The Viewing Transformation

Technical Memo No. 84

Alvy Ray Smith

Pixar
P. O. Box 13719
San Rafael, CA 94913

June 24, 1983
(Revised May 4, 1984)

This document was reentered by Alvy Ray Smith in Microsoft Word on Apr 22, 1999. Spelling and punctuation are generally preserved, but trivially minor spelling errors are corrected. Otherwise additions or changes made to the original are noted inside square brackets or in footnotes.

Introduction

The *viewing transformation* is the operation that maps a perspective view of an object in world coordinates into a physical device's display space. In general, this is a complex operation which is best grasped intellectually by the typical computer graphics technique of dividing the operation into a concatenation of simpler operations.

The terms world, model, and eye space will refer to coordinate systems in object space coordinates. *World space* refers to the 3-D universe in which objects are embedded. *Model space* refers to a coordinate system embedded in world space but with its origin attached to a particular model. *Eye space* is similar but has its origin at the eyepoint from which a view is desired. I will tend to use world, model, and object space interchangeably.

The terms device, screen, and display space will be used interchangeably for coordinate systems in image space coordinates. *Screen space* refers to a coordinate system attached to a display device with the *xy* plane coincident with the display surface. It can be thought of as a 2-D space with a third dimension carried along for auxiliary purposes (e.g., depth cueing).

Orthographic projection will be mentioned from time to time, but the memo concentrates on perspective projection. Other projections such as elevation oblique or plan oblique will not be treated.

We assume that we are to view a scene of objects in object space, or world coordinates, and that it is to displayed in some image space, or device coordinates. The coarsest subdivision of the transformation task is

1. Clip the scene against the viewing frustum for the desired perspective.
2. Transform all remaining points into the desired image space.

This has the drawback that clipping against an arbitrary frustum is difficult.

Reversing the order of the two steps is another possibility. It has the flaw that all points must be transformed whether they are ultimately visible or not. But, more importantly, it has the *serious* flaw that it requires clipping after perspective. As explained in Appendix A, there are two ways of applying perspective. In one of them, the "serious" flaw is a *fatal* flaw because points behind the eye may be mapped into the visible space by the perspective step.

So it is typical to divide the problem into three major steps:

3. Transform the scene into a normalized set of viewing coordinates for which clipping is easy.
4. Clip in this normalized space.
5. Complete the transformation of remaining points, including the perspective projection.

In the case where the database is huge, as we expect those in Computer Imagery to be, we want the transformation of step (1) to be as simple as possible since all points pass through it. The ease of clipping in normalized space must offset the cost of this transformation else it will not get used. Alternatively, simple culling procedures might be invoked to cut down the database before invoking the first transformation above.

Figure 1 from [FOLVAN] illustrates the process we have conventionalized. This process assumes three kinds of transformations:

1. Modeling transformations for manipulating a model's database in world coordinates. Presumably these would be applied in some type of modeling program, or model editor. The results would be input to the left side of the Figure 1.
2. Viewing transformations for manipulating a simulated camera's view of a model. These are mainly what we are talking about here. They include clipping, perspective projection, and mapping to display space.
3. Image transformations move an already clipped object about in device space.

Here we shall not be particularly concerned with modeling transformations. Their results will be changed, as indicated in the leftmost box of the Figure 1, into a normalized space as a first step for viewing. Our only concern with image transformations will be that we take them into account and leave hooks for their invocation.

The approach here will be to present a convention for specifying a view then follow with a convention for transforming a conventional view. The conventional view specification is the more important as far as conventions are concerned. The other part is a justification for and a guide to the use of the convention.

Conventional View Spec

The *conventional view* is that defined by the following conventional structure:

```
typedef double WorldType;
typedef WorldPoint { WorldType x, y, z; };
typedef ViewPlanePoint { WorldType u, v; };
#define PERSPECTIVE 1
#define ORTHOGRAPHIC 2

struct ViewStructure {
    WorldPoint ViewPoint;          /* center of projection point */
    WorldPoint ViewNormal;         /* view plane normal vector */
    WorldPoint ViewUp;             /* view up vector */

    WorldType ViewDistance;        /* from ViewPoint */
    WorldType NearDistance;        /* from ViewPoint */
    WorldType FarDistance;         /* from ViewPoint; 0 means infinite*/

    ViewPlanePoint WindowCenter;   /* relative view plane center */
    ViewPlanePoint WindowHalfsize; /* window u, v halfsizes */

    int ProjectionType;            /* PERSPECTIVE or ORTHOGRAPHIC */
};
```

This differs from [MCHCAR] only in sticking with ViewUp rather than changing to ViewRight as advocated there (for no particular advantage that I can see for perspective or orthographic projections). I have also dropped their parameters *G* and *F* which are not needed for perspective or orthographic projections.

To complete the spec, a normalized set of device coordinates needs to be specified. Let the conventional *normalized device coordinates (NDC)* be

$$-1. \le x \le 1.,$$
$$-1. \le y \le 1.,$$
$$0. \le z \le 1.,$$

with *x* left, *y* up, and *z* in—i.e., a left-handed system. This is a departure from the Core standard [CORE79] which defines NDC as *x*, *y*, and *z* each on [0., 1.]. Our choice provides a cleaner presentation, and an extremely simple transformation maps our NDC to theirs.

Finally, let the conventional normalized clipping region for perspective projection be as shown in the Figures 2 and 3 (from [MCHCAR]). The frustum of Figure 2 between “Front” and “Back” is the *canonical viewing volume (frustum)*. Notice that I have chosen to use “near” and “far” rather than the “front” and “back” of [FOLVAN], [CORE79], and [MCHCAR].

The most important part of this spec is the struct ViewStructure. This is the structure that is passed from program to program. It is the structure that is as-

sumed kept up-to-date by any program changing the view. Since a far distance of 0 is disallowed, this value is used to denote the frequently desired case of the far plane at infinity. Notice that all numbers in ViewStructure are in world space.

We ahve not opted to specify ViewStructure with such traditional parameters as aspect ratio and field-of-view angle because field-of-view angle, as usually defined, is meaningless for off-center windows. But since the traditional parameters will get used, they are conventionalized as follows:

1. *Aspect ratio* will be the ratio of width to height. It is therefore equal to WindowHalfsize.u / WindowHalfsize.v.
2. *Field-of-view angle* will be the full horizontal angle of the viewing frustum *for a centered window*. Thus it is equal to 2*arctan(WindowHalfsize.u / ViewDistance).

Notice that the field-of-view angle by this definition does not change as the viewing window translates across the view plane.

Elaboration of the View Spec

The notion is straightforward: Specify the location of the ViewPoint, or eyepoint, in world space. Then specify which direction you're looking with ViewNormal. Orient the view by specifying ViewUp. The component of ViewUp perpendicular to ViewNormal gives the "up" direction. Finally, specify where the view plane is to be along the direction ViewNormal from ViewPoint—i.e., specify ViewDistance. The perspective projection of objects onto this view plane is what will be mapped into a display.

All that remains to do is to specify what part of the infinite view plane is to be mapped to the display. Since displays tend to be rectangular, it is typical to specify a (rectangular) window in the view plane as the portion of interest. In our convention, this will be done by utilizing a *uv*-coordinate system in the view plane with the origin being where a ray from the ViewPoint in the direction ViewNormal intersects the view plane. Let **n** be the unit normal in direction ViewNormal, and let **v** be the unit normal in the up direction. Thus

$$\mathbf{v} = \frac{\mathbf{UP} - (\mathbf{UP}\cdot\mathbf{n})\mathbf{n}}{\left|\mathbf{UP} - (\mathbf{UP}\cdot\mathbf{n})\mathbf{n}\right|},$$

where **UP** is an abbreviation for vector ViewUp. So *v* is the coordinate along **v** and *u* along **u**, where

$$\mathbf{u} = \mathbf{n} \times \mathbf{v}.$$

Similarly, *n* is the coordinate along **n**. Notice that the *uvn* system is a left-handed one. Figures 4-7 from [FOLVAN] illustrate this arrangement. In these figures, VRP is our *uv* origin, VPN is our ViewNormal, and VUP is our ViewUp. Notice that the window may be off center.

The main difference between the Core standard and our convention is the method of specifying the view plane. In the Core standard, the so-called "view reference point" (cf. VRP in Figures 4-7) is specified explicitly. The distances of

the view, near, and far planes are all measured relative this point. The origin of the *uv*-coordinate system in the view plane is located where a line parallel ViewNormal and passing through the view reference point intersects the view plane. As pointed out in [MCHCAR], this specification can exhibit unnatural consequences. [FOLVAN] ameliorates this somewhat by requiring the view reference point to lie in the view plane, but the "slipping window" problem permitted by the Core spec still exists. The [MCHCAR] approach we have adopted avoids this problem.

