CS559: Computer Graphics

Lecture 12: OpenGL - Transformation

Li Zhang

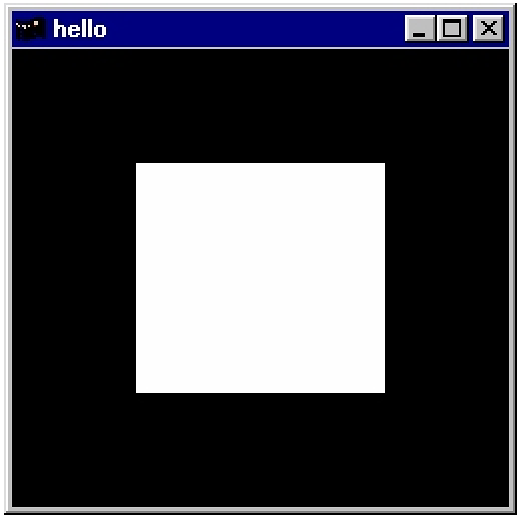
Spring 2008

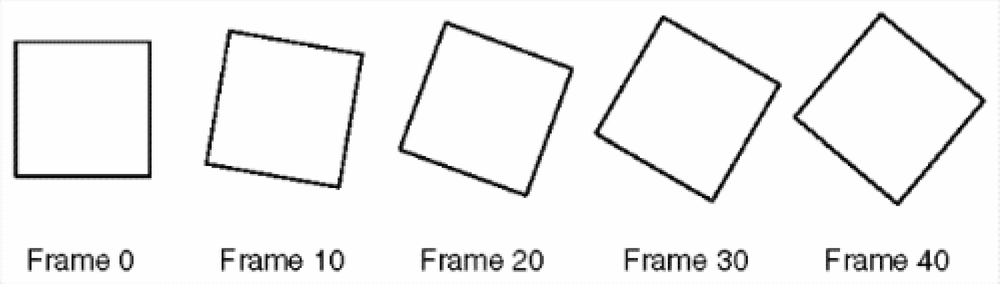
Today

- Transformation in OpenGL

- Reading
  - Chapter 3

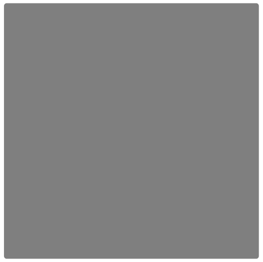
Last time



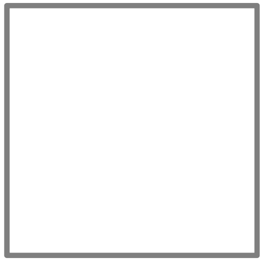


Primitive Details

- glPolygonMode(*GLenum face, GLenum mode);*
  - face: GL_FRONT, GL_BACK
  - mode: GL_POINT, GL_LINE, GL_FILL



```
glPolygonMode(GL_FRONT, GL_FILL);
glRectf(0, 0, 100, 100);
```



```
glPolygonMode(GL_BACK, GL_LINE);
glRectf(0, 0, 100, 100);
```

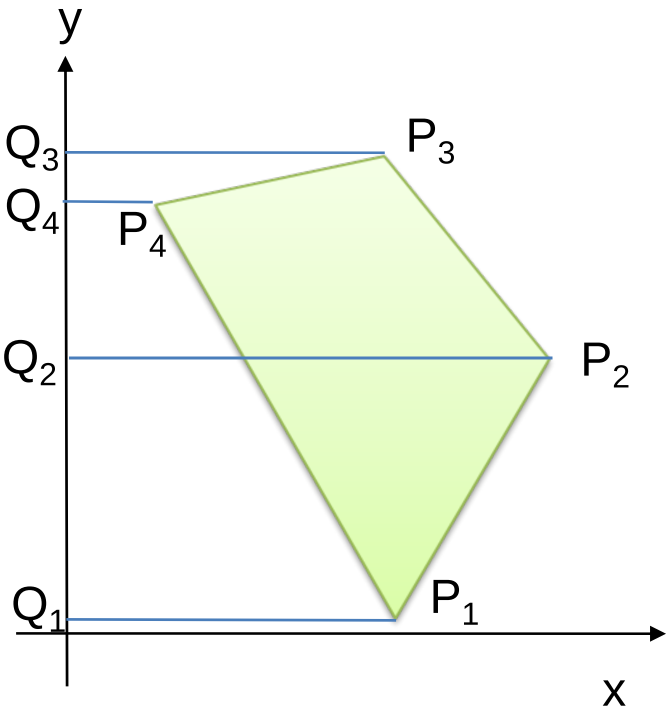
Primitive Details

- Determine Polygon Orientation

$$area(P_1P_2P_3P_4)$$

$$= area(P_1P_2Q_2Q_1) + area(P_2P_3Q_3Q_2)$$

$$- area(P_3P_4Q_4Q_3) - area(P_4P_1Q_4Q_1)$$



$$area = \sum_{n=1}^{N} \frac{1}{2}(x_i + x_{i+1})(y_{i+1} - y_i)$$

$$= \frac{1}{2}\left(\sum_{n=1}^{N} x_i y_{i+1} - \sum_{n=1}^{N} x_i y_i + \sum_{n=1}^{N} x_{i+1} y_{i+1} - \sum_{n=1}^{N} x_{i+1} y_i \right)$$

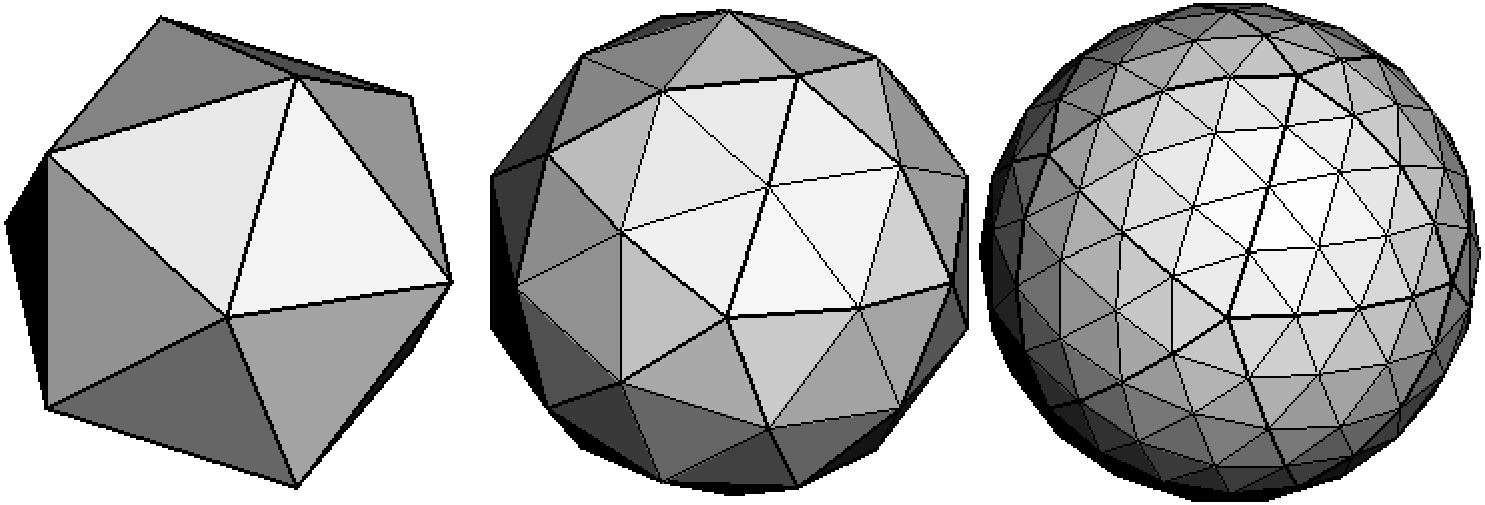
$$\boxed{= \frac{1}{2}\sum_{n=1}^{N}\left(x_i y_{i+1} - x_{i+1} y_i\right)}$$

Orientation == sign of the area

Stokes' Theorem

$$\iint_R dxdy = \oint_{\partial R} xdy$$

Icosahedron



Icosahedron

```
//initial icosahedron
Static float t[20][3][3] = {…};

void display(void)
{









}
```

