The Visual Computer

Extension of the Nicholls–Lee–Nichols algorithm to three dimensions

Václav Skala, Duc Huy Bui

Department of Computer Science and Engineering, University of West Bohemia, Univerzitní 22, Box 314, 306 14 Plzeň, Czech Republic
e-mail: {skala, bui}@kiv.zcu.cz

A new algorithm for clipping a line segment against a pyramid in E^3 is presented. This algorithm avoids computation of intersection points that are not end points of the output line segment. It also solves all cases more effectively. The performance of this algorithm is shown to be consistently better than that of existing algorithms, including the Cohen–Sutherland, Liang–Barsky, and Cyrus–Beck algorithms.

Key words: Line clipping – Computer graphics – Algorithm complexity – Geometric algorithms – Algorithm complexity analysis

Correspondence to: V. Skala

1 Introduction

Let us assume that two points $A(x_A, y_A, z_A)$ and $B(x_B, y_B, z_B)$ are given, and we wish to compute the intersection of the line segment AB with the unitary clipping pyramid, defined as the set of all points (x, y, z) such that $-z \leq x \leq z$ and $-z \leq y \leq z$ $(z \geq 0)$. The intersection is either empty or a line segment whose end points we must compute.

Many algorithms for clipping a line or a line segment in E^3 have been published; see the standard textbooks (Cyrus and Beck 1978; Liang and Barsky 1983, 1984; Foley et al. 1990; Skala 1996, 1997) for the main references. For a long time, the Cohen-Sutherland (CS) algorithm and its extensions to E^3 (Foley et al. 1990) were used for the line segment clipping algorithms nearly exclusively. Later, the Liang-Barsky (LB) and Cyrus-Beck (CB) algorithms were proposed, and they are based on the line parametric representation. Because the line segment clipping against a pyramid is used in all graphics packages, a new algorithm based on Nicholl-Lee-Nicholl algorithm has been developed.

Before describing the proposed algorithm for a line segment clipping, it is necessary to unify some definitions.

2 Definitions

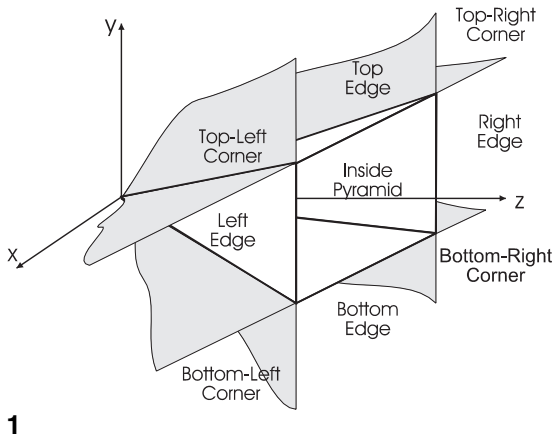
The planes $x = -z$, $x = z$, $y = -z$, and $y = z$ are called the right, left, bottom, and top boundaries of the unitary pyramid, respectively. We say that:

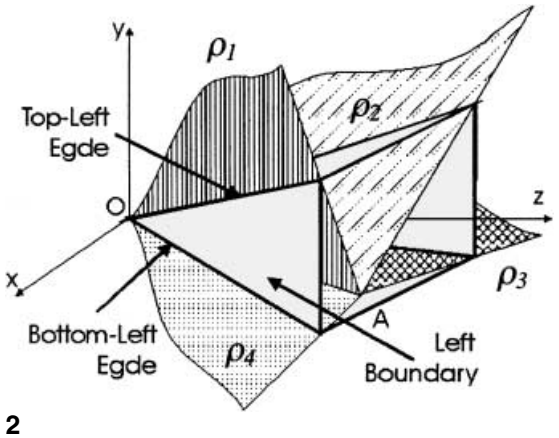
- A point or a line segment is visible if it lies entirely inside the given pyramid,
- A point or a line segment is invisible if it lies entirely outside the given pyramid,
- A line segment is partially visible if it lies partly inside and partly outside the given pyramid.