The *viewing frustum* which results from passing the near and far clipping planes through a view volume, as shown in Figures 6-7, is illustrated in Figure 8. The purpose of the leftmost transformation in Figure 1 is to map a frustum such as this onto the normalized frustum shown in Figure 2. This *normalizing transformation* is detailed in the next section.

How to Use the View Spec

Here we show how to turn the view spec into the normalizing transformation. All that we are assuming about the world coordinate system is that it is right-handed. So the normalizing transformation must take this generality into account.

The steps to be performed are the following:

1. Translate the center of projection, ViewPoint, to the origin of the world system with matrix **A** (see below).
2. Transform the world system to *eye space* which is the system corresponding to Figure 2 with the eyepoint, or ViewPoint, at the origin. So X_c, Y_c, and Z_c in that figure correspond respectively to directions **u**, **v**, **n**. Thus eye space is left-handed. This coordinate transformation is given by matrix **B**. See Appendix C for a refresher on how to derive this matrix.
3. Skew (shear) the window center to the Z_c axis with the matrix **C**. Notice that lengths in the view plane are not changed by this transformation.
4. Scale so that the far face of the view volume is the unit square and lies at $Z_c = 1$. This is done with matrix **D**.

The concatenation $\mathbf{N} = \mathbf{ABCD}$ is the normalizing transformation. (We assume everywhere that vectors are row vectors and matrices multiply them from the right—i.e., computer graphics normal form.)

The four matrices are easy to specify if we introduce some simplifying symbols. Let **V** be the ViewPoint vector in world space. Let **0** be the vector $\begin{bmatrix}0 & 0 & 0\end{bmatrix}$ and let $\mathbf{I}_3$ be the 3×3 identity matrix. Let a superscript T indicate[1] the transpose of a vector. Let n, f, and d represent the near, far, and view distances, respectively.

[1] "represent" in the original

Finally, let c_u and c_v be the window center uv coordinates and s_u and s_v be the window halfsizes. Then

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_3 & \mathbf{0}^T \\ -\mathbf{V} & 1 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_3 & \mathbf{0}^T \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-c_u}{d} & \frac{-c_v}{d} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} \frac{d}{s_u f} & 0 & 0 & 0 \\ 0 & \frac{d}{s_v f} & 0 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

C is derived by requiring the center of the viewing window $\begin{bmatrix} c_u & c_v & d & 1 \end{bmatrix}$ be mapped by a skew into the point $\begin{bmatrix} 0 & 0 & d & 1 \end{bmatrix}$. Since z is to remain fixed with x and y being skewed, two entries of a skewing matrix are required. Let them be r and s. Then $c_u + rd = 0$ and $c_v + sd = 0$, giving the r and s shown in matrix **C**.

D is also easily derived. After the skew by **C**, the viewing volume is centered along the **n** axis. Since this skew preserves lengths along constant z planes, the viewing window extends from $-s_u$ to s_u in x and from $-s_v$ to s_v in y in the $z = d$ plane. The far face of the viewing volume, at $z = f$, extends from $-X$ to X in x and from $-Y$ to Y in y, where X and Y can be determined from similar right triangles to be

$$X = \frac{s_u f}{d} \quad Y = \frac{s_v f}{d}.$$

This plus the fact that the far face lies at distance f requires the scaling matrix **D** shown to obtain the canonical viewing volume. The case where f is infinite is handled in Appendix D.

[MCHCAR] shows that the product **CD** can be expressed as the product of

$$\mathbf{E} = \begin{bmatrix} \frac{1}{f} & 0 & 0 & 0 \\ 0 & \frac{1}{f} & 0 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{F} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -c_u & -c_v & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} \frac{1}{s_u} & 0 & 0 & 0 \\ 0 & \frac{1}{s_v} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

So the viewing parameters of ViewStructure may be nicely partitioned into six matrices with one view spec parameter per matrix.

Going the other way, the normalizing transformation in collapsed form is

$$\mathbf{N} = \mathbf{N}_L \mathbf{N}_R = \begin{bmatrix} \mathbf{u}^T & \mathbf{v}^T & \mathbf{n}^T & \mathbf{0}^T \\ -\mathbf{u}\cdot\mathbf{V} & -\mathbf{v}\cdot\mathbf{V} & -\mathbf{n}\cdot\mathbf{V} & 1 \end{bmatrix} \begin{bmatrix} \frac{d}{s_u f} & 0 & 0 & 0 \\ 0 & \frac{d}{s_v f} & 0 & 0 \\ \frac{-c_u}{s_u f} & \frac{-c_v}{s_v f} & \frac{1}{f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

where $\mathbf{N}_L = \mathbf{AB}$ and $\mathbf{N}_R = \mathbf{CD}$. For a fixed camera, $\mathbf{N}_L$ and $\mathbf{N}_R$ may be premultiplied. For the general case of a moving camera, only $\mathbf{N}_R$ is precomputed since $\mathbf{N}_L$ carries all the changes.

The normalizing transformation for orthographic projections can be shown [MCHCAR] to be

$$\mathbf{N}_o = \mathbf{AB} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -c_u & -c_v & -n & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{s_u} & 0 & 0 & 0 \\ 0 & \frac{1}{s_v} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$\mathbf{N}_o$ transforms the viewing volume (a rectangular box) into the normalized viewing volume shown in Figure 3. This may also be collapsed for efficient use with a moving camera to

$$\mathbf{N}_o = \mathbf{N}_L \mathbf{N}_{oR} = \begin{bmatrix} \mathbf{u}^T & \mathbf{v}^T & \mathbf{n}^T & \mathbf{0}^T \\ -\mathbf{u}\cdot\mathbf{V} & -\mathbf{v}\cdot\mathbf{V} & -\mathbf{n}\cdot\mathbf{V} & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{s_u} & 0 & 0 & 0 \\ 0 & \frac{1}{s_v} & 0 & 0 \\ 0 & 0 & \frac{1}{f-n} & 0 \\ \frac{-c_u}{s_u} & \frac{-c_v}{s_v} & \frac{-n}{f-n} & 1 \end{bmatrix}.$$

The Rest of the Viewing Transformation

So far we have covered just the leftmost box of Figure 1. The second box is where the clipping takes place. Details of how to clip to the canonical viewing volume are given in [FOLVAN]. There are several directions we could proceed at this point:

1. The perspective projection of points now inside the canonical viewing volume is applied (see Appendix A). This includes a divide by the homogeneous coordinate and reduces the 3-D data to 2-D information in the view plane—to be exact, in the view window in the view plane. These could then be directly mapped into the desired display space. We shall not further explore this possibility here.

2. As above, but the points after perspective projection are first mapped into NDC (normalized device coordinates) before a final additional mapping to a specific device. This has the advantage of device independence. For example, points in NDC could either be mapped to a Picture System screen or to a framebuffer display.

3. Either (1) or (2) is okay if no further manipulations of the image—e.g., no image transformations—are to be performed. Another example of further manipulation is hidden surface removal. Otherwise the other boxes in Figure 1 need to be implemented. The notion is to undo the squeezing of line-of-sight rays to a single point, the ViewPoint, into an orthogonal space where the line-of-sight rays are parallel, hence where arithmetic and geometry are easy (particulary for hidden surface comparisons). After any manipulations are performed in this space, a simple mapping into NDC is executed and then a mapping to display space as in (2).

Spreading the view volume into a rectangular prism is illustrated in Figure 9 from [FOLVAN]. The corresponding matrix is (see Appendix B)

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1-z_{\min}} & 1 \\ 0 & 0 & \dfrac{-z_{\min}}{1-z_{\min}} & 0 \end{bmatrix},$$

where $z_{\min}$ is the z coordinate of point $\begin{bmatrix} 0 & 0 & n & 1 \end{bmatrix}$ in eye space transformed by **CD**. So

$$z_{\min} = \frac{n}{f}$$

Working backwards give **P** in terms of the viewing parameters:

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{f}{f-n} & 1 \\ 0 & 0 & \dfrac{-n}{f-n} & 0 \end{bmatrix}.$$

This mapping deforms the canonical viewing frustum into a rectangular prism of width 2, height 2, and depth 1, centered on the z axis—i.e., NDC.

