Transformations and projections

Transformations play an important role in computer graphics: they allow to describe mappings between different coordinate systems, move or scale 3D models, can be used to concisely describe parallel or perspective projections. Here, we focus on transformations performed by graphics hardware in the vertex processing stage. The purpose of such transfromations is to translate the *model coordinates* (in which a 3D object is modelled) into the *world coordinates* ('global' coordinate system describing the whole virtual world we would like to render on the screen) and then to translate the world coordinates into the *screen coordinates* (essentially, the location of the projection onto the screen plus depth information).

Both of the transformations described above can be conveniently described using matrices.

1 2D linear transformations as 2x2 matrices

Linear transformations of the plane can be represented by two by two matrices. Here are a few examples

- Rotation around the origin by the angle of α, $\begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$.
- Scaling by a factor of c_x along the x-axis and by a factor of c_y along the y-axis $\begin{bmatrix} c_x & 0 \\ 0 & c_y \end{bmatrix}$.
- Uniform scaling by a factor c (if $c = 2$ it makes everything two times bigger) $\begin{bmatrix} c & 0 \\ 0 & c \end{bmatrix}$.
- Symmetry about the origin is equivalent to uniform scaling by a factor of -1 $\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$.
- Symmetry about the x-axis scales y by a factor of -1 and leaves the x coordinate unchanged $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.
- Similarly, symmetry about the y-axis has the matrix $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$.
- Shear along the x-axis has the matrix $\begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}$, and shear along y - $\begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix}$ (Figure 1 shows how it works).

A problem with with 2x2 matrices as a representation of 2D transformations is that translations (which are obviously useful to represent e.g. movement of

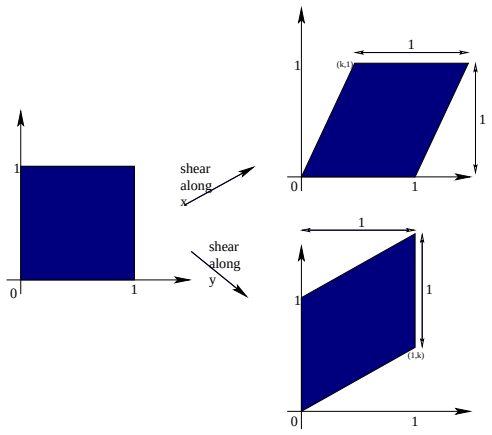

Figure 1: Shear transformations and how they act on a unit square.

objects) do not fit into this framework: there is no way to represent a nontrivial translation as a 2×2 matrix (since any transformation written by a 2×2 matrix maps the origin into itself, unlike a translation by any nonzero vector). Homogenous coordinates are a way around this problem: by adding one extra third coordinate they allow to represent translations as 3x3 matrices. In a moment we'll see that they also allow to represent 2D perspective projections as a 3x3 matrices, which makes them even more useful for graphics.

2 Translations as matrices using homogenous coordinates

We'll think of a 2D point (x, y) as a point in 3D, namely $(x, y, 1)$. This immediately allows to represent a translation by a vector $[T_x, T_y]$ as the 3x3 matrix

$$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}.$$

It maps the point $(x, y, 1)$ (the 3D point corresponding to (x, y)) to $(x + T_x, y + T_y, 1)$, the 3D point corresponding to the translaterd point $(x + T_x, y + T_y)$. All transformations discussed in the previous section can be represented in homogenous coordinates as 3x3 matrices: just complete its 2x2 matrix to a 3x3 one by adding one row and one column like this (replace the stars with the entries of the 2x2 matrix):

$$\begin{bmatrix} * & * & 0 \\ * & * & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3 3D case

All that has been said about the 2D case carries over to the 3D case. A 3D point (x, y, z) in homogenous coordinates has coordinates $(x, y, z, 1)$. Translations are represented as 4×4 matrices

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Rotations around the coordinate axes have matrices

$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Shears have matrices of the form

$$\begin{bmatrix} 1 & \star & \star & 0 \\ \star & 1 & \star & 0 \\ \star & \star & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

one of the stars replaced by a nonzero entry and all other ones - by zeroes.

