การแปลงในสามมิติ

418341: สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 6

ประมุข ขันเงิน

pramook@gmail.com

ในปริภูมิสามมิติ

- พิกัดในสามมิติแทนด้วยลำดับ (x,y,z)
- หรือด้วย (x,y,z,w) ถ้าอยู่ในรูป homogeneous coordinate
- homogeneous coordinate (x,y,z,w) หมายถึงพิกัด $(x/w,\ y/w,\ z/w)$ ในปริภูมิสามมิติ

ในปริภูมิสามมิติ (ต่อ)

- พิกัดในสามมิติสามารถเขียนได้อีกแบบหนึ่งในรูป matrix

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- มีความหมายเหมือนกับ homogeneous coordinate $(x,y,z,1)$

การแปลงในปริภูมิสามมิติ

- การแปลงแอฟไฟน์สามชนิดที่เรียนผ่านมา
  - การเลื่อนแกนขนาน (translation)
  - การย่อขยาย (scaling)
  - การหมุน (rotation)

  สามารถแทนได้ด้วย matrix 4 คูณ 4

การเลื่อนแกนขนาน

- สัญลักษณ์ $T_{a,b,c}$
- ส่งพิกัด (x,y,z) ไปยังพิกัด (x+a, y+b, z+c)
- มี matrix เป็น

$$\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

การย่อขยายขนาด

- สัญลักษณ์ $S_{a,b,c}$
- ส่งพิกัด (x,y,z) ไปยังพิกัด (ax, by, cz)
- นี่เป็นการย่อขยายรอบพิกัด (0,0,0) เนื่องจากพิกัด (0,0,0) ไม่เปลี่ยนแปลง
- มี matrix เป็น

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

การหมุน

- เวลาหมุนจะต้องบอกสองอย่าง
  - **แกน**ที่จะใช้หมุน
  - มุมที่จะใช้หมุน
- เวลาระบุแกนเราจะระบุด้วยเวกเตอร์ (a,b,c)
- **แกน**คือเส้นตรงที่เกิดจากจุดทั้งหมดที่อยู่ในรูป (at, bt, ct) เมื่อ t เป็นจำนวนจริงใดๆ
- แกนจะผ่านจุด (0,0,0) เสมอ
- เวลาทำการหมุน จุดที่อยู่บนแกนจะไม่เคลื่อนที่
- มุมที่จะใช้หมุนส่วนใหญ่จะใช้สัญลักษณ์ θ

การหมุนรอบแกน Z

- แกน Z คือเซตของพิกัดต่างๆ ที่อยู่ในรูป (0, 0, t)
- สามารถระบุได้ด้วยเวกเตอร์ (0,0,1)
- สัญลักษณ์ $R_{\theta,0,0,1}$
- ส่งพิกัด $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ ไปยังพิกัด $\begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ z \\ 1 \end{bmatrix}$

การหมุนรอบแกน Z (ต่อ)

- มี matrix เป็น

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

การหมุนรอบแกน X

- แกน X คือเซตของพิกัดต่างๆ ที่อยู่ในรูป (t, 0, 0)
- สามารถระบุได้ด้วยเวกเตอร์ (1,0,0)
- สัญลักษณ์ $R_{\theta,1,0,0}$
- ส่งพิกัด $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ ไปยังพิกัด $\begin{bmatrix} x \\ y\cos\theta - z\sin\theta \\ y\sin\theta + z\cos\theta \\ 1 \end{bmatrix}$

การหมุนรอบแกน **X** (ต่อ)

- มี matrix เป็น

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

การหมุนรอบแกน y

- แกน z คือเซตของพิกัดต่างๆ ที่อยู่ในรูป (0, t, 0)
- สามารถระบุได้ด้วยเวกเตอร์ (0,1,0)
- สัญลักษณ์ $R_{\theta,0,1,0}$
- ส่งพิกัด $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ ไปยังพิกัด $\begin{bmatrix} z\sin\theta + x\cos\theta \\ y \\ z\cos\theta - x\sin\theta \\ 1 \end{bmatrix}$

การหมุนรอบแกน y (ต่อ)

- มี matrix เป็น

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

การหมุนรอบแกนใดๆ

- สัญลักษณ์ $R_{\theta,a,b,c}$
- มี matrix เป็น

$$\begin{bmatrix} a^2(1-C)+C & ab(1-C)+cS & ac(1-C)+bS & 0 \\ ba(1-C)+cS & b^2(1-C)+C & bc(1-C)+aS & 0 \\ ca(1-C)+bS & cb(1-C)+aS & c^2(1-C)+C & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

เมื่อ $C = \cos\theta$ และ $S = \sin\theta$

การแปลง affine

- การแปลง affine คือการแปลงที่สามารถเขียนอยู่ในรูป matrix

$$\begin{bmatrix} a & d & i & l \\ b & e & j & m \\ c & f & k & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ระบบพิกัด

- ระบบพิกัดเป็นตัวกำหนดว่าพิกัดใดแทนจุดใด
- พิกัดและจุด?
  - พิกัดคือลำดับของเลขสามตัว: **(x,y,z)**
  - จุดคือจุดที่เราเห็นด้วยตา
- ระบบพิกัดในสามมิติมีส่วนประกอบอยู่สามส่วน
  - จุดออริจิน **o**: จุดนี้จะแทนด้วยพิกัด (0,0,0) ในระบบพิกัด
  - เวกเตอร์สามตัว **i, j,** และ **k** สำหรับกำหนดทิศทางแกน **x, y,** และ **z** ตามลำดับ

ระบบพิกัด (ต่อ)

- พิกัด **(x,y,z)** ในระบบพิกัดนี้จึงหมายถึงจุด

$$\mathbf{o} + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

กล่าวคือมันคือจุดที่อยู่ห่างจากจุด **o**
ไปตามแนวเวกเตอร์ **i** เป็นระยะ **x** เท่าของความยาวเวกเตอร์ **I**
ไปตามแนวเวกเตอร์ **j** เป็นระยะ **y** เท่าของความยาวเวกเตอร์ **j**
ไปตามแนวเวกเตอร์ **k** เป็นระยะ **z** เท่าของความยาวเวกเตอร์ **k**

ระบบพิกัด

- เขียนได้อีกแบบหนึ่งว่าพิกัด (x,y,z) หมายถึงจุด

$$\begin{bmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{o}\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

ระบบพิกัดกับการแปลง

- พิจารณาการแปลง affine

$$M = \begin{bmatrix} a & d & i & l \\ b & e & j & m \\ c & f & k & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ระบบพิกัดกับการแปลง (ต่อ)

- มันส่งพิกัด $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ ไปยังพิกัด $\begin{bmatrix} ax + dy + iz + l \\ bx + ey + jz + m \\ cx + fy + kz + n \\ 1 \end{bmatrix}$

ระบบพิกัดกับการแปลง (ต่อ)

- พูดได้อีกอย่างหนึ่งคือ M ส่งจุด

$$\begin{bmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{o}\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

ไปยังจุด

$$\begin{bmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{o}\end{bmatrix}\begin{bmatrix}ax+dy+iz+l\\ bx+ey+jz+m\\ cx+fy+kz+n\\ 1\end{bmatrix}=\begin{bmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{o}\end{bmatrix}\begin{bmatrix}a & d & i & l\\ b & e & j & m\\ c & f & k & n\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}$$

ระบบพิกัดกับการแปลง (ต่อ)

- แต่เราอาจมองได้อีกว่า

$$\begin{bmatrix}\mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{o}\end{bmatrix}\begin{bmatrix} a & d & i & l \\ b & e & j & m \\ c & f & k & n \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

มีค่าเท่ากับ

$$\begin{bmatrix} a\mathbf{i}+b\mathbf{j}+c\mathbf{k} & d\mathbf{i}+e\mathbf{j}+f\mathbf{k} & i\mathbf{i}+j\mathbf{j}+k\mathbf{k} & o+l\mathbf{i}+m\mathbf{j}+n\mathbf{k}\end{bmatrix}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

ระบบพิกัดกับการแปลง (ต่อ)

- ดังนั้นการแปลง M จึงสามารถมองได้ว่าเป็นการเปลี่ยนระบบพิกัดจากระบบพิกัดที่
  - มี **o** เป็นจุดออริจิน
  - มี **i** เป็นตัวกำหนดทิศทางแกน **x**
  - มี **j** เป็นตัวกำหนดทิศทางแกน **y**
  - มี **k** เป็นตัวกำหนดทิศทางแกน **z**

  เป็นระบบพิกัดที่
  - มี **o+li+mj+nk** เป็นจุดออริจิน
  - มี **ai+bj+ck** เป็นตัวกำหนดทิศทางแกน **x**
  - มี **di+ej+fk** เป็นตัวกำหนดทิศทางแกน **y**
  - มี **ii+jj+kk** เป็นตัวกำหนดทิศทางแกน **z**

ระบบพิกัดกับการแปลง (ต่อ)

- หรือกล่าวได้อีกอย่างหนึ่งคือ
  - จุดออริจินใหม่คือจุดที่มีพิกัด (l,m,n) ในระบบพิกัดเดิม
  - เวกเตอร์แกน x ใหม่ คือเวกเตอร์ (a,b,c) ในระบบพิกัดเดิม
  - เวกเตอร์แกน y ใหม่ คือเวกเตอร์ (d,e,f) ในระบบพิกัดเดิม
  - เวกเตอร์แกน z ใหม่ คือเวกเตอร์ (i,j,k) ในระบบพิกัดเดิม

ระบบพิกัดกับการแปลง (ต่อ)

$$\begin{bmatrix} a & d & i & l \\ b & e & j & m \\ c & f & k & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

แกน x ใหม่

ระบบพิกัดกับการแปลง (ต่อ)

$$\begin{bmatrix} a & d & i & l \\ b & e & j & m \\ c & f & k & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

แกน y ใหม่

ระบบพิกัดกับการแปลง (ต่อ)

$$\begin{bmatrix} a & d & i & l \\ b & e & j & m \\ c & f & k & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

แกน z ใหม่

ระบบพิกัดกับการแปลง (ต่อ)

$$\begin{bmatrix} a & d & i & l \\ b & e & j & m \\ c & f & k & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

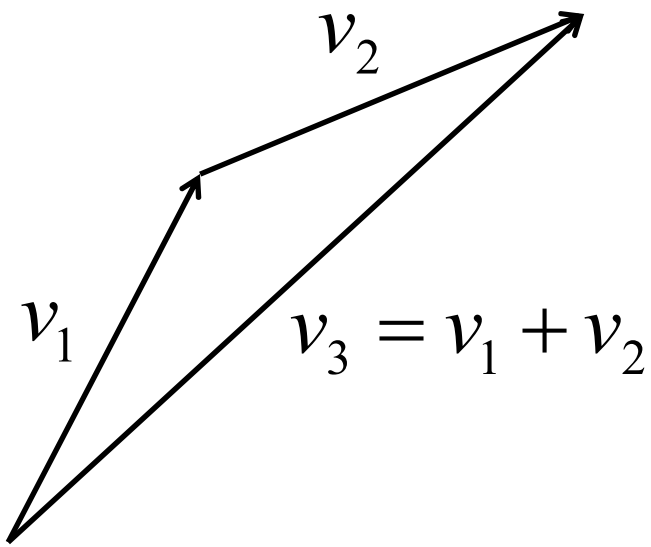
จุด origin ใหม่

Homogeneous Coordinate กับเวกเตอร์

- Homogeneous coodinate สามารถใช้แทนได้ทั้ง**จุด**และ**เวกเตอร์**
- ถ้า w ใน (x,y,z,w) เป็น 1 แสดงว่ามันแทนจุด
  - ถ้ามันไม่ใช่ 1 ให้เอา w ไปหาทุกตัวเพื่อทำให้มันเป็น 1 เสีย
- ถ้า w ใน (x,y,z,w) เป็น 0 แสดงว่ามันแทนเวกเตอร์ (ทิศทาง)