Icosahedron

```
//initial icosahedron
Static float t[20][3][3] = {…};

void display(void)
{
  //clear buffer
  //set up viewport and frustum

  if (animation) angle+=0.3;
  if (angle>360) angle-=360.0;

  glLoadIdentity();
  glRotatef(angle,1,0,1);









}
```

Icosahedron

```
//initial icosahedron
Static float t[20][3][3] = {…};

void display(void)
{
  //clear buffer
  //set up viewport and frustum

  if (animation) angle+=0.3;
  if (angle>360) angle-=360.0;

  glLoadIdentity();
  glRotatef(angle,1,0,1);

  // subdivide each face of the triangle
  for (int i = 0; i < 20; i++)
  {
    Subdivide(t[i][0], t[i][1], t[i][2], subdiv);
  }







}
```

Icosahedron

```
//initial icosahedron
Static float t[20][3][3] = {…};

void display(void)
{
  //clear buffer
  //set up viewport and frustum

  if (animation) angle+=0.3;
  if (angle>360) angle-=360.0;

  glLoadIdentity ();
  glRotatef(angle,1,0,1);

  // subdivide each face of the triangle
  for (int i = 0; i < 20; i++)
  {
    Subdivide(t[i][0], t[i][1], t[i][2], subdiv);
  }



  glFlush();
  glutSwapBuffers();
}
```

Icosahedron

```
void Subdivide(GLfloat v1[3], GLfloat v2[3], GLfloat v3[3], int depth)
{

}
```

Icosahedron

```
void Subdivide(GLfloat v1[3], GLfloat v2[3], GLfloat v3[3], int depth)
{
  if (depth == 0) {
     glColor3f(0.5,0.5,0.5);
     glBegin(GL_TRIANGLES);
     glVertex3fv(v1);  glVertex3fv(v2);  glVertex3fv(v3);
     glEnd();
  }
















}
```

Icosahedron

```
void Subdivide(GLfloat v1[3], GLfloat v2[3], GLfloat v3[3], int depth)
{
  if (depth == 0) {
     glColor3f(0.5,0.5,0.5);
     glBegin(GL_TRIANGLES);
     glVertex3fv(v1);  glVertex3fv(v2);  glVertex3fv(v3);
     glEnd();
  }
  else
  {
    GLfloat v12[3], v23[3], v31[3];
    for (int i = 0; i < 3; i++) {
       v12[i] = (v1[i]+v2[i])/2.0;
       v23[i] = (v2[i]+v3[i])/2.0;
       v31[i] = (v3[i]+v1[i])/2.0;
     }
     Normalize(v12); Normalize(v23); Normalize(v31);



  }
}
```

Icosahedron

```
void Subdivide(GLfloat v1[3], GLfloat v2[3], GLfloat v3[3], int depth)
{
  if (depth == 0) {
     glColor3f(0.5,0.5,0.5);
     glBegin(GL_TRIANGLES);
     glVertex3fv(v1);  glVertex3fv(v2);  glVertex3fv(v3);
     glEnd();
  }
  else
  {
    GLfloat v12[3], v23[3], v31[3];
    for (int i = 0; i < 3; i++) {
       v12[i] = (v1[i]+v2[i])/2.0;
       v23[i] = (v2[i]+v3[i])/2.0;
       v31[i] = (v3[i]+v1[i])/2.0;
     }
     Normalize(v12); Normalize(v23); Normalize(v31);
     Subdivide(v1, v12, v31, depth-1);
     Subdivide(v2, v23, v12, depth-1);
     Subdivide(v3, v31, v23, depth-1);
     Subdivide(v12, v23, v31, depth-1);
  }
}
```

