UNIVERSITY OF WEST BOHEMIA
Faculty of Applied Sciences
Department of Computer Sciences

Comparison of Tetrahedron Subdivision schemes of Volumetric Data

DIPLOMA THESIS

Plzeň, Czech Republic
May 1999

Alexandre Brusi Terrades

1. Index

1. INDEX **2**

2. ABSTRACT **3**

3. SOURCES OF VOLUME DATA **3**

4. VOLUME VISUALIZATION STEPS **4**

5. VOLUME VISUALIZATION METHODS **4**

6. DATA CLASSIFICATION AND TRAVERSALS **5**

7. SHADING **6**

8. MARCHING CUBES **9**

9. MARCHING TETRAHEDRA **11**

A. DECOMPOSITION IN 5 OR 6 TETRAHEDRA 11
B. DECOMPOSITION IN 12, 24 AND 48 TETRAHEDRAS 13
C. TETRAHEDRAL TESSELLATION OF SPACE FROM THE CENTRED CUBIC LATTICE 15

10. IMPLEMENTATION FIRST ALGORITHM **16**

11. IMPLEMENTATION SECOND ALGORITHM **21**

12. COMPARISON OF TWO METHODS **25**

A. CONSTRUCTION TIME 26
B. NUMBER OF TRIANGLES 28
C. RADIUS ERROR FROM EACH VERTEX 29
D. RADIUS ERROR FROM CENTRE OF GRAVITY 31
E. AREA ERROR 32
F. VOLUME ERROR 33
G. IMAGE COMPARISON 36

13. CONCLUSIONS **37**

14. BIBLIOGRAPHY **38**

APPENDIX A. IMAGES 40
APPENDIX B. CDROM CONTENT 50

2. Abstract

There exist various methods for extracting isosurfaces from volume data.

A Marching Tetrahedra algorithm with a decomposition in six tetrahedra and Marching Tetrahedra with a tetrahedral tessellation of the space from the cubic lattice are implemented and are compared.

There is a short introduction to visualization of volume data, which are the volume visualization steps, methods of volume visualization, data classification and Shading.

Introduction of Marching Cubes method and various Marching Tetrahedra methods are explained, too.

Both implemented methods are described in more detail before the comparison section.

Comparison of the both methods are made in different aspects: isosurface extraction process time, number of triangles generated and when approximation of a sphere is made, radius, area and volume errors. There is a comparison of images generated.

Finally some conclusions are presented.

3. Sources of volume data

Volume data sets are often acquired by scanning a material of interest using Magnetic Resonance Imaging (MRI), Computed aided Tomography (CT), Positron Emission Tomography (PET), and/or Sonogram machines. Laser scan confocal and other high power microscopes are also used to acquire data [4].

All of these data sets can be treated similarly even though they are generated by diverse means. The amount of structure in the data set usually determines which volume visualization algorithm will create the most informative images.

Particular data acquisition devices, such as CT and MRI scanners, are good at sampling a specific characteristic of a substance. This characteristic is usually different for each kind of device used. For example, MRI scans shows soft tissue tumours that do not show up in CT scans.

MRI can be used for investigating blood vessels [3] for surgical planning and in the diagnosis of diseases that change blood flow. CT data can be used for Cranioplasty (the procedure of repairing defects, usually holes, in the skull with cranial implants [10]).

One of the applications of visualization algorithms is in medical data. 3D medical data differs from traditional computer graphics in two ways; (1) the data from which images are formed are captured from real objects rather than synthesised by a designer or generated mathematically from a computer model, (2) accurate depiction is more important than pleasing images. There are two volume visualization paradigms used in medical imaging; surface rendering and volume rendering [13].

4. Volume Visualization steps

The initial step in every procedure is data acquisition [4]. The next common step is to put the data slices into a form that can be worked with and then to process each slice so that it covers a good distribution of values, is high in contrast, and is free of noise and out-of-range values. Of course, the same set of processes must be applied to every slice.

Furthermore, the data set is reconstructed so that the ratio of the dimensions is proportional to the ratio of the dimensions of the measured substance or substance being simulated. This may involve interpolating between values in adjacent slices to construct new slices, replicating existing slices, interpolating to estimate missing values, or scan-converting an irregular grid or non-orthogonal grid onto a Cartesian grid by interpolation. At this point three-dimensional enhancement may be applied, such as a slight blurring of the data values. Next, a data-classification or threshold is performed.

After data-classification, a mapping operation is applied to map the elements into geometric or display primitives. This is the stage that varies the most between algorithms. At this point the primitives can be stored, manipulated, intermixed with externally defined primitives, shaded, transformed to screen space, and displayed.

Most of the fundamental volume visualization algorithms include only a subset of the above steps.

5. Volume visualization methods

Volume visualization algorithms can be divided into two categories, direct volume rendering (DVR) algorithms and surface-fitting (SF) algorithms [5].

DVR algorithms are categorised by mapping elements directly into screen space without using geometric primitives as an intermediate representation. DVR methods are especially appropriate for creating images from data sets containing amorphous features like clouds,

fluids, and gases. One disadvantage of using DVR methods is that the entire data set must be traversed each time an image is rendered. A low resolution and quality of a DVR image over time is called progressive refinement [4].

SF algorithms (sometimes called feature-extraction or isosurfaces) typically fit (usually planar) surface primitives such as polygons or patches to constant-value contour surfaces in volumetric data sets. The user begins by choosing a threshold value and then geometric primitives are automatically fit to the high-contrast contours in the volume that match the threshold. Cells whose corner-values are all above the chosen threshold (cell is inside) or all below the threshold (cell is outside) are discarded and have no effect on the final image. Showing just the cells falling on the threshold is sometimes useful, but can be a problem. Another consideration is the huge number of surface primitives generated for large volumetric data sets.
The SF approach includes contour-connecting, marching cubes, marching tetrahedra, dividing cubes, and others. SF methods are typically faster than DVR methods since SF methods only traverse the volume once to extract surfaces. After extracting the surfaces, rendering hardware and rendering methods can be used to quickly render the surface primitives each time the user changes a viewing or lighting parameter. Changing the SF threshold value is time consuming because it requires that all of the cells be revisited to extract a new set of surface primitives.

SF methods suffer from several problems such as occasional false positive and negative surface pieces, and incorrect handling of small features and branches in the data. Note that such artifacts could be incorrectly viewed by scientists as features in the data.

Table 1 shows a taxonomy of a well-known algorithms [4]:

Volume Visualization Algorithms		
Surface-fitting	**Direct Volume Rendering**	
	Projection methods	Image-order methods
Opaque cubes (Cuberille) Contour connecting Marching Cubes Marching Tetrahedra	V-buffer Splatting	Ray-casting Cell integration Sabella method

Table 1

6. Data classification and traversals

Data classification is the most difficult function that a volume visualization user has to perform. Data classification means choosing a threshold if a SF algorithm is to be used or choosing colour (brightness) and opacity (light attenuation) values to go with each possible range of data values to meaningful colours. The DVR opacity table is used to expose that part of the volume most interesting to the user and to make transparent those parts that are not interesting [4].

In DVR each element in a volume contributes colour to the final image. The amount of colour contributed by an element is determined by the colour and opacity of the current element and the colours and opacities of all of the elements between the current element and the image plane. To get a reasonable image, the user must choose the tables with care.

Once a scientific data set has been classified, images are typically created in one of two ways: either using an image-order traversal of the pixels in the image plane or an object-order traversal of the elements in the volumetric data set. Object-order traversals operate by calculating, for each element, the projection and contribution of that element to the pixels in the image plane. Most SF and some DVR algorithms use object-order traversals and some algorithms use a combination of both types of traversal.

Object-order traversals can proceed through the volume either front-to-back or back-to-front. The advantage of front-to-back traversals is that the elements in the back do not have to be traversed if the elements in the front have already created a completely opaque image. The advantage of back-to-front traversals is that the user can watch the image progressing, seeing the structures in the back that will eventually be hidden by the structures in the front.

Image-order traversals usually proceed in scanline order. Alternatively, pixels can be processed in a random order so that the user can watch the image being refined as the missing pixels are filled in.

7. Shading

Normals vectors are used for surfaces that are described with polygonal data such that surfaces appear smooth rather than faceted [16].

To find the normal for a flat polygon it's taken three vertices v_1,v_2 and v_3 of the polygon that do not lie in a straight line. The cross product $[v_2-v_1]x[v_3-v_1]$ is perpendicular to the polygon.

The calculation of the normal vector (u,v,w) in 3D space would be for the triangle illustrated in Fig.1, where $v_1 = (x_1, y_1, z_1)$, $v_2 = (x_2, y_2, z_2)$, $v_3 = (x_3, y_3, z_3)$:

$$(u,v,w) = [v_2-v_1]x[v_3-v_1] = [(x_2- x_1), (y_2- y_1), (z_2- z_1)] \times [(x_3- x_1), (y_3- y_1), (z_3- z_1)] =$$
$$= ((y_2-y_1)(z_3-z_1)-(z_2-z_1)(y_3-y_1),\ (x_3-x_1)(2z-z_1)-(x_2-x_1)(z_3-z_1),\ (x_2-x_1)(y_3-y_1)-(y_2-y_1)(x_3-x_1))$$

The normalised vector is:

$$\left(\frac{u}{N}, \frac{v}{N}, \frac{w}{N}\right)$$

Where,

$$N = \sqrt{u^2 + v^2 + w^2}$$

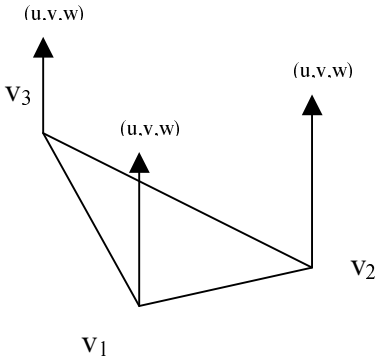


Fig.1

A more realistic and detailed presentation is obtained if the color information present in the data is taken into account. As a consequence of the tomographic acquision process, the color levels in the 3D neighbourhood of a surface voxel are representing the relative proportions of different materials inside the voxels [14].

For surface rendering techniques, the gradient is used as an estimation of the surface normal which is used for shading [7,9].

The gradient of a 3D intensity function is the partial derivative of that intensity function with respect to all three directions. Given a function *f(x,y,z)* the gradient is [11]:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$$

In volume rendering, a 3D data set consists of discrete samples of *f(x,y,z)* called voxels. If this function *f(x,y,z)* is not known, the gradient is calculated using this voxels. For simplicity only is considered the case when voxel is a cuboid.

Gradient within a cell can be estimated by interpolation between the gradients at the corners of the cell. The gradient at a point in a volume can be used to approximate the normal to an imaginary surface passing through the point.

Gradient estimation can be analysed in a manner similar to interpolation. When the gradient is needed at a location other than a given voxel point, some kind of reconstruction filter has to be used to estimate the derivative (in each direction) of the *f(x,y,z)*

.

Several methods exist to estimate a local gradient in volume data sets [2,7,9,14].

One of the most commonly used methods in volume rendering, referred as the central difference method [14], calculates the components of a gradient G for a surface voxel at *(i,j,k)*, using the values of its six neighbourhood along the main axes as:

$$G_x = \frac{g(i+1, j, k) - g(i-1, j, k)}{\Delta x}$$

$$G_y = \frac{g(i, j+1, k) - g(i, j-1, k)}{\Delta y}$$

$$G_z = \frac{g(i, j, k+1) - g(i, j, k-1)}{\Delta z}$$

Normalization of G yields the surface normal.

Fig.2 illustrates the above operation.

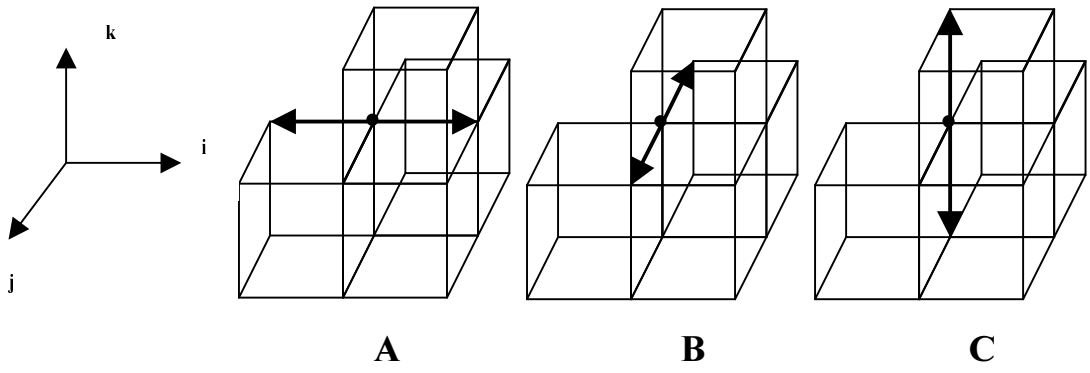


Fig.2

Gradient in the point marked with a black circle in **A** the i direction, **B** the j direction, **C** the k direction.

Mathematically the central difference method can be described as follows:

Suppose we have the values *f(-1)*, *f(0)*, *f(1)*, *f(2)*. Using the central difference method, gradients are calculated on voxel positions as follows:

$$g(0) = \frac{f(1) - f(-1)}{\Delta x} \qquad g(1) = \frac{f(2) - f(0)}{\Delta x}$$

Where g(x) denotes the gradient at position x.

