A Line Generation Algorithm over 3D Body-centered Cubic Lattice

Lijun He, Shuang Liu*, Jian Yun, Yongkui Liu
School of Computer Science and Engineering, Dalian Nationalities University, Dalian 116600, China
Email: dhnuliushuang@126.com

Abstract—New line generation algorithm is proposed for generating lines over 3D body-centered cubic lattice, a kind of optimal lattice in 3D space. The main contribution in this paper is employing the 3D Bresenham algorithm, a popular algorithm for generating 3D lines on a cubic lattice, to produce the BCC lattice occupied by 3D lines, with the help of the adjacent parallelepiped space, having the same center and basis vectors with the BCC lattice. The adjacent parallelepiped line is easy to generate by employing the existed 3D cubic Bresenham algorithm. Due to the one-to-one correspondence between the parallelogram cells of parallelepiped space and the voxels of the BCC space, the 3D BCC line generation algorithm is gained. The whole procedure is characterized by a simple discriminator and a derivation for this discriminator given in the paper confirms that all calculations can be realized using only integer arithmetic which is to implement on computer.

Index Terms—body-centered cubic, lattice, voxel, line generation

I. INTRODUCTION

In three dimensions, the digital three dimensional objects are usually generated on cubic grid on which the cubes are called voxels. The quick development of volume graphics has lead to growing interest in the 3D line generation algorithms that convert a 3D continuous line into a discrete line representation. Being basic but very important in volume graphics, these algorithms are used for synthesizing voxel-based objects in volume graphics as the 3D line itself is also used to build block for voxelizing more complex objects. The 3D line generation algorithms are also employed for ray traversal in voxel space [1, 2]. It is extended from Bresenham’s algorithm [3] to three-dimensional straight lines [4]. Similarly, an assortment of 3D scan-conversion algorithms [5] is introduced to scan-convert 3D geometric objects into their discrete voxel-map representation within the cubic lattice. Then a tripod algorithm which generates an exact 3D discrete line on the cubic lattice [6] is considered. Furthermore, an algorithm for naïve 3D discrete line drawing [7] is given. An algorithm based on a linear incremental algorithm which recognizes any set of points on the cubic lattice [8] is proposed. More recent studies [9] propose a volumetric drawing system that directly extracts and renders linear features that lay on isosurfaces within the volume.

Ray tracing techniques that cast rays (lines) through a volume of voxels are based on algorithms that generate the set of voxels visited by the continuous ray. The discrete three dimensional objects are usually generated on a cubic lattice on which the cubes are called voxels.

Although the cubic lattice seems to be more natural, alternatives to the cubic lattice in 3D have also been considered. Two honeycomb graphical models in which the voxels are hexagonal prisms are introduced in [10]. Three-dimensional lattice where the voxels are rhombic dodecahedra and truncated octahedron are called FCC (face-centered cubic) lattice and BCC (body-centered cubic) lattice, respectively. Here voxel is defined as Voronoi region of the lattice and in this paper voxel is always refers to the Voronoi region of the lattice. The BCC lattice and FCC lattice are the three-dimensional equivalents of the two-dimensional hexagonal lattice. Many recent studies have focused on BCC lattice and FCC lattice in [11-14]. Some properties and advantages of lattice based on BCC lattice and FCC lattice have been studied, e.g., in [15]. Moreover, an algorithm for computing surface skeletons on the BCC lattice and the FCC lattice is presented in [16].

The BCC lattice is the three-dimensional equivalents of the two-dimensional hexagonal lattice in the sense of the favorable volumetric sampling pattern due to its optimal spectral sphere packing property. There are several advantages for the BCC lattice over the cubic lattice.

1) On the BCC lattice, Their voxels are more "sphere-like" than cubes [14]. Therefore, surfaces consisting of BCC voxels will often appear smoother compared to the cubic lattice.

2) Voxels of the BCC lattice only have face-adjacencies (no edge-adjacency or vertex-adjacency). In other words, there are no neighboring BCC voxel pairs that share only edges or vertices. This property of BCC lattice can greatly simplify analyses and algorithms.

3) The BCC lattice has higher packing densities than the cubic lattice [13, 17]. The conclusion is that the BCC lattice is the best lattice for sampling in 3D. This leads to less memory usage and speeds up traditional volume rendering algorithms by the same ratio.

