POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

DE
IB

2024

Dipartimento di Elettronica, Informazione e Bioingegneria

Computer Graphics

Milano, 2024

Computer Graphics

- 3D coordinates and basic transforms

Homogeneous coordinates

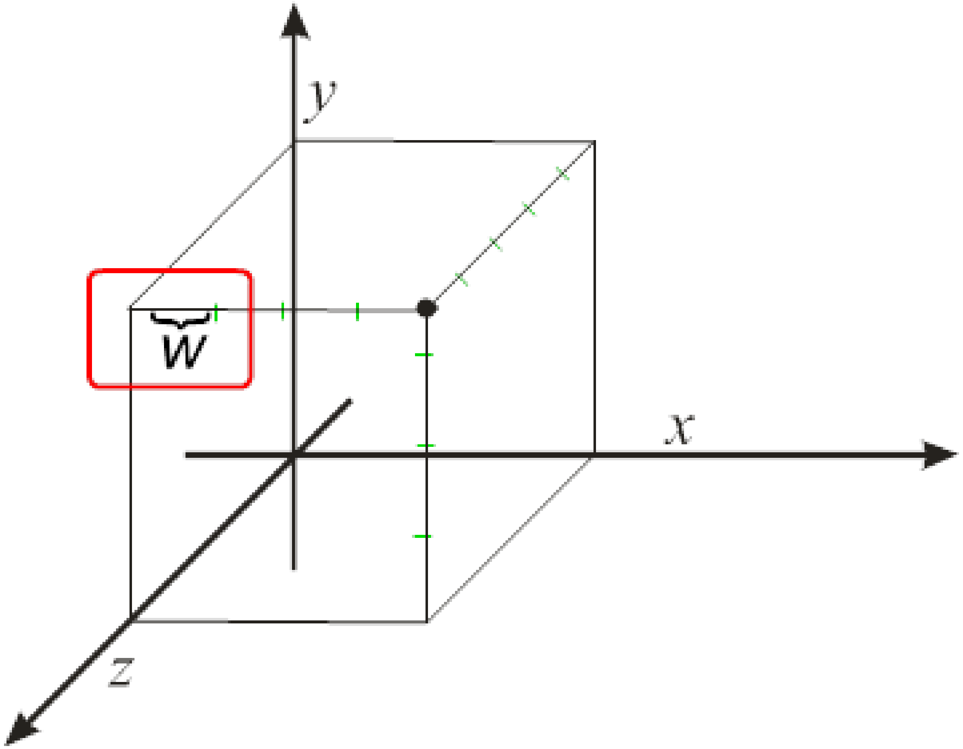
In order to represent objects in a 3D space, an appropriate coordinate system must be chosen.

For reasons that will be clear when we will deal with transformations and projections of 3D coordinates, we use a special system called *homogeneous coordinates*.

In homogeneous coordinates, a point in the 3D space is characterized by *four* values: *x, y, z* and *w*.

Homogeneous coordinates

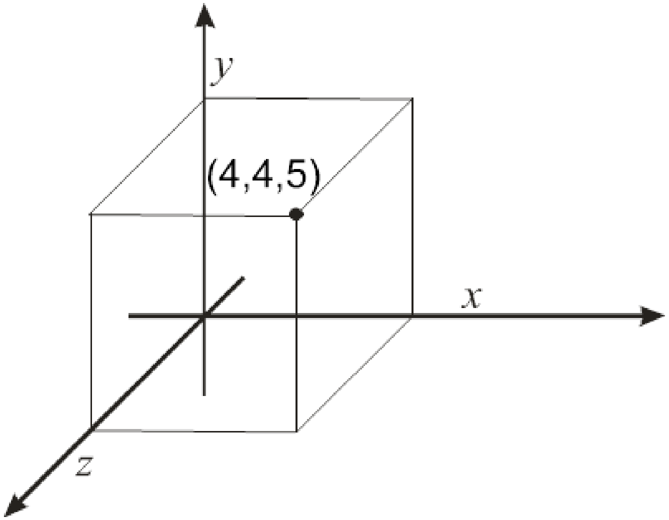
The coordinates *x*, *y* and *z* represents the position of the point in the 3D space, while coordinate *w* defines a *scale*: the units of measure used by the other three coordinates.



Homogeneous coordinates

Since we have four values to define a point in a 3D space, we have an infinite number of coordinates that identify the same position.

In particular, *all tuples of four values that are linearly dependent* represent the same point in the 3D space.

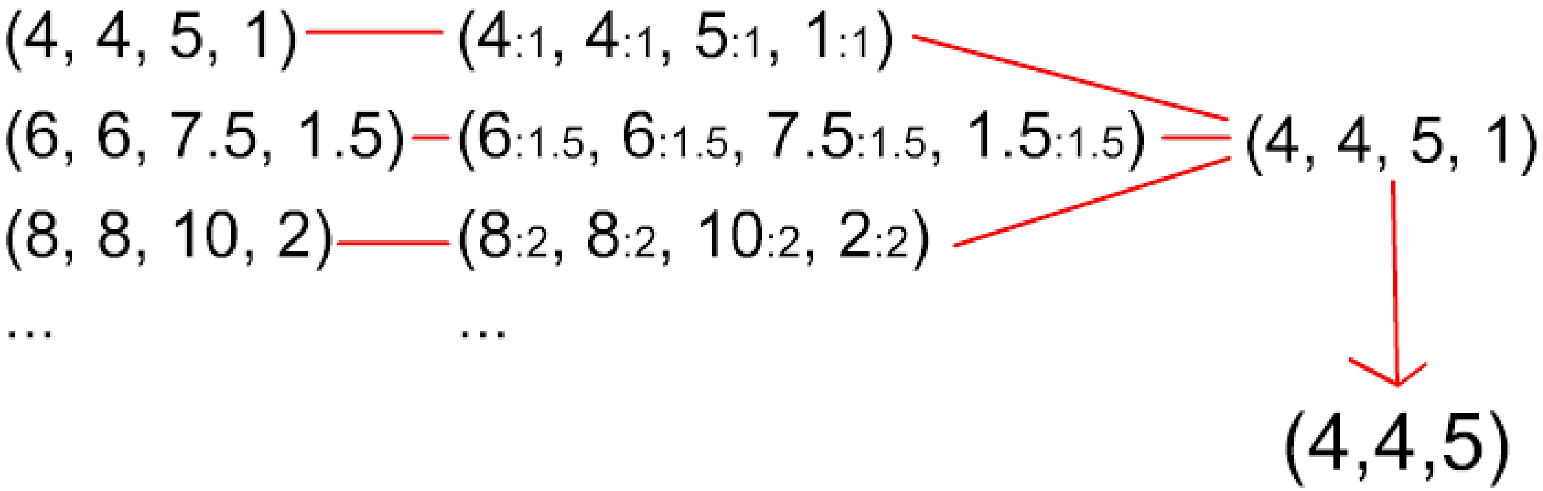


(2, 2, 2.5, 0.5)
(4, 4, 5, 1)
(5, 5, 6.25, 1.25)
(6, 6, 7.5, 1.5)
(-8, -8, -10, -2)
....

Homogeneous coordinates

The *x, y,* and *z* coordinates of the vector with *w = 1* identify the “real” position of the point in the 3D space.

Since all the vectors representing the same point are linearly dependent, we can obtain the one with *w = 1* by simply dividing first three components (*x, y, z)* by *w,* the fourth one.



(4, 4, 5, 1) — (4:1, 4:1, 5:1, 1:1)

(6, 6, 7.5, 1.5) — (6:1.5, 6:1.5, 7.5:1.5, 1.5:1.5) — (4, 4, 5, 1)

(8, 8, 10, 2) — (8:2, 8:2, 10:2, 2:2)

...

...

(4,4,5)

Homogeneous coordinates

In other words, we can find the *(x',y',z')* Cartesian coordinates corresponding to any point in homogeneous coordinates *(x,y,z,w)* in this way:

$$(x, y, z, w) \quad \rightarrow \quad (x', y', z') = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w} \right)$$

Conversely, we can simply transform a point with Cartesian coordinates *(x,y,z)* into homogeneous coordinates by adding a fourth component *w=1*.

$$(x, y, z) \quad \rightarrow \quad (x, y, z, 1)$$

Homogeneous coordinates

Example:

