CAPITULO 2

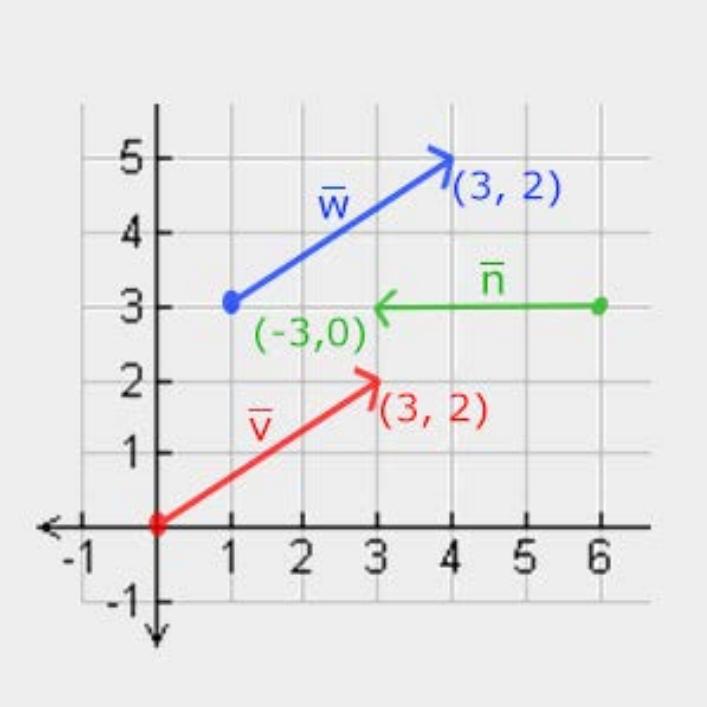
OBJETOS GEOMÉTRICOS Y TRANSFORMACIONES

2.2 Transformaciones Geométricas en 2D

2.2.4 Core Concepts on Transformations 2D/3D

T2D/3D - Vectors

A **Vector** has a direction and a magnitude (also known as its strength or length).



When describing vectors mathematicians generally prefer to describe vectors as character symbols with a little bar over their head like $\bar{v}$.

$$\bar{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

If we want to visualize vectors as positions we can imagine the origin of the direction vector to be (0,0,0)

Using vectors we can thus describe directions and positions in 2D and 3D space

T2D/3D - Vectors

Scalar vector operations

A scalar is a single digit. When adding/subtracting/multiplying or dividing a vector with a scalar we simply add/subtract/multiply or divide each element of the vector by the scalar. For addition it would look like this:

$$\begin{pmatrix}1\\2\\3\end{pmatrix} + x \rightarrow \begin{pmatrix}1\\2\\3\end{pmatrix} + \begin{pmatrix}x\\x\\x\end{pmatrix} = \begin{pmatrix}1+x\\2+x\\3+x\end{pmatrix}$$

Where + can be +,−,· or ÷ where · is the multiplication operator.

Vector negation

Negating a vector results in a vector in the reversed direction.

$$-\bar{v} = -\begin{pmatrix}v_x\\v_y\\v_z\end{pmatrix} = \begin{pmatrix}-v_x\\-v_y\\-v_z\end{pmatrix}$$

T2D/3D - Vectors

Addition and subtraction

Addition of two vectors is defined as **component-wise** addition, that is each component of one vector is added to the same component of the other vector like so

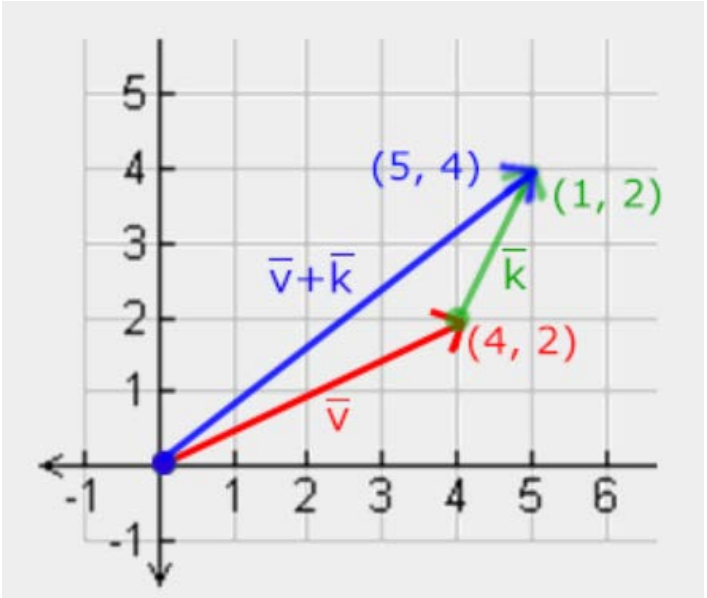
$$\bar{v} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \bar{k} = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix} \rightarrow \bar{v} + \bar{k} = \begin{pmatrix} 1+4 \\ 2+5 \\ 3+6 \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \\ 9 \end{pmatrix}$$

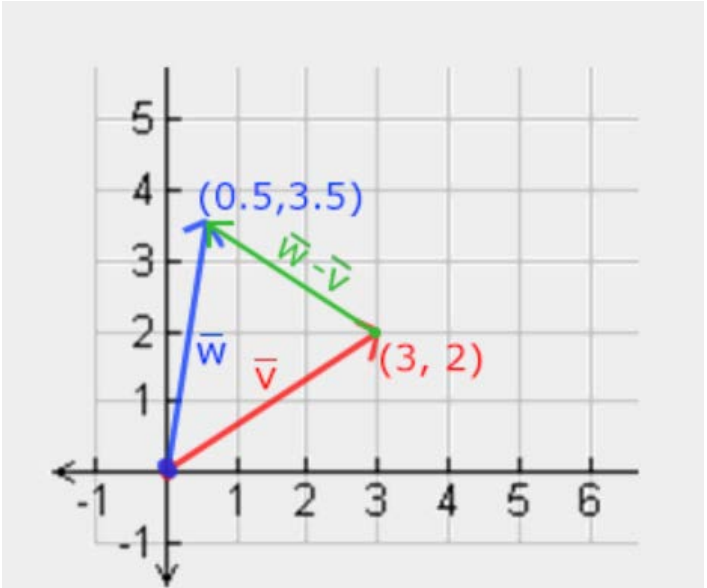
vector subtraction is the same as addition with a negated second vector:

$$\bar{v} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \bar{k} = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix} \rightarrow \bar{v} + -\bar{k} = \begin{pmatrix} 1+(-4) \\ 2+(-5) \\ 3+(-6) \end{pmatrix} = \begin{pmatrix} -3 \\ -3 \\ -3 \end{pmatrix}$$

Example:

v=(4,2) and k=(1,2)





T2D/3D - Vectors

Length

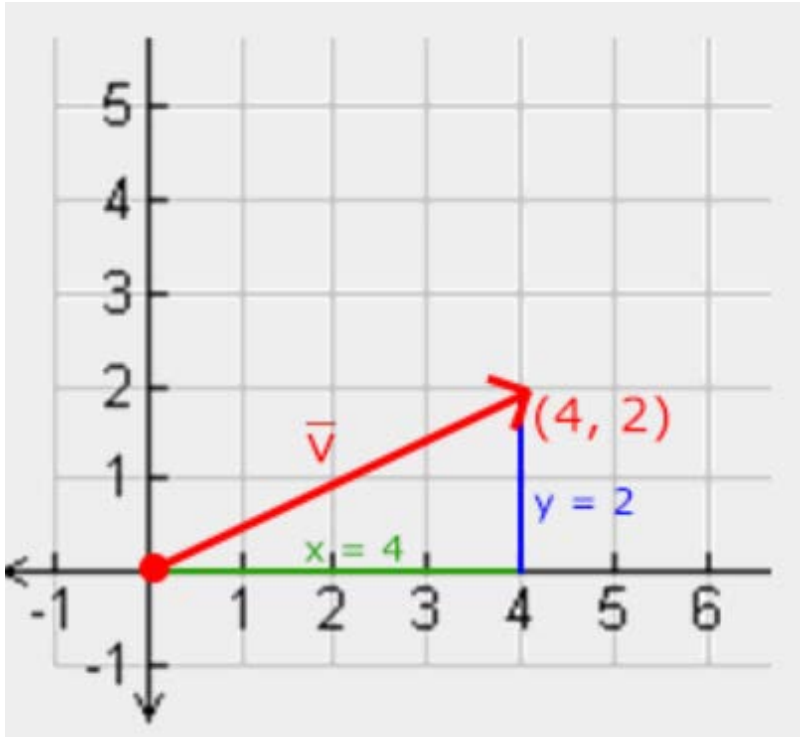
Since the length of the two sides (x, y) are known and we want to know the length of the tilted side $\bar{v}$ we can calculate it using the Pythagoras theorem as

$$||\bar{v}|| = \sqrt{x^2 + y^2}$$

A unit vector has one extra property and that is that its length is exactly 1.

$$\hat{n} = \frac{\bar{v}}{||\bar{v}||}$$

We call this normalizing a vector.



T2D/3D – Vectors

Vector-vector multiplication

Dot product

$$\bar{v} \cdot \bar{k} = ||\bar{v}|| \cdot ||\bar{k}|| \cdot \cos\theta$$

if v¯ and k¯ are unit vectors: $\hat{v} \cdot \hat{k} = 1 \cdot 1 \cdot \cos\theta = \cos\theta$

Now the dot product **only** defines the angle between both vectors.

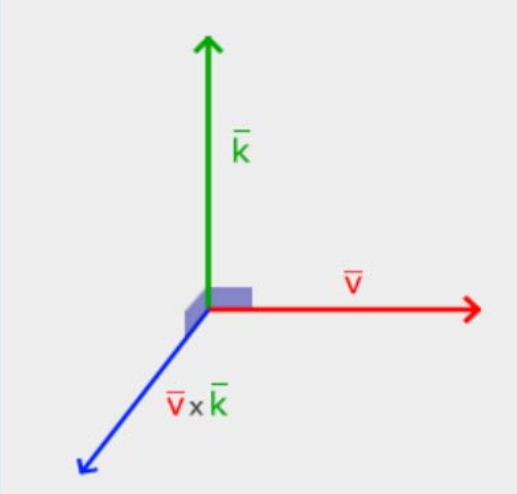
So how do we calculate the dot product?

$$\begin{pmatrix} 0.6 \\ -0.8 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = (0.6 * 0) + (-0.8 * 1) + (0 * 0) = -0.8$$

To calculate the degree between both these unit vectors we use the inverse of the cosine function cos−1 and this results in 143.1 degrees. We now effectively calculated the angle between these two vectors.

Cross product

The cross product is only defined in 3D space and takes two non-parallel vectors as input and produces a third vector that is orthogonal to both the input vectors.



$$\begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix} \times \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix} = \begin{pmatrix} A_y \cdot B_z - A_z \cdot B_y \\ A_z \cdot B_x - A_x \cdot B_z \\ A_x \cdot B_y - A_y \cdot B_x \end{pmatrix}$$

you'll get another vector that is orthogonal to your input vectors.