So far

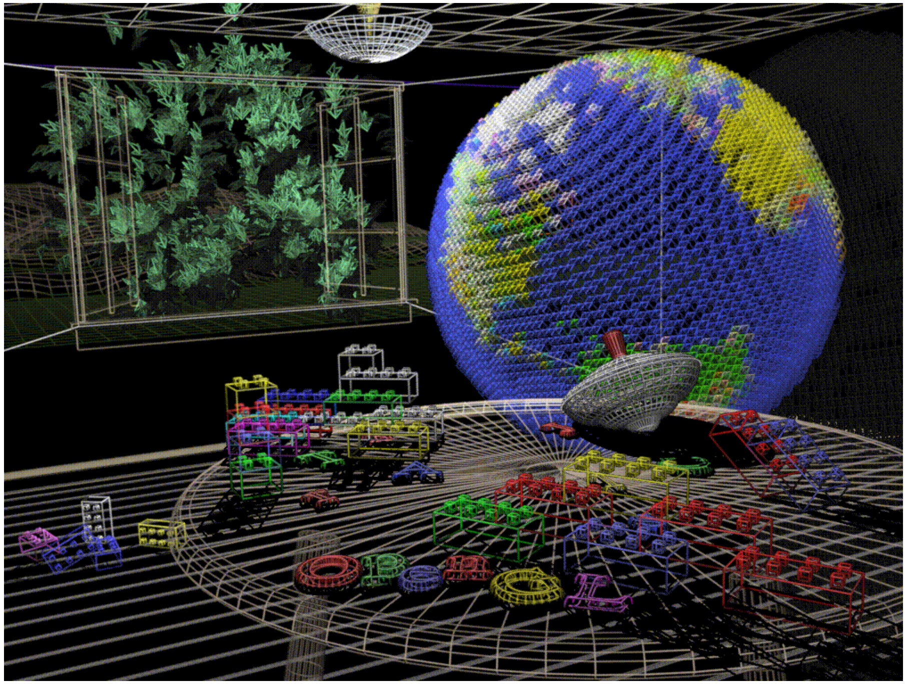
- Fixed Camera location

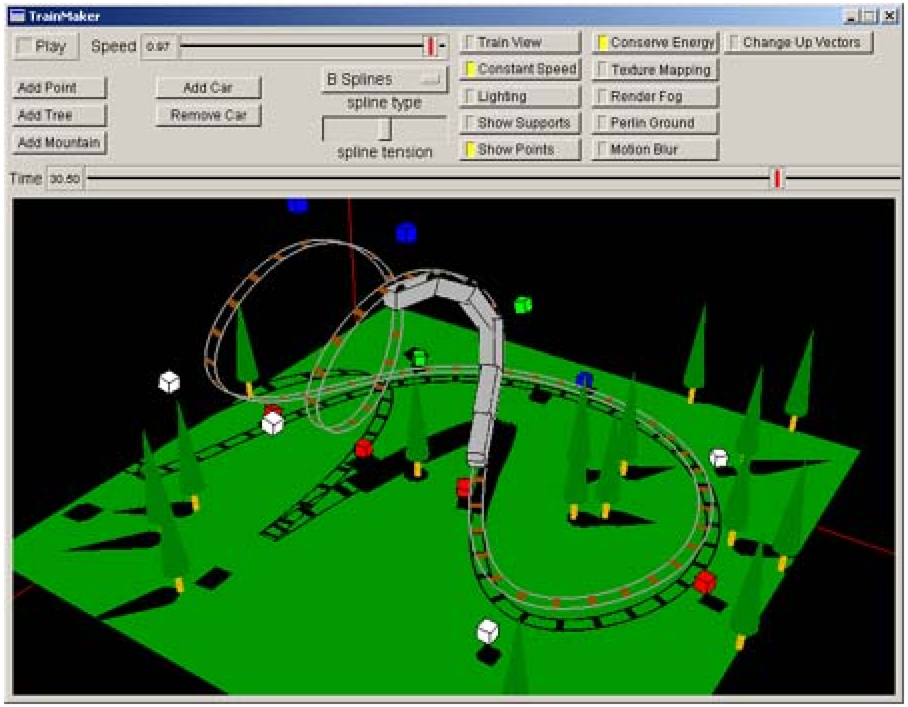
  => Change Camera location

- Single object

  => Multiple objects

Examples of multiple objects





☐ Play

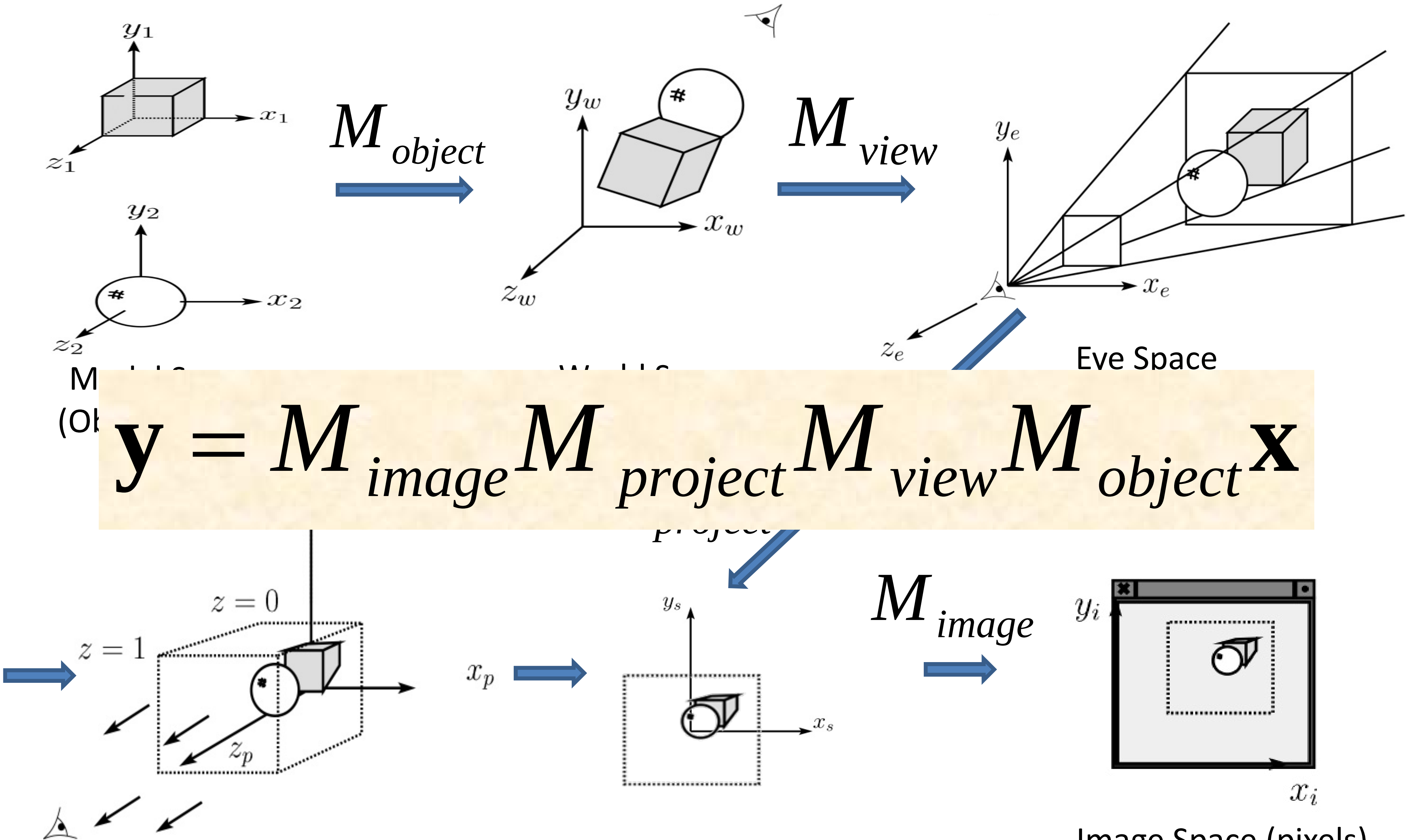
Speed 0.87

B Splines
spline type

☐ Train View
☑ Constant Speed
☐ Lighting
☐ Show Supports
☑ Show Points
☑ Conserve Energy
☐ Texture Mapping
☐ Render Fog
☐ Perlin Ground
☐ Motion Blur
☐ Change Up Vectors

Time 30.50

3D Geometry Pipeline



M_{object}

M_{view}

Eye Space

$$\mathbf{y} = M_{image} M_{project} M_{view} M_{object} \mathbf{x}$$

$M_{project}$

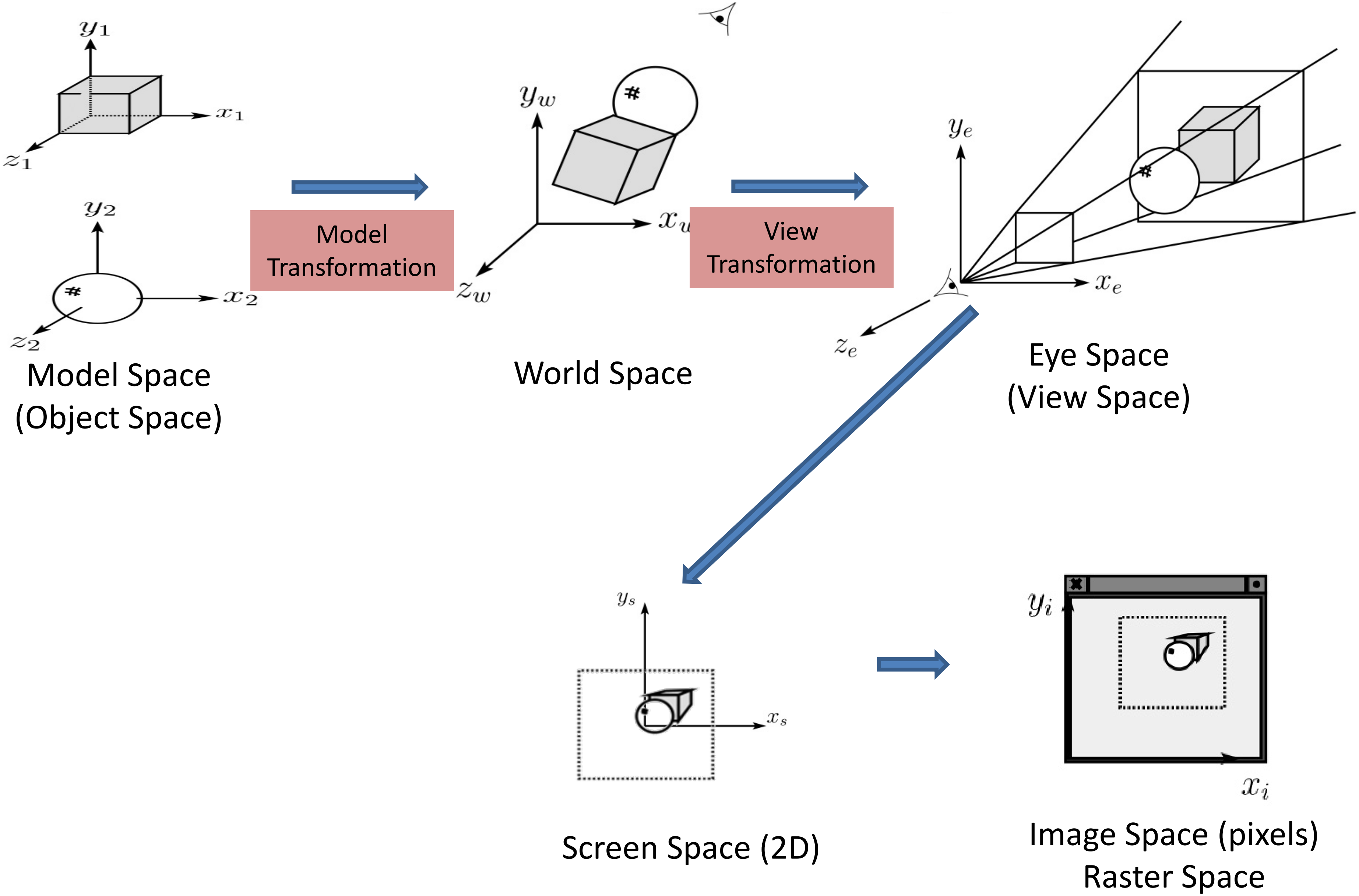
M_{image}

Canonical View Space

Screen Space (2D)

Image Space (pixels)
Raster Space

3D Geometry Pipeline



Model Transformation

View Transformation

Model Space
(Object Space)