Describing scalings and symmetries (about origin, coordinate axes and coordinate planes) in the 3D case is left to the reader.

4 Composing transformations given by matrices in homogenous coordinates

Why use matrices in homogenous coordinates? First, they are flexible enough to represent rigid transformations and scalings, which are most commonly used to map model coordinates into world coordinates. And then, the matrix representation allows to build more complex transformations from simple ones by means of composition (equivalent to matrix multiplication).

For example, the matrix of the 2D rotation by angle α around a point (x_0, y_0) can be computed as follows. Notice that the rotation is equivalent to a sequence of three elementary transformations whose matrices we already know:

1. Translation by $(-x_0, -y_0)$ (which takes the center of rotation to the origin)

2. Rotation by α around the origin

3. Inverse of the first transformation, i.e. translation by (x_0, y_0).

The matrix of the composition can be computed as the product of the three components (written right to left):

$$\begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix}.$$

By carrying out the matrix multiplication we can obtain one 3×3 matrix representing the rotation about (x_0, y_0). Let us stress here that the resulting transformation is not among the elementary transformations we listed previously, but it has the same representation: a 3×3 matrix.

5 Projections

There are two types of projections that are commonly used in computer graphics: parallel and perspective. Parallel projection is somewhat simpler to deal with, but it is less natural: most pictures of the 'real world' (like photographs) we are used to are based on perspective projections. However, parallel projections are often used for visualizing more abstract data where realism is not that important.

Below we discuss the two kinds of projections in 2D.

5.1 Parallel projections

A typical approach to projections is to reduce them to a simple case, called the *canonical view*, which we describe first. Later on, we will show how it is possible to reduce any projection (both parallel and perspective) to the canonical view by means of a sequence of transformations.

5.1.1 Canonical view

Canonical view is a special type of parallel projection, depicted in Figure 2. The near and far clipping planes allow to restrict viewing to objects which are not too close and, at the same time, not too far from the screen in the world space. This allows to save time by rejecting objects which are too far (in many cases, e.g. walkthroughs in large buildings such objects are not visible anyway, e.g. are behind walls; or they are barely visible or unimportant because they are too distant from the viewpoint). The clipping planes play an important role in depth resolution (points on the front clipping plane have zero depth, points on the far clipping plane have depth of 1). The front clipping plane also allows to view interesting cross-sections through complex objects by clipping away what's in the front.

The handling of the canonical view is very simple: for a point (x, y) one simply uses its first coordinate as the coordinate of the projected point (on the screen) and the second coordinate as the depth. The two coordinates together form what is often referred to as *screen coordinates*, containing information

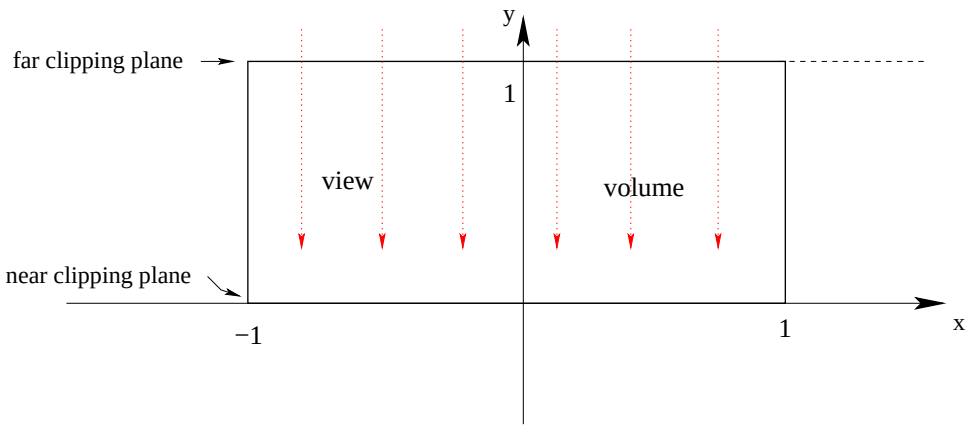

Figure 2: Canonical view. This is a parallel projection, with the screen extending along the x-axis from $(-1, 0)$ to $(1, 0)$. The projection direction is vertically down. The near and far clipping planes (in 2D they are lines, of course) are $y = 0$ (x-axis) and $y = 1$.

