Viewing

Creating and Viewing a Scene

- How to view the geometric models that you can now draw with OpenGL

- Two key factors:
  - Define the position and orientation of geometric objects in 3D space (creating the scene)
  - Specify the location and orientation of the viewpoint in the 3D space (viewing the scene)

- Try to visualize the scene in 3D space that lies deep inside your computer

A Series of Operations Needed

- Transformations
  - Modeling, viewing and projection operations
- Clipping
  - Removing objects (or portions of objects) lying outside the window
- Viewport Transformation
  - Establishing a correspondence between the transformed coordinates (geometric data) and screen pixels

The Camera Analogy

- Position and aim the Camera at the scene
  - Viewing transformation: Position the viewing volume in the world
- Arrange the scene to be photograph into the desired composition
  - Modeling transformation: Position the models in the world
- Choose a camera lens or adjust the zoom to adjust field of view
  - Projection transformation: Determine the shape of the viewing volume
- Determine the size of the developed (final) photograph
  - Viewport transformation

Transformation Matrix

- Transformation is represented by matrix multiplication
- Construct a 4x4 matrix ***M*** which is then multiplied by the coordinates of each vertex ***v*** in the scene to transform them to new coordinates ***v***'

$$v' = Mv$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} x \\ y \\ y \\ w \end{bmatrix}$$

Homogenous Coordinates:

$v = (x, y, z, w)^T$

Relation between Cartesian and homogeneous coordinates:

$x_c = x/w, \quad y_c = y/w, \quad z_c = z/w$

Different Matrices

Identity Matrix

$$M_I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation Matrix

$$M_T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation Matrix (about *x*-axis)

$$M_R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling Matrix

$$M_S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Order of Matrix Multiplication

- Each transformation command multiplies a new matrix M by the current matrix C
  - Last command called in the program is the first one applied to the vertices

    glLoadIdentity();

    glMultMatrixf(N);

    glMultMatrixf(M)

    glMultMatrix(L)

    glBegin(GL_POINTS);

    glVertec3f(v);

    glEnd();

    The transformed vertex is INMLv

    Transformations occur in the opposite order than they applied

- Transformations are first defined and then objects are drawn

Coordinate Systems

- Grand, fixed coordinate system
  - Geometric models are transformed in the fixed coordinate system
  - Matrix multiplication occur in the opposite order from how they appear in the code, e.g.,

    glMultMatrixf(T);

    glMultMatrixf(R);

    The order is T(Rv)

- Local coordinate system
  - The system is tied to the object you are drawing
  - All operations occur relative to this moving coordinate system
  - Matrix multiplications appear in the natural order, e.g,

    R(Tv)
  - Useful for applications such as robot arms

General Purpose Transformation Commands

- void **glMatrixMode**(GLenum *mode*);
  - Specifies which matrix will be modified, using GL_MODELVIEW or GL_PROJECTION for *mode*
- Multiplies the current matrix ***C*** by the specified matrix ***M*** and then sets the result to be the current matrix

  Final matrix will be ***CM***
  - Combines previous transformation matrices with the new one
  - But you may not want such combinations in many cases
- void **glLoadIdentity**(*void*);
  - Sets the current matrix to the 4x4 identity matrix
  - Clears the current matrix so that you avoid compound transformation for new matrix

More Commands

- void **glLoadMatrix**(const *TYPE* **m*);
  - Specifies a matrix that is to be loaded as the current matrix
  - Sets the sixteen values of the current matrix to those specified by *m*

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}$$

- void **glMultMatrix**(const *TYPE* **m*);
  - Multiplies the matrix specified *M* by the current matrix and stores the result as the current matrix

Modeling Transformations

- Positioning and orienting the geometric model
  - MTs appear in display function
- Translate, rotate and/or scale the model
  - Combine different transformations to get a single matrix
  - Order of matrix multiplication is important
- Affine transformation $v' = Av + b$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

