


Tutorial 3

CS3241 Computer Graphics (AY23/24)

September 12, 2023

Wong Pei Xian

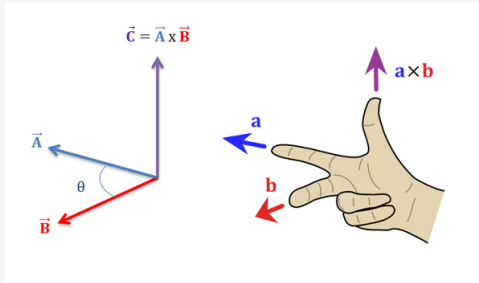
 e0389023@u.nus.edu

Question 1a

pollev.com/peixian

Given three points A, B, and C in 3D space, write an expression for the **normal vector** of the plane that contains the three points.

Normal vector



$$\mathbf{a} := B - A$$

$$\mathbf{b} := C - A$$

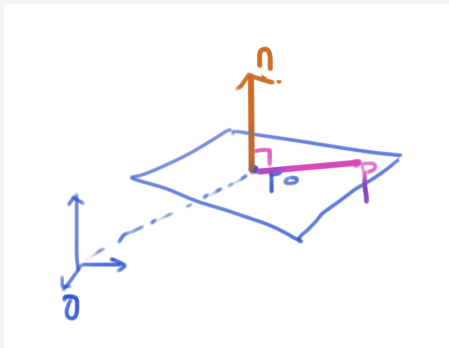
$$\mathbf{n} := \mathbf{a} \times \mathbf{b}$$

$$= (B - A) \times (C - A)$$

Question 1b

Given a point $R = [r_x \ r_y \ r_z]^T$ on a plane Π and a normal vector $n = [n_x \ n_y \ n_z]^T$ of the plane, write the implicit-form equation of the plane in the form $ax + by + cz + d = 0$.

Implicit form



$$n \cdot ([x \ y \ z]^T - R) = 0 \quad (1)$$

$$n_x x + n_y y + n_z z - n \cdot R = 0 \quad (2)$$

Question 1c

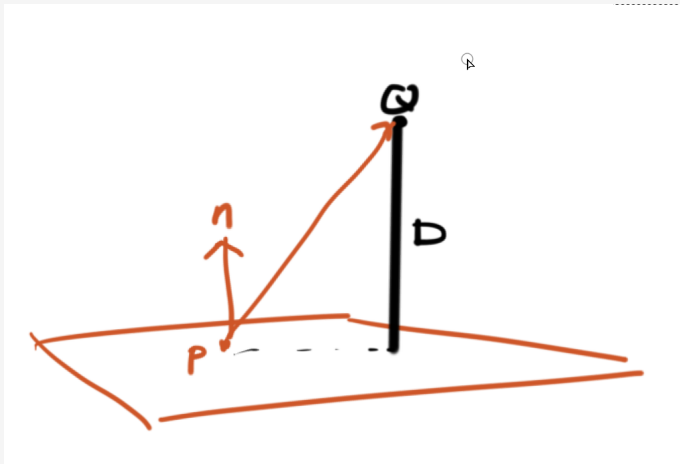
What is the perpendicular distance of the point Q from the plane
 $ax + by + cz + d = 0$?

Perpendicular distance of point to plane

Let $Q = [q_x \ q_y \ q_z]^T$. Then the distance D from point to plane is

$$D = \left| \frac{aq_x + bq_y + cq_z + d}{\sqrt{a^2 + b^2 + c^2}} \right| \quad (3)$$

Proof?



Outline of proof

$$\begin{aligned} D &= \|PQ\| \cos \theta \\ &= \|PQ\| \text{norm}(n) \cdot \text{norm}(PQ) \\ &= \|PQ\| \frac{n \cdot PQ}{\|n\| \|PQ\|} \\ &= \frac{n \cdot (OQ - OP)}{\|n\|} \\ &= \frac{n \cdot OQ - n \cdot OP}{\|n\|} \\ &= \frac{n_x x + n_y y + n_z z - (-d)}{\sqrt{a^2 + b^2 + c^2}} \end{aligned}$$

Question 2a

pollev.com/peixian

What does the homogenous coordinates $[6, 4, 2, 0.5]^T$ represent?

