OpenGL: Transformations

Slides adapted from Angel: Interactive Computer Graphics 4E © Addison-Wesley 2005

1

Objectives

- Learn how to carry out modelling transformations in OpenGL
  - Scaling
  - Rotation
  - Translation
- Introduce OpenGL matrix modes
  - Model-view
  - Projection
- OpenGL 3D Viewing

2

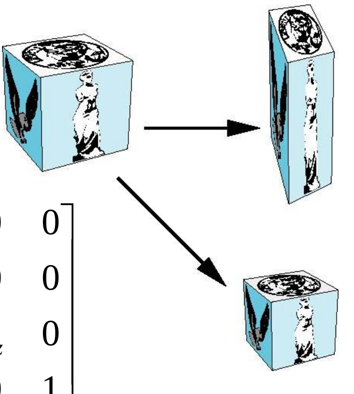
Scaling

Expand or contract along each axis (fixed point of origin)

$x' = s_x x$
$y' = s_y y$
$z' = s_z z$

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

$$\mathbf{S} = \mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



3

Example ..

```
GLfloat CubeVertices[][3] = {{-1.0,-1.0,-1.0},
  {1.0,-1.0,-1.0}, {1.0,1.0,-1.0}, {-1.0,1.0,-1.0},
  {-1.0,-1.0,1.0}, {1.0,-1.0,1.0}, {1.0,1.0,1.0},
  {-1.0,1.0,1.0}};
```

- `colorcube();  // not using scaling factor`
- `glScaled(0.3, 0.3, 0.3);`
  `colorcube();`

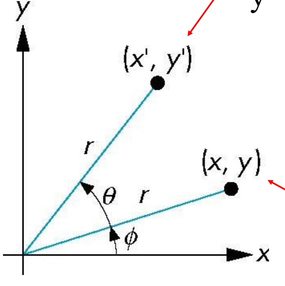
4

Rotation (2D)

Consider rotation about the origin by θ degrees

- radius stays the same, angle increases by θ

$x' = r \cos(\phi + \theta)$
$y' = r \sin(\phi + \theta)$



$$\boxed{\begin{aligned} x' &= x \cos\theta - y \sin\theta \\ y' &= x \sin\theta + y \cos\theta \end{aligned}}$$

$x = r \cos\phi$
$y = r \sin\phi$

Queen Mary University of London

5

Rotation about the z axis

- Rotation about z axis in three dimensions leaves all points with the same z
  - Equivalent to rotation in two dimensions in planes of constant z

$$x' = x \cos\theta - y \sin\theta$$
$$y' = x \sin\theta + y \cos\theta$$
$$z' = z$$

  - or in homogeneous coordinates

$$\mathbf{p}' = \mathbf{R}_z(\theta)\mathbf{p}$$

Queen Mary University of London

6

Rotation Matrix

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

```
glRotated(θ, 0.0, 0.0, 1.0);
```

Queen Mary University of London

7

Rotation about x and y axes

- Same argument as for rotation about *z* axis
  - For rotation about x axis, x is unchanged
  - For rotation about y axis, y is unchanged

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

```
glRotated(θ, 1.0, 0.0, 0.0);
```

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

```
glRotated(θ, 0.0, 1.0, 0.0);
```

Queen Mary University of London

8

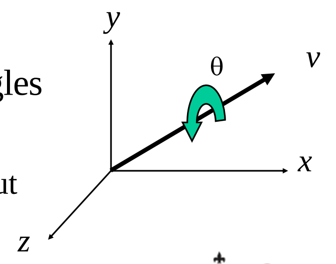
General Rotation About the Origin

A rotation by θ about an arbitrary axis can be decomposed into the concatenation of rotations about the *x*, *y*, and *z* axes

$$\mathbf{R}(\theta) = \mathbf{R}_z(\theta_z)\,\mathbf{R}_y(\theta_y)\,\mathbf{R}_x(\theta_x)$$

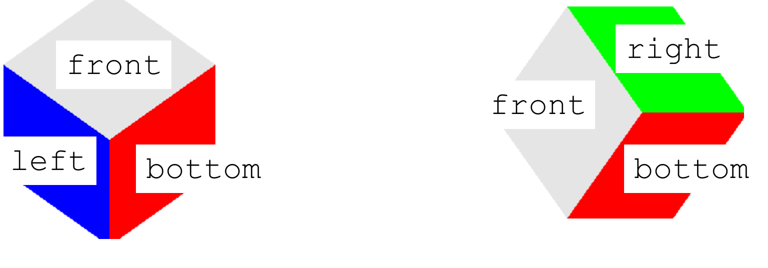
$\theta_x\ \theta_y\ \theta_z$ are called the Euler angles

Note that rotations do not commute.
We can use rotations in another order but with different angles.



