



Xi'an Jiaotong-Liverpool University
西交利物浦大學

CPT205 Computer Graphics

Transformation Pipeline and

Geometric Transformations

Lecture 04
2024-25

Yong Yue and Nan Xiang

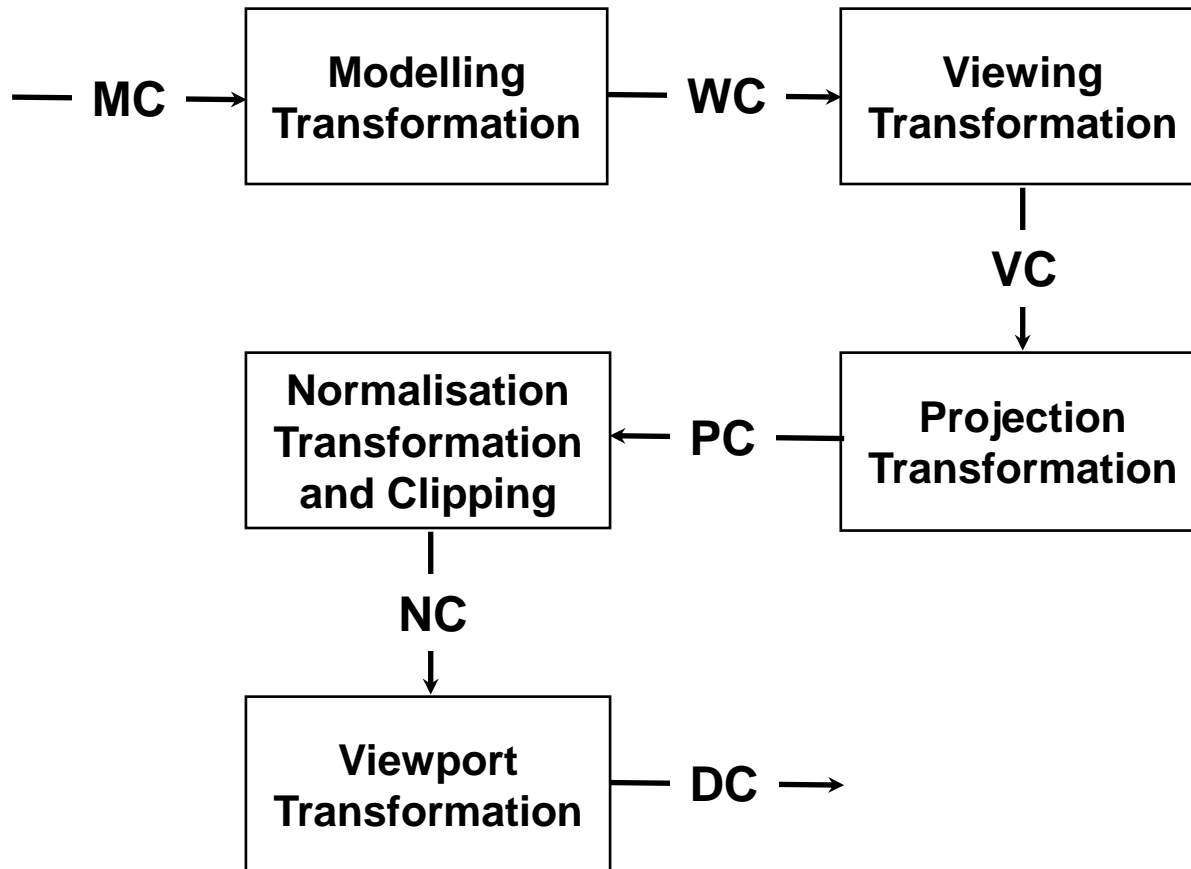
Topics for today

- Transformation pipeline
- Standard transformations
 - Translation
 - Rotation
 - Scaling
 - Reflection
 - Shearing
- Homogeneous co-ordinate transformation matrices
- Composite (arbitrary) transformation matrices from simple transformations
- OpenGL functions for transformations

Transformation pipeline (1)

- The Transformation Pipeline is the series of transformations (alterations) that must be applied to an object before it can be properly displayed on the screen.
- The transformations can be thought of as a set of processing stages. If a stage is omitted, very often the object will not look correct. For example if the *projection* stage is skipped then the object will not appear to have any depth to it.
- Once an object has passed through the pipeline it is ready to be displayed as either a wire-frame item or as a solid item.

