CS 174A
Discussion 1B-Week 5

02/06/2020

Homogeneous Coordinates

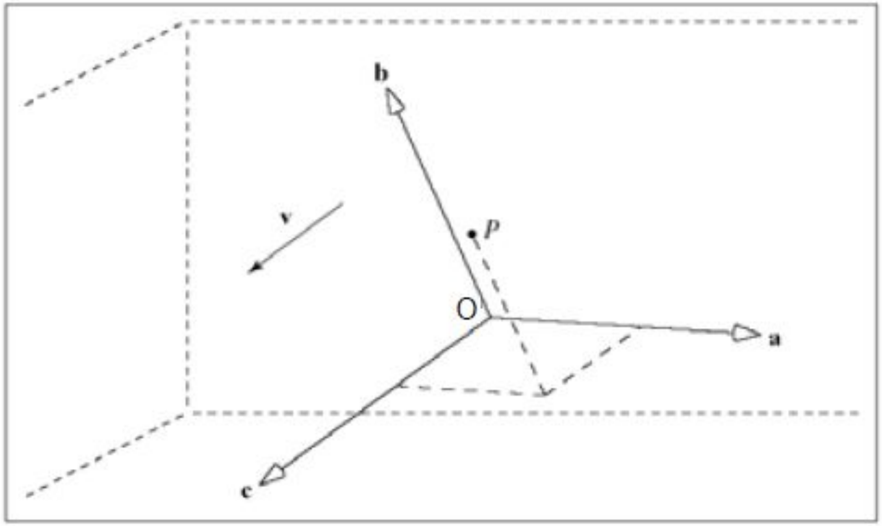
Homogeneous Coordinates

- Vectors and points are both presented as 4×1 column matrices:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix}$$

Homogeneous Representation of Points and Vectors

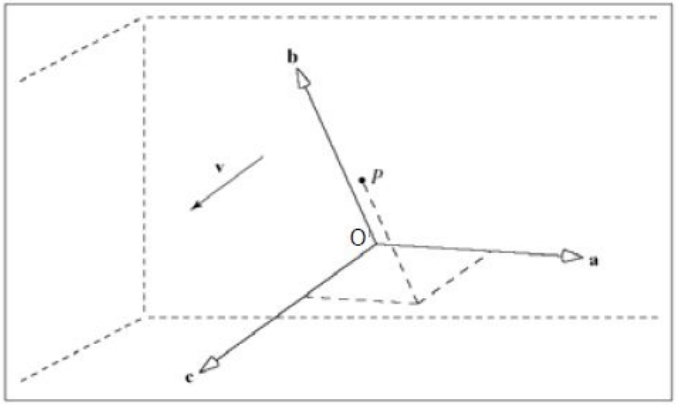
- Suppose we have a coordinate system represented by unit vectors of a,b and c as well as coordinate O for the origin



Homogeneous Representation of Points and Vectors

- Then a point $p = (p_1, p_2, p_3)$ can be represented as:

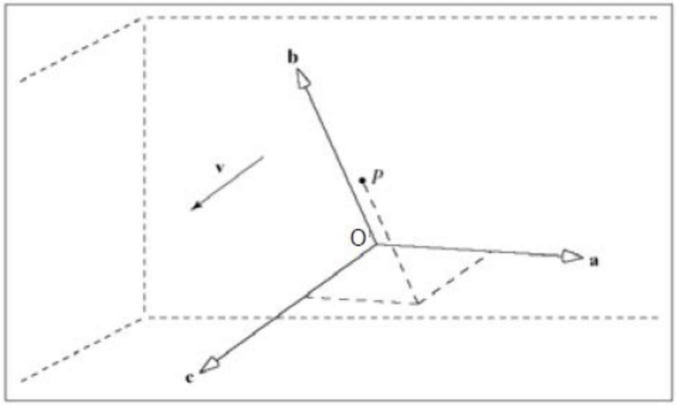
$$P = O + p_1\mathbf{a} + p_2\mathbf{b} + p_3\mathbf{c}$$