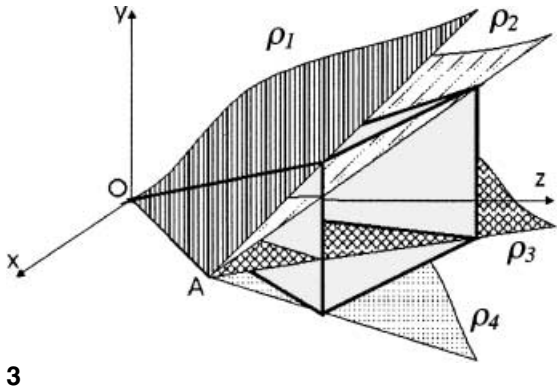
If the line segment is invisible, then no part of the line segment appears in the output, and the line segment is said to be rejected by the clipping algorithm.

The boundaries of the pyramid divide the cartesian positive half space $(z \geq 0)$ into nine regions. Regions that are bounded by only two boundaries are called the corner regions, and regions that are bounded by three boundaries are called the edge regions (Fig. 1).

The Visual Computer (2001) 17:236–242







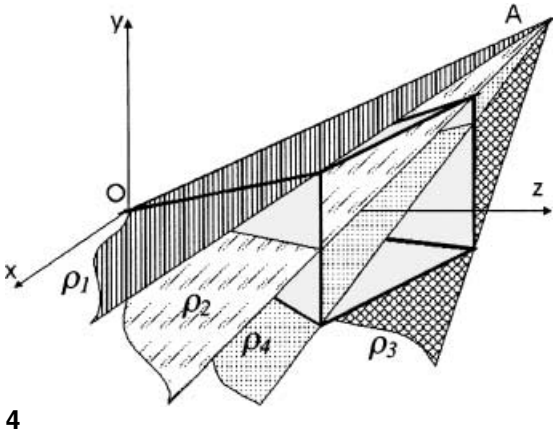


Fig. 1. Subdivision of the positive half space into regions
Fig. 2. Subregions for the case when point A lies inside the pyramid
Fig. 3. Subregions for the case when point A is in the left edge region
Fig. 4. Subregions for the case when point A is in top right corner region

3 Proposed pyramidal clipping algorithm

The proposed pyramidal clipping (PC) algorithm uses an approach similar to that of the Nicholl–Lee–Nicholl algorithm (Nicholl et al. 1987) that was derived for the E^2 case only. Given a line segment AB, for simplicity, we assume that both points A and B are in the positive half space. The end point A therefore can lie inside of the pyramid, in an edge region or in a corner region. For each of these cases, we can divide the positive half space into certain number of subregions (Figs. 2–4). These subregions are bounded by the pyramid boundaries and planes determined by point A and one edge of the pyramid. These planes are denoted as $\varrho_1, \varrho_2, \varrho_3$, and ϱ_4, clockwise from the top left edge (Figs. 2–4). All other cases can be obtained from one of these cases in Figs. 2–4 by rotating the scene around the z axis.

With these definitions, the algorithm can be described by the following basic steps:

- Characterize the location of point A among the nine regions.
- Characterize the location of point B among the appropriate subregions.

- Compute the intersection points according to the characterization.

The main advantage of this approach is that we can determine which pyramid boundaries are intersected, and we therefore avoid unnecessary computation of invalid intersection points.
The proposed PC algorithm is explained in more detail by the top-down approach. At the beginning, we must determine whether the end point A of the given line segment is beyond the right boundary, beyond the left boundary, or between those two boundaries. The main procedure, in Pascal-like code is:

```
procedure Clip(x_A, y_A, z_A, x_B, y_B, z_B : real);
begin
  if x_A < -z_A then    case 3.1
                {point A is beyond right boundary}
  else if x_A <= z_A then case 3.2
                {point A is between 2 boundaries}
    else                case 3.3
                {point A is beyond left boundary}
end;
```

We will use the exit command in the following parts of the algorithm to denote the end of the procedure and to avoid "**else if**" sequences that are unnecessary for the algorithm explanation.

3.1 Point A is beyond the right boundary

If point B is also beyond the right boundary, it is not necessary to characterize point A further because the line segment is invisible. Therefore, we must check whether point B is beyond the right boundary before proceeding. After that, we must test whether point A lies either in the corner region or in the edge region. This section of the algorithm can be implemented as follows:

```
begin {case 3.1}
  if x_B < -z_B then
          EXIT; {Line segment is rejected}
  if y_A > z_A then case 3.1.1
                {point A is in the top right
                          corner region}
  else if y_A >= -z_A then case 3.1.2
                {point A is in the right edge
                                region}
    else case 3.1.3
                {point A is in the bottom right
                              corner region}
end {case 3.1};
```

