Applied Mathematical Sciences, Vol. 8, 2014, no. 24, 1177 - 1186
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ams.2014.4133

Line – Convex Polyhedron Intersection Using Vertex Connections Table

A. M. Konashkova

Ural Federal University
19 Mira St., Ekaterinburg, 620002, Russia

Abstract

Previously developed algorithm for intersection a line and a convex hull of points cloud is modified for the case of convex polyhedron. The algorithm uses only vertex connections table in contrast to all other known algorithms and provides a solution of the problem in case of unknown face list. This case appears in CAD and real-time geometric calculations. The algorithm is compared numerically with the most famous Cyrus-Beck algorithm and the direct algorithm.

Keywords: line, intersection, convex polyhedron, vertex, edge, face

1. Introduction

Intersection of lines, rays and segments against various geometrical objects is widely used in radiative heat transfer, computational geometry and computer graphics. In radiative heat transfer such objects are metal bars, furnace walls and mechanical assemblies. In computer graphics such objects are buildings, interior objects and animated characters [6]. Line/Ray – polyhedron intersection is central problem in ray tracing for rendering and also for radiation obstruction modeling [10]. There are many algorithms to solve this problem, but all of them use representation of a polyhedron as a set of polygonal or triangular faces. In other words those algorithms use face list. The face list is a table that contains numbers of vertices for each face. This table is stored in two-dimensional array, i.e. called *Faces*, such that *Faces(i,j)* – is a number of *j*-th vertex of *i*-th face. For a polyhedron in fig.1 the face list is shown in table 1.

This paper first presents alternative solution of line – convex polyhedron intersection problem using only vertex connections table (table 2). The algorithm doesn't use faces and line-face intersections. It is expected that such approach may be more flexible for various geometrical calculations.

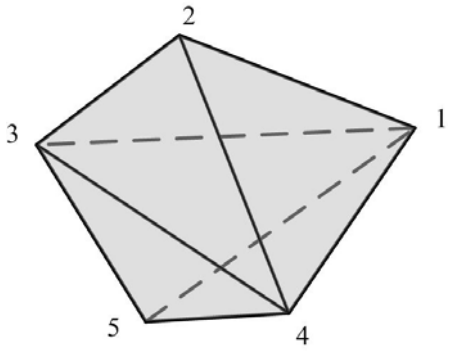


Fig. 1 Example of polyhedron

Table 1 Polyhedron's face list

Face	Vertices		
F1	1	2	3
F2	1	2	4
F3	1	3	5
F4	1	4	5
F5	2	3	4
F6	3	4	5

Table 2 Polyhedron's vertex connections table

Vertex	Connected vertices					Number of connections
V1	2	3	4	5		4
V 2	1	3	4			3
V 3	1	2	4	5		4
V 4	1	2	3	5		4
V 5	1	3	4			3

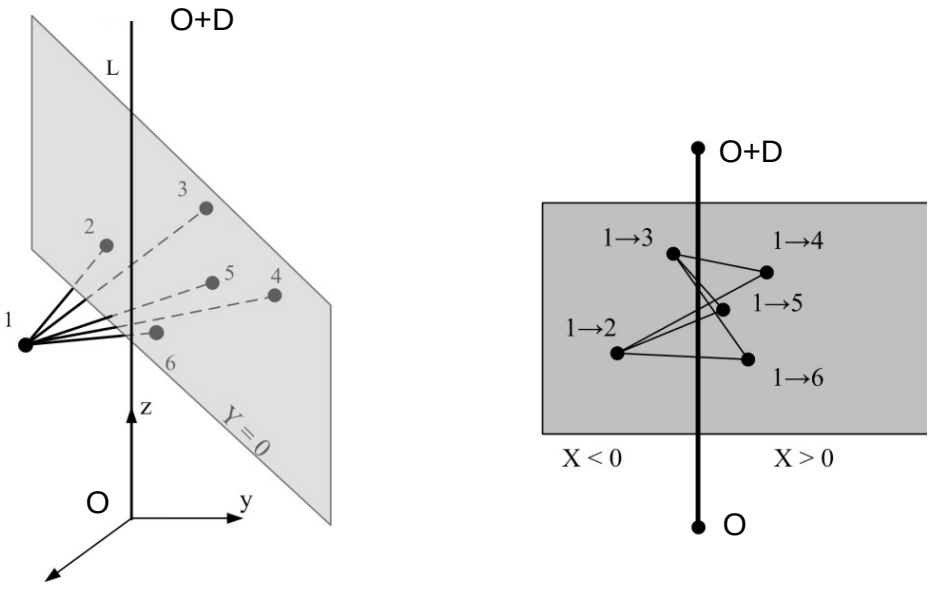
2. Previous work

Proposed intersection algorithm is based on previously developed algorithm of intersection a line and a convex hull of points cloud without convex hull construction [4].