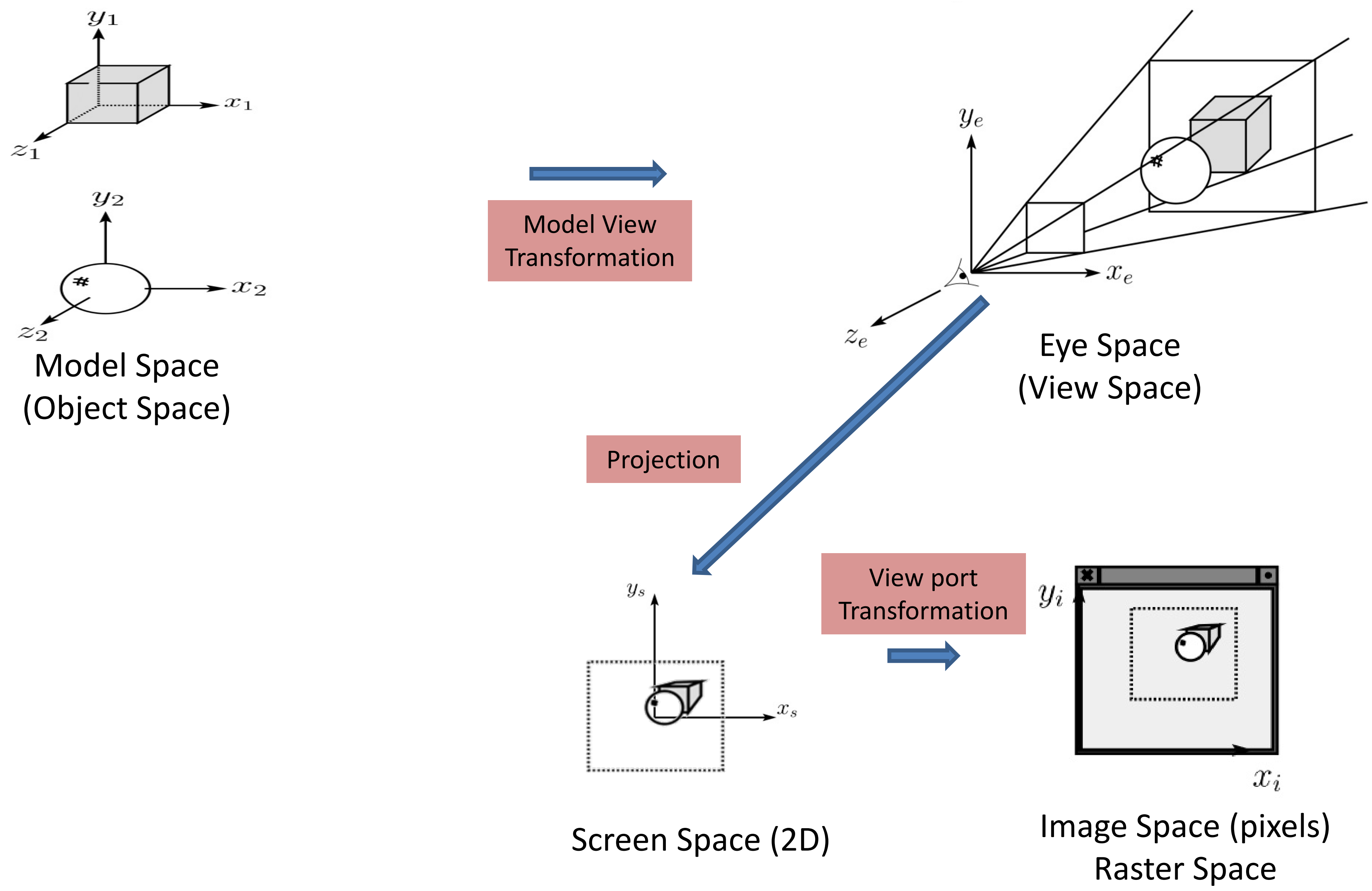
World Space

Eye Space
(View Space)

Screen Space (2D)

Image Space (pixels)
Raster Space

3D Geometry Pipeline



Model Space
(Object Space)

Model View Transformation

Eye Space
(View Space)

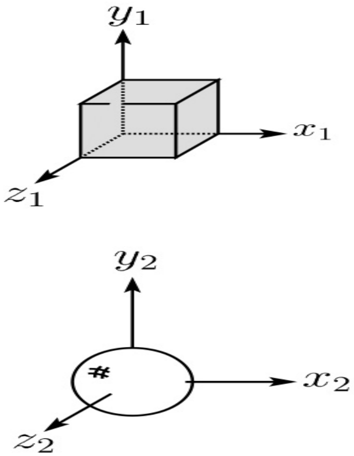
Projection

Screen Space (2D)

View port Transformation

Image Space (pixels)
Raster Space

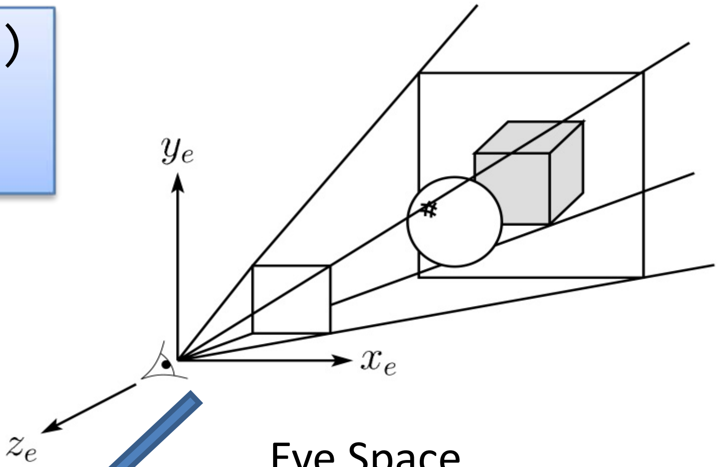
3D Geometry Pipeline



Model Space
(Object Space)

```
glMatrixMode(GL_MODELVIEW)
glLoadIdentity();
glRotate(…);
```

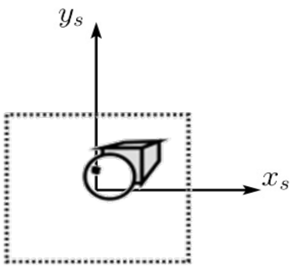
Model View Transformation



Eye Space
(View Space)

Projection

```
glMatrixMode(GL_Projection)
glLoadIdentity();
glFrustum(…);
```



Screen Space (2D)

```
glViewport(…);
```

View port Transformation

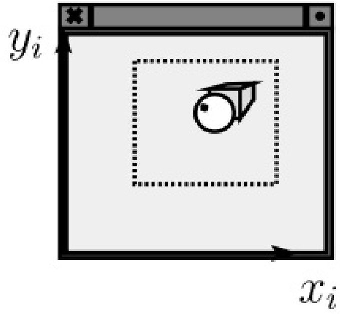
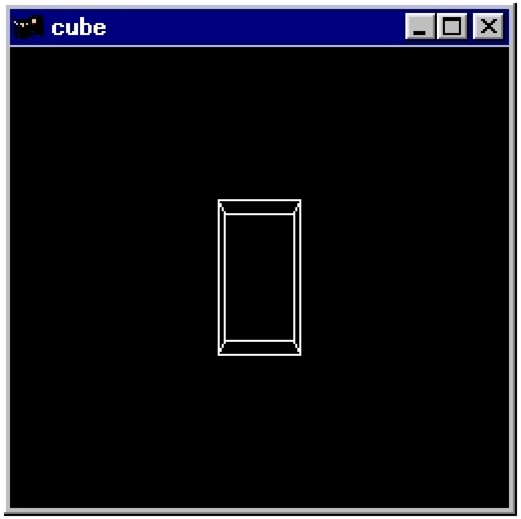


