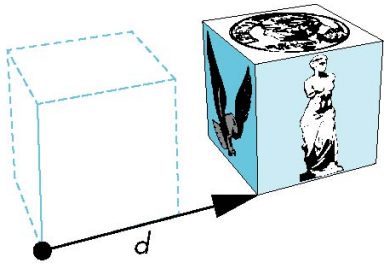
Transformations (Part 1)
A transformation, T, maps points to other points and/or vectors to other vectors.

- Affine Transformations
  - Preserves lines
  - Importance in graphics is that we need only to transform endpoints of line segments and let implementation draw the line segment between the transformed endpoints
  - This is an essential property for all the transformations we discuss in computer graphics

- **Translate**



  - Move (translate, displace) a point, ***p***, to a new location, **q**
  - Displacement determined by a vector ***d: q=p+d***
  - **3 degrees of freedom**

Translation Matrix: $M\_row = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$

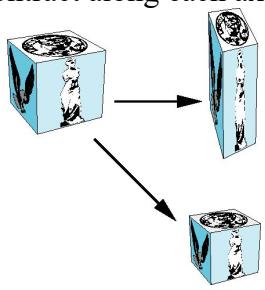
  - Convert to column major order:

$M\_column = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{bmatrix}$

  - How to apply this matrix in translating a point p to q? ➔ post-multiply M by p
  - Practice Examples:
    - Example 1: Suppose a point p=(1, 2, 1, 1) has been translated 2 units along x, 3 units along y, and -2 units along z, what is the translated point q?
    - Example 2: Suppose a point p=(3, 5, 2, 1) has been translated along vector d=(-1, 3, -2, 0) to the new point q, what is q?
    - Example 3: Translate a point p=(-1, 0, 0, 1) to point q=(1, 0, 0, 1) in a sequence of 100 steps → animate the movement of a point moving across the canvas
    - Given a square having its four vertices defined as
      - A: (-0.2, -0.2), B: (-0.2, 0.3), C: (0.3, 0.3), D: (0.3, -0.2)

      Where is the square after it translates along the vector d=(-0.1, -0.1, 0)?
    - Example 4: Animate the movement of a square of size 0.1 that moves diagonally across the canvas, from the lower left corner of the canvas to the upper right corner.
  - WebGL function call?
    - **var t = translate(dx, dy, dz); // t will be assigned the 4x4 translate matrix**

Scale -- Expand or contract along each axis (fixed point of origin)

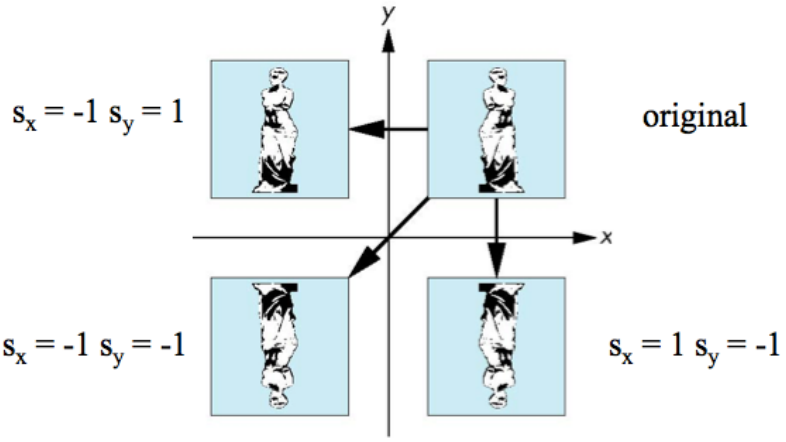


S=

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **p'=S*p**

- Practice Examples: suppose the cubes shown above are all centered at the origin:
  - What was the scaling transformation matrix that transformed the original cube into the top right cube, assuming the top right cube is twice as tall along Y-axis and half as wide along X-axis, and no change along the Z-axis?
  - What was the scaling transformation matrix that transformed the original cube into the lower right cube, assuming the lower right cube is ½ of size as before along X, Y and Z axis?
- Scaling and reflection



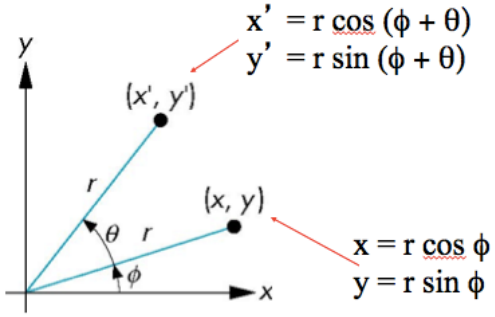
- WebGL function call?
  - var s= **scale4(sx, sy, sz) // s will be assigned 4x4 scaling matrix**

```
function scale4(sx, sy, sz) {
   var s=mat4();
    s[0][0]=sx;
    s[1][1]=sy;
    s[2][2]=sz;

    return s;
}
```

- **Rotation**
  The direction of **rotation** can be **clockwise** (cw) or **counterclockwise** (ccw). ... **Counterclockwise** is the **positive rotation** direction and **clockwise** is the negative direction.

