Algorithms and Data Structures

COMP261
3D Rendering 1

Yi Mei

yi.mei@ecs.vuw.ac.nz

Outline

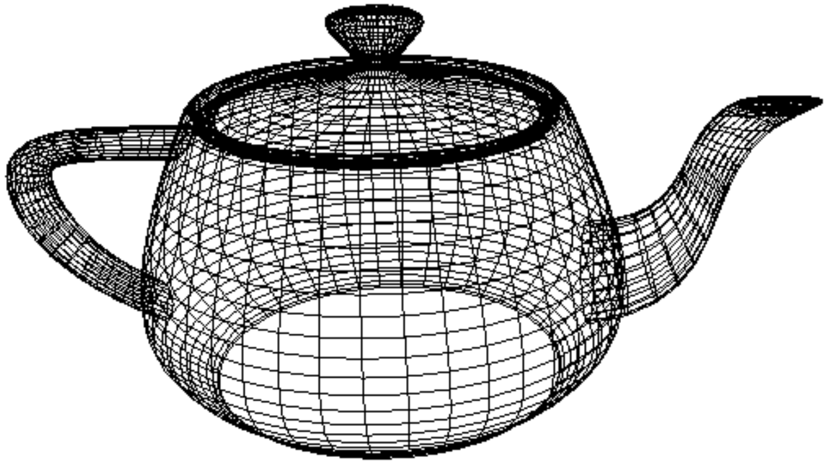
- 3D Rendering
- Polygons as rendering unit
- 3D coordinate system to represent polygons
- Change coordinates when changing viewing perspective

3D Graphics

- Movies, Animations, Games, …

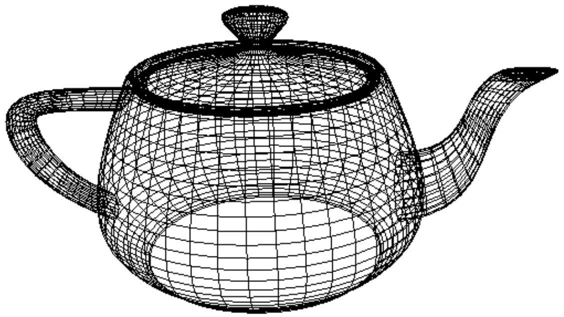
3D Graphics

- **Rendering**: display 3D objects on a 2D screen
- Need to consider
  - Shape
  - Surface properties
  - Material/mass
  - Movement/animation
  - Light sources
  - …

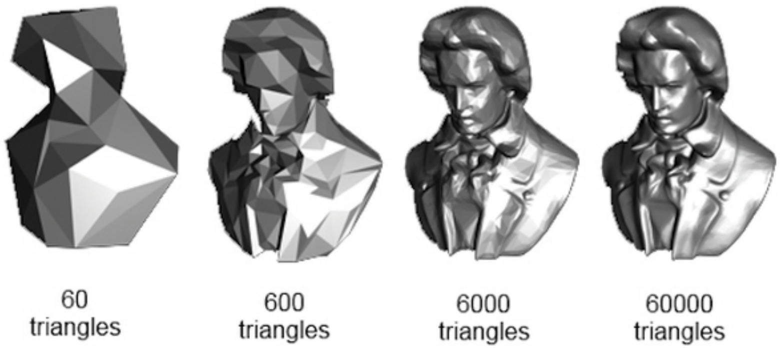


3D Rendering

- We only draw the surfaces that we can see
- We don't care about the hidden parts
- Key question
  - Identify the visible surfaces
  - Render the surfaces in computer