If the gradient is required on position x with x between 0 and 0.5, the approximation with the central difference method equals:

$$g(x) = (1-x)g(0) + xg(1) = (1-x)\frac{1}{2}(f(1) - f(-1)) + x\frac{1}{2}(f(2) - f(0)) = \frac{1}{2}[(f(1) - f(-1)) + x(f(-1) - f(0) - f(1) + f(2))]$$

Alternatively, all 26 neighbours of a voxel can be used to calculate the gradient. This is usually a better estimation of the gradient, but takes more time to calculate. Another disadvantage is that additional smoothing is introduced.

There is a slightly different method than the central difference method. The gradient using two neighbouring values. In that case the gradient is calculated in between the original voxel positions. Estimating the gradient on sample positions requires a linear interpolation between the closest gradient values. In [9] this method is referred as the intermediate difference method.

8. Marching Cubes

In the marching cube algorithm, a cube is formed by eight adjacent data points on two consecutive slices [5]. First of all the user specifies a threshold value. For this value, some voxels will be entirely inside (Fig.3 (A)) or outside (Fig.3 (B)) the corresponding isosurface and some voxels will be intersected by the isosurface (Fig.3 (D)).

Cubes are processed in a row-column and slice by slice order, see Fig.4.

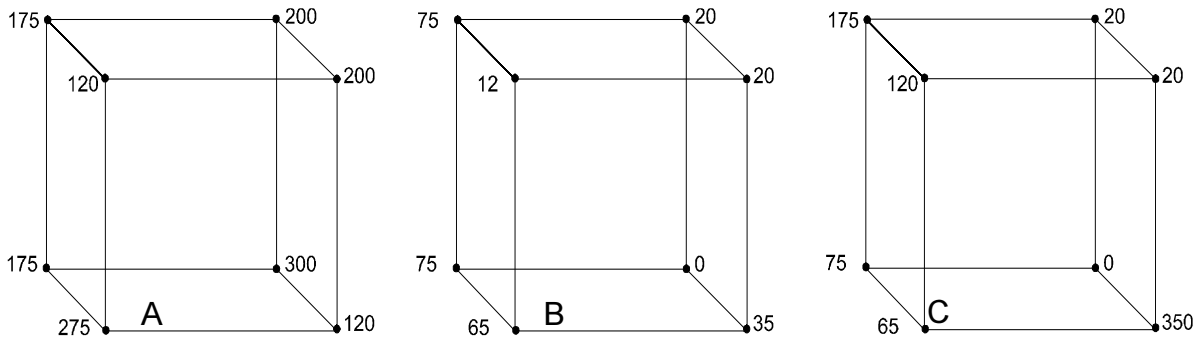


Fig.3

If threshold is set to 100: (a) cube is entirely inside (b) cube is entirely outside (c) intersection of the isosurface with the voxel.

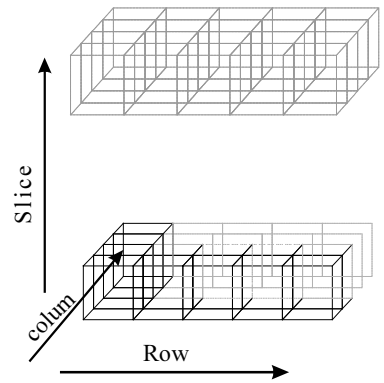


Fig.4

Voxels intersected by the threshold are examined and a set of one or more polygons is produced, which are then output for rendering; cubes with all vertices in the same *on* state (values are above the threshold) or *off* state (values are below the threshold) are not used for the generation of the contour isosurface.

Each of the voxel's 8 vertices can be either above or below of the isosurface value, thus, there are $2^8 = 256$ possible ways in which a surface can intersect the voxel [1].
This number can be reduced to 128 by assuming that the vertices in on and vertices in off, can be switched. The number can be reduced to 15 patterns if inversion symmetry and rotation is used.
(Actually the number can be further reduced to 14 if reflection symmetry is taken into account too [2]).

Fig.5 illustrates the 15 patterns.

Points on the edges intersected by the surface can be calculated via interpolation of the data. Furthermore, unit normals can also be calculated at the grid points and subsequently interpolated onto the surface elements.
We can create triangular patches that divide the cube into regions within the isosurface and regions outside. By connecting the patches from all cubes on the isosurface boundary, we get a surface representation.

A problem facing the marching cubes algorithm is that ambiguity may arise when cubes of certain patterns are adjacent to each other, leading to holes in the resulting contour surface [2].

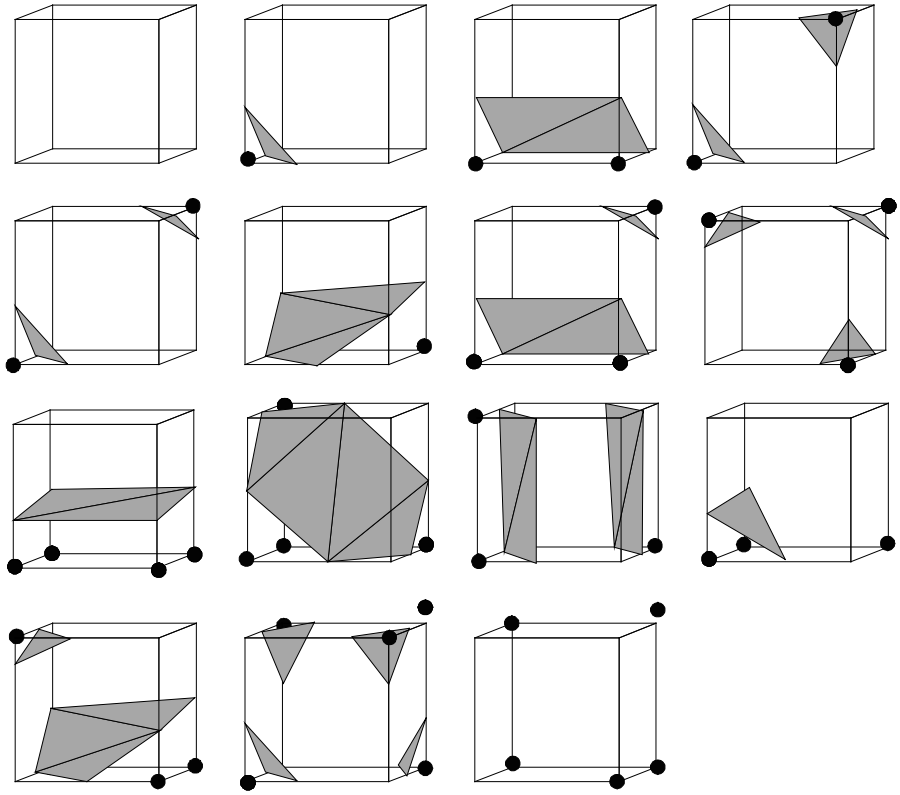


Fig.5

9. Marching Tetrahedra

The MT algorithm is a modification of the Marching Cubes [8].

It has been observed that the pattern mismatch problem can be overcome by the use of tetrahedral instead of the cubic cells.
The Marching Tetrahedra algorithm splits up each cube in the Marching Cube algorithm into various tetrahedra first and then intersections of the isosurface and tetrahedra edges are computed.

There have been presented various decomposition methods.

A. Decomposition in 5 or 6 tetrahedra

A lattice is defined as an infinite array of points in space with identical environments, including orientation. It offers a natural starting point for constructing a repeating tessellation of space. There are altogether only 14 types of lattices in 3D space, called the Bravais lattices. These 14 lattice types can be classified into seven systems which are characterised by different symmetry elements.

There are 3 types of lattices in the cubic system: the cubic primitive, the body-centred, and the face-centred cubic lattices, see Fig.6.

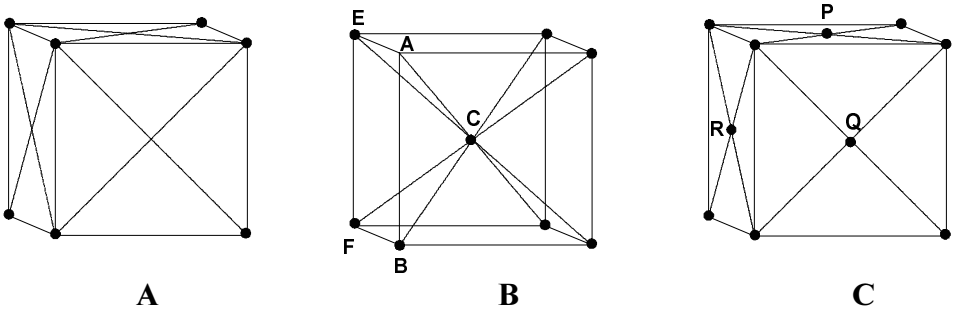


Fig.6

The three cubic lattices: (a) the cubic primitive (b) the body-centred cubic, (c) the face-centred cubic. For clarity purposes, features on backward facing faces (which are identical to the forward facing faces) are not shown in (a) and (c).

For the primitive lattice, after the first neighbours are connected, cubes are formed.

The smallest number of tetrahedrons into which a cube can be partitioned is five [15].
The demonstration is easy, tetrahedral faces are triangular, so each cube face requires at least two tetrahedral faces. So at least 12 tetrahedral faces are needed in all.
At most three faces of a tetrahedron can be mutually orthogonal (and no two faces can be parallel), so at most 3 faces from each tetrahedron can contribute towards these 12. So we require at least 4 tetrahedra to provide the cube faces. But these tetrahedra each have volume at most 1/6 (1/3 x face area x 1, and face area is at most 1/2).
So if we have only 4 tetrahedra in total then their total volume is less than the cube's volume. Contradiction. Hence we need at least 5 tetrahedra.

The decomposition in 5 tetrahedra can be made by cutting the cubic primitive up along some of the lines joining the second neighbours (Fig.7). The cube is break down to four identical tetrahedras plus a fifth (a regular tetrahedron) at the centre that is twice as big as any of the other four.

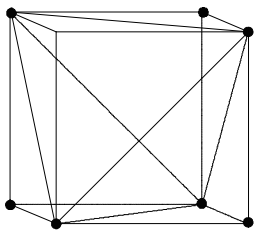


Fig.7

Subdivision the cube in five or six tetrahedra are well-known used decompositions. In Fig.8 there are two examples of the decomposition in six tetrahedra and one of five.

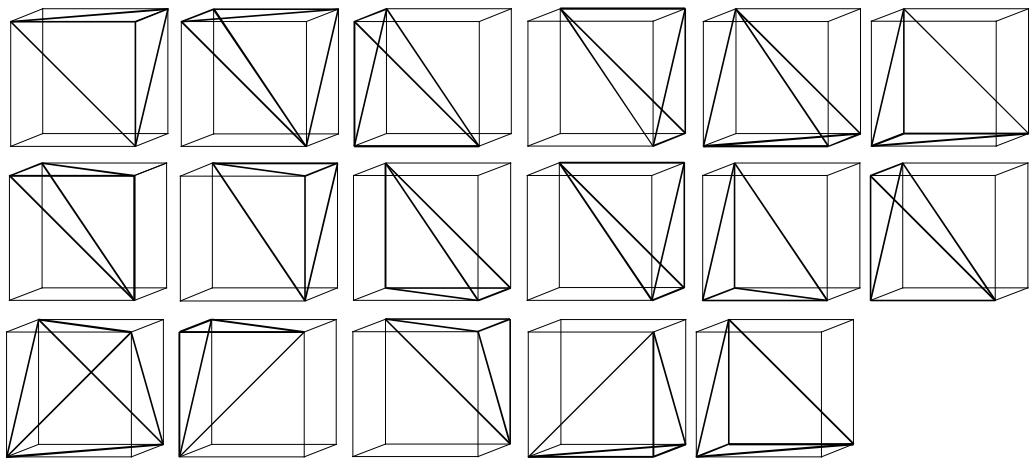


Fig.8

5-6 tetrahedra decompositions have several disadvantages. The tetrahedra are generally uneven, often mutually different or asymmetrical (i.e. some of the cell vertices contribute in more resulting of the cell vertices contribute in more resulting tetrahedra tan others), result of which is that boundary representation generated using such decompositions is not visually acceptable.

B. Decomposition in 12, 24 and 48 tetrahedras

There were proposed three decompositions that full fill the following requirements, see [2]:

- The decomposition is symmetrical (in the meaning explained in the previous paragraph).
- There is no information loss in comparison with the original voxel scene. This is very important in medicine, where no changes of original data are permitted.
- Tetrahedra are at the most two shapes: the less different shapes the tetrahedra have, the simpler is a possible hardware implementation.

One of the new decompositions is shown in Fig.9.

