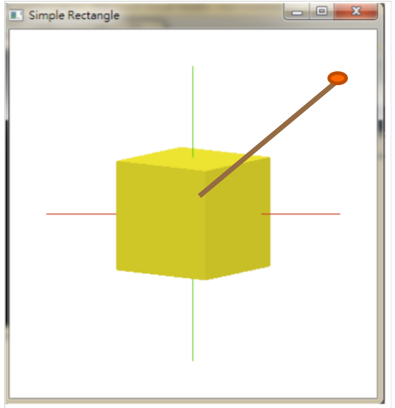
Transformation

PartII

multiply matrix

Goal



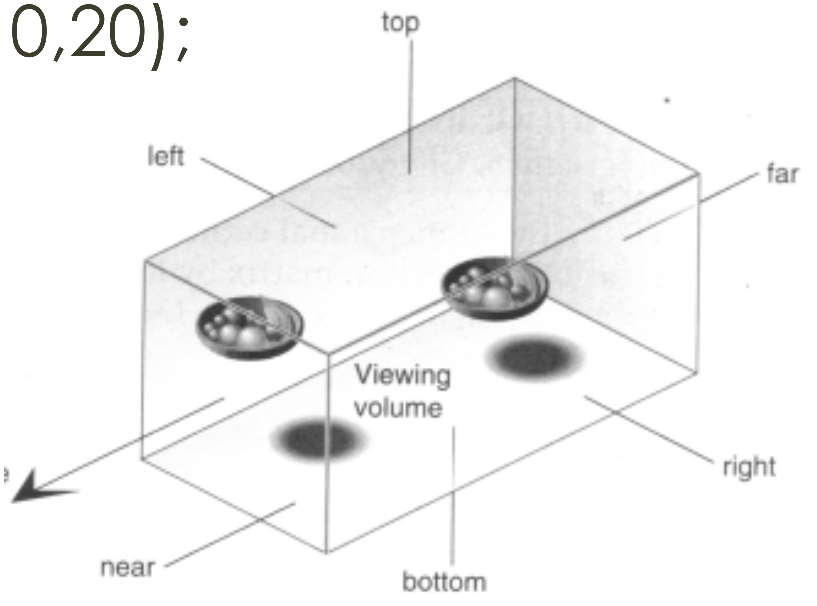
1. Rotate along x, y, z respectively.
   - use your own key setting
2. Translate along x, y, z respectively
   - use your own key setting
3. Reset to origin
   - use your own key setting
4. Arbitrary Rotation:
   - Draw the last dot where your mouse click on (20%)
   - Draw the line between the origin and the last dot (10%)
   - Rotate along the line

- Write comments in your code about your key setting
- Do not use glRotate, glTranslate in your code
- Turn in your code

Note: The rotational matrix is provided in this pdf (for arbitrary rotation), use it for this Lab assignment.

glOrtho

- glOrtho(-10,10,-10,10,-10,20);

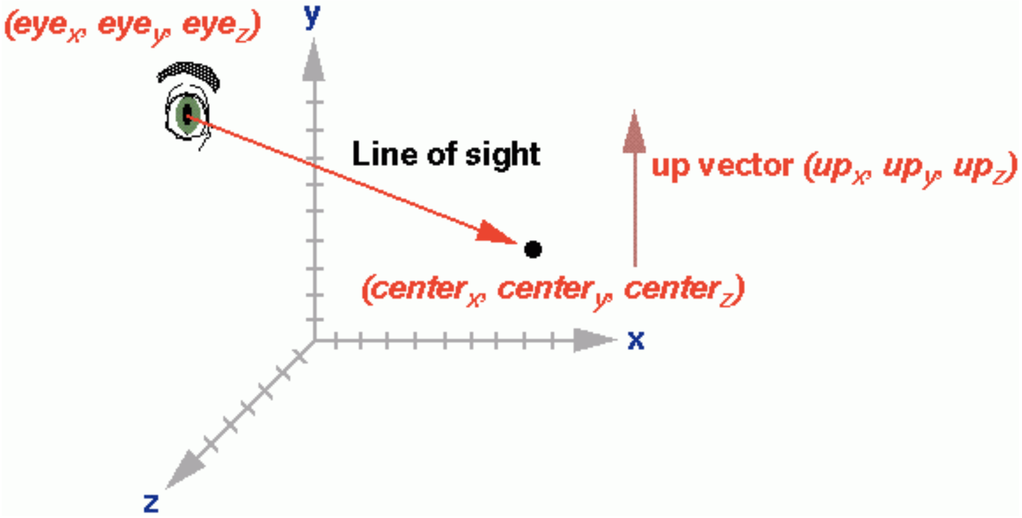


The last two parameters specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

