Algorithms and Data Structures

COMP261
3D Rendering 2

Yi Mei

yi.mei@ecs.vuw.ac.nz

Outline

- Composite transformation
- Draw objects as polygons
  - Find visible/invisible polygons
  - Shading

Composite Transformation

- So far we know how to do a single transformation (translation, scaling, rotation) using unified transformation operator, but what if we have multiple transformation together?
  - E.g. translation + scaling, translation + rotation
- Easily achieved by **composite matrix multiplication**

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

Translate-and-rotate matrix

Rotate

Translate

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

Translate-and-rotate matrix

Composite Transformation

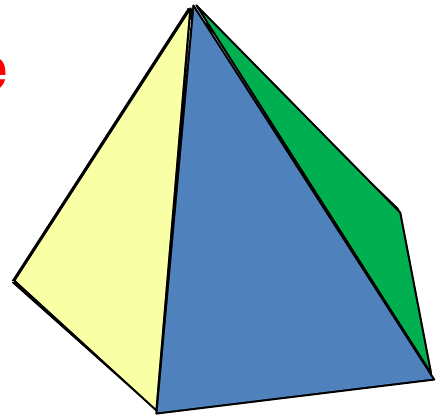
- Apply the transformation matrix from right to left
  - First transformation to the right most, last to the left most

```
// Calculate the composite transformation matrix
Input: a sequence of 4x4 transformation matrices (M_1, ..., M_k)
Output: the 4x4 composite transformation matrix CM
CM = M_2 * M_1;
for (i = 3:k) {
   CM = M_i * CM;
}
```

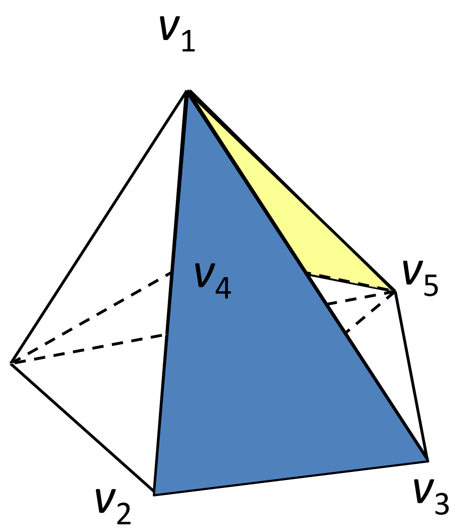
- Order of transformations
  - Rotation -> Scaling -> Translation

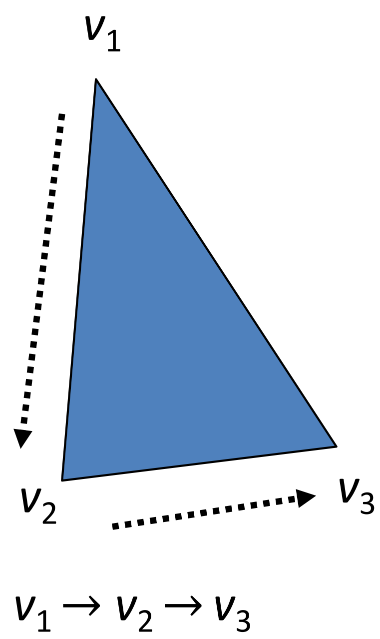
Draw Objects as Polygons

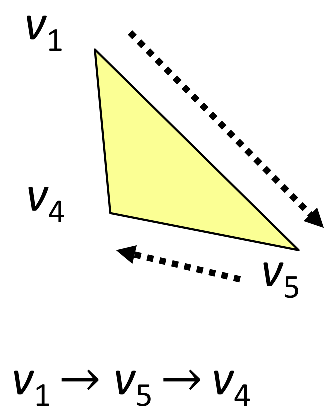
- We only draw the visible polygons (surfaces)
- Need to find out which polygons are visible/invisible
- Use 3D coordinate system + cross product



- Assumptions:
  - The viewer looks along z-axis
  - Order the polygon vertices as anti-clockwise when facing the viewer





