

ZÁPADOČESKÁ UNIVERZITA

Americká 42, 306 14 Plzeň

AN EFFICIENT ALGORITHM FOR LINE
CLIPPING BY CONVEX AND
NON-CONVEX POLYHEDRONS IN E³

Václav Skala

Preprinty vědeckých prací

Preprint č. 64

1994

**An Efficient Algorithm for Line Clipping
by Convex and Non-convex Polyhedrons in E^3**

Václav Skala

Department of Informatics and Computer Science

University of West Bohemia

Americká 42, Box 314, 306 14 Plzeň

Czech Republic

e-mail: skala@kiv.zcu.cz

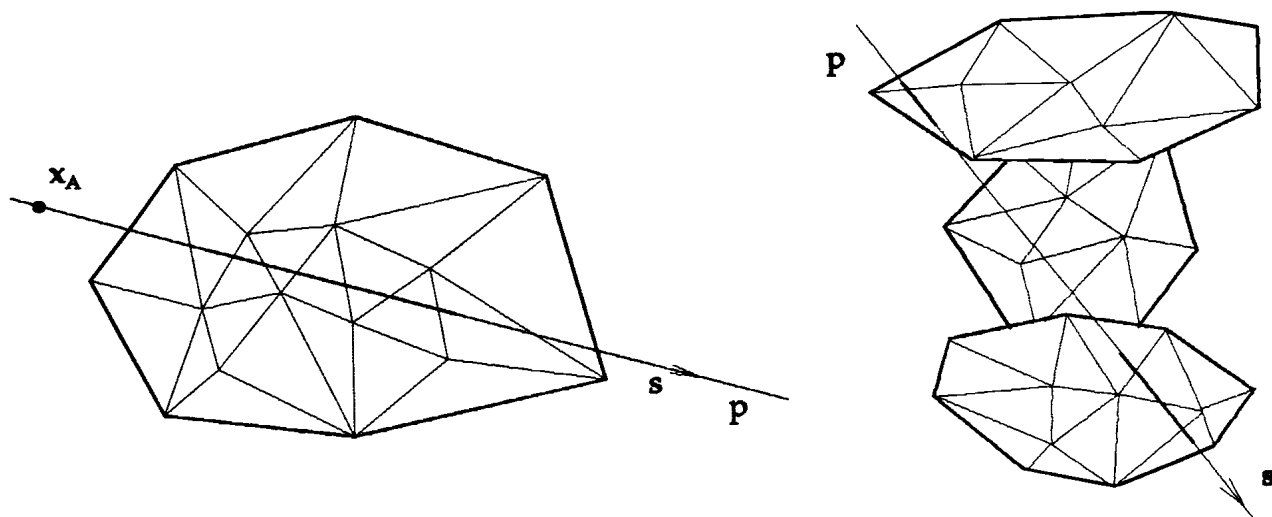
Abstract

A new algorithm for clipping lines against convex polyhedron with $O(N)$ complexity is given with modification for non-convex polyhedron. The suggested algorithm is faster for higher number of facets of the given polyhedron than the traditional Cyrus-Beck's algorithm. Some principal results of comparisons of all algorithms are shown and give some imagination how the proposed algorithm could be used effectively.

Keywords: Line Clipping, Convex Polyhedron, Non - Convex Polyhedron, Computer Graphics, Algorithm Complexity, Geometric Algorithms, Algorithms Analysis.

1. Introduction

A problem of line clipping against convex polyhedron in E^3 can be solved by Cyrus-Beck's algorithm (CB) [1] for three dimensional case. Many algorithms for line clipping in E^2 have been published so far with $O(N)$ or $O(\lg N)$ complexities, see [3], [4] for all known references and comparison of algorithms. Nevertheless algorithms for E^3 case are mostly based on the CB algorithm and restricted to convex polyhedrons, or based on direct intersection computation of a facet (usually a triangular facet) of the given convex or non-convex polyhedrons and the given line p . Many algorithms for line clipping are restricted to orthogonal or pyramidal volumes, see [2]. Because the line clipping problem solution is a bottleneck of many packages and applications it would be desirable to use the fastest algorithm even it is of complexity $O(N)$.