As the BCC lattice has advantages above, some researchers investigated BCC lattice in more detail and
adapted several volume rendering methods to it. Carr et al. [18] proposed marching octahedral, modified marching octahedral and modified marching hexahedra to generate isosurfaces on the BCC lattice. Theušl et al. [17] implemented splatting on BCC lattice. Sweeney et al. adapted the shear-warp algorithm to the BCC lattice. Dornhofer modified Fourier Domain Rendering for use on the BCC lattice. A vectorial algorithm [19] is established by Ibáñez et al. to trace discrete straight lines on nonorthogonal lattices in any dimension including the BCC lattice. Ray casting algorithm provides results of very high quality, usually it is considered to provide the best image quality with the slowest speed. So it is very important to accelerate the ray casting algorithm to derive the high quality of image. The aim of this paper is to give a more efficient line generation algorithm on the BCC lattice which can speed optimization of volumetric ray casting process. The algorithm is based on the adjunct parallelepiped lattice and the 3D cubic Bresenham’s line drawing algorithm. The contribution of the new algorithm is that up to three voxels can be produced in a step. As will be shown, the algorithm can be implemented using integer representation only. In this way, it is faster, and the accumulation of rounding errors is eliminated completely.

At first, we will briefly consider BCC lattice in section 2. The adjunct parallelepiped space of the BCC system and the correspondence between them are introduced in section 3. The line generation algorithm is proposed on 3D BCC lattice in detail in section 4. The experimental results are considered in section 5 and the paper is concluded in section 6.

II. BCC (BODY-CENTERED CUBIC) LATTICE

It is known that \( \mathbb{R}^3 \) can be uniformly tiled by parallelepipeds, hexagonal prisms, rhombic dodecahedrons, or truncated octahedrons. The corresponding three-dimensional lattice where the voxels are truncated octahedrons is called the BCC (body-centered cubic) lattice.

A BCC lattice can be constructed from a pre-existing cubic lattice. The following procedure is just one possible way of constructing a BCC lattice, and is by no means the only one. Starting with a unit cubic lattice, a second identical lattice is added with a translation of \((0.5, 0.5, 0.5)\), so that the new lattice nodes fall into the center of the original cubic voxels. The Voronoi regions so called voxels around all of these lattice nodes have the shape of a truncated octahedron. Consider a discrete space \( \mathbb{N} \) consisting of slices composed by truncated octahedron, consecutively placed one by one. In contrast to the case of cubic lattice, voxels here vertex-adjacency and edge-adjacency are both impossible. Thus, a connected discrete object is always face-adjacent and tunnel-free.

The voxels form a voxelized space \( \mathbb{N} \) based on the BCC lattice. A coordinate system is defined on space \( \mathbb{N} \), as follows. The coordinate system is simply one choice among many other possible coordinate systems. A voxel is chosen and its center \( O \) is defined to be the origin of the coordinate system. The origin’s coordinates are all zeros, i.e., \( O = (0, 0, 0) \). Next three coordinate axes \( OX, OY \) and \( OZ \) are fixed as shown in Figure 1, where \( OX, OY \) and \( OZ \) are “non-orthogonal” among themselves. And that \( OX, OY \) are orthogonal to adjacent two hexagons of the chosen truncated octahedron (voxel), while \( OZ \) is orthogonal to the square face that is adjacent to the above two hexagonal faces. The basis vectors \( u, v \) and \( w \) of the coordinate system are aligned with the coordinate axes, where the vector \( u \) is parallel to the \( OX \) axis and it connects the center of the origin voxel to the center of the adjacent voxel through the hexagonal face that is orthogonal to the \( OX \) axis. The vectors \( v \) and \( w \) are as similar as vector \( u \). It must be point out that the vectors \( u, v, w \) are not unit vectors. In particular, \( |u|=\sqrt{3}/2, |v|=\sqrt{3}/2, |w|=1 \).

The centers of the voxels of the BCC space form a lattice \( L \), as \( u, v \) and \( w \) form a basis of \( L \). The BCC coordinate system thus defined is denoted by \( OXYZ \). The coordinate axes \( OX, OY \) and \( OZ \) define three planes by pairs, the plane \( OXY \) (defined by the axes \( OX \) and \( OY \)), the plane \( OXZ \) (defined by the axes \( OX \) and \( OZ \)), and the plane \( OYZ \) (defined by the axes \( OY \) and \( OZ \)). The three planes divide the space into eight octants, denoted Octant I, Octant II, Octant III, Octant IV, Octant V, Octant VI, Octant VII and Octant VIII (see Figure 2). A voxel with center point coordinates \((x, y, z)\) are defined in terms of the basis vectors \( u, v, w \) as \( P=x.u+y.v+z.w \). The sign of voxel’s center point coordinate (briefly called voxel’s coordinate) is depending on the Octant in which the point lays.

