UDC 004.925

N. Jaworski, P. Shmigelskyi, I. Farmaga
Lviv Polytechnic National University, Department of Computer-Aided Design

VISUALIZATION OF REPRESENTATIVE VOLUME ELEMENTS OF COMPOSITE MATERIALS CELLULAR MODELS BASED ON THREE DIMENSIONAL TEXTURING

На основі використання технологій тривимірного текстурування запропоновано метод візуалізації елементарних об'ємів коміркових моделей композиційних матеріалів, що на відміну від відомих підходів відображає композицію взаємно ортогональних площин текстурних перерізів, з можливістю обертання не текстури, а самого об'єкта. Це дає можливість краще деталізувати кінцеве зображення під час збереження відносної простоти обчислень.

Ключові слова: об'ємний рендеринг, композиційний матеріал, елементарний об'єм, коміркова модель.

Basing on three dimensional texturing technologies was proposed a visualization method of representative volume elements of composite materials cellular models. In contrast to the known approaches, the method renders a composition of mutually orthogonal texture cut planes with the ability to rotate no texture, but the object itself. This enables better detailed final image with maintaining of the relative calculations simplicity.

Key words: volume rendering, composite material, representative volume element, cellular model.

Introduction

The development of composite materials design methods requires every time greater detalization of corresponding physical and mathematical models. Active usage of composite structures microlevel detailed models causes the development of visualization tools with the purpose for analysis or verification at the design process. Since the work with a composite material microlevel model provides processing of enough large amounts of data, the actual task of engineering design systems implementation, which can be used on ordinary personal computers, is appearing.

One of the common ways to research complex composite structures is to do their description by microlevel cellular models, which represent a three-dimensional matrix of some scalars. Such volumetric data arrangement gives the ability to use volume rendering techniques for visualization, including three-dimensional texturing, which are hardware supported by common personal computer graphic devices.

This paper proposes a visualization method of representative volume elements of composite materials cellular models, which in contrast to known approaches is using mutually orthogonal texture cut planes rendering with the ability to rotate no texture, but the object itself. This enables better detailed final image with maintaining of the relative calculations simplicity.

Composite Materials Cellular Models Representative Volume Elements

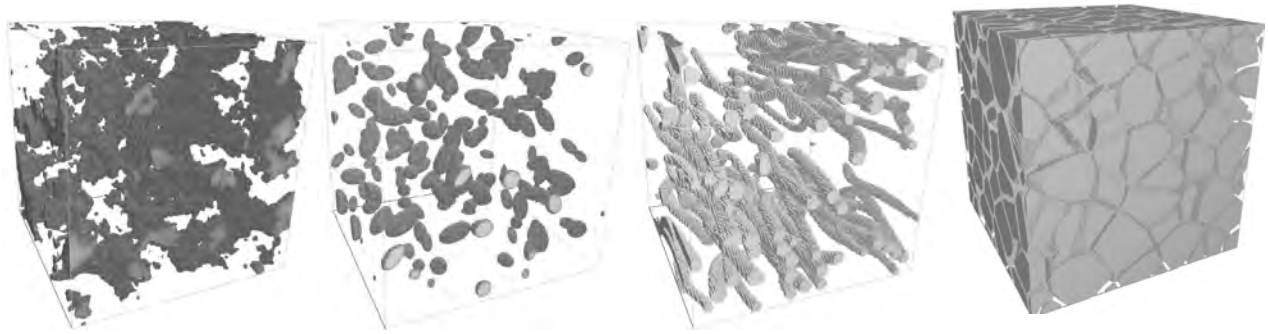
Microlevel models involve the construction so-called representative volume element (RVE) – usually a volume $\Omega \subset R^3$ of heterogeneous material, sufficiently large to describe it statistically, i.e. the smallest composite material volume, for which macroscopic representation of spatial characteristics is sufficiently accurate model of effective response on corresponding outer influence [1]. In construction of the RVEs is convenient enough to use the cellular structure models, in the form of a large number of

regular voxel-cells that simultaneously represent a regular finite-element discretization [2], [3]. Advantages of approach include: simplicity and relatively small number of computations in discretization; the possibility of direct usage of domain decomposition methods for calculations and corresponding effective implementation on devices with big number of computing nodes [4]; universality, which allows one to construct in one way such composite material complex structure models, as a model of random scalar fields, random cellular models, models with deterministic inclusions, and the combination of these models with the ability to build functional transition layers.

