3D Transforms and projections

Inversion of transformations

Transformations can be inverted to return an object to its original position, size or orientation.

A nice feature about using matrices is that the inverse transformation can be obtained by inverting the matrix.

If point *p'* is obtained by applying the transformation coded into matrix *M* to point *p*, then point *p* can be obtained back from point *p'* by multiplying it with the inverse of the matrix M^{-1}.

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

$$p' = M \cdot p$$

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

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

$$p = M^{-1} \cdot p'$$

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

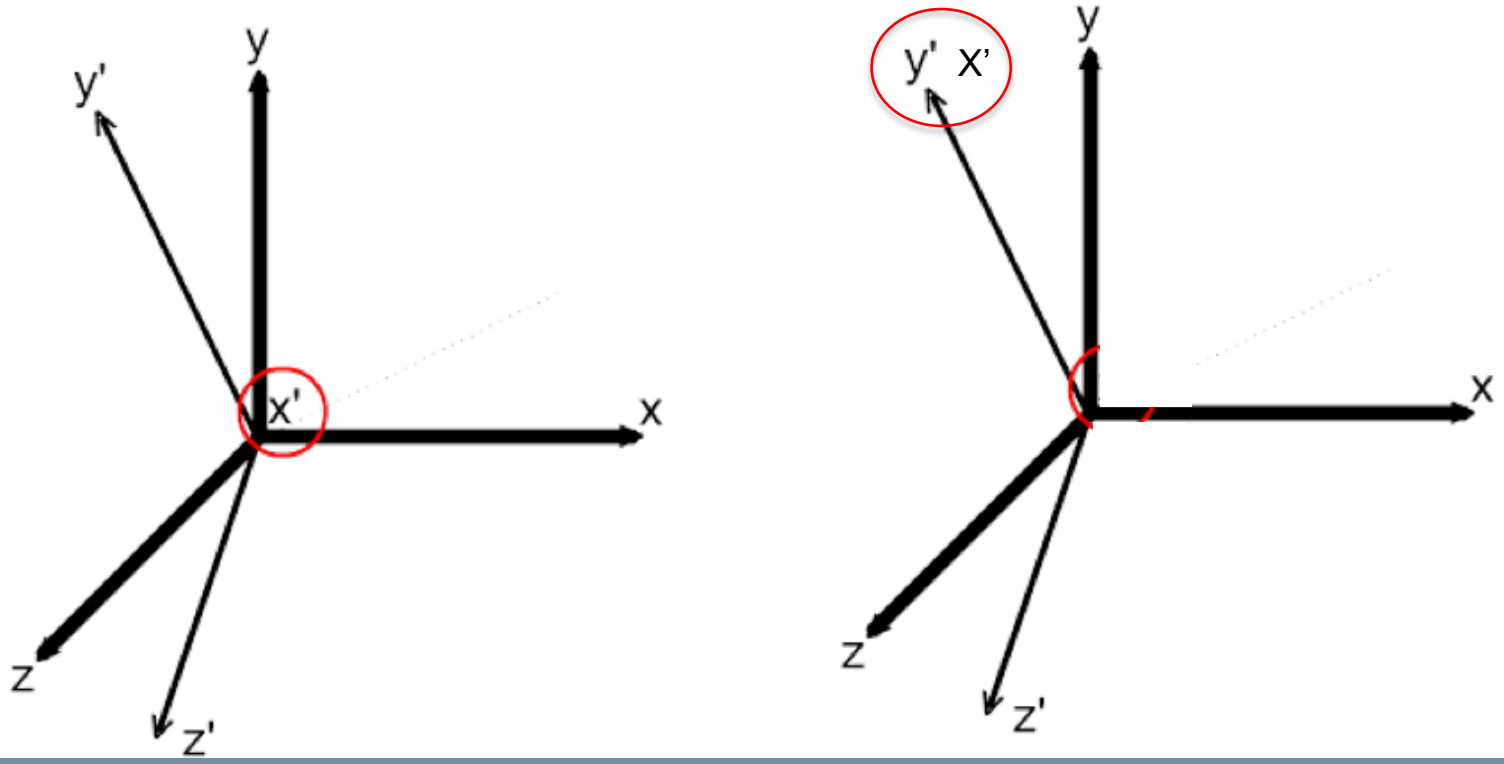
Inversion of transformations

Since the last row is |0 0 0 1|, it can be proven that a transform matrix M is invertible, if its sub-matrix composed by the first 3 rows and 3 columns is invertible.

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

Inversion of transformations

This is generally the case, except when for some reason one or more of the projected axis degenerates to be of zero length, or when two axis perfectly overlaps.



Inversion of transformations

The inverse of a general transformation matrix can be obtained by applying numerical inversion techniques. Closed formulas also exists.

$$M_C = \left|\begin{array}{cccc} v_x & v_y & v_z & c \\ 0 & 0 & 0 & 1 \end{array}\right| = \left|\begin{array}{cc} R_C & c \\ 0 & 1 \end{array}\right|$$

$$M_V = \left[M_C\right]^{-1} = \left|\begin{array}{cc} \left(R_C\right)^T & -\left(R_C\right)^T \cdot c \\ 0 & 1 \end{array}\right|$$

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

$$\operatorname{adj}(\mathbf{A}) = \begin{pmatrix} +\begin{vmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{vmatrix} & -\begin{vmatrix} A_{12} & A_{13} \\ A_{32} & A_{33} \end{vmatrix} & +\begin{vmatrix} A_{12} & A_{13} \\ A_{22} & A_{23} \end{vmatrix} \\ -\begin{vmatrix} A_{21} & A_{23} \\ A_{31} & A_{33} \end{vmatrix} & +\begin{vmatrix} A_{11} & A_{13} \\ A_{31} & A_{33} \end{vmatrix} & -\begin{vmatrix} A_{11} & A_{13} \\ A_{21} & A_{23} \end{vmatrix} \\ +\begin{vmatrix} A_{21} & A_{22} \\ A_{31} & A_{32} \end{vmatrix} & -\begin{vmatrix} A_{11} & A_{12} \\ A_{31} & A_{32} \end{vmatrix} & +\begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \end{pmatrix}$$

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\det(A) = ad - bc$$

det(A)= a11(a22*a33 - a32*a23) - a21(a12*a33 - a32*a13) + a31(a12*a23 - a22*a13)

$$A^{-1} = \frac{adj\,(A)}{det\,(A)}$$

However the inverse of the transformations previously presented can be determined with simple rules.

Inversion of transformations

For translation:

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

$$\left[T(d_x, d_y, d_z)\right]^{-1} = T(-d_x, -d_y, -d_z) = \begin{vmatrix} 1 & 0 & 0 & -d_x \\ 0 & 1 & 0 & -d_y \\ 0 & 0 & 1 & -d_z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Inversion of transformations

For scaling:

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

$$\left[S(s_x, s_y, s_z)\right]^{-1} = S(1/s_x, 1/s_y, 1/s_z) = \begin{vmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Note that *1/s* can be computed only if *s != 0*.

This explains why a transformation cannot be inverted if the length of one axis is reduced to 0.

Inversion of transformations

For rotation:

$$R_x(\alpha) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad \left[R_x(\alpha)\right]^{-1} = R_x(-\alpha) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$R_y(\alpha) = \begin{vmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad \left[R_y(\alpha)\right]^{-1} = R_y(-\alpha) = \begin{vmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$R_z(\alpha) = \begin{vmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad \left[R_z(\alpha)\right]^{-1} = R_z(-\alpha) = \begin{vmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Inversion of transformations

Example:

Consider a scaling of 2.5 on the y-axis and 1/3 on z-axis. The transform matrix and its inverse are the following:

$$S(1,2.5,1/3) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 2.5 & 0 & 0 \\ 0 & 0 & 0.33333 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$\left[S(1,2.5,1/3)\right]^{-1} = S(1,1/2.5,1/(1/3)) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.4 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Composition of transformations

During the creation of a scene, an object is subject to several transformations.

The application of a sequence of transformation is called **composition**.

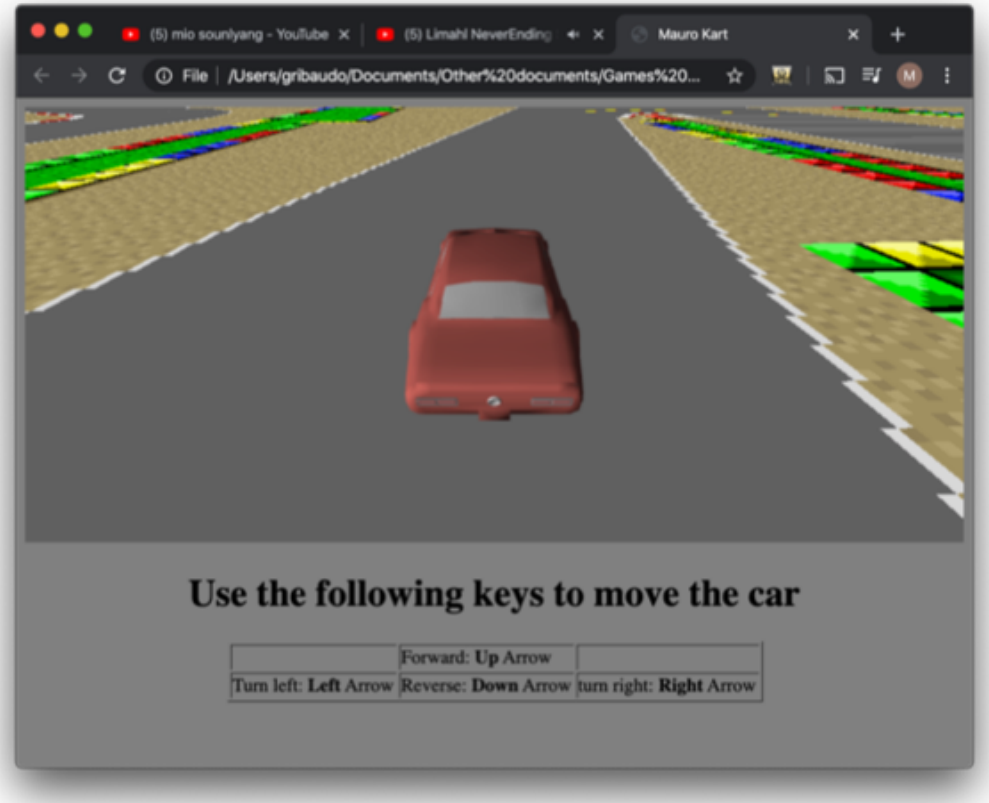
Usually, an object should be translated and rotated in all directions to be positioned in the scene.