The main idea of the second algorithm is construction of sections of the points cloud. First section of the points cloud by a plane containing the given line is constructed. A new planar points cloud is obtained. Intersection points of the line and initial points cloud are the same as intersection points of the line and obtained planar points cloud. New section of the planar points cloud by the given line is constructed. As a result, a 1D points set belonging to the line is obtained. Bounds of the 1D point set are the intersection points of the line and the convex hull of initial points cloud.

The algorithm:

1. Let us consider the line $L = O + t \cdot D$. Use transformation of coordinates to place O in the origin and $O + 1.0 \cdot D$ on the z-axis. In practice, x or y axis may be used.
2. Separate points into two groups: $P_{y<0}$ - with y coordinate less than zero, and $P_{y\geq 0}$ - with y coordinate not less than zero. If one of two groups is empty then L doesn't intersect the points cloud (its convex hull).
3. Connect each $p_i \in P_{y<0}$ with $p_j \in P_{y\geq 0}$ and compute $N_{y<0} \cdot N_{y\geq 0}$ points $P_{y=0}$, which lie at the polyhedron section by plane $y = 0$ (fig. 2a). The set $P_{y=0}$ is non convex, therefore 2D algorithms of line-polygon intersection can't be used.



a b

Fig. 2 Illustration of the algorithm: a – steps 2,3; b – steps 4,5; «1-2» - point that obtained by connection initial points 1 and 2.

4. Separate points $P_{y=0}$ into two groups: $P_{y=0,x<0}$ - with x coordinate less than zero, и $P_{y=0,x\geq 0}$ - with x coordinate not less than zero (fig. 2b). If one of two groups is empty then L doesn't intersect the points cloud (its convex hull).

5. Connect each $p_i \in P_{y=0,x<0}$ with $p_j \in P_{y=0,x\geq 0}$ and compute $N_{y=0,x<0} \cdot N_{y=0,x\geq 0}$ points $P_{x,y=0}$. They lie at the line L and have only z coordinate.
6. Find min z and max z coordinate of points $P_{x,y=0}$. Here z parameter is the same as parameter t in algorithms that use parametric line equation $L = O + t \cdot D$. So, we can compute intersection points $Po\text{int}_1 = O + \min z \cdot D$, $Po\text{int}_2 = O + \max z \cdot D$.

3. Modification of the algorithm for convex polyhedron

Obviously, developed algorithm of intersection a line and a convex hull of points cloud will work correctly if point cloud is already convex (convex polyhedron). In this case more efficient algorithm can be proposed. New algorithm has two main differences: transformation of coordinates is not used and vertices are connected with respect to vertex connection table.

The algorithm:

1. For line L compute two planes equations $F_1(x) = 0$ and $F_2(x) = 0$. We can always choose planes so that in plane equation $Ax + By + Cz + D = 0$ one of coefficients A,B or C is set to zero. For details see [7].
2. Compute distance D_1 from all vertices v to plane F_1.
3. Separate vertices into two groups: $V_{D1<0}$, and $V_{D1\geq 0}$ (fig. 3). If one of two groups is empty then line doesn't intersect the polyhedron. Otherwise go to step 4.
4. For each vertex $v_i \in V_{D1<0}$ a row in vertex connections table is available. Choose all vertices $v_j \in V_{D1\geq 0}$ from this row and connect them with v_i. Compute coordinates of points $P_{D1=0}$. The best way is to use the formula: $P_{D1=0} = \dfrac{v_i \left|D_1(v_j)\right| + v_j \left|D_1(v_i)\right|}{\left|D_1(v_i)\right| + \left|D_1(v_j)\right|}$. Noting, that $D_1(v_i) < 0$ and $D_1(v_j) \geq 0$, one obtain: $P_{D1=0} = \dfrac{v_i D_1(v_j) - v_j D_1(v_i)}{D_1(v_j) - D_1(v_i)}$.
5. Points $P_{D1=0}$ are unordered polygon vertices. Compute distance D_2 from all points $P_{D1=0}$ to plane F_2.

