Slovak University of Technology in Bratislava
Department of Mathematics and Constructive Geometry

Faculty of Civil Engineering
Academic year: 2014/2015
Reg. No.: SvF-5343-7433

STU SvF

DIPLOMA THESIS TOPIC

Student: **Bc. Balázs Kósa**
Student's ID: 7433
Study programme: Mathematical and Computational Modeling
Study field: 9.1.9. applied mathematics
Thesis supervisor: prof. RNDr. Karol Mikula, DrSc.

Topic: **3D point cloud surface reconstruction by using level set methods**

Specification of Assignment:

A mathematical model and efficient and stable numerical method for surface reconstruction from unorganized 3D point cloud will be created. The method will be implemented in C language and tested on artificial and real data.

Assignment procedure from: 01. 10. 2014
Date of thesis submission: 21. 05. 2015

L. S.

Bc. Balázs Kósa
Student

prof. RNDr. Radko Mesiar, DrSc.
Head of department

prof. RNDr. Magdaléna Komorníková, PhD.
Study programme supervisor

Affidavit

I declare that I developed this master thesis by myself, only with the help of the cited literature and the assistance of my thesis supervisor.

In Bratislava May 21, 2015 .....

student's signature

Acknowledgment

I want to thank my thesis supervisor prof. RNDr. Karol Mikula, DrSc. for the help, dedicated time during consultations, expert guidance and valuable advices in the elaboration of this thesis. Special thanks to my family for their support and classmates for the common study.

Abstrakt

V rámci práce sme vytvorili matematický model a numerickú metódu na rekonštrukciu plôch z 3D mračien bodov pomocou tzv. level-set metódy. Prezentovaná metóda rieši rekonštrukciu plôch výpočtom distančnej funkcie k útvaru, ktorý je reprezentovaný mračnom bodov, s použitím tzv. Fast Sweeping Metódy a riešenia advečnej rovnice s krivostnou časťou, ktorá vytvorí evolúciu počiatočnej podmienky do finálneho stavu. Pre numerickú diskretizáciu sme navrhli novú bezpodmienečne stabilnú metódu, ktorá využíva semi-implicitnú co-volume schému pre krivostnú časť a implicitný upwind pre advektívnu časť modelu. Metóda bola naprogramovaná v jazyku C, a testovaná na reprezentatívnych príkladoch a komplexných reálnych dátach.

Abstract

In this work we created a mathematical model and numerical method for surface reconstruction from 3D point cloud data, using the level-set method. The presented method solves surface reconstruction by the calculation of the distance function to the shape, represented by the point cloud, using the so called Fast Sweeping Method, and the solution of advection equation with curvature term, which creates the evolution of an initial condition to the final state. For the numerical discretization of the model we suggested a novel unconditionally stable method, in which the semi-implicit co-volume scheme is used in curvature part and implicit upwind scheme in advective part. The method was implemented in the programming language C and tested on representative examples as well as complex real data.

Contents

1 Introduction

The aim of our work is to create a reliable numerical method which can easily create computerized 3D models from point cloud data that resembles the original object as much as possible. These type of data can be obtained by 3D scanning or by photogrammetric methods. Papers as [1, 2] have shown us that for these type of tasks the level-set method is suitable. We follow basic ideas from these papers, but we take a different approach in the solution of the partial differential equation presented here.

After we finished the theoretical deduction of our method we implemented it in the language C with the use of the programming environment of Visual Studio 2013, so we could prove that it works not only in theory but in practice as well. As you will read along you will notice several exempla pictures of results for the different sections in our work. These are direct outputs from our application processed in the freely available open-source visualization software Paraview. With the help of this software we can easily compare the initial point cloud data and our results, to confirm that our assumptions regarding this new numerical method are right.

In the following sections we will present a detailed breakdown of our method and its numerical discretization and solution as well as results which we achieved so far.

2 Mathematical formulation

The level set method, which we are using is based on the solution of the advection equation with the curvature term

$$\begin{gathered} u_t - \nabla d \cdot \nabla u - \delta \left|\nabla u\right| \nabla \cdot \left(\frac{\nabla u}{\left|\nabla u\right|}\right) = 0 \\ (x,t) \in \Omega \times [0,T] \end{gathered} \tag{2.1}$$

where $v = -\nabla d$ is the advective velocity defined by the gradient of the distance function d, the parameter $\delta \in [0,1]$ before the curvature part determines its influence to the result and Ω is the computational domain. This equation is coupled with homogeneous Neumann boundary conditions and an initial condition which we will discuss later in this paper.

3 Numerical solution

For the numerical solution of the model created from point cloud data, denoted by $\Omega_0 \subset \Omega$ and determined by equation (2.1) the following steps have to be executed. First we have to calculate the distance function to the point cloud, then we have to find a surface containing Ω_0 which will be the initial condition for the generation of the final solution of the equation. The final model will be represented by an isosurface of the calculated function $u(x)$ with value 0.5.