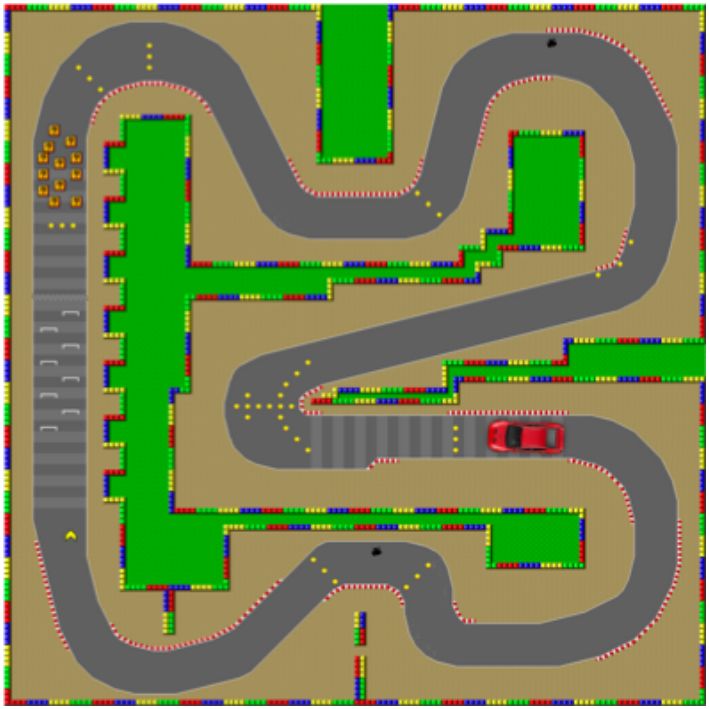
Moreover, rotations among arbitrary axis and scaling with different centers can be performed by composing different transformations.

Thanks to the properties of matrix product, composition of transformations can be be done in a very efficient way.

Composition of transformations

Let us consider a simple racing game, where we want to position a car running on a flat track.

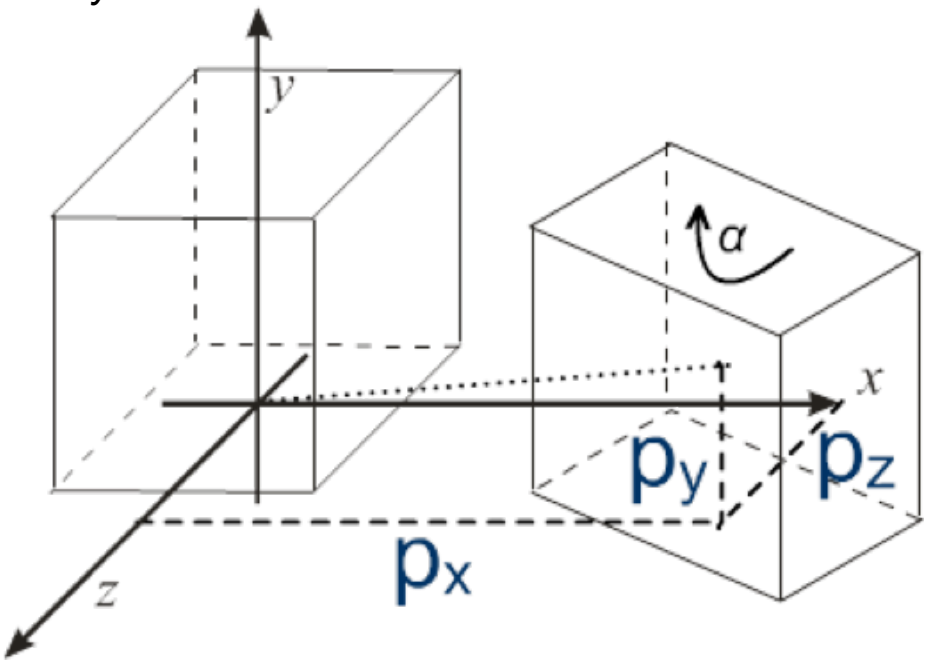


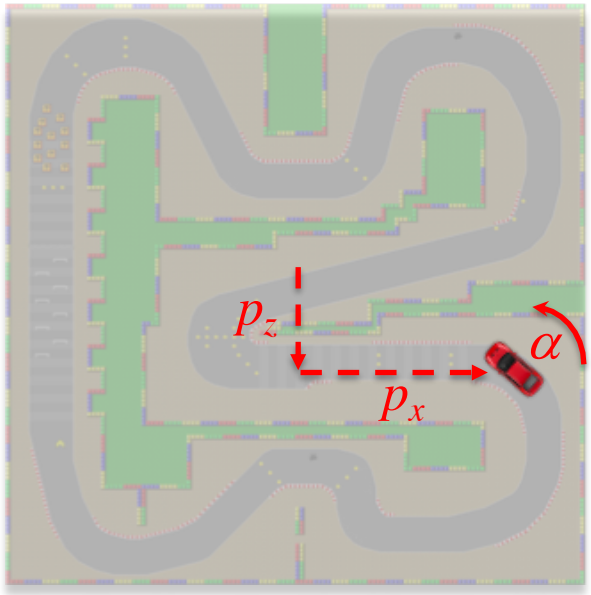


Composition of transformations

To simplify the presentation, we will display the car as a cube, with its sides parallel to the *x*, *y*, and *z* axes, and with its center in the origin.

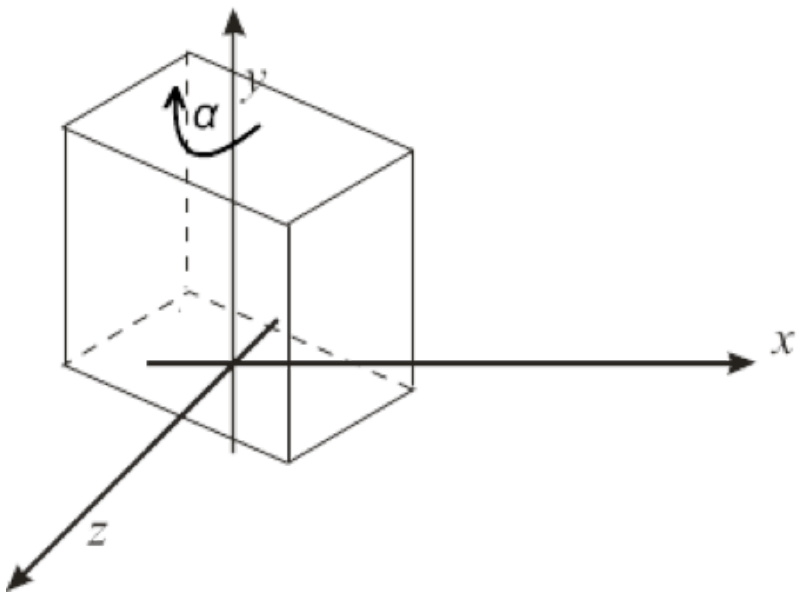
The goal is to translate the car, so to have its center in position (p_x, p_y, p_z), and its sides are oriented at an angle α around the *y* axis.





Composition of transformations

The goal can be reached by first rotating all the car vertices of angle α around the *y*-axis (which can be obtained by a rotation matrix $R_y(\alpha)$).

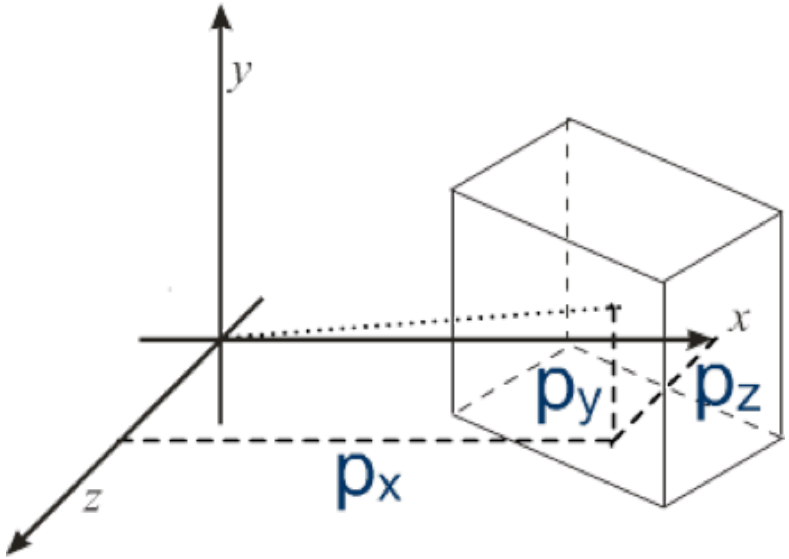


$$R_y(\alpha) = \begin{vmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$p' = R_y(\alpha) \cdot p$$

Composition of transformations

Then the rotated car can be translated in position (p_x, p_y, p_z) using a translation matrix $T(p_x, p_y, p_z)$.



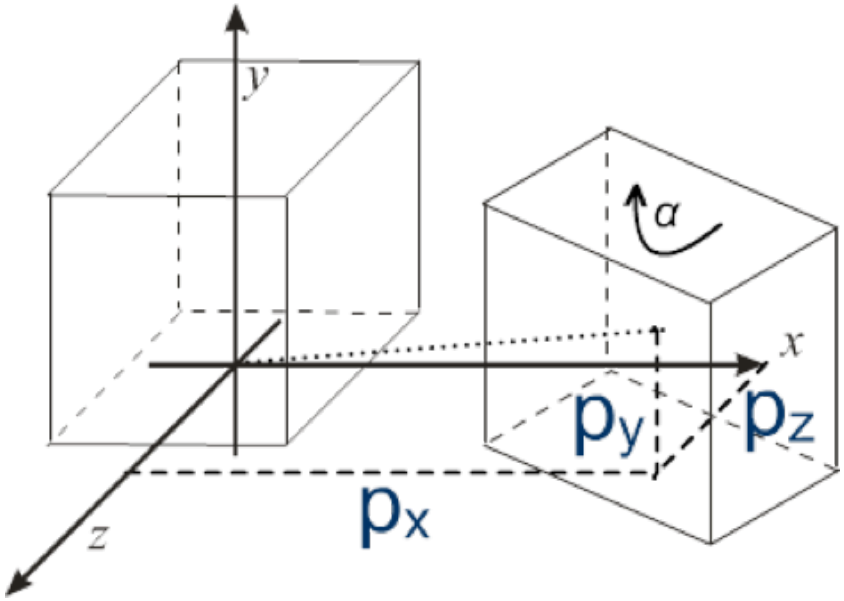
$$T(p_x, p_y, p_z) = \begin{vmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$p' = R_y(\alpha) \cdot p$$

$$p'' = T(p_x, p_y, p_z) \cdot p' = T(p_x, p_y, p_z) \cdot R_y(\alpha) \cdot p$$

Composition of transformations

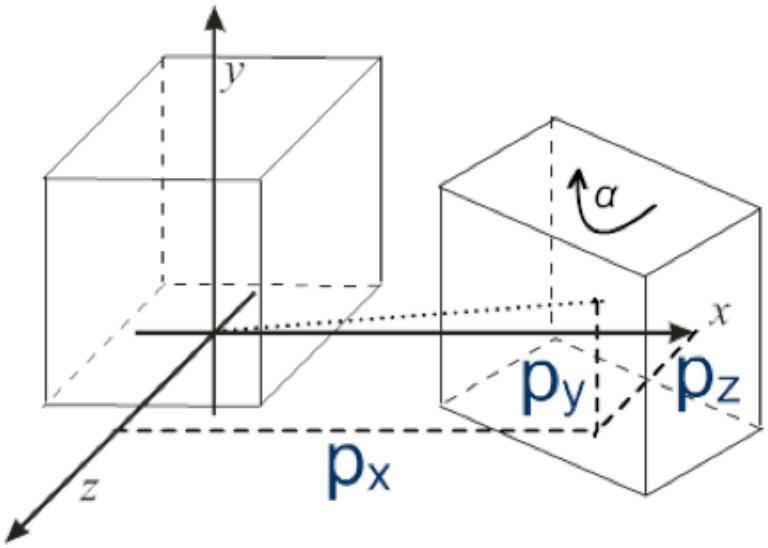
Note that matrices appear in reverse order with respect to the transformations they represent.



