Applied Mathematical Sciences, Vol. 7, 2013, no. 103, 5139 - 5149
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ams.2013.37372

Intersection of a Line and a Convex Hull of Points Cloud

R. P. Koptelov

Ural Federal University
19 Mira St., Ekaterinburg, 620002, Russia
r-koptelov@mail.ru

A. M. Konashkova

Ural Federal University
19 Mira St., Ekaterinburg, 620002, Russia
a_konashkova@mail.ru

Copyright © 2013 R. P. Koptelov and A. M. Konashkova. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

An algorithm for intersection a line and a convex hull of points cloud is presented. The algorithm doesn't require the convex hull construction. The points cloud can be arbitrary and not sorted, no topology, face list or edge list is known. The algorithm uses only vertices coordinates. Standard transformation of coordinates is performed and the points cloud is bisected by two perpendicular planes. Yielded 1D points set lies at the line. Bounds of the set are intersection points of the points cloud and the line. The algorithm was compared against the obvious algorithm which uses intersection of the line and all possible faces (sets of three points). Presented algorithm is much faster than the obvious one.

Keywords: line, intersection, convex hull, 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].

In usual applications static geometrical models are used which are represented by planar (triangle, quadrilateral, polygon) or volumetric (tetrahedron, hexahedron, cube, sphere) objects. For such models the topology – face list, edge list or a graph – is already known or can be determined once before all operations with the model. Here, face list is a table containing vertices numbers for each face, edge list is a table containing vertices numbers for each edge. The topology can be determined by construction of the convex hull. There are many algorithms of convex hull construction in 2D, 3D and kD: Graham's method, «divide and conquer» method, «gift wrapping» and others – see classic book [5]. But really determining the topology is often not needed because it consumes too much time and because it is often used only as a preprocessing step before intersection tests. Such case can be occurred in animation and visualization of non rigid objects. For such objects their topology can't be determined beforehand, so handling of non rigid objects becomes more and more actual problem [11].

This paper presents an algorithm of intersection a line and a convex hull of points cloud without the convex hull construction. The paper is organized as following:

1) Known algorithms of intersection of a line and convex polyhedron are described;
2) Obvious approach for line – convex hull intersection is given. The algorithm intersects a line against all combinations of 3 points assuming each 3 points may be a face of the convex hull (face representation);
3) A new approach for line – convex hull intersection is given. In contrast to face representation, the algorithm assumes that each 2 points may be an edge of the convex hull (edge representation);
4) Performance comparison of the algorithms 2 and 3.

2 Known algorithms of line – convex polyhedron intersection

2.1 Early algorithms with O(N) complexity

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

The CB algorithm uses point coordinates **O** at the line, the direction vector **D**, faces normals $\mathbf{N_i}$: $i=1\ldots N$, 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*.

