Appendix 5: Perspective Correct Interpolation & Interpolation of Linear Functions in Homogeneous Space

We start with interpolation of texture coordinates which are meant to be linearly interpolated across the polygons. The same algorithm applies to other linear functions as well.

Say we would like to texture a rectangle made up of two triangles with a regular texture board texture. To render the triangles consider the transformations that the vertices go through. Essentially the vertices of the triangle are projected to the 2D screen space through a perspective projection followed by a perspective division. Consider one of the triangles. Say the vertices in the view space are $({}^{v}x_i,\ {}^{v}y_i,\ {}^{v}z_i, 1.0), i \in [0, 1, 2]$. Then the vertices $(\ {}^{p}x_i,\ {}^{p}y_i,\ {}^{z}z_i,\ {}^{p}w_i)$ are given by:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} {}^{v}x_i \\ {}^{v}y_i \\ {}^{v}z_i \\ 1.0 \end{bmatrix} = \begin{bmatrix} {}^{v}x_i \\ {}^{v}y_i \\ {}^{v}z_i \\ {}^{v}z_i \end{bmatrix} \equiv \begin{bmatrix} {}^{p}x_i \\ {}^{p}y_i \\ {}^{p}z_i \\ {}^{p}z\ _i \end{bmatrix} \qquad \text{A 6.1}$$

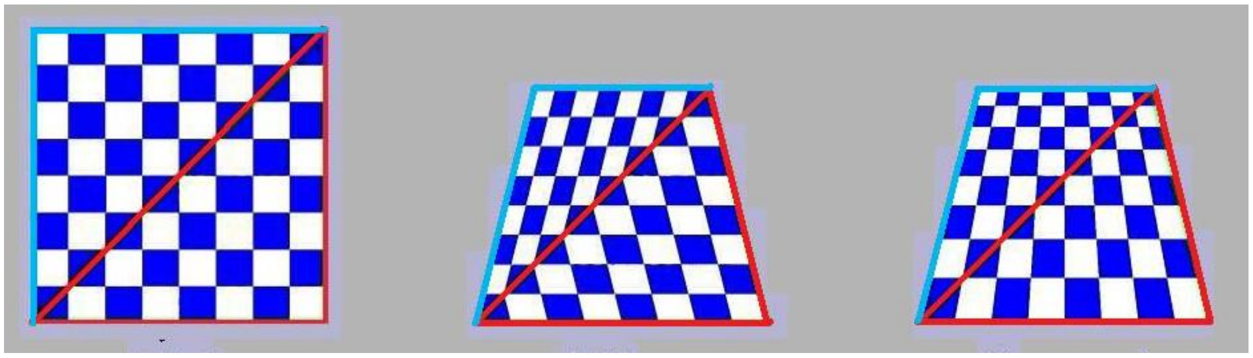
The vertices ${}^{p}x_i$ are in the perspective homogeneous space. To convert them to the screen space, we carry out a perspective division

$$({}^{s}x_i,\ {}^{s}y_i\ ) \equiv \left(\frac{{}^{p}x_i}{{}^{p}z_i}, \frac{{}^{p}x_i}{{}^{p}z_i} \right) \qquad \text{A 6.2}$$

where $({}^{s}x_i,\ {}^{s}y_i)$ are the screen coordinates of the vertices. In order to render the triangle, we transform each vertex of the triangle to the screen coordinates and then determine the so called screen coverage of the triangle – the screen area covered by the triangle. For each pixel within the coverage area, we need to figure out the fragment value based on linear interpolation. For completely determining the fragment value at each pixel, we need to map a texel from the texture, for each pixel. To sample the texture we need the texture coordinates corresponding to the pixel and those coordinates need to be interpolated form the texture coordinate values that are provided as vertex elements at the vertices of the primitive.

The texture coordinates are to be linearly interpolated; but this can be done in two ways. We can take the texture coordinates at the vertices in the view space to be the texture coordinates at the vertices transformed to the screen space interpolate those values to find the texture coordinates at the site of the pixel. For reasons that will be clarified, this method we refer to as being "inconsistent with the polygon interpolation". The result of such an inconsistent interpolation is shown in the middle picture in Figure 6-1 Affine and perspective correct texture maps. The other alternative is to interpolate the texture coordinates in a manner that is consistent with the polygon interpolation. The result of this is shown in the last picture in the Figure.