Construction of the RVE is the task of composite materials structure modeling that prior to physical processes analysis in these materials [5]. Classically, the heterogeneous systems microlevel model differential balance equations that describe physical processes, consider the structure of the CM as a combination of component material characteristics that are represented by equations coefficients and topology of the material, which is described by the integration borders where these equations are defined. The pair:

$$(\Omega, \mathbf{D}) = \bigcup_p \left(\Omega_p, \mathbf{D}_p \right) \tag{1}$$

where $\mathbf{D}_p$ – the set of characteristics of p-th component, and Ω_p – corresponding geometric area, i.e. its topology; completely describes the composite material microlevel structure model. By using this formalization, the RVE can be conveniently presented as a cubic matrix of scalar intensities, i.e. cells that accept scalar values in a certain range, for example from 0 to 1. With a large number of cells, by defining the intensities intervals as a separate composite phases Ω_p and giving them an appropriate characteristics set $\mathbf{D}_p$ it is possible to construct the model of complex structure (*Fig. 1*).



a *b* *c* *d*

Fig. 1. Example of representative volume elements in the form of a 256x256x256 elements matrix that represents composite materials microlevel structure models: a – scalar random fields; b – random ellipsoid particles; c – fibers; d – cellular structures

Volume Rendering

Volume rendering – the technique which is used to obtain a flat two-dimensional image from discrete volumetric data set [6], which is commonly represented as a three-dimensional scalar fields. Volumetric models can be obtained either by constructing a polygonal mesh based on input data, e.g. by using marching cubes algorithm [7], or by so-called direct volume rendering. There are several approaches for the implementation of last:

- the direct approach – the direct mapping of each volumetric data element by converting them into the appropriate pixel projection into graphics device frame buffer;
- volume ray casting, which like a two-dimensional ray tracing technique provides: "ray casting" phase that defines three-dimensional elements on which the ray that released from the image projected plane falls; sampling phase in which the position of found elements will be simplified and aligned along the ray; shading phase in which the color of each element will be found depending on its orientation relative to the camera; compositing phase in which all found colors will be arranged in some way into the resulting image pixel;

- splatting (throwing snowballs), in which the volumetric data elements are drawn in the order of distance from the projection plane in the form of some primitives, such as disks, color and transparency of which vary depend on diameters;
- texture-based volume rendering, which uses the hardware texturing tools for direct volumetric data rendering.

Thanks to domain decomposition and significant development of hardware graphics devices, volume rendering tasks can be relatively easily implemented in acceptable time on ordinary personal computers. Particular attention is, given to the three-dimensional texturing tools [8], [9], which are included in the standard libraries such as OpenGL starting from version 1.2 or DirectX from version 10.

Composite Materials Representative Volume Elements Rendering Implementation

Unlike to known [8], [9] volume rendering approaches that are based on three-dimensional texturing, which implements volume ray casting and displays a certain number of texture cut planes with the ability of that texture rotation, taking into account the power of modern ordinary personal computers graphics cards, here is proposed a method that renders a composition of mutually orthogonal texture cut planes with the ability to rotate no texture, but the object itself. This enables better detailed final image with maintaining of the relative calculations simplicity.