T2D/3D – Matrices

Addition and subtraction

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 \\ 1 & 6 \end{bmatrix} - \begin{bmatrix} 2 & 4 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 4-2 & 2-4 \\ 1-0 & 6-1 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 1 & 5 \end{bmatrix}$$

Matrix-scalar products

$$2 \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 & 2 \cdot 2 \\ 2 \cdot 3 & 2 \cdot 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

Matrix-matrix multiplication

- You can only multiply two matrices if the number of columns on the left-hand side matrix is equal to the number of rows on the right-hand side matrix.
- Matrix multiplication **is not commutative** that is **A·B≠B·A**

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 & 0 \\ 0 & 8 & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 4 & 2 & 1 \\ 2 & 0 & 4 \\ 9 & 4 & 2 \end{bmatrix} = \begin{bmatrix} 4 \cdot 4 + 2 \cdot 2 + 0 \cdot 9 & 4 \cdot 2 + 2 \cdot 0 + 0 \cdot 4 & 4 \cdot 1 + 2 \cdot 4 + 0 \cdot 2 \\ 0 \cdot 4 + 8 \cdot 2 + 1 \cdot 9 & 0 \cdot 2 + 8 \cdot 0 + 1 \cdot 4 & 0 \cdot 1 + 8 \cdot 4 + 1 \cdot 2 \\ 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 9 & 0 \cdot 2 + 1 \cdot 0 + 0 \cdot 4 & 0 \cdot 1 + 1 \cdot 4 + 0 \cdot 2 \end{bmatrix}$$

$$= \begin{bmatrix} 20 & 8 & 12 \\ 25 & 4 & 34 \\ 2 & 0 & 4 \end{bmatrix}$$

T2D/3D – Matrix-Vector multiplication

Identity matrix

In **OpenGL** we usually work with **4x4 transformation matrices** for several reasons and one of them is that most of the vectors are of size 4. The most simple transformation matrix that we can think of is the identity matrix.

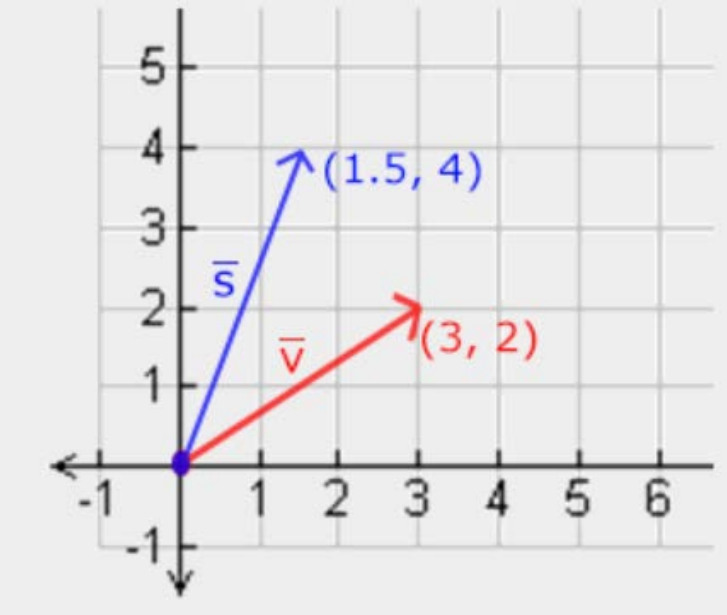
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 \\ 1 \cdot 2 \\ 1 \cdot 3 \\ 1 \cdot 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

You might be wondering what the use is of a transformation matrix that does not transform? The identity matrix is usually a starting point for generating other transformation matrices

T2D/3D – Scaling

When we're scaling a vector we are **increasing the length of the arrow** by the amount we'd like to scale, **keeping its direction the same.**

Let's try scaling the vector **$\bar{v}$=(3,2)**. We will scale the vector along the **x-axis by 0.5**, thus making it twice as narrow; and we'll scale the vector by **2 along the y-axis**,



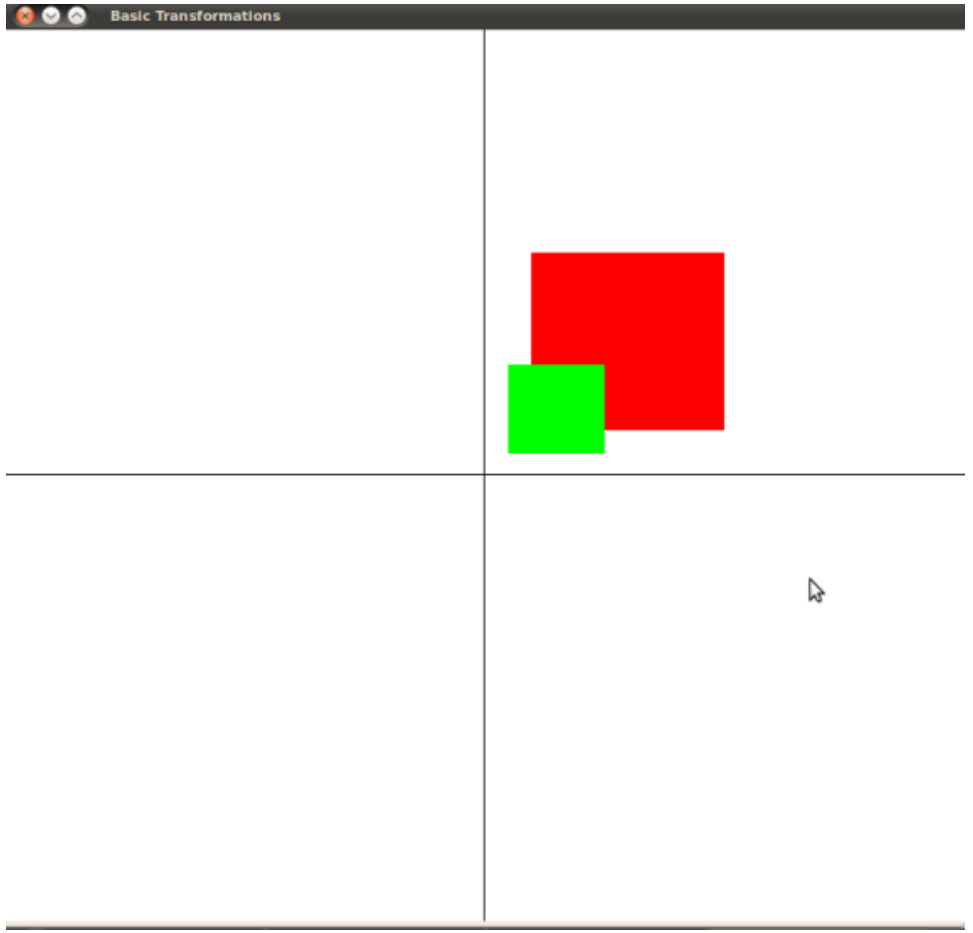
- Keep in mind that **OpenGL usually operates in 3D space** so for this 2D case we could set the **z-axis scale to 1**, leaving it unharmed.
- The scaling operation we just performed is a **non-uniform scale**, because the scaling factor is not the same for each axis.
- If the scalar would be equal on all axes it would be called a **uniform scale**.

T2D/3D – Scaling

If we represent the scaling variables as **(S1,S2,S3)** we can define a scaling matrix on any vector **(x,y,z)** as:

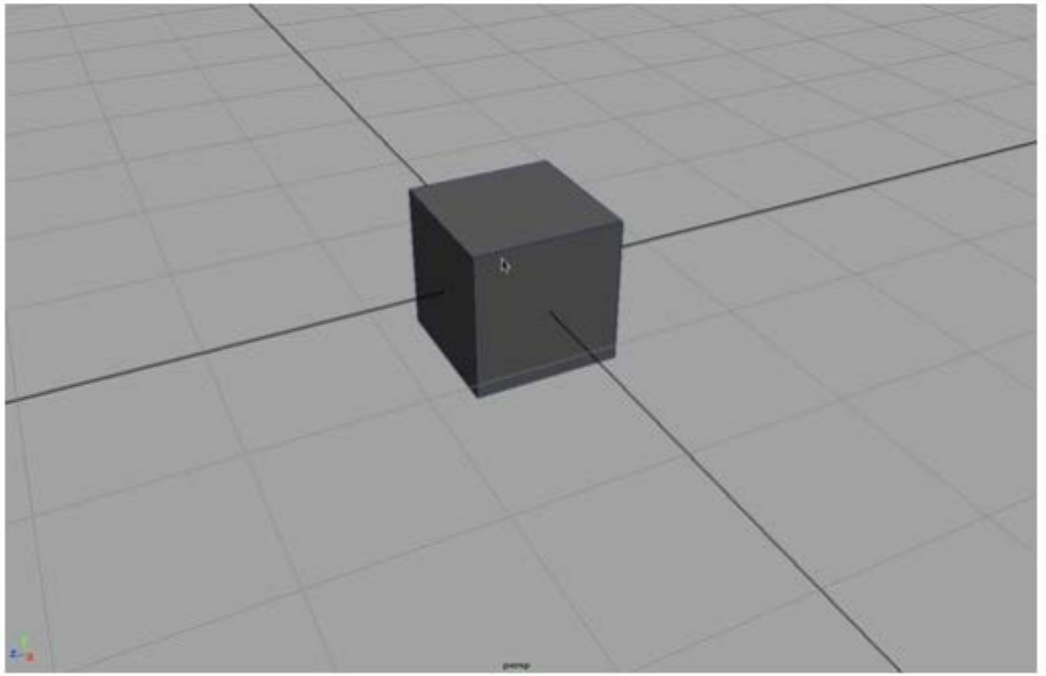
$$\begin{bmatrix} S_1 & 0 & 0 & 0 \\ 0 & S_2 & 0 & 0 \\ 0 & 0 & S_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} S_1 \cdot x \\ S_2 \cdot y \\ S_3 \cdot z \\ 1 \end{pmatrix}$$

Note that we keep the 4th scaling value 1. The w component is used for other purposes as we'll see later on.