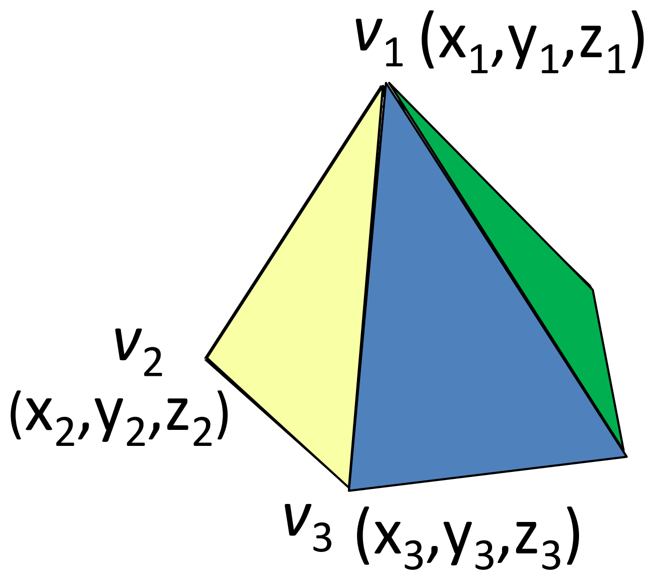


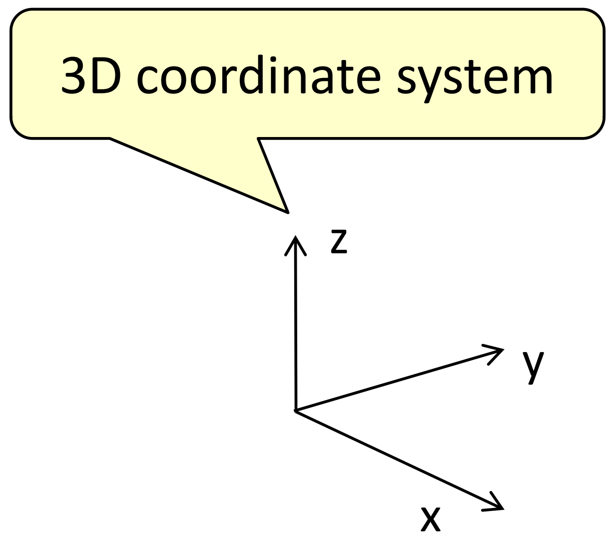
- **Rendering unit**
  - Pixels? - Elementary unit, but too slow
  - Polygons to approximate surfaces
    - Triangle
    - Square
    - …
  - **Use triangles to approximate surfaces**



Representation of Polygon

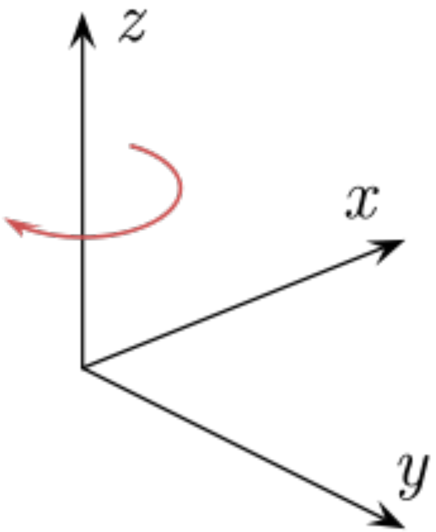
- A polygon is a list of vertices
- Each vertices is a point in a 3D space
  - Location: (x, y, z) coordinate



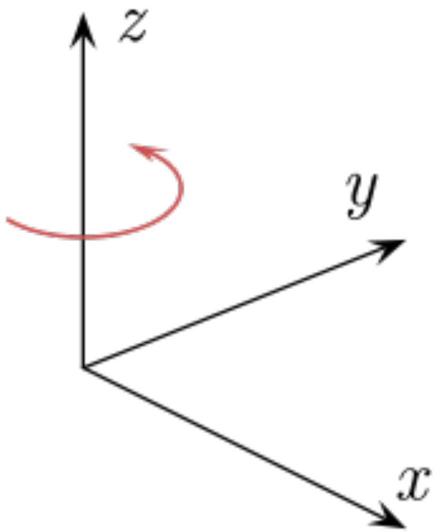


3D Coordinate System

- A standard 3D coordinate system based on **right-hand rule**
  - Order x, y and z
  - Use right hand, thumb point to the **z** axis
  - Curl of other fingers from x to y (anti-clockwise viewing from z axis)