about the location of the projected point on the screen and the depth (of course, in 3D we would have two coordinates of the projected point and one depth coordinate). For points within the view volume, this yields depths between 0 and 1 and the coordinates of the projection on the screen - between -1 and 1. Only points having depth between 0 and 1 are rendered, sice all other points are outside the view volume.

5.1.2 General parallel projection can be reduced to canonical view

Any parallel projection can be reduced to the canonical view by a sequence of transformations as shown in Figure 3.

5.2 Perspective projection

5.2.1 A simple case

Perhaps the most interesting property of the homogenous coordinates is that they allow to represent perspective projections as matrices. Figure 4 shows the simplest possible case of a perspective projection we'll work out first.

Notice that it sends the point (x, y) to $(x/y, 1)$. Thus, it requires division, an operation not provided by the matrix formalism. To help us write the projection as a 3x3 matrix we'll use the homogenous coordinate for scaling the first two: we'll think of a point (x, y, t) as $(x/t, y/t)$. With this convention, the perspective

projection can be wtitten as the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

This is a matrix which maps $(x, y, 1)$ ((x, y) in homogenous coordinates) to (x, y, y) which, by our scaling convention, corresponds to $(x/y, 1)$.

To get an even more interesting representation of the perspective projection we can use the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 1 & 0 \end{bmatrix},$$

which represents the transformation mapping (x, y) to $(x/y, (y-1)/y) = (x/y, 1-1/y)$. It can be shown to map lines into lines (why?). The x-coordinate of the result is exactly the x-coordinate of the perspectively projected input point. The y-coordinate can be interpreted as the the depth: as y increases from 0 to infinity, $(y-1)/y$ increases monotonously from minus infinity to 1. Thus, the half-plane $y > 0$ is mapped onto the half-plane $y < 1$. The order of the y-coordinates is preserved by that mapping: if a point p_1 is above (has larger y-coordinate) another point p_2 then the image of p_1 is above the image of p_2. Therefore, something like $(y-1)/y$, can be a good candidate for the depth: it reflects the order of points with respect to the eye and can be computed from vertices by linear interpolation (we'll scale it to be between 0 and 1 for points between the near and far clipping plane before using as the depth value though). The qualitative picture showing how the transformation described above acts on lines through the origin is shown in Figure 5. See Figure 6 for a similar picture showing how it acts on horizontal lines.

Another interesting property of perspective transformation is that it maps families of parallel lines onto families of lines intersecting at one point of the screen as shown in Figure 7. This should not be a surprise, since any parallel lines in the 'real world' appear to come together at the horizon – think about lanes on a straight highway, or look at Figure 8.

5.2.2 An arbitrary perspective projection

The general perspective view setup is shown in Figure 9.

A lot like in the case of parallel projection, the 2D counterpart of the view volume is bounded by four lines (see Fig 9):

- The *far clipping plane*: no object (or part of an object) behind that plane will not be drawn,
- The *near clipping plane*: no object in front of it will be drawn,
- Two lines defining the field of view through the screen.

Of course, in 3D we have the near and far planes and the field of view is bounded by four planes, each passing through the eye and one of the edges of the screen.

6 3D case

As usual, everything that we said about 2D case carries over to 3D. In particular, every view can be reduced to the canonical view using transformations represented using 4×4 matrices in homogenous coordinates. The most general way to describe a 3D view would be through defining the viewpoint, the screen (a parallelogram in 3D) and the distances between front and back clipping planes and the viewpoint. Of course, the front and back clipping planes need to be parallel to the screen and on the same side of the viewpoint (i.e. they should be intersected by any ray stabbing the screen).

