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

CPT205 Computer Graphics

Transformation Pipeline and Geometric Transformations

Lecture 04
2024-25

**Yong Yue and Nan Xiang**

Topics for today
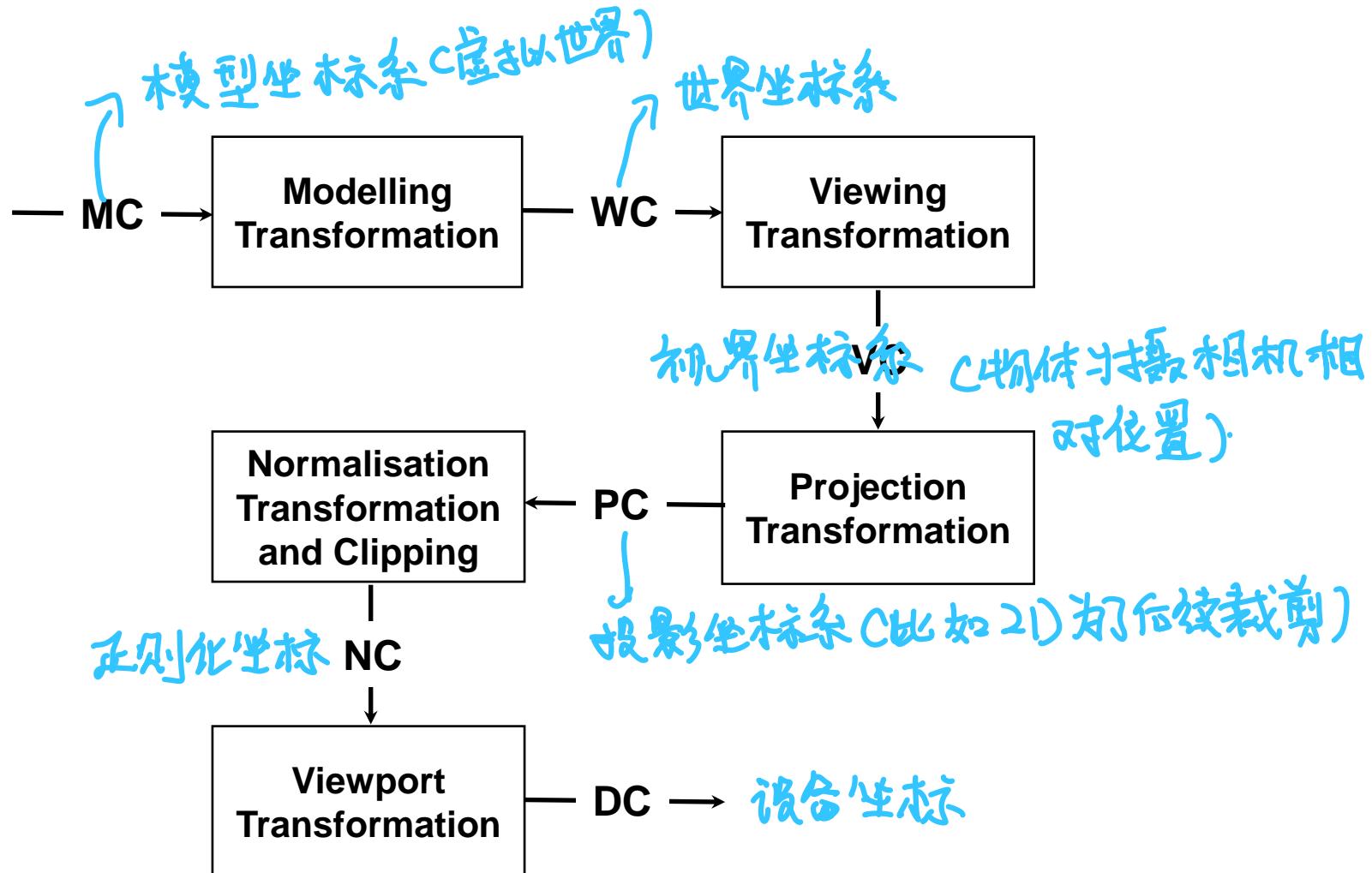
- ➢ Transformation pipeline
- ➢ Standard transformations
  - Translation
  - Rotation
  - Scaling
  - Reflection
  - Shearing
- ➢ Homogeneous co-ordinate transformation matrices
- ➢ Composite (arbitrary) transformation matrices from simple transformations
- ➢ OpenGL functions for transformations
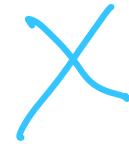
Transformation pipeline (1)

