POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

DEIB

2024

Dipartimento di Elettronica, Informazione e Bioingegneria

Computer Graphics

Milano, 2024

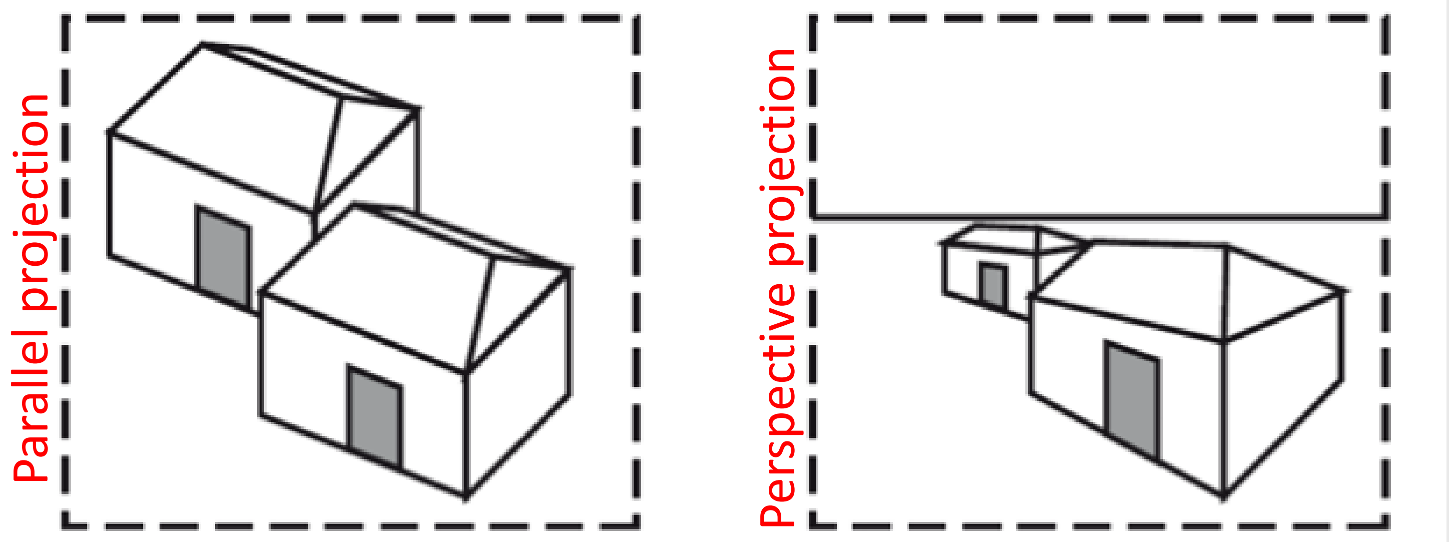
Computer Graphics

- Perspective Projections

Perspective

Parallel projections do not change the apparent size of an object with the distance from the observer. This type of projection is generally used for technical drawings.

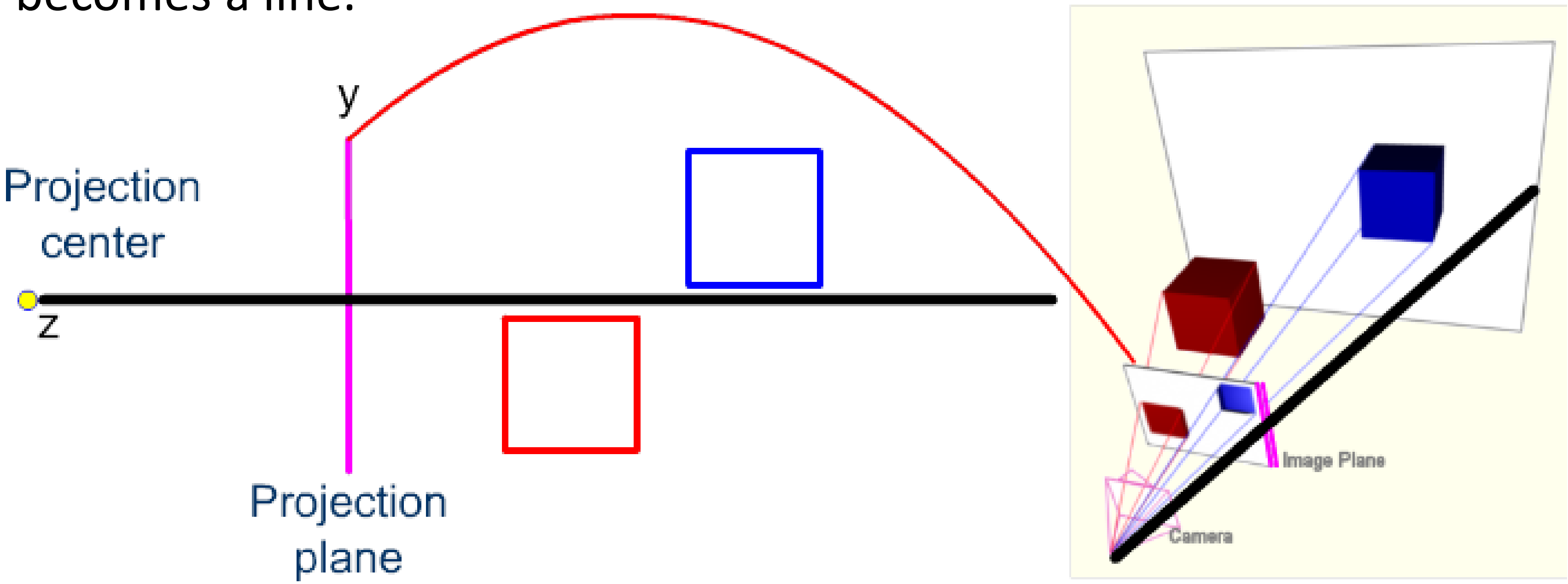
Perspective projections represent an object with a different size depending on its distance from the projection plane. This makes it more suitable for immersive visualizations.



Perspective

The magnification effect is due to the fact that all the projection rays pass through the same point.

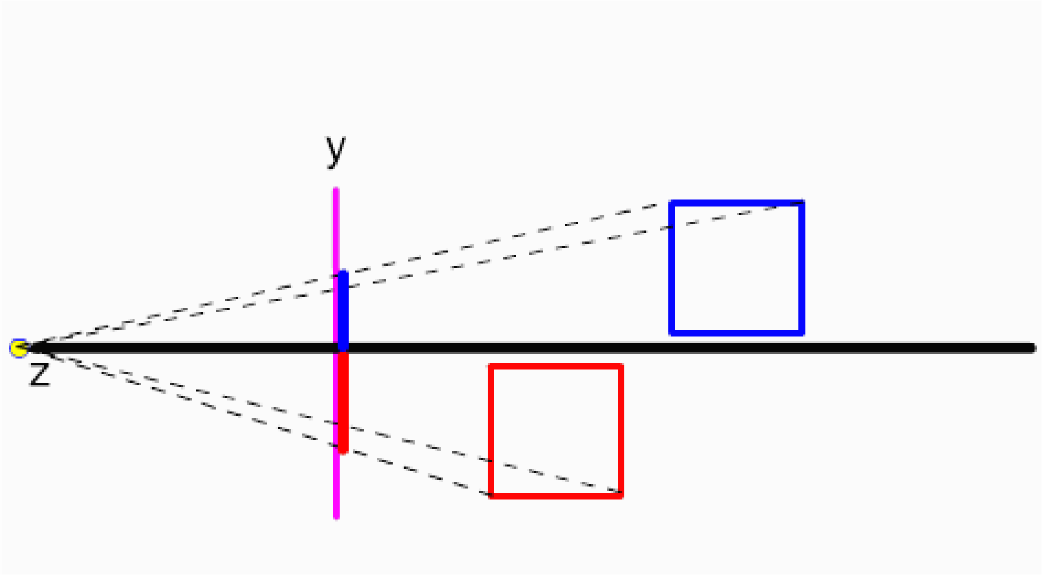
To simplify the discussion, let us focus on a side view: the projection plane becomes a line.

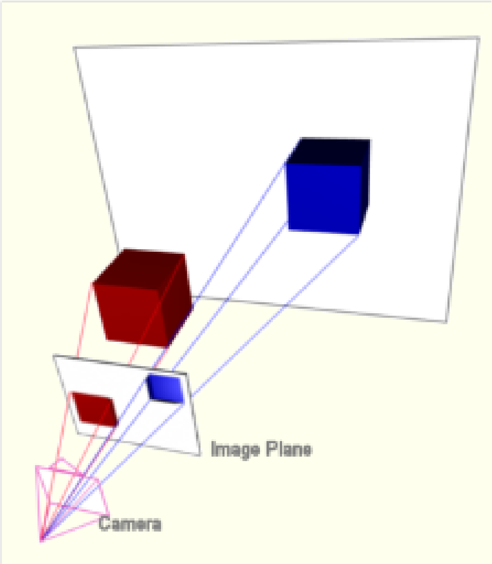


Perspective

Rays intersects the projection plane at different points depending on the distance of the object.

If we compare segments coming from objects with the same size but at different distances, we can see that the ones that are closer to the plane have a larger projection.