Transformation pipeline (2)



Transformation pipeline (3)

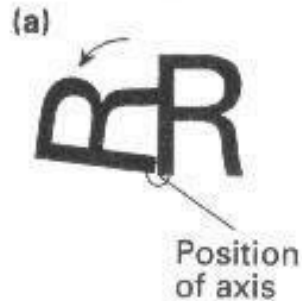
- Modelling Transformation - to place an object into the Virtual World.
- Viewing Transformation - to view the object from a different vantage point in the virtual world.
- Projection Transformation - to see depth in the object.
- Viewport Transformation - to temporarily map the volume defined by the "window of interest" plus the front and rear clipping planes into a unit cube. When this is the case, certain other operations are easier to perform.
- Device Transformation - to map the user defined "window of interest" (in the virtual world) to the dimensions of the display area.

Transformation pipeline (4)

- We start when the object is loaded from a file and is ready to be processed.
- We finish when the object is ready to be displayed on the computer screen.
- We should be able to draw a picture of a simple object, say a cuboid, and show visually what happens to it as it passes through each pipeline stage.

Types of geometric transformation

Rotation



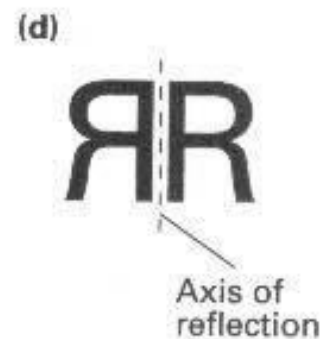
Translation



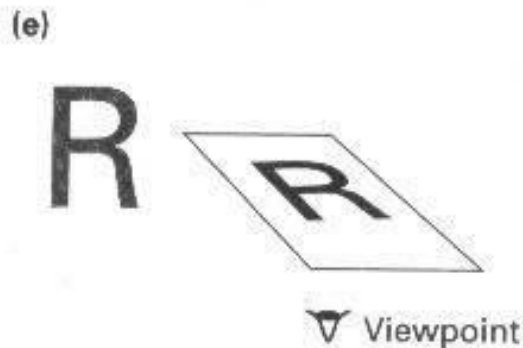
Scaling



Reflection



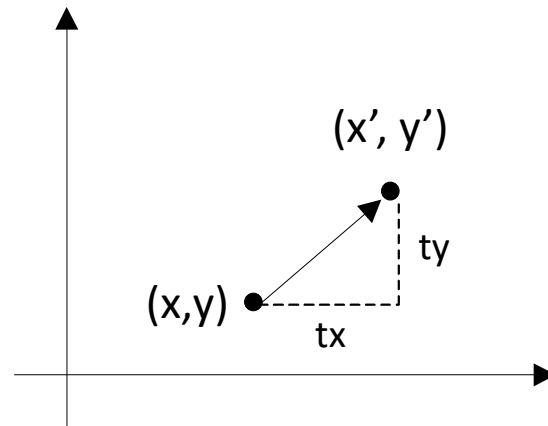
Shearing



2D translation (1)

- Translating a point from $P(x, y)$ to $P'(x', y')$ along vector T
- Importance in computer graphics – we need to only transform the two endpoints of a line segment and let the implementation draw the line segment between the transformed endpoints

$$\begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases}$$



2D translation (2)