➢ 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)

模型坐标系（虚拟世界）

世界坐标系

MC → **Modelling Transformation** → WC → **Viewing Transformation**

视界坐标系 VC （物体对摄相机相对位置）

Normalisation Transformation and Clipping ← PC ← **Projection Transformation**

投影坐标系（比如2D）为后续裁剪）

正则化坐标 NC

Viewport Transformation — DC → 设备坐标

Transformation pipeline (3)

➢ <u>Modelling Transformation</u> - to place an object into the Virtual World.

➢ <u>Viewing Transformation</u> - to view the object from a different vantage point in the virtual world.

➢ <u>Projection Transformation</u> - to see depth in the object.

➢ <u>Viewport Transformation</u> - 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.

➢ <u>Device Transformation</u> - to map the user defined "window of interest" (in the virtual world) to the dimensions of the display area.
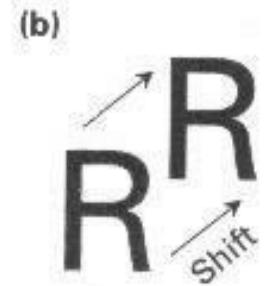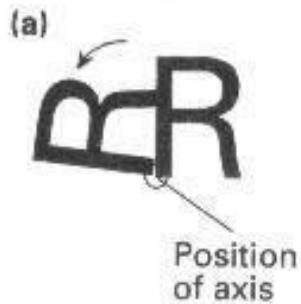
Transformation pipeline (4)

- ➢ 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
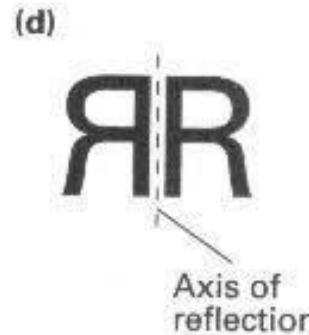
Rotation
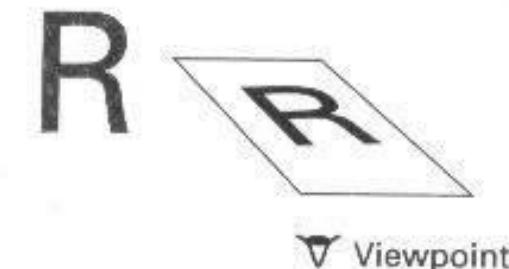
Translation 平移

Scaling

Reflection
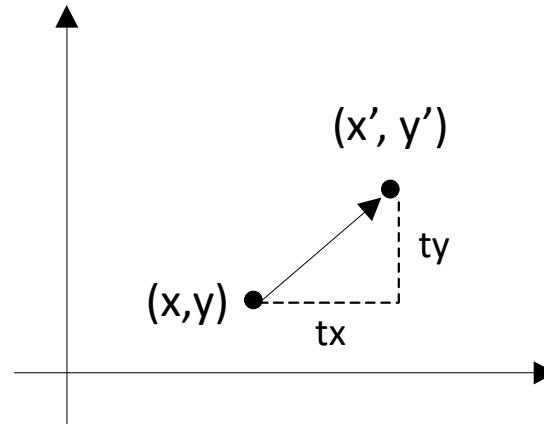
Shearing 切变

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)