Homogeneous Representation of Points and Vectors

- Similarly, a vector $v = (v_1, v_2, v_3)$ can be represented as:

$$\mathbf{v} = v_1\mathbf{a} + v_2\mathbf{b} + v_3\mathbf{c}$$

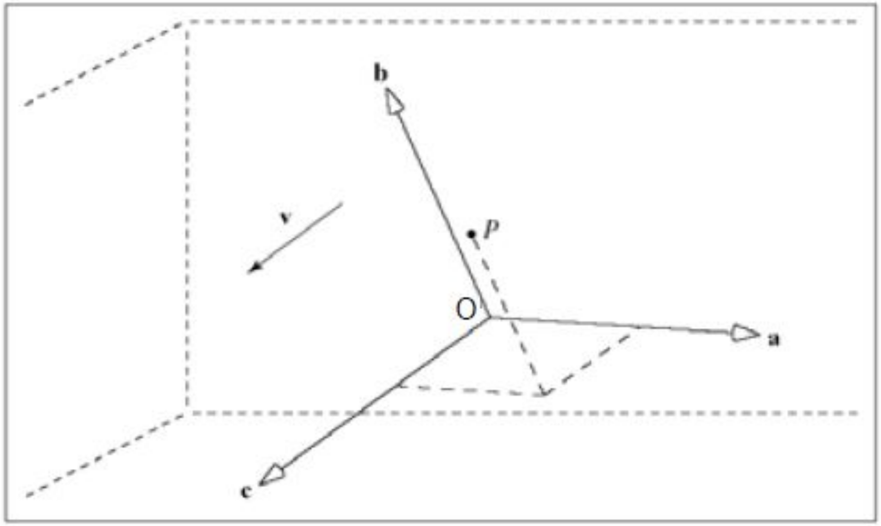


Homogeneous Representation of Points and Vectors

- In homogeneous coordinates, we can represent them as:

$$\mathbf{v} = [\mathbf{a}\ \ \mathbf{b}\ \ \mathbf{c}\ \ O]\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix}$$

$$P = [\mathbf{a}\ \ \mathbf{b}\ \ \mathbf{c}\ \ O]\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix}$$



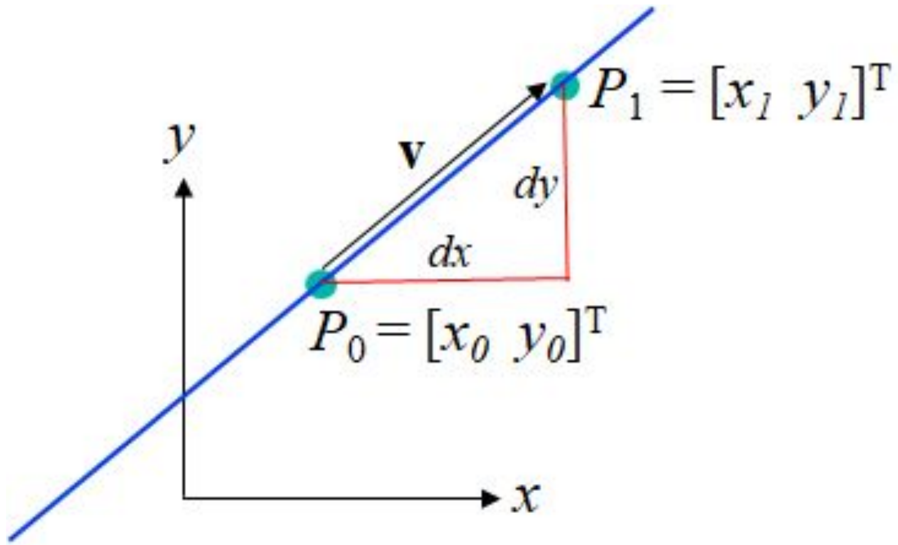
Homogeneous Representation of Points and Vectors

- Points and vectors are both represented as 4×1 column matrices. Does it make sense to just add them together ? what's the outcome ?

$$[p_1, p_2, p_3, 1]^{\mathrm{T}} + [v_1, v_2, v_3, 0]^{\mathrm{T}} = [p_1 + v_1,\ p_2 + v_2,\ p_3 + v_3,\ 1]^{\mathrm{T}}$$