3.1 Calculation of the distance function

For the calculation of distance function we use the Fast sweeping method, as introduced in [3], which solves the Eikonal equation with boundary conditions which in our case has the following form

$$\begin{aligned} |\nabla d(x)| &= f(x) \quad x \in \Omega \\ d(x) &= 0 \quad x \in \Omega_0 \subset \Omega \end{aligned} \tag{3.1}$$

where $f(x) = 1$. For introducing the method we will use the following notation. $x_{i,j,k}$ will be used for the grid point of Ω , h is the size of the edges of a grid cell and $d_{i,j,k}$ denotes the numerical solution at $x_{i,j,k}$. The discretization of (3.1) at interior grid points is done according to the Godunov upwind difference scheme:

$$\begin{gathered} \left[(d_{i,j,k} - d_{x\,min})^+\right]^2 + \left[(d_{i,j,k} - d_{y\,min})^+\right]^2 + \left[(d_{i,j,k} - d_{z\,min})^+\right]^2 = h^2 \\ i = 1, ..., I-1,\ j = 1, ..., J-1,\ k = 1, ..., K-1 \\ d_{x\,min} = min\left(d_{i,j-1,k},\ d_{i,j+1,k}\right) \\ d_{y\,min} = min\left(d_{i-1,j,k},\ d_{i+1,j,k}\right) \\ d_{z\,min} = min\left(d_{i,j,k-1},\ d_{i,j,k+1}\right) \end{gathered} \tag{3.2}$$

$$(x)^+ = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

At the boundary of Ω we use one sided difference.

The initialization of $d(x)$ is done the following way. For every grid cell which contains a point from the point cloud we calculate the exact distance between the vertexes of the element and the point and set the values of $d_{i,j,k}$ to the calculated

distance. For the values to be correct we have to check if there is a smaller distance for the vertexes in the neighboring grid cells. The obtained values will be fixed in the main process of the algorithm. This way we enforce the boundary condition $d(x) = 0$ for $x \in \Omega_0 \subset \Omega$. At all the other grid points a large positive number is assigned to $d_{i,j,k}$.

After the initialization is done the algorithm continues with Gauss-Seidel iterations with alternating sweeping orderings. For each non fixed grid point $x_{i,j,k}$ we compute the solution of (3.2) from the neighboring values $d_{i,j-1,k}, d_{i,j+1,k}, d_{i-1,j,k}, d_{i+1,j,k}, d_{i,j,k-1}, d_{i,j,k+1}$ and then we update $d_{i,j,k}$ if the solution is smaller than the current value. For three dimension we sweep the computational domain with eight alternating orderings:

$$\begin{array}{ll} 1.\ i = 1:I,\ j = 1:J,\ k = 1:K & 2.\ i = I:1,\ j = 1:J,\ k = 1:K \\ 3.\ i = I:1,\ j = J:1,\ k = 1:K & 4.\ i = I:1,\ j = J:1,\ k = K:1 \\ 5.\ i = I:1,\ j = 1:J,\ k = K:1 & 6.\ i = 1:I,\ j = 1:J,\ k = K:1 \\ 7.\ i = 1:I,\ j = J:1,\ k = K:1 & 8.\ i = 1:I,\ j = J:1,\ k = 1:K \end{array} \tag{3.3}$$

The unique solution, denoted by $\bar{x}$, to the equation

$$\left[(x-a_1)^+\right]^2 + \left[(x-a_2)^+\right]^2 + \left[(x-a_3)^+\right]^2 = h^2 \tag{3.4}$$

where $a_1 = d_{x\,min},\ a_2 = d_{y\,min},\ a_3 = d_{z\,min}$ can be found as follows. We order a_1, a_2, a_3 in increasing order. For generality we assume $a_1 \le a_2 \le a_3$. There is an integer p, $1 \le p \le 3$, such that $\bar{x}$ is the unique solution that satisfies

$$(x-a_1)^2 + (x-a_2)^2 + (x-a_3)^2 = h^2 \quad and \quad a_p < \bar{x} < a_{p+1} \tag{3.5}$$

To find $\bar{x}$ we start with $p = 1$. If $\tilde{x} = a_1 + h \le a_2$ then $\bar{x} = \tilde{x}$. Otherwise we have to find the solution of the quadratic equation

$$(x-a_1)^2 + (x-a_2)^2 = h^2$$

that satisfies $\tilde{x} > a_2$. We always take the maximum of the two solutions as our $\tilde{x}$. If $\tilde{x} \le a_3$ then $\bar{x} = \tilde{x}$. If we still doesn't have a $\tilde{x}$ which satisfies all the conditions as the third step we compute the solution of the quadratic equation

$$(x-a_1)^2 + (x-a_2)^2 + (x-a_3)^2 = h^2$$

which will satisfy (3.5).

In Figure 1 we visualize the calculated distance function on the planes $z = 0$, $y = 0$ and $x = 0$, comparing it to the initial point cloud data. We can see that distance function crates surfaces with constant values in both outer and inner regions of the grid surrounding the object.