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

where $\mathbf{P}(x, y)$ and $\mathbf{P'}(x', y')$ are the original and new positions, and $\mathbf{T}$ is the distance translated.
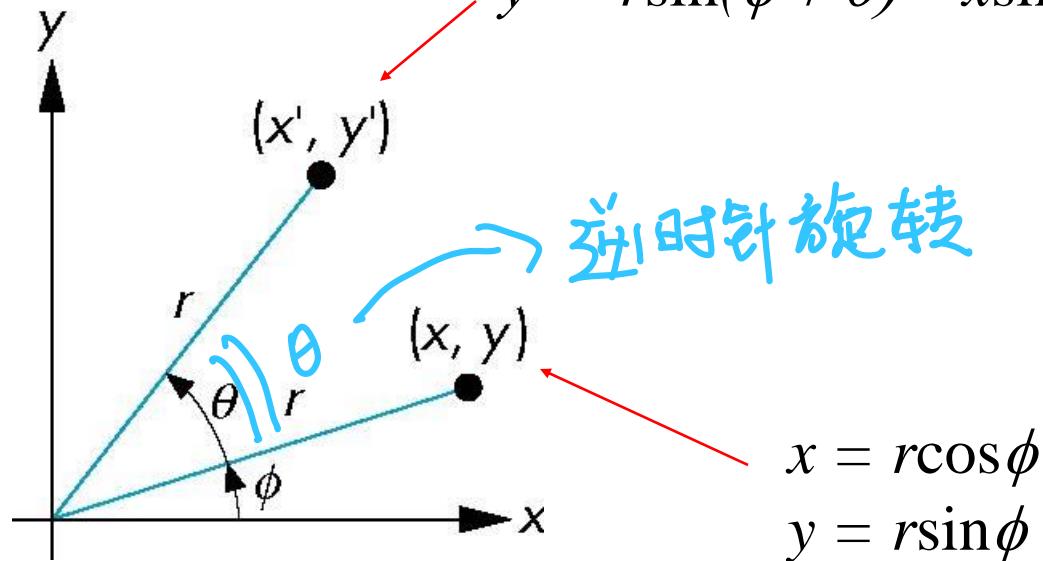
$$\mathbf{P'} = \mathbf{P} + \mathbf{T} \qquad \text{or} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

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.

$$x' = r\cos(\phi + \theta) = x\cos\theta - y\sin\theta$$
$$y' = r\sin(\phi + \theta) = x\sin\theta + y\cos\theta$$

逆时针旋转

$$x = r\cos\phi$$
$$y = r\sin\phi$$

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)

P' = R · P

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

so

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$
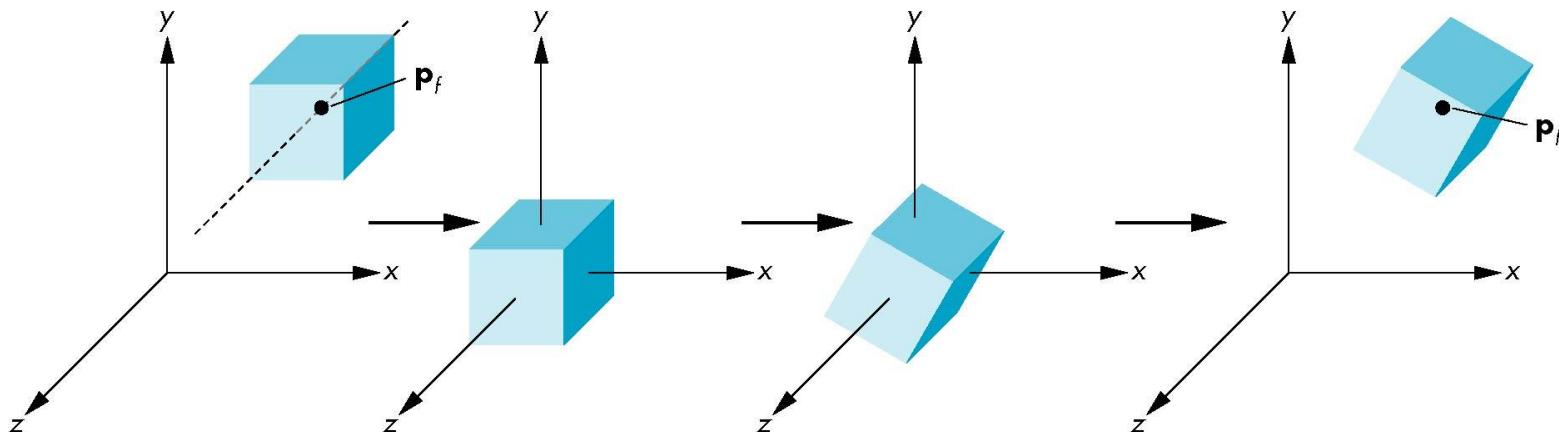
where θ is the rotation angle and ϕ is the angle between the *x*-axis and the line from the origin to (*x*, *y*).