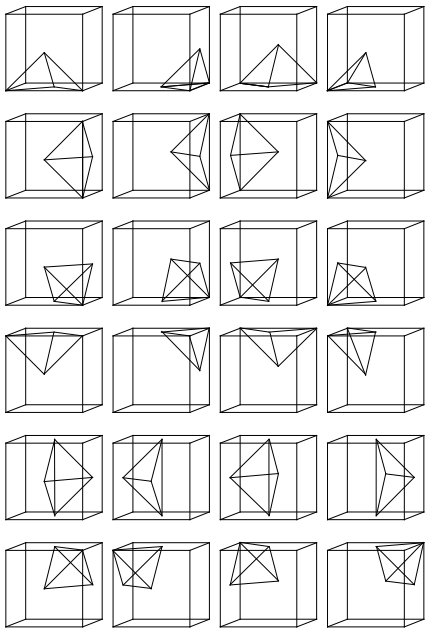


Fig.9

In addition to the values in the original eight cell vertices, some new vertices are used. In Fig.9, the centre of each cell side is given a value that is an average value of the nearest 4 vertices values (is based in the face-centred cubic lattice, see Fig 6 (A)).
Another decomposition can be made when also a value of the cell's centre of gravity is computed as an average value from the known 8 vertices as a result of trilinear interpolation in the cell (is based in the face-centred and the body-centred cubic lattice). A decomposition into 48 tetrahedra can be achieved from the previous decomposition by adding centres of the cell edges and thus splitting each tetrahedron in two.

Since there is no information available about values between the cell vertices, linear interpolations seem to be the best approximation.
Such interpolations have no influence on the eight known voxel values. This fact is important especially when the algorithm is used for medical data visualization purposes.

Now let's consider the different patterns of the vertices on-off states of each tetrahedron. When all four vertices are at the same state, no surface element needs to be constructed.
When three vertices assume a state different from the fourth one, a surface can be constructed parallel to the triangle defined by the three vertices.
Since all triangles are identical on the tetrahedron, there is only one pattern.
Finally, when two vertices assume a state different from the other two, there can be two (not significantly different) possibilities: (a) identical vertices are joined by a long edge, or (b) identical vertices are joined by a short edge. In either case, a quadrilateral surface element can be constructed bisecting the tetrahedron. Hence we see that there are altogether only four possible patterns for each tetrahedron, as opposed to 15 in the marching cubes set up.

In volume and surface criteria it is proved the 24 and 48 tetrahedra decomposition is useless than the 5, 6 and 12 ones. Nevertheless, the visual impression is obviously better, if finer subdivision is used. According to their experience, the results indicate that error measurement should be based on a gradient change, because a human eye is sensitive to edges and gradient changes of a surface [8].

C. Tetrahedral tessellation of space from the centred cubic lattice

A new method is presented with a tetrahedral tessellation of space from the body-centred cubic lattice in the manner illustrated in Fig.10, see [2]. All tetrahedra obtained in this way have the same shape although their orientations differ.

There are two types of edges on this tessellation of tetrahedra. The short edges join a lattice point with one of its eight nearest neighbours. Examples are AC, AD, BC and BD in Fig.10.
The long edges join a lattice point with one of its six second-nearest neighbours. Examples are AB and CD in Fig.10.

There are two long edges and four short edges on each tetrahedron. There are four tetrahedra surrounding each long edge and six tetrahedra surrounding each short edge. Similar to the previous algorithm, the intersections of the contour surface on the edges of the tetrahedra are interpolated to improve the quality of the surface.

There is only one tetrahedral shape involved in this decomposition and symmetric overall.

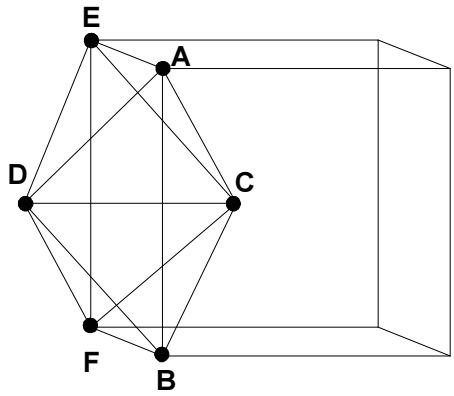


Fig.10

10. Implementation first algorithm

The implementation presented here is the decomposition of the cube in 6 tetrahedra.

The decomposition used was illustrated is the first in Fig.8 but for simplicity it's redrawn in Fig.11

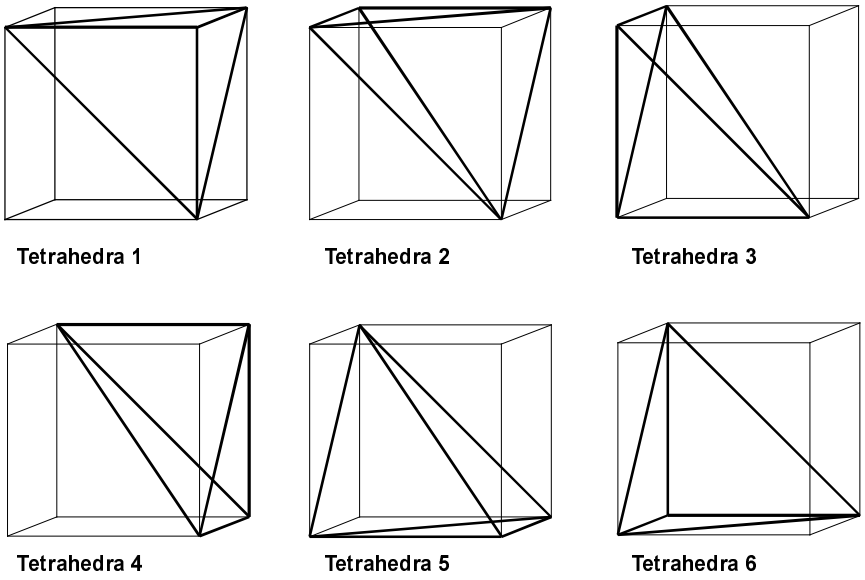


Fig.11

All the explanations of this implementation will use the labels for the vertex of the cube shown in Fig.12.

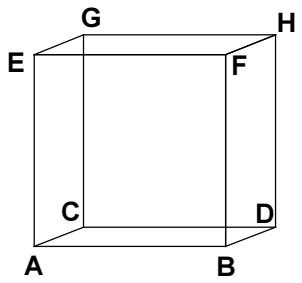


Fig.12

The algorithm of the principal code is:

```
For each cube do
   If (one vertex in the cube has a different value) then
      Calculate values
      If (one vertex of BEFH has a different value) then process Tetrahedra1;
      If (one vertex of BEGH has a different value) then process Tetrahedra2;
      If (one vertex of ABEG has a different value) then process Tetrahedra3;
      If (one vertex of BDGH has a different value) then process Tetrahedra4;
      If (one vertex of ABDG has a different value) then process Tetrahedra5;
      If (one vertex of ACDG has a different value) then process Tetrahedra6;
   End If
End Do
```

principal code

Using the decomposition of the cube into six tetrahedra, there are 16 combinations of the on-off vertex's state (one vertex has on state when the volume associated with the vertex is above the threshold and off state when it's below and there are 2^4 combinations).
This vertex's state can be classified in the next cases:

1. One case with all four vertex in off.
2. Four cases with only one vertex in on.
3. Six cases with two vertex in on.
4. Four vertex with three vertex in on.
5. One case with all four vertex in on.

This 16 cases can be reduced to 8 if interchanges of the on-off state are taken into account; this eight patterns can be grouped in the following way:

A. One pattern represent first and last cases.
B. Four patterns represents the second and fourth cases.
C. Three patterns represents the six cases where two vertex are on and anyone is symmetric of the other two.

This eight patterns are illustrated in Fig.13.

In the A group, where all four vertex are on or off, the isosurface doesn't pass throw any edge of the tetrahedra, so there isn't drawn any triangle inside the tetrahedra, Fig.13 (A)

In the B group, where one vertex is different with the other, the isosurface pass throw the tetrahedra and one vertex is outside/inside the isosurface. So there is drawn one triangle inside the tetrahedra, Fig.13 (B,C,D,E).

Triangle's set up differs between the case where one vertex is in on state and where one vertex is in the off state, in the orientation of the normal vector of the triangle. When there is only one vertex in off state the normal vector directs forward the off vertex and when there are three vertex in the off sate the normal are contrary to the on vertex direction.

Fig 14 shows the two cases of Fig 13 (B).

In the C group, there is drawn two triangles inside the tetrahedra. Triangles are drawn again with the normal direct to the off vertex, Fig.13 (F,G,H)

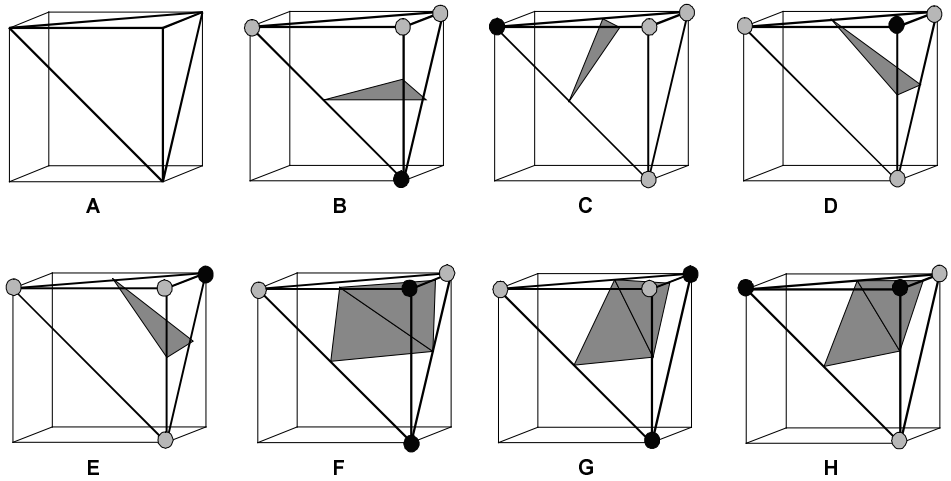


Fig 13.

Eight patterns of the on-off state of the first tetrahedra. Triangles of each state are drawn inside.
For generality in the figure, direction of normal vector is not drawn, so triangle's configuration is valid if on state is interchanged with the off state. Normal's direction can be drawn going from the triangle to the off vertex.

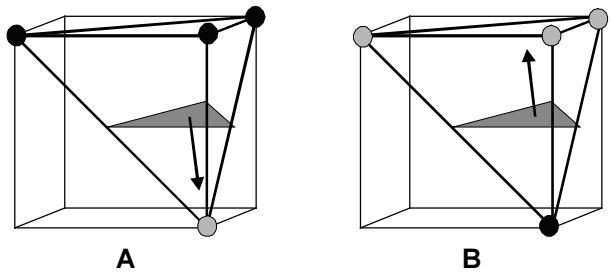


Fig 14.

Direction of the normal vector varies between the symmetric on-off patterns. Black circumferences of this figure represent the on state.

Next code shows the implementation of the method Tetrahedra1, referenced in the *principal code*.
This method differentiates 14 possible cases; the two cases where all vertices have the same state aren't handled because as it was pointed before it isn't drawn any triangle.

There is a differentiation with one vertex's state combination and the symmetric (the interchange of on-off values), in the construction of the triangles because of normal's direction.
Normal vector's direction in Open GL can be defined by the order of the vertex definition, so normal vector direction of a polygon with a vertex's definition in counterclockwise is the opposite of the vertex defined in clockwise.

```
/* Solution for fundamental cases */

If B is different to the other vertex then
   If B is below the threshold then
      Draw Triangle BE,BF,BH
   Else
      Draw_Triangle BE,BH,BF

If E is different to the other vertex then
   If E is below the threshold then
      Draw_Triangle BE,EF,EH
   Else
      Draw_Triangle BE,EH,EF

If F is different to the other vertex then
   If F is below the threshold then
      Draw_Triangle BF,FH,EF
   Else
      Draw_Triangle BF,EF,FH

If H is different to the other vertex then
   If H is below the threshold then
      Draw_Triangle BH,EH,FH
   Else
      Draw_Triangle BH,FH,EH

If B and E are different to the other vertex then
   If B and E are below the threshold then
      Draw_Triangle EH,BH,BF
      Draw_Triangle EH,BF,EF
   Else
      Draw Triangle EH,BF,BH
      Draw Triangle EH,EF,BF

If B and H are different to the other vertex then
   If B and H are below the threshold then
      Draw Triangle BE,EH,FH
      Draw Triangle BE,FH,BF
   Else
      Draw_Triangle BE,FH,EH
      Draw_Triangle BE,BF,FH

If E and H are different to the other vertex then
   If E and H are below the threshold then
      Draw Triangle BE,EF,BH
      Draw_Triangle BH,EF,FH
   Else
      Draw_Triangle BE,BH,EF
      Draw_Triangle BH,FH,EF
```

Method Tetrahedra1

Normal vectors and coordinates of each vertex in the same cube, are calculate once and are used for all the tetrahedras because all tetrahedras of the same cube have the same values.

If two consecutives cubes are handled, the second one can use part of the values of the previous cube.
For example if vertex coordinates and normals computation of the left cube in Fig.14 are calculated, cube in the right have some common values *(A'=B, C'=D, E'=F, G'=H)* and it isn't necessary to compute them again.

Execution time is improved with this optimisation, it's important to remark that execution time is not reduced to the half because this computations are made only when a cube is taking into account (one or more vertex have a different state).

