

An Efficient Ray-Quadrilateral Intersection Test

Ares Lagae Philip Dutré

Department of Computer Science
Katholieke Universiteit Leuven

Abstract

We present a new and efficient method to compute the intersection point between a convex planar quadrilateral and a ray. Contrary to the Schlick-Subrenat intersection test, the bilinear coordinates of the intersection point are computed only for rays that hit the quadrilateral. Rays that do not hit the quadrilateral are rejected early. Our method is up to 40% faster compared to the algorithm presented by Schlick and Subrenat. The intersection test we present is based on the Möller-Trumbore ray-triangle intersection algorithm. The new test is at least as fast as two ray triangle intersection tests, and yields bilinear coordinates with no discontinuities in the isoparametrics. Source code implementing the intersection test is available online.

1 Introduction

The ray-quadrilateral intersection problem consists of determining whether a ray intersects a convex planar quadrilateral. In most cases, the problem also consists of computing the bilinear coordinates of the intersection point. These coordinates are used to interpolate shading normals and texture coordinates across the quadrilateral.

As Schlick and Subrenat [SS95] pointed out, considering a quadrilateral as two separate triangles introduces discontinuities in the isoparametrics. This leads to interpolation artefacts such as distorted texture mappings and incorrect shading, as illustrated in figure 1. Therefore, computing a ray-quadrilateral intersection is often preferred over splitting a quadrilateral into triangles.

Schlick and Subrenat [SS95] presented a simple but effective method to solve the ray-quadrilateral intersection problem. The ray is first intersected with the plane containing the quadrilateral. If the ray intersects the plane, the bilinear coordinates of the intersection point are computed. These coordinates determine whether the ray also intersects the quadrilateral.

The ray-quadrilateral intersection test we present in this paper is based on the Möller-Trumbore ray-triangle intersection method [MT97]. It is up to 40% faster compared to the Schlick-Subrenat ray-quadrilateral intersection algorithm [SS95]. We also show that computing one ray-quadrilateral intersection is at least as fast as computing two separate ray-triangle intersections. This means that when ray tracing a scene consisting of quadrilaterals, you should probably

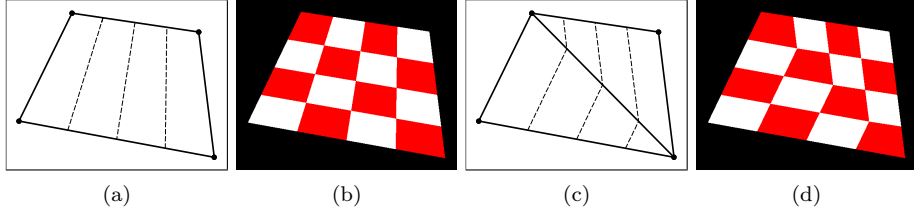


Figure 1: (a) A quadrilateral and its isoparametrics for $u = 0.25$, $u = 0.5$ and $u = 0.75$. (b) The texture mapped quadrilateral. (c) The isoparametrics become discontinuous when splitting the quadrilateral into triangles. (d) The texture mapped triangles. (This figure is based on [SS95, figure 2].)

keep them intact instead of splitting them into triangles. Source code implementing the intersection test is available online.

2 Overview

Computing the bilinear coordinates of a point with respect to a quadrilateral involves solving a quadratic system of two equations, which is an expensive operation. If the ray intersects the plane containing the quadrilateral, but not the quadrilateral itself, our method avoids computing the bilinear coordinates of the intersection point.

Our method can be summarized as follows: consider the two triangles defined by a quadrilateral and one of its diagonals. The barycentric coordinates of the intersection point with respect to these triangles determine whether the ray intersects the quadrilateral or not. If the ray intersects the quadrilateral, the barycentric coordinates of the intersection point with respect to one of these triangles are transformed to bilinear coordinates with respect to the quadrilateral.

3 Intersection Test

Let V_{00} , V_{10} , V_{11} and V_{01} be the vertices of the convex planar quadrilateral $Q = \langle V_{00}, V_{10}, V_{11}, V_{01} \rangle$, listed in counterclockwise order. Each point $Q(u, v)$ in the plane of Q can be written as

$$Q(u, v) = (1 - u)(1 - v)V_{00} + u(1 - v)V_{10} + uvV_{11} + (1 - u)vV_{01}, \quad (1)$$

where (u, v) are the bilinear coordinates of $Q(u, v)$ with respect to Q . If $0 \leq u \leq 1$ and $0 \leq v \leq 1$, then $Q(u, v)$ lies within Q . This equation describes a bilinear mapping, which is a mapping of the unit square into a quadrilateral. This mapping is computed by first linearly interpolating along the top and bottom edges of the quadrilateral, and then linearly interpolating between the two interpolated points.