$$p'' = T(p_x, p_y, p_z) \cdot R_y(\alpha) \cdot p$$

Composition of transformations

(In case of the *matrix-on-the-right* convention the order of transformation is from left to right)



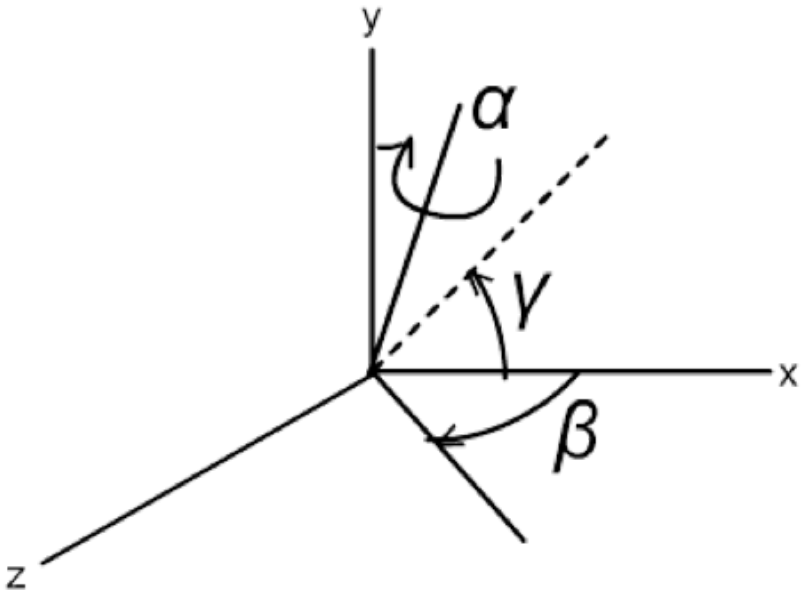
In this course, we will never use the matrix-on-the-right convention.

$$p'' = p \cdot R_y(\alpha)^T \cdot T(p_x, p_y, p_z)^T$$

Transformations around an arbitrary axis or center

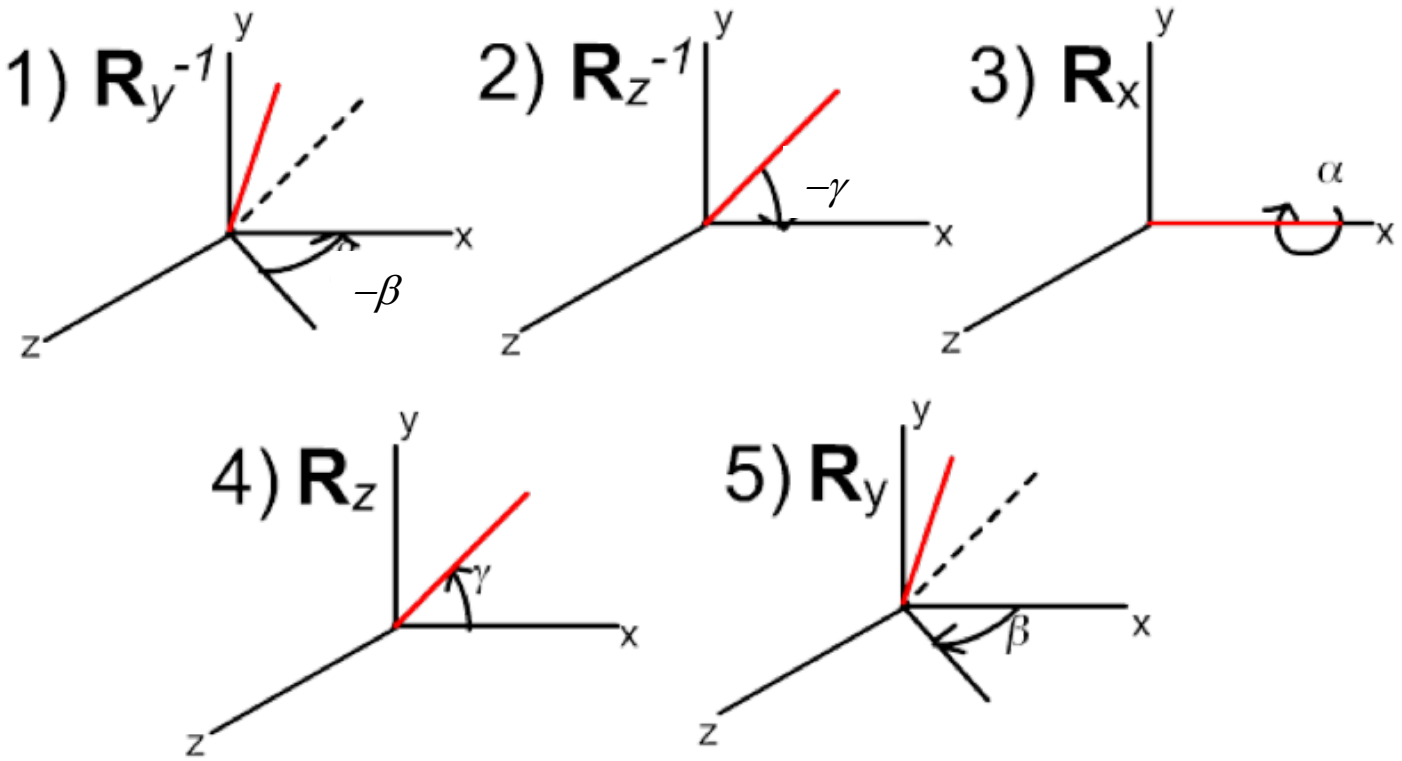
Now consider the rotation of an object of an angle α about an arbitrary axis that passes through the origin.

In the considered scenario, the x-axis can be aligned to the arbitrary axis by first rotating an angle γ around the z-axis, and then an angle β around the y-axis.



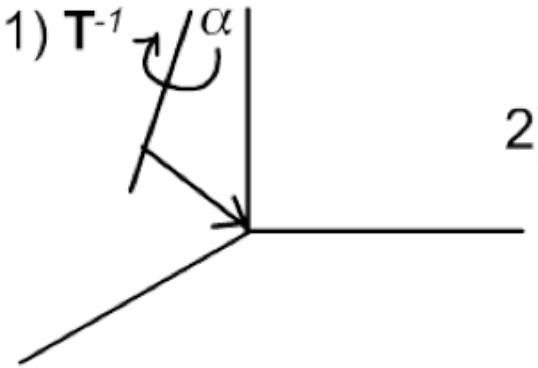
Transformations around an arbitrary axis or center

The arbitrary axis rotation can be obtained by combining 5 elementary transformations:

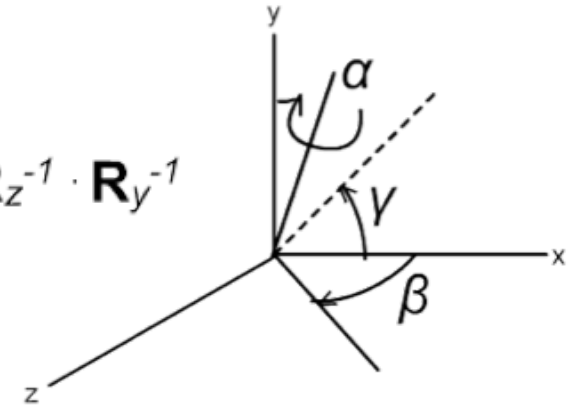


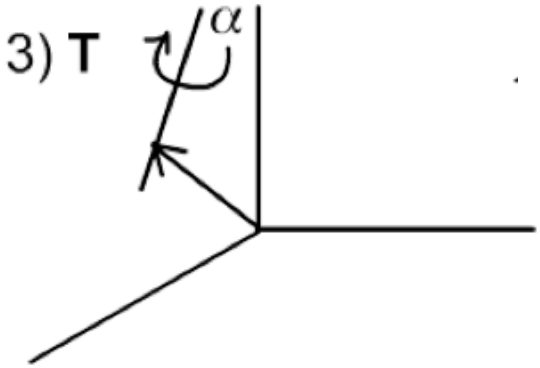
Transformations around an arbitrary axis or center

If the axis does not pass through the origin, a translation T must be applied to make it pass through (0,0,0).



2) $\mathbf{R}_y \cdot \mathbf{R}_z \cdot \mathbf{R}_x \cdot \mathbf{R}_z^{-1} \cdot \mathbf{R}_y^{-1}$





Transformations around an arbitrary axis or center

To summarize, a rotation of α about an arbitrary axis passing through point (p_x, p_y, p_z) and such that the x-axis can be aligned to it by first rotating an angle γ around the z-axis, and then an angle β around the y-axis, can be computed as:

$$p' = T(p_x, p_y, p_z) \cdot R_y(\beta) \cdot R_z(\gamma) \cdot R_x(\alpha) \cdot R_z(\gamma)^{-1} \cdot R_y(\beta)^{-1} \cdot T(p_x, p_y, p_z)^{-1} \cdot p$$

Similar procedures can be followed if we know different rotation sequences, that might also align the arbitrary axis with another of the three main axis (i.e. how to align it with the z-axis).

Transformations around an arbitrary axis or center

Similar considerations can be applied for scaling an object along an arbitrary direction, with an arbitrary center.

They can also be applied to generalize shear, and to perform symmetries about arbitrary planes, axes or centers.

Since there is an extremely large number of possibilities, determining the right sequence of transformations to obtain a specific result is left to the programmer.

Transformations around an arbitrary axis or center

Example:

A sequence of transformations that performs a uniform scaling of 2, centered at (3,2,1) is the following:

$$p' = T(p_x, p_y, p_z) \cdot S(s_x, s_y, s_z) \cdot T(p_x, p_y, p_z)^{-1} \cdot p$$

$$p' = T(3,2,1) \cdot S(2,2,2) \cdot T(3,2,1)^{-1} \cdot p$$

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

Matrix transformations: is it worth it?

One of the question that usually comes up when seeing transform matrix is: is it worth it?

For example, in the car example just given, if I know the original position of a vertex of the object, (x, y, z), wouldn't be easier to directly compute it?

$$\begin{cases} x' = p_x + x \cdot \cos\alpha - z \cdot \sin\alpha \\ y' = p_y + y \\ z' = p_z + z \cdot \cos\alpha + x \cdot \sin\alpha \end{cases}$$

Properties of composition of transformations

The product of two matrices, and of a matrix and a vector is associative:

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

A, B and C are 4x4 matrices.

$$A \cdot (B \cdot p) = (A \cdot B) \cdot p$$

p is a vector

We can exploit this feature to factorize all the transforms in a single matrix.

Properties of composition of transformations

Instead of repeatedly multiplying the coordinates by different matrices, a single matrix corresponding to the product of all the transformations can be computed (using the associative property).

Then the composed matrix can be used to apply all the transformations in a single product.

