COSC 4332 Computer Graphics

Lab 2
3D and Interaction

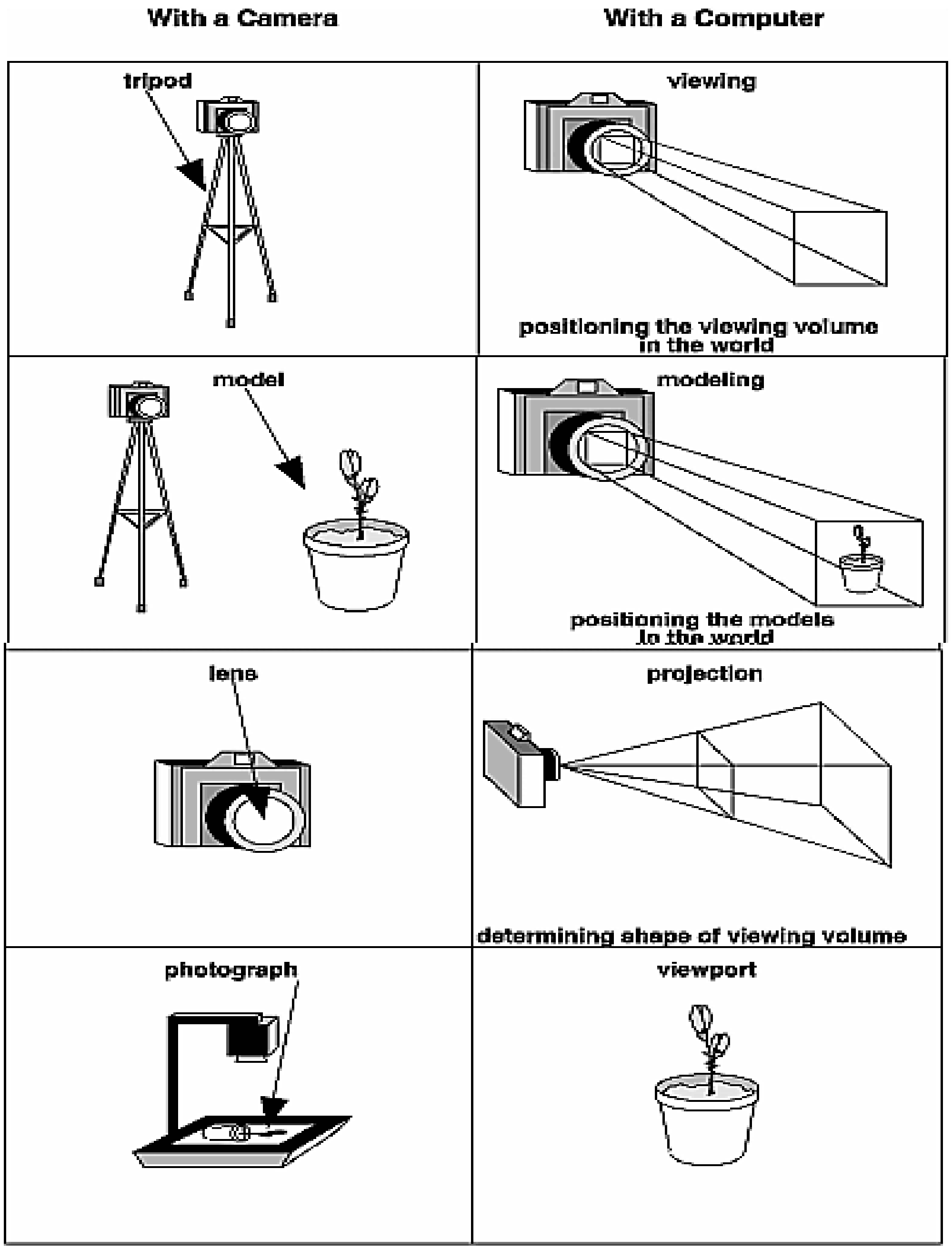
Dr. Khaled Rabieh

Outline

1. OpenGL

    1. 3D basics and Keyboard/ Mouse Interaction

The Camera Analogy



Set up your tripod and pointing the camera at the scene (viewing transformation).