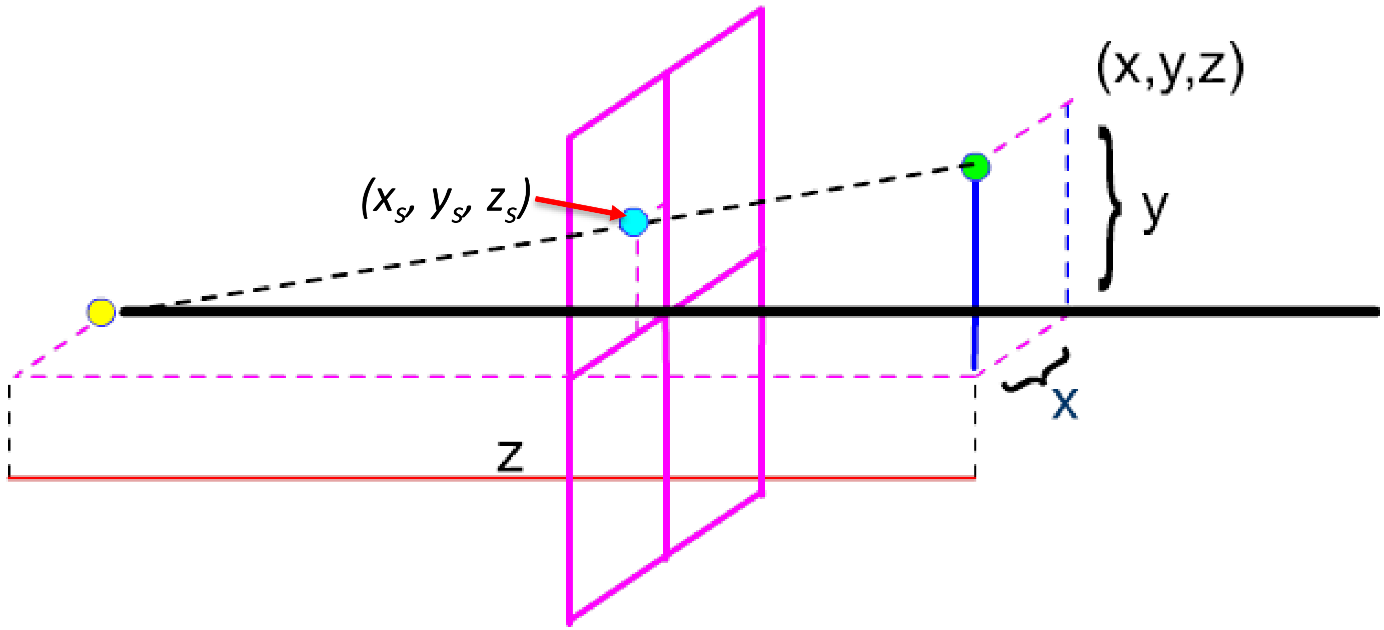




Perspective

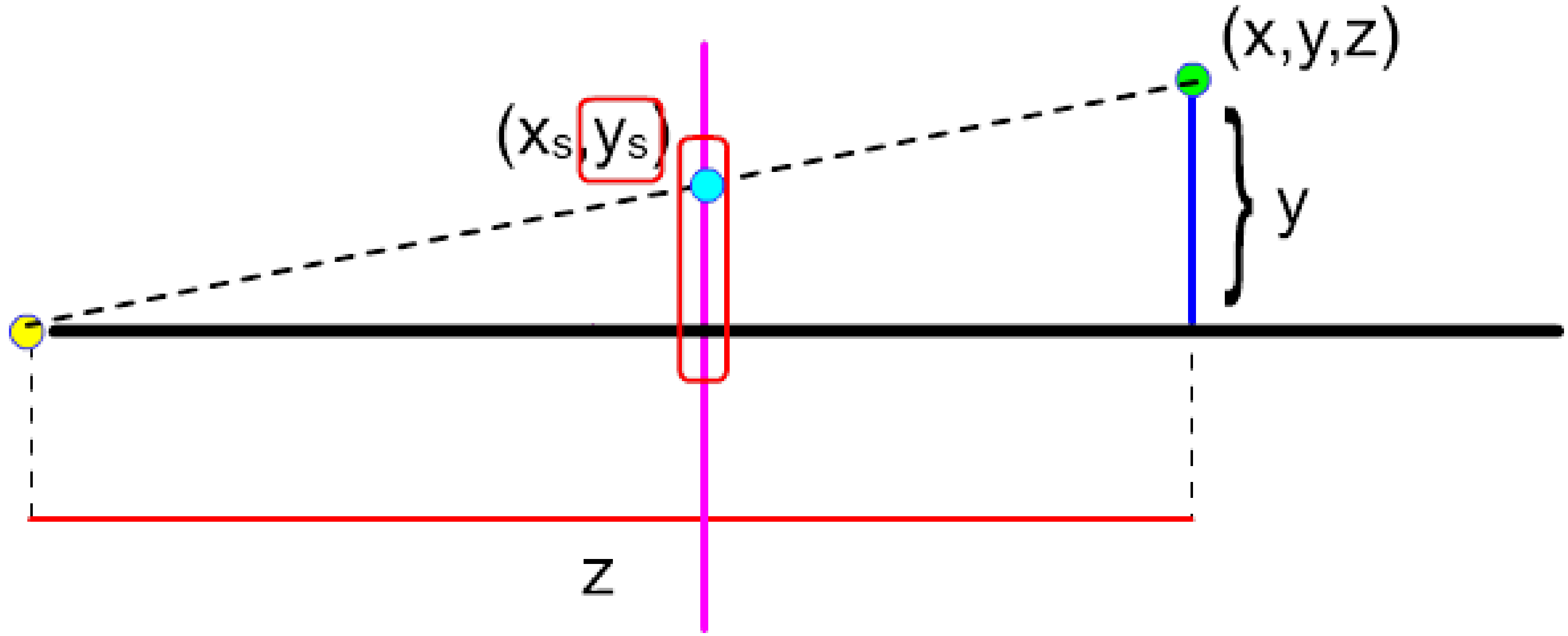
A point with coordinates *(x,y,z)* in the space, is projected on the plane to a point with *Normalized Screen Coordinates* (x_s, y_s, z_s), where z_s is required to sort points according to the distance from the viewer, as for parallel projections.

Let us initially we focus on y_s, then on x_s and finally on z_s.



Perspective

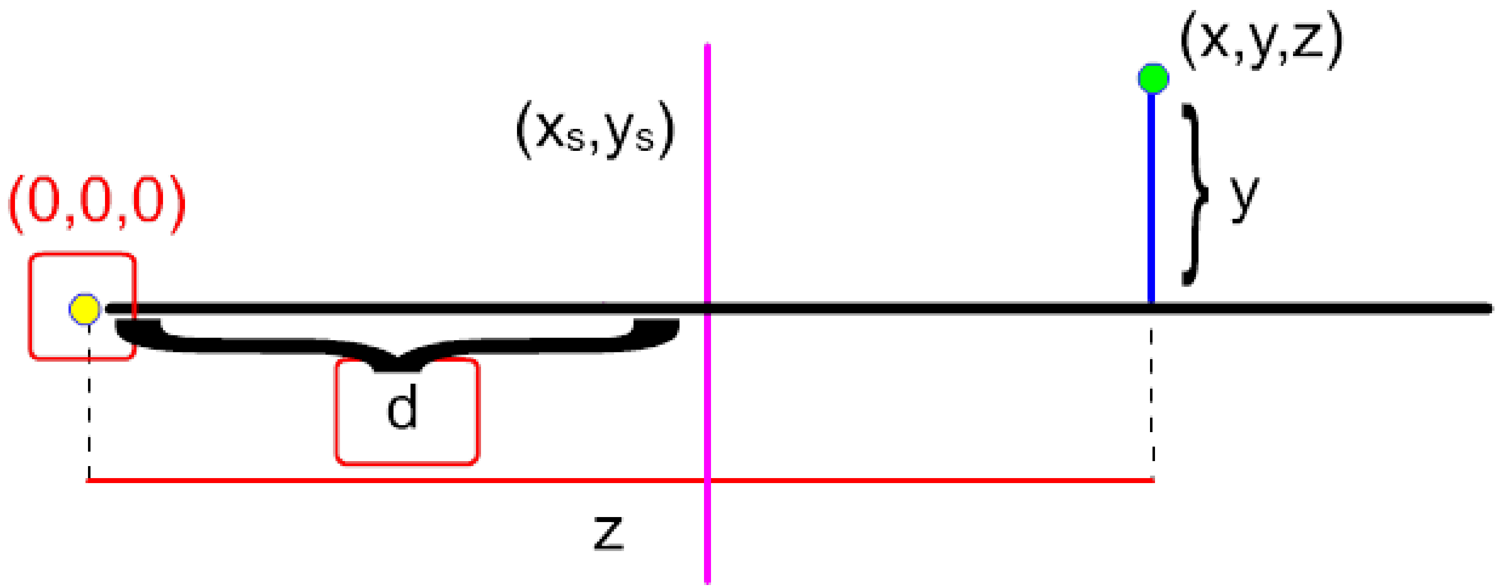
In particular, focusing on *the y-axis*, we can determine the normalized screen coordinate y_s as the projection of the vertical component of the point *(x,y,z)* on the plane.



Perspective

To simplify the computation, we put the center of projection in the origin *(0,0,0)*.