จุดกับเวกเตอร์

- จุด คือ “ตำแหน่ง”
- เวกเตอร์ คือ “ทิศทาง”
- คุณเอาเวกเตอร์สองเวกเตอร์มาบวกกันได้

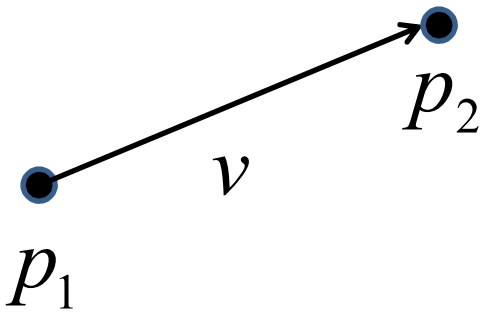


จุดกับเวกเตอร์ (ต่อ)

- แต่คุณเอาจุดสองจุดมาบวกกันไม่ได้

$$p_1 + p_2 = ???$$

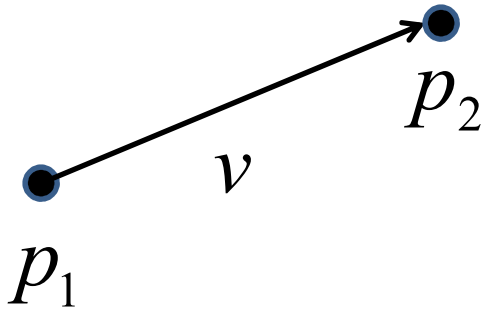
- แต่เอาจุดมาบวกกันเวกเตอร์ได้ จะได้จุดอีกจุดหนึ่ง



$$p_1 + v = p_2$$

จุดกับเวกเตอร์ (ต่อ)

- ในทำนองเดียวกัน คุณสามารถหาผลต่างของจุดได้ ซึ่งจะได้ผลลัพธ์ออกมาเป็นเวกเตอร์



$$p_1 + v = p_2$$

- ยกตัวอย่างเช่น $\begin{bmatrix} 3 \\ 4 \\ 5 \\ 1 \end{bmatrix} - \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 3 \\ 0 \end{bmatrix}$

จุดกับเวกเตอร์ (ต่อ)

- การแปลง **affine** มีผลต่อจุดและเวกเตอร์ต่างกัน

$$\begin{bmatrix} a & d & i & l \\ b & e & j & m \\ c & f & k & n \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} (ax+dy+iz)+l \\ (bx+ey+jz)+m \\ (cx+fy+kz)+n \\ 1 \end{bmatrix}$$

แต่

$$\begin{bmatrix} a & d & i & l \\ b & e & j & m \\ c & f & k & n \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} ax+dy+iz \\ bx+ey+jz \\ cx+fy+kz \\ 0 \end{bmatrix}$$

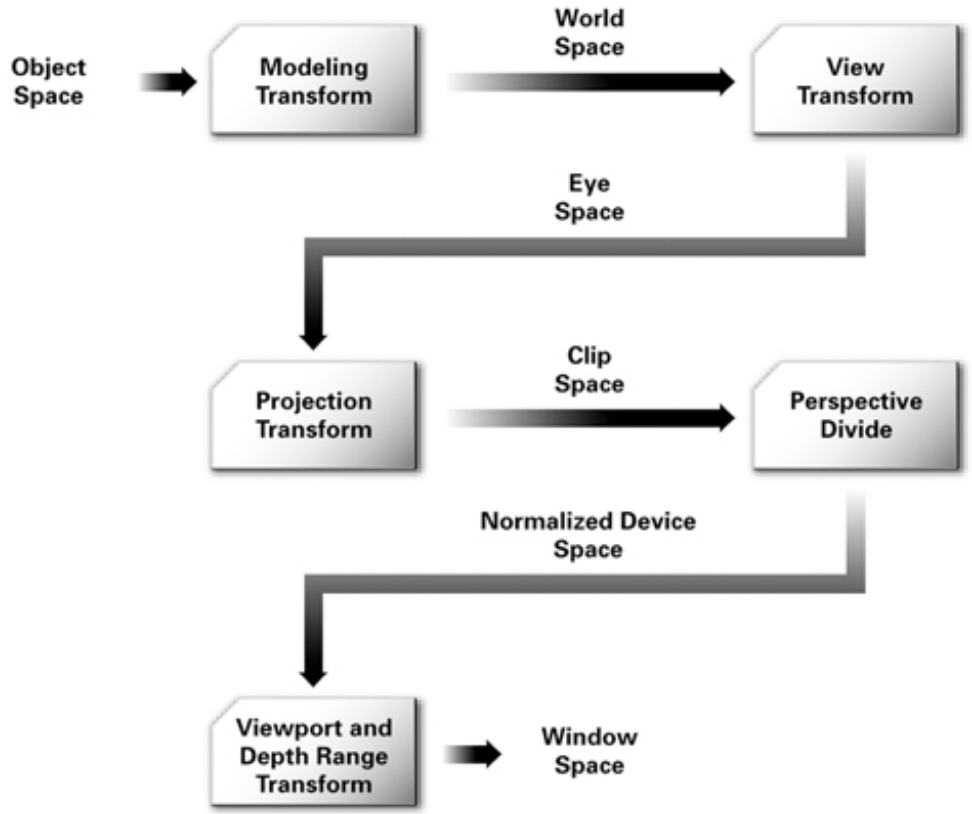
จุดกับเวกเตอร์ (ต่อ)

- ให้
  - M เป็นการแปลง affine
  - p เป็นจุด
  - ให้ v เป็นเวกเตอร์
- ได้ว่า
  - Mp เป็นจุด
  - Mv เป็นเวกเตอร์
- ในการแปลงจุดจะมีการเลื่อนแกนขนานติดมาด้วย
- แต่ในการแปลงเวกเตอร์ จะไม่มีการเลื่อนแกนขนานติดมาด้วย

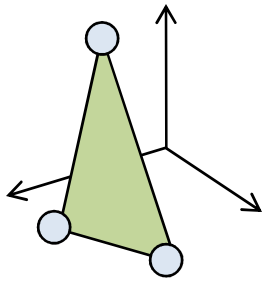
TRANSFORMATIONS IN THE GRAPHICS PIPELINE

OpenGL Vertex Transformations

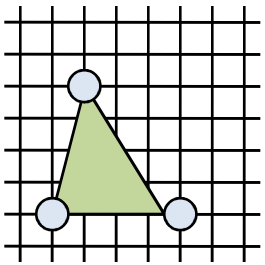
- ลำดับของ transform ที่ vertex หนึ่งจะต้องผ่านไปก่อนที่มันจะถูกเปลี่ยนเป็น fragment



OpenGL Vertex Transformation (ต่อ)



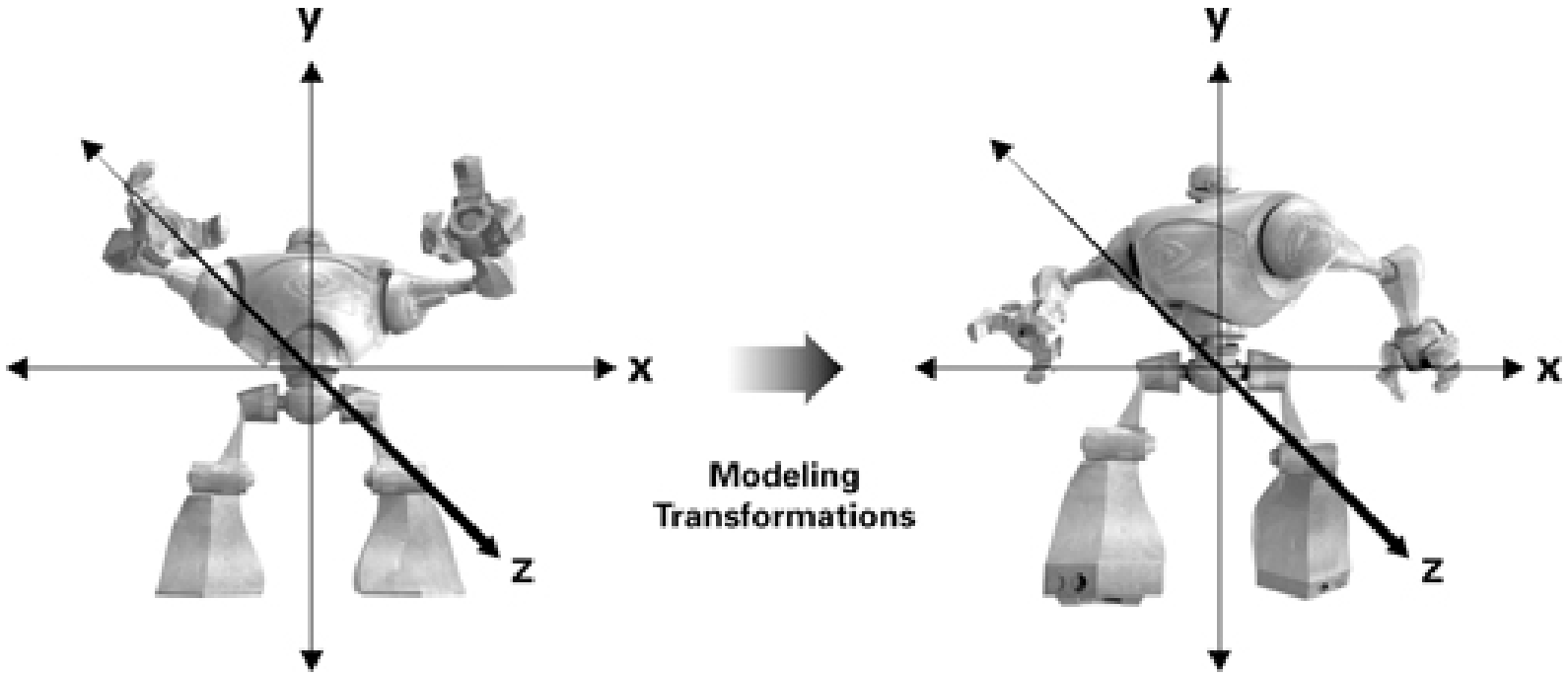
$$\begin{bmatrix} x_w \\ y_w \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \text{Viewport} \\ \text{Transform} \end{bmatrix} \begin{bmatrix} \text{Perspective} \\ \text{Divide} \end{bmatrix} \begin{bmatrix} \text{Projection} \\ \text{Transform} \end{bmatrix} \begin{bmatrix} \text{View} \\ \text{Transform} \end{bmatrix} \begin{bmatrix} \text{Model} \\ \text{Transform} \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$