A translate of point $\begin{bmatrix} -1 & -1 & 0 & 1 \end{bmatrix}$ to the origin followed by a scale by .5 in x and y converts this prism into Core standard NDC, with x, y, and z each on [0.,1.]. The following matrix accomplishes this task:

$$\mathbf{K} = \begin{bmatrix} .5 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ .5 & .5 & 0 & 1 \end{bmatrix}.$$

Then there must be a mapping to the desired device. Let $X_{\min}$, $X_{\max}$, $Y_{\min}$, and $Y_{\max}$ define the desired portion of a display screen in image coordinates (but real). The z coordinate may be mapped into a desired range, $Z_{\min}$ to $Z_{\max}$, also. This map is given by

$$\mathbf{L} = \begin{bmatrix} X_{\max} - X_{\min} & 0 & 0 & 0 \\ 0 & Y_{\max} - Y_{\min} & 0 & 0 \\ 0 & 0 & Z_{\max} - Z_{\min} & 0 \\ X_{\min} & Y_{\min} & Z_{\min} & 1 \end{bmatrix}.$$

So the mapping from NDC into *real* screen space is given by

$$\mathbf{KL} = \begin{bmatrix} \frac{X_{max} - X_{min}}{2} & 0 & 0 & 0 \\ 0 & \frac{Y_{max} - Y_{min}}{2} & 0 & 0 \\ 0 & 0 & Z_{max} - Z_{min} & 0 \\ \frac{X_{max} + X_{min}}{2} & \frac{Y_{max} + Y_{min}}{2} & Z_{min} & 1 \end{bmatrix}.$$

Finally, there is the mapping of real screen space to integer screen space, or *pixel space*. The model we shall use for this mapping is the following[2]: A pixel at integer location (*i, j*) is assumed to represent the real space

$$i-.5 \le x \le i+.5 \quad j-.5 \le y \le j+.5 .$$

The interpretation of the model is that the smoothing filter that is used for sampling the real space to obtain the sampled pixel space has a central lobe with unit extent in *x* and *y* and centered at (*i, j*). The lobe extent is measured between the two zero crossings nearest the filter origin. A common—but frequently inadequate—filter is the box filter which is nothing but a central "lobe" of unit height everywhere and covering precisely the square of space defined above. So the mapping from real screen space to pixel space is

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \rightarrow \text{floor}\left(\begin{bmatrix} x+.5 & y+.5 & z+.5 & 1 \end{bmatrix}\right),$$

where *z* is treated just like *x* and *y*, and

$$\text{floor}\left(\begin{bmatrix} a & b & c & d \end{bmatrix}\right) \rightarrow \begin{bmatrix} \text{floor}(a) & \text{floor}(b) & \text{floor}(c) & \text{floor}(d) \end{bmatrix}$$

is the extension of Unix function *floor()* to vectors. The analytic part of this function can be expressed by the matrix

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ .5 & .5 & .5 & 1 \end{bmatrix}.$$

The application of this matrix is then followed by the *floor()* function. It returns doubles which may be assigned to integers with no loss of precision. **M** is nothing more than a translate by half a pixel.

Screen Mapping Details

Unfortunately, not all displays fit into the neat form assumed in the derivation above. In fact, our raster displays have *y* positive down. The coordinate transformation which takes NDC into this inverted form is derived as $\mathbf{J}_{3r}$ in Appendix C. Our calligraphic display has far *z* mapped to lower intensity than near *z*. If intensity value is taken as the screen map in the *z* direction, then the screen normal points out of the screen. Similarly, the transformation taking NDC into

[2] I no longer advocate this mapping. See my Microsoft Technical Memo 6, *A Pixel is* Not *a Little Square, A Pixel is* Not *a Little Square, A Pixel is* Not *a Little Square! (And a Voxel is* Not *a Little Cube)*, Jul 1995, for details.

this form is derived in Appendix C as $\mathbf{J}_{3c}$. The full transformation is given in either case by

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_3 & \mathbf{0}^T \\ 0 & 1 \end{bmatrix}.$$

So the complete *screen mapping* from NDC to pixel space is $\mathbf{S} = \mathbf{JKLM}$ followed by the *floor()* function. Using $\mathbf{J}_{3r}$, as derived in Appendix C, gives the *raster screen mapping*:

$$\mathbf{S}_r = \begin{bmatrix} \frac{X_{\max} - X_{\min}}{2} & 0 & 0 & 0 \\ 0 & \frac{Y_{\min} - Y_{\max}}{2} & 0 & 0 \\ 0 & 0 & Z_{\max} - Z_{\min} & 0 \\ \frac{X_{\max} + X_{\min} + 1}{2} & \frac{Y_{\max} + Y_{\min} + 1}{2} & Z_{\min} + \frac{1}{2} & 1 \end{bmatrix}.$$

$\mathbf{J}_{3c}$, also derived in Appendix C, yields the calligraphic screen mapping:

$$\mathbf{S}_c = \begin{bmatrix} \frac{X_{\max} - X_{\min}}{2} & 0 & 0 & 0 \\ 0 & \frac{Y_{\max} - Y_{\min}}{2} & 0 & 0 \\ 0 & 0 & Z_{\min} - Z_{\max} & 0 \\ \frac{X_{\max} + X_{\min} + 1}{2} & \frac{Y_{\max} + Y_{\min} + 1}{2} & Z_{\min} + \frac{1}{2} & 1 \end{bmatrix}.$$

These are named for the type of display to which they typically, but not necessarily, correspond.

Nonsquare Pixel Spacing[3]

Our Ikonas framebuffer displays have a unit *x* length which is greater than the unit *y* length in the natural coordinate system for the display which is its pixel location coordinates. The ratio of unit *x* length to unit *y* length is defined to be the *pixel aspect ratio (PAR)* of a device.

The PAR can be determined from a window of known horizontal and vertical resolution, HRES and VRES, and known aspect ratio, AR:

$$PAR = AR * \frac{VRES}{HRES}.$$

This is usually derived from the aspect ratio of the full display and the full horizontal and vertical resolutions. For example, with the Ikonas display, HRES is 512 pixels, VRES is 488 pixels, and the aspect ratio is the standard video aspect ratio of 4/3; hence

$$PAR_{Ikonas} = \frac{4}{3} * \frac{488}{512} \approx 1.27.$$

[3] "Nonsquare Pixels" in the original, a terminology I now strongly eschew.

Interestingly, the PAR does not explicitly enter into the viewing transformation above. Its importance becomes evident when windows other than full screen windows are displayed on a device. If the screen limits X_{min}, X_{max}, etc., are such that the aspect ratio matches that stored in the corresponding view struct, there is no problem; the viewing transformation as already described handles the non-squareness of the pixels. But in order to define the screen limits in the first place, given a known aspect ratio, requires the PAR. Similarly, to derive the aspect ratio from a given set of screen limits also requires the PAR, in general.

Conventional Display Spec

The data necessary for completely specifying the screen mapping for a display resides in the *conventional display spec* given by the following conventional structure:

```
typedef double ScreenType;
typedef struct { ScreenType x, y, z; } ScreenPoint;

struct DisplayStructure {
    ScreenPoint FullScreenMin;          /* in pixel space */
    ScreenPoint FullScreenMax;          /* in pixel space */
    double FullScreenAspectRatio;       /* full screen width to height */
    ScreenPoint ScreenMin;              /* current, in pixel space */
    ScreenPoint ScreenMax;              /* current, in pixel space */
    WorldPoint ScreenNormal;            /* screen normal vector in NDC */
    WorldPoint ScreenUp;                /* screen up vector in NDC */
};
```

Screen right is always assumed to be right ($[1 \quad 0 \quad 0]$ in NDC). The matrix **J** can be determined from this assumption and from ScreenNormal and ScreenUp in the struct. Windowing is permitted in *z* as well as in *x* and *y*, but linear distortion in *z* is ignored. That is, there is no need for a width-to-depth pixel aspect ratio. FullScreenMin holds X_{min}, Y_{min}, and Z_{min}; FullScreenMax holds X_{max}, Y_{max}, and Z_{max}. Thus the PAR for a device described with such a structure is

$$PAR = FullScreenAspectRatio * \frac{Y_{max} - Y_{min} + 1}{X_{max} - X_{min} + 1}.$$

Summary

The entire viewing transformation on a point **p** in model space is given by

$$\mathbf{pNPS} = \mathbf{pABEFGHPJKLM}.$$

Modeling transformations are implicit in **p**. If **T** is a modeling transformation and **q** is a point in the "raw" model database, then $\mathbf{p} = \mathbf{qT}$. In general, a homogeneous divide (division by the homogeneous coordinate) should follow transformation by **T**. Clipping occurs on the points **pN**. Division by the homogeneous coor-

dinate occurs to points **pNP** to obtain NDC. The screen mapping **S** is either $\mathbf{S}_c$ or $\mathbf{S}_r$. The *floor()* function is applied to points **pNPS** to obtain integer display coordinates. Any image transformations **X** or hidden surface calculations are applied to points **pNP** in NDC. If **X** maps NDC to NDC, the full transform is **pNPXS**. See Figure 10 (cf., Figure 1). **N** is conveniently decomposed into $\mathbf{N}_L\mathbf{N}_R$ corresponding to the parts which do and do not change with camera motion.