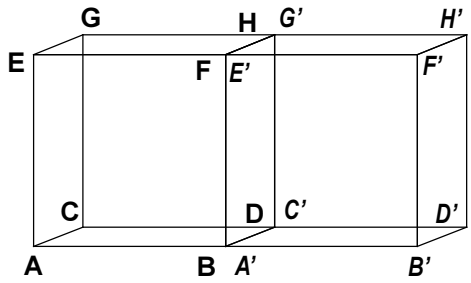


Fig.15

Two consecutive cubes with their labels.

Parameters of the method Draw_Triangle are the three vertexes of the triangle. This method uses a method called Draw_Point and draws a triangle.

The code of the method Draw_Triangle, is:

```
Draw_Triangle ( line1, line2, line3)
    Draw_Point(line1)
    Draw_Point(line2)
    Draw_Point(line3)
```

The Draw_Point method calculates the position of each vertex and normals values.

Position vertex are calculated using interpolation and normal values are approximated with the gradients, using the central difference method.

The code of the method Draw_Point is:

```
Draw Point ( line )
  proportion = (Threshold - origin value) / (end value - origin value)
  normalx = origin_normalx + (end_normalx - origin_normalx) * proportion
  normaly = origin_normaly + (end_normaly - origin_normaly) * proportion
  normalz = origin_normalz + (end_normalz - origin_normalz) * proportion
  vertexx = vertex originx + (end vertexx - vertex originx) * proportion
  vertexy = vertex originy + (end vertexy - vertex originy) * proportion
  vertexz = vertex_originz + (end_vertexz - vertex_originz) * proportion
```

Threshold is the value defined before starting the isosurface extraction.
origin_value is the value of the first vertex of the line passed as a parameter.
end_value is the value of the second vertex of the line passed as a parameter.
origin_normalx, origin_normaly, origin_normalz are the gradient value calculated using the central difference method in the first vertex position of the line in the x, y and z directions respectively.
end_normalx, end_normaly, end_normalz are the gradient value calculated using the central difference method in the second vertex position of the line in the x, y and z directions respectively.

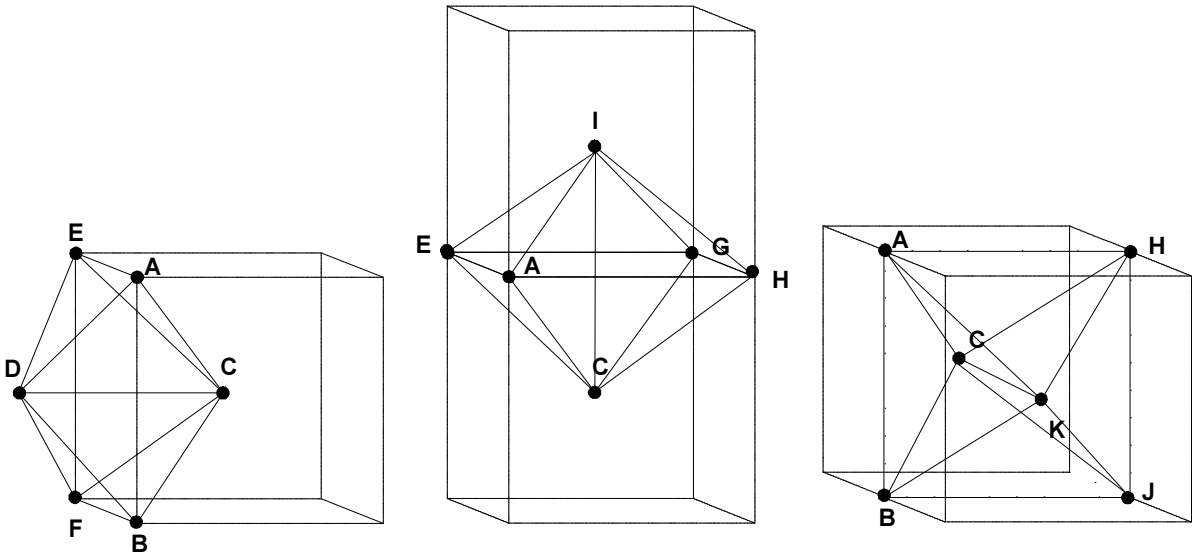
After Draw_Point method operations are performed, there have been the following assignations:

normalx, normaly, normalz store the interpolation of the gradients values and so the gradient correspond to the position (vertex, vertexy, vertexz).
vertex, vertexy, vertexz store the position in the line where the isosurface pass through.

11. Implementation second algorithm

The second implementation is the tetrahedral tessellation of space from the centred cubic lattice, presented in section 9(C).

In Fig.16 is illustrated all tetrahedra involved in this method.



A B C

Fig 16

There are two types of edges in this tessellation of the tetrahedra. The short edges join a lattice point with one of its eight nearest neighbours. Examples are AC, AD, BC and BD in Fig.16.
The long edges of a lattice point with one of its six second-nearest neighbours. Examples are AB and CD in Fig.16.

There are two long edges and four short edges for each tetrahedron. There are four tetrahedra surrounding each long edge and six tetrahedra surrounding each short edge.

There is only one tetrahedral shape involved in this decomposition and symmetric overall.

Each cube participate in 24 different tetrahedras and each tetrahedra is constructed using two cubes, so it's like to say this is a decomposition in 12 tetrahedras.

Figure 17 illustrates all the12 tetrahedras handled when one cube is processed.

The cube in the right hand of this one will full fill the right space of the cube actually processed, the back cube will full fill the back space and finally the cube in the next slice above the actual cube will full fill the space in the top of the actual cube.

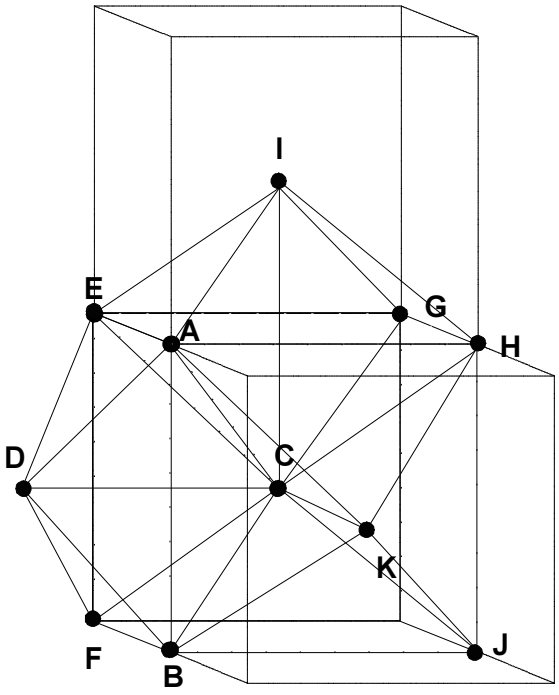


Fig 17

Values in the centre of the cube (points C, D and I) are calculated by averaging the eight nearest values.

Gradients aren't obtained in a identical way as in the previous implementation; because the points at *(i+1, j, k)* etc. are not the ones nearest to the lattice point at *(i,j,k)*. Instead, there are eight nearest neighbouring points located at *(i+1/2, j+1/2, k+1/2)* etc. Hence we take into account the functional values at these positions, too [2]. The gradient in the point *(i,j,k)* is set:

$$
\begin{aligned}
G_x = c\left(\frac{g(i+1, j, k) - g(i-1, j, k)}{2a}\right) \\
+ \frac{1-c}{4}\frac{g(i+\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2}) - g(i-\frac{1}{2}, j+\frac{1}{2}, k+\frac{1}{2})}{a} \\
+ \frac{1-c}{4}\frac{g(i+\frac{1}{2}, j+\frac{1}{2}, k-\frac{1}{2}) - g(i-\frac{1}{2}, j+\frac{1}{2}, k-\frac{1}{2})}{a} \\
+ \frac{1-c}{4}\frac{g(i+\frac{1}{2}, j-\frac{1}{2}, k-\frac{1}{2}) - g(i-\frac{1}{2}, j-\frac{1}{2}, k-\frac{1}{2})}{a} \\
+ \frac{1-c}{4}\frac{g(i+\frac{1}{2}, j-\frac{1}{2}, k+\frac{1}{2}) - g(i-\frac{1}{2}, j-\frac{1}{2}, k+\frac{1}{2})}{a}
\end{aligned}
$$

where $c \in \langle 0,1 \rangle$ is a weight parameter between zero and one and a is the distance in the x, y and z directions. Different values for c were tried, see [2] but the conclusion was that it had a little effect on the appearance of the isosurface.

Similar to the previous method, the intersections of the contour surface on the edges of the tetrahedra are interpolated to improve the quality of the surface.

12. Comparison of two methods

In the following sections comparison of the two previous methods is presented.

Decomposition in 6 tetrahedras algorithm will be referenced as First algorithm and tetrahedral tessellation of space from the centred cubic lattice will be referenced as Second algorithm.

The A section compare the time used to create the isosurface from the volumetric data.

Next section compare the number of triangles used for representing the isosurface.

In C, D and E sections differences between a perfect sphere and a sphere constructed using each method are measured.
Various comparison are made: radius from centre of the sphere to each vertex, radius from centre of the sphere to the centre of gravity, area of the triangles used for the creation of the isosurface and finally volume defined by the isosurface.

The last section will show images from simple and complex volumetric data and visual impression will be used to extract some new conclusions.

A. Construction time

One important aspect when comparing two methods is execution time.

Results in Table 2 are from program's execution with only the improvements explained in the 10 section were one cube uses the common vertex values of the previous processed cube.

Image		Time (seconds)	
Name	Size	First Algorithm	Second Algorithm
Syn_64	64x64x64	2	4
Engine	256x256x110	53	159
3dhead	256x256x109	104	215
51Sphere	51x100x100x100	124	432

Table 2

Isosurface extraction time from volumetric data

From this results it can be observed:

A) First algorithm always spend less time than the Second

This difference can be because of various reasons:

- Second algorithm must calculate the central cube positions and gradients plus the same calculations performed by the first algorithm when one cube is being processed.
- Calculation of the gradients are more complex in the second algorithm (it's performed the same operation like in the first method plus the central difference method in four more positions).
- Second algorithm splits the cube in more tetrahedras so there are involved more calculus.
- In the Second algorithm, during the evaluation process where the method decide weather the cube is going to be processed, three central cube values are calculated by averaging eight cube vertexes and this operation is performed for each cube of the model.

B) Time ratio of two methods isn't constant (sometimes it's two times bigger, sometimes three, etc).

This variation of the ratio can be because there is a difference with the threshold election; some tetrahedras would be taken into account or leave to be taken into account and this difference isn't changing in the same magnitude in both methods because the second one use the interpolated points and this change the comportment.

It's possible to decrease the execution time using the improvement explained in the 10 section for the values calculated from different columns and different slices (actual implementation only use the calculated values of the previous processed cube and it's only in the row direction).
With this modification the second method would benefit more than the first because operations have a higher cost.

It's important to know how much time is spend by each method without handling any tetrahedra. (constant time for explore all cubes without counting the time when the cube is handled).
The way to overtake this value was using one volumetric data (3dhead with a size of 256x256x109) and execute the isosurface process extraction with a threshold bigger than any vertex value and so any tetrahedra was handled.
The results of this execution were: the first algorithm spend 11 seconds to cross all the data volume looking for which cubes contribute in the isosurface construction and 48 seconds for the Second algorithm.

With this results it's clear there is a different constant time associated to each method.

As it was pointed before both methods makes a similar first cube evaluation; both methods find values of eight positions but the second method process a second group of operations (average the 8 values for three different places).
So the second group of operations spend about 37 seconds.

Using the improvement of the 10 section in columns and slices this values are calculated once and the second method would reduce from 37 seconds to 12.

Execution time of both implementations are bigger than other experimental implementations. The reason is because both programs are implemented using Visual C++ and there are two classes, one has all the volumetric data and the second one implements both methods. This second class access to volumetric data with the public methods of the first class.
This class structure isn't the most appropriate if the most important is execution time but it's more elegant than accessing directly to a data vector.
The finality of this implementation wasn't the execution speed and it was important to have a good structure and if both methods have the same accessing data delay it doesn't change substantially the comparison results.

The work has not been directed to the fastest visualization but to the understanding what error behaviour can be expected from two fundamental approaches of isosurface generation.

B. Number of triangles

In Chart 1 and Table 3, it clear that the second algorithm always generates more triangles than the first algorithm.

The ratio of triangle generation is between 1.5 and 3.
This bigger triangle's generation amount involve more size of memory -if triangles are keep in a list, and more execution time.

This result must be one of the reasons of the difference in generation time discussed in the previous section.
The cube in the second method participate in the double amount of tetrahedras than the first method and it involves in general a bigger triangle amount.

