POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

D E
I B

2023

Dipartimento di Elettronica, Informazione e Bioingegneria

Computer Graphics

Milano, 2023

Computer Graphics

- Parallel Projections

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 separate 2D images sent to the left and to the right eye.

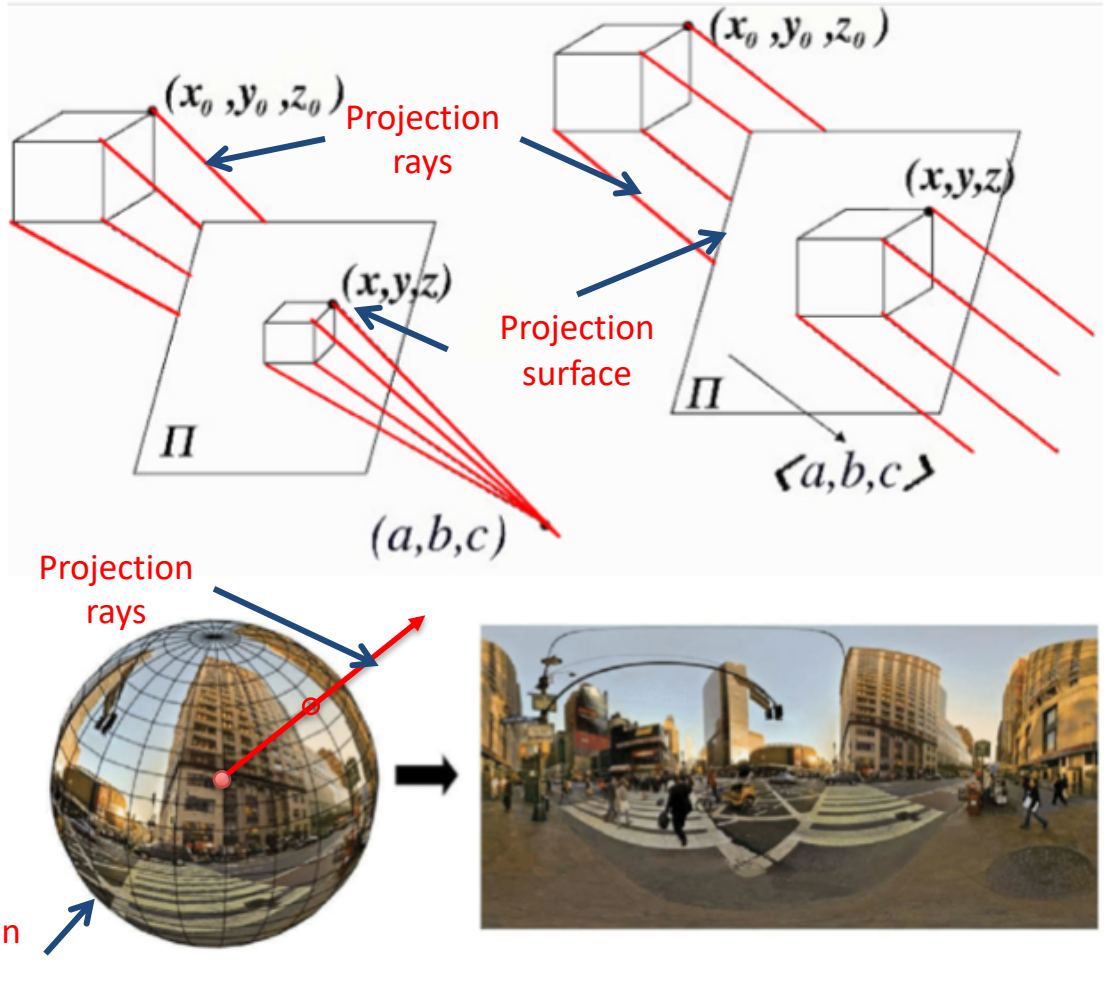
3D computer graphics uses geometrical primitives to describe objects in a three dimensional space.

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

Projections

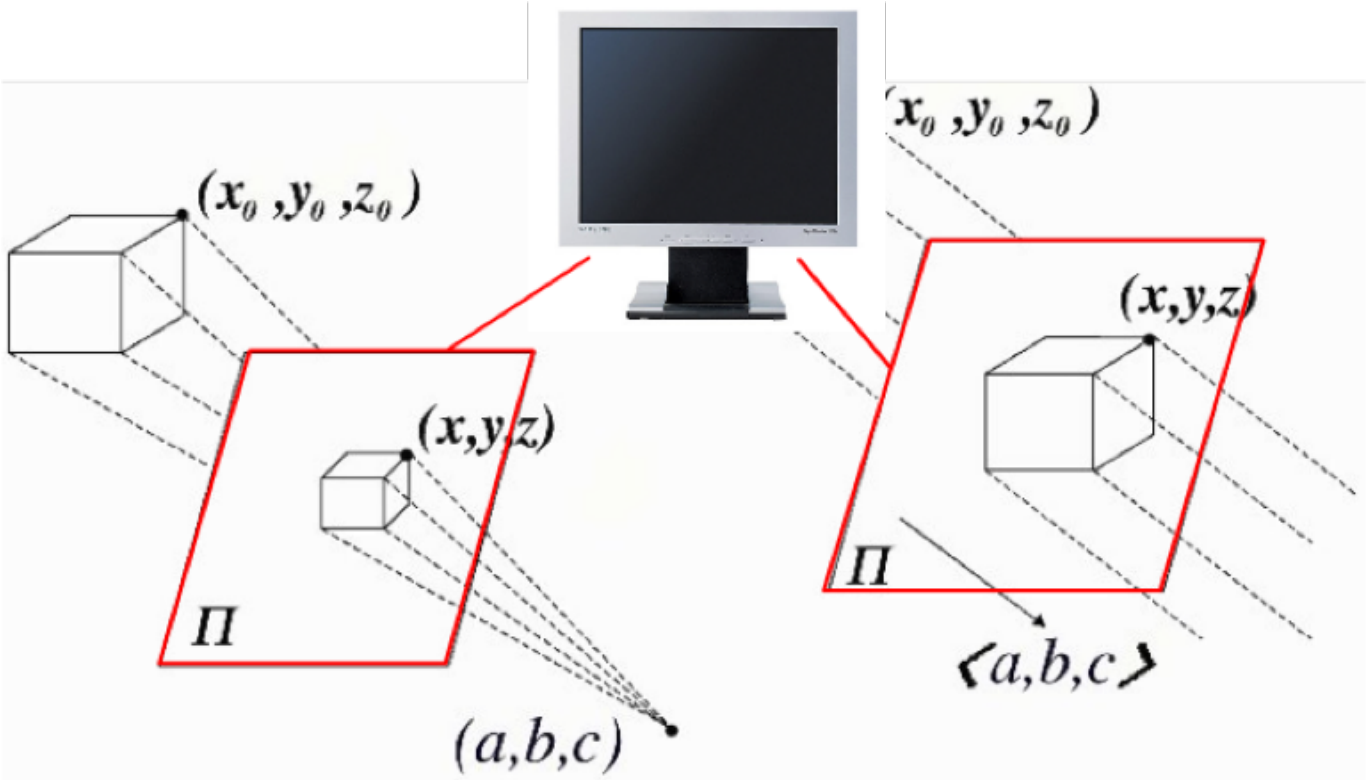
The technique used to construct a 2D image from a 3D scene is called *projection*.

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



Projections

This surface is generally a plane. It is called the *projection plane*, and a rectangular area defined on it corresponds to the screen. We will only consider planar projections.



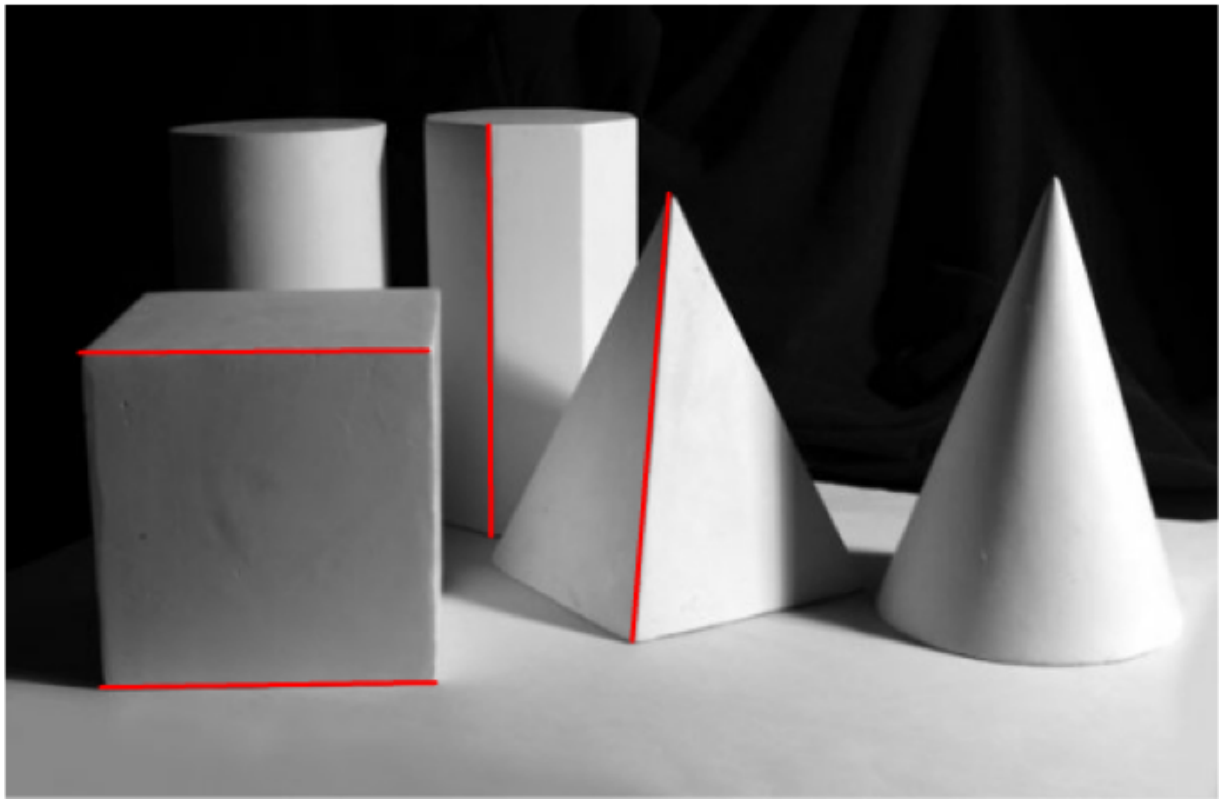
Projections

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

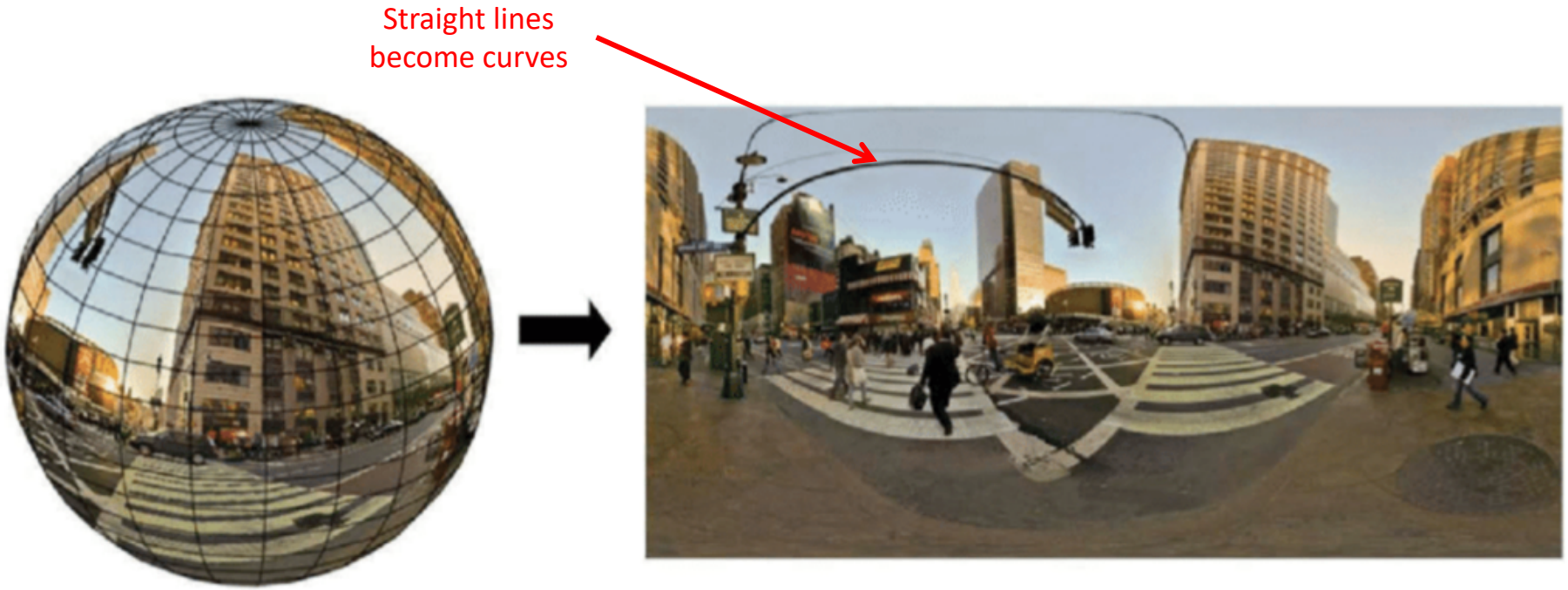
Projections

One important properties of projection planes, is that the projection of linear segments in the 3D scene remain straight.



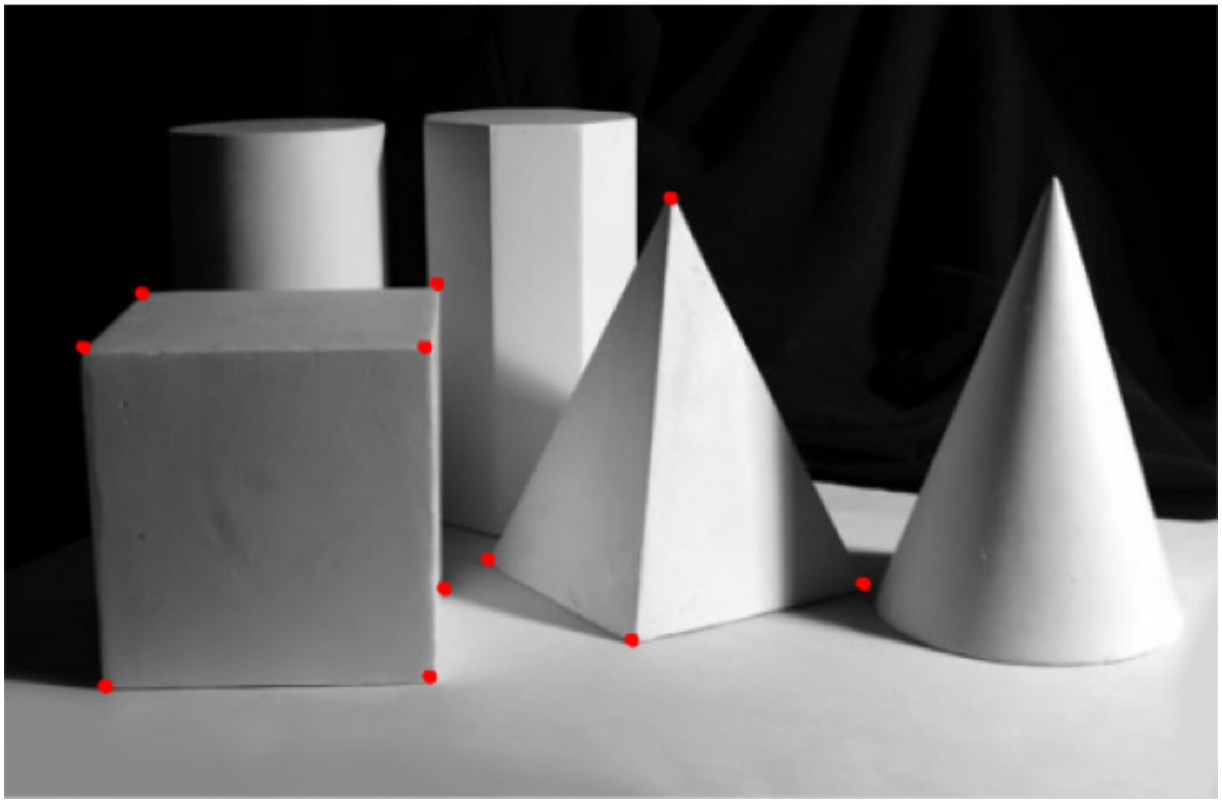
Projections

Note that this is no longer the case with projections on different surfaces



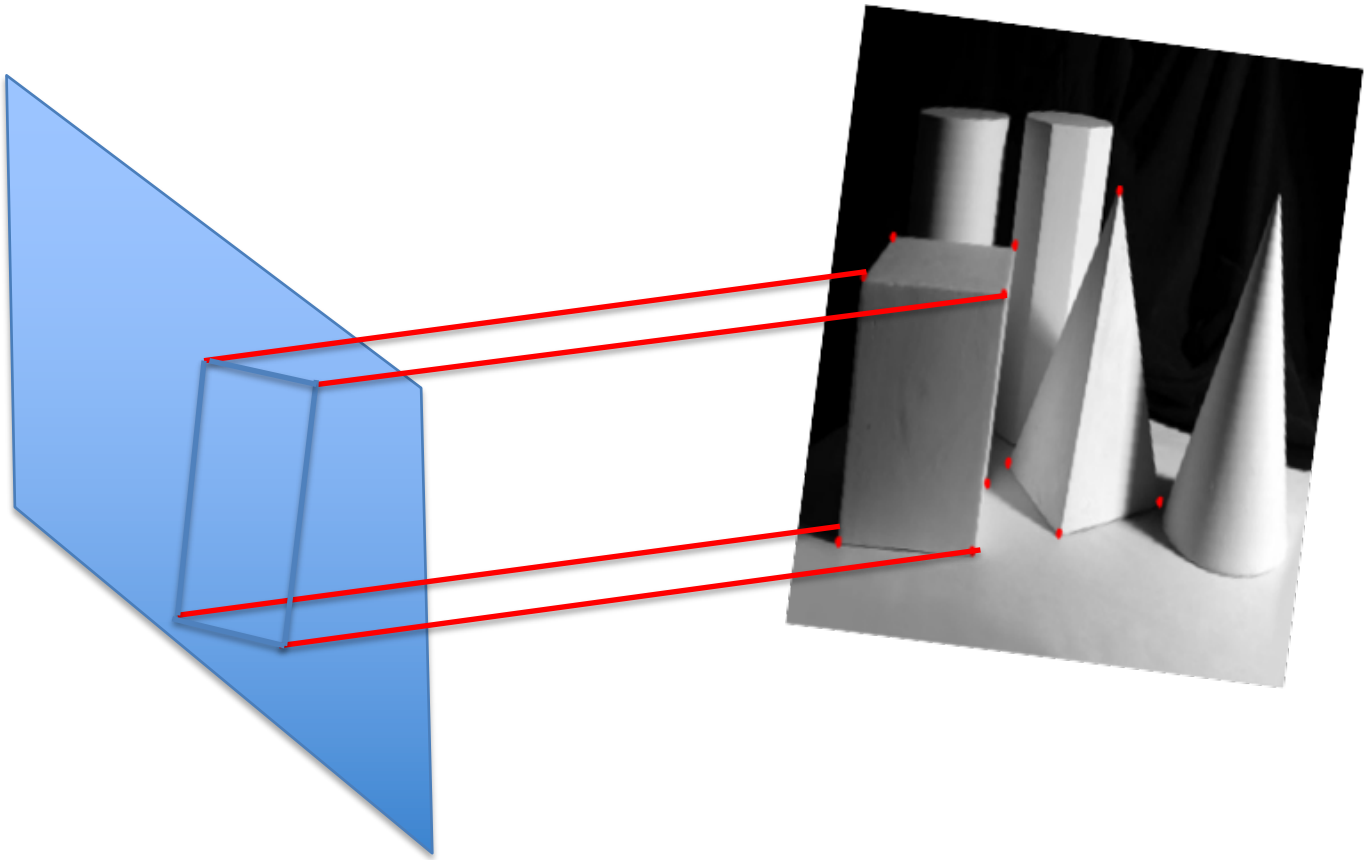
Projections

Since the projected segments connect the projections of their end points…



Projections

It is sufficient to connect the projected vertices to recreate a 2D representation of the corresponding 3D Objects.