Q 's diagonal $V_{10}V_{01}$ determines two triangles, $T = \langle V_{10}, V_{01}, V_{00} \rangle$ and $T' = \langle V_{01}, V_{10}, V_{11} \rangle$ (see figure 2). Each point $T(\alpha, \beta)$ in the plane of T (and Q) can

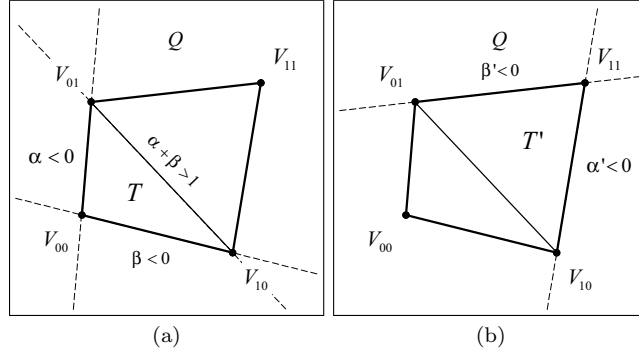


Figure 2: The barycentric coordinates with respect to (a) T and (b) T' of the intersection point are used to reject rays that do not intersect the quadrilateral.

be written as

$$T(\alpha, \beta) = V_{00} + \alpha(V_{10} - V_{00}) + \beta(V_{01} - V_{00}), \quad (2)$$

where (α, β) are the barycentric coordinates of $T(\alpha, \beta)$ with respect to T . If $\alpha \geq 0$, $\beta \geq 0$ and $\alpha + \beta \leq 1$, then $T(\alpha, \beta)$ lies within T .

The barycentric coordinates of the intersection point with respect to T are computed using the approach outlined in [MT97]. The parametric equation of a ray R with origin O and direction D is given by

$$R(t) = O + tD, \quad (3)$$

with $t \geq 0$. The barycentric coordinates and the ray parameter t of the intersection point of the ray R and the plane of T are obtained by solving $R(t) = T(\alpha, \beta)$, or

$$O + tD = V_{00} + \alpha(V_{10} - V_{00}) + \beta(V_{01} - V_{00}), \quad (4)$$

for α , β and t . We only solve for α and β . Solving for t is postponed until we are sure that the ray intersects the quadrilateral.

If $\alpha \geq 0$, $\beta \geq 0$ and $\alpha + \beta \leq 1$, the ray intersects T and thus Q . If $\alpha < 0$ or $\beta < 0$, the ray hits neither T nor Q . If $\alpha \geq 0$, $\beta \geq 0$ and $\alpha + \beta > 1$, the ray does not hit T , but it cannot be classified with respect to Q . In this case, the barycentric coordinates (α', β') of the intersection point with respect to the triangle T' are computed. If $\alpha' \geq 0$ and $\beta' \geq 0$, the ray hits T' and thus Q , otherwise it misses both T' and Q . The process of ray rejection is illustrated in figure 2.

For rays that intersect the quadrilateral, the bilinear coordinates of the intersection point are computed as follows. Let $(\alpha_{11}, \beta_{11})$ be the barycentric coordinates of V_{11} with respect to T . These can be computed by solving

$$(V_{11} - V_{00}) = \alpha_{11}(V_{10} - V_{00}) + \beta_{11}(V_{01} - V_{00}) \quad (5)$$

for α_{11} and β_{11} . This equation expands to a linear system of three equations in two unknowns, one equation being a linear combination of the other two. The

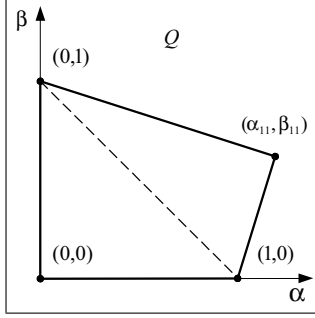


Figure 3: The quadrilateral Q in barycentric coordinate space.