Line clipping by convex and non-convex polyhedrons in E^3
Figure 1

```

procedure Clip_3D_Cyrus_Beck (  $x_A$  ,  $x_B$  );
begin { !! all vectors  $n_i$  precomputed !! algorithm shortened }
   $t_{\min} := 0.0$ ;   $t_{\max} := 1.0$ ;       $s := x_B - x_A$ ;
  { for a line initialization:       $t_{\min} := -\infty$ ;   $t_{\max} := \infty$ ; }
   $i := 1$  ;
  while  $i \leq N$  do                {  $N$  is a number of facets }
  begin {  $n_i$  is a normal vector of the  $i$ -th facet }
    { and  $n_i$  must point out of the convex polyhedron }
     $\xi := s^T n_i$ ;     $s_i := x_i - x_A$ ;
    if  $\xi \neq 0.0$  then
    begin  $t := s_i^T n_i / \xi$ ;
      if  $\xi > 0.0$  then  $t_{\max} := \min ( t , t_{\max} )$ 
      else  $t_{\min} := \max ( t , t_{\min} )$ 
    end
    else { line is parallel to the  $i$ -th facet }
      Special case solution;
     $i := i + 1$ 
  end;

```

```

if  $t_{\min} > t_{\max}$  then { the line doesn't intersect the polyhedron }
    EXIT; { !!  $\langle t_{\min}, t_{\max} \rangle = \emptyset$  }
{ recompute end-points of the line segment if changed }
{ for lines points  $x_A$  ,  $x_B$  must be recomputed always }
if  $t_{\max} < 1.0$  then  $x_B := x_A + s t_{\max}$ ;
if  $t_{\min} > 0.0$  then  $x_A := x_A + s t_{\min}$ ;
SHOW_LINE(  $x_A$  ,  $x_B$  );
end { Clip_3D_Cyrus_Beck };

```

Algorithm 1

The main disadvantage of the CB algorithm is direct line intersection computation for all planes which form the boundary of the given convex polyhedron. It means that $N-2$ of intersection computations are wasted, if N is a number of facets of the given convex polyhedron.

The main advantages of the CB algorithm are its stability and that facets that forms the polyhedron surface might be generally polygons and.

The efficiency of the CB algorithm is given by a simple algorithm for direct intersection computation of a line with a plane, see alg.1. It is obvious that with growing number of facets of the given polyhedron the efficiency decreases as many invalid intersection points are computed.

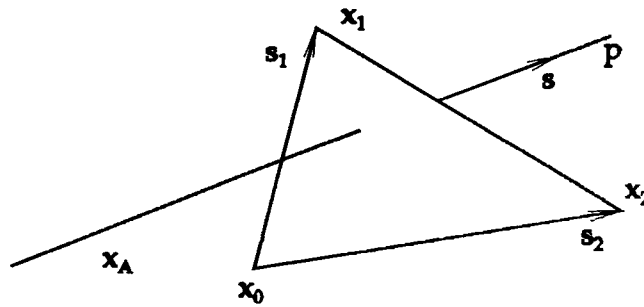
Any facet of a surface of any convex polyhedron can be imagined as a non-linear transformation

$$x = x(p, q) \quad p \in \langle p_A, p_B \rangle \quad \& \quad q \in \langle q_A, q_B \rangle$$

from E^2 (parametric space) to E^3 .

It can be seen that number of facets grows with square of u and v interval partitioning (if we consider a grid of 20×20 triangles in (p, q) space we will obtain a polyhedron with 760 non-degenerated facets as a sphere approximation!).

Let us consider only triangular facets of the given polyhedron for the following (generally it is not necessary). For the given line p (the line p varies and polyhedron is more or less stable!) it is necessary to find an effective test whether the line p intersects the given triangle, see fig.2.



Line p intersects the triangular facet

Figure 2

The intersection points for convex and non-convex polyhedrons can be directly computed as a solution of parametric equations

$$\mathbf{x}(t) = \mathbf{x}_A + \mathbf{s} t \quad t \in (-\infty, \infty) \quad (\alpha)$$

$$\mathbf{x}(p, q) = \mathbf{x}_0 + \mathbf{s}_1 p + \mathbf{s}_2 q \quad (\beta)$$

$$p, q \in \langle 0, 1 \rangle \ \& \ p + q \leq 1$$

e.g. in the matrix form

$$\begin{bmatrix} \mathbf{s}_1 & \mathbf{s}_2 & -\mathbf{s} \end{bmatrix} \cdot \begin{bmatrix} p \\ q \\ t \end{bmatrix} = \mathbf{x}_A - \mathbf{x}_0$$

