Parallel plug-in for 3D reconstruction of medical images

Abstract

In this paper we present our new parallel plug-in for creating virtual models of human organs and other parts of a human body. The plug-in is incorporated into Blender environment and Python scripts are used for its basic functionality . For more demanding operations parallel implementation in C++ language with hybrid technique combining OpenMP and MPI is used. The implementation can exploit computational power of both non-accelerated as well as Intel Xeon Phi accelerated compute nodes of a cluster. In this paper we describe the plug-in construction and its basic functionality. The figures illustrate the process of the 3D model reconstruction of a mandible.

Keywords: Medical imaging, Blender plug-in, Surface Reconstruction, Metaballs, Python, Intel Xeon Phi

1. Introduction

The rapid development of medical imaging and image processing methods leads to the development of the techniques that can be used to create virtual models of human body. The models of human organs are now widely required by doctors for diagnostic purposes and planning of patient treatments. For surgeons such models can be of high importance when very complicated surgical operations are planned.

In this paper parallel tool for processing of medical images is presented. This tool was developed to create 3D virtual model of human organs from CT (Computed Tomography) and MRI (Magnetic Resonance Imaging) scans. It exploits advanced techniques for model reconstruction, parallel implementation techniques of more demanding methods running on the supercomputer, and processing and rendering of results in the real time.

In our development an open-source software Blender is used. Tool presented in this paper is created as a new internal plug-in of Blender. GUI and basic behaviour of the plug-in are implemented in the Python scripting language. More demanding tools are implemented in C++ using libraries and directives for parallel programming like MPI and OpenMP. The control menu, which has a workflow structure, is placed in Blender's UV/Image Editor. Using this menu a required 3D model can be created in a few easy steps (loading of data, denoising, segmentation, boundary extraction and model reconstruction). The support for auxiliary calculations is performed by our new application called Blender-client.

In Section 2 we describe the creation and control of the internal plug-in and its component Blender-client. In Section 3 the workflow of the 3D model reconstruction and its basic components is outlined. In the final section, methods for surface reconstruction by the Metaballs and Poisson surface reconstruction method are depicted and their advantages and disadvantages are illustrated.

2. Internal plug-in

Our plug-in is based on the Blender version 2.75. To compile it the latest Intel library 2016.01 (MPI compiler) is used in combination with GCC version 4.9.3.

Using Intel compiler gives us the possibility to take all advantages of the Intel libraries such as vectorization, compilation for the Intel Xeon Phi coprocessor (MIC) and the exploitation of the latest standard of OpenMP and MPI for parallelization. It also allows distribution of calculations not only across the nodes, but also to the MIC coprocessors.

Our plug-in is made as a Blender module which is created and registered in the following manner:

```
static PyObject *poissonReconstruction_func
(PyObject * /*self*/, PyObject *args)
{
  ...
  if (!PyArg_ParseTuple(args, "OOff", &pyBoundary,
        &pyBoundaryVector, ...))
  {
    return NULL;
  }
  ...
  PyObject* vertices = PyList_New(vs.size());
  PyObject* faces = PyList_New(fs.size());
  ...
  return Py_BuildValue("OO", vertices, faces);
}

static PyMethodDef methods[] =
{
  ...
  { "poissonReconstruction",
  (PyCFunction)poissonReconstruction_func, METH_VARARGS, "" },
  ...
}

static struct PyModuleDef module =
{
  PyModuleDef_HEAD_INIT,
  "_xxx_dicom",
  ...
  methods,
  ...
}

static PyObject *XXX_initPython(void)
{
  PyObject *mod = PyModule_Create(&module);
  return (void*)mod;
}

static struct _inittab bpy_internal_modules[] =
```