Example ..

How would you achieve this ?



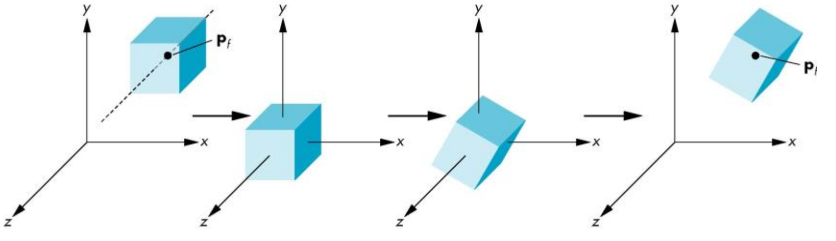
Rotation About a Fixed Point other than the Origin

Move fixed point to origin

Rotate

Move fixed point back

$\mathbf{M} = \mathbf{T}(p_f)\,\mathbf{R}(\theta)\,\mathbf{T}(-p_f)$

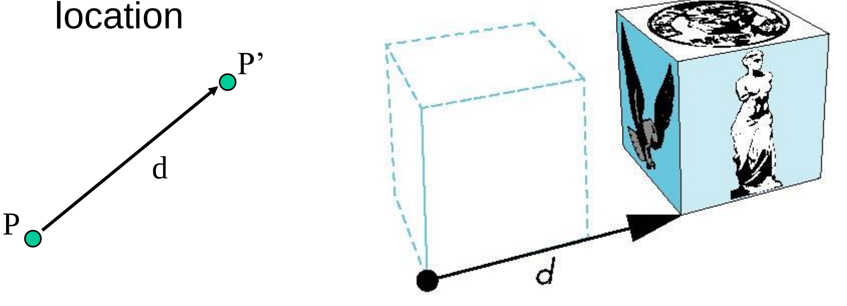


Queen Mary University of London

11

Translation

- Move (translate, displace) a point to a new location



- Displacement determined by a vector d
  - Three degrees of freedom
  - P'=P+d

Queen Mary University of London

12

Translation Matrix

We can also express translation using a
4 x 4 matrix **T** in homogeneous coordinates
p'=Tp where

$$\mathbf{T} = \mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
glTranslated(dx, dy, dz);
```

This form is better for implementation because all affine transformations can be expressed this way and multiple transformations can be concatenated together

Queen Mary
University of London

13

Example ..

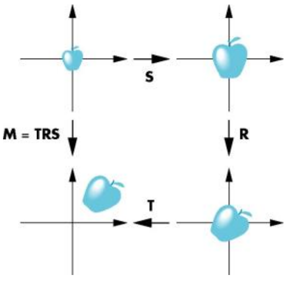
```
glTranslated(-0.5, -0.5, 0.0);
glRotated(45.0, 1.0, 0.0, 0.0);
glRotated(45.0, 0.0, 0.0, 1.0);
glScaled(0.3, 0.3, 0.3);
colorcube();