Left-hand rule

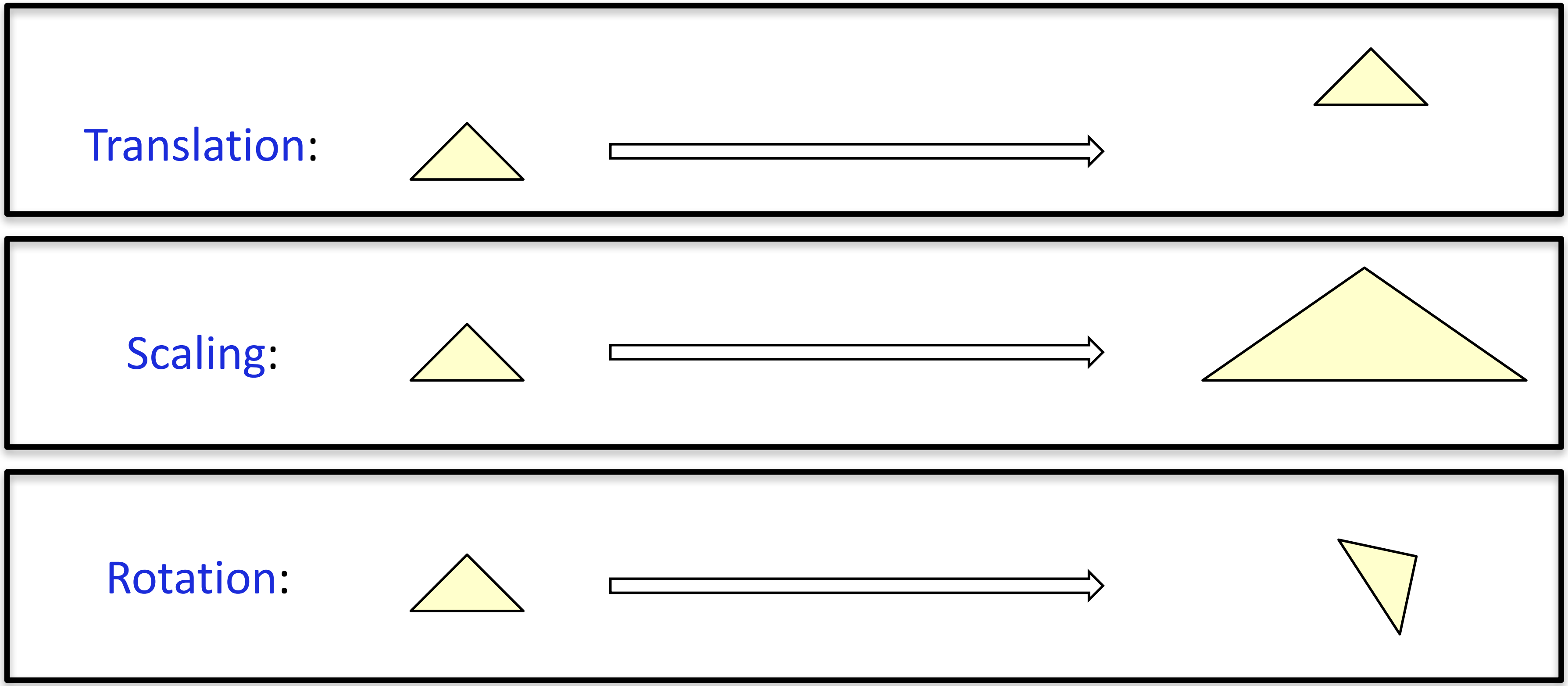


Right-hand rule

Right-hand rule

Change Viewer's Perspective

- Change polygon representation when changing viewer's perspective: change coordinate of each vertex
  - Translation, scaling, rotation



- Do them by **linear algebra**

Linear Algebra: basics

- *n*-dimensional Vector: $\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$
- *m*-row-*n*-column Matrix $\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$
- *n*-dim vector can be seen as *n*-row-1-column matrix
- Matrix Addition (same rows and columns)
  - Add corresponding element

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & \cdots & a_{1n}+b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

Linear Algebra: basics

- Matrix Multiplication
  - Condition: #columns of matrix 1 = #rows of matrix 2
    - Matrix 1 = m-row-n-col, Matrix 2 = n-row-k-col
  - To calculate element (i, j) of the multiplied matrix, first get row i of matrix 1 and column j of matrix 2
    - $\vec{a}_i^{\langle row \rangle} = (a_{i1}, \ldots, a_{in}),\ \vec{b}_j^{\langle col \rangle} = \left(b_{1j}, \ldots, b_{nj}\right)^T$
  - Do the inner product of the two same-dim vectors
    - $\vec{a}_i^{\langle row \rangle} \cdot \vec{b}_j^{\langle col \rangle} = a_{i1}b_{1j} + \cdots + a_{in}b_{nj}$
  - $(m\text{-row-}n\text{-col}) \times (n\text{-row-}k\text{-col}) \rightarrow (m\text{-row-}k\text{-col})$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \times \begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{bmatrix} = \begin{bmatrix} \vec{a}_1^{\langle row \rangle} \cdot \vec{b}_1^{\langle col \rangle} & \cdots & \vec{a}_1^{\langle row \rangle} \cdot \vec{b}_k^{\langle col \rangle} \\ \vdots & \ddots & \vdots \\ \vec{a}_m^{\langle row \rangle} \cdot \vec{b}_1^{\langle col \rangle} & \cdots & \vec{a}_m^{\langle row \rangle} \cdot \vec{b}_k^{\langle col \rangle} \end{bmatrix}$$

