# Perspective-Correct Interpolation

Kok-Lim Low

Department of Computer Science
University of North Carolina at Chapel Hill
Email: lowk@cs.unc.edu

March 12, 2002

## 1 INTRODUCTION

We will derive (and prove) a method to achieve *perspective-correct interpolation* by linear interpolation in the *screen space*. During *rasterization* of *linear graphics primitives*, such as lines and polygons, straightforward screen-space linear interpolation of vertex *attributes* generally does not produce perspective correct results.

In traditional raster graphics, attributes, such as colors, texture coordinates and normal vectors, are usually associated with the vertices of the graphics primitives [1, 2]. In 3D space, the *value* of each attribute varies *linearly* across each graphics primitive. However, this linear variation of attribute values in 3D space does not translate into similar linear variation in the screen space after the 3D vertices have been projected onto a 2D image plane (or the screen) [†] by a perspective projection. Therefore, if we apply straightforward linear interpolation to these attribute values in the screen space, generally, we get incorrect results in the image. Figure 1 shows such an example.

screen space 中，c 为 ab 中点，因此 c 的插值参数为 0.5。c 对应的几何空间中点 C 并不是 AB 中点，因此若直接使用 screen space 中 c 的插值参数对 AB 插值得到 C，则无法得到正确的插值结果。
这里 C 的插值可以指代任意三维空间的属性插值

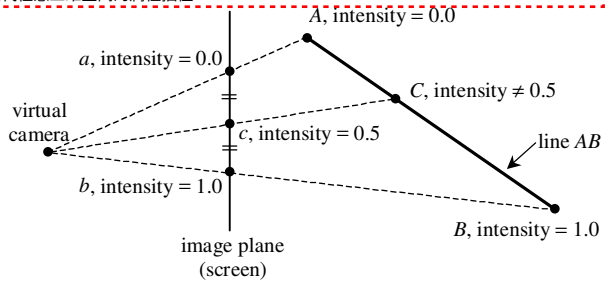


***Figure 1**: Straightforward linear interpolation of attribute values in the screen space (or in the image plane) does not always produce perspective-correct results.*

For easier illustration, Figure 1 only shows a line in 2D space being projected onto a 1D image plane, but the same argument can be applied to a 3D linear geometric primitive projected onto a 2D image plane. In the diagram, vertices *A* and *B* of a line *AB* in 2D space are projected onto points *a* and *b*, respectively, in a 1D image plane. The attribute values at the vertices are intensity values. At *A*, intensity value is 0.0, and at *B*, intensity value is 1.0. It follows that the intensity values at *a* and *b* are 0.0 and 1.0, respectively. Suppose *c* is the midpoint between *a* and *b* in the image plane. If we linearly interpolate in the image plane (or in the screen space) the intensity values at *a* and *b*, then the intensity value at *c* is 0.5. However, if we "unproject" the point *c* onto a point *C* on the line *AB*, we can see that *C* is not necessary the midpoint between *A* and *B*. Since the intensity value varies linearly from *A* to *B* (in 3D space), the intensity value at *C* should not be 0.5 if it is not the midpoint between *A* and *B*.

In spite of this, it is still possible to obtain perspective correct results by linearly interpolating in the screen space. This can be done by interpolating the values of some functions of the attributes, instead of interpolating the attributes directly. Each interpolated result is then transformed by another function (the inverse function) to get the final attribute value at the desired point in the screen space. You will see that these functions make use of the *z*-values of the vertices.

## 2 INTERPOLATING *z*-VALUES

Before a perspective projection is done to project the 3D vertices of a primitive onto a 2D image plane, many graphics rendering systems assume that the *virtual camera* is already located at the origin of the 3D space, looking in the $-z$ or $+z$ direction. This defines a coordinate system called the *camera coordinate system*. The fixed viewing direction of the camera also allows us to always have an image plane perpendicular to the *z*-axis, which significantly simplifies the perspective projection computation. Without loss of generality, we will assume that the camera is looking in the $+z$ direction, and the image plane is at a distance of *d* in front of the camera (see Figure 2).

Many raster graphics rendering systems use *z*-buffer [1, 2] to perform hidden-surface removal. This requires that at every pixel on the screen that a primitive is projected onto, the *z*-coordinate (*z*-value) of the corresponding 3D point on the primitive must be known. However, because only the primitive's vertices are actually projected, their corresponding 2D image points in the screen space are the only places in the screen space where *z*-values are known. For faster rasterization, we would want to derive the *z*-values at other pixels using the known *z*-values at the vertices' image points.

The *z*-values can be treated as an attribute whose values vary linearly across a 3D linear primitive. Like the example we have seen in Figure 1, *straightforward* linear interpolation of *z*-values in the screen space does not always produce perspective-correct results. However, we will see in the followings that we can actually linearly interpolate the reciprocals of the *z*-values to achieve the correct results.

Similar to Figure 1, Figure 2 shows a line in a 2D camera coordinate system being projected onto a 1D image plane. The caption explains the symbols that we will use in the formula derivation. In the figure, *s* is the interpolation parameter in the image plane, and *t* is the interpolation parameter on the primitive.

[†] Technically, an image plane is not the same as the screen space. A 2D translation and a 2D scale are usually required to map a rectangular region of the image plane to a region in the screen space. Since linear interpolation in the image plane works similarly in the screen space, we will not try to distinguish the two in the arguments.

