RUHUNA JOURNAL OF SCIENCE
Vol 7: 12-20, June 2016
ISSN: 1800-279X

An approach for line clipping against a convex polyhedron

K. R. Wijeweera[1], S. R. Kodituwakku[2]

[1]*Department of Computer Science, Faculty of Science, University of Ruhuna*
[2]*Department of Statistics and Computer Science, Faculty of Science, University of Peradeniya*
[1, 2]*Postgraduate Institute of Science, University of Peradeniya*

Correspondence: [1]krw19870829@gmail.com

Received: 18th October 2015, Revised: 11th June 2016, Accepted: 24th June 2016

Abstract. Line clipping operation is a bottleneck in most of computer graphics applications. There are situations when millions of line segments need to be clipped against convex polyhedrons with millions of facets. An algorithm to clip line segments against a convex polyhedron is proposed in this work. The salient feature of the proposed algorithm is that it minimizes the number of computations by ignoring unnecessary intersection calculations. The other advantage of the proposed algorithm is that it needs minimum details about the convex polyhedron; the equations of the facets and the centroid. Therefore, it improves the efficiency of the algorithm. The line segment may have zero length (a point) or positive length. When line segment is just a point which is outside with respect to at least one facet, it should be rejected as the line segment is outside the convex polyhedron. When the line segment is parallel to a facet and one of its end points is outside, that line segment is also completely outside and it should also be rejected. Unless the line segment belongs to none of the above two cases, it should be pruned against each facet in a certain order. In this case, the intersection points with only some of the facets need to be computed and some other intersection calculations can be ignored. If the line segment is completely outside then it becomes a single point. That means the two end points coincide. But due to the precision error they do not exactly coincide. Therefore, approximate equality should be tested. By using this property, completely outside line segments can be identified. Having two end points outside does not necessarily keep the line segment completely outside. The widely used Cyrus Beck algorithm computes all the intersection points with each facet of the polyhedron while the proposed algorithm successfully avoids some of the intersection point calculations. In the best case; it is capable of avoiding all the unnecessary intersection calculations. An experimental comparison between the Cyrus Beck algorithm and the proposed algorithm was carried out. Random polyhedrons were created with different number of facets. Random points were generated and they were considered as end points of line segments. For a given polyhedron, the number of clock cycles consumed to clip 10^8 number of line segments was computed using the Cyrus Beck algorithm and the proposed algorithm. For a polyhedron with four vertices, the proposed algorithm is 1.02 times faster than the Cyrus Beck algorithm. For a polyhedron with nine vertices, the proposed algorithm is 1.16 times faster than the Cyrus Beck algorithm.

When the number of facets is large, then the performance of the proposed algorithm is significantly faster since more intersection calculations are avoided. The Skala algorithm is also an efficient algorithm which requires the order of facets also as the input to the algorithm. The proposed algorithm is faster than Skala algorithm when the number of facets of the polyhedron is less than 100 according to the experimental results. The proposed approach could be very useful for applications where large number of lines to be clipped. It can also be applied in linear programming as well since it can be extended to arbitrary dimensions.

Keywords. Computational Geometry, Convex Analysis, Computer Graphics Programming, Coordinate Geometry, Linear Programming

1 Introduction

In computer graphics, line clipping is one of the most important operations. The concept of line clipping is applied in many application areas. If a part of a given scene needs to be extracted then line clipping is used as a primitive operation to perform extraction (D. Hearn *et al*, 1998). Especially in medical applications certain regions needs to be clipped against polyhedrons (W. Huang, 2010).

The line clipping process is a bottleneck in many applications. Therefore, optimizing the performance of line clipping is very important to solve this problem. When spherical or cylindrical volumes are approximated using polyhedrons there will be a large number of facets in the resultant polyhedron (W. Huang, 2010). Then the line segment has lots of intersections with facets of that polyhedron. Calculation of an intersection point involves a higher computational cost and it has a significant impact on the performance of a clipping algorithm. The proposed algorithm uses a mechanism to avoid some of the intersection calculations. In the best case, it is capable of avoiding all the unnecessary intersection calculations.