  - Rotation about the origin by θ degrees along z-axis, radius stays the same, angle increases by θ



$cos(\theta + \Phi) = cos(\theta)\ cos(\Phi) - sin(\theta)\ sin(\Phi)$
$sin(\theta + \Phi) = sin(\theta)\ cos(\Phi) + cos(\theta)\ sin(\Phi)$
➔

$x' = x \cos\theta - y \sin\theta$
$y' = x \sin\theta + y \cos\theta$
$z' = z$

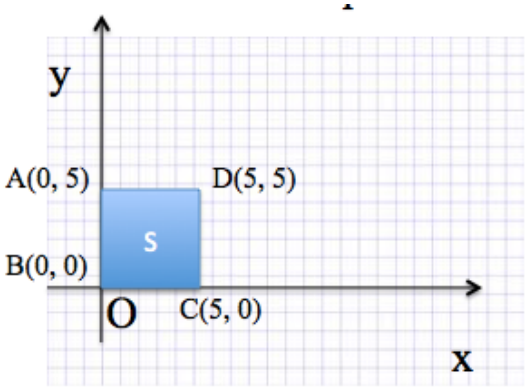
  - Rotation Matrix (all these are rotations about the origin)
    - $\mathbf{R} = \mathbf{R}_z(q) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
    - $\mathbf{R} = \mathbf{R}_x(q) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
    - $\mathbf{R} = \mathbf{R}_y(q) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- Convert to column major order: $\mathbf{R}_z(q)$

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

Practice Examples:

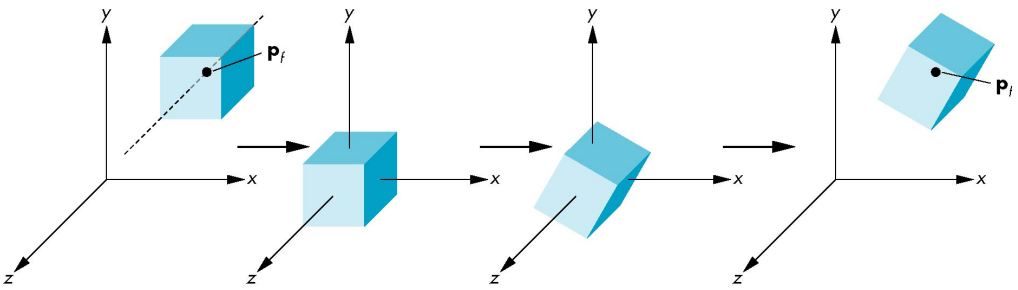
- Example 1: Point B=(1, 1, 1, 1) rotates about the Z-axis around the origin for 30 degrees to point B'. What are the coordinates for point B'?
- Example 2: Segment AB with end points A=(4, 6, 0, 1), B=(1, 2, 0, 1), rotates about the Z-axis around origin for 30 degrees, where is the resulting segment A'B' located?
- Example 3: Point C=(2, 3, 4, 1) rotates about the X-axis around the origin for -25 degrees to point C'. Show the coordinates for point C'.
- Example 4: Given the following square defined with points A, B, C, D. Rotate this square 45 degrees counter-clockwise about the z-axis around the origin. What is the transformation matrix for this rotation? Where will the square end up after the rotation?



  - How to implement rotation in WebGL?
    - var r= rotate (40, 0, 0, 1); // r will be assigned the 4x4 rotation matrix

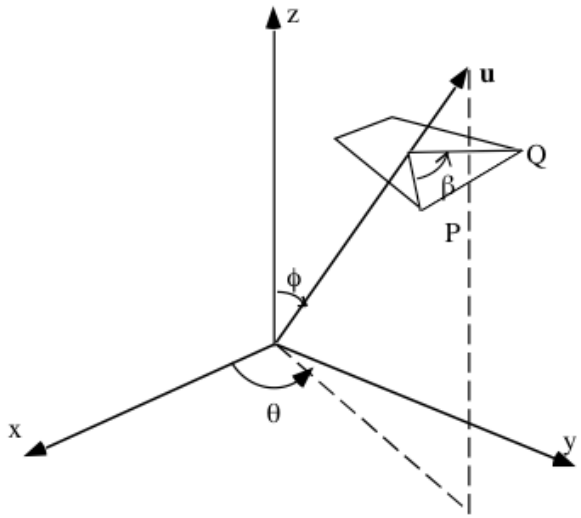
- Rotation about a fixed point other than the origin
  - Steps:
    1. Move fixed point to origin $T(-p_f)$
    2. Rotate about the origin $\mathbf{R}(q)$
    3. Move fixed point back to its original position $\mathbf{T}(p_f)$

    ➔ $\mathbf{M} = \mathbf{T}(p_f)\ \mathbf{R}(q)\ \mathbf{T}(-p_f)$



  - Example 1: Given a point A=(1, 3, 1, 1) rotates about Z-axis for 30 degrees around the point p=(2, 2, 1, 1), and ends at point A'. What are the coordinates of A'?
  - How to implement this in WebGL?
    - What is the corresponding transformation matrix?

- Rotation about an arbitrary vector/axis **u**
  - Rotate point P around vector **u,**
  - Assume the rotation is on the plane perpendicular to to vector u,
  - After the rotation, P ends at Q.



  - *Any* 3D rotation around an axis (passing through the origin) can be obtained from the product of five matrices for the appropriate choice of Euler angles;
    - 3 values (Euler Angles) are required to completely specify a rotation

$$\boldsymbol{R_u(\beta) = R_x(\theta)\, R_y(\Phi)\, R_z(\beta)\, R_y(-\Phi)\, R_x(-\theta)}$$

    - Which sequence of rotations does this represent?
    - Order of transformation, when a composite matrix is used to transform a point, which transformation is performed first? Left to right.
    - The combined rotation matrix:

$$R_u(\beta) = \begin{pmatrix} c+(1-c)u_x^2 & (1-c)u_yu_x - su_z & (1-c)u_zu_x + su_y & 0 \\ (1-c)u_xu_y + su_z & c+(1-c)u_y^2 & (1-c)u_zu_y - su_x & 0 \\ (1-c)u_xu_z - su_y & (1-c)u_yu_z + su_x & c+(1-c)u_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

    c=cos(β), s=sin(β)

  - How to implement this in WebGL? var r = rotate (β, u_x, u_y, u_z);