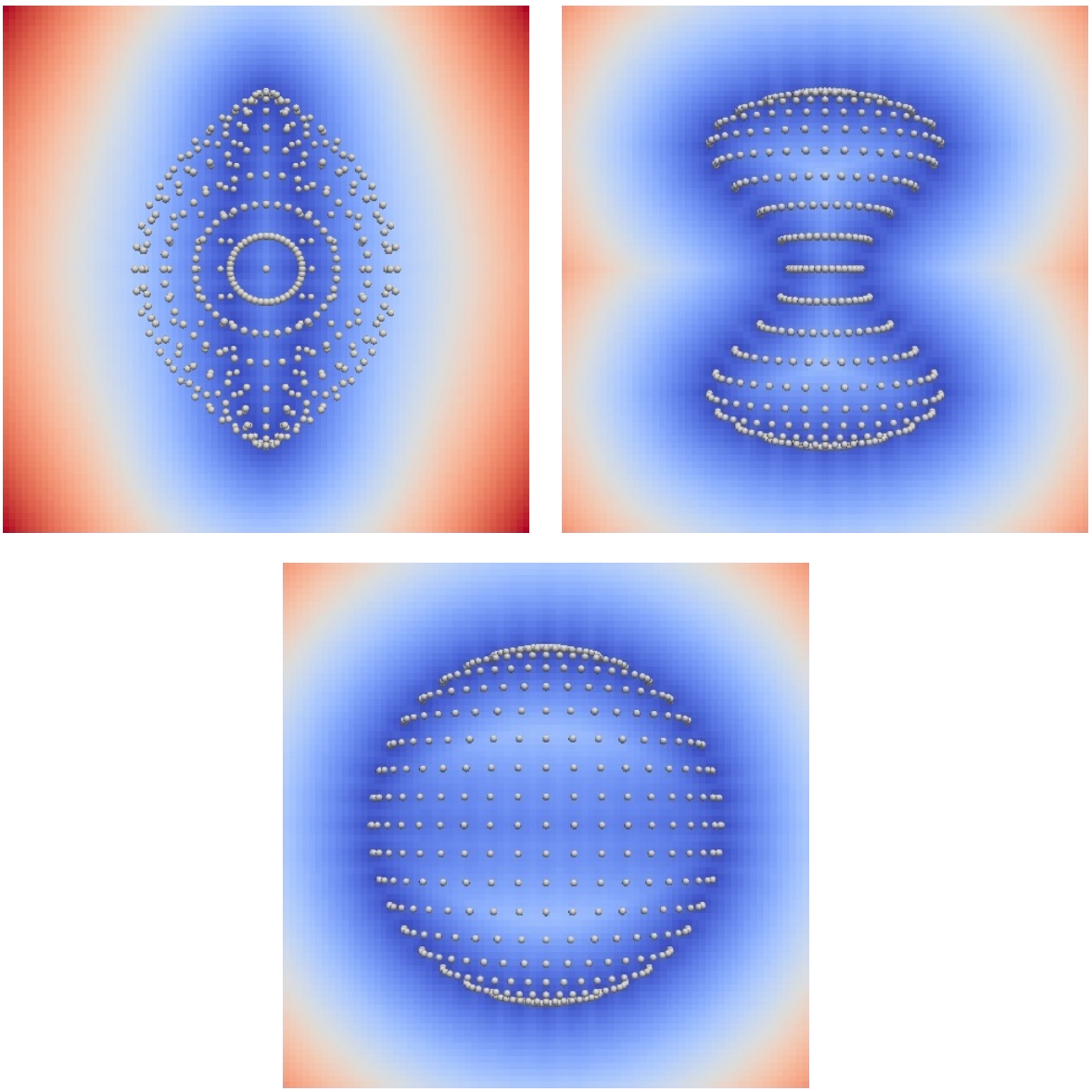

Figure 1: Distance function with point cloud data. On the top left we see the distance function on the plane $z = 0$, on the top right the plane $y = 0$ and on the bottom the plane $x = 0$.

3.2 Numerical scheme for advection equation with curvature term

Now that the distance function is calculated we proceed with the discretization of the equation (2.1). We will do this analogically to the discretization used in [4].

3.2.1 Time discretization

For the time discretization we have to choose uniform discrete time step, denoted by τ. We can replace the time derivative in (2.1) with a backward difference. Then we can formulate our semi-implicit time discretization the following way:

Let τ be a fixed number and u^0 the initial surface of our model. Then at every discrete time $t_n = n\tau,\ n = 1, ..., N$ we search for the function u^n as the solution of the equation

$$\frac{u^n - u^{n-1}}{\tau} - \nabla d \cdot \nabla u^n - \delta \left|\nabla u^{n-1}\right| \nabla \cdot \left(\frac{\nabla u^n}{|\nabla u^{n-1}|} \right) = 0 \tag{3.6}$$

3.2.2 Spatial discretization

Our model consists of a 3D grid, which is built of voxels with cubic shape and an edge size h. We will interpret the spatial discretization of the level set function u as the numerical values $u_{i,j,k}$ at the voxel centers. In order to easily calculate the gradient of the level-set equation $|\nabla u^{n-1}|$ in every time step of (3.6) we induct a 3D tetrahedral grid into the voxel structure and take a piecewise linear approximation of $u\,(x)$ on such a grid. This way we obtain a constant value of the gradient for each tetrahedron, by which we can construct a simple and clear fully discrete system of equations.

