Computer Graphics

Chapter 7
2D Geometric Transformations

Chapter 7
Two-Dimensional Geometric Transformations

Part III.

- OpenGL Functions for Two-Dimensional Geometric Transformations
- OpenGL Geometric Transformation Programming Examples

OpenGL Geometric Transformation Functions

- Be careful of manipulating the matrix in OpenGL
  - OpenGL uses 4X4 matrix for transformation.
  - The 16 elements are stored as 1D in *column-major order*

$$\begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}$$

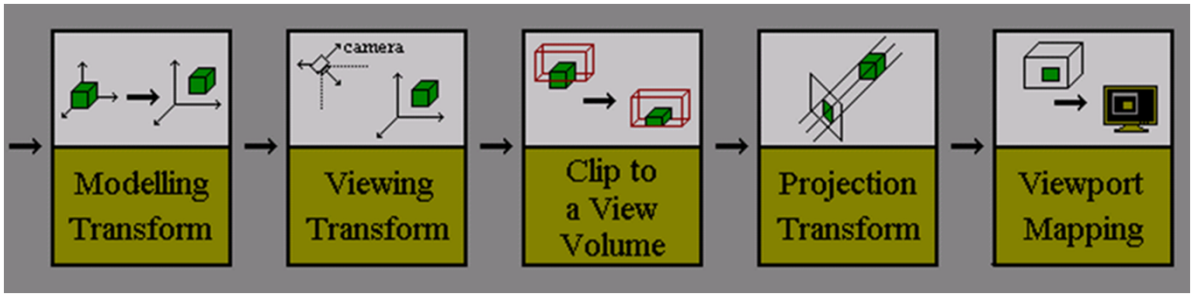
OpenGL transform matrix

  - C and C++ store matrices in *row-major order*
  - If you declare a matrix to be used in OpenGL as

    GLfloat M[4][4]; to access the element in row i and column j, you need to refer to it by M[j][i]; or, as

    GLfloat M[16]; and then you need to convert it to conventional *row-major order*.