OpenGL Routines for MTs

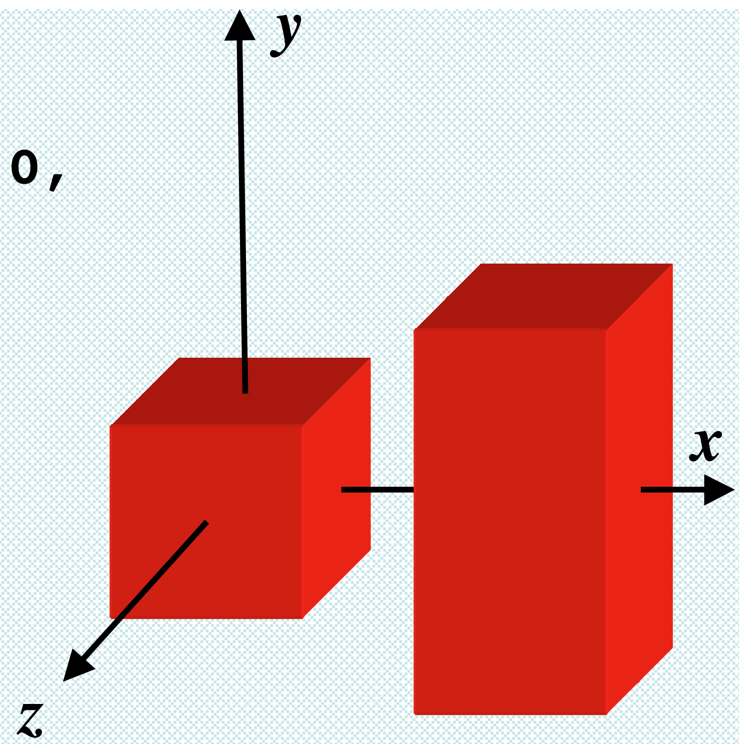
- void **glTranslate**{fd}(*TYPE x*, *TYPE y*, *TYPE z*);
  - Moves (translates) an object by given *x*, *y* and *z* values

- void **glRotate**{fd}(*TYPE angle*, *TYPE x*, *TYPE y*, *TYPE z*);
  - Rotates an object in a counterclockwise direction by *angle* (in degrees) about the rotation axis specified by vector (*x*,*y*,*z*)

- void **glScale**{fd}(*TYPE x*, *TYPE y*, *TYPE z*);
  - Shrinks or stretches or reflects an object by specified factors in x, y and z directions

Transformed Cube

```
void {display}
{
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0,0.0,5.0, 0.0,0.0,0.0,
    0.0,1.0,0.0);
    glutSolidCube(1);
    glTranslatef(3, 0.0, 0.0);
    glScalef(1.0, 2.0, 1.0);
    glutSolidCube(1);
}
```



First cube is centered at (0,0,0)

Second cube is at (3,0,0)
and its y-length is scaled twice

Viewing Transformations

- Specify the position and orientation of viewpoint

- Often called before any modeling transformations so that the later take effect on the objects first
  - Defined in *display or reshape* functions

- Default: Viewpoint is situated at the origin, pointing down the negative z-axis, and has an up-vector along the positive y-axis

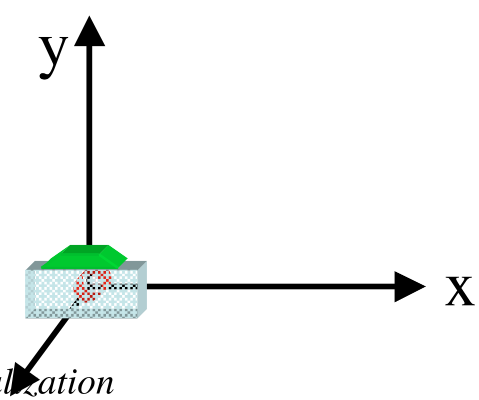
- VTs are generally composed of translations and rotations