The 3D tetrahedral finite element grid is created with the following approach. Every voxel is divided into six pyramid shaped elements with base surface given by the voxel's walls and vertex by the voxel center. Each one of these pyramids is joined with the neighboring pyramids with whom they have a mutual base surface. These newly formed octahedrons are then split into four tetrahedrons as seen in Figure 2. In our new grid $\mathcal{T}_h$ the level-set function will be updated only at the centers of the voxels, they will represent so called degree of freedom (DF) nodes.

For the tetrahedral grid we construct a co-volume mesh, which will consist of cells p associated only with DF nodes of $\mathcal{T}_h$. We denote all neighboring DF nodes q of p

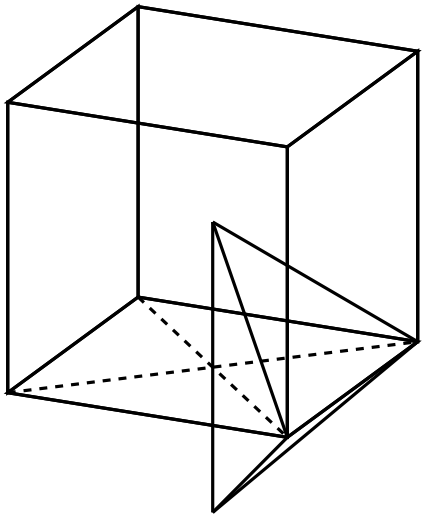

Figure 2: Our initial voxel grid cell with a tetrahedral grid cell

with C_p. The DF nodes q are all connected to the DF node p by a mutual edge of four tetrahedrons, which is denoted by σ_{pq} with the length h_{pq}. Each co-volume p is bounded by a plane for every $q \in C_p$ which is perpendicular to σ_{pq} and is denoted by e_{pq}. The set of tetrahedrons which have σ_{pq} are denoted by ε_{pq}. For every $T \in \varepsilon_{pq}$ c_{pq}^T is the area of the intersection of e_{pq} and T. N_p will be the set of tetrahedrons that have DF node p as a vertex. On this grid u_h will be a piecewise linear function. Then we can use the notation $u_p = u_h(x_p)$, where x_p denotes the coordinates of DF node p.

Now that we have all the notations which are needed we can begin the derivation of the spatial discretization of (3.6). We will do this by using the following modified form of the equation:

$$\frac{u^n - u^{n-1}}{\tau} + v \cdot \nabla u^n = \delta \left|\nabla u^{n-1}\right| \nabla \cdot \left(\frac{\nabla u^n}{|\nabla u^{n-1}|}\right) \tag{3.7}$$

where $v = -\nabla d$.

As the first step we will integrate (3.7) over every co-volume p.

$$\int_p \frac{u^n - u^{n-1}}{\tau} dx + \int_p v \cdot \nabla u^n dx = \int_p \delta \left|\nabla u^{n-1}\right| \nabla \cdot \left(\frac{\nabla u^n}{|\nabla u^{n-1}|}\right) dx \tag{3.8}$$

For the first part of left hand side of (3.8) we get the approximation in the form

$$\int_p \frac{u^n - u^{n-1}}{\tau} dx = m(p) \frac{u_p^n - u_p^{n-1}}{\tau} \tag{3.9}$$

where $m(p)$ is a measure in $\mathbb{R}^d$ of the co-volume p. To derive the second part of the left hand side we use the following approach. This part can be written in an equivalent

form by

$$v \cdot \nabla u = \nabla (uv) - u \nabla \cdot v$$

$$\int_p v \cdot \nabla u^n dx = \int_p \nabla (u^n v)\, dx - \int_p u^n \nabla \cdot v\, dx$$

Using the divergence theorem we get

$$\begin{aligned} &\int_p \nabla (u^n v)\, dx - \int_p u^n \nabla \cdot v\, dx = \int_{\partial p} u^n v \cdot n\, d\sigma - u_p^n \int_{\partial p} v \cdot n\, d\sigma = \\ &\sum_{q \in C_p} u_{pq}^n \int_{e_{pq}} v \cdot n\, d\sigma - \sum_{q \in C_p} u_p^n \int_{e_{pq}} v \cdot n\, d\sigma \end{aligned} \tag{3.10}$$

where u_{pq}^n is the value of the level-set function on the surface e_{pq}. We substitute $\int_{e_{pq}} v \cdot n\, d\sigma$ in (3.10) with v_{pq} which by solving the integral will have the value $v_{pq} = h_{pq}^2 v \cdot n$. By this we finally get

$$\int_p v \cdot \nabla u^n dx = \sum_{q \in C_p} v_{pq} \left(u_{pq}^n - u_p^n \right) \tag{3.11}$$