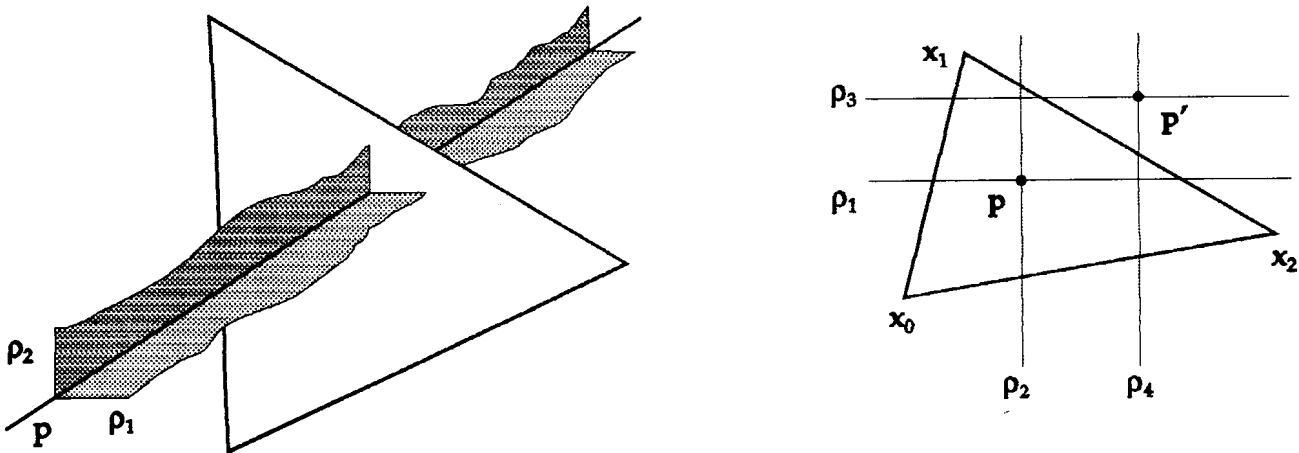
where α is an equation for a given line,

β is an equation for a given triangular facet, see fig.3.

Unfortunately a test which rejects all lines not intersecting a triangular facet is nearly as complex as the direct computation of the intersection point between the extended facet and the line, used in the CB algorithm. In order to substantially speed up line clipping in E^3 , we will instead develop a simple test, which on the other hand only rejects some of the lines not intersecting the facet.

2. Proposed algorithm

A line p can be defined as an intersection of two non-parallel planes ρ_1 and ρ_2 . It can be seen that if the line p intersects the given triangle then planes ρ_1 and ρ_2 intersect the given triangle, too, but not vice versa (planes ρ_3 and ρ_4), see fig.4.



Usage of two planes for line definition

Figure 3

It is possible to test all triangles (facets) of the given polyhedron against ρ_1 and ρ_2 planes. If both planes ρ_1 and ρ_2 intersect the given triangle (facet) then compute detailed intersection test, see alg.2. The intersection of the given plane ρ_i and the triangle exists if and only if two vertices \mathbf{x}_A and \mathbf{x}_B of the triangle exist so that

$$\text{sign}(F_i(\mathbf{x}_A)) \neq \text{sign}(F_i(\mathbf{x}_B))$$

where $F_i(\mathbf{x}) = a_i x + b_i y + c_i z + d_i$
 is a separation function for the i -th plane ρ_i , $i=1,2$

and $F_i(\mathbf{x}) = 0$
 is an equation for the i -th plane ρ_i , $i=1,2$.

Unfortunately there is some principal inefficiency in this implementation of the proposed solution as the separation function $F_1(\mathbf{x})$, resp. $F_2(\mathbf{x})$ are computed many times than needed because every vertex is shared by at least three triangles.

Therefore it seems to be convenient if the values $\text{sign}(F_1(\mathbf{x}_k))$ are precomputed (\mathbf{x}_k is the k -th vertex of the i -th facet) and stored in a separate vector, see alg.3. It means that the proposed solution might significantly improve the efficiency, especially for higher N .

```

procedure CLIP_3D (  $\mathbf{x}_A, \mathbf{x}_B$  );
begin
   $t_{\min} := 0;$      $t_{\max} = 1;$      $\mathbf{s} := \mathbf{x}_B - \mathbf{x}_A;$ 
  { for a line segment  $t_{\min} := -\infty;$      $t_{\max} = \infty;$  }
  {  $\rho_1 : a_1x + c_1z + d_1 = 0$      $\rho_2 : b_2y + c_2z + d_2 = 0$  }
   $i := 1;$      $j := 0;$ 
  while ( $i \leq N$ ) do
    begin
      {  $\mathbf{x}_k$  means a k-th vertex of the i-th triangle }
      if  $\text{sign}(F_1(\mathbf{x}_0)) = \text{sign}(F_1(\mathbf{x}_1))$  then
        if  $\text{sign}(F_1(\mathbf{x}_0)) = \text{sign}(F_1(\mathbf{x}_2))$  then goto 1;
        { do nothing -  $\rho_1$  does not intersect the i-th triangle }
      if  $\text{sign}(F_2(\mathbf{x}_0)) = \text{sign}(F_2(\mathbf{x}_1))$  then
        if  $\text{sign}(F_2(\mathbf{x}_0)) = \text{sign}(F_2(\mathbf{x}_2))$ 
          { do nothing -  $\rho_2$  does not intersect the i-th triangle }
          then goto 1;
        { both planes  $\rho_1, \rho_2$  intersect the i-th triangle }
        { detailed test SOLVE finds a value t }
        if SOLVE (  $\mathbf{x}_A, \mathbf{s}, i, t_{j+1}$  ) then  $j := j + 1;$ 
          { otherwise  $t_{j+1}$  will be rewritten }
1:     $i := i + 1;$ 
    end { while };
    if  $j = 0$  then EXIT; {no one triangle intersected by the line }
    if  $t_1 > t_2$  then SWAP ( $t_1, t_2$ ); { swaps values  $t_1 \leftrightarrow t_2$  }
     $t_{\min} := \max(t_{\min}, t_1);$      $t_{\max} := \min(t_{\max}, t_2);$ 
    if  $t_{\min} \leq t_{\max}$  then SHOW_LINE( $\mathbf{x}(t_{\min}), \mathbf{x}(t_{\max})$ );
end { CLIP_3D };