Arrange the scene to be photographed into the desired composition (modeling transformation).

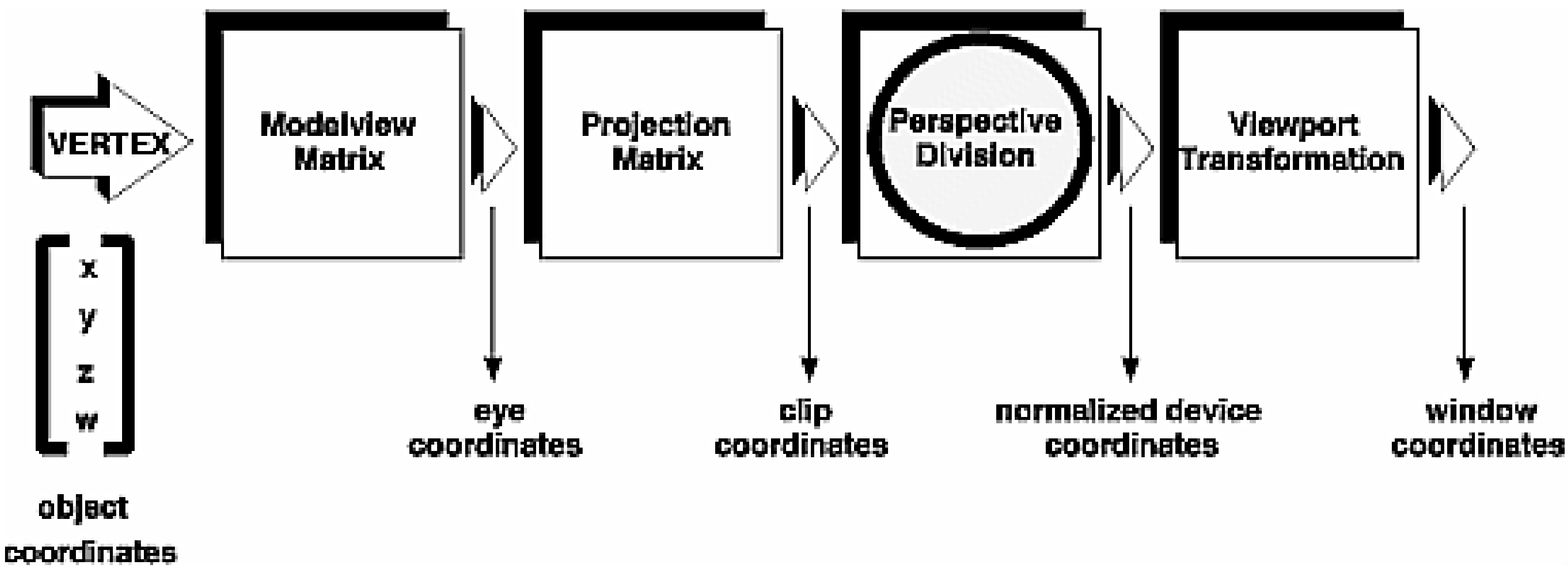
Choose a camera lens or adjust the zoom (projection transformation).

Determine how large you want the final photograph to be - for example, you might want it enlarged (viewport transformation).

http://www.glprogramming.com/red/chapter03.html#name1

The Vertex Transformations

To specify viewing, modeling, and projection transformations, you construct a 4 * 4 matrix **M**, which is then multiplied by the coordinates of each vertex *v* in the scene to accomplish the transformation **v'=Mv**