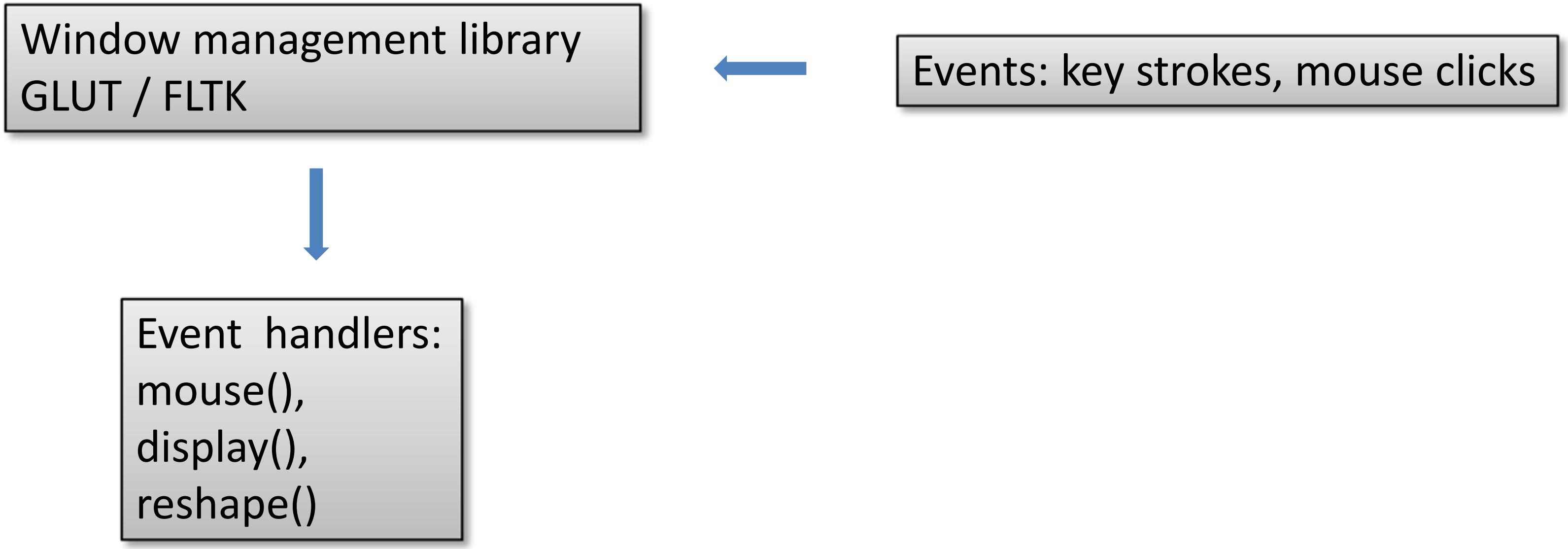
Image Space (pixels)
Raster Space

Cube.cpp



Cube.cpp

- Event-Driven Programming



Window management library
GLUT / FLTK

Events: key strokes, mouse clicks

Event handlers:
mouse(),
display(),
reshape()

Cube.cpp

```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

```
void keyboard(unsigned char key, int x, int y) {
   switch (key) {
      case 27:  //correspond to ESC
         exit(0);
         break;
   }
}
```

Cube.cpp

```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

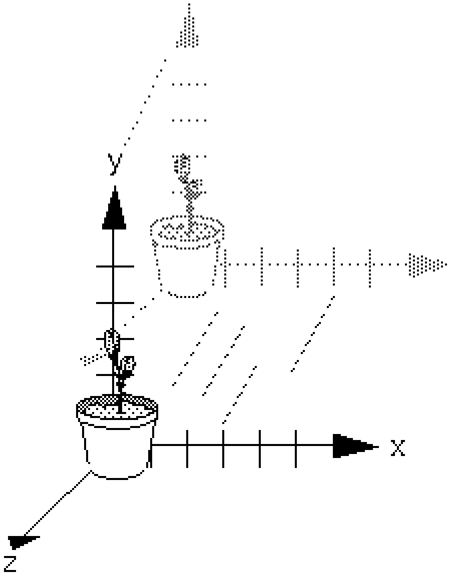
```
void reshape (int w, int h) {
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
}
```

Cube.cpp

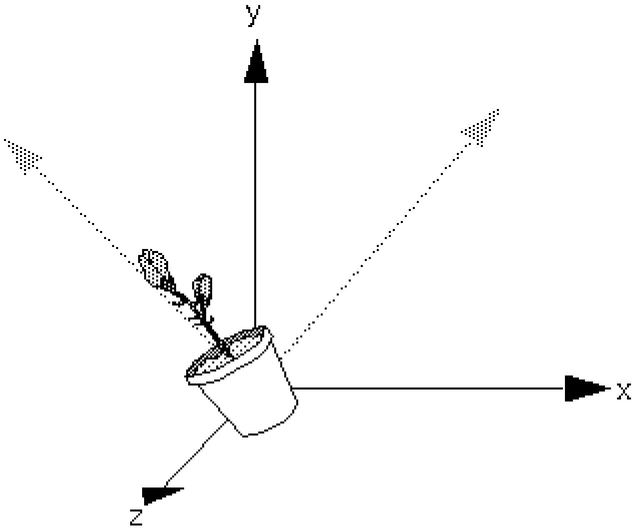
```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
```

```
void display(void) {
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
   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();
}
```

Modeling Transformation



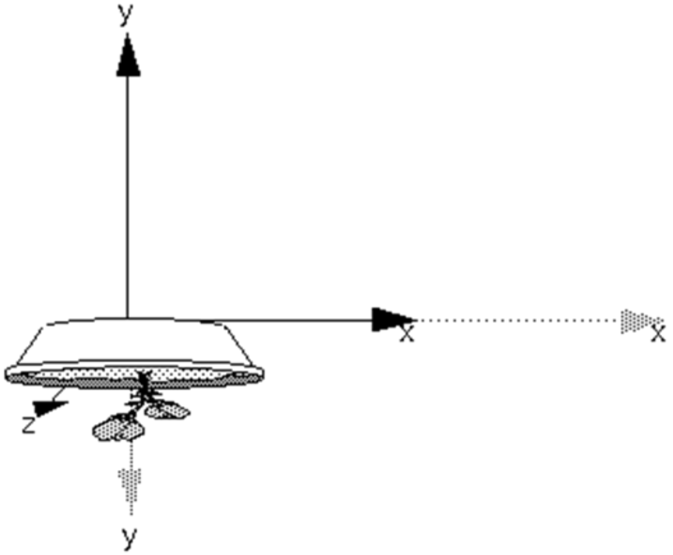
```
glTranslatef(float x, float y, float z)
```



```
glRotatef(float angle, float x, float y, float z)
```

```
glScalef(float x, float y, float z)
```

```
glScalef(2.0, -0.5, 1.0)
```



General Modeling Transform

```
glLoadIdentity()
```

```
glLoadMatrixf(float *M)
```

```
glMultiMatrixf(float *M)
```

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

double M[4][4];
M[2][1] corresponds to???

M[10]

Matrix Chain

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(N);                     /* apply transformation N */
glMultMatrixf(M);                     /* apply transformation M */
glMultMatrixf(L);                     /* apply transformation L */
glBegin(GL_POINTS);
glVertex3f(v);                        /* draw transformed vertex v */
glEnd();                              /* which is (N*(M*(L*v))) */
```

Matrix Chain

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glMultMatrixf(V);

glMultMatrixf(R);

glMultMatrixf(T);

glBegin(GL_POINTS);
glVertex3f(v);
glEnd();
```