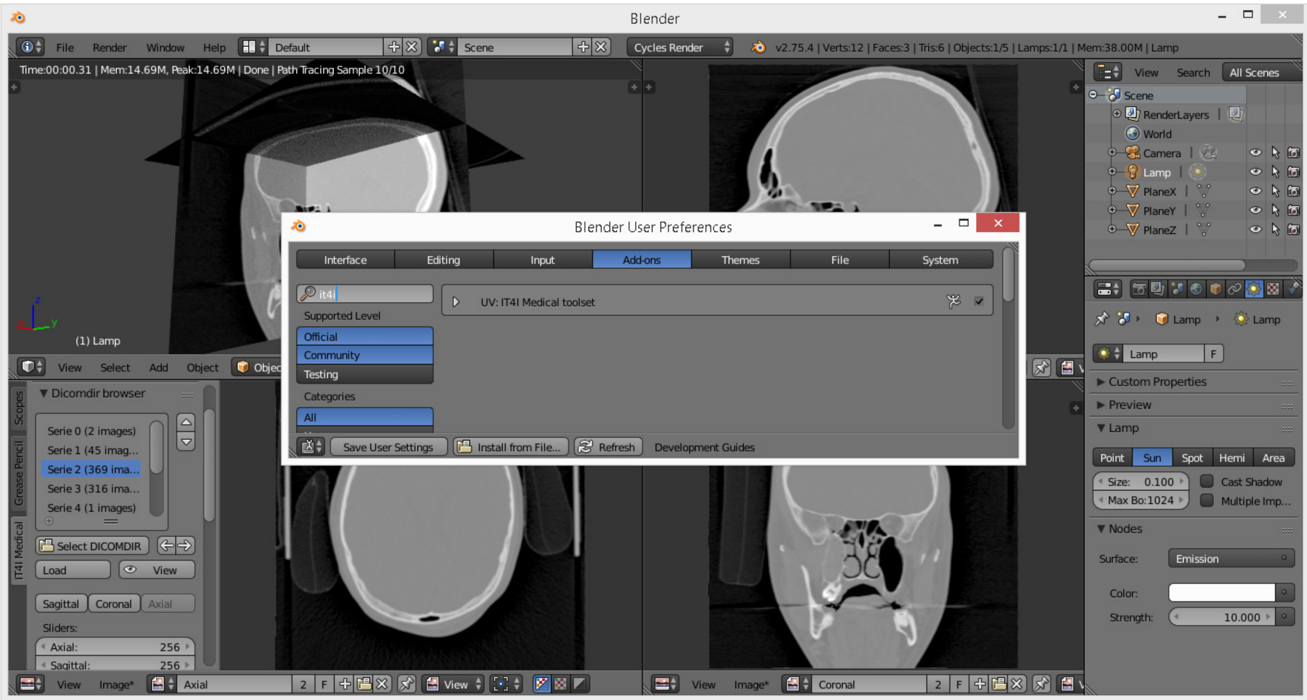

Figure 1: Blender environment with our plug-in

```
{
  ...
  { "_xxx_dicom", XXX_initPython },
};
```

Then it is enough to create Python script "__init__.py" and save it to directory "blender/2.75/scripts/addons/xxx". Example of such file could look like this:

```
bl_info = {
  "name": "XXX Medical toolset",
  "category": "UV",
  "author": "",
  "version": (1,0,0),
  "description": "XXX Medical toolset"
  }

import bpy, _xxx_dicom
....
# Draw the panel for poisson method
class UIPoisson(Panel):
  bl_idname = 'Poisson_panel_11'
  bl_category = "XXX Medical"
  bl_space_type = 'IMAGE_EDITOR'
  bl_region_type = 'TOOLS'
  bl_label = "Perform poisson reconstruction"

  def draw(self, context):
    layout = self.layout
    scn = bpy.context.scene

    row = layout.row()
    row.prop(context.scene, "poisson_depth", slider=False)
    row.prop(context.scene, "poisson_smooth", slider=False)

    row = layout.row()
    row.operator("custom.poisson_rec", icon="OBJECT_DATAMODE",
                 text="Make poisson reconstruction")

# Select Make poisson reconstruction button
class UIPoisson_rec(bpy.types.Operator):
  bl_idname = "custom.poisson_rec"
  bl_label = "Poisson rec"
  bl_description = "Create poisson reconstruction from data"

  def execute(self, context):
    (verts, faces) =
        _XXX_dicom.poissonReconstruction(boundary,
                        boundaryVector, ...)

# Create mesh and object
    createMesh(verts, faces)

    return {"FINISHED"}
```

Graphical elements were created by above mentioned script directly in Blender environment as it is depicted in Figure 1.