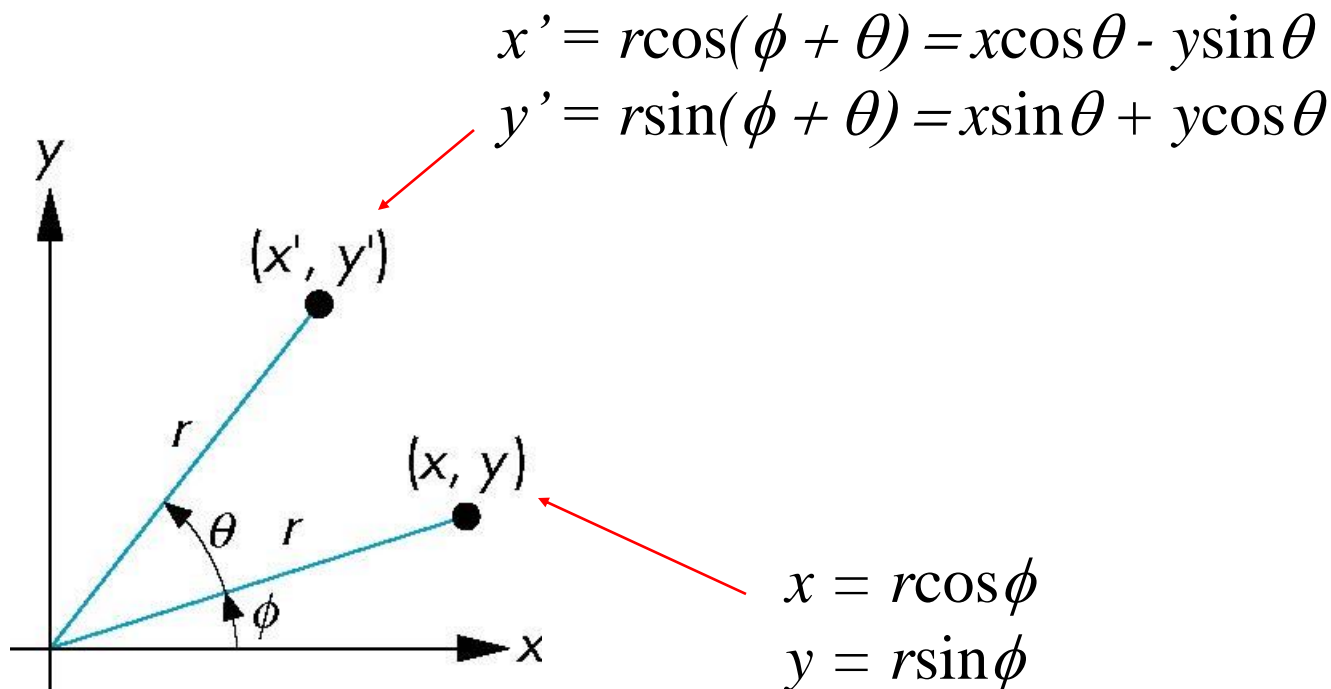
$$\mathbf{P} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{P}' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

where $\mathbf{P}(x, y)$ and $\mathbf{P}'(x', y')$ are the original and new positions, and \mathbf{T} is the distance translated.

$$\mathbf{P}' = \mathbf{P} + \mathbf{T} \quad \text{or} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

2D rotation (1)

Rotating a point from $P(x, y)$ to $P'(x', y')$ about the origin by angle θ - radius stays the same, and angle increases by θ .



2D rotation (2)

$$\begin{cases} x' = r \cos(\phi + \theta) = r \cos \phi \cos \theta - r \sin \phi \sin \theta \\ y' = r \sin(\phi + \theta) = r \cos \phi \sin \theta + r \sin \phi \cos \theta \end{cases}$$

$$\begin{cases} x = r \cos \phi \\ y = r \sin \phi \end{cases}$$

$$\begin{cases} x' = x \cos \theta - y \sin \theta \\ y' = x \sin \theta + y \cos \theta \end{cases}$$

2D rotation (3)

$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P}$$

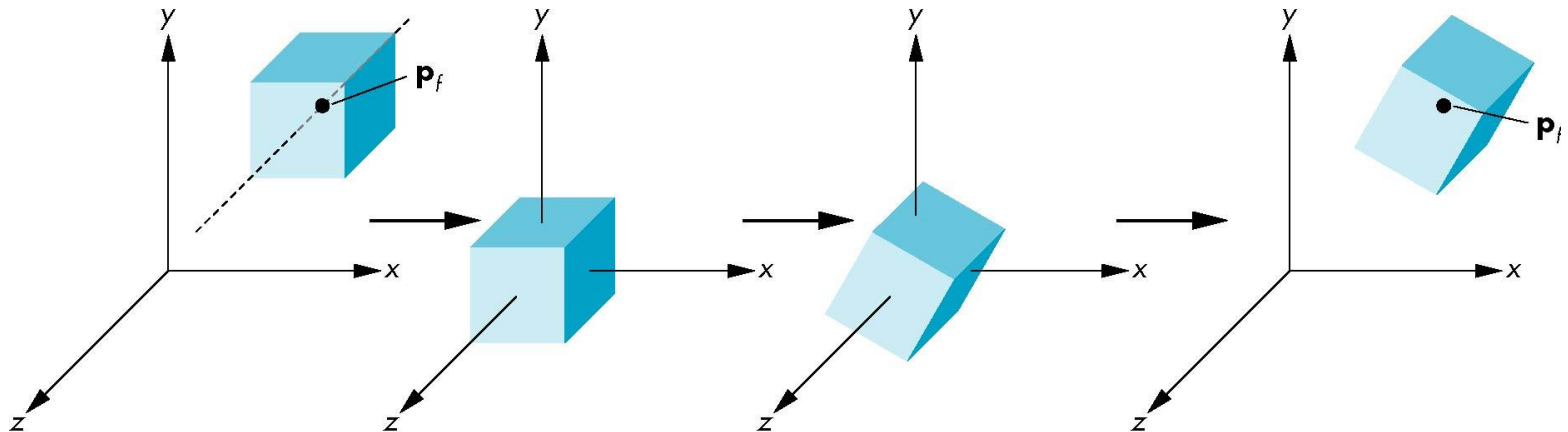
$$\mathbf{R} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad \text{so} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