Lines

- A line in 2D can have 3 different representations:
  - Explicit

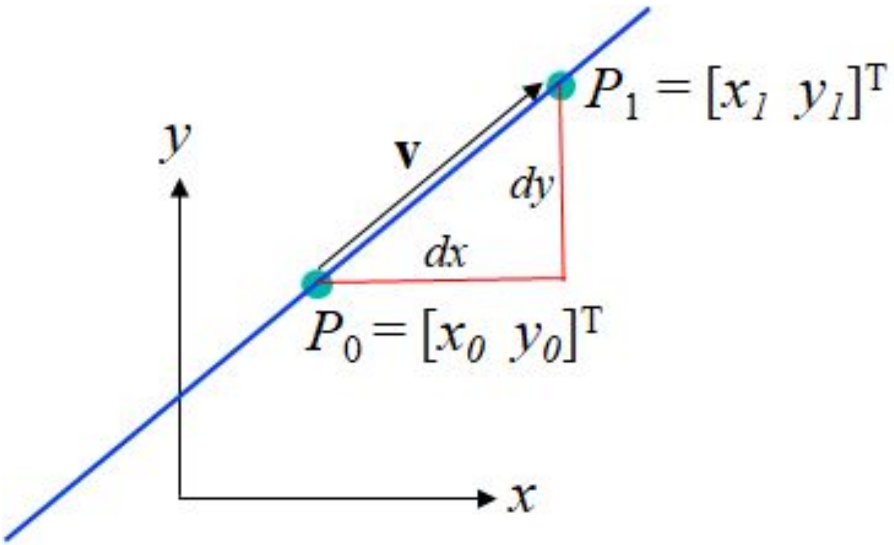


$$y = \alpha x + \beta$$

$$y = m(x - x_0) + y_0; \quad m = \frac{dy}{dx} = \frac{y_1 - y_0}{x_1 - x_0}$$

Lines

- A line in 2D can have 3 different representations:
  - Implicit



$$f(x,y) = (x - x_0)dy - (y - y_0)dx$$

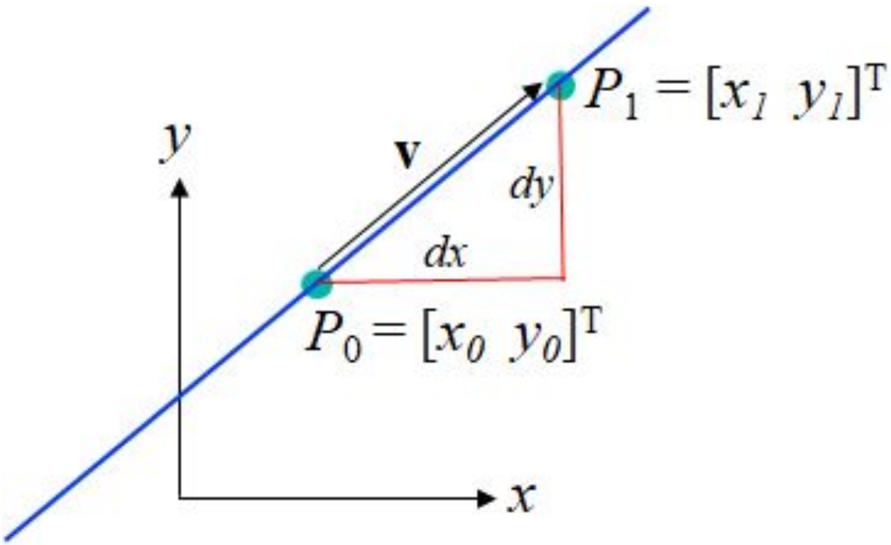
if $f(x,y) = 0$ then (x,y) is **on** the line

$f(x,y) > 0$ then (x,y) is **below** the line

$f(x,y) < 0$ then (x,y) is **above** the line

Lines

- A line in 2D can have 3 different representations:
  - Parametric



$$x(t) = x_0 + t(x_1 - x_0)$$

$$y(t) = y_0 + t(y_1 - y_0)$$

$$t \in [0,1] \text{ for line segment, or } t \in [-\infty, \infty] \text{ for infinite line}$$

$$P(t) = P_0 + t(P_1 - P_0) \quad \text{or} \quad P(t) = P_0 + t\,\mathbf{v}$$

$$P(t) = (1-t)P_0 + tP_1$$

Transformations

Transforms

- Linear Transformation: function T: $R^n \rightarrow R^m$ is a linear transform if it satisfies:

$$T(c_1\vec{u} + c_2\vec{v}) = c_1 T(\vec{u}) + c_2 T(\vec{v})$$

- T can be obviously represented by a matrix.
- Intuitively, linear transforms leave the origin untouched.

UCLA

Linear Transforms

- Linear Transformation can be compactly written as matrix multiplications:

$$Q = \mathcal{T}(P)$$

$$= \begin{bmatrix} m_{11}P_x + m_{12}P_y \\ m_{21}P_x + m_{22}P_y \end{bmatrix}$$

$$= \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix}$$

$$= \mathbf{M}P$$

Linear Transforms

- What kind of transformations can we get from the following ?