Question 2a

What does the homogenous coordinates $[6, 4, 2, 0.5]^T$ represent?

$$= \begin{bmatrix} x & y & z & w \\ x/w & y/w & z/w & 1 \end{bmatrix}$$

Question 2b

Why does OpenGL (and many other graphics APIs) use homogeneous coordinates to represent points?

Why homogenous coordinates?

1. Different representations for points and vectors.
2. 4×4 Matrix multiplication
3. Perspective projection with matrix mult. and *perspective division*.

Point vector distinguishing

For **vector**; $w = 0$ e.g. $[x \ y \ z \ 0]$.

For **point**; $w = 1$ e.g. $[x \ y \ z \ 1]$.

In this week's lecture, you will learn that to transform **normal vectors** instead of points, we use the matrix

$$M_n = (M_t^{-1})^T$$

where M_t is the upper left 3×3 submatrix.

Question 3

pollev.com/peixian

Which of the followings is the matrix that rotates objects about the fixed 3D point $[2 \ 3 \ 4]^T$, where the rotation axis is parallel to and in the same direction as the x-axis, and the rotation angle is θ ?

- A. $\mathbf{T}(-2, -3, -4) \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T}(2, 3, 4)$
- B. $\mathbf{T}(-2, -3, -4) \cdot \mathbf{R}_x(-\theta) \cdot \mathbf{T}(2, 3, 4)$
- C. $\mathbf{T}(2, 3, 4) \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T}(-2, -3, -4)$
- D. $\mathbf{T}(2, 3, 4) \cdot \mathbf{R}_x(-\theta) \cdot \mathbf{T}(-2, -3, -4)$
- E. $\mathbf{T}(-2, -3, -4) \cdot \mathbf{R}_x(\theta)$

Things to note about transformation

Order of Transformations

- Conceptually, the transformation on the right is applied first

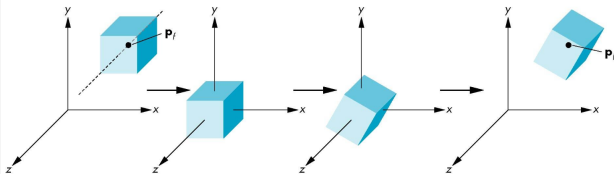
$$\mathbf{p}' = \mathbf{ABCp} = \mathbf{A(B(Cp))}$$

Things to note about transformation

Rotation About a Fixed Point Other than the Origin

1. Move fixed point to origin
2. Rotate
3. Move fixed point back

$$\mathbf{M} = \mathbf{T}(\mathbf{p}_f) \mathbf{R}(\theta) \mathbf{T}(-\mathbf{p}_f)$$



Ans: $T(2, 3, 4) \cdot R(\theta) \cdot T(-2, -3, -4)$.

Question 4

What is the inverse matrix of

$$M = \begin{bmatrix} s_1 r_{11} & s_1 r_{12} & s_1 r_{13} & t_1 \\ s_2 r_{21} & s_2 r_{22} & s_2 r_{23} & t_2 \\ s_3 r_{31} & s_3 r_{32} & s_3 r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 1: Decompose

You can tell there is a S , R , and T matrix multiplied together in some order to give M .

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, S = \begin{bmatrix} s_{11} & 0 & 0 & 0 \\ 0 & s_{22} & 0 & 0 \\ 0 & 0 & s_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T = \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 2: Order

pollev.com/peixian

$M = ?$

Step 2: Order

$$M = TSR.$$

$$\text{Thus } M^{-1} = R^{-1}S^{-1}T^{-1}.$$

How to get here

1. Trial and error

2. Hints

- In M , the last column $[t_1 \ t_2 \ t_3 \ 1]^T$ is not transformed.
- i.e. T applied **last**
- $M = TRS$ or $M = TSR$