where θ is the rotation angle and ϕ is the angle between the x-axis and the line from the origin to (x, y) .

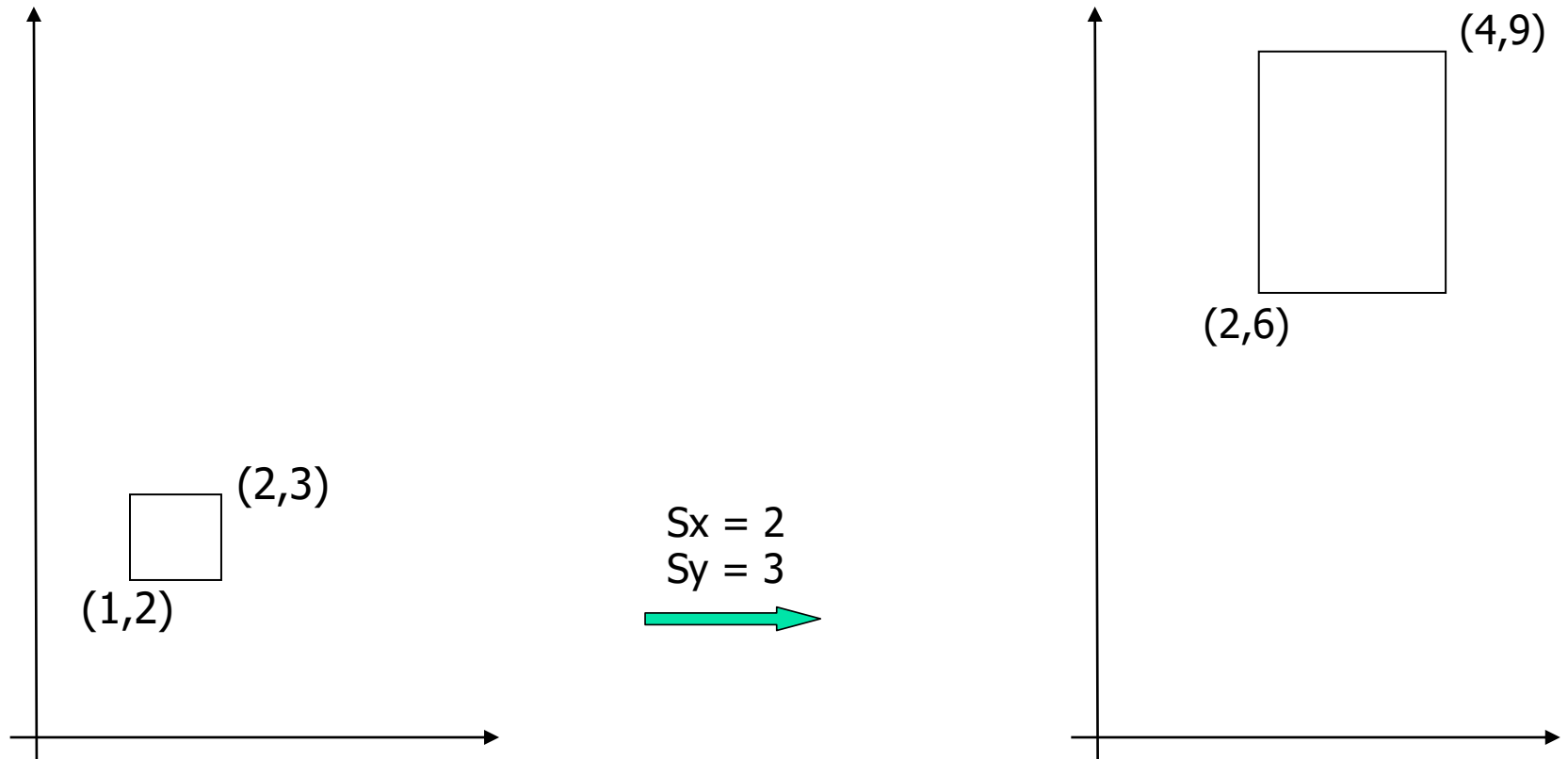
Notice that the rotation point (or pivot point) is the coordinate origin.