Notice that the rotation point (or pivot point) is <u>the co-ordinate origin</u>.
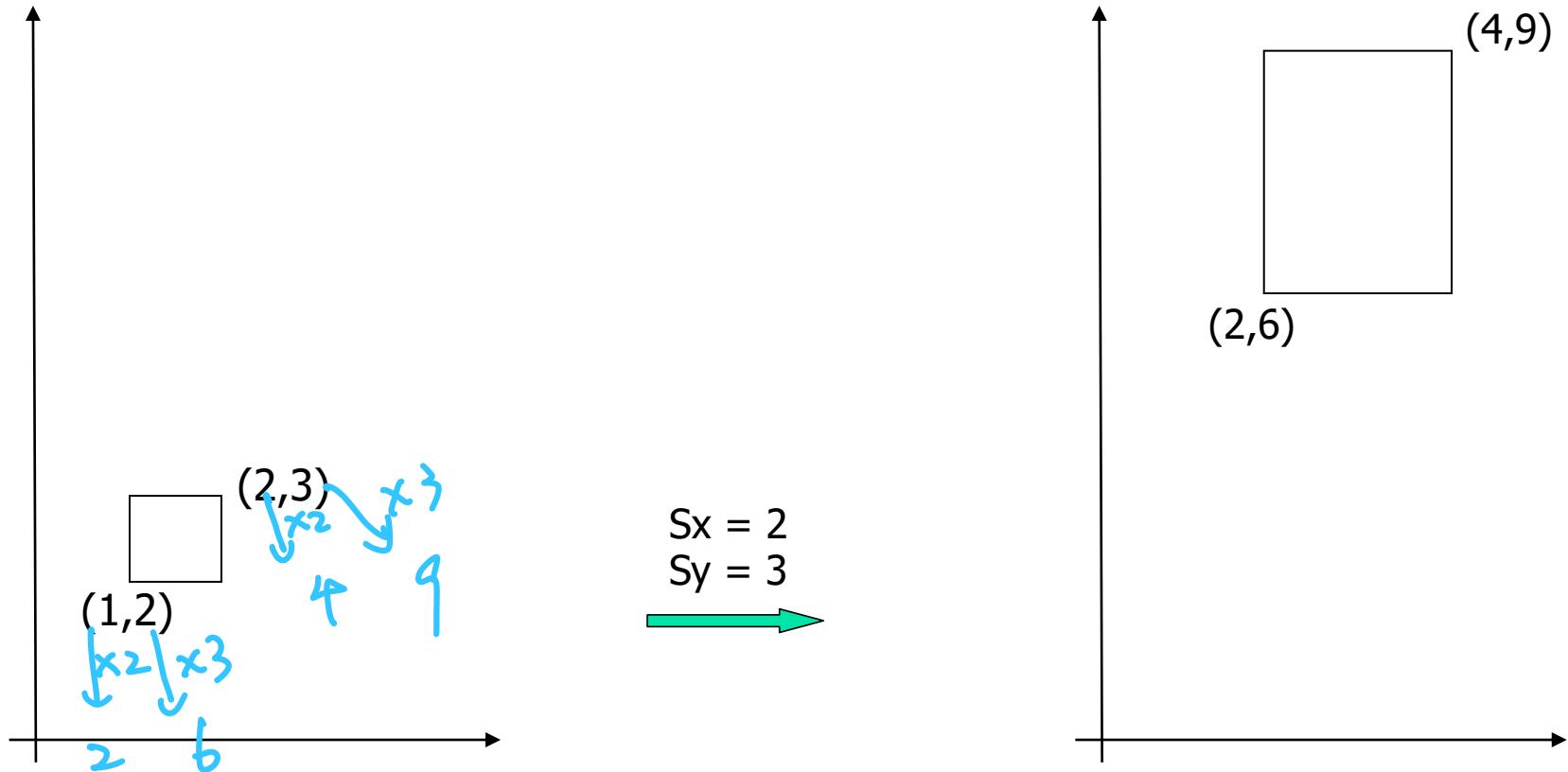
Rotation about a fixed point rather than the origin