The Cartesian coordinate:

$$A_c(1, 3, -5)$$

corresponds in homogeneous coordinates to:

$$A_h = (1, 3, -5, 1)$$

The homogeneous coordinates:

$$B_h(1, 3, -5, 2), \quad C_h(6, -3, 2, 1/2)$$

correspond, in Cartesian coordinates, respectively to:

$$B_c = (1, 3, -5, 2) = (1/2, 3/2, -5/2) = (0.5, 1.5, -2.5)$$

$$C_c = (6, -3, 2, 1/2) = (6/0.5, -3/0.5, 2/0.5) = (12, -6, 4)$$

Affine transforms

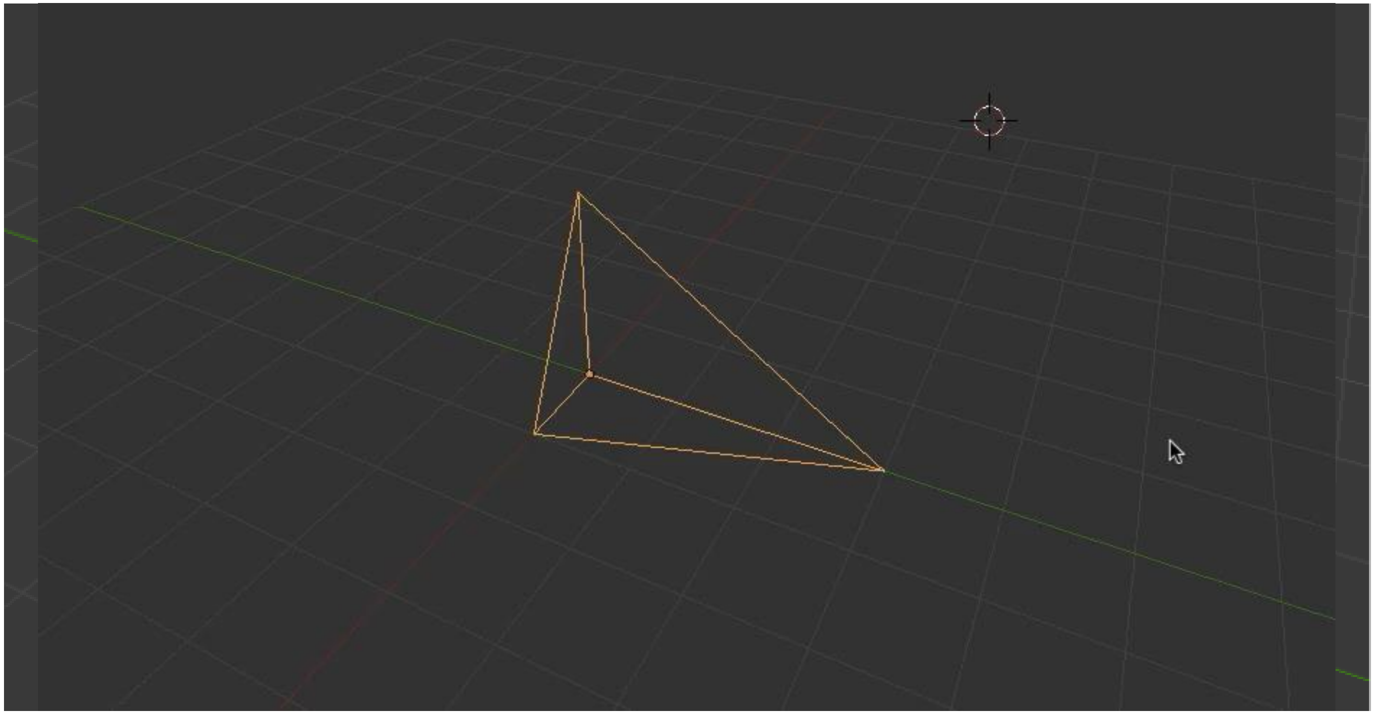
The process of varying the coordinates of the points of an object is defined as a **transformation**.

Transformations in 3D can be quite complex, since all the points of the object might be repositioned in a three-dimensional space.

However, there is an important and large set of transformations that can be summarized with a mathematical concept known as the *affine transforms*.

Affine transforms

Objects are defined in the 3D space through the coordinates of their points.



By applying an affine transformations to the coordinates of their points, objects can be moved, rotated or scaled in the 3D space.

Affine transforms

The affine transforms are usually grouped in four classes:

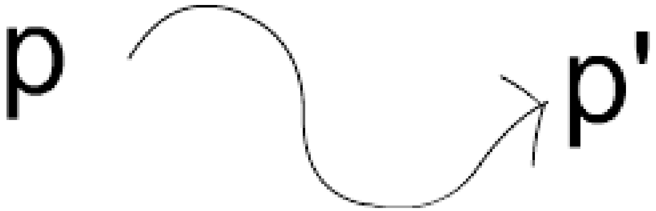
- *Translation*
- *Scaling*
- *Rotation*
- *Shear*

To translate, rotate or scale an object, the same transformation is applied to all its points.

The transformed object is obtained by reconstructing the geometric primitive with the new points.

Affine transforms

In the following, we will call *p=(x,y,z,w)* the original vertex, and with *p'=(x',y',z',w')* the transformed one.



4x4 Matrix transforms

When using homogeneous coordinates, *4x4 matrices* can express the considered geometrical transforms.

The new vertex *p'* can be computed from the old point *p* by simply multiplying it with the corresponding transform matrix *M*.

The basic transformations we are considering, are constructed to maintain the fourth component of the resulting vector unchanged.

$$p = (x, y, z, 1)$$

$$p' = M \times p^T$$

$$p' = \left(x', y', z', 1\right)$$

4x4 Matrix transformations: on the left or on the right?

Note that two opposite conventions can be used:

- The *Transform Matrix* is on the left
- The *Transform Matrix* is on the right.

Matrix-on-the-left

$$p = (x, y, z, 1)$$

$$p' = \left(M \times p^T\right)^T$$

$$p' = \left(x', y', z', 1\right)$$

Matrix-on-the-right

$$p = (x, y, z, 1)$$

$$p' = p \times M^T$$

$$p' = \left(x', y', z', 1\right)$$

We will use the convention with the *matrix-on-the-left*, since it is the most used in text-books. We will also consider only column vectors, and drop the "^{T}" operator in the matrix products.

Be careful however that many libraries used to ease the development of 3D applications are better suited for the matrix-on-the-right convention.

Identity transform

The simplest transform, is the one that does not perform any change on the points of an object.

This is the *Identity Transform*.

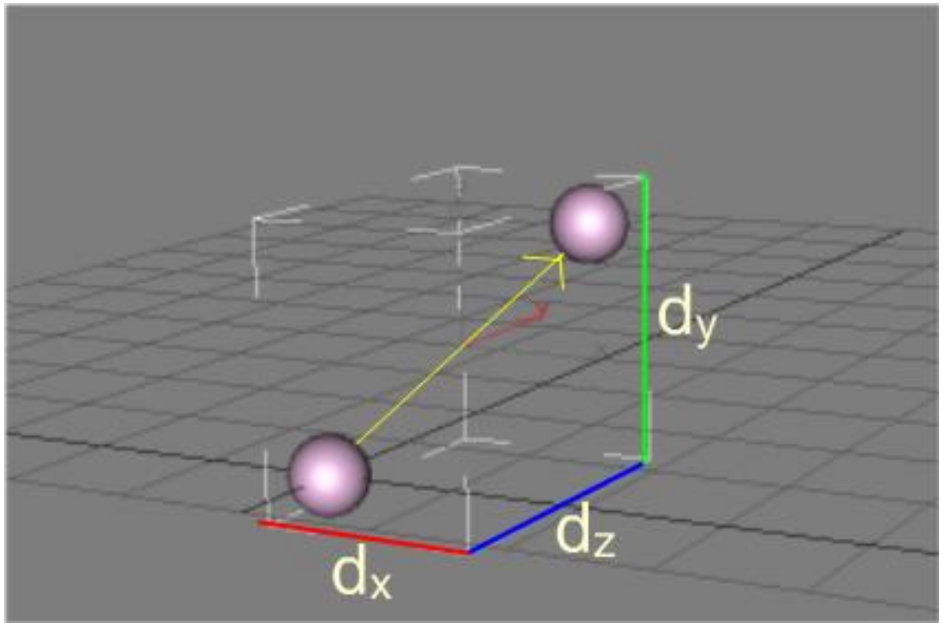
As the name suggests, it can be implemented with a 4x4 identity matrix.

$$I = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Translation

Translation *moves* the points of an object, while maintaining its size and orientation.

Let us imagine moving an object of d_x units along the x-axis, d_y on the y-axis and d_z on the z-axis.



Translation

The new coordinates can be obtained by simply adding the corresponding movement to each axis:

$$x' = x + d_x$$

$$y' = y + d_y$$

$$z' = z + d_z$$

Since in homogeneous coordinates derived from cartesian ones the fourth component is always *w=1*, the translation matrix $T(d_x, d_y, d_z)$ can be obtained by putting d_x, d_y and d_z on the last column of the identity matrix.

$$T(d_x, d_y, d_z) = \begin{vmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Translation

Example:

Consider a translation of +2 on the x-axis and -2 on z-axis:
The corresponding matrix is T(2,0,-2), and it can be used in the following way to transform points:
A(1, 2, 3), B(-4, 2, -1) .

$$T(2,0,-2) = \begin{vmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$A' = \begin{vmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 \\ 2 \\ 3 \\ 1 \end{vmatrix} = \begin{vmatrix} 3 \\ 2 \\ 1 \\ 1 \end{vmatrix}$$

$$B' = \begin{vmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} -4 \\ 2 \\ -1 \\ 1 \end{vmatrix} = \begin{vmatrix} -2 \\ 2 \\ -3 \\ 1 \end{vmatrix}$$

Scaling

Scaling modifies the *size* of an object, while maintaining constant its position and its orientation.