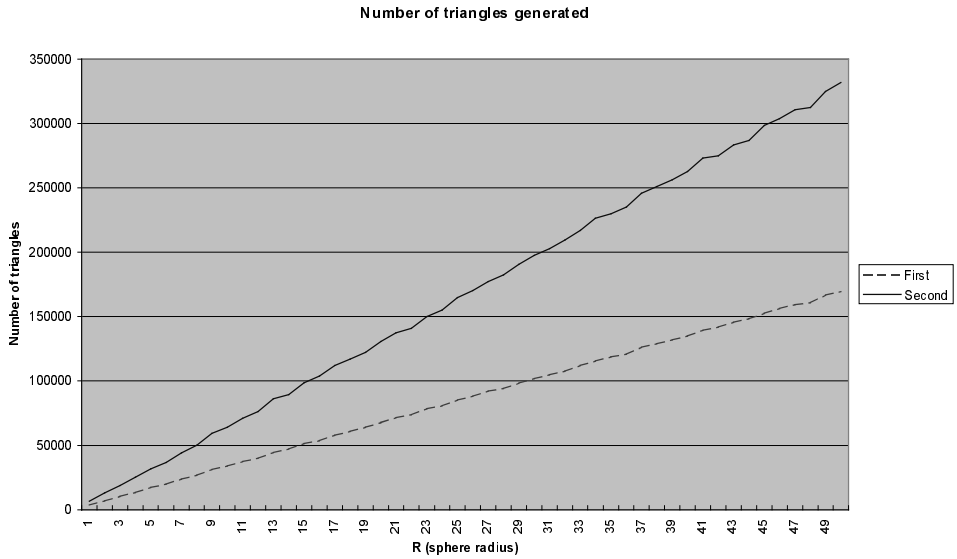


Chart 1

Number of triangles when construction of 51 spheres with a radius varying from 1 to 501 in steps of 10 is performed (henceforth 51Sphere).

Image	Number of triangles	
	First Algorithm	*Second Algorithm*
Syn_64	171276	269184
Engine	5744340	8333868
3dHead	12713532	17652630

Table 3

Number of triangles used for isosurface construction of several images

C. Radius error from each vertex

In this section there is presented the error committed when approximating a sphere, with a measure of quality the distance from the centre of the sphere to the vertexes of each triangle.

This error is calculated with the following formula:

$$\sum_{i=1}^{n} \frac{R_i - R_r}{R_r}$$

Where, n is the number of triangles multiplied by 3, R_r is the radius of the sphere and R_r is the distance from the centre of the sphere (c_x,c_y,c_z) to a vertex of a triangle (v_x,v_y,v_z):

$$r_i = \sqrt{(v_x - c_x)^2 + (v_y - c_y)^2 + (v_z - c_z)^2}$$

With Chart 2 it can be observed that radius error in the second algorithm is bigger than in the first algorithm.

In Chart 3 there are presented the radius error divided by the number of triangles generated. The first method has associated a less radius error. It means that the first algorithm is a better approximation if radius error is taken into account.

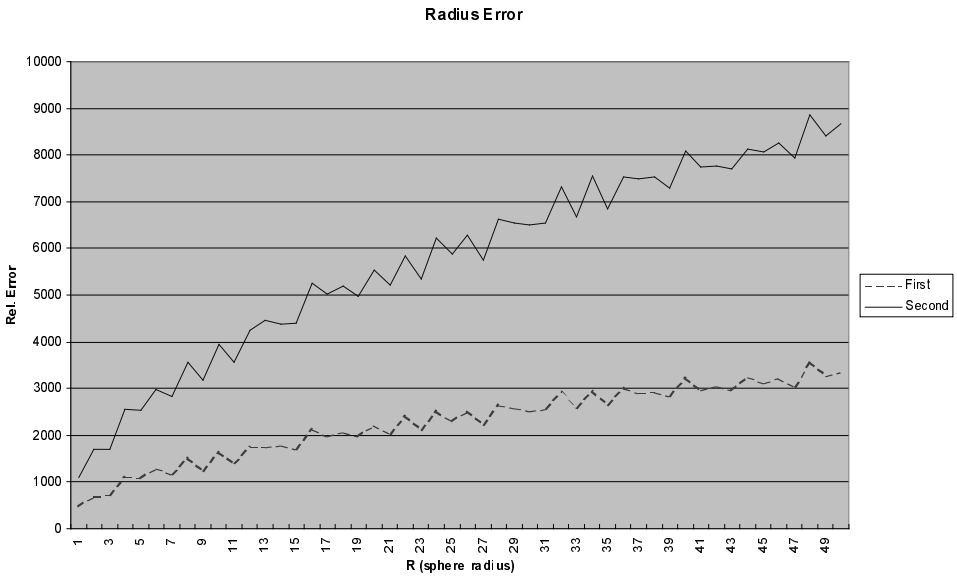


Chart 2

Radius Error when construction of the 51Sphere is performed.

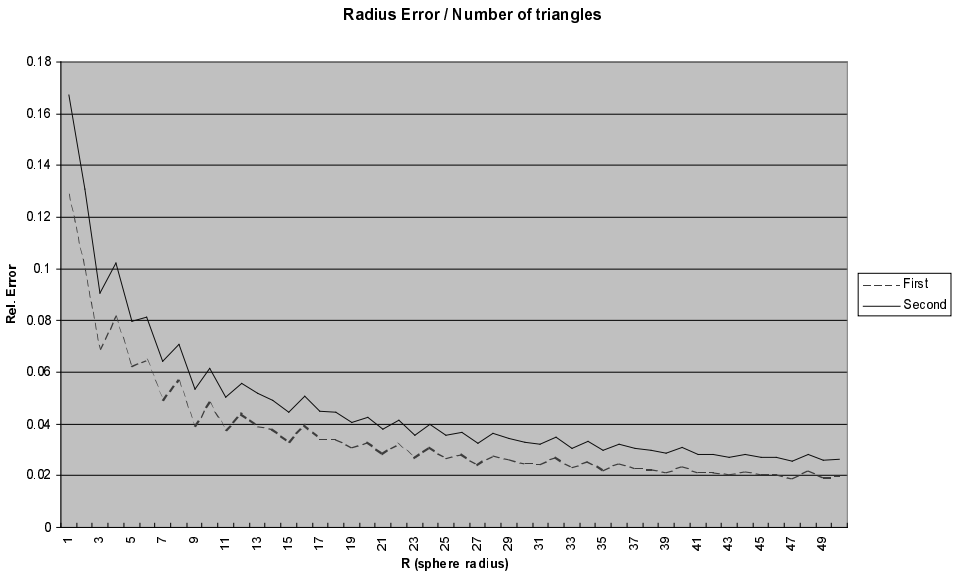


Chart 3

Radius Error divided by the number of triangles used when construction of the 51Sphere is performed.

D. Radius error from centre of gravity

In this section R_i is calculated from the centre of the sphere to the centre of gravity of each triangle.

Results from this approach are similar as the Radius error from each vertex.

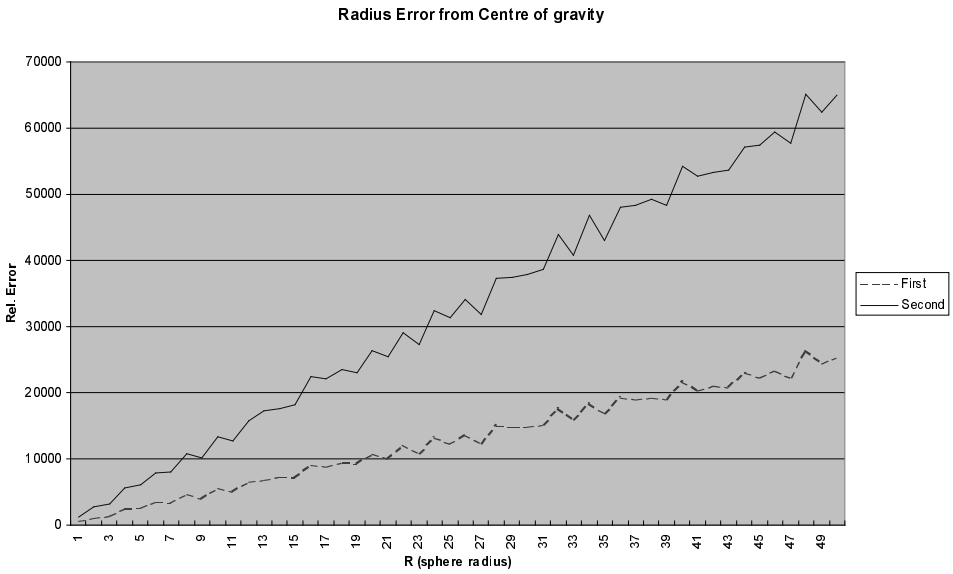


Chart 4

Radius Error from centre of gravity obtained when construction of the 51Sphere is performed.

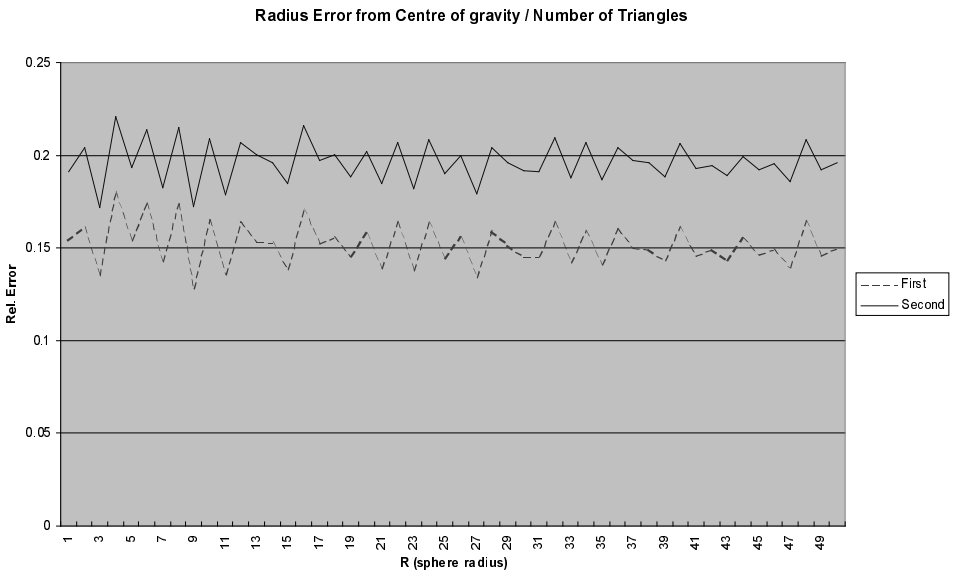


Chart 5

Radius Error divided by the number of triangles used when construction of the 51Sphere is performed.

E. Area error

In this section there is presented comparison of triangle's area when creating the isosurface with the theoretical value.

The magnitude of the cross product of two vectors is the area of the parallelogram they determine: If A and B are vectors, the $|A \times B|$ is the area of the parallelogram with sides A and B. Since any triangle can be viewed as half of a parallelogram, this gives an immediate method of computing the area from coordinates. Just let A=b-a and B=c-a. then the area is half the area of AxB. The cross product can be computed from the following determinant, where i, j and k are unit vectors in the x, y, z directions respectively [12]:

$$\begin{vmatrix} i & j & k \\ A_0 & A_1 & A_2 \\ B_0 & B_1 & B_2 \end{vmatrix} = (A_1B_2 - A_2B_1)i + (A_2B_0 - A_0B_2)j + (A_0B_1 - A_1B_0)k$$

The theoretical area is calculated by $S_r = 4\pi r^2$ and Relative Error by $E = \frac{S_d - S_r}{S_r}$

It can be concluded with results shown in Chart 6 that first method is a better approximation of area.

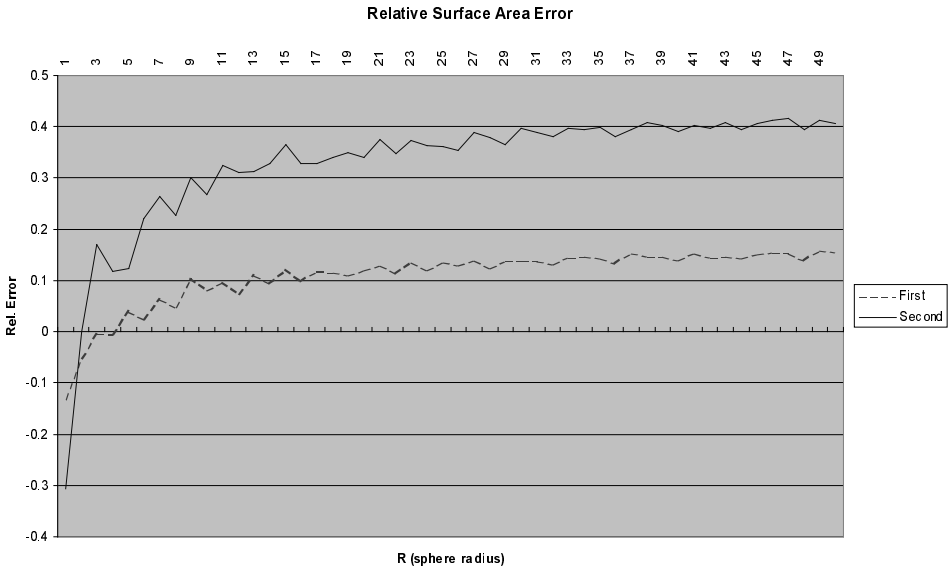


Chart 6

Relative surface area error of each sphere when construction of the 51Sphere is performed.

F. Volume error

In this section there is presented conclusions about the volume limited by the isosurface.

In the 10 section, all different combinations of the on-off tetrahedral vertex state were grouped in three patterns. This three patterns were differentiated by the direction of normal's vector.

When calculating the surface's volume, defined by the triangles drawn inside the tetrahedra, this patterns can be used.