2 Materials and Methods

In this section, the proposed line clipping algorithm is presented. The convex polyhedron is represented using the set of infinite planes formed by extending its facets. The number of facets of the convex polyhedron is stored in facets variables. The equation of a plane in general form is a * x + b * y + c * z + d = 0 where a, b, c and d are constants. Therefore, the i^{th} facet can be represented as a[i] * x + b[i] * y + c[i] * z + d[i] = 0, where i = 1, 2..., (number of facets – 1). The line segment in three dimensions can be represented by its end points. The end points are represented as (x[i], y[i], z[i]), where i = 0, 1. The coordinates of the Centroid (xc, yc, zc) of the

clipping volume should be provided to the algorithm as an input. When the polyhedron is convex its Centroid lies inside it.

2.1 Mathematical background of the proposed algorithm

The general equation of a straight line joining the end points (x[0], y[0], z[0]) and (x[1], y[1], z[1]) can be be written as,
(x – x[0])/(x[1] - x[0]) = (y – y[0])/(y[1] - y[0]) = (z – z[0])/(z[1] - z[0]) = t.
This can be rewritten in the following form.
x = x[0] + (x[1] – x[0]) * t;
y = y[0] + (y[1] – y[0]) * t;
z = z[0] + (z[1] – z[0]) * t.

The general equation of the i^{th} facet is a[i] * x + b[i] * y + c[i] * z + d[i] = 0.
The intersection point of the i^{th} plane and the given line segment (if exists), can be calculated as
a[i] * {x[0] + (x[1] – x[0]) * t} + b[i] * {y[0] + (y[1] – y[0]) * t} + c[i] * { z[0] + (z[1] – z[0]) * t} + d[i] = 0
→ a[i] * x[0] + b[i] * y[0] + c[i] * z[0] + d[i] + {(a[i] * x[1] + b[i] * y[1] + c[i] * z[1]) – (a[i] * x[0] + b[i] * y[0] + c[i] * z[0])} * t = 0
→ p[0] + (p[1] – p[0]) * t = 0, where p[0] = a[i] * x[0] + b[i] * y[0] + c[i] * z[0] + d[i] and p[1] = a[i] * x[1] + b[i] * y[1] + c[i] * z[1] + d[i]
→ t = p[0]/(p[0] - p[1])

Let k = a[i] * xc + b[i] * yc + c[i] * zc + d[i].
If (p[i] * k < 0) then the (x[i], y[i], z[i]) is outside the convex polyhedron.

2.2 Pseudo Code of the Proposed Algorithm

```
FUNCTION: clip(a[], b[], c[], d[], facets, xc, yc, zc, x[], y[], z[])
BEGIN
IF {(x[0] = x[1]) AND (y[0] = y[1]) AND (z[0] = z[1])}
        FOR {i = 0 TO i = facets -1}
                IF {(a[i]*x[0]+b[i]*y[0]+c[i]*z[0]+d[i])*(a[i]*xc+b[i]*yc+c[i]*zc+d[i])<0}
                        GOTO line;
                END IF
        END FOR
        SAVE (x[0], y[0], z[0]);
ELSE
        FOR {i = 0 TO i = facets – 1}
                FOR {j = 0 TO j = 1}
                        p[j] = a[i] * x[j] + b[i] * y[j] + c[i] * z[j] + d[i];
                END FOR
                k = a[i] * xc + b[i] * yc + c[i] * zc + d[i];
                IF {p[0] = p[1]}
                        IF (p[0] * k < 0)
                                GOTO line;
                        END IF
```

```
                    ELSE
                        t = p[0]/(p[0] - p[1]);
                        L = x[1] - x[0]; M = y[1] - y[0]; N = z[1] - z[0];
                        x0 = x[0]; y0 = y[0]; z0 = z[0];
                        FOR {j = 0 TO j = 1}
                            IF {p[j] * k < 0}
                                x[j] = x0 + L * t;
                                y[j] = y0 + M * t;
                                z[j] = z0 + N * t;
                            END IF
                        END FOR
                    END IF
                    IF {(abs(x[0]-x[1])<1) AND (abs(y[0]-y[1])<1) AND (abs(z[0]-z[1])<1)}
                        GOTO line;
                    END IF
            END FOR
            SAVE (x[0], y[0], z[0], x[1], y[1], z[1]);
END IF
line:
END
```