https://www.opengl.org/sdk/docs/man2/xhtml/glOrtho.xml

gluLookAt

- gluLookAt(0,0,10.0f ,0,0,0, 0,1,0);



(eye_x, eye_y, eye_z)
y
Line of sight
up vector (up_x, up_y, up_z)
$(center_x, center_y, center_z)$
x
z

```
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 glRotatef(angle, 1,0,0);
 glTranslatef(tx,ty,tz);
//draw the object
glutSolidCube(6);
```

glMultiMatrix

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity;
glMultMatrixf(rotMatrix);
glMultMatrixf(translateMatrix);

//draw_the_object
glutSolidCube(6);
```

```
GLfloat rotMatrix[] = {
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        0.0, 0.0, 0.0, 1.0   };
```

https://www.opengl.org/sdk/docs/man2/xhtml/glRotate.xml

glRotatef(angel, x, y,z)

produces a rotation of `angle` degrees around the vector $x\ y\ z$.

$$\begin{pmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ yx(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ xz(1-c)-ys & yz(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Where $c = cos\ (angle)$, $s = sin(angle)$, and $||(x, y, z)|| = 1$ (if not, the GL will normalize this vector).

http://www.euclideanspace.com/maths/geometry/rotations/conversions/angleToMatrix/index.htm

```
public void rotationMatrix() {
    double c = cos(angle);
    double s = sin(angle);
    double t = 1.0 - c;

     //  if axis is not already normalised then uncomment this
     // double magnitude = sqrt(x*x + y*y + z*z);
     // if (magnitude==0) throw error;
     // x /= magnitude;
     // y /= magnitude;
     // z /= magnitude;

    m00 = c + x*x*t;
    m11 = c + y*y*t;
    m22 = c + z*z*t;

    m10 = x*y*t + z*s;
    m01 = x*y*t - z*s;

    m20 = x*z*t - y*s;
    m02 = x*z*t + y*s;

    m21 = y*z*t + x*s;
    m12 = y*z*t - x*s;
}
```

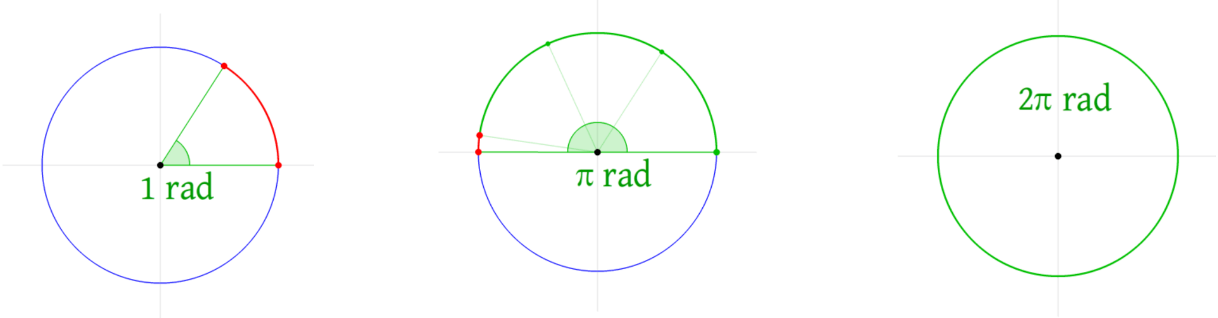
Example code (concept only)

Degree to radians conversion

```
#define PI 3.14159265

int main ()
{
    double  degree, result;
    degree = 60.0;
    result = cos ( degree * PI / 180.0 );   //   = 2PI /360
    printf ("The cosine of %f degrees is %f.\n", degree, result );
    return 0;