The general composite materials cellular models representative volume elements visualization algorithm

Input:	An array of real numbers within 0 to 1 diapason with size N^3, where N – the number of cells along RVE side.
1:	Create a memory buffer with size N^3x4 bytes (byte per each RGBA channel).
2:	Fill the buffer according to input data by using grayscale palette or rainbow palette depending on what RVE is describing – composite structure or some simulated potential field. If some cell hasn't be rendered, assign zero value to corresponding alpha-channel.
3:	By using corresponding API, create a three-dimensional texture from filled memory buffer.
4:	Configure texture rendering parameters.
5:	For each RVE cut plane, by corresponding API tools, draw textured quad – 3N quads in total.
Output:	RVE image.

Before drawing process begin, it is necessary to enable by corresponding API tools the alpha test with parameter to draw elements, alpha channel of which is greater than zero. If there are not used transparency effects and colors blending, then have to be enabled texture rendering parameters:

- clamping – clamp border texels to edge;
- texel filter – draw nearest;
- texture environment mode – modulate textures;
- enable lights;
- enable depth test;
- disable blending.

If there are uses transparency effects and colors blending the rendering process occurs in two stages [10]: first into the graphics device frame buffer is copied information about cells transparency, and then according to it transparency is copied information about colors. For the first stage have to be enabled texture rendering parameters:

- clamping – clamp border texels to edge;
- texel filter – draw nearest;

- texture environment mode – replace textures;
- enable depth test;
- blending function – (source alpha : 1 – source alpha);
- separated blending function – for RGB – (0 : 1), for alpha channel – (1 : 0).

For the second stage:

- clamping – clamp whole texture;
- texel filter – linear interpolation;
- texture environment mode – replace textures;
- enable depth test;
- blending function – (1 : 1 – source alpha);
- separated blending function – for RGB – (destination alpha : 1 – destination alpha), for alpha channel – (0 : 0).

Described steps were implemented under Windows 7 x64 with C++11 by using Qt SDK 5.4.1, MinGW 4.9.2 compiler, and OpenGL 3.0 + Extension Wrangler Library. Graphics device was AMD Radeon HD 6300M Series.

The fragment of OpenGL initialization function:

```
QGLFormat aGLFormat;
aGLFormat.setSampleBuffers(true);
aGLFormat.setVersion(aGLFormat.majorVersion(),aGLFormat.minorVersion());
QGLFormat::setDefaultFormat(aGLFormat);
glClearDepth(1.0f);
glDepthFunc(GL_LEQUAL);
glEnable(GL_ALPHA_TEST);
glAlphaFunc(GL_GREATER, 0.0f);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_NORMALIZE);
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
GLfloat light0_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
GLfloat light0_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0f);
glEnable(GL_LIGHT0);
```

The fragment of texture preparation function, for composite structure visualization:

```
GLbyte *RGBABuff = new GLbyte[N*N*N*4];
for(long i = 0; i< N*N*N; ++i){
    RGBABuff[i * 4 + 0] = 255;
    RGBABuff[i * 4 + 1] = 255;
    RGBABuff[i * 4 + 2] = 255;
    RGBABuff[i * 4 + 3] = (ptrToRVEdata[i] > innerCutLevel) ? 255 : 0;
}
glBindTexture(GL_TEXTURE_3D, textureIDs[0]);
// see https://www.opengl.org/sdk/docs/man3/xhtml/glTexImage3D.xml
glTexImage3D(
       GL_TEXTURE_3D,      // target, copy data to device
       0,                  // level
       GL_RGBA,            // internalFormat
       N,                  // width
       N,                  // height
       N,                  // depth
       0,                  // border
       GL_RGBA,            // format
       GL_UNSIGNED_BYTE,   // type
       RGBABuff );         // ptr to data
delete[] RGBABuff;
```