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} = \begin{bmatrix} m_{11}P_x + m_{12}P_y \\ m_{21}P_x + m_{22}P_y \end{bmatrix}$$

Linear Transforms

- Let's look at translation in more details. Translation can be formally described as:

$$Q = P + t$$

But this is not the same as :

$$Q = MP$$

Affine Transforms

- Translation is not a linear transformation.
- It's an affine transformation.
- In essence, we can represent

affine transformation = linear + translation

Affine Transforms

- Transforming points:

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{array}{c} \begin{array}{cc} \text{Rotation/Scaling/Shearing} & \text{Translation} \end{array} \\ \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{bmatrix} \end{array} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Affine Transforms

- Transforming vectors:

$$\begin{bmatrix} W_x \\ W_y \\ 0 \end{bmatrix} = \begin{array}{c} \begin{array}{cc} \text{Rotation/Scaling/Shearing} & \text{Translation} \end{array} \\ \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{bmatrix} \end{array} \begin{bmatrix} V_x \\ V_y \\ 0 \end{bmatrix}$$

Affine Transforms

- Scaling

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

$$\text{Uniform scaling:} \quad s_x = s_y$$

Inverse

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Affine Transforms

- Rotation

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Inverse

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

Affine Transforms

- Shear in the x-direction

$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Inverse

$$\begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & -a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Affine Transforms

- Translation

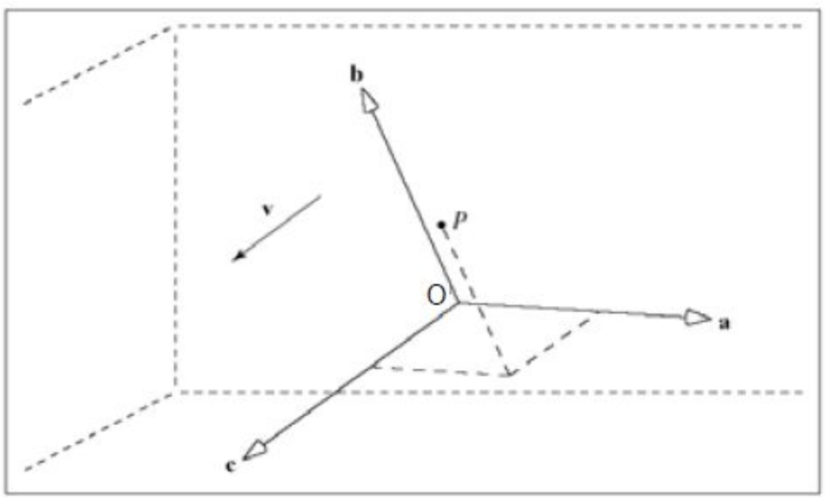
$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

Inverse

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Homogeneous Representation of Points and Vectors

- Vector v and point P can be represented in terms of



$$\begin{array}{c} \begin{array}{cc} \text{Rotate/Scale/Shear} & \text{Translate} \end{array} \\ \mathbf{M} = \begin{bmatrix} \boxed{\begin{matrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{matrix}} & \boxed{\begin{matrix} m_{14} \\ m_{24} \\ m_{34} \end{matrix}} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \end{array}$$

$$\begin{array}{c} \begin{array}{cccc} \text{Basis Vector 1} & \text{Basis Vector 2} & \text{Basis Vector 3} & \textit{Origin Point} \end{array} \\ \mathbf{M} = \begin{bmatrix} \boxed{\begin{matrix} m_{11} \\ m_{21} \\ m_{31} \\ 0 \end{matrix}} & \boxed{\begin{matrix} m_{12} \\ m_{22} \\ m_{32} \\ 0 \end{matrix}} & \boxed{\begin{matrix} m_{13} \\ m_{23} \\ m_{33} \\ 0 \end{matrix}} & \boxed{\begin{matrix} m_{14} \\ m_{24} \\ m_{34} \\ 1 \end{matrix}} \end{bmatrix} \end{array}$$