绕 非原点旋转

➢ Move the fixed point to the origin

➢ Rotate the object

➢ Move the fixed point back to its initial position

➢ **M** = **T**(p_f) **R**(θ) **T**$(-p_f)$

2D scaling (1)

(4,9)

(2,6)

(2,3) ×3
×2
4   9

(1,2)
×2 ×3
2   6

Sx = 2
Sy = 3

When an object is scaled, <u>both the size and location</u> change.

2D scaling (2)

$$\begin{cases} x' = x \cdot s_x \\ y' = y \cdot s_y \end{cases}$$
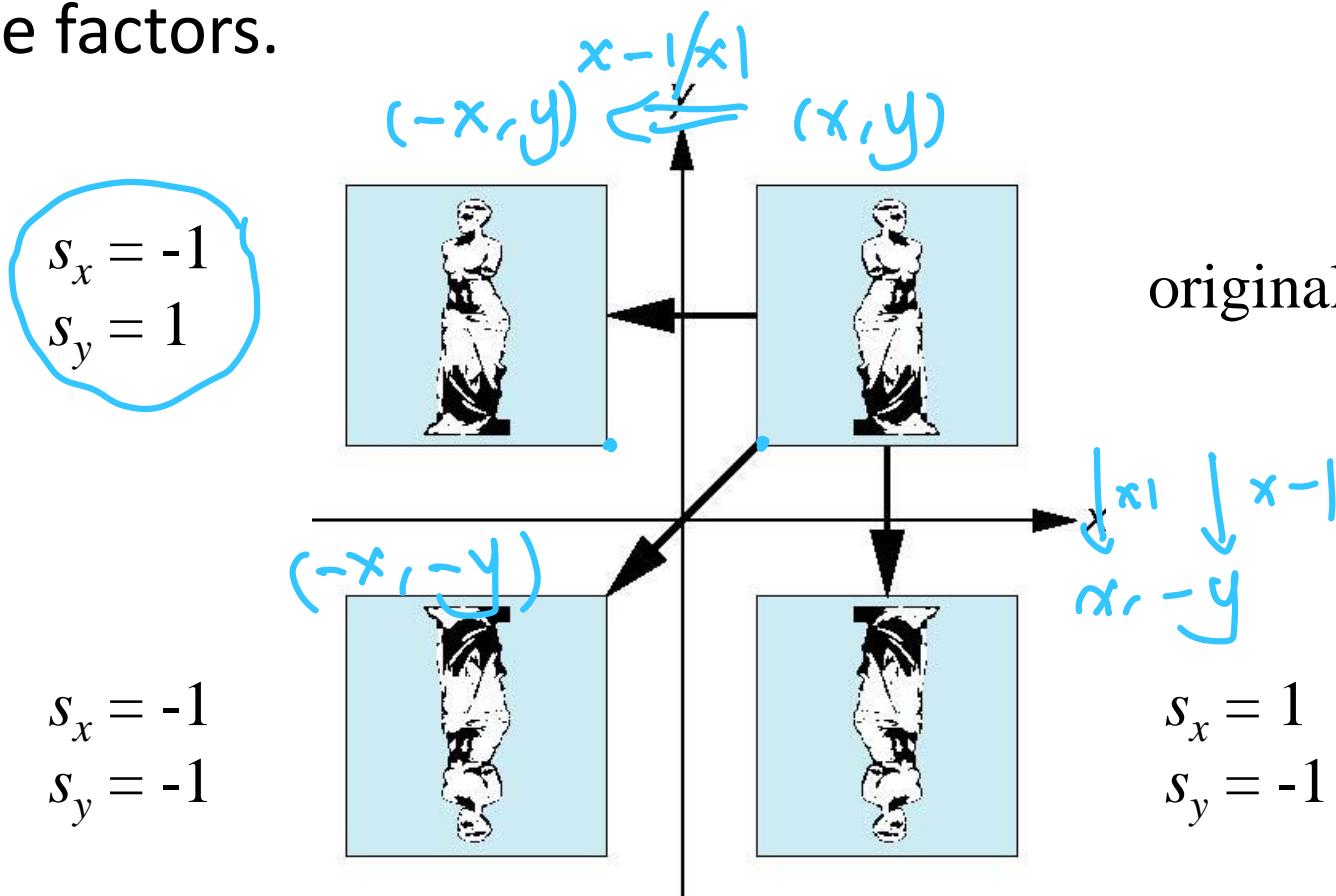
**P' = S · P**   so $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