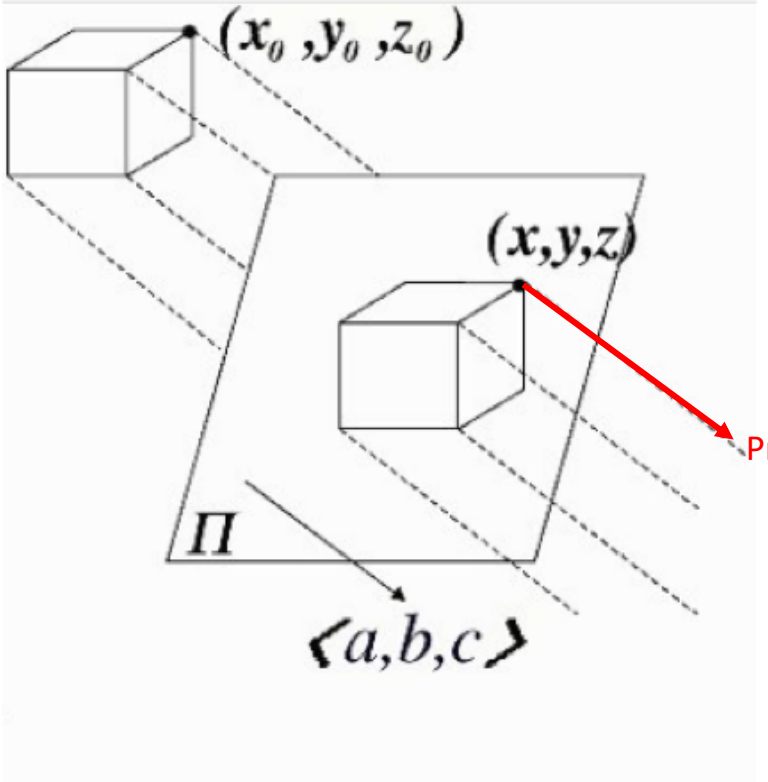
Projection types

We will consider two types of planar projections:

- *parallel projections*
- *perspective projections*

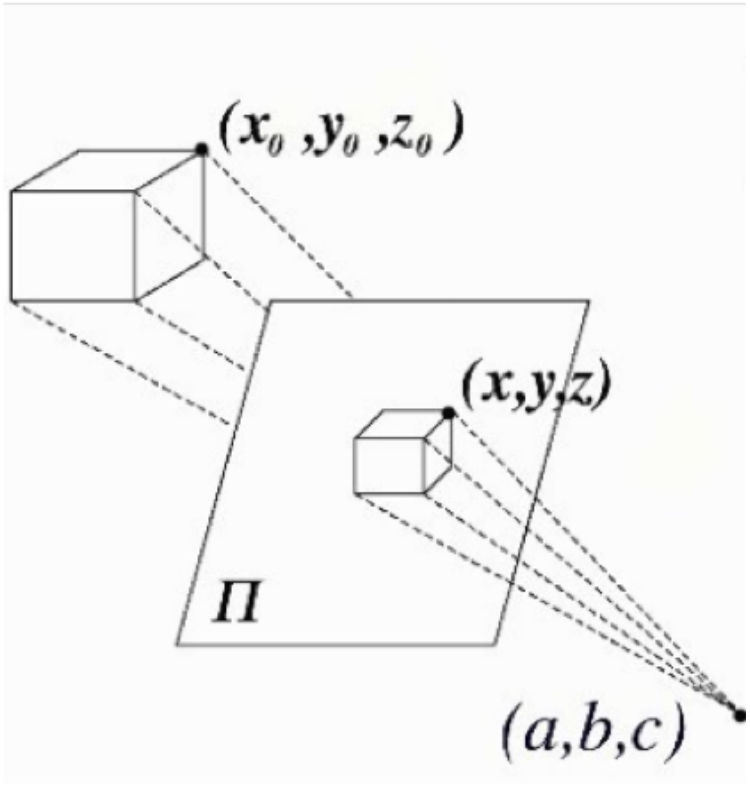
Projection types

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*.



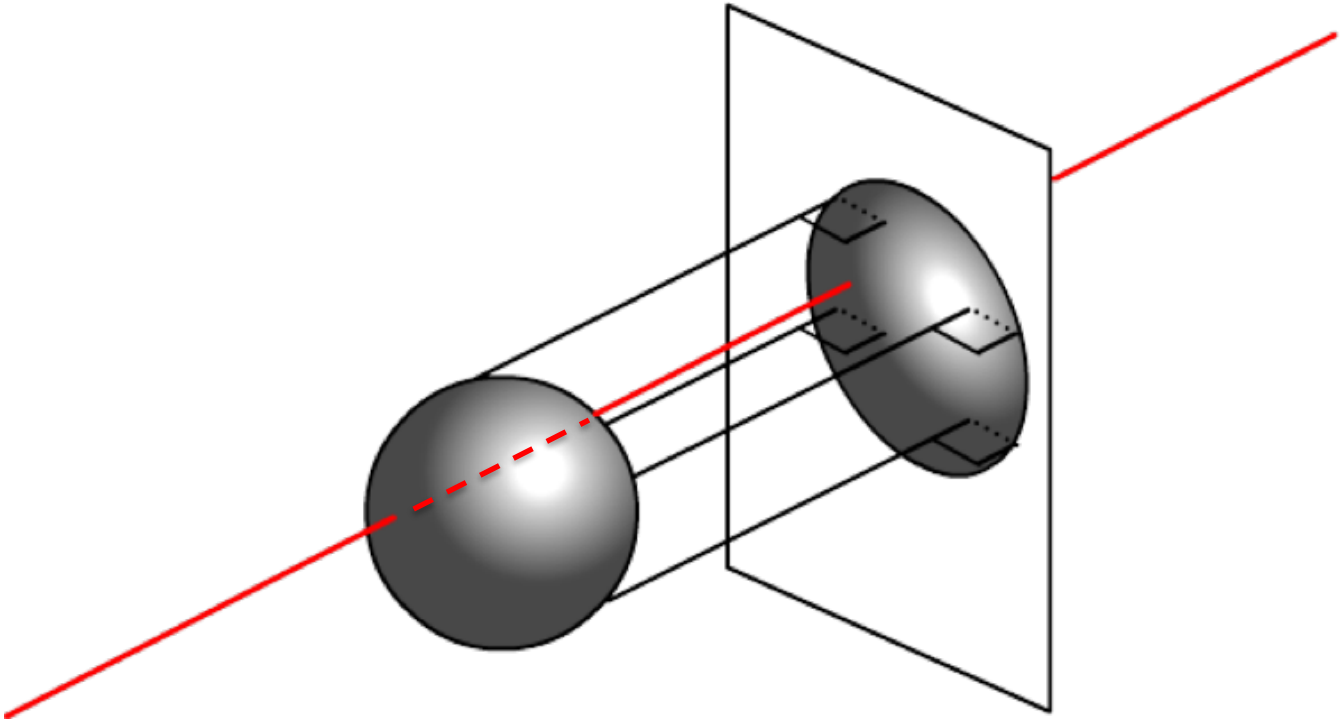
Projection properties

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.

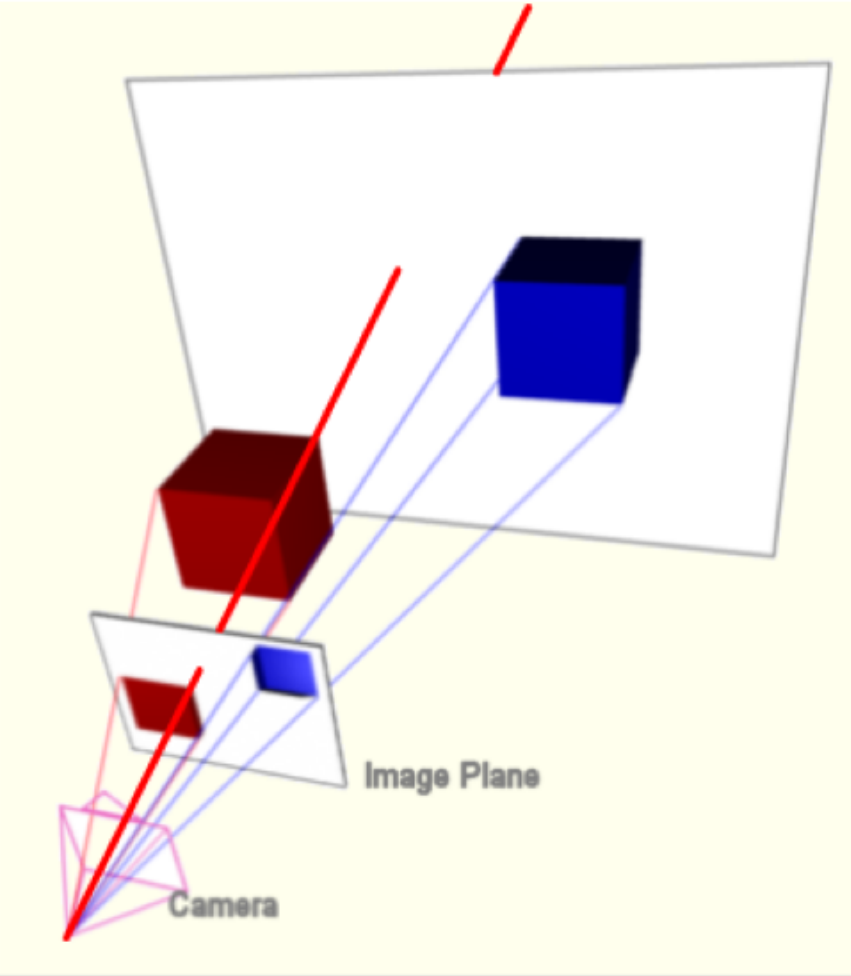
Projection properties

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:

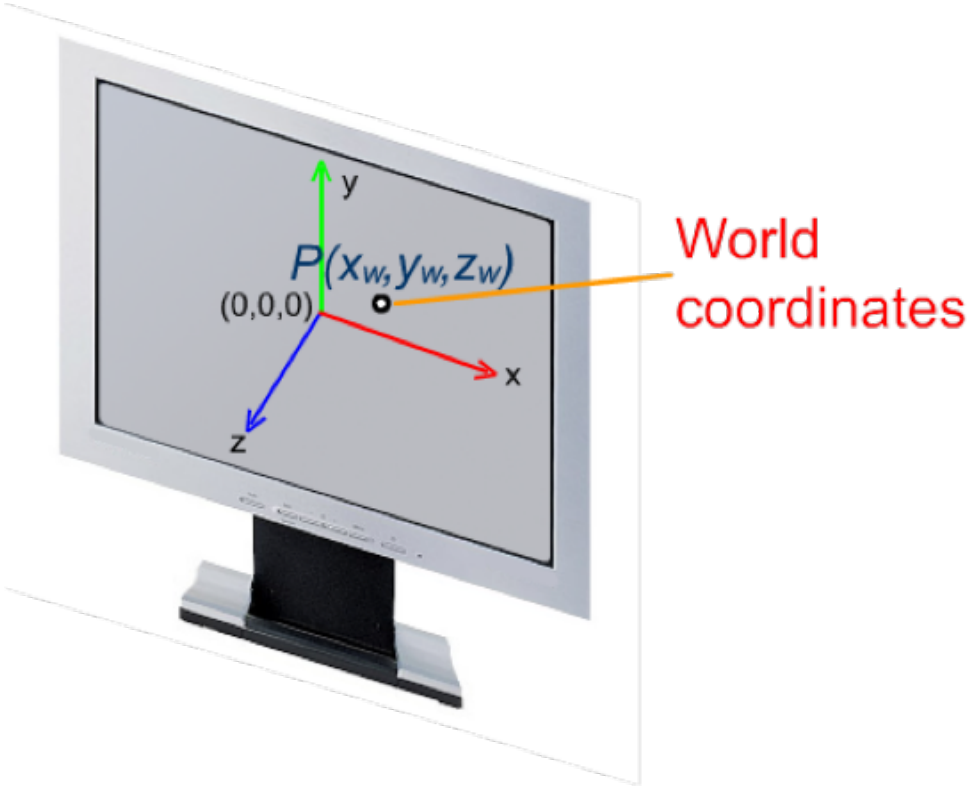
World Coordinates

Into:

3D Normalized Screen Coordinates

World Coordinates

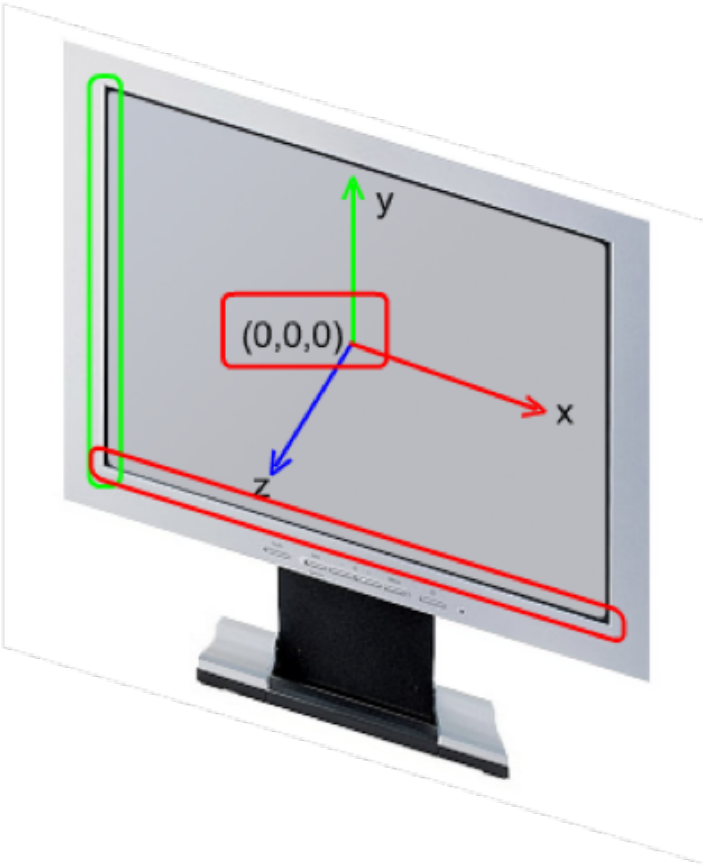
The coordinates system that describes the objects in the 3D space is called *World Coordinates* (or *global coordinates*).



World Coordinates

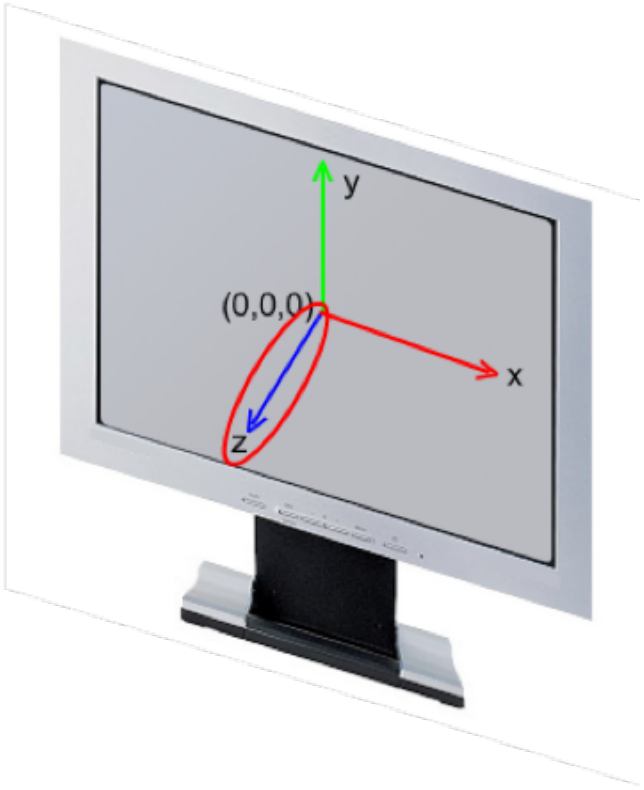
World Coordinates is a right-handed Cartesian coordinate system with the origin that is mapped to the center of the screen.

