Teoria i praktyka programowania gier komputerowych

3D projection

Piotr Fulmański

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki, Polska

8 grudnia 2014

Spis treści

View (Camera) Space

Suppose that all the models are positioned and oriented in the world (using different transformation matrix – for the 3D case we will discuss transformation matrix in the next lecture). The next thing to consider is the location of the camera. Thus we need another matrix that tells the graphics card how to transform the models from world space into a coordinate space that is relative to the camera. The matrix most commonly used for this is a look-at matrix . In a look-at matrix, the position of the camera is expressed in addition to the three coordinate axes of the camera.

View (Camera) Space

For a row-major left-handed coordinate system, the look-at matrix is expressed as

$$\begin{bmatrix} L_x & U_x & F_x & 0 \\ L_y & U_y & F_y & 0 \\ L_z & U_z & F_x & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

where L is the left or x-axis, U is the up or y-axis, F is the forward or z-axis, and T is the translation.

View (Camera) Space

To calculate this matrix three inputs are needed:

- the position of the camera (the eye),
- the position of the target the camera is looking at,
- and the up vector of the camera.

View (Camera) Space

```
function CreateLookAt( Vector3 eye, Vector3 target, Vector3 Up )
    Vector3 F = Normalize( target - eye )
    Vector3 L = Normalize(CrossProduct( Up , F ))
    Vector3 U = CrossProduct( F , L )
    Vector3 T

    T.x = -DotProduct( L , eye )
    T.y = -DotProduct( U , eye )
    T.z = -DotProduct( F , eye )

    // Create and return look-at matrix from F, L, U, and T
end
```

Podstawowym przekształceniem w grafice trójwymiarowej jest *rzutowanie*, gdyż komputerowa wizualizacja jakiegokolwiek obiektu przestrzennego wymaga odwzorowania go na płaski ekran monitora. Metody rzutowania możemy podzielić na dwie grupy

- rzuty równoległe (rysunek techniczny; zachowuje równoległość prostych, stosunek długości odcinków równoległych itd),
- rzuty perspektywicze nazywane też rzutami perspektywicznymi (realistyczna wizualizacja obiektów, wrażenie głębi).

Przy rzutowaniu na płaszczyznę Π obrazem punktu p jest punkt p' przecięcia się tej płaszczyzny z prostą (nazywaną *promieniem rzutowania*) przechodzącą przez p. Oczywiście takich prostych jest nieskończenie wiele, ale w przypadku

- rzutu równoległego ustalamy, że wszystkie proste mają ten sam kierunek rzutowania d (jeśli jest on prostopadły do rzutni, to rzut nazywamy *ortogonalnym*);
- rzutu perspektywicznego ustalamy, że wszystkie proste (promienie rzutowania) mają wspólny punkt e (odległość tego punktu od rzutni Π decyduje o „deformacji" rysunku).

In practice znalezienie współrzędnych rzutu p' punktu p sprowadza się do rozwiązania zadania wyznaczenia przeciecia płaszczyzny Π i prostej (promienia rzutowania) określonej punktem p i kierunkiem d przy rzucie równoległym lub punktem p i środkiem rzutowania e w przypadku rzutu perspektywicznego. Zauważmy przy tym, że milcząco założyliśmy, iż

- położenie wszelkich obiektów wyrażone jest w jednym i tym samym układzie współrzędnych: w układzie, w którym został opisany rysowany obiekt (*układ danych*, *world-space coordinates*);
- wszystkie obiekty umiejscowione są na scenie w taki sposób abyśmy nie musieli przejmować się ich widzialnością (tzn. zakładamy, że wszystkie obiekty dają się narysować).

Unfortunately real life is not so simple. Jak już zauważyliśmy, stosując uprzednio opisane podejście otrzymujemy współrzędne punktu p' w układzie trójwymiarowym, w którym został opisany rysowany obiekt (world-space coordinates). My natomiast potrzebujemy współrzędnych w układzie dwuwymiarowym określonym na płaszczyźnie rzutowania Π. Metod wyznaczenia takich współrzędnych jest wiele – w dalszej części omówimy:

- intuitive approach (screen space = view space),
- ostrosłup widzenia (view frustum).

Dodatkową trudnością jest to, że zwykle musimy pracować z więcej niż dwoma układami współrzędnych (zwykle przynajmniej dwa, gdyż mamy układ danych oraz układ związany z punktami płaszczyzny Π).

3D projection

In practice: intuitive approach (screen space = view space): rzut równoległy: szczególny przypadek: rzut prostopadły (ortographic projection)

Najbardziej intuicyjnym rozwiązaniem jest przyjęcie, iż rzutnia Π pokrywa się z płaszczyzną $z = 0$ układu obserwatora. Zalety takiego wyboru układu najlepiej widać przy rzucie ortogonalnym, kiedy rzutem punktu $p = (x, y, z)$ jest punkt $p' = (x, y, 0)$.