glTranslated(0.5, 0.5, 0.0);
glRotated(45.0, 0.0, 1.0, 0.0);
glRotated(45.0, 0.0, 0.0, 1.0);
glScaled(0.3, 0.3, 0.3);
colorcube();
```

Queen Mary
University of London

14

Instancing

- In modelling, we often start with a simple object centered at the origin, oriented with the axis, and at a standard size (a "model")
- We apply an *instance transformation* to its vertices to
  - Scale
  - Orient
  - Locate



Queen Mary University of London

15

Objectives

- Learn how to carry out modelling transformations in OpenGL
  - Scaling
  - Rotation
  - Translation
- Introduce OpenGL matrix modes
  - Model-view
  - Projection
- OpenGL 3D Viewing

Queen Mary University of London

16

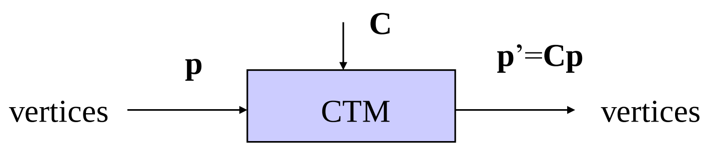
OpenGL Matrices

- In OpenGL matrices are part of the state
- Multiple types
  - Model-View (**`GL_MODELVIEW`**)
  - Projection (**`GL_PROJECTION`**)
  - Texture (**`GL_TEXTURE`**) (ignore for now)
  - Color(**`GL_COLOR`**) (ignore for now)
- Single set of functions for manipulation
- Select which to manipulate with:
  - **`glMatrixMode(GL_MODELVIEW);`**
  - **`glMatrixMode(GL_PROJECTION);`**

17

Current Transformation Matrix (CTM)

- Conceptually there is a 4 x 4 homogeneous coordinate matrix, the *current transformation matrix* (CTM) that is part of the state and is applied to all vertices that pass down the pipeline (recall that they are created by glVertex)



18

CTM operations

- The CTM can be altered either by loading a new CTM or by postmutiplication

Load an identity matrix: $\mathbf{C} \leftarrow \mathbf{I}$ `glLoadIdentity`
Load an arbitrary matrix: $\mathbf{C} \leftarrow \mathbf{M}$ **glLoadMatrixf**

Load a translation matrix: $\mathbf{C} \leftarrow \mathbf{T}$
Load a rotation matrix: $\mathbf{C} \leftarrow \mathbf{R}$
Load a scaling matrix: $\mathbf{C} \leftarrow \mathbf{S}$

Postmultiply by an arbitrary matrix: $\mathbf{C} \leftarrow \mathbf{CM}$ **glMultMatrixf**
Postmultiply by a translation matrix: $\mathbf{C} \leftarrow \mathbf{CT}$ `glTranslatef`
Postmultiply by a rotation matrix: $\mathbf{C} \leftarrow \mathbf{C\,R}$ `glRotatef`
Postmultiply by a scaling matrix: $\mathbf{C} \leftarrow \mathbf{C\,S}$ `glScalef`

Queen Mary University of London

19

e.g. Rotation about a Fixed Point

Start with identity matrix: $\mathbf{C} \leftarrow \mathbf{I}$
Move fixed point to origin: $\mathbf{C} \leftarrow \mathbf{CT}$
Rotate: $\mathbf{C} \leftarrow \mathbf{CR}$
Move fixed point back: $\mathbf{C} \leftarrow \mathbf{CT}^{-1}$

Result: $\mathbf{C} = \mathbf{TR\,T}^{-1}$ which is **backwards**. **WRONG!**

This result is a consequence of doing postmultiplications.

Queen Mary University of London

20

Reversing the Order

We want $\mathbf{C} = \mathbf{T}^{-1}\,\mathbf{R}\,\mathbf{T}$
so we must do the operations in the following order

$\mathbf{C} \leftarrow \mathbf{I}$
$\mathbf{C} \leftarrow \mathbf{C}\mathbf{T}^{-1}$
$\mathbf{C} \leftarrow \mathbf{C}\mathbf{R}$
$\mathbf{C} \leftarrow \mathbf{C}\mathbf{T}$

Each operation corresponds to one function call in the program.

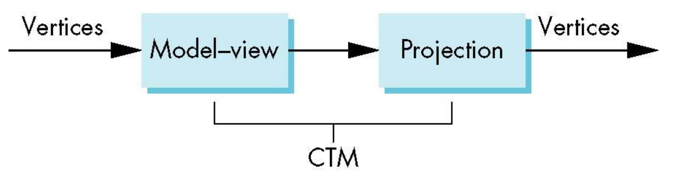
Note that the last operation specified is the first executed in the program

Example (revisited)

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslated(-0.5, -0.5, 0.0);
glRotated(45.0, 1.0, 0.0, 0.0);
glRotated(45.0, 0.0, 0.0, 1.0);
glScaled(0.3, 0.3, 0.3);
colorcube();
glLoadIdentity();
glTranslated(0.5, 0.5, 0.0);
glRotated(45.0, 0.0, 1.0, 0.0);
glRotated(45.0, 0.0, 0.0, 1.0);
glScaled(0.3, 0.3, 0.3);
colorcube();
```

