous Coordinates

- Basic: a trick to unify/simplify computations.

- Deeper: projective geometry
  - Interesting mathematical properties
  - Good to know, but less immediately practical
  - We will use some aspect of this when we do perspective projection

Homogeneous Coordinates

- Add an extra component. 1 for a point, 0 for a vector:

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \qquad \vec{\mathbf{v}} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}$$

- Combine **M** and **d** into single 4x4 matrix:

$$\begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- And see what happens when we multiply…

Homogeneous Point Transform

- Transform a point:

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{xx}p_x + m_{xy}p_y + m_{xz}p_z + d_x \\ m_{yx}p_x + m_{yy}p_y + m_{yz}p_z + d_y \\ m_{zx}p_x + m_{zy}p_y + m_{zz}p_z + d_z \\ 0+0+0+1 \end{bmatrix}$$

$$\mathbf{M} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \vec{\mathbf{d}}$$

- Top three rows are the affine transform!
- Bottom row stays 1

Homogeneous Vector Transform

- Transform a vector:

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \\ 0 \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} m_{xx}v_x + m_{xy}v_y + m_{xz}v_z + 0 \\ m_{yx}v_x + m_{yy}v_y + m_{yz}v_z + 0 \\ m_{zx}v_x + m_{zy}v_y + m_{zz}v_z + 0 \\ 0+0+0+0 \end{bmatrix}$$

$$\mathbf{M} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

- Top three rows are the linear transform
  - Displacement **d** is properly ignored
- Bottom row stays 0

Homogeneous Arithmetic

- Legal operations always end in 0 or 1!

vector+vector: $\begin{bmatrix}\vdots\\0\end{bmatrix}+\begin{bmatrix}\vdots\\0\end{bmatrix}\Rightarrow\begin{bmatrix}\vdots\\0\end{bmatrix}$

vector-vector: $\begin{bmatrix}\vdots\\0\end{bmatrix}-\begin{bmatrix}\vdots\\0\end{bmatrix}\Rightarrow\begin{bmatrix}\vdots\\0\end{bmatrix}$

scalar*vector: $s\begin{bmatrix}\vdots\\0\end{bmatrix}\Rightarrow\begin{bmatrix}\vdots\\0\end{bmatrix}$

point+vector: $\begin{bmatrix}\vdots\\1\end{bmatrix}+\begin{bmatrix}\vdots\\0\end{bmatrix}\Rightarrow\begin{bmatrix}\vdots\\1\end{bmatrix}$

point-point: $\begin{bmatrix}\vdots\\1\end{bmatrix}-\begin{bmatrix}\vdots\\1\end{bmatrix}\Rightarrow\begin{bmatrix}\vdots\\0\end{bmatrix}$

point+point: $\begin{bmatrix}\vdots\\1\end{bmatrix}+\begin{bmatrix}\vdots\\1\end{bmatrix}\Rightarrow\begin{bmatrix}\vdots\\2\end{bmatrix}$

scalar*point: $s\begin{bmatrix}\vdots\\1\end{bmatrix}\Rightarrow\begin{bmatrix}\vdots\\s\end{bmatrix}$

$\left\{\begin{matrix}\text{weighted average}\\\text{affine combination}\end{matrix}\right\}$ of points: $\frac{1}{3}\begin{bmatrix}\vdots\\1\end{bmatrix}+\frac{2}{3}\begin{bmatrix}\vdots\\1\end{bmatrix}\Rightarrow\begin{bmatrix}\vdots\\1\end{bmatrix}$

Homogeneous Transforms

- Rotation, Scale, and Translation of points and vectors unified in a single matrix transformation:

$$\mathbf{p}' = \mathbf{M}\ \mathbf{p}$$

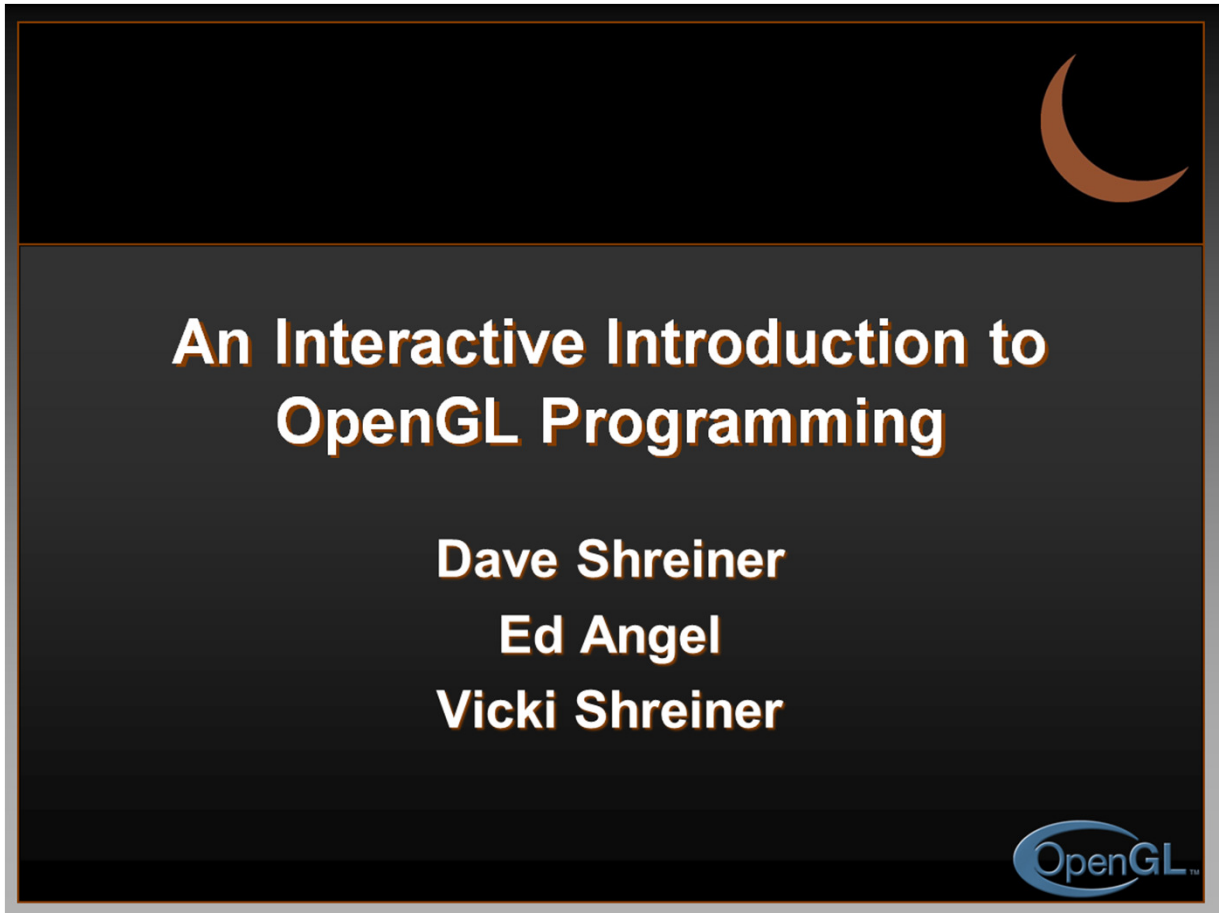
- Matrix has the form:
  - Last row always 0,0,0,1

$$\begin{bmatrix} m_{xx} & m_{xy} & m_{xz} & d_x \\ m_{yx} & m_{yy} & m_{yz} & d_y \\ m_{zx} & m_{zy} & m_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Transforms compose by matrix multiplication!
  - Same caveat: order of operations is important
  - Same note: Transforms operate right-to-left

Introduction to OpenGL

- Using slides from SIGGRAPH course:



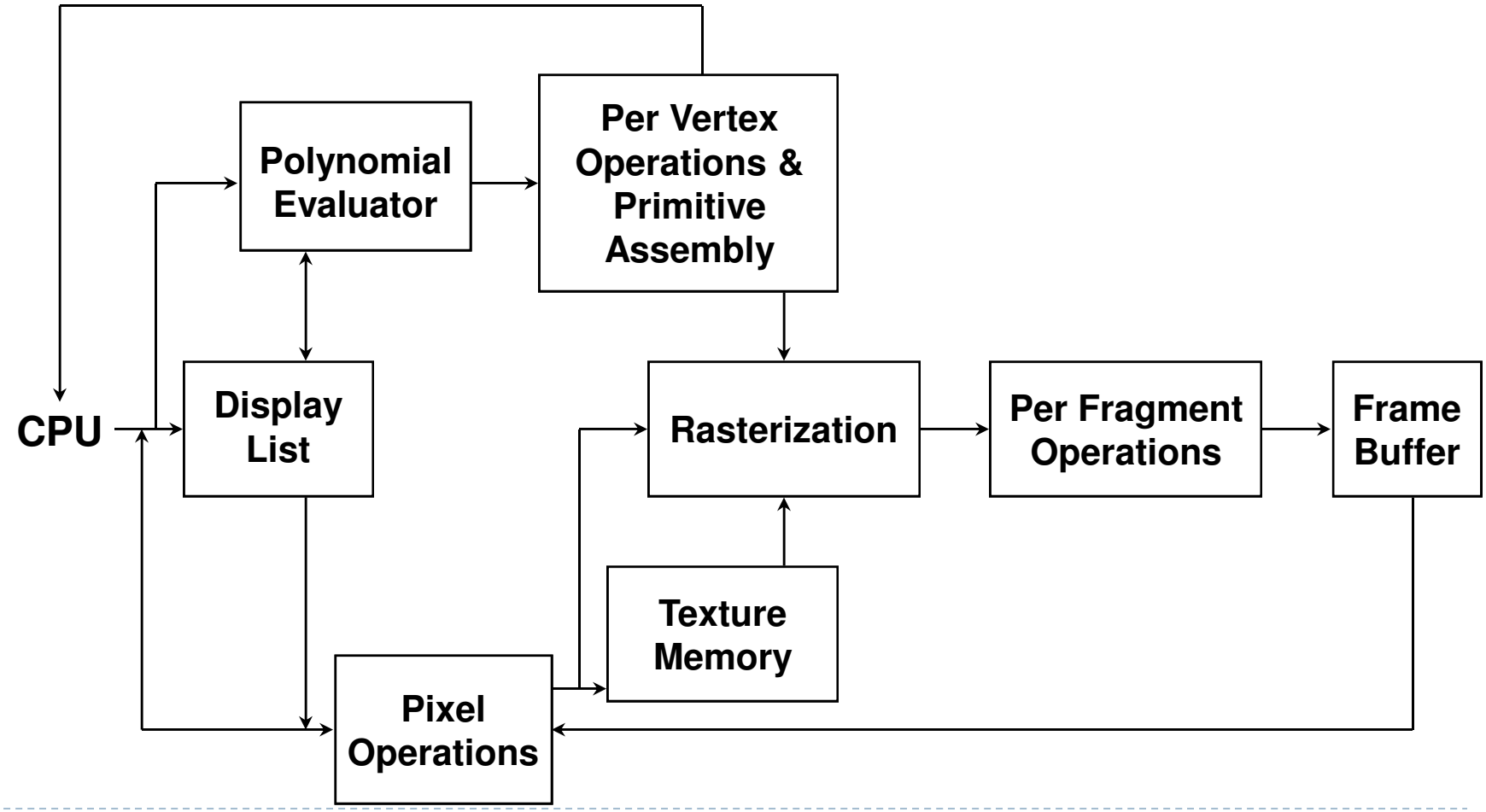
OpenGL and GLUT Overview

- What is OpenGL & what can it do for me?
- OpenGL in windowing systems
- Why GLUT
- A GLUT program template

What Is OpenGL?

- Graphics rendering API
  - high-quality color images composed of geometric and image primitives
  - window system independent
  - operating system independent

OpenGL Architecture



OpenGL as a Renderer

- 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.

Related APIs

- GLU (OpenGL Utility Library)
  - part of OpenGL
  - NURBS, tessellators, quadric shapes, etc.
- GLUT (OpenGL Utility Toolkit)
  - portable windowing API
  - not officially part of OpenGL

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 Basics

- 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

```
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();
}
```

OpenGL Initialization

- Set up whatever state you're going to use

```
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 );
```

Rendering Callback

- Do all of your drawing here

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();
}
```

Idle Callbacks

- Use for animation and continuous update

```
glutIdleFunc( idle );

void idle( void )
{
  t += dt;
  glutPostRedisplay();
}
```

User Input Callbacks

- Process user input