6. Separate points $P_{D1=0}$ into two groups: $P_{D2<0}$ and $P_{D2\geq 0}$ (fig. 4). If one of two groups is empty then line *L* doesn't intersect the polyhedron. Otherwise line *L* intersects the polyhedron.
7. Find two points $P_{\min D2}$ and $P_{\max D2}$ with min and max D_2 and compute normal form of the line equation $F_3(x)=0$. Separate points $P_{D1=0}$ into four groups: $P_{D2<0,D3\leq 0}$, $P_{D2<0,D3\geq 0}$, $P_{D2>0,D3\leq 0}$, $P_{D2>0,D3\geq 0}$ (fig. 4). Points $P_{\min D2}$ is included in both groups $P_{D2<0,D3\leq 0}$ and $P_{D2<0,D3\geq 0}$, and $P_{\max D2}$ is included in both groups $P_{D2>0,D3\leq 0}$, $P_{D2>0,D3\geq 0}$.
8. Find points $P_{\text{down left}} \in P_{D2<0,D3\leq 0}$ with maximum value D_2 and $P_{\text{down rightt}} \in P_{D2\geq 0,D3\leq 0}$ with minimum value D_2. These two points are located on opposite sides of D_2. Noting that $L = D_1 \cap D_2$, find two points on opposite sides of *L*. So, first line – polyhedron intersection point is a point of intersection the line *L* with segment $P_{\text{down left}}P_{\text{down rightt}}$.
9. Find points $P_{\text{up left}} \in P_{D2<0,D3\geq 0}$ with maximum value D_2 and $P_{\text{up rightt}} \in P_{D2\geq 0,D3\geq 0}$ with minimum value D_2. Second line – polyhedron intersection point is a point of intersection the line *L* with segment $P_{\text{up left}}P_{\text{up rightt}}$.

The described algorithm uses only vertices coordinates and vertex connections table, and no faces reconstruction from the table is used. This algorithm has *O(N)* complexity.

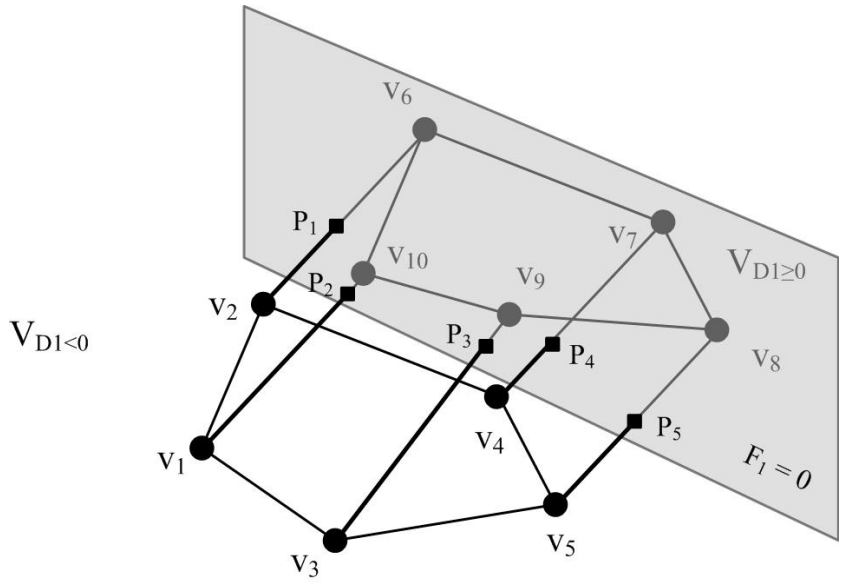


Fig. 3 Illustration of the proposed algorithm. Constructing a polyhedron section by plane $F_1(x)=0$ (steps 3-4)