Volume calculation (V_d) have been made in the following way:

A Group. All vertex of the tetrahedra have the same state:

- All vertex have the on state (all tetrahedra is inside the surface). The volume is Vol_t were Vol_t is the volume of the tetrahedra. Fig 18 (A)

- All vertex have the on state off (all tetrahedra is outside the surface). The volume is 0. Fig 18 (B)

B Group. One vertex v_t of the tetrahedra has a different state:

- If v_t has the on state (this vertex is inside of the surface and the other three are outside). The volume is $Vol_{a,b,c,t}$ were $Vol_{a,b,c,t}$ is the volume of the tetrahedra defined with the vertex v_t and the three points of the triangle drawn. Fig 18 (C)

- If v_t has the off state. The volume is $Vol_t - Vol_{a,b,c,t}$. Fig 18 (D)

C Group. Two vertex v_{t1} and v_{t2} of the tetrahedra have a different state:

- If v_{t1} and v_{t2} have the on state. The volume is $Vol_{a,b,c,t1} + Vol_{a,b,d,t2} + Vol_{a,b,t1,t2}$ Fig 18 (E) In Fig.19 this three tetrahedras are drawn for clarification.

- If v_{t3} and v_{t4} have the off state. The volume is $Vol_t - (Vol_{a,b,c,t1} + Vol_{a,b,d,t2} + Vol_{a,b,t1,t2})$ Fig 18 (F)

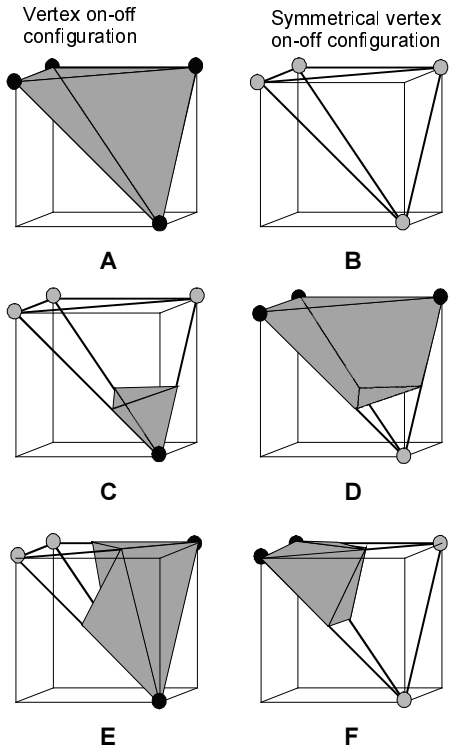


Fig 18

Example of volumes calculated for each configuration.

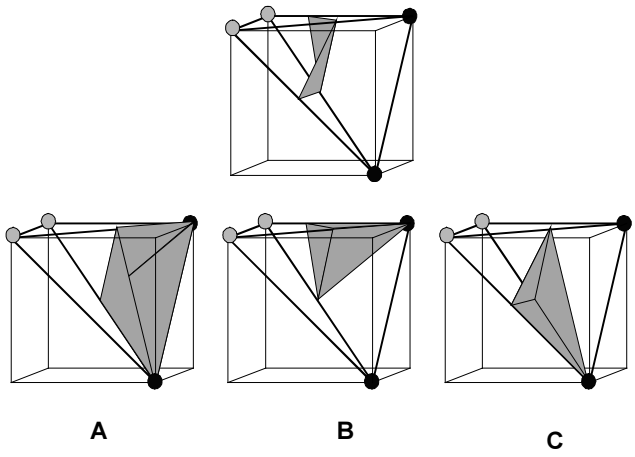


Fig 19

Black circumferences represent the on state.
The above cube shows the two triangles drawn inside the second tetrahedra.
The A, B and C cubes shows the three cubes used to calculate the volume.

The theoretical volume is calculated as:

$$V_r = \frac{4}{3}\pi r^3$$

The relative error is:

$$E_r = \frac{(V_d - V_r)}{V_r}$$

From Chart 7 it can be observed, the first algorithm is a better approximation of the volume than the second.

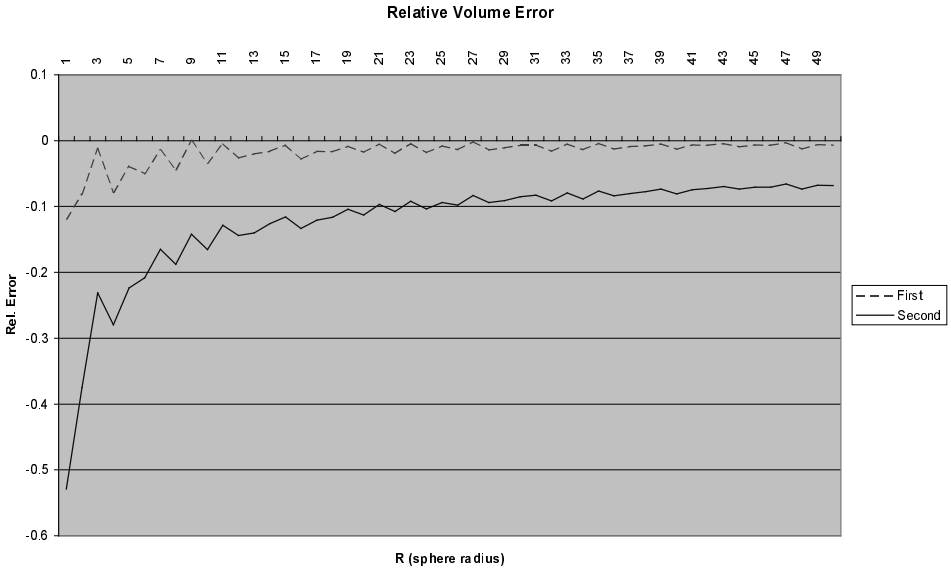


Chart 7

Relative volume error of each sphere when construction of the the 51Sphere is performed

G. Image Comparison

When a little cube is approximated, the first algorithm has a better result, see Fig.1 in appendix A

When a medium cube is drawn it can be observed that in the centre of each face the cube is drawn with the same appearance but in the each edge the second algorithm is worst, see Fig.2.
The reason of this result is because positions outside the cube have a zero value and positions inside have the same value as the threshold and when the averaging values of a tetrahedra with one vertex inside the object and the other outside of a long edge, there is drawn triangle going to the vertex that's inside the surface, see Fig.5 and 6.

But when drawing a big cube the results of both methods are identical, see Fig.3.

If a cube is constructed with a different threshold than the exact value, the shape of the object varies a lot and the second has an image with a similar quality of the first, see Fig.4.

Fig.7 illustrates two spheres and the sphere made by the second algorithm has a better shadows.

The second method is better than the first one when a complex model is generated.

Fig.8 and 9 shows that images generated by the second algorithm are better: the ear and nose are more defined, the contour of the eye is better, the back of the head has more realistic relief, the mouth is better and the lips have a better shape.

Fig.10 and 11 shows that there aren't big differences but the image generated by the second algorithm has a better shadows.

Finally in Image 12, the image generated by the second algorithm has a better contour of the line; it's more straight and there is a better impression of the frontier of two edges.
In the second image there is a better impression in the curb zone and it's more lineal; in the image generated by the first algorithm it's more faceted.

13. Conclusions

A comparison of the Marching Tetrahedra algorithm with a decomposition in six tetrahedra and Marching Tetrahedra with a tetrahedral tessellation of the space from the cubic lattice has been presented.

When the measure of quality is generation time or number of triangles, the decomposition in 6 tetrahedras has a better results.

Drawing spheres with a different radius, distance error from each vertex of the triangles, distance error from the centre of gravity, area and volume error were studied and the experimental results proved that it is useless to implement the tessellation of the cube using the centre cubic lattice with this criteria.

Nevertheless, the visual impression is better when a complex model is generated with the second the tetrahedral tessellation of the space from the cubic lattice.

This results are similar as the comparison made in [8] where five and six tetrahedra decomposition were compared with 24 and 48.

A similar explanation can be made about the better image results with the second decomposition: human eye is sensitive to edges and gradient changes of a surface.

14. Bibliography

[1] Chong-wei Xu, “Surface-fitting Algorithms”
http://www.cs.gasou.edu/~xuc/hypervis/vistech/volume/surface.htm

[2] S. L. Chan and E. O. Purisima, 1998 “A new Tetrahedral Tessellation scheme for Isosurface Generation”, Computer and Graphics, Vol. 22, No 1, app. 83-90.

[3] Andreas Pommert, Michael Bomans, Karl Heinz Hohne, 1992 “Volume Visualization in Magnetic Resonance Angiography“, The Institute of Electrical and Electronics Engineers, Inc.

[4] T. Todd Elvins, April 1992 “A Survey of Algorithms for Volume Visualization“, San Diego Supercomputer Center, Advanced Scientific Visualization Laboratory.

[5] Jung-Hong Chuang and Woan-Chiaun Lee, 1995 “Efficient Generation of Isosurfaces in Volume Rendering”, Computer & Graphics, Vol. 19, No 6, pp 805-813.

[6] M. Bosma, J. Smith and J. Terwisscha van Scheltinga, 1995 “Super Resolution Volume Rendering Hardware“, Proc. 10th Workshop Graphics Hardware, Vol EG95 HW, Eurographics Technical Report Series.

[7] Takanori Nagae, Takeshi Agui, and Hiroshi Nagahashi, 1995 “Comparisons of triangulated isosurfaces constructed from volume data”, The Fourth International Conference on CAD/CG, 23-25 October 1995, Huazhong University of Science and Technology, Wuhan China
http://www.shobi-u.ac.jp/~tnagae/pub/wuhan95/

[8] Pavel BAXA, Václav SKALA, Roman MOUČEK, “Error Estimation for Isosurfaces“, In Proceedings of COMPUGRAPHICS '97, 1997.

[9] Mark J. Bentum, Barthold B. A. Lichtenbelt, Tom Malzbender, 1996 “Frequency Analysis of Gradient Estimators in Volume Rendering“, IEE Transactions on visualization and Computer Graphics, Vol. 2, N. 3, September 1996

[10] Jonathan C. Carr, W. Richard Fright, Richard K. Beatson, 1997 “Surface interpolation with radial basis function for medical imaging“, IEE Transactions on medical imaging.

[11] Kenneth I Joy, “Gradients and Normals”, Visualization and Graphics Research Laboratory, Department of Computer Science, University of California, http://graphics.cs.ucdavis.edu/

[12] Joseph O’Rourke, 1998 “Computational Geometry”, Second Edition, Cambridge University Press.

[13] Jonathan C. Carr, Andrew H. Gee, Richard W. Prager, Kevin J. Dalton, “Quantitative visualization of surfaces from volumetric data“.

[14] Allen Kent, James G. Williams, 1993 ”Encyclopedia of computer science and technology”, Marcel Dekker, Inc. Volume 28, supplement 13.

[15] Russian Problems, Solution 53, http://www.kalva.demon.co.uk/rsoln/rsoln53.html

[16] OpenGL Programming Guide.

Appendix A. Images

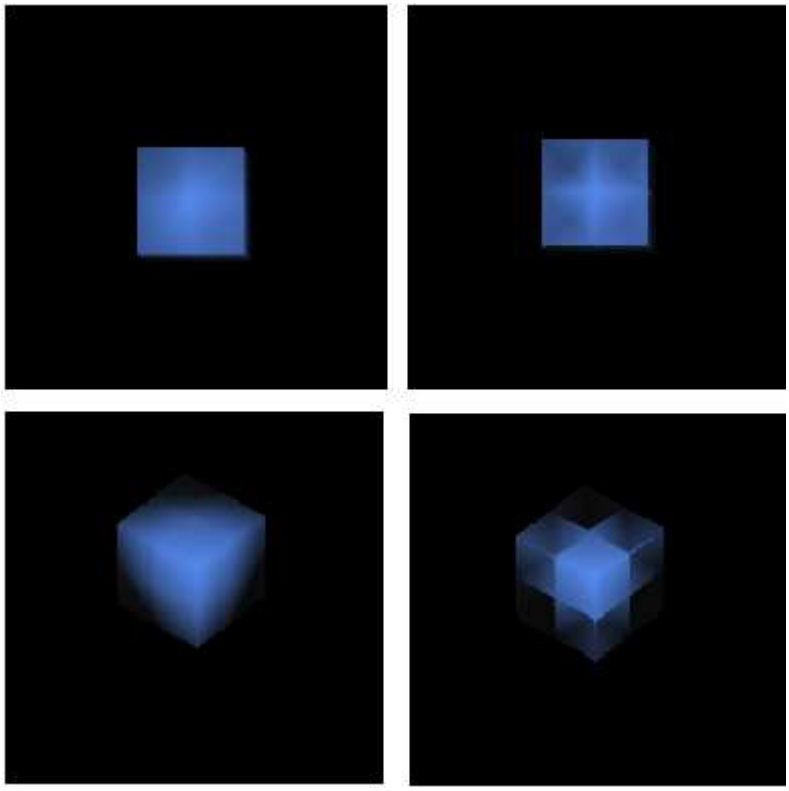


Figure 1

Approximation of a cube with a edge size 2. Vertex of the cube have 100 value and the others have 0.
Threshold is 100
Left images are obtained from the First algorithm and right images from the Second Algorithm.