- Define a custom utility for VTs in specialized applications

Using GLU Routine for VT

- void **gluLookAt**(GLdouble *eyex*, GLdouble *eyey*, GLdouble *eyez*, GLdouble *centerx*, GLdouble *centery*, GLdouble *centerz*, GLdouble *upx*, GLdouble *upy*, GLdouble *upz*);
  - Defines a viewing matrix and multiplies it by the current matrix
  - *eyex,eyz,eyz* = position of the viewpoint
  - *centerx,centery,centerz* = any point along the desired line of sight
  - *upx,upy,upz* = up direction from the bottom to the top of vewing volume

  gluLookAt(0.0,0.0,5.0, 0.0,0.0,-10.0, 0.0,1.0,0.0);



Using glTranslate and glRotate for VT

- Use modeling transformation commands to emulate viewing transformation

- **glTranslatef**(0.0, 0.0, -5.0)
  - Moves the objects in the scene -5 units along the *z*-axis
  - This is equivalent to moving the viewpoint +5 units along the *z*-axis
- **glRotatef**(45.0, 0.0, 1.0, 0.0);
  - Rotates objects (local coordinates) by 45 degrees about *y*-axis
  - To view objects from the side
  - This is equivalent to rotating camera in opposite sense

- Total effect is equivalent to
  gluLookAt (3.53,0.0,3.53, 0.0,0.0,0.0, 0.0,1.0,0.0);

Modelview Matrix

- Modeling and viewing transformations are complimentary so they are combined to the modelview matrix mode

- To activate the modelview transformation

  glMatrixMode(GL_MODELVIEW);

  glLoadIdentity();

  glTranslate();

  glRotate();

- Default *mode* is set at modelview
  - Needs to be specified only if the other *mode (projection) is* activated and you want to go back to *modelview mode*

Example 1

- Modeling and Viewing Transofrmations

Projection Transformations

- Call **glMatrixMode**(GL_PROJECTION);

  glLoadIdentity();
  - activate the projection matrix
  - PT is defined in *reshape* function

- To define the field of view or viewing volume
  - how an object is projected on the screen
  - which objects or portions of objects are clipped out of the final image

Two Types of Projection

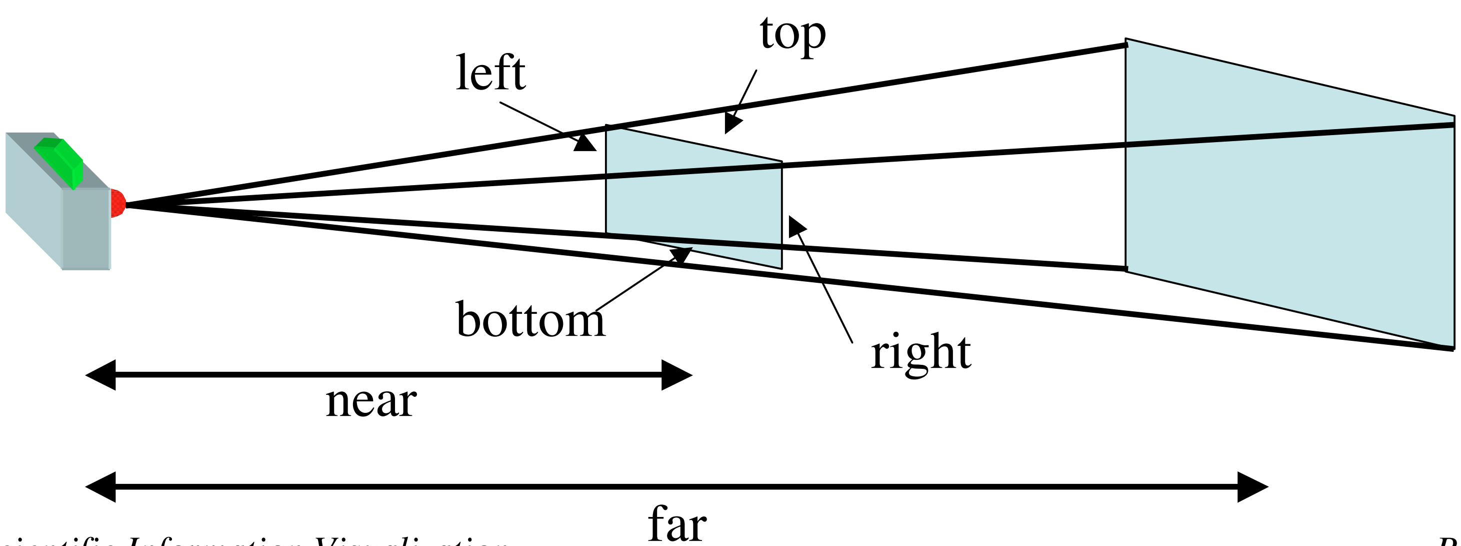
- Perspective projection
  - Foreshortening:

    The farther an object is from the camera, the smaller it appears in the final image
  - Gives a realism: How our eyes work
  - Viewing volume is frustum of a pyramid

- Orthographic projection
  - Size of object is independent of distance
  - Viewing volume is a rectangular parallelepiped (a box)

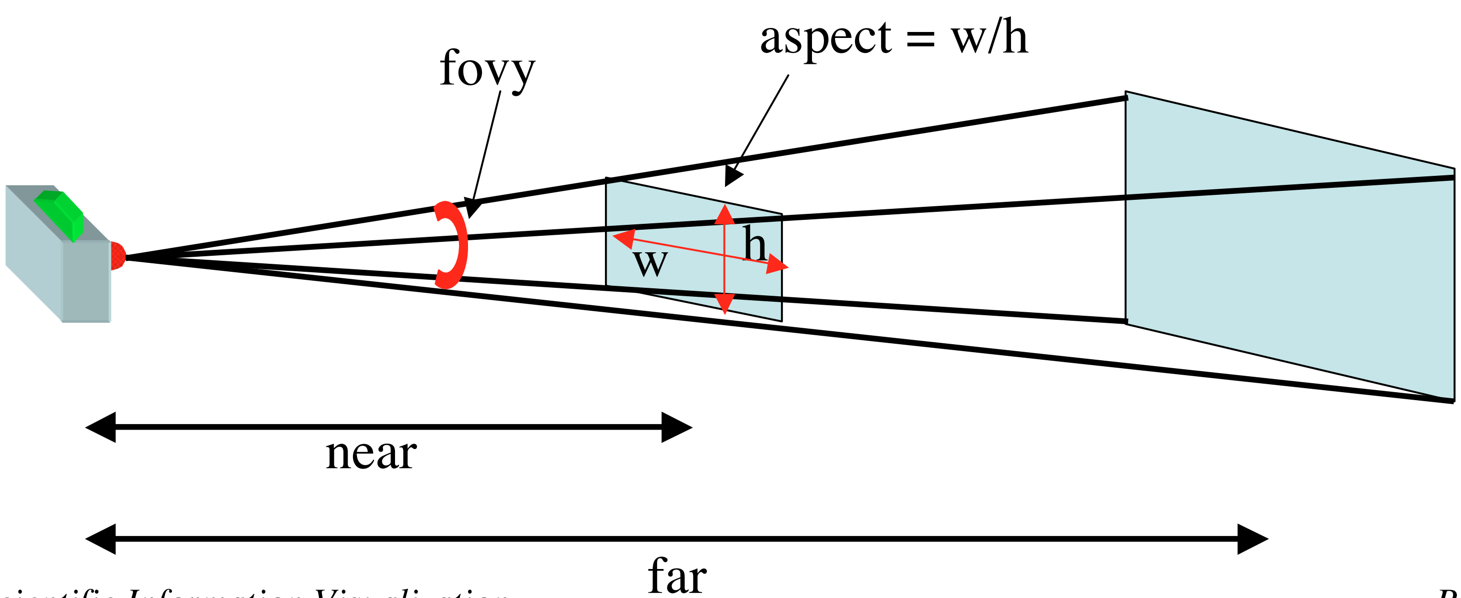
glFrustum

- void **glFrustum**(GLdouble *left*, GLdouble *right*, GLdouble *bottom*, GLdouble *top*, GLdouble *near*, GLdouble *far*);
  - Creates a matrix for perspective-view frustum
  - The frustum's viewing volume is defined by the coordinates of the lower-left and upper-right corners of the near clipping plane