3D projection

In practice: intuitive approach (screen space = view space): rzut równoległy: szczególny przypadek: rzut prostopadły (ortographic projection): computer graphics

In computer graphics, one of the most common matrices used for orthographic projection can be defined by a 6-tuple, (left, right, bottom, top, near, far), which defines the clipping planes. These planes form a box with the minimum corner at (left, bottom, -near) and the maximum corner at (right, top, -far).
The box is translated so that its center is at the origin, then it is scaled to the unit cube which is defined by having a minimum corner at (-1,-1,-1) and a maximum corner at (1,1,1).
In this case the orthographic transform can be given by the following matrix:

$$\begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & \frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D projection

In practice: intuitive approach (screen space = view space): rzut równoległy: szczególny przypadek: rzut prostopadły (ortographic projection): computer graphics

In [Mad, 2014] on page 75 you can find orthographic projection matrix defined as

$$\begin{bmatrix} \frac{2}{width} & 0 & 0 & 0 \\ 0 & \frac{2}{height} & 0 & 0 \\ 0 & 0 & \frac{1}{far-near} & 0 \\ 0 & 0 & \frac{near}{near-far} & 1 \end{bmatrix}$$

With this matric the box is translated so that its center is ..., then it is scaled to the unit cube which is defined by having a minimum corner at (...) and a maximum corner at (...).

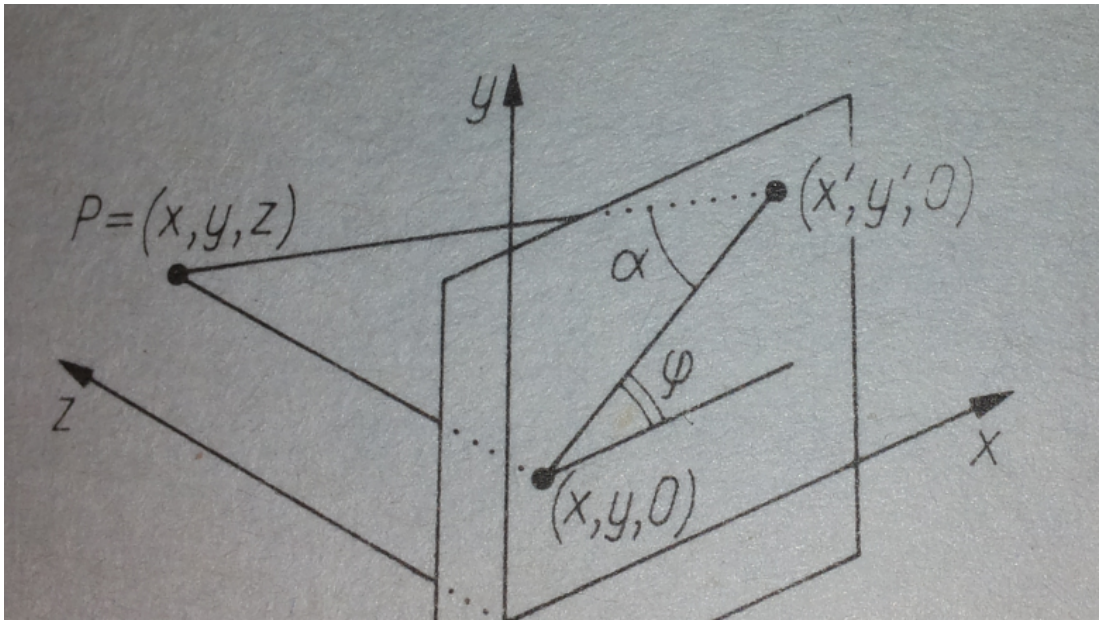
3D projection

In practice: intuitive approach (screen space = view space): rzut równoległy

Uogólniając rzut ortogonalny na dowolny rzut równoległy w którym kierunek rzutu równoległego tworzy z rzutnią Π kat α otrzymujemy następujace współrzędne x' i y' rzutu punktu $p = (x, y, z)$