Rotation about a fixed point rather than the origin

- Move the fixed point to the origin
- Rotate the object
- Move the fixed point back to its initial position
- $\mathbf{M} = \mathbf{T}(\mathbf{p}_f) \mathbf{R}(\theta) \mathbf{T}(-\mathbf{p}_f)$



2D scaling (1)



When an object is scaled, both the size and location change.

2D scaling (2)

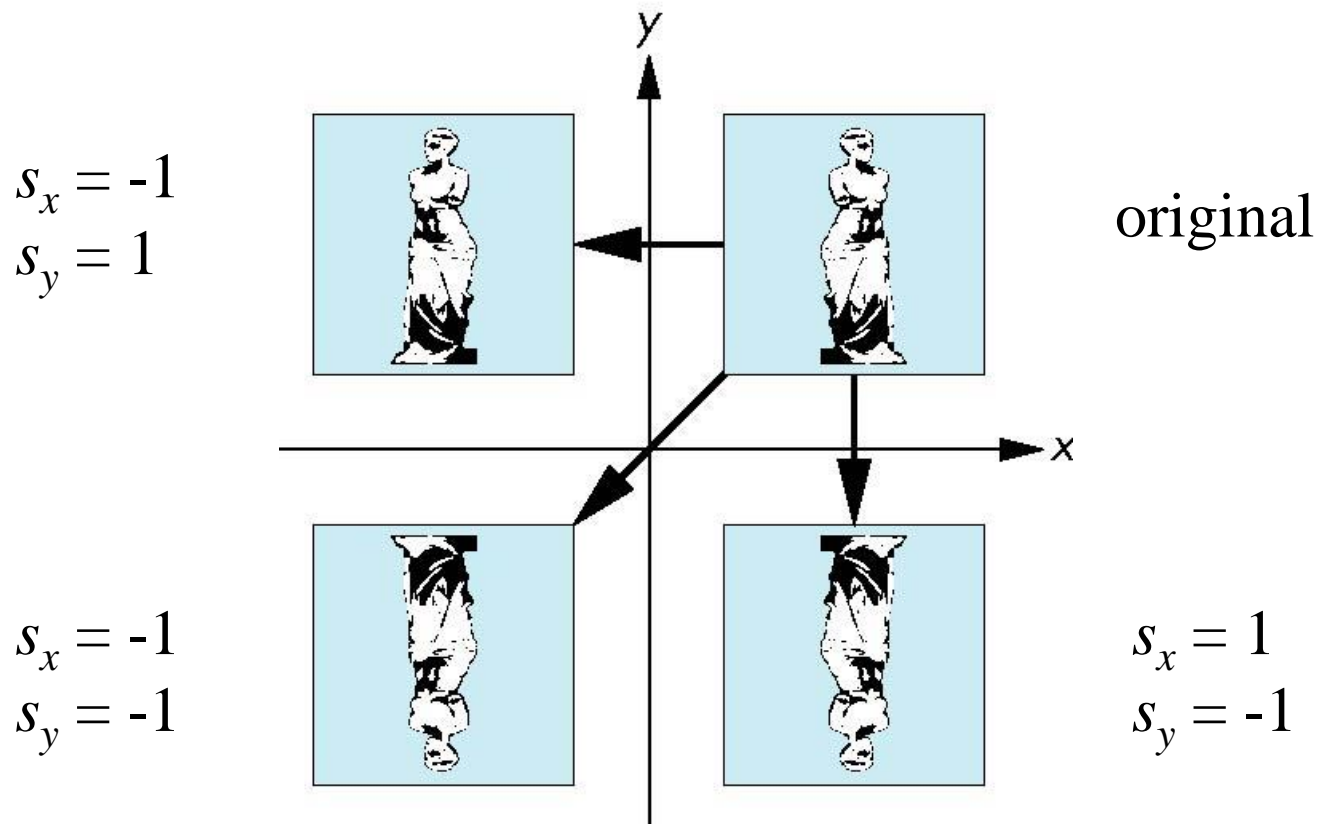
$$\begin{cases} x' = x \cdot s_x \\ y' = y \cdot s_y \end{cases}$$

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P} \quad \text{so} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

where \mathbf{P} , and \mathbf{P}' are the original and new positions, and s_x and s_y are the scaling factors along the x - and y -axes.