OpenGL Transformations

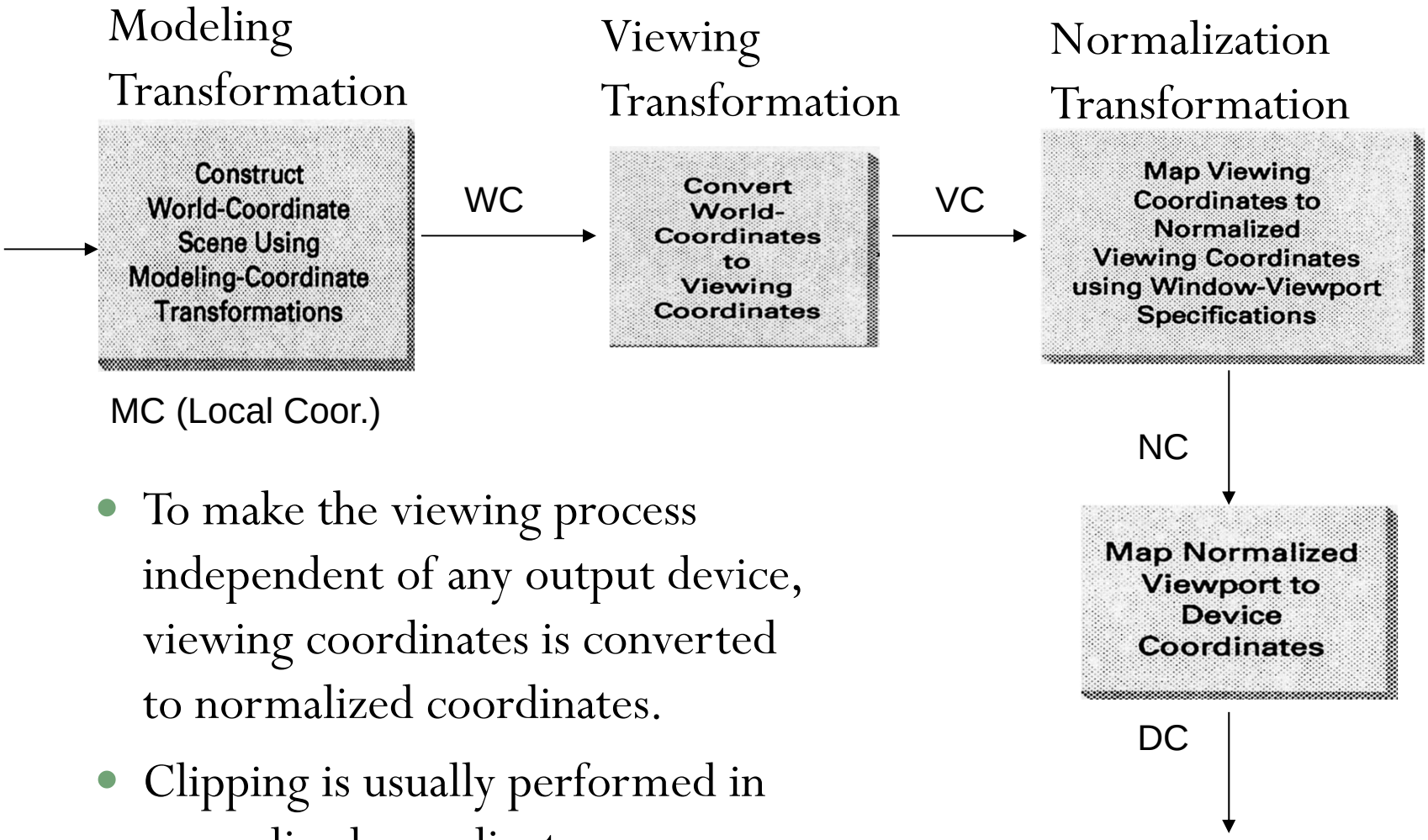


- Three types
  - Modeling, Viewing and Projection

Transformation	Use
Viewing	Specifies the location of the viewer or camera
Modeling	Moves objects around scene
Modelview	Describes the duality of viewing and modeling transformations
Projection	Clips and sizes the viewing volume
Viewport	Scales final output to the window

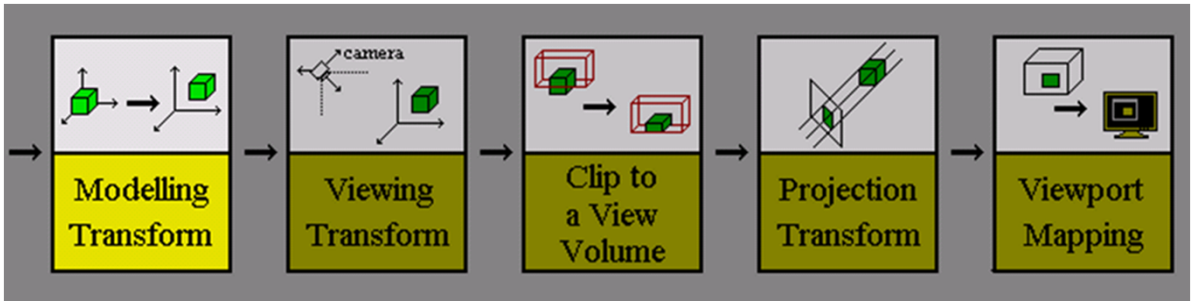
(From OpenGL Super Bible)

Standard 2D Viewing Pipeline



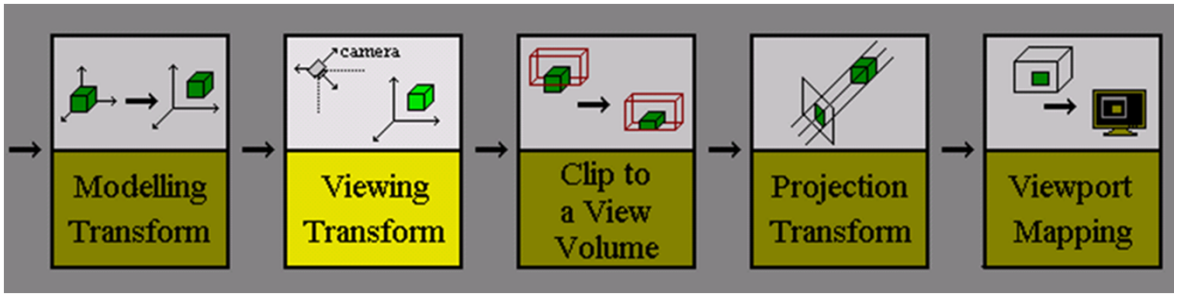
- To make the viewing process independent of any output device, viewing coordinates is converted to normalized coordinates.
- Clipping is usually performed in normalized coordinates.