A shortcut for the case where no image transformations or hidden surface calculations are to be performed is represented by **pNQS'** where **Q** is defined in Appendix A and $\mathbf{S}' = \mathbf{RS}$ is the corresponding mapping to display space where

$$\mathbf{R} = \begin{bmatrix} \frac{f}{d} & 0 & 0 & 0 \\ 0 & \frac{f}{d} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

For this shortcut the homogeneous coordinate division is performed on points **pNQ**, and *z* information is ignored in the screen map.

A far plane at infinity is not handled by the formulation above but may be dealt with by the viewing transformation

$$\mathbf{pN}_\infty\mathbf{P}_\infty\mathbf{S} = \mathbf{pABEF}_\infty\mathbf{GHP}_\infty\mathbf{JKLM}$$

(see Appendix D), where $\mathbf{N}_\infty = \mathbf{N}_L\mathbf{N}_{R_\infty}$. This is a *different* transformation from that above and must be treated with care. In particular, the normalization is to the canonical *infinite* viewing frustum, so the clipping algorithm used must be altered in the way it handles clipping in z (but not in x and y). The shortcut method is represented by $\mathbf{pN}_\infty\mathbf{P}_\infty\mathbf{S}$ in this case.

Handy Ways to Generate a View

There are a variety of ways to write an interface to drive the viewing transformation. A typical way is to provide two routines *View(distance, azimuth, pitch, roll)* and *Perspective(fieldofview, aspectratio, near, far)* from which all the viewing parameters can be derived. The first establishes the location and direction of the viewing frustum; the second defines its size and shape. The two routines correspond to the two matrices $\mathbf{N}_L$ and $\mathbf{N}_R$ derived above; the parameters that change as a camera moves are concentrated in one routine.

Before describing these routines in detail, the special angles *azimuth*, *pitch*, and *roll* will be defined. There are several ways to define them (and name them) so it is important to settle on one set of definitions. The definitions here are based on those used in flying aircraft over terrain with *x* east, *y* north, and *z* up (following a suggestion by Loren Carpenter).

Azimuth is the angle of a radius vector about the origin of world space in the *xy* plane. It is 0 degrees at the positive *y* axis and increases positively clockwise about the *z* axis (i.e., follows lefthand rule about *z* axis). A mnemonic for this an-

gle is a compass at the origin, lying on the *xy* plane. Azimuth is 0 degrees when the needle points north; 90 degrees is east.

Pitch is the angle of a radius vector out of the *xy* plane. So it is 0 at the *xy* plane, positive above it, and negative below.

Roll is the angle about a view normal as seen looking along the normal. It is 0 when the rightward vector is to the right, parallel the *xy* plane. It increases positively counterclockwise looking in the positive normal direction. Thus roll follows the lefthand rule about the view normal.

When these three angles are relative another origin other than the world space origin, then the coordinate system against which they are measured is assumed to be a simple translation of the world space system to the new origin—e.g., the eyepoint. Note that the common alternative names “yaw”, “elevation”, and “twist” are not allowed by these definitions.

View() defines the location of the ViewPoint by giving its *distance* from the world space origin along a radius vector at angle *azimuth* in the *xy* plane and at angle *pitch* from the *xy* plane. The vector from the origin to this point defines the ViewNormal. ViewUp is derived from the angle *roll* measured about the ViewNormal. ViewUp is assumed perpendicular to ViewNormal.

A variation on *View()* which is more versatile is *FullView(vx, vy, vz, azimuth, pitch, roll)* which places the ViewPoint at $[vx \quad vy \quad vz \quad 1]$ and then locates the ViewNormal and ViewUp with the three angles as before but with respect to the ViewPoint as origin. So the ViewNormal is not restricted to be collinear with the line from world space origin to ViewPoint.

Perspective() defines the viewing frustum, assuming a centered window. This is a common assumption but does not fully utilize the view spec. The *fieldofview* and *aspectratio* are related to the WindowHalfsize and ViewDistance as explained before. Since *near* and *far* are given as distances of the near and far clipping planes in eye space, they can be used directly for NearDistance and FarDistance in ViewStructure. Another assumption this particular interface makes is that the view plane and the near clipping are the same. This is another common restriction on the generality provided by the view spec. ViewDistance is simply NearDistance. Setting ProjectionType to PERSPECTIVE completes the view spec in this example.

Bill Reeves has provided routines of the flavor of those above plus several others for driving the viewing transformation. For example, he also provides the routines *LookAt(vx, vy, vz, px, py, pz, roll)* and *Camera(xr, yr, zr, xn, yn, zn, xup, yup, zup, deye)* in place of *View()* and *Window(wleft, wright, wtop, wbottom, near, far)* in place of *Perspective()*.

LookAt() locates the ViewPoint at $[vx \quad vy \quad vz \quad 1]$. The ViewNormal direction is that defined by a vector from ViewPoint to $[px \quad py \quad pz \quad 1]$. *Roll* is interpreted as before. This routine is handy for keeping a single point centered on the screen as the camera flies an elaborate path.

Camera() is the fanciest of the three routines. The point [*xr yr zr* 1] is any convenient reference point. The vector [*xn yn zn* 1] defines the ViewNormal. The vector [*xup yup zup* 1] defines ViewUp. In this case ViewUp is not necessarily perpendicular to ViewNormal, as permitted by the view spec. The ViewPoint is located distance *deye* from reference point along and in the direction of the ViewNormal.

Window() computes the view window from the left, right, top, and bottom positions of its edges. Hence it is more general than *Perspective()*, allowing non-centered windows.

Notice that none of the interfaces above fully utilize the view spec. The combination of *Camera()* and *Window()* comes closest, lacking only the capability of specifying a view plane distinct from the near plane.

The Core recommendation for an interface is more "gut-level" with routines such as the following which are self-explanatory:

SetViewPoint(x, y, z)
SetViewNormal(x, y, z)
SetViewUp(x, y, z)
SetViewDistance(d)
SetViewDepth(near, far)
SetWindow(cu, cv, su, sv)
SetProjection(type)

For each of these routines there is a corresponding inquiry routine such as *InquireViewPoint(px, py, pz)* where *px*, *py*, and *pz* are the addresses of variable to be filled with the current coordinates of the ViewPoint.

The Core standard also has default values for all parameters. Here is a set of defaults for ViewStructure:

ViewPoint: 0 0 0 (at world space origin)
ViewNormal: 0 1 0 (in conventional world space)
ViewUp: 0 0 1 (in conventional world space)
ViewDistance: 1 (45° full horizontal field of view)
NearDistance: 1 (view and near planes coincident)
FarDistance: 1e5 (larger than 64K but less than infinity)
WindowCenter: 0 0 (on axis)
WindowHalfsize: 0.41421356 0.31066017 (4/3 aspect ratio)
Projection: PERSPECTIVE

We have adopted a set of low-level routines, inquiry routines, and defaults as part of our convention. Similarly, we have adopted low-level routines and defaults for the conventional display structures. For example, the raster display structure defaults to

FullScreenMin: 0 0 0

FullScreenMax: 511 487 65535 (assuming two 8-bit channels for *z*)
FullScreenAspectRatio: 1.3333333333
ScreenMin: 0 0 0
ScreenMax: 511 487 65535
ScreenNormal: 0 0 1
ScreenUp: 0 –1 0

And the calligraphic display structure would default to

FullScreenMin: -2048 –2048 0
FullScreenMax: 2047 2047 255
FullScreenAspectRatio: 1
ScreenMin: -2048 –2048 0
ScreenMax: 2047 2047 255
ScreenNormal: 0 0 -1
ScreenUp: 0 1 0

References

[CORE79]

Status Report of the Graphic Standards Planning Committee: Chapter 5: Viewing Operations and Coordinate Transformations, **Computer Graphics**, Vol 13, No 3, Aug 1979 (SIGGRAPH ’79).