```

Algorithm 2

Because a line can intersect the convex polyhedron only in two points it is possible to extend the condition in the while statement, see alg.2, by condition and ($j < 2$). It can be seen that the detailed test (SOLVE procedure that uses parametric equations) is invoked only on a relatively few facets.

```

procedure CLIP_3D_MOD (  $\mathbf{x}_A, \mathbf{x}_B$  );
begin
     $t_{\min} := 0;$      $t_{\max} = 1;$      $i := 1;$      $j := 0$ 
    { for a line  $t_{\min} := -\infty;$      $t_{\max} = \infty;$  }
    {  $\rho_1 : a_1x + c_1z + d_1 = 0$      $\rho_2 : b_2y + c_2z + d_2 = 0$  }
    for  $k := 1$  to  $N_v$  do    {  $N_v$  number of vertices }
         $Q_k := \text{sign}(F_1(\mathbf{x}_k));$     {  $Q_k$  is a vector of int or char types }
    while ( $i \leq N$ ) and ( $j < 2$ ) do
        begin
            {  $i\mathbf{x}_k$  means a k-th vertex of the i-th triangle }
            { INDEX(i,k) gives the index of k-th vertex }
            { of the i-th triangle, i.e.  $i\mathbf{x}_k = \mathbf{x}_{\text{INDEX}(i,k)}$  }
            if  $Q_{\text{INDEX}(i,0)} = Q_{\text{INDEX}(i,1)}$  then
                if  $Q_{\text{INDEX}(i,0)} = Q_{\text{INDEX}(i,2)}$  then goto 1;
                { do nothing  $\rho_1$  does not intersect the i-th triangle }
                if  $\text{sign}(F_2(\mathbf{x}_{\text{INDEX}(i,0)})) = \text{sign}(F_2(\mathbf{x}_{\text{INDEX}(i,1)}))$  then
                    if  $\text{sign}(F_2(\mathbf{x}_{\text{INDEX}(i,0)})) = \text{sign}(F_2(\mathbf{x}_{\text{INDEX}(i,2)}))$ 
                        then goto 1;    {  $\rho_2$  does not intersect the triangle }
                    { both planes  $\rho_1, \rho_2$  intersect the i-th triangle }
                    { detailed test SOLVE finds a value t }
                    if SOLVE (  $\mathbf{x}_A, s, i, t_{j+1}$  ) then  $j := j + 1;$ 
                        { otherwise  $t_{j+1}$  will be rewritten }
        1:  $i := i + 1;$ 
        end;
        if  $j = 0$  then EXIT;    { no intersection }
        if  $t_1 > t_2$  then SWAP ( $t_1, t_2$ );
         $t_{\min} := \max(t_{\min}, t_1);$      $t_{\max} := \min(t_{\max}, t_2);$ 
        if  $t_{\min} < t_{\max}$  then SHOW_LINE( $\mathbf{x}(t_{\min}), \mathbf{x}(t_{\max})$ );
    end { of CLIP_3D_MOD };

```

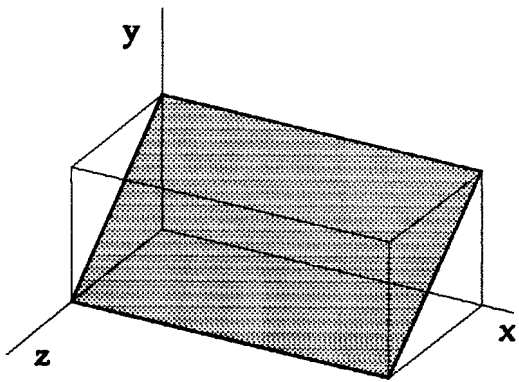
Algorithm 3

Planes ρ_1 and ρ_2 can be taken as parallel to any selected two coordinate axes, see fig.4. If those planes ρ_1 and ρ_2 are observed from the line p they are orthogonal. In that case the functions $F_i(\mathbf{x})$ can be simplified so that