barycentric coordinates of the vertices of Q with respect to T are $(0,0)$, $(1,0)$, $(\alpha_{11}, \beta_{11})$ and $(0,1)$. In barycentric coordinate space, the bilinear mapping that maps the unit square into Q is given by

$$\begin{aligned}\alpha &= uv(\alpha_{11} - 1) + u \\ \beta &= uv(\beta_{11} - 1) + v.\end{aligned}\quad (6)$$

These formulas are obtained by solving the system of equations

$$\begin{aligned}T(0,0) &= Q(0,0) \\ T(1,0) &= Q(1,0) \\ T(\alpha_{11}, \beta_{11}) &= Q(1,1) \\ T(0,1) &= Q(0,1).\end{aligned}\quad (7)$$

The bilinear coordinates of a point in the plane of Q are obtained by applying the inverse of this bilinear mapping on the barycentric coordinates of that point. The inverse of this bilinear mapping is computed by solving the system of equation 6 for u and v . u is obtained by solving the quadratic equation

$$-(\beta_{11} - 1)u^2 + (\alpha(\beta_{11} - 1) - \beta(\alpha_{11} - 1) - 1)u + \alpha = 0, \quad (8)$$

and once u is known, v is obtained with the equation

$$v = \frac{\beta}{u(\beta_{11} - 1) + 1}.\quad (9)$$

Note that when α_{11} or β_{11} (or both) are equal to 1, the quadratic system of equation 6 degrades to a linear one. In the case where $(\alpha_{11}, \beta_{11})$ equals $(1,1)$, Q is a parallelogram and the bilinear mapping (and its inverse) is the identity transformation. In this case, the bilinear coordinates of the intersection point are equal to its barycentric coordinates. If either α_{11} or β_{11} is equal to 1, Q is a trapezium. In the case where α_{11} equals 1, $u = \alpha$ and v is given by equation 9. In the case where β_{11} equals 1, $v = \beta$ and u is given by

$$u = \frac{\alpha}{v(\alpha_{11} - 1) + 1}.\quad (10)$$

If the quadrilateral has to be intersected with many rays, the barycentric coordinates of V_{11} with respect to T can be precomputed, because they are independent of the ray.

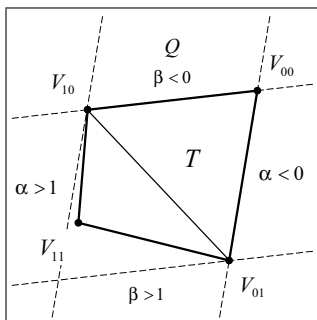


Figure 4: By reordering the vertices of Q , more rays can be rejected using only the barycentric coordinates with respect to T .

Another optimization consists of reordering the vertices of Q such that V_{11} lies within the parallelogram determined by T . This is easily done by inspecting the barycentric coordinates of V_{11} with respect to T , and permuting the vertices of necessary. Rays that do not intersect the parallelogram can be rejected using only the barycentric coordinates with respect to T , and the barycentric coordinates with respect to T' will need to be computed for less rays. This optimization is illustrated in figure 4.

4 Implementation

The pseudocode of the ray-quadrilateral intersection algorithm is shown in figure 5. The first block of code rejects rays that are parallel to Q , and rays that intersect the plane of Q either on the left of the line $V_{00}V_{01}$ or on the right of the line $V_{00}V_{10}$. The third and fifth return statement in this block of code reject rays that intersect the plane of Q outside of the parallelogram determined by T . These are only needed when vertex reordering is used. The second block of code rejects rays that intersect the plane of Q either on the left of the line $V_{11}V_{10}$ or on the right of the line $V_{11}V_{01}$. This block of code is only executed if the ray does not hit T . The third block of code computes the ray parameter t of the intersection point. The fourth block of code solves the system given by equation 5. To avoid numerical instability, the equation corresponding to the component of largest magnitude of Q 's normal is discarded. When the barycentric coordinates of V_{11} with respect to T are precomputed, this block of code is removed from the intersection algorithm. Finally, the last block of code computes u , the first bilinear coordinate, by solving equation 8. Because the ray hits Q , it is not necessary to check for $\Delta < 0$, and at least one of the roots lies between 0 and 1. Once u is known, v is calculated according to equation 9. Some efficiency might be gained by replacing the divisions by multiplications by precomputed inverses, and postponing these multiplications as long as possible.