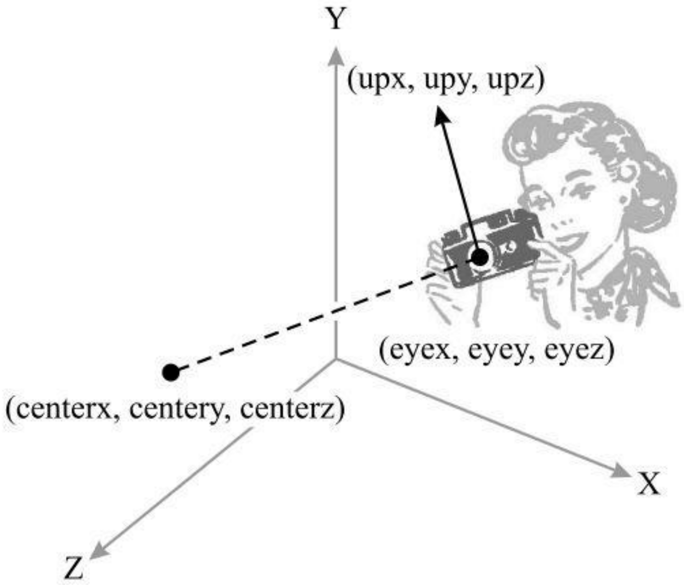
The viewing and modeling transformations you specify are combined to form the modelview matrix

Viewing and Modeling Transformation

```
void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glLoadIdentity ();              /* clear the matrix */
           /* viewing transformation  */
   gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
   glScalef (1.0, 2.0, 1.0);      /* modeling transformation */
   glutWireCube (1.0);
   glFlush ();
}
```

gluLookAt(). The arguments for this command indicate

- The camera / eye position
- where the camera looks
- Which way is up.
- The arguments used here place the camera at (0, 0, 5), aim the camera lens towards (0, 0, 0), and specify the up-vector as (0, 1, 0). The up-vector defines a unique orientation for the camera.



The Projection Transformation

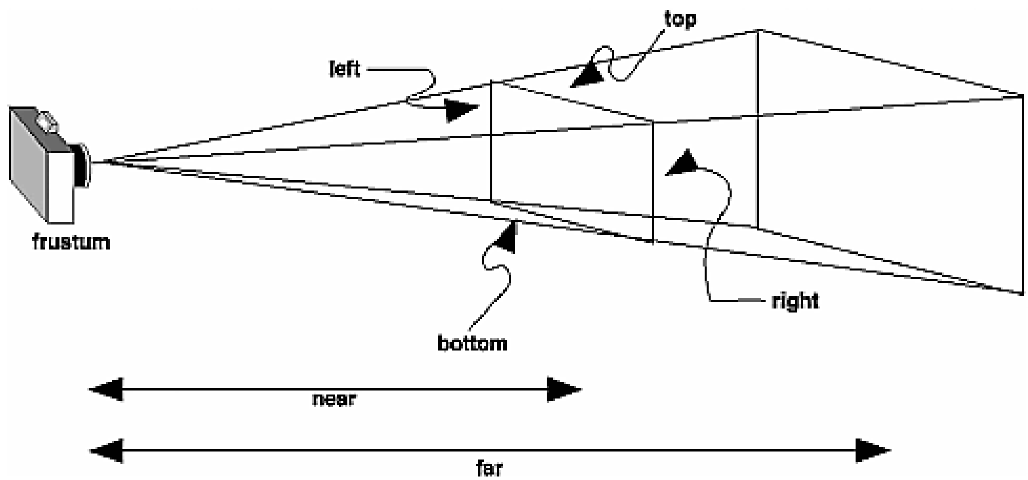
- Specifying the projection transformation is like choosing a lens for a camera.
- Think of this transformation as determining what the field of view is
  - what objects are inside it and to some extent how they look
- This is equivalent to choosing among
  - wide-angle,normal, and telephoto lenses
- For example, with a wide-angle lens, you can include a wider scene in the final photograph than with a telephoto lens, but a telephoto lens allows you to photograph objects as though they're close