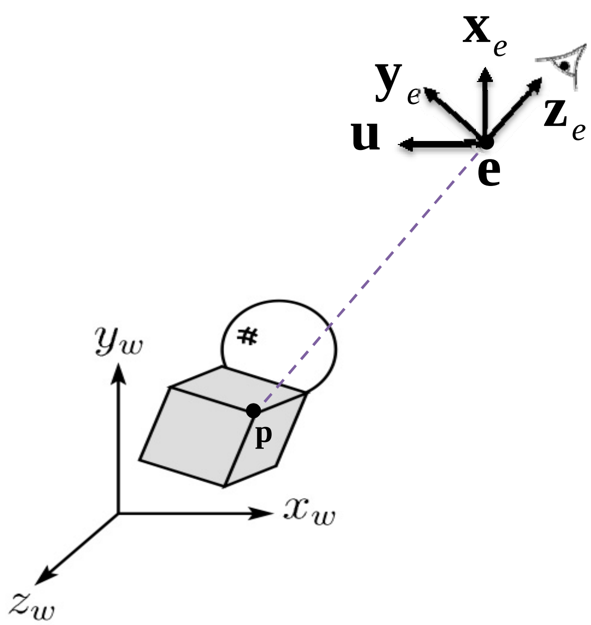
⟺

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

gluLookat(…);

glRotatef(…);

glTranslatef(…);

glBegin(GL_POINTS);
glVertex3f(v);
glEnd();
```

gluLookAt

```
gluLookAt(
    float eyex, float eyey, float eyez,
    float px, float py, float pz,
    float upx, float upy, float upz )
```

1. Give eye location e
2. Give target position p
3. Give upward direction u



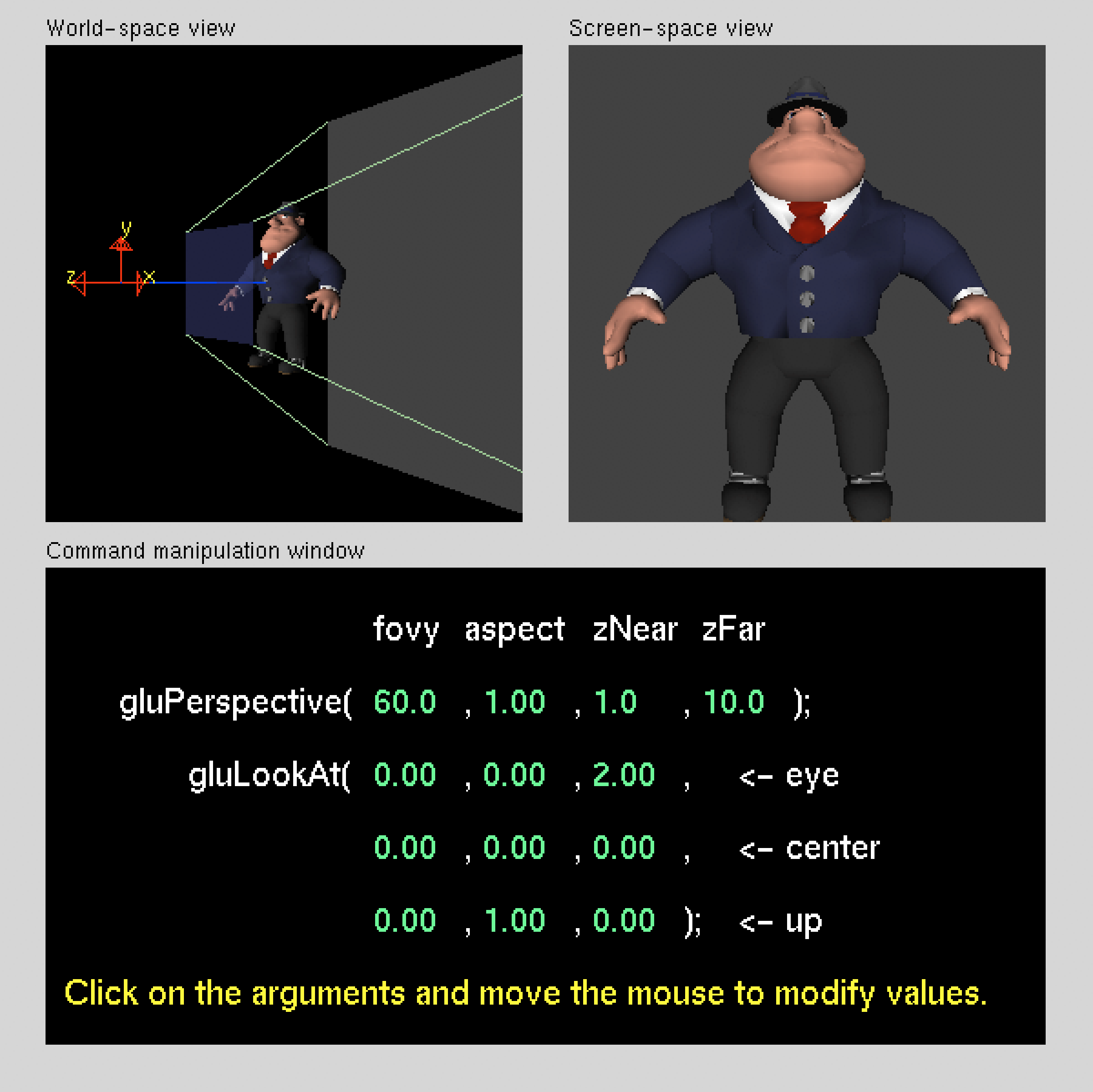
$$\mathbf{z}_e = -\frac{\mathbf{p}-\mathbf{e}}{|\mathbf{p}-\mathbf{e}|}$$

$$\mathbf{x}_e = \frac{\mathbf{u}\times\mathbf{z}_e}{|\mathbf{u}\times\mathbf{z}_e|}$$

$$\mathbf{y}_e = \frac{\mathbf{z}_e\times\mathbf{x}_e}{|\mathbf{z}_e\times\mathbf{x}_e|}$$

$$\mathbf{M}_v = \begin{bmatrix} - & \mathbf{x}_e^{\mathrm{T}} & - & 0 \\ - & \mathbf{y}_e^{\mathrm{T}} & - & 0 \\ - & \mathbf{z}_e^{\mathrm{T}} & - & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & | \\ 0 & 1 & 0 & -\mathbf{e} \\ 0 & 0 & 1 & | \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

gluLookAt

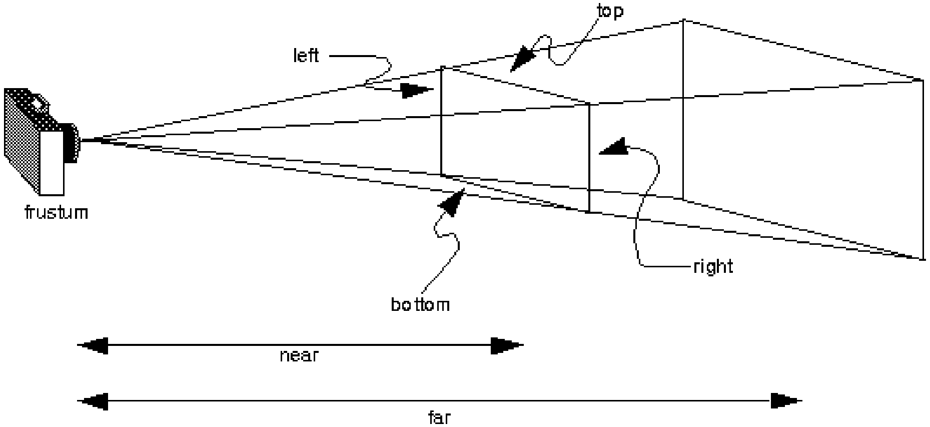


```
                 fovy   aspect   zNear   zFar

gluPerspective(  60.0  , 1.00   , 1.0    , 10.0   );

