PRIMITIVES INTERSECTION WITH CONFORMAL 5D GEOMETRY

Eduardo Roa
eduroam@ldc.usb.ve
Víctor Theoktisto
vtheok@usb.ve

Laboratorio de Computación Gráfica e Interacción
Universidad Simón Bolívar, Caracas-VENEZUELA.

Abstract. *Conformal algebra in conformal geometric space allows for undifferentiated algebraic treatment of first class members such as points, vectors, areas (defined by bivectors) and volumes (defined by trivectors). We have derived a novel unified approach for all types of collisions in conformal space, based in a reformulation of euclidean (3D) collision queries mapped to conformal space (5D). The algebraic formulation of collisions/intersections in conformal space was then prototyped in MATLAB to verify the accuracy of the algorithms for an optimized implementation in GPU*

Key words: Conformal Geometry, Collision Detection, Geometric Algebra

1 INTRODUCTION

INtersection among primitives (such as lines, planes and spheres) are core concepts for any collision algorithm, and essential in several Computer Graphics areas such as dynamics, simulation, and graphics rendering [1]. We propose a new unified treatment described herein for all types of collisions: line segment-line segment, line segment-triangle, line segment-sphere, triangle-sphere and sphere-sphere, using Conformal 5D Geometry. A related work [2] shows the implementation in GPU of this algorithms to optimize computation times.

Conformal Geometry uses the geometric algebra framework, which allows for a simple and compact representation of all primitives, adapting easily to the way of how these objects are represented in computer graphics. In its relation to collision detection it has the great advantage of unifying all object intersections by just one formula.

2 GEOMETRIC ALGEBRA

OF the varied approaches for mathematical description in Computer Graphics, the most recent has been the Geometric Algebra formulation, of which a complete description may be found in the work of Dorst [3] [4], and Vince [5].

This algebra has three main operators: the known inner (*dot*) product ($\mathbf{x}\cdot\mathbf{y}$), the outer (*bivector*) product ($\mathbf{x}\wedge\mathbf{y}$), and the geometric product ($\mathbf{xy}$). As a vector space, it shares other properties of a vector algebra, such as Euclidean distance, invariance, etc.

Definition 2.1 *For vectors* $\mathbf{a}$ *and* $\mathbf{b}$*, the outer product* $\mathbf{a}\wedge\mathbf{b}$ *defines an oriented hyperplane, or bivector, with magnitude the signed area of the counterclockwise parallelogram* $\|\mathbf{a}\wedge\mathbf{b}\| = \|\mathbf{a}\|\|\mathbf{b}\|\sin\theta$.

Definition 2.2 *The* geometric product $\mathbf{ab}$ *is a notation for the sum of its inner and outer products.*

$$\mathbf{ab} \;=\; \mathbf{a}\cdot\mathbf{b} \;+\; \mathbf{a}\wedge\mathbf{b} \tag{1}$$

From these definitions are derived the following algebraic properties

$$\begin{aligned}
a\wedge a &= 0 & a^2 &= a\cdot a = \|a\|^2\\
(\lambda a)\wedge b &= \lambda(a\wedge b),\ \text{for scalar } \lambda & (\lambda a)b &= \lambda(ab),\ \text{for scalar } \lambda\\
b\wedge a &= -a\wedge b = a\wedge(-b) & ba &= b\cdot a + b\wedge a = a\cdot b - a\wedge b\\
a\wedge(b+c) &= a\wedge b + a\wedge c & a(b+c) &= ab+ac\\
a\cdot b &= \tfrac{1}{2}(ab+ba) & a\wedge b &= \tfrac{1}{2}(ab-ba)
\end{aligned}$$

3 CONFORMAL GEOMETRY

Conformal Geometry [3, 6] describes an elegant algebraic space for geometric visualization in $\mathbb{R}^3$, being homogeneous, supporting point and lines at infinity, preserving angles and distances, and defining concisely points, lines, planes and spheres.