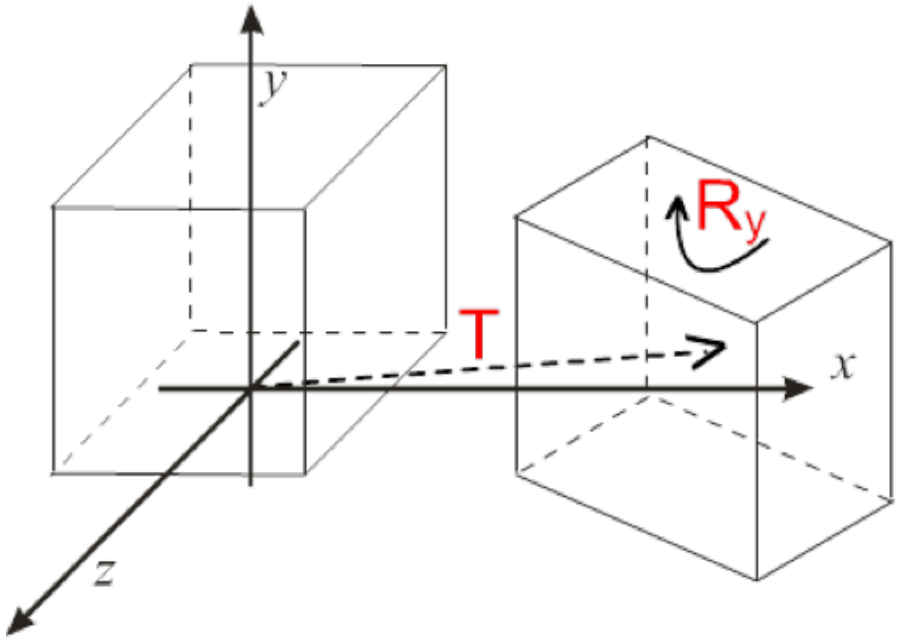
$$p' = T(p_x, p_y, p_z) \cdot R_y(\beta) \cdot R_z(\gamma) \cdot R_x(\alpha) \cdot R_z(\gamma)^{-1} \cdot R_y(\beta)^{-1} \cdot T(p_x, p_y, p_z)^{-1} \cdot p$$

$$R_{p_x,p_y,p_z,\beta,\gamma}(\alpha) = T(p_x, p_y, p_z) \cdot R_y(\beta) \cdot R_z(\gamma) \cdot R_x(\alpha) \cdot R_z(\gamma)^{-1} \cdot R_y(\beta)^{-1} \cdot T(p_x, p_y, p_z)^{-1}$$

$$p' = R_{p_x,p_y,p_z,\beta,\gamma}(\alpha) \cdot p$$

Properties of composition of transformations

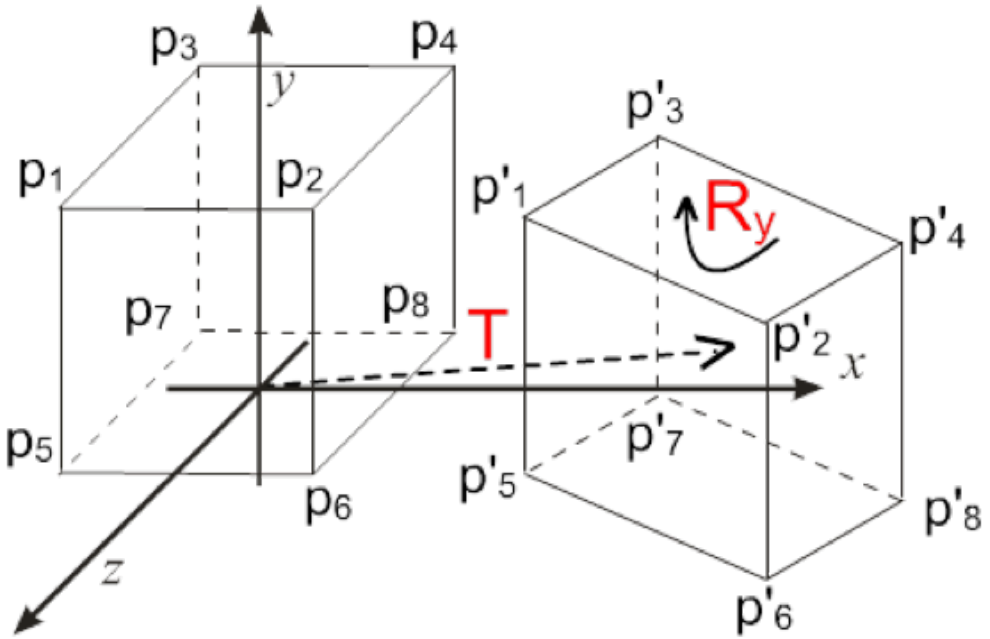
Computing the composed matrix greatly improves the performance of executing multiple transformations.



$$p' = T \cdot R_y \cdot p$$

Properties of composition of transformations

Usually, an object is composed by a large number of points (in current applications, between 10^4 and 10^6) that require the same transformation.



$$p'_1 = T \cdot R_y \cdot p_1$$
$$p'_2 = T \cdot R_y \cdot p_2$$
$$\vdots$$
$$p'_8 = T \cdot R_y \cdot p_8$$

Properties of composition of transformations

The transformation matrix is computed once per object.

Then all the n_v points are transformed by multiplying with the same matrix.

$$p'_1 = T \cdot R_y \cdot p_1$$
$$p'_2 = T \cdot R_y \cdot p_2$$
$$\vdots$$
$$p'_8 = T \cdot R_y \cdot p_8$$

2 n_v MxV products

$$M = T \cdot R_y$$
$$p'_1 = M \cdot p_1$$
$$p'_2 = M \cdot p_2$$
$$\vdots$$
$$p'_8 = M \cdot p_8$$

n_v (+4) MxV products

Composition of transformation: inversion

Recall that the inverse of the product of two matrices, is the product of the inverse matrices in the opposite order:

$$(A \cdot B)^{-1} = B^{-1} \cdot A^{-1}$$

This can be used to compute the inverse of a composed transform.

Example:

$$\left[R_{p_x,p_y,p_z,\beta,\gamma}(\alpha)\right]^{-1} = \left[T(p_x,p_y,p_z)\cdot R_y(\beta)\cdot R_z(\gamma)\cdot R_x(\alpha)\cdot R_z(\gamma)^{-1}\cdot R_y(\beta)^{-1}\cdot T(p_x,p_y,p_z)^{-1}\right]^{-1} =$$

$$= \left[T(p_x,p_y,p_z)^{-1}\right]^{-1}\cdot\left[R_y(\beta)^{-1}\right]^{-1}\cdot\left[R_z(\gamma)^{-1}\right]^{-1}\cdot R_x(\alpha)^{-1}\cdot R_z(\gamma)^{-1}\cdot R_y(\beta)^{-1}\cdot T(p_x,p_y,p_z)^{-1} =$$

$$= T(p_x,p_y,p_z)\cdot R_y(\beta)\cdot R_z(\gamma)\cdot R_x(\alpha)^{-1}\cdot R_z(\gamma)^{-1}\cdot R_y(\beta)^{-1}\cdot T(p_x,p_y,p_z)^{-1} = R_{p_x,p_y,p_z,\beta,\gamma}(-\alpha)$$

Composition of transformation: inversion

Example:

An object is translated of *(1, 2, 3)*, then rotated 30° around the *y* axis.

The inverse transform can be computed without inverting the matrix as follows:

$$M = R_y(30^\circ) \cdot T(1,2,3)$$

$$M^{-1} = T(1,2,3)^{-1} \cdot R_y(30^\circ)^{-1}$$

$$M^{-1} = \begin{vmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 0.866 & 0 & -0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0.866 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0.866 & 0 & -0.5 & -1 \\ 0 & 1 & 0 & -2 \\ 0.5 & 0 & 0.866 & -3 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Composition of transformation: not commutative

Note that matrix product is not commutative.

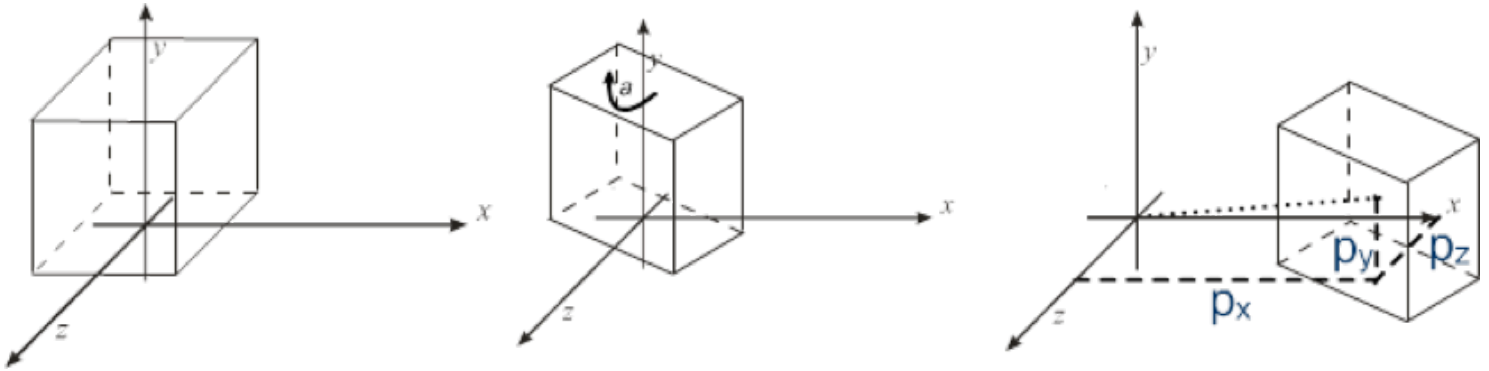
$$A \cdot B \neq B \cdot A$$

This means that the order of transformation is important.

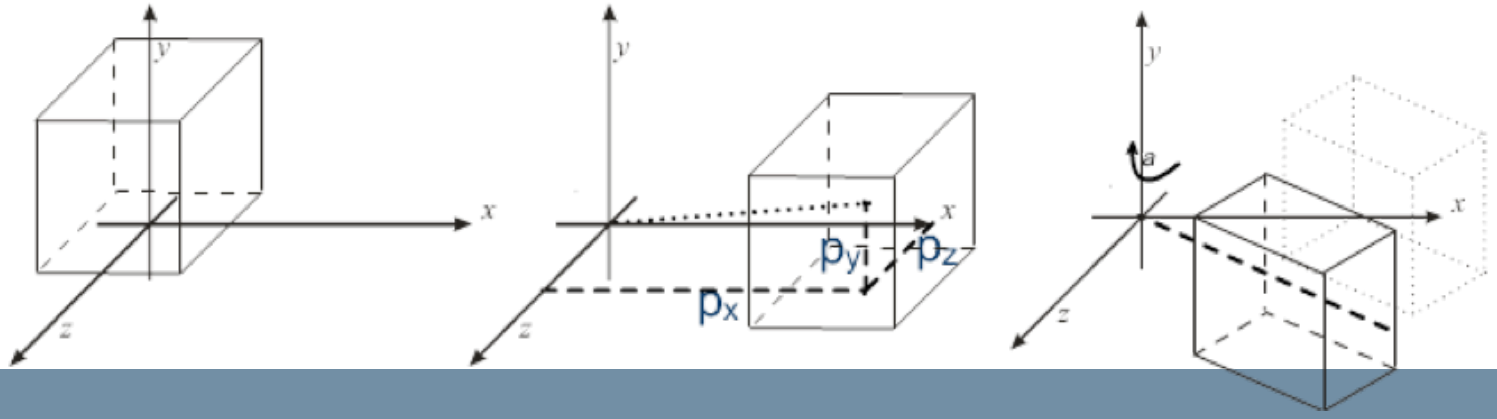
It also means that two transformations cannot be swapped without obtaining a different result.

Composition of transformation: not commutative

For example if we first rotate a cube, and then perform a translation we obtain one result.



If we maintain the same parameters, but invert the order of the transformations, we obtain a different result.



Projections

In 3D computer graphics, the goal is to represent a three-dimensional space on a screen.

The screen has only two dimensions.