Modeling Transform

- Object space คือระบบพิกัดที่ศิลปินทำการขึ้นโมเดลมาให้
- World space คือระบบพิกัดกลางของฉากที่โมเดลหลายๆ โมเดลมาอยู่ร่วมกัน
- Modeling transform ทำหน้าที่เปลี่ยน vertex จากที่อยู่ใน object space มาอยู่ใน world space
- ในขณะเดียวกันมันอาจจะเปลี่ยนแปลงหน้าตาหรือท่าทางของโมเดลได้ด้วย

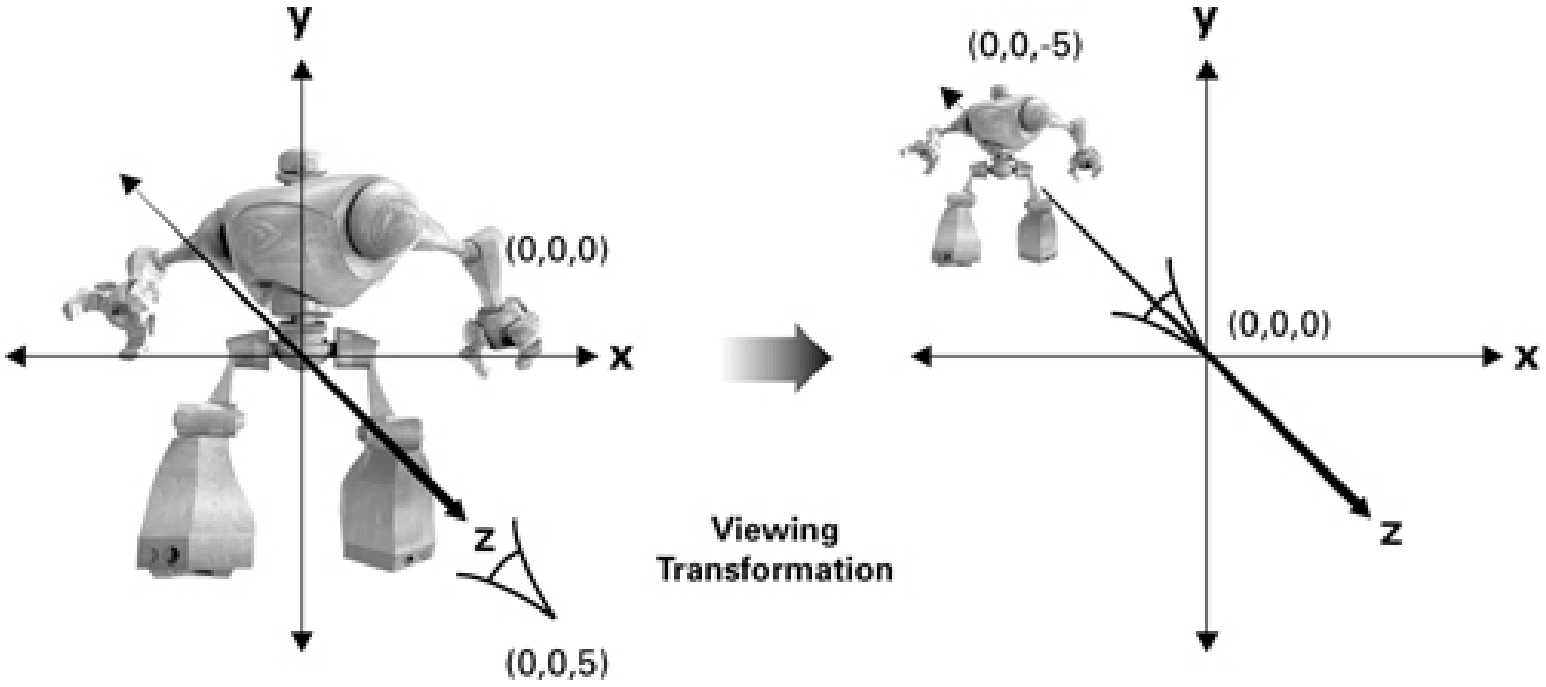
Modeling Transform (ต่อ)



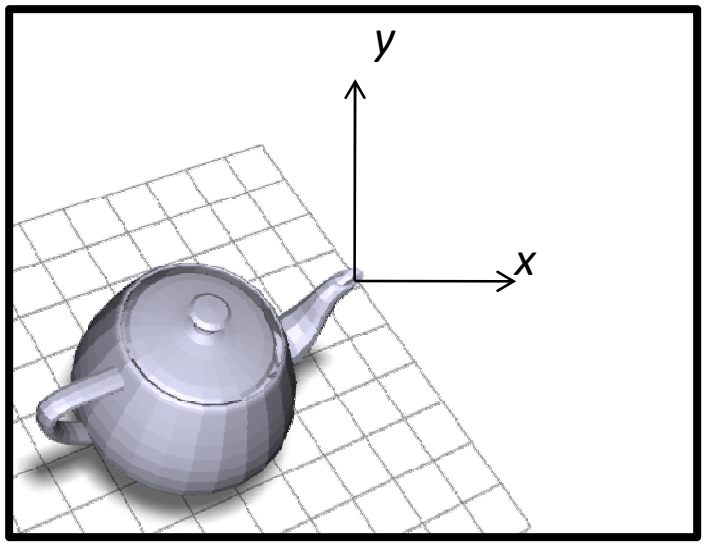
View Transform

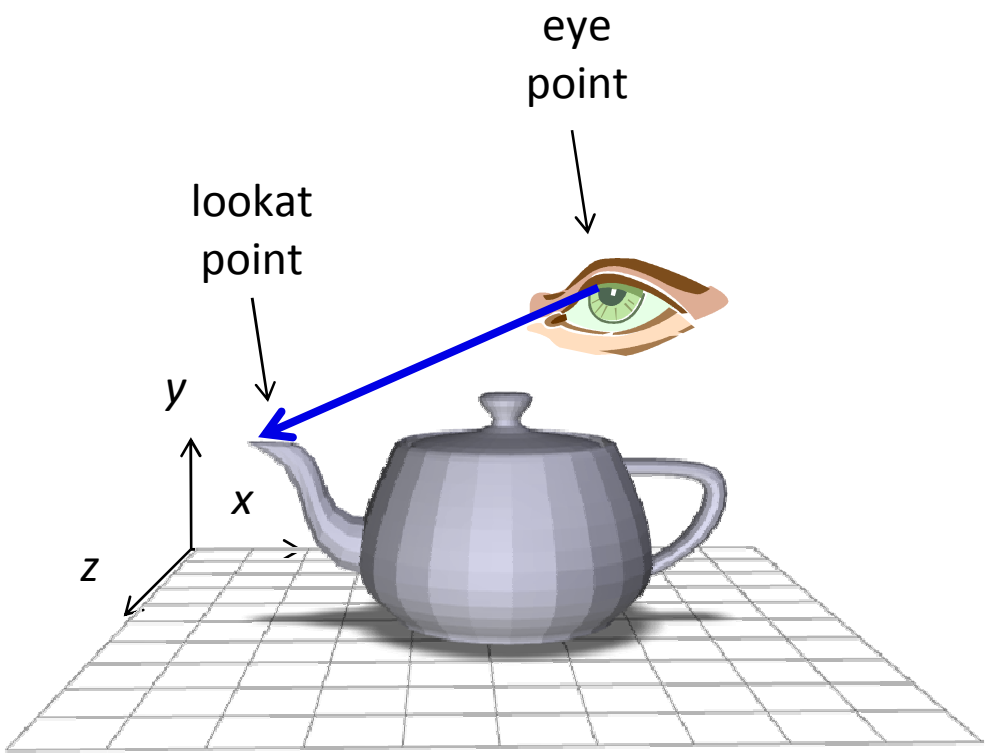
- View transform ใช้ในการเซตมุมกล้อง
- Eye space คือระบบพิกัดที่
  - ตาเราอยู่ที่จุด (0,0,0)
  - เรามองไปในทิศทางแกน z ทางลบ (ในทิศทางของเวกเตอร์ **–k**)
  - ทิศทางแกน y คือ “ด้านบน”
- View transform เปลี่ยน vertex ที่อยู่ใน world space มาอยู่ใน eye space

View Transform (ต่อ)



View Transform (ต่อ)





Modelview Matrix

- OpenGL รวมขั้นตอนการทำ modeling transform และ view transform เข้าด้วยกันเป็นขั้นตอนเดียว
- แทนการแปลงจาก object space ไปเป็น eye space ด้วย **modelview matrix**

$$\left[Modelview \right] = \left[View \right] \left[Model \right]$$

การจัดการกับ Modelview Matrix

- เปลี่ยน mode ของ matrix เป็น modelview matrix ด้วยคำสั่ง glMatrixMode(GL_MODELVIEW)
- หลังจากนั้นใช้คำสั่งอื่นๆ
  - glLoadIdentity
  - glTranslate[fd]
  - glScale[fd]
  - glRotate[fd]
  - glMultMatrix[fd]

คำสั่งเกี่ยวกับ matrix

- glLoadIdentity()
  - ทำให้ค่าของ matrix ใน mode ปัจจุบันที่ OpenGL จำไว้เป็น identity matrix
- glTranslate[fd](a,b,c)
  - สมมติว่า matrix ใน mode ปัจจุบันคือ M
  - คำสั่งนี้จะทำให้ matrix ปัจจุบันกลายเป็น $MT_{a,b,c}$
- glScale[fd](a,b,c)
  - คำสั่งนี้จะทำให้ matrix ปัจจุบันกลายเป็น $MS_{a,b,c}$

คำสั่งเกี่ยวกับ matrix (ต่อ)

- glRotate[fd](a, x, y, z)
  - a คือ มุมที่จะหมุน หน่วยเป็นองศา (ไม่ใช่เรเดียน!)
  - x, y, และ z ระบุแกนที่จะหมุน
  - คำสั่งนี้จะทำให้ matrix ปัจจุบันกลายเป็น $MR_{a,\mathrm{x},y,z}$

คำสั่งเกี่ยวกับ matrix (ต่อ)