[FOLVAN]

James D Foley and Andries Van Dam, *Chapter 8: Viewing in Three Dimensions*, **Fundamentals of Interactive Computer Graphics**, Addison-Wesley Publishing Company, Menlo Park, CA, 1982.

[MCHCAR]

James C Michener and Ingrid B Carlbom, *Natural and Efficient Viewing Parameters*, **Computer Graphics**, Vol 14, No 3, Jul 1980 (SIGGRAPH ’80), 238-245.

Appendix A: Don’t Clip After Perspective

Perspective can be applied in two ways. The first is a 3-D to 2-D *perspective projection* which maps all points, even those behind the eye, into a constant *z* plane in front of the eye. If the view plane is at distance *D* from the eyepoint, or ViewPoint, which is at the origin and looking out positive *z*, then the perspective projection is accomplished by the operation

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} \frac{X}{W} & \frac{Y}{W} & \frac{Z}{W} \end{bmatrix}$$

where

$$\begin{bmatrix} X & Y & Z & W \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \mathbf{Q}.$$

Since we are particularly interested in the canonical view volume of Figure 2 for which $D = \frac{d}{f}$,

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{f}{d} \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Regardless of x and y, the resulting z' is D. So all points with negative z—i.e., all points behind the eye—are mapped to points with positive z.

Matrix **Q** is easy to derive. Let x' and y' be the location in the view plane of the projection of point $[x \quad y \quad z \quad 1]$. Thus the point and its projection lie on a line collinear with the eyepoint. By similar right triangles,

$$\frac{x'}{D} = \frac{x}{z} \quad \frac{y'}{D} = \frac{y}{z}.$$

So

$$x' = \frac{xD}{z} \quad y' = \frac{yD}{z}.$$

Clearly, $z' = D$. But matrix **Q** times the given point, followed by a homogeneous coordinate division, transforms the given point to $[x' \quad y' \quad D \quad 1]$.

The second perspective method is a 3-D to 3-D mapping called the *perspective transformation* which preserves z ordering of visible points and delays projection into 2-D until a later step. See Appendix B for derivation of the perspective transformation matrix **P**. Clipping of lines that pass from before the eye to behind the eye becomes tricky, although not impossible as in the perspective projection case. It can be shown that **P** maps points behind the eye into points beyond the far clipping plane. In particular, for the canonical viewing volume, point $[x \quad y \quad z \quad 1]$ is mapped by **P** into a point with

$$z' = \left(\frac{f}{f-n}\right)\left(1 + \frac{n}{f}|z|\right).$$

Thus $z' > 1$ for $z < 0$. So the endpoints of a line passing from a visible point to a point behind the eye transform into two points which, if naively connected, form a transformed line segment which is not correct. If care is exercised (see [BLINWL] for details), the two points may be connected by passing through infinity to get the correct line segment. It is probably best simply to have clipped before encountering this problem and hence avoiding it.

Of course, **P** and **Q** must give the same results in x and y (to within a scale factor $\frac{d}{f}$ for the canonical viewing volume).

Reference

[BLINWL]

James C Blinn and Martin E Newell, *Clipping Using Homogeneous Coordinates*, **Computer Graphics**, Vol 12, No 3, Jul 1978 (SIGGRAPH '78), 245-251.

Appendix B: Perspective Matrix Derivation

To derive the perspective transformation matrix **P** for the canonical view volume, we want to solve for the 16 elements of

$$\mathbf{P} = \begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix}.$$

One laborious method is to derive a set of simultaneous equations which determine **P** from the known mappings of several points—e.g., from the eight corner mappings: $[1\ \ 1\ \ 1]$ to $[1\ \ 1\ \ 1]$, ..., $[z_{min}\ \ z_{min}\ \ z_{min}]$ to $[1\ \ 1\ \ 0]$, ... Another method is based on the observation that **P** will induce only linear size changes in planes of constant z—no nonlinear perspective distortion. So the far plane face is unchanged by **P**, implying (see Figure 9)

$$ax + by + c + d = Wx$$

for some homogeneous coordinate W and all x and y in the face. Thus

$$a = W \quad b = 0 \quad d = -c$$

Similarly, the other three relationships yield

$$e = i = j = m = n = 0 \quad f = W \quad h = -g \quad k + l = o + p = W.$$

So at this step **P** has form:

$$\mathbf{P} = \begin{bmatrix} W & 0 & 0 & 0 \\ 0 & W & 0 & 0 \\ c & g & k & o \\ -c & -g & W-k & W-o \end{bmatrix}.$$

The $z = z_{min}$ face goes through a scale change by $\frac{1}{z_{min}}$ in x and y when mapped by **P** into $z = 0$. Hence

$$Wx + c(z_{min} - 1) = \frac{W'x}{z_{min}}$$

for some homogeneous coordinate W'. Thus

$$c = 0 \quad W' = Wz_{min}.$$

Similarly, from the other three relationships

$$g = 0 \quad k = \frac{W}{1 - z_{min}} \quad o = W.$$

So

$$\mathbf{P} = \begin{bmatrix} W & 0 & 0 & 0 \\ 0 & W & 0 & 0 \\ 0 & 0 & \frac{W}{1-z_{\min}} & W \\ 0 & 0 & \frac{-z_{\min} W}{1-z_{\min}} & 0 \end{bmatrix}.$$

Since the *W* divides out in the homogeneous coordinate division, it may be set to 1.

Appendix C: Coordinate Transformations

This is a quick review of the basic theory. We are given points in a coordinate system with orthogonal axes *x*, *y*, and *z*. We wish to express these points in terms of another orthogonal coordinate system—that is, we wish to perform a coordinate transformation on the points. Suppose the unit vectors defining the three new axes are **u**, **v**, and **n**.

For example, **u** might be the vector $[1 \quad 0 \quad 0]$ in the *xyz* space; **v** might be $[0 \quad 0 \quad 1]$; and **n** might be $[0 \quad 1 \quad 0]$. These are orthogonal unit vectors. At Lucasfilm, *conventional world space* is the right-handed *xyz* system with *z* up, *x* right, and *y* in. (Think of a map of the country with *x* east, *y* north, and *z* up.) Using this as an example, **u** lies along the *x* axis, **v** lies along the *z* axis, and **n** lies along the *y* axis. So the *uvn* system is left-handed with **u** left, **v** up, and **n** in. This is the eye space definition at Lucasfilm.

We want to find a matrix $\mathbf{B}_3$ such that $\mathbf{u}\mathbf{B}_3 = [1 \quad 0 \quad 0]$. Thus the point **u** is transformed by $\mathbf{B}_3$ into a principal axis. Similarly, we want $\mathbf{v}\mathbf{B}_3 = [0 \quad 1 \quad 0]$ and $\mathbf{n}\mathbf{B}_3 = [0 \quad 0 \quad 1]$. This can be expressed by the following matrix equation:

$$\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{n} \end{bmatrix} \mathbf{B}_3 = \mathbf{I}.$$

Therefore

$$\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{n} \end{bmatrix} = \mathbf{B}_3^{-1}.$$

Hence

$$\mathbf{B}_3 = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{n} \end{bmatrix}^{-1}.$$

Since **u**, **v**, and **n** are orthogonal unit vectors,

$$\begin{bmatrix}\mathbf{u}\\ \mathbf{v}\\ \mathbf{n}\end{bmatrix}^{-1} = \begin{bmatrix}\mathbf{u}^T & \mathbf{v}^T & \mathbf{n}^T\end{bmatrix}$$

as used in the text.

Thus, for the example above,

$$\mathbf{B}_3 = \begin{bmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}.$$

Another example is the transformation from NDC to "upside down" NDC, convenient for screen mappings for raster displays which typically have *y* positive down instead of up. Thus **u** in terms of NDC space is $[1\ \ 0\ \ 0]$, $\mathbf{v} = [0\ \ -1\ \ 0]$, and $\mathbf{n} = [0\ \ 0\ \ 1]$. Therefore

$$\mathbf{J}_{3r} = \begin{bmatrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

is the desired coordinate transformation. Similarly, for our calligraphic display **u** is $[1\ \ 0\ \ 0]$, $\mathbf{v} = [0\ \ 1\ \ 0]$, and $\mathbf{n} = [0\ \ 0\ \ -1]$. Therefore

$$\mathbf{J}_{3c} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & -1\end{bmatrix}.$$

Appendix D: Far Plane at Infinity

A far plane at infinity causes problems with the viewing transformation as presented in the text. The normalizing transformation **N** maps all finite points into the point $[0\ \ 0\ \ 0\ \ 1]$. In particular, $\mathbf{N}_R$ becomes all 0's except for $\mathbf{N}_R[3][3] = 1$. So the canonical viewing frustum is useless in this case. Here we consider changes to the viewing transformation which make it useful for the interesting case of a far plane at infinity.