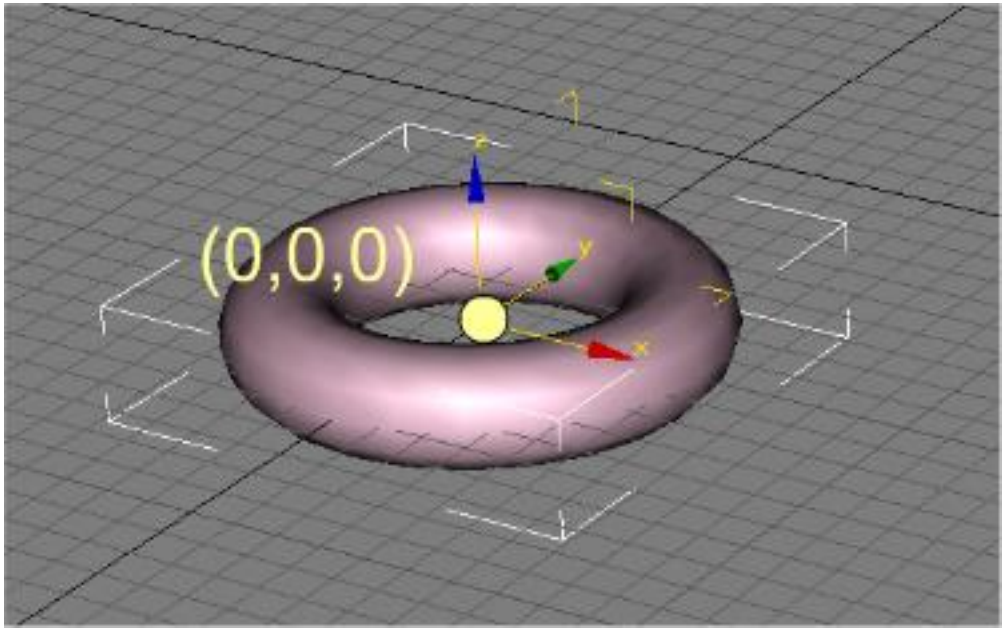
Scaling can be used to obtain several effects:

- Enlarge
- Shrink
- Deform
- Mirror
- Flatten

Scaling

The scale transforms have a center: a point that is not moved during the transformation.

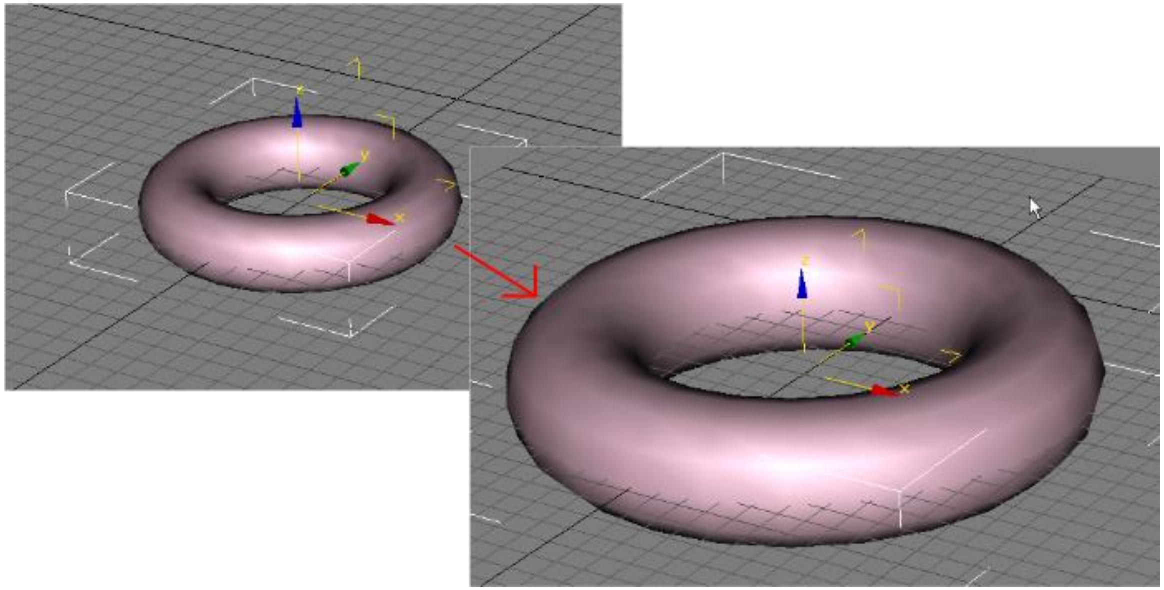
Initially, we consider the center of the scaling located in the origin of the 3D space (i.e. at *x = y = z = 0*).



Scaling

Proportional scaling enlarges or shrinks an object of the same amount *s* in all the directions.

For this reason proportional scaling maintains the proportions of the objects while changing its size.



Scaling

Multiplying the three coordinates of the points with a factor *s* performs proportional scaling.

$$x' = s \times x$$

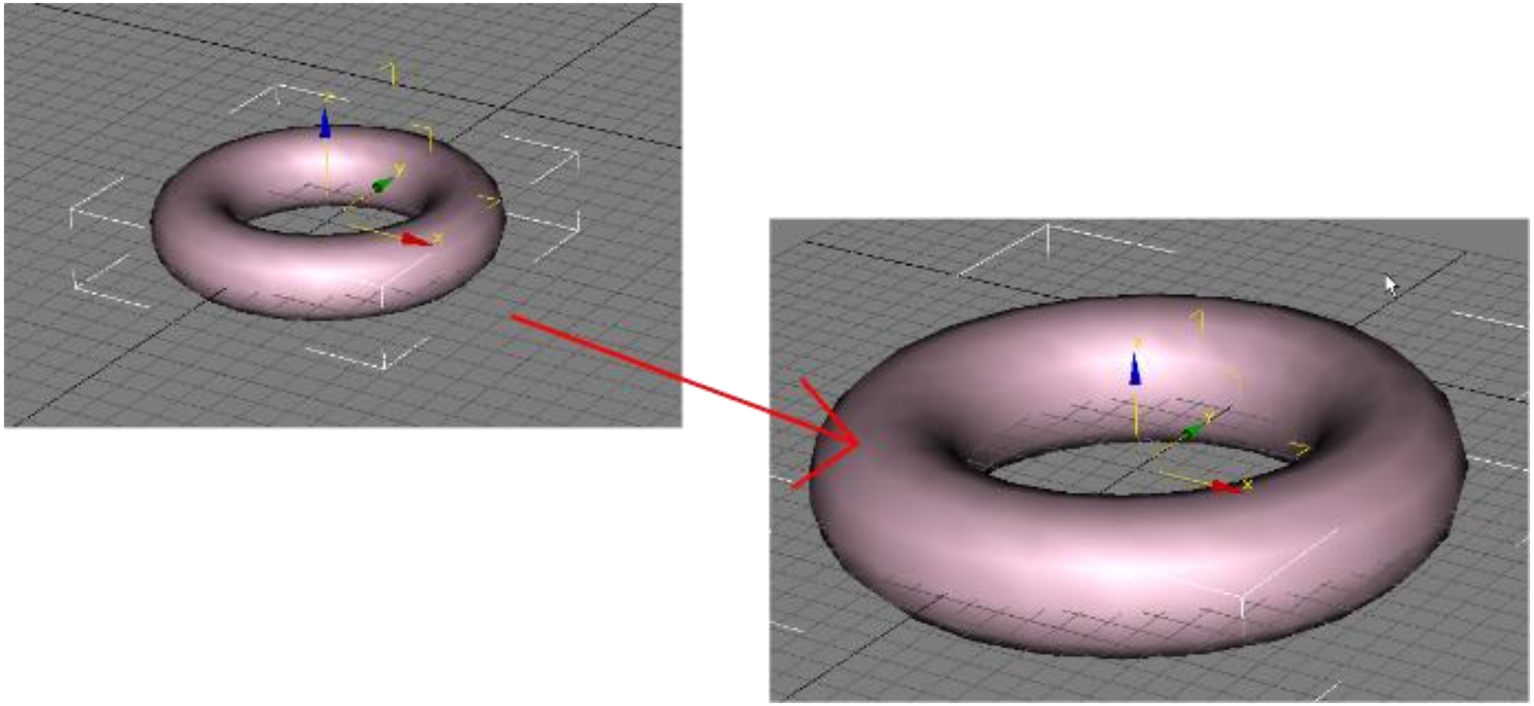
$$y' = s \times y$$

$$z' = s \times z$$

Depending on the value of *s*, the transformation can either enlarge or shrink an object.