     gluLookAt(  0.00  , 0.00   , 2.00   ,      <- eye

                 0.00  , 0.00   , 0.00   ,      <- center

                 0.00  , 1.00   , 0.00   );     <- up
```

Click on the arguments and move the mouse to modify values.

glFrustum

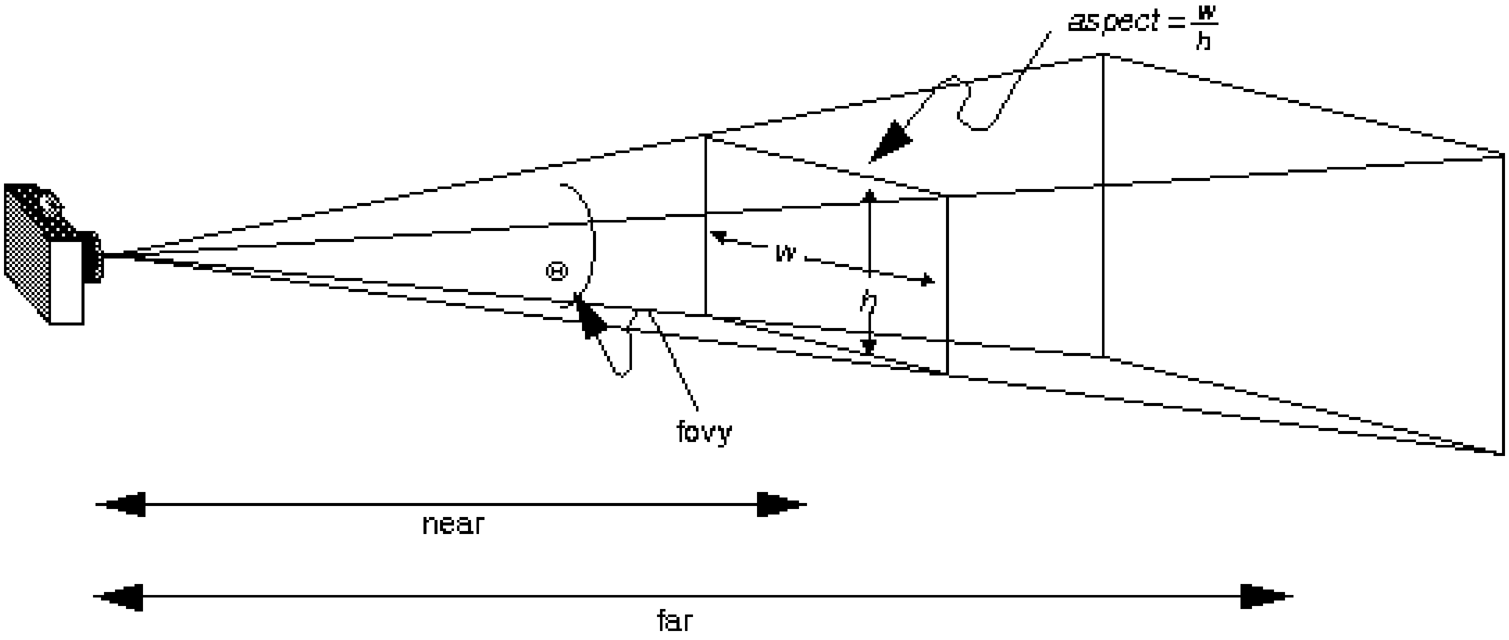
```
glFrustum(double left, double right, double bottom, double top, double near, double far)
```



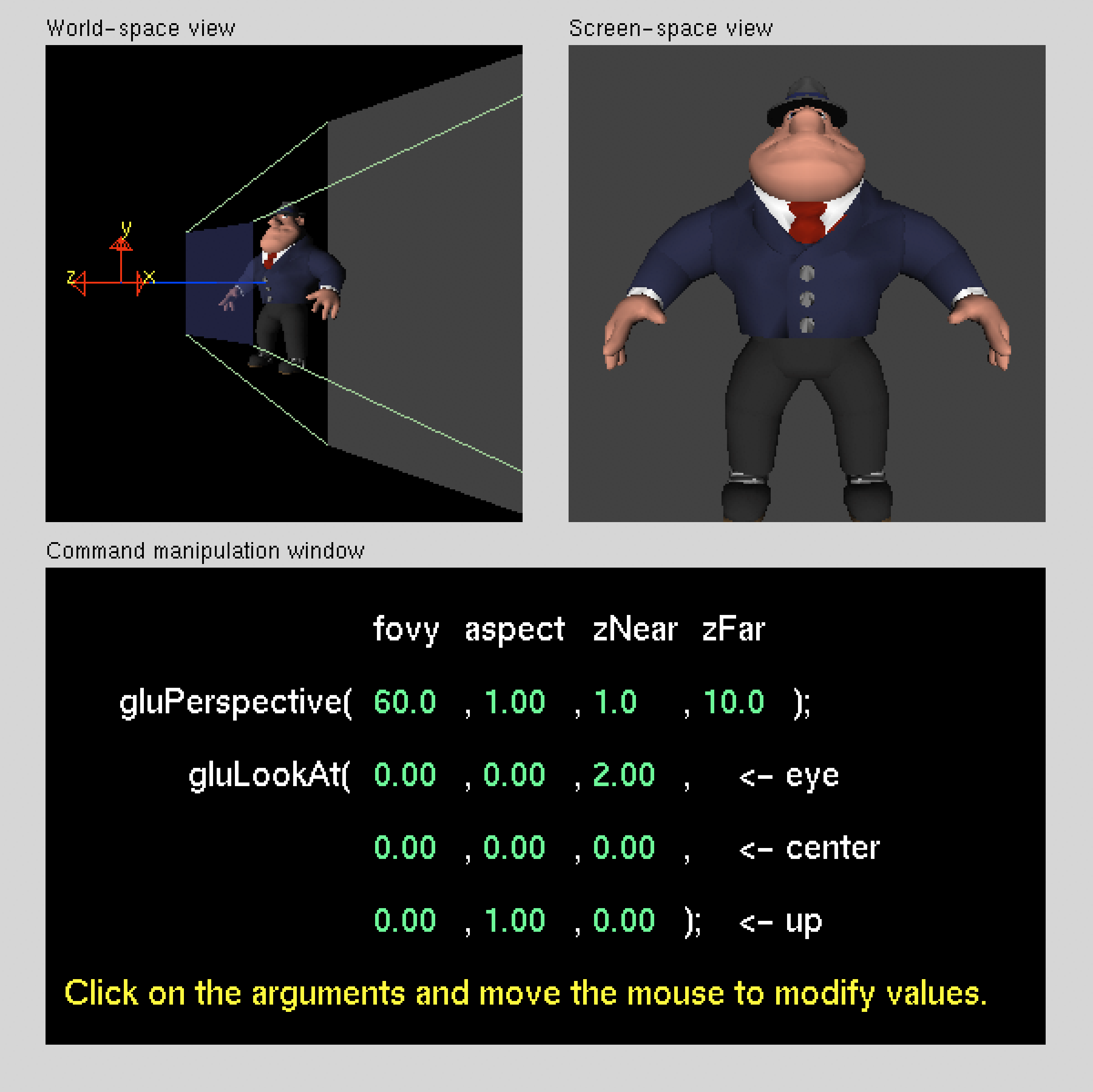
$$\mathbf{M}_{view->canonical} = \mathbf{M}_O \mathbf{M}_P = \begin{bmatrix} \frac{2}{(r-l)} & 0 & 0 & \frac{-(r+l)}{(r-l)} \\ 0 & \frac{2}{(t-b)} & 0 & \frac{-(t+b)}{(t-b)} \\ 0 & 0 & \frac{2}{(n-f)} & \frac{-(n+f)}{(n-f)} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & (n+f) & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

gluPerspective

```
gluPerspective(double fovy, double aspect, double zNear, double zFar)
```



gluPerspective

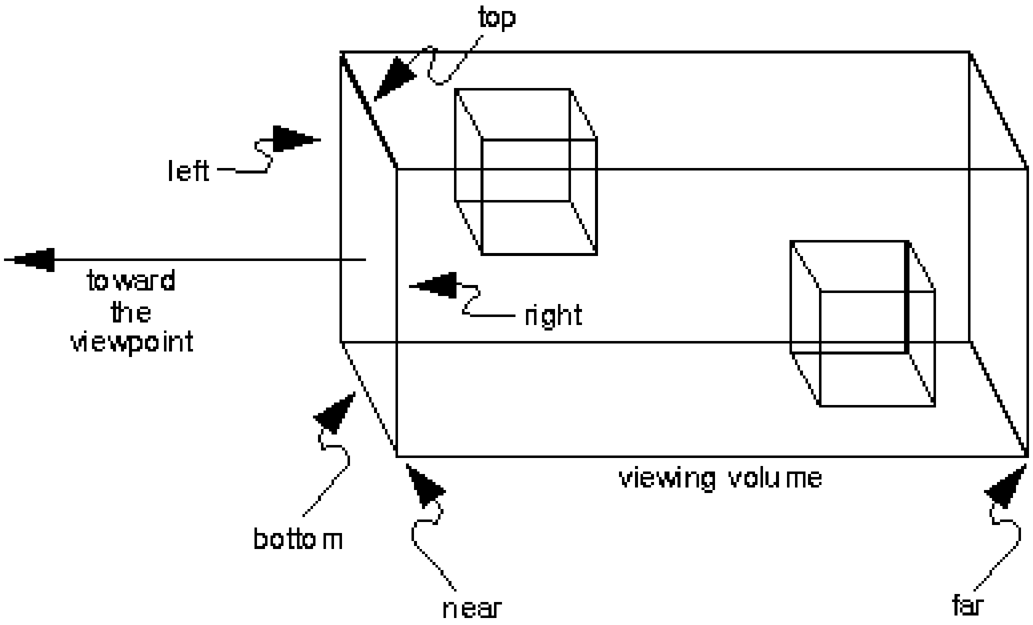