where **P**, and **P'** are the original and new positions, and s_x and s_y are the scaling factors along the x- and y-axes.
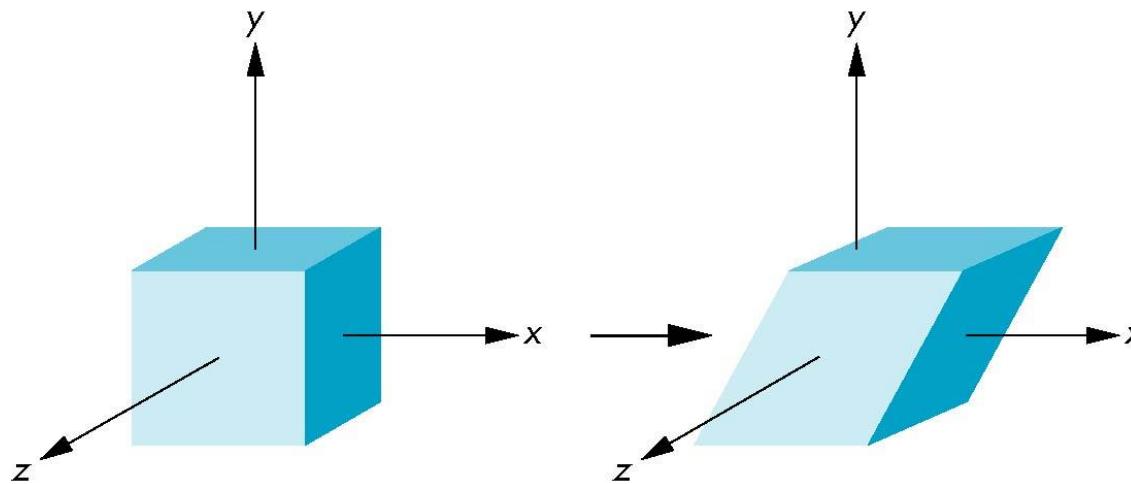
2D reflection

Special case of scaling - corresponding to negative scale factors.