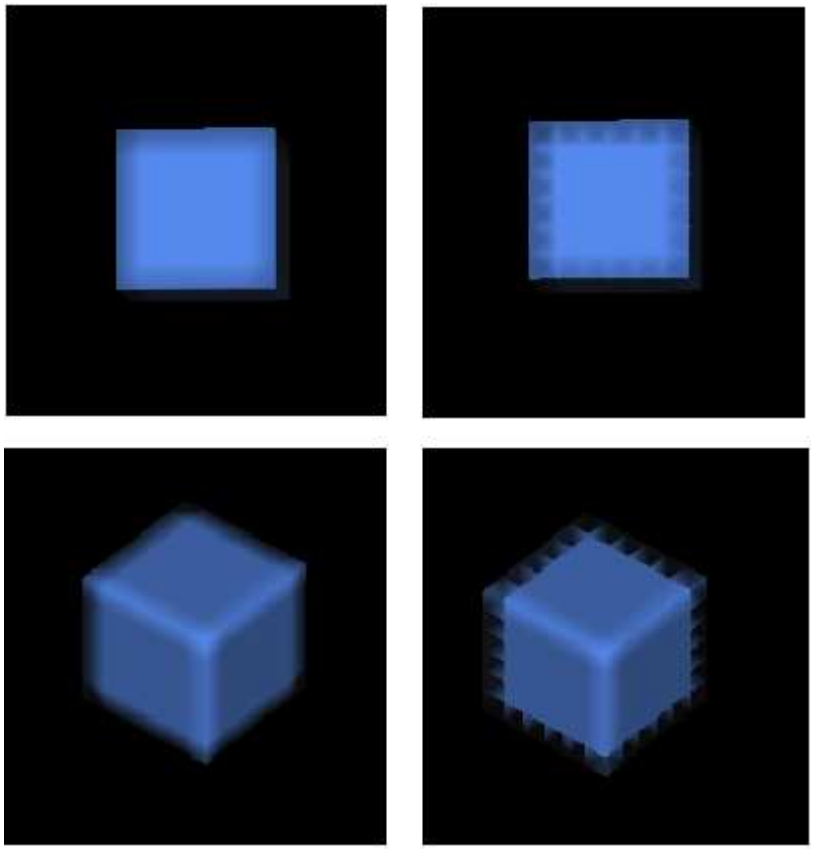


Figure 2

Approximation of a cube with a edge size 6. Vertex of the cube have 100 value and the others have 0.
Threshold is 100
Left images are obtained from the First algorithm and right images from the Second Algorithm.

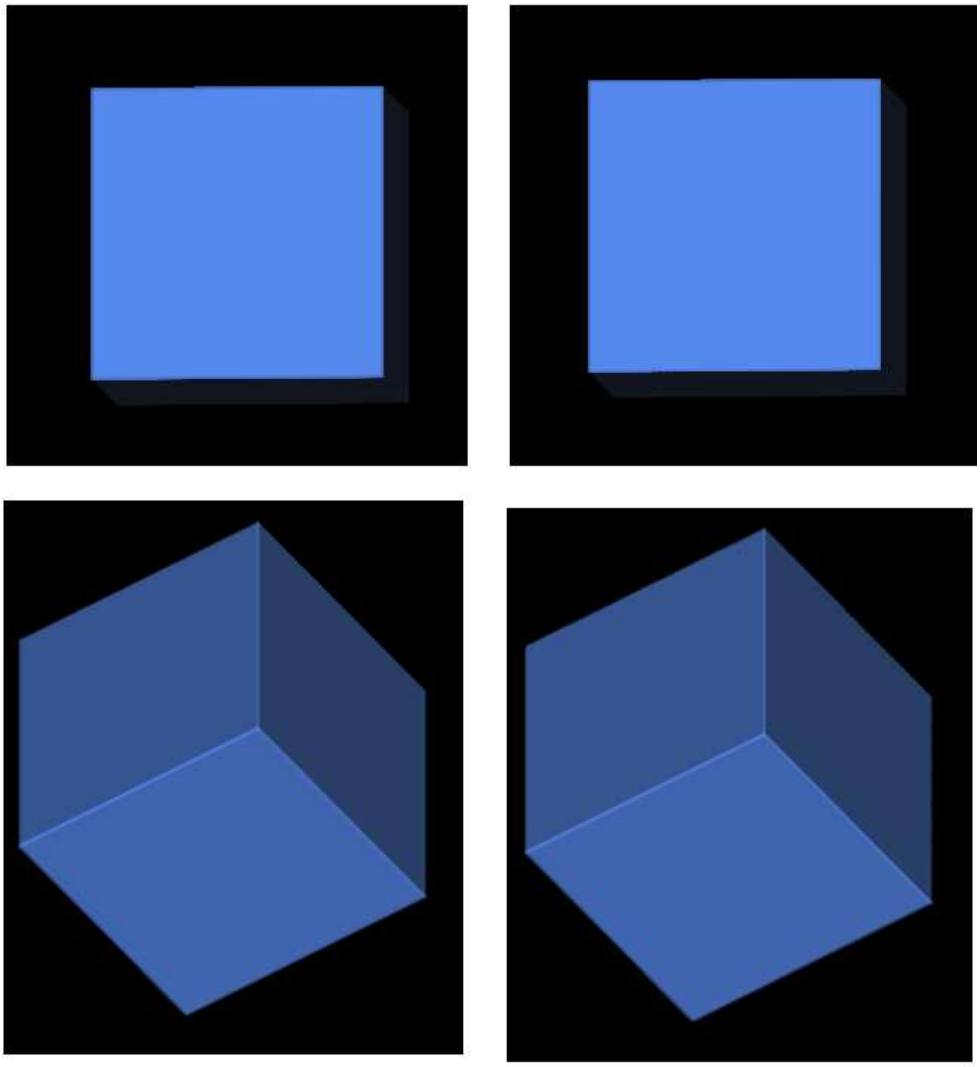


Figure 3

Approximation of a cube with a edge size 80. Vertex of the cube have 100 value and the others have 0.
Threshold is 100
Left images are obtained from the First algorithm and right images from the Second Algorithm.

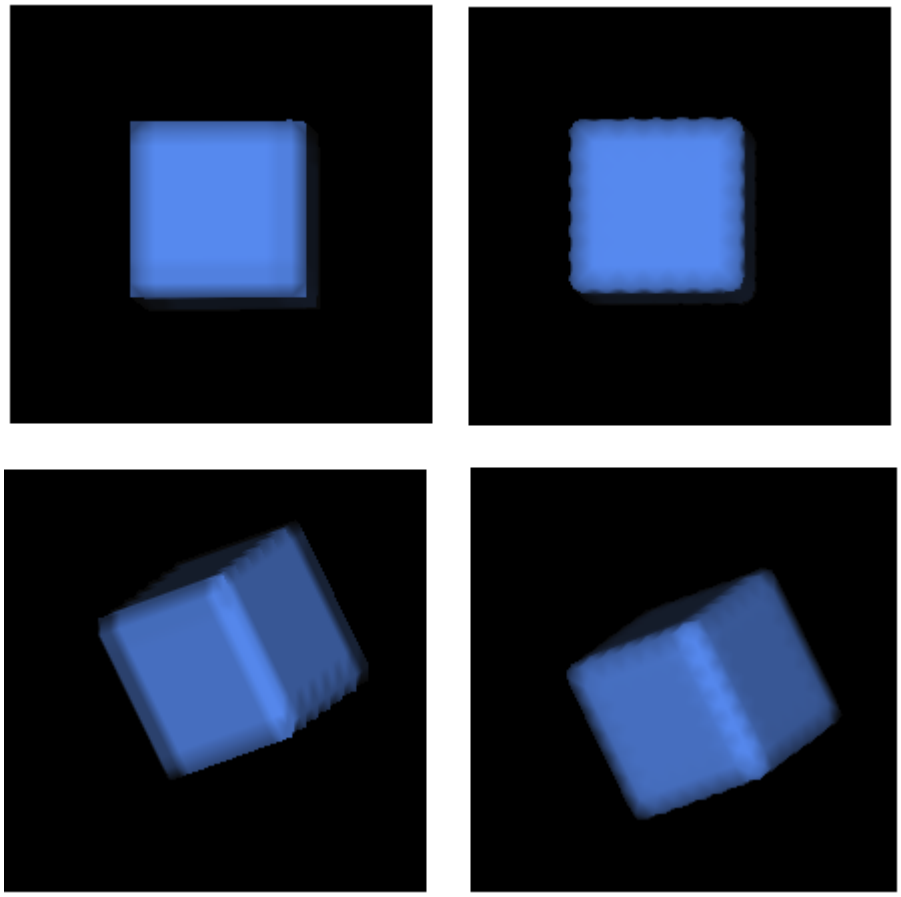


Figure4

Approximation of a cube with a edge size 6. Vertex of the cube have 100 value and the others have 0.
Threshold is 50
Left images are obtained from the First algorithm and right images from the Second Algorithm.

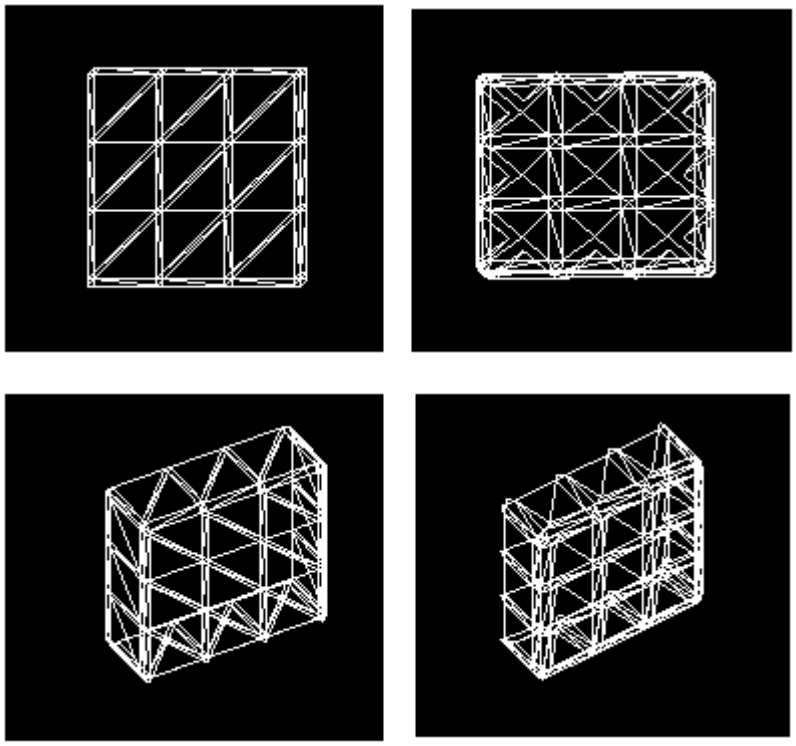


Figure 5

Approximation of a cube with a edge size 3. Vertex of the cube have 100 value and the others have 0.
Threshold is 100
Left images are obtained from the First algorithm and right images from the Second Algorithm.

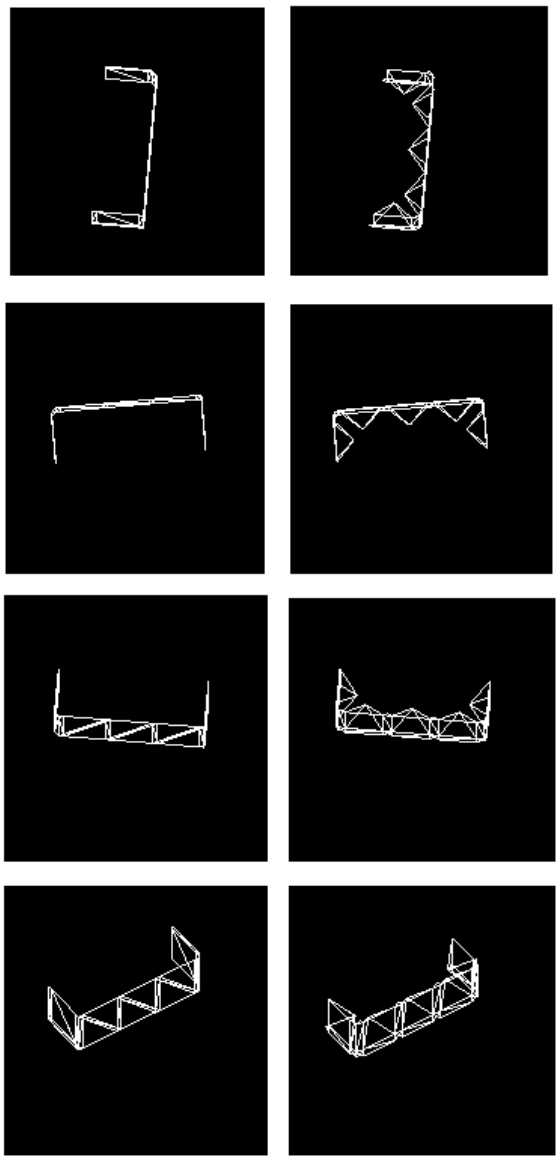


Figure 6

Details of Image 4
Left images are obtained from the First algorithm
and right images from the Second Algorithm.