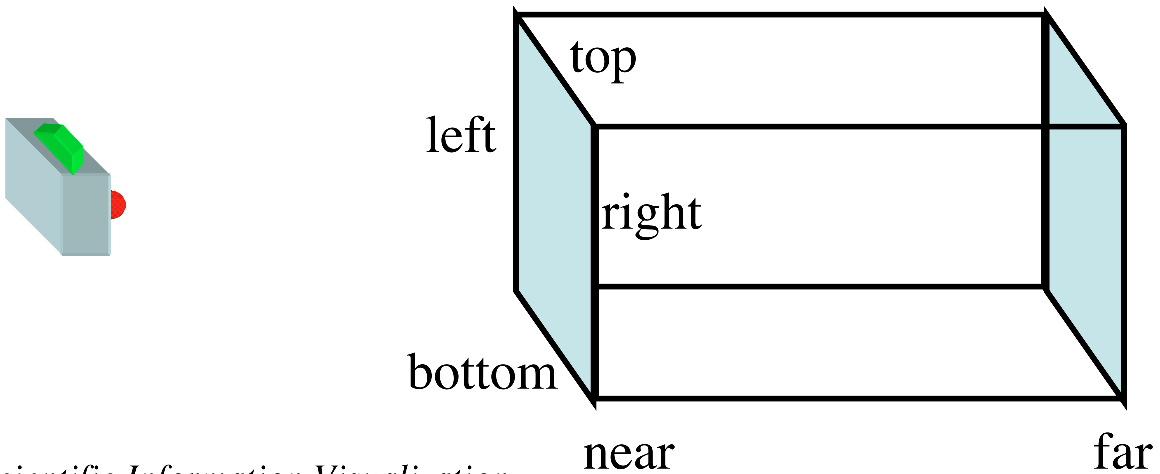
gluPerspective

- void **gluPerspective**(GLdouble *fovy*, GLdouble *aspect*, GLdouble *near*, GLdouble *far*);
  - Creates a matrix for a symmetric perspective-view frustum
  - Frustum is defined by *fovy* (angle in *yz* plane) and *aspect ratio*
  - Near and far clipping planes



Orthographic Projection

- Void **glOrtho**(GLdouble *left*, GLdouble *right*, GLdouble *bottom*, GLdouble *top*, GLdouble *near*, GLdouble *far*);
  - Creates an orthographic parallel viewing volume



Viewing Volume Clipping

- Clipping
  - Frustum defined by six planes (left, right, bottom, top, near, and far
  - Clipping is effective after modelview and projection transformations

- Further restricting the viewing volume by specifying additional clipping planes (up to 6)

- **glClipPlane**(GLenum *plane*, const GLdouble **equation*)
  - Defines a clipping plane.
  - The *equation* argument points to the coefficients of the plane equation $Ac+By+Cz+D=0$
  - Only points that satisfy $(A\ B\ C\ D)M^{-1}(x_e\ y_e\ z_e\ w_e)^T >= 0$ are kept.
  - The *plane* argument is GL_CLIP_PLANEi, where is labels the clipping plane
  - Needs to be enabled and disabled