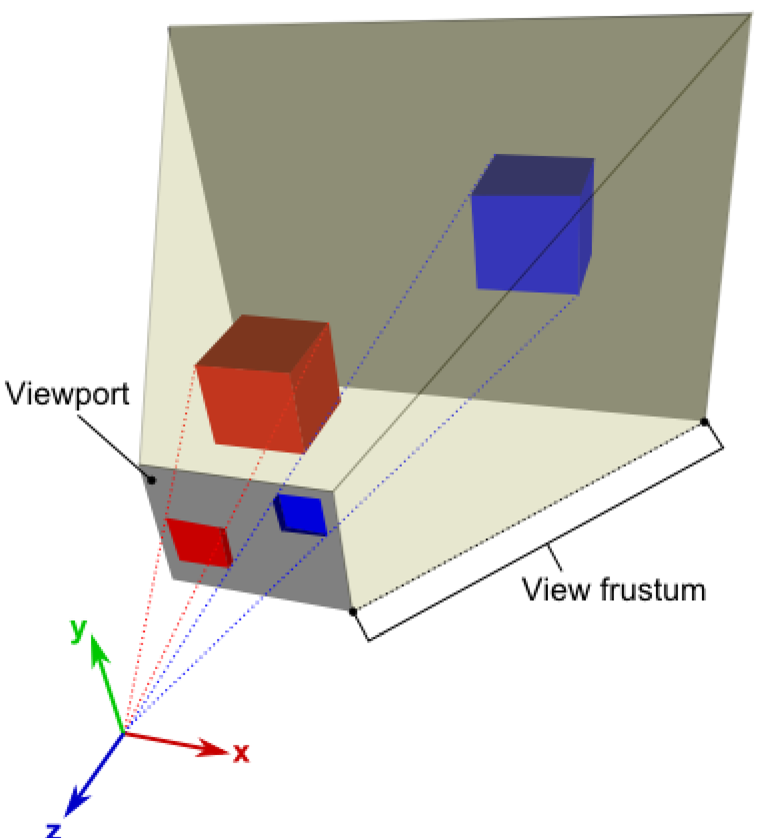
The Projection Transformation

- Two basic types of projections are provided for you by OpenGL
- *Perspective projection*
  - Matches how you see things in daily life.
  - Makes objects that are farther away appear smaller
  - for example, it makes railroad tracks appear to converge in the distance.
- *Orthographic projection*
  - Maps objects directly onto the screen without affecting their relative size.

Perspective Projection

glFrustum(*l, r, b, t, n, f*)