3.1.1 Point A is in the top right corner region, and point B is not beyond the right boundary

If point B is above the top boundary, the line segment is invisible, and no further computation is needed. Therefore, we need to check this condition first, and then characterize point B so that we can distinguish between the case when point B is beyond the left boundary and the case when point B is inside the pyramid or in the bottom edge region (Fig. 4). The following pseudo-code shows how it can be implemented:

```
begin {case 3.1.1}
  if y_B > z_B then
          EXIT; {Line segment is rejected}
  if x_B > z_B then case 3.1.1.1
                {point B is in the left edge or in the
                          bottom left corner region}
    else case 3.1.1.2
                {point B is inside of the pyramid or in
                              the bottom edge region}
end {case 3.1.1};
```

3.1.1.1 Point A is in the top right corner region and point B is in the left edge region or in the bottom left corner region

If point B is above the plane ϱ_1, the line segment is rejected (Fig. 4). Therefore, we must check this condition first, and then distinguish the case when point B is in the left edge region and the case when point B is in the bottom left corner region. In the case of the left edge region, one intersection point lies on the pyramid's left boundary. The location of point B against the plane ϱ_2 specifies, that the second intersection point lies on the top or on the right boundary. This way, only the appropriate intersection point is computed. In the case of bottom left corner region, point B is compared to the plane ϱ_3 first to eliminate the case when the line segment is rejected. After that, we compare the position of point B to the plane ϱ_4 to determine the location of the first intersection point (on the bottom or on the left boundary). At the end, a similar comparison with the plane ϱ_2 determines the second intersection point. An implementation can be as follows:

```
begin {case 3.1.1.1}
  Δx := x_B - x_A; Δy := y_B - y_A; Δz := z_B - z_A;
  if ((x_A - z_A) * (Δz - Δy)
                    > (y_A - z_A) * (Δz - Δx))
    then EXIT; {Line segment is rejected}
```

```
{first intersection point computation}
if y_B > −z_B then {B is in the left edge region}
    t_1 := (x_A − z_A)/(Δz − Δx)
else {B is in the bottom left corner region}
 begin
 if ((x_A + z_A) * (Δz + Δy)
                        > (y_A + z_A) * (Δz + Δx))
    then EXIT; {Line segment is rejected}
 if ((z_A − x_A) * (Δy + Δz)
                        > (z_A + y_A) * (Δz − Δx))
    then {intersection with left boundary}
         t_1 := (x_A − z_A)/(Δz − Δx)
 else {intersection with bottom boundary}
         t_1 := −(y_A + z_A)/(Δz + Δy)
 end;
{second intersection point computation}
if ((x_A + z_A) * (Δz − Δy)
                        > (z_A − y_A) * (Δz + Δx))
 then {intersection with top boundary}
         t_2 := (y_A − z_A)/(Δz − Δy)
else {intersection with right boundary}
         t_2 := −(x_A + z_A)/(Δz + Δx)
end {case 3.1.1.1};
```

3.1.1.2 Point A is in the top right corner region, and point B is inside the pyramid or in the bottom edge region

In the case when point B is inside the pyramid, the position of point B against the plane ϱ_2 specifies whether the intersection point lies either on the top or on the right boundary. In the case when point B is in the bottom edge region, point B must be compared with the plane ϱ_3 first to eliminate the situation in which the line segment is rejected. If the line segment AB is not rejected by the clipping algorithm, then one intersection point lies on the bottom boundary. The other intersection point lies on the top or on the right boundary according to the position of point B against the plane ϱ_2. This section can be implemented as follows:

```
begin {case 3.1.1.2}
  Δx := x_B − x_A; Δy := y_B − y_A; Δz := z_B − z_A;
  {first intersection point computation}
  if y_B < −z_B then {B in bottom edge region}
   begin
   if ((x_A + z_A) * (Δz + Δy)
                        > (y_A + z_A) * (Δz + Δx))
      then EXIT; {Line segment is rejected}
   t_1 := −(y_A + z_A)/(Δz + Δy)
   end
  else {B is inside the pyramid}
   t_1 := 1;
  {second intersection point computation}
  if ((x_A + z_A) * (Δz − Δy)
                        > (z_A − y_A) * (Δz + Δx))
   then {intersection with top boundary}
   t_2 := (y_A − z_A)/(Δz − Δy)
  else {intersection with right boundary}
   t_2 := −(x_A + z_A)/(Δz + Δx)
end {case 3.1.1.2};
```