The fragment of texture preparation function, for potential field visualization:

```
GLbyte *RGBABuff = new GLbyte[N*N*N*4];
float delta = maxPotentialValue - minPotentialValue;
for(long i = 0; i< N*N*N; ++i){
    int r, g, b;
    float val = (ptrToRVEpotentialField[i] - minPotentialValue) / delta;
    if(val >= 0.0f && val <= 1.0f)
        RGBABuff[i * 4 + 3] = potentialFieldAlphaLevel;
    else{
        if(val < 0.0f) val = 0.0f;
        if(val > 1.0f) val = 1.0f;
        RGBABuff[i * 4 + 3] = 0;
    }
    grayscaleToRainbow(val, r, g, b);
    RGBABuff[i * 4 + 0] = r;
    RGBABuff[i * 4 + 1] = g;
    RGBABuff[i * 4 + 2] = b;
}
glBindTexture(GL_TEXTURE_3D, textureIDs[1]);
glTexImage3D(GL_TEXTURE_3D,0,GL_RGBA,N,N,N,0,GL_RGBA,GL_UNSIGNED_BYTE, RGBABuff);
delete[] RGBABuff;
```

The rainbow palette function:

```
void grayscaleToRainbow(const float gray, int &r, int &g, int &b) noexcept {
    float inv = (1.0f-gray)*4.0f;       //invert and group
    int X = std::floor(inv);            //this is the integer part
    int Y = std::floor(255*(inv-X));    //fractional part from 0 to 255
    switch(X){
        case 0: r = 255;        g = Y;          b = 0;     break;
        case 1: r = 255 - Y;    g = 255;        b = 0;     break;
        case 2: r = 0;          g = 255;        b = Y;     break;
        case 3: r = 0;          g = 255 - Y;    b = 255;   break;
        case 4: r = 0;          g = 0;          b = 255;   break;
    }
}
```

The fragment of texture rendering function, without transparency effects and colors blending:

```
glNewList(firstDisplayListID, GL_COMPILE);
glEnable(GL_TEXTURE_3D);
glBindTexture( GL_TEXTURE_3D, textureIDs[0]);
glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
glEnable(GL_LIGHTING);
glEnable(GL_DEPTH_TEST);
glDisable(GL_BLEND);
glColor4f(1.0f, 1.0f, 1.0f, 1.0f);  // material color (for lightning)
glBegin(GL_QUADS);
for ( float fIndx = 0.0f; fIndx <= 1.0f; fIndx += 1.0 / N){
    glNormal3f(0.0f, 0.0f, 1.0f);
    glTexCoord3f(0.0f, 0.0f, fIndx);   glVertex3f(0.0f, 0.0f, fIndx);
    glTexCoord3f(1.0f, 0.0f, fIndx);   glVertex3f(1.0f, 0.0f, fIndx);
    glTexCoord3f(1.0f, 1.0f, fIndx);   glVertex3f(1.0f, 1.0f, fIndx);
    glTexCoord3f(0.0f, 1.0f, fIndx);   glVertex3f(0.0f, 1.0f, fIndx);
    glNormal3f(0.0f, 1.0f, 0.0f);
    glTexCoord3f(1.0f, fIndx, 0.0f);   glVertex3f(1.0f, fIndx, 0.0f);
    glTexCoord3f(0.0f, fIndx, 0.0f);   glVertex3f(0.0f, fIndx, 0.0f);
    glTexCoord3f(0.0f, fIndx, 1.0f);   glVertex3f(0.0f, fIndx, 1.0f);
    glTexCoord3f(1.0f, fIndx, 1.0f);   glVertex3f(1.0f, fIndx, 1.0f);
    glNormal3f(1.0f, 0.0f, 0.0f);
    glTexCoord3f(fIndx, 0.0f, 0.0f);   glVertex3f(fIndx, 0.0f, 0.0f);
    glTexCoord3f(fIndx, 1.0f, 0.0f);   glVertex3f(fIndx, 1.0f, 0.0f);
    glTexCoord3f(fIndx, 1.0f, 1.0f);   glVertex3f(fIndx, 1.0f, 1.0f);
    glTexCoord3f(fIndx, 0.0f, 1.0f);   glVertex3f(fIndx, 0.0f, 1.0f);
}
glEnd();
glEnable(GL_BLEND);
glDisable(GL_DEPTH_TEST);
glDisable(GL_LIGHTING);
glDisable(GL_TEXTURE_3D);
glEndList();
```