In the upwind approach the set C_p can be divided into $C_p = C_p^{in} \cup C_p^{out}$, where $C_p^{in} = \{q \in C_p,\, v_{pq} < 0\}$ which consists of the inflow boundaries and $C_p^{out} = \{q \in C_p,\, v_{pq} > 0\}$ consisting of the outflow boundaries. By dividing C_p we can set the values u_{pq}^n to u_q^n if $q \in C_p^{in}$ and to u_p^n if $q \in C_p^{out}$. After these modification we can rewrite the sum right hand side of (3.11) to

$$\sum_{q \in C_p} v_{pq} \left(u_{pq}^n - u_p^n \right) = \sum_{q \in C_p^{in}} v_{pq} \left(u_q^n - u_p^n \right) + \sum_{q \in C_p^{out}} v_{pq} \left(u_p^n - u_p^n \right)$$

As we see the outflow part will be zero and after we rewrite the inflow part for simpler implementation we get the final form of the second part of the left hand side of the equation (3.8)

$$\int_p v \cdot \nabla u^n dx = \sum_{q \in C_p} min\left(v_{pq}, 0\right) \left(u_q^n - u_p^n \right) \tag{3.12}$$

Now what remains is the discretization of the right hand side of (3.8). Again we use the divergence theorem to get

$$\int_p \delta \left| \nabla u^{n-1} \right| \nabla \cdot \left(\frac{\nabla u^n}{|\nabla u^{n-1}|} \right) dx = \delta \left| \nabla u_p^{n-1} \right| \sum_{q \in C_p} \int_{e_{pq}} \frac{1}{|\nabla u^{n-1}|} \frac{\partial u^n}{\partial n} d\sigma \tag{3.13}$$

The integral part $\int_{e_{pq}} \frac{1}{|\nabla u^{n-1}|} \frac{\partial u^n}{\partial n} d\sigma$ and $\left| \nabla u_p^{n-1} \right|$ from (3.13) will be approximated numerically using piecewise linear reconstruction of u^{n-1} on the tetrahedral grid $\mathcal{T}_h$, thus we get

$$\delta \left| \nabla u_p^{n-1} \right| \sum_{q \in C_p} \left(\sum_{T \in \varepsilon_{pq}} c_{pq}^T \frac{1}{\left| \nabla u_T^{n-1} \right|} \right) \frac{u_q^n - u_p^n}{h_{pq}}$$

$$M_p = \left|\nabla u_p^{n-1}\right| = \sum_{T \in N_p} \frac{m\left(T \cap p\right)}{m\left(p\right)} \left|\nabla u_T^{n-1}\right|$$

and the final form of the equation (3.7) will be

$$\begin{aligned} m\left(p\right) \frac{u_p^n - u_p^{n-1}}{\tau} + \sum_{q \in C_p} min\left(v_{pq}, 0\right) \left(u_q^n - u_p^n\right) = \\ \delta\, M_p \sum_{q \in C_p} \left(\sum_{T \in \varepsilon_{pq}} c_{pq}^T \frac{1}{\left|\nabla u_T^{n-1}\right|} \right) \frac{u_q^n - u_p^n}{h_{pq}} \end{aligned} \tag{3.14}$$

From this form we are able to derive the system of linear equations which we will solve at every time step. For the linear equations we will define the regularized gradients by

$$\left|\nabla u_T\right|_\varepsilon = \sqrt{\varepsilon^2 + \left|\nabla u_T\right|^2} \tag{3.15}$$

After we arrange all the parts of the equation (3.14) we get the following coefficients

$$a_{pq}^{n-1} = \left(min\left(v_{pq}, 0\right) - \delta\, M_p \frac{1}{h_{pq}} \sum_{T \in \varepsilon_{pq}} c_{pq}^T \frac{1}{\left|\nabla u_T^{n-1}\right|_\varepsilon} \right) \tag{3.16}$$

thus we can formulate our semi-implicit co-volume scheme:

Let $u_p^0,\ p = 1, ..., M$ be given discrete initial values of the level-set function. Then, for $n = 1, ..., N$ we look for $u_p^n,\ p = 1, ..., M$, satisfying

$$u_p^n + \frac{\tau}{m\left(p\right)} \sum_{q \in N_p} a_{pq}^{n-1} \left(u_q^n - u_p^n\right) = u_p^{n-1} \tag{3.17}$$

With addition of the homogeneous Neumann boundary conditions to our fully discrete scheme we obtain a system of linear equations for which we can declare the following statement.