![Figure 1. The axes of the coordinate system OXYZ.](image1)

![Figure 2. The octants divided by axes OX, Oy and OZ.](image2)
A BCC lattice line between voxels \((x_1, y_1, z_1)\) and \((x_2, y_2, z_2)\) is defined as an ordered sequence of voxels starting at coordinates \((x_1, y_1, z_1)\) and finishing at coordinates \((x_2, y_2, z_2)\) where a given voxel is included in the sequence a single time (no loops), and every voxel is connected through a face adjacency to the next voxel in the sequence.

III. ADJUNCT PARALLELEPIPED SPACE

To extend the Bresenham algorithm in 3D to BCC lattice, a sheared voxelized space called the adjunct parallelepiped space is needed to trace out the major voxels intersected, then remaps to the Voronoi cells of the samples on the BCC lattice. We will explain how to build the adjunct parallelepiped space in this section.

With the voxelized space based on the BCC lattice \(N\) defined in section 2, one can define an auxiliary space, called the adjunct parallelepiped space and denoted \(N_1\). Each voxel \(h(x, y, z)\) in the space \(N\) is associate to a parallelepiped \(p(x, y, z)\) in the space \(N_1\), called adjunct to \(h(x, y, z)\), with the same center and with sides determined by the basis vectors of the space \(N\), i.e., \(u, v\) and \(w\) (see Figure 3). This correspondence defines a parallelepiped discretized space \(N_1\) adjunct to \(N\) with the same center and basis vectors.

Figure 3. Voxel \(h(x, y, z)\) and its adjunct parallelepiped \(p(x, y, z)\).

Now considering more in detail the relation between the spaces \(N\) and \(N_1\), the adjacency of their cells is studied. We label a voxel from \(N\) (resp. a parallelepiped from \(N_1\)) by the coordinates of its center. Let \(h(i, j, k)\) be a voxel from \(N\) defined in terms of the basis vectors \(u, v, w\) and \(p(i, j, k)\) the corresponding parallelepiped from \(N_1\). Now with the help of Figure 4 we get the following neighbor relations of parallelepipeds and the corresponding voxels shown in table 1. There are 13 adjacencies listed which correspond to half of the 26 possible adjacencies in a cubic-like space such as the adjunct parallelepiped space when considering face, edge and vertex adjacencies. The adjacencies can be reduced to 13 due to the reflection symmetries. The 26 adjacencies of the parallelepiped are more than the 14 possible adjacencies of the truncated octahedron and that that is the reason why 12 of the parallelepiped adjacencies lead to pairs of disjoint BCC voxels. Again, due to reflection symmetries, the table lists only six out of these 12 cases.

### TABLE I. NEIGHBOR RELATIONS OF PARALLELEPIPEDS AND ADJACENCIES OF THEIR CORRESPONDING VOXELS

<table>
<thead>
<tr>
<th>Form of the pairs</th>
<th>Parallelepipeds neighbor relation</th>
<th>Voxels neighbor relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>((i, j, k)) and ((i+1, j, k))</td>
<td>Face-adjacency</td>
<td>Face-adjacency</td>
</tr>
<tr>
<td>((i, j, k)) and ((i, j+1, k))</td>
<td>Face-adjacency</td>
<td>Face-adjacency</td>
</tr>
<tr>
<td>((i, j, k)) and ((i, j, k+1))</td>
<td>Face-adjacency</td>
<td>Face-adjacency</td>
</tr>
<tr>
<td>((i, j, k)) and ((i+1, j+1, k+1))</td>
<td>Face-adjacency</td>
<td>Face-adjacency</td>
</tr>
<tr>
<td>((i, j, k)) and ((i+1, j-1, k-1))</td>
<td>Face-adjacency</td>
<td>Face-adjacency</td>
</tr>
<tr>
<td>((i, j, k)) and ((i+1, j+1, k+1))</td>
<td>Face-adjacency</td>
<td>Face-adjacency</td>
</tr>
</tbody>
</table>

Figure 4. Illustration of the neighbor type of parallelepipeds and the corresponding voxels. (a) pair of the form \((i, j, k)\) and \((i+1, j, k)\), (b) pair of the form \((i, j, k)\) and \((i, j+1, k)\), (c) pair of the form \((i, j, k)\) and \((i+1, j+1, k)\), (d) pair of the form \((i, j+1, k)\) and \((i+1, j, k)\), (e) pair of the form \((i, j+1, k)\) and \((i+1, j, k+1)\), (f) pair of the form \((i, j, k)\) and \((i+1, j, k+1)\), (g) pair of the form \((i, j, k)\) and \((i+1, j+1, k+1)\), (h) pair of the form \((i, j+1, k)\) and \((i+1, j+1, k+1)\), (i) pair of the form \((i, j, k)\) and \((i+1, j+1, k)\), (j) pair of the form \((i, j, k)\) and \((i+1, j+1, k+1)\), (k) pair of the form \((i, j, k)\) and \((i+1, j, k+1)\), (l) pair of the form \((i+1, j, k+1)\) and \((i+1, j+1, k)\).