CTM in OpenGL

- OpenGL has a model-view and a projection matrix in the pipeline which are concatenated together to form the CTM
- Can manipulate each by first setting the correct matrix mode



Rotation, Translation, Scaling

Load an identity matrix:

- `glLoadIdentity()`

Multiply on right (postmultiplication):

- `glRotatef(theta, vx, vy, vz)`

`theta` in degrees, (**`vx, vy, vz`**) define axis of rotation

- `glTranslatef(dx, dy, dz)`
- `glScalef(sx, sy, sz)`

Each has a float (f) and double (d) format (e.g. **`glScaled`**)

Example (1)

- Rotation about z axis by 30 degrees with a fixed point of (1.0, 2.0, 3.0)

- Remember that the last matrix specified in the program is the first applied

Example (2)

- Instantiate by scaling, rotating and translating.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(-0.3, 0.2, 0.0);
glRotatef(30.0, 0.0, 0.0, 1.0);
glRotatef(30.0, 1.0, 0.0, 0.0);
glRotatef(30.0, 0.0, 1.0, 0.0);
glScalef(0.5, 0.2, 0.3);
colorcube()
```

- Remember that the last matrix specified in the program is the first applied

Arbitrary Matrices

- Can load and multiply by matrices defined in the application program

```
glLoadMatrixf(m)
glMultMatrixf(m)
```

- The matrix **m** is a one dimension array of 16 elements which are the components of the desired 4 x 4 matrix stored by <u>columns</u>
- In **glMultMatrixf**, **m** multiplies the existing matrix on the right

Reading Back Matrices

- Can also access matrices (and other parts of the state) by *query* functions

```
glGetIntegerv
glGetFloatv
glGetBooleanv
glGetDoublev
glIsEnabled
```

- For matrices, we use:

```
GLfloat m[16];
glGetFloatv(GL_MODELVIEW, m);
```

Using the OpenGL matrices

- In OpenGL the model-view matrix is used to
  - Position the camera
    - Can be done by rotations and translations but it is often easier to use **`gluLookAt`**
  - Build models of objects
- The projection matrix is used to define the view volume and to select a camera lens

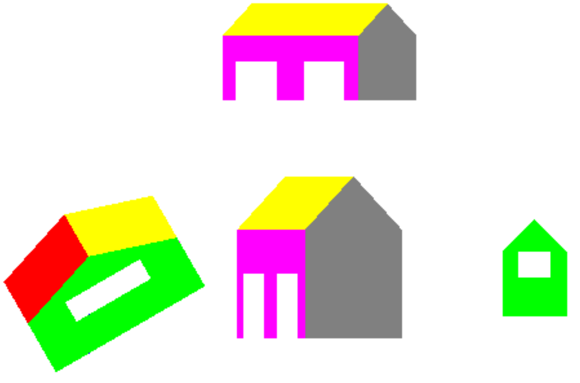
Matrix Stacks

- In many situations we want to save transformation matrices for later use
  - E.g when traversing hierarchical data structures
- OpenGL maintains stacks for each type of matrix
  - Access present type (as set by **`glMatrixMode)`** by

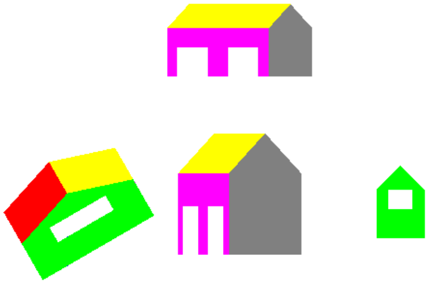
```
glPushMatrix()
glPopMatrix()
```

Example

Use instance transformation to create the following scene:




```
void display()
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glRotatef(theta[0], 1.0, 0.0, 0.0);
  glRotatef(theta[1], 0.0, 1.0, 0.0);
  glRotatef(theta[2], 0.0, 0.0, 1.0);
  glPushMatrix();
  glScalef (0.5, 0.5, 0.5);
  barn();
  glPopMatrix();
  glTranslatef (0.0, 1.0, 0.0);
  glScalef (0.3, 0.3, 1.0);
  barn();
  glLoadIdentity();
  glTranslatef (1.0, 0.0, 0.0);
  glScalef (0.3, 0.3, 0.3);
  barn();
  glLoadIdentity();
  glTranslatef (-1.0, 0.0, 0.0);
  glRotatef(30.0, 0.0, 0.0, 0.1);
  glRotatef(30.0, 1.0, 0.0, 0.0);
  glScalef (0.8, 0.3, 0.3);
  barn();
  glutSwapBuffers();
}
```



Objectives

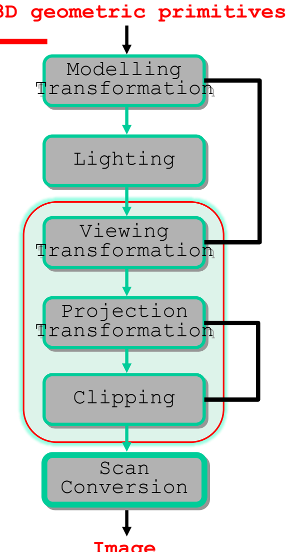
- Learn how to carry out modelling transformations in OpenGL
  - Scaling
  - Rotation
  - Translation
- Introduce OpenGL matrix modes
  - Model-view
  - Projection
- OpenGL 3D Viewing

Queen Mary University of London

33

Computer Viewing

- There are three aspects of the viewing process, all of which are implemented in the pipeline:
  - Positioning the camera
    - Setting the model-view matrix
  - Selecting a lens
    - Setting the projection matrix
  - Clipping
    - Setting the view volume



Queen Mary University of London

34

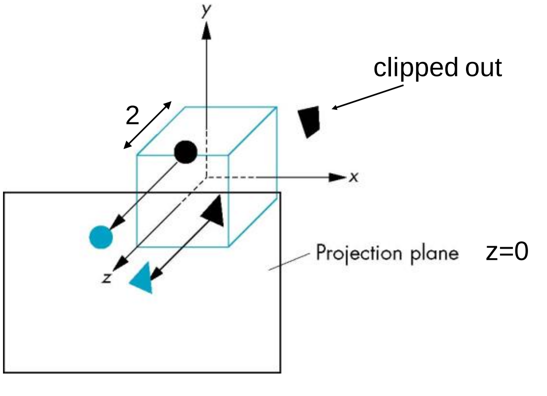
The OpenGL Camera

- In OpenGL, initially the object and camera frames are the same
  - Default model-view matrix is an identity
- The camera is located at origin and points in the negative z direction
- OpenGL also specifies a default view volume that is a cube with sides of length 2 centered at the origin
  - Default projection matrix is an identity

35

Default Projection

Default projection is orthogonal



36

Moving the Camera Frame

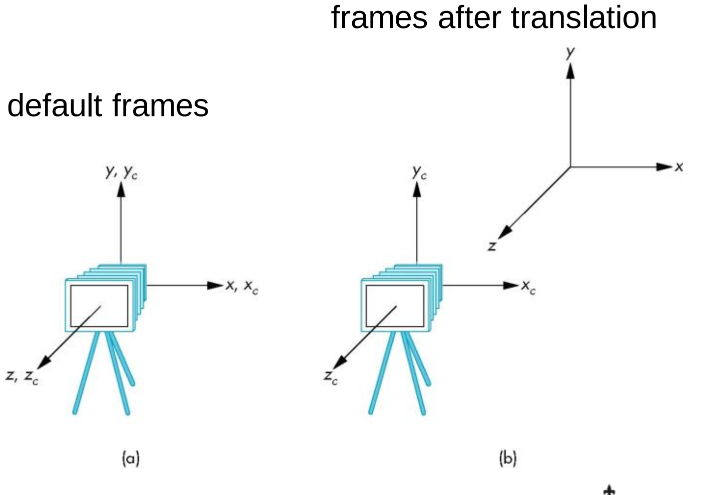
- We can either:
  - Move the camera in the positive z direction
    - Translate the camera frame
  - Move the objects in the negative z direction
    - Translate the world frame
- Both of these views are equivalent and are determined by the model-view matrix