Point vs Coordinate Transform

- If we transform point P to Q by applying n transformations:

$$Q = \mathbf{M}_n \ldots \mathbf{M}_2 \mathbf{M}_1 P$$

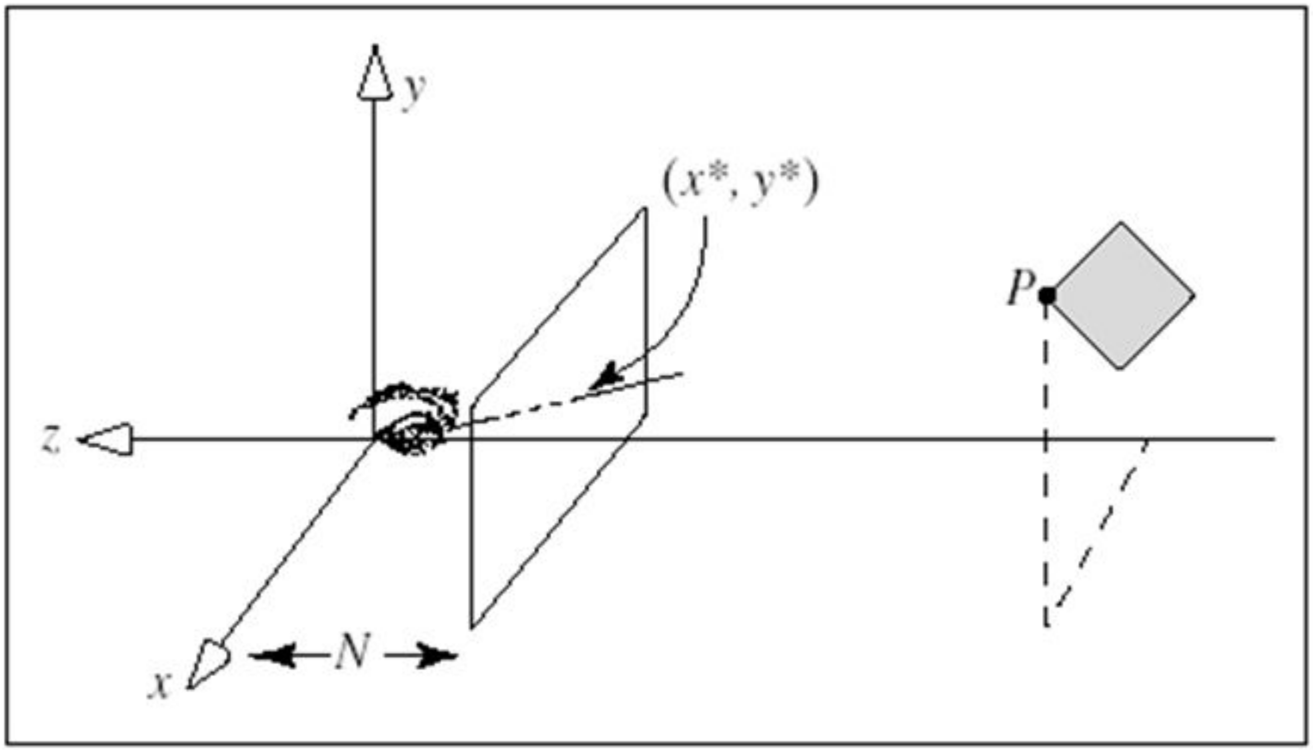
Then, you can also view this as transforming the canonical coordinate system:

$$P = \mathbf{M}_1^{-1} \mathbf{M}_2^{-1} \ldots \mathbf{M}_n^{-1} Q$$

Projections

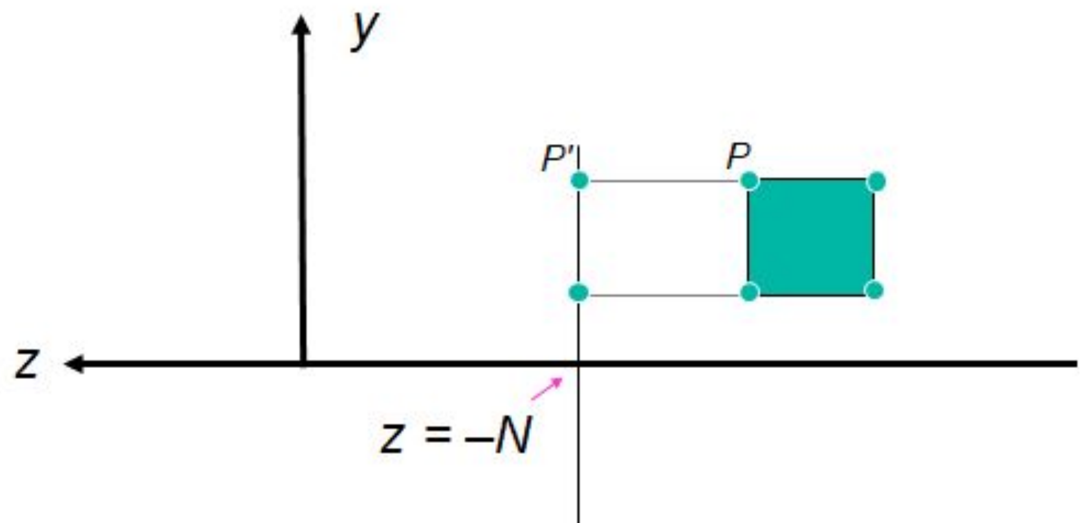
Orthographic Projection

- Assuming the camera is at (0,0,0) and image plan is at Z=-N



Orthographic Projection

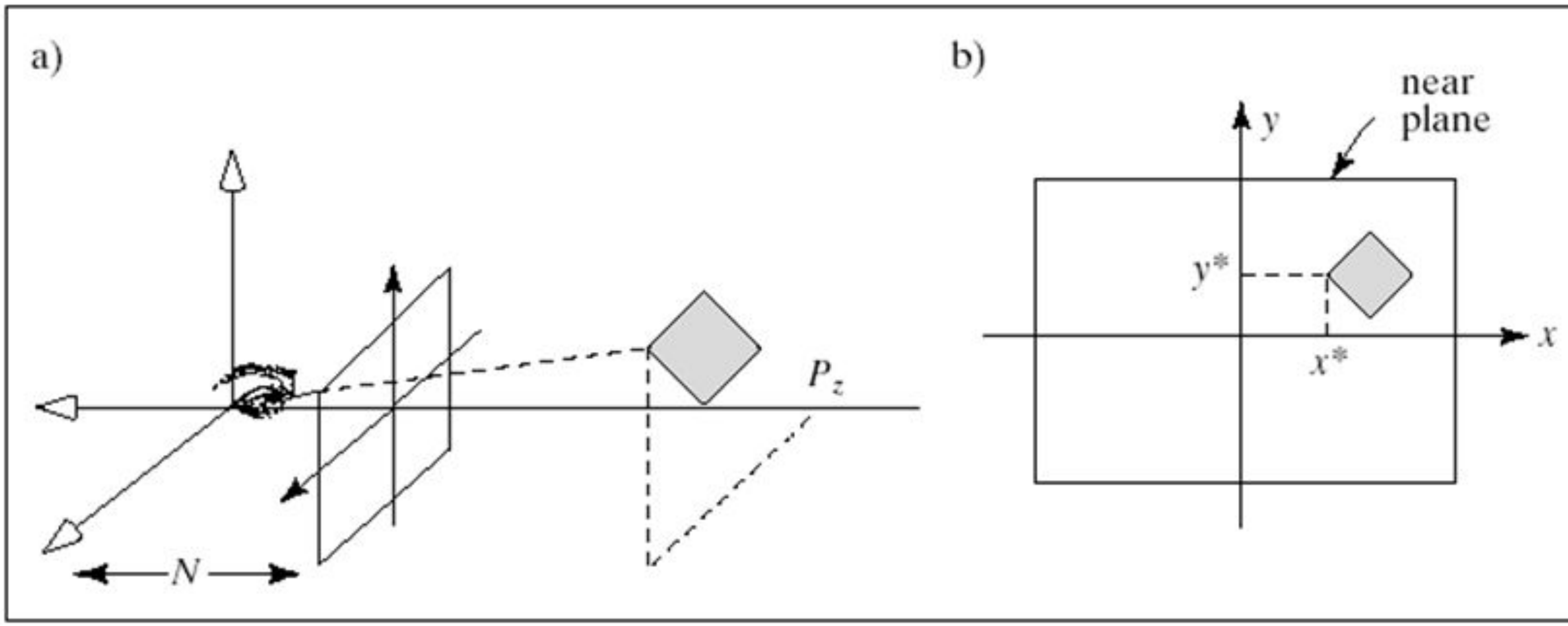
- Assuming the camera is at (0,0,0) and image plan is at Z=-N



$$\begin{bmatrix} P'_x \\ P'_y \\ P'_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -N \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