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

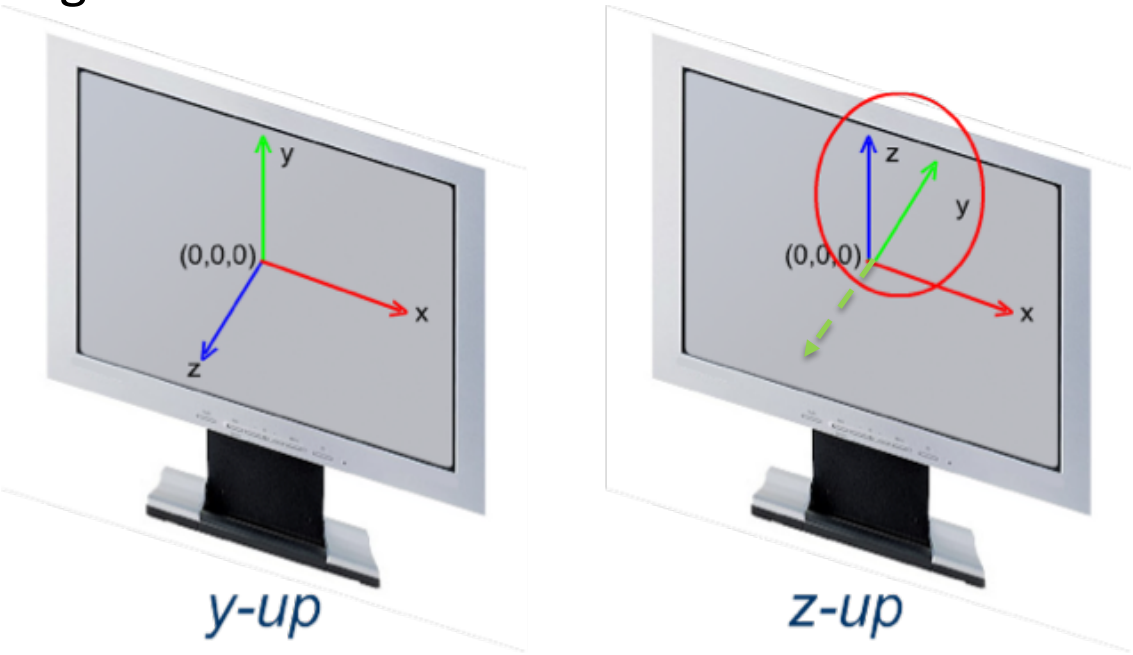
The *z-axis* is directed “out of the screen”, towards the viewer.



World Coordinates

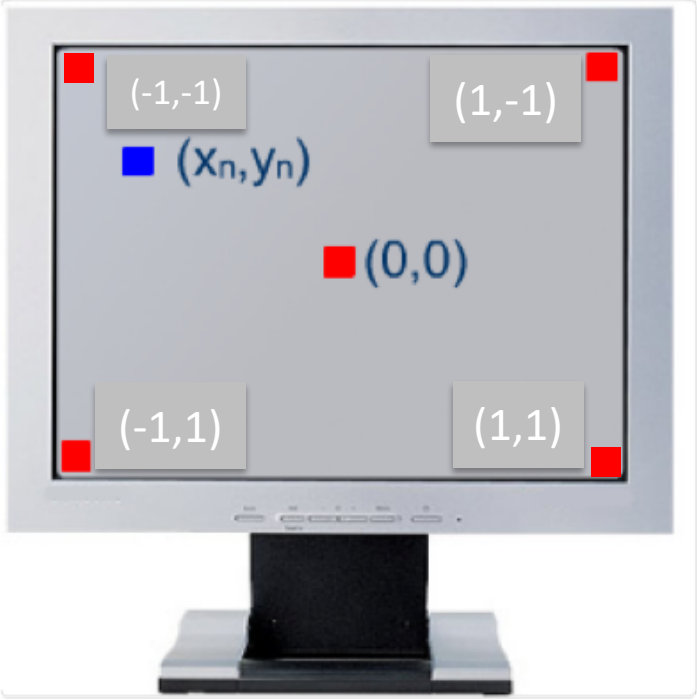
Note that many applications (i.e. *Blender*) use another convention for World Coordinates called *"z-up"*. In this case the *z-axis* is oriented along the vertical screen direction, and the *y-axis* points "inside the screen". Some applications even use a left-handed system, with the y axis pointing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

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



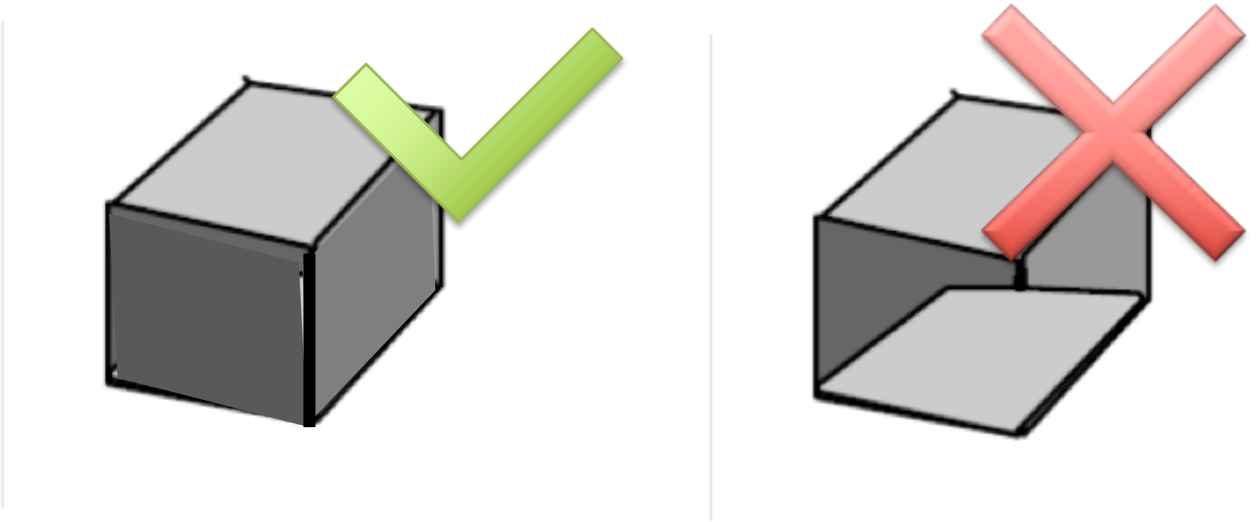
3D Normalized Screen Coordinates

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 (as we will see later).

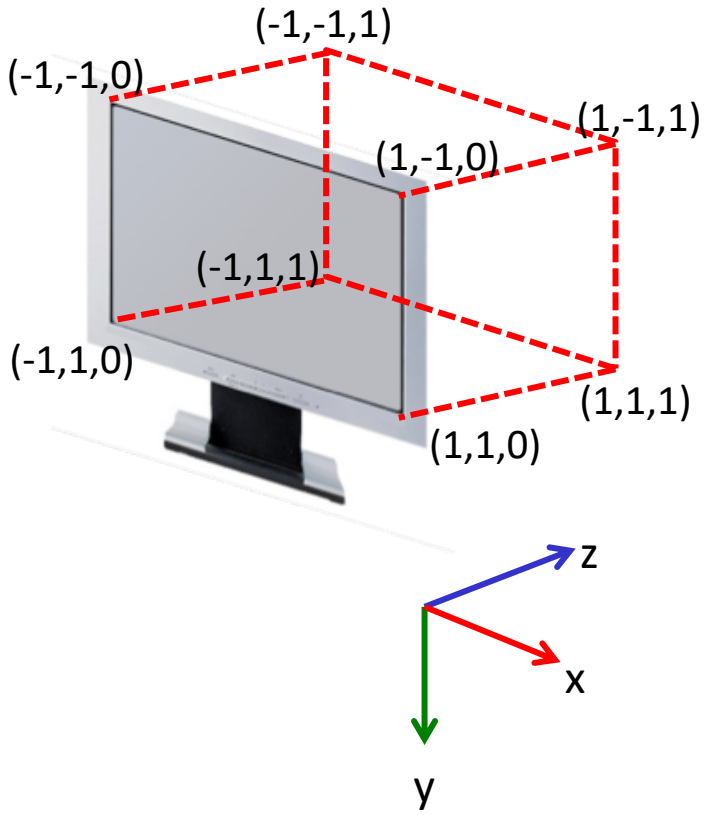


3D Normalized Screen Coordinates

3D Normalized Screen Coordinates have a third component, which for Vulkan time is in the [0,1] range.

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

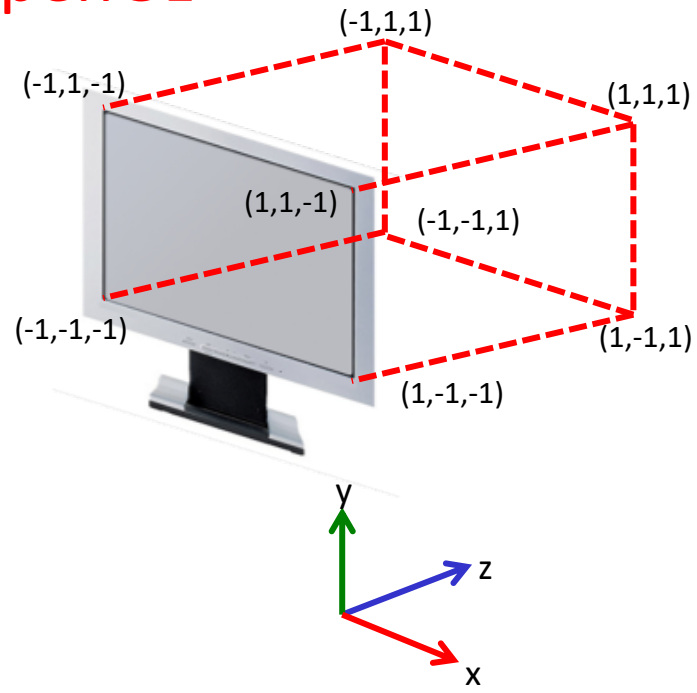
In the following, all normalized screen coordinates will be considered to be 3D.



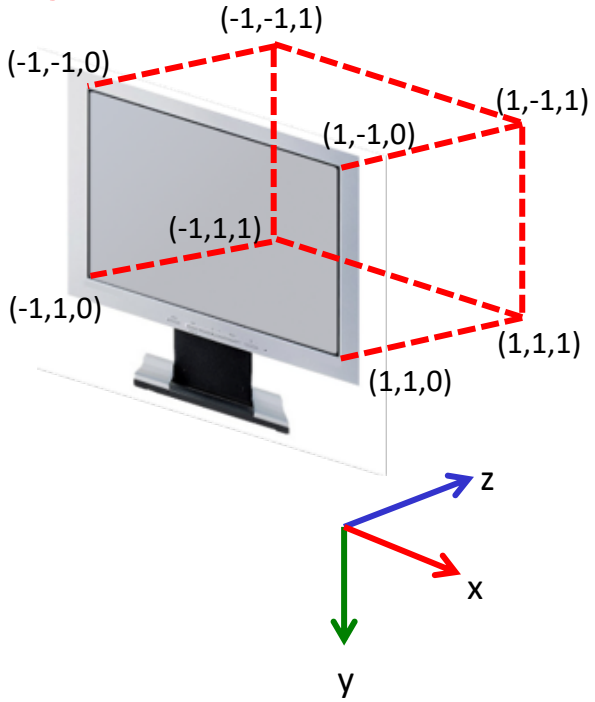
3D Normalized Screen Coordinates

Please note that other low level graphics engines, such as OpenGL, adopts completely different normalized screen coordinates systems.

OpenGL

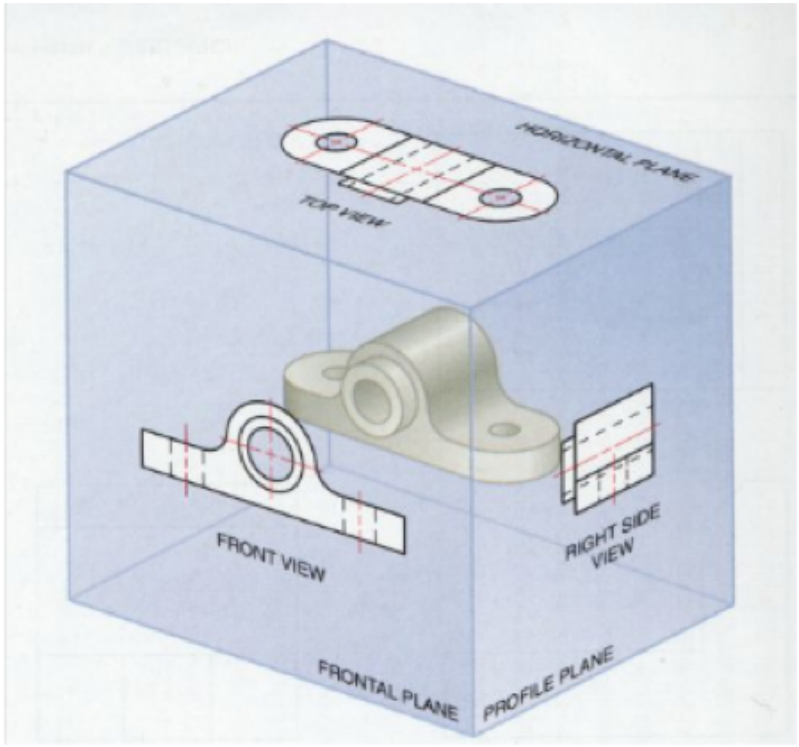


Vulkan



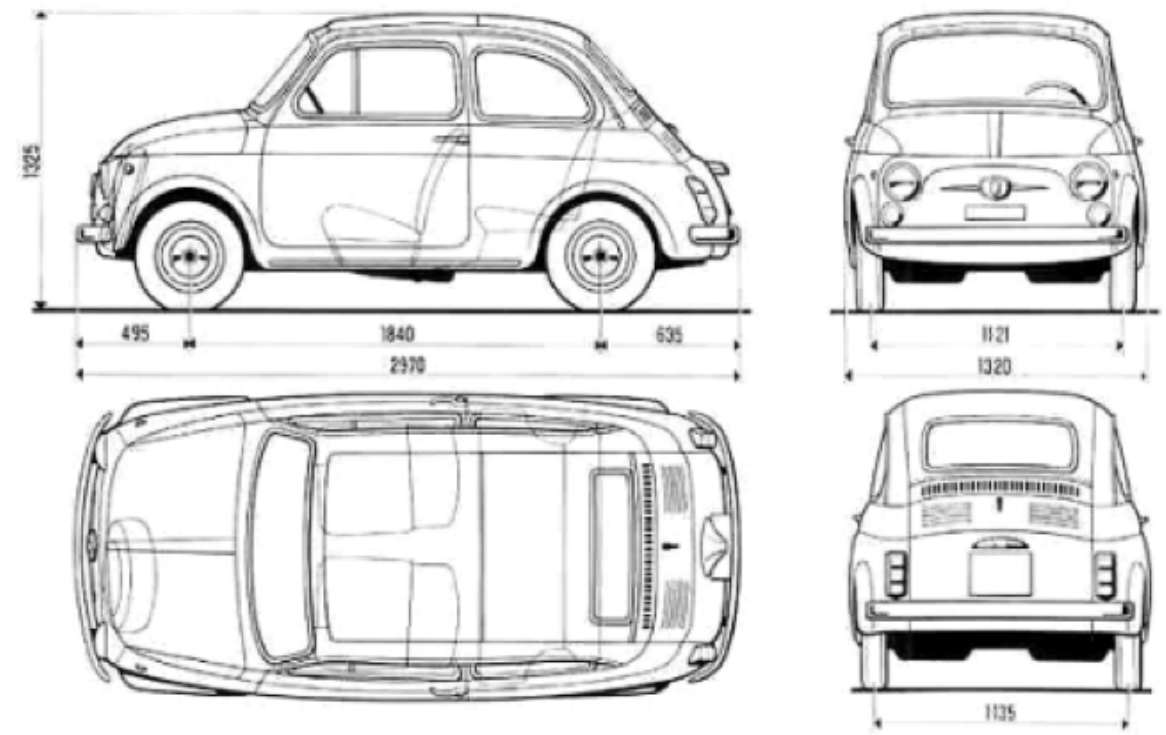
Parallel Projections

Orthogonal Projections are projections where the plane is either the *xy*, *yz* or *zx-plane*, and the projection rays are perpendicular to it.