7 Accuracy of the z-buffer

The nonlinear transformation of the z-coordinate allows linear interpolation of depth (which is good) but also has some consequences which might not be so desirable. One of them is that the accuracy of depth is not uniform and depends on the locations of the front and back clipping planes. Here is why. Let's examine the 2D case (3D works the same way), with view as described in Section 5.2.1. Let's say that the front and back clipping lines are at $y = y_f$, $y = y_b$ and the y-coordinates of triangles we would like to draw are in the range $[y_{min}, y_{max}]$. The depth values for the triangles' vertices are in the range $I_{obj} = [1 - 1/y_{min}, 1 - 1/y_{max}]$ and the depth range for the entire view volume is $I_{total} = [1 - 1/y_f, 1 - 1/y_b]$. The internal representation of depth uses fixed point numbers rather than floats. The I_{total} interval is scaled to $[0, 1]$ and a fixed number of bits (in current graphi cs cards, this might be 24 or 32 bits) is used to represent the depth value. After the scaling, I_{obj} scales down to an interval of length

$$L = \frac{1/y_{min} - 1/y_{max}}{1/y_f - 1/y_b}.$$

For example, if $y_{max} = 1$ and $y_{min} = 0.5$, this reduces to

$$L = \frac{1}{1/y_f - 1/y_b}.$$

Notice that the magnitude of this expression heavily depends on where the front and back clipping lines are. If the back clipping plane is $y = 0.001$, then L has to be smaller than $0.001 \approx 2^{-10}$, which practically means losing ten bits of accuracy in depth (since 10 most significant bits of the vertices are the same for most (or even all) vertices that are to be processed). Increasing y_b will also increase L, but, in a way, its effect would not be as bad: no matter how big y_b is, L is less than y_f.

To sum up: for most efficient use of z-buffer bits, enclose the objects in the scene between front and back clipping planes as tightly as possible. Most importantly, don't put the front clipping plane too close to the origin, even if this might be a tempting way to ensure that everything (even objects very close to the viewpoint) shows up on the screen.