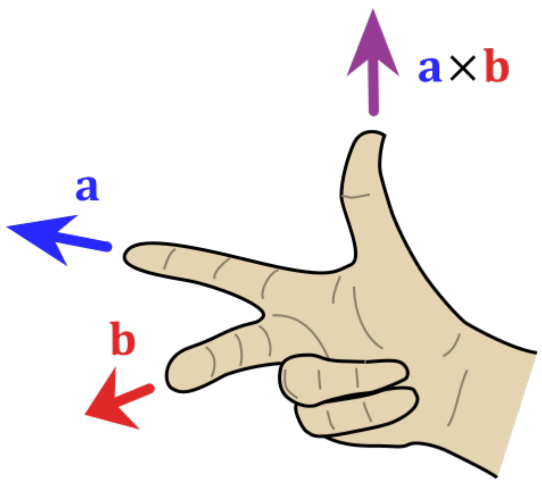


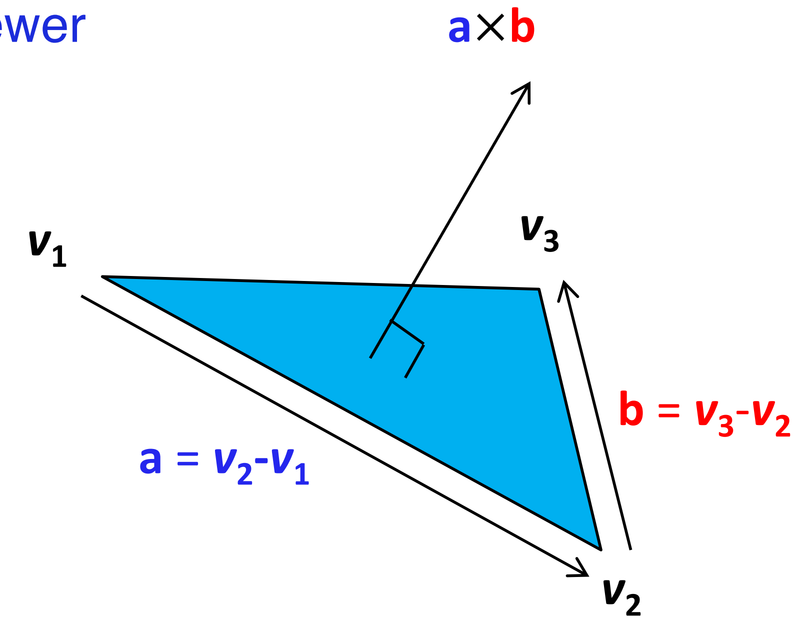
Cross Product

- **Cross product** is an operation between vectors. It returns another vector that is perpendicular with the input vectors.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - y_1 x_2 \end{bmatrix}$$

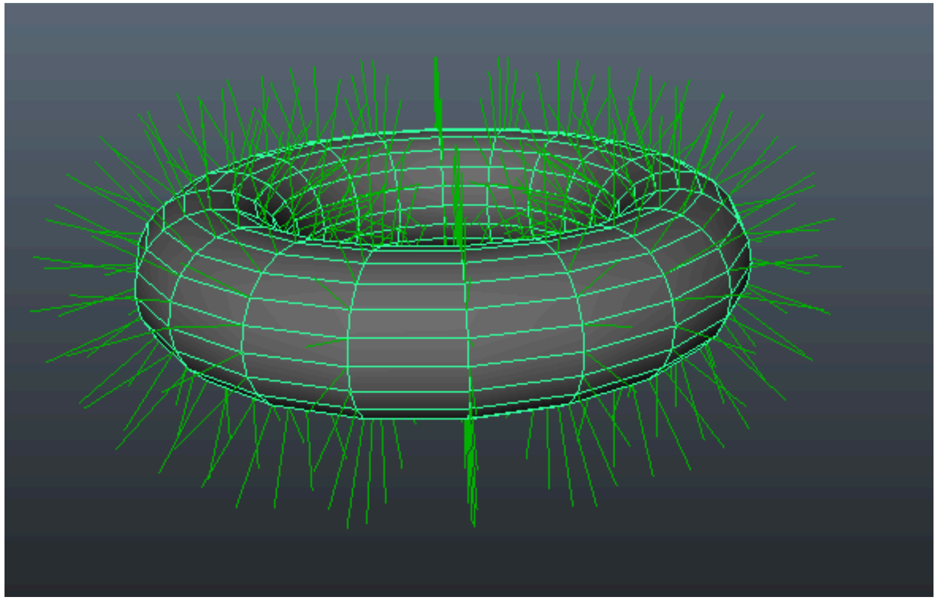
- The three vectors follow the right-hand rule.
- Use cross product to get which direction the polygon is facing
  - **v_1 -> v_2 -> v_3** is anti-clockwise
  - **(v_2- v_1) x (v_3- v_2)** is facing the viewer