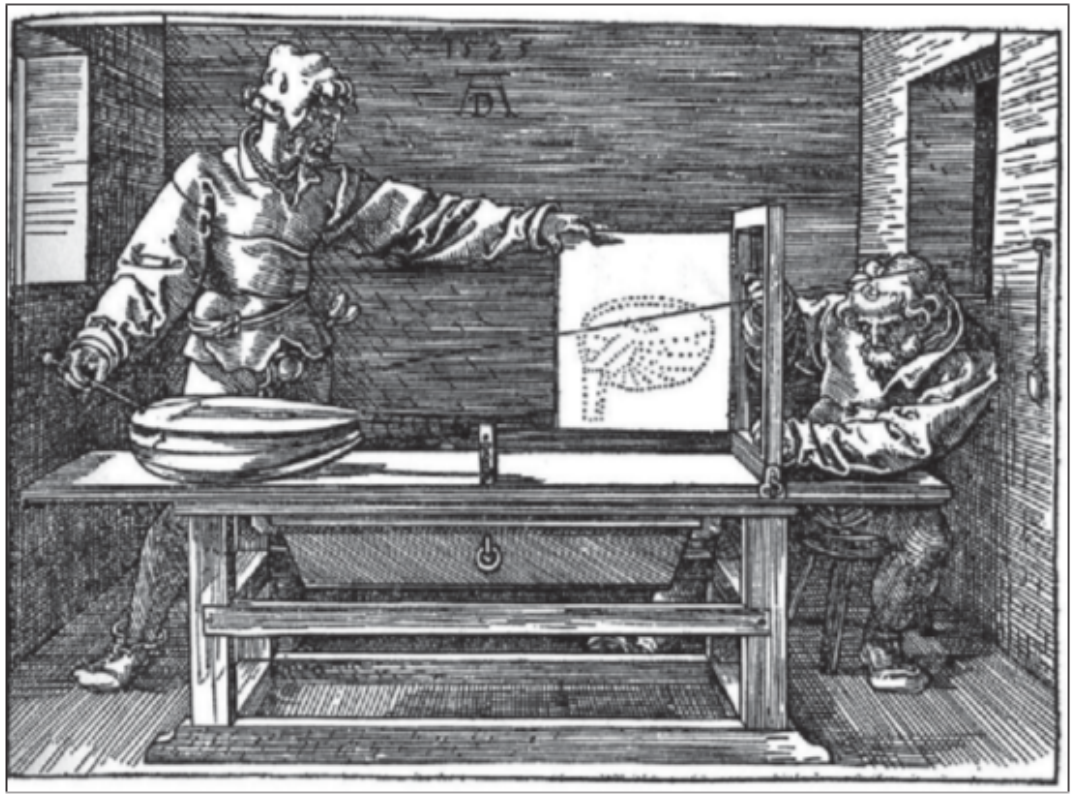
$$\begin{aligned} x' &= x + r\cos\varphi \\ y' &= y + r\sin\varphi \end{aligned}$$

gdzie $r = x\,\mathrm{ctg}\,\alpha$ a φ jest kątem nachylenia prostej przechodzącej przez $(x, y, 0)$ i $(x', y', 0)$ do osi x. Uzyskanie wartości dla α i φ nie jest trudne, gdyż zwykle kierunek d wyraża się właśnie przez podanie tych kątów a nie odpowiednich składowych tworzących wektor d.



3D projection: A Dürer Woodcut

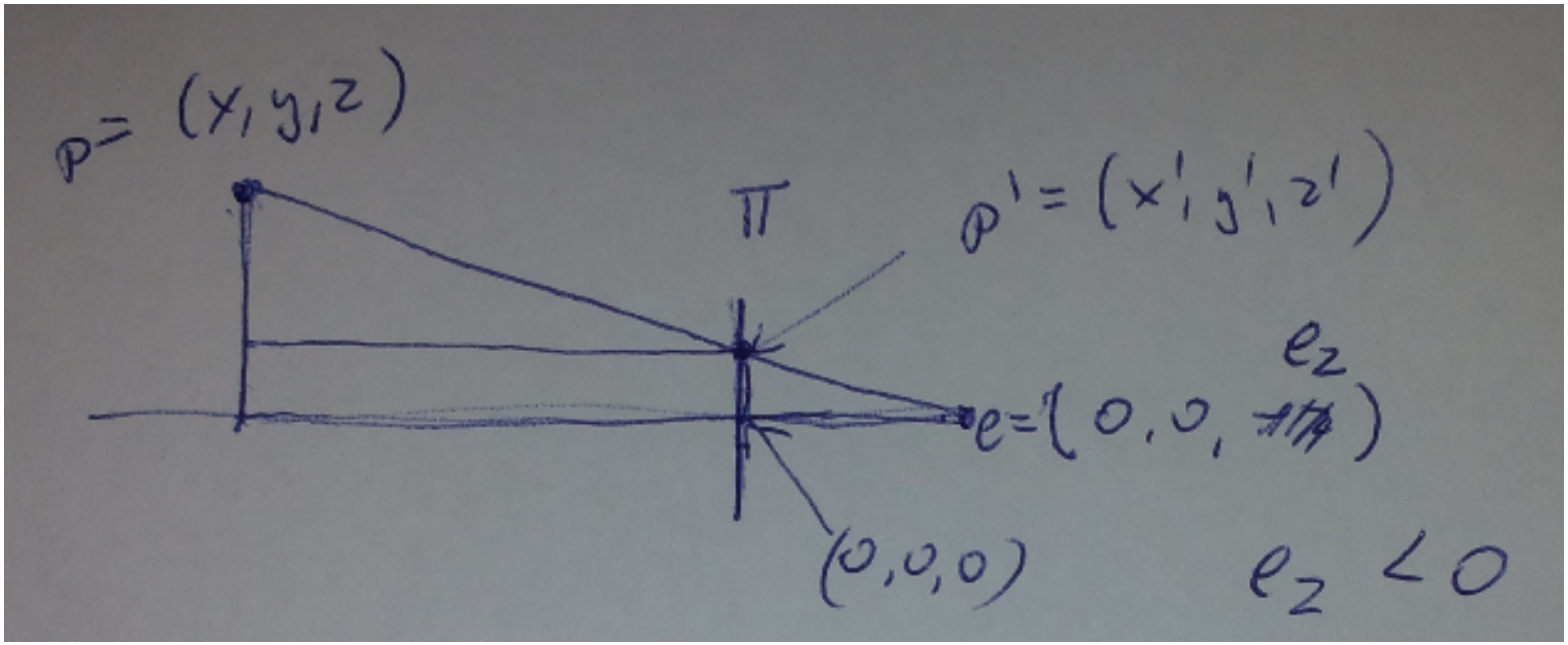
Intuitive approach (screen space = view space): rzut perspektywiczny



3D projection

In practice: intuitive approach (screen space = view space): rzut perspektywiczny

Utożsamiamy obserwatora ze środkiem rzutowania e, który znajduje się na ujemnej części układu w punkcie $(0, 0, -e_z)$ (a więc korzystamy z lewoskrętnego układu współrzędnych, left-handed coordinate system).