2.1. Blender client

For faster execution of most time consuming algorithms, we have created a client system, which performs the most demanding tasks such as Marching Cubes method. The basic principle of the client system is shown in Figure 2. The Blender plug-in is used in Offload mode for computation on MIC. The client itself works in symmetric mode. In this mode MPI programs are executed on both host computer (CPU) and MIC accelerator. Series of such clients working in symmetric mode can be established. Work division and collection of results is maintained using MPI.

Since MIC has a different architecture and requires different binary file produced by the Intel compiler, two different files have to be compiled before MPI program is executed.

3. Basic elements of the plug-in

The plug-in contains user friendly menu which has a workflow structure. It is placed in the UV/Image Editor, see Figure 1. By this menu a required 3D model can be created in a few easy steps:

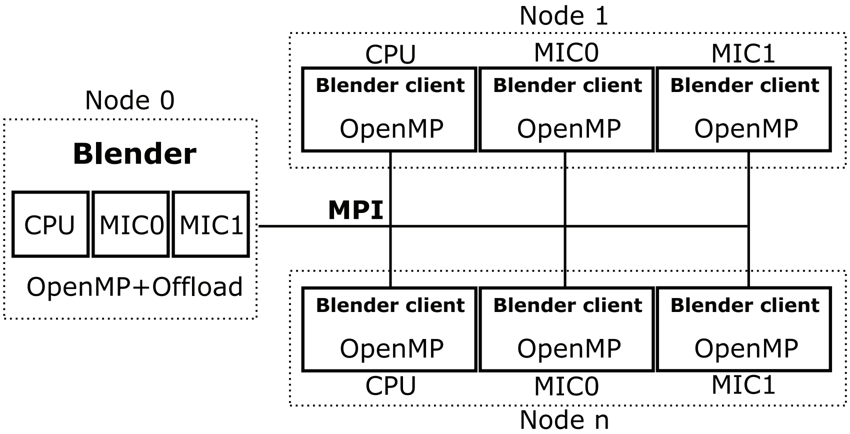

Figure 2: Overview of the communication between Blender and its client

Figure 3: Loaded data

1. loading data from DICOM format (GDCM libraries are used),
2. application of a filter to reduce image noise: e.g., Gaussian blur, BM3D filter, anisotropic filter,
3. image segmentation (e.g., thresholding, K-means clustering),
4. boundary extraction, computation of normals,
5. 3D model reconstruction (e.g., Metaballs and Poisson surface reconstruction method),
6. visualization

3.1. Input data

In the first step of the process we have to obtain the image data stored in the DICOM format, see Figure 3. DICOM (Digital Imaging and Communications in Medicine) is a standard for storing, displaying and distributing medical data obtained from CT, MRI or ultrasound. For loading DICOM file we have integrated the library Grassroots DICOM (GDCM) directly into Blender. GDCM is a C++ library for DICOM medical files.

The data from CT are stored as a images, usually as axial slices at mutual axial distance. The distance between two axial slices is mostly in range between 0.6 and 5.0 mm. For better accuracy of final model smaller values are preferred.

After the data loading, volume of data could be restricted to selected area of interest, see Figure 4, by providing tools such as box cutting tool or sphere cutting tool.

3.2. Image denoising

In present version of the plug-in, three different types of image filters are used for denoising. The Gaussian smoothing filter [1], anisotropic diffusion filter [2] and BM3D filter [3]. The results after denoising images by Gaussian smoothing are shown in Figure 4.

3.3. Image segmentation

After images are pre-processed by denoising, image segmentation is performed. The segmentation simplifies representation of the image so localization of objects and boundaries in the image can be created. We are using two segmentation methods. Simple image thresholding [1] and k-means clustering [4]. They can be used separately or combined together.