Step 3: Substitute and Compute

$$\mathbf{M}^{-1} = \mathbf{R}^{-1} \cdot \mathbf{S}^{-1} \cdot \mathbf{T}^{-1} = \begin{bmatrix} r_{11} & r_{21} & r_{31} & 0 \\ r_{12} & r_{22} & r_{32} & 0 \\ r_{13} & r_{23} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1/s_1 & 0 & 0 & 0 \\ 0 & 1/s_2 & 0 & 0 \\ 0 & 0 & 1/s_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -t_1 \\ 0 & 1 & 0 & -t_2 \\ 0 & 0 & 1 & -t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}^{-1} = \begin{bmatrix} r_{11}/s_1 & r_{21}/s_2 & r_{31}/s_3 & -(r_{11}t_1/s_1) - (r_{21}t_2/s_2) - (r_{31}t_3/s_3) \\ r_{12}/s_1 & r_{22}/s_2 & r_{32}/s_3 & -(r_{12}t_1/s_1) - (r_{22}t_2/s_2) - (r_{32}t_3/s_3) \\ r_{13}/s_1 & r_{23}/s_2 & r_{33}/s_3 & -(r_{13}t_1/s_1) - (r_{23}t_2/s_2) - (r_{33}t_3/s_3) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Question 5

Let $M_1 = TR$.

Let $M_2 = RT$.

Is $M_1 = M_2$? Prove it.

Experiment with hand

1. With left hand, define 3 axes. This is **object space**.
2. Your left hand is now at **world space** origin
 - 2.1 Translate your hand by $T = [0, 0, 5]$.
 - 2.2 Rotate your hand by $R_x(90^\circ)$.
 - 2.3 Note where is your hand now in **world space**.
3. Now reset your left hand to the **world space** origin
 - 3.1 Rotate your hand by $R_x(90^\circ)$.
 - 3.2 Translate your hand by $T = [0, 0, 5]$.
4. *Is your hand in the space world space position as before?*

Proof and counterexample

In general $TR \neq RT$.

i.e. $M_1 \neq M_2$. Counterexample:

$$T(1, 0, 0)R_z(\pi/2)P(1, 0, 0) \neq R_z(\pi/2)T(1, 0, 0)P(1, 0, 0)$$

$$P(1, 1, 0) \neq P(0, 2, 0)$$

(Note: $P(x, y, z)$ refers to the point (x, y, z) .)

Question 6

The computation of $\sin \theta$ and $\cos \theta$ is considered relatively slow for some real-time rendering applications.

However, when the angle θ is very small, we can make use of the small-angle approximation: $\sin \theta \approx \theta$, and $\cos \theta \approx 1 - \theta_{small}^2/2$, for better speed.

Write the 4x4 matrix for rotation about the z-axis by a very small rotation angle θ_{small} , which is given in radians.

Substitute in

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\cos \theta \approx 1 - \theta_{small}^2/2$$

$$\sin \theta \approx \theta_{small}$$

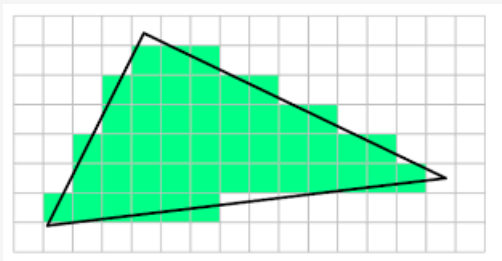
Substitute in

$$\mathbf{R}_Z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\approx \begin{bmatrix} 1 - \theta_{small}^2/2 & -\theta_{small} & 0 & 0 \\ \theta_{small} & 1 - \theta_{small}^2/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Question 7

In OpenGL, what is the purpose of transforming vertices to window space?

For the rasterizer!



Question 8

pollev.com/peixian

What are the matrices applied to each vertex $v_1 \dots v_6$?

Question 8

For the following application program code fragment, write down the transformations that are actually applied to each vertex v_i .

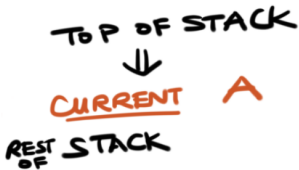
```
glMatrixMode( GL_MODELVIEW );
glLoadMatrixd( A );
glPushMatrix();
    glBegin( GL_POINTS ); glVertex3dv( v1 ); glEnd();
    glMultMatrixd( B );
    glPushMatrix();
        glMultMatrixd( C );
        glBegin( GL_POINTS ); glVertex3dv( v2 ); glEnd();
    glPopMatrix();
    glBegin( GL_POINTS ); glVertex3dv( v3 ); glEnd();
    glPushMatrix();
        glMultMatrixd( D );
        glPushMatrix();
            glMultMatrixd( E );
            glBegin( GL_POINTS ); glVertex3dv( v4 ); glEnd();
        glPopMatrix();
        glBegin( GL_POINTS ); glVertex3dv( v5 ); glEnd();
    glPopMatrix();
glPopMatrix();
glMultMatrixd( F );
glPushMatrix();
    glMultMatrixd( G );
glPopMatrix();
glBegin( GL_POINTS ); glVertex3dv( v6 ); glEnd();
```