Modeling Transformations



- *Modeling transformations:* to manipulate/create your model and the particular objects within it.
  - Move objects into place, rotates them, and scales them, etc.
  - The final appearance of your scene or object can depend greatly **on the order** in which the modeling transformations are applied.

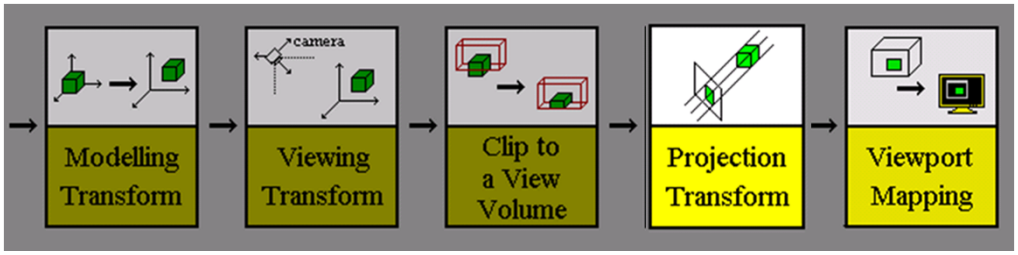
Viewing Transformation



- *Viewing transformation:* to place & point a camera to view the scene.
  - By default, the point of observation is at the origin (0,0,0) looking down the negative z-axis ("into" the monitor screen).
    - Objects drawn with positive z values would be behind the observer.
  - You can put the point of observation anywhere you want, and looking in any direction.

Transformation demo - Nate

Projection and Viewport Transformations



- *Projection transformation*: applied to your final Modelview orientation in which way to project.
  - Defines how a constructed scene (after all the modeling is done) is translated to the final 2D image on the viewing plane.
  - Defines the viewing volume and establishes clipping planes.
  - Two types
    - *Orthographic*
    - *Perspective*
- *Viewport transformation:* maps the 2D projection result of your scene to a window somewhere on your screen.

OpenGL Transformations

- In OpenGL, all the transformations are described as a **multiplication of matrices**.
  - The mathematics behind these transformations are greatly simplified by the mathematical notation of the matrix.
  - Each of the transformations can be achieved by multiplying a **matrix** that contains the vertices, by a **matrix** that describes the transformation.

OpenGL Geometric Transformation Functions

- OpenGL matrix operation function

  void **glMatrixMode**(Glenum *mode*);

  - Specify which matrix is the current matrix
  - *mode:* GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE;

    GL_COLOR (if ARB_imaging extension is supported).

    e.g.: glMatrixMode (GL_MODELVIEW);

- OpenGL matrix operations

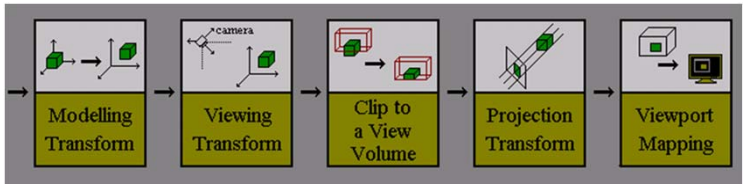
  glMatrixMode (GL_MODELVIEW); → Set up the matrix for geometric transformations

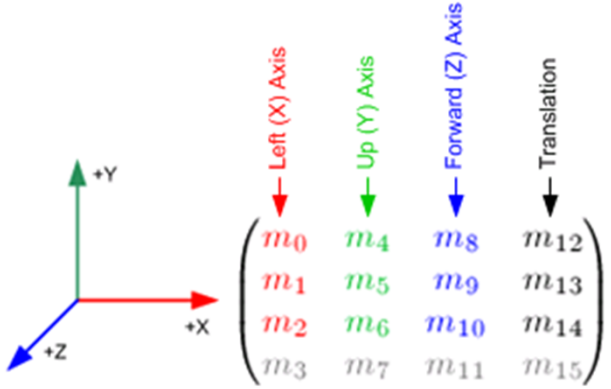
  glLoadIdentity (); // assign identity matrix to the current matrix