© 2013 ACADEMY PUBLISHER
IV. Voxelization Algorithm for 3D BCC Line

The algorithm is similar to the Bresenham algorithm, which has been originally designed for the cubic lattice. The key idea is to perform the line-drawing in the parallelepiped space exploiting the one-to-one correspondence between the parallelepipeds and the truncated octahedron voxels of the BCC lattice. The obtained line is sometimes disconnected, therefore additional BCC voxels are identified which guarantee the one-to-one correspondence between the obtained line and the desired line.

Let the two endpoints of the line to be voxelized on the BCC lattice have the BCC coordinates \((x_1, y_1, z_1)\) and \((x_2, y_2, z_2)\). Then \((x_1, y_1, z_1)\) and \((x_2, y_2, z_2)\) are also the corresponding parallelepiped coordinates of the two endpoints. Let \(\Delta x = |x_2 - x_1|\), \(\Delta y = |y_2 - y_1|\) and \(\Delta z = |z_2 - z_1|\). \(x_{\text{sign}} = \text{SIGN}(x_2 - x_1)\), \(y_{\text{sign}} = \text{SIGN}(y_2 - y_1)\) and \(z_{\text{sign}} = \text{SIGN}(z_2 - z_1)\). To draw a BCC lattice line between \((x_1, y_1, z_1)\) and \((x_2, y_2, z_2)\). The algorithm solves the problem by the following steps,

Step 1. Select the octant in which the line lays as introduced in section 2. Initialize current parallelepiped and corresponding voxel at the point \((x_0, y_0, z_0)\)

Step 2. Determine the next parallelepiped in the adjacent parallelepiped space,

Step 3. Determine the next BCC voxels with the help of the one-to-one correspondence between the parallelepipeds and the BCC voxels.

Step 4. If voxel \((x_0, y_0, z_0)\) has not yet been reached, go to step 2. Otherwise, the procedure terminates.

A. Determine the Next Parallelepiped

If \(\Delta x > \Delta y\) and \(\Delta x > \Delta z\) then \(X\) is called the driving axis. The procedure of determining the next parallelepiped is divided into three parts in regard to whether \(X\), \(Y\), or \(Z\) is the driving axis. Without loss of generality, the discussion of the decision procedure will be limited to the case \(X\) is the driving axis. The algorithm takes unit steps along the direction with the driving axis coordinate \(X\). Here unit step means the coordinate of the driving axis \(X\) is always increased by one, but the other two coordinates \(Y\) and \(Z\) may be increased or not.

At each \(X\) step, the algorithm determines the \(Y\)-coordinate and the \(Z\)-coordinate of the next parallelepiped nearest the desired line path. To determine this, two integer decision variables \(e_{xy}\) and \(e_{xz}\) are computed incrementally. The decision variables \(e_{xy}\) and \(e_{xz}\) are initialized and incrementally updated for the subsequent step the same way the decision variable is maintained in the Bresenham algorithm. The decision variable \(e_{xy}\) and \(e_{xz}\) are initialized by \(e_{xy} = 2\Delta y - \Delta x\) and \(e_{xz} = 2\Delta z - \Delta x\).

Parallelepiped \(P_{\text{current}} = p(i, j, k)\) represents the parallelepiped that had been selected to be the closest to the desired line. At the current step the algorithm decides whether the next parallelepiped closest to the line should be \(A = p(i+x_{\text{sign}}, j, k)\), \(B = p(i+x_{\text{sign}}, j+y_{\text{sign}}, k)\), \(C = p(i+x_{\text{sign}}, j, k+z_{\text{sign}})\), \(D = p(i+y_{\text{sign}}, j+y_{\text{sign}}, k+z_{\text{sign}})\)(Figure 5). The decision is performed in two steps, first for \(y\), in the \(xy\) plane, \(e_{xy}\) is tested. If \(e_{xy} < 0\) then the line is closer to \(j\) than to \(j+y_{\text{sign}}, y\) is not incremented, and the candidates for the next parallelepiped are either \(A\) or \(C\) (depending on \(e_{xy}\)). If however, \(e_{xy} \geq 0\), i.e., either \(B\) or \(D\) is lying closer to the desired line, then \(y\) is incremented. Then for \(z\), in the \(xz\) plane, is similar to that of \(y\). If \(e_{xz} < 0\), then \(z\) is not incremented (either \(A\) or \(B\) is selected), otherwise \(C\) or \(D\) is selected. Thus there are four possibilities,