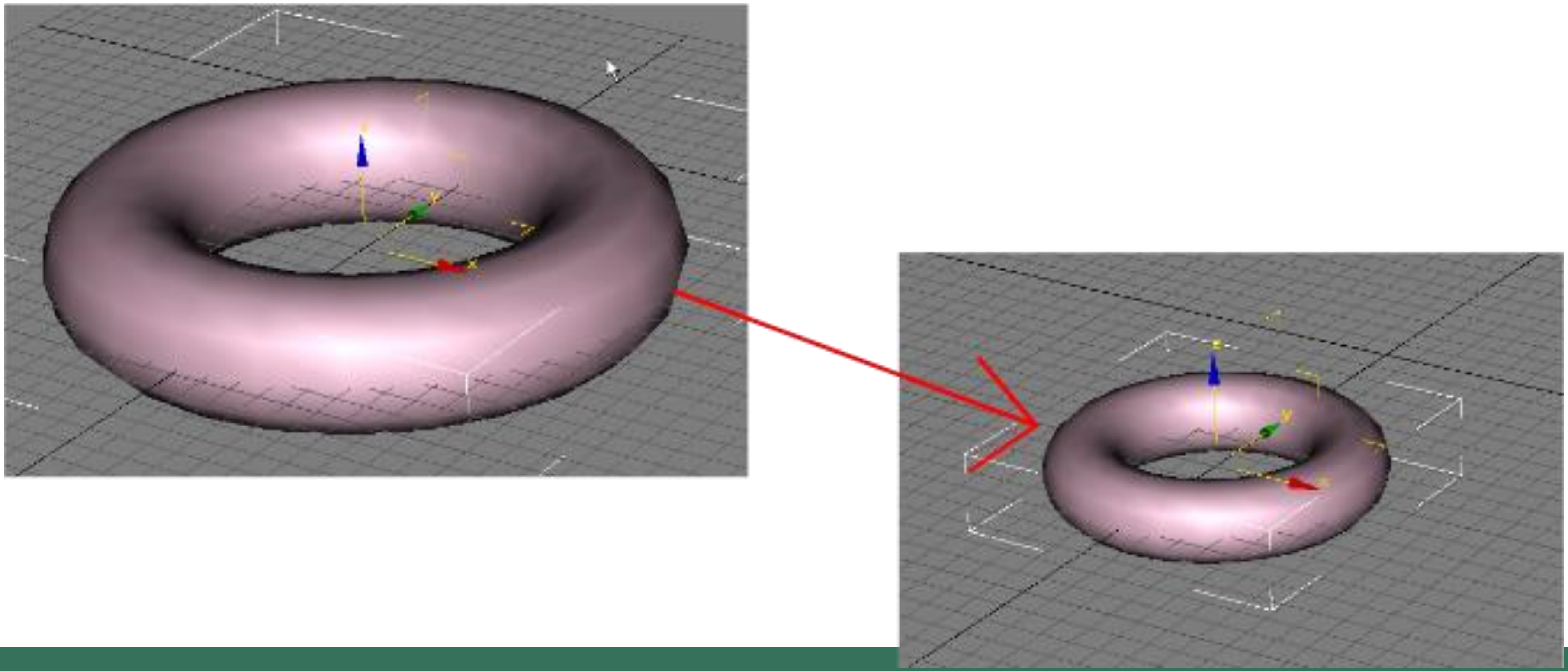
Scaling

A factor *s > 1*, enlarges *s* times the object. For example, *s=2* doubles the size.



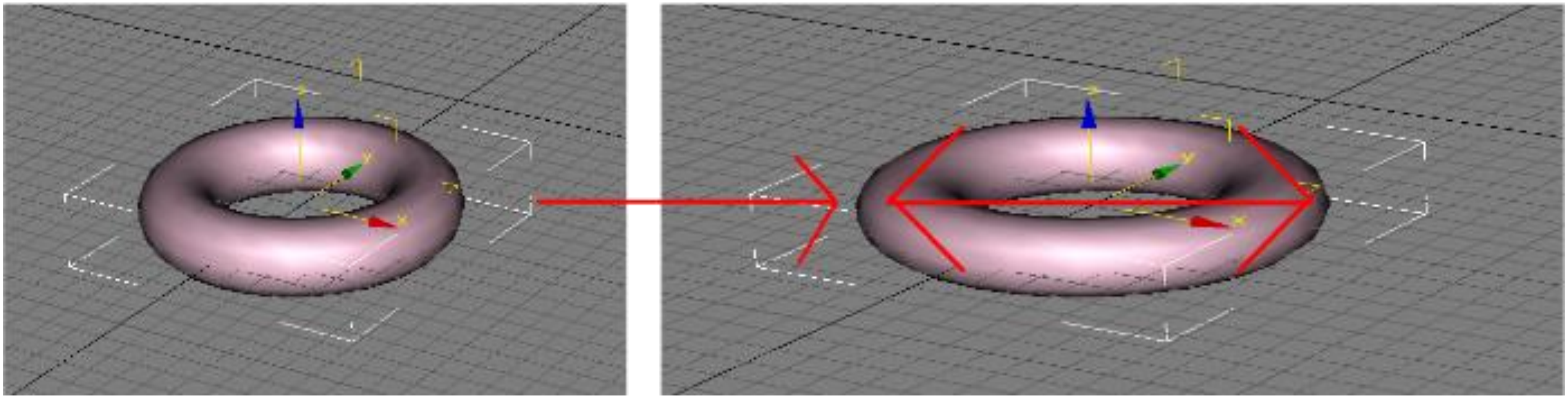
Scaling

A factor $0 < s < 1$, shrinks $1/s$ times the object. For example, $s = \frac{1}{2} = 0.5$ halves the object.



Scaling

Non-proportional scaling deforms an object by using a different scaling factors s_x, s_y and s_z for each axis. It can be used to enlarge or shrink an object only in one direction.



Initially we will suppose that non-proportional scaling can only occur with respect to the three main axis. Later we will see how to apply non-proportional scaling in arbitrary directions.

Scaling

The new coordinates can be simply computed as:

$$x' = s_x \times x$$

$$y' = s_y \times y$$

$$z' = s_z \times z$$

The transform matrix *S(s*

placing the scaling factors s_x, s_y and s_z on the diagonal:

$$S(s_x, s_y, s_z) = \begin{vmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Proportional scaling is obtained using identical scaling factors $s_x = s_y = s_z = s$.

Scaling

Example:

A scaling of 2.5 on the y-axis and 0.5 on z-axis is performed by matrix S(1, 2.5, 0.5), and it can be used in the following way to transform points A(1, 2, 3), B(-4, 2, -1) .

$$A' = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 2.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 \\ 2 \\ 3 \\ 1 \end{vmatrix} = \begin{vmatrix} 1 \\ 5 \\ 1.5 \\ 1 \end{vmatrix}$$

$$S(1, 2.5, 0.5) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 2.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$B' = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 2.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} -4 \\ 2 \\ -1 \\ 1 \end{vmatrix} = \begin{vmatrix} -4 \\ 5 \\ -0.5 \\ 1 \end{vmatrix}$$

Scaling - mirroring

Mirroring can be obtained by *using negative scaling factors*.

In particular, three possible types of mirroring can be done:

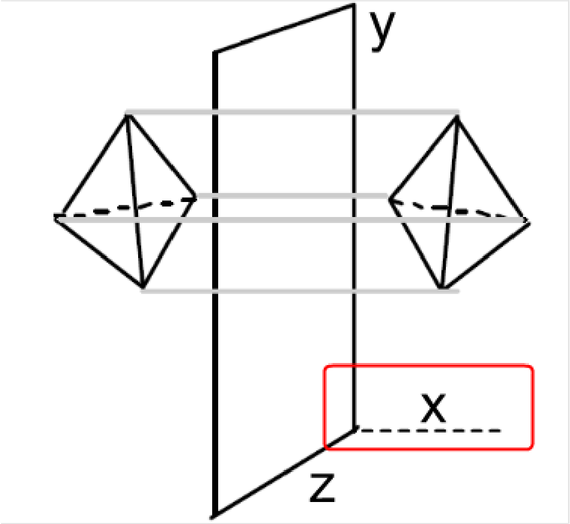
- Planar
- Axial
- Central

Again, we will initially assume that mirror occurs around a plane or axis that passes through the origin and it is aligned to the *x*, *y* or *z* axes.

Scaling - mirroring

Planar mirroring creates the symmetric object with respect to a plane.

It is obtained by assigning *-1* to the scaling factor of the axis perpendicular to the plane (*x* for plane *yz*).



$$s_x = -1$$

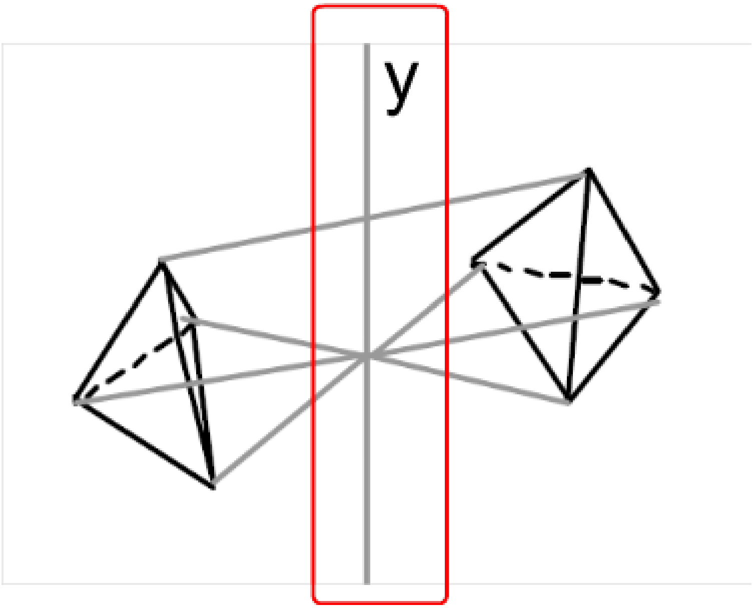
$$s_y = 1$$

$$s_z = 1$$

Scaling - mirroring

Axial mirroring creates the symmetric object with respect to an axis.