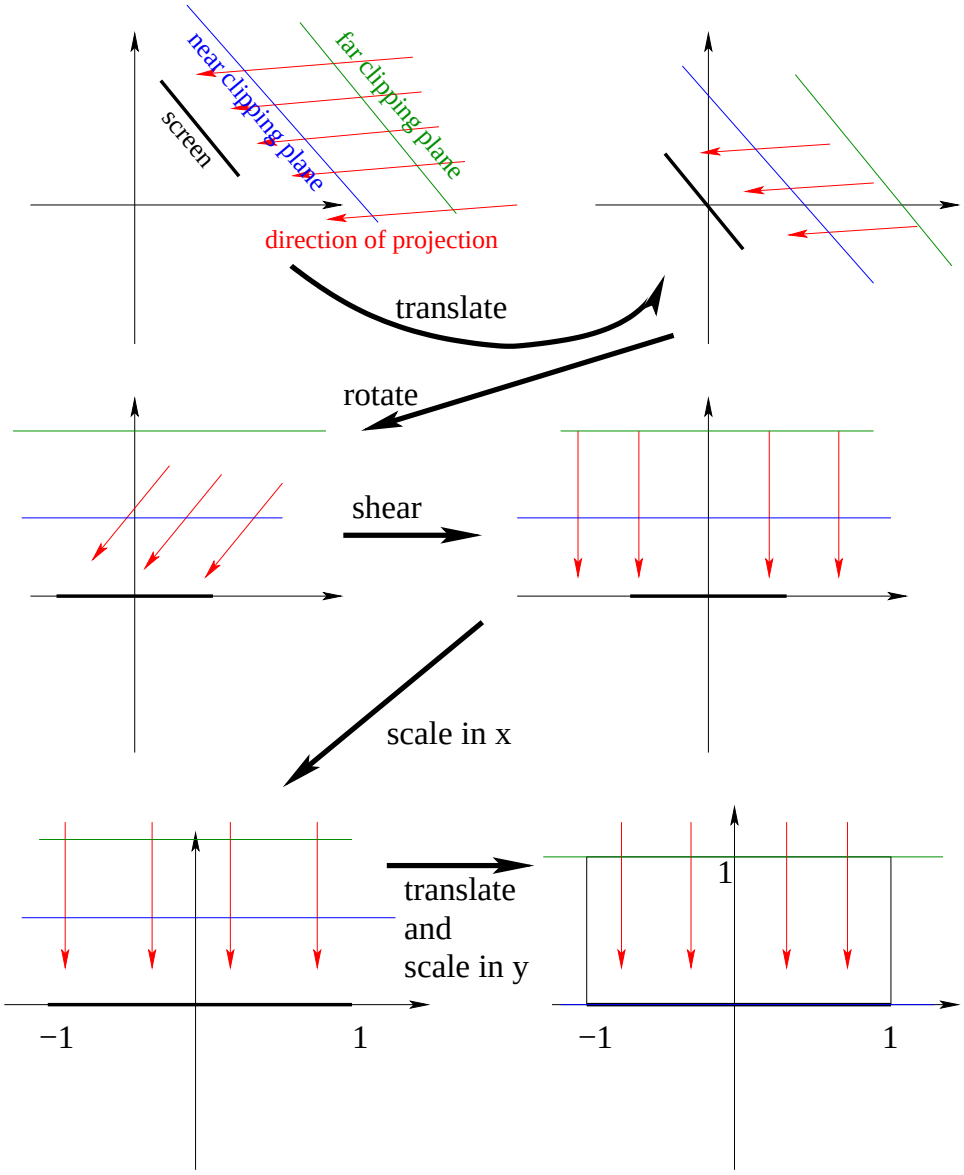

Figure 3: The process of reducing any parallel projection to a canonical view. First, we apply translation which takes the center of the screen to the origin. Then, we rotate to make the screen overlap with the x-axis. A shear transformation makes the projection direction vertical down. Then, scaling makes the screen extend from -1 to 1. Finally, translation and scaling along y takes the near and far clipping planes to $y = 0$ and $y = 1$.

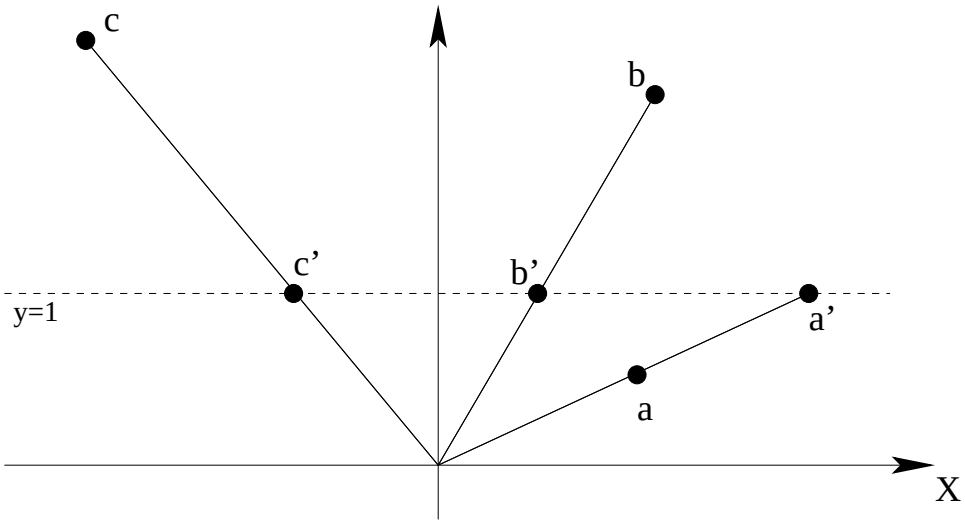

Figure 4: A perspective projection; the center of the projection (think of it as the location of the eye) is the origin; the 'screen' onto which points are projected is the dashed line described by $y = 1$.