3.1.2 Point A is in the right edge region, and point B is not beyond the right boundary

We need to distinguish the cases when point B is below the bottom boundary (point B is in bottom left corner region or in the bottom edge region), or above the top boundary (point B is in top left corner region or in the top edge region) or between top and bottom boundaries. An implementation can be as follows:

```
begin {case 3.1.2}
  if y_B < −z_B then case 3.1.2.1
          {point B is in the bottom-left corner
              or in the bottom edge region}
  else if y_B <= z_B then case 3.1.2.2
          {point B is in the left edge region or
              inside of the pyramid}
     else case 3.1.2.3
          {point B is in the top left corner
              or in the top edge region}
end {case 3.1.2}
```

3.1.2.1 Point A is in the right edge region, and point B is in the bottom left corner or in the bottom edge region

The location of point B against the plane ϱ_3 helps us to eliminate the case when the line segment is rejected. If point B is in the bottom edge region, then the intersection points are on the right and the bottom boundaries. If point B is in the bottom left corner region, then one intersection point lies on the right boundary and the second intersection point's location (either on the left or on the bottom boundary) is determined by the location of point B against the plane ϱ_4.

```
begin {case 3.1.2.1}
  if ((x_A + z_A) * (Δz + Δy)
                        > (y_A + z_A) * (Δz + Δx))
     then EXIT; {Line segment is rejected}
  {first intersection point computation}
```

```
  if x_B > z_B then {B in the bottom left corner}
      if ((z_A − x_A) * (Δz + Δy)
                          > (z_A + y_A) * (Δz − Δx))
       then {intersection with left boundary}
          t_1 := (x_A − z_A)/(Δz − Δx)
      else {intersection with bottom boundary}
          t_1 := −(y_A + z_A)/(Δz + Δy)
  else {B in bottom edge region}
          {intersection with bottom boundary}
      t_1 := −(y_A + z_A)/(Δz + Δy);
  {second intersection is on right boundary}
  t_2 := −(x_A + z_A)/(Δz + Δx)
end {case 3.1.2.1};
```

3.1.2.2 Point A is in the right edge region, and point B is inside of the pyramid or in the left edge region

In this case, one intersection point is on the right boundary and the second one (if point B is in the left edge region) is on the left boundary; see following pseudo-code:

```
begin {case 3.1.2.2}
  {first intersection is on right boundary}
  t_1 := −(x_A + z_A)/(Δz + Δx);
  {second intersection point computation}
  if x_B > z_B then {point B in left edge region}
     t_2 := (x_A − z_A)/(Δz − Δx)
  else t_2 := 1
end {case 3.1.2.2};
```

3.1.2.3 Point A is in the right edge region, and point B is in the top left corner or in the top edge region

This case is similar to the case in Sect. 3.1.2.1.

3.1.3 Point A is in the bottom right corner region, and point B is not beyond the right boundary

The case is similar to the case in Sect. 3.1.1.

3.2 Point A is between the left and the right boundaries

In this case, we need to characterize the location of point A to specify whether point A lies inside the pyramid or in an edge region. The following pseudo-code shows how it can be done:

```
begin {case 3.2}
  if y_A > z_A then case 3.2.1
                {A is in the top edge region}
  else if y_A < −z_A then case 3.2.2
                {A is in the bottom edge region}
     else case 3.2.3 {A is inside the pyramid}
end {case 3.2};
```

We need to consider only the case when point A is inside the pyramid (case 3.2.3). The cases, in which point A is in the top (case 3.2.1) or bottom edge region (case 3.2.2), are similar to the case when point A is in the right edge region (case 3.1.2).

3.2.3 Point A is inside the pyramid

If point B lies in an edge region, then the boundary, on which the intersection point lies, is determined (Fig. 2), and the appropriate intersection point is computed. If point B lies in a corner region, then a comparison of the location of point B with an appropriate plane ϱ_i is necessary before the appropriate intersection point is computed. An implementation can be illustrated as follows:

```
begin {case 3.2.3}
  if x_B < −z_B then
     if y_B > z_B then case 3.2.3.1
           {B is in the top right corner region}
     else if y_B >= −z_B then case 3.2.3.2
           {B is in the right edge region}
        else case 3.2.3.3
        {B is in the bottom right corner region}
  else if x_B > z_B then
     if y_B > z_B then case 3.2.3.4
           {B is in the top left corner region}
  else if y_B < −z_B then case 3.2.3.5
        {B is in the bottom left corner region}
        else case 3.2.3.6
           {B is in the left edge region}
else
  if y_B > z_B then case 3.2.3.7
           {B is in the top edge region}
  else if y_B < −z_B then case 3.2.3.8
           {B is in the bottom edge region}
     else case 3.2.3.9
        {B is inside pyramid; the whole line
              segment is visible}
end {case 3.2.3};
```