2.3 Analysis of the Proposed Algorithm

In this section, the line number of the pseudo code (within the parenthesis) is used for description.

The input line segment can have two possibilities. The line segment can be a point (Line 3) or a line segment with non-zero length (Line 10). If the line segment is just a point and it lies outside at least one facet it can be directly rejected. In this way, the computational cost can be reduced. If the line segment has a non-zero length then line segment should be pruned against each facet (Line 11). If the line segment is parallel to a facet then the intersection point does not exist. When p[0] = p[1] the line segment is parallel to that facet (Line 16). In such cases, the facet is ignored, if the end points of the line segment lie inside the facet. And the line segment is ignored, if the end points lie outside the facet (Line 17 to Line 19).

The positions of end points after the line segment get pruned by each facet is dynamically calculated (Line 26 to Line 28). If the end points of the line segment coincides that means the line segment is completely outside with respect to the corresponding facet (Line 32), the original line segment is rejected as it is completely outside the clipping volume (S. R. Kodituwakku *et al*, 2013). Due to the precision error in computers testing exact equality is impossible. Therefore, approximate equality is tested. If abs(x[1] – x[0]) < 1 then it is assumed that x[1] = x[0].

3 Results

In order to validate the proposed algorithm, it was experimentally compared against Cyrus Beck (CS535, 2012) and Skala (V. Skala, 1997) line clipping

algorithms. The three algorithms were implemented using C++ programming language and following hardware and software resources were used.

Computer: Intel(R) Pentium(R) Dual CPU; E2180 @ 2.00 GHz; 2.00 GHz, 0.98 GB of RAM; IDE: Turbo C++; Version 3.0; Copyright(c) 1990, 1992 by Borland International, Inc;

Let T_{CB}, T_{SK} and T are the execution times of Cyrus Beck algorithm, Skala algorithm and the proposed algorithm respectively. Then the coefficients of efficiency can be defined as; $v1 = T_{CB}/T$; $v2 = T_{SK}/T$.

Random polyhedrons were created with different number of facets. The Table 1 shows some of those data. There Pn denotes a random polyhedron with n number of vertices. Random points were generated and they were considered as end points of line segment. The line generating space is a cube in which the volume is twice as that of the given polyhedron. Therefore the interior volume of the polyhedron is equal to the exterior volume of the polyhedron. The center of the polyhedron coincides with the center of the cube space. The purpose is to equally distribute the line segments in the space. For a given polyhedron, the number of clock cycles consumed to clip 10^8 number of line segments was computed. And then corresponding efficiency coefficients v1 and v2 were also computed.

Table 1. Number of clock cycles comparison for uniform line segments

Polyhedron	Cyrus Beck	Skala	Proposed	v1	v2
P4	4388	5048	4321	1.0155	1.1682
P5	4821	5492	4511	1.0687	1.2175
P6	5178	5857	4761	1.0875	1.2302
P7	5646	6328	4935	1.1440	1.2823
P8	5905	6556	5144	1.1479	1.2745
P9	6271	6892	5384	1.1647	1.2801

According to the set of polyhedrons shown in Table 1, the proposed algorithm is faster than both Cyrus Beck and Skala algorithms when line segments are generated randomly and uniformly in the space.