1) If \(e_{xy} < 0\) and \(e_{xz} < 0\), the next parallelepiped is thus \(P_{\text{next}} = p(i+x_{\text{sign}}, j, k)\). The decision variables \(e_{xy}\) and \(e_{xz}\) are incrementally updated by \(e_{xy} = e_{xy} + 2\Delta y\) and \(e_{xz} = e_{xz} + 2\Delta z\).

2) If \(e_{xy} < 0\) and \(e_{xz} \geq 0\), the next parallelepiped is thus \(P_{\text{next}} = p(i+x_{\text{sign}}, j, k+z_{\text{sign}})\). The decision variables \(e_{xy}\) and \(e_{xz}\) are incrementally updated by \(e_{xy} = e_{xy} + 2\Delta y\) and \(e_{xz} = e_{xz} + 2(\Delta z - \Delta x)\).

3) If \(e_{xy} \geq 0\) and \(e_{xz} < 0\), the next parallelepiped is thus \(P_{\text{next}} = p(i+x_{\text{sign}}, j+y_{\text{sign}}, k)\). The decision variables \(e_{xy}\) and \(e_{xz}\) are incrementally updated by \(e_{xy} = e_{xy} + 2(\Delta y - \Delta x)\) and \(e_{xz} = e_{xz} + 2\Delta z\).

4) If \(e_{xy} \geq 0\) and \(e_{xz} \geq 0\), the next parallelepiped is thus \(P_{\text{next}} = p(i+x_{\text{sign}}, j+y_{\text{sign}}, k+z_{\text{sign}})\). The decision variables \(e_{xy}\) and \(e_{xz}\) are incrementally updated by \(e_{xy} = e_{xy} + 2(\Delta y - \Delta x)\) and \(e_{xz} = e_{xz} + 2(\Delta z - \Delta x)\).

Figure 5. 3D Bresenham algorithm in parallelepiped lattice
B. Determine the Next Voxels on the BCC Lattice

With the decision of the next parallelepiped \( P_{\text{next}} \) above, one can choose the corresponding BCC voxel \( V_{\text{next}} \). Unfortunately, sometimes the current voxel and the chosen next voxel are disjoint while the corresponding parallelepipeds are edge-adjacent or vertex-adjacent. With a reference to table 1 we distinguish the cases based on the relation of the neighbor parallelepipeds and the voxels.

Case 1. The current and next parallelepipeds are face-adjacent, and the corresponding voxels are face-adjacent, as well (see Figure 4(a)-(c)).

Case 2. The current and next parallelepipeds are edge-adjacent, while the corresponding voxels are face-adjacent (see Figure 4(d)-(f)).

Case 3. The current and next parallelepipeds are vertex-adjacent, while the corresponding voxels are face-adjacent (see Figure 4(g)).

Case 4. The current and next parallelepipeds are edge-adjacent, while the corresponding voxels are disjoint (see Figure 4(h)-(j)).

Case 5. The current and next parallelepipeds are vertex-adjacent, while the corresponding voxels are disjoint (see Figure 4(l)-(m)).

For the above case 1, case 2 or case 3, the corresponding current and next voxels are face-adjacent. The next voxel of the BCC lattice line is reached. For the case 4 and case 5, the corresponding current and next voxels are disjoint. In order to eliminate the possibility for such kind of undesirable effects, in what follows we introduce the method to select the additional voxels to connect the disjoint two neighbor voxels. The method for case 4 and case 5 will be addressed in subsection C and subsection D, respectively.

C. Voxels Disjoint, while Parallelepipeds are Edge-adjacent

Consider the case that the current and next voxels are disjoint, while the corresponding parallelepipeds are edge-adjacent. With the help of table 1, the pair of voxels or parallelepipeds has one of the following three forms, \((i, j, k)\) and \((i, j+1, k+1)\), \((i, j, k)\) and \((i+1, j, k+1)\), \((i, j, k)\) and \((i+1, j+1, k)\). Without loss of generality, the decision will be limited to the form \((i, j, k)\) and \((i+1, j+1, k)\). The other two are similar.

To connect the voxels \(h(i, j, k)\) and \(h(i+1, j+1, k)\), there are two candidate voxels \(h(i+1, j, k)\) and \(h(i, j+1, k)\). Consider the \(XY\) plane at \(z=k\), the decision is based on whether the line segment is above point \(B\) in Figure 6.