Theorem. *There exists unique solution* $(u_1^n, ..., u_M^n)$ *of* (3.17) *for any* $\tau > 0$, $\varepsilon > 0$, *and for every* $n = 1, ..., N$. *The system matrix is a strictly diagonally dominant M-matrix. For any* $\tau > 0$, $\varepsilon > 0$, *the following* L_∞ *stability holds:*

$$\min_p\ u_p^0 \leq \min_p\ u_p^n \leq \max_p\ u_p^n \leq \max_p\ u_p^0,\ \ 1 \leq n \leq N. \tag{3.18}$$

Proof. First we will prove the inequality for the maximum. Let $u_{p\,max}^n$ be the maximum of the time step n achieved in the DF node p. If we appoint this value to the equation (3.17) we get:

$$u_{p\,max}^n + \frac{\tau}{m\left(p\right)} \sum_{q \in N_p} a_{pq}^{n-1} \left(u_q^n - u_{p\,max}^n\right) = u_p^{n-1}$$

. Since the coefficient $a_{pq}^{n-1} \leq 0$ and $u_q^n \leq u_{p\,max}^n$, thus $\left(u_q^n - u_{p\,max}^n\right)$ is either zero or negative, the second part of the left hand side is non-negative. Therefore $u_{p\,max}^n \leq u_p^{n-1}$, so we can write

$$\max_p \; u_p^n \leq \max_p \; u_p^{n-1} \tag{3.19}$$

To prove the inequality for the minimum, we can apply the same argumentation. If we appoint value $u_{p\,min}^n = \min\left(u_1^n, ..., u_M^n\right)$ to (3.17) the second part will be non-positive and $u_p^{n-1} \leq u_{p\,min}^n$, thus we can write that

$$\min_p \; u_p^{n-1} \leq \min_p \; u_p^n \tag{3.20}$$

If we apply (3.19) and (3.20) to every time step $1 \leq n \leq N$ we get

$$\min_p \; u_p^0 \leq \min_p \; u_p^n \leq \max_p \; u_p^n \leq \max_p \; u_p^0$$

by which we proved our theorem.

The number of time steps N is determined by the difference of the solution in the current and the previous time step in discrete L^2 norm. The computation is stopped if this difference is less than the prescribed tolerance, which we usually set to 10^{-6}. Then the stopping time $T = N\tau$.

3.3 Calculating the coefficients of the linear system

Now that we formulated our scheme for the easy replicability and simplicity of implementation we will write the co-volume scheme in a "finite-difference notation". We will associate our 3D co-volume p with the index triplet (i, j, k), where i represents the y axis, j the x axis and k the z axis. Analogically the values u_p^n will be associated with $u_{i,j,k}^n$. In each co-volume p, the set N_p consist of 24 tetrahedrons on which we will compute absolute value of gradient denoted by $G_{i,j,k}^l,\ l = 1, ..., 24$. Furthermore to keep the formulas as comprehensible as possible we will introduce a notation seen in Figure 3.

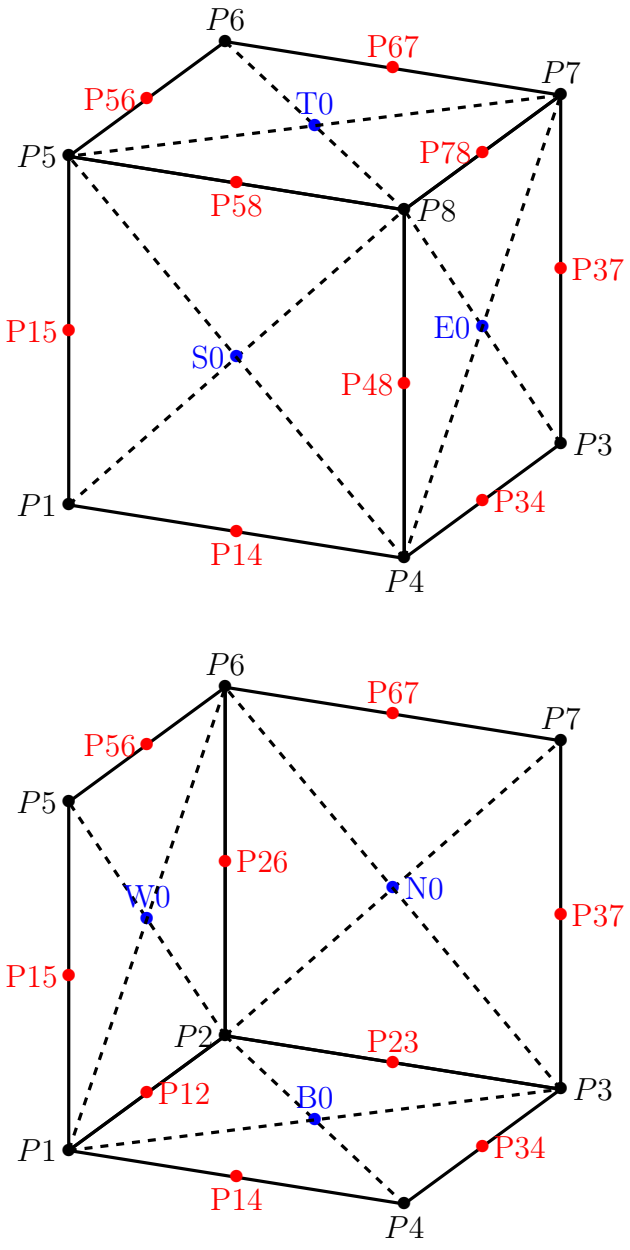

Figure 3: Notation for the additional points of a grid cell used for the easier formulation of the coefficient computation