The main idea, as before, is to map an arbitrary frustum to one convenient for clipping. The "canonical" frustum we recommend here is the same as before for *x* and *y* but leaves an infinite far plane at infinity. The previous derivation mapped the far plane into the $z = 1$ plane. This one maps the view plane into the $z = 1$ plane and the viewing window into a square from –1 to 1 in both *x* and *y*. So the *canonical infinite view frustum* is bounded by the same planes top, bottom, left, and right as with the (noninfinite) canonical view frustum, but the far face is at infinity and the view window now occupies the space occupied by the far face in the noninfinite case.

An argument parallel to that for deriving **D** says that the desired mapping for the view plane in the canonical infinite view frustum scales *x* by $\frac{1}{s_u}$, *y* by $\frac{1}{s_v}$, and *z* by $\frac{1}{d}$. So

$$\mathbf{D}_\infty = \begin{bmatrix} \frac{1}{s_u} & 0 & 0 & 0 \\ 0 & \frac{1}{s_v} & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Hence

$$\mathbf{N}_{R_\infty} = \begin{bmatrix} \frac{1}{s_u} & 0 & 0 & 0 \\ 0 & \frac{1}{s_v} & 0 & 0 \\ \frac{-c_u}{s_u d} & \frac{-c_v}{s_v d} & \frac{1}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

A, **B**, and **C** are unaffected, so $\mathbf{N}_L$ is unaffected. The decomposition of $\mathbf{N}_\infty = \mathbf{N}_L \mathbf{N}_{R_\infty}$ into matrices partitioned by view parameter is the same as before with

$$\mathbf{E}_\infty = \mathbf{I},$$

$$\mathbf{F}_\infty = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

and **G** and **H** unchanged.

The perspective projection is derived just as in Appendix A but in this case $D = 1$ so

$$\mathbf{Q}_\infty = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The perspective transformation now must map the canonical infinite view frustum onto NDC. A 2-step derivation yields the transformation: First, derive a mapping which maps $z_{\min}$ into 0 and leave $z = 1$ unchanged. An argument paral-

lel that of Appendix B shows that the desired mapping is the same as **P** in its $z_{\min}$ formulation, but since

$$z_{\min} = \frac{n}{d}$$

in this formulation,

$$\mathbf{P}'_{\infty} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{d}{d-n} & 1 \\ 0 & 0 & \frac{-n}{d-n} & 0 \end{bmatrix}.$$

This matrix maps the infinite far plane into $z = \frac{d}{d-n}$. So a prescaling of z by $\frac{d-n}{d}$ gives the desired mapping to NDC:

$$\mathbf{P}_{\infty} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & \frac{-n}{d} & 0 \end{bmatrix}.$$

$\mathbf{P}_{\infty}$ and $\mathbf{Q}_{\infty}$ give identical results in x and y so

$$\mathbf{R}_{\infty} = \mathbf{I}.$$

The screen mapping portion of the viewing transformation is not affected.

The case of orthographic projection may be treated similarly. The canonical viewing volume is extended to infinity as for the perspective projection case and the normalization matrix is reworked to map the near plane to $z = 0$ and the view plane to $z = 1$. The result is

$$\mathbf{N}_{oR_{\infty}} = \begin{bmatrix} \frac{1}{s_u} & 0 & 0 & 0 \\ 0 & \frac{1}{s_v} & 0 & 0 \\ 0 & 0 & \frac{1}{d-n} & 0 \\ \frac{-c_u}{s_u} & \frac{-c_v}{s_v} & \frac{-n}{d-n} & 1 \end{bmatrix}.$$

Whereas d did not figure in the orthographic transformation at all in the finite case, it takes on a principal role here. The case $d \equiv n$ is treated specially. It suffices to set the [2][2] element to 1 and the [3][2] element to $-n$. This maps the near plane to $z = 0$ and the view plane to $z = 1$ but leaves the far plane at infinity. Notice that the screen mapping in this case is not valid for z. Its z component may be

thought of as a linear scale and translate, however. The Z_{max} factor becomes a scaling control. The minimum *z* will be Z_{min}, however.

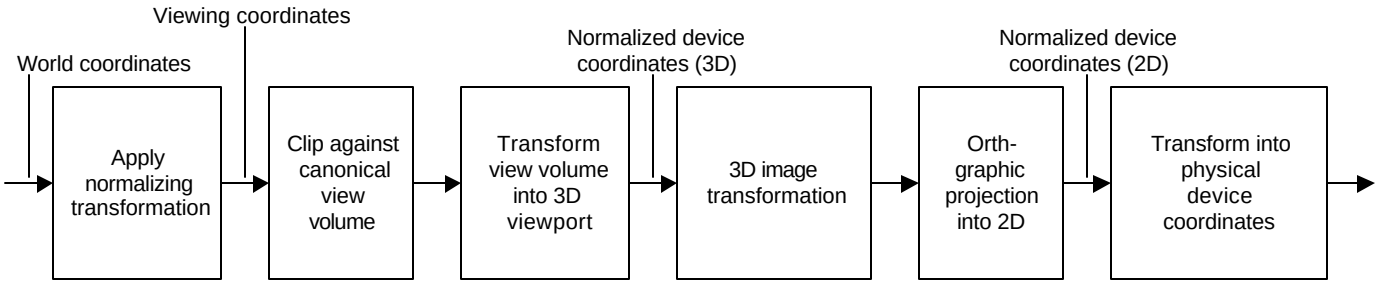

Figure 1. 3D viewing process extended to include 3D image transformations (from [FOLVAN]).

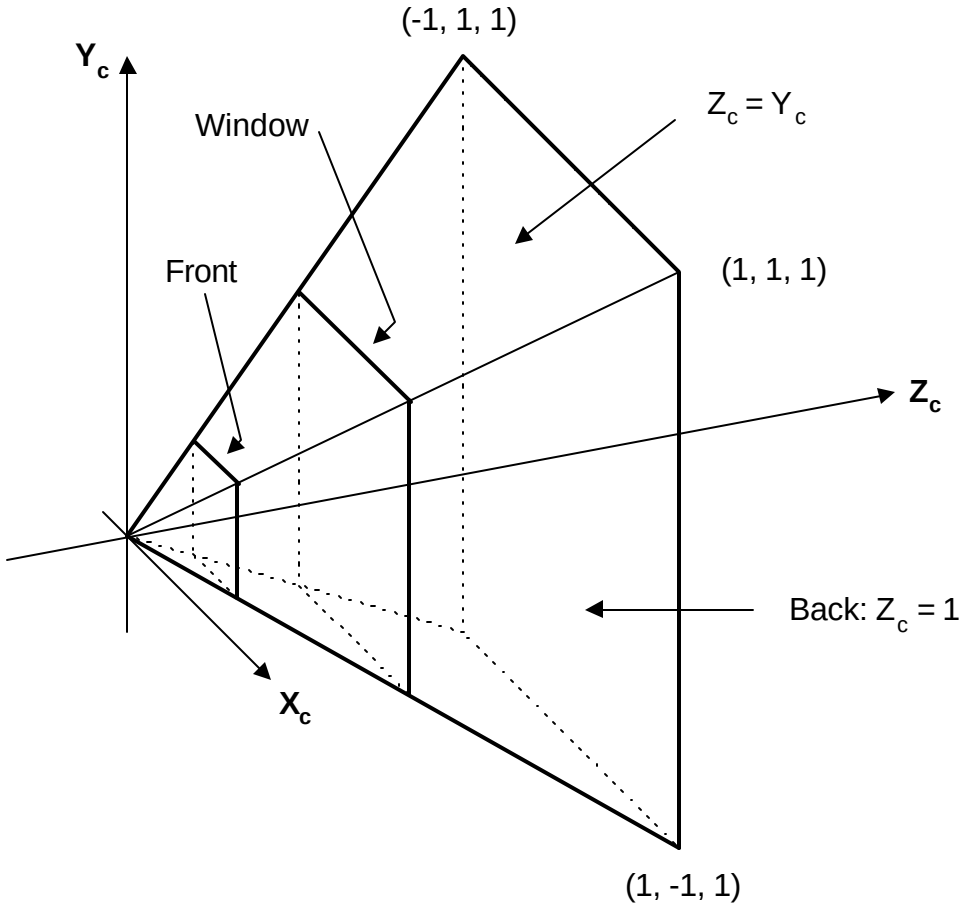

Figure 2. Normalized clipping region for perspective projection. $0 \le |X_c|, |Y_c| \le Z_c$; Front $\le Z_c \le 1$. (from [MCHCAR]).