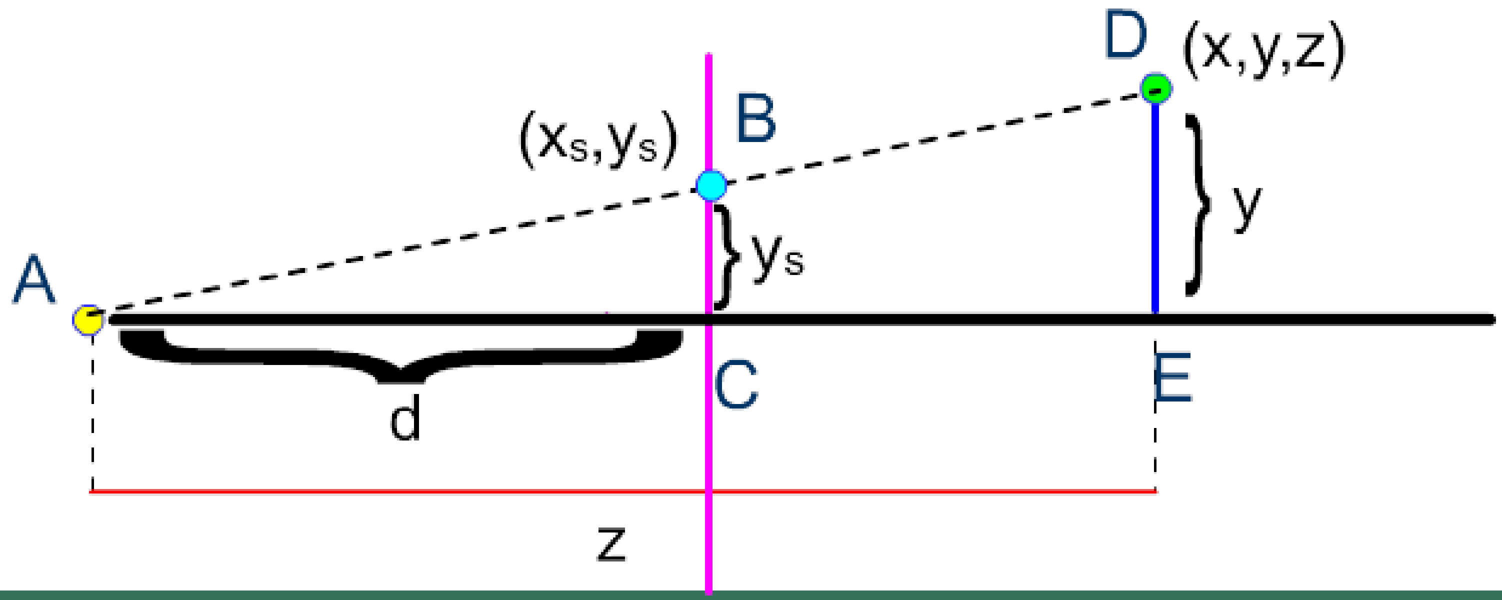
The projection plane is located at a distance *d* on the *z-axis* from the projection center.



Perspective

If we look at the picture below, we can observe two similar triangles: *ABC* and *ADE*.

Here, y_s represents the height of the smaller triangle, and the world coordinate y the height of the larger one.

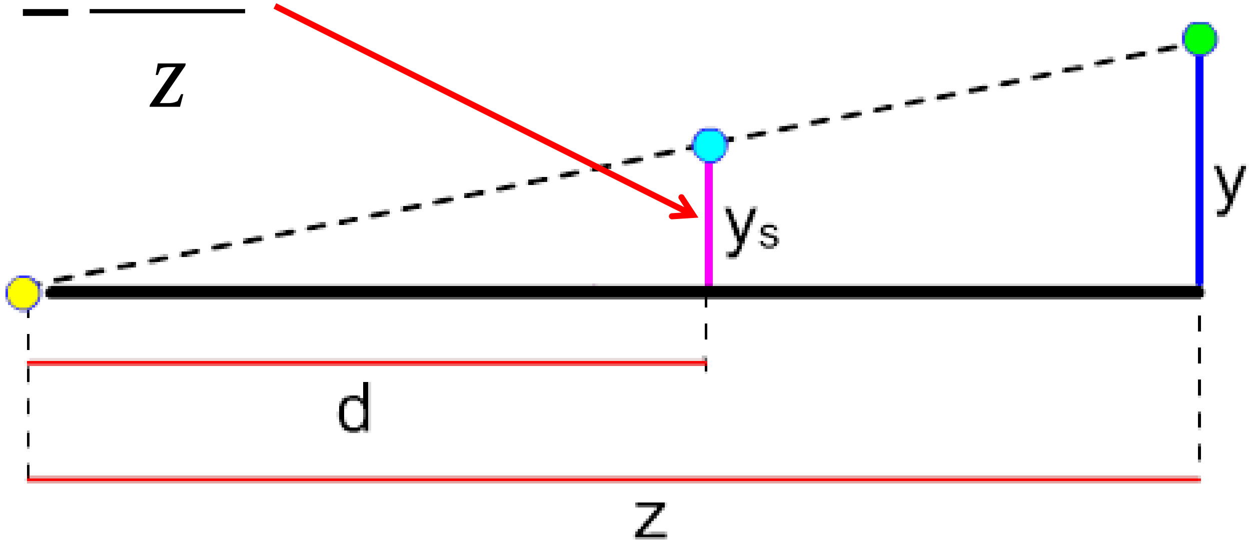


Since the two triangles are similar, there is a linear proportion among the lengths of their edges.

In particular we have:

$$y_s : d = y : z$$

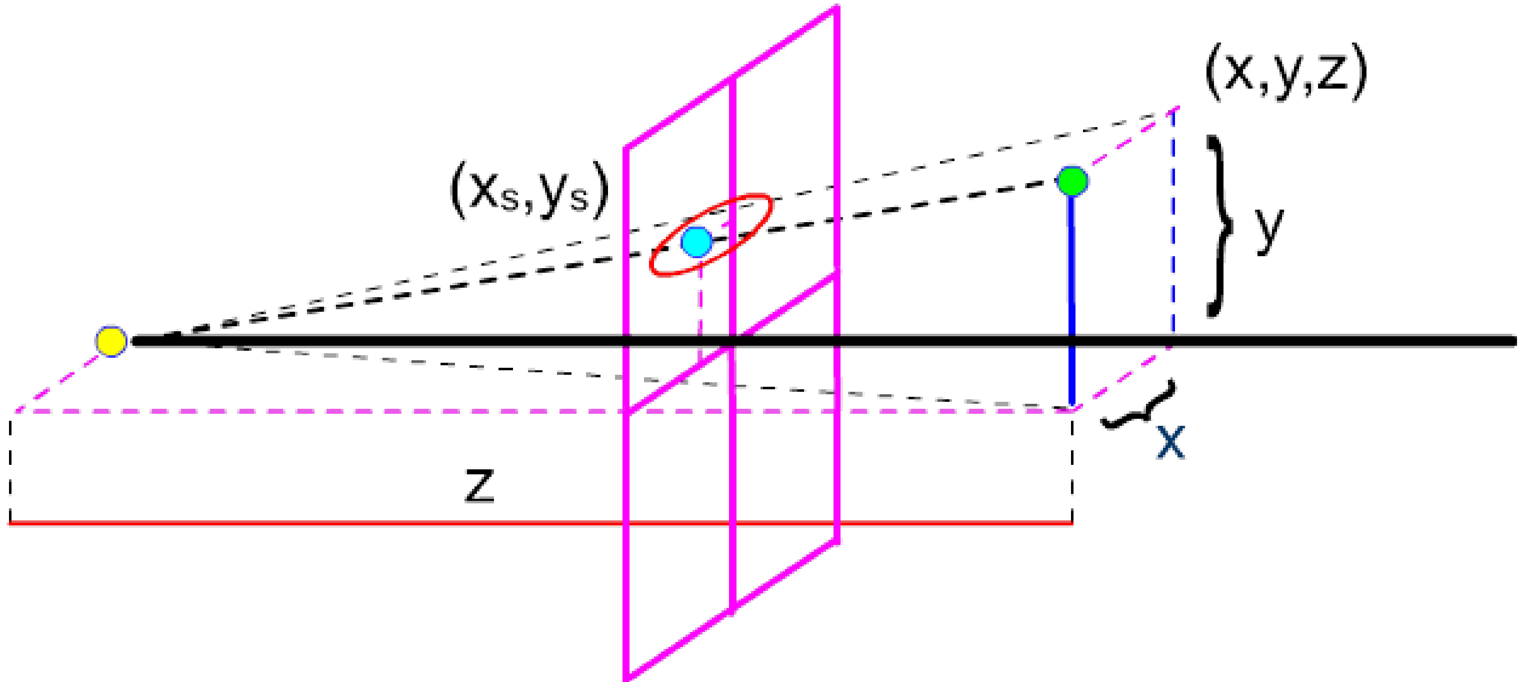
$$y_s = \frac{d \times y}{z}$$



Perspective

The same reasoning can also be repeated for x_s:

$$x_s = \frac{d \times x}{z} \qquad y_s = \frac{d \times y}{z}$$



Perspective

Parameter *d* represents the distance of the center of projection from the projection plane.