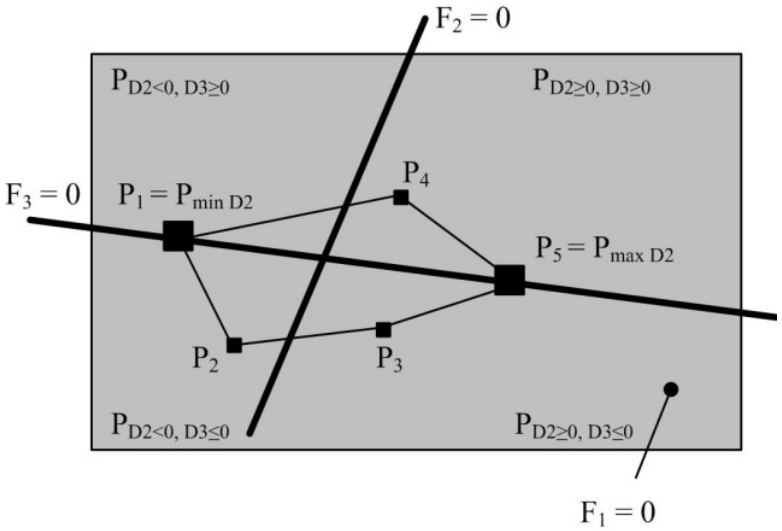


Fig. 4 Illustration of the proposed algorithm. Points $P_{Dl=0}$ are separated into four groups: $P_{D2<0,D3\le 0} = \{P_1, P_2\}$, $P_{D2<0,D3\ge 0} = \{P_1\}$, $P_{D2>0,D3\le 0} = \{P_3, P_5\}$, $P_{D2>0,D3\ge 0} = \{P_4, P_5\}$. Here $P_{\text{down left}} = P_2$, $P_{\text{down right}} = P_3$, $P_{\text{up left}} = P_1$, $P_{\text{up right}} = P_4$. Two line – polyhedron intersection points are $L \cap P_2P_3$ and $L \cap P_1P_4$

4. Known algorithms of line – convex polyhedron intersection for comparison

There are two general algorithms with *O(N)* complexity: direct computational algorithm and the Cyrus-Beck algorithm. Both of them are very popular up to date.

The direct computational algorithm [2] performs direct line – triangle intersection for each triangular face of the given polyhedron while two intersections are not found. This algorithm is well known and very useful.

The algorithm uses point coordinates **O** at the line, the direction vector **D**, a list of *Nv* polyhedron vertices coordinates $\mathbf{P_i}$, the number of vertices *Nv,* and the face list as input data. The outputs are t_1,t_2, and the intersection points $\mathbf{Point_1}$ and $\mathbf{Point_2}$.

The Cyrus-Beck algorithm [1] uses the fact that a convex polyhedron can be understood as the intersection of halfspaces. Boundaries of these halfspaces are formed by planes in which faces of the polyhedron lie. Suppose we have a convex polyhedron and a line with some parametrization. Searching for the intersection of these geometrical objects, we can divide the bounding planes of the polyhedron into two groups according to the orientation of their normal vectors. Among the planes oriented towards the observer, we search for the point of intersection with the maximal parameter value t_l. Among planes of the other group, the minimal

parameter value t_2 is found. If $t_1 > t_2$, the intersection of the polyhedron with the given line does not exist. If $t_1 \leq t_2$, compute the points of intersection [1]. This process is illustrated in fig. 5 for 2D case.