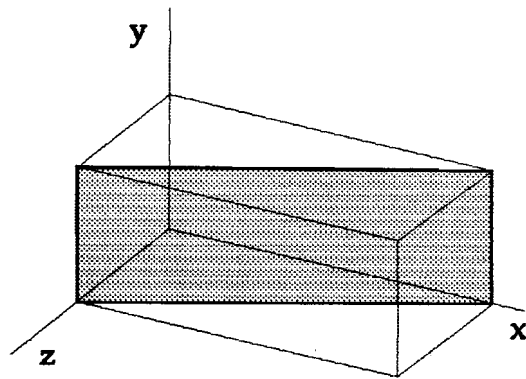
$$F_1(\mathbf{x}) = a_1x + c_1z + d_1 \quad \text{for plane } \rho_1$$

$$F_2(\mathbf{x}) = b_2y + c_2z + d_2 \quad \text{for plane } \rho_2$$

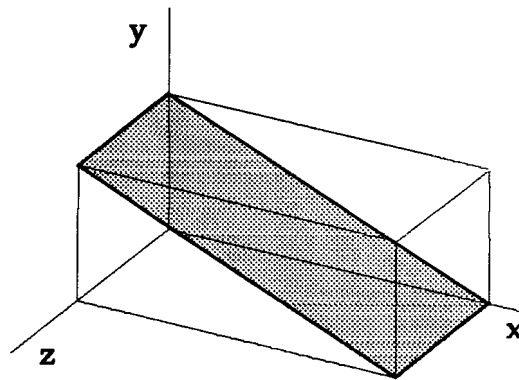
because planes ρ_1 and ρ_2 are selected as parallel to the y -axis and x -axis of the coordinate system. It can be seen that we get a singular case if the given line is parallel to the x -axis or y -axis. Therefore we should find a criterion how to select planes ρ_1 and ρ_2 and how to avoid such a singular cases, see fig. 4.



Plane parallel to x axis
(a)



Plane parallel to y axis
(b)



Plane parallel to z axis
(c)

Definition of "diagonal planes"

Figure 4

Let us suppose that we can generally select two planes ρ_1 and ρ_2 , see alg.4, from "diagonal planes" $\rho'_1, \rho'_2, \rho'_3$ which are defined as

$$\rho'_1: \mathbf{n}_1^T \mathbf{x} + d_1 = 0, \text{ where } \mathbf{n}_1 = s \times \mathbf{e}_x, \mathbf{e}_x = [1, 0, 0]^T$$

$$\rho'_2: \mathbf{n}_2^T \mathbf{x} + d_2 = 0, \text{ where } \mathbf{n}_2 = s \times \mathbf{e}_y, \mathbf{e}_y = [0, 1, 0]^T$$

$$\rho'_3: \mathbf{n}_3^T \mathbf{x} + d_3 = 0, \text{ where } \mathbf{n}_3 = s \times \mathbf{e}_z, \mathbf{e}_z = [0, 0, 1]^T$$

Planes ρ_1 and ρ_2 are selected so that

$$\rho_1, \rho_2 \in \{ \rho'_1, \rho'_2, \rho'_3 \}$$

ii := index of maximal value $\{ |s_x|, |s_y|, |s_z| \}$;

case of

$$\begin{aligned} 1: \text{ begin } & \mathbf{n}_1 = s \times \mathbf{e}_y; \quad d_1 := -\mathbf{x}_1^T \mathbf{x}_A; \\ & \mathbf{n}_2 = s \times \mathbf{e}_z; \quad d_2 := -\mathbf{n}_2^T \mathbf{x}_A; \quad \{ \text{ see fig. 4.a } \} \\ & \text{ end;} \end{aligned}$$

$$\begin{aligned} 2: \text{ begin } & \mathbf{n}_1 = s \times \mathbf{e}_z; \quad d_1 := -\mathbf{n}_1^T \mathbf{x}_A; \\ & \mathbf{n}_2 = s \times \mathbf{e}_x; \quad d_2 := -\mathbf{n}_2^T \mathbf{x}_A; \quad \{ \text{ see fig. 4.b } \} \\ & \text{ end;} \end{aligned}$$

$$\begin{aligned} 3: \text{ begin } & \mathbf{n}_1 = s \times \mathbf{e}_x; \quad d_1 := -\mathbf{n}_1^T \mathbf{x}_A; \\ & \mathbf{n}_2 = s \times \mathbf{e}_y; \quad d_2 := -\mathbf{n}_2^T \mathbf{x}_A; \quad \{ \text{ see fig. 4.c } \} \\ & \text{ end;} \end{aligned}$$

end;