  //… to apply any transformation matrix to transform your scene …

Model-View Matrix: GL_MODELVIEW

- GL_MODELVIEW



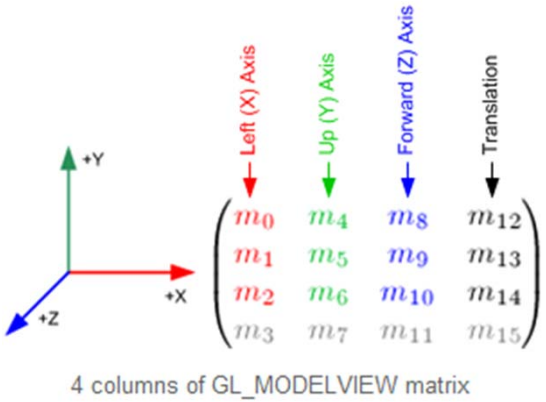


4 columns of GL_MODELVIEW matrix

- Store and combine the geometric transformations to models and viewing-coordinate system
  - Combine viewing matrix and modeling matrix into one matrix
- Viewing transformation
  - For example: **gluLookAt()**
- Modeling transformation: OpenGL transformation functions
  - Translation transformation: *m12, m13, m14*
  - Other Euclidean/affine transformations, such as rotation or scaling: *(m0, m1, m2), (m4, m5, m6) and (m8, m9, m10)*

Model-View Matrix: GL_MODELVIEW

- GL_MODELVIEW



4 columns of GL_MODELVIEW matrix

$$M = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

glMatrixMode (GL_MODELVIEW);

glLoadIdentity (); // assign identity matrix to the current matrix

These 3 sets

- (*m0*, *m1*, *m2*) : +X axis, *left* vector, (1, 0, 0) by default
- (*m4*, *m5*, *m6*) : +Y axis, *up* vector, (0, 1, 0) by default
- (*m8*, *m9*, *m10*) : +Z axis, *forward* vector, (0, 0, 1) by default

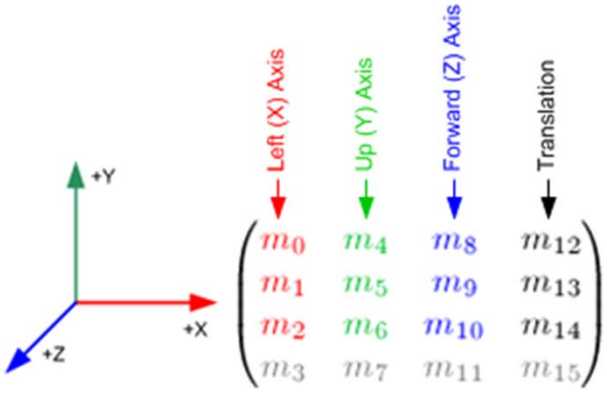
are actually representing 3 orthogonal axes.

glLoadMatrix* (elements16); // replace the current matrix by your own

Model-View Matrix: GL_MODELVIEW

Example

```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
GLfloat elems [16];
GLint k;
for (k = 0; k < 16; k++)
        elems [k] = float (k);
glLoadMatrixf (elems);
```



4 columns of GL_MODELVIEW matrix

$$M = \begin{bmatrix} 0.0 & 4.0 & 8.0 & 12.0 \\ 1.0 & 5.0 & 9.0 & 13.0 \\ 2.0 & 6.0 & 10.0 & 14.0 \\ 3.0 & 7.0 & 11.0 & 15.0 \end{bmatrix}$$

glMatrixMode()

http://www.opengl.org/sdk/docs/man/

OpenGL Geometric Transformation Functions

- Basic OpenGL geometric transformations on the matrix:

glTranslate* (tx, ty, tz);

[glTranslatef (25.0, -10.0, 0.0);] for 2D, set tz = 0.