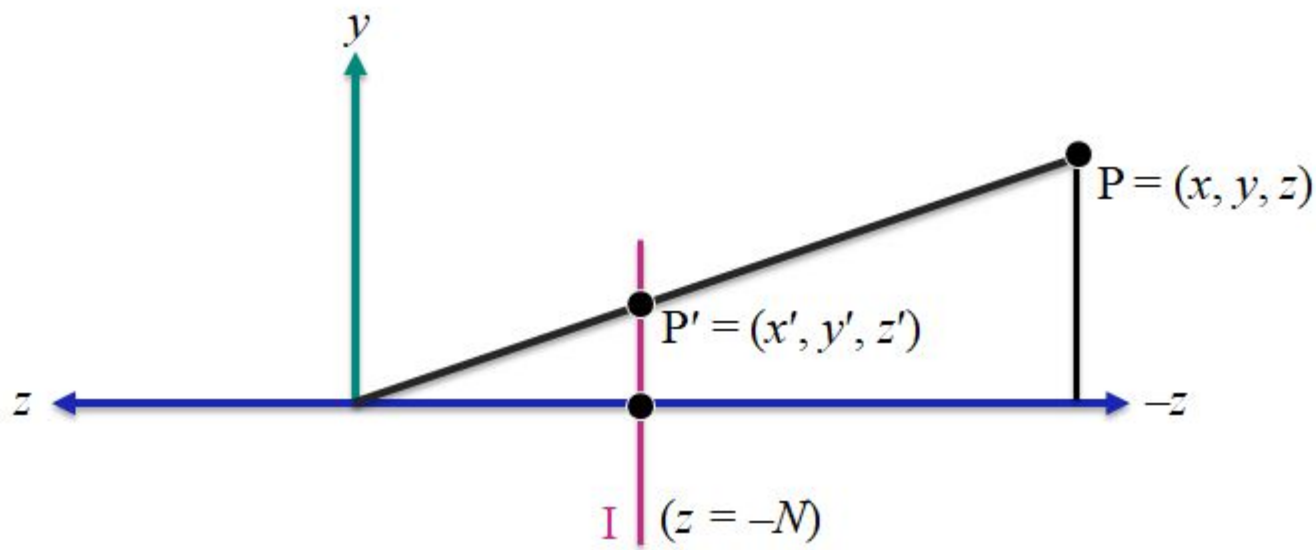
Perspective Projection

- Assuming the camera is at (0,0,0) and image plan is at Z=-N



Perspective Projection

- Basic calculations



$$P_y' = P_y N / -P_z$$

$$P_x' = P_x N / -P_z$$

$$P_z' = -N$$

Perspective Projection

- Basic calculations

$$\begin{bmatrix} P'_x \\ P'_y \\ P'_z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x N/(-P_z) \\ P_y N/(-P_z) \\ -N \\ 1 \end{bmatrix} \xrightarrow[-P_z/N]{\times} \begin{bmatrix} P_x \\ P_y \\ P_z \\ -P_z/N \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/N & 0 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

Therefore:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/N & 0 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \xrightarrow[\text{homogenize}]{\text{and then:}} \begin{matrix} \div \\ -P_z/N \end{matrix} \begin{bmatrix} P'_x \\ P'_y \\ P'_z \\ 1 \end{bmatrix}$$

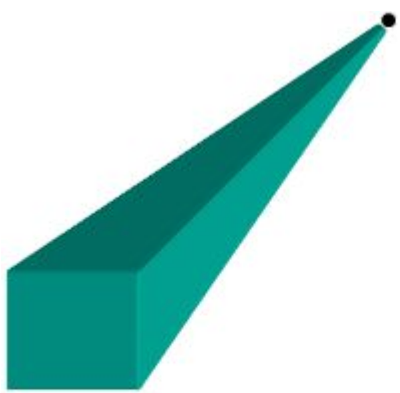
Matrix $\mathbf{M}$

$\div -P_z/N$

Homogenization step:
"Perspective Division"
(divide by w = $-P_z/N$)

Perspective Projection

- Key points to remember:
  - Perspective projection is not a linear ! (obviously since we are doing the perspective division)
  - Lines remain lines
  - Parallel lines may or may not remain parallel
    - If lines are parallel to the image plane, they will remain parallel
    - If not parallel to the image plane, they will converge to a vanishing point



OpenGL

OpenGL

- Transformations are retained as part of the graphics state (Current Transformation Matrix or CTM).
- Transformations are cumulative and the order matters.
- “ glLoadIdentity ” sets the CTM to the identity matrix, for a “fresh start”.
- When a command such as “ glRotate ” issued, the appropriate transformation matrix is updated.
- It doesn’t overwrite the old CTM. It updates CTM by matrix multiplication.

```
// Example 1
Display(){
…
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0,0.0,-6.0);
glRotatef(45.0,0.0,1.0,0.0);
glScalef(2.0, 2.0, 2.0);
DrawCube();
…}
```

```
// Example II
Display(){
…
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(45.0,0.0,1.0,0.0);
glTranslatef(0.0,0.0,-6.0);
glScalef(2.0, 2.0, 2.0);
DrawCube();
…}
```

OpenGL

- glPushMatrix();
  - – Save the state.
  - – Push a copy of the CTM onto the stack.
  - – The CTM itself is unchanged.