Definition 3.1 *A conformal space* $\mathbb{R}^{p+1,q+1}$ *of* $p+1$ *positive dimensions and* $q+1$ *negative dimensions is built from a* $\mathbb{R}^{p,q}$*. A point* $x = ue_1 + ve_2 + we_3$ *in* $\mathbb{R}^3$ *is a null vector* $X = P(x)$ *in* $\mathbb{R}^{4,1}$ *(inner product* $X\cdot X = 0$*, for* $X \neq 0$*), having the orthonormal base* $\{e_1, e_2, e_3, e, \bar{e}\}$.

$$\begin{aligned}
&e_1\cdot e_1 = 1, \quad e_2\cdot e_2 = 1, \quad e_3\cdot e_3 = 1, && n = e+\bar{e}, \quad \bar{n} = e-\bar{e}\\
&e\cdot e = 1, \quad \bar{e}\cdot\bar{e} = -1 && X = P(x) = 2x + x^2 n - \bar{n}
\end{aligned}$$

with n and $\bar{n}$ representing null vectors at *infinity* and at the *origin*. The primitives shown on Table 1 are derived from these null vectors.

Table 1: *Algebraic primitives built in Conformal Geometry 5D*

Primitive	Representation	Geometric Interpretation
Circle	$\mathrm{C} = P_1\wedge P_2\wedge P_3$	Defined by three noncollinear points in the perimeter of the circle
Line	$\mathrm{L} = P_1\wedge P_2\wedge n$	Defined by two nonidentical points on the line plus the point at ∞
Sphere	$\mathrm{S} = P_1\wedge P_2\wedge P_3\wedge P_4$	Four noncoplanar points on the surface of the sphere
Plane	$\Pi = P_1\wedge P_2\wedge P_3\wedge n$	Three noncollinear points define a triangle plus the point at ∞
Parallelogram	$\mathrm{A} = q_1\wedge q_2$	Area formed by two anchored vectors
Parallelepiped	$\mathrm{V} = q_1\wedge q_2\wedge q_3$	Volume formed by three anchored vectors
Pseudoscalar	$\mathrm{I} = e_1\wedge e_2\wedge e_3\wedge e\wedge\bar{e}$	The cannonical rotor for the $\mathbb{R}^{4,1}$ of the conformal space vector base

Definition 3.2 *A* k-blade *or* k-vector *is the outer product of k vectors:*

$$v_1 \wedge v_2 \wedge \ldots \wedge v_{k-1} \wedge v_k = V \in \mathbb{R}^{n,m}, \;\; k \leq n+m \tag{2}$$

Definition 3.3 *A* pseudoscalar *is the highest order blade in the space $\mathbb{R}^{n,m}$, represented by I ($I^2 = -1$), and it is analogous to* **i**, *the imaginary $90°$ counterclockwise canonical rotor of $\mathbb{C}$, the complex plane.*

3.1 INTERSECTIONS IN CONFORMAL SPACE

Intersections in the conformal geometry model [6] have the advantage of being expressed by just one formula for all primitives, as shown in Table 2.

Definition 3.4 *The* meet *operator ($\vee$) denotes a* multivector *expression of up to 32 terms (in $\mathbb{R}^{4,1}$) representing the geometric intersection of two multivectors. It has the 5-term multivector $\mathbf{I} = \mathbf{e_1 e_2 e_3 e \bar{e}}$ as pseudoescalar for that space.*

$$\begin{aligned} B &= (X \vee Y) = (I\,X) \cdot Y, \quad B^2 = \|B\|^2 \\ B &= \beta_0 + \beta_{e_1} e_1 + \ldots + \beta_{e_1 e_2} e_1 e_2 + \ldots + \beta_{e_1 e_2 e_3 e \bar{e}} e_1 e_2 e_3 e \bar{e} \end{aligned} \tag{3}$$

The work of Roa [7] shows the calculus made for each intersection expressing value for B^2:

Table 2: *Primitive intersections in $\mathbb{R}^{4,1}$ conformal space*

Primitives	Conformal representation
Line – Plane	$B = \Pi_1 \vee L_1 = (I\,\Pi_1) \cdot L_1$
Plane – Plane	$B = \Pi_1 \vee \Pi_2 = (I\,\Pi_1) \cdot \Pi_2$
Line – Sphere	$B = S_1 \vee L_1 = (I\,S_1) \cdot L_1$
Plane – Sphere	$B = S_1 \vee \Pi_1 = (I\,S_1) \cdot \Pi_1$
Sphere – Sphere	$B = S_1 \vee S_2 = (I\,S_1) \cdot S_2$

$$\text{if } B^2 \begin{cases} > 0, & \text{intersection at least at two points} \\ = 0, & \text{intersection at a tangent point} \\ < 0, & \text{primitives do not intersect} \end{cases}$$

For spheres (and circles), the radius ρ and the center ε are given by the expressions $\rho^{\mathbf{2}} = \frac{-\mathbf{S^2}}{(\mathbf{S} \wedge \mathbf{n})^2}$ and $\varepsilon = \mathbf{SnS}$

4 ALGORITHMS

In computer graphics the intersection of a plane with a sphere or a line with a plane, is straightforward, but not very interesting. More useful are the intersections of triangles with spheres, or line segments with triangles. The following algorithms solve these intersections using the formulas given in Table 2.