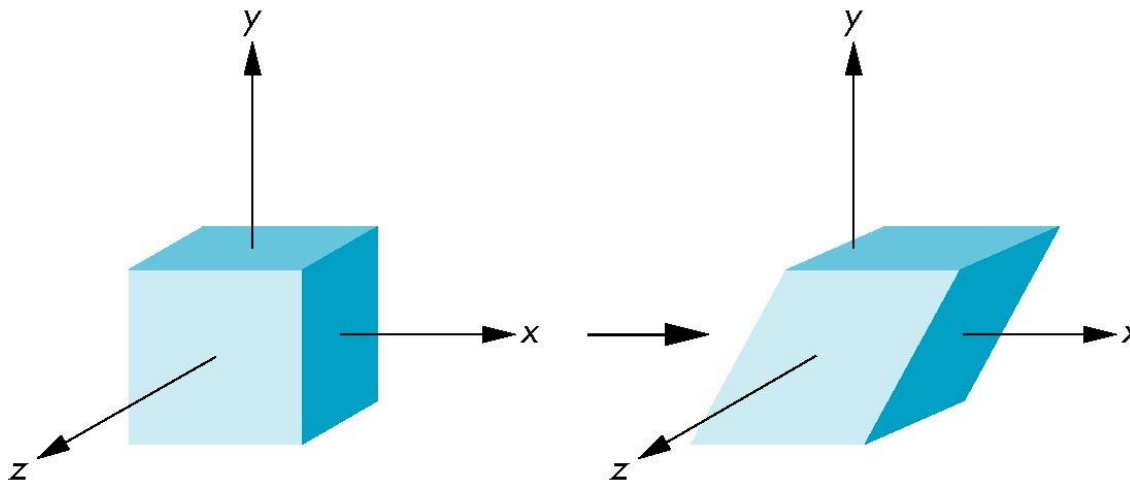
2D reflection

Special case of scaling - corresponding to negative scale factors.



2D shearing (1)

Equivalent to pulling faces in opposite directions.



2D shearing (2)

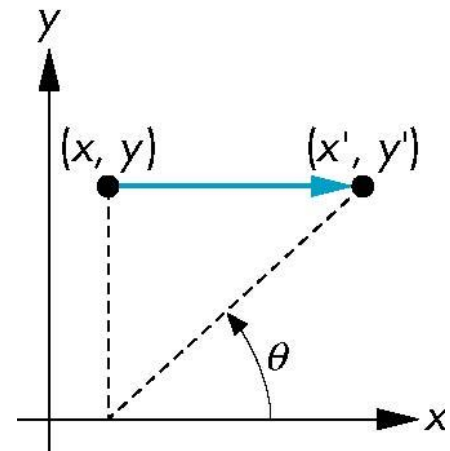
Consider simple shearing along the x axis.

$$x' = x + y \cot \theta$$

$$y' = y$$

$$z' = z$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & \cot \theta \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$



2D homogeneous co-ordinates

- By expanding the 2x2 matrices to 3x3 matrices, homogeneous co-ordinates are used.
- A homogeneous parameter is applied so that each 2D position is represented with homogeneous co-ordinates $(h \cdot x, h \cdot y, h)$.
- The homogeneous parameter has a non-zero value, and can be set to 1 for convenient use.
- This makes all the transformation matrices into the same format, i.e. 3x3 matrices, including the translation matrix.
- The homogenous transformation matrices can then be combined into a composite transformation matrix.

2D homogeneous matrices (1)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2D \text{ translation})$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2D \text{ rotation})$$

2D homogeneous matrices (2)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2D \text{ scaling})$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \cot \theta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2D \text{ shearing})$$

2D composite transformation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} rs_{xx} & rs_{xy} & trs_x \\ rs_{yx} & rs_{yy} & trs_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where elements rs are the multiplicative rotation-scaling terms in the transformation (which involve only rotation angles and scaling factors);

elements trs are the translation terms, containing combination of translation distances, pivot-point and fixed-point co-ordinates, rotation angles and scaling parameters.

3D translation

3D translations and scaling can be simply extended from the corresponding 2D methods.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D co-ordinate axis rotations (1)

- The extension from 2D rotation methods to 3D rotation is less straightforward (because this is about an arbitrary axis instead of an arbitrary point).
- Equivalent to rotation in two dimensions in planes of constant z (i.e. about the origin).

