A Faster Algorithm for 3D Discrete Lines

V. Boyer and J.-J. Bourdin

G.R.I.S. and Laboratoire d'Intelligence Artificielle, Université Paris 8, France
boyer,jj@ai.univ-paris8.fr

Abstract
In a three–dimensional space considered as a set of uniform voxels, an object is defined within a volume represented by a subset of voxels. The illumination of a scene requires the determination of overlapping of lines and objects. This task is a large part of the computation times in illumination methods. The algorithms used for it are based on the DDA method. We present an original method to determine the voxels crossed by a line. The speed of this method is then compared with the speed of Kaufman's method[10].
Keywords: *[Computer Graphics]: Line Approximation, Discrete Ray–Tracing, Voxel, Three–Dimensional DDA, Picture/Image Generation.*

1. Introduction

As usual[11, 10] we represent the three–dimensional space by elementary uniform cubes (voxels). Objects are included in voxel–based volumes. Illumination methods require to compute lines. For ray–tracing these lines are light rays[8]. For radiosity the lines computed connect objects together. In these two cases, for each line the overlapping of each voxel must be detected.

The computation time of a scene's illumination depends of these overlapping computations. Most algorithms are adaptations of well known 2D algorithms. For example Stolte[13], Cohen et al.[7, 10] adapt the 2D DDA method presented by Bresenham in 1965[4].

This paper presents an adaptation of the last improvements of the 2D line drawing methods to 3D lines computation.

2. 3D Line

"Line" denotes the discrete straight segment between two points. Let $P(x_p, y_p, z_p)$ and $Q(x_q, y_q, z_q)$ be the extremities of the line. As these coordinates are integers, the segment is an exact translation of the line $((0,0,0)(u,v,t))$, where:

$$\begin{cases} u = x_q - x_p, \\ v = y_q - y_p, \\ t = z_q - z_p. \end{cases}$$

The projections of the 3D line along the axis produce three 2D lines: (u,v,t) transforms into (u,v) along Z–axis, (u,t) along Y–axis and (v,t) along X–axis. Two of such projections suffice to compute the 3D line.

3. Previous method

Kaufman's[10] method is based on the projection propriety. Two linear interpolations are computed with Bresenham's DDA method[4]. Two error values are computed to measure the distance between a point of the continuous line and the current discrete point. The first one measures an ordinate difference and the second one a height difference. An increment system is used to compute incrementally the next error values.

4. 2D line improvement

Bresenham's method[4] is improvable. Double step (Rokne[12]), triple step (Graham[9]), repeating gcd times a regular pattern (Angel[1]) are well known improvements. Castle[6] uses a combinatory approach. Run

Length Encoding (RLE) by Bresenham[5] computes directly the length of each monotonous span. Berstel reduced the problem to the analysis of continuous fractions[2].

Boyer et al.[3] use an inner symmetry to compute only one half of the line. They use also an adaptation of RLE. These different spans are then placed on the line by a DDA algorithm. Symmetric properties permit to work only in one hexadecant. These choices shorten significantly the computation. The speed–up ratio to Bresenham's algorithm is 21.

5. A New 3D algorithm

Our purpose was to adapt these last improvements to 3D space. Let v/u and t/v be "reduced slopes" (i.e. the gcd's of u, v and t, v are 1). The span length is approximately $m = u/v$. The square span (S) is 0^m1 and the diagonal span (D) is $0^{m-1}1$. The spans are distributed homogeneously with a DDA method. Vertical moves are coded "2" and will be placed during this distribution according to the sign of δ_z which is computed incrementally. See for example the computation of the line ((0,0,0) (21,11,4)):

$$u = 27, v = 11, t = 4, m = 2, \mathrm{S} = 001, \mathrm{D} = 01$$

i	1	2	3	4	5	6
span	S	D	S2	D	S	D2
moves	001	01	0012	01	001	012
i	6	7	8	9	10	11
span	D2	S	D	S2	D	D2
moves	012	001	01	0012	01	012

Table 1: *Span computation trace*

The symmetry demonstrated for the 2D discrete line[3] is preserved for 3D. Spans are disposed symmetrically and only the extremes ($i = 1$ and $i = v$) are different (respectively S and D2). So the computation of half the line produces the whole line. The execution time, linear with regard to v, is divided by 2. Below we present the algorithm (benchmarks are based on an implementation in C, Figure 1). Parameters are:

$i = v$	number of spans
$j = v - u\%v$	number of diagonal spans
$k = t$	number of vertical moves
S	square span
D	diagonal span