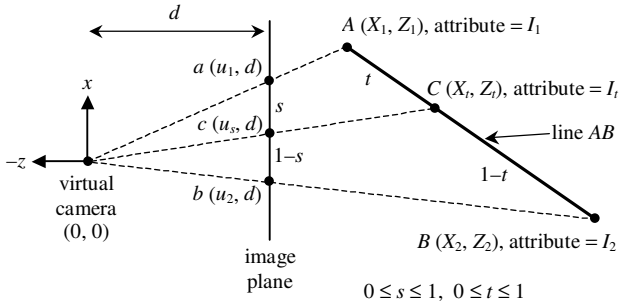


***Figure 2**: The virtual camera is looking in the +z direction in the camera coordinate system. The image plane is at a distance of d in front of the camera. A, B and C are points on the primitive with attribute values $I_1$, $I_2$ and $I_t$ respectively, and their images on the image plane are a, b and c, respectively. s and t are parameters used for linear interpolation.*

Our objective is to derive formula to correctly interpolate, in the screen space, the $z$-values. The same derivation can be directly applied to the case of a 3D linear primitive projected onto a 2D image plane.

Referring to Figure 2, by similar triangles, we have

$$\frac{X_1}{Z_1} = \frac{u_1}{d} \;\Rightarrow\; X_1 = \frac{u_1 Z_1}{d}, \tag{1}$$

$$\frac{X_2}{Z_2} = \frac{u_2}{d} \;\Rightarrow\; X_2 = \frac{u_2 Z_2}{d}, \tag{2}$$

$$\frac{X_t}{Z_t} = \frac{u_s}{d} \;\Rightarrow\; Z_t = \frac{dX_t}{u_s}. \tag{3}$$

By linearly interpolating in the image plane (or screen space), we have

$$u_s = u_1 + s(u_2 - u_1). \tag{4}$$

By linearly interpolating across the primitive in the camera coordinate system, we have

$$X_t = X_1 + t(X_2 - X_1), \tag{5}$$

$$Z_t = Z_1 + t(Z_2 - Z_1), \tag{6}$$

Substituting (4) and (5) into (3),

$$Z_t = \frac{d\left(X_1 + t\left(X_2 - X_1\right)\right)}{u_1 + s(u_2 - u_1)}. \tag{7}$$

Substituting (1) and (2) into (7),

$$\begin{aligned} Z_t &= \frac{d\left(\dfrac{u_1 Z_1}{d} + t\left(\dfrac{u_2 Z_2}{d} - \dfrac{u_1 Z_1}{d}\right)\right)}{u_1 + s(u_2 - u_1)} \\ &= \frac{u_1 Z_1 + t(u_2 Z_2 - u_1 Z_1)}{u_1 + s(u_2 - u_1)}. \end{aligned} \tag{8}$$

Substituting (6) into (8),

$$Z_1 + t(Z_2 - Z_1) = \frac{u_1 Z_1 + t(u_2 Z_2 - u_1 Z_1)}{u_1 + s(u_2 - u_1)}, \tag{9}$$

which can be simplified into

$$t = \frac{sZ_1}{sZ_1 + (1-s)Z_2}. \tag{10}$$

Substituting (10) into (6), we have

$$Z_t = Z_1 + \frac{sZ_1}{sZ_1 + (1-s)Z_2}(Z_2 - Z_1), \tag{11}$$

which can be simplified to

$$Z_t = \frac{1}{\dfrac{1}{Z_1} + s\left(\dfrac{1}{Z_2} - \dfrac{1}{Z_1}\right)}. \tag{12}$$

Equation (12) tells us that the $z$-value at point $c$ in the image plane can be correctly derived by just *linearly interpolating* between $1/Z_1$ and $1/Z_2$, and then compute the reciprocal of the interpolated result. For $z$-buffer purpose, the final reciprocal need not even be computed, because all we need is to reverse the comparison operation during $z$-value comparison.

## 3 INTERPOLATING ATTRIBUTE VALUES

Here, we want to derive formula to correctly interpolate, in the screen space, the other attribute values.

Refer to Figure 2 again. By linearly interpolating the attribute values across the primitive in the camera coordinate system, we get

$$I_t = I_1 + t(I_2 - I_1). \tag{13}$$

Substituting (10) into (13), we have

$$I_t = I_1 + \frac{sZ_1}{sZ_1 + (1-s)Z_2}(I_2 - I_1), \tag{14}$$

which can be rearranged into

$$I_t = \left(\frac{I_1}{Z_1} + s\left(\frac{I_2}{Z_2} - \frac{I_1}{Z_1}\right)\right) \Big/ \left(\frac{1}{Z_1} + s\left(\frac{1}{Z_2} - \frac{1}{Z_1}\right)\right). \tag{15}$$

From (12), we can see that the denominator in (15) is just $1/Z_t$. Therefore,

$$I_t = \left(\frac{I_1}{Z_1} + s\left(\frac{I_2}{Z_2} - \frac{I_1}{Z_1}\right)\right) \Big/ \frac{1}{Z_t}. \tag{16}$$

What (16) means is that the attribute value at point $c$ in the image plane can be correctly derived by just *linearly interpolating* between $I_1/Z_1$ and $I_2/Z_2$, and then divide the interpolated result by $1/Z_t$, which itself can be derived by linear interpolation in the screen space as shown in (12).

## REFERENCES


[1] James D. Foley, Andries van Dam, Steven K. Feiner and John F. Hughes. *Computer Graphics: Principles and Practice, Second Edition.* Addison-Wesley, 1990.

[2] Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner (OpenGL Architecture Review Board). *OpenGL Programming Guide, Third Edition: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley, 1999.