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (\text{About } z\text{-axis})$$

3D co-ordinate axis rotations (2)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{(About y-axis)}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \text{(About x-axis)}$$

General rotation about the origin

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

$$\mathbf{R}(q) = \mathbf{R}_z(q_z) \mathbf{R}_y(q_y) \mathbf{R}_x(q_x)$$

where q_x , q_y and q_z are called the Euler angles.

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

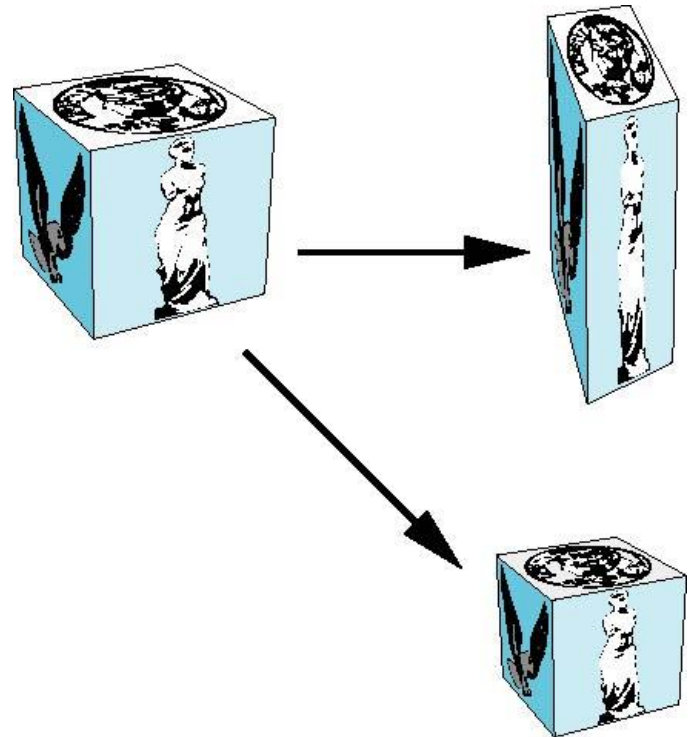
3D scaling

$$x' = s_x x$$

$$y' = s_y y$$

$$z' = s_z z$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



3D composite transformation (1)

- As with 2D transformation, a composite 3D transformation can be formed by multiplying the matrix representations for the individual operations in the transformation sequence.
- There are further forms of transformation, namely reflection and shearing which can be implemented with the other three transformations.
- Translation, scaling, rotation, reflection and shearing are all affine transformations in that transformed point $P'(x',y',z')$ is a **linear combination** of the original point $P(x,y,z)$.

3D composite transformation (2)

- Matrix multiplication is associative

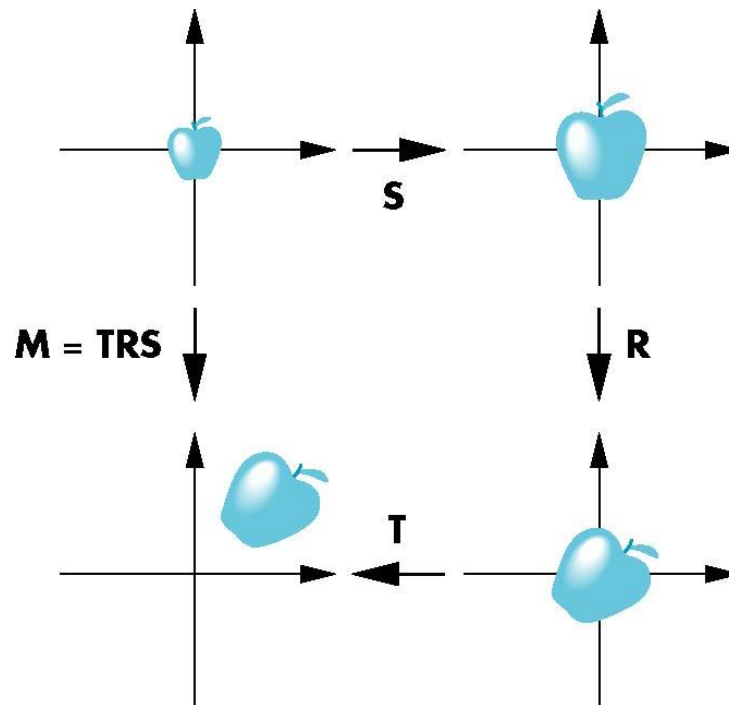
$$M3 \cdot M2 \cdot M1 = (M3 \cdot M2) \cdot M1 = M3 \cdot (M2 \cdot M1)$$

