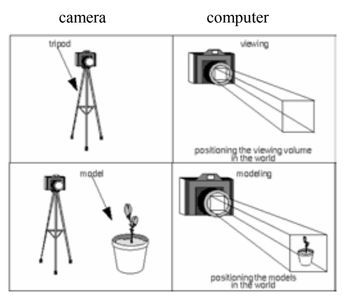
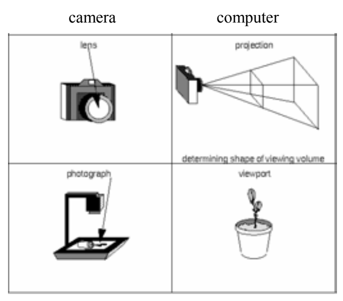
OpenGL Transformations

The Camera Analogy


camera
computer
tripod
viewing
positioning the viewing volume in the world
model
modeling
positioning the models in the world

The Camera Analogy



OpenGL Pipeline

- 4 steps pipeline :
  - Modelview
  - Projection
  - Perspective subdivision
  - Viewport

OpenGL Matrix Mode

- *`void glMatrixMode(mode)`*
  - *`GL_PROJECTION`* used to define projection matrix
  - *`GL_MODELVIEW`* used to define both model and camera transformation
- Matrix operations apply on the current matrix mode.
  - Caution: possible to define projection matrix in *`GL_MODELVIEW`* mode

Matrix Manipulation

- Assign the identity matrix to the current matrix:
  `void glLoadIdentity()`
- Declare float array to hold matrix data:
  `GLfloat m[16]; // 4x4 matrix`
- OpenGL holds the elements of 4x4 matrices in a 16 array:

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \Leftrightarrow [m_{00} \quad m_{10} \quad m_{20} \quad m_{30} \quad m_{01} \quad m_{11} \quad \cdots \quad m_{23} \quad m_{33}]$$

Matrix Manipulation

- Assign the current matrix values of matrix m:
  `void glLoadMatrix{fd}(m)`
- Multiply the current matrix by matrix m:
  `void glMultMatrix{fd}(m)`
- Get the value of 'matrix' into 'm':
  `void glGetFloatv(matrix, m)`
  - `GL_MODELVIEW_MATRIX`
  - `GL_PROJECTION_MATRIX`

Modelview Transformations

- `void glTranslate{fd}(x,y,z)`
- `void glRotate{fd}(angle,x,y,z)`
  Note: direction of rotation is according to right-hand rules.
- `void glScale{fd}(sx,sy,sz)`
- `void gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)`

Multiplication

- Let's denote the current matrix as **A**, calling:
  `glMultMatrix*()`, `glTranslate*()`, `glRotate*()`, or `glScale*()`

Perform a multiplication of **A** by another matrix, $\mathbf{A}_1$, from the **<u>right</u>**, resulting in:

$$\mathbf{A} = \mathbf{A} * \mathbf{A}_1$$

Question: what if we wanted the result to be:

$$\mathbf{A} = \mathbf{A}_1 * \mathbf{A} \quad ?$$

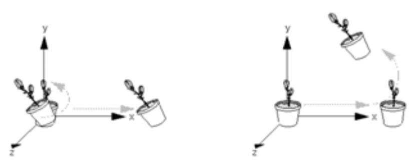
Multiplication

- Multiply the current ModelView matrix, **A**, by a rotation matrix, $\mathbf{A}_1$, from the **left**:

```
GLfloat  m[16];
glMatrixMode(GL_MODELVIEW);
glGetFloatv(GL_MODELVIEW_MATRIX, m);
glLoadIdentity();
// The rotation matrix multiplication
glRotated (45, 1, 0, 0);
glMultMatrixf(m);
```

- Result: $\mathbf{A} = \mathbf{A}_1 * \mathbf{A}$

Transformation Order

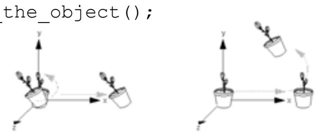


Thinking about Transformations

- Grand, fixed coordinate system:
  - Think of the multiplications as occurring in the opposite order from how they appear in the code
- Local coordinate system is tied to the object you're drawing
  - All operations occur relative to this changing coordinate system