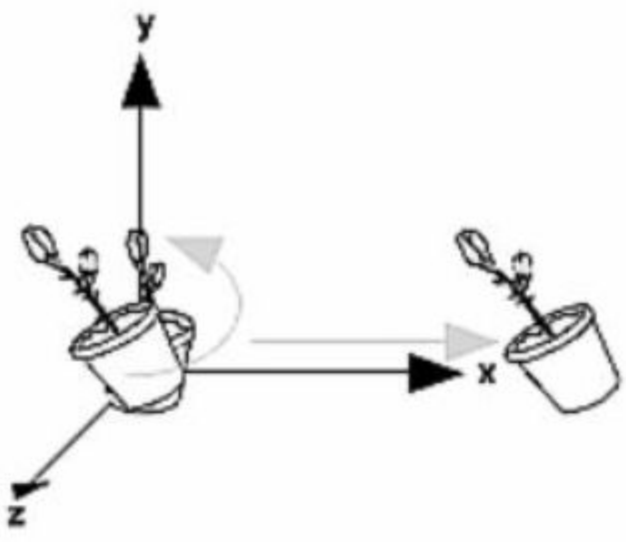
- glPopMatrix(); –
  - Restore the state, as it was at the last Push. –
  - Overwrite the CTM with the matrix at the top of the stack.

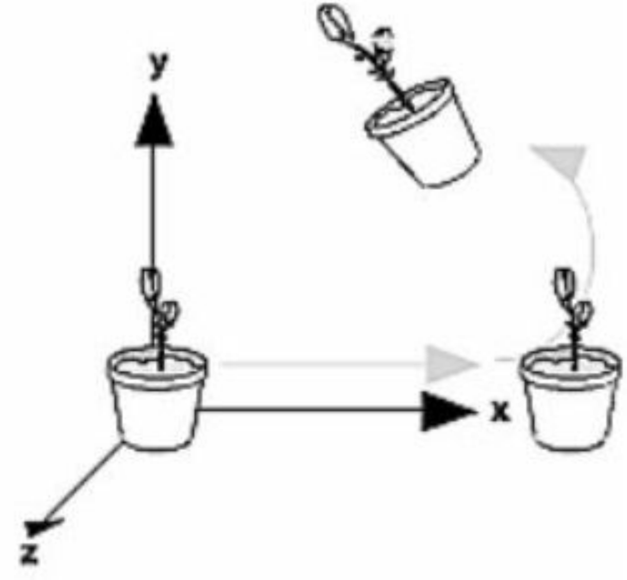
- glLoadIdentity(); –
  - Overwrite the CTM with the identity matrix.

OpenGL

- Assuming that R and T represent rotation and translation matrices, which one of the following scenarios occur after executing the following code in OpenGL?

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glMultMatrixf(T);                 /* translation */
glMultMatrixf(R);                 /* rotation */
draw_the_object();
```





OpenGL

- What happens here?

```
glLoadIdentity();
glColor3f(1.0, 1.0, 1.0);
draw_triangle();                  /* solid lines */

glEnable(GL_LINE_STIPPLE);        /* dashed lines */
glLineStipple(1, 0xF0F0);
glLoadIdentity();
glTranslatef(-20.0, 0.0, 0.0);
draw_triangle();

glLineStipple(1, 0xF00F);         /*long dashed lines */
glLoadIdentity();
glScalef(1.5, 0.5, 1.0);
draw_triangle();

glLineStipple(1, 0x8888);         /* dotted lines */

glLoadIdentity();
glRotatef (90.0, 0.0, 0.0,
1.0); draw_triangle ();
glDisable (GL_LINE_STIPPLE);
```

OpenGL

- What happens here?

```
glLoadIdentity();
glColor3f(1.0, 1.0, 1.0);
draw_triangle();                     /* solid lines */

glEnable(GL_LINE_STIPPLE);           /* dashed lines */
glLineStipple(1, 0xF0F0);
glLoadIdentity();
glTranslatef(-20.0, 0.0, 0.0);
draw_triangle();

glLineStipple(1, 0xF00F);            /*long dashed lines */
glLoadIdentity();
glScalef(1.5, 0.5, 1.0);
draw_triangle();

glLineStipple(1, 0x8888);            /* dotted lines */

glLoadIdentity();
glRotatef (90.0, 0.0, 0.0,
1.0); draw_triangle ();
glDisable (GL_LINE_STIPPLE);
```