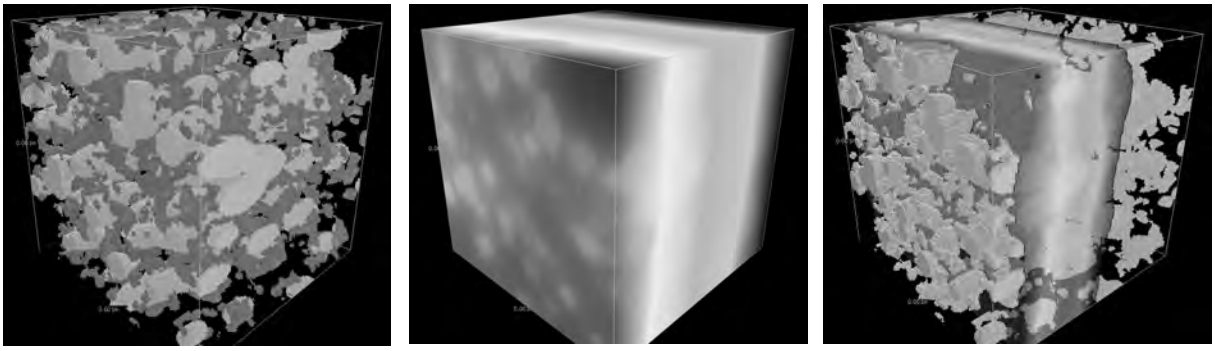
The fragment of texture rendering function, with transparency effects and colors blending:

```
glNewList(firstDisplayListID+1, GL_COMPILE);
glEnable(GL_TEXTURE_3D);
glBindTexture(GL_TEXTURE_3D, textureIDs[1]);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
glEnable(GL_DEPTH_TEST);
for (int i = 0; i<2; ++i){
    if (i == 0){ // Clear alpha buffer
        glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
        glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
        glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
        glBlendFuncSeparate(GL_ZERO, GL_ONE, GL_ONE, GL_ZERO);
    } else if (i == 1){ // Draw pixels
        glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameterf(GL_TEXTURE_3D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_S, GL_CLAMP);
        glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_T, GL_CLAMP);
        glTexParameteri(GL_TEXTURE_3D, GL_TEXTURE_WRAP_R, GL_CLAMP);
        glBlendFuncSeparate(GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA, GL_ZERO, GL_ZERO);
    } else {
        glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
    }
    glBegin(GL_QUADS);
    for ( float fIndx = 0.0f; fIndx <= 1.0f; fIndx += 1.0 / N) {
        glTexCoord3f(0.0f, 0.0f, fIndx);   glVertex3f(0.0f, 0.0f, fIndx);
        glTexCoord3f(1.0f, 0.0f, fIndx);   glVertex3f(1.0f, 0.0f, fIndx);
        glTexCoord3f(1.0f, 1.0f, fIndx);   glVertex3f(1.0f, 1.0f, fIndx);
        glTexCoord3f(0.0f, 1.0f, fIndx);   glVertex3f(0.0f, 1.0f, fIndx);
        glTexCoord3f(1.0f, fIndx, 0.0f);   glVertex3f(1.0f, fIndx, 0.0f);
        glTexCoord3f(0.0f, fIndx, 0.0f);   glVertex3f(0.0f, fIndx, 0.0f);
        glTexCoord3f(0.0f, fIndx, 1.0f);   glVertex3f(0.0f, fIndx, 1.0f);
        glTexCoord3f(1.0f, fIndx, 1.0f);   glVertex3f(1.0f, fIndx, 1.0f);
        glTexCoord3f(fIndx, 0.0f, 0.0f);   glVertex3f(fIndx, 0.0f, 0.0f);
        glTexCoord3f(fIndx, 1.0f, 0.0f);   glVertex3f(fIndx, 1.0f, 0.0f);
        glTexCoord3f(fIndx, 1.0f, 1.0f);   glVertex3f(fIndx, 1.0f, 1.0f);
        glTexCoord3f(fIndx, 0.0f, 1.0f);   glVertex3f(fIndx, 0.0f, 1.0f);
        }
    glEnd();
}
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glDisable(GL_DEPTH_TEST);
glDisable(GL_TEXTURE_3D);
glEndList();
```