3.3.1. Thresholding

The thresholding method is used to transform the original image to the reduced version of the image by eliminating selected pixel intensity values [1]. Specifically values grater than T_{max} and lower that T_{min}. The values between T_{min} and T_{max} remain unchanged.

Figure 4: Denoising and cutting image

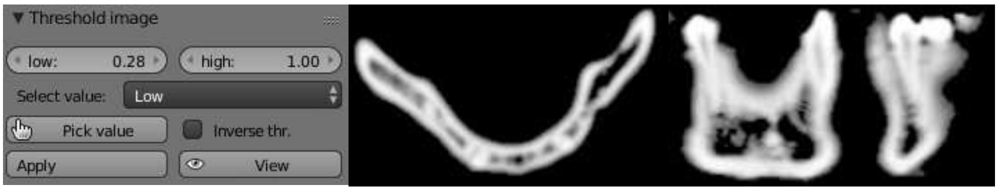

Figure 5: Thresholding

We can apply this segmentation technique repeatedly on the original images with different settings and then merge the results to provide advanced version of the thresholding method.

3.3.2. K-means clustering

K-means clustering is method originated in signal processing and is often used in data mining [4]. Generally this method can be employed in different areas including image processing as an image segmentation technique. This method divides pixels into k clusters according to some similar features like an intensity of a pixel and distance of the pixel intensity from a centroid pixel intensity.

The parallelization of this method by using Intel Xeon Phi was introduced in the paper [5].

3.4. Finding connected segments, boundary extraction and computation of normals

The subsequent task after image segmentation is finding only the connected areas of selected segment and extraction of its boundary, see Figure 6 and Figure 7. Boundary of the segment is represented by a set of pixels obtained by the flood algorithm. This boundary is used for 3D reconstruction using both methods, Poisson reconstruction and Metaballs. For application of Poisson method, computation of the normal vector $n_i = (n_x, n_y, n_z)$ in each point of the boundary is also necessary. The enumeration of the normal vector is depicted in Figure 8.

3.5. Surface reconstruction by Poisson method

As mentioned in previous subsection we are using Poisson method as a technique for reconstruction of the 3D surface. Original name is Screened Poisson Surface Reconstruction [6, 7]. Advantage of this method is that it is not sensitive to noise, because it uses the whole set of points at once. The basic procedure consists of three steps: computation of gradient $\nabla\chi_M$, computation of indicator function χ_M and extraction of iso-surface ∂M (using Marching Cubes method). The gradient is computed from normals $\vec{V}$ that serve as an input to the method.

We need to formulate and solve the Poisson problem. After supplying a set of vectors $\vec{V}$ we need to find a function $\tilde{\chi}$ using the equation $\nabla\tilde{\chi} = \vec{V}$. Since the set $\vec{V}$ is not integrable, we use an operator of divergence and a least squares method for formulating Poisson equation

$$\Delta\tilde{\chi} = \nabla \cdot \vec{V}. \tag{1}$$

After discretization we get

$$\chi(p) = \sum_{i=1}^{N} x_i B_i(p), \tag{2}$$

where $B_i : \mathbb{R}^3 \to \mathbb{R}$ is a Bspline basis function.

To find the coefficients x_i we have to solve a system of linear equations $A_{ij} x_i = b_i$ (A is a sparse symmetric matrix) where

$$\begin{aligned} A_{ij} &= \int_M \left\langle \nabla B_i(q), \nabla B_j(q) \right\rangle dq \\ b_i &= \int_M \left\langle \nabla B_i(q), \vec{V}(q) \right\rangle dq \end{aligned} \tag{3}$$

In our solution we are using original implementation from the authors of the method. This version is already optimized and parallelized. The resulting 3D model coming from the Poisson reconstruction is shown in Figure 9.