$x - 1/x1$

$(-x, y) \Longleftarrow (x, y)$

$s_x = -1$
$s_y = 1$

original

$|x1 \quad |x-1$

$(-x, -y)$

$x, -y$

$s_x = -1$
$s_y = -1$

$s_x = 1$
$s_y = -1$

2D shearing (1)

Equivalent to pulling faces in opposite directions.

2D shearing (2)
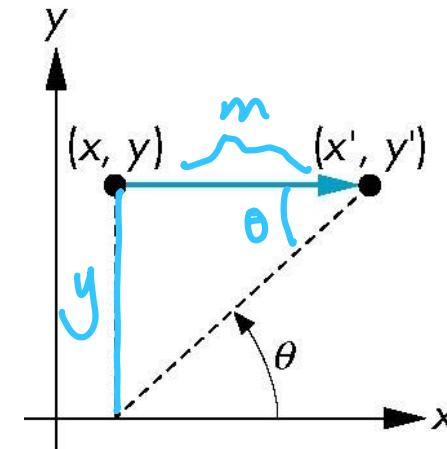
Consider simple shearing along the *x* axis.

$$x' = x + \underset{m}{\underline{y\cot\theta}}$$
$$y' = y$$
$$z' = z$$

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

$$\frac{y}{m} = \tan\theta \implies \frac{m}{y} = \cot\theta$$

2D homogeneous co-ordinates

复合变换

➢ 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)

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

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

2D homogeneous matrices (2)

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

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

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

(About z-axis)

3D co-ordinate axis rotations (2)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

(About y-axis)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

(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.
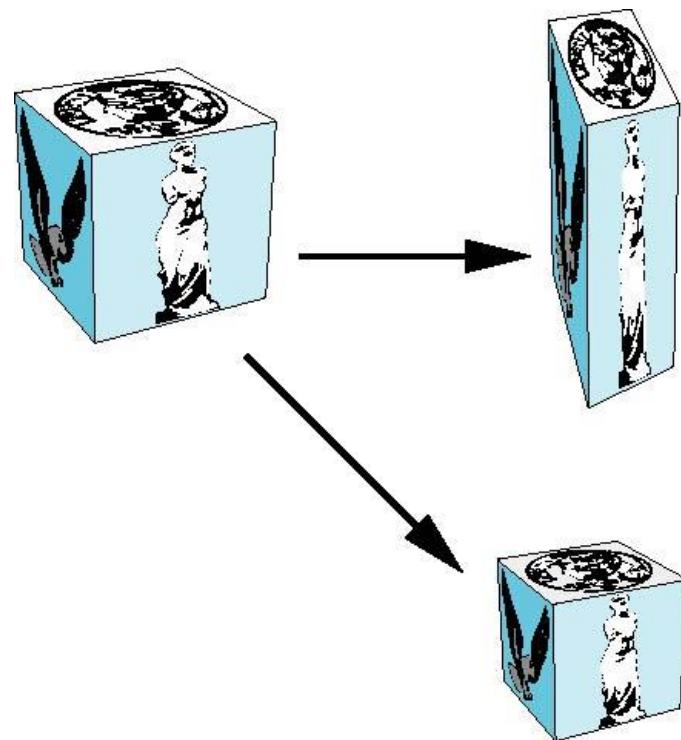
3D scaling

$$x' = s_x x$$
$$y' = s_y y$$
$$z' = s_z z$$

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

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·M2·M1 = (M3·M2)·M1 = M3·(M2·M1)