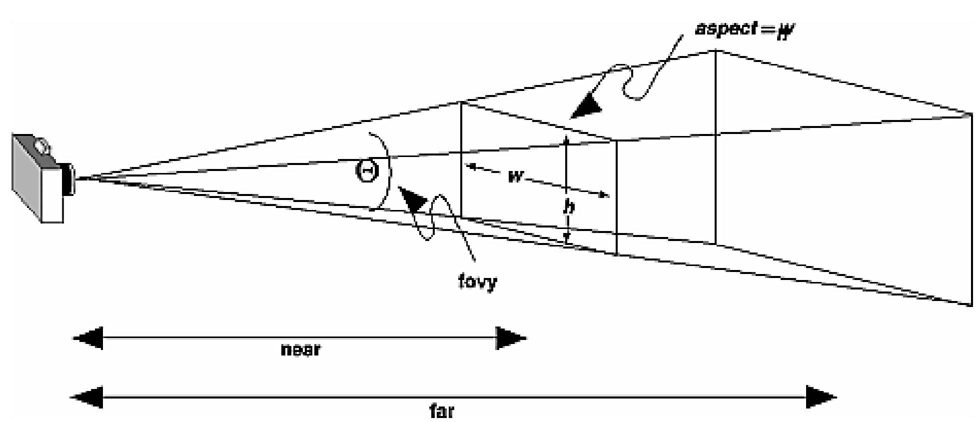




http://www.xojo3d.com/tut002.php

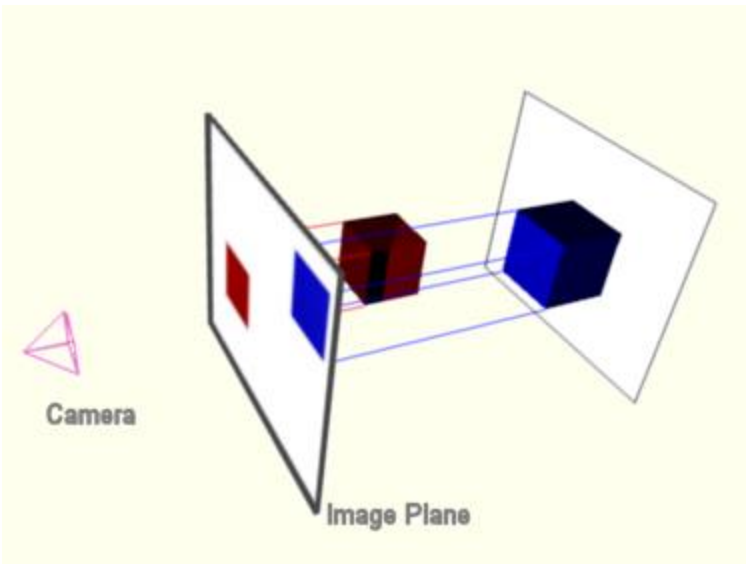
gluPerspective

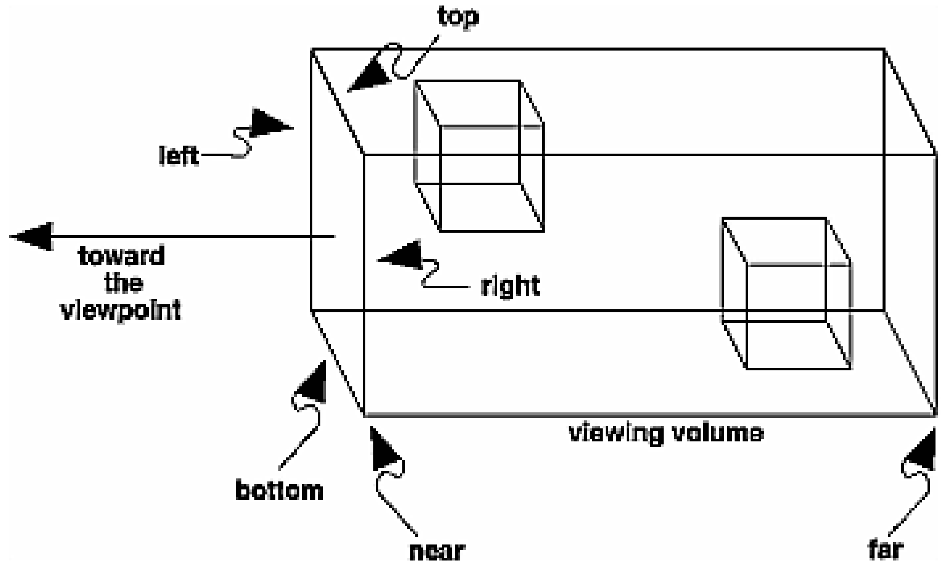
- **gluPerspective**(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);
  - Creates a matrix for a symmetric perspective-view frustum and multiplies the current matrix by it.
  1. fovy is the angle of the field of view in the x-z plane; its value must be in the range [0.0,180.0].
  2. Aspect is the aspect ratio of the frustum, its width divided by its height.
  3. Near and far values the distances between the viewpoint and the clipping planes, along the negative z-axis. They should always be positive



Orthographic Projection

glOrtho(*l, r, b, t, n, f* **)**





$$R = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } R^{-1} = \begin{bmatrix} \frac{r-l}{2} & 0 & 0 & \frac{r+l}{2} \\ 0 & \frac{t-b}{2} & 0 & \frac{t+b}{2} \\ 0 & 0 & \frac{f-n}{-2} & \frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

General Transformation Commands

- **glMatrixMode()**
  - Specifies whether the modelview, projection, or texture matrix will be modified, using the argument GL_MODELVIEW, GL_PROJECTION, or GL_TEXTURE for mode.
  - Note that only one matrix can be modified at a time.
  - By default, the modelview matrix is the one that's modifiable, and all three matrices contain the identity matrix.
- void **glLoadIdentity**(void);
  - Sets the currently modifiable matrix to the 4×4 identity matrix.