3.2.3.1 Point A is inside the pyramid, and point B is in top right corner region

The comparison of point B with the plane ϱ_2 specifies which boundary (top or right) is to be used to

compute the intersection point. An implementation can be as follows:

begin {case 3.2.3.1}
  if $((x_A + z_A) * (\Delta z - \Delta y)$
    $> (z_A - y_A) * (\Delta z + \Delta x))$
  then {intersection with top boundary}
    $t_1 := (y_A - z_A)/(\Delta z - \Delta y)$
  else {intersection with right boundary}
    $t_1 := -(x_A + z_A)/(\Delta z + \Delta x)$;
  $t_2 := 0$;
end {case 3.2.3.1};

The cases 3.2.3.3–3.2.3.5 are similar to case 3.2.3.1.

3.2.3.2 Point A is inside the pyramid, and point B is in right edge region

The appropriate intersection point (the intersection point on the right boundary) is calculated. The following pseudo-code shows how it can be implemented:

begin {case 3.2.3.2}
    $t_1 := -(x_A + z_A)/(\Delta z + \Delta x)$;
    $t_2 := 0$
end {case 3.2.3.2};

The cases 3.2.3.6–3.2.3.8 can be solved similarly.

3.3 Point A is beyond the left boundary

This case can be solved similarly to the case when point A is beyond the right boundary (case 3.1).
Finally, we can easily compute the end points of the output line segment from the parameter value t as follow:

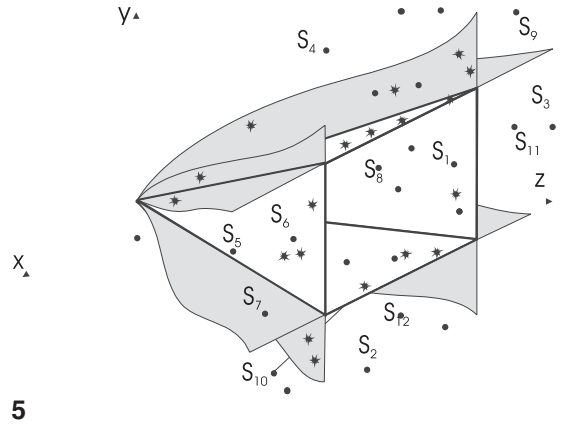
$x := t * \Delta x + x_A$;
$y := t * \Delta y + y_A$;
$z := t * \Delta z + z_A$.

It can be seen that all possible cases have been solved, and the complete algorithm can be obtained by substitution of the appropriate codes in all procedures.

4 Experimental results

To be able to compare the CS, LB, and CB algorithms with the proposed PC algorithm and evaluate the efficiency of the PC algorithm, we introduce three coefficients of efficiency as:

$$\nu_1 = \frac{T_{\mathrm{CS}}}{T_{\mathrm{PC}}},\ \nu_2 = \frac{T_{\mathrm{LB}}}{T_{\mathrm{PC}}},\ \nu_3 = \frac{T_{\mathrm{CB}}}{T_{\mathrm{PC}}},$$