}
```



360 degree = 2PI

radian: the length of a corresponding arc of a unit circle

Transformation Matrix

- All modeling transformations are represented as 4x4 matrices

- Identity matrix

```
GLfloat rotMatrix[] = {
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        0.0, 0.0, 0.0, 1.0   };
```

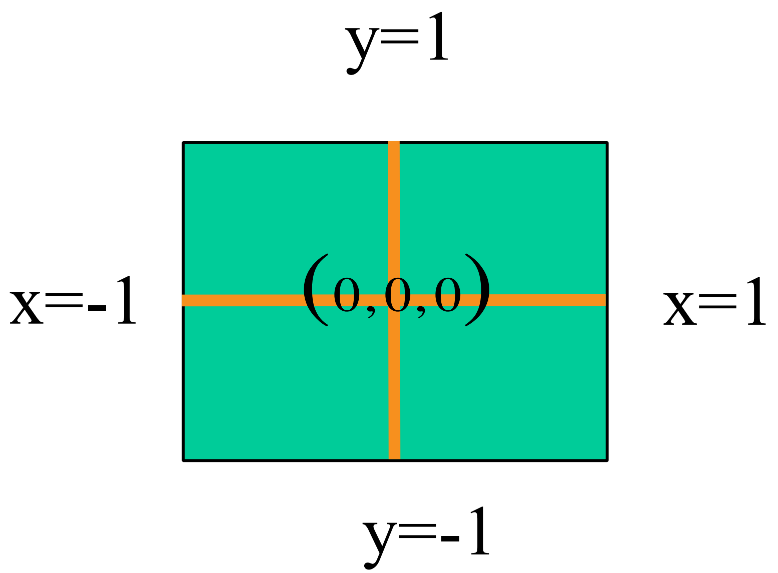
```
rotMatrix[0] = 1;  rotMatrix[4] = 0;   rotMatrix[8] = 0;  rotMatrix[12] = 0;
rotMatrix[1] = 0;  rotMatrix[5] = 1;   rotMatrix[9] = 0;  rotMatrix[13] = 0;
rotMatrix[2] = 0;  rotMatrix[6] = 0;  rotMatrix[10] = 1;  rotMatrix[14] = 0;
rotMatrix[3] = 0;  rotMatrix[7] = 0;  rotMatrix[11] = 0;  rotMatrix[15] = 1;
```

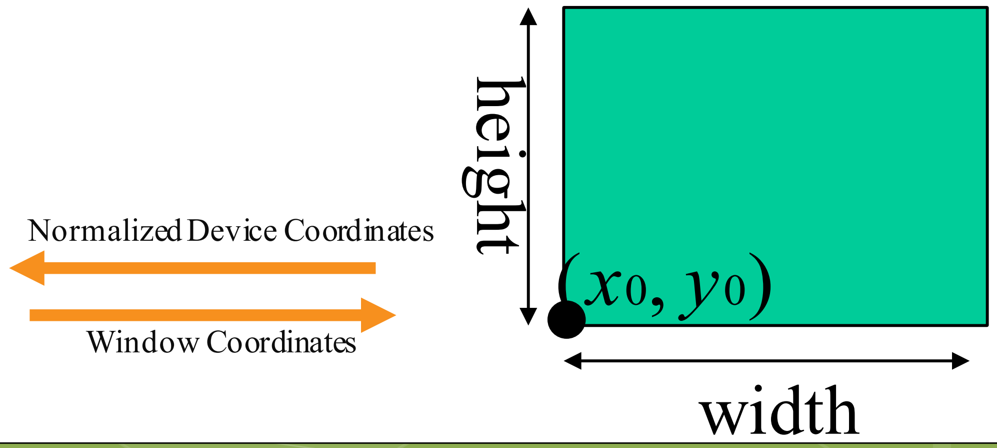
Mouse Click Location

- Click at (win$_x$, Win$_y$)
- Convert it to OpenGL's coordinate (x, y)
- Draw the dot

$$x_w = (x_{nd} + 1)\left(\frac{width}{2}\right) + x_0$$

$$y_w = (y_{nd} + 1)\left(\frac{height}{2}\right) + y_0$$


y=1
x=-1
(0,0,0)
x=1
y=-1


Normalized Device Coordinates
Window Coordinates
height
(x0, y0)
width