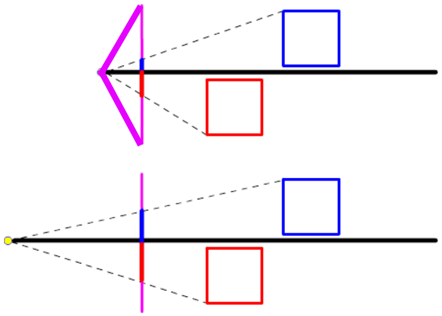
It can be used to simulate the focal length of the lens of a camera.

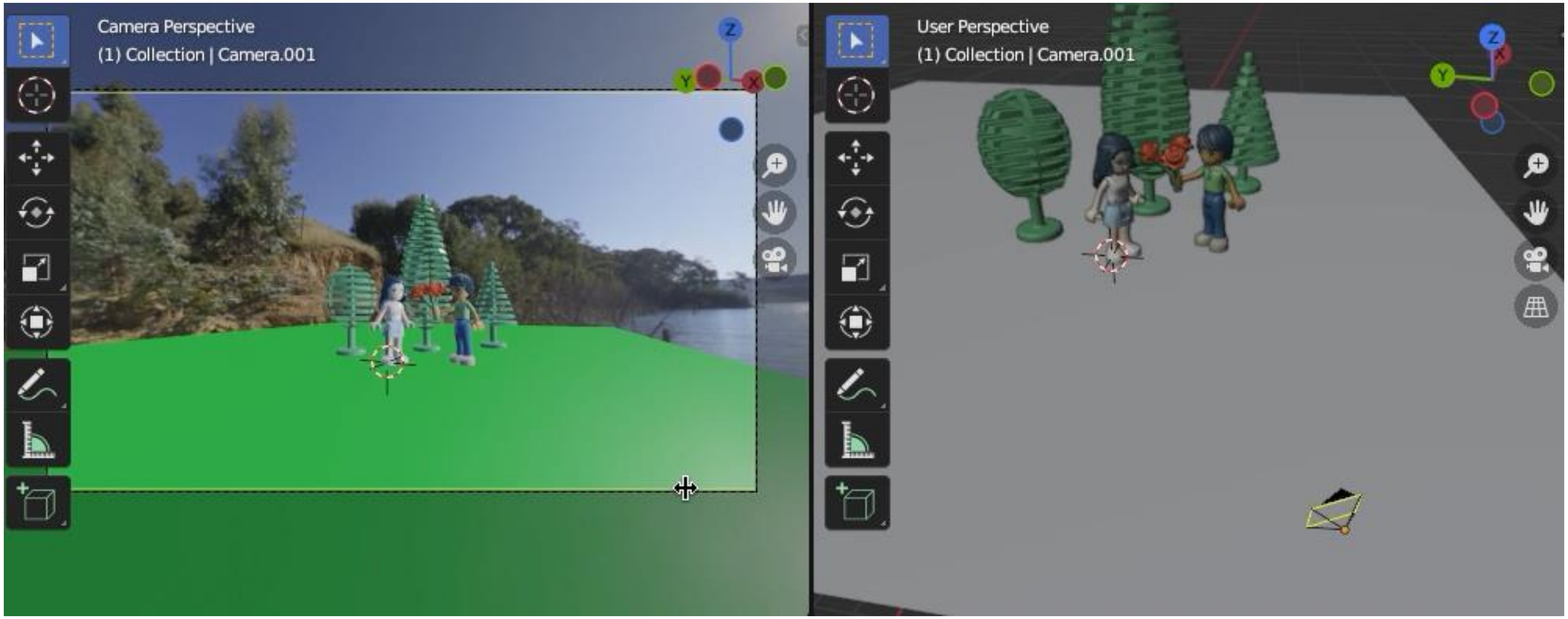
In particular, changing *d* has the effect of performing a zoom.

Perspective

Short *d* corresponds to wide-lens: it emphasizes the distances of the objects from the plane.

It also allows to capture a larger number of objects in the view, producing "smaller" objects in the images.

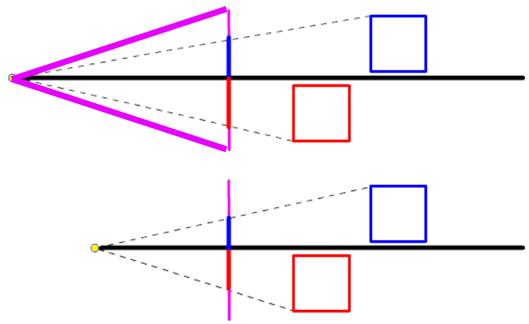


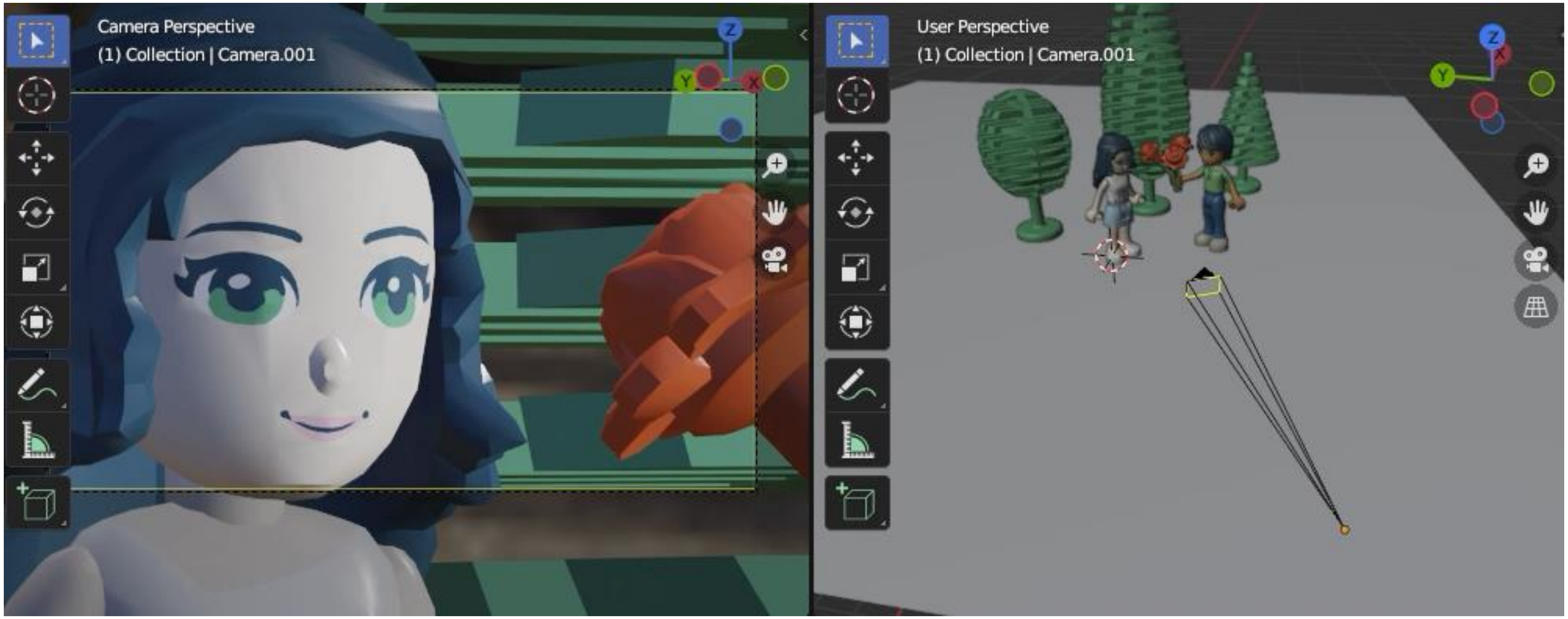


Perspective

Large *d* corresponds to tele-lens, reducing the differences in size for objects at different distances.

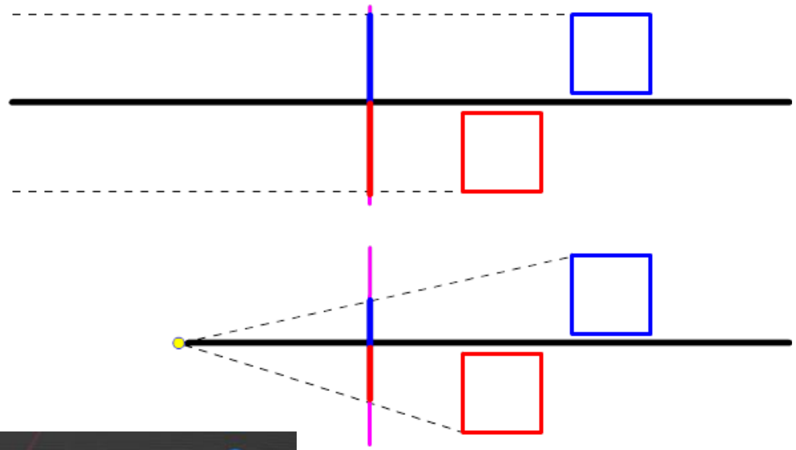
It also has the effect of reducing the number of objects visible in the scene, producing "enlarged" views.