Korzystając jedynie z własności podobieństwa trójkątów bez kłopotu znajdujemy zależności pomiędzy p i p'

$$\frac{y}{z + e_z} = \frac{y'}{e_z}$$

$$y' = y \frac{e_z}{z + e_z}$$

gdzie $\frac{e_z}{z+e_z}$ nazywane jest skalą podobieństwa. Oznaczając ją przez s otrzymujemy ostatecznie

$$\begin{aligned} x' &= xs, \\ y' &= ys, \end{aligned}$$

In 3D computer graphics, the viewing frustum or view frustum is the region of space in the modeled world that may appear on the screen; it is the field of view of the notional camera. The exact shape of this region varies depending on what kind of camera lens is being simulated, but typically it is a frustum of a rectangular pyramid (hence the name). The planes that cut the frustum perpendicular to the viewing direction are called the near plane and the far plane. Objects closer to the camera than the near plane or beyond the far plane are not drawn. Often, the far plane is placed infinitely far away from the camera so all objects within the frustum are drawn regardless of their distance from the camera.

W tym podejściu środek rzutowania e pokrywa się z wierzchołkiem ostrosłupa widzenia a w konsekwencji screen space (czyli rzutnia Π) nie jest tym samym co układ obserwatora (view space). Zadania stawiane przed nami w tym podejściu to:

- wyznaczenie wierzchołków lub płaszczyzn ograniczających ostrosłu widzenia;
- rozstrzyaganie czy punkt p leży wewnątrz ostrosłupa widzenia (w szczególności interesuje nas współrzędna z – jej znajomość pozwoli nam wyznaczać obiekty zasłaniane – o której nie mieliśmy żadnej informacji w poprzednim podejściu);
- poszukiwanie współrzędnych rzutu perspektywicznego p' punktu p w układzie ekranu (screen space) a więc na rzutni Π.

Before we start calculation notice that the view frustum is defined by two sets of parameters

- Set of view parameters:
  - the camera position,
  - the look and up vector.

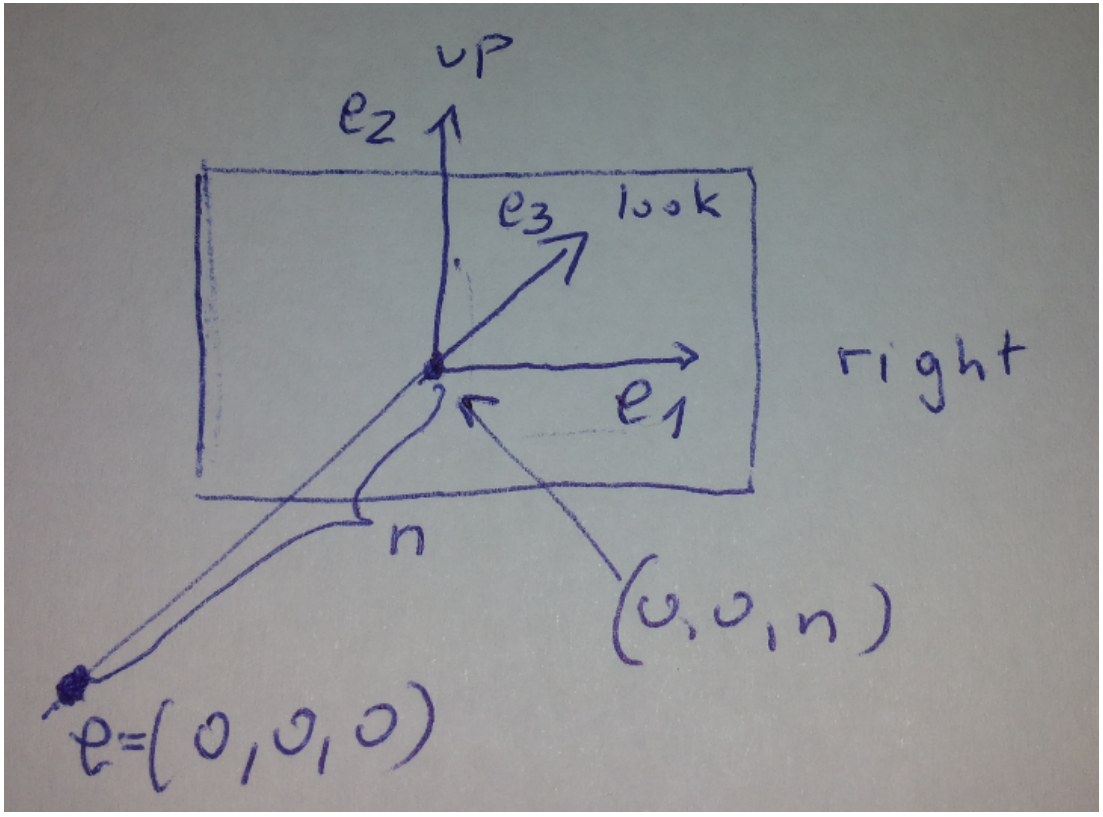
  The up vector determines the tilt of the scene. These parameters determine the general direction and orientation for the view frustum.
- Set of projection parameters:
  - The field of view (fov) parameter determines how much of the scene will be visible.
  - The near plane and far plane parameters are distances that define clipping planes.
  - The aspect ratio parameter defines the ratio between the width and height of the view volume (width is defined in the direction of the right vector, height is defined in the direction of the up vector).