Notice that if the polygon is orthogonal to the viewing direction, the rendering looks correct irrespective of which interpolation is employed, as shown in the first picture in the Figure. Clearly the result we seek is the one from the interpolation of texture coordinates that is consistent with the polygon interpolation.



Polygon orthogonal to the viewing direction – texture interpolation inconsistency does not matter and no artifacts observed.

Polygon oblique to viewing direction & texture interpolation *inconsistent* with the polygon interpolation - using affine interpolation.

Polygon oblique to viewing direction & texture interpolation *consistent* with the polygon interpolation – using perspective correct interpolation.

Figure 8-13 Affine and perspective correct texture maps

Let us represent any function that needs to be linearly interpolated along the primitive, including the texture coordinates *u* or *v*, byϕ. Say $({}^s x, {}^s y)$ is an interpolated point on the screen space triangle. We need ${}^s\phi({}^s x, {}^s y)$. One approach is to linearly interpolate ϕ in screen space as:

A 6.3

$${}^s\phi = \alpha\ {}^s x + \beta\ {}^s y + \gamma$$

However such an interpolation would lead to wrong results (2). This is because the function ϕ is meant to be interpolated linearly along the primitive in the coordinate system in which the primitive is specified. As it happens, all coordinate systems in the pipeline till we hit the perspective division are affine with each other (that is, we can transform back and forth between those coordinate systems through affine transformations) and hence interpolating ϕ in any one of those coordinate systems linearly is equivalent. However the perspective divide takes the quantities to a space that is not affine with the rest of the coordinate spaces before. So the results of interpolating the quantity after the perspective division are not the same as doing so before the perspective division.

Thus there is a conceptual division in the rendering pipeline between the different coordinate systems used – those which are encountered before the perspective division, which we refer to as *object affine spaces,* and those after the perspective division which we refer to as *screen affine spaces* (2). This grouping is shown in Figure 6-2.

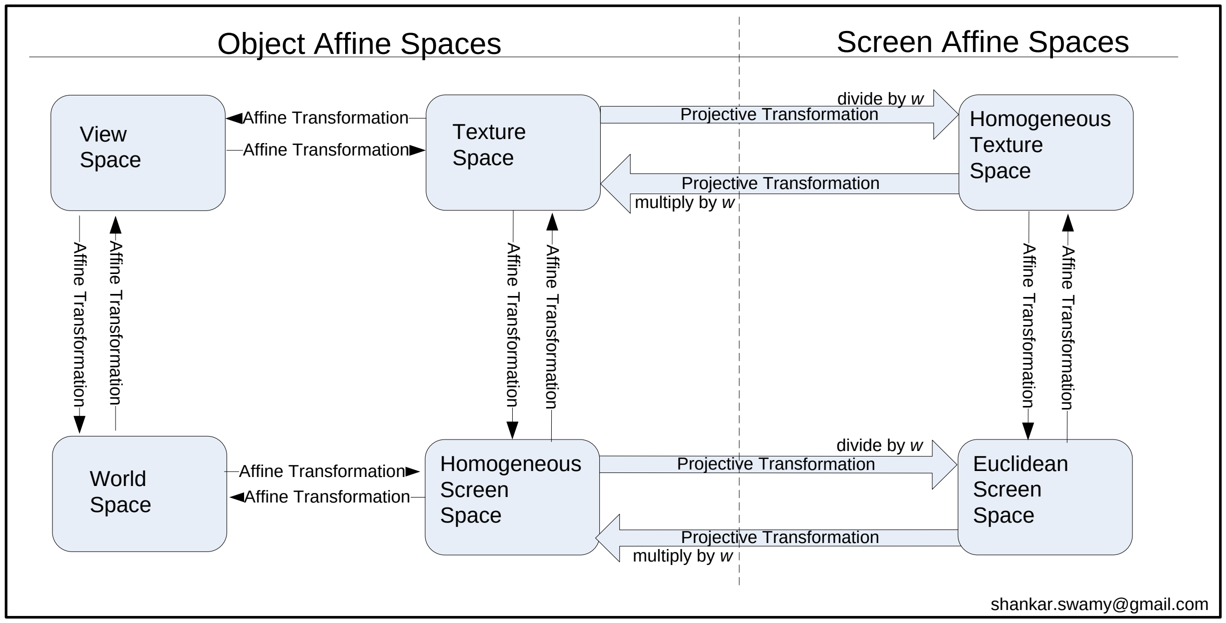


Figure 8-14 Object affine and screen affine coordinate systems in the rendering pipeline