Change Vertex Coordinates

- Translation

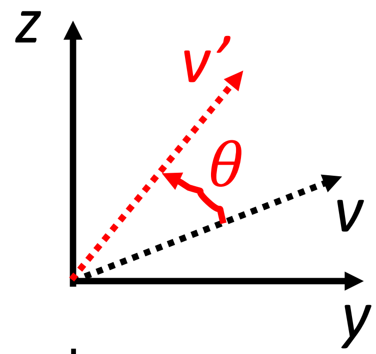
Original vertex coordinates

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \Rightarrow \begin{bmatrix} x + \Delta x \\ y + \Delta y \\ z + \Delta z \end{bmatrix}$$

New vertex coordinates

- Scaling

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \begin{bmatrix} x \cdot s_x \\ y \cdot s_y \\ z \cdot s_z \end{bmatrix}$$



- Rotation
  - In the *y-z* plane, rotate θ angle anti-clockwise, keep x fixed

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \begin{bmatrix} 1x + 0y + 0z \\ 0x + \cos\theta \cdot y - \sin\theta \cdot z \\ 0x + \sin\theta \cdot y + \cos\theta \cdot z \end{bmatrix}$$

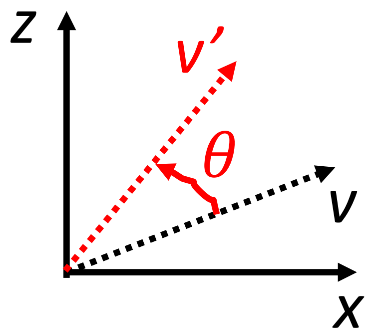
Change Vertex Coordinates

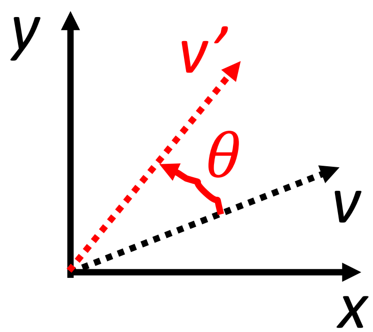
- Rotation
  - In the *x-z* plane, keeping *y* fixed

$$\begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \begin{bmatrix} \cos\theta \cdot x - \sin\theta \cdot z \\ y \\ \sin\theta \cdot x + \cos\theta \cdot z \end{bmatrix}$$

  - In the *x-y* plane, keeping *z* fixed

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \begin{bmatrix} \cos\theta \cdot x - \sin\theta \cdot y \\ \sin\theta \cdot x + \cos\theta \cdot y \\ z \end{bmatrix}$$





Unified Transformation Operator

- Transformation operators
  - **Translation**: add a vector
  - **Scaling**: multiply a matrix on the left
  - **Rotation**: multiply a matrix on the left
- A unified transformation operator can simplify the process
  - A single method/function for all the transformation scenarios
  - Can we rewrite translation as multiply a matrix on the left?
    - Adding a vector -> multiplying a matrix on the left

- Need an extra dimension

$$\begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x + \Delta x \\ y + \Delta y \\ z + \Delta z \\ 1 \end{bmatrix}$$

Unified Transformation Operator

- Also add an extra dimension for other transformations

Scaling:

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

Rotation:

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

- 4x4 transformation matrix:
  - **Translation**: 1s in the diagonal, shifts are on the last column, all others are 0.
  - **Scaling**: top-left corner is original 3x3 matrix, last diagonal element is 1, all others are 0.
  - **Rotation**: top-left corner is the original 3x3 matrix, last diagonal element is 1, all others are 0.

Unified Transformation Operator

Input: original point = {x, y, z}, 4x4 transformation matrix T
Output: new point {x', y', z'} after transformation

```
Initialise newpoint[4] = {0, 0, 0, 1};
//multiply (x, y, z, 1) with the 4x4 transformation matrix T on the left
for (row = 0 to 3) {
    for (col = 0 to 3) {
        newpoint[row] += T[row][col]*point[col];
    }
}
// only keep the first 3 elements
newpoint = newpoint[0:2];
return newpoint;
```

Summary

- Use triangle polygon to render (good balance between accuracy and time complexity)
- 3D coordinate to represent polygons (right-hand system)
- Coordinate transformation using linear algebra
  - Translation
  - Scaling
  - Rotation