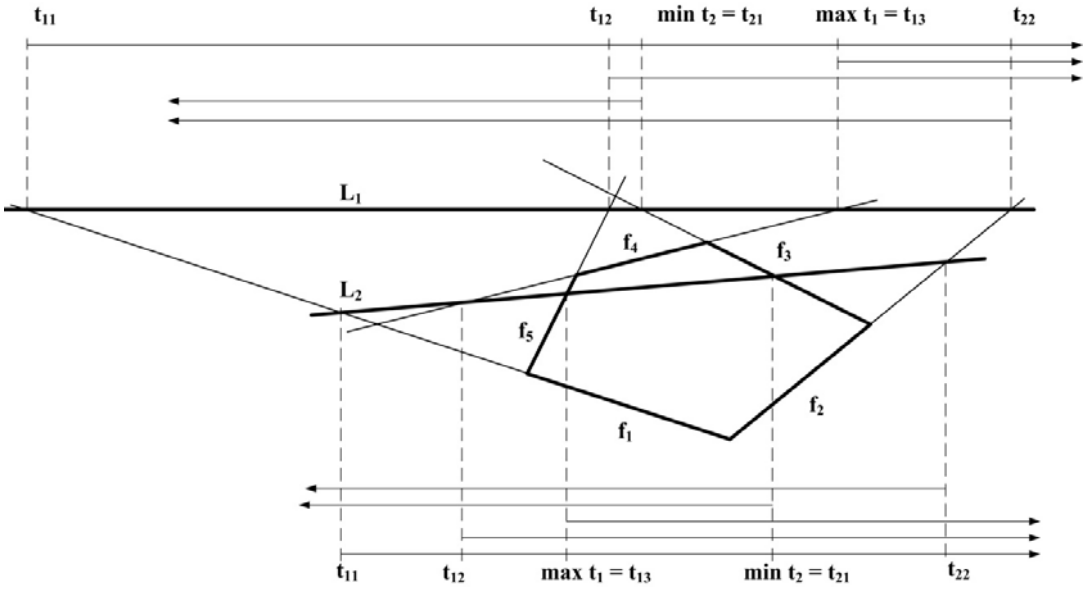


FIg. 5 Finding intervals along parameter t, where line intersects the polyhedron given by facets f_1-f_5. Line L_1 doesn't intersect the polyhedron because $\max(t_1)>\min(t_2)$. Line L_2 intersects the polyhedron because $\max(t_1)<\min(t_2)$

The CB algorithm uses point coordinates **O** at the line, the direction vector **D**, faces normals $\mathbf{N_i}$: $i=1…N$, a list of d_i such that $d_i = -\mathbf{N_i}\cdot\mathbf{P_i}$, and the number of faces N as input data. The outputs are t_1,t_2, and the intersection points $\mathbf{Point_1}$ and $\mathbf{Point_2}$. Each face is presented by a halfspace in this algorithm, so, the Cyrus-Beck algorithm *needs the face list*.

5. Performance comparison

The proposed algorithm was tested against popular Cyrus-Beck algorithm and the direct computational algorithm. Algorithms were tested for number of polyhedron vertices Nv from 4 to 120. For each Nv, 100 randomly generated convex polyhedra were used, polyhedra were inscribed into the cube that has minimal bounds $x,y,z = -1$ and maximal bounds $x,y,z = +1$. For each Nv and for each polyhedron, $2\cdot10^5$ lines were used, so, $2\cdot10^7$ lines were used for each Nv. Data sets of two points that define a line were generated such that first and second points of each line lie on two different cube faces, see fig. 6. Such lines and

polyhedra vertices arrangement is the most right and practical one because in practice line – object intersection calculation should be always the second step after intersection the line with polyhedron bounding box. The algorithm described in [5] was used as the line-triangle intersection test in the direct computational algorithm.

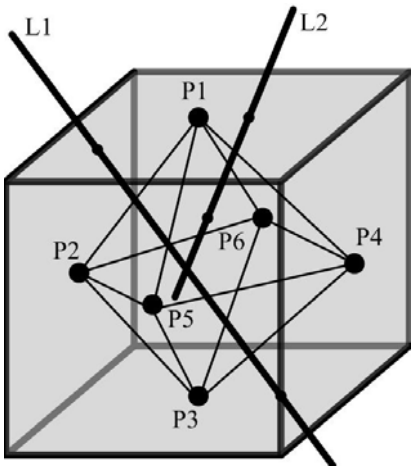


Fig. 6 Lines and points arrangement. Data sets of two points that define a line are located in the bounding box of points cloud

Runtimes of algorithms in seconds per million lines are given in fig. 7. All tests were implemented in Fortran on Intel Pentium II 1.83 GHz. It can be seen that efficiency of the proposed algorithm is comparable to the most useful Cyrus-Beck algorithm's efficiency. New algorithm is the fastest one for $Nv >$ 70-80. This proves that using the vertex connections table is applicable in practice. In case of unknown face list proposed algorithm will be the most efficient.

6. Conclusions

A new algorithm for line – polyhedron intersection is developed and tested. The algorithm uses only vertex connections table and is suitable in cases when polyhedron faces list is unknown. These cases appear in CAD and in real time geometric calculations.