Given that quantities ϕ such as texture coordinates are meant to be interpolated in affine spaces, but the coordinates $({}^{s}x,\ {}^{s}y)$ at which we need to interpolate ϕ are known to us in the screen space creates an issue of implementation complexity. Essentially we need to find ϕ given by

A 6.4

$$\phi = a\ {}^{v}x + b\ {}^{v}y + c\,{}^{v}z$$

for $({}^{v}x,\ {}^{v}y,\ {}^{v}z)$ that corresponds to screen space coordinates $({}^{s}x,\ {}^{s}y)$. The apparent problem here is that we do not know the view space coordinates $({}^{v}x,\ {}^{v}y,\ {}^{v}z)$ corresponding to $({}^{s}x,\ {}^{s}y)$.

However the homogeneous representation of the view space coordinates is such that

A 6.5

$$({}^{s}x,\ {}^{s}y) \equiv \left(\frac{{}^{hs}x}{{}^{hs}w}, \frac{{}^{hs}y}{{}^{hs}w}\right)$$

where $({}^{hs}x,\ {}^{hs}y,\ {}^{hs}w)$ are the homogeneous screen space coordinates. If we look at how the view space coordinates are projected to the screen space via perspective projection, the coordinate systems in the pipeline are set up such that the projection is essentially a direct divide by the z-coordinate:

A 6.6

$$({}^{s}x,\ {}^{s}y) \equiv \left(\frac{{}^{v}x}{{}^{v}z}, \frac{{}^{v}y}{{}^{v}z}\right)$$

From equations A 6.5 and A 6.6 we notice that wherever we use ${}^{v}x,\ {}^{v}y$ and ${}^{v}z$ in the RHS of equation A 6.4, we can instead use ${}^{hs}x,\ {}^{hs}y$ and ${}^{hs}w$ and still achieve the same effect as linear interpolation in the view space. We rewrite equation A 6.4 to get:

A 6.7

$$\phi = a\ {}^{hs}x + b\ {}^{hs}y + c\,{}^{hs}w$$

In equation A 6.7, the constants a, b and c are the same as those in equation A 6.4. What this means is that we can any function that is meant to be linearly interpolated in the view space can be so interpolated in the homogeneous screen space with the same effect. Dividing the equation A 6.7 throughout by w, we get the linearly interpolated value of ϕ in the screen space.

$$\frac{\phi}{w} = a \quad {}^{s} x + b \quad {}^{s} y + c \qquad \text{A 6.8}$$

Equation A 6.8 says that linear interpolation of the quantity $\frac{\phi}{w}$ in the 2D screen space is equivalent to doing so in the view space. Since ϕ is any arbitrary function, this should hold for the constant function $\phi = 1$ and hence we have

$$\frac{1}{w} = a \quad {}^{s} x + b \quad {}^{s} y + c \qquad \text{A 6.9}$$

We can use equations A 6.8 and A 6.9 to interpolate $\frac{\phi}{w}$ and $\frac{1}{w}$ independently and arrive at the interpolation of ϕ (8), (2).

In Figure 6-1, the first picture shows the case where the perspective correct interpolation does not matter. As a matter of implementation, that cases have an unique signature that all the vertices of the primitive (all three of them for a triangle primitive, for example) have the same value of w. This signature helps avoid redundant perspective correct interpolation overheads in the implementation.

Interpolating this way is consistent with geometric transformations used in interpolating the primitive across the pixels during the rasterization (2). The results of texture coordinates interpolated in such a manner is shown in Figure 6-1 in the picture on the extreme right.

The downside to always using this interpolation of course is that it requires a division at each pixel. Traditionally this has been a significant overhead and hence the architectures tended to interpolate only the texture coordinates in a consistent manner.

Olano and Greer (8) have implemented a scan conversion method on Pixel-Planes 5 with all parameters interpolated across the triangle perspective correct. Olano and Greer used a method of scan conversion that completely eliminated clipping and that method required that the interpolation of all parameters across the triangle be perspective correct.

But inconsistently interpolated texture coordinates probably produce the most visible artifacts in rendering. Colors, normals, positions are also susceptible to such errors; but in those cases, the results are often not as conspicuous. When these quantities are used for calculating visible features like Phong or Goraud shading there are no exact standards to compare against as they are anyway approximations. Most commercial rasterizers tend to interpolate only the texture coordinate to save the additional divisions per pixel needed for each parameter.