OpenGL's matrix stacks API

- `glMatrixMode`: sets matrix being modified to either `ModelView` or `Projection`.
- `glLoadMatrix`/`glLoadIdentity`: clears entire matrix stack and replaces with loaded matrix/identity matrix I
- `glMultMatrix`: **postmultiplies** matrix to the current stack
 - Note that the stack grows from left to right as a result.
- `glPushMatrix`: creates a new "sub-stack" of matrices (see next slides for actual implementation)
- `glPopMatrix`: pops-off the previous "sub-stack" of matrices

Understanding OpenGL's matrix stacks

```
glLoadMatrixd( A );
```

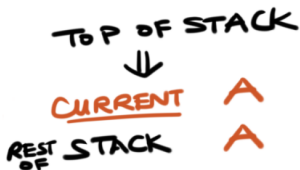


In fact, OpenGL works with a stack of stack of matrices. For clarity, the outer stack will be referred to as the stack, and the matrices being multiplied together will be referred to as the product.

On `glLoadMatrix`, the stack is initialized with one product (the single matrix *A* in this example).

Understanding OpenGL's matrix stacks

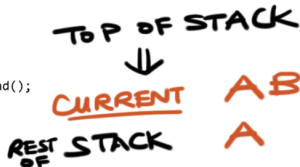
```
glLoadMatrixd( A );  
glPushMatrix();
```



When `glPushMatrix` is called, the top product of the stack is duplicated and added to the top of the stack.

Understanding OpenGL's matrix stacks

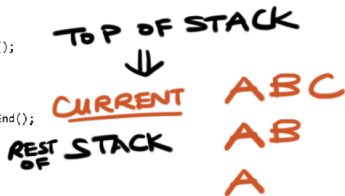
```
LoadMatrixd( A );  
PushMatrix();  
glBegin( GL_POINTS ); glVertex3dv( v1 ); glEnd();  
glMultMatrixd( B );
```



When `glMultMatrix` is called, the matrix *B* is post-multiplied to the current product at the top of the stack *A*.

Understanding OpenGL's matrix stacks

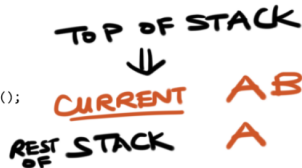
```
glLoadMatrixd( A );  
glPushMatrix();  
glBegin( GL_POINTS ); glVertex3dv( v1 ); glEnd();  
glMultMatrixd( B );  
glPushMatrix();  
glMultMatrixd( C );  
glBegin( GL_POINTS ); glVertex3dv( v2 ); glEnd();
```



This is similarly applied here.

Understanding OpenGL's matrix stacks

```
glLoadMatrixd( A );  
glPushMatrix();  
glBegin( GL_POINTS ); glVertex3dv( v1 ); glEnd();  
glMultMatrixd( B );  
glPushMatrix();  
glMultMatrixd( C );  
glBegin( GL_POINTS ); glVertex3dv( v2 ); glEnd();  
glPopMatrix();
```



When `glPopMatrix` is called, the top-most product in the stack is removed.

Note that

1. `glMultMatrix` **post-multiplies** the current matrix
2. `glPushMatrix` makes a copy of the topmost matrix of the stack, and makes it the current matrix
3. `glPopMatrix` deletes the top-most matrix and replaces it with the next one on the stack.
4. `glLoadIdentity` replaces the **current matrix** with I .

If too confusing, the high level understanding from here (slide 34) suffices.

Attendance taking

Thanks! Get the slides here after the tutorial.



<https://trxe.github.io/cs3241-notes>

Feel free to ask me questions about Assignment 1.