- Transformation products are **not always commutative**

$$A \cdot B \neq B \cdot A$$

Instancing

- In modelling, we often start with a simple object centred at the origin, oriented with an axis, and of a standard size.
- We apply an *instance transformation* to its vertices to
 - Scale
 - Orient
 - Locate



OpenGL matrices

- In OpenGL, matrices are part of the state.
- Multiple types
 - Model-View (`GL_MODELVIEW`)
 - Projection (`GL_PROJECTION`)
 - Texture (`GL_TEXTURE`)
 - Color (`GL_COLOR`)
- Single set of functions for manipulation
- Select which to be manipulated by
 - `glMatrixMode(GL_MODELVIEW) ;`
 - `glMatrixMode(GL_PROJECTION) ;`

Current transformation matrix

- Conceptually there is a 4x4 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.
- The CTM is defined in the user program and loaded into a transformation unit.
- The CTM can be altered either by loading a new CTM (e.g. Identity matrix) or by postmultiplication.
- OpenGL has a model-view and a projection matrix in the pipeline which are concatenated together to form the CTM.
- The CTM can manipulate each by first setting the correct matrix mode.

CTM operations

The CTM can be altered either by loading a new CTM or by postmultiplication

Load an identity matrix: $\mathbf{C} \leftarrow \mathbf{I}$

Load an arbitrary matrix: $\mathbf{C} \leftarrow \mathbf{M}$

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: $\mathbf{C} \leftarrow \mathbf{CM}$

Postmultiply by a translation matrix: $\mathbf{C} \leftarrow \mathbf{CT}$

Postmultiply by a rotation matrix: $\mathbf{C} \leftarrow \mathbf{CR}$

Postmultiply by a scaling matrix: $\mathbf{C} \leftarrow \mathbf{CS}$

Arbitrary Matrices

- We can load and multiply by matrices defined in the application program
 - `glLoadMatrixf(m)`
 - `glMultMatrixf(m)`
- Matrix **m** is a one-dimension array of 16 elements which are the components of the desired 4 x 4 matrix stored by columns.
- In `glMultMatrixf`, **m** multiplies the existing matrix on the right.

Matrix stacks

- CTM is not just one matrix but a matrix stack with the “current” at top.
- In many situations we want to save transformation matrices for use later
 - Traversing hierarchical data structures
 - Avoiding state changes when executing display lists
- Pre 3.1 OpenGL maintains stacks for each type of matrix
 - Access present type (as set by **glMatrixMode**) by
glPushMatrix()
glPopMatrix()
- Right now just 1-level CTM.

Matrix stacks

➤ We can also access matrices (and other parts of the state) with *query* functions

- `glGetIntegerv`
- `glGetFloatv`
- `glGetBooleanv`
- `glGetDoublev`
- `glIsEnabled`

➤ For matrices, we use as

- `double m[16];`
- `glGetFloatv(GL_MODELVIEW, m);`

OpenGL transformation functions (1)

- **glTranslate***
Specify translation parameters
- **glRotate***
Specify rotation parameters for rotation about any axis through the origin
- **glScale***
Specify scaling parameters with respect to the co-ordinate origin
- **glMatrixMode**
Specify current matrix for geometric-viewing, projection, texture or colour transformations
- **glLoadIdentity**
Set current matrix to identity
- **glLoadMatrix* (elems)**
Set elements of current matrix

OpenGL transformation functions (2)

- **glMultMatrix* (elems)**
Post-multiply the current matrix by the specified matrix
- **glGetIntegerv**
Get max stack depth or current number of matrices in the stack for selected matrix mode
- **glPushMatrix**
Copy the top matrix in the stack and store copy in the second stack position
- **glPopMatrix**
Erase top matrix in stack and move second matrix to top stack
- **glPixelZoom**
Specify 2D scaling parameters for raster operations

Example of rotation

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

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glTranslatef(1.0, 2.0, 3.0);  
glRotatef(30.0, 0.0, 0.0, 1.0);  
glTranslatef(-1.0, -2.0, -3.0);
```

- Note that the last matrix specified in the program is the first applied.

Summary

- Transformation pipeline
- Standard transformations
 - Rotation
 - Translation
 - Scaling
 - Reflection
 - Shearing
- Homogeneous co-ordinate transformation matrices
- Composite (arbitrary) transformation matrices from simple transformations
- OpenGL functions for transformations