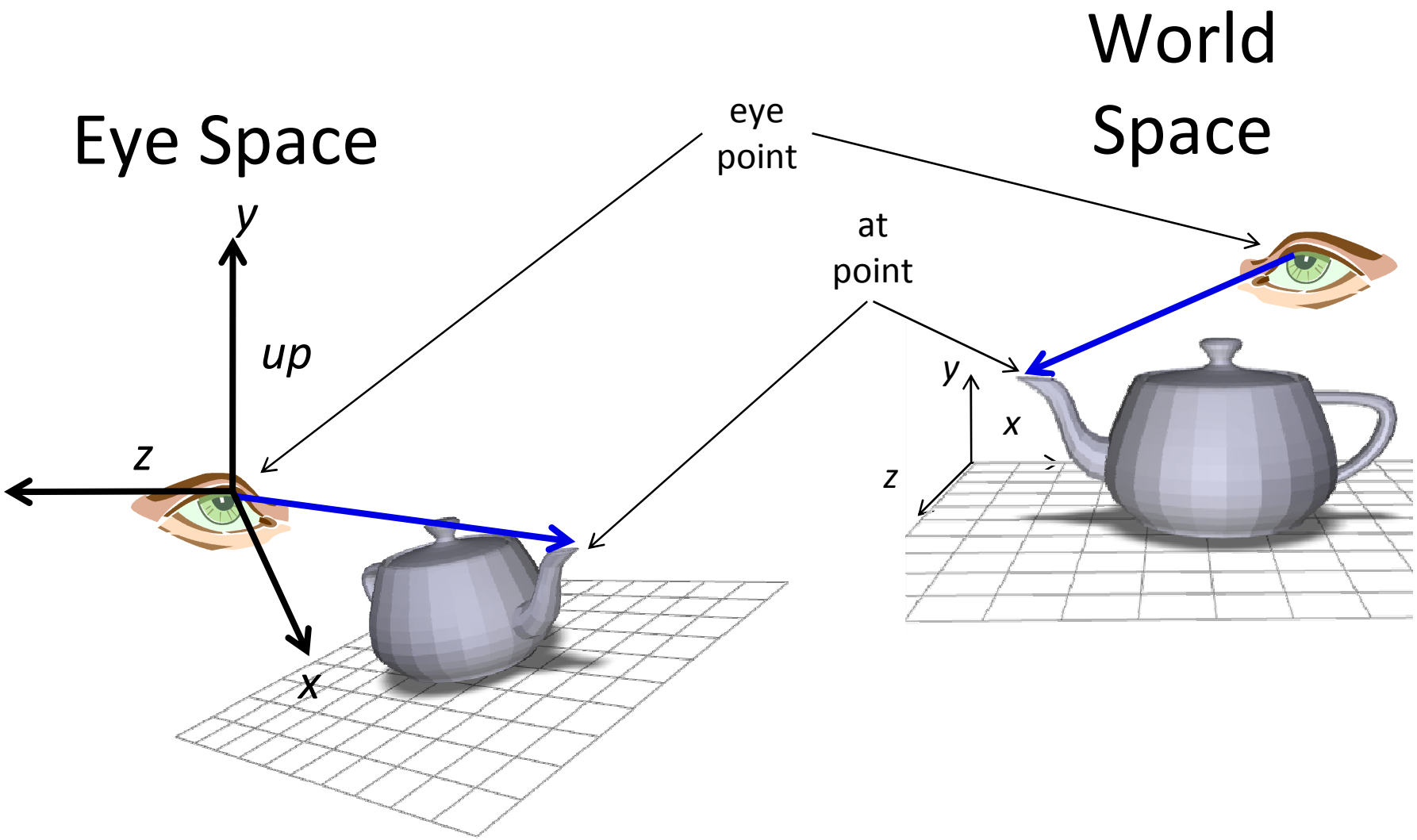
- glMultMatrix[fd](m)
  - m คือ list ของเลข 16 ตัว
  - สมมติว่าให้ m = [a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p]
  - คำสั่งนี้จะทำให้ matrix ปัจจุบันกลายเป็น

$$M\begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix}$$

LookAt Transform

- การเซตมุมกล้องอย่างง่ายแบบหนึ่ง
- บอก
  - eye = ตำแหน่งของตา
  - at = ตำแหน่งที่ตามอง
  - up = ทิศทางด้านบน

การเปลี่ยนระบบพิกัดของ LookAt Transform



gluLookAt

- gluLookAt(eyeX, eyeY, eyeZ, atX, atY, atZ, upX, upY, upZ)
  - คูณ matrix ของ mode ปัจจุบันด้วย matrix ที่ transform ระบบพิกัดโดยทำให้
    - จุด (0,0,0) ในระบบพิกัดใหม่คือจุด eye
    - ทิศทาง –z ของระบบพิกัดใหม่คือทิศทางจากจุด eye ไปยังจุด at
      - กล่าวคือแกน z มีทิศทางเดียวกับเวกเตอร์ eye – at
    - ทิศทางของแกน y จะคล้ายๆ กับทิศทาง up

ตัวอย่าง

- ต้องการเซตมุมกล้องให้กล้องอยู่ที่จุด (-5,-5,-5) แล้วมองไปที่จุด (0,0,0) และมีเวกเตอร์ (1,-1,0) เป็นด้านบน