3.6. Metaballs method

The development of a technology for creating realistic and visually interesting images of three-dimensional shapes goes back to 80's of the 20th century [8]. The

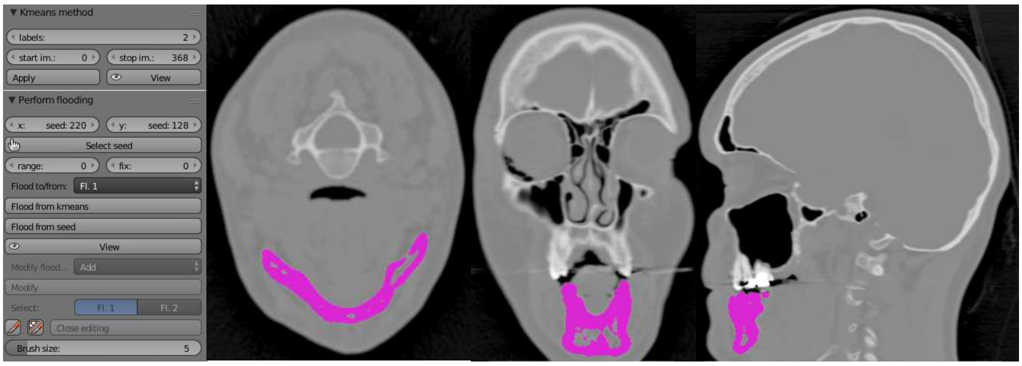

Figure 6: K-means clustering

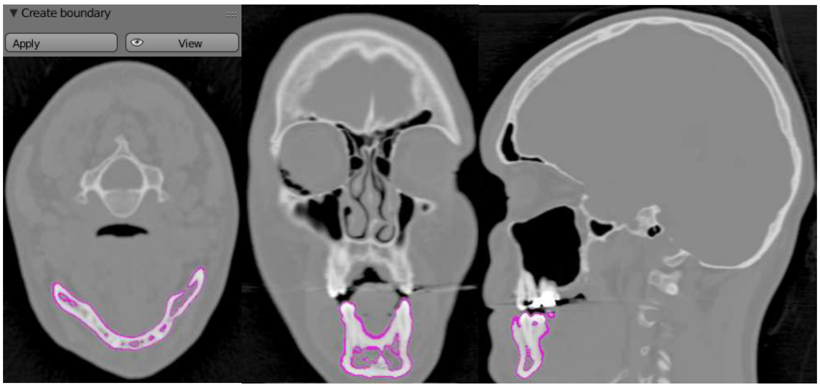

Figure 7: Boundary extraction

method is based on the construction of the iso-surfaces. For each voxel with coordinates (x, y, z) the value of the potential function

$$g(x,y,z) = \sum_{i=1}^{N} \begin{cases} f_i(x,y,z) & for\ f_i(x,y,z) \leq 1, \\ 0 & other, \end{cases} \tag{4}$$

is computed. In the previous formula N is a total number of the metaballs and $f_i(x, y, z)$ represents a function of the i-th metaball. In our implementation ellipsoidal metaballs with f_i in the form

$$f_i(x,y,z) = \frac{(x - x_i^0)^2}{(r_i^x)^2} + \frac{(y - y_i^0)^2}{(r_i^y)^2} + \frac{(z - z_i^0)^2}{(r_i^z)^2} \tag{5}$$

are used. $r_i^x, r_i^y, r_i^z > 0$ determine the length of the semi-axes and (x_i^0, y_i^0, z_i^0) are coordinates of the centre of the i-th metaball. The choice of the ellipsoids is rational, since the distance between two neighbouring slices of CT is much bigger then the size of the pixels in one image.

The metaballs were originally introduced for a visualization of the objects without necessity of the surface mesh generation. This save both the computational time and the memory requirements, since such meshes could have several millions of triangles. However, our plug-in should have a possibility to generate high quality polygonal mesh for post-processing purposes such as generation of the STL file for a 3D print of the model.

To create a polygonal mesh from voxels the Marching Cubes method is used [9]. This method passes through the input scalar field and from the eight neighbouring points simultaneously (imaginary cube) computes the polygons which represents a corresponding part of the iso-surface. This method is time demanding and so the efficient parallel implementation is necessary. The results are depicted in Figure 9.

3.7. Visualization

For visualization, Blender Cycles engine is used, see Figure 9. Although the GPU acceleration is already implemented in Cycles, its functionality is limited. To accelerate it, we use Intel Many Integrated Core architecture (MIC).