Algorithm 4

It can be seen that for case ii = 1 we are getting

$$\mathbf{n}_1 = [-s_z, 0, s_x]^T, \quad \mathbf{n}_2 = [s_y, -s_x, 0]^T$$

and then

$$F_1(\mathbf{x}) = \mathbf{n}_1^T \mathbf{x}_A + d_1 = a_1x + c_1z + d_1, \quad c_1 \neq 0$$

$$F_2(\mathbf{x}) = \mathbf{n}_2^T \mathbf{x}_A + d_2 = a_2x + b_2y + d_1, \quad b_2 \neq 0$$

For ii = 2

$$\mathbf{n}_1 = [s_y, -s_x, 0]^T, \quad \mathbf{n}_2 = [0, s_z, -s_y]^T$$

and then

$$F_1(\mathbf{x}) = \mathbf{n}_1^T \mathbf{x}_A + d_1 = a_1x + b_1y + d_1, \quad a_1 \neq 0$$

$$F_2(\mathbf{x}) = \mathbf{n}_2^T \mathbf{x}_A + d_2 = b_2y + c_2z + d_1, \quad c_2 \neq 0$$

$$n_1 = [0, s_z, -s_y]^T, \quad n_2 = [-s_z, 0, s_x]^T$$

$$F_1(\mathbf{x}) = n_1^T \mathbf{x}_A + d_1 = b_1 y + c_1 z + d_1, \quad b_1 \neq 0$$

$$F_2(\mathbf{x}) = n_2^T \mathbf{x}_A + d_2 = a_2 x + c_2 z + d_1, \quad a_2 \neq 0$$

Further simplification of $F_i(\mathbf{x})$ functions is possible because for case $ii = 1$

$$c_1 \neq 0 \quad \text{and} \quad b_2 \neq 0$$

Therefore we can introduce modified separation functions $F'_i(\mathbf{x})$, $i = 1, 2$ as

$$F'_1(\mathbf{x}) = F_1(\mathbf{x}) / c_1 = a'_1 x + z + d'_1$$

$$F'_2(\mathbf{x}) = F_2(\mathbf{x}) / b_2 = a'_2 x + y + d'_2$$

and similarly for other cases.

This modification speeds up the proposed algorithm as two additions and four multiplications are saved for each facet. Precomputations of $F'_i(\mathbf{x})$ functions are made for each clipped line only. That is convenient as the number of tested facets of the given polyhedron is expected to be high (for a sphere approximation might reach 10^4).

3. Theoretical analysis

For considering efficiency of any algorithm we have to take into account at least floating point operations and their timing, see tab.1. Before making any comparisons it is necessary to point out that time needed for each operation ($:=, <, \pm, *, /$) does differ from computer to computer.

operation	$:=$	$<$	\pm	$*$	$/$
float	33	50	16	20	114
int	5	9	3	26	44

Time is in 1/10 sec for 10^5 operations for PC 486/33 MHz

Table 3

Let us denote N number of facets of the given convex polyhedron. Then it is possible to express the time complexity T_{CB} of the CB algorithm, see alg.1, as

$$T_{CB} = (6 , 3 , 4 , 9 , 1) * N$$

and computational time can be estimated using tab.1 as

$$T_{CB} = 706 * N$$

The time complexity T_0 of the algorithm that directly computes the intersection point of the given line p with a triangle using parametric equations can be estimated as

$$T_0 = (5.5 , 2.5 , 18 , 15 , 1.5) * N$$

and using tab.1 we obtain computational time

$$T_0 = 1065.5 * N$$

It means that this algorithm will be slower than CB algorithm.

The time complexity T of the proposed algorithm can be estimated:

- for the worst case as

$$T = (2 , 3.5 , 12 , 12 , 0) * N$$

- for an average case as

$$T = (2 , 2 , 6 , 6 , 0) * N$$

Using tab.1 we can evaluate the expression for T as

$$T = 673 * N \quad \text{for the worst case}$$

$$T = 382 * N \quad \text{for an average case}$$

Let us introduce coefficients of efficiency as

$$\nu_1 = \frac{T_{CB}}{T} \qquad \nu_2 = \frac{T_0}{T}$$

where T_{CB} , T_0 , T are execution times needed by the CB algorithm, algorithm that uses parametric equations and the proposed algorithm (CLIP_3D_MOD).