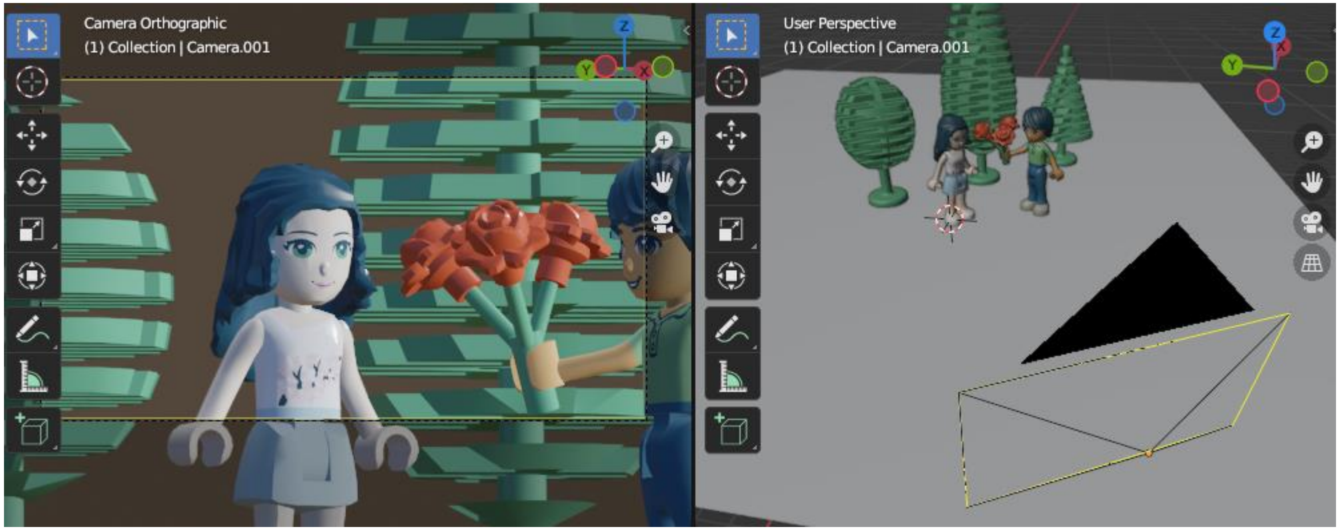




Perspective

Parallel (orthographic) projections can be obtained from perspective as *d* tends to the infinity.

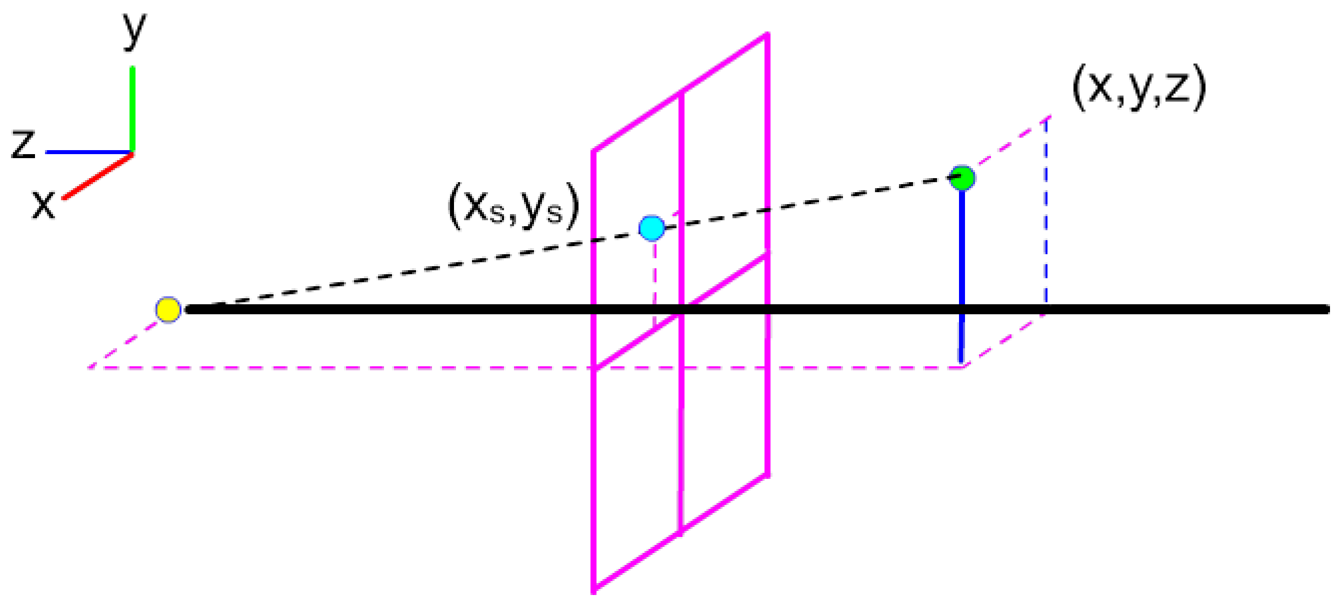




Perspective Projections Matrices

Thanks to homogeneous coordinates, perspective projections can be obtained with a matrix-vector product as well.

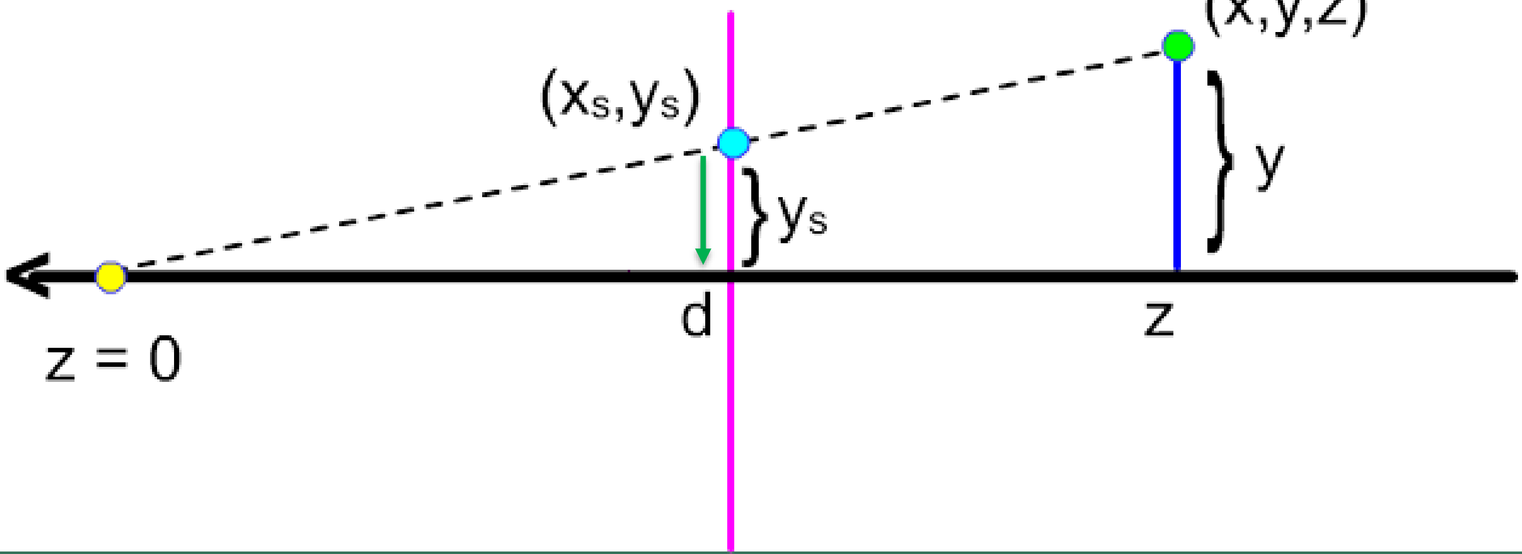
First of all we have to note that the considered world coordinate system is oriented in the opposite direction on the *z-axis*: the *z* coordinates are indeed negative.



Perspective Projections Matrices

The considered formula needs a change of sign to account for the real direction of the *z-axis*. In Vulkan, also the *y-axis* should be mirrored: however it will be simpler to do this in a separate last step.

$$x_s = \frac{d \times x}{-z} \qquad y_s = \frac{d \times y}{-z}$$



Perspective Projections Matrices

The projection matrix for perspective, with center in the origin and projection plane at distance *d* on the z-axis, can be defined as:

$$P_{persp} = \begin{vmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & -1 & 0 \end{vmatrix}$$

Since:

$$\begin{vmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & -1 & 0 \end{vmatrix} \times \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix} = \begin{vmatrix} d \times x \\ d \times y \\ d \times z \\ -z \end{vmatrix}$$

Perspective Projections Matrices

Note that the last row of this transform matrix is no longer equal to *|0 0 0 1|*. This also makes the product of matrix P_{persp} with an homogenous coordinate resulting in a vector with component $w \neq 1$.

$$\begin{vmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ \boxed{0 \quad 0 \quad -1 \quad 0} \end{vmatrix} \times \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix} = \begin{vmatrix} d \times x \\ d \times y \\ d \times z \\ \boxed{-z} \end{vmatrix}$$

If we divide by the *w* component to obtain the equivalent Cartesian coordinate, we have:

$$x_s = \frac{d \times x}{-z} \qquad y_s = \frac{d \times y}{-z}$$

$$\begin{vmatrix} d \times x & d \times y & d \times z & -z \end{vmatrix} = \begin{vmatrix} \frac{d \times x}{-z} & \frac{d \times y}{-z} & \frac{d \times z}{-z} & 1 \end{vmatrix} = \begin{vmatrix} x_s & y_s & -d & 1 \end{vmatrix}$$

The previous technique has a disadvantage: the *z* component of the equivalent Cartesian coordinate is always *-d* and the information about the distance from the projection plane is completely lost.