4.1 LINE SEGMENT-TRIANGLE INTERSECTION

The first step is to determine the intersection of the line containing the segment with the plane containing the triangle in the conformal model. Using the equation [7]:

$$\begin{aligned} \mathbf{B} \;\; = \;\; & (\omega_2\beta_3 + \omega_1\beta_4 - \omega_4\beta_1) e_1 e \; + \; (\omega_2\beta_3 + \omega_1\beta_4 - \omega_4\beta_1) e_1 \bar{e} \; + \; (\omega_3\beta_1 - \omega_2\beta_2 + \omega_1\beta_5) e_2 e \\ + \;\; & (\omega_3\beta_1 - \omega_2\beta_2 - \omega_1\beta_5) e_2 \bar{e} \; + \; (-\omega_3\beta_3 + \omega_4\beta_2 + \omega_1\beta_6) e_3 e \; + \; (-\omega_3\beta_3 + \omega_4\beta_2 + \omega_1\beta_6) e_3 \bar{e} \\ + \;\; & (-\omega_3\beta_4 - \omega_4\beta_5 - \omega_2\beta_6) e \bar{e}, \qquad \textit{with scalar discriminant } \mathbf{B^2} = (\omega_3\beta_4 + \omega_4\beta_5 + \omega_2\beta_6)^2 \end{aligned} \tag{4}$$

where the β's y ω's are the coefficients of the corresponding multivectors L_1 and Π_1. A nonnegative B^2 signals a potential collision. If the intersection occurs, it then proceeds to intersect the line segment with each edge of the triangle. Algorithm **1** describes the procedure for calculating the intersection.

Algorithm 1: Line Segment-Triangle intersection

```
kernel Segment_Triangle_Intersect(segment L1, plane P1)
  Normalize(L1);   Normalize(P1)
  [a,e3er] = ConformalIntersectLinePlane(L1,P1)
  L3 = Line(L0.point1,L0.point2)
  L2 = Line(P1.point3,P1.point1)
  [out1,ind1] = ConformalIntersectSegmentSegment(L2, L3 )
  L2 = Line(P1.point1,P1.point2)
  [out2,ind2] = ConformalIntersectSegmentSegment(L2, L3 )
  L2 = Line(P1.point3,P1.point2)
  [out3,ind3] = ConformalIntersectSegmentSegment(L2, L3 )
    // out# = 1 (segments intersect); 0 (they do not)
    // ind#: scalar coefficient of e vector
  if (a == 0) then    //line and plane parallel
    if e3er = 0 then //line lies on the triangle's plane
      //verify segment intersection with other triangles
      return (out1==1) or (out2==1) or (out3==1)
    else return 0      // No intersection found
    end if
  else //whether both points are in same side of plane
    sign1 = trivector(P1.point2-P1.point1,P1.point3-P1.
          point1,L1.point1-P1.point1)
    sign2 = trivector(P1.point2-P1.point1,P1.point3-P1.
          point1,L1.point2-P1.point1)
    if (sign1 == sign2) then
      return 0          // Segment does not touch plane
    end if              // Segment crosses the plane
    return result = (ind1>0 and ind2>0 and ind3<0) or
                    (ind1<0 and ind2<0 and ind3>0);
  end if
```

Algorithm 2: Triangle-Triangle intersection

```
kernel Triangle_Triangle_Intersect(triangle P1, triangle P2
      )
  //Verify if all points are at same side of plane
  if not VerifySameSidePoints(P1) then
    return 0 // No Intersection
  end if
  Normalize(P1);   Normalize(P2)
  r1 = Line(P2.point1,P2.point2)
  r2 = Line(P2.point2,P2.point3)
  r3 = Line(P2.point3,P2.point1)
  // out# = 1, intersection exists , 0 no intersection ,
  out1 = ConformalIntersectSegmentPlane(r1, P1 )
  out2 = ConformalIntersectSegmentPlane(r2, P1 )
  out3 = ConformalIntersectSegmentPlane(r3, P1 )
  if (out1 == 1) or (out2 == 1) or (out3 == 1) then
    return 1  // intersection exists
  end if
  r1 = Line(P1.point1,P1.point2)
  r2 = Line(P1.point2,P1.point3)
  r3 = Line(P1.point3,P1.point1)
  out1 = ConformalIntersectSegmentPlane(r1, P2 )
  out2 = ConformalIntersectSegmentPlane(r2, P2 )
  out3 = ConformalIntersectSegmentPlane(r3, P2 )
  return  (out1 == 1) or (out2 == 1) or (out3 == 1)
```