Visible/Invisible Polygons

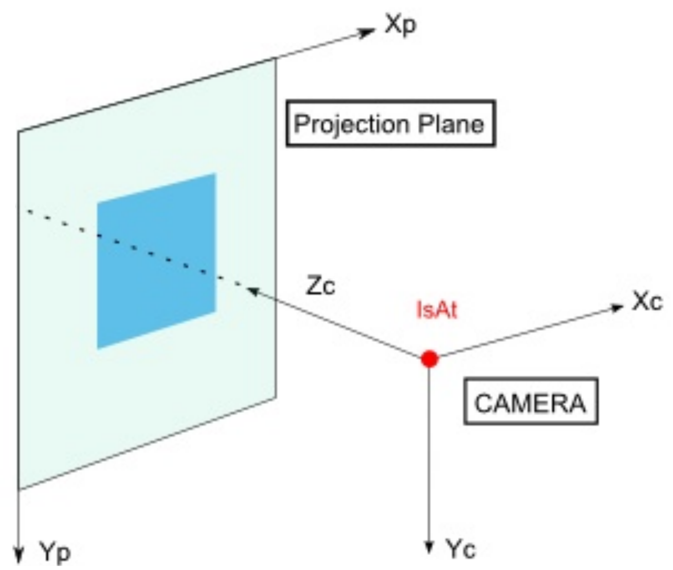
- The cross product $(v_2 - v_1) \times (v_3 - v_2)$ is called the normal of the polygon



- Whether an object is visible or not?
  - Direction of the normal of the polygon
  - Direction of the viewer

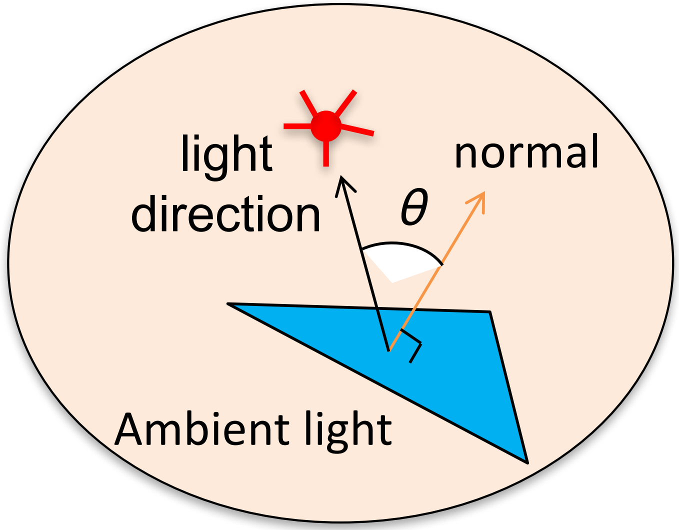
Visible/Invisible Polygons

- **Assumption**: viewer is viewing along the z-axis
- A polygon is visible to viewer, if its normal has negative z coordinate value
- A polygon is invisible to viewer, if its normal has positive z coordinate value



Shading

- Shading for the visible polygons is the light reflected from the surface. It depends on
  - Direction and color of light resources
  - Reflectance
  - Matte/Shiny surface
  - Color, texture of the surface
  - …



- A simple method:
  - Assume matte, uniform reflectance for red, green, blue
  - Assume some ambient light: intensity (0, 1]
    - ambientlight = ambient light intensity * reflectance
  - Assume an indicent light source: intensity (0, 1], and its direction
  - Diffuse reflection depends on incident light source direction
    - incidentlight = incident light intensity * reflectance * cos(θ)
  - light = ambientlight + incidentlight