Depending on the position of line segments, they can be classified into three groups; line segment is completely outside the polyhedron, line segment is completely inside the polyhedron, line segment is intersecting the polyhedron.

The experiment was done with the same set of polyhedrons in Table 1 for 10^8 number of random line segments generated outside the polyhedron. The results are shown in Table 2.

Table 2. Number of clock cycles comparison for outside line segments

Polyhedron	Cyrus Beck	Skala	Proposed	v1	v2
P4	1782	1318	1283	1.3889	1.0273
P5	1971	1431	1657	1.1895	0.8667
P6	2134	1523	1742	1.2250	0.8743
P7	2317	1648	1887	1.2279	0.8733
P8	2415	1702	1971	1.2253	0.8635
P9	2578	1795	2068	1.2466	0.8680

According to the results shown in Table 2, the proposed algorithm is faster than the Cyrus Beck algorithm and slower than Skala algorithm when line segments are generated outside the polyhedron.

The experiment was done with the same set of polyhedrons in Table 1 for 10^8 number of random line segments generated inside the polyhedron. The results are shown in Table 3.

Table 3. Number of clock cycles comparison for inside line segments

Polyhedron	**Cyrus Beck**	**Skala**	**Proposed**	**v1**	**v2**
P4	1694	1243	972	1.7428	1.2788
P5	1867	1354	1276	1.4632	1.0611
P6	2032	1429	1361	1.4930	1.0500
P7	2208	1555	1489	1.4829	1.0443
P8	2287	1591	1541	1.4841	1.0324
P9	2453	1691	1629	1.5058	1.0381

According to the experimental results shown in Table 3, the proposed algorithm is faster than both Cyrus Beck and Skala algorithms when line segments are generated inside the polyhedron.

The experiment was done with the same set of polyhedrons in Table 1 for 10^8 number of random line segments generated intersecting the polyhedron. The results are shown in Table 4.

Table 4. Number of clock cycles comparison for intersecting line segments

Polyhedron	**Cyrus Beck**	**Skala**	**Proposed**	**v1**	**v2**
P4	6371	6567	4754	1.3401	1.3814
P5	6888	7142	6038	1.1408	1.1828
P6	7317	7617	6446	1.1351	1.1817
P7	7878	8223	6962	1.1316	1.1811
P8	8191	8527	7215	1.1353	1.1818
P9	8626	8961	7583	1.1375	1.1817

According to the experimental results shown in Table 4, the proposed algorithm is faster than both Cyrus Beck and Skala algorithms when line segments are generated intersecting the polyhedron.

The results shown in Table1, Table2, Table 3 and Table 4 indicate that the proposed algorithm is faster than the Cyrus Beck algorithm for any line segment. And the Skala algorithm is faster than the proposed algorithm only when the line segment is outside the polyhedron. The results in Table 2 and Table 3 indicate that the computational cost for clipping a line segment outside the polyhedron and the computational cost for clipping a line segment inside the polyhedron are approximately equal. Therefore the line segments can be divided into two fundamental categories: 0% intersections (line segment is either completely inside or completely outside the polyhedron) and 100% intersections (line segment is intersecting the polyhedron).

Then the algorithms were compared in order to observe what happens when the number of facets grows very large. Therefore convex polyhedrons were generated with N number of triangular facets (Skala, 1997). The number of clock cycles consumed to clip 10^8 line segments with 0% intersections was measured for each polyhedron. The experimental results are shown in Table 5.

Table 5. 0% intersections

N	10	20	50	100	200	500	1000
CB	2751	7638	23831	50115	100852	255178	511887
SK	5805	8557	17720	22615	34841	82209	160752
PP	1902	3809	9515	19024	38052	95092	190192
v1	1.4464	2.0053	2.5046	2.6343	2.6504	2.6835	2.6914
v2	3.0521	2.2465	1.8623	1.1888	0.9156	0.8645	0.8452

The results in Table 5 show that the proposed algorithm is faster than Cyrus Beck algorithm for a polyhedron with any number of facets when line segments are generated either completely inside or completely outside the polyhedron. Also the proposed algorithm is faster than Skala algorithm in this situation only when the number of facets of the polyhedron is less than 100.