```
glTranslatef(0.0, 0.0, d);
```

Queen Mary University of London

37

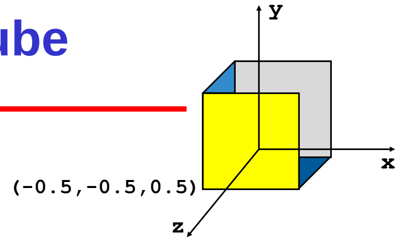
Moving Camera away from Origin



Queen Mary University of London

38

Colorcube



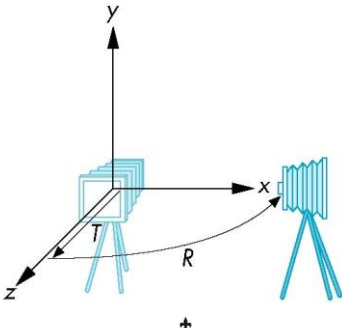

```
1.  glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, 1.0);
    colorcube();
2.  glTranslatef(0.0, 0.0, -1.0);
```

Queen Mary University of London

39

Moving the Camera

- We can move the camera (in fact the objects) to any desired position by a sequence of rotations and translations
- Example: side view
  - Rotate
  - Move away from origin
  - Model-view matrix C = TR

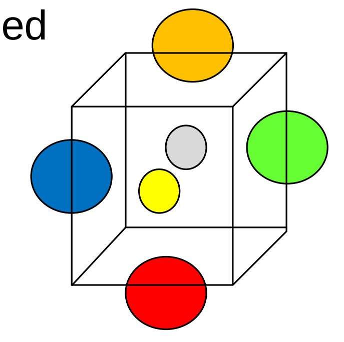


Queen Mary University of London

40

Example OpenGL code

- Remember that last transformation specified is first to be applied



```
glMatrixMode(GL_MODELVIEW)

• glLoadIdentity();
  glRotatef(90.0, 0.0, 1.0, 0.0);

• glLoadIdentity();
  glTranslatef(0.0, 0.0, -1.0);
  glRotatef(90.0, 0.0, 1.0, 0.0);
```

Queen Mary University of London

41

The LookAt Function

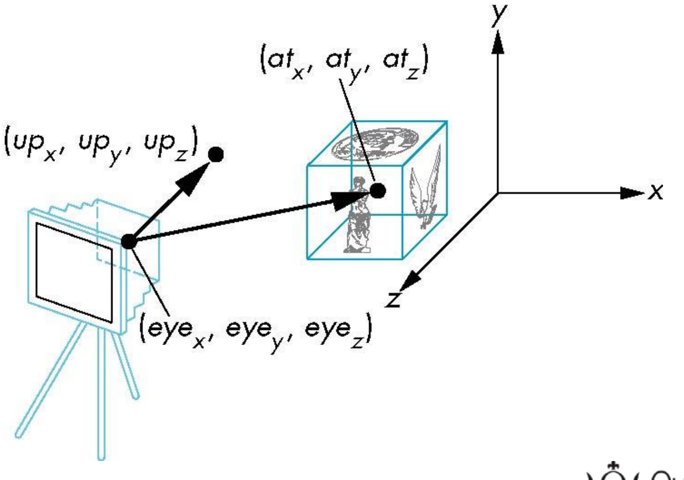
- The GLU library contains the function gluLookAt to form the required modelview matrix through a simple interface
- Note the need for setting an up direction
- Still need to initialize
  - Can concatenate with modeling transformations

Queen Mary University of London

42

gluLookAt

```
gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz)
```



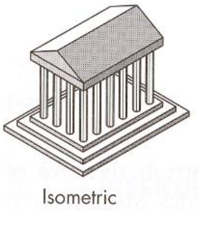
Queen Mary University of London

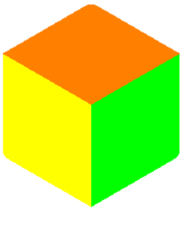
43

gluLookAt: example

- Example: isometric view of cube aligned with axes

```
glMatrixMode(GL_MODELVIEW):
glLoadIdentity();
gluLookAt(1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
```





Queen Mary University of London

44

Other Viewing APIs

- The LookAt function is only one possible API for positioning the camera
- Others include
  - View reference point, view plane normal, view up (PHIGS, GKS-3D)
  - Yaw, pitch, roll
  - Elevation, azimuth, twist
  - Direction angles

45