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(-5,-5,-5,0,0,0,1,-1,0);
```

(วาดรูปอะไรอย่างอื่นต่อไป)

Projection Transformation

- เปลี่ยน eye space เป็น clip space
- พิกัดใน clip space จะใช้เป็นตัวบอกว่าเราจะเห็น vertex ใด หรือไม่เห็น vertex ใด
- กระบวนการตัดสินใจ: vertex ที่เห็นจะต้องมี
  - $-1 \le x \le 1$
  - $-1 \le y \le 1$
  - $-1 \le z \le 1$
- Projection transform ยังมีผลต่อลักษณะภาพที่เราเห็นอีกด้วย

Projection Transform ใน OpenGL

- OpenGL จะจำ matrix ของ projection transform เอาไว้
- เวลาต้องการเปลี่ยนแปลง projection matrix ให้เปลี่ยน mode ของ matrix เป็น GL_PROJECTION ด้วยคำสั่ง glMatrixMode(GL_PROJECTION);
- หลังจากนั้นใช้คำสั่งในการเปลี่ยนแปลง matrix อื่นแบบเดิม เช่น glLoadIdentity(), glMultMatrix(…), ฯลฯ
- ส่วนมากเราจะสั่ง glLoadIdentity() ทันทีหลังจากสั่ง glMatrixMode(GL_PROJECTION) เสร็จแล้ว เพื่อเคลียร์ค่า projection matrix ก่อนใส่ค่าใหม่

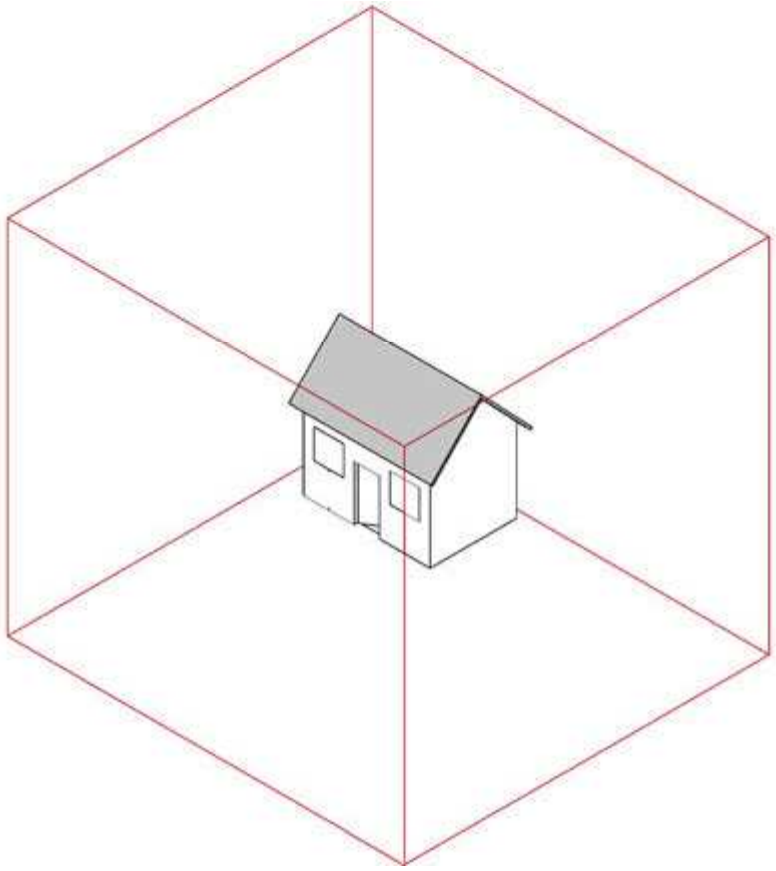
Projection Transformation ที่สำคัญ 2 แบบ

- Orthographic Projection
- Perspective Projection

Orthographic Projection

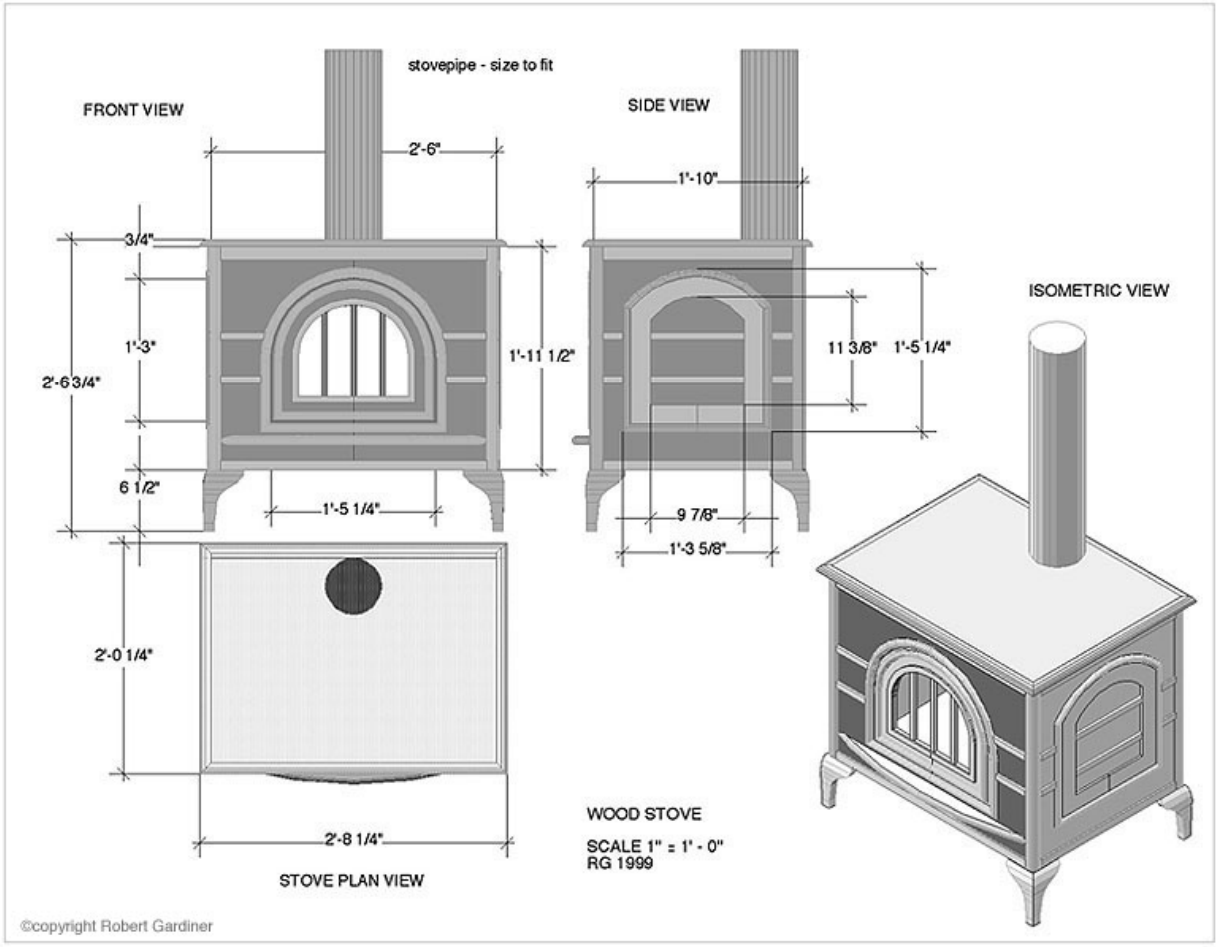
- ปริมาตรของบริเวณที่เห็นเป็นปริซึม
- ไม่มี foreshortening กล่าวคือ ไม่ว่าวัตถุจะอยู่ใกล้ไกลก็เห็นขนาดเท่ากันหมด
- หลังจากฉาก เส้นขนานยังเป็นเส้นขนานอยู่
- ใช้ในโปรแกรมช่วยเขียนแบบ/CAD เนื่องจากขนาดของวัตถุเป็นเรื่องสำคัญ

Orthographic Projection (ต่อ)



http://www2.arts.ubc.ca/TheatreDesign/crslib/drft_1/orthint.htm

Orthographic Projection (ต่อ)



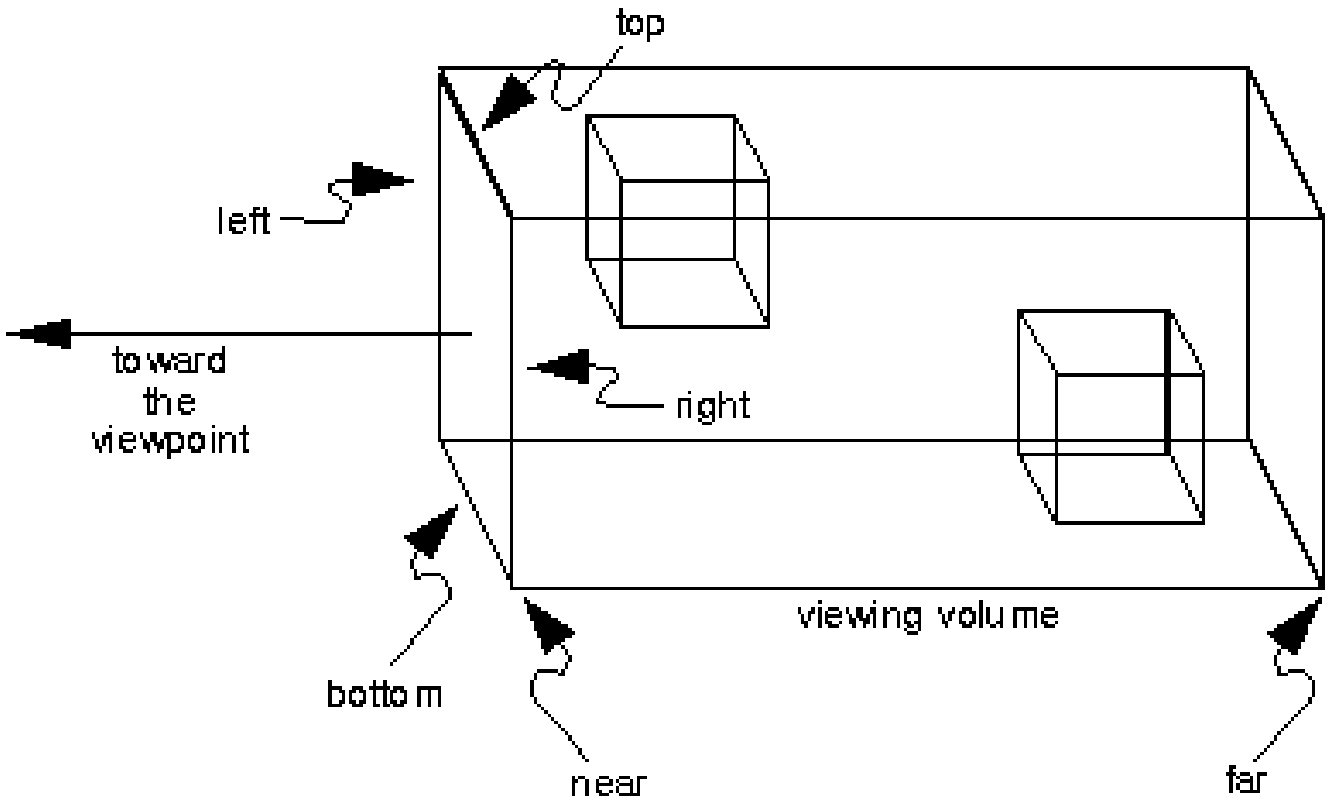
http://www2.arts.ubc.ca/TheatreDesign/crslib/drft_1/cad/wdstv.htm

การนิยาม Orthographic Projection

- นิยามได้โดยการนิยามปริซึมของปริมาตรที่เราต้องการมองเห็น
- ปริซึมนี้สามารถนิยามได้ด้วยตัวเลข 3 คู่
  - left และ right --- ขอบเขตในแนวแกน x
  - top และ bottom --- ขอบเขตในแนวแกน y
  - near และ far --- ขอบเขตในแนวแกน -z (เพราะเรามองในแนว -z)
- ค่าทั้งหกเป็นพิกัดใน eye space