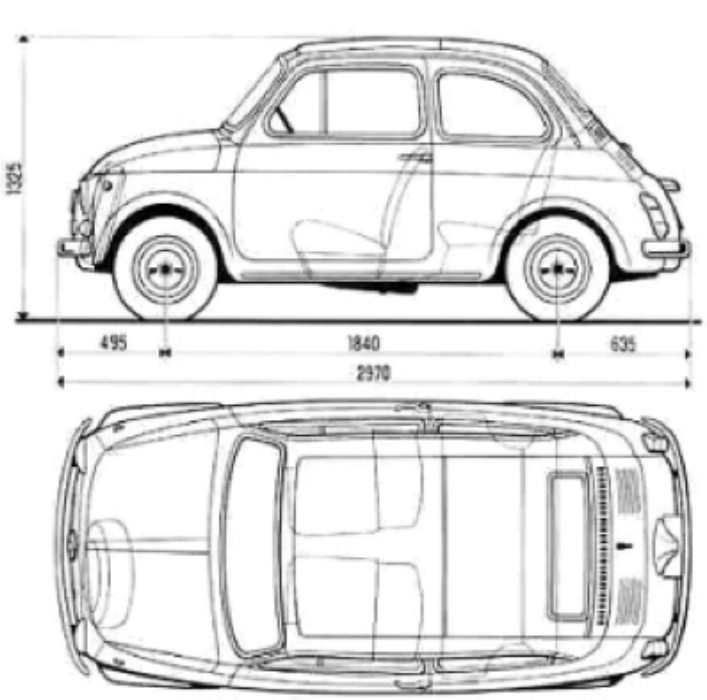
Parallel Projections

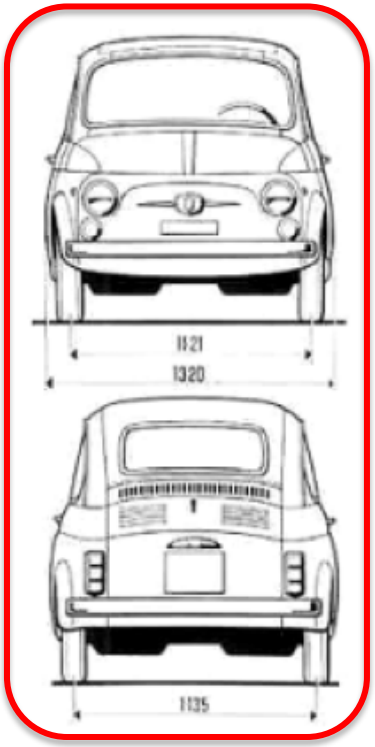
They are mainly used for technical drawings.

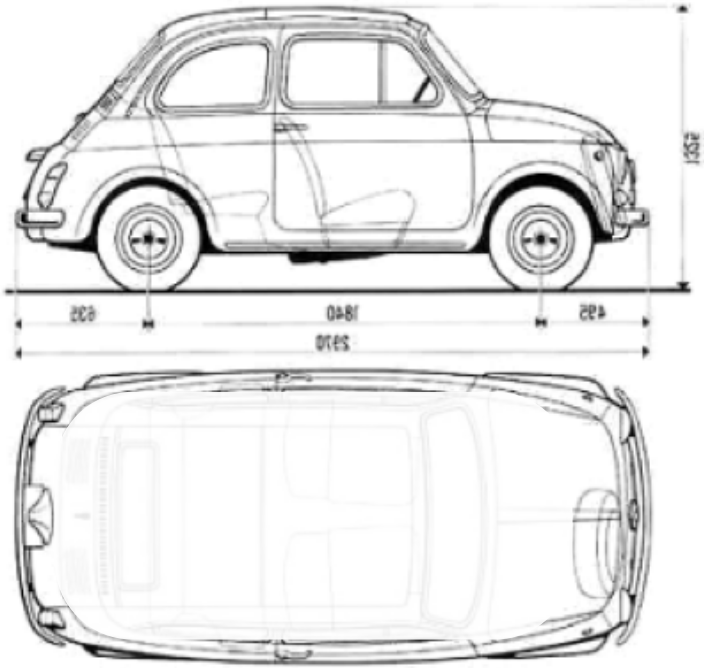


Parallel Projections

Note that there are two different projection directions for each plane: this creates a different ordering in z-component of normalized screen coordinates, leading sometimes to completely different images.

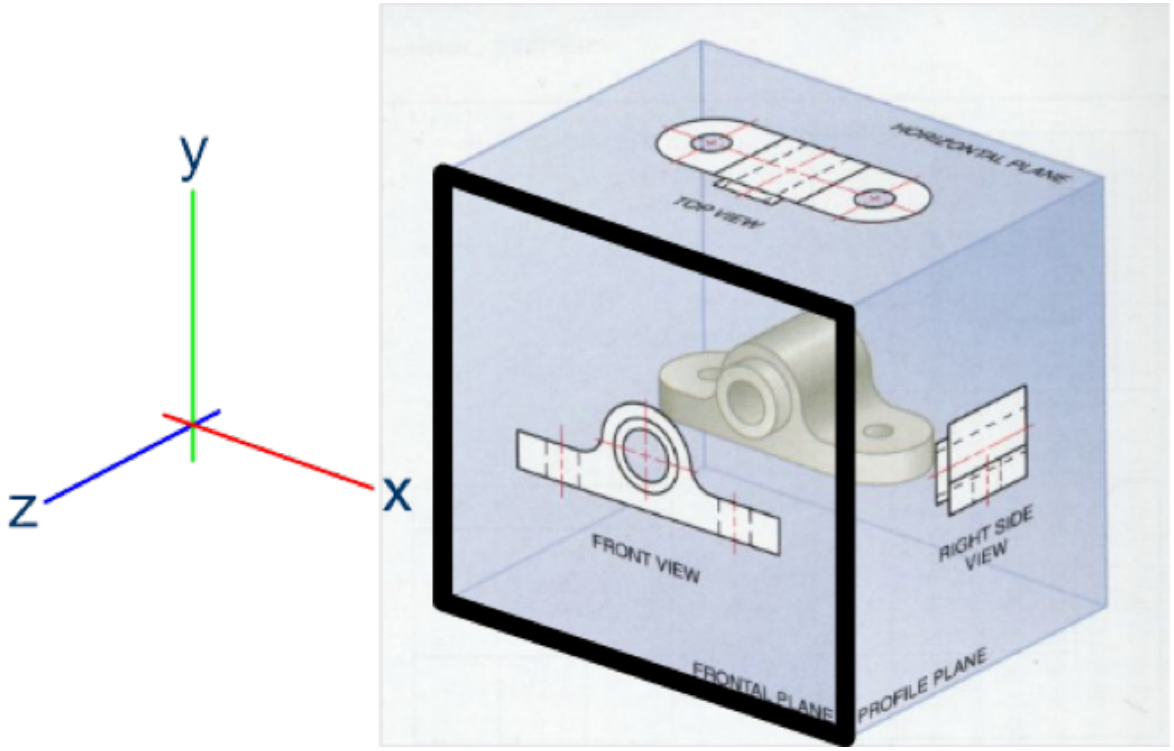






Parallel Projections

We start focusing on a projection plane corresponding to the *xy-plane,* which is perpendicular to the *z-axis*. Visible objects have negative values in their *z* coordinate.



In the following lessons we will relax this assumption by adding a few extra transforms among the ones previously seen.

Near and far planes

The screen boundaries implicitly limit the portion of 3D world that can be visualized on a screen for the horizontal and vertical axes.

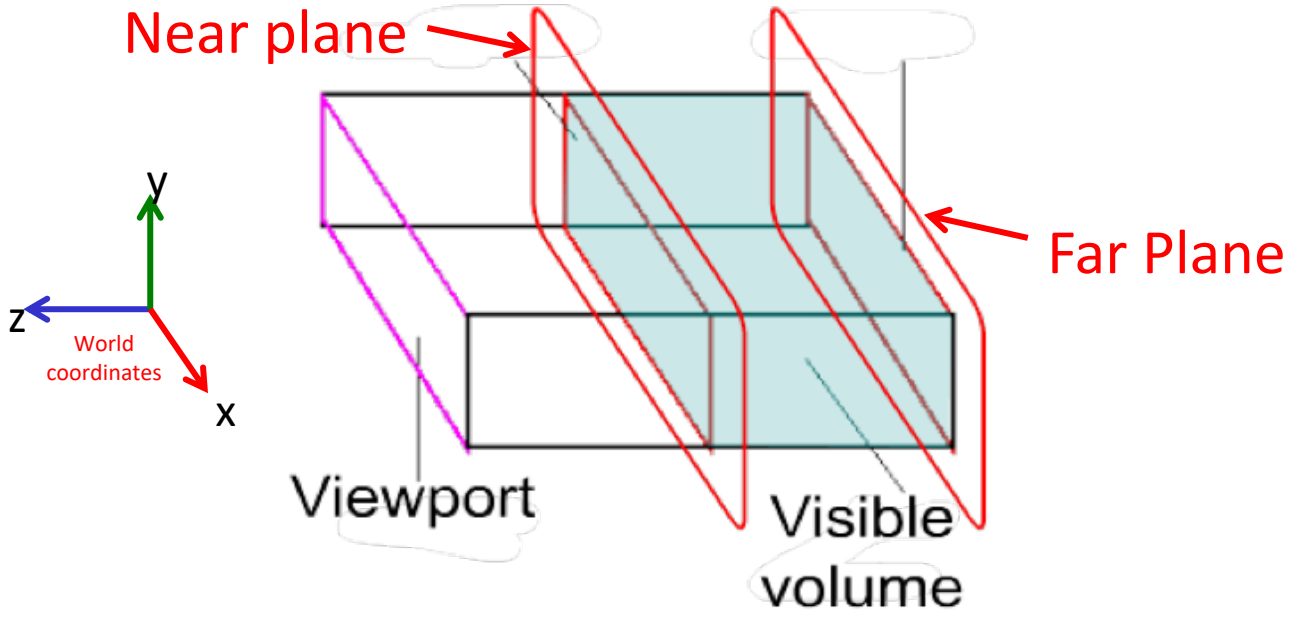
There is however the need to limit the range of a scene also for the depth axis:

- To avoid displaying object behind the observer.
- To avoid showing objects too far away from the considered spot.
- To allow the *z-axis* of the normalized screen coordinates be confined in the 0 to +1 range.

Near and far planes

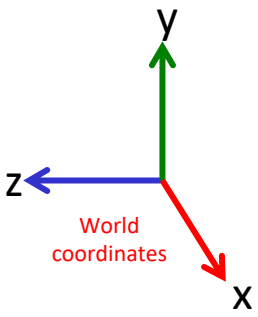
These limits defines two planes:

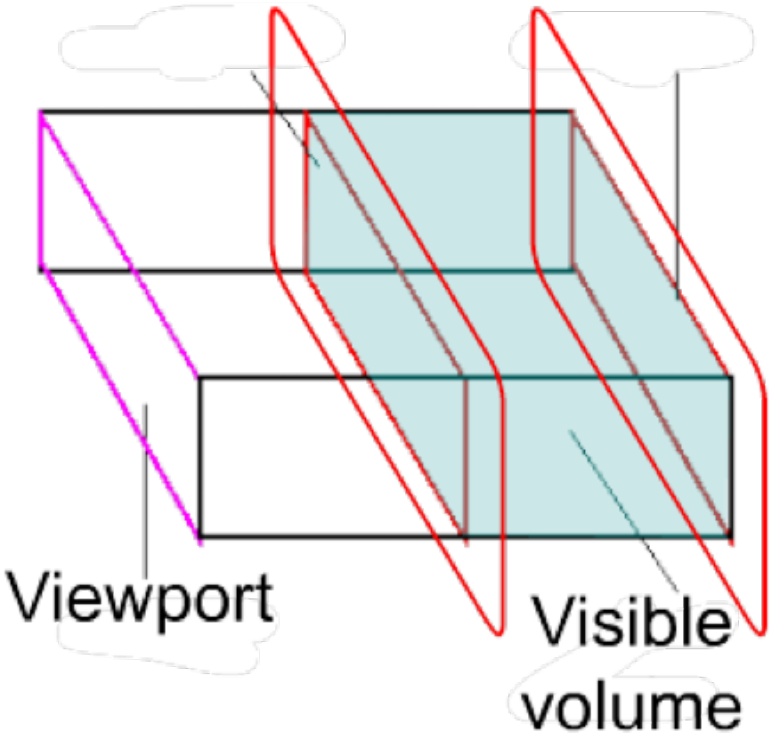
- The plane with the *closest z component (maximum signed, minimum absolute value)* is called the **near plane**.
- The one with *the farthest z component (minimum singed, maximum absolute value)* is called the **far plane**.



Near and far planes

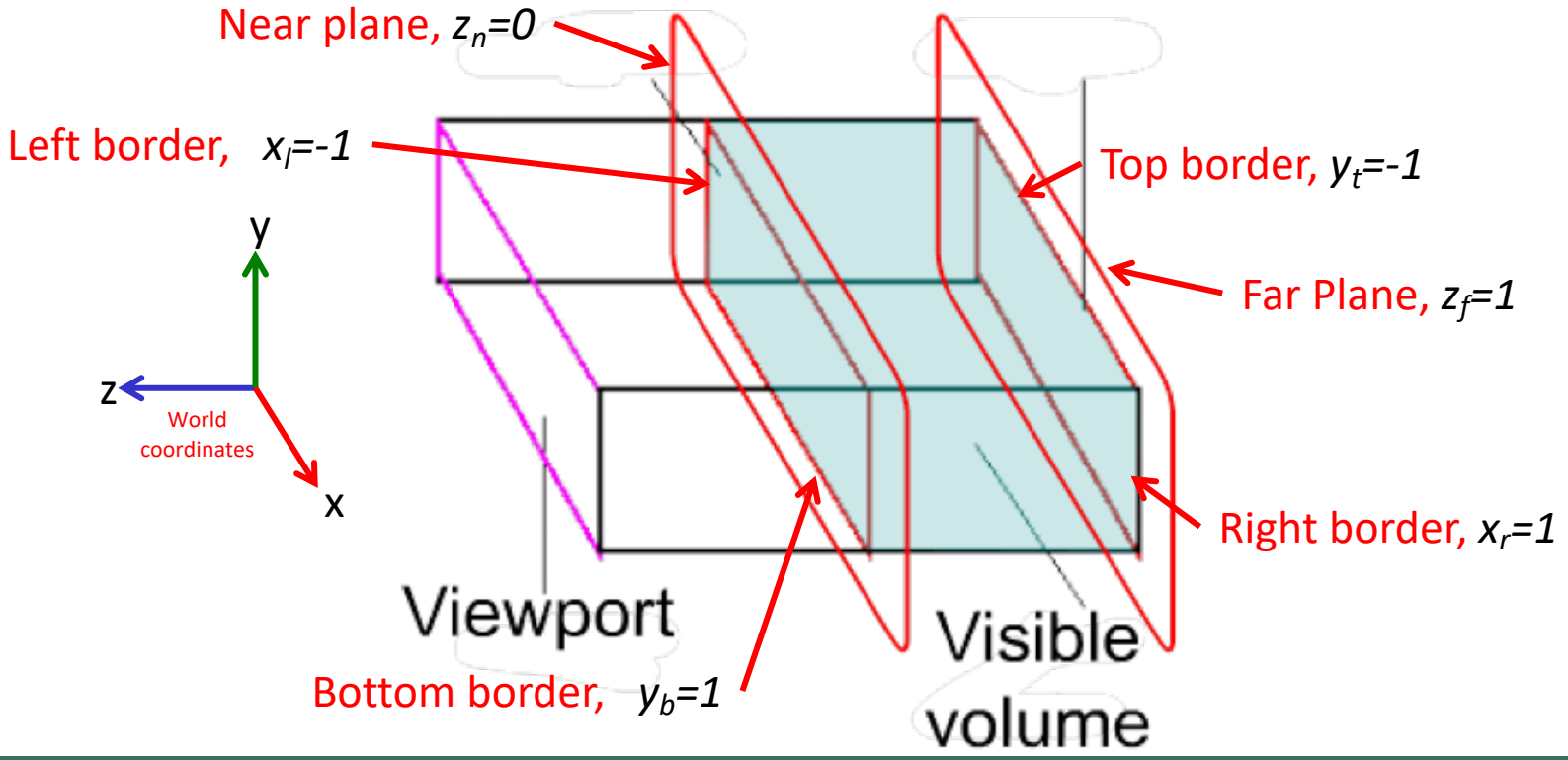
Only the points that are contained inside the box delimited by the left, right, top, bottom screen borders, and by the near and far planes on the depth, can be visualized after the projection.





Near and far planes

The *Normalized Screen Coordinates* are defined such that the near plane, the left and the bottom sides of the screen are projected respectively to $z_n=0$ $x_l=-1$ $y_t=-1$, and the far plane, the right and the top sides to $z_f=1$ $x_r=1$ $y_b=1$.



Orthogonal Projections Matrices

Orthogonal projections can be implemented by normalizing the *x, y, z* coordinates of the projection box in the *(-1,1), (-1,1) and (0,1)* range.

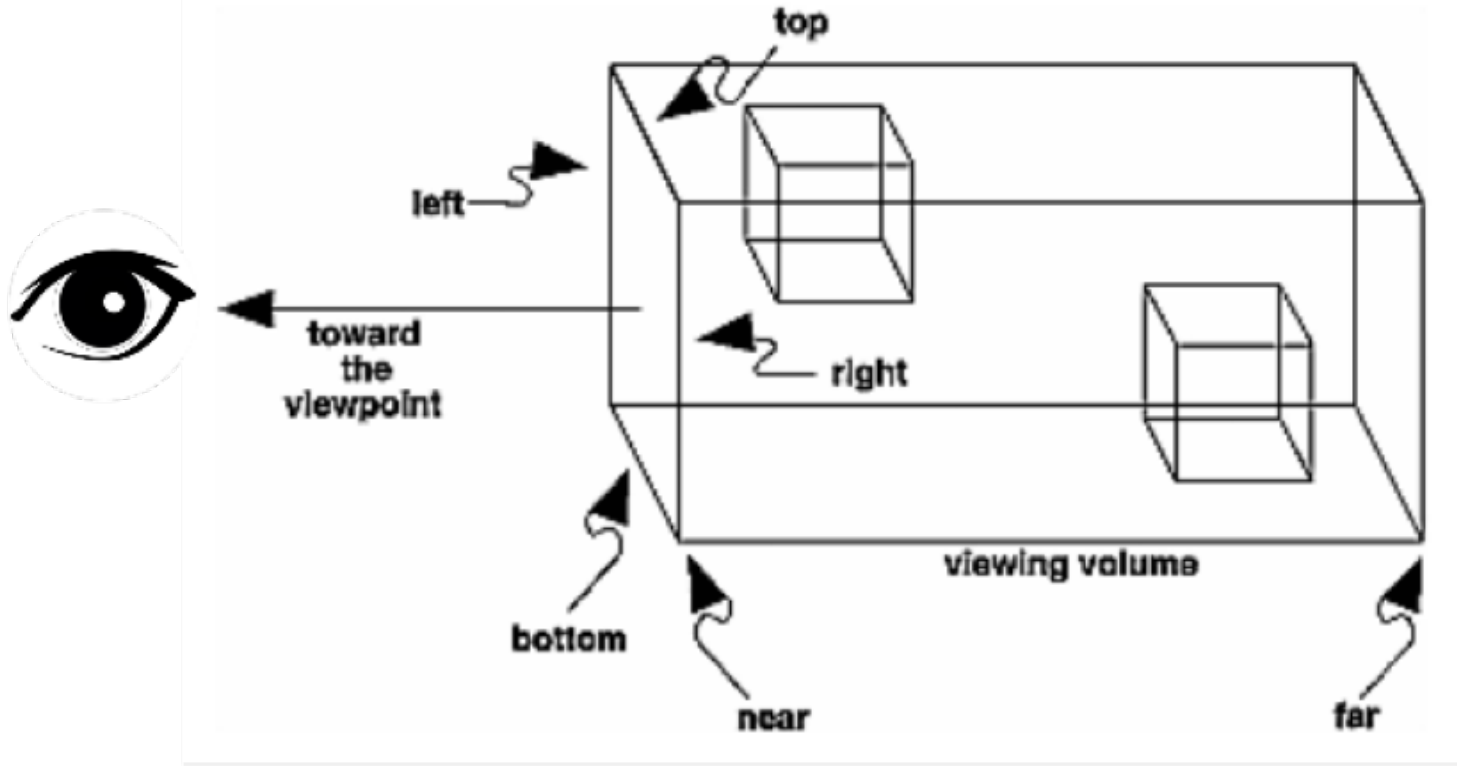
As transformations previously introduced, the orthogonal projections can be obtained with the product with a suitable a matrix.