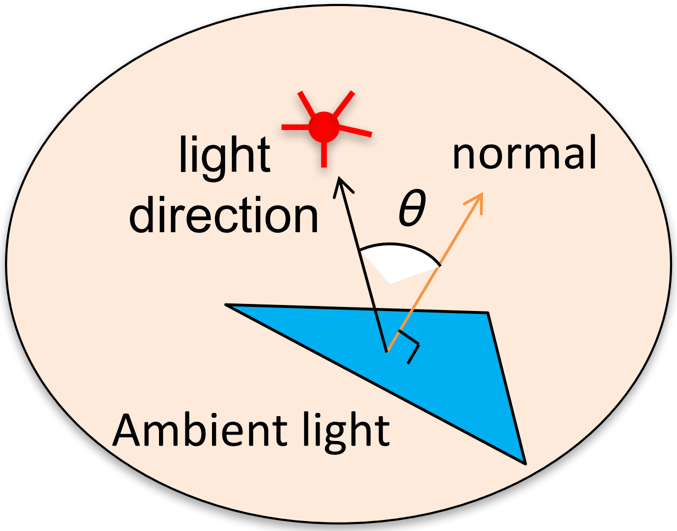
Shading

- Calculating $\cos(\theta)$
- Based on the laws of cosines:

Dot product

- $$\cos(\theta) = \frac{lightDirection \cdot normal}{|lightDirection| \times |normal|}$$

Length of the vector



Shading Computation

Input:
- three vertices ordered anti-clockwise when facing the viewer: $\boldsymbol{v}_i, i = 1,2,3$
- Ambient light intensity $\boldsymbol{AL} = (AL.r, AL.g, AL.b)$, each color is within the range (0, 1]
- Incident light intensity $\boldsymbol{IL} = (IL.r, IL.g, IL.b)$, each color is within the range (0, 1]
- Incident light direction $\boldsymbol{D} = (D.x, D.y, D.z)$
- Reflectance $\boldsymbol{R} = (R.r, R.g, R.b)$, each color within the range [0, 255]

Output: the shading color $(S.r, S.g, S.b)$

// calculate normal

$\boldsymbol{a} = \boldsymbol{v}_2 - \boldsymbol{v}_1, \boldsymbol{b} = \boldsymbol{v}_3 - \boldsymbol{v}_2$

$\boldsymbol{n} = \boldsymbol{a} \times \boldsymbol{b}$ ← Cross product

// calculate $\cos(\theta)$

$\cos(\theta) = \frac{\boldsymbol{n} \cdot \boldsymbol{D}}{|\boldsymbol{n}| \times |\boldsymbol{D}|}$ ← Dot product

// calculate the shading

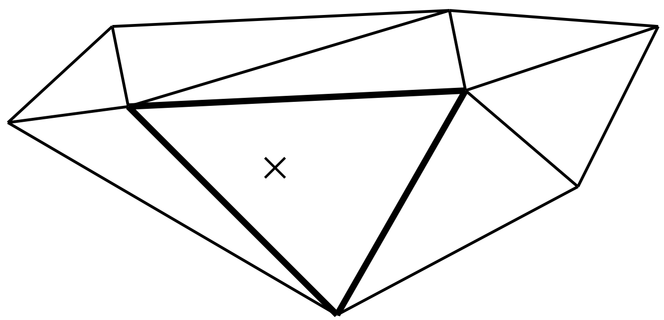
for (c in {r, g, b}) {

$S.c = AL.c \times R.c + IL.c \times R.c \times \cos(\theta);$

}

Advanced Shading

- Light reflected from a polygon:
  - could be uniform (if assume each polygon is a flat, uniform surface)
    - $\Rightarrow$ compute once for whole polygon
  - could vary across surface (if polygons approximate a curved surface)



  - Can interpolate from the vertices:
    - use "vertex normals" (average of surfaces at vertex)
    - either interpolate shading from vertices
    - or interpolate normals from vertices and compute shading

Summary

- Composite transformation
  - Calculate from right-hand side, first operation matrix on the right most
  - Rotation -> Scaling –> Translation
- Visible/Invisible polygons
  - Calculate normal using cross product
  - Assume viewer's direction (z-axis)
  - Check the z-value of normal (positive -> visible, negative -> invisible)
- Shading
  - Calculate shading color based on
    - Ambient light intensity
    - Incident light intensity and direction
    - Reflectance