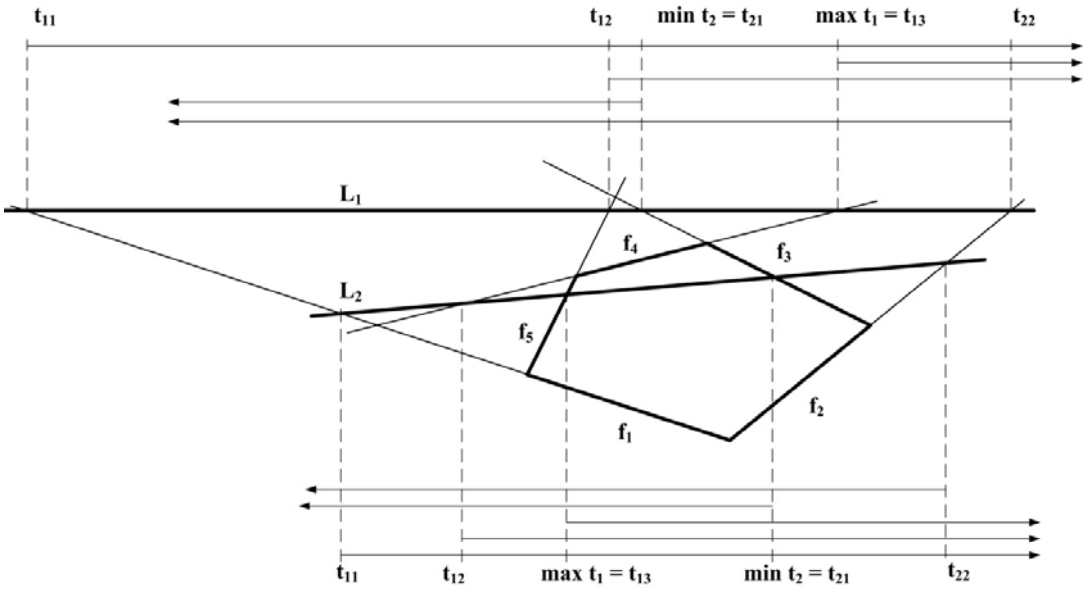


FIg. 1 Finding intervals along parameter t, where line intersects the polyhedron given by facets f_1-f_5. Line L_1 doesn't intersect the polyhedron because $\max(t_1)>\min(t_2)$. Line L_2 intersects the polyhedron because $\max(t_1)<\min(t_2)$

2.2 Plane tested algorithms

The main idea to accelerate two early algorithms is to reject polyhedron faces before the main computing. The one of possible ways is to test line – bounding volume intersection first. However, this strategy is applied usually to all geometry but not to individual polyhedron faces.

Another idea was proposed by V. Skala in [7] for triangular faces. A line L_1 can be defined as an intersection of two nonparallel planes p_1 and p_2. If the line L_1 intersects the given triangle then planes p_1 and p_2 intersect the given triangle, too, but if planes p_1 and p_2 intersect the triangle then the line can intersect (line L_1 and planes p_1 and p_2) or miss the triangle (line L_2 and planes p_3 and p_4), see fig.2. Then we can test each triangle of the given polyhedron against p_1 and p_2 planes before detailed line – triangle or line – halfspace intersection computation. If both planes intersect the given triangle (facet) then use detailed intersection test. The intersection of the given plane p_i and the triangle exists if and only if two vertices $\mathbf{x_j}$ and $\mathbf{x_k}$ of the triangle exist so that $sign(F_i(x_j)) \neq sign(F_i(x_k))$, where $F_i = 0$ is an equation for the *i*-th plane p_i, *i=1,2*.

This test can be applying with direct computational algorithm or with Cyrus-Beck algorithm and computing time may be decreased 2-4 times for 100 polyhedron faces or 34 – 6.1 times for 1000 polyhedron faces [2].

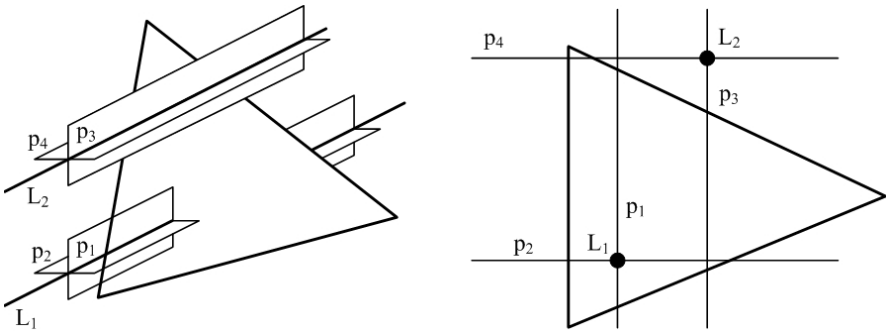


Fig. 2 Usage of two planes for line definition

The rejection test allows one to determine an edge intersected by planes p_1 or p_2 and next triangle shared by this edge. Thus we can test not all triangles against plains p_1 and p_2, but only ring of triangles which intersected by plane p_1. By following from one triangle to next triangle with common edge (see fig. 3) we can test only $O(\sqrt{N})$ triangles. This algorithm described in details in [10].

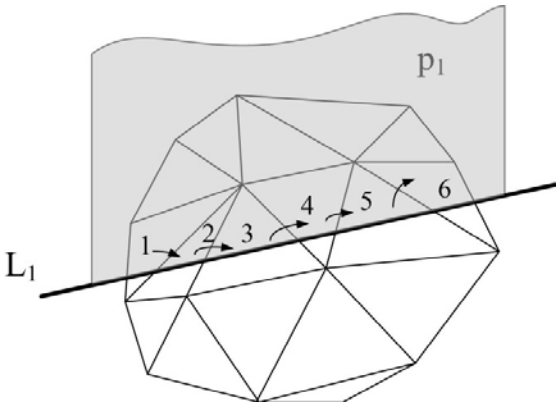


Fig. 3 Testing sequence of triangles with common edge

2.3 Dual space algorithms

A line in E^2 can be described by an equation $ax+by+c=0$ and rewritten as $y=kx+q$, if $|k|\le 1$, $b\neq 0$ or $x=my+p$, if $|m|<1$, $a\neq 0$.