It is obtained by assigning *-1* to all the scaling factors except the one of the considered axis (*x* and *z* for y-axis).



$$s_x = -1$$

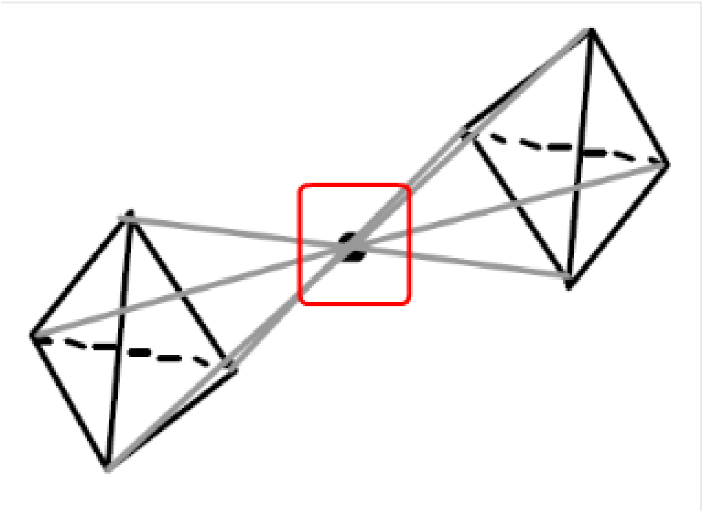
$$s_y = 1$$

$$s_z = -1$$

Scaling - mirroring

Central mirroring creates the symmetric object with respect to the origin.

It is obtained by assigning *-1* to all the scaling factors.



$$s_x = -1$$

$$s_y = -1$$

$$s_z = -1$$

Scaling - flattening

A scaling factor of *0* in any direction, flattens the image along that axis. This however should be handled with care, since it will make the objects loose one dimension.

To simplify our discussion, we will almost always suppose that the scaling coefficients are different from *0*.

$$x' = s_x \times x \qquad s_x \neq 0$$

$$y' = s_y \times y \qquad s_y \neq 0$$

$$z' = s_z \times z \qquad s_z \neq 0$$

Scaling

Example:

Consider mirroring about the *xy*-plane.

The corresponding transform matrix is S(1,1,-1), and it can be used to transform points A(1, 2, 3), B(-4, 2, -1) in the following way:

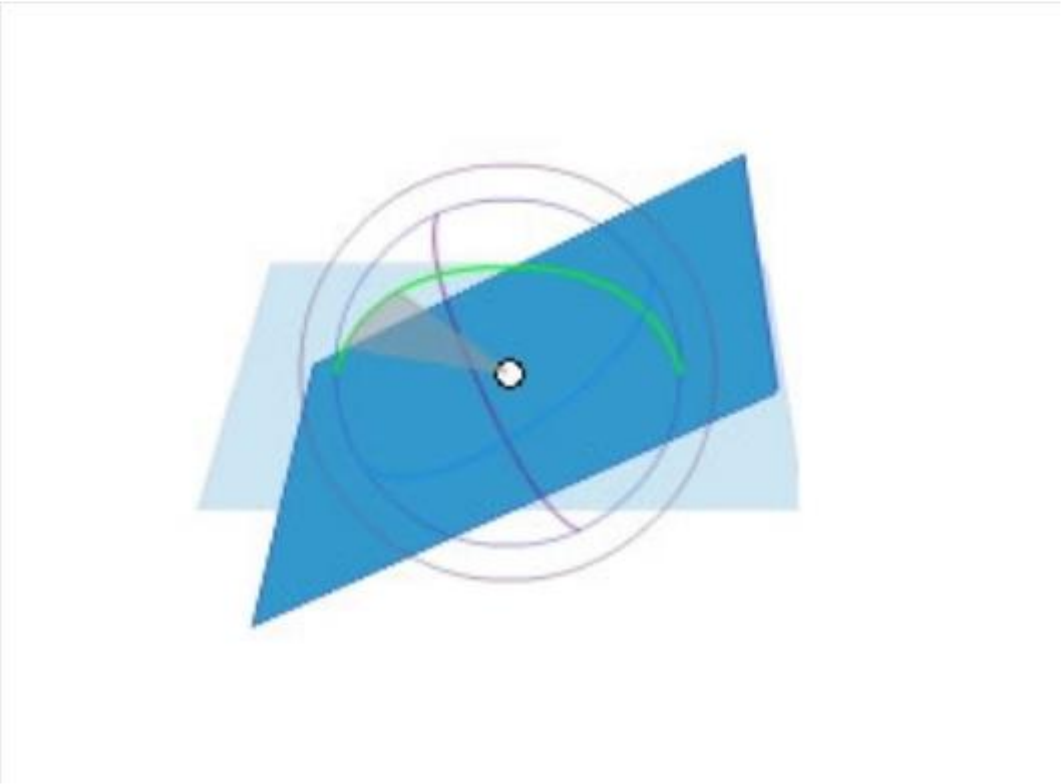
$$S(1,1,-1)=\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$A'=\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}\begin{vmatrix} 1 \\ 2 \\ 3 \\ 1 \end{vmatrix}=\begin{vmatrix} 1 \\ 2 \\ -3 \\ 1 \end{vmatrix}$$

$$B'=\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}\begin{vmatrix} -4 \\ 2 \\ -1 \\ 1 \end{vmatrix}=\begin{vmatrix} -4 \\ 2 \\ 1 \\ 1 \end{vmatrix}$$

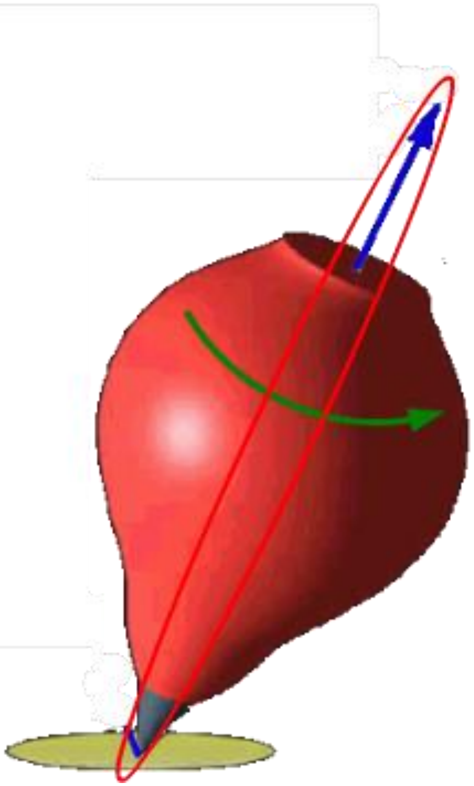
Rotation

Rotation varies the *orientation* of an object, leaving its position and size unchanged.



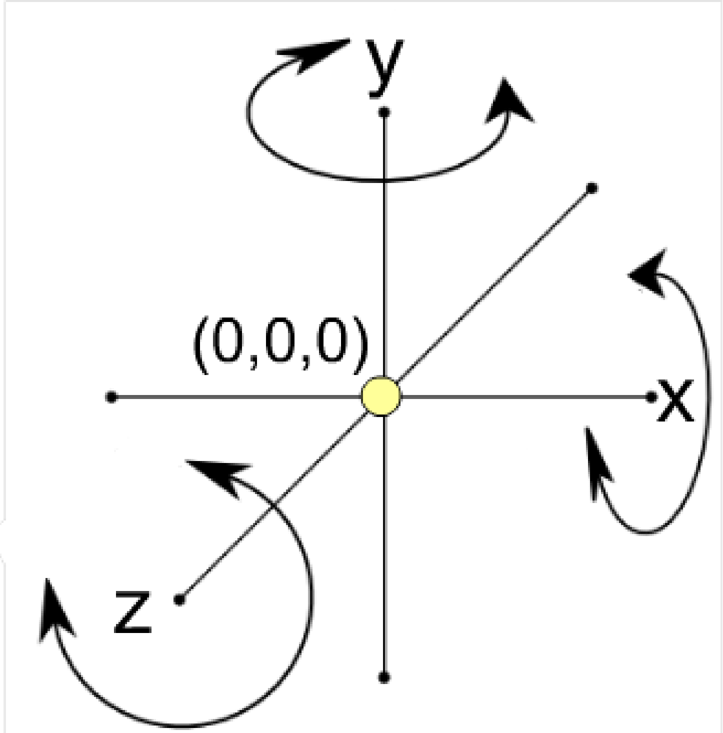
Rotation

Rotation is always performed along an *axis*: a line where points are unaffected by the transformation.