Algorithm efficiency is comparable to simple face-based algorithms efficiency and even better for large number of polyhedron vertices. There are faster face-based algorithms [3,8,9], but they all use face list. Anyway, proposed algorithm is the first one that doesn't use polyhedron face list, and there is a hope that edge-based approach will provide new solutions of geometric and graphical

problems. Possible subjects for future work is development of edge-based algorithm with O($\sqrt{N}$) complexity like the one of face-based algorithms [9].

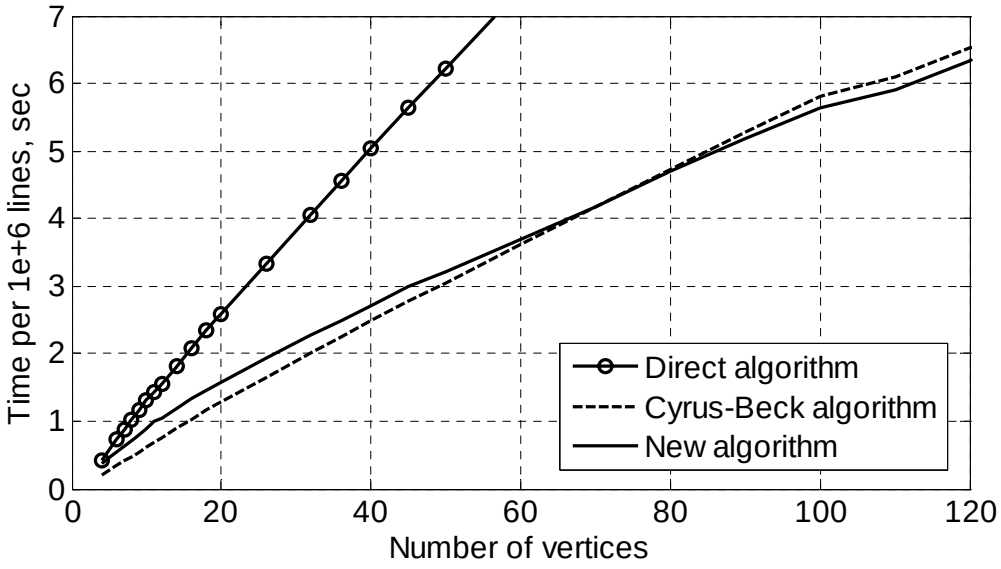


Fig. 7 Algorithms performance

References

[1] M. Cyrus, J. Beck, Generalized two and three dimensional clipping. Computers & Graphics, (1979), 3, 23-28.

[2] I. Kolingerova 3D - Line Clipping Algorithms - A Comparative Study, The Visual Computer, Vol.11, No.2, (1994), pp.96-104.

[3] I. Kolingerova, Convex polyhedron-line intersection detection using dual representation. The Visual Computer 13(1), (1997), pp. 42–49.

[4] R. Koptelov, A. Konashkova, Intersection Of A Line And A Convex Hull Of Points Cloud. Applied Mathematical Sciences, 7(103), (2013), pp. 5139-5149.

[5] T. Müller and B. Trumbore, Fast, Minimum Storage Ray-Triangle Intersection, Journal of Graphics Tools, pp. 37-46, (1997), pp.22-28.

[6] P. Shirley, Fundamentals of computer graphics , Second Edition, AK Peters, (2005) 623 p.

[7] V. Skala, An Efficient Algorithm for Line Clipping by Convex and Non-Convex Polyhedrons in E3, Computer Graphics Forum, Vol.15, No.1, (1996), pp.61-68.

[8] V. Skala, Line Clipping in E3 with Expected Complexity O(1), Machine Graphics and Vision, Poland Academy of Sciences, (1996).

[9] V. Skala, A Fast Algorithm for Line Clipping by Convex Polyhedron in E3, Computers & Graphics, Pergamon Press, Vol.21, No.2, (1997), pp.209-214.

[10] I. Wald, W.R. Mark, J. Gunther, S. Boulos, T. Ize et al. State of the Art in Ray Tracing Animated Scenes, Computer graphics forum, 28(6), (2009), pp. 1691-1722.

Received: January 20, 2014