  These parameters determine the shape for the view frustum.

View frustum

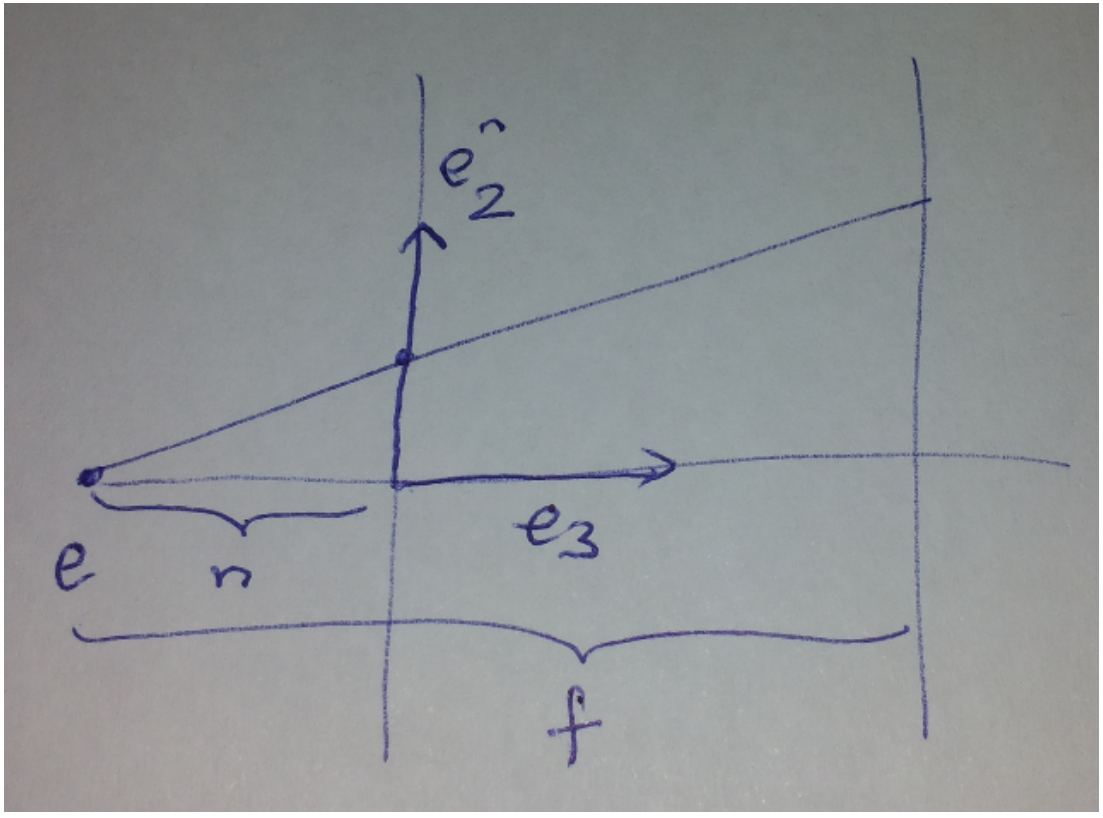
Height of view frustum

Assume that view space is defined by unit vectors $e_1 = (1, 0, 0)$, $e_2 = (0, 1, 0)$ and $e_3 = (0, 0, 1)$ coresponding to right, up and look vector respectively. Then it is very easy to calculate the centres of the near and far clip plane. Denote by e top of the frustum – typicaly $e = (0, 0, 0)$, n – distance to near clipping plane, f distance to far clipping plane.



View frustum

Height of view frustum



With the above we have

$$n_c = e + ne_3 = (0,0,0) + n(0,0,1) = (0,0,n)$$

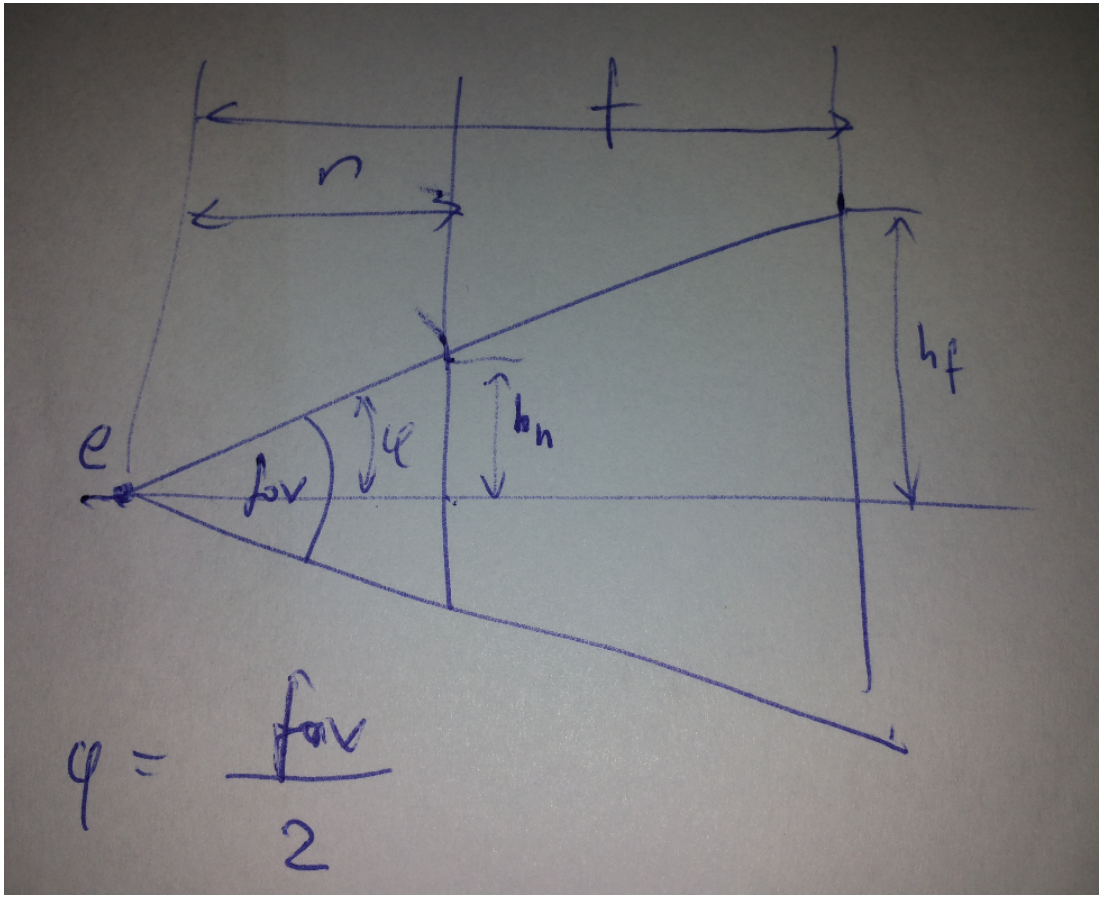
Finally

$$\begin{aligned} n_c &= e + ne_3, \\ f_c &= e + fe_3, \end{aligned}$$

View frustum

Height of view frustum

We can now use centre points as base points to calculate the corners of the near and far clip plane.



The half height of the near and far plane are given by

$$\begin{aligned} h_n &= n\,\mathrm{tg}(\varphi), \\ h_f &= f\,\mathrm{tg}(\varphi), \end{aligned}$$

where φ is a half the field of view in radians.

For the width of the near and far plane we need to take the aspect ratio into account

$$\begin{aligned} w_n &= h_n a, \\ w_f &= h_f a, \end{aligned}$$

where a is a given aspect ration (which sholud fulfil the following equation $a = \frac{w_n}{h_n} = \frac{w_f}{h_f}$).

View frustum

Coordinates of the 8 points that define the view frustum volume

$$
\begin{aligned}
p_{nrb} &= n_c - h_n e_2 + w_n e_1, \\
p_{nrt} &= n_c + h_n e_2 + w_n e_1, \\
p_{nlt} &= n_c + h_n e_2 - w_n e_1, \\
p_{nlb} &= n_c - h_n e_2 - w_n e_1, \\
\\
p_{frb} &= f_c - h_f e_2 + w_f e_1, \\
p_{frt} &= f_c + h_f e_2 + w_f e_1, \\
p_{flt} &= f_c + h_f e_2 - w_f e_1, \\
p_{flb} &= f_c - h_f e_2 - w_f e_1.
\end{aligned}
$$

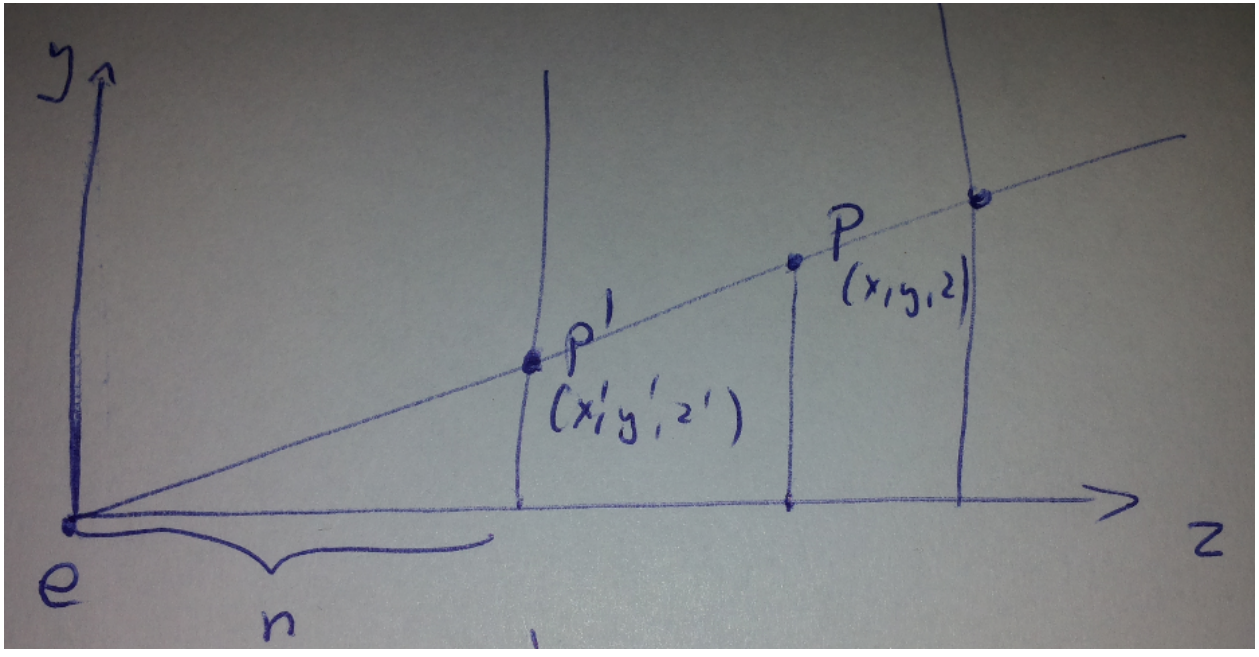
View frustum

Homogeneous clip space. Transformation of x and y

Now we are going to transform a vertex in the view frustum to a homogeneous coordinate in the unit cube. Assume that the view frustum is defined as a pyramid starting at point $e = (0, 0, 0)$ and left-handed coordinate system is used.
We start with the formula for the x and y value. The projected x and y coordinates can be found by projecting the x and y values of the vertices on the near plane and dividing by half the near plane width and height (see figure and compare with previous part of this lecture).

View frustum

Homogeneous clip space. Transformation of x and y



As we have seen before, the formula for the y' coordinate is easy to derive by application of congruent triangles

$$\frac{y'}{n} = \frac{y}{z},$$

where n is the length of n_c vector. In consequence

$$y' = \frac{yn}{z}.$$

Now have a y' value in the range $[-h_n, +h_n]$ for vertices inside the view frustum. For a correct DirectX or OpenGL projection transform we need to calculate a y' value in the range of $[-1, +1]$. To do this, we just need to divide the y' value with half of the near height. Because the half height of the near plane, previously calculated, is given by

$$h_n = n\,\mathrm{tg}(\varphi),$$

then

$$y'' = \frac{yn}{zn\,\mathrm{tg}(\varphi)},$$

and finally

$$y'' = \frac{y}{z\,\mathrm{tg}(\varphi)}$$

The final y'' coordinate is inside the $[-1, +1]$ range for vertices inside the view frustum.

The x coordinate is calculated in exactly the same way as y, but we need to take the aspect ratio of the view frustum into account

$$\frac{x'}{n} = \frac{x}{z},$$

where n is the length of n_c vector. In consequence

$$x' = \frac{xn}{z}.$$

Again, for vertices inside the view frustum, x' is in the range $[-w_n, +w_n]$ and this range needs to be converted to the range $[-1, +1]$. The aspect ratio for a perspective transform which defines the width of the near plane relative to the height of the near plane is equal to

$$a = \frac{w_n}{h_n}.$$

so we have

$$w_n = ah_n,$$

$$w_n = an\,\mathrm{tg}(\varphi).$$

In consequence we have

$$x'' = \frac{xn}{zan\,\mathrm{tg}(\varphi)},$$

and finally In consequence we have

$$x'' = \frac{x}{za\,\mathrm{tg}(\varphi)}.$$

Again for vertices inside the view frustum, x'' is inside the range $[-1, +1]$.

As always we want to apply the formulas via a 4x4 matrix. The following matrix is only valid for the transformation of the x and y coordinates (matrix is in row major order)

$$[x, y, z, 1] \begin{bmatrix} \frac{1}{a\,\mathrm{tg}(\varphi)} & 0 & 0 & 0 \\ 0 & \frac{1}{\mathrm{tg}(\varphi)} & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[x', y', z', w'] = \left[\frac{x}{a\,\mathrm{tg}(\varphi)}, \frac{y}{\mathrm{tg}(\varphi)}, z, z \right].$$

If we divide this homogeneous coordinate by the w' coordinate to convert to a normal 3D coordinate, we get the following result

$$\left[\frac{x'}{w'}, \frac{y'}{w'}, \frac{z'}{w'}\right] = \left[\frac{x}{aw' \operatorname{tg}(\varphi)}, \frac{y}{w' \operatorname{tg}(\varphi)}, \frac{w'}{w'}\right].$$

$$\left[\frac{x'}{w'}, \frac{y'}{w'}, \frac{z'}{w'}\right] = \left[\frac{x}{az \operatorname{tg}(\varphi)}, \frac{y}{z \operatorname{tg}(\varphi)}, 1\right].$$

The x and y coordinate are correct, however the z coordinate is messed up, it is always one. So we have to change something in the perspective matrix to create a valid z value.