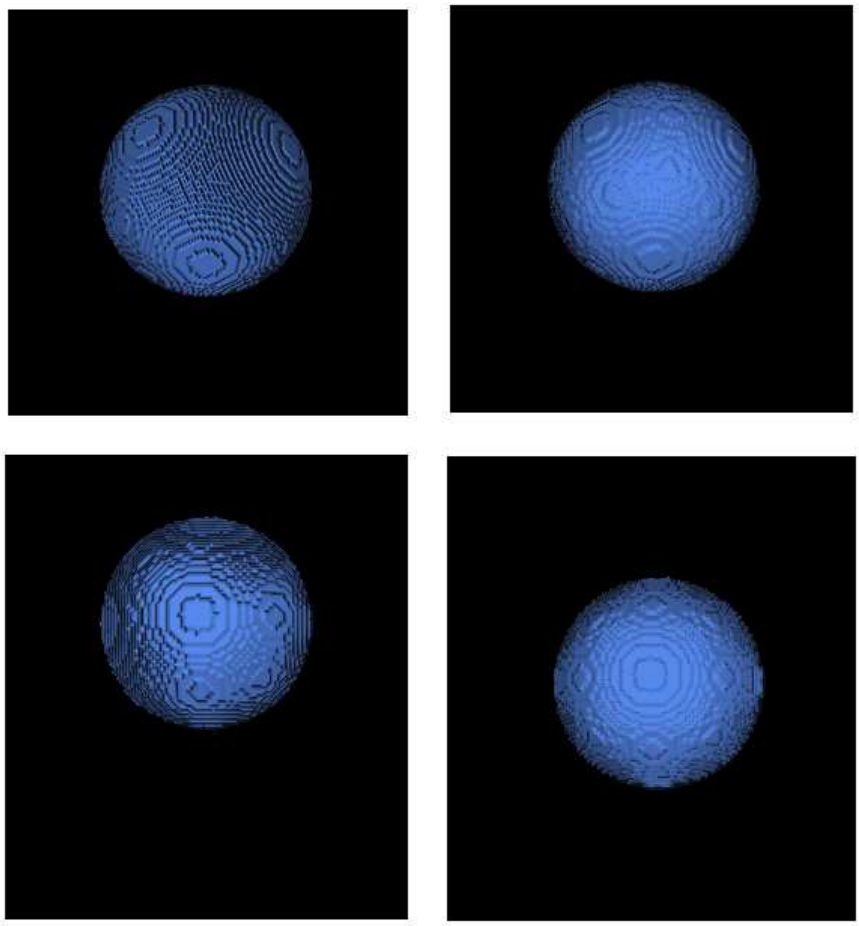


Figure 7

Circumference with light. Vertexes of the sphere have 100 value and the others have 0.
Sphere radius is $\sqrt{810}$
Left images are obtained from the First algorithm

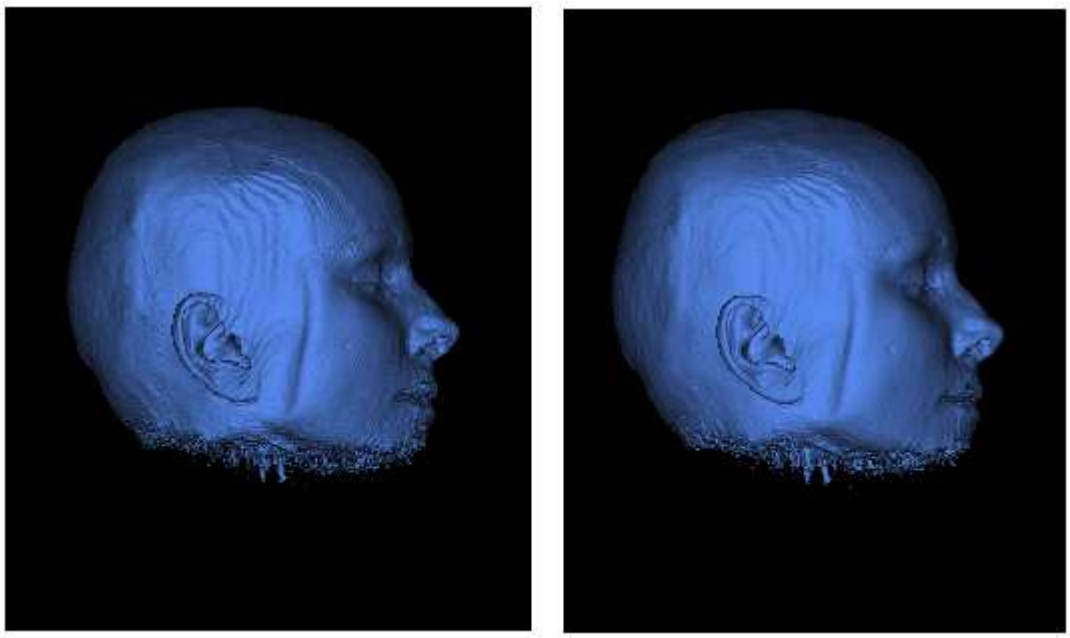


Figure 8

3dhead, image of the left side is from First Algorithm.

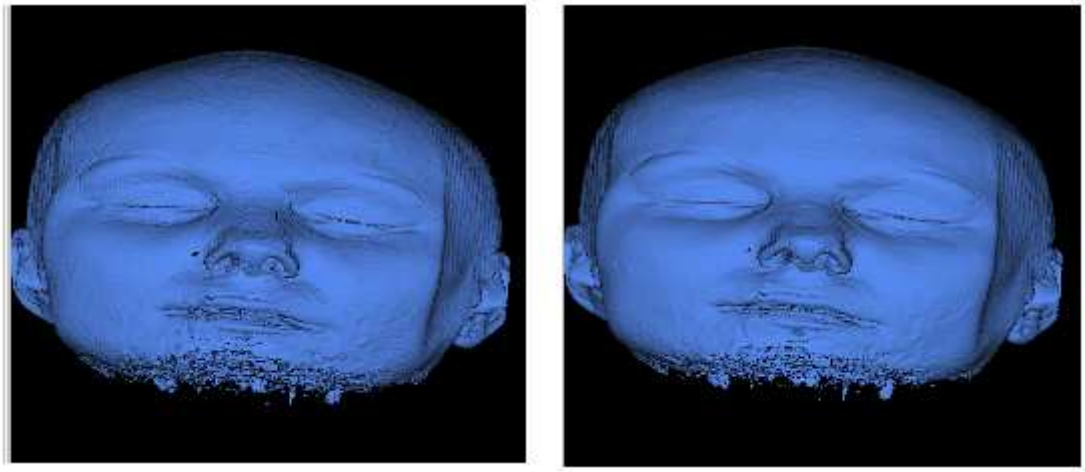


Figure 9

3dhead, image of the left side is from First Algorithm.

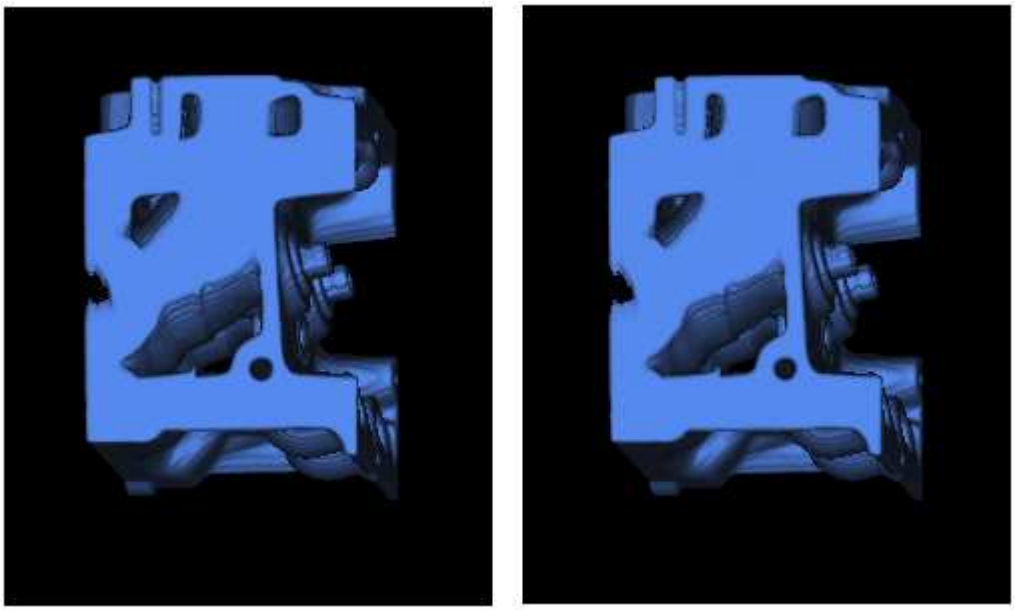


Figure 10

Engine, image of the left side is from First Algorithm.

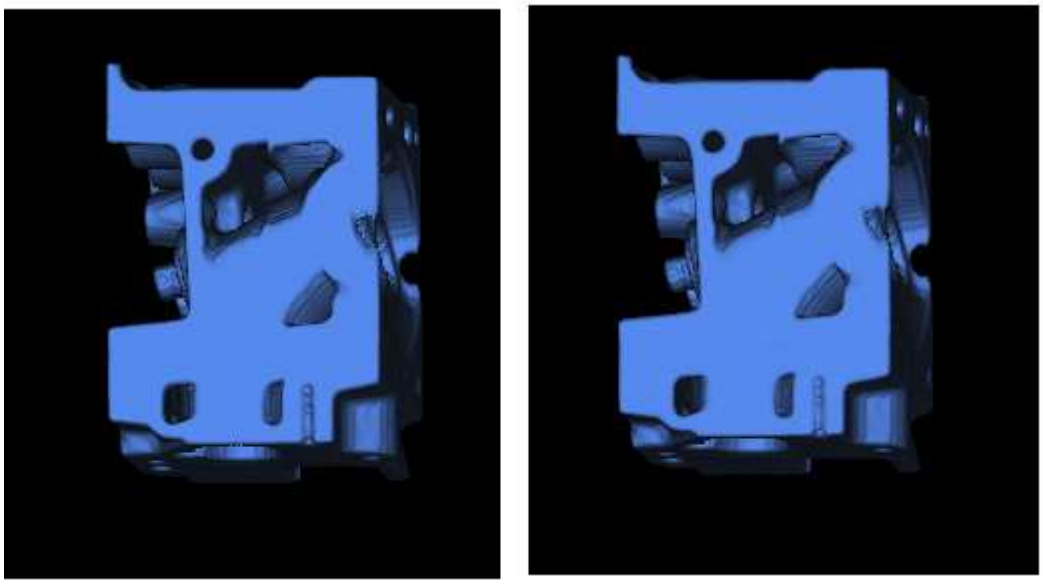


Figure 11

Engine, image of the left side is from First Algorithm.

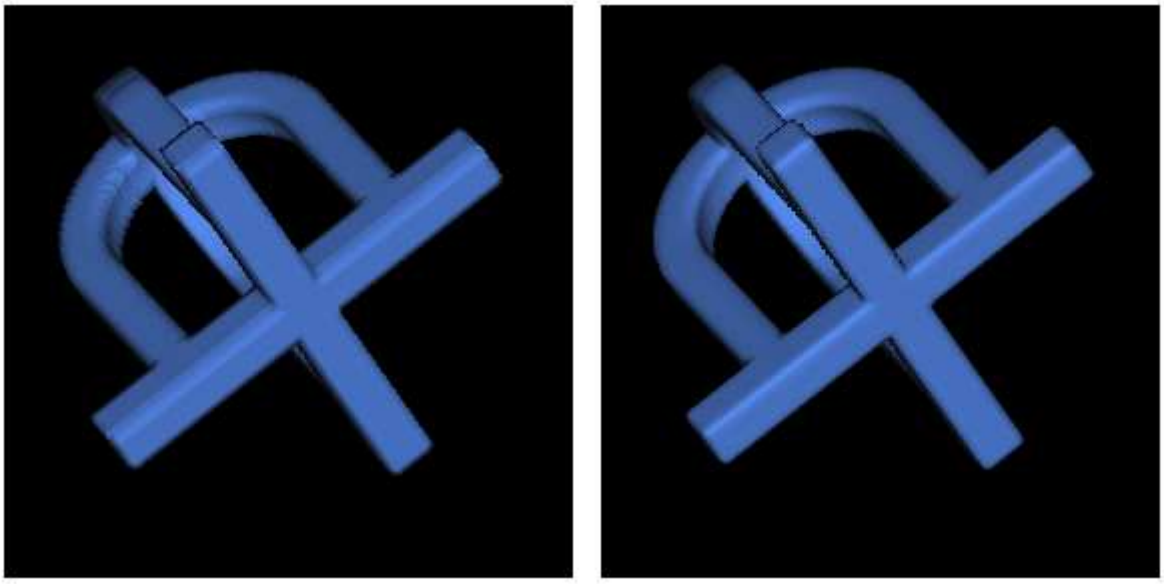


Figure 12

Syn64, image of the left side is from First Algorithm.

Appendix B. CDROM content

Enclosed CDROM contains:

- Word97 version of the thesis (not for public use)
- RTF version of the thesis (not for public use)
- Postcript version of the thesis
- Programs in the source format (not for public use)
- Programs in the executable version
- Papers used for the thesis, found in internet

All files are accessible via main.htm page in the root directory