In particular, we define the *projection matrix* P_{ort} , as the matrix that multiplied with a world coordinate p_W computes the corresponding normalized screen coordinate p_N.

$$p_N = P_{ort} \cdot p_W$$

Orthogonal Projections Matrices

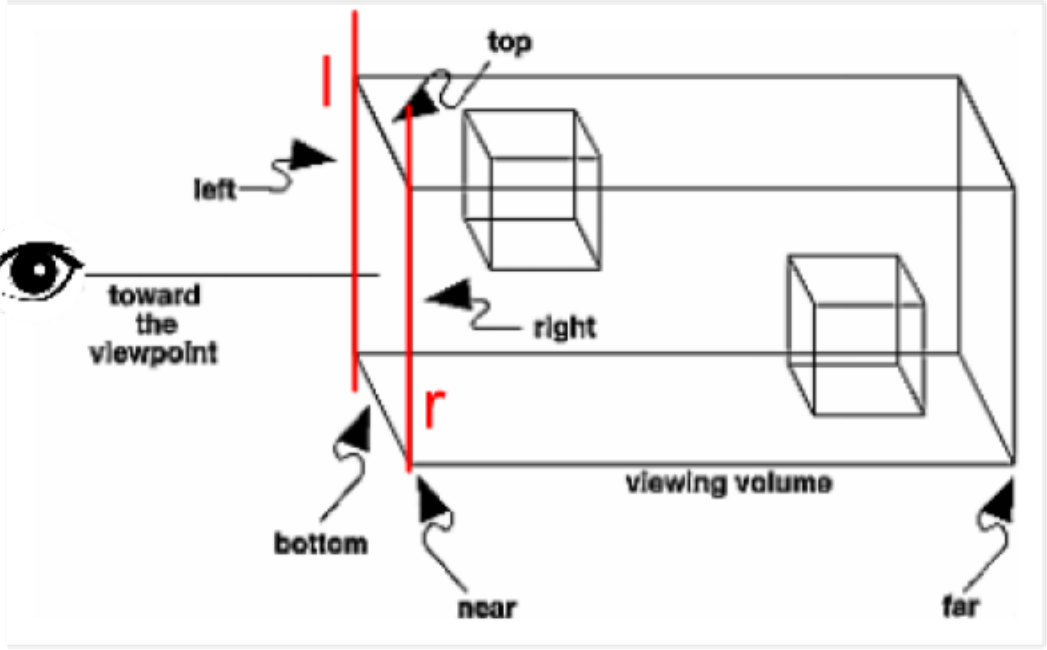
In order to obtain normalized coordinates, we need to know which are the coordinates of the borders of the screen in the 3D space.

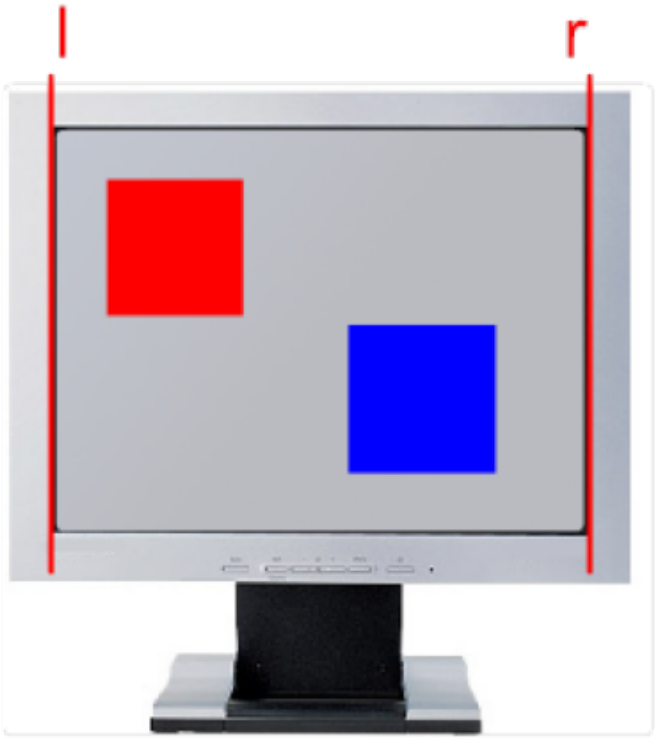


Orthogonal Projections Matrices

We call *l* and *r* the *x* coordinates in the 3D space of locations displayed on the left and right borders of the screen.

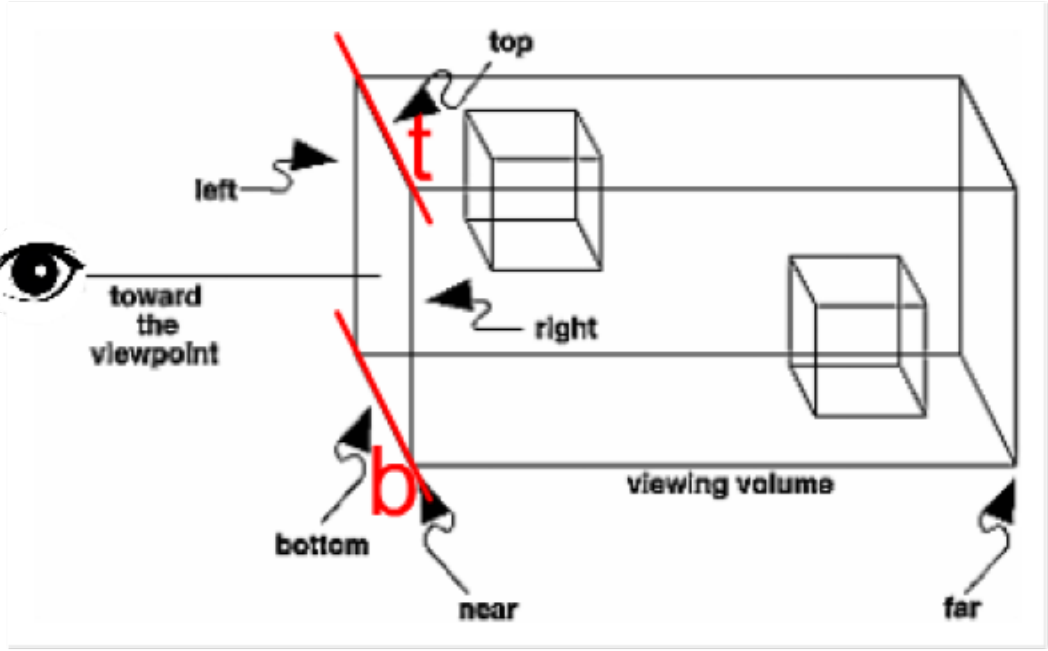
Everything on the left of *l* will be cut out of the screen, and nothing to the right of r will be visualized.

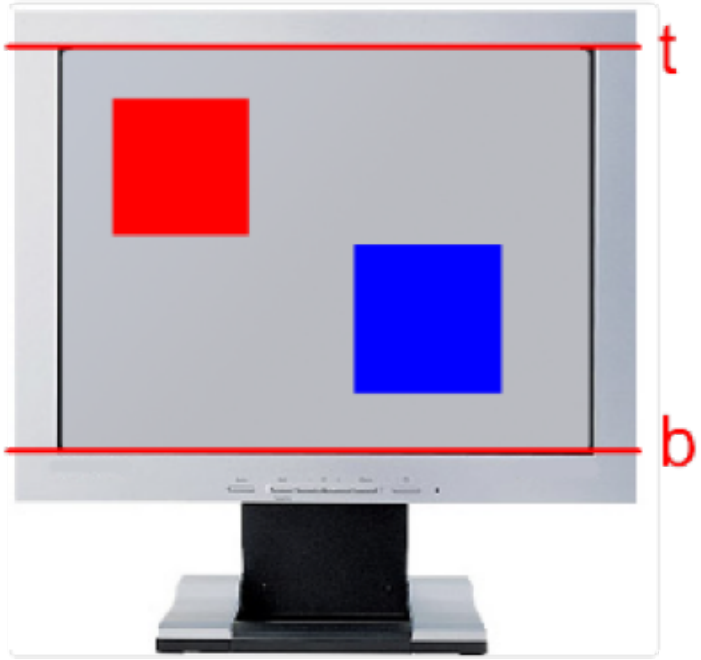




Orthogonal Projections Matrices

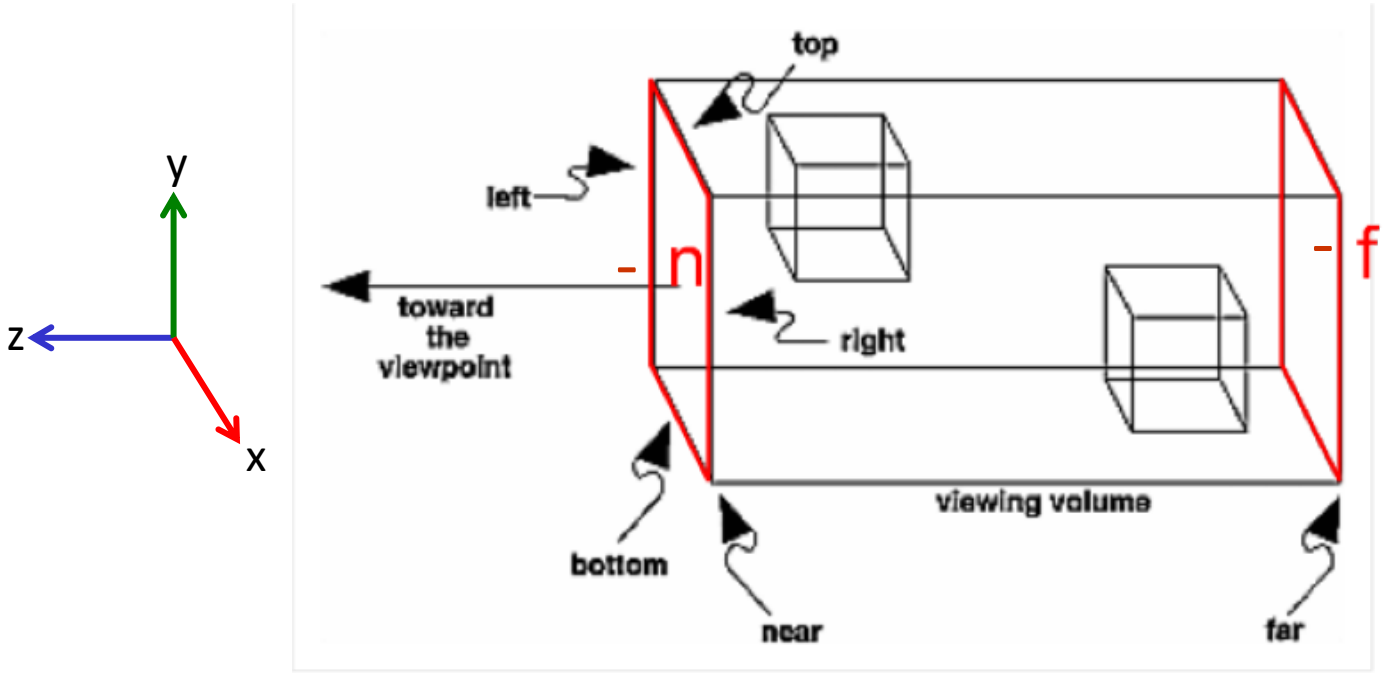
Similarly we use *t* and *b* for the *y* coordinates of the top and bottom borders of the screen in the 3D space.





Orthogonal Projections Matrices

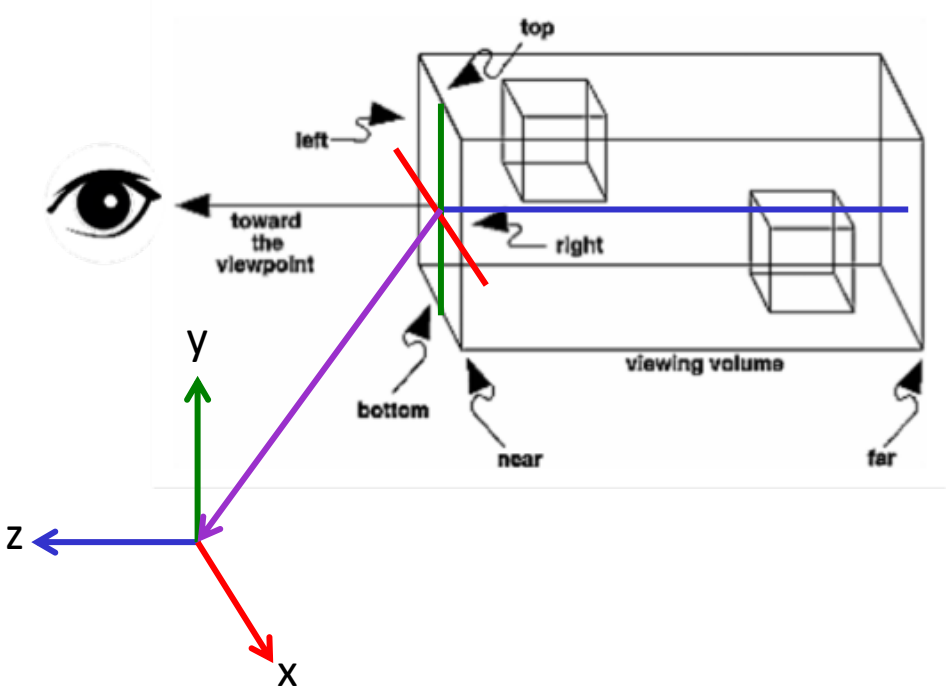
We also call *-n* and *-f* the *z* coordinates of the near and far planes in the 3D space.



Since the *z-axis* is oriented in the opposite direction with respect to the viewer, two positive distances *n* and *f* are generally used instead of the negative coordinates to identify the two planes.

Orthogonal Projections Matrices

The orthogonal projection matrix P_{ort} is computed in three steps: first we move the center of the projection box at the near plane in the origin with a translation T_{ort}:



$$T_{ort} = \begin{vmatrix} 1 & 0 & 0 & -\frac{r+l}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -(-n) \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Orthogonal Projections Matrices

The second step normalizes the coordinates between -1 and 1 (x and y-axis) or 0 and 1 (z-axis) by performing a scale transformation S_{ort}:

$$S_{ort} = \begin{vmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

The last step corrects the direction of the axes, by inverting the y and z-axis with the mirror transformation M_{ort}:

$$M_{ort} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Orthogonal Projections Matrices

Applying the composition of transformations seen in the previous lessons, we can define the orthogonal projection matrix P_{ort} as follows:

$$P_{ort} = M_{ort} \cdot S_{ort} \cdot T_{ort} = \begin{vmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{l-r} \\ 0 & \frac{2}{b-t} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{1}{n-f} & \frac{n}{n-f} \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Aspect Ratio

The proportions of the physical screen are called **aspect ratio**: that is the ratio between the horizontal and vertical dimensions.

$$a = D_x / D_y$$

Aspect Ratio

In this definition, the measures for the horizontal and vertical sizes, D_x and D_y, are *metrical units and* not pixels.

If the pixels are square, both metrical units and pixels would lead to the same result.

If pixels are not square, only the actual display proportions can create images that are not distorted.

Normalized screen coordinates do not account for the aspect ratio: the projection matrix takes care of this factor and properly scale the images to make them appear with the correct proportions.

Orthogonal Projections Matrices and Aspect Ratio

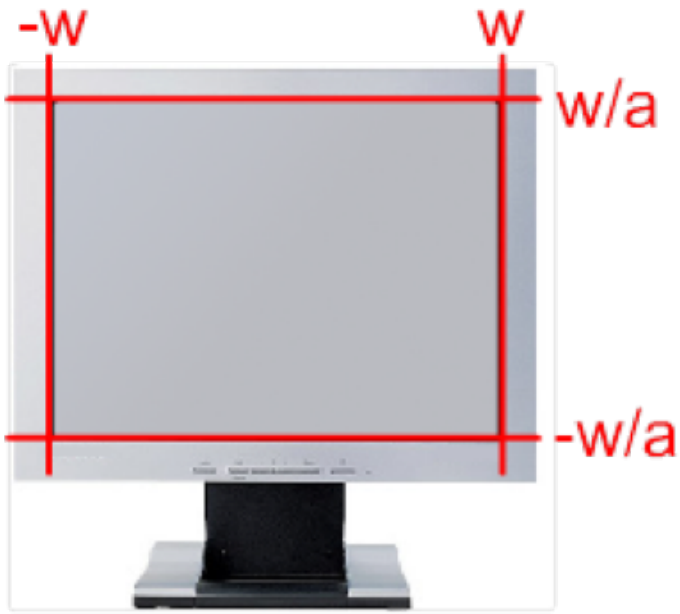
In order to produce images with the correct proportions *l, r, t* and *b* must be consistent with the aspect ratio *a* of the monitor.