That is to determine whether \(t_{\text{v}}\) is greater than \(1/2\). Notice that \(t_{\text{v}} - 1/2 = d_{\text{v}} - h - 1/2 = (1/2 + e_{\text{v}} - \Delta y)/2\Delta y - 1/2 = e_{\text{v}} - \Delta y/2\Delta y\) and \(\Delta y\) is greater than zero, the decision comes to compare \(e_{\text{v}}\) with \(\Delta y\). If \(e_{\text{v}} < \Delta y\) then the next two voxels are thus \(h(i+1, j, k)\) and \(h(i+1, j+1, k)\). Otherwise the next two voxels \(h(i, j+1, k)\) and \(h(i+1, j+1, k)\) are preferred.

D. Voxels Disjoint, while Parallelepipeds are Vertex-adjacent

Consider the case that the current and next voxels are disjoint, while the corresponding parallelepipeds are vertex-adjacent. With the help of table 1, the pair of voxels or parallelepipeds has one of the following three forms,

1. \((i, j, k)\) and \((i+1, j-1, k+1)\)
2. \((i, j, k)\) and \((i+1, j+1, k+1)\)
3. \((i, j, k)\) and \((i+1, j+1, k)\)

The form 1 and 2 are similar in that each need one additional voxel to eliminate the disjointing, and we will focus on form 1). The form 3) needs two additional voxels. The detail for form 1) and form 3) will be addressed in subsection E and subsection F, respectively.

E. Form \((i, j, k)\) and \((i+1, j-1, k+1)\)

Note the current voxel \(h(i, j, k)\) and next voxel \(h(i+1, j-1, k+1)\) are disjoint. To adjoin them, there are four candidate voxels \(h(i, j, k-1)\), \(h(i, j, k), h(i+1, j-1, k)\) and \(h(i+1, j, k)\). The decision is performed in two steps, in the \(xz\) plane, \(e_{\text{v}} < \Delta x\) is tested. There two cases, \(e_{\text{v}} < \Delta x\) or \(e_{\text{v}} \geq \Delta x\).

If \(e_{\text{v}} < \Delta x\) then the line is closer to \((i+1, k)\) than to \((i, k+1)\), and the candidates for the next voxel are thus either \(h(i+1, j, k)\) or \(h(i+1, j-1, k+1)\) (depending on \(e_{\text{v}}\)). Then in the \(zy\) plane if \(e_{\text{v}} = \Delta y - e_{\text{v}} \Delta z < -\Delta x \Delta y\) (i.e., \(d_{\text{v}} + d < 0\) see Figure 7), then \(y\) is incremented and \(h(i+1, j-1, k)\) is selected. If, however, \(e_{\text{v}} = \Delta y - e_{\text{v}} \Delta z \geq -\Delta x \Delta y\), then \(y\) is not incremented and \(h(i+1, j, k)\) is selected.

If, however, \(e_{\text{v}} \geq \Delta x\), i.e., either \(h(i, j, k+1)\) or \(h(i+1, j, k)\) is lying closer to the desired line. Then in the \(ZY\) plane if \(e_{\text{v}} = \Delta y - e_{\text{v}} \Delta z < \Delta x \Delta y\) (i.e., \(d_{\text{v}} + d < 1\) see Figure 7), then \(y\) is incremented and \(h(i+1, j+1, k)\) is selected. If, however, \(e_{\text{v}} = \Delta y - e_{\text{v}} \Delta z \geq \Delta x \Delta y\), then \(y\) is not incremented and \(h(i+1, j, k+1)\) is selected.

F. Form \((i, j, k)\) and \((i+1, j+1, k+1)\)

Note the current voxel \(h(i, j, k)\) and next voxel \(h(i+1, j+1, k+1)\) are disjoint. There are three candidate voxels next to \(h(i, j, k)\) which are \(h(i+1, j, k)\), \(h(i+1, j, k)\) and \(h(i, j+1, k)\). The decision is performed in two steps, first for \(y\),

---

Figure 6. Illustration of decision in XY plane while the neighbor parallelepipeds are edge-adjacent.
in the xy plane, \( e_{ys} < \Delta y \) is tested. Then for \( z \), in the xz plane, \( e_{sz} < \Delta z \) is tested. There are four possibilities,

Possibility 1, If \( e_{ys} < \Delta y \) and \( e_{sz} < \Delta z \), then the voxel next to \( h(i, j, k) \) is thus \( h(i+1, j, k) \).