It is enough to call created display lists for rendering:

```
glCallList(_firstDisplayListID);
```

The examples of visualization are shown on *Fig. 1, 2.*



a *b* *c*

Fig. 2. An example of composite materials cellular models RVE visualization (128×128×128): a – composite material structure; b – simulated temperature field; c – combination of composite structure and simulated temperature field with applying a cut on some temperature level

Conclusions

In this paper, basing on three dimensional texturing technologies was proposed a visualization method of representative volume elements of composite materials cellular models. In contrast to the known approaches, the method renders a composition of mutually orthogonal texture cut planes with the ability to rotate no texture, but the object itself. This enables better detailed final image with maintaining of the relative calculations simplicity.

References

1. Kanit T. Determination of the size of the representative volume element for random composites: statistical and numerical approach / T. Kanit, S. Forest, I. Galliet, V. Mounoury, D. Jeulin // Int. Journ. of Solids and Structures. – 2003. – Vol. 40. – P. 3647–3679. 2. Torquato S. Random Heterogeneous Materials. Microstructure and Macroscopic Properties / Torquato S. – New-York: Springer, 2002. – 556 p. 3. Jaworski N. Effective Thermal Characteristics Synthesis Microlevel Models in the Problems of Composite Materials Optimal Design / N. Jaworski // ECONTECHMOD: an international quarterly journal on economics of technology and modelling processes. – 2015. – Vol. 4. – № 2. – P. 3–12. 4. Jaworski N. Architecture of the Composite Materials Distributed Heterogeneous Computer-Aided Design System / N. Jaworski, M. Lobur, I. Farmaga, K. Kurzydlowski // IEEE Proc. of the XII-th Int. Conf. "The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM'2013)". – Polyana-Svalyava (Ukraine), 2013. – P. 440–442. 5. Jaworski N. Thermal Analysis Methods for Design of Composite Materials with Complex Structure / N. Jaworski, I. Farmaga, O. Matviykiv, M. Lobur, P. Spiewak, L. Ciupinski, K. Kurzydlowski // ECS Transactions. – 2014. – Vol. 59. – № 1. – P. 513–523. 6. Kaufman A. Overview of Volume Rendering [In: The Visualization Handbook, eds. C. Hansen, C. Johnson] / Kaufman A., Mueller K. – Elsevier Butterworth-Heinemann, 2005. – P. 127–174. 7. Chernyaev E. Marching Cubes 33: Construction of Topologically Correct Isosurfaces / E. Chernyaev // Saint-Petersburg: GRAPHICON'95. – 1995. – 9 p. 8. Lewiner T. VSVR: a very simple volume rendering implementation with 3D textures / Lewiner T. – Preprint MAT 16/06, communicated on June 26-th, 2006 to the Department of Mathematics, PUC – Rio de Janeiro (Brazil), 2006. 9. Augustine D. Getting started with Volume Rendering using OpenGL: Step by step explanation of 3D image rendering using OpenGL, June 30 2013 [electronic resource]: <http://www.codeproject.com/Articles/352270/Getting-started-with-Volume-Rendering>, title from screen, accessed October 8 2015. 10. Spencer A. Use glBlendFuncSeparate for alpha masking in tex.c, Jan 30 2012 [electronic resource]: <https://mail.gnome.org/archives/commits-list/2012-February/msg05204.html>, title from screen, accessed October 9 2015.