In particular, we must have that:

$$a = \frac{r-l}{t-b} \quad \Rightarrow \quad r-l = a \cdot (t-b)$$

Orthogonal Projections Matrices and Aspect Ratio

In most of the cases, the projection box is centered in the origin both horizontally and vertically : if this happens, only the half-width of the box *w*, the near plane *n*, the far plane *f*, and the aspect ratio a are needed.

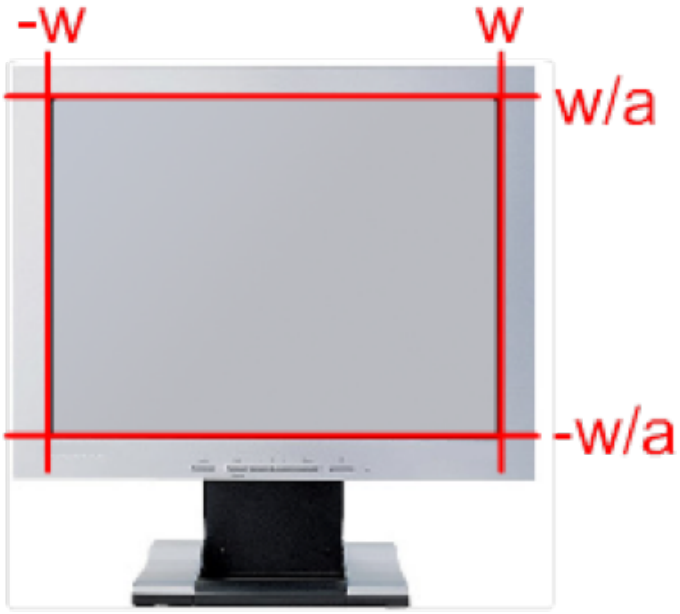


$$l = -w$$
$$r = w$$
$$t = \frac{w}{a}$$
$$b = -\frac{w}{a}$$

$$P_{ort} = \begin{vmatrix} \frac{1}{w} & 0 & 0 & 0 \\ 0 & \frac{-a}{w} & 0 & 0 \\ 0 & 0 & \frac{1}{n-f} & \frac{n}{n-f} \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Orthogonal Projections Matrices and Aspect Ratio

In case of orthogonal projections, the near plane can be put at *n=0* (this will not be possible for the perspective projections that will be presented next time). In this case, the projection matrix can be simplified even more, becoming a non-uniform scaling matrix of factors *S(1/w, -a/w, -1/f)* :



$$P_{ort} = \begin{vmatrix} \frac{1}{w} & 0 & 0 & 0 \\ 0 & \frac{-a}{w} & 0 & 0 \\ 0 & 0 & \frac{-1}{f} & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Orthogonal Projections Matrices: coordinates

World Coordinates are *Homogenous coordinates* but *Normalized Screen Coordinates* are *Cartesian coordinates*: the conversion procedure from one system to the other (division by the fourth component) seen in the previous lessons must be performed.

In case of parallel projection, since it is implemented using a sequence of conventional transform matrices, the last element of P_N is always one.

This mean that the *Normalized Screen Coordinates* can be obtained by simply discarding the fourth element of the vector obtained after multiplying the homogeneous coordinates with the projection matrix.

Orthogonal Projections Matrices: example

Example:

Consider a projection with half width *w=400*, aspect ratio *4:3*, near plane at *n=0*, and far plane at *f = 1000*. Find the normalized screen coordinates A_N of a point *A* at *(100, 200, -300)*.

$$A_N = P_{ort} \cdot A = \begin{vmatrix} \frac{1}{400} & 0 & 0 & 0 \\ 0 & \frac{-1}{300} & 0 & 0 \\ 0 & 0 & \frac{-1}{1000} & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 100 \\ 200 \\ -300 \\ 1 \end{vmatrix} = \begin{vmatrix} 0.25 \\ -0.666 \\ 0.3 \\ 1 \end{vmatrix}$$

$A_N = (0.25, -0.666, 0.3)$

Orthographic projections in GLM

GLM gives the the `ortho()` function to compute the orthographic projection matrix specifying the boundaries:

```
glm::mat4 Port = glm::ortho(l, r, b, t, n, f);
```

Where `l, r, b, t, n, f` are the positions in world coordinates respectively of the left, right, bottom, top, near and far boundaries of the visible region.

Keep however in mind that such procedure was created for the Normalized Screen Coordinates conventions of OpenGL, and the result will not work correctly in Vulkan without further modifications.

Orthographic projections in GLM

In order to make it work, the following additions must be done:

```
#include <iostream>
#include <cstdlib>

#define GLM_FORCE_DEPTH_ZERO_TO_ONE
#define GLM_FORCE_RADIANS

#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>

...

  Port = glm::scale(glm::mat4(1.0), glm::vec3(1,-1,1)) *
         glm::ortho(l, r, b, t, n, f);
```

This directive, before including the library, forces the z-axis of the normalized screen coordinates in the zero-one range.

This added matrix product flips the y-axis to match the Vulkan conventions.

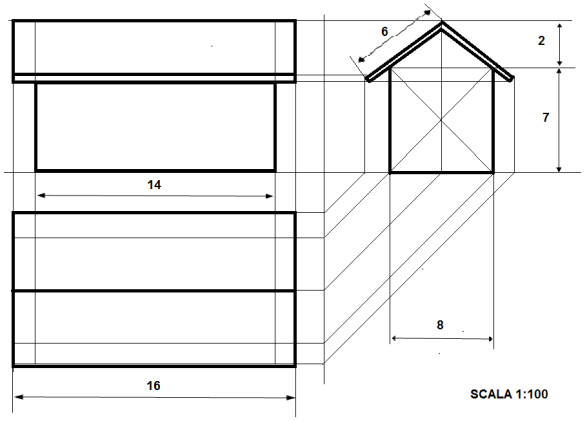
Axonometric projections

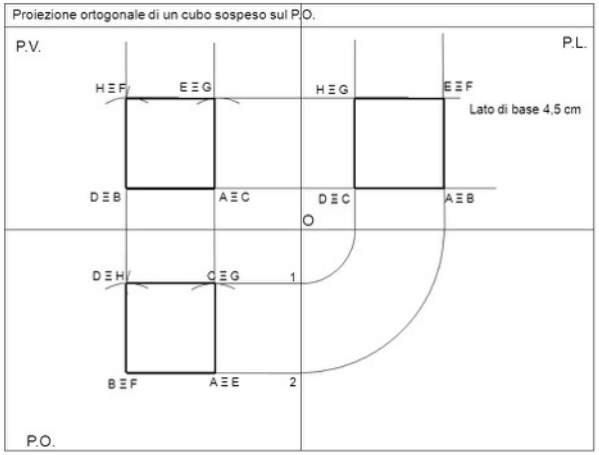
Most of the objects have shapes somehow aligned with the sides of a box.

Parallel projections hide faces perpendicular to the projection plane.

A cube, oriented along the three main axis, appears as a square: this limits the perception of depth.

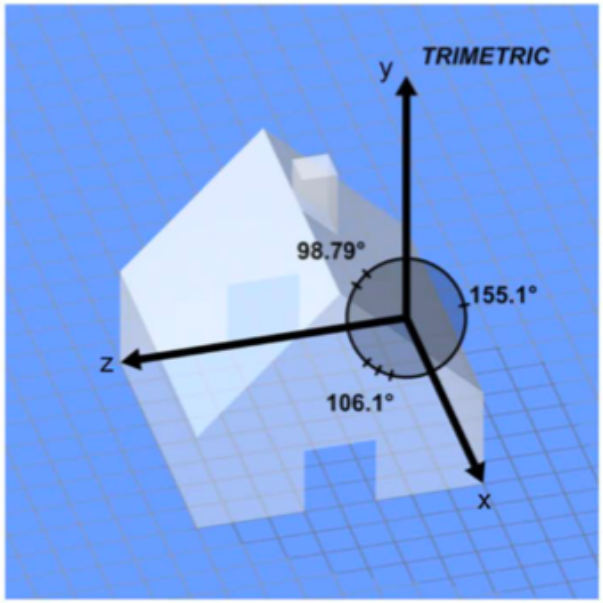
Axonometric projections, overcome this limitation, by showing all the faces of a cube at the same time.



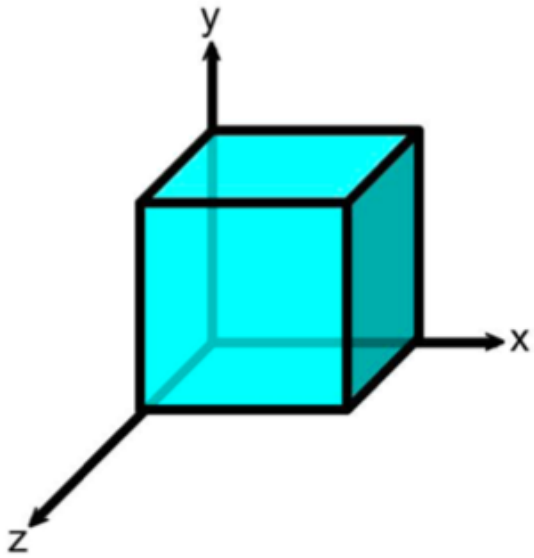


Axonometric projections

Axonometric projections allow to see all the faces of box shaped objects by either rotating the projection plane with respect to the three main axis, or by making the projection plane no longer perpendicular to the projection rays.



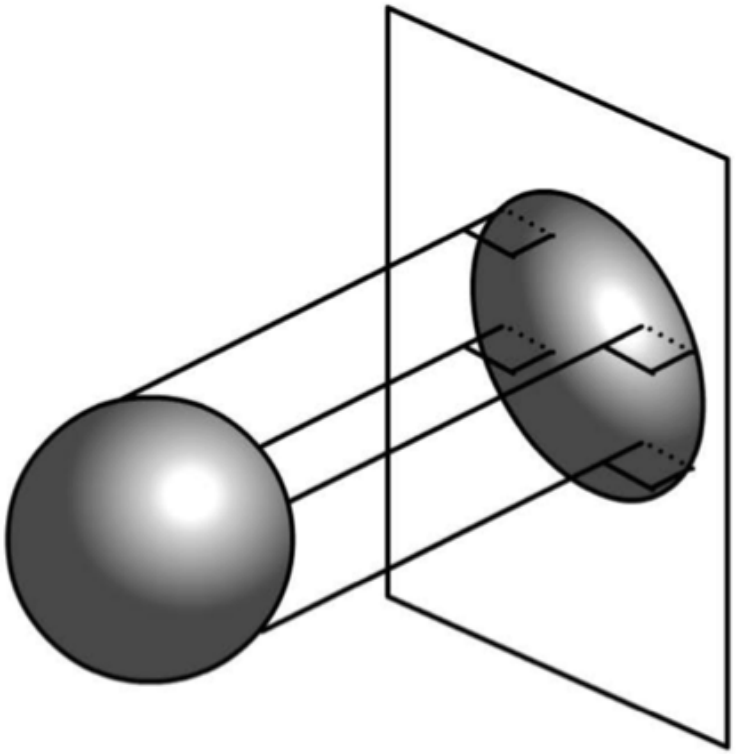
Rotating the projection plane



Projection rays no longer perpendicular

Axonometric projections

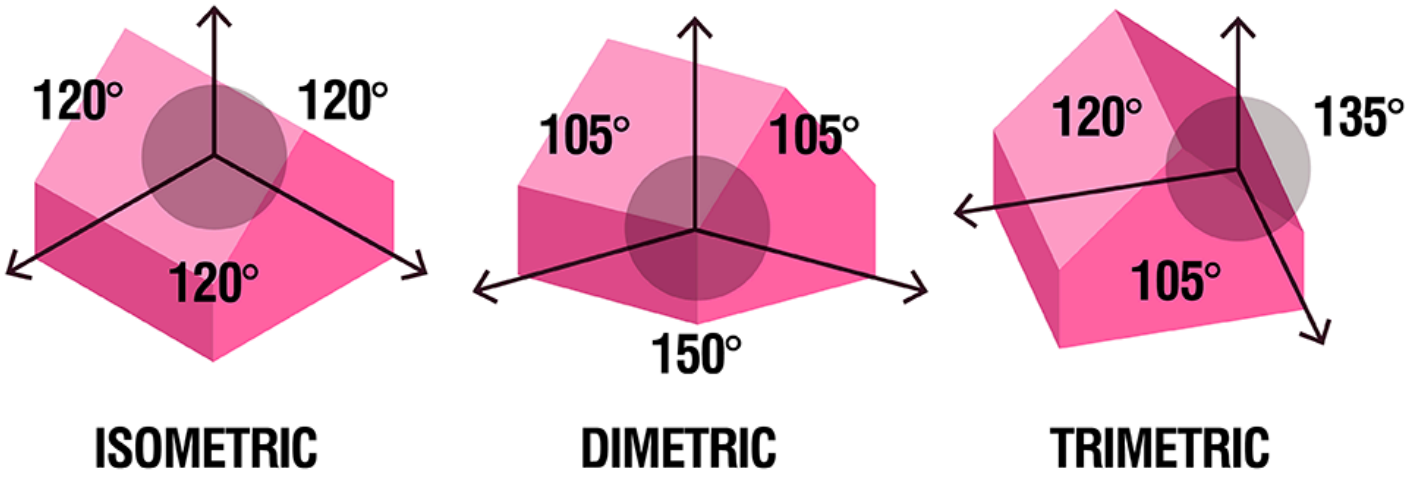
Axonometric projections where rays are perpendicular to the projection plane are called *Orthographic projections*.



Axonometric projections

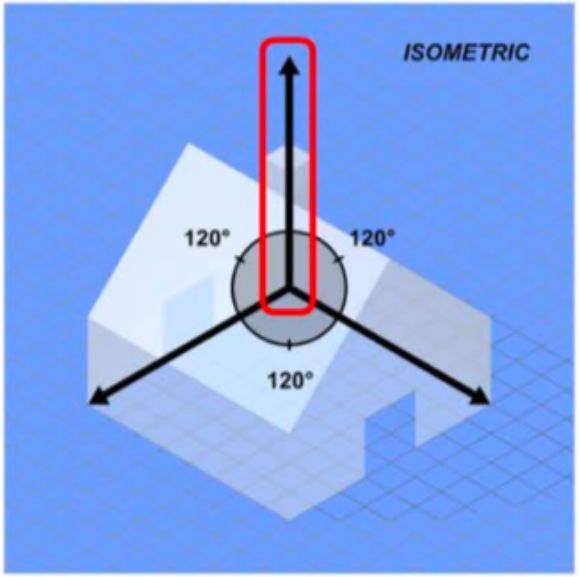
The most important types of orthographic axonometric projections are:

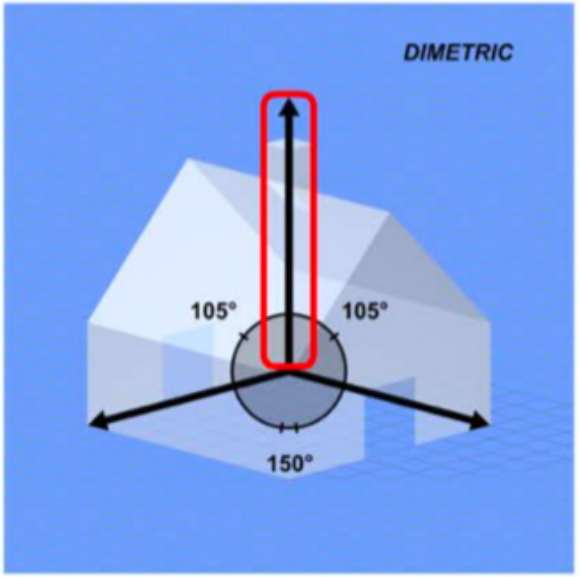
- **Isometric**
- **Dimetric**
- **Trimetric**

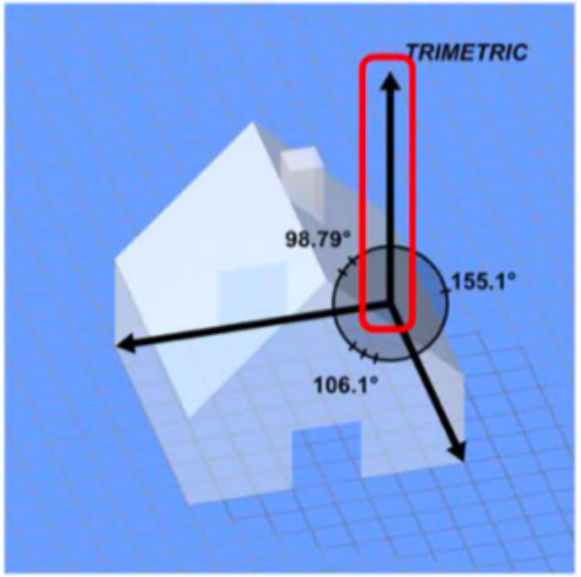


Axonometric projections

One main property of axonometric projections is maintaining the up axis (*y-axis* in our case) parallel to the vertical direction of the screen.

