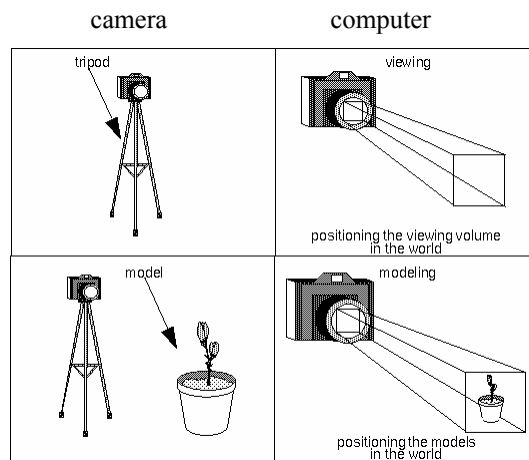


OpenGL Transformations

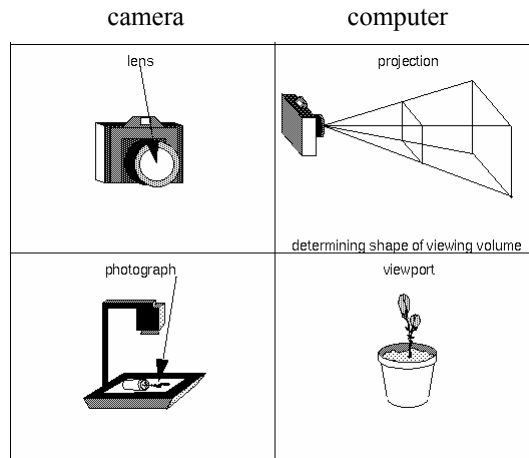
1

The Camera Analogy



2

The Camera Analogy



3

OpenGL Pipeline

⌘ 4 steps pipeline :

- ☒ Modelview
- ☒ Projection
- ☒ Perspective subdivision
- ☒ Viewport

4

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

5

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:

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

6

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
```

7

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)*

8

Multiplication

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

Perform a multiplication of **A** by another matrix,
A₁, from the **right**, resulting in:

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

Question: what if we wanted the result to be:

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

9

Multiplication

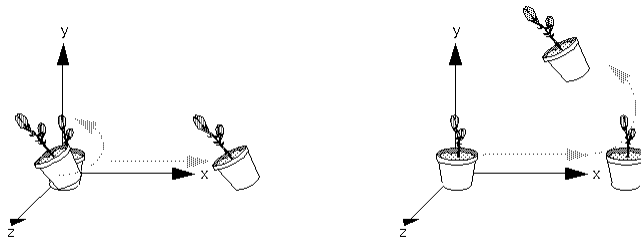
⌘ Multiply the current ModelView matrix, **A**, by a
rotation matrix, **A₁**, from the **left**:

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

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

10

Transformation Order



11

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

12

Example

⌘ Rotation about the origin and a translation along the x -axis:

```
⌘ glMatrixMode(GL_MODELVIEW);  
⌘ glLoadIdentity();  
⌘ glTranslatef(T); /* translation */  
⌘ glRotatef(R); /* rotation */  
⌘ draw_the_object();
```



13

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:

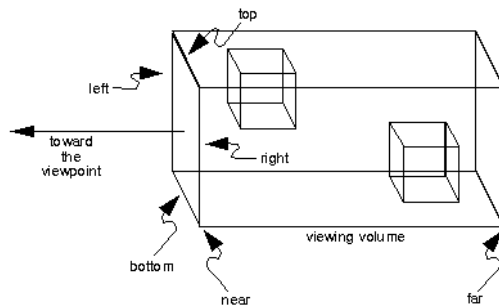
```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();
```

14

Orthographic Projections

⌘ `void glOrtho(left, right, bottom, top, near, far)`

⌘ `void gluOrtho2D(left, right, bottom, top);`

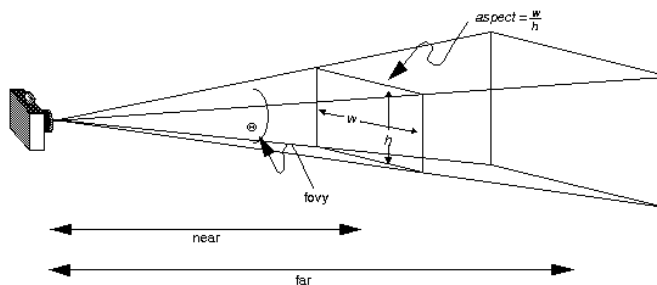


15

Perspective Projections

⌘ `void glFrustum(left, right, bottom, top, near, far);`

⌘ `void gluPerspective(fovy, aspect, near, far);`



16

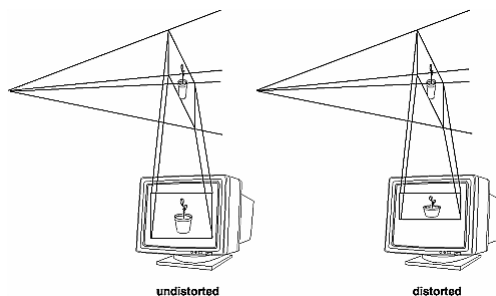
Viewport Transformation

- ⌘ 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)`

17

Viewport Transformation

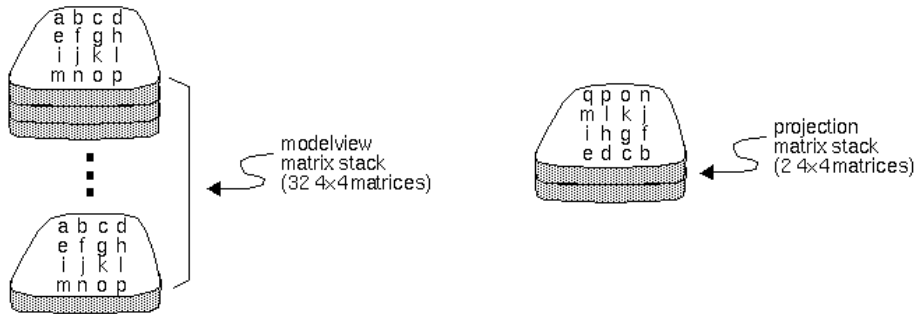
- ⌘ `glViewport(x, y, width, height)`



18

Matrix Stacks (top of the iceberg)

- ⌘ Useful for constructing hierarchical models



19

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();
```

20

Matrix stack example

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

21

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);
}
```

22

Example II

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

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