Distance from Point to Orthogonal Frustum

David Eberly
Magic Software, Inc.
http://www.magic-software.com

Created: May 8, 2000

1 Introduction

The algorithm for computing the distance from a point to an orthogonal frustum is based on determining the Voronoi regions for the faces, edges, and vertices of the frustum. The region containing the point is computed. The nearest point on the frustum in that region is also computed. From this the distance can be calculated. The concepts are illustrated first in 2D. The 3D case is slightly more difficult to visualize, but is a straightforward generalization of the Voronoi idea.

2 2D Distance

The orthogonal frustum has origin $\vec{E}$, unit–length direction vector $\vec{D}$, and perpendicular unit–length vector $\vec{L}$. The near line has normal $\vec{D}$ and contains the point $\vec{E} + n\vec{D}$ for some $n > 0$. The far line has normal $\vec{D}$ and contains the point $\vec{E} + f\vec{D}$ for some $f > n$. The four vertices of the frustum are $\vec{E} + n\vec{D} \pm \ell\vec{L}$ for some $\ell > 0$, and $\vec{E} + (f/n)(n\vec{D} \pm \ell\vec{L})$. Let $\vec{P}$ be the point whose distance to the frustum is required. The point can be written in the frustum coordinate system as

$$\vec{P} = \vec{E} + x_0\vec{L} + x_1\vec{D},$$

so $x_0 = \vec{L} \cdot (\vec{P} - \vec{E})$ and $x_1 = \vec{D} \cdot (\vec{P} - \vec{E})$. It is sufficient to demonstrate the construction for $x_0 \geq 0$. For if $x_0 < 0$, a reflection can be made by changing sign on x_0, the closest point can be calculated, then a reflection on that point yields the closest point to the original. Figure 1 shows the portion of the frustum in the first quadrant.

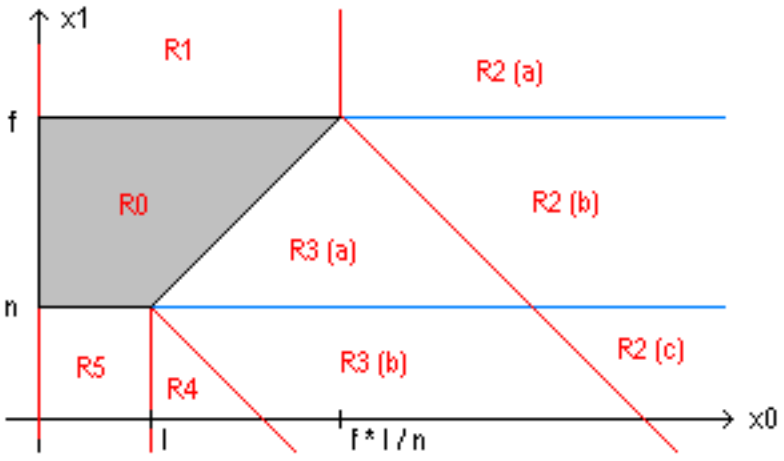

Figure 1. Portion of frustum in first quadrant.

The Voronoi regions are marked in red. Region R_0 contains those points inside the frustum. Region R_1 contains those points closest to the top edge of the frustum. Region R_2 contains those points closest to the vertex $(f\ell/n, f)$ of the frustum. That region is split into three subregions based on $\vec{D}$ component being larger than f, between n and f, or smaller than n. Region R_3 contains those points closest to the slanted edge of the frustum. That region is split into two subregions based on $\vec{D}$ component being between n and f or smaller than n. Region R_4 contains those points closest to the vertex (ℓ, n) of the frustum. Finally, region R_5 contains those points closest to the bottom edge of the frustum.