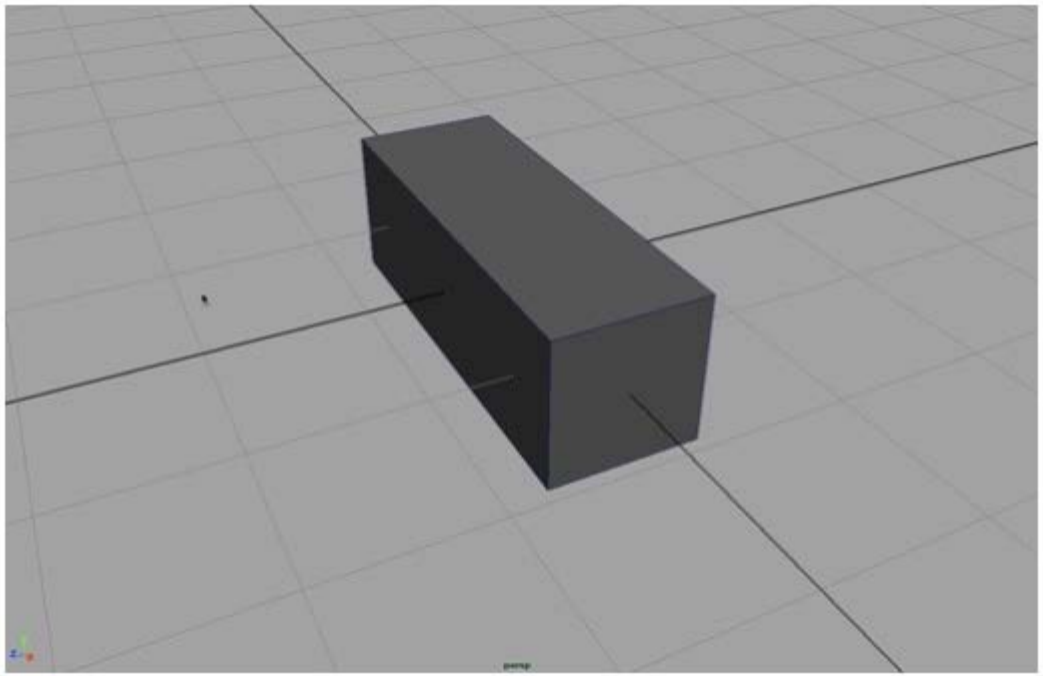
T2D/3D – Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



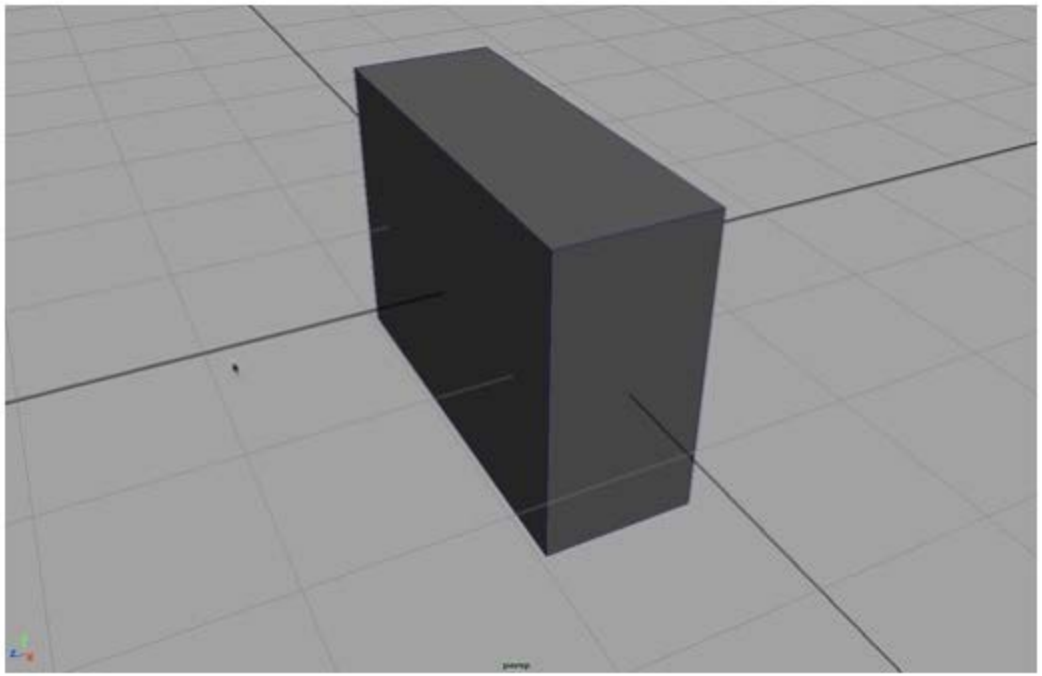
T2D/3D – Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



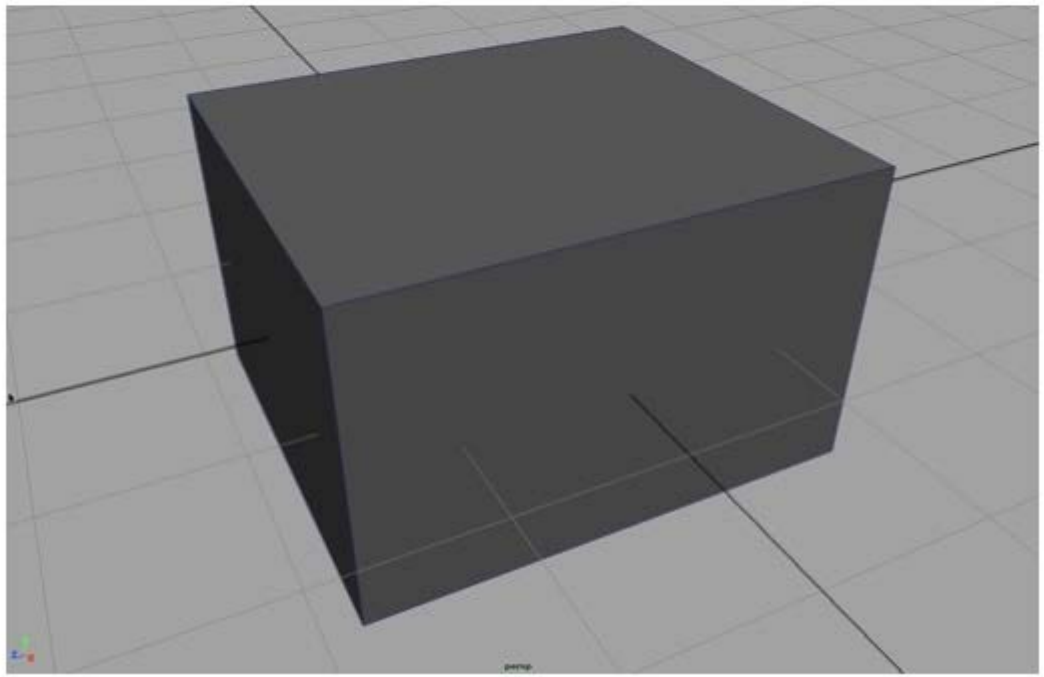
T2D/3D – Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



T2D/3D – Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



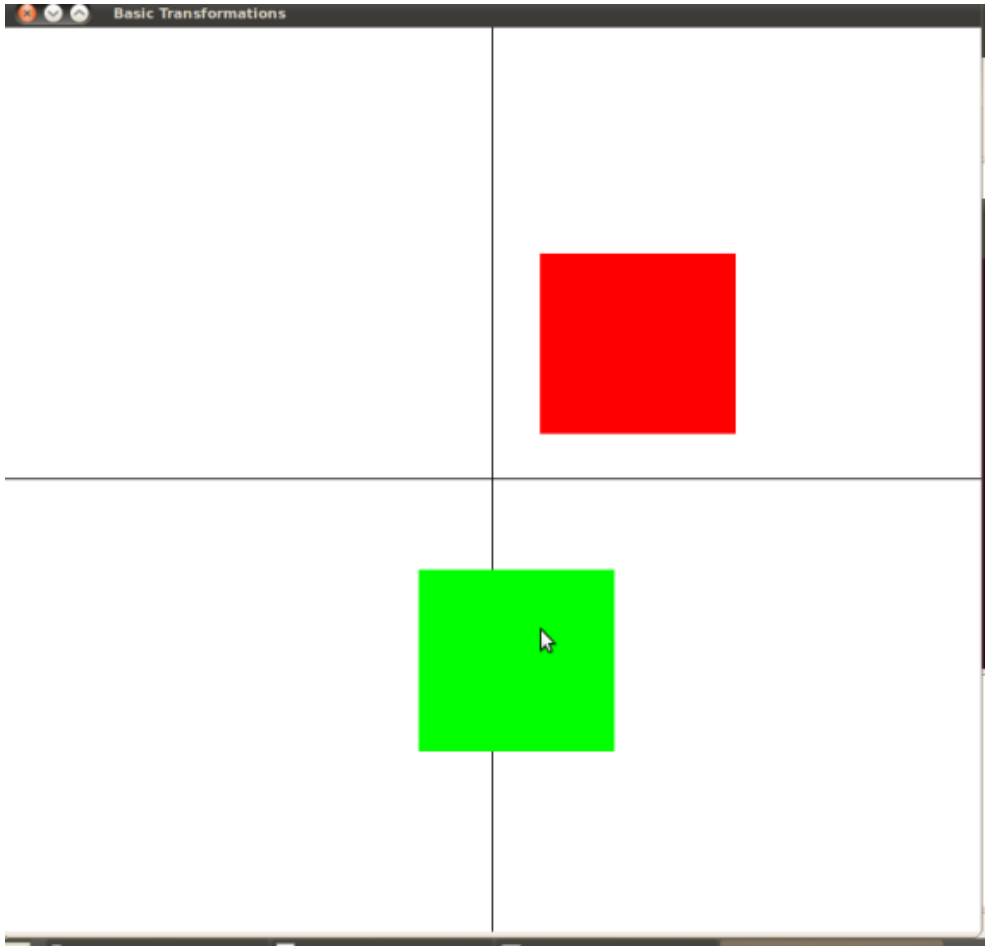
T2D/3D – Translation

Translation is the process of adding another vector on top of the original vector to return a new vector with a different position, thus moving the vector based on a translation vector → vector addition

Just like the scaling matrix there are several locations on a **4-by-4 matrix** that we can use to perform certain operations and for translation those are the **top-3 values of the 4th column**. If we represent the translation vector as **(Tx,Ty,Tz)** we can define the translation matrix by:

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix}$$

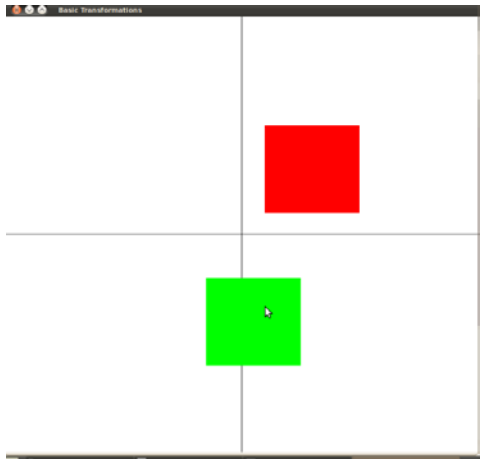
This wouldn't have been possible with a 3-by-3 matrix.



T2D/3D – Translation

With a translation matrix we can move objects in any of the 3 axis directions (x, y, z), making it a very useful transformation matrix for our transformation toolkit.