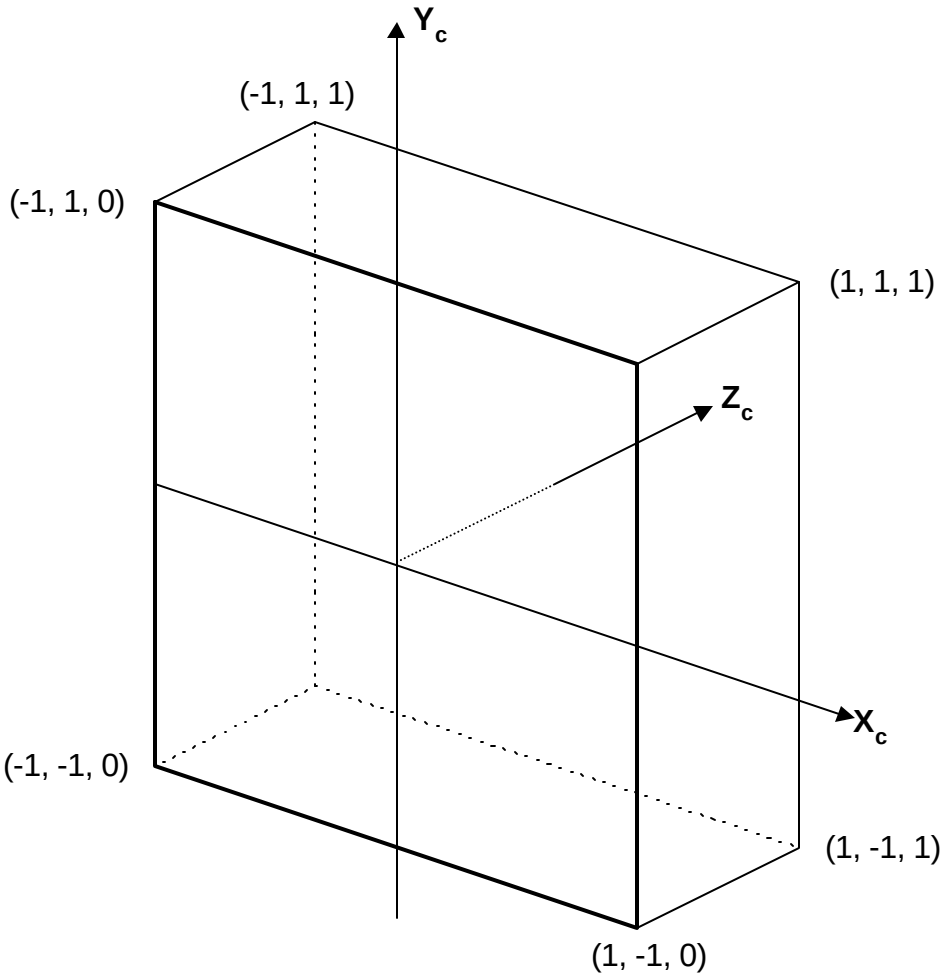

Figure 3. Normalized clipping region for parallel projection. -1 <_ Xc,Yc <_ 1 and 0 <_ Zc <_ 1. (from [MCHCAR])

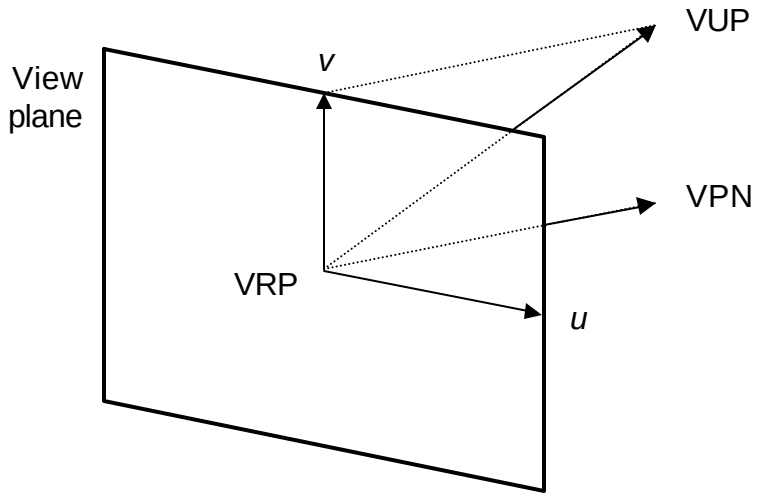

Figure 4. *uv*-system in the view plane (from [FOLVAN]).

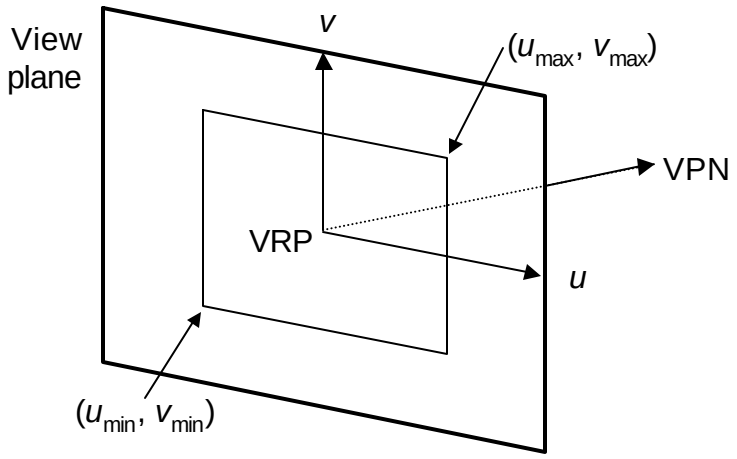

Figure 5. Window in *uv*-coordinates (from [FOLVAN]).

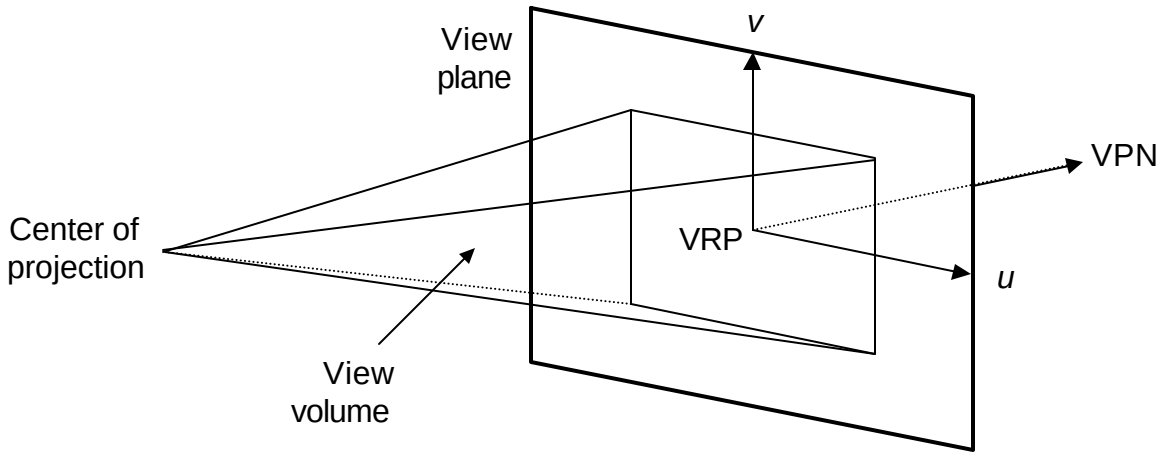

Figure 6. Semi-infinite pyramid view volume for perspective projection (from [FOLVAN]).