4.2 TRIANGLE-TRIANGLE INTERSECTION

For triangle-triangle intersections, we do not use the plane-plane intersection from the conformal model. Instead, we take each edge of first triangle and intersect it with the edges of the second one using the algorithm **1**. Then we take each edge of the second triangle and repeat the process above with the first triangle. Algorithm **2** describes this procedure.

4.3 LINE SEGMENT-SPHERE INTERSECTION

The first step checks whether just one endpoint on the line segment is inside the sphere, therefore an intersection occurs. If both points are outside the sphere, we proceed to apply the calculus using the conform model, checking the sign of B^2 [7]. For this case that there is a crossing between the line

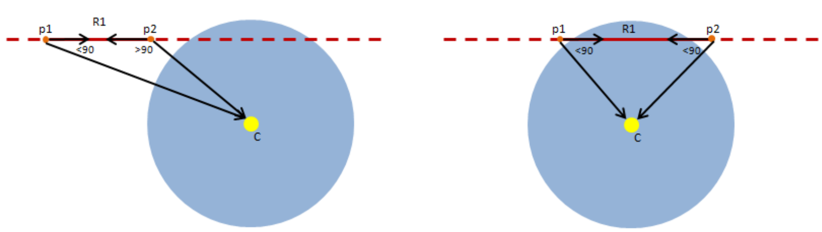

Figure 1: *Left: one of the angles between the vectors is* $> 90°$. *Right: both angles are acute*

segment and the sphere, but both endpoints of the line segment are outside of it, we consider the angles among the vectors indicated in Figure 1. If both angles are acute, then the line segment has pierced the sphere. The procedure is indicated in the algorithm **3**.

Algorithm 3: Line Segment-Sphere intersection

```
kernel Segment_Sphere_Intersect(segment R1, sphere E1 )

  ChangedCoordinatedSphere(E1)
  ChangedCoordinatedLine(R1)
  b = CheckPointInsideSphere(R1)
  if (b==true)
    return 1 //Intersection exist
  end if
  Normalize(R1)
  a = IntersectionConformModelLineSphere(R1,E1)
  // a = 0 segment is tangent, a >= 0 No intersect
  if (a<=0)
      if (a==0) return 1
      if (a>=0) return 0
  else
      vecOP1 = -R1.punto1
      vecOP2 = -R1.punto2
      vecP1P2 = R1.punto2 - R1.punto1
      vecP2P1 = R1.punto1 - R1.punto2
      cos1 = CosineAngle(vecOP1,vecP1P2)
      cos2 = CosineAngle(vecOP2,vecP2P1)
      if (cos1 < 0) && (cos2 < 0)
          return 0
      else // both angles are acute, intersection exist
          return 1
      end if
  end if
```

Algorithm 4: Triangle-Sphere intersection

```
kernel Triangle_Sphere_Intersect(triangle P1, sphere E1 )

  ChangedCoordinatedSphere(E1)
  ChangedCoordinatedPlane(P1)
  r = CheckPointInsideSphere(P1)
  if (r==true)
      return 1 // Intersection exist
  end if
  Normalize(P1)
  r = CalculateIntersectionConformalModelPlaneSphere(P1,E1)
  // r < 0 no intersection, r > 0 posible intersection
  if (r < 0)
     return 0
  else
     // Check if one line of the triangle intersect the
         sphere
     r1 = Line(P1.punto2,P1.punto1)
     r2 = Line(P1.punto3,P1.punto2)
     r3 = Line(P1.punto1,P1.punto3)
     res1 = Segment_Sphere_Intersect(r1, E1);
     res2 = Segment_Sphere_Intersect(r2, E1);
     res3 = Segment_Sphere_Intersect(r3, E1);
     if (res1 = 1) or (res2 = 1) or (res3 = 1)
        return 1 // Intersection exist
     else
        // Calculate point with minimal distance of the
            center sphere
        [d, q] = PointMinimalDistanceSphere(E1,P1) // d =
            distance, q point in the plane
        if (d <= r)
            v12 = P1.point2 - P1.point1
            v10 = q - P1.punto1
            v23 = P1.point3 - P1.point2
            v20 = q - P1.punto2
            v31 = P1.point1 - P1.point3
            v30 = q - P1.point3
            signed1 = bivector(v12,v10)
            signed2 = bivector(v23,v20)
            signed3 = bivector(v31,v30)
            //If signed are equal the point q is inside
                triangle
            if (signed1 = signed2) and (signed2 = signed3)
                return 1
            else
                return 0
            end if
        else
           return 0
        end if
     end if
  end if
```