It means, that the line L in E^2 can be represented using an asymmetrical model of dual space [3] representation as a point $D(L)=\{k,q\}\in D(E^2)$ or $D(L)=\{m,p\}\in D(E^2)$. Inversely, a point in E^2 can be represented as a line in $D(E^2)$. A point in E^3 can be represented as a plane in $D(E^2)$, and inversely, a plane in E^3 can be represented as a point in $D(E^2)$.

These relations allow us to transform a line – polyhedron intersection test into point in polygon test problem. Dual space representation was used for line clipping in [2,3,8]. V. Skala proposed a line clipping algorithm with *O(1)* complexity [9]. The line clipping problem is reduced into point in polygon problem. And point in polygon problem is solved by constant time search on uniform grid.

This is a great advancement for solving the problem if a large time for precomputing is available and the face list is known. Anyway, dual space algorithms also use the face list of polyhedron.

3. Intersection a line and a convex hull of points cloud without convex hull construction

3.1 The obvious algorithm

Potentially any three points may be a face of the convex hull. Obvious algorithm intersects a line and each potential face and calculates the value of $\mathbf{t_i}$ – distance from point on the line to the intersection point. Max and min of all $\mathbf{t_i}$ represent two intersections point of the line and the convex hull of points cloud. Loop over each set of three points (3 nested loops over point number) leads to $O(N^3)$ complexity.

This algorithm can be simplified if coordinates of intersection points are not needed. In this case the algorithm will be terminated if first line-triangle intersection is found.

3.2 New algorithm

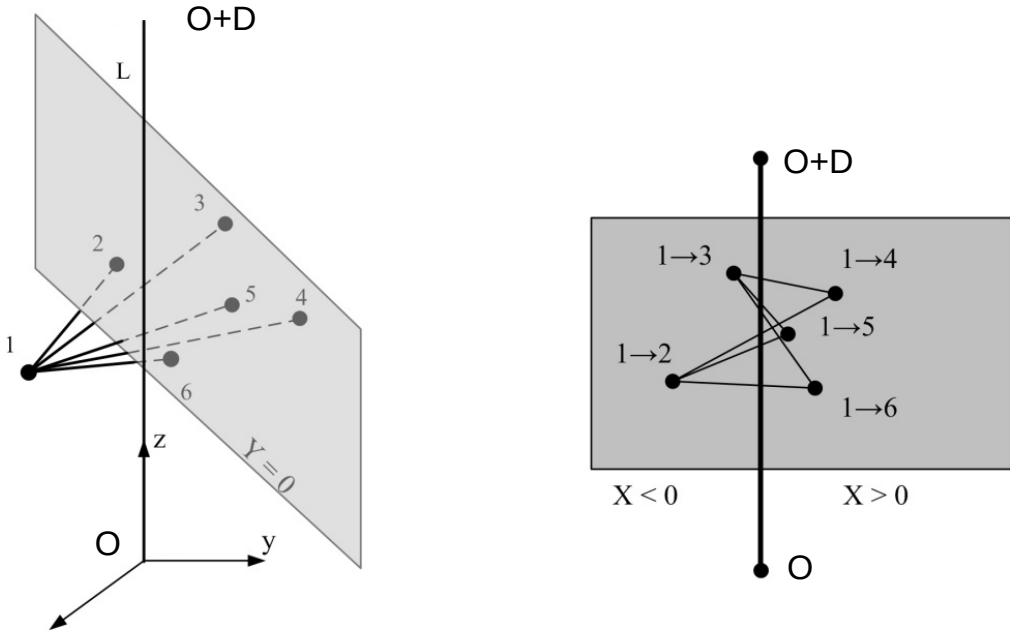
The main ideas of the algorithm are refusing of face representation of convex hull and construction of sections of the points cloud. First section of the points cloud by a plane containing the given line is constructed. A new planar points cloud is obtained. Intersection points of the line and initial 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. 4a). The set $P_{y=0}$ is non convex, therefore 2D algorithms of line-polygon intersection can't be used.