Example2: Clipping

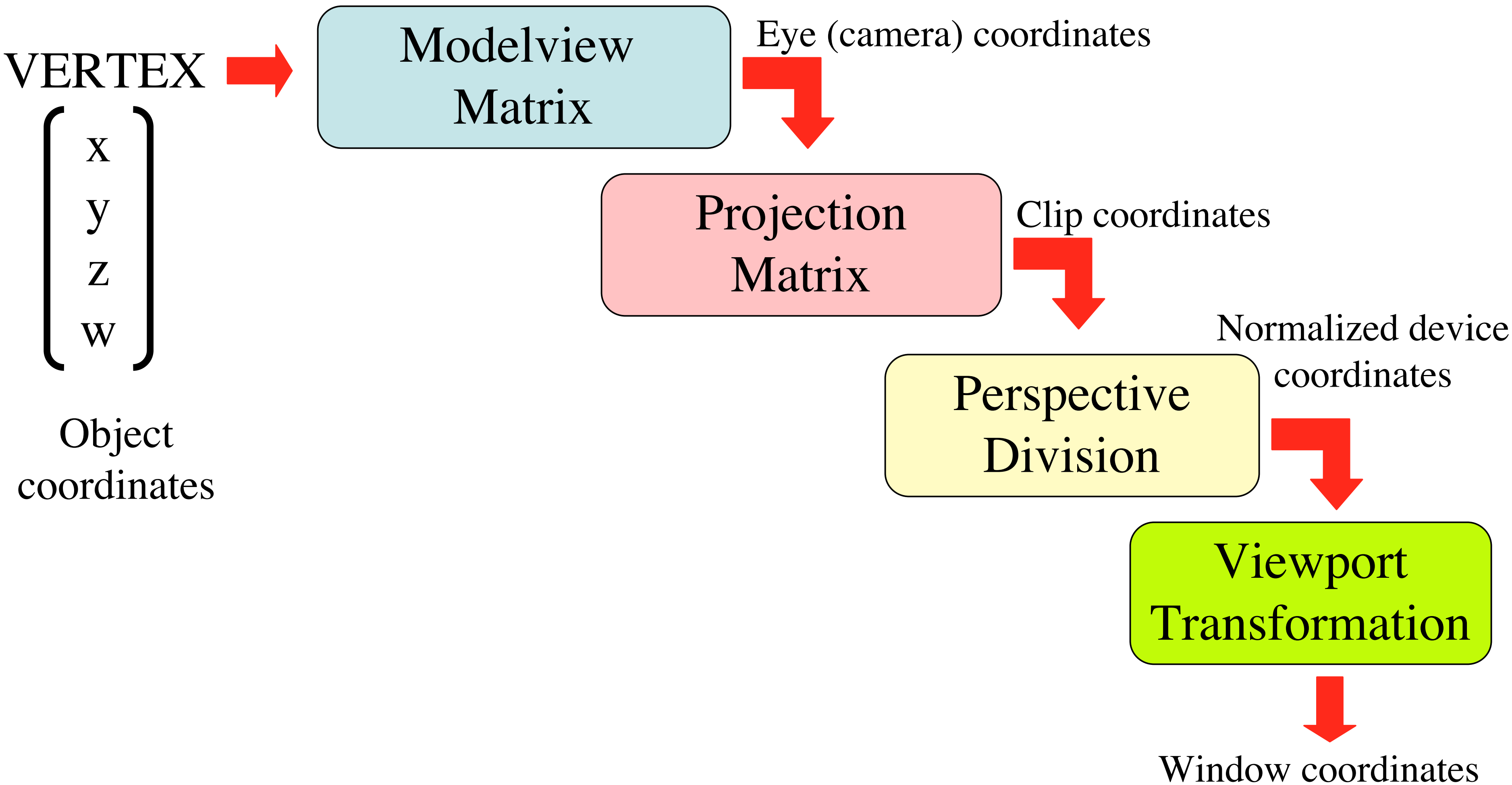
```
void display (void)
{
    GLdouble eqn0[4] = {0.0, 1.0, 0.0, 0.0);
    GLdouble eqn1[4] = {1.0, 0.0, 0.0, 0.0);

    glClearColor (0.0, 0.0, 0.0, 0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 0.0, 0.0);

    glClipPlane (GL_CLIP_PLANE0, eqn0);
    glEnable (GL_CLIP_PLANE0);
    glClipPlane (GL_CLIP_PLANE1, eqn1);
    glEnable (GL_CLIP_PLANE1);

    glutWireSphere(1.0, 20, 16);
    glFlush();
}
```

Viewport Transformation

- Viewport is a rectangular region of window where the image is drawn
  - Measured in window coordinates
  - Reflects the position of pixels on the screen relative to lower-left corner of the window