Load and Mult Matrices

void ***glLoadMatrix****{fd}(const TYPE *m);*

Sets the sixteen values of the current matrix to those specified by m.

void ***glMultMatrix****{fd}(const TYPE *m);*

Multiplies the matrix specified by the sixteen values pointed to by m by the current matrix and stores the result as the current matrix.

Mathematical and Programming Notes

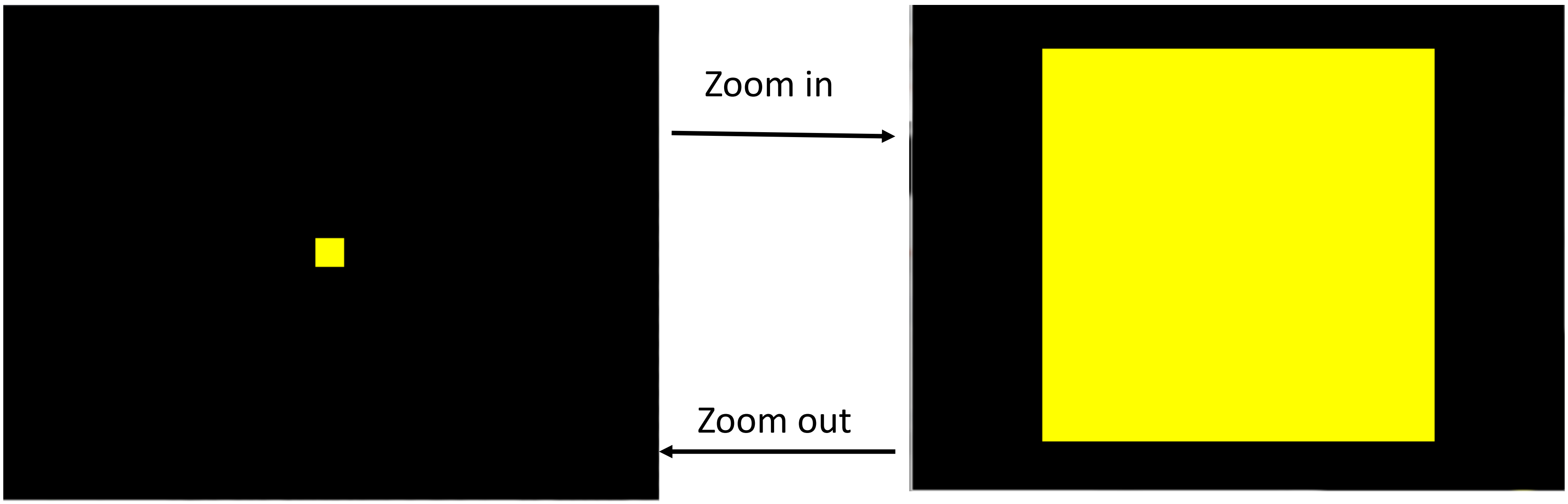
- OpenGL uses column instead of row vectors
- Let **C** be the current matrix and call **glMultMatrix*(M)**.