4. Drawbacks in using Poisson reconstruction and possible solution

In terms of surface reconstruction quality, there is a significant drawback hidden in the principle of the Poisson method. The method provides satisfactory results, when reconstructing surfaces of thick objects, see Figure 10. In specific medical data, we are dealing with objects with walls as thin as 1 pixel. It is for example reconstruction of very thin bones, e.g. bones which create the orbital floor. In such cases the method can not resolve the enveloping

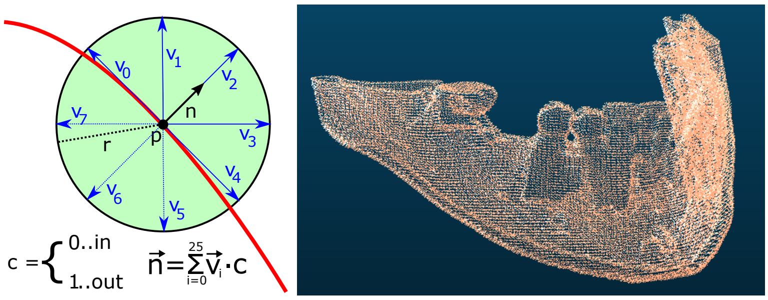

Figure 8: The enumeration of the normal $n_i = (n_x, n_y, n_z)$ from 26× vectors v_i in selected point of the boundary (left) and the result in all points of the boundary (right).

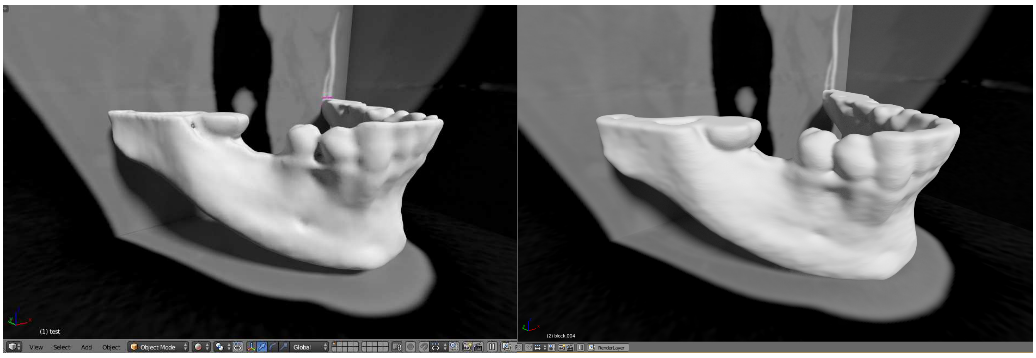

Figure 9: Comparison Poisson method (left) with Metaballs (right)

surface correctly and provides misleading or totally incorrect results. For this reason different approach capable to overcome such problem have been searched and found in application of Metaballs.

5. Comparison of Poisson reconstruction and Metaballs method

We provide qualitative comparison of Poisson and Metaballs method on two examples. First example is illustrative and describes the problem of the Poisson method while reconstructing the thin wall, see Figure 10. Effective solution to this problem is shown by Metaballs method also in Figure 10. Second example provides possible detailed reconstruction of mandible by both methods, see Figure 9.

6. Conclusion

In the paper we have introduced plug-in for Blender software. Plug-in is using powerful methods for medical image processing and 3D reconstruction of selected objects. Metaballs method has been presented as a suitable solution for 3D reconstruction of problematic parts of human body. As a show case we have provided comparison of the previously used Poisson method and Metaballs method. We have also shown possible advantages of Metaballs method over Poisson method. Due to high computational demands of selected methods extension to utilization of HPC resources is also effectively solved within the plug-in. User have several options. Use multi-core option, one node solution for the calculations or extend his work by using our Blender-client to large number of computing nodes with plenty of processing cores including Xeon Phi accelerators.

7. Future work

Our goal is to prepare an application, which can create a 3D model in real-time. To accelerate all used methods necessary to complete this task we develop the parallel version of the code.

