GPU Ray Casting of Virtual Globes Supplement

Patrick Cozzi\textsuperscript{*} \hspace{1cm} Frank Stoner\textsuperscript{†}

Analytical Graphics, Inc.

\textbf{1 Introduction}

This supplement derives the equations for scaled space operations used to optimize GPU ray casting of ellipsoids. First, computation of the viewport-aligned ellipsoid bounding polygon is detailed. This polygon is used to reduce ray misses, and thus reduce fill rate. Next, the ray/ellipsoid test in the scaled space is derived. Finally, sample code for computing the bounding polygon is provided in C# and sample code for the ray/ellipsoid test is provided in GLSL.

\section{The Front Face of the Ellipsoid}

A point on the surface of an ellipsoid with semiaxis lengths (a, b, c) and with its center and principal axes coincident with a Cartesian reference frame satisfies the scalar equation

\[ \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1 \]

This equation can be expressed in matrix form as

\[ \mathbf{s}^T \mathbf{D}^T \mathbf{D} \mathbf{s} = 1 \]  \hspace{1cm} (1)

where

\[ \mathbf{s} = \begin{pmatrix} x/a \\ y/b \\ z/c \end{pmatrix} \quad \mathbf{D} = \begin{pmatrix} 1/a \\ 0 \\ 0 \\ 0 \\ 1/b \\ 0 \\ 0 \\ 0 \\ 1/c \end{pmatrix} \]

From a point, \( \mathbf{p} \), which is exterior to the ellipsoid, the front face of the ellipsoid is bounded by the set of points which satisfy the tangency condition

\[ \mathbf{v}^T \mathbf{p} = 0 \]  \hspace{1cm} (2)

where \( \mathbf{v} \) is the vector from \( \mathbf{p} \) to the point of tangency \( \mathbf{s} \), and \( \mathbf{v} \) is the gradient of the ellipsoid surface at \( \mathbf{s} \).

\[ \mathbf{v} = 2 \mathbf{D}^T \mathbf{D} \mathbf{s} \]

Equation (2) can be expressed in terms of \( \mathbf{p} \) and \( \mathbf{s} \) as

\[ 2 (\mathbf{s} - \mathbf{p})^T \mathbf{D}^T \mathbf{D} \mathbf{s} = 0 \]

Expanding and simplifying using equation (1), this expression becomes

\[ \mathbf{p}^T \mathbf{D}^T \mathbf{D} \mathbf{s} = 1 \]  \hspace{1cm} (3)

Together, equations (1) and (3) provide the two conditions that constrain the points comprising the boundary curve of the front face of the ellipsoid from the vantage point \( \mathbf{p} \); they must lie on the surface of the ellipsoid from equation (1) and they must satisfy the modified tangency relationship from equation (3).

\subsection{The Transformed Representation of the Front Face}

The matrix \( \mathbf{D} \) represents a simple linear, scaling transformation. Defining an alternate pair of vectors

\[ \mathbf{q} = \mathbf{D} \mathbf{p}, \quad \mathbf{q} = \mathbf{D} \mathbf{s} \]

equations (1) and (3) become

\[ \mathbf{r}^T \mathbf{q} = 1 \]  \hspace{1cm} (4)

\[ \mathbf{q}^T \mathbf{r} = 1 \]  \hspace{1cm} (5)

Observe that equation (4) is the relationship which indicates that \( \mathbf{r} \) is appropriately notated as a unit vector. Also observe that equation (5) is a simple projection (dot product) operation and can also be expressed in terms of the vector magnitudes (where \( r = 1 \)) and included angle, \( \rho \), as

\[ \mathbf{q}^T \mathbf{r} = q \cos(\rho) \]  \hspace{1cm} (6)

Together, equations (5) and (6) indicate that

\[ \cos(\rho) = \frac{1}{q} \]  \hspace{1cm} (7)
In this transformed representation (Figure 1a), the two conditions that constrain the points comprising the boundary curve of the front face of the ellipsoid require that the points lie on the surface of a unit sphere from equation (4) and that they have a fixed projection onto the direction along \( \hat{q} \) from equation (7). Equivalently, the boundary curve is formed from the intersection of the surface of the unit sphere and the plane perpendicular to and at a fixed distance \( \frac{1}{q} \) along the unit vector \( \hat{q} \). Moreover, the boundary curve in this transformed representation is a circle with radius \( \sqrt{1 - \frac{1}{q^2}} \).

3 Determining a Bounding Shape for the Front Face

A simple yet conservative approximation to the circular boundary curve in the transformed space is an N-sided regular polygon with sides tangent to the circle (Figure 1b). The \( i \)th vertex of this polygon will have coordinates in the transformed space given by

\[
\mathbf{h}_i = \frac{1}{q^2} q + h (\cos(\alpha_i) \hat{q} + \sin(\alpha_i) \hat{q} \times \hat{q})
\]

where

\[
\alpha_i = i \frac{2\pi}{N} \quad \text{for } i = 0, ..., N - 1
\]

\[
h = \sqrt{1 - \frac{1}{q^2}}
\]

and \( \hat{q} \) is a direction chosen to be perpendicular to \( \hat{q} \). The vertices are transformed to the reference frame of the ellipsoid by

\[
\mathbf{g}_i = \mathbf{D}^{-1} \mathbf{h}_i
\]

where the inverse transformation \( \mathbf{D}^{-1} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} \).