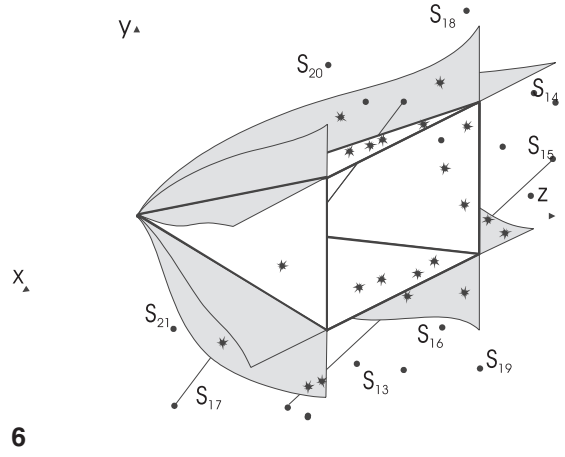


Fig. 5. Generated line segments
Fig. 6. Generated line segments

where T_{CS}, T_{LB}, T_{CB}, and T_{PC} denote the time consumed by the CS algorithm, LB algorithm, CB algorithm, and the proposed PC algorithm, respectively.
For experimental verification, $8 \cdot 10^7$ different line segments were randomly generated for each of the 21 cases shown in Figs. 5 and 6. The tests were performed on a PC Intergraph Pentium II, 400 MHz 512 MB RAM, 256 KB cache. The obtained results are presented in Table 1, which shows that the proposed algorithm is just as fast as the CS algorithm when the line segment lies inside the pyramid, and it is significantly faster in all other cases. It can be seen that the speed-up varies from 1.17 to 2.08 approximately for these cases. Table 1

Table 1. Experimental results

Case	ν_1	ν_2	ν_3
s1	1.01	2.54	2.40
s2	1.29	2.83	2.69
s3	1.51	1.85	1.75
s4	1.52	1.84	1.75
s5	1.25	1.74	1.63
s6	1.31	1.35	1.25
s7	1.57	1.26	1.17
s8	1.18	1.78	1.63
s9	1.55	1.64	1.51
s10	1.53	1.56	1.46
s11	1.54	1.22	1.14
s12	1.17	1.68	1.58
s13	1.25	1.28	1.19
s14	1.18	1.69	1.57
s15	2.08	1.53	1.74
s16	1.38	1.44	1.62
s17	1.50	1.23	1.13
s18	1.63	1.09	1.00
s19	1.18	1.25	1.15
s20	1.49	1.23	1.12
s21	1.32	1.30	1.22

also shows that the PC algorithm is significantly faster than the LB and CB algorithms for most cases.

5 Conclusion

The new line segment clipping algorithm against a given pyramid in E^3 has been developed, verified, and tested. This algorithm is convenient for all applications if the line segment clipping in E^3 is to be used. Experiments have shown that the new algorithm is never slower than the CS, LB, and CB algorithms, and is generally faster; up to twice as fast in some cases.

Acknowledgements. The authors would like to express their thanks to all who contributed to this work, especially to colleagues, recent MSc and PhD students of Computer Graphics at the University of West Bohemia in Pilsen, who stimulated this work. Special thanks go to the anonymous reviewers for very useful comments and recommendations. V. Skala's work was supported by the Ministry of Education of the Czech Republic: project GA AV A2030801. D.H. Bui's work was supported by the Ministry of Education of the Czech Republic: project VS 97155.

References

1. Bui DH, Skala V (1998) Fast algorithms for clipping lines and line segments in E^2. Visual Comput 14:31–37
2. Cyrus M, Beck J (1978) Generalized two- and three-dimensional clipping. Comput Graph 3:23–28
3. Foley DJ, Dam A van, Feiner SK, Hughes JF (1990) Computer graphics – principles and practice, 2nd edn. Addison Wesley
4. Liang YD, Barsky BA (1983) An analysis and algorithm for polygon clipping. Commun ACM 26:868–877
5. Liang YD, Barsky BA (1984) A new concept and method for line clipping. ACM TOG 3:1–22
6. Nicholl TM, Lee DT, Nicholl RA (1987) An efficient new algorithm for 2D line clipping: its development and analysis. ACM Comput Graph 21:253–262
7. Skala V (1996) An efficient algorithm for line clipping by convex and non-convex polyhedra in E^3. Comput Graph Forum 15:61–68
8. Skala V (1997) A fast algorithm for line clipping by convex polyhedron in E^3. Comput Graph 21:209–214

VÁCLAV SKALA received his MSc from the Institute of Technology, Plzeň, in 1975, his PhD from the Czech Technical University, Prague, in 1981. In 1988 he was appointed as a Reader at the Institute of Technology, Plzeň and was habilitated at the University of West Bohemia in 1989. He was appointed as a full Professor of Computer Science at the University of West Bohemia, Plzeň in 1996. His research interests are in fundamental algorithms and algorithms for visualization, parallel processing, and computational geometry. He is the organizer of the WSCG conferences on Computer Graphics, Visualization and Computer Vision.

HUY DUC BUI completed his MSc at the University of West Bohemia in 1995; he worked as a Research Assistant and finished his PhD studies at the University of West Bohemia in 1999. His current research interests are algorithms for computer graphics, scientific computation, and parallel and distributed processing.