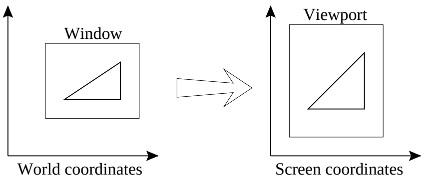
Outline

- Window to viewport transformations.
- How OpenGL does it.
- Efficiency.
- Inverse transformations.

Window to viewport transformations

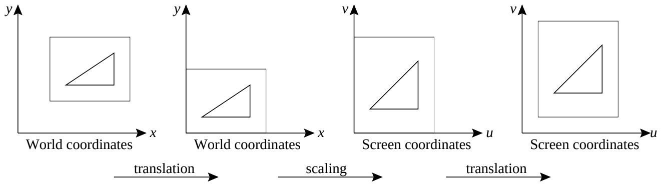
- The objects and primitives represented in the application model will be stored in world coordinates.
- Usually in terms of logical units for whatever the objects represent.
- To display the appropriate images on the screen (or other devices) it is necessary to map from world coordinates to screen or device coordinates.
- This transformation is known as the window to viewport transformation.

Window to viewport transformations



- Think of it as the window (on the world) to viewport (on screen) transformation.

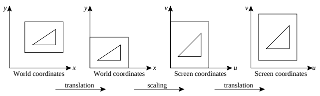
Window to viewport transformations



- In general the window to viewport transformation will involve a scaling and translation.

Window to viewport transformations

- Non uniform scalings result in the world coordinate window and viewport having different aspect ratios.
- The screen window (that is the viewport) typically covers only part of the screen.
- Generally the region will be clipped in world coordinates and then transformed.



Window to viewport transformations

- The transformation will be given by:
  - a translation;

$$T_x = \begin{bmatrix} 1 & 0 & -x_{\text{min}} \\ 0 & 1 & -y_{\text{min}} \\ 0 & 0 & 1 \end{bmatrix};$$

  - a scaling:

$$S_{xu} = \begin{bmatrix} \frac{u_{\text{max}}-u_{\text{min}}}{x_{\text{max}}-x_{\text{min}}} & 0 & 0 \\ 0 & \frac{v_{\text{max}}-v_{\text{min}}}{y_{\text{max}}-y_{\text{min}}} & 0 \\ 0 & 0 & 1 \end{bmatrix};$$

  - and finally another translation;

$$T_u = \begin{bmatrix} 1 & 0 & u_{\text{min}} \\ 0 & 1 & v_{\text{min}} \\ 0 & 0 & 1 \end{bmatrix}.$$

Window to viewport transformations

- Combination, $M_{xu} = T_u S_{xu} T_x =$

$$M_{xu} = \begin{bmatrix} \frac{u_{\text{max}}-u_{\text{min}}}{x_{\text{max}}-x_{\text{min}}} & 0 & -x_{\text{min}}\frac{u_{\text{max}}-u_{\text{min}}}{x_{\text{max}}-x_{\text{min}}} + u_{\text{min}} \\ 0 & \frac{v_{\text{max}}-v_{\text{min}}}{y_{\text{max}}-y_{\text{min}}} & -y_{\text{min}}\frac{v_{\text{max}}-v_{\text{min}}}{y_{\text{max}}-y_{\text{min}}} + v_{\text{min}} \\ 0 & 0 & 1 \end{bmatrix}.$$

- Note the inverted order of transformation – T_x will be applied first.
- Use of homogeneous coordinates has allowed a single transformation matrix to be written.

Window to viewport transformations – OpenGL

- In OpenGL GLUT sets up the viewport (i.e. screen window) position and size, OpenGL defines the mapping:

```
glutInitWindowSize(400,200);
glutInitWindowPosition(100,150);
glViewport(0, 0, 400, 200);
```

- The window (on the world) is set by the projection matrix – in 2D we have:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,400.0,0.0,200.0);
```

Efficiency – your job

- The composition of transformations involving translation, scaling and rotation leads to transformation matrices M of the general form:

$$M = \begin{bmatrix} r_{1,1} & r_{1,2} & t_x \\ r_{2,1} & r_{2,2} & t_y \\ 0 & 0 & 1 \end{bmatrix} .$$

- How many elementary operations $(+, -, *, /)$ are required to multiply a vector $[x, y, w]'$ by this matrix?

Efficiency

- The composition of transformations involving translation, scaling and rotation leads to transformation matrices M of the general form:

$$M = \begin{bmatrix} r_{1,1} & r_{1,2} & t_x \\ r_{2,1} & r_{2,2} & t_y \\ 0 & 0 & 1 \end{bmatrix} .$$

- Multiplying a point $[x, y, w]'$ by this matrix would take nine multiplies and six adds.

Efficiency

- Using the fixed structure in the final row of the matrix, the actual transformation can be written:

$$x^* = r_{1,1}x + r_{1,2}y + t_x ,$$

$$y^* = r_{2,2}y + r_{2,1}x + t_y ,$$

- This takes four multiplies and four adds.
- Parallel hardware adders and multipliers effectively remove this concern.

Inverse transformations

- If a general transformation (which may be composite) is given by the 3×3 matrix M, then the inverse transformation, which maps the new image back to the original, untransformed one is given by M^{-1}.
- The matrix inverse is defined so that $MM^{-1} = I$ where I is the 3×3 identity matrix,

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} .$$

- Inverses only exist for one to one transformations, for many operations they are not defined.

Inverse transformations

- The translation matrix has inverse:

$$T^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} ,$$

- The scaling matrix has inverse:

$$S^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix} .$$

Inverse transformations

- You can easily determine the inverse transformation matrix in Matlab.

```
>> M = [ 2 0 -3; 0 1 4; 0 0 1]

M =  2     0    -3
     0     1     4
     0     0     1

>> inv(M)

ans = 0.5      0    1.5
        0    1.0   -4.0
        0      0    1.0
```

Inverse of composite transformations

- For a composite transformation matrix $C = AB$ the inverse is less obvious.
- We know $(AB)^{-1} = B^{-1}A^{-1}$ from linear algebra thus:

$$C^{-1} = B^{-1}A^{-1} .$$

- This is logical – the inverse transformations are applied in the opposite order.

Summary

- Having finished this lecture you should:
  - be able to write down the transformation matrices for the window to viewport transformation;
  - know how OpenGL implements the window to viewport transformation;
  - be able to improve the efficiency of applying transformations;
  - understand the role and concept of an inverse transformation.
- It may help to experiment in OpenGL with changing the window to viewport transformation.