  After multiplication, the final matrix is always **CM**.
- Matrices are defined like this

$$M = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

Interaction

Write a program to zoom in and zoom out according to the user mouse motion



The ModelView Matrix

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear color and depth buffers
    glMatrixMode(GL_MODELVIEW);      // To operate on model-view matrix
                                     // Render a color-cube consisting of 6 quads with different colors
    glLoadIdentity();                  // Reset the model-view matrix

    gluLookAt(0, 0, -g_fViewDistance, 0, 0, -1, 0, 1, 0);

    glBegin(GL_QUADS);                 // Begin drawing the color cube with 6 quads
                                       // Top face (y = 1.0f)
                                       // Define vertices in counter-clockwise (CCW) order with normal pointing out
    glColor3f(0.0f, 1.0f, 0.0f);     // Green
    glVertex3f(1.0f, 1.0f, -1.0f);
    glVertex3f(-1.0f, 1.0f, -1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(1.0f, 1.0f, 1.0f);

    // Bottom face (y = -1.0f)
    glColor3f(1.0f, 0.5f, 0.0f);     // Orange
    glVertex3f(1.0f, -1.0f, 1.0f);
    glVertex3f(-1.0f, -1.0f, 1.0f);
    glVertex3f(-1.0f, -1.0f, -1.0f);
    glVertex3f(1.0f, -1.0f, -1.0f);

    // Front face  (z = 1.0f)
    glColor3f(1.0f, 0.0f, 0.0f);     // Red


    // Right face (x = 1.0f)
    glColor3f(1.0f, 0.0f, 1.0f);     // Magenta
    glVertex3f(1.0f, 1.0f, -1.0f);
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(1.0f, -1.0f, 1.0f);
    glVertex3f(1.0f, -1.0f, -1.0f);
    glEnd();  // End of drawing color-cube


    glutSwapBuffers();  // Swap the front and back frame buffers (double buffering)
```

The Perspective Matrix

```
void reshape(GLsizei width, GLsizei height) {  // GLsizei for non-negative integer
                                               // Compute aspect ratio of the new window
    if (height == 0) height = 1;                // To prevent divide by 0
    GLfloat aspect = (GLfloat)width / (GLfloat)height;

    // Set the viewport to cover the new window
    glViewport(0, 0, width, height);

    // Set the aspect ratio of the clipping volume to match the viewport
    glMatrixMode(GL_PROJECTION);  // To operate on the Projection matrix
    glLoadIdentity();             // Reset
    gluPerspective(45.0f, aspect, 0.01f, 100.0f); // Enable perspective projection with fovy, aspect, zNear and zFar
}
```

Mouse Interaction

- Register two functions for mouse
  - Mouse press and mouse movement

```
/* Main function: GLUT runs as a console application starting at main() */
int main(int argc, char** argv) {
    glutInit(&argc, argv);                   // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE);        // Enable double buffered mode
    glutInitWindowSize(640, 480);            // Set the window's initial width & height
    glutInitWindowPosition(50, 50);          // Position the window's initial top-left corner
    glutCreateWindow(title);                 // Create window with the given title
    glutDisplayFunc(display);                // Register callback handler for window re-paint event
    glutReshapeFunc(reshape);                // Register callback handler for window re-size event
    glutMouseFunc(MouseButton);
    glutMotionFunc(MouseMotion);
    initGL();                                // Our own OpenGL initialization
    glutMainLoop();                          // Enter the infinite event-processing loop
    return 0;
}
```

Mouse motion and click capture events

```
void MouseButton(int button, int state, int x, int y)
{
    // Respond to mouse button presses.
    // If button1 pressed, mark this state so we know in motion function.
    if (button == GLUT_LEFT_BUTTON)
    {
        g_bButton1Down = (state == GLUT_DOWN) ? TRUE : FALSE;
        g_yClick = y -  g_fViewDistance;
        printf("g_yClick is %d\n", g_yClick);

    }
}
void MouseMotion(int x, int y)
{
    // If button1 pressed, zoom in/out if mouse is moved up/down.
    if (g_bButton1Down)
    {
        g_fViewDistance = (y - g_yClick);
        printf("The eye location, y are : %f and %d\n", -g_fViewDistance,y);
        if (g_fViewDistance < 0)
            g_fViewDistance = 0;
        glutPostRedisplay();
    }
}
```

Thank You

Questions

Khaled Rabieh