The pseudocode for determining the Voronoi region for (x_0, x_1) is given below.

```
if ( x1 >= f )
{
    if ( x0 <= f*l/n )
        point in R1;
    else
        point in R2a;
}
else if ( x1 >= n )
{
    t = Dot((n,-l),(x0,x1));
    if ( t <= 0 )
        point in R0;
    else
    {
        t = Dot((l,n),(x0,x1));
        if ( t <= Dot((l,n),(f*l/n,f)) )
            point in R3a;
        else
            point in R2b;
```

```
        }
    }
    else
    {
        if ( x0 <= l )
            point in R5;
        else
        {
            t = Dot((l,n),(x0,x1));
            if ( t <= Dot((l,n),(l,n)) )
                point in R4;
            else if ( t <= Dot((l,n),(f*l/n,f)) )
                point in R3b;
            else
                point in R2c;
        }
    }
```

The closest point to (x_0, x_1) in R_1 is (x_0, f). The closest point in R_2 is $(f\ell/n, f)$. The closest point in R_4 is (ℓ, n). The closest point in R_5 is (x_0, n). Region R_3 requires projecting out the $(n, -\ell)$ component from (x_0, x_1). The closest point is $(x_0, x_1) - [(nx_0 - \ell x_1)/(\ell^2 + n^2)](n, -\ell)$.

3 3D Distance

The orthogonal view frustum has origin $\vec{E}$. Its coordinate axes are determined by left vector $\vec{L}$, up vector $\vec{U}$, and direction vector $\vec{D}$. The vectors in that order form a right–handed orthonormal system. The extent of the frustum in the $\vec{D}$ direction is $[n, f]$ where $0 < n < f$. The four corners of the frustum in the near plane are $\vec{E} \pm \ell\vec{L} \pm \mu\vec{U} + n\vec{D}$. The four corners of the frustum in the far plane are $\vec{E} + (f/n)(\pm\ell\vec{L} \pm \mu\vec{U} + n\vec{D})$.

Let $\vec{P}$ be the point whose distance to the frustum is required. The point can be written in the frustum coordinate system as

$$\vec{P} = \vec{E} + x_0\vec{L} + x_1\vec{U} + x_x\vec{D},$$

so $x_0 = \vec{L} \cdot (\vec{P} - \vec{E})$, $x_1 = \vec{U} \cdot (\vec{P} - \vec{E})$, and $x_2 = \vec{D} \cdot (\vec{P} - \vec{E})$. It is sufficient to demonstrate the construction for $x_0 \geq 0$ and $x_1 \geq 0$. The idea is the same as in the 2D case, reflect the x_0 and x_1 components, find the closest point, then reflect its x_0 and x_1 components back to the original quadrant.

The naming conventions for the frustum components are N for near, F for far, U for up, and L for left. The top face of the frustum is labeled the F–face. It has two edges, the UF–edge that is in the direction of $\vec{L}$ and the LF–edge that is in the direction of $\vec{U}$. It also has a vertex, the LUF–vertex at $(f\ell/n, f\mu/n, f)$. The bottom face of the frustum is labeled the N–face. It has two edges, the UN–edge that is in the direction of $\vec{L}$ and the LN–ege that is in the direction of $\vec{U}$. It also has a vertex, the LUN–vertex at (ℓ, μ, n). The remaining two faces are the L–face whose normal is $(n, 0, -\ell)$ and the U–face whose normal is $(0, n, -\mu)$. Finally there is the LU–edge that is shared by the L–face and the U–face. Figure 2 illustrates the Voronoi region boundaries in red. The blue lines indicate the near and far planes that split some of the Voronoi regions.