References

[1] Sonka, M., Hlavac, V., Boyle, R.: Image Processing, Analysis and Machine Vision. Thomson, (2006)

[2] Perona, P., Malik, J.: Scale-space and Edge Detection Using Anisotropic Diffusion. In: IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 12, no. 7, pp. 629–639. (1990)

[3] Dabov, K., Foi, A., Katkovnik, V., Egiazarian, K.: Image Denoising by Sparse 3D Transform-Domain Collaborative Filtering. In: IEEE Trans. Image Process., vol. 16, no. 8, pp. 20802095, (2007).

[4] MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press. pp. 281297

[5] P. Strakos, M. Jaros, T. Karasek, L. Riha, M. Jarosova, T. Kozubek, P. Vavra, T. Jonszta, Parallelization of the Image Segmentation Algorithm for Intel Xeon Phi with Application in Medical Imaging, in P. Ivnyi, B.H.V. Topping, (Editors), Proceedings of the Fourth International Conference on

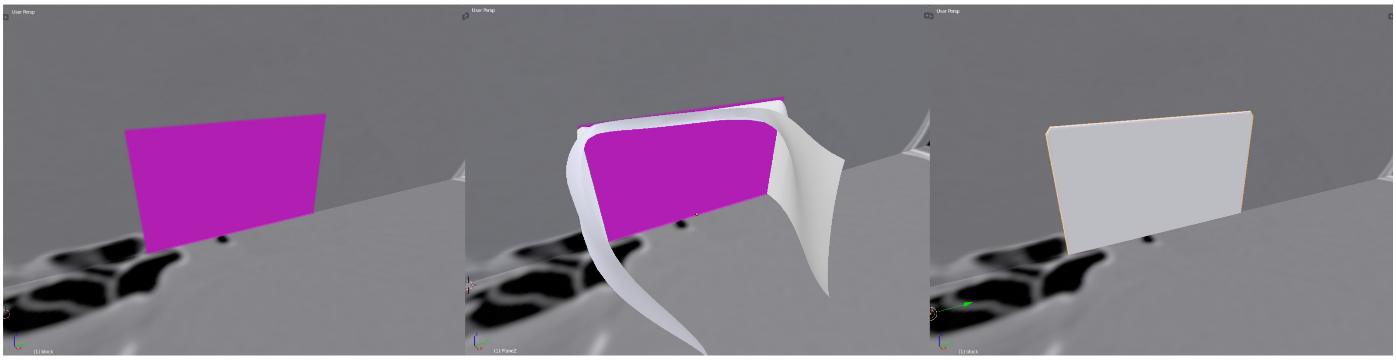

Figure 10: The reconstruction of thin surface: the segmented area (left), the reconstruction by poisson (middle), the reconstruction by metaballs

Parallel, Distributed, Grid and Cloud Computing for Engineering, Civil-Comp Press, Stirlingshire, UK, Paper 7, 2015. doi:10.4203/ccp.107.7.

[6] Michael Kazhdan, Matthew Bolitho, Hugues Hoppe. Poisson surface reconstruction, Symposium on Geometry Processing 2006, 61-70.

[7] Michael Kazhdan, Hugues Hoppe. Screened Poisson surface reconstruction, ACM Trans. Graphics, 32(3), 2013. (Presented at SIGGRAPH 2013.)

[8] James F. Blinn. 1982. A Generalization of Algebraic Surface Drawing. ACM Trans. Graph. 1, 3 (July 1982), 235-256. DOI=http://dx.doi.org/10.1145/357306.357310

[9] Lorensen W. E., Cline H. E., Marching Cubes: A high resolution 3D surface construction algorithm. Computer Graphics, Vol. 21, Nr. 4, July 1987.

[10] M. Jaros, L. Riha, T. Karasek, P. Strakos, A. Vasatova, M. Jarosova, T. Kozubek, Acceleration of Blender Cycles Path-Tracing Engine using Intel Many Integrated Core Architecture, Proceedings of the 14th International Conference on Computer Information Systems and Industrial Management Applications, Warsaw, Poland, 2015.