Example

- Rotation about the origin and a translation along the *x*-axis:
- `glMatrixMode(GL_MODELVIEW);`
- ` glLoadIdentity();`
- `glMultMatrixf(T);  /* translation */`
- `glMultMatrixf(R);  /* rotation */`
- `draw_the_object();`



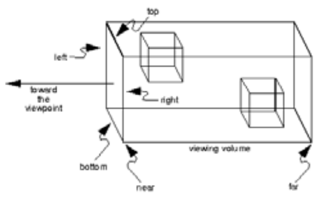
Projection Transformation

- Defines a *viewing volume*, used in two ways:
  - Determines how an object is projected onto the screen (perspective / orthographic)
  - Defines which objects or portions of objects are clipped out of the final image
- Usually a projection transformation is **not** combined with another transformation matrix:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

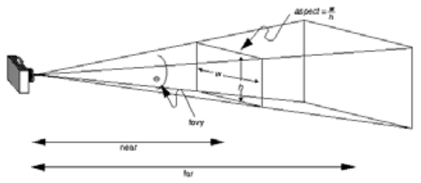
Orthographic Projections

- *`void glOrtho(left, right, bottom, top, near, far)`*
- *`void gluOrtho2D(left, right, bottom, top);`*



Perspective Projections

- *`void glFrustum(left, right, bottom, top, near, far);`*
- *`void gluPerspective(fovy, aspect, near, far);`*

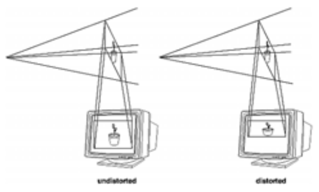


Viewport Transformation

- Determines size and proportions of the display window
- Aspect ratio of viewport should generally equal aspect ratio of viewing volume
- Application should detect window resize events and modify the viewport
- *`glViewport(x, y, width, height)`*

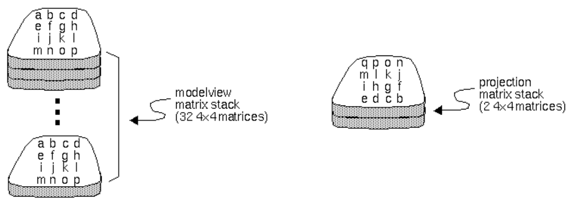
Viewport Transformation

- *`glViewport(x, y, width, height)`*



Matrix Stacks (top of the iceberg)

⌘Useful for constructing hierarchical models


a b c d
e f g h
i j k l
m n o p
modelview matrix stack (32 4×4 matrices)
q p o n
m l k j
i h g f
e d c b
projection matrix stack (2 4×4 matrices)

Matrix Stacks (top of the iceberg)

⌘OpenGL maintains two matrix stacks: **Modelview** and **Projection**

⌘Put a copy of current matrix on the top of the stack:

```
void glPushMatrix();
```

⌘Remove the matrix that is on top of the stack. Underlying matrix is now on top.

```
void glPopMatrix();
```

Matrix stack example

```
void DrawCar() {
     DrawBody();
     glPushMatrix();
     glTranslatef(40, 0, 0);
     DrawWheel();
     glPopMatrix();
     glPushMatrix();
     glTranslatef(-40, 0, 0);
     DrawWheel();
     glPopMatrix();
  }
```

Example II

Substitute the current pipeline transformations with transformations that draw vertices in screen coordinates:

```
void DeleteTrans() {
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glLoadIdentity();
  glMatrixMode(GL_PROJECTION);
  glPushMatrix();
  glLoadIdentity();
  gluOrtho2D(0, screen_width, 0,
  screen_height);
}
```

Example II

Restore the transformations settings that were prior to calling to DeletTrans():

```
void RestoreTrans() {
  glMatrixMode(GL_MODELVIEW);
  glPopMatrix();

  glMatrixMode(GL_PROJECTION);
  glPopMatrix();
}
```