5 Results & Discussion

We did three different tests to compare our ray-quadrilateral intersection algorithm with the one of Schlick and Subrenat, and with the Möller-Trumbore ray-triangle intersection method.

```

bool intersect(O, D, V00, V10, V11, V01, u, v, t)
{
    // Reject rays using the barycentric coordinates of
    // the intersection point with respect to T.
    E01 = V10 - V00
    E03 = V01 - V00
    P = D × E03
    det = E01 · P
    if (|det| < ε) return false
    T = O - V00
    α = (T · P)/det
    if (α < 0) return false
    if (α > 1) return false
    Q = T × E01
    β = (D · Q)/det
    if (β < 0) return false
    if (β > 1) return false

    // Reject rays using the barycentric coordinates of
    // the intersection point with respect to T'.
    if ((α + β) > 1) {
        E23 = V01 - V11
        E21 = V10 - V11
        P' = D × E21
        det' = E23 · P'
        if (|det'| < ε) return false
        T' = O - V11
        α' = (T' · P')/det'
        if (α' < 0) return false
        Q' = T' × E23
        β' = (D · Q')/det'
        if (β' < 0) return false
    }

    // Compute the ray parameter of the intersection
    // point.
    t = (E03 · Q)/det
    if (t < 0) return false

    // Compute the barycentric coordinates of V11.
    E02 = V11 - V00
    N = E01 × E03
    if ((|N.x| ≥ |N.y|)
        and (|N.x| ≥ |N.z|)) {
        α11 = (E02.y * E03.z - E02.z * E03.y)/N.x
        β11 = (E01.y * E02.z - E01.z * E02.y)/N.x
    } else if ((|N.y| ≥ |N.x|)
        and (|N.y| ≥ |N.z|)) {
        α11 = (E02.z * E03.x - E02.x * E03.z)/N.y
        β11 = (E01.z * E02.x - E01.x * E02.z)/N.y
    } else {
        α11 = (E02.x * E03.y - E02.y * E03.x)/N.z
        β11 = (E01.x * E02.y - E01.y * E02.x)/N.z
    }

    // Compute the bilinear coordinates of the
    // intersection point.
    if (|α11 - 1| < ε) {
        u = α
        if (|β11 - 1| < ε) v = β
        else v = β/(u * (β11 - 1) + 1)
    } else if (|β11 - 1| < ε) {
        v = β
        u = α/(v * (α11 - 1) + 1)
    } else {
        A = -(β11 - 1)
        B = α(β11 - 1) - β(α11 - 1) - 1
        C = α
        Δ = B2 - 4AC
        Q = -1/2 * (B + sign(B)√Δ)
        u = Q/A
        if ((u < 0) or (u > 1)) u = C/Q
        v = β/(u(β11 - 1) + 1)
    }

    return true
}

```

Figure 5: Pseudo-code of the ray-quadrilateral intersection test.

The first test measures the time needed to ray trace a single quadrilateral using a minimal ray tracer. The test was repeated for a batch of 1000 randomly oriented, convex planar quadrilaterals. The area of the quadrilaterals is uniformly distributed between 0% and 100% of the area of the viewport of the ray tracer. Figure 6(a) shows the results. Our algorithm is 23% to 40% faster than the one of Schlick and Subrenat, and 13% slower to 12% faster (depending on which optimizations are used) than two applications of the Möller-Trumbore ray-triangle intersection algorithm. For reference purposes, we have also included the time needed for a single ray-triangle intersection. We have repeated the test using different batches of random quadrilaterals with a fixed area. Figure 6(b) shows the results. The method of Schlick and Subrenat performs roughly equally well on each of those batches. In contrast, our method becomes faster as the area of the quadrilaterals decreases. This clearly shows the effect of early ray rejection.

The second test matches the situation in a real ray tracer more closely (in particular, the presence of acceleration structures). This test measures the time needed to intersect a quadrilateral with approximately 15000 randomly generated rays that intersect its bounding box. Again, this test was repeated for a batch of 1000 random quadrilaterals. Figure 6(c) shows the results. Our algorithm is 15% to 28% faster than the one of Schlick and Subrenat, and up to 14% faster than two applications of the Möller-Trumbore ray-triangle intersection test.

The third and final test consisted of implementing the proposed intersection method in our renderer¹, which formerly used the Schlick and Subrenat intersection test. As a result, we have observed a decrease in total rendering time

¹RENDERPARK (<http://www.renderpark.be>)

up to 5 percent.

The algorithm of Schlick and Subrenat and our method both process quadrilaterals with parallel sides in a more efficient way. Therefore, we have verified that our method also outperforms the algorithm of Schlick and Subrenat for trapeziums and parallelograms.