Rotation

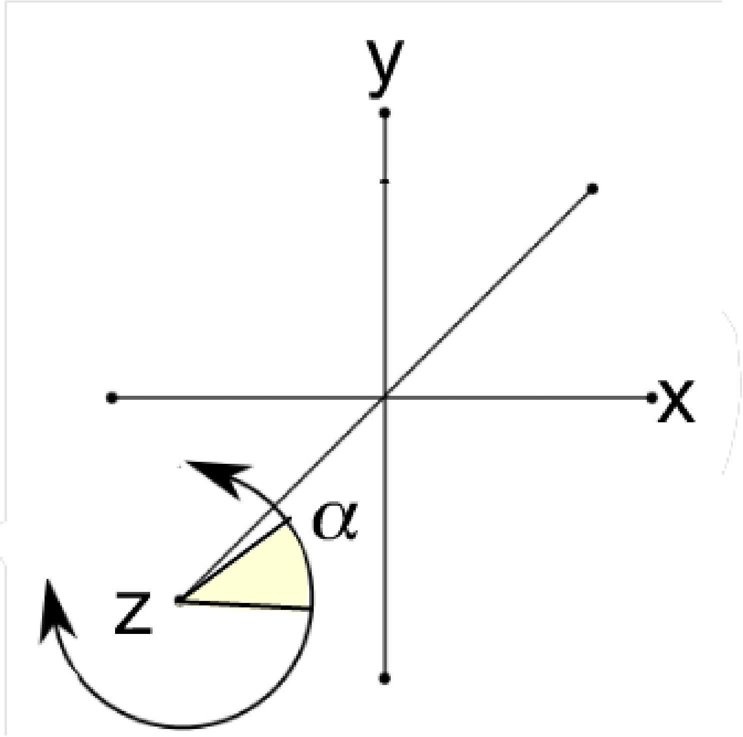
Again, we will begin by considering only rotations about the three reference axis and passing through the origin.



Rotation

A rotation of an angle α about the z-axis can be computed as shown below. Note that points on the z-axis are unaffected.

A simple way to recall this formula is to first rotate the x component as if it was a point on the x-axis (i.e with *y=0*), then rotate the y component as a point on the y-axis (i.e with *x=0*) and finally sum up the two.



$$x' = x \times \cos a - y \times \sin a$$

$$y' = x \times \sin a + y \times \cos a$$

$$z' = z$$

Rotation

Thanks to homogeneous coordinates, the rotations of an angle α around the *z-axis* can be expressed with a matrix.

Rotations around the *x-axis* and the *y-axis* can be performed in a similar way, and expressed by the following matrices.

$$x' = x$$
$$y' = y \times \cos a - z \times \sin a$$
$$z' = y \times \sin a + z \times \cos a$$

x-axis

$$R_x(a) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos a & -\sin a & 0 \\ 0 & \sin a & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$x' = x \times \cos a + z \times \sin a$$
$$y' = y$$
$$z' = -x \times \sin a + z \times \cos a$$

y-axis

$$R_y(a) = \begin{vmatrix} \cos a & 0 & \sin a & 0 \\ 0 & 1 & 0 & 0 \\ -\sin a & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$x' = x \times \cos a - y \times \sin a$$
$$y' = x \times \sin a + y \times \cos a$$
$$z' = z$$

z-axis

$$R_z(a) = \begin{vmatrix} \cos a & -\sin a & 0 & 0 \\ \sin a & \cos a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Rotation

Example:

Consider rotation of *90°* on the y-axis. The corresponding matrix $R_y(90^o)$ is written as follows, and can be used to transform points A(1, 2, 3), B(-4, 2, -1) as shown.

$$R_y(90^\circ) = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

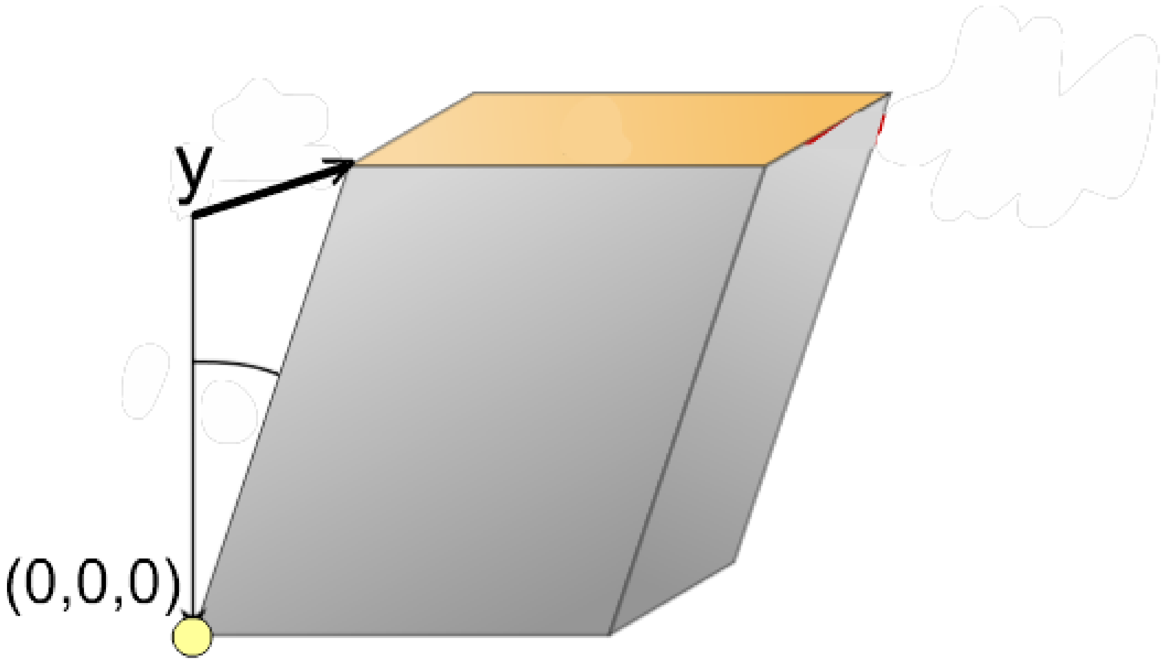
$$A' = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 \\ 2 \\ 3 \\ 1 \end{vmatrix} = \begin{vmatrix} 3 \\ 2 \\ -1 \\ 1 \end{vmatrix}$$

$$B' = \begin{vmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} -4 \\ 2 \\ -1 \\ 1 \end{vmatrix} = \begin{vmatrix} -1 \\ 2 \\ 4 \\ 1 \end{vmatrix}$$

Shear

The **shear** transform *bends* an object in one direction.

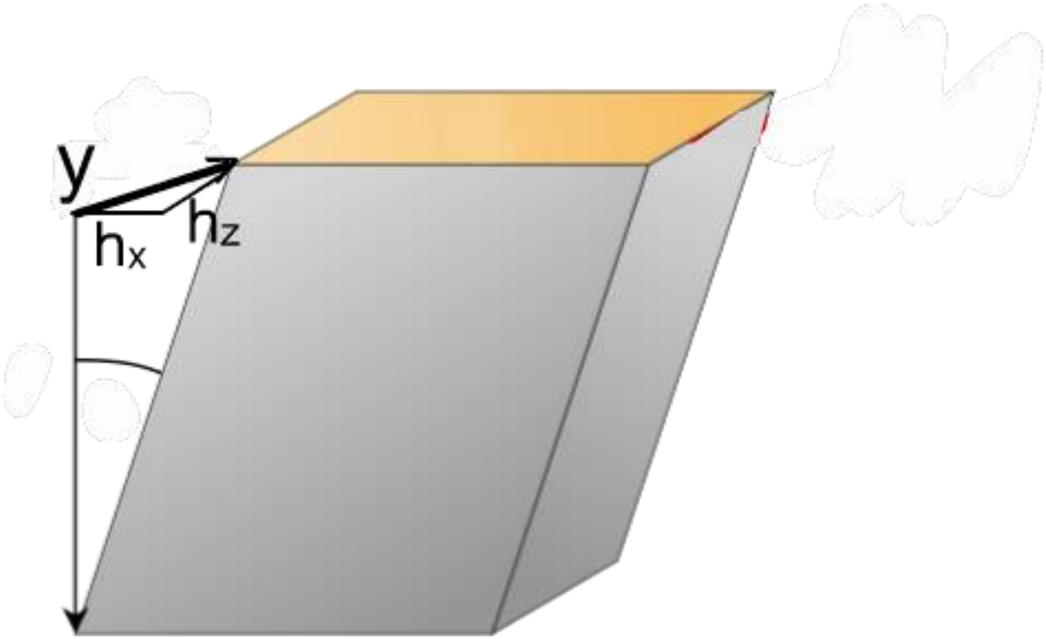
Shear is performed along an axis and has a center. We initially focus on the y-axis passing through the origin.



Shear

As the value of y-axis increases, the object is linearly bent into a direction specified by a 2D vector (in this case defined by two values: h_x and h_z).

The coordinates of the transformed point can be computed as shown:



$$x' = x + y \times h_x$$

$$y' = y$$

$$z' = z + y \times h_z$$

Shear

A matrix formulations can be given, and the same transform can be considered also along the *x-axis* and the *z-axis* :

x-axis

$$H_x(h_y, h_z) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ h_y & 1 & 0 & 0 \\ h_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

y-axis

$$H_y(h_x, h_z) = \begin{vmatrix} 1 & h_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & h_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

z-axis

$$H_z(h_x, h_y) = \begin{vmatrix} 1 & 0 & h_x & 0 \\ 0 & 1 & h_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Shear

Example:

In the following you can see matrix *$H_x(1,0.5)$* that performs a shear along the x-axis which bends the object at the rate of 1 on the y-axis and 0.5 on the z-axis. The matrix is used to transform points A(1, 2, 3), B(-4, 2, -1) .

$$H_x(1,0.5) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0.5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$A' = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0.5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}\begin{vmatrix} 1 \\ 2 \\ 3 \\ 1 \end{vmatrix} = \begin{vmatrix} 1 \\ 3 \\ 3.5 \\ 1 \end{vmatrix}$$

$$B' = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0.5 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}\begin{vmatrix} -4 \\ 2 \\ -1 \\ 1 \end{vmatrix} = \begin{vmatrix} -4 \\ -2 \\ -3 \\ 1 \end{vmatrix}$$

Matrix transformations

Note that the last row in all the 4x4 transformation matrices we have presented is always | 0 0 0 1 |.