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix}$$



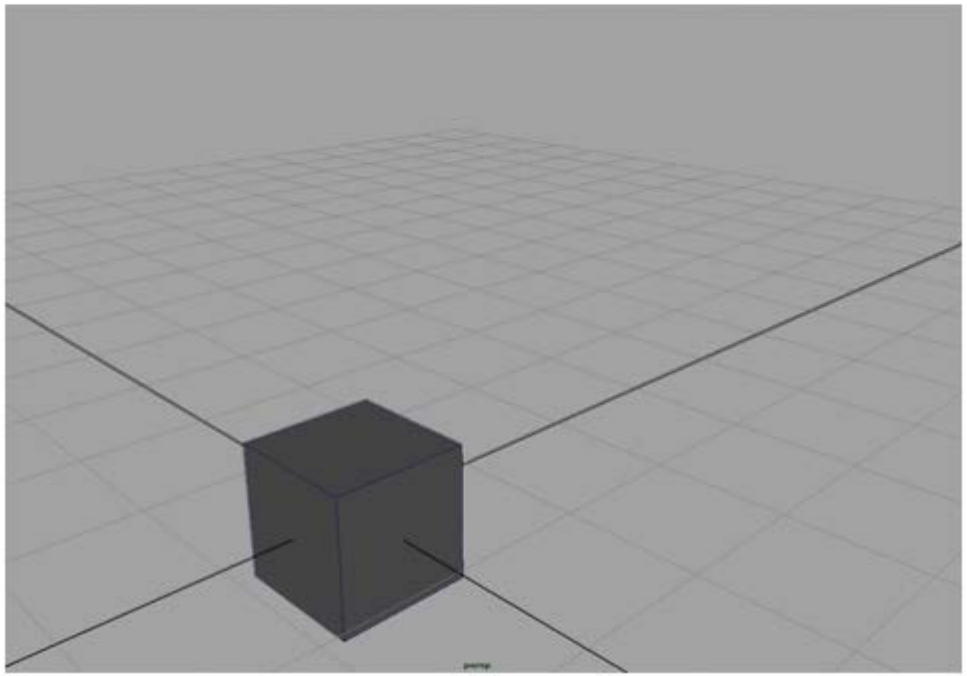
Homogeneous coordinates

The `w` component of a vector is also known as a homogeneous coordinate. To get the 3D vector from a homogeneous vector we divide the `x`, `y` and `z` coordinate by its `w` coordinate. We usually do not notice this since the `w` component is `1.0` most of the time. Using homogeneous coordinates has several advantages: it allows us to do matrix translations on 3D vectors (without a `w` component we can't translate vectors) and in the next chapter we'll use the `w` value to create 3D perspective.

Also, whenever the homogeneous coordinate is equal to `0`, the vector is specifically known as a direction vector since a vector with a `w` coordinate of `0` cannot be translated.

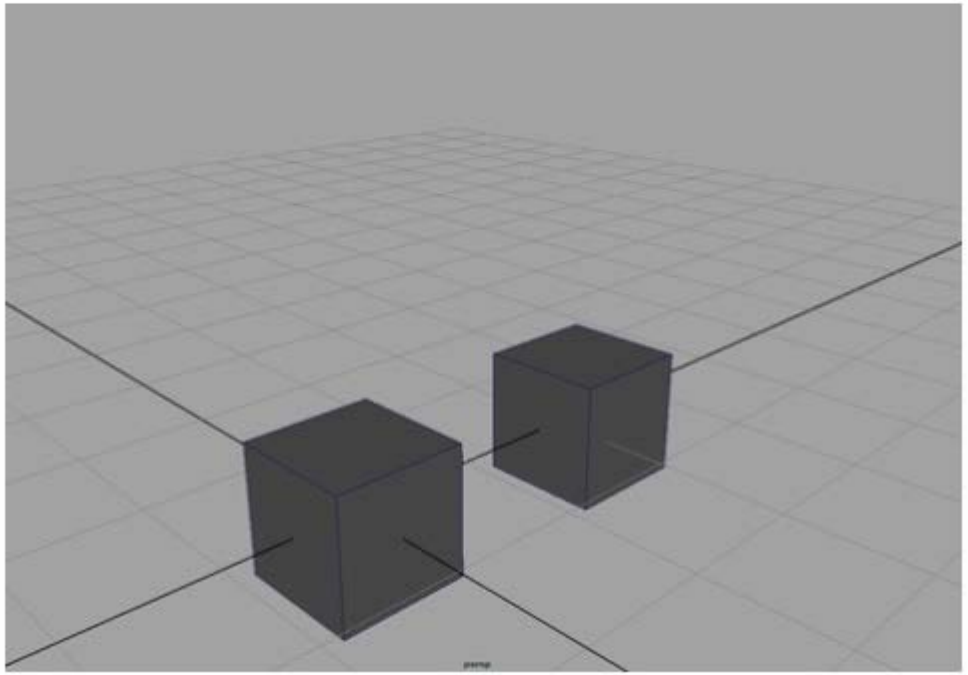
T2D/3D – Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



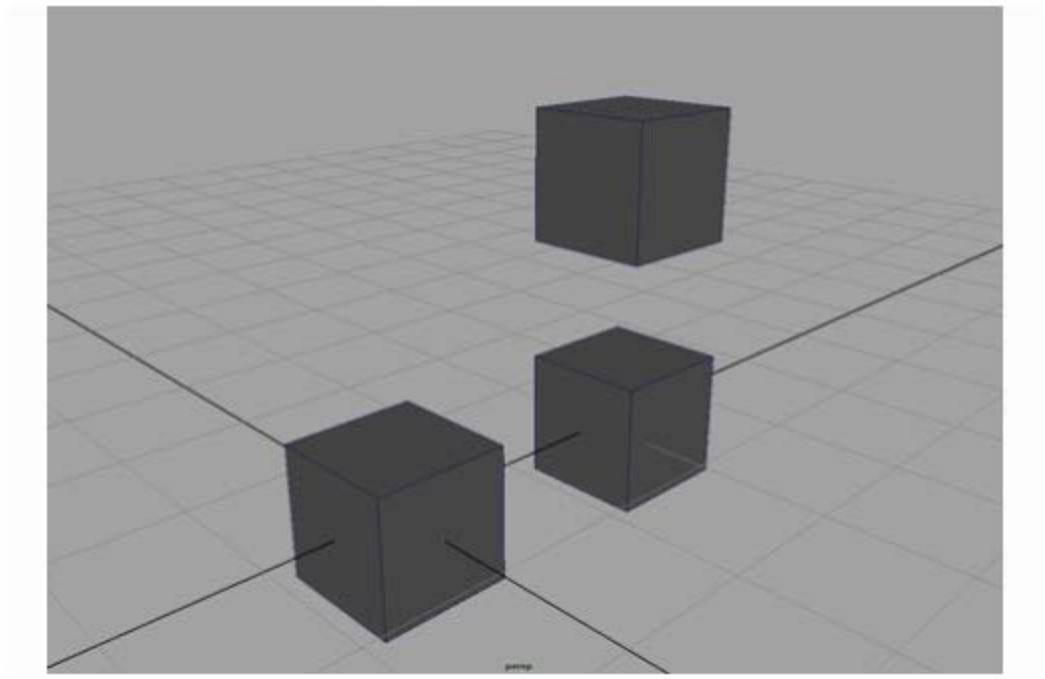
T2D/3D – Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



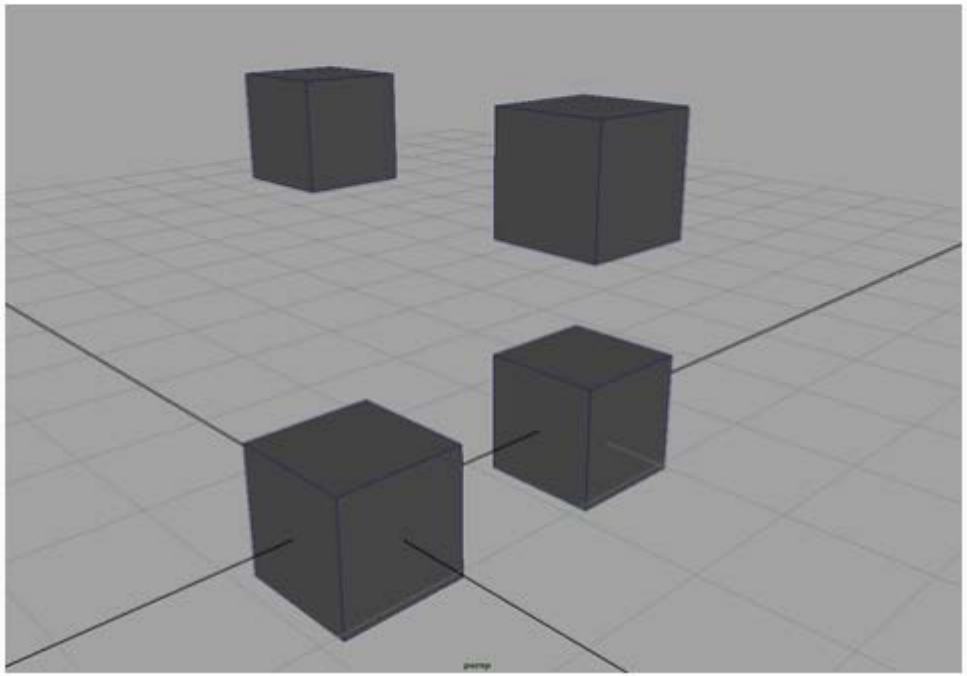
T2D/3D – Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



T2D/3D – Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

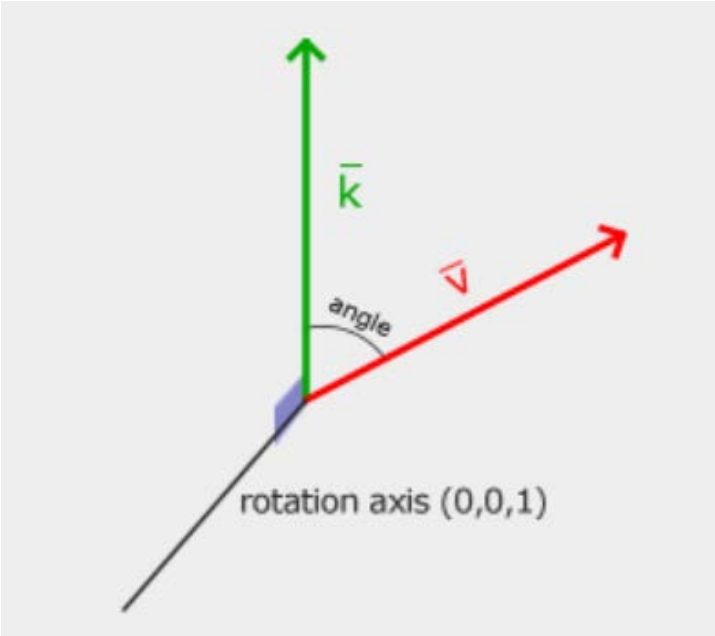


T2D/3D – Rotation

A **rotation** in 2D or 3D is represented with an angle. An angle could be in degrees or radians where a whole circle has 360 degrees or 2 PI radians.