Even when considering stereoscopic images, the sensation of the third dimension is given by the way in which the human brain interprets two different 2D images sent to the left and to the right eyes.

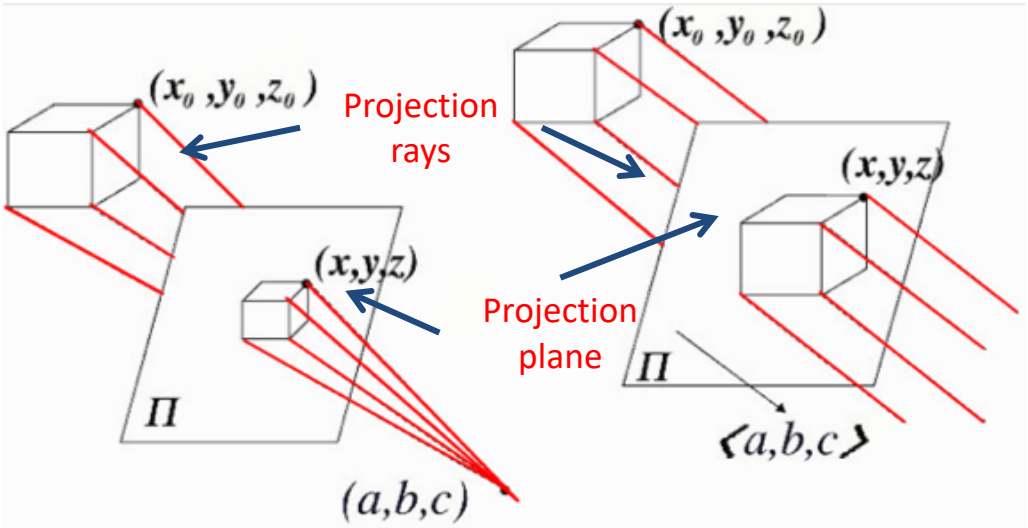
3D computer graphics uses geometrical primitives to describe objects defined in three dimensions.

Then it produces a 2D representation of the scene, and shows it on the screen.

Projections

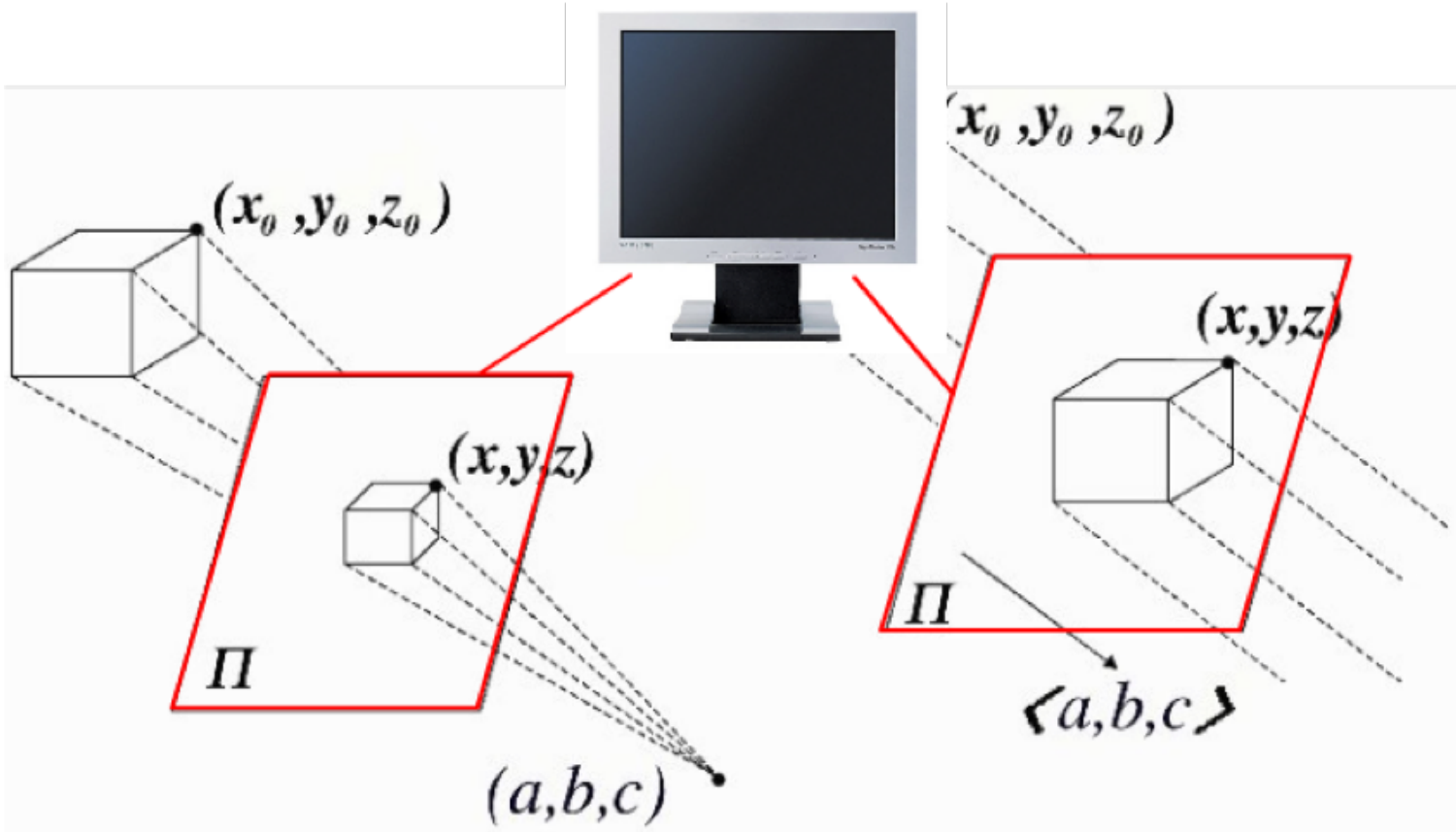
The technique used to construct a 2D image from a 3D scene is called *projection*.

The 2D representation of the 3D object is defined by the intersection of a set of *projection rays* with a plane called the *projection plane*.



Projections

The projection plane represents the screen.



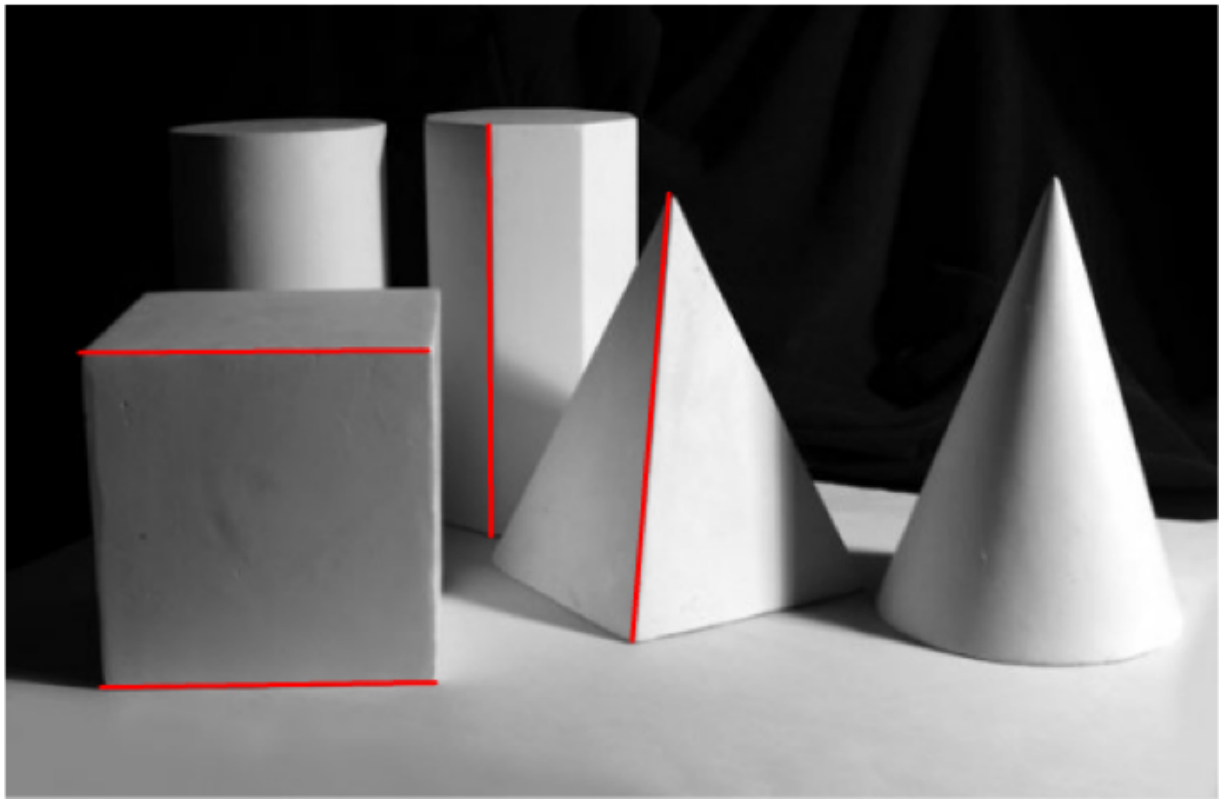
Projections

The idea of correctly represent a 3D object over a 2D surface with projections was studied in the XVI century.

DÜRER, Albrecht (1471-1528). Institutiones geometricae - 1522

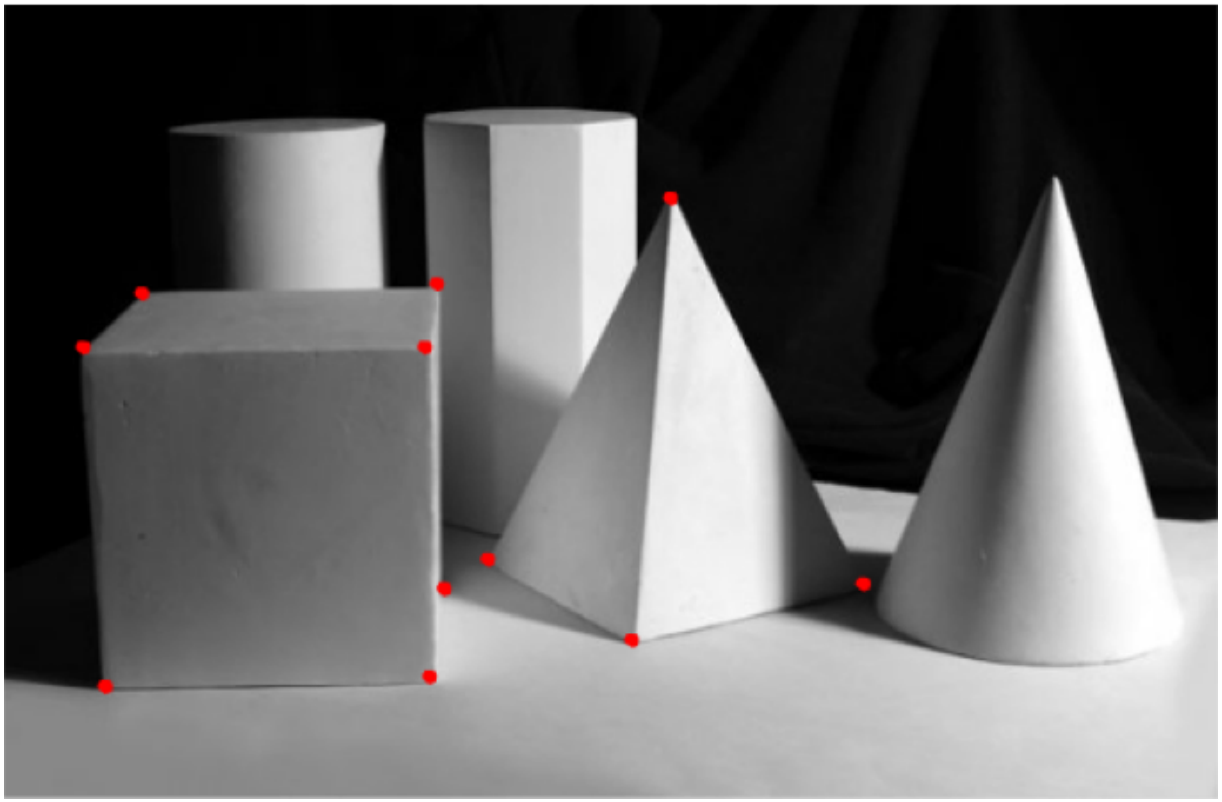
Projections

One important property is that the projections of linear segments remain linear segments.