- ปริซึมที่นิยามคือ

$$\{(x,y,z) : left \le x \le right,\ top \le y \le bottom,$$
$$near \le -z \le far\}$$

ปริซึมปริมาตรที่มองเห็น



การนิยาม Orthographic Projection (ต่อ)

- Matrix ของ orthographic projection ต้องทำอะไรบ้า
  - ส่ง x = left ไป x = -1
  - ส่ง x = right ไป x = 1
  - ส่ง y = bottom ไป y = -1
  - ส่ง y = top ไป y = 1
  - ส่ง z = -far ไป z = 1
  - ส่ง z = -near ไป z = -1

Matrix ของ Orthographic Projection

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

คำสั่ง OpenGL เกี่ยวกับ Orthographic Projection

- glOrtho(left, right, bottom, top, near, far)
  - คูณ matrix ปัจจุบันด้วย matrix ของ orthographic projection ในหน้าก่อน
  - ก่อนใช้ควรเรียก

    glMatrixMode(GL_PROJECTION)

    glLoadIdentity()

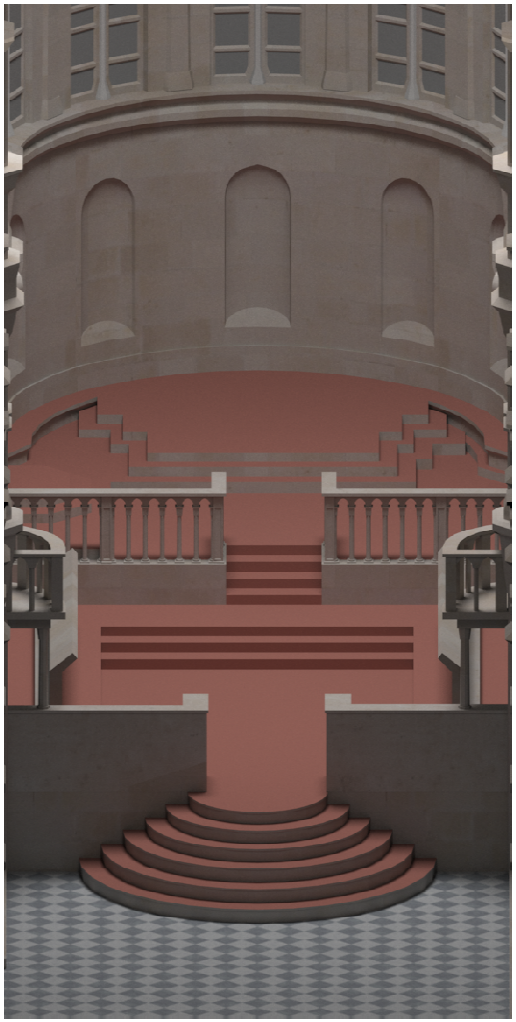
    ก่อนเพื่อเปลี่ยน mode และเคลียร์ค่า projection matrix เดิม
- glOrtho2D(left, right, bottom, top)
  - เหมือนกับ glOrtho แต่ให้ค่า near เป็น 0 และ ค่า far เป็น 1

Perspective Projection

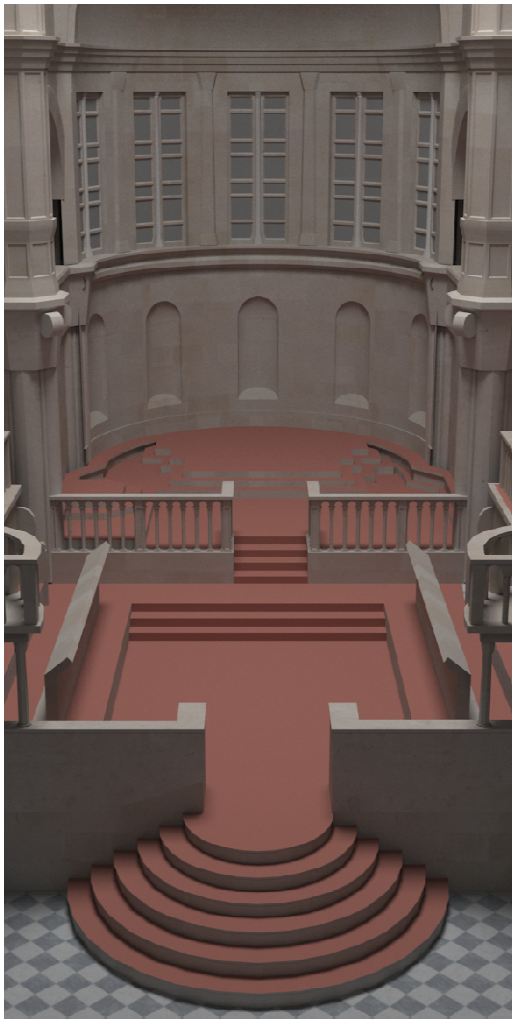
- ปริมาตรของบริเวณที่เห็นเป็น frustum (ปีระมิดยอดตัด)
- มี foreshortening กล่าวคือ อะไรที่อยู่ใกล้จะเห็นใหญ่กว่า
- หลังจากฉายแล้ว เส้นขนานอาจจะไม่ขนานกันเหมือนเดิม
- ให้ความเป็นสามมิติ เพราะเหมือนกับที่ตาคนทำงาน ทำให้เหมือนเข้าไปอยู่ในฉากจริงๆ
- ใช้กับโปรแกรมทางความบันเทิง

Perspective Projection (ต่อ)

Perspective Projection (cont.)



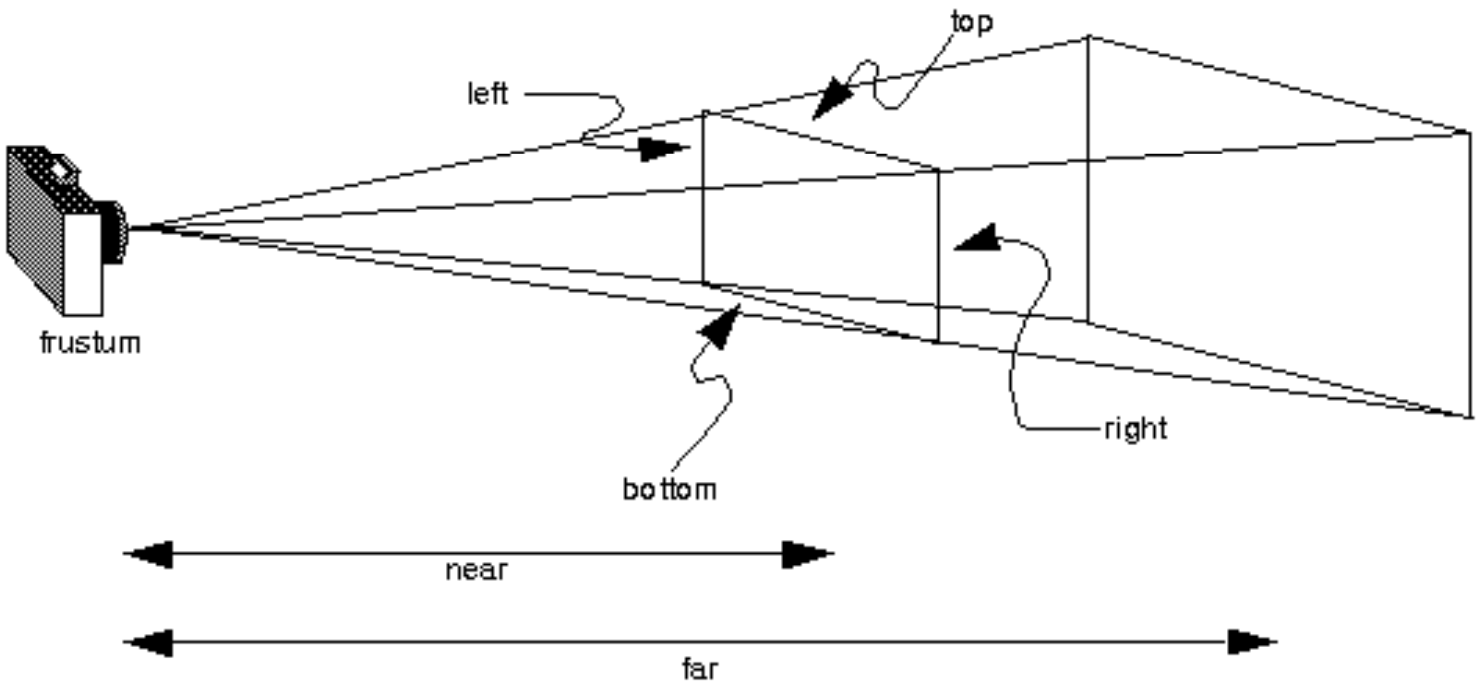
orthographic



perspective

การนิยาม Perspective Projection

- นิยามด้วยเลข 6 ตัวเหมือนกับ orthographic projection



การนิยาม Perspective Projection (ต่อ)

- ปริมาตรที่มองเห็นคือปีระมิดยอดตัดที่มียอดเป็นสี่เหลี่ยม

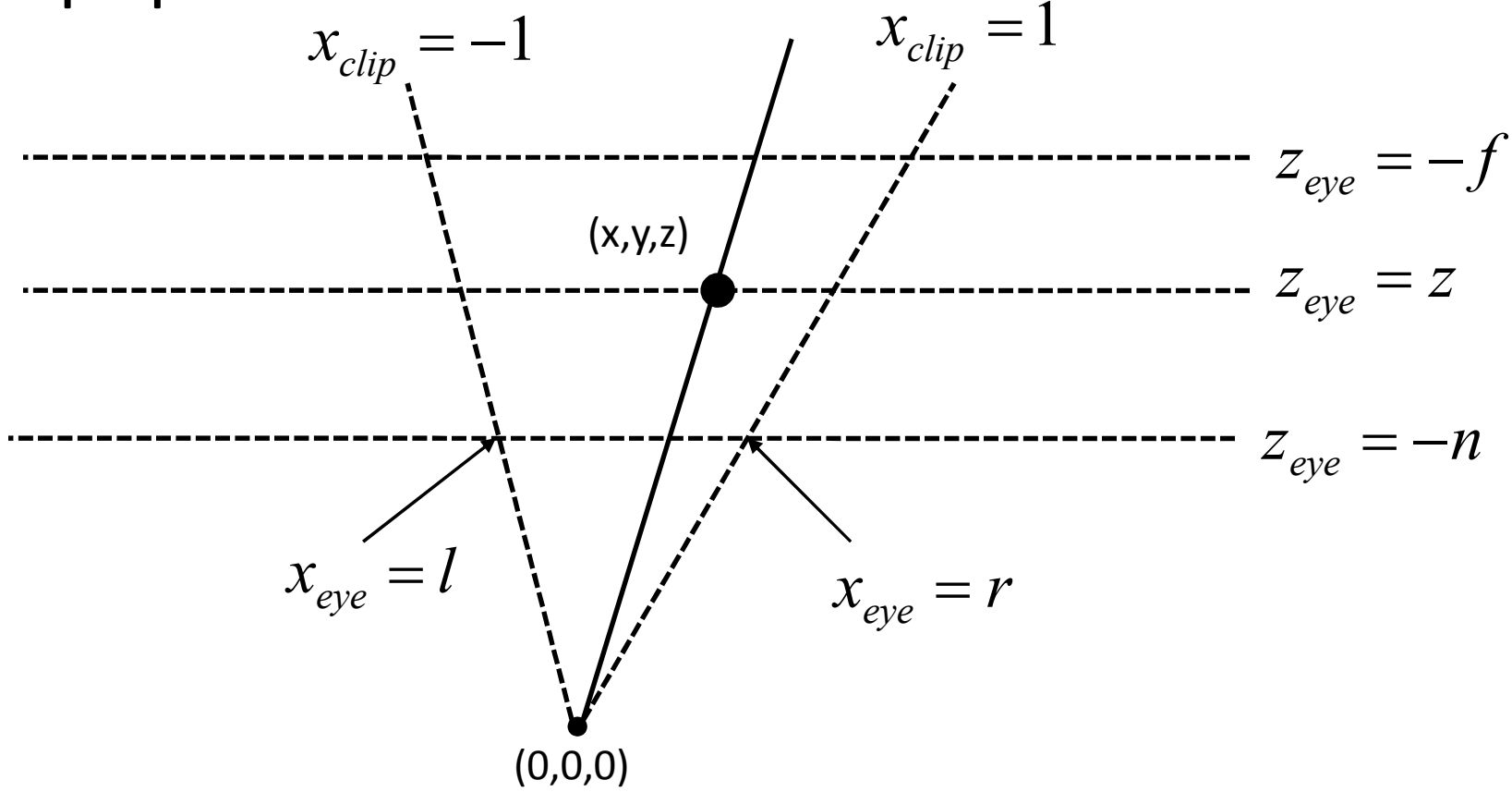
  $\{(x,y,z) : left \le x \le right, bottom \le y \le top,$

  $z = -near\}$

  ซึ่งยอดของมันถูกฉายต่อไปจนถึง $z = -far$

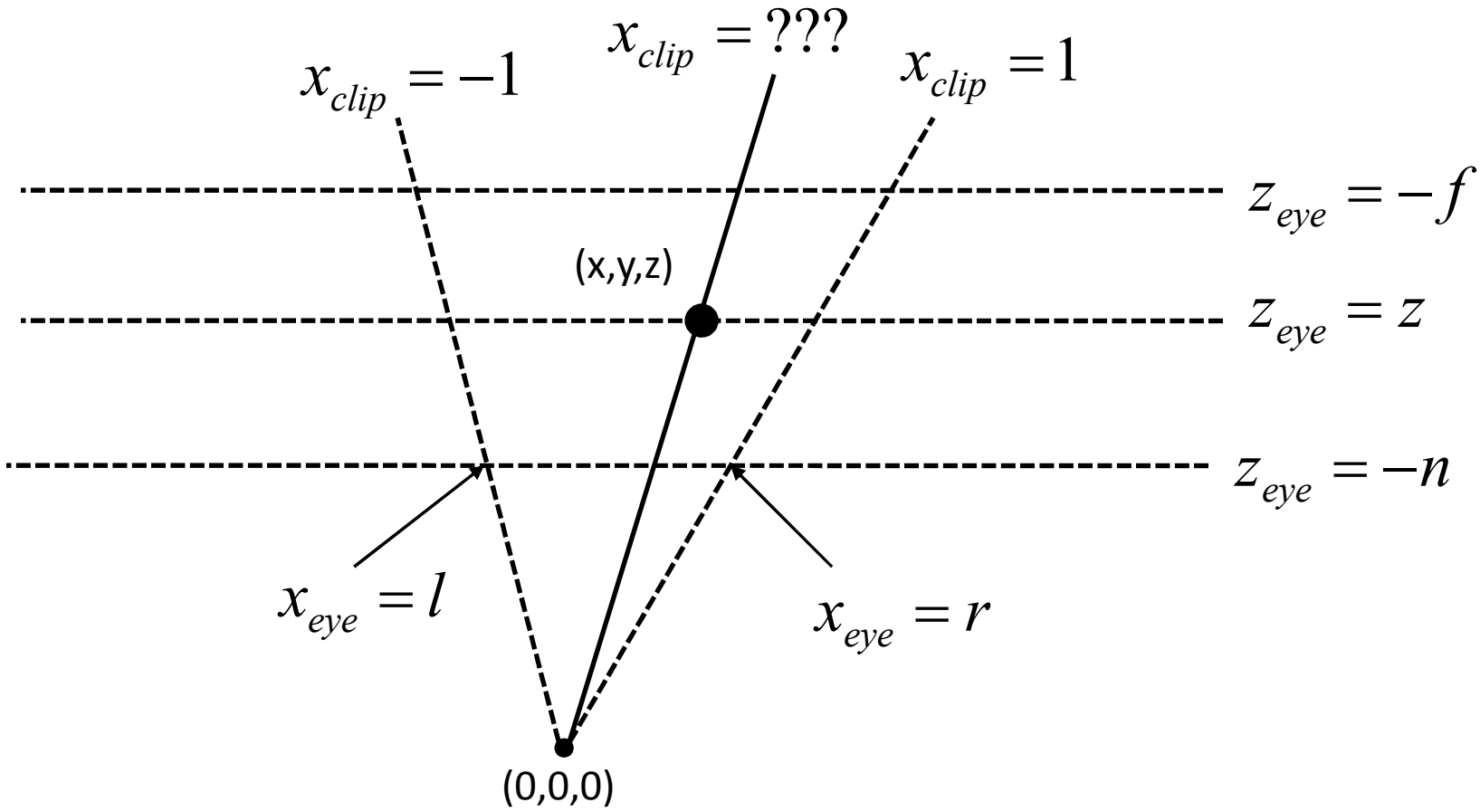
การนิยาม Perspective Projection (ต่อ)