Example:

Rotating half a circle rotates us 360/2 = 180 degrees and rotating 1/5th to the right means we rotate 360/5 = 72 degrees to the right. This is demonstrated for a basic 2D vector where **$\bar{v}$ is rotated 72 degrees to the right,** or clockwise, from **$\bar{k}$:**



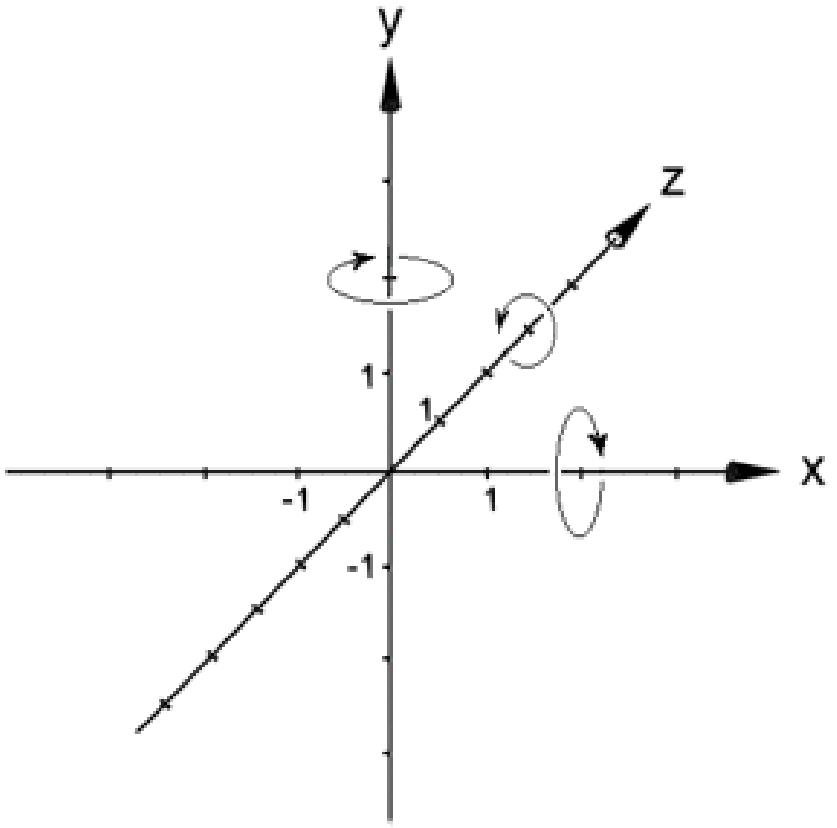
Most rotation functions require an angle in radians, but luckily degrees are easily converted to radians:

```
angle in degrees = angle in radians * (180 / PI)
angle in radians = angle in degrees * (PI / 180)
```

Where `PI` equals (rounded) `3.14159265359`.

T2D/3D – Rotation

- Rotations in 3D are specified with an angle and a rotation axis. The angle specified will rotate the object along the rotation axis given.
- When rotating 2D vectors in a 3D world for example, we set the rotation axis to the z-axis (try to visualize this).



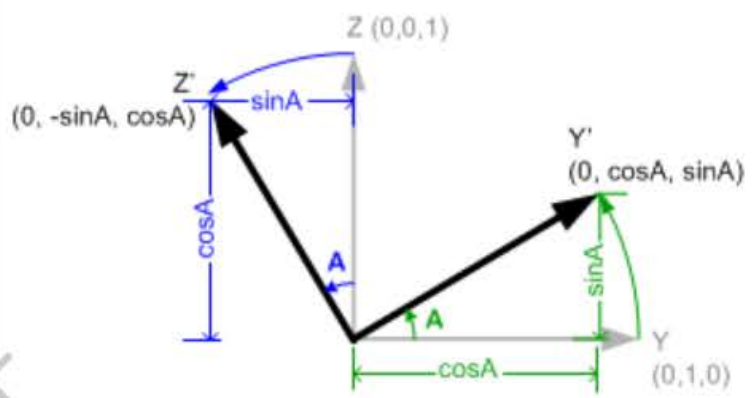
Left handed system of coordinates with positive sense of rotation

- Using trigonometry it is possible to transform vectors to newly rotated vectors given an angle.
- This is usually done via a smart combination of the **sine** and **cosine** functions (commonly abbreviated to **sin** and **cos**).

T2D/3D – Rotation

A rotation matrix is defined for each unit axis in 3D space where the angle is represented as the theta symbol Θ

Rotation about Left(X) Axis (Pitch)



Rotation about Left(X) Axis

Initial value of up(Y) and forward(Z) axes are (0, 1, 0) and (0, 0, 1). If left(X) axis rotates A degree, then new up(Y') axis becomes (0, cosA, sinA) and forward(Z') becomes (0, -sinA, cosA). The new axes are inserted as column components of the 3x3 rotation matrix. Then, the rotation matrix becomes;

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos A & -\sin A \\ 0 & \sin A & \cos A \end{pmatrix}$$

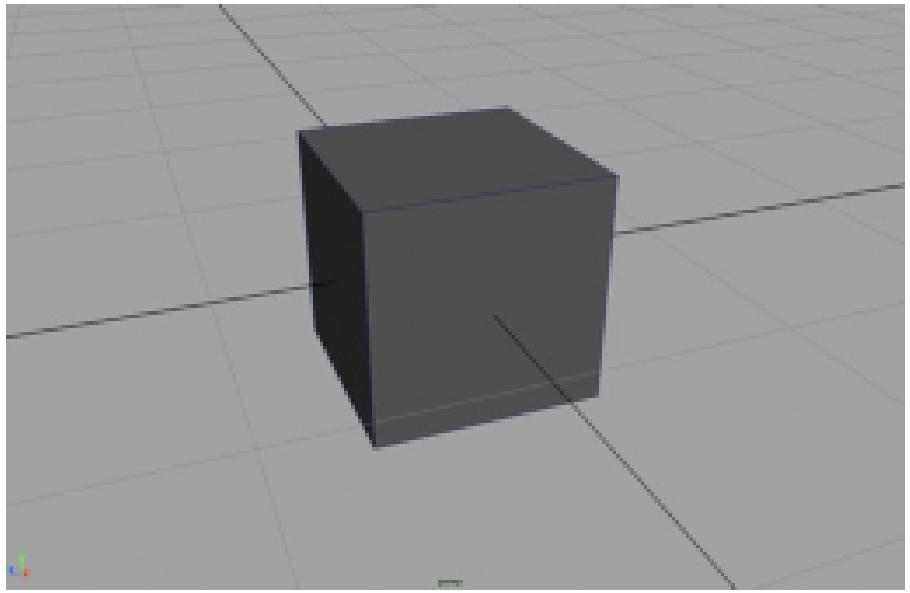
Rotation around the X-axis:

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

T2D/3D – Rotation

Rotation about **x** axis

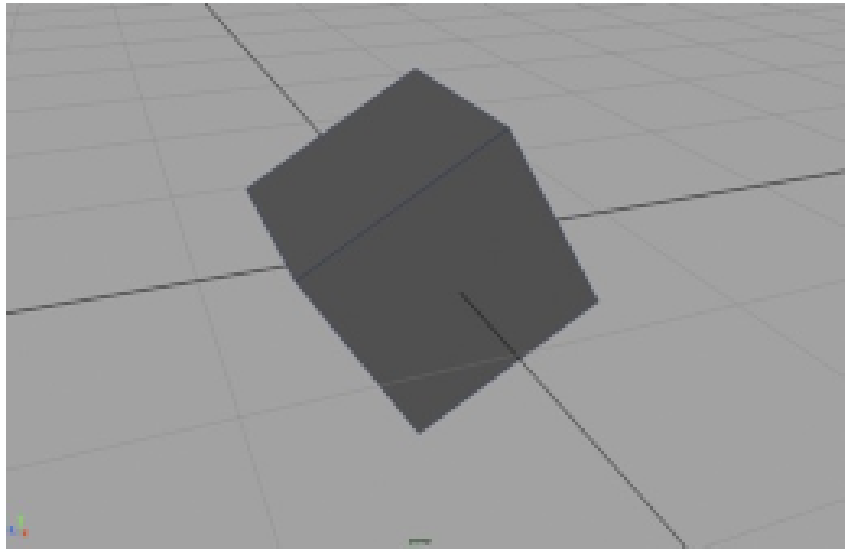
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



T2D/3D – Rotation

Rotation about **x** axis

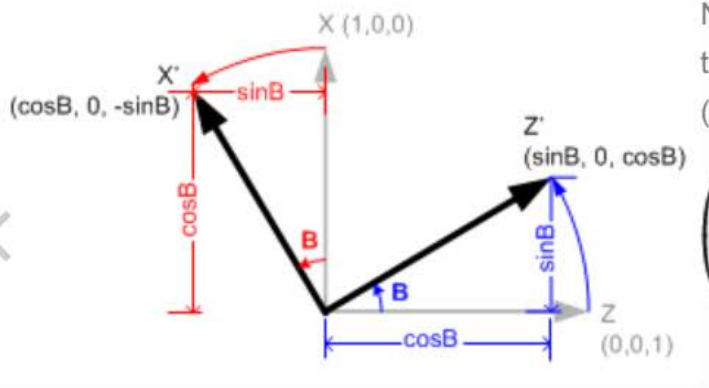
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



T2D/3D – Rotation

A rotation matrix is defined for each unit axis in 3D space where the angle is represented as the theta symbol Θ

Rotation about Up(Y) Axis (Yaw, Heading)



Rotation about Up(Y) Axis

Now, we rotate around up vector, which is facing toward you, with B angle . Left(X) axis is transformed from (1, 0, 0) to X' (cosB, 0, -sinB). And forward(Z) axis is from (0, 0, 1) to Z' (sinB, 0, cosB).

$$\begin{pmatrix} \cos B & 0 & \sin B \\ 0 & 1 & 0 \\ -\sin B & 0 & \cos B \end{pmatrix}$$

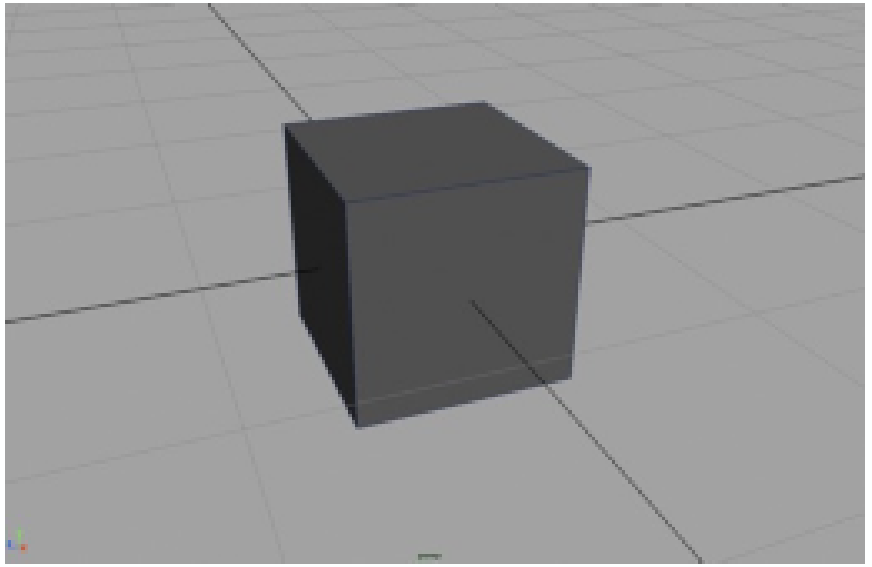
Rotation around the Y-axis:

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

T2D/3D – Rotation

Rotation about **y** axis

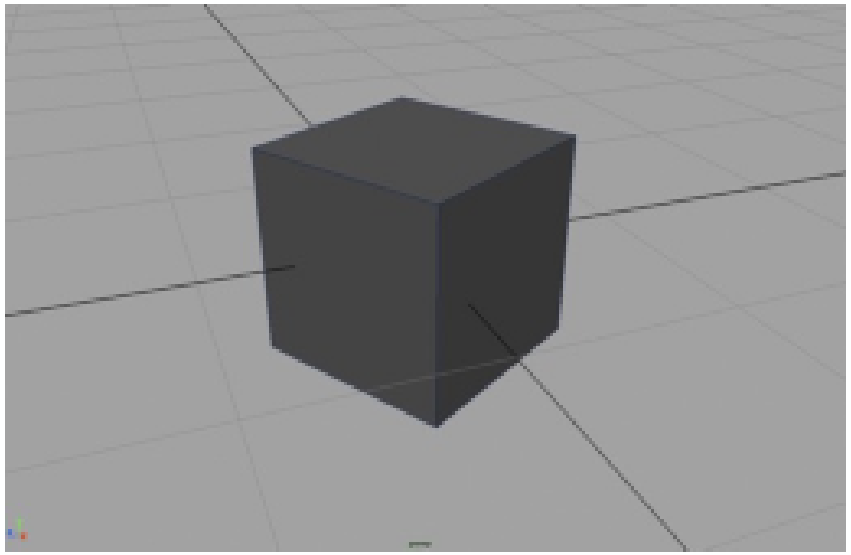
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



T2D/3D – Rotation

Rotation about **y** axis

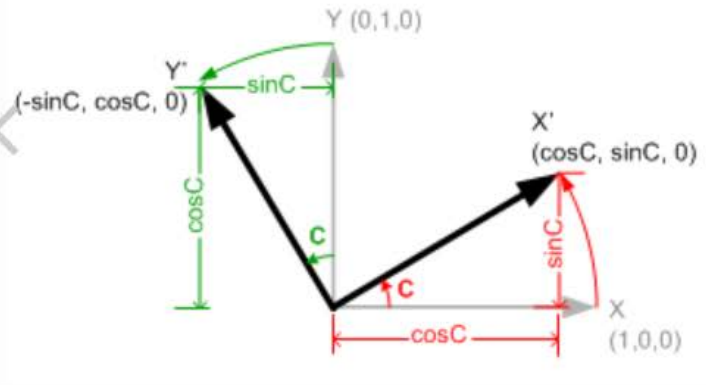
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



T2D/3D – Rotation

A rotation matrix is defined for each unit axis in 3D space where the angle is represented as the theta symbol Θ

Rotation about Forward(Z) Axis (Roll)



Rotation about Forward(Z) Axis

If we rotate forward(Z) axis with angle C degree, the original left(X) (1, 0, 0) axis becomes X' (cosC, sinC, 0), and up(Y) (0, 1, 0) axis becomes Y' (-sinC, cosC, 0).

$$\begin{pmatrix} \cos C & -\sin C & 0 \\ \sin C & \cos C & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

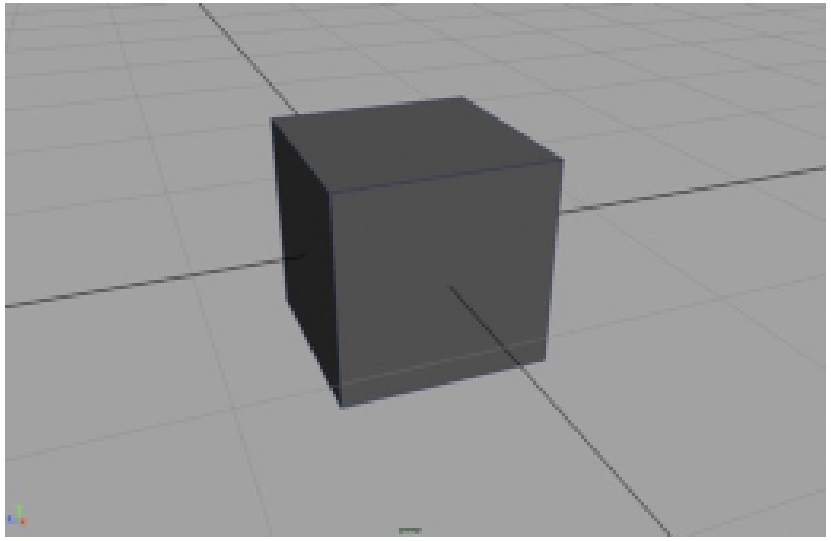
Rotation around the Z-axis:

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

T2D/3D – Rotation

Rotation about **z** axis

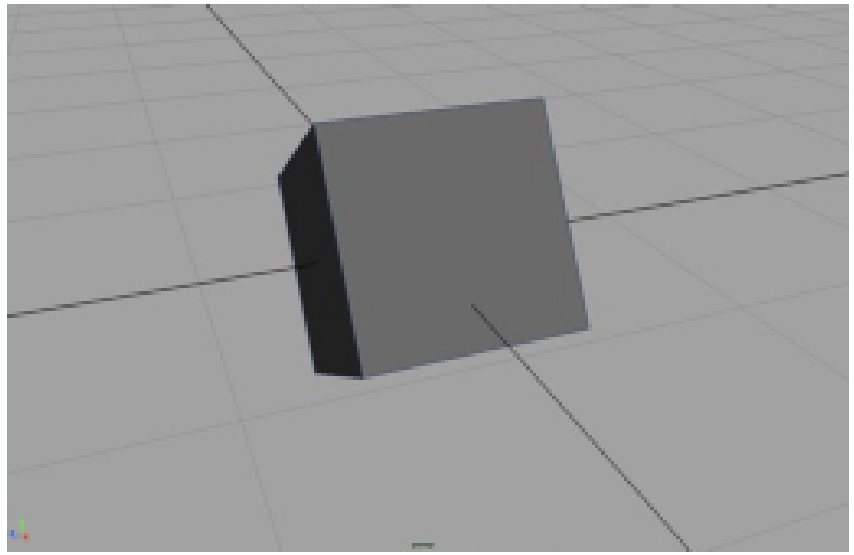
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



T2D/3D – Rotation

Rotation about **z** axis

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



T2D/3D – Rotation

- Using the rotation matrices we can transform our position vectors around one of the three unit axes.
- To rotate around an arbitrary 3D axis we can combine all 3 them by first rotating around the X-axis, then Y and then Z for example.

$$
\begin{aligned}
&R_X R_Y R_Z \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos A & -\sin A \\ 0 & \sin A & \cos A \end{pmatrix} \begin{pmatrix} \cos B & 0 & \sin B \\ 0 & 1 & 0 \\ -\sin B & 0 & \cos B \end{pmatrix} \begin{pmatrix} \cos C & -\sin C & 0 \\ \sin C & \cos C & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos B & 0 & \sin B \\ \sin A \sin B & \cos A & -\sin A \cos B \\ -\cos A \sin B & \sin A & \cos A \cos B \end{pmatrix} \begin{pmatrix} \cos C & -\sin C & 0 \\ \sin C & \cos C & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos B \cos C & -\cos B \sin C & \sin B \\ \sin A \sin B \cos C + \cos A \sin C & -\sin A \sin B \sin C + \cos A \cos C & -\sin A \cos B \\ -\cos A \sin B \cos C + \sin A \sin C & \cos A \sin B \sin C + \sin A \cos C & \cos A \cos B \end{pmatrix}
\end{aligned}
$$

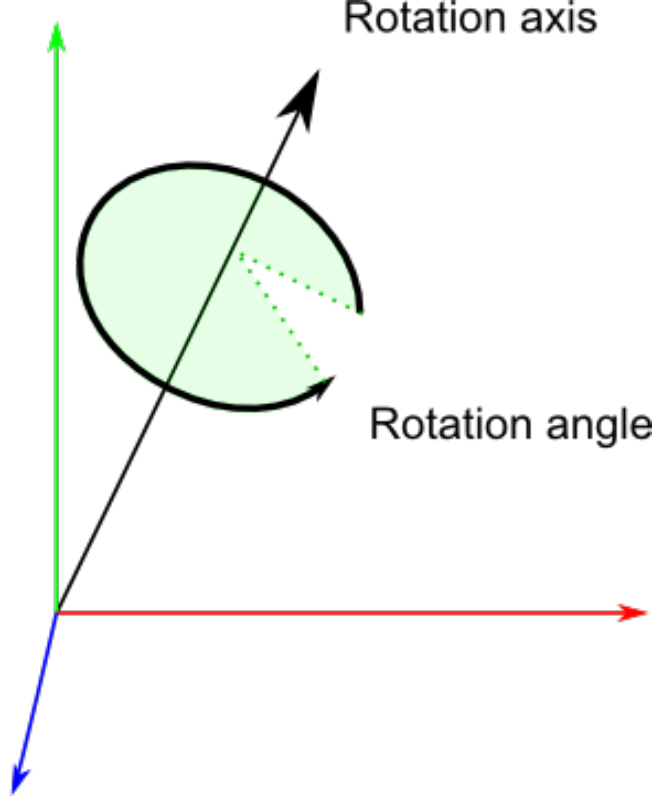
However, this quickly introduces a problem called **Gimbal lock**.

https://www.youtube.com/watch?v=zc8b2Jo7mno

T2D/3D – Rotation

- A better solution is to rotate around an arbitrary unit axis **e.g. (0.662,0.2,0.722) (note that this is a unit vector)** right away instead of combining the rotation matrices.

- Such a (verbose) matrix exists and is given below with **(Rx,Ry,Rz)** as the arbitrary rotation axis:



$$\begin{bmatrix} \cos\theta + {R_x}^2(1-\cos\theta) & R_xR_y(1-\cos\theta) - R_z\sin\theta & R_xR_z(1-\cos\theta) + R_y\sin\theta & 0 \\ R_yR_x(1-\cos\theta) + R_z\sin\theta & \cos\theta + {R_y}^2(1-\cos\theta) & R_yR_z(1-\cos\theta) - R_x\sin\theta & 0 \\ R_zR_x(1-\cos\theta) - R_y\sin\theta & R_zR_y(1-\cos\theta) + R_x\sin\theta & \cos\theta + {R_z}^2(1-\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T2D/3D – Combining Matrices

- The true power from using matrices for transformations is that we can **combine multiple transformations** in a single matrix thanks to matrix-matrix multiplication.