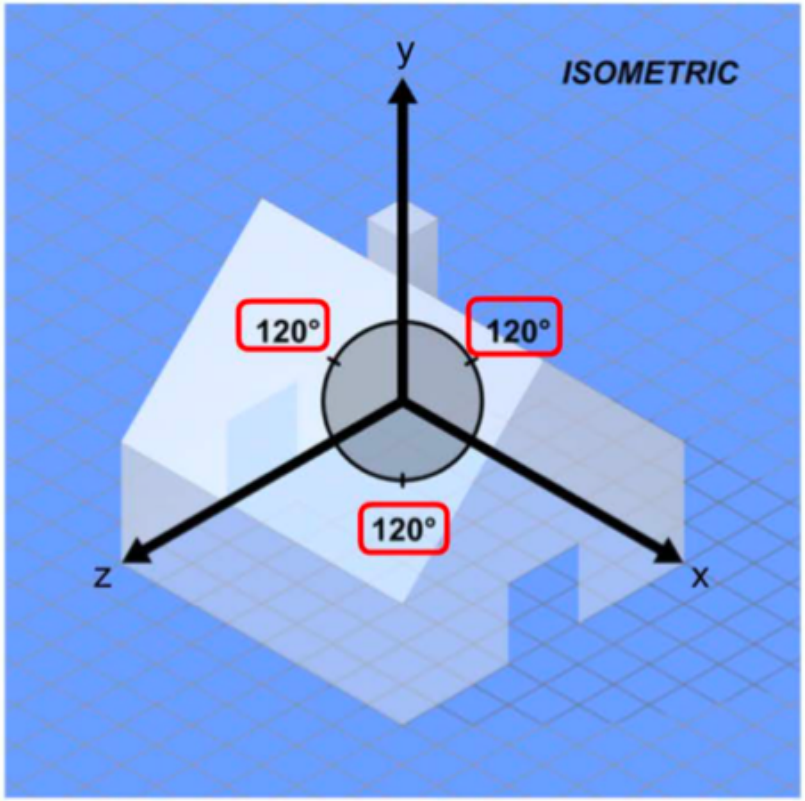






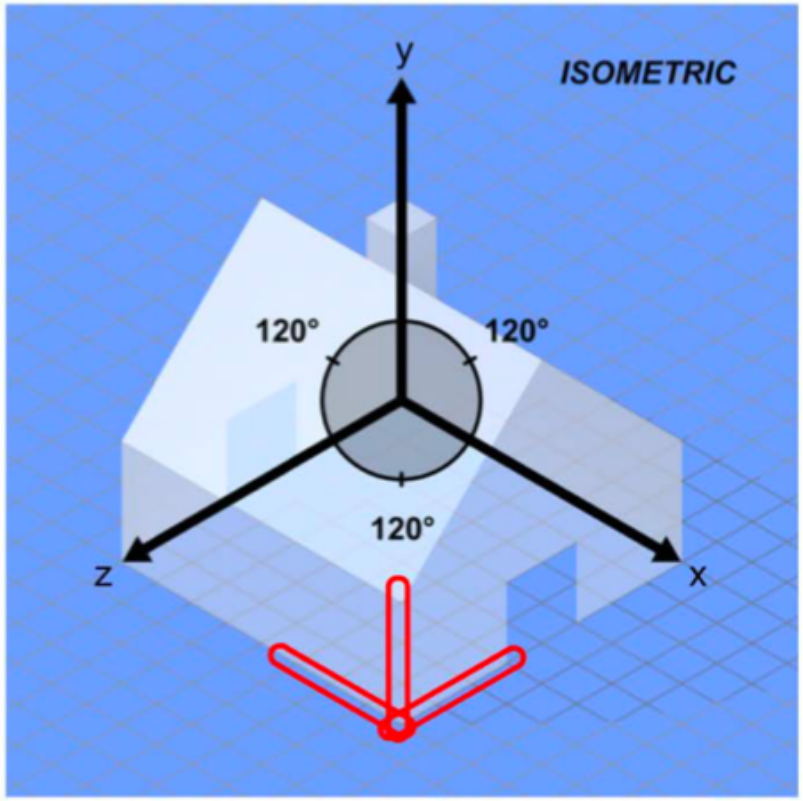
Axonometric projections: Isometric

In *isometric* projections the three axes are angled at 120º one from the other.



Axonometric projections: Isometric

Segments of equal lengths parallel to the main axis in the 3D space have equal lengths also in the 2D projected plane.



Axonometric projections: Isometric

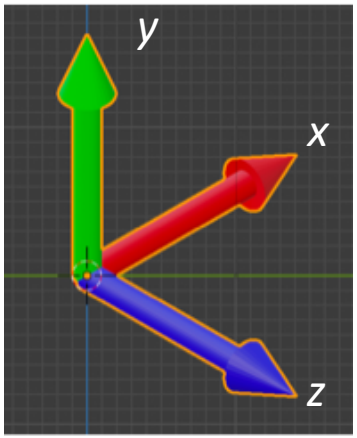
Isometric projections are obtained by applying a rotation of $\pm 45^o$ around the *y-axis*, followed by a rotation of $\pm 35.26^o$ around the *x-axis*, before applying the parallel projection previously seen.

Note that in this case the projection matrix must be specified with the half-width and the aspect ratio, since the border of the box shown on screen are no longer oriented along the main axis.

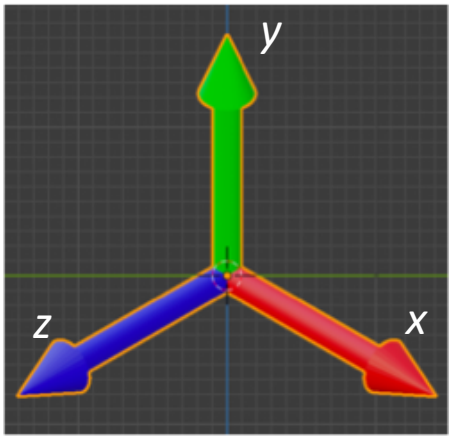
$$\begin{vmatrix} \frac{1}{w} & 0 & 0 & 0 \\ 0 & \frac{-a}{w} & 0 & 0 \\ 0 & 0 & \frac{1}{n-f} & \frac{n}{n-f} \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 35.26^o & \sin 35.26^o & 0 \\ 0 & -\sin 35.26^o & \cos 35.26^o & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \cos 45^o & 0 & \sin 45^o & 0 \\ 0 & 1 & 0 & 0 \\ -\sin 45^o & 0 & \cos 45^o & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Axonometric projections: Isometric

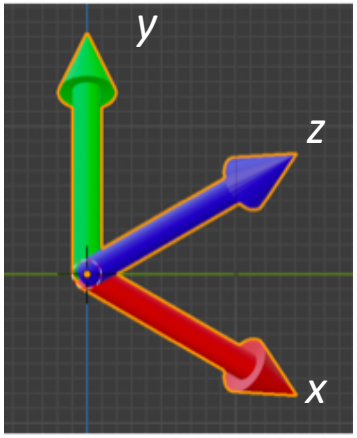
Note also that the sign of the rotations around the x and y axis, creates four radically different, but equally correct, axonometric projections.



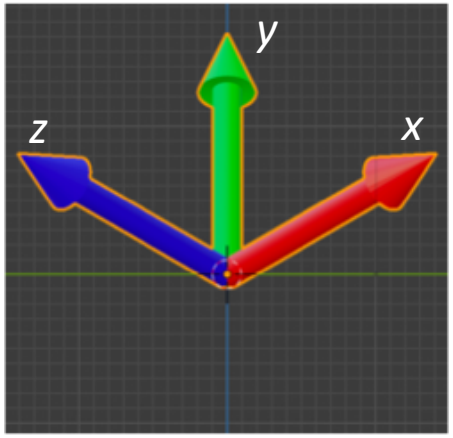
+45º, +35.26º



-45º, +35.26º



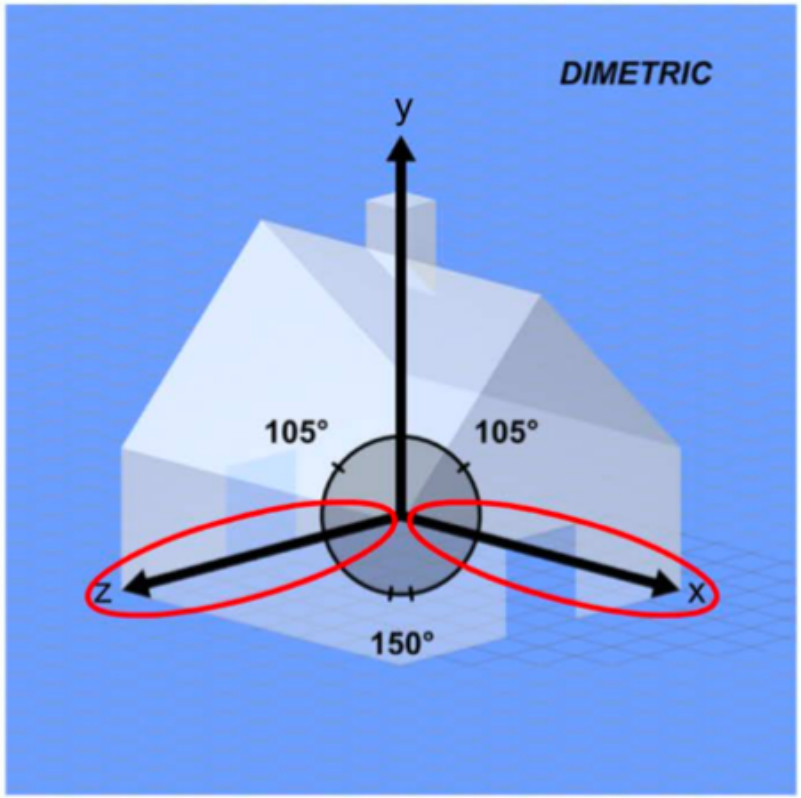
+45º, -35.26º

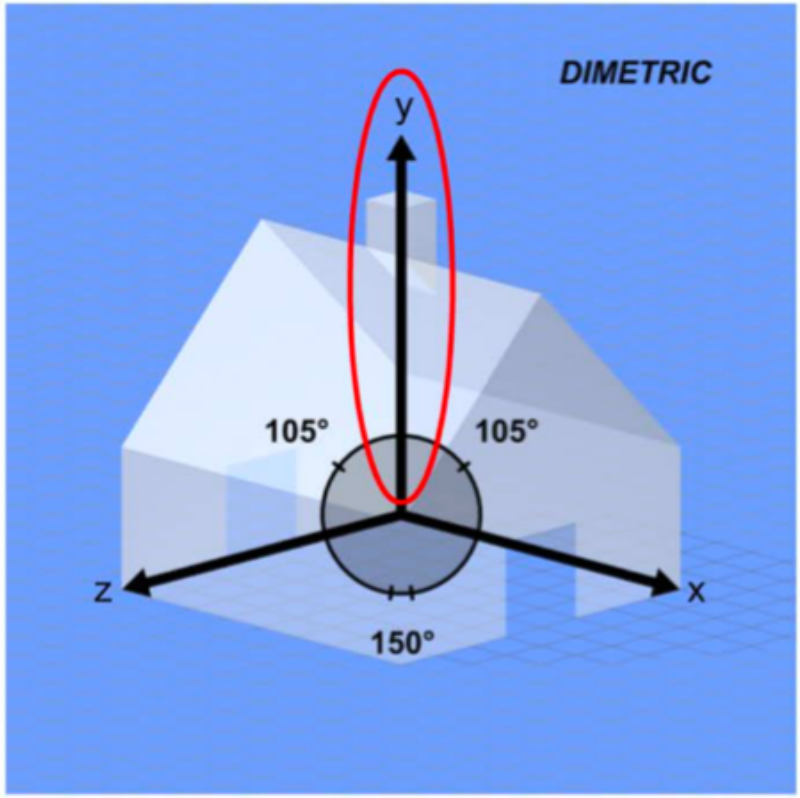


-45º, -35.26º

Axonometric projections: Dimetric

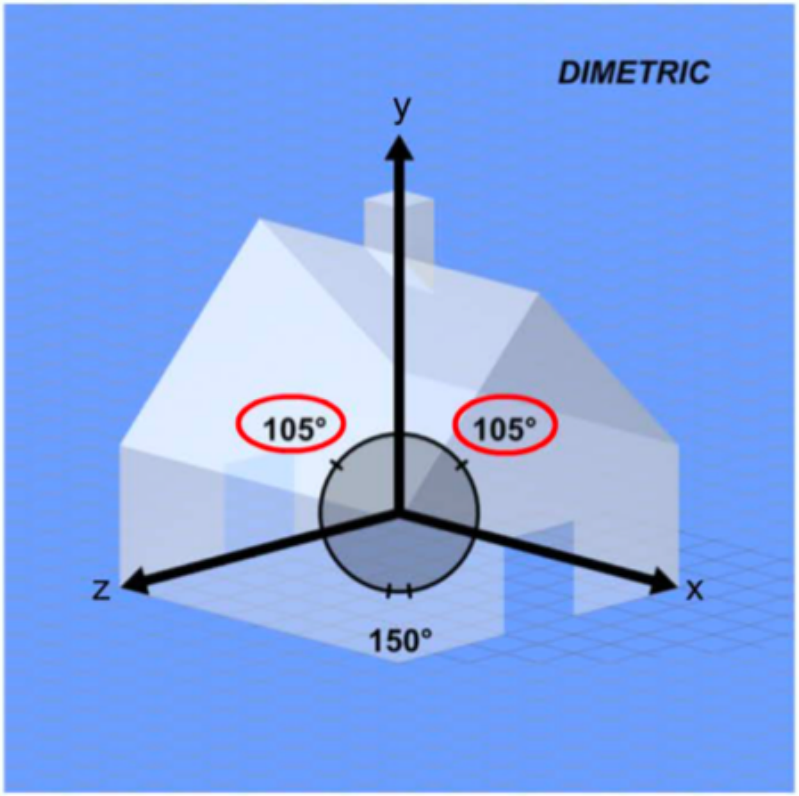
Dimetric projections have two different units: one for the *x* and *z-axis* and one for the *y-axis*.





Axonometric projections: Dimetric

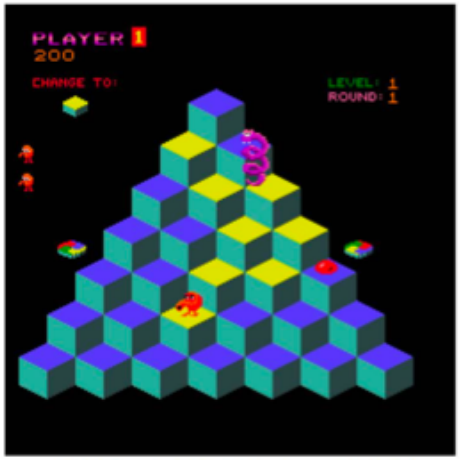
The angle between the *x* and *y-axis* is equal to the one between the *y* and *z-axis*, but different from the one between the *z* and *x-axis*



Axonometric projections: Dimetric

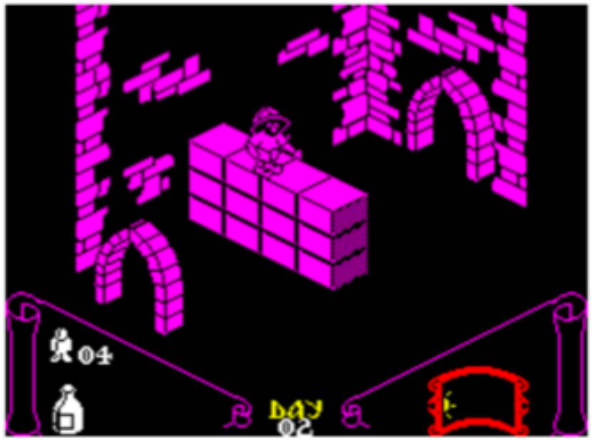
Due to its ease of implementation using only integer arithmetic, It was used a lot in 80's, both for arcade games such as Q*Bert or Zaxxon, and in many home games such as Knight Lore on the ZX Spectrum.

Currently, it is still widely used in retro-style games and applications.