PutSpan and PutReverseSpan are functions displaying the spans respectively at the beginning and

```
void SymetricCompute(int i, int j, int k,
                     int H[], int 0[]) {
  int io, ih, iv;
  int d, dz;

  io = j−i;
  ih = j;
  iv = k−i;
  d = 2*j−i;
  dz = 2*z−i;

  PutSpan(H); PutReverseSpan(2);
  PutReverseSpan(0);
  i = (i−1)/2;
  while (i−−) {
    if (delta>=0) {
      PutSpan(0); PutReverseSpan(0);
      delta+=io;
    } else {
      PutSpan(H); PutReverseSpan(H);
      delta+=ih;
    }
    if (dz>=0){
      PutSpan(2); PutReverseSpan(2);
      delta+=k;
    } else
      delta+=iv;
  }
}
```

Figure 1: *3D straight lines algorithm*

at the end of the line. Further speed gain is due to the integration of Rokne's[12] "double step" into this algorithm. The vertical 2–proximity, is well adapted to the line/voxels computation.

6. Benchmarks

Every line starting at the origin and ending within a pyramid is computed. The pyramid is: ((0,0,0), (x,0,0), (x,x/2,0), (x,x/2,x/2), (x,0,x/2)). Three algorithms are tested: Kaufman's[10], the algorithm presented above and the same with double step method[12]. The times given in the following table were obtained by the total time field of the profile Unix program on a Silicon Graphic Elan Workstation. Times of Table 2 are given in seconds.

As gcd computations are slow, the corresponding improvement was abandoned in 3D algorithm. The double step increases the speed–up (20%).

7. Conclusion

Our new method is clearly the fastest. As the computing is done span by span and no more point by point, we have decreased the number of operations

x	Kaufman's	by Spans	New
100	2,50	0,37	0,32
500	1621,93	192,50	152,58
1000	26123,60	3001,48	2359,23

Table 2: *Results in user-time*

necessary to compute linear bi-interpolation. Between Kaufman's algorithm and ours the speed–up is at least 7. This value increases if the line has a gentle slope. It increases also with the length of the line: for a 1000 voxels line the ratio is 11. It will permit the use of a larger amount of objects or a reduction of the scale. Using this algorithm for a set of linear bi–interpolations in a discrete space composed of uniforms voxels, we can obtain a better interactivity between users and computers.

References

1. E. Angel and D. Morrison. Speeding up bresenham's algorithm. *IEEE Coputer Graphics & Applications*, 11:16–17, November 1991.
2. J. Berstel. *Mots, Mélanges offerts MP. Schützenbzereger*, chapter Tracé de droites, fractions continues et morphismes itérés. Hermès, 1990.
3. V. Boyer, J. Tayeb, and J.-J. Bourdin. Une accélération du tracé de droites. In *I.A.C : Intelligence Artificielle et Complexité*, pages 142–149, February 1997.
4. J.E. Bresenham. Algorithm for computer control of a digital plotter. *IBM System Journal*, 4(1):25–30, 1965.
5. J.E. Bresenham. Run length slice algorithm for incremental lines. In *Fundemental Algorithms in Computer Graphics*, pages 59–102. Springer-Verlag, 1985.
6. C.M.A. Castle and M.L.V. Pitteway. An application of euclid's algorithm to drawing straigth lines. In *Fundemental Algorithms in Computer Graphics*, pages 135–139. Springer-Verlag, 1985.
7. D. Cohen and A. Kaufman. Scan-conversion algorithm for linear and quadratic objects. *Volume Visualisation*, pages 280–300, 1991.
8. J.D. Foley, A. Van Dam, S. Feiner, and J. Hughes. *Computer Graphics, Principes and Practice*. Addison Wesley, 1990.
9. P. Graham and S.S. Iyengar. Double and triple step incremental generation of lines. In *IEEE Computer Graphics & Application*, pages 49–53, May 1994.
10. A. Kaufman. Efficient algorithms for scan-converting 3d polygons. *Computer & Graphics*, 12(2):213–219, 1988.
11. C. Metge and R. Caubet. Une structure de données discrète pour les scènes de radiosité. In *Deuxièmes journées de l'AFIG*, pages 43–51, Toulouse, 1994.
12. J.G. Rokne, B. Wyvill, and X. Wu. Fast line scan-conversion. *ACM Transaction on Graphics*, 9(4):376–388, October 1990.
13. N. Stolte and R. Caubet. Discrete ray-tracing of huge voxel spaces. In *Eurographics'95*, volume 14 of *3*, pages 383–394, 1995.