The DirectX and OpenGL pipeline requires that the z value after the perspective transform is in the range [0,1] (or [-1,1] for OpenGL). The z value before previously discussed transform is in the range $[near, far]$. We start with a simple range conversion that will transform the $[n, f]$ range to the range $[0, 1]$.
Because the shift $z - n$ transform the $[n, f]$ range to the range $[0, f - n]$, so

$\frac{z-n}{f-n}$ transform the $[n, f]$ range to the range $[0, 1]$.

To fit this conversion into the transformation matrix we have to split it into 2 terms, a factor to scale z with and a factor to shift

$$\frac{z}{f-n} + \frac{-n}{f-n}.$$

Taking both factors, our matrix takes the form

$$[x, y, z, 1] \begin{bmatrix} \frac{1}{a\,\mathrm{tg}(\varphi)} & 0 & 0 & 0 \\ 0 & \frac{1}{\mathrm{tg}(\varphi)} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 1 \\ 0 & 0 & \frac{-n}{f-n} & 0 \end{bmatrix}$$

$$[x', y', z', w'] = \left[\frac{x}{a\,\mathrm{tg}(\varphi)}, \frac{y}{\mathrm{tg}(\varphi)}, \frac{z-n}{f-n}, z \right].$$

Unfortunately, this is still not correct. When we convert the homogeneous coordinate back to a normal 3D coordinate we get

$$\left[\frac{x'}{w'}, \frac{y'}{w'}, \frac{z'}{w'} \right] = \left[\frac{x}{az\,\mathrm{tg}(\varphi)}, \frac{y}{z\,\mathrm{tg}(\varphi)}, \frac{z-n}{(f-n)z} \right].$$

Notice, that the homogeneous conversion which results the form

$$\left[\frac{x'}{w'}, \frac{y'}{w'}, \frac{z'}{w'}\right] = \left[\frac{x}{az\,\mathrm{tg}(\varphi)}, \frac{y}{z\,\mathrm{tg}(\varphi)}, \frac{z-n}{(f-n)z}\right]$$

reintroduces undesirable term $(\frac{1}{z})$ into the transformed z coordinated, which breaks our conversion. Fortunately there is a quick fix to restore the range for the z values: it is enough multiply the shift factor with the f value (because division by z introduced range $[0, \frac{1}{f}]$)

$$[x, y, z, 1] \begin{bmatrix} \frac{1}{a\,\mathrm{tg}(\varphi)} & 0 & 0 & 0 \\ 0 & \frac{1}{\mathrm{tg}(\varphi)} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 1 \\ 0 & 0 & \frac{-nf}{f-n} & 0 \end{bmatrix}$$

$$[x, y, z, 1] \begin{bmatrix} \frac{1}{a\,\mathrm{tg}(\varphi)} & 0 & 0 & 0 \\ 0 & \frac{1}{\mathrm{tg}(\varphi)} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 1 \\ 0 & 0 & \frac{-nf}{f-n} & 0 \end{bmatrix}$$

$$[x', y', z', w'] = \left[\frac{x}{a\,\mathrm{tg}(\varphi)}, \frac{y}{\mathrm{tg}(\varphi)}, \frac{(z-n)f}{f-n}, z\right]$$

$$\left[\frac{x'}{w'}, \frac{y'}{w'}, \frac{z'}{w'}\right] = \left[\frac{x}{az\,\mathrm{tg}(\varphi)}, \frac{y}{z\,\mathrm{tg}(\varphi)}, \frac{(z-n)f}{(f-n)z}\right]$$

Unfortunately the division by z introduces the non linearity of the z buffer. This non linearity is usually sold as beneficial for the precision of the z buffer close to the camera, however it dramatically reduces the resolution of the z buffer a bit further from the camera. For example, if the near plane is set to 1 and the far plane is set to 100 than when z is 40 the resulting transformed z' is allready at 0.98, leaving only a small amount of resolution to the following 60% of the scene.

Because our transformatio transforms the vertices to the range of $[-1, 1]$ we need to convert the range $[-1, 1]$ to the range $[0, \text{window width}]$ for the x coordinate and to the range $[0, \text{window height}]$ for the y coordinate. The window width and window height are expressed in pixel units. The formula for this conversion is

$$x' = \frac{xw}{2} + \frac{w}{2},$$

$$y' = \frac{yh}{2} + \frac{h}{2}.$$

The same expressed as a matrix

$$[x, y, z, 1] \begin{bmatrix} \frac{w}{2} & 0 & 0 & 0 \\ 0 & \frac{h}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{w}{2} & \frac{h}{2} & 0 & 1 \end{bmatrix}.$$