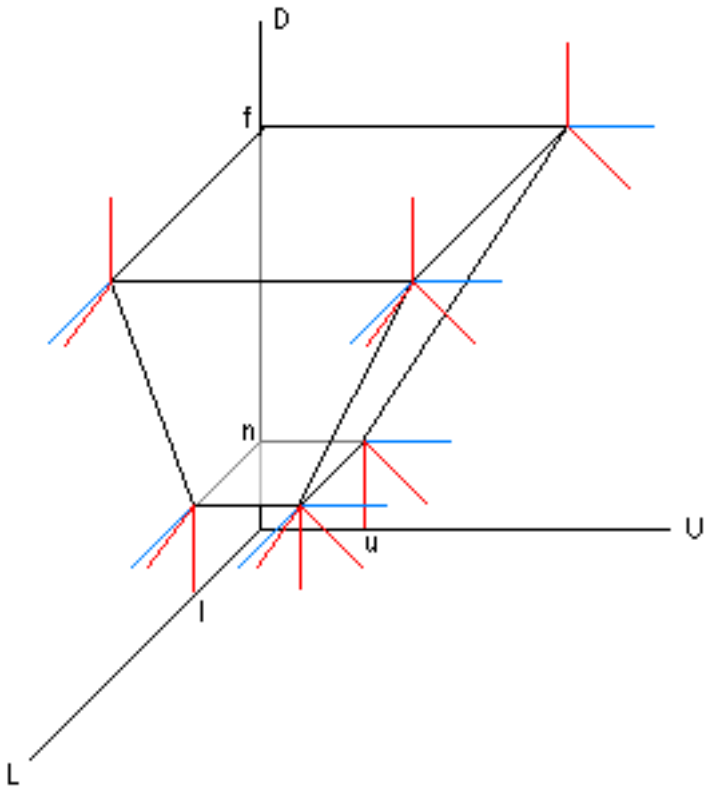

Figure 2. Portion of frustum in first octant.

The pseudocode for determining the Voronoi region for $(x_0, x_1, x_2))$ is given below.

```
if ( x2 >= f )
{
    if ( x0 <= f*l/n )
    {
        if ( x1 <= f*u/n )
            F-face is closest;
        else
            UF-edge is closest;
    }
    else
    {
        if ( x1 <= f*u/n )
            LF-edge is closest;
        else
            LUF-vertex is closest;
    }
}
else if ( x2 <= n )
```

```
{
    if ( x0 <= l )
    {
        if ( x1 <= u )
            N-face is closest;
        else
        {
            t = u*x1 + n*x2;
            if ( t >= (f/n)*(u*u+n*n) )
                UF-edge is closest;
            else if ( t >= u*u+n*n )
                U-face is closest;
            else
                UN-edge is closest;
        }
    }
    else
    {
        if ( x1 <= u )
        {
            t = l*x0 + n*x2;
            if ( t >= (f/n)*(l*l+n*n) )
                LF-edge is closest;
            else if ( t >= l*l+n*n )
                L-face is closest;
            else
                LN-edge is closest;
        }
        else
        {
            r = l*x0 + u*x1 + n*x2;
            s = u*r - (l*l+u*u+n*n)*x1;
            if ( s >= 0.0 )
            {
                t = l*x0 + n*x2;
                if ( t >= (f/n)*(l*l+n*n) )
                    LF-edge is closest;
                else if ( t >= l*l+n*n )
                    L-face is closest;
                else
                    LN-edge is closest;
            }
            else
            {
                s = l*r - (l*l+u*u+n*n)*x0;
                if ( s >= 0.0 )
                {
                    t = u*x1 + n*x2;
```

```
                    if ( t >= (f/n)*(u*u+n*n) )
                        UF-edge is closest;
                    else if ( t >= u*u+n*n )
                        U-face is closest;
                    else
                        UN-edge is closest;
                }
                else
                {
                    if ( r >= (f/n)(l*l+u*u+n*n) )
                        LUF-vertex is closest;
                    else if ( r >= l*l+u*u+n*n )
                        LU-edge is closest;
                    else
                        LUN-vertex is closest;
                }
            }
        }
    }
}
else
{
    s = n*x0 - l*x2;
    t = n*x1 - u*x2;
    if ( s <= 0 )
    {
        if ( t <= 0 )
            point inside frustum;
        else
        {
            t = u*x1 + n*x2;
            if ( t >= (f/n)*(u*u+n*n) )
                UF-edge is closest;
            else
                U-face is closest;
        }
    }
    else
    {
        if ( t <= 0 )
        {
            t = l*x0 + n*x2;
            if ( t >= (f/n)*(l*l+n*n) )
                LF-edge is closest;
            else
                L-face is closest;
        }
        else
```

```
        {
            r = l*x0 + u*x1 + n*x2;
            s = u*r - (l*l+u*u+n*n)*x1;
            if ( s >= 0 )
            {
                t = l*x0 + n*x2;
                if ( t >= (f/n)*(l*l+n*n) )
                    LF-edge is closest;
                else
                    L-face is closest;
            }
            else
            {
                t = l*r - (l*l+u*u+n*n)*x0;
                if ( t >= 0 )
                {
                    t = u*x1 + n*x2;
                    if ( t >= (f/n)*(u*u+n*n) )
                        UF-edge is closest;
                    else
                        U-face is closest;
                }
                else
                {
                    if ( r >= l*l+u*u+n*n )
                        LUF-vertex is closest;
                    else
                        LU-edge is closest;
                }
            }
        }
    }
}
```

The closest point in each region is obtained by projection onto that component.