Now we will explain what these new symbols mean and how their values are computed. For every vertex of a square cubic element we use $P1, ..., P8$ to denote the average values of u_n at these points, which are calculated the following way:

$$P1 = \tfrac{1}{8}\left(u_{i,j,k} + u_{i,j-1,k} + u_{i-1,j,k} + u_{i-1,j-1,k} + u_{i,j,k-1} + u_{i,j-1,k-1} + u_{i-1,j,k-1} + u_{i-1,j-1,k-1}\right)$$
$$P2 = \tfrac{1}{8}\left(u_{i,j,k} + u_{i,j-1,k} + u_{i+1,j,k} + u_{i+1,j-1,k} + u_{i,j,k-1} + u_{i,j-1,k-1} + u_{i+1,j,k-1} + u_{i+1,j-1,k-1}\right)$$
$$P3 = \tfrac{1}{8}\left(u_{i,j,k} + u_{i,j+1,k} + u_{i+1,j,k} + u_{i+1,j+1,k} + u_{i,j,k-1} + u_{i,j+1,k-1} + u_{i+1,j,k-1} + u_{i+1,j+1,k-1}\right)$$
$$P4 = \tfrac{1}{8}\left(u_{i,j,k} + u_{i,j+1,k} + u_{i-1,j,k} + u_{i-1,j+1,k} + u_{i,j,k-1} + u_{i,j+1,k-1} + u_{i-1,j,k-1} + u_{i-1,j+1,k-1}\right)$$
$$P5 = \tfrac{1}{8}\left(u_{i,j,k} + u_{i,j-1,k} + u_{i-1,j,k} + u_{i-1,j-1,k} + u_{i,j,k+1} + u_{i,j-1,k+1} + u_{i-1,j,k+1} + u_{i-1,j-1,k+1}\right)$$
$$P6 = \tfrac{1}{8}\left(u_{i,j,k} + u_{i,j-1,k} + u_{i+1,j,k} + u_{i+1,j-1,k} + u_{i,j,k+1} + u_{i,j-1,k+1} + u_{i+1,j,k+1} + u_{i+1,j-1,k+1}\right)$$
$$P7 = \tfrac{1}{8}\left(u_{i,j,k} + u_{i,j+1,k} + u_{i+1,j,k} + u_{i+1,j+1,k} + u_{i,j,k+1} + u_{i,j+1,k+1} + u_{i+1,j,k+1} + u_{i+1,j+1,k+1}\right)$$
$$P8 = \tfrac{1}{8}\left(u_{i,j,k} + u_{i,j+1,k} + u_{i-1,j,k} + u_{i-1,j+1,k} + u_{i,j,k+1} + u_{i,j+1,k+1} + u_{i-1,j,k+1} + u_{i-1,j+1,k+1}\right)$$

We will also need the average values between each of these points for which we use these notations:

$$P12 = \frac{1}{2}(P1 + P2) \quad P14 = \frac{1}{2}(P1 + P4) \quad P15 = \frac{1}{2}(P1 + P5)$$
$$P23 = \frac{1}{2}(P2 + P3) \quad P26 = \frac{1}{2}(P2 + P6) \quad P34 = \frac{1}{2}(P3 + P4)$$
$$P37 = \frac{1}{2}(P3 + P7) \quad P48 = \frac{1}{2}(P4 + P8) \quad P56 = \frac{1}{2}(P5 + P6)$$
$$P58 = \frac{1}{2}(P5 + P8) \quad P67 = \frac{1}{2}(P6 + P7) \quad P78 = \frac{1}{2}(P7 + P8)$$

The values at the points where the edges σ_p of the tetrahedral elements intersects the plains e_{pq}, for every $q \in C_p$, are marked as $N0, S0, E0, W0$ for the corresponding cardinal directions and $B0, T0$ for bottom and top. These values are calculated as the average between two neighboring co-volumes p.

$$N0 = \frac{1}{2}\left(u_{i,j,k} + u_{i+1,j,k}\right) \quad S0 = \frac{1}{2}\left(u_{i,j,k} + u_{i-1,j,k}\right)$$
$$E0 = \frac{1}{2}\left(u_{i,j,k} + u_{i,j+1,k}\right) \quad W0 = \frac{1}{2}\left(u_{i,j,k} + u_{i,j-1,k}\right)$$
$$T0 = \frac{1}{2}\left(u_{i,j,k} + u_{i,j,k+1}\right) \quad B0 = \frac{1}{2}\left(u_{i,j,k} + u_{i,j,k-1}\right)$$

With these new points we are ready to derive the values $G^l_{i,j,k},\ l = 1, ..., 24$ with some simple equations.

