Homogeneous Rasterization

Matthias Zwicker

October 9, 2008

1 Introduction

This document gives a short summary of the homogeneous rasterization technique proposed by Olano and Greer [2]. The main advantage of this algorithm is that it does not require a homogeneous division at the triangle vertices to compute the interpolation parameters for vertex attributes. The algorithm is elegant and simple to implement.

The main idea of homogeneous rasterization is to first look at the problem of interpolating arbitrary linear functions across triangles. Once we have understood this problem, we can implement both *perspective correct interpolation* and *rasterization* using this idea. Rasterization is achieved by defining specific *edge functions* that allow us to determine if a pixel is inside or outside a triangle.

2 Interpolating Linear Functions on Triangles

Assume that you have transformed triangle vertices such that the projection to pixel coordinates is simply achieved by dividing by the homogeneous coordinate. Let us denote these transformed vertex coordinates by (x, y, z, w), and pixel coordinates after homogeneous division by $(x/w, y/w)$.

The main observation is that any linear function on the triangle, *before homogeneous division*, can be written in the form $u(x, y, w) = a_u x + b_u y + c_u w$. We call the coefficients a_u, b_u, c_u the *interpolation parameters* for the function u. We show how to determine them below. Note that we could also have used the z coordinate here. You will see why we use the w coordinate instead. The coordinates (x, y, w) can also be interpreted as *2D homogeneous coordinates.* Our goal is now to express the interpolated function u in terms of pixel coordinates $(x/w, y/w)$, i.e., we want to compute the function $u(x/w, y/w)$. This setup is illustrated in Figure 1.

Any linear function on a triangle is given by its values at the triangle vertices. We denote the coordinates of the triangle vertices by $(x_i, y_i, w_i), i = 0, 1, 2$, and the function values by $u_i, i = 0, 1, 2$. We solve for the interpolation parameters a_u, b_u, c_u by formulating constraints that the values at the vertices are

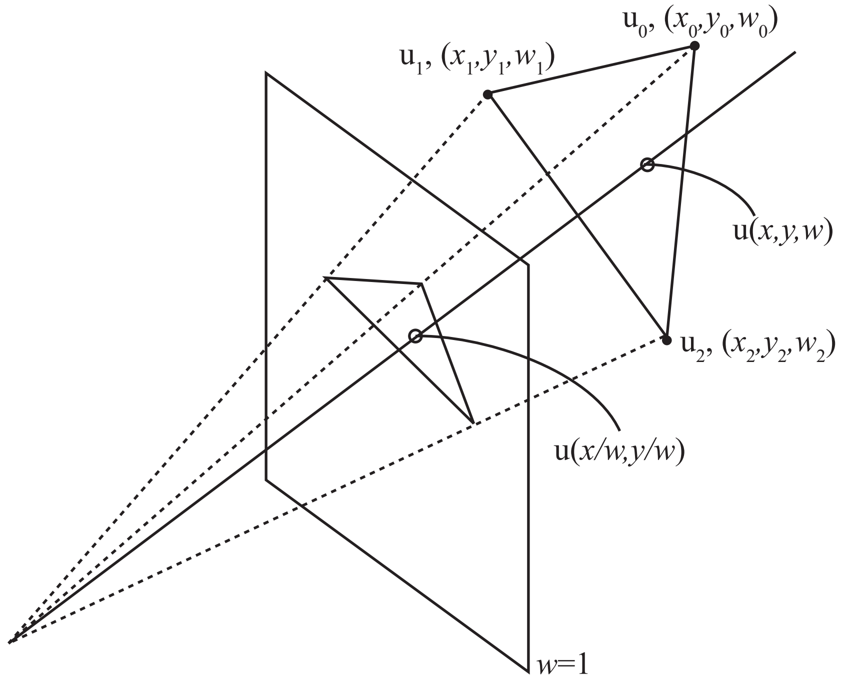

Figure 1: The setup for interpolating linear functions on triangles parameterized by pixel coordinates $(x/w, y/w)$.