Possibility 2, If \( e_{ys} < \Delta y \) and \( e_{sz} \geq \Delta z \), the voxel next to \( h(i, j, k) \) is thus \( h(i, j+1, k) \).

Possibility 3, If \( e_{ys} \geq \Delta y \) and \( e_{sz} < \Delta z \), the voxel next to \( h(i, j, k) \) is thus \( h(i, j, k+1) \).

Possibility 4, If \( e_{ys} \geq \Delta y \) and \( e_{sz} \geq \Delta z \), the voxel to \( h(i, j, k) \) is thus \( h(i+1, j+1, k+1) \).

First considering the case of \( e_{ys} < \Delta y \) and \( e_{sz} < \Delta z \), then the voxel next to \( h(i, j, k) \) is thus \( h(i+1, j+1, k+1) \).

Then there are two possible paths from \( h(i+1, j, k) \) to \( h(i+1, j+1, k+1) \). By an analysis, we obtain a simple discriminator \( e_{ys} = e_{ys} \Delta y - e_{sz} \Delta z \) where \( e_{ys} \) and \( e_{sz} \) are the Bresenham’s error in the XY plane and XZ plane. If \( e_{ys} \) is positive, \( z \) coordinate should be increased before \( y \) coordinate, the selected three voxels next to \( h(i, j, k) \) is \( h(i+1, j, k), h(i+1, j+1, k+1) \). When \( e_{ys} \) is negative or equal zero, the selected three voxels next to \( h(i, j, k) \) is \( h(i, j+1, k), h(i+1, j+1, k+1) \) and \( h(i, j+1, k+1) \). The deriving process of \( e_{sz} \) follows. According to Figure 7, which shows the voxels in YZ plane at \( x+1 \), if the line is above the point \( E \), i.e. \( d_{ys} + d > \frac{1}{2} \), the \( z \) coordinate should be increased first; otherwise, the \( y \) coordinate should be increased first. As shown in Figure 7, \( d \) can be easily calculated as

\[
d = \left( \frac{1}{2} - d_{ys} \right) \Delta z / \Delta y
\]

and the formula \( d_{ys} + d > \frac{1}{2} \) can be rewritten as

\[
d_{ys} + \left( 1 - \frac{1}{2} - d_{ys} \right) \Delta z / \Delta y > \frac{1}{2}
\]

Bresenham’s error \( d_{ys} \) (or \( d_{sz} \)) is transformed to \( e_{ys} \) (or \( e_{sz} \)).

\[
\begin{align*}
\frac{d_{ys}}{2} &= \frac{1}{2} + \frac{e_{ys}}{2\Delta y} \\
\frac{d_{sz}}{2} &= \frac{1}{2} + \frac{e_{sz}}{2\Delta z}
\end{align*}
\]

Substituting the expressions of \( d_{ys} \) and \( d_{sz} \) in (2) into (1) gives,

\[
\frac{1}{2} + \frac{e_{ys}}{2\Delta y} + \frac{1}{2} - \left( \frac{1}{2} + \frac{e_{sz}}{2\Delta z} \right) \Delta z / \Delta y > \frac{1}{2}
\]

i.e.

\[
\frac{e_{ys} \Delta y - e_{sz} \Delta z}{2\Delta y \Delta z} > 0
\]

Since both \( \Delta z \) and \( \Delta y \) are positive (absolute values), as a discriminator, (3) equals

\[
e_{ys} = e_{ys} \Delta y - e_{sz} \Delta z > 0
\]

If \( e_{ys} < \Delta y \) and \( e_{sz} \geq \Delta z \), the voxel next is thus \( h(i, j, k + 1) \) which is unfortunately disjoint to \( h(i+1, j+1, k+1) \). To adjoin them, there are two candidate voxels \( h(i+1, j+1, k+1) \) and \( h(i, j+1, k+1) \) and \( h(i, j+1, k+1) \). Because of \( e_{ys} < \Delta y \), \( h(i+1, j+1, k+1) \) is selected.

If \( e_{ys} \geq \Delta y \) and \( e_{sz} < \Delta z \), the voxel next is thus \( h(i, j+1, k) \) which is unfortunately disjoint to \( h(i+1, j+1, k+1) \). To adjoin them, there are two candidate voxels \( h(i+1, j+1, k) \) and \( h(i, j+1, k+1) \). Because of \( e_{sz} \leq \Delta z \), \( h(i+1, j+1, k) \) is selected.