Now it is possible to estimate the asymptotic behavior of the proposed algorithm based on theoretical complexity estimations:

- for the worst case as

$$\nu_1 = \frac{T_{CB}}{T} = \frac{706 * N}{673 * N} = 1.04$$

$$\nu_1 = \frac{T_{CB}}{T} = \frac{706 * N}{382 * N} = 1.85$$

The obtained theoretical estimations and comparisons show that the proposed algorithm should be faster to the CB algorithm significantly.

4. Experimental results

The proposed algorithm has been tested against Cyrus-Beck's algorithm and algorithm that uses parametric equations. Data sets of two points that define a line have been randomly and uniformly generated inside a sphere in order to eliminate an influence of rotation. More than 10^4 randomly generated lines were used. Convex polyhedrons were generated as N-sided convex polyhedrons that consist of triangles and were inscribed into a smaller sphere.

Results obtained from experiments are shown in tab.2 and tab.3 for two fundamental cases when any line does not intersect the given polyhedron (0%) and when all lines intersect (100%) the given polyhedron. In general case the efficiency depends approximately linearly on the percentage of lines that intersect the given polyhedron. All tests were made on PC 486/33 MHz.

N	8	48	224	960
0%	1.01	0.92	1.18	1.22
100%	0.5	0.79	1.08	1.61

Efficiency coefficients ν_1 if condition $j < 2$ was not used

Table 2

N	8	48	224	960
0%	1.41	2.08	2.13	2.23
100%	1.07	1.58	1.90	2.15

Efficiency coefficients ν_2 condition $j < 2$ was not used

Table 3

The efficiency of the proposed algorithm against CB algorithm for $N \geq 224$ is

$$v_1 \in \langle 1.08, 1.66 \rangle \quad (\text{condition } j < 2 \text{ was not used}),$$

$$v_1 \in \langle 1.64, 2.49 \rangle \quad (\text{condition } j < 2 \text{ was used})$$

and the efficiency of the proposed algorithm against direct intersection computation

$$v_2 \in \langle 1.9, 2.23 \rangle \quad (\text{condition } j < 2 \text{ was not used}),$$

$$v_2 \in \langle 1.5, 2.9 \rangle \quad (\text{condition } j < 2 \text{ was used})$$

It means that the proposed algorithm for $N \geq 224$ was always faster than the CB algorithm.

In the course of the algorithm experimental evaluation was found that the proposed algorithm is sensitive to implementation of the $\text{sign}(x)$ function. We actually need just determine if $x \geq 0$. In that case it is possible just to test a sign bit of the float point representation of value x .

The proposed algorithm can be generalized for a non-triangular facets by replacing procedure SOLVE by a more general procedure or by using a similar approach as the CB algorithm does, e.g. use the CB algorithm just for all facets that were intersected by both planes ρ_1 and ρ_2 . In case that normal vectors of all facets can be precomputed we can obtain an additional speed up of the proposed algorithm if convex polyhedron is considered and all facets are oriented.

Tab.4 shows the final experimental results if $\text{sign}(x)$ function and all sign comparisons were implemented carefully and the SOLVE procedure was implemented as a modification of the CB algorithm for the case when normal vectors for all facets were precomputed, see alg.5.