```
glutKeyboardFunc( keyboard );
```

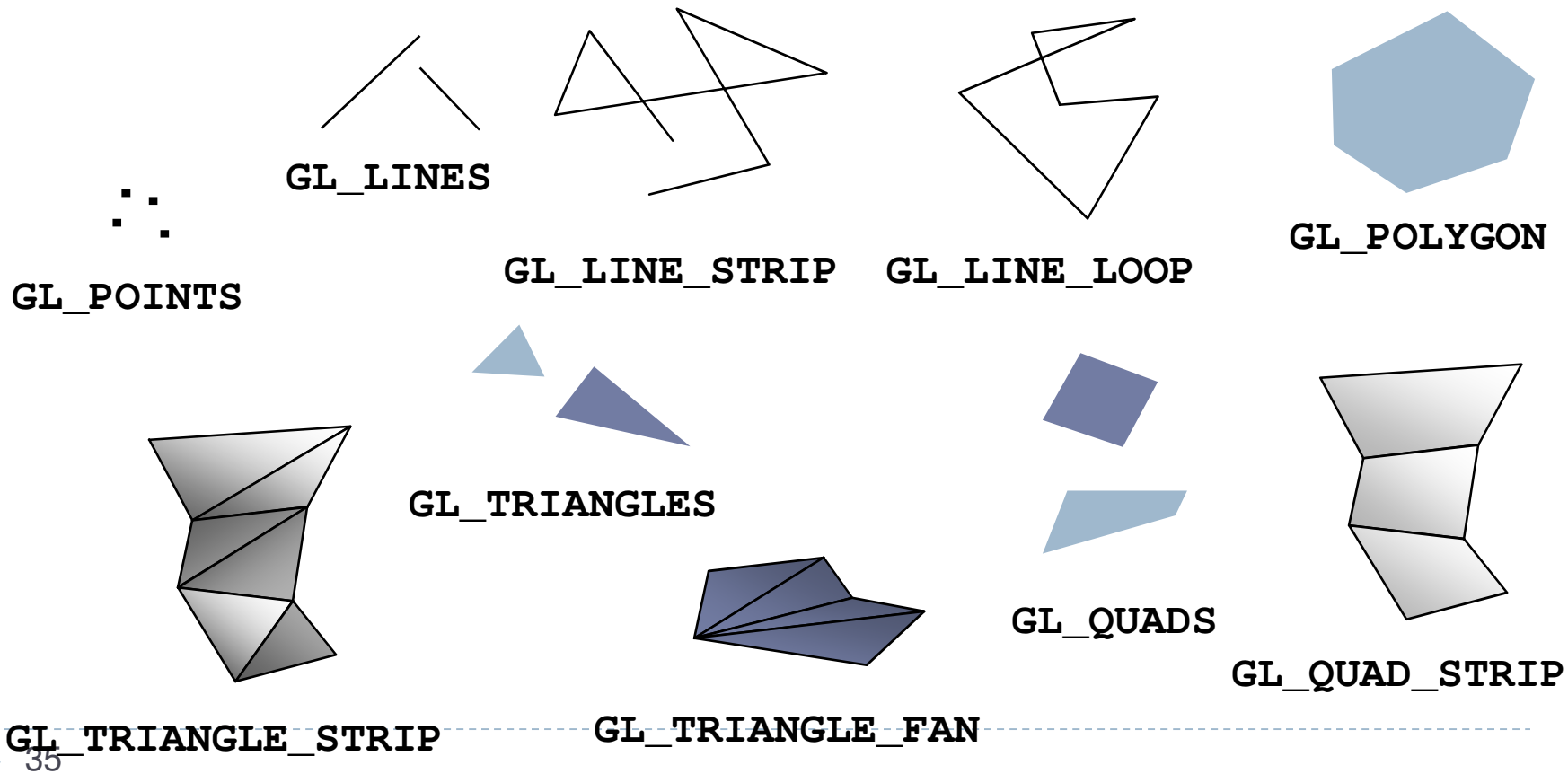
```
void keyboard( char key, int x, int y )
{
  switch( key ) {
    case 'q' : case 'Q' :
      exit( EXIT_SUCCESS );
      break;