Generally the $G^l_{i,j,k}$ can be calculated the following way

$$G^l_{i,j,k} = \sqrt{\left(\frac{\partial u_{i,j,k}}{\partial x}\right)^2 + \left(\frac{\partial u_{i,j,k}}{\partial y}\right)^2 + \left(\frac{\partial u_{i,j,k}}{\partial z}\right)^2} \tag{3.21}$$

where the derivatives are calculated on the l th tetrahedron of the co-volume p. According to this formula and Figure 3 for the tetrahedrons intersected by the bottom surface of a voxel grid cell we get:

$$dB = \frac{u_{i,j,k} - u_{i,j,k-1}}{h}$$

$$G^1_{i,j,k} = \sqrt{\left(\frac{P2 - P1}{h}\right)^2 + \left(\frac{B0 - P12}{0.5h}\right)^2 + dB^2}$$

$$G^2_{i,j,k} = \sqrt{\left(\frac{B0 - P14}{0.5h}\right)^2 + \left(\frac{P4 - P1}{h}\right)^2 + dB^2}$$

$$G^3_{i,j,k} = \sqrt{\left(\frac{P3 - P4}{h}\right)^2 + \left(\frac{P34 - B0}{0.5h}\right)^2 + dB^2}$$

$$G^4_{i,j,k} = \sqrt{\left(\frac{P23 - B0}{0.5h}\right)^2 + \left(\frac{P3 - P2}{h}\right)^2 + dB^2}$$

Analogically for the top surface

$$dT = \frac{u_{i,j,k+1} - u_{i,j,k}}{h}$$

$$G^5_{i,j,k} = \sqrt{\left(\frac{P6 - P5}{h}\right)^2 + \left(\frac{T0 - P56}{0.5h}\right)^2 + dT^2}$$

$$G^6_{i,j,k} = \sqrt{\left(\frac{T0 - P58}{0.5h}\right)^2 + \left(\frac{P8 - P5}{h}\right)^2 + dT^2}$$

$$G^7_{i,j,k} = \sqrt{\left(\frac{P7 - P8}{h}\right)^2 + \left(\frac{P78 - T0}{0.5h}\right)^2 + dT^2}$$

$$G^8_{i,j,k} = \sqrt{\left(\frac{P67 - T0}{0.5h}\right)^2 + \left(\frac{P7 - P6}{h}\right)^2 + dT^2}\ ,$$

for the north wall

$$dN = \frac{u_{i+1,j,k} - u_{i,j,k}}{h}$$

$$G^{9}_{i,j,k} = \sqrt{dN^2 + \left(\frac{P7 - P6}{h}\right)^2 + \left(\frac{P67 - N0}{0.5h}\right)^2}$$

$$G^{10}_{i,j,k} = \sqrt{dN^2 + \left(\frac{P37 - N0}{0.5h}\right)^2 + \left(\frac{P7 - P3}{h}\right)^2}$$

$$G^{11}_{i,j,k} = \sqrt{dN^2 + \left(\frac{P3 - P2}{h}\right)^2 + \left(\frac{N0 - P23}{0.5h}\right)^2}$$

$$G^{12}_{i,j,k} = \sqrt{dN^2 + \left(\frac{N0 - P26}{0.5h}\right)^2 + \left(\frac{P6 - P2}{h}\right)^2},$$

the south wall

$$dS = \frac{u_{i,j,k} - u_{i-1,j,k}}{h}$$

$$G^{13}_{i,j,k} = \sqrt{dS^2 + \left(\frac{P8 - P5}{h}\right)^2 + \left(\frac{P58 - S0}{0.5h}\right)^2}$$

$$G^{14}_{i,j,k} = \sqrt{dS^2 + \left(\frac{P48 - S0}{0.5h}\right)^2 + \left(\frac{P8 - S4}{h}\right)^2}$$

$$G^{15}_{i,j,k} = \sqrt{dS^2 + \left(\frac{P4 - P1}{h}\right)^2 + \left(\frac{S0 - P14}{0.5h}\right)^2}$$

$$G^{16}_{i,j,k} = \sqrt{dS^2 + \left(\frac{S0 - P15}{0.5h}\right)^2 + \left(\frac{P5 - P1}{h}\right)^2},$$

the east wall

$$dE = \frac{u_{i,j+1,k} - u_{i,j,k}}{h}$$

$$G^{17}_{i,j,k} = \sqrt{\left(\frac{P3 - P4}{h}\right)^2 + dE^2 + \left(\frac{E0 - P34}{0.5h}\right)^2}$$

$$G^{18}_{i,j,k} = \sqrt{\left(\frac{P37 - E0}{0.5h}\right)^2 + dE^2 + \left(\frac{P7 - P3}{h}\right)^2}$$

$$G^{19}_{i,j,k} = \sqrt{\left(\frac{P8 - P7}{h}\right)^2 + dE^2 + \left(\frac{P78 - E0}{0.5h}\right)^2}$$

$$G^{20}_{i,j,k} = \sqrt{\left(\frac{E0 - P48}{0.5h}\right)^2 + dE^2 + \left(\frac{P8 - P4}{h}\right)^2},$$

the west wall

$$dw = \frac{u_{i,j,k} - u_{i,j-1,k}}{h}$$

$$G^{21}_{i,j,k} = \sqrt{\left(\frac{P2 - P1}{h}\right)^2 + dW^2 + \left(\frac{W0 - P12}{0.5h}\right)^2}$$

$$G^{22}_{i,j,k} = \sqrt{\left(\frac{P26 - W0}{0.5h}\right)^2 + dW^2 + \left(\frac{P6 - P2}{h}\