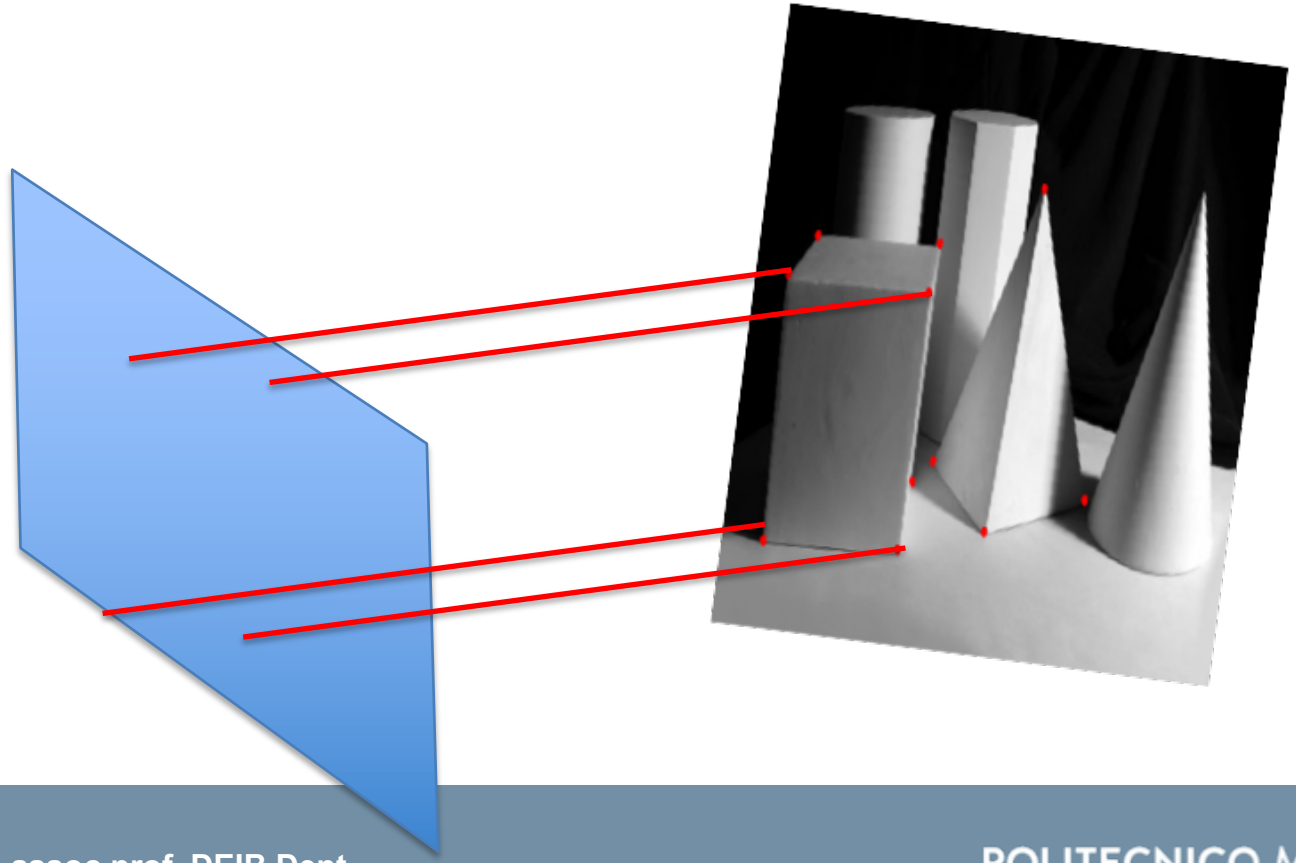
Projections

Moreover, the projected segments connect the projections of the end points.



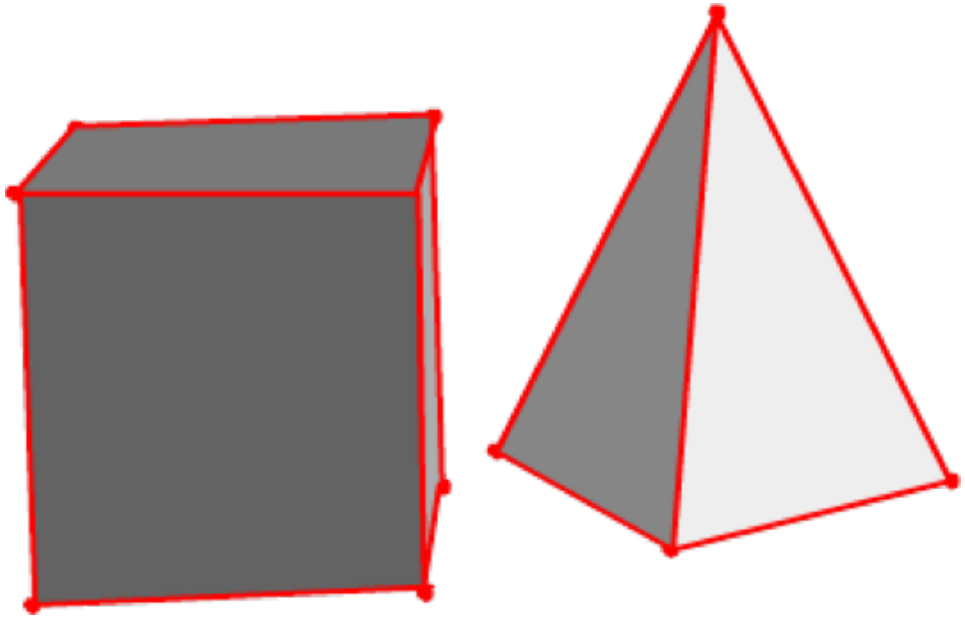
Projections

To recreate a 2D representation of 3D polyhedron is thus sufficient to project its vertices.



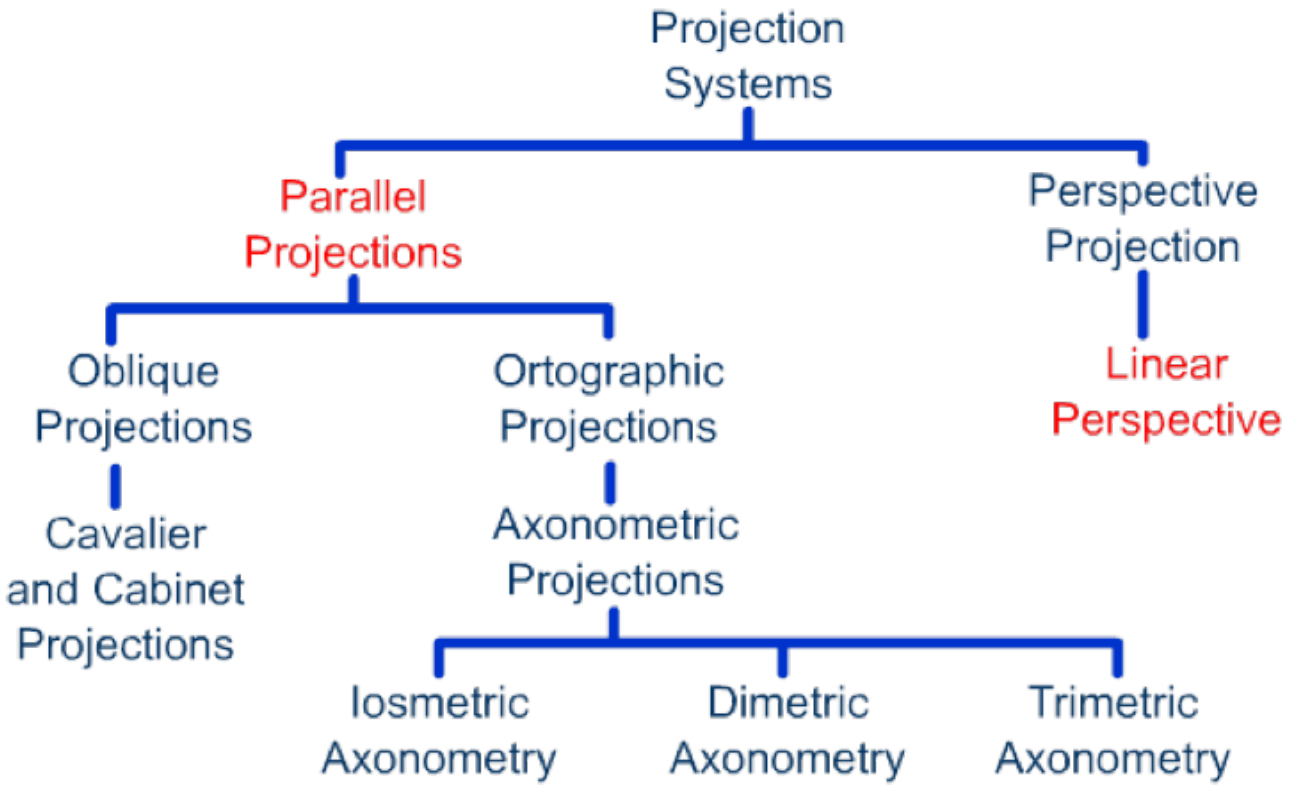
Projections

And connect them with straight segments or polygons in 2D.



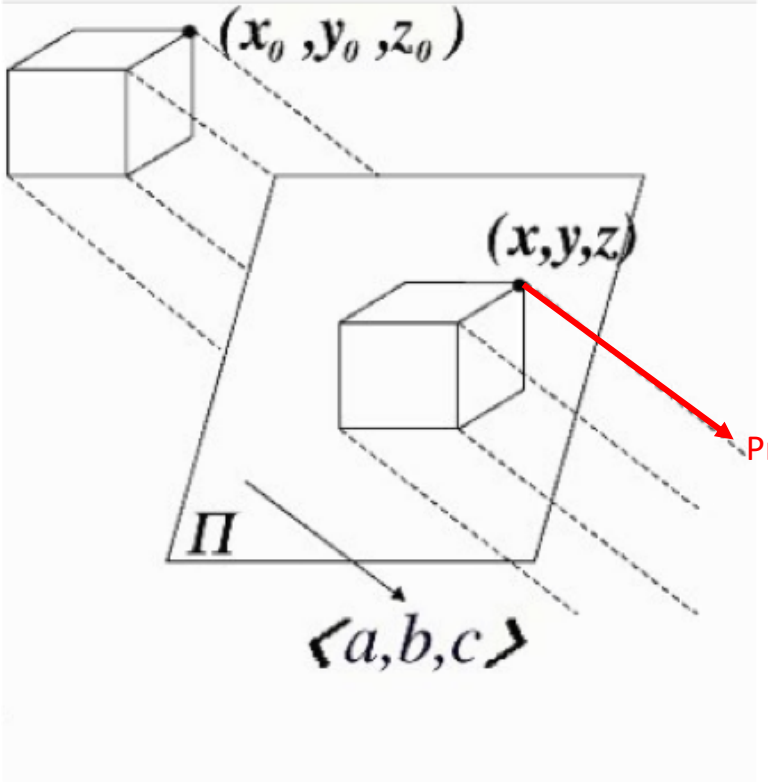
Projection types

Many types of projections have been defined in the literature. In this course we will consider only *parallel projections* and *perspective*.