    case 'r' : case 'R' :
      rotate = GL_TRUE;
      break;
  }
}
```

Elementary Rendering

- Geometric Primitives
- Managing OpenGL State
- OpenGL Buffers

OpenGL Geometric Primitives

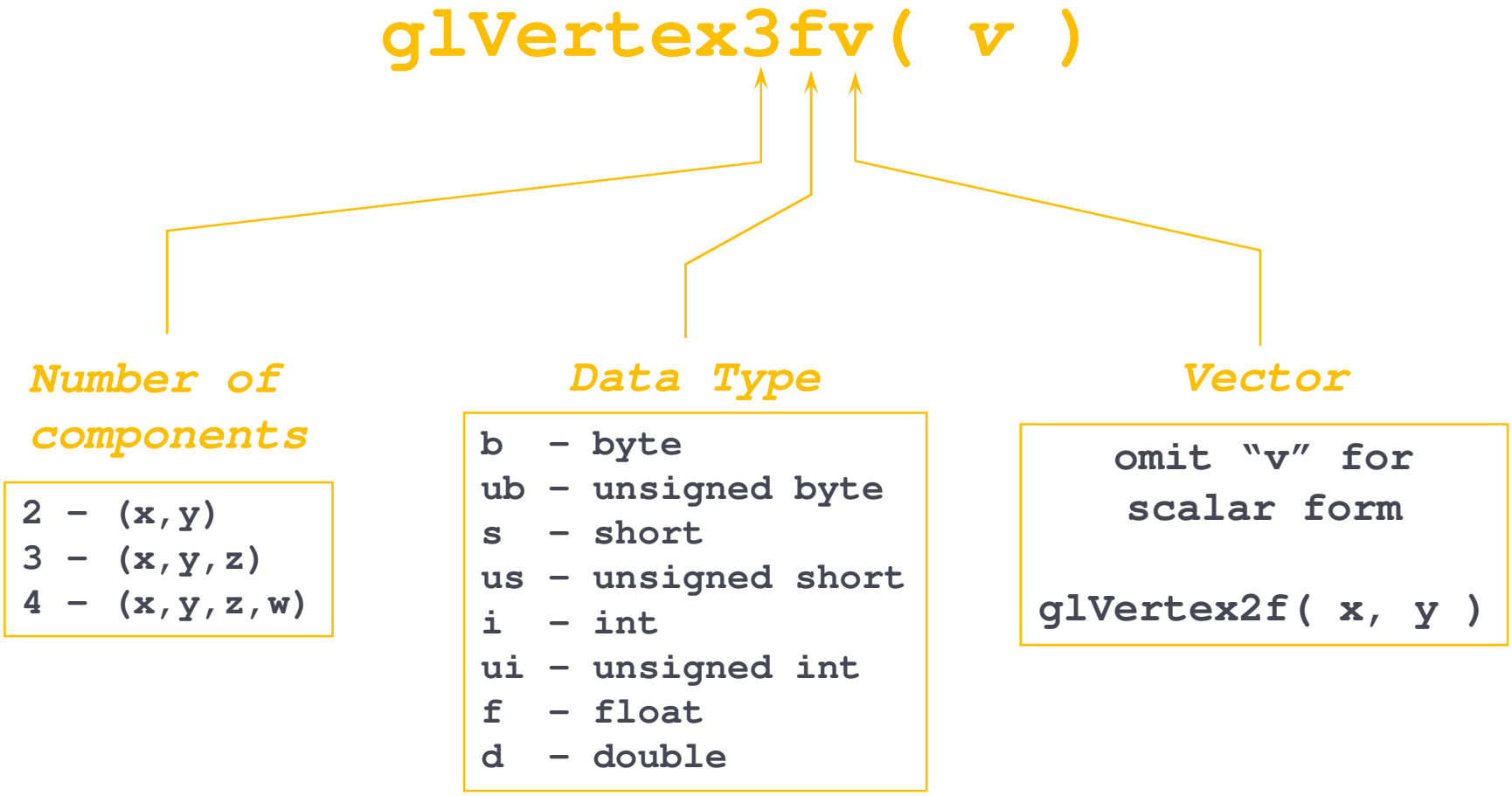
- All geometric primitives are specified by vertices



Simple Example