- ให้จุด (x,y,z) มาใน eye space แล้วมันจะถูกแปลงเป็นอะไรใน clip space?



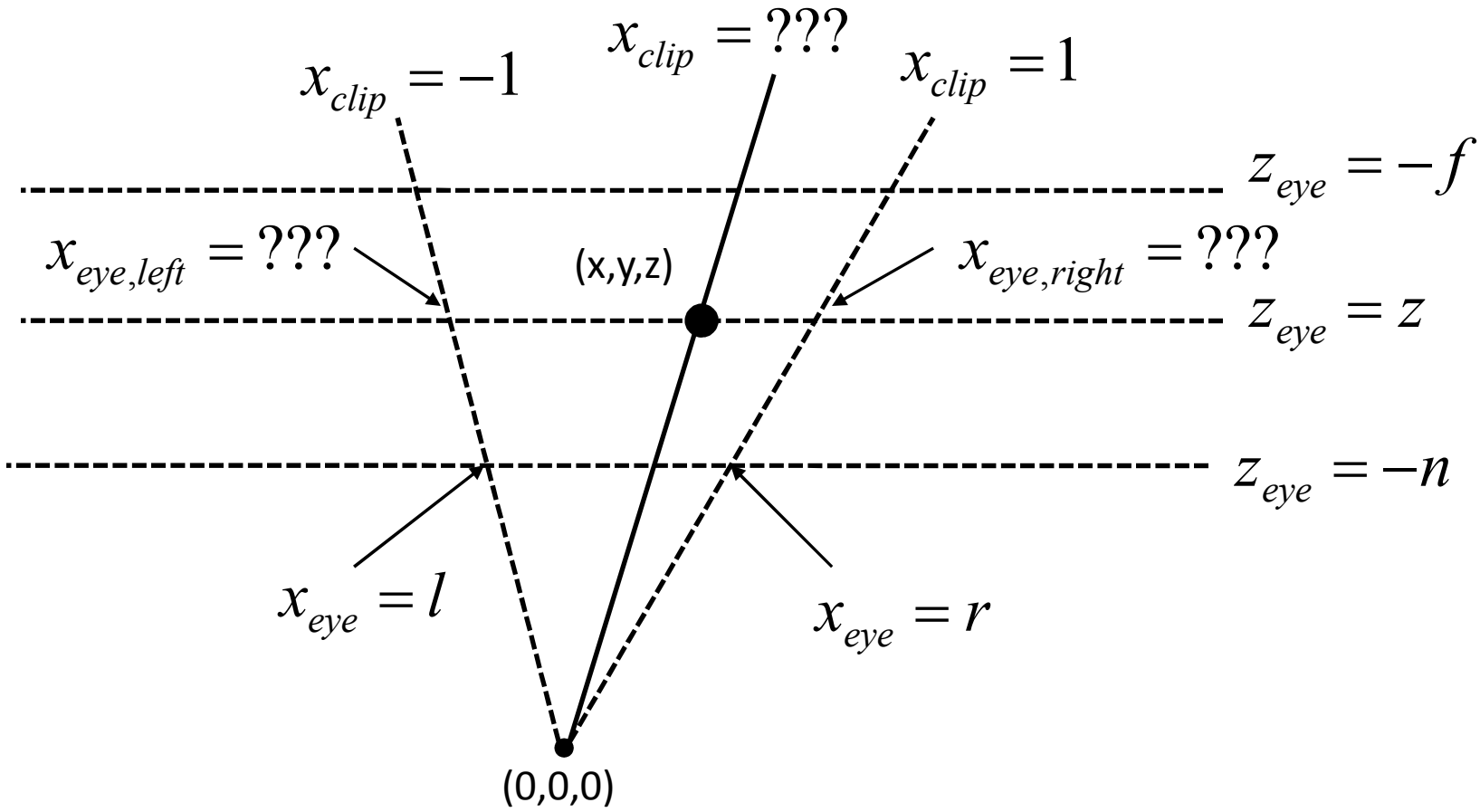
การนิยาม Perspective Projection (ต่อ)

- หา x ใน clip space



การนิยาม Perspective Projection (ต่อ)

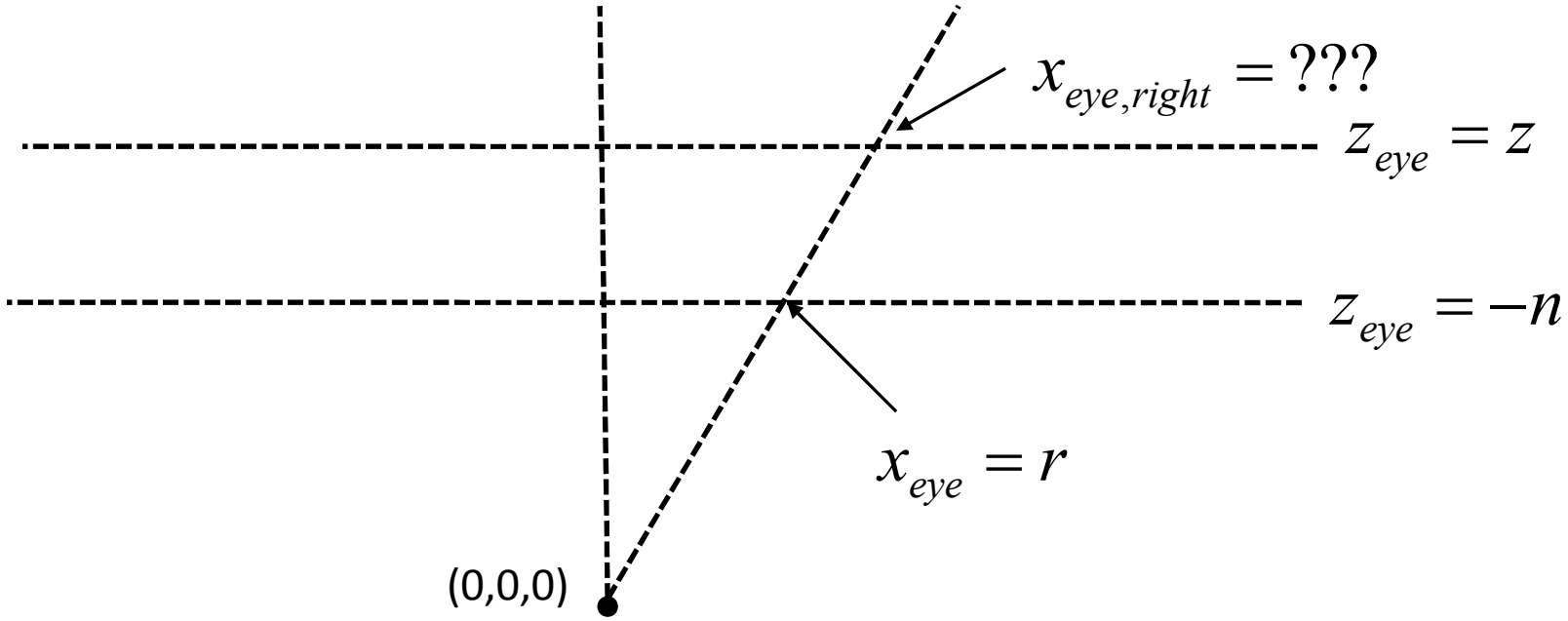
- เริ่มจากการหา x ใน eye space ของจุดปลายสองจุด



การนิยาม Perspective Projection (ต่อ)

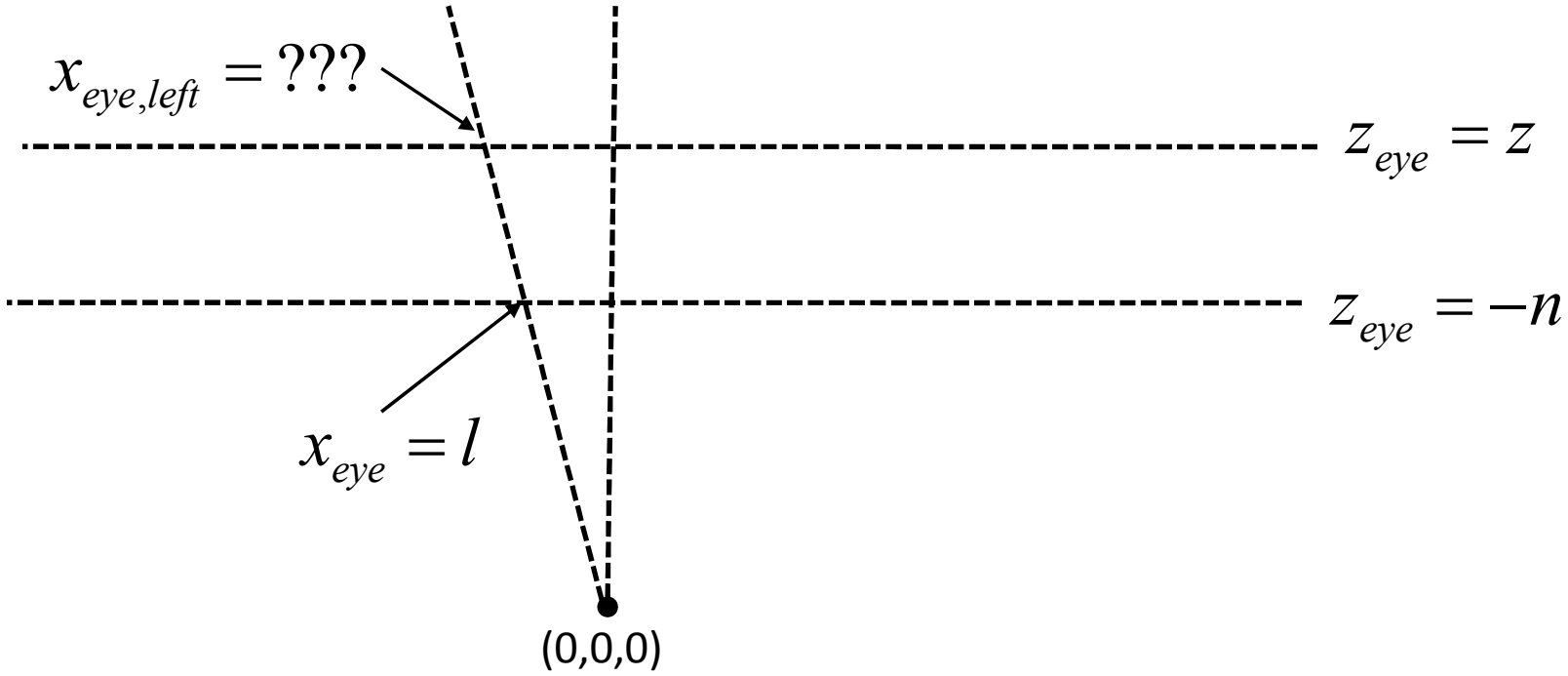
- อาศัยความรู้เรื่องสามเหลี่ยมคล้าย ได้ว่า $\frac{x_{eye,right}}{r} = \frac{z}{-n}$

  ดังนั้น $x_{eye,right} = -\frac{zr}{n}$



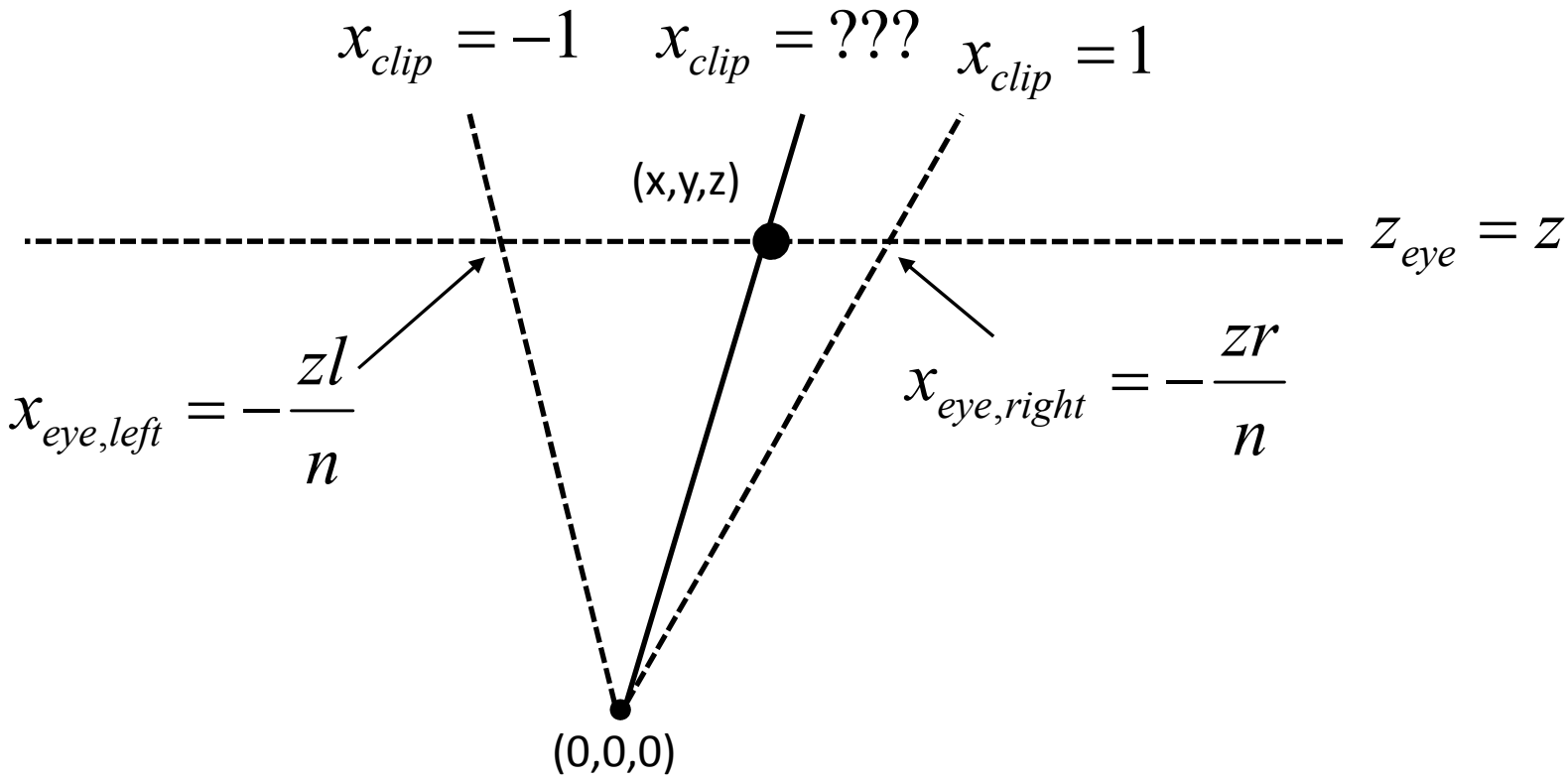
การนิยาม Perspective Projection (ต่อ)

- ทำนองเดียวกัน $\frac{x_{eye,left}}{l} = \frac{z}{-n}$ ดังนั้น $x_{eye,left} = -\frac{zl}{n}$



การนิยาม Perspective Projection (ต่อ)

- เรารู้ว่า $x_{clip} = ax + b$ สำหรับค่าคงที่ a และ b บางตัว



การนิยาม Perspective Projection (ต่อ)

- เนื่องจากถ้า $x = -zl/n$ แล้ว $x_{clip} = -1$

  และถ้า $x = -zr/n$ แล้ว $x_{clip} = 1$
- ได้ว่า

$$-1 = -a\frac{zl}{n} + b$$

$$1 = -a\frac{zr}{n} + b$$

- เมื่อแก้สมการออกมาจะได้ว่า

$$a = -\frac{2n}{(r-l)z}$$

$$b = -\frac{r+l}{r-l}$$

การนิยาม Perspective Projection (ต่อ)

- กล่าวคือ

$$x_{clip} = -\frac{2n}{(r-l)z}x - \frac{r+l}{r-l}$$

- ในทำนองเดียวกันเราก็จะได้ว่า

$$y_{clip} = -\frac{2n}{(t-b)z}y - \frac{t+b}{t-b}$$

การนิยาม Perspective Projection (ต่อ)

- แล้ว z_{clip} ควรจะมีค่าเท่าไหร่?
- ค่า z_{clip} จะถูกใช้เป็น “ความลึก” ของ fragment
- z_{clip} จะต้องมีคุณสมบัติสองประการ
  - ถ้า z น้อย z_{clip} ก็ต้องน้อยตามไปด้วย
  - perspective matrix จะต้องส่งเส้นตรงไปยังเส้นตรง
- ตัวอย่าง z_{clip} ที่ใช้ไม่ได้
  - $z_{clip}=z$
  - $z_{clip} = \sqrt{x^2 + y^2 + z^2}$

การนิยาม Perspective Projection (ต่อ)

- z_{clip} ที่ OpenGL ใช้มีรูป

$$z_{clip} = A + \frac{B}{z}$$

- เนื่องจาก $z_{clip} = -1$ ถ้า $z = -n$

  และ $z_{clip} = 1$ ถ้า $z = -f$

  จะได้ว่า

$$-1 = A - \frac{B}{n}$$

$$1 = A - \frac{B}{f}$$