interpolated. This leads to a system of equations,

$$\begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix} \begin{bmatrix} a_u \\ b_u \\ c_u \end{bmatrix} = M \begin{bmatrix} a_u \\ b_u \\ c_u \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix}.$$

The interpolation parameters are found by inverting the matrix,

$$\begin{bmatrix} a_u \\ b_u \\ c_u \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & w_0 \\ x_1 & y_1 & w_1 \\ x_2 & y_2 & w_2 \end{bmatrix}^{-1} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} = M^{-1} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix}.$$

We first observe that we can easily interpolate the function $1/w$ given pixel coordinates $x/w, y/w$. To achieve this, we choose a constant function on the triangle, i.e., $u_i = 1, i = 0, 1, 2$. We solve for its set of coefficients denoted by a_1, b_1, c_1 as shown above. This means we have $1 \equiv a_1 x + b_1 y + c_1 w$ for points on the plane defined by the triangle, or $1/w = a_1 x/w + b_1 y/w + c_1$. To make clear that this means "given pixel coordinates, compute $1/w$", we write this explicitly as a function $f(x/w, y/w)$ of pixel coordinates, i.e., $f(x/w, y/w) = a_1 x/w + b_1 y/w + c_1 = 1/w$.

We now show how to interpolate an arbitrary function u, which may represent color, texture coordinates or any other vertex attribute. We denote the coefficients by a_u, b_u, c_u and solve for them as shown above. Homogeneous division yields $u/w = a_u x/w + b_u y/w + c_u$. Or as above, $g(x/w, y/w) = a_u x/w + b_u y/w + c_u = u/w$.

So far we have shown how to interpolate both $1/w$ and u/w in terms of pixel coordinates $x/w, y/w$. We now get the desired function $u(x/w, y/w) = g(x/w, y/w)/f(x/w, y/w) = u/w \cdot 1/w$! Therefore, we have already solved the perspective correct interpolation problem.

3 Edge Functions

Our next goal is to define linear functions on triangles that allow us to determine whether a pixel is inside or outside the triangle. We define an edge function as a linear function that is zero along one edge, and positive on the opposite vertex. It is clear that this function is positive inside the triangle, and negative in the half-plane that lies outside the triangle across the edge. A point lies in the triangle if all edge functions are positive. Therefore, we can rasterize a triangle by simply interpolating all three edge functions at each pixel. If they are all positive, the pixel is inside the triangle.

One can also omit the division by $1/w$ when evaluating the edge functions, since we are only interested in the sign of the functions. We denote the edge functions by $\alpha = a_\alpha x + b_\alpha y + c_\alpha$, $\beta = a_\beta x + b_\beta y + c_\beta$, and $\gamma = a_\gamma x + b_\gamma y + c_\gamma$. If $0 < \alpha/w, \beta/w, \gamma/w$, the pixel is inside the triangle and the triangle lies in front of the eye (w is positive). If $0 > \alpha/w, \beta/w, \gamma/w$, the pixel is inside the triangle, but the triangle lies behind the eye (w is negative). Otherwise, the pixel is outside the triangle.

Note that because the edge functions are zero at two vertices and one at the third vertex, the coefficients of each edge function correspond to one column of M^{-1}.

4 Implementation

The homogeneous rasterization algorithm requires the inversion of a 3×3 matrix. This involves the computation of the matrix determinant, which provides some useful information that can be exploited. If the determinant is negative the triangle is back facing. If the determinant is zero, the triangle has zero area in homogeneous space, or it is viewed edge on. In both cases it does not need to be drawn.

Key to the efficiency of the algorithm is a fast *binning* technique. Binning is the process of determining for which pixels the edge functions should be evaluated in the first place. A simple and efficient way is to use axis aligned bounding boxes for the triangles. Bounding boxes can be computed without homogeneous division using the method proposed by Blinn [1]. Multi-level bounding boxes can further speed up the process.

References

[1] James Blinn. Jim blinn's corner: Calculating screen coverage. *IEEE Computer Graphics & Applications*, 16(3), 1996.

[2] Marc Olano and Trey Greer. Triangle scan conversion using 2d homogeneous coordinates. In *HWWS '97: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 89–95, 1997.