```
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



Specifying Geometric Primitives

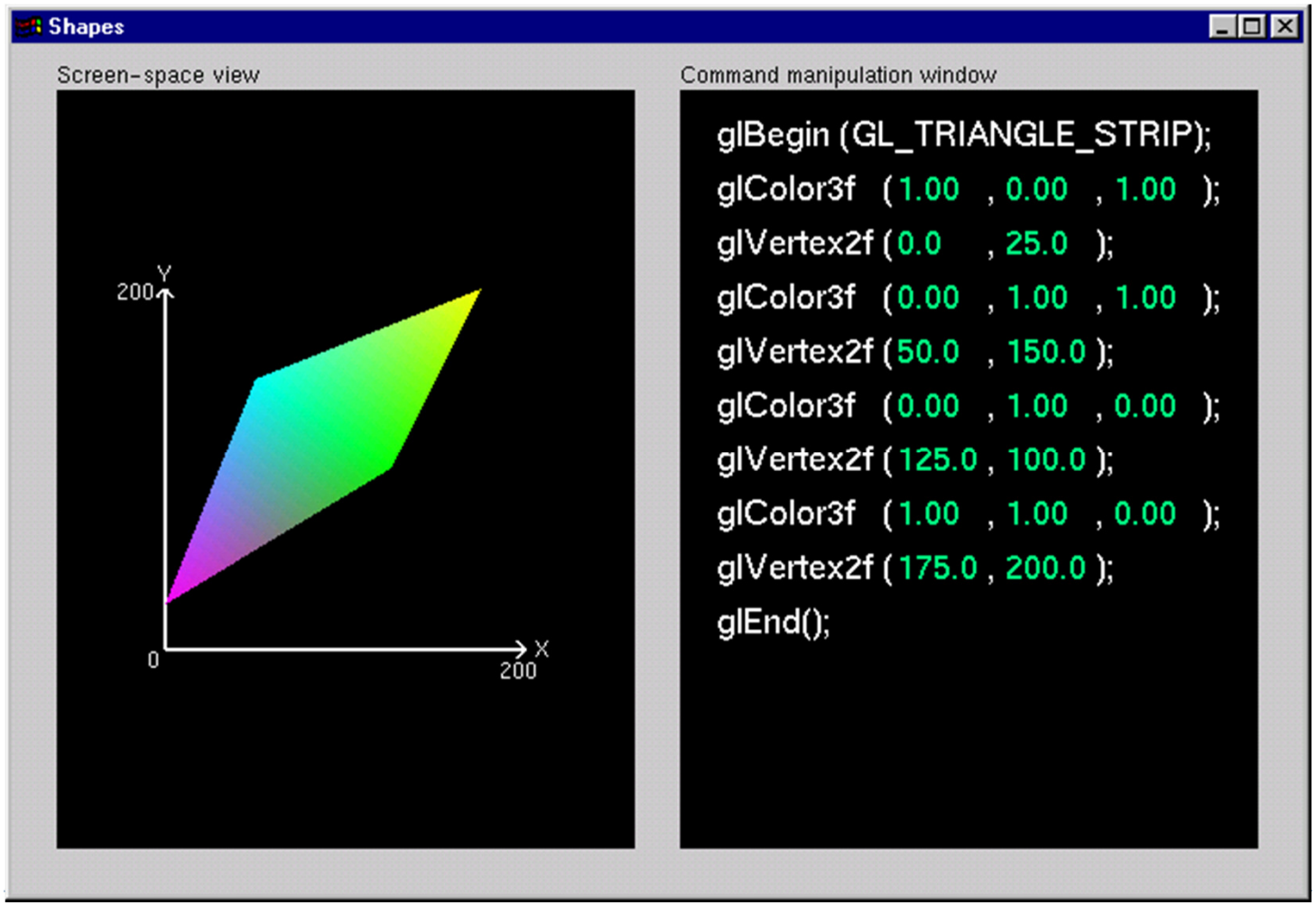
- Primitives are specified using

```
glBegin( primType );
glEnd();
```

  - *primType* determines how vertices are combined

```
GLfloat red, greed, blue;
Glfloat coords[3];
glBegin( primType );
for ( i = 0; i < nVerts; ++i ) {
  glColor3f( red, green, blue );
  glVertex3fv( coords );
}
glEnd();
```

Shapes Tutorial



OpenGL's State Machine

- All rendering attributes are encapsulated in the OpenGL State
  - rendering styles
  - shading
  - lighting
  - texture mapping

Manipulating OpenGL State

- Appearance is controlled by current state

```
for each ( primitive to render )
{
    update OpenGL state
    render primitive
}
```

- Manipulating vertex attributes is most common way to manipulate state

```
glColor*() / glIndex*()
glNormal*()
glTexCoord*()
```

Controlling current state

- Setting State

```
glPointSize( size );
glLineStipple( repeat, pattern );
glShadeModel( GL_SMOOTH );
```

- Enabling Features

```
glEnable( GL_LIGHTING );
glDisable( GL_TEXTURE_2D );
```