This does not allow to define proper *3D Normalized Screen Coordinates* with a z_s component that reflects the distance of the point from the view plane.

Perspective Projections Matrices

Since z_s is used just for sorting the primitives, it is enough to add an element equal to 1 in the third row of the fourth column of the matrix to obtain a z_s component that changes monotonically with the distance of the point.

$$P_{persp} = \begin{vmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & \boxed{1} \\ 0 & 0 & -1 & 0 \end{vmatrix}$$

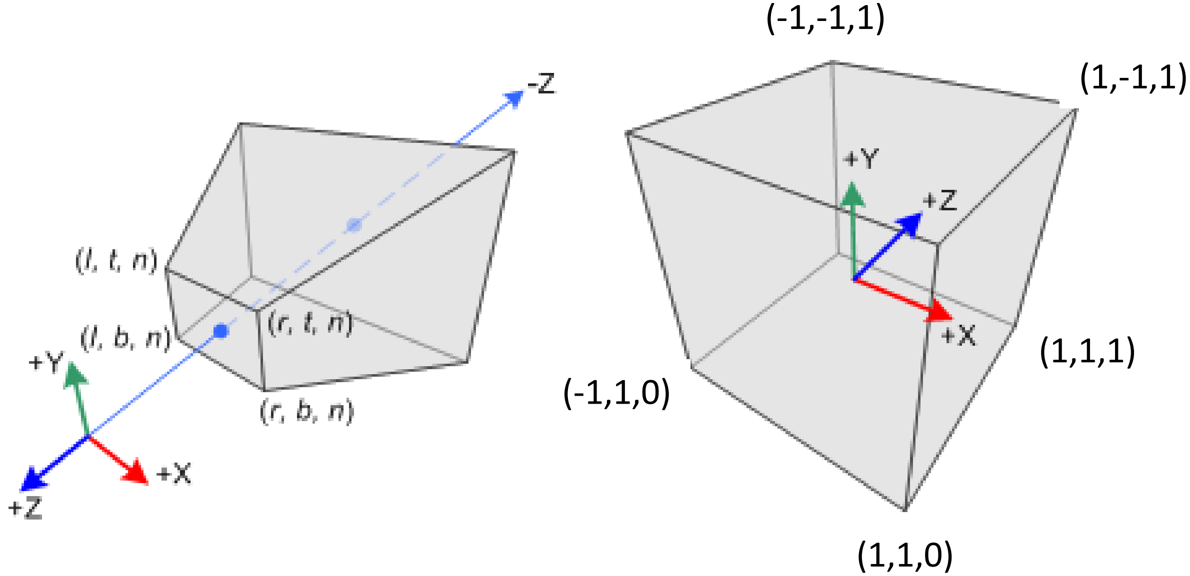
$$\begin{vmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 1 \\ 0 & 0 & -1 & 0 \end{vmatrix} \times \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix} = \begin{vmatrix} d \times x \\ d \times y \\ d \times z + 1 \\ -z \end{vmatrix}$$

$$x_s = \frac{d \times x}{-z} \qquad y_s = \frac{d \times y}{-z}$$

$$\begin{vmatrix} d \times x & d \times y & d \times z + 1 & -z \end{vmatrix} = \begin{vmatrix} \frac{d \times x}{-z} & \frac{d \times y}{-z} & \frac{d \times z + 1}{-z} & 1 \end{vmatrix} = \begin{vmatrix} x_s & y_s & -d \boxed{- \frac{1}{z}} & 1 \end{vmatrix}$$

Perspective Projections Matrices

As for parallel projections, the visible area of the 3D world corresponds to the one for which the normalized screen coordinates are in the [-1, +1] range for x and y, and [0, 1] for the z. The next steps are adding extra transforms to make sure that this range corresponds to the world area we are interested to show.

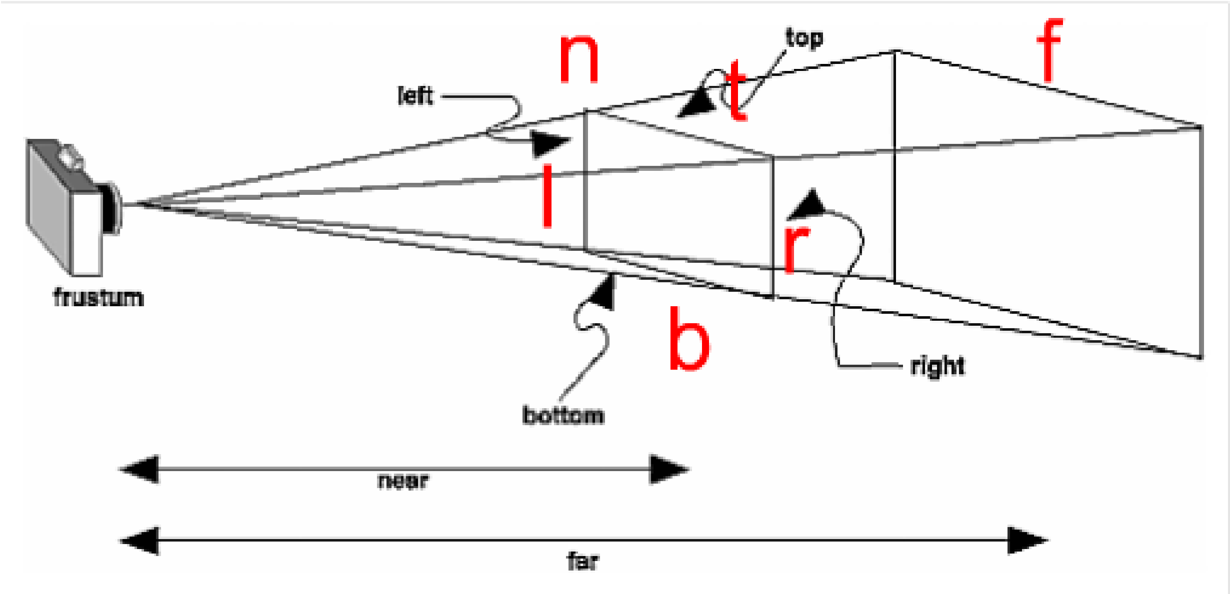


Perspective Projections Matrices

Let us call *n* the distance from the origin of the near plane.

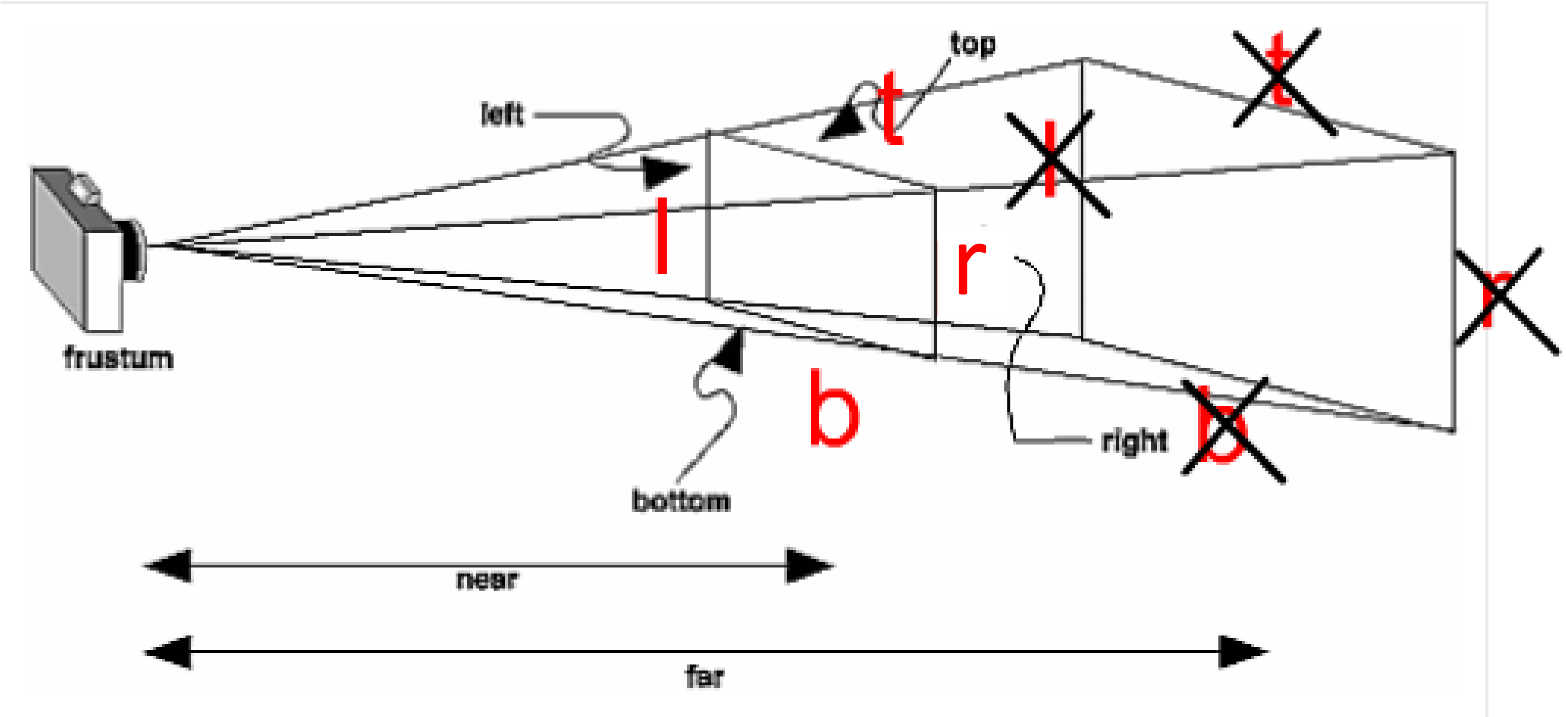
We call *l*, *r*, *t*, and b the coordinates of the left, right, top and bottom edges of the projection plane in the world space at the near plane.