- Let's see if we can generate a **transformation matrix** that combines **several transformations**.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(rx) & -\sin(rx) & 0 \\ 0 & \sin(rx) & \cos(rx) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(ry) & 0 & \sin(ry) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(ry) & 0 & \cos(ry) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -tx \\ 0 & 1 & 0 & -ty \\ 0 & 0 & 1 & -tz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translate by - viewDistance in z direction

Translate origin to rotationCenter

Rotate by rx radians about the x-axis

Rotate by ry radians about the y-axis

Translate rotationCenter to origin

T2D/3D – Combining Matrices

- Say we have a **vector (x,y,z)** and we want to **scale it by 2 and then translate it by (1,2,3**).
- We need a translation and a scaling matrix for our required steps.
- The resulting transformation matrix would then look like:

$$Trans.Scale = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

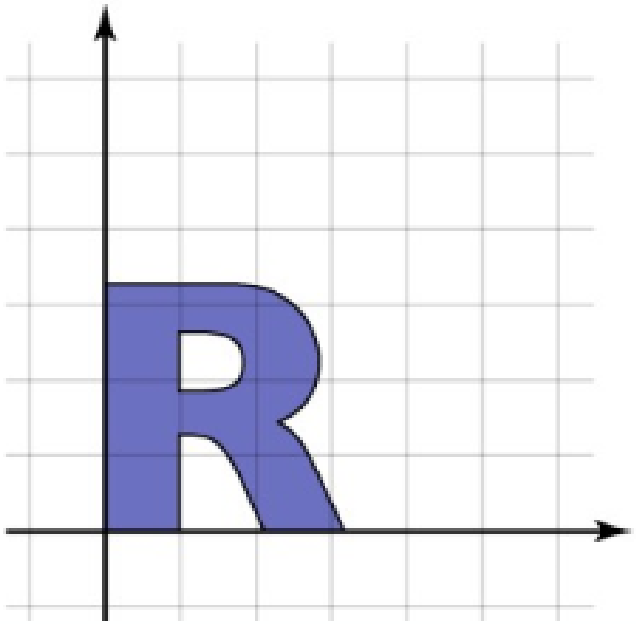
T2D/3D – Combining Matrices

$$Trans.Scale = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

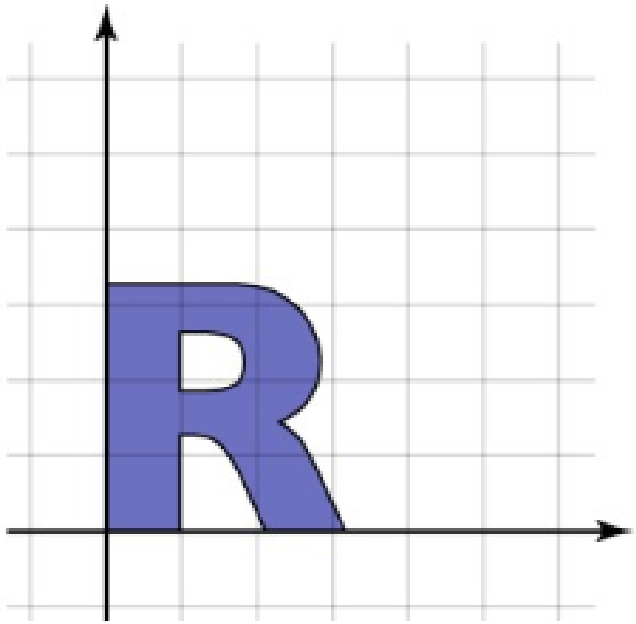
- Note that we first do a translation and then a scale transformation when multiplying matrices.
- **Matrix multiplication is not commutative**, which means **their order is important**.
- When multiplying matrices the right-most matrix is first multiplied with the vector so you should read the multiplications from right to left.

T2D/3D – Combining Matrices

- In general **not** commutative: order matters!



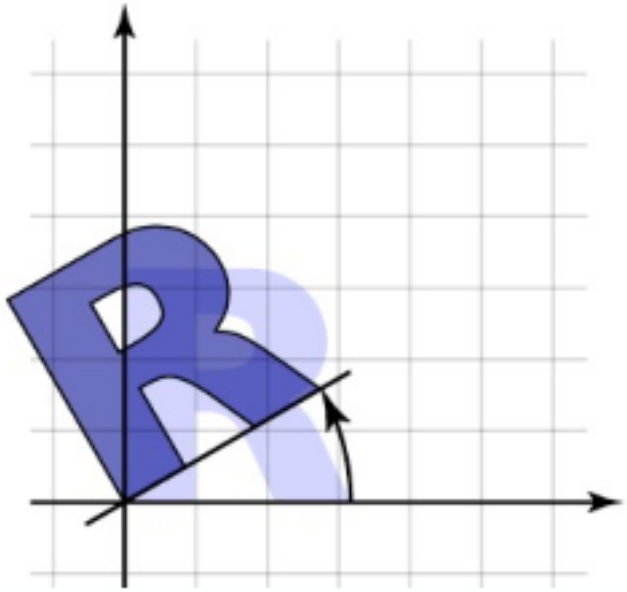
rotate, then translate



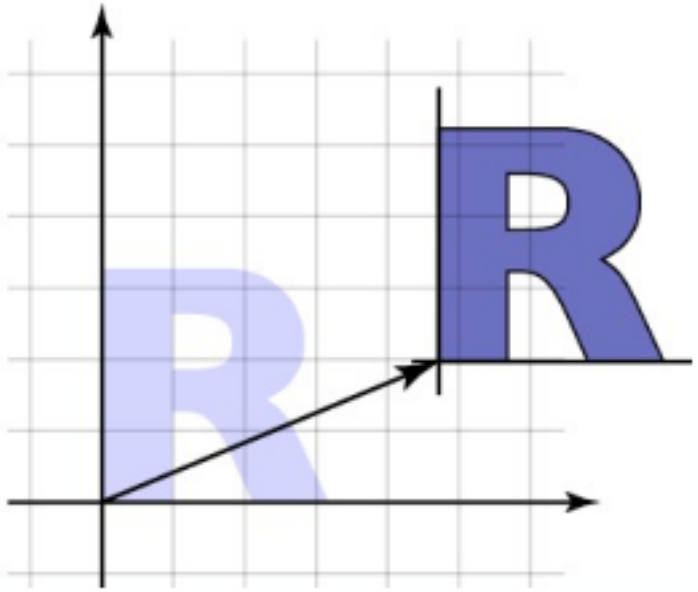
translate, then rotate

T2D/3D – Combining Matrices

- In general **not** commutative: order matters!



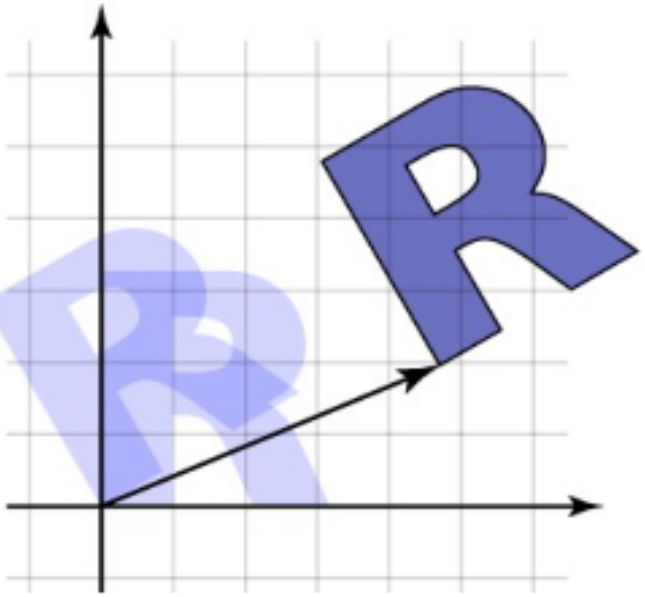
rotate, then translate



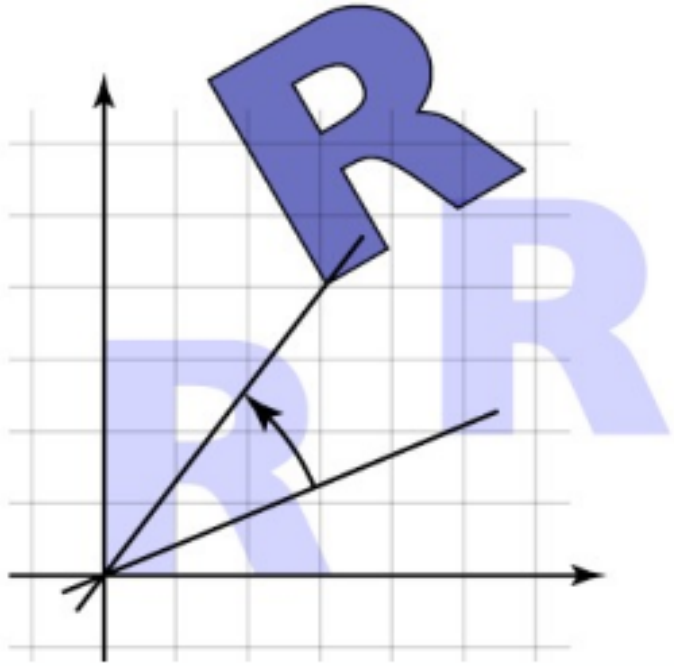
translate, then rotate

T2D/3D – Combining Matrices

- In general **not** commutative: order matters!



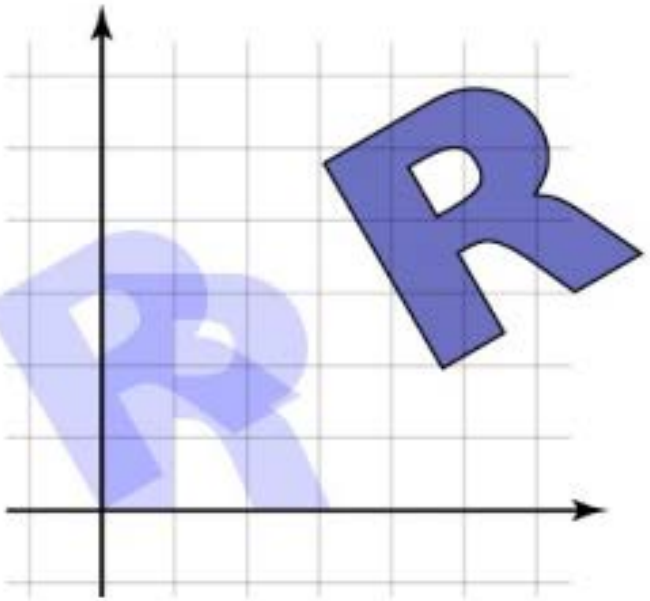
rotate, then translate



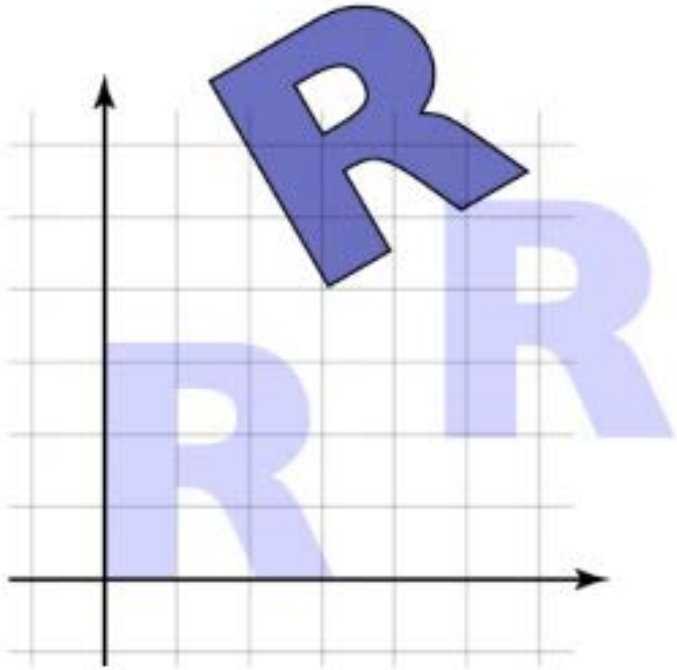
translate, then rotate

T2D/3D – Combining Matrices

- In general **not** commutative: order matters!