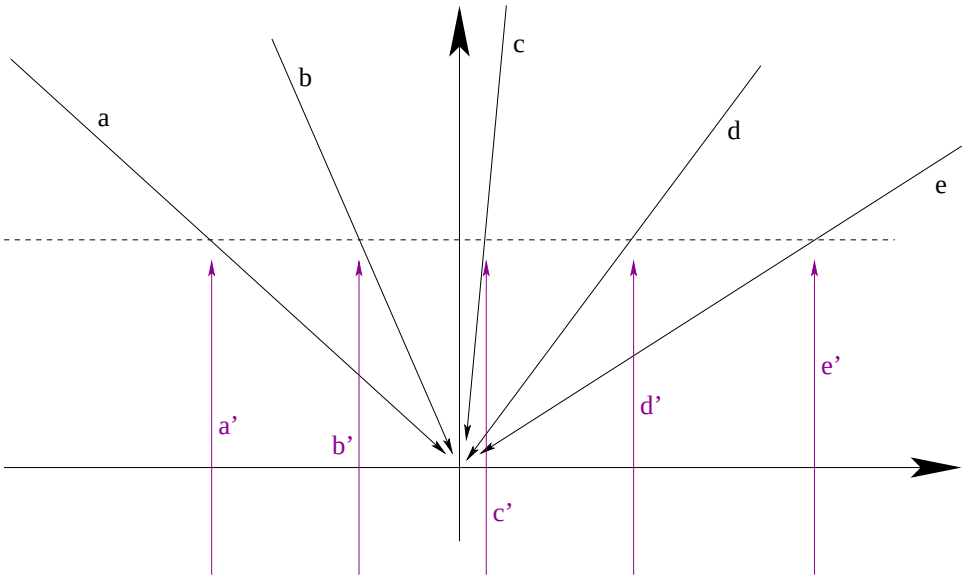

Figure 5: The black rays that approach the origin are mapped into the magenta colored vertical rays. Note that a magenta ray converges to the intersection of the screen and the corresponding black ray. This is the point on the screen that all points on the black ray project to.

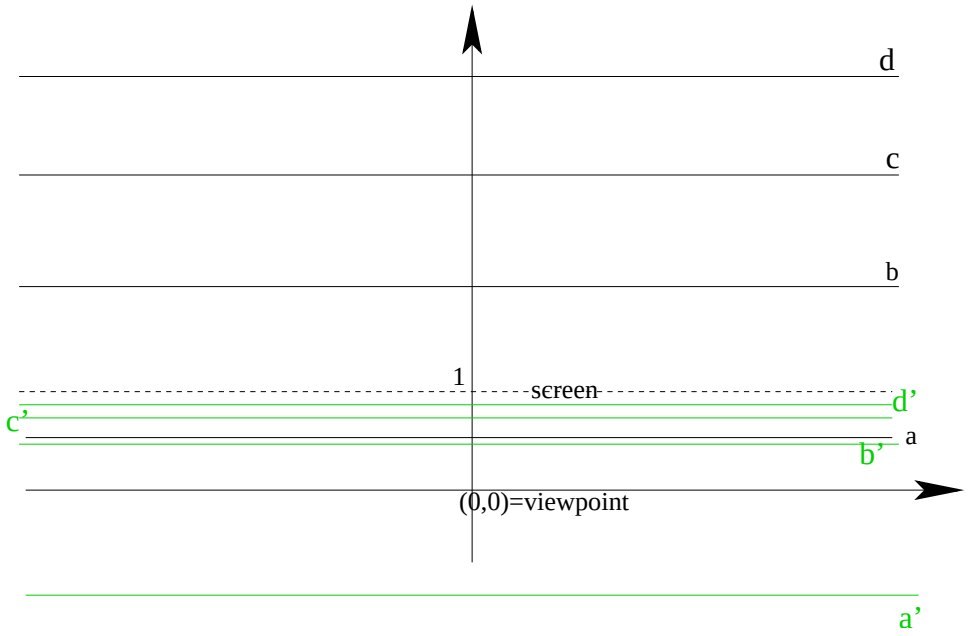

Figure 6: The black horizontal lines are mapped to the corresponding green horizontal lines. Imagine that you put the black line on the x-axis and move it up with constant speed (black arrow on the left). Then, the image of the black line (which is also a horizontal line) is going to move up with decreasing speed. The green lines are going to converge to the screen (line $y = 1$).