This ensures that the *w* coordinate is unchanged by the transformation (and in particular it is kept *w=1*).

$$M = \begin{vmatrix} n_{xx} & n_{yx} & n_{zx} & d_x \\ n_{xy} & n_{yy} & n_{zy} & d_y \\ n_{xz} & n_{yz} & n_{zz} & d_z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$p = (x, y, z, 1)$$

$$p' = M \times p$$

$$p' = \left(x', y', z', \boxed{1}\right)$$

Matrix transformations

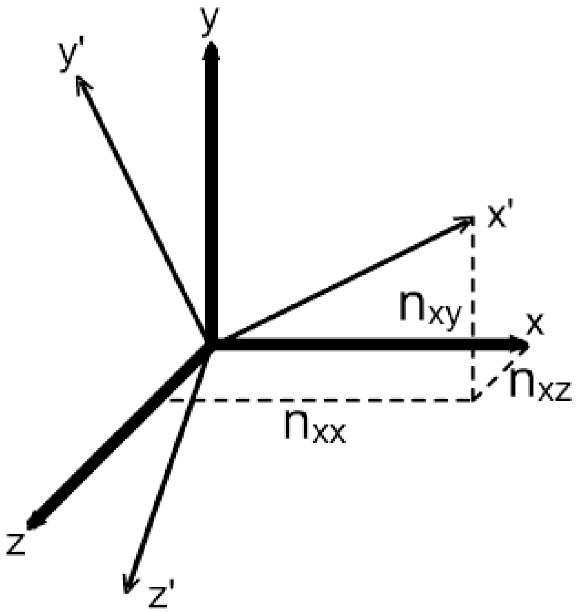
The upper part of a transform matrix, can be divided into a 3x3 sub-matrix $\mathbf{M}_\mathbf{R}$ that represents the rotation, scaling and shear factors of the transform, and a column vector $\mathbf{d}^\mathrm{T}$ that encodes the translation.

$$M = \left| \begin{array}{ccc|c} n_{xx} & n_{yx} & n_{zx} & d_x \\ n_{xy} & n_{yy} & n_{zy} & d_y \\ n_{xz} & n_{yz} & n_{zz} & d_z \\ \hline 0 & 0 & 0 & 1 \end{array} \right| = \left| \begin{array}{c|c} M_R & \mathbf{d}^T \\ \hline \mathbf{0} & 1 \end{array} \right|$$

Matrix transformations

In particular, the matrix product exchanges the three Cartesian axis of the original coordinate system, with three new directions.

The columns of $\mathbf{M_R}$ represent the directions and sizes of the new axes in the old reference system (when translated to the origin).

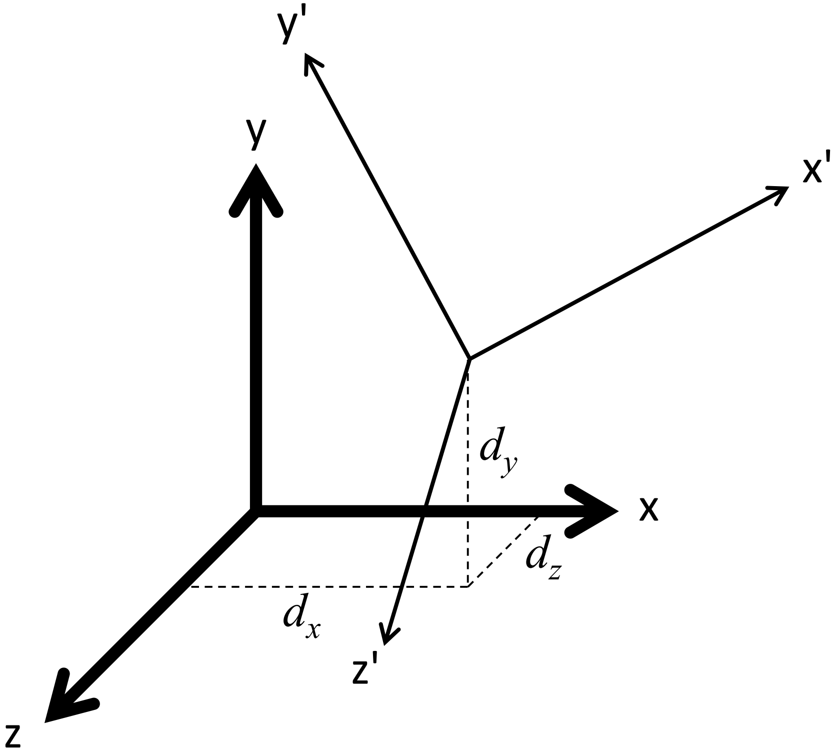


$$M_R = \begin{vmatrix} \boxed{\begin{matrix} n_{xx} \\ n_{xy} \\ n_{xz} \end{matrix}} & \begin{matrix} n_{yx} \\ n_{yy} \\ n_{yz} \end{matrix} & \begin{matrix} n_{zx} \\ n_{zy} \\ n_{zz} \end{matrix} \end{vmatrix}$$

Matrix transformations

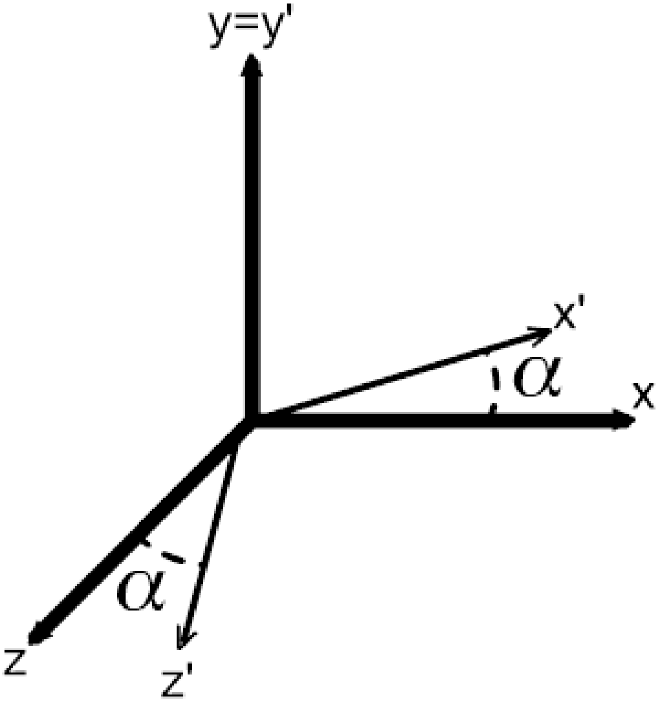
Vector **d**^{T} represents the position of the origin of the new coordinates system in the old one.

$$M = \left|\begin{array}{c:c} M_R & \boxed{\begin{matrix} d_x \\ d_y \\ d_z \end{matrix}} \\ \hdashline \mathbf{0} & 1 \end{array}\right|$$



Matrix transformations

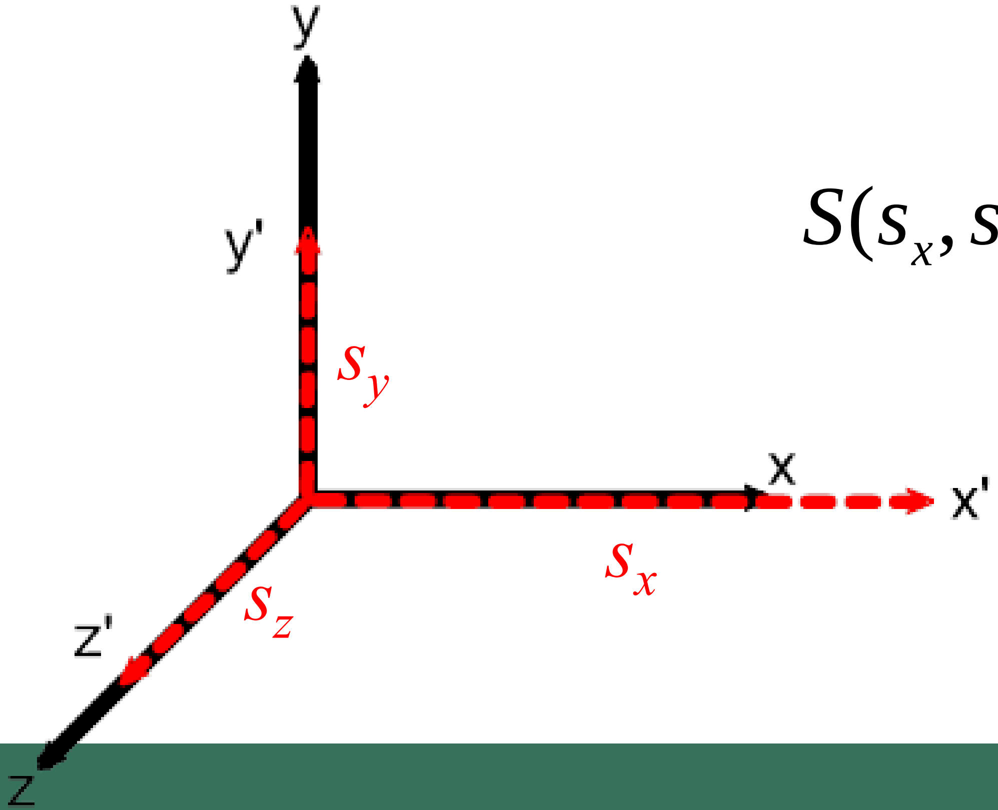
Rotations maintain the size and the angles of the axes constant, but change their directions.