- Post-multiplies the current matrix by a matrix that moves the object by the given x-, y-, and z-values

glScale* (sx, sy, sz);

[glScalef (2.0, -3.0, 1.0);]

- Post-multiplies the current matrix by a matrix that scales an object about the origin. **None** of sx, sy or sz is **zero**.

glRotate* (theta, vx, vy, vz);

[glRotatef (90.0, 0.0, 0.0, 1.0);]

- Post-multiplies the current matrix by a matrix that rotates the object in a counterclockwise direction. vector v=(vx, vy, vz) defines the orientation for the rotation axis that passes though the coordinate origin. (the rotation center is (0, 0, 0))

OpenGL: Order in Matrix Multiplication

```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ( );       //Set current matrix to the identity.
glMultMatrixf (elemsM2); //Post-multiply identity by matrix M2.
glMultMatrixf (elemsM1); //Post-multiply M2 by matrix M1.
glBegin (GL_POINTS)
    glVertex3f (vertex);
glEnd( );
```

Modelview matrix successively contains:

I(identity), M2, M2· M1

The concatenated matrix is:

M=M2·M1

The transformed vertex is:

M2·(M1 · vertex)

In OpenGL, a transformation sequence is applied in reverse order of which it is specified.

OpenGL: Order in Matrix Multiplication

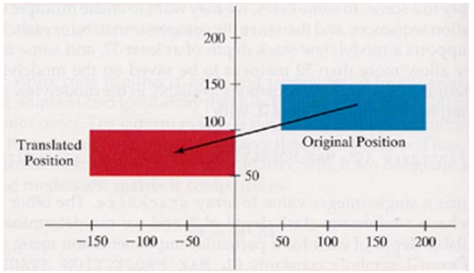
- Example

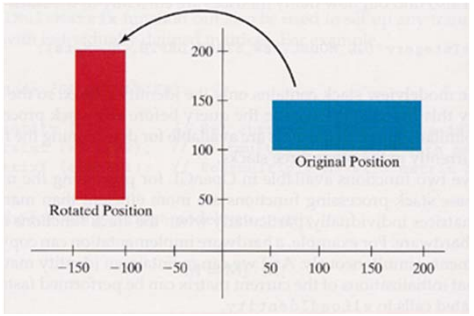
```
// rotate object 30 degrees around Z-axis
glRotatef(30.0, 0.0, 0.0, 1.0);
// move object to (2.0,  3.0, 0.0)
glTranslatef(2.0, 3.0, 0.0);
drawObject();
```

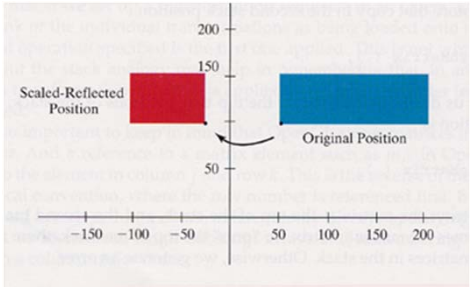
The object will be **translated** first then **rotated**.

OpenGL Geometric Trans. Programming Examples

```
glMatrixMode (GL_MODELVIEW); //Identity matrix

glColor3f (0.0, 0.0, 1.0);          // Set current color to blue
glRecti (50, 100, 200, 150);        // Display blue rectangle.

glColor3f (1.0, 0.0, 0.0);          // Red
glTranslatef (-200.0, -50.0, 0.0);  // Set translation parameters.
glRecti (50, 100, 200, 150);        // Display red, translated rectangle.

glLoadIdentity ();          // Reset current matrix to identity.
glRotatef (90.0, 0.0, 0.0, 1.0);    // Set 90-deg, rotation about z axis.
glRecti (50, 100, 200, 150);        // Display red, rotated rectangle.

glLoadIdentity ();          // Reset current matrix to identity.
glScalef (-0.5, 1.0, 1.0);          // Set scale-reflection parameters.
glRecti (50, 100, 200, 150);        // Display red, transformed rectangle.
```







Summary

- Basic 2D geometric transformations
  - Translation
  - Rotation
  - Scaling
  - Reflection, shearing…
  - Combination of these transformations
- Homogeneous coordinate representation
- OpenGL geometric transformation functions
  - GL_MODELVIEW matrix
  - Order in multiple matrix multiplication
  - Example