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. 4b). 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 $Point_1 = O + \min z \cdot D$, $Point_2 = O + \max z \cdot D$.

Illustration of the algorithm executing for non-convex points set is given in fig. 5.



a b

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

In the worst case P is separated by plane y = 0 into equal parts: $N_{y<0} = N_{y\geq 0} = N/2$, where N - number of points. $P_{y=0}$ also is separated by plane x = 0 into equal parts: $N_{y=0,x<0} = N_{y=0,x\geq 0} = \left(N_{y<0} N_{y\geq 0}\right)/2 = N^2/8$. Total

number of connections is $N_{y=0,x<0} \cdot N_{y=0,x\geq 0} = N^4/64$. So, the worst case complexity of the algorithm is O(N^4). In the best case: $N_{y<0} = 1$, $N_{y\geq 0} = N-1$ and $N_{y=0,x<0} = 1$, $N_{y=0,x\geq 0} = (N-1)-1$. So, the best case complexity of the algorithm is O(N).

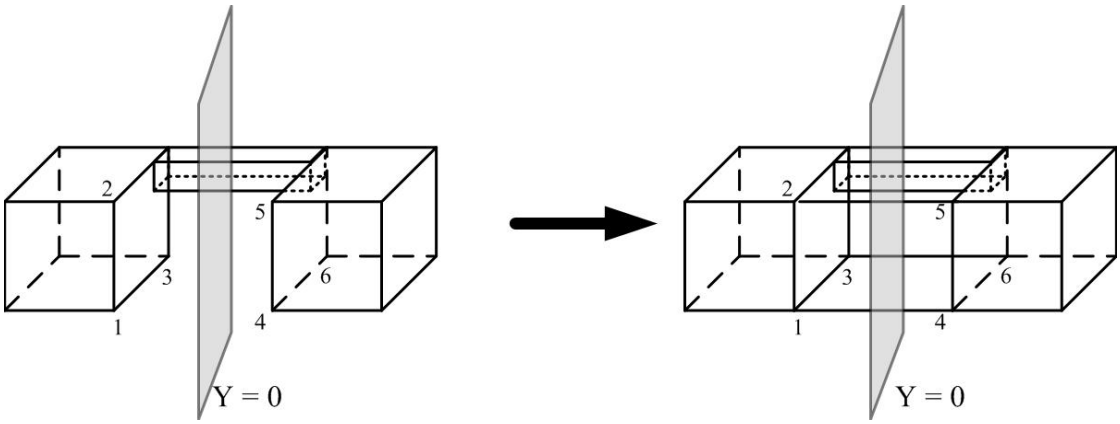


Fig. 5 Illustration of the algorithm executing for non-convex points set. Initial points set – two cubes connected with a parallelepiped. If each $p_i \in P_{y<0}$ connect with $p_j \in P_{y\geq 0}$, then points 1 and 4, 2 and 5, 3 and 6 will be connected and the convex hull of points set will be used further.

This algorithm is suitable not only for calculations with non rigid points clouds or if face list and edge list are unknown, but also for CAD applications. For example, one can manually pick a point subset on the screen without connecting the points and send the data to the intersection calculations module, while connection the points by hand is time consuming and must not be made by man.

The algorithm can also be simplified if intersection points are not needed. In this case steps 1-3 are not changed. Step 4 has to be changed: 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. If one of two groups is empty then L doesn't intersect the points cloud (its convex hull), otherwise there is an intersection.

5. Performance comparison

The new algorithm was tested against the obvious one. Algorithms were tested for number of points *N* from 4 to 100. For each *N*, 100 randomly generated

points clouds were used, the clouds were inscribed into the cube that has minimal bounds $x,y,z = -1$ and maximal bounds $x,y,z = +1$. For each N and for each points cloud, $2 \cdot 10^5$ lines were used, so $2 \cdot 10^7$ lines were used for each N. 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 points 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 object's bounding box.

Runtimes of algorithms in seconds per million lines are given in table 1. All tests were implemented in Fortran on Intel Pentium II 1.83 GHz. The algorithm described in [4] was used as the line-triangle intersection test in the obvious algorithm.