Note that our algorithm does not check for degenerate quadrilaterals. Although our intersection test could be extended to gracefully handle degenerate cases, checking for degenerate quadrilaterals is typically done as a preprocess, and therefore does not justify a more expensive intersection test.

Our intersection test is numerically somewhat more robust than the one of Schlick and Subrenat. Because less operations are needed to reject non-intersecting rays, our algorithm is less prone to rounding errors. A small drawback of our intersection test is that consistency between the test result and the bilinear coordinates is not guaranteed.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive remarks, and the people of our research group for their clever suggestions and careful proofreading. The first author is funded as a Research Assistant by the Fund of Scientific Research - Flanders, Belgium (Aspirant F.W.O.- Vlaanderen).

References

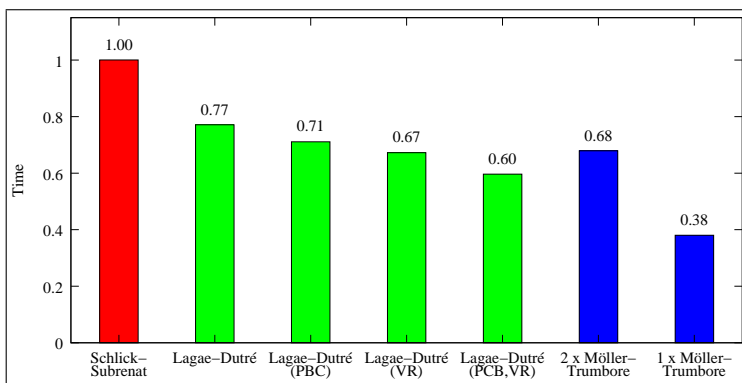
- [MT97] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.
- [SS95] Christophe Schlick and Gilles Subrenat. Ray intersection of tessellated surfaces: Quadrangles versus triangles. In Alan Paeth, editor, *Graphics Gems V*, pages 232–241. Academic Press, San Diego, 1995.

Web Information

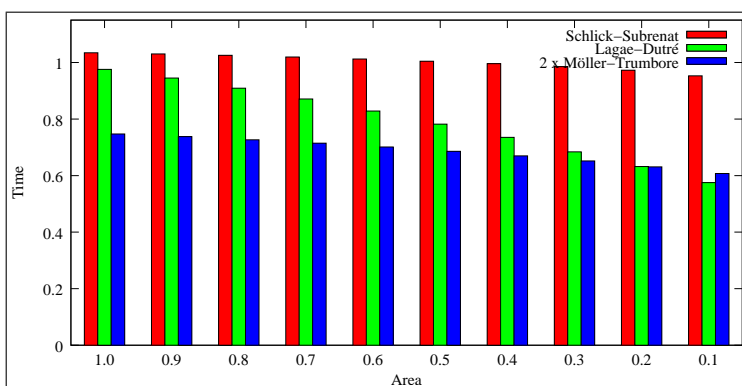
Source code implementing the intersection test described in this paper can be found at <http://www.acm.org/jgt/papers/LagaeDutre05>.

Ares Lagae, Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200 A, B-3001 Heverlee, Belgium
(ares.lagae@cs.kuleuven.be).

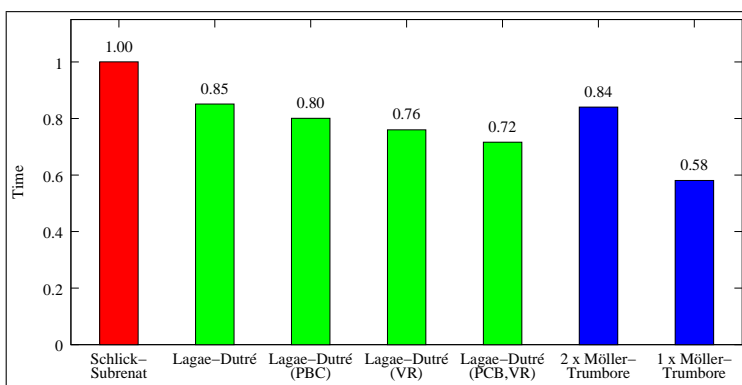
Philip Dutré, Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200 A, B-3001 Heverlee, Belgium
(philip.dutre@cs.kuleuven.be).



(a)



(b)



(c)

Figure 6: The relative performance of the different ray-quadrilateral intersection algorithms compared (see section 5). PCB stands for *precomputed barycentric coordinates*, and VR for *vertex reordering*.