Finally, we call *f* the distance from the origin of the far plane.



Perspective Projections Matrices

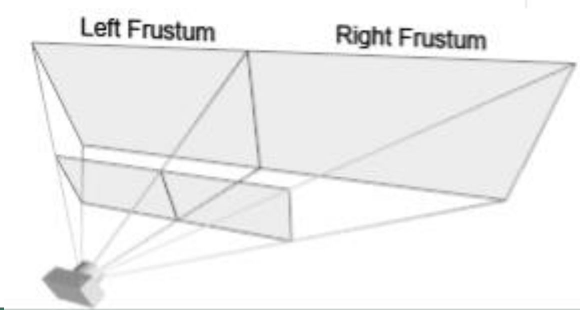
Note that, due to the perspective, the coordinates of the borders of the screen at the far plane are different from the ones at the near plane, and the visible area is not a box but a *frustum*.



Perspective Projections Matrices

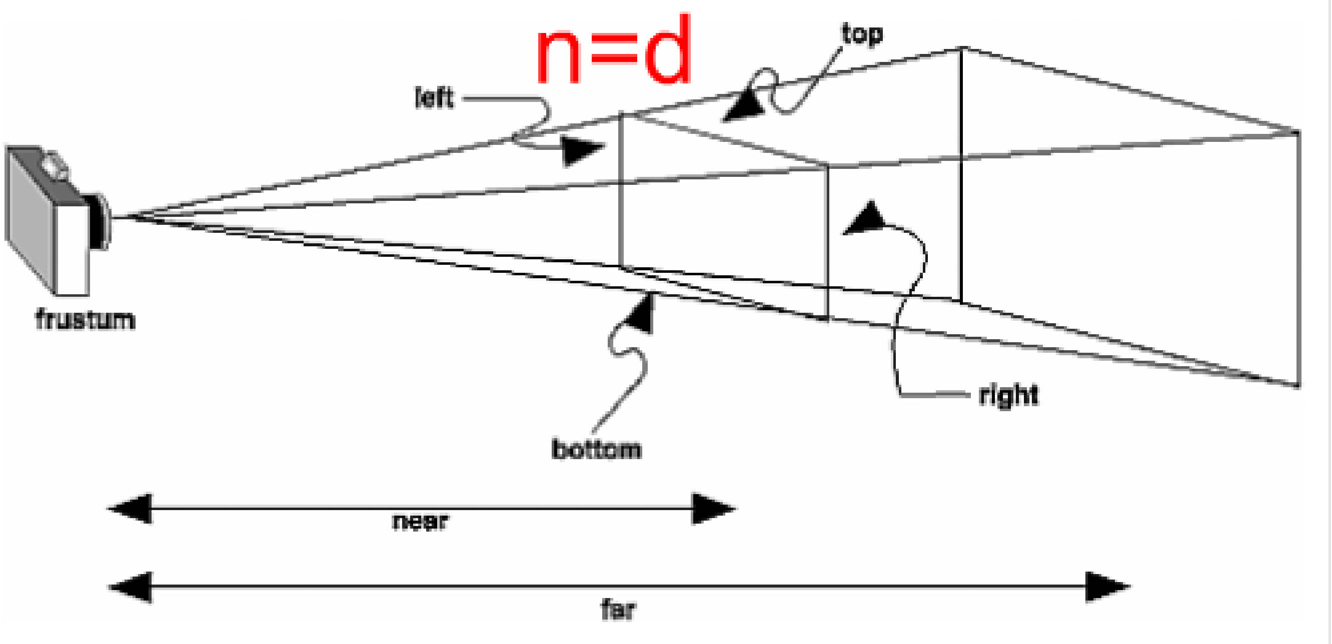
Note also that the coordinates of the borders of the screen do not need to be symmetric.

This can be used to compute "shifted" viewports to share, for example, a projection over two adjacent screens.



Perspective Projections Matrices

Since the coordinates of the border of the screen are specified at the near plane, the value of *n* corresponds to the distance of the projection plane *d*. Note that for perspective, it must be *n > 0*.



Perspective Projections Matrices

The first step is thus writing the projection matrix just introduced, replacing *d* with *n*.

$$U_{persp} = \begin{vmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 1 \\ 0 & 0 & -1 & 0 \end{vmatrix}$$

Perspective Projections Matrices

To obtain Normalized Screen Coordinates, further transformations should be chained.

Since *l*, *r*, *t* and *b* are given at the near plane, we start computing the projections of the top-left and bottom-right corners at the near plane.

$$\begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} l \\ t \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} n \times l \\ n \times t \\ -n \times n + 1 \\ n \end{bmatrix} \qquad \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} r \\ b \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} n \times r \\ n \times b \\ -n \times n + 1 \\ n \end{bmatrix}$$

$$\begin{bmatrix} n \times l & n \times t & -n^2+1 & n \end{bmatrix} = \begin{bmatrix} l & t & \frac{1}{n} - n & 1 \end{bmatrix} \qquad \begin{bmatrix} n \times r & n \times b & -n^2+1 & n \end{bmatrix} = \begin{bmatrix} r & b & \frac{1}{n} - n & 1 \end{bmatrix}$$

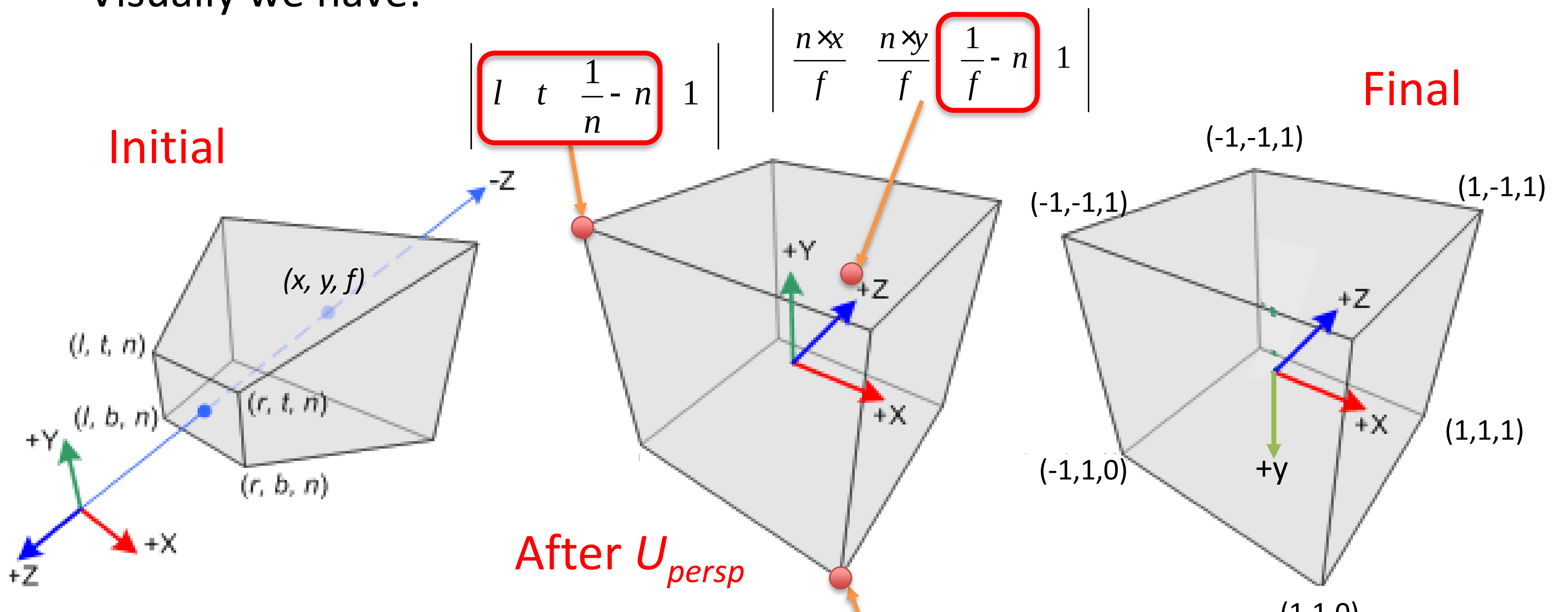
Perspective Projections Matrices

We must also compute the projected coordinate of a point at the far plane (*z=-f*) to determine the proper normalization for the *z axis*.

$$\begin{vmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n & 1 \\ 0 & 0 & -1 & 0 \end{vmatrix} \times \begin{vmatrix} x \\ y \\ -f \\ 1 \end{vmatrix} = \begin{vmatrix} n \times x \\ n \times y \\ -n \times f + 1 \\ f \end{vmatrix}$$

$$\begin{vmatrix} n \times x & n \times f & -n \times f + 1 & f \end{vmatrix} = \begin{vmatrix} \frac{n \times x}{f} & \frac{n \times y}{f} & \boxed{\frac{1}{f} - n} & 1 \end{vmatrix}$$