4 Ray Intersection in the Transformed Representation

A vector (or ray) \( \mathbf{d} \) from vantage point \( \mathbf{p} \) in the reference frame of the ellipsoid has a representation \( \mathbf{b} \) and direction \( \mathbf{b} \) in the transformed frame of

\[
\mathbf{b} = \mathbf{D} \mathbf{d} \\
\mathbf{b} = \frac{\mathbf{b}}{\sqrt{\mathbf{b}^\top \mathbf{b}}}
\]

Define the projection (dot product) of \( \hat{q} \) onto this direction to be

\[
t = -\hat{q}^\top \mathbf{b}
\]

Then the geometry of the intersection (Figure 1c) requires that

\[
q^2 = (1 - x^2) + t^2
\]

Solving for \( x \) produces

\[
x = \sqrt{t^2 - w^2}
\]

where \( w^2 = q^2 - 1 \).

The point of intersection on the front face of the sphere is then given by

\[
\mathbf{r} = \mathbf{q} + (t - x) \mathbf{b}
\]

The point of intersection in the reference frame of the ellipsoid is \( \mathbf{s} = \mathbf{D}^{-1} \mathbf{r} \) and the gradient (normal vector) to the ellipsoid surface at the point of intersection is \( \mathbf{n} = 2 \mathbf{D} \mathbf{r} \).

There are a couple of relationships involving \( t \) which are worth noting. First, \( t > 0 \) indicates the transformed direction \( \mathbf{b} \) is pointing back toward the sphere from \( \mathbf{q} \) and could possibly intersect the sphere. Otherwise, no intersection can occur along the positive \( \mathbf{b} \) direction. Second, \( t^2 = w^2 \) indicates that we have the geometry indicated in Figure 1a where \( \mathbf{r} \) is a point of tangency. So, if \( t^2 \geq w^2 \) then \( \mathbf{b} \) will intersect or be tangent to the sphere. Otherwise, the direction \( \mathbf{b} \) will completely miss the sphere.

5 Bounding Polygon Computation Code

The following C# function, `BoundingPolygon()`, computes the vertices of the bounding polygon in the reference frame of the ellipsoid. The input parameter \( q \) is the scaled camera position, pre-computed as \( q = D \cdot p \). The input \( \mathbf{D}^{-1} \) is the diagonal matrix with semiaxis lengths as the diagonal elements. This is used for descaling. The input \( n \) determines the number of sides of the polygon, and should be at least three. \( n \) determines the trade off between CPU/vertex processing and fragment processing; higher values of \( n \) result in a tighter fit polygon, which reduces ray misses at the cost of extra vertex data and CPU/vertex processing.

```csharp
public static Cartesian[] BoundingPolygon(Cartesian q, Matrix3By3 DInverse, int n)
{
    double qMagSquared = q.MagnitudeSquared;
    double qMag = Math.Sqrt(qMagSquared);
    // Compute the orthonormal basis defined
    // by q and its most orthogonal axis.
    Cartesian a1 = q / qMag;
    UnitCartesian r = a1.MostOrthogonalAxis;
    Cartesian a2 = r.Cross(a1).Normalize();
    Cartesian a3 = a1.Cross(a2);
    // Compute the scaling and translation
    // in the orthonormal basis.
    double scaling = Math.Sqrt(1.0 - 1.0 / qMagSquared);
    double trans = 1.0 / qMag;
    // Compute the parameters of the
    // bounding regular convex polygon.
    double piOverN = Math.PI / n;
    double angle = 2.0 * piOverN;
    double r = scaling / Math.Cos(piOverN);
    // Compute the reference vectors used to
    // generate the vertices in the reference
    // frame of the ellipsoid.
    Cartesian xTemp = trans * DInverse * a1;
    Cartesian yTemp = r * DInverse * a2;
    Cartesian zTemp = r * DInverse * a3;
    // Generate the vertices.
    Cartesian[] result = new Cartesian[n];
    for (int i = 0; i < n; ++i)
    {
        double temp = i * angle;
        result[i] = xTemp +
                    Math.Cos(temp) * yTemp +
                    Math.Sin(temp) * zTemp;
    }
}
```
6 Ray/Ellipsoid Intersection Code

The following GLSL function, `RayIntersectEllipsoid()`, intersects a ray with origin \( q \) in the scaled space \( q = D \times p \) and un-normalized direction `rayDirection` with an ellipsoid centered at \((0, 0, 0)\) with radii `radii`. `wMagnitudeSquared` is precomputed as \( q\text{MagnitudeSquared} - 1.0 \), where \( q\text{MagnitudeSquared} = \text{dot}(q, q) \). Note that in the case of intersection, the returned surface normal is not normalized.

```cpp
struct Intersection {
    bool intersects;
    vec3 surfacePoint;
    vec3 surfaceNormal;
};

Intersection RayIntersectEllipsoid(
    vec3 q,
    float wMagnitudeSquared,
    vec3 oneOverRadii,
    vec3 radii,
    vec3 rayDirection
) {
    vec3 bUnit = normalize(
        rayDirection * oneOverRadii);
    float t = -dot(bUnit, q);"}