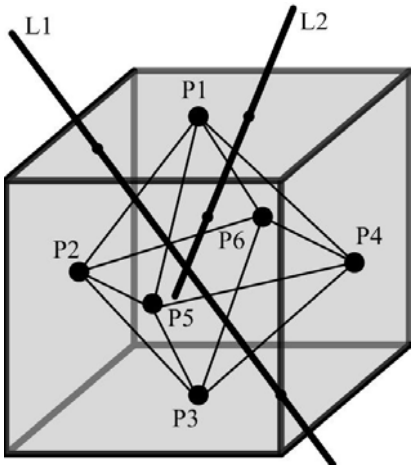


Fig. 6 Lines and points arrangement. Data sets of two points that define a line are located in the bounding box of points cloud

It can be seen that both for calculation of intersection points and for intersection test the proposed algorithm is faster than the obvious one for N>4. If N=4 then the points cloud is a tetrahedron and its topology is really known: each three points form a face of the tetrahedron and each two points form an edge. If only intersection test is required, the proposed algorithm is 10 times faster than the obvious one if N=18 and it is 34 times faster if N=100. If calculation of intersection points is required, performance of the proposed algorithm grows as N increases to 16 (5.9 times superiority), but further the performance decreases.

6. Conclusions

The algorithm for intersection a line and a convex hull of points cloud is presented. The algorithm doesn't require of convex hull construction. The algorithm is interesting for computer graphics, readiative heat transfer and compu-

tational geometry. The main ideas of the algorithm are refusing of face representation of convex hull and construction of sections of the points cloud. The proposed algorithm was tested against the obvious algorithm which intersects a line and each set of three points (each potential face). It is shown that the proposed algorithm has superior performance especially for intersection tests. Possible subject for future work is application of the algorithm for convex polyhedron with known edge list and unknown face list. It is expected that the algorithm will have O(N) complexity and its performance will be comparable with those of known line – convex polyhedron intersection algorithms.

Table 1. Algorithms performance

Number of points	**Obvious algorithm**		**Proposed algorithm**	
	Test of intersection	**Calculation of intersection points**	**Test of intersection**	**Calculation of intersection points**
4	0.36	0.47	0.49	0.56
5	**0.70**	**1.06**	**0.61**	**0.69**
6	1.19	2.07	0.66	0.87
7	1.86	3.60	0.77	1.12
8	2.72	5.65	0.87	1.42
9	3.79	8.25	0.99	1.81
10	4.98	11.66	1.11	2.31
11	6.41	15.88	1.24	2.97
12	8.13	20.99	1.37	3.71
14	12.14	34.20	1.65	5.83
16	17.61	51.97	1.97	8.85
18	23.67	76.40	2.35	13.02
20	29.97	106.40	2.69	19.32
26	56.38	238.00	4.00	51.28
32	94.33	450.90	5.69	113.69
36	130.09	648.30	6.81	178.23
40	176.72	893.70	8.22	263.11
45	228.28	1328.60	10.37	422.89
50	288.00	1741.30	12.18	699.20
60	441.53	3057.20	16.92	1381.10
70	642.91	4855.90	22.26	2516.90
80	888.42	7263.40	28.92	4343.70
90	1151.15	10445.50	36.70	6653.20
100	1524.37	14476.40	44.50	10534.90

References

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

[2] I. Kolingerova 3D - Line Clipping Algorithms - A Comparative Study, The Visual Computer, 11(2), (1994), 96-104.

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

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

[5] F.P. Preparata, M.I. Shamos, Computational Geometry - An Introduction, Springer-Verlag, 1985.

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

[7] V. Skala, An Efficient Algorithm for Line Clipping by Convex and Non-Convex Polyhedrons in E3, Computer Graphics Forum, 15(1), (1996), 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, Trading Time for Space: an O(1) Average time Algorithm for Point-in-Polygon Location Problem. Theoretical Fiction or Practical Usage? Machine Graphics and Vision, 5(3), (1996), 483-494.

[10] V. Skala, A Fast Algorithm for Line Clipping by Convex Polyhedron in E3, Computers & Graphics, Pergamon Press, 21(2), (1997), 209-214.

[11] 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), 1691-1722.

Received: July 5, 2013