4.4 TRIANGLE-SPHERE INTERSECTION

This algorithm requires three steps. If all three vertexes of the triangle are inside, then there are no intersections. If just one or two points of the triangle are inside the sphere, an intersection exists. If these conditions are not met, then the three points of the triangle are outside the sphere.

We proceed to calculate the intersection in the conformal model, checking the sign of B^2 [7], signaling a potential plane-sphere collision. Then we proceed to see if one of the triangle's segments intersects the sphere. If it's affirmative the intersection occurs. However, it may be possible that no segment intersects the sphere and still the intersection exists (see figure 2). To solve this case, we calculate the point **q** (figure 2) whose distance is minimal respect to the center. If that point **q** lies inside the sphere, then the triangle intersects it, otherwise there is no collision. The procedure is indicated in the algorithm **4**.

4.5 SPHERE-SPHERE INTERSECTION

The simplest of all intersections, just verifying the sign of B^2 [7] in the last equation of Table **2**.

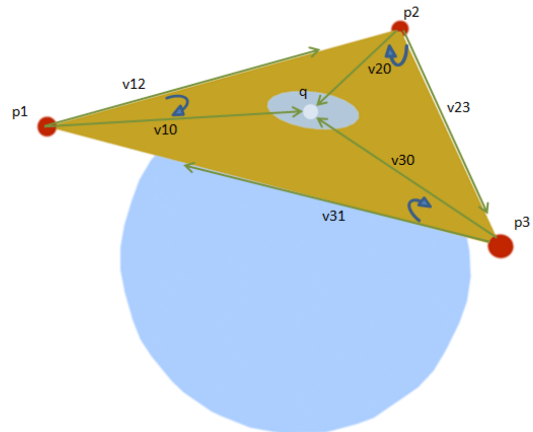

Figure 2: *Triangle-Sphere Intersection*

5 CONCLUSIONS

WE found a elegant procedure to deal with collisions among primitives in a unified manner under the conformal model, based on reformulating the collisions of $\mathbb{R}^3$ euclidean space in the corresponding conformal $\mathbb{R}^{4,1}$ geometric space. In our algorithms we also used several geometric formulations in $\mathbb{R}^3$ to complement our results. All the algorithms were prototyped and implemented in MATLAB to prove accuracy and correctness. When implemented using the GPU (Graphics Processor Unit) [2], it accounts for greatly accelerated computation times, in some cases allowing for real-time collision detection.

Future work involves simplifying and speeding up even more the calculation of these formulas, and investigating the geometric meaning of each coefficient of the resulting intersection multivector.

REFERENCES

[1] C. Ericson, *Real Time Collision Detection*. San Francisco, CA: Morgan Kaufmann, 2005.

[2] E. Roa, V. Theoktisto, M. Fairén, and I. Navazo, “GPU Collision Detection in Conformal Geometric Space,” in *V Ibero-American Symposium in Computer Graphics SIACG'2011*, pp. 153–157, Universidade do Algarve, Portugal, June 2011.

[3] L. Dorst, D. Fontijne, and S. Mann, *Geometric Algebra for Computer Science*. San Francisco, CA: Morgan Kaufmann, 2007.

[4] L. Dorst and S. Mann, “Geometric Algebra: A Computational Framework for Geometrical Applications (Part 1),” *IEEE Computer Graphics and Applications*, vol. 22, pp. 24–31, 2002.

[5] J. Vince, *Geometric Algebra for Computer Graphics*. London, UK: Springer, 2008.

[6] C. Doran and A. Lasenby, *Geometric Algebra for Physicists*. Cambridge, UK: Cambridge, 2009.

[7] E. Roa, “GPU-based Operations in 5D Conformal Geometric Space,” Master's thesis, Universidad Simón Bolívar, Caracas, Venezuela, May 2011.