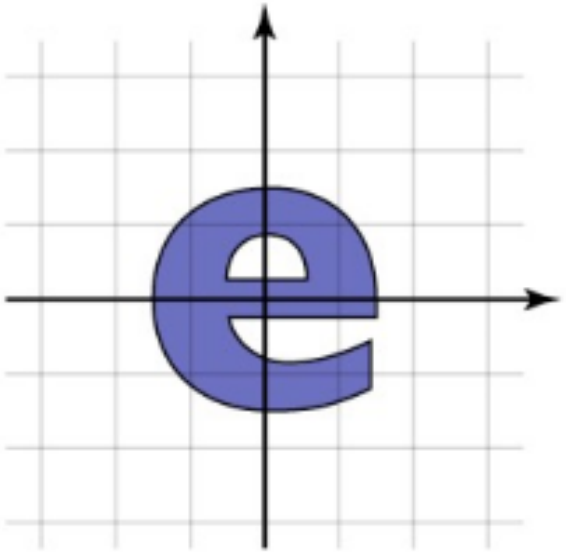
rotate, then translate



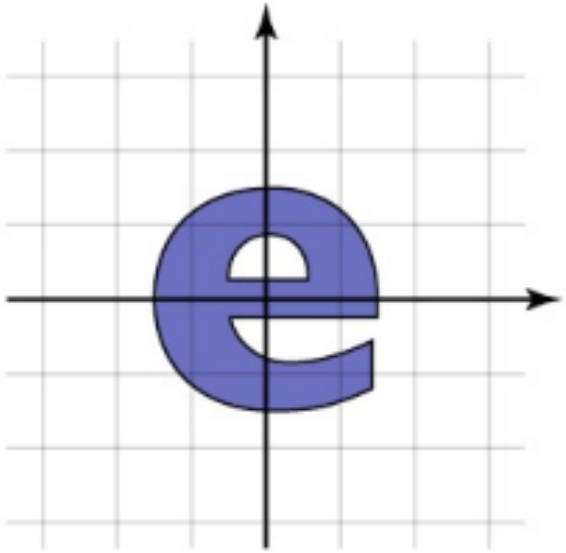
translate, then rotate

T2D/3D – Combining Matrices

- **Another example**



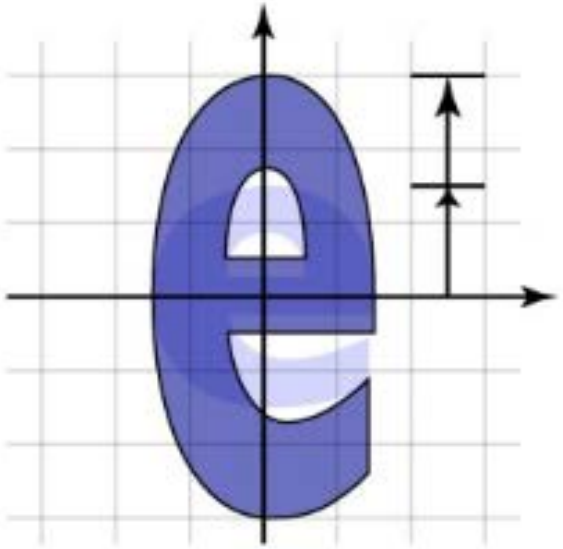
scale, then rotate



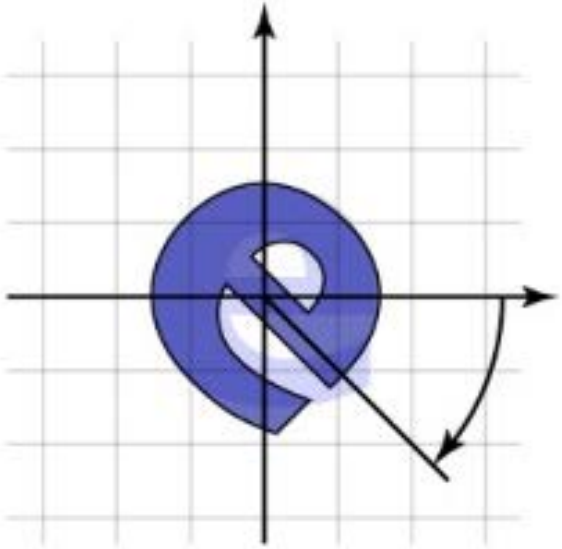
rotate, then scale

T2D/3D – Combining Matrices

- Another example



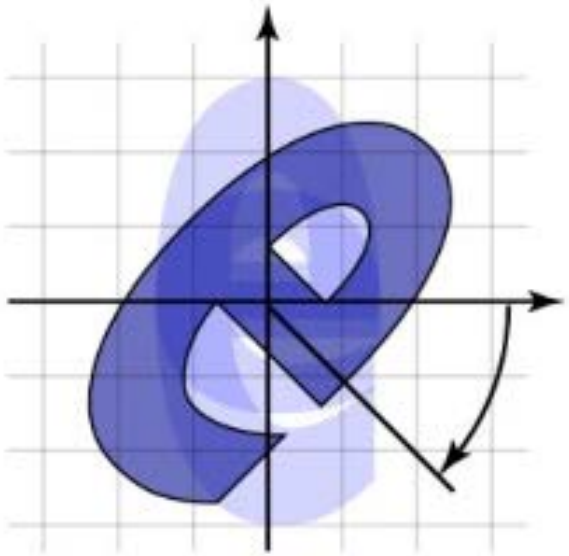
scale, then rotate



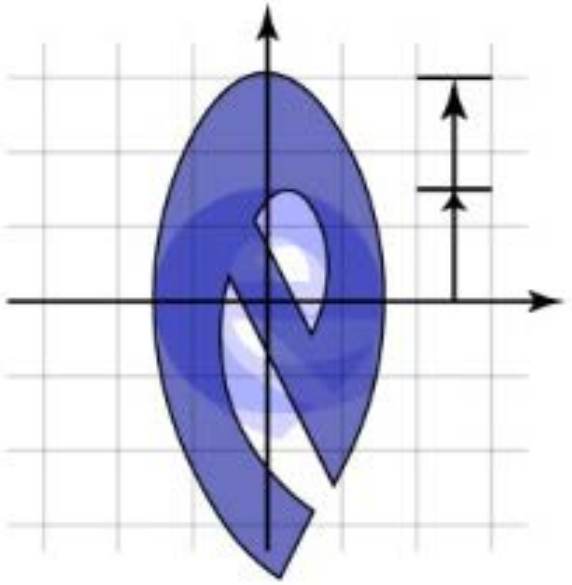
rotate, then scale

T2D/3D – Combining Matrices

- Another example



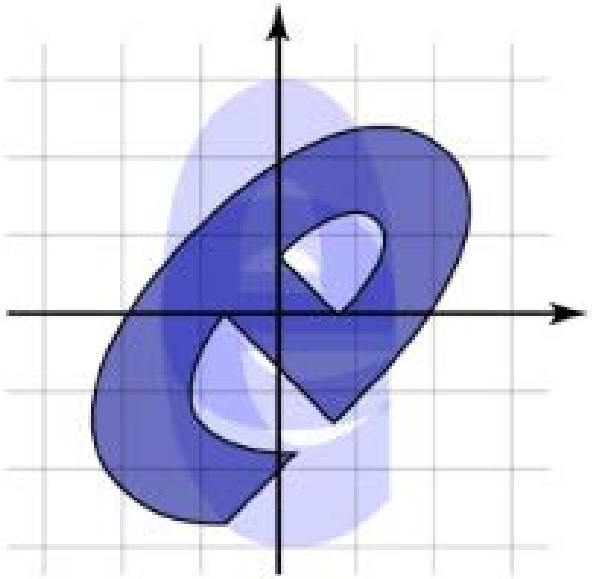
scale, then rotate



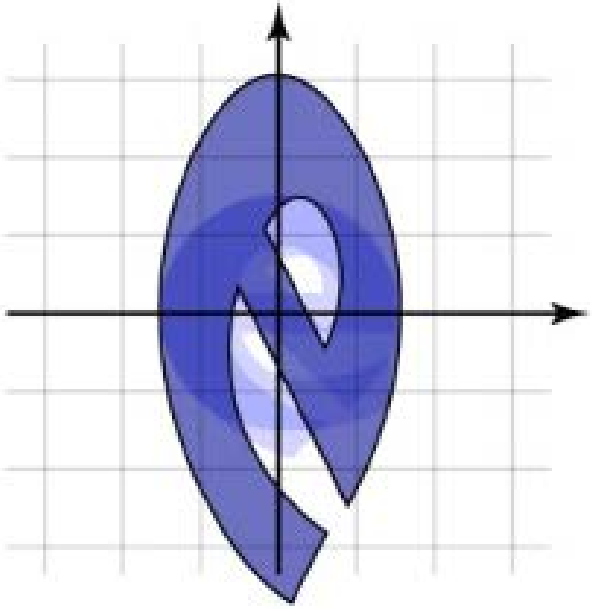
rotate, then scale

T2D/3D – Combining Matrices

- Another example



scale, then rotate



rotate, then scale

T2D/3D – Combining Matrices

$$Trans.Scale = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It is advised to first do (1) scaling operations, then (2) rotations and lastly (3) translations when combining matrices otherwise they might (negatively) affect each other.

For example, if you would first do a translation and then scale, the translation vector would also scale!

Running the final transformation matrix on our vector results in the following vector:

$$\begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 2x+1 \\ 2y+2 \\ 2z+3 \\ 1 \end{bmatrix}$$

The vector is first scaled by two and then translated by (1,2,3).