การนิยาม Perspective Projection (ต่อ)

- เมื่อแก้สมการออกมาแล้วจะได้ว่า

$$B = \frac{2fn}{f-n}$$

$$A = \frac{f+n}{f-n}$$

- กล่าวคือ

$$z_{clip} = \frac{f+n}{f-n} + \frac{2fn}{(f-n)z}$$

การนิยาม Perspective Projection (ต่อ)

- กล่าวคือ perspective projection matrix จะต้องส่ง

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \text{ ไปยัง } \begin{bmatrix} -\frac{2n}{(r-l)z}x - \frac{r+l}{r-l} \\ -\frac{2n}{(t-b)z}y - \frac{t+b}{t-b} \\ \frac{f+n}{f-n} + \frac{2fn}{(f-n)z} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \frac{2n}{r-l}x + \frac{r+l}{r-l}z \\ \frac{2n}{t-b}y + \frac{t+b}{t-b}z \\ -\frac{f+n}{f-n}z - \frac{2fn}{f-n} \\ -z \end{bmatrix}$$

Matrix ของ Perspective Projection

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

คำสั่ง OpenGL เกี่ยวกับ Perspective Projection

- glFrustum(left, right, bottom, top, near, far)
  - คูณ matrix ปัจจุบันด้วย matrix ของ perspective projection ในหน้าก่อน
  - ก่อนใช้ควรเรียก

    glMatrixMode(GL_PROJECTION)

    glLoadIdentity()

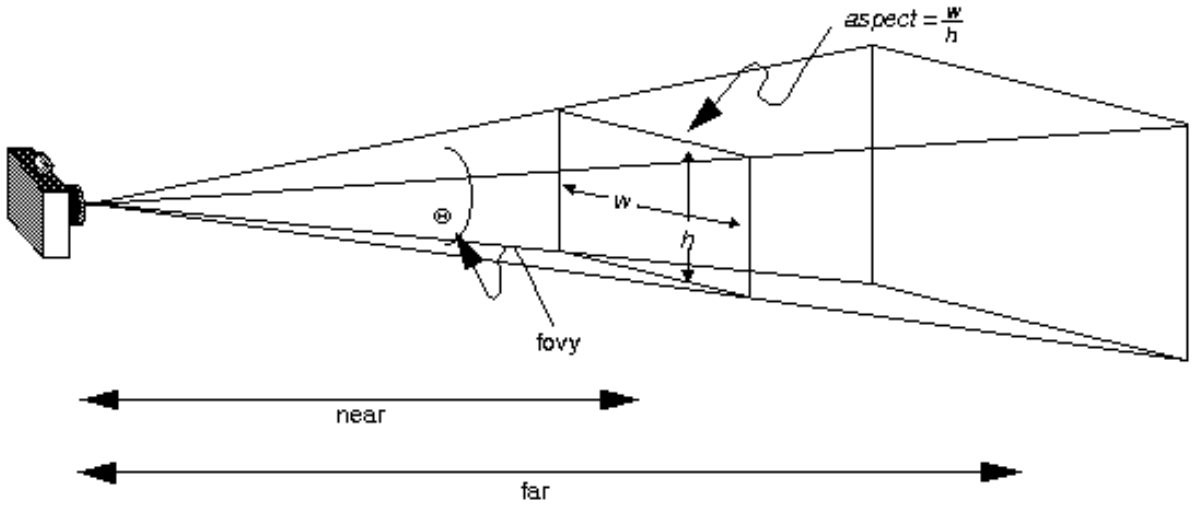
    ก่อนเพื่อเปลี่ยน mode และเคลียร์ค่า projection matrix เดิม

คำสั่ง OpenGL เกี่ยวกับ Perspective Projection (ต่อ)

- gluPerspective(fovy, aspect, near, far)
  - คูณ matrix ปัจจุบันด้วย perspective projection matrix เช่นเดียวกับ glFrustrum
  - มีผลเหมือนกับสั่ง glFrustum โดยได้
    - top = near * tan(fovy / 2)
    - bottom = -top
    - right = aspect * top
    - left = -right

คำสั่ง OpenGL เกี่ยวกับ Perspective Projection (ต่อ)

- fovy ย่อมาจาก field of view Y หมายถึงความกว้างของมุมมองตามแนวแกน y (มีหน่วยเป็นองศา)
- aspect คือ aspect ratio ของหน้าตัดของปีระมิด
- ปีระมิดที่ gluPerspective สร้างมีหน้าตาเป็นดังข้างล่าง



คำสั่ง OpenGL เกี่ยวกับ Perspective Projection (ต่อ)

- สังเกตว่าคำสั่ง glFrustum สามารถสร้างปีระมิดที่ไม่สมมาตรรอบแกน z ได้
- แต่ปีระมิดที่สร้างด้วย gluPerspective จะเป็นปีระมิดที่สมมาตรรอบแกน z เสมอ