If \( e_{ys} \geq \Delta y \) and \( e_{sz} \geq \Delta z \), the voxel to is thus \( h(i, j+1, k) \) or \( h(i, j, k + 1) \). The decision is based on the discriminator \( e_{ys} = e_{ys} \Delta y - e_{sz} \Delta z \) in the yz plane, \( e_{ys} < 0 \) is tested. If \( e_{ys} < 0 \) then \( h(i, j, k + 1) \) and \( h(i, j+1, k+1) \) are selected, otherwise \( h(i, j+1, k+1) \) and \( h(i+1, j+1, k) \) are selected.

V. ANALYSES AND COMPARISONS OF ALGORITHMS

The proposed algorithm uses only integer arithmetic, and only addition, subtraction, shifting, and comparison are employed except uses two multiplications in the decision for the two voxels to adjoin the line. At the same time, with the decision, three voxels are generated in only one step.

In order to prove the efficiency of the new algorithm, it is compared to the existing line tracing algorithm on BCC lattice, Ibáñez-Hamitouche-Roux’s algorithm (IHR) [19].

The operations at each step in the Ibáñez-Hamitouche-Roux’s algorithm, as published in their paper, are two vector addition and selecting the index of the minimum element in a vector. The number of numerical operations in each step is counted and shown in table II. Here as the operations for updating the \( X \), \( Y \) and \( Z \) coordinates of voxels being pierced are the same for both algorithms, they are omitted during the counting. As can be seen from table II, the number of the new algorithm’s arithmetic operations is smaller than the Ibáñez-Hamitouche-Roux’s algorithm.
Both algorithms tested were implemented in C++ and executed on Intel's Pentium 4 2.53GHz. For testing, a parallelepiped was embedded in the lattice of voxels. The line was voxelized by connecting the center of the parallelepiped and the points on the parallelepiped. By changing the edge length of the parallelepiped, different line segments were voxelized. The comparison of CPU time for both tested algorithms is shown in table 3. The times include the execution-time in the main loop and the time for both tested algorithms is shown in table 3. The comparison of CPU time for both tested algorithms is shown in table 3. The times include the execution-time in the main loop and the execution-time needed to set the initial values outside the main loop. All input/output operations are excluded. As shown in table 3, the proposed algorithm is more efficient than the Ibáñez-Hamitouche-Roux’s algorithm.

<table>
<thead>
<tr>
<th>Operation</th>
<th>IHR algorithm</th>
<th>New algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Division</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Multiple</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Increase/decrease</td>
<td>7 3 (1.5)</td>
<td>4(1.33)</td>
</tr>
<tr>
<td>Comparison</td>
<td>3 3 (5.2)</td>
<td>6(2)</td>
</tr>
</tbody>
</table>

Both algorithms tested were implemented in C++ and executed on Intel's Pentium 4 2.53GHz. For testing, a parallelepiped was embedded in the lattice of voxels. The line was voxelized by connecting the center of the parallelepiped and the points on the parallelepiped. By changing the edge length of the parallelepiped, different line segments were voxelized. The comparison of CPU time for both tested algorithms is shown in table 3. The times include the execution-time in the main loop and the execution-time needed to set the initial values outside the main loop. All input/output operations are excluded. As shown in table 3, the proposed algorithm is more efficient than the Ibáñez-Hamitouche-Roux’s algorithm.

### References


Lijun He received the B.S. degree in computational mathematics in 1996 from Xiangan University, Xiangan, China. She received the M.S. degree in computational...
mathematics in 1999 from National University of Defense Technology, Changsha, China. She is currently an assistant professor in the School of Computer Science and Engineering at Dalian Nationalities University, Dalian, China. Her research interests are computer graphics and data processing.

Yongkui Liu is a Professor in the School of Computer Science and Engineering at Dalian Nationalities University, Dalian, China. He received the B.S. degrees in computer science from Jilin University, Changchun, China, and received his M.S. degrees in computer science from Shenyang University of Technology, Shenyang, China. He has his Ph. D degree in computer science from Zhejiang University, Hangzhou, China. His research interests lie in Computer Graphics, Pattern Recognition and Image Processing.

Shuang Liu, born in Jinzhou, Liaoning, China, received her Ph.D degree of Traffic Information Engineering & Control from Dalian Maritime University in 2006. She currently works as an Associate Professor at the college of computer science & engineering, Dalian Nationalities University, Dalian, China. Her research interests include computer education, intelligent transportation system and machine learning algorithms.

Jian Yun was born in Baotou, Inner Mongolia, China in 1975. He received his M.S. degree in Computer Science from from Inner Mongol University, China in 2003, and received the Ph.D degree in linguistics in the College of Humanities and Communications, Shanghai Normal University, Shanghai, China in 2010. His research interests lie in Computer Graphics and Image Processing.