```
                 fovy   aspect   zNear   zFar

gluPerspective(  60.0  , 1.00   , 1.0    , 10.0   );

     gluLookAt(  0.00  , 0.00   , 2.00   ,     <- eye

                 0.00  , 0.00   , 0.00   ,     <- center

                 0.00  , 1.00   , 0.00   );    <- up
```

Click on the arguments and move the mouse to modify values.

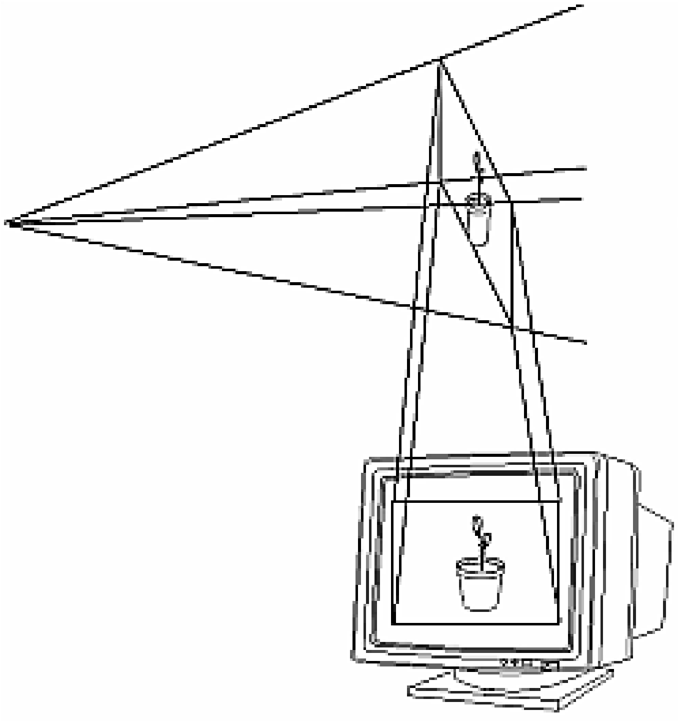
Orthographic Projection

```
glOrtho(double left, double right, double bottom,
double top, double near, double far);
```

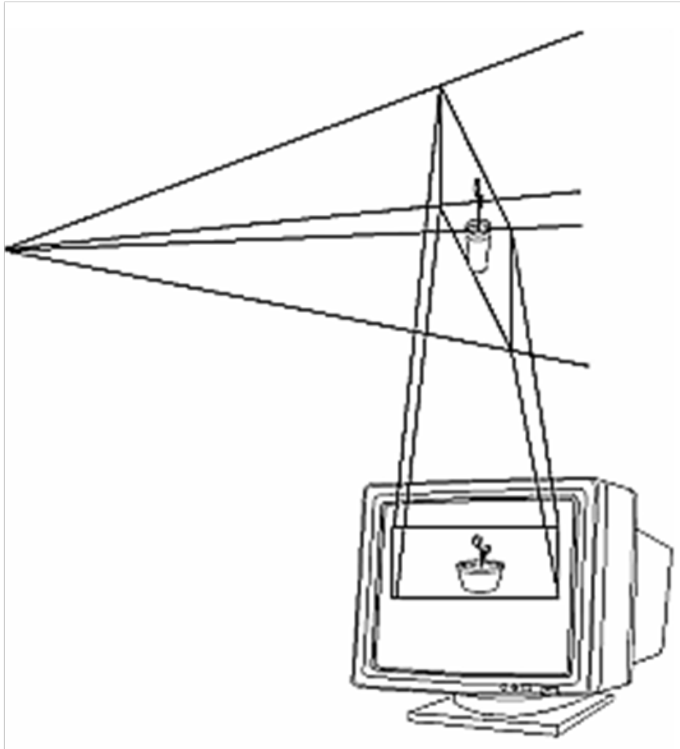


Viewport Transformation

```
glViewport(int x, int y, int width, int height);
```



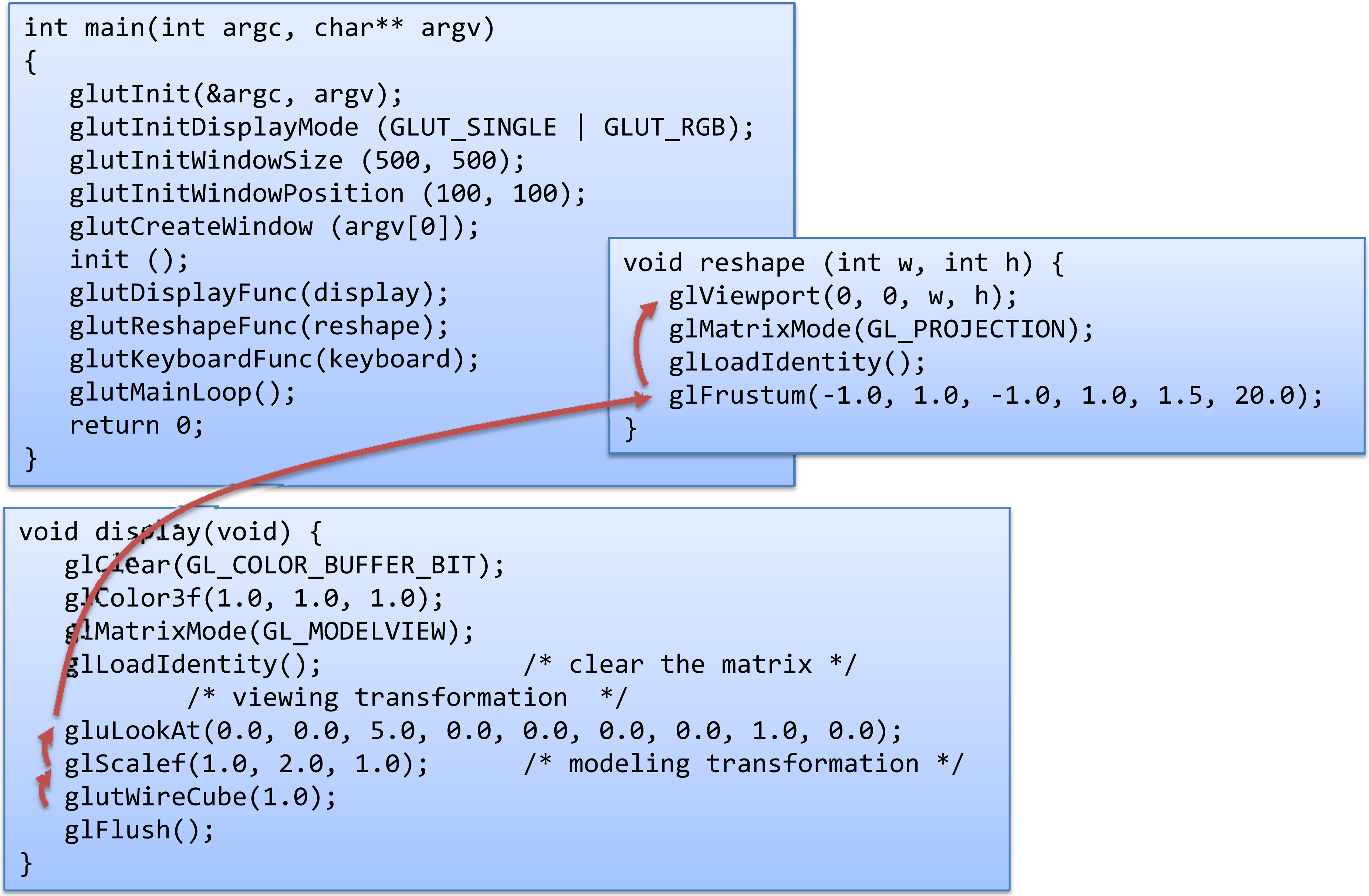
undistorted



distorted

```
glFrustum(double left, double right, double bottom, double top, double near, double far)
gluPerspective(double fovy, double aspect, double zNear, double zFar)
```

Revisit cube.cpp



```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

```
void reshape (int w, int h) {
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
}
```

```
void display(void) {
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
   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();
}
```