➤ Transformation products are not always commutative

A·B ≠ B·A

Instancing
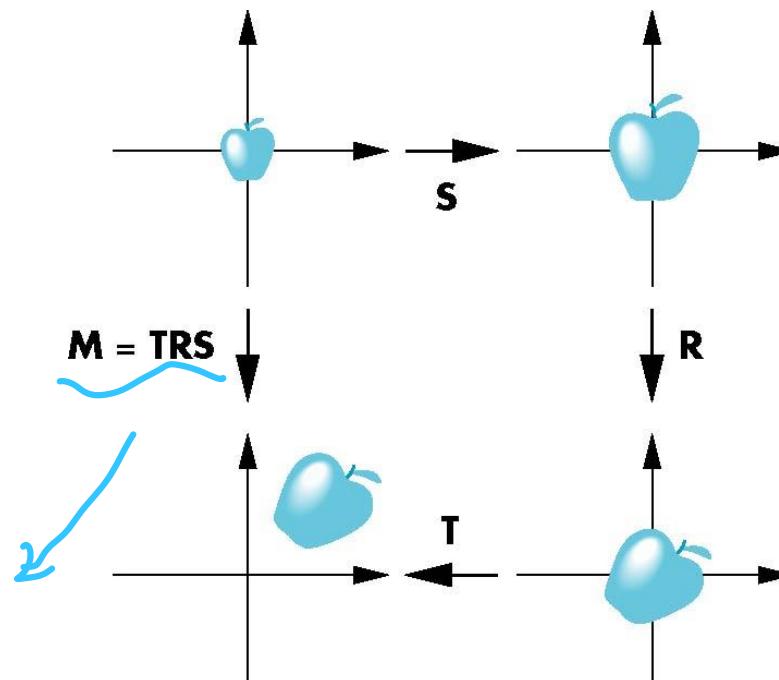
➢ In modelling, we often start with a simple object centred at the origin, oriented with an axis, and of a standard size.

➢ We apply an *instance transformation* to its vertices to

- Scale
- Orient
- Locate

$$P_1 = S P_0$$

$$P_2 = R P_1$$

$$P_3 = T P_2$$

$$\Rightarrow P_3 = T R S P_0$$

S

M = TRS

R

T

OpenGL matrices

➤ In OpenGL, matrices are part of the state.

➤ Multiple types

→ 模型变换

- Model-View (**GL_MODELVIEW**)
- Projection (**GL_PROJECTION**)
- Texture (**GL_TEXTURE**)
- Color (**GL_COLOR**)

➤ <u>Single</u> 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 postmutiplication. 左乘

➢ 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 postmutiplication

Load an identity matrix: $C \leftarrow I$
Load an arbitrary matrix: $C \leftarrow M$

任意的

Load a translation matrix: $C \leftarrow T$
Load a rotation matrix: $C \leftarrow R$
Load a scaling matrix: $C \leftarrow S$

Postmultiply by an arbitrary matrix: $C \leftarrow CM$
Postmultiply by a translation matrix: $C \leftarrow CT$
Postmultiply by a rotation matrix: $C \leftarrow CR$
Postmultiply by a scaling matrix: $C \leftarrow CS$

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. $C \leftarrow CM$

Matrix stacks

➢ CTM is not just one matrix but a <u>matrix stack</u> with the "current" at top.

➢ In many situations we want to <u>save transformation matrices</u> 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**

查询

➢ For matrices, we use as

- **double m[16];**
- **glGetFloatv(GL_MODELVIEW, m);** 获得矩阵数据

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** 确定做啥 Mode
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
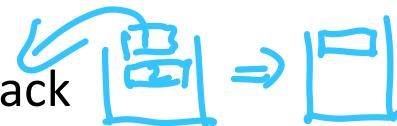
➢ **glPushMatrix**
Copy the top matrix in the stack and store copy in the second stack position

➢ **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) R对着 PC(1,2,3)放缩B Z轴 30度·

必须加完抛回去

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.0, 2.0, 3.0);
glRotatef(30.0, 0.0, 0.0, 1.0);  围绕Z轴 30度
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