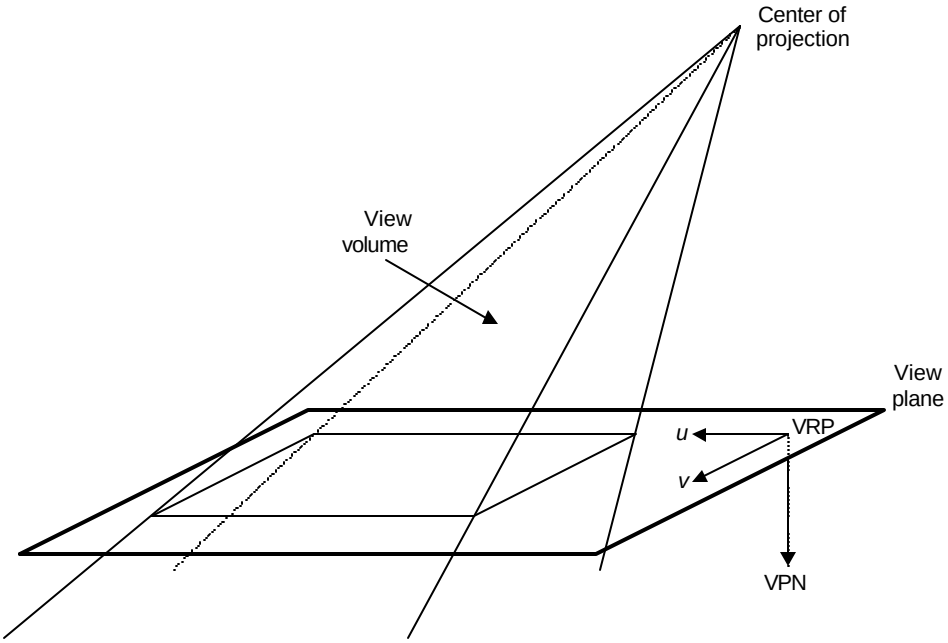

Figure 7. Another semi-infinite pyramid view volume for perspective projection (from [FOLVAN]).

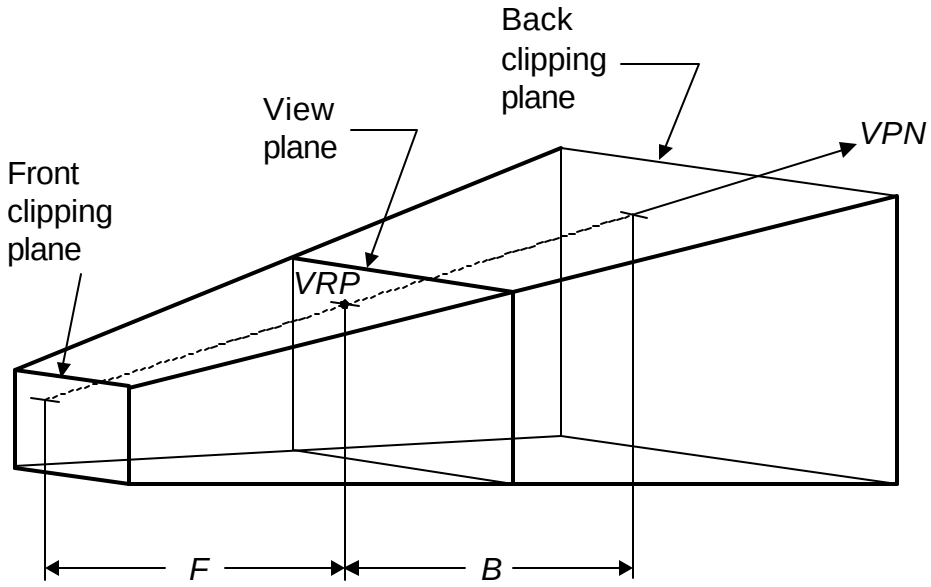

Figure 8. Truncated view volume (from [FOLVAN]).

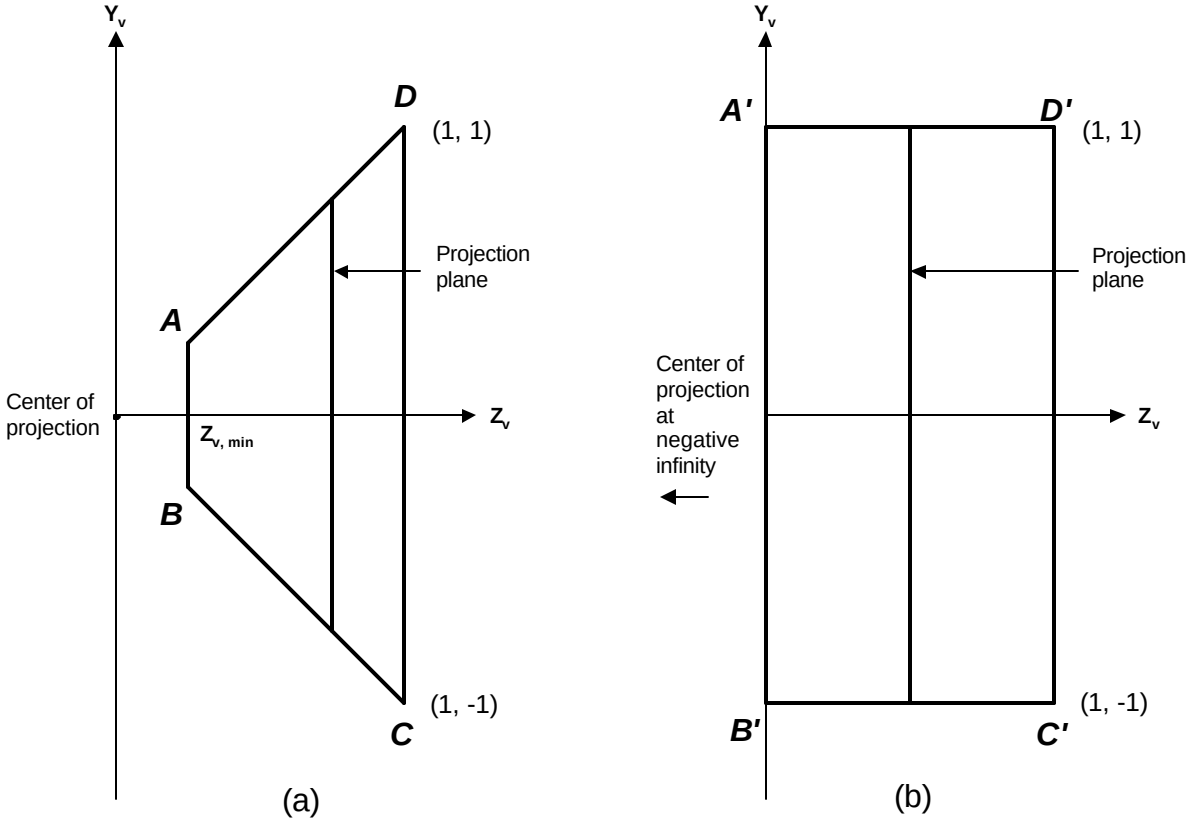

Figure 9. Side views of normalized perspective view volume (a) before and (b) after application of matrix **P** (from [FOLVAN]).

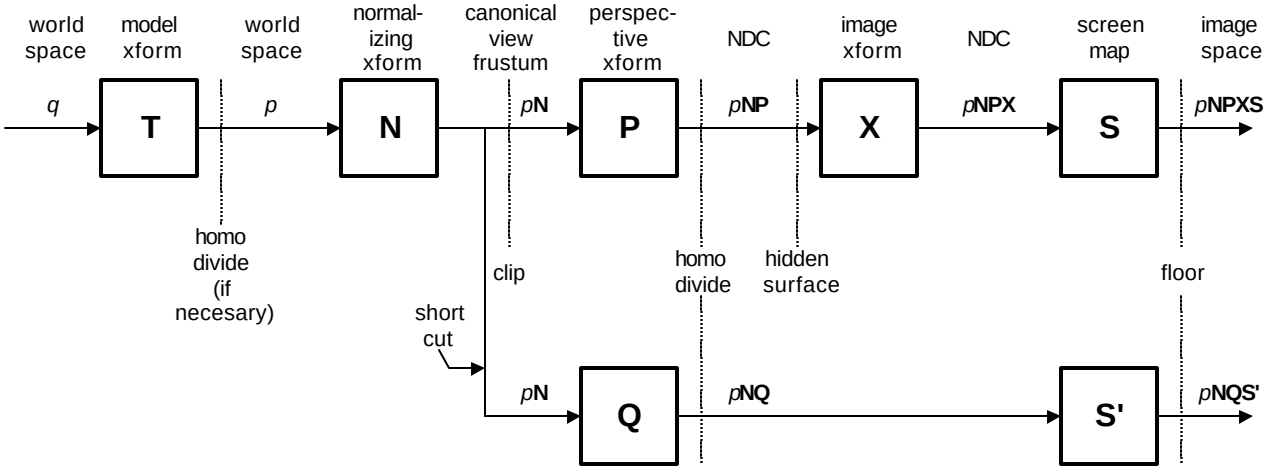

Figure 10. The complete viewing transformation.