Visually we have:



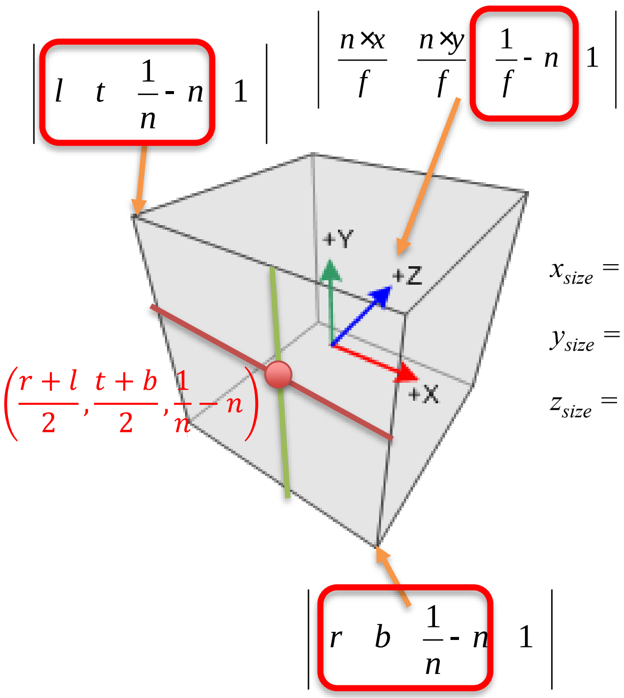
$$x_{size} = r - l \qquad y_{size} = t - b$$

$$z_{size} = \frac{1}{f} - n - \left(\frac{1}{n} - n\right) = \frac{n-f}{nf}$$

Perspective Projections Matrices

Following what we have seen in the previous for parallel projections, we obtain:

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



$x_{size} = r - l$

$y_{size} = t - b$

$z_{size} = \frac{n-f}{nf}$

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

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

Perspective Projections Matrices

Combining all the elements, we obtain:

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

Perspective Projections Matrices

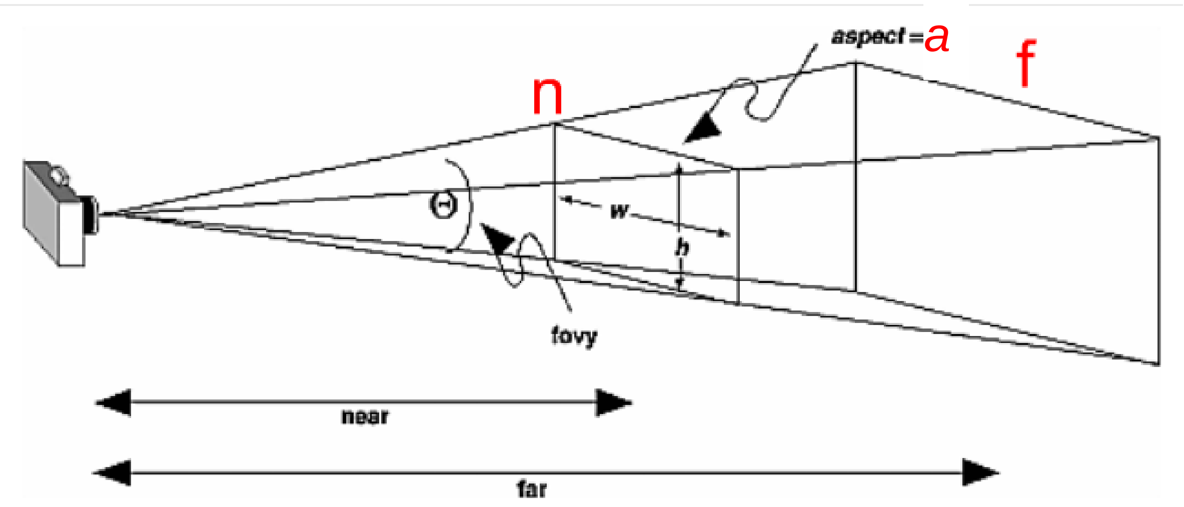
As for parallel projection, the values *l*, *r*, *t* and *b* must be consistent with the aspect ratio *a* of the monitor.

$$r - l = a \cdot (t - b)$$

Projection matrix: camera

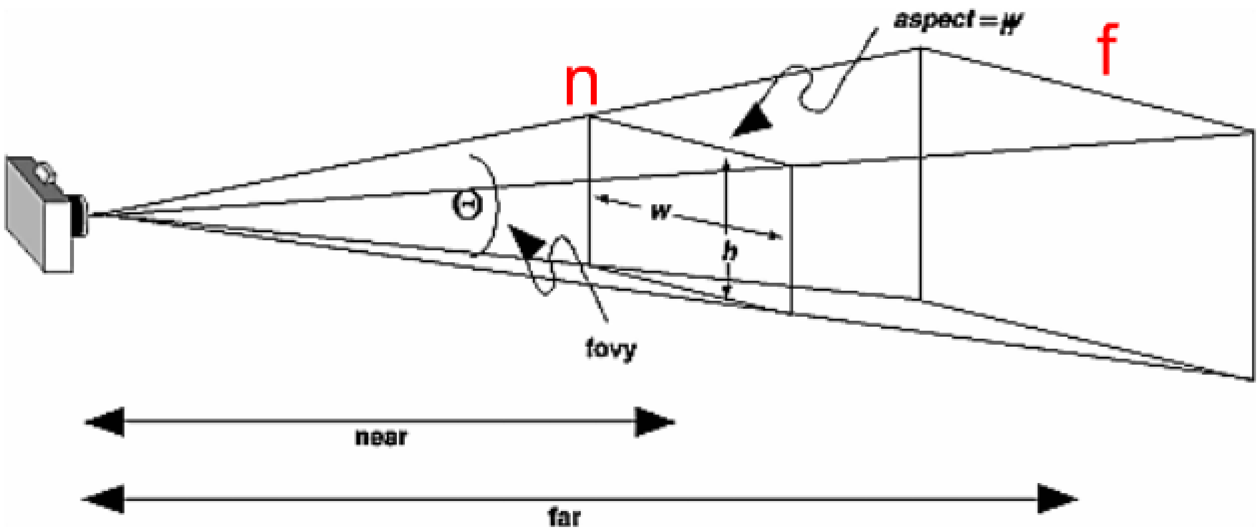
In most of the cases, a set of parameters similar to the one available in a camera is used.

This includes the distances *n* and *f* of the near and far planes, the angle *Θ* at the top of the frustum (known as "*field of view*" *fov-y*) over the y-axis, and the aspect ratio *a* of the screen.



Projection matrix: camera

From the previous definitions we have:



$$t = n \cdot \tan\frac{\Theta}{2}$$

$$b = -n \cdot \tan\frac{\Theta}{2}$$

$$l = -a \cdot n \cdot \tan\frac{\Theta}{2}$$

$$r = a \cdot n \cdot \tan\frac{\Theta}{2}$$

$$r - l = a \cdot (t - b)$$

Projection matrix: camera

Plugging the values *l*, *r*, *t* and *b* in the P_{persp} matrix we obtain:

$$t = n \times \tan\frac{Q}{2}$$

$$b = -n \times \tan\frac{Q}{2}$$

$$l = -a \times n \times \tan\frac{Q}{2}$$

$$r = a \times n \times \tan\frac{Q}{2}$$

$$P_{persp} = \begin{vmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{b-t} & \frac{t+b}{b-t} & 0 \\ 0 & 0 & \frac{f}{n-f} & \frac{nf}{n-f} \\ 0 & 0 & -1 & 0 \end{vmatrix}$$

$$P_{persp} = \begin{vmatrix} \frac{1}{a \cdot \tan\frac{\Theta}{2}} & 0 & 0 & 0 \\ 0 & \frac{-1}{\tan\frac{\Theta}{2}} & 0 & 0 \\ 0 & 0 & \frac{f}{n-f} & \frac{nf}{n-f} \\ 0 & 0 & -1 & 0 \end{vmatrix}$$

Perspective projection matrices in GLM

GLM provides the `frustum()` function to compute the perspective projection matrix specifying the boundaries:

```
glm::mat4 Port = glm::frustum(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.

Perspective projection matrices in GLM

The `perspective()` function computes the perspective projection matrix specifying Fov and aspect ratio:

```
glm::mat4 Port = glm::perspective(fov, a, n, f);
```

Where `fov, a, n, f` are respectively the vertical field of view, the aspect ratio, the near and the far plane distances.

Perspective projection matrices in GLM

As for the `ortho()` function, procedures where created for OpenGL, and need to be wrapped for being used in Vulkan:

```
#define GLM_FORCE_DEPTH_ZERO_TO_ONE
#define GLM_FORCE_RADIANS
#include <glm/glm.hpp>

M2glm = glm::scale(glm::mat4(1.0), glm::vec3(1,-1,1)) *
        glm::frustum(l, r, b, t, n, f);

M1glm = glm::perspective(fovy, a, n, f);
M1glm[1][1] *= -1;
```

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.

Since the Vulkan and OpenGL matrices for Fov/Aspect ratio projection differ only for the sign of the element in the second row / second column, in this special case it could be more convenient to just change the sign of this element instead of applying a mirroring transform.

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)