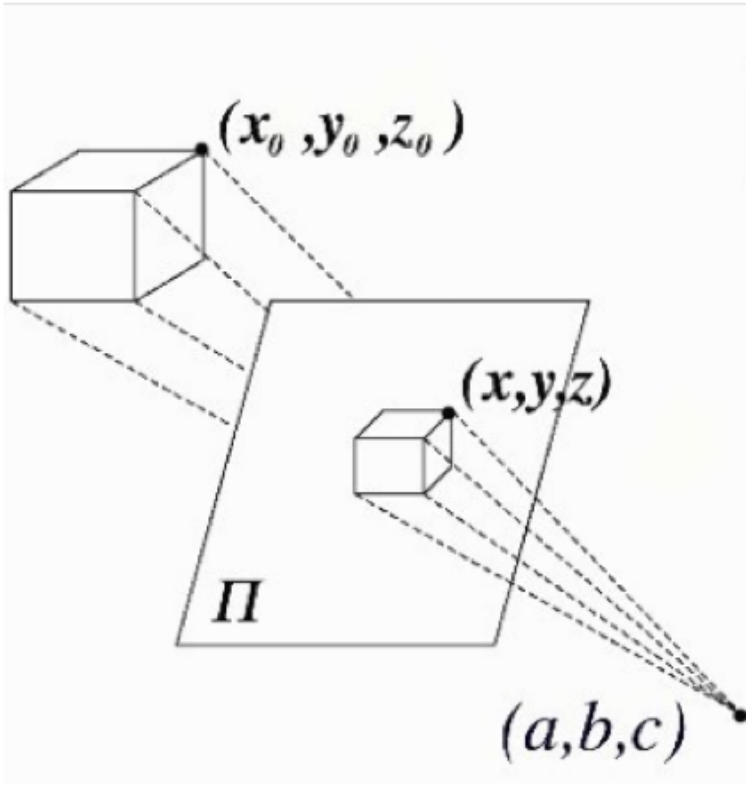
Projection types

In *parallel projections*, all rays are parallel to the same *direction*.



Projection types

In *perspective projections*, all the rays pass through a point, called the *center of projection*.



Projection properties

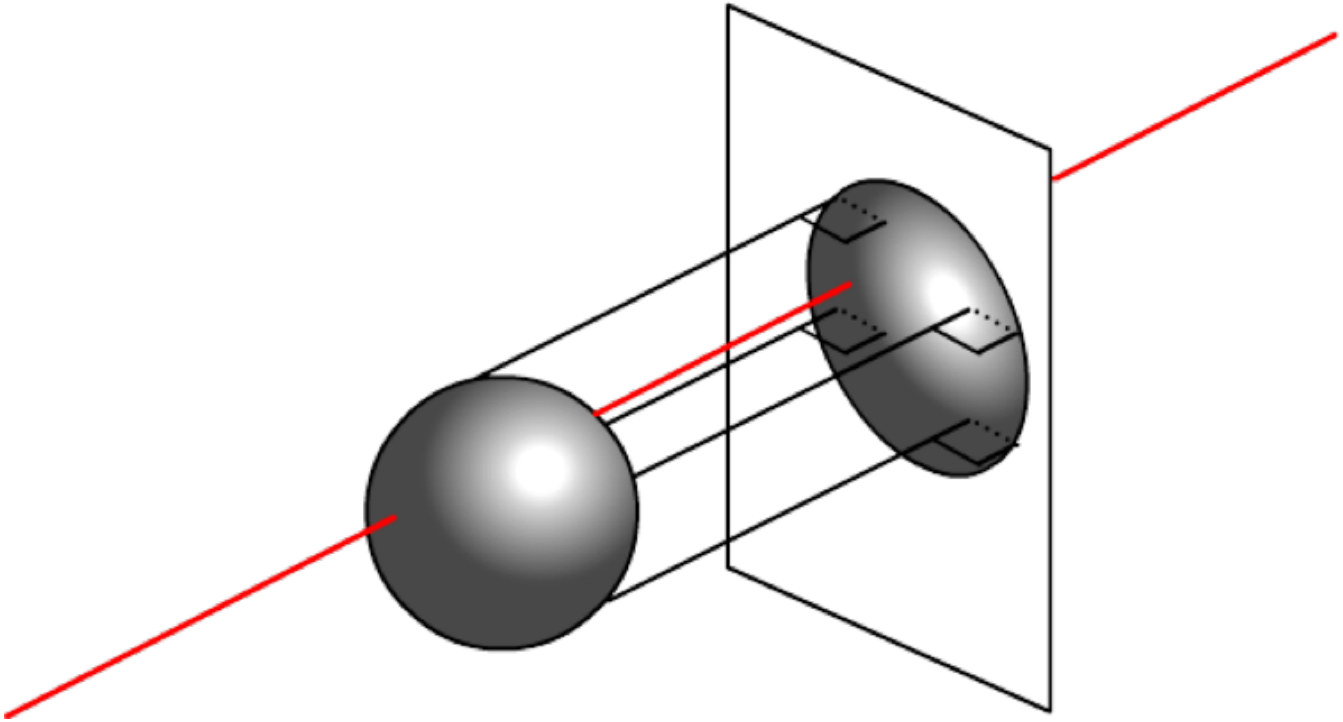
For both parallel and perspective projections, a point on the screen corresponds to an infinite number of coordinates in the space.

This is a direct consequence of loosing a component when moving from a 3D spatial system, to a 2D screen system.

In particular, in both parallel and perspective projections any pixel on the screen corresponds to a line of points in the 3D space.

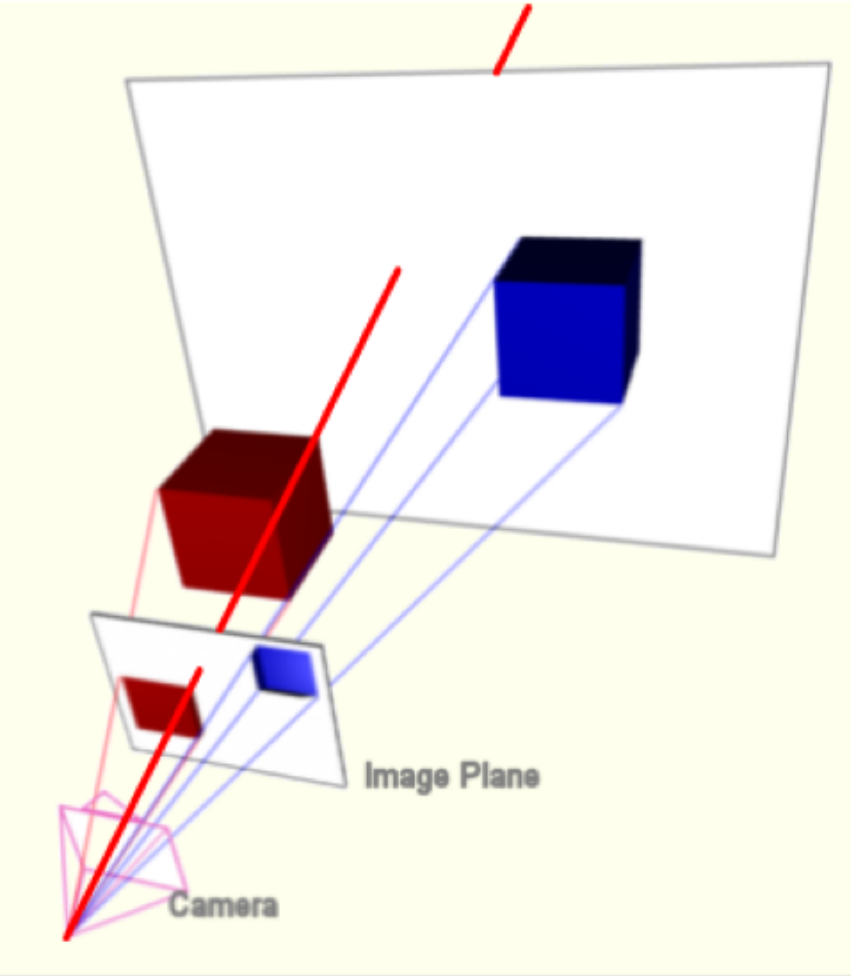
Projection properties

In parallel projections, all points that pass through a line parallel to the projection ray are mapped to same pixel.



Projection properties

In perspective projection, all points that are aligned with both the projected pixel and the center of projection are mapped to the same location.



Projection in 3D computer applications

In 3D computer applications, the projections are implemented with a conversion of 3D coordinates between two reference systems.

In particular projections transform:

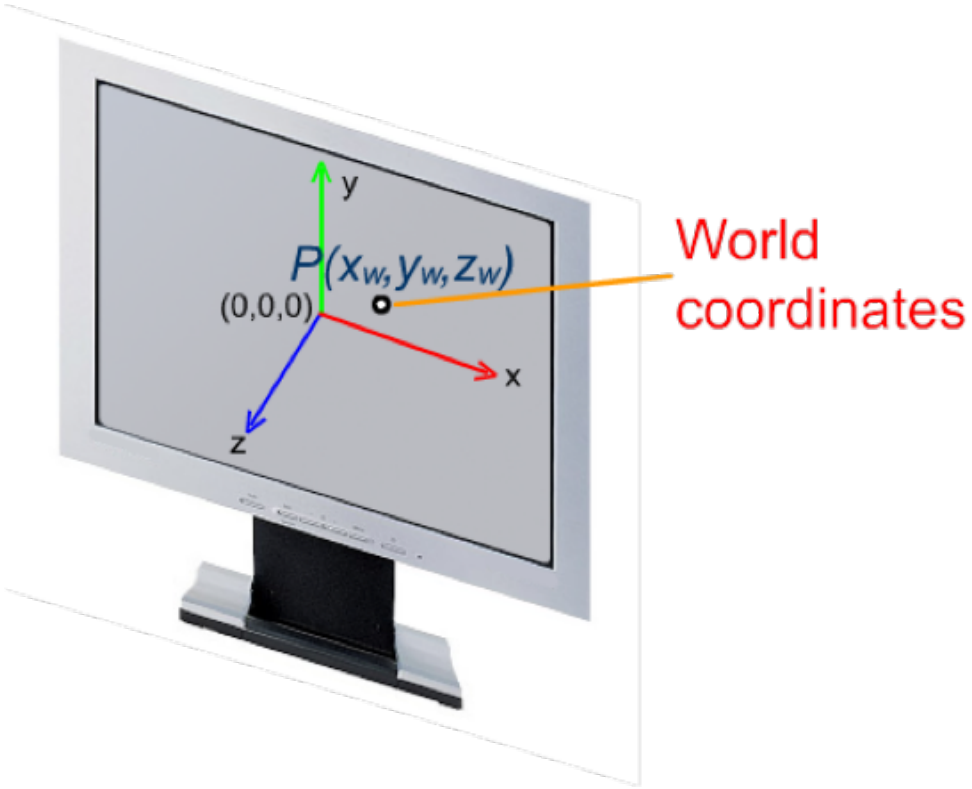
World Coordinates

Into:

3D Normalized Screen Coordinates

World Coordinates

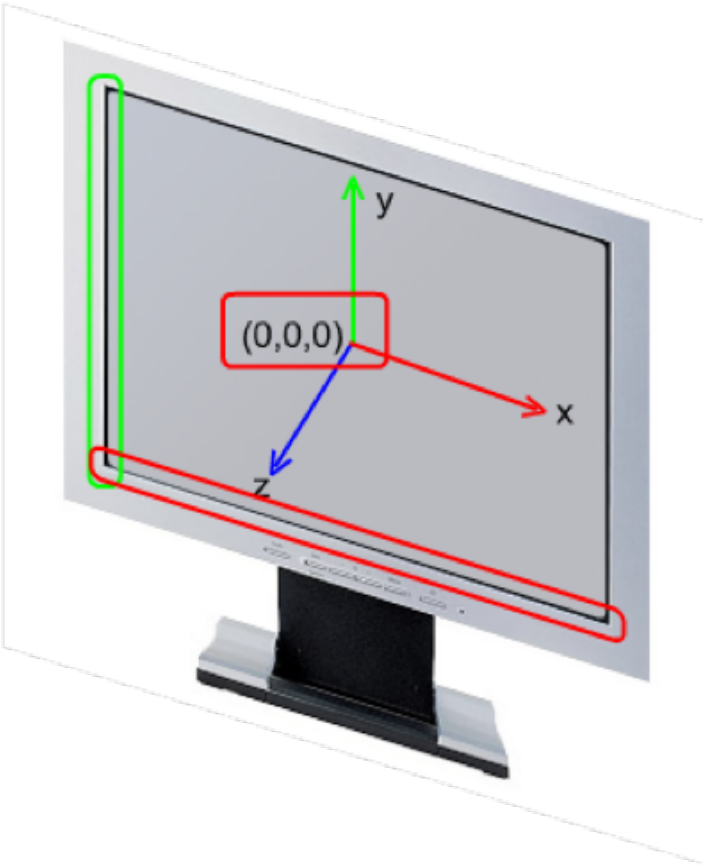
The coordinates system that describes the objects in the 3D space is called *World Coordinates* (or *global coordinates*).



World Coordinates

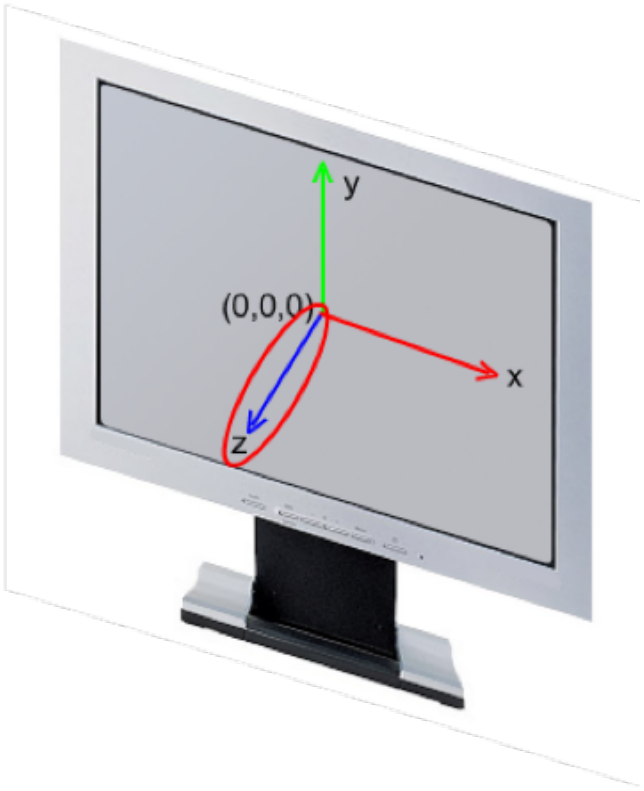
World Coordinates is a right-handed Cartesian coordinate system with the origin in the center of the screen.

The version we will considered in this course is called *“y-up”: it* has the *x* and *y-axes* parallel respectively to the horizontal and vertical edges of the screen.



World Coordinates

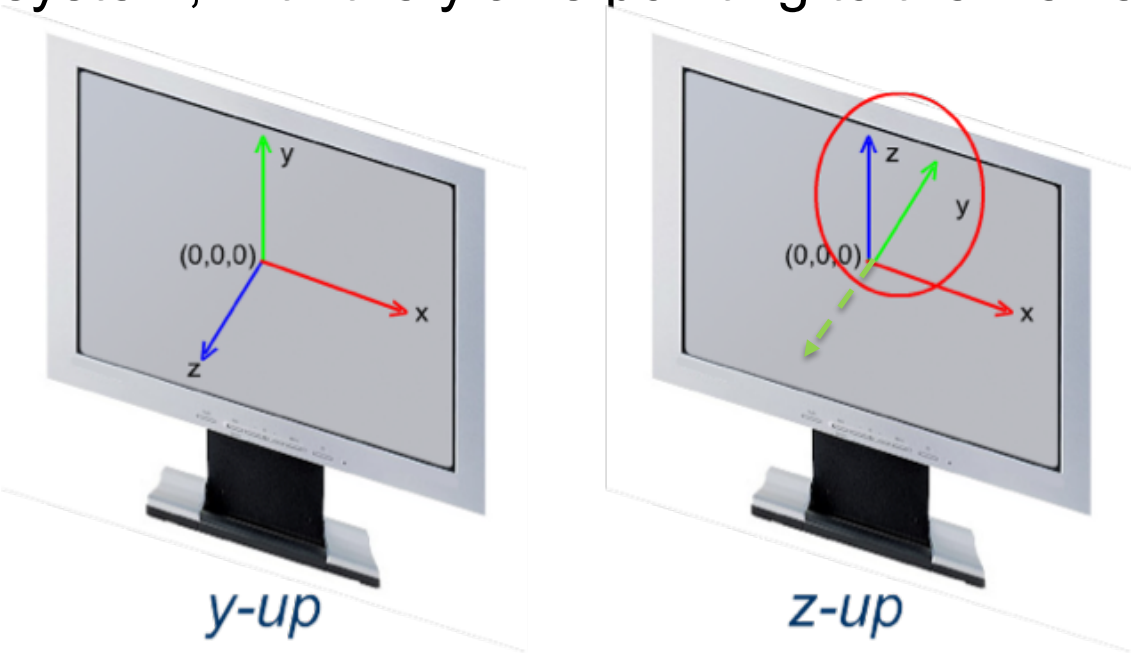
The *z-axis* is directed “out of the screen”, towards the viewer.



World Coordinates

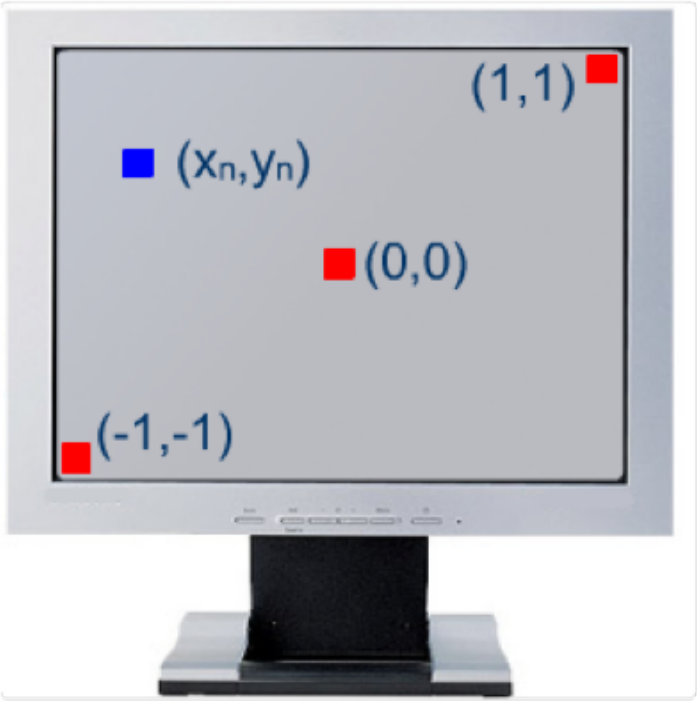
Note that many applications (i.e. *Blender*) use another convention for World Coordinates called *"z-up"*. In this case the *z-axis* is oriented along the vertical screen direction, and the *y-axis* "inside the screen". Some applications even use a left-handed system, with the y axis pointing to the viewer.

In this course we focus on the y-up system only.



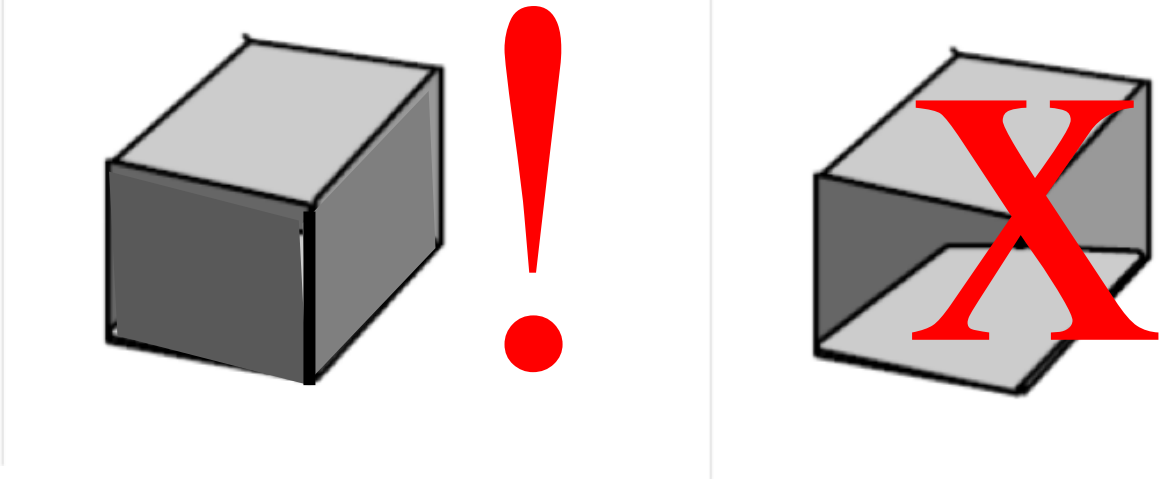
3D Normalized Screen Coordinates

As introduced, *Normalized Screen Coordinates* allow to specify the positions of points on a screen (or on a window) in a device independent way.



3D Normalized Screen Coordinates

However, even if the screen is a 2D surface, pixels coming from 3D images must be characterized by a “distance from the viewer” to allow sorting the surfaces in a correct order, and prevent the construction of unrealistic images (this will be covered later in the course).



3D Normalized Screen Coordinates

3D Normalized Screen Coordinates have a third component, also ranging in the same extents (i.e. -1,+1).

Coordinates with smaller *z* value are considered to be closer to the viewer.

Note that this system is "*left-handed*", with the *z-axis* oriented in the opposite direction with respect to *World Coordinates*.

In the following, all normalized screen coordinates will be considered to be in 3D.