- void **glViewport**(GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*);
  - Defines a pixel rectangle in the window into which the final image is mapped
  - Aspect ratio of a viewport = aspect ratio of the viewing volume, so that the projected image is undistorted
  - **glViewport** is called in *reshape* function

Vertex Transformation Flow



Matrix Stacks

- OpenGL maintains stacks of transformation matrices
  - At the top of the stack is the current matrix
  - Initially the topmost matrix is the identity matrix
  - Provides an mechanism for successive remembering, translating and throwing

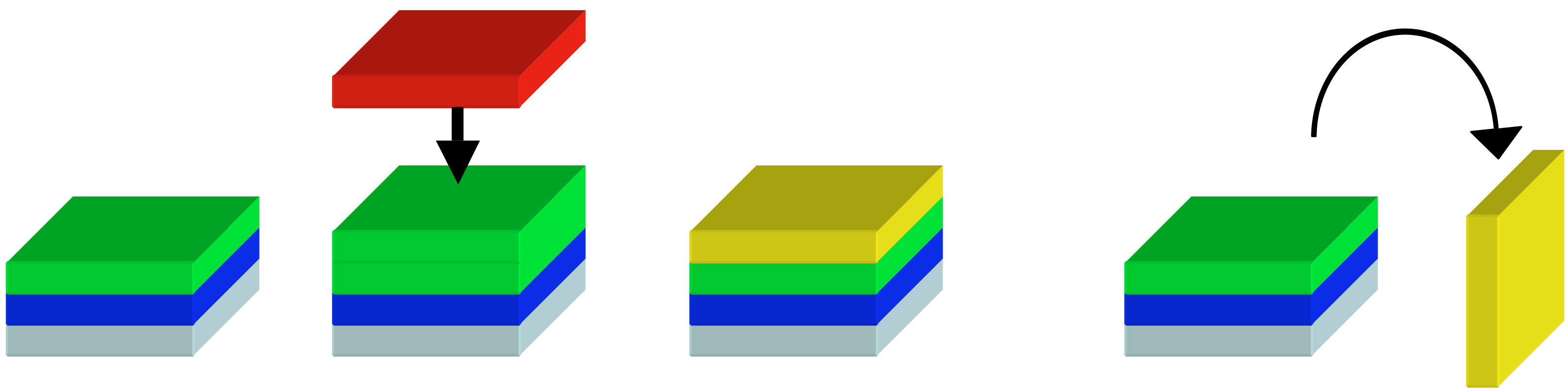
    Get back to a previous coordinate system

- Modelview matrix stack
  - Has 32 matrices or more on the stack
  - Composite transformations

- Projection matrix stack
  - is only two or four levels deep

Pushing and Popping the Matrix Stack

- void **glPushMatrix**(void);
  - Pushes all matrices in the current stack down one level
  - Topmost matrix is copied so its contents are duplicated in both the top and second-from-the-top matrix
  - Remember where you are



- void **glPopMatrix**(void);
  - Eliminates (pops off) the top matrix (destroying the contents of the popped matrix) to expose the second-from-the-top matrix in the stack
  - Go back to where you were

Example 3: Building A Solar System

- How to combine several transformations to achieve a particular result

- Solar system (with a planet and a sun)
  - Setup a viewing and a projection transformation
  - Use **glRotate** to make both grand and local coordinate systems rotate
  - Draw the sun which rotates about the grand axes
  - **glTranslate** to move the local coordinate system to a position where planet will be drawn
  - A second **glRotate** rotates the local coordinate system about the local axes
  - Draw a planet which rotates about its local axes as well as about the grand axes (i.e., orbiting about the sun)

Commands to Draw the Sun and Planet

```
glPushMatrix ();

glRotatef (year, 0.0, 1.0, 0.0);
glutWireSphere (1.0, 20, 16);

glTranslatef (2.0, 0.0, 0.0);
glRotatef (day, 0.0, 1.0, 0.0);

glutWireSphere (0.2, 10, 8);

glPopMatrix ();
```