Q*Bert (Gottlieb - 1982)

Zaxxon (SEGA - 1982)



Knight Lore (Ultimate Play The Game - 1984)

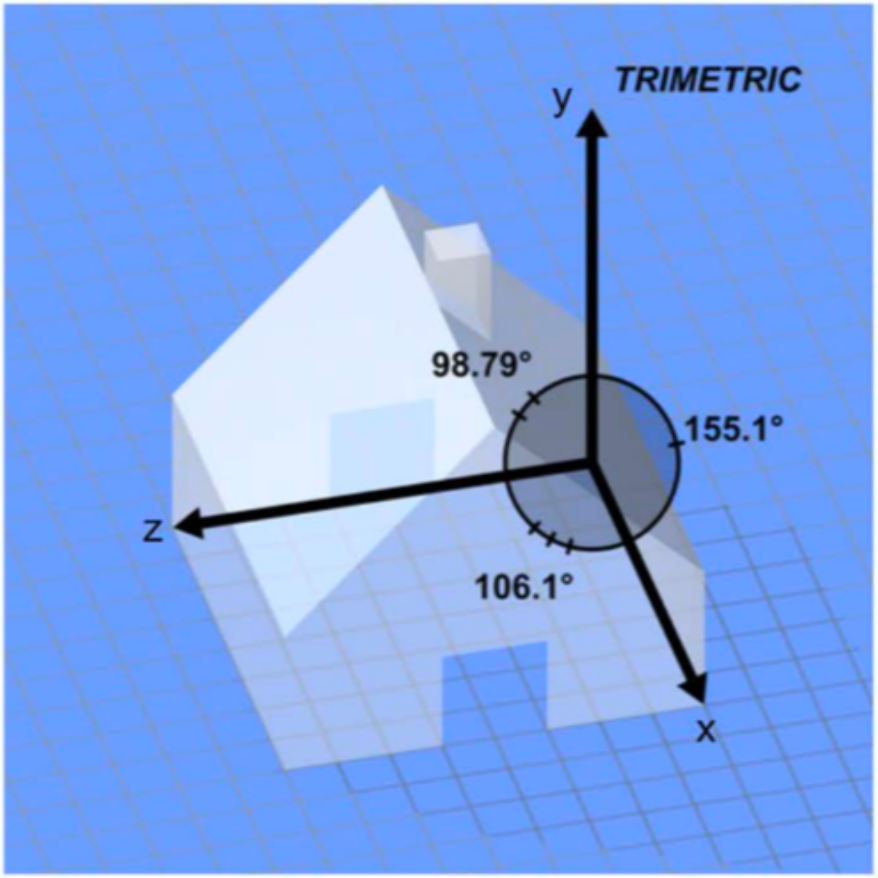
Axonometric projections: Dimetric

Dimetric projections are obtained by applying a rotation of $\pm 45^o$ around the *y-axis*, followed by an arbitrary rotation α around the *x-axis*, before applying the basic parallel projection.

$$\begin{vmatrix} \frac{1}{w} & 0 & 0 & 0 \\ 0 & \frac{-a}{w} & 0 & 0 \\ 0 & 0 & \frac{1}{n-f} & \frac{n}{n-f} \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \cos 45^o & 0 & \sin 45^o & 0 \\ 0 & 1 & 0 & 0 \\ -\sin 45^o & 0 & \cos 45^o & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

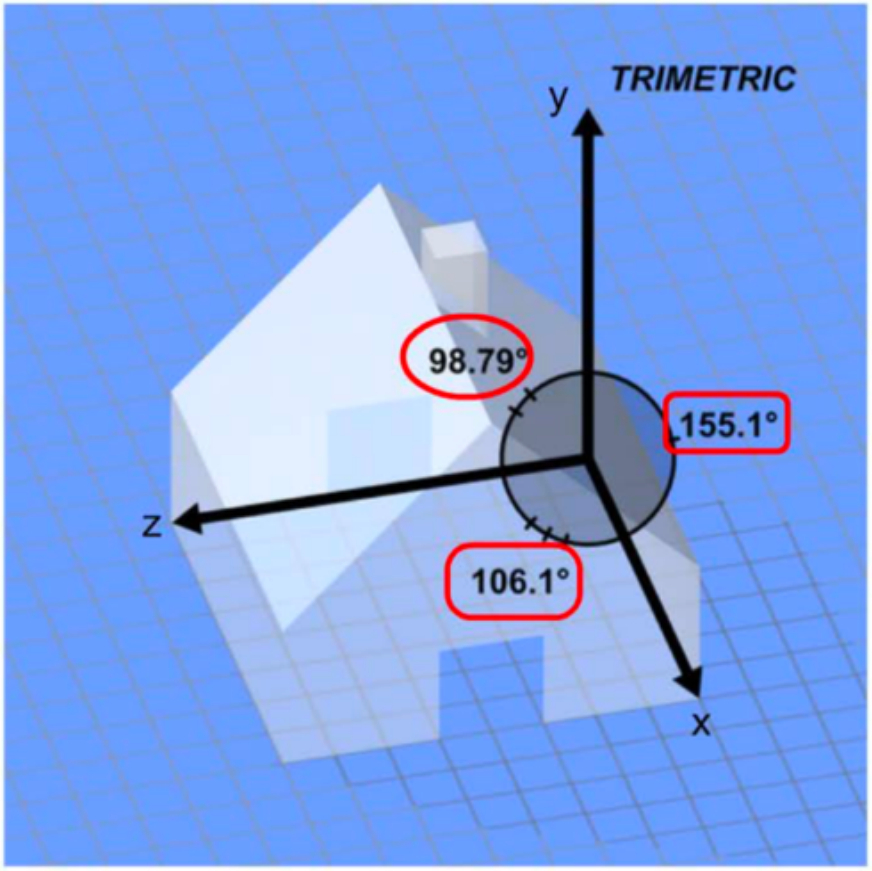
Axonometric projections: Trimetric

Trimetric projections allow a different unit for the three axes.



Axonometric projections: Trimetric

All the angles between the projections of the axes are different.



Axonometric projections: Trimetric

An example of a game using trimetric projections was the first Fallout.

Fallout (Interplay Entertainment - 1997)

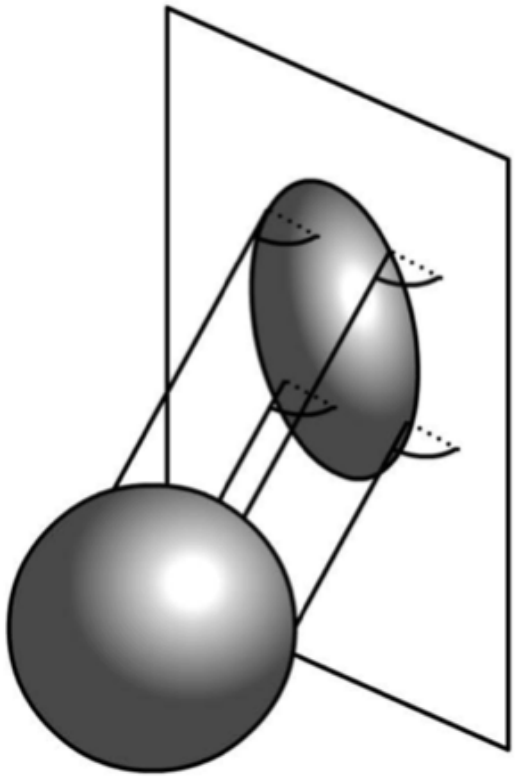
Axonometric projections: Trimetric

Trimetric projections are obtained by applying an arbitrary rotation β around the *y-axis*, followed by an arbitrary rotation α around the *x-axis*, before applying the parallel projection previously seen.

$$\begin{vmatrix} \frac{1}{w} & 0 & 0 & 0 \\ 0 & \frac{-a}{w} & 0 & 0 \\ 0 & 0 & \frac{1}{n-f} & \frac{n}{n-f} \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

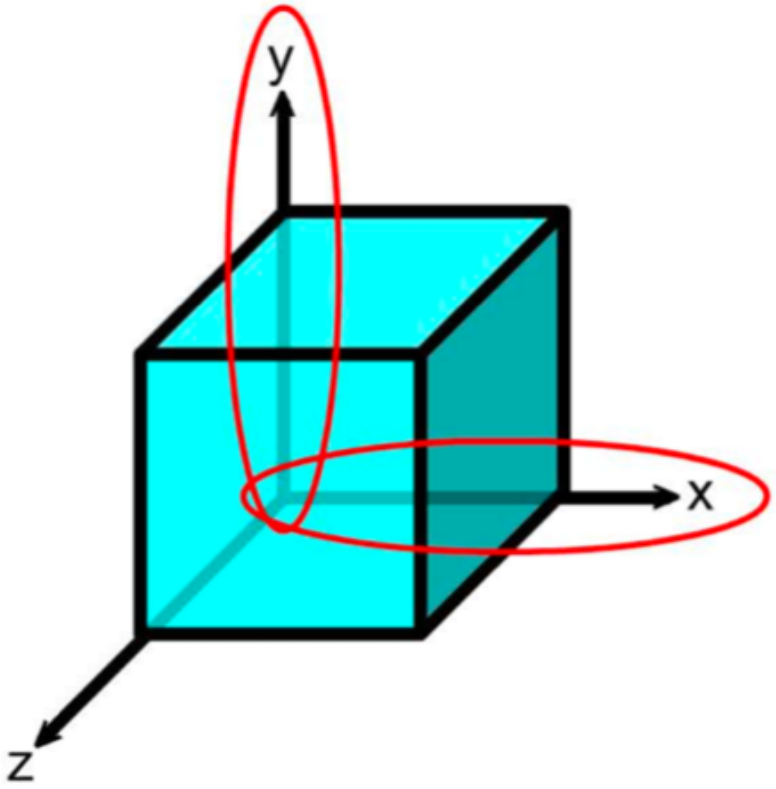
Oblique projections

In oblique projections rays are parallel, but oblique with respect to the projection plane.



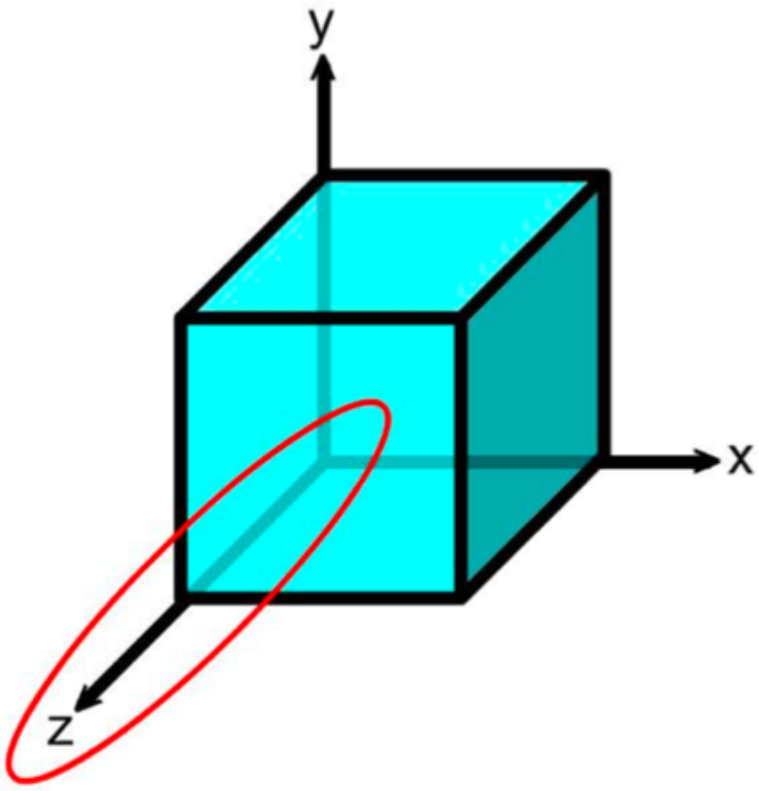
Oblique projections

This has the effect that two of the three axes (namely *x* and *y*) are parallel to the screen.



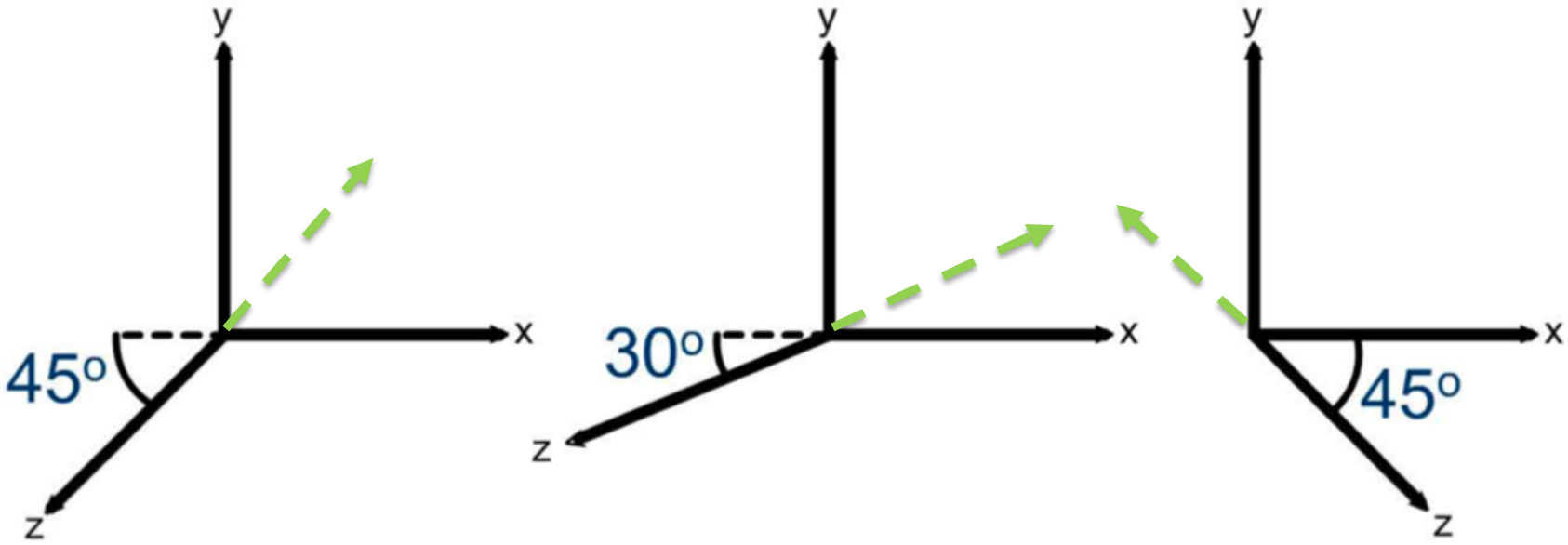
Oblique projections

The third one (the *z-axis*) is oriented at an angle with respect to the previous two.



Oblique projections

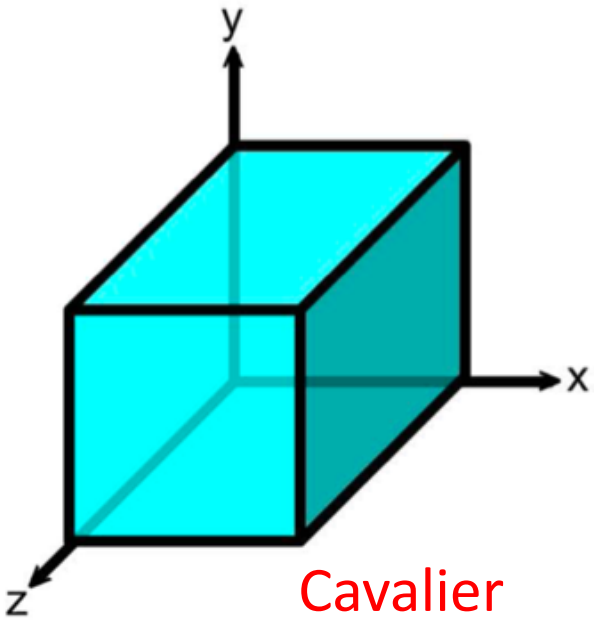
The *z-axis* could be angled in different ways, but it is usually 45º, 30º or 60º, and can be oriented in both directions.



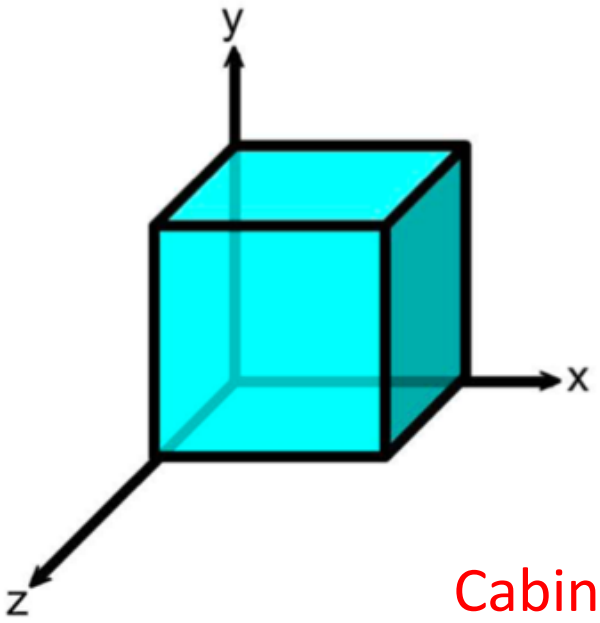
Oblique projections

The length of the *z-axis* can either be equal to the one of the two other axes or halved.

If the length is maintained, the projection is called *Cavalier*, otherwise it is called *Cabinet*.



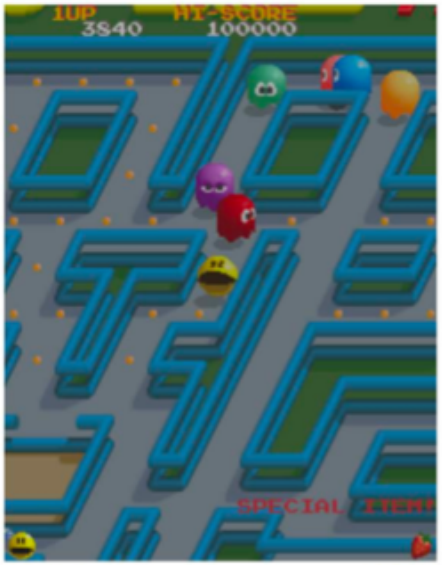
Cavalier



Cabinet

Oblique projections

It was used in some arcade and PC games again for it ease to implement using only integer arithmetic.



Pac-Mania (Namco - 1987)

The staff of Karnath (Ultimate Play The Game - 1984)

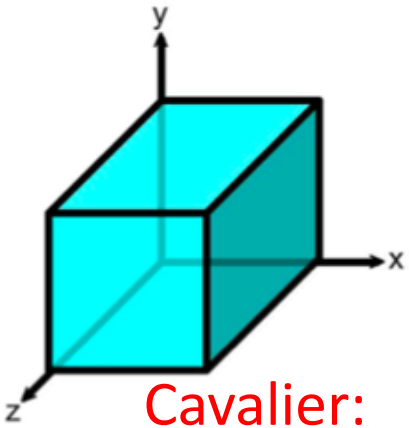
Oblique projections

Oblique projections can be obtained by applying a shear along the *z-axis* before the orthogonal projection.

The shear factor will determine the angle of projection, and whether it will be Cavalier or Cabinet.

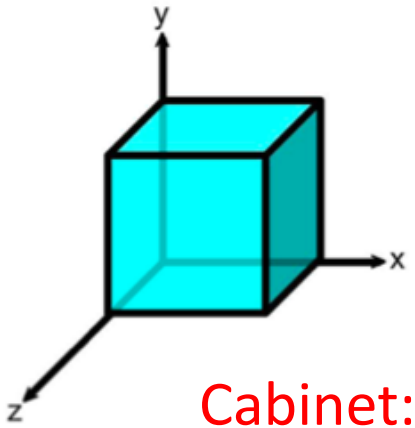
In particular, it can be expressed considering the angle α of the axis, and the corresponding reduction factor ρ.

$$\begin{vmatrix} \frac{1}{w} & 0 & 0 & 0 \\ 0 & \frac{-a}{w} & 0 & 0 \\ 0 & 0 & \frac{1}{n-f} & \frac{n}{n-f} \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & -\rho\cos\alpha & 0 \\ 0 & 1 & -\rho\sin\alpha & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$



Cavalier:

α=45º, ρ=1



Cabinet:

α=45º, ρ=0.5

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)