$$R_y(a) = \left| \begin{array}{ccc:c} \cos a & 0 & \sin a & 0 \\ 0 & 1 & 0 & 0 \\ -\sin a & 0 & \cos a & 0 \\ \hdashline 0 & 0 & 0 & 1 \end{array} \right|$$

Matrix transformations

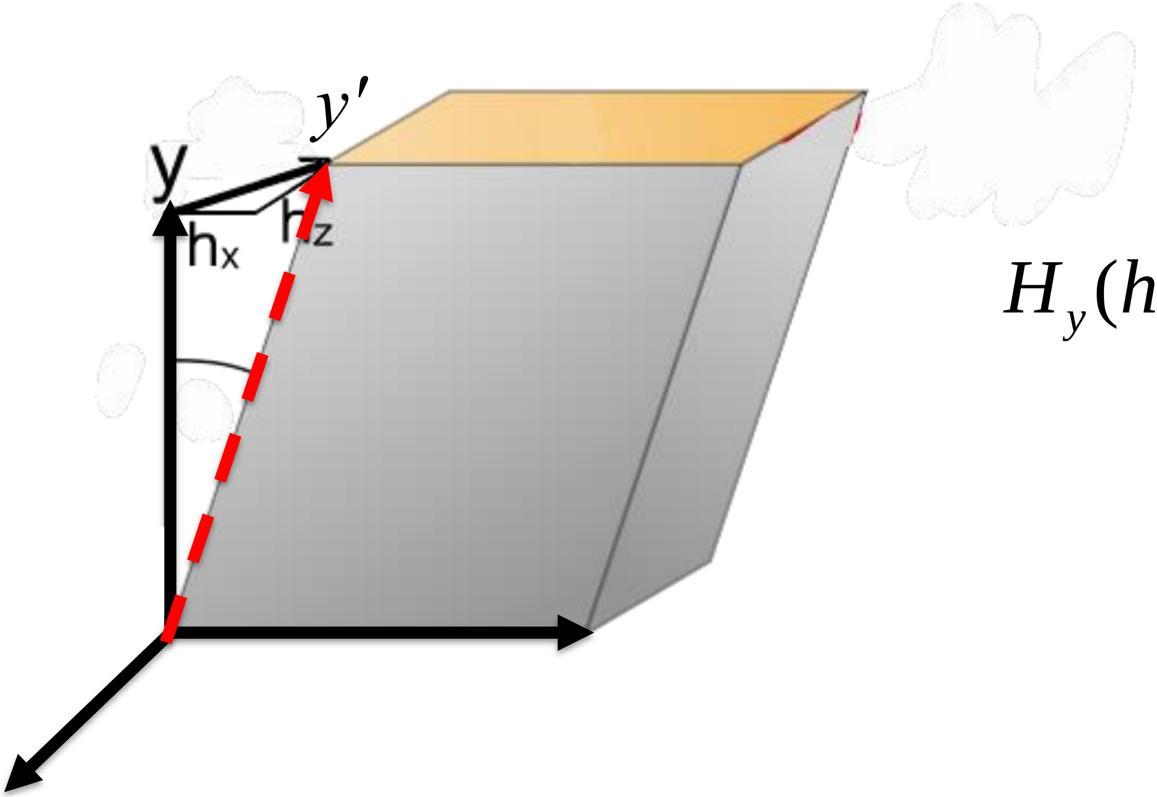
Scaling increases or decreases the size of the axes, while maintaining their original directions.



$$S(s_x, s_y, s_z) = \left|\begin{array}{ccc:c} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right|$$

Matrix transformations

Shear bends the axis along which the transform is performed.

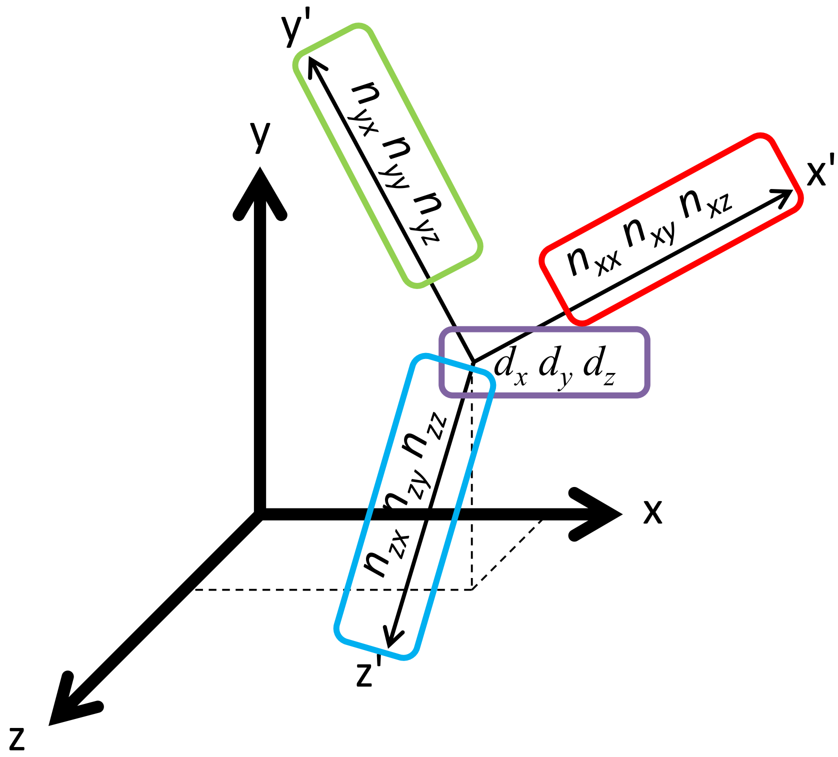


$$H_y(h_x,h_z)=\begin{vmatrix} 1 & \boxed{h_x} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & h_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Matrix transformations

In many cases it is easier to define a transformation by specifying its new center, and the new directions of its axes.

$$M = \left|\begin{array}{cccc:c} n_{xx} & n_{yx} & n_{zx} & d_x \\ n_{xy} & n_{yy} & n_{zy} & d_y \\ n_{xz} & n_{yz} & n_{zz} & d_z \\ \hdashline 0 & 0 & 0 & 1 \end{array}\right.$$


y'
n_{yx} n_{yy} n_{yz}
y
x'
n_{xx} n_{xy} n_{xz}
d_x d_y d_z
n_{zx} n_{zy} n_{zz}
x
z'
z

Matrix transformations

Note that with the matrix-on-the-right convention, all the transform matrices are transposed.

$$M^T = \left|\begin{array}{ccc|c} \multicolumn{3}{c|}{\boxed{n_{xx} \quad n_{xy} \quad n_{xz}}} & 0 \\ n_{yz} & n_{yy} & n_{yz} & 0 \\ n_{zx} & n_{zy} & n_{zz} & 0 \\ \hline d_x & d_y & d_z & 1 \end{array}\right| = \left|\begin{array}{c|c} M_R^T & \mathbf{0} \\ \hline \mathbf{d} & 1 \end{array}\right|$$

In this course, we will never use the matrix-on-the-right notation.

Matrix transformations

A way to determine which convention is used (if unknown), is by looking at a non-zero translation transform: if the matrix has the last *column* |0 0 0 1|, then matrix-on-the-right is used.

If the last *row* is |0 0 0 1|, then it is matrix-on-the-left (the one considered here).

Matrix-on-the-left

$$M = \left| \begin{array}{ccc:c} n_{xx} & n_{yx} & n_{zx} & d_x \\ n_{xy} & n_{yy} & n_{zy} & d_y \\ n_{xz} & n_{yz} & n_{zz} & d_z \\ \hdashline 0 & 0 & 0 & 1 \end{array} \right| = \left| \begin{array}{c:c} M_R & \mathbf{d}^T \\ \hdashline \mathbf{0} & 1 \end{array} \right|$$

In this course, we will never use the matrix-on-the-right convention.

Matrix-on-the-right

$$M^T = \left| \begin{array}{ccc:c} n_{xx} & n_{xy} & n_{xz} & 0 \\ n_{yz} & n_{yy} & n_{yz} & 0 \\ n_{zx} & n_{zy} & n_{zz} & 0 \\ \hdashline d_x & d_y & d_z & 1 \end{array} \right| = \left| \begin{array}{c:c} M_R^T & \mathbf{0} \\ \hdashline \mathbf{d} & 1 \end{array} \right|$$

Marco Gribaudo

Associate Professor

CONTACTS

Tel. +39 02 2399 3568
marco.gribaudo@polimi.it
https://www.deib.polimi.it/eng/home-page

(Remember to use the phone, since mails might require a lot of time to be answered. Microsoft Teams messages might also be faster than regular mails)