For the same set of polyhedrons in Table 5, the experiment was carried out when there are 100% intersections. The results are shown in Table 6.

Table 6. 100% intersections

N	10	20	50	100	200	500	1000
CB	3978	9473	24752	50731	102076	258237	516781
SK	16492	19561	28109	30863	42487	90481	160748
PP	2648	4621	9772	19151	38415	96134	191902
v1	1.5023	2.0500	2.5330	2.6490	2.6572	2.6862	2.6929
v2	6.2281	4.2331	2.8765	1.6116	1.1060	0.9412	0.8377

The results in Table 6 show that the proposed algorithm is faster than Cyrus Beck algorithm for a polyhedron with any number of facets when line segments are generated intersecting the polyhedron. Also the proposed algorithm is faster than Skala algorithm in this situation only when the number of facets of the polyhedron is less than 500.

3 Discussion

The Cyrus Beck algorithm needs to compute all the intersection points with each facet of the polyhedron (CS 535, 2012). But the proposed algorithm avoids some of those intersection points depending on the sequence each facet is met inside the FOR loop. Therefore theoretically the proposed algorithm is faster. When the number of facets increases the speed of the proposed algorithm is more significant since the number of avoided intersections points also increase.

Experimental results shown also suggest that the proposed algorithm is faster. The required inputs to the proposed algorithm are equations of the facets and the Centroid of the polyhedron. But for the Cyrus Beck algorithm, it needs even the vertices of the polyhedron.

The Skala algorithm uses the known order of facets as an extra input. The proposed algorithm is faster than the Skala algorithm when the number of facets is less than 100.

The processing power of the CPU has the impact on the number of clock cycles needed to execute the algorithm. Doubling the processing power of the CPU will halve the execution time. This happens since all the three algorithms are sequential. All three algorithms use a single FOR loop to access each facet. Therefore the computational complexity of the three algorithms is O (N) where N is the number of facets. The Skala algorithm has the expected computational complexity O (sqrt (N)) using the known order of facets. The asymptotic computational cost comparison does not work for this kind of algorithms since they have equal asymptotic computational cost for larger values of N. Therefore experimental comparison was used.

4 Conclusion

A novel algorithm to clip line segments against a convex polyhedron was proposed. The theoretical and experimental analyses proved that the proposed algorithm is faster than the Cyrus Beck algorithm for a polyhedron with any number of facets. And also the proposed algorithm is faster than the Skala

algorithm for polyhedrons in which the number of facets is less than 100. The algorithm can be successfully extended to arbitrary dimensions to clip line segments against convex polytopes. Then each facet should be represented as $a_1x_1+a_2x_2+\ldots+a_nx_n + b = 0$.

References

D. Hearn and M. P. Baker (1998), *Computer Graphics*, C Version, 2nd Edition, Prentice Hall, Inc., Upper Saddle River, pp. 224-248.

W. Huang (2010), The Line Clipping Algorithm Based on Affine Transformation, *Intelligent Information Management*, Volume. 2, pp. 380-385.

S. R. Kodituwakku, K. R. Wijeweera, M. A. P. Chamikara (2013), An Efficient Algorithm for Line Clipping in Computer Graphics Programming, *Ceylon Journal of Science (Physical Sciences)*, Volume. 17, pp. 1 – 7.

CS 535 NOTES CYRUS-BECK CLIPPING ALGORITHM, http://cs1.bradley.edu/public/jcm/cs535CyrusBeck.html; accessed on 20 July 2012.

V. Skala (1997), A Fast Algorithm for Line Clipping by Convex Polyhedron in E^3, Computers & Graphics, Pergamon Press, Vol. 21, No. 2, pp. 209-214.