N	10	20	100	200	500
0%	1.28	2.08	2.69	2.89	3.17
100%	0.3	0.57	1.57	2.06	2.67

Efficiency coefficients v_1 if condition $j < 2$ was not used

Table 4

```

begin { for a line  $t_{\min} := -\infty$ ;  $t_{\max} = \infty$ ; }
     $t_{\min} := 0$ ;  $t_{\max} = 1$ ;  $i := 1$ ;  $j := 0$ 
    {  $\rho_1 : a_1x + c_1z + d_1 = 0$   $\rho_2 : b_2y + c_2z + d_2 = 0$  }
    for  $k := 1$  to  $N_V$  do {  $N_V$  number of vertices }
         $Q_k := \text{sign}(F_1(\mathbf{x}_k))$ ; {  $Q_k$  is a vector of int or char types }
    while ( $i \leq N$ ) and ( $j < 2$ ) do
    begin {  $\mathbf{x}_k^i$  means a k-th vertex of the i-th triangle }
        { INDEX(i,k) gives the index of k-th vertex }
        { of the i-th triangle, i.e.  $\mathbf{x}_k^i = \mathbf{x}_{\text{INDEX}(i,k)}$  }
        if  $Q_{\text{INDEX}(i,0)} = Q_{\text{INDEX}(i,1)}$  then
            if  $Q_{\text{INDEX}(i,0)} = Q_{\text{INDEX}(i,2)}$  then goto 1;
            { do nothing  $\rho_1$  does not intersect the i-th triangle }
        if  $\text{sign}(F_2(\mathbf{x}_{\text{INDEX}(i,0)})) = \text{sign}(F_2(\mathbf{x}_{\text{INDEX}(i,1)}))$  then
            if  $\text{sign}(F_2(\mathbf{x}_{\text{INDEX}(i,0)})) = \text{sign}(F_2(\mathbf{x}_{\text{INDEX}(i,2)}))$ 
                then goto 1; {  $\rho_2$  does not intersect the triangle }
        { both planes  $\rho_1, \rho_2$  intersect the i-th triangle }
        { one step of the CB algorithm }
        { ----- }
        {  $\mathbf{n}_i$  is a normal vector of the i-th facet }
        { and  $\mathbf{n}_i$  must point out of the convex polyhedron }
         $\xi := \mathbf{s}^T \mathbf{n}_i$ ;  $\mathbf{s}_i := \mathbf{x}_i - \mathbf{x}_A$ ;
        if  $\xi \neq 0.0$  then
            begin  $t := \mathbf{s}_i^T \mathbf{n}_i / \xi$ ;
                if  $\xi > 0.0$  then  $t_{\max} := \min(t, t_{\max})$ 
                    else  $t_{\min} := \max(t, t_{\min})$ 
            end
            else Special case; { line is parallel to the i-th facet }
            { ----- }
1:  $i := i + 1$ ;
    end;
    if  $t_{\min} \leq t_{\max}$  then SHOW_LINE( $\mathbf{x}(t_{\min}), \mathbf{x}(t_{\max})$ );
end { of CLIP_3D_MOD };

```

Algorithm 5

5. Modification for Non-Convex Polyhedrons

It can be seen that the proposed algorithm can be used for clipping lines against non-convex polyhedrons, too. For this case only the condition $j < 2$ must be removed from the while statement in the suggested algorithm, see alg.3., and the lines marked by (*) must be replaced by a sequence, see alg.4. Of course, it is necessary to sort all values t_j . Intervals $\langle t_{2j-1}, t_{2j} \rangle$ define parts of the given line which lies inside of the given non-convex polyhedron.

```

SORT_ALL_VALUES ( { ti } );
for k := 1 to j/2 do
  if  $\langle t_{2k-1}, t_{2k} \rangle \cap \langle 0, 1 \rangle \neq \emptyset$  then
    SHOW_LINE(x(t2k-1), x(t2k));

```

Non-convex case modification for alg.3.

Algorithm 4

If line clipping is considered instead of line segment then the part $\langle 0, 1 \rangle$ in if statement has to be left out.

The complexity of the proposed modified algorithm in this case will be

$$O(N) + O(k \lg k)$$

where k is a number of intersections of the given line and the given polyhedron (for situation on Fig.1.b is $k = 4$), N is a number of triangles (facets).

In experimental algorithm verification non-convex polyhedrons were generated by a special 3D graphics editor which enabled to prepare quite complex objects. In the case of non-convex polyhedron direct solution of parametric equations were only used and the efficiency coefficient ν_2 is always higher than one for all N . All experiments showed that

$$\nu_2 \in \langle 1.07, 2.15 \rangle$$

The new efficient algorithm of $O(N)$ complexity for clipping lines against convex polyhedron was developed with modification for non-convex polyhedron case, too. The proposed algorithm does not strictly require triangular facets and can be easily modified for non-triangular facets using a different SOLVE procedure for general polygonal facets. The given facets should be oriented. All tests were implemented in C++ on PC 486/33 MHz.

7. Acknowledgments

The author would like to express his thanks to students of Computer Graphics courses at the University of West Bohemia in Plzen and Charles's University in Prague for their suggestions and critical comments that stimulated this project, especially to P.Sebránek and J.Jirák for verifications of all tests and implementation of the proposed algorithm, to P.Bláha for making the 3D graphics editor for preparing polyhedrons.

8. References

- [1] Cyrus, M., Beck, J.: Generalized Two and Three Dimensional Clipping, Computers & Graphics, Vol.3, No.1, pp.23-28, 1979.
- [2] Foley, D.J., van Dam, A., Feiner, S.K., Huges, J.F.: Computer Graphics - Principles and Practice, Addison Wesley, 2nd ed., 1990.
- [3] Skala, V.: An Efficient Algorithm for Line Clipping by Convex Polygon, Computers & Graphics, No.4, Vol.17. pp.417-421, 1993.
- [4] Skala, V.: $O(\lg N)$ Line Clipping Algorithm in E^2 , accepted for publication, Computers & Graphics, Pergamon Press, No.4, Vol.18, 1994.