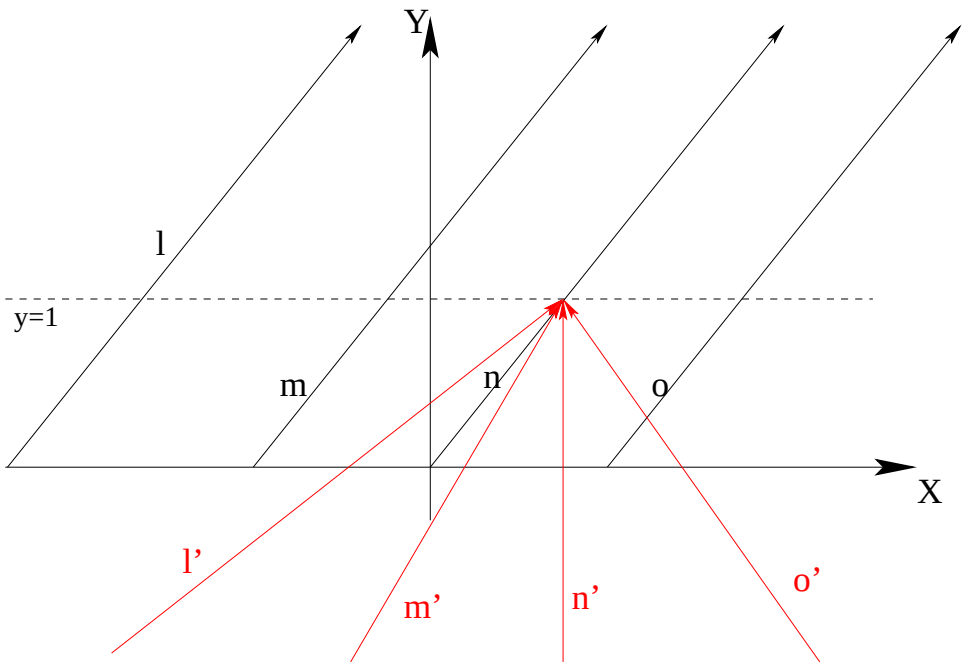

Figure 7: The black lines (in a family of parallel lines) are mapped into the red lines, all of which meet at one point on the screen. This is the point where the black line n passing through the eye intersects the screen. Notice that the entire line n projects to a single point – thus the x-coordinate of the projection is constant and the corresponding line n' is vertical.

Figure 8: Projections of parallel (here: vertical) lines intersect at one point.

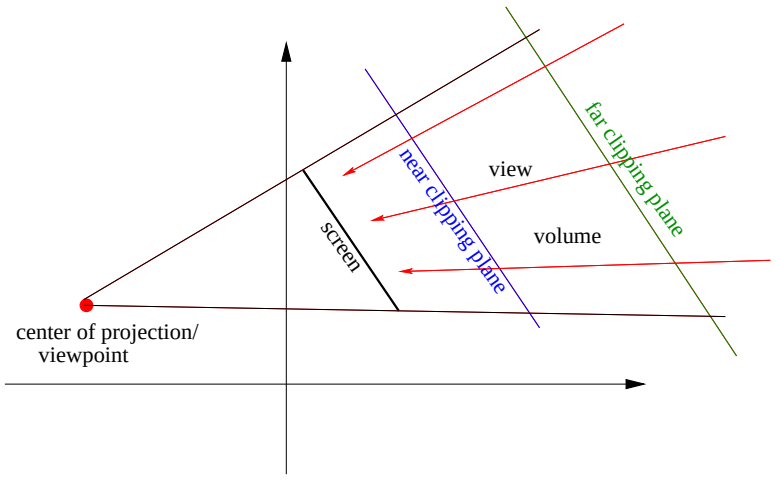

Figure 9: View volume: the red lines bound the field of view through the screen.

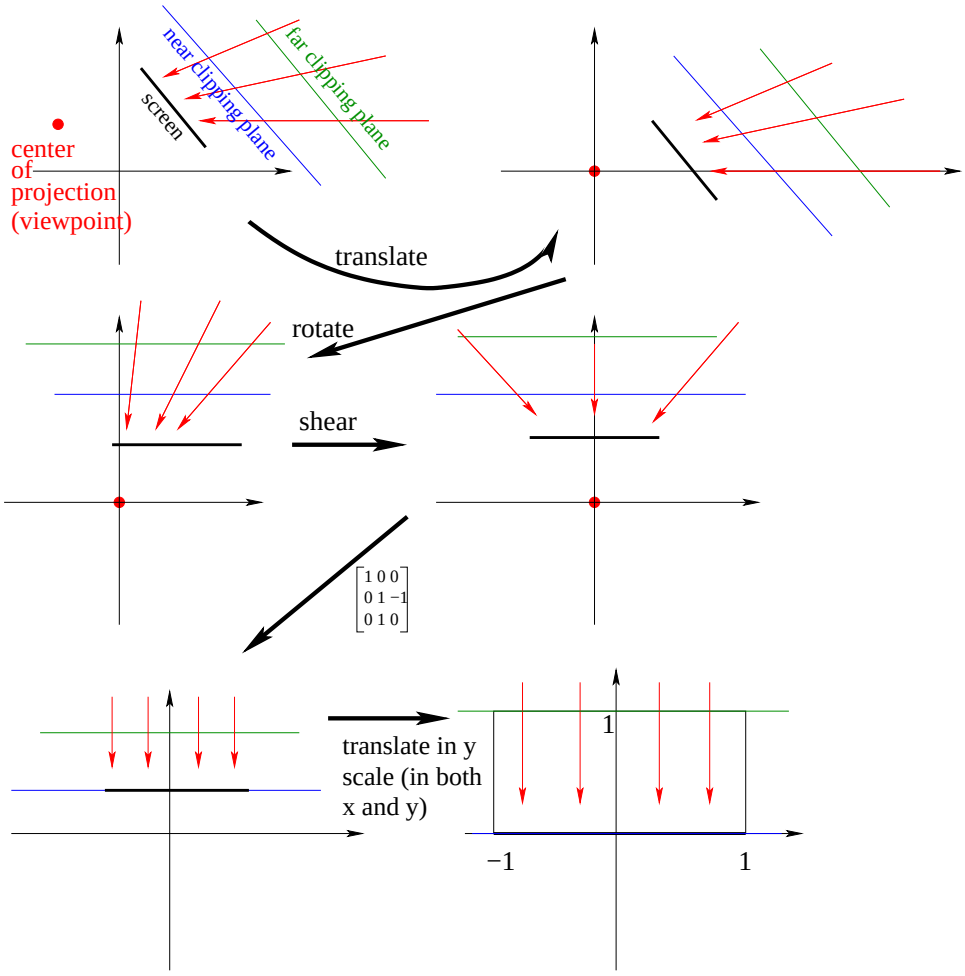

Figure 10: The process of reducing any perspective projection to the canonical view. First, we apply translation which takes the center of projection (viewpoint) to the origin. Then, we rotate to make the screen parallel to the x-axis. A shear transformation takes the center of the screen to the y-axis. Applying perspective projection matrix turns the perspective projection into a parallel projection, but it reverses the projection direction. Finally, translation in y and scaling along y (by a negative number) takes the near and far clipping planes to $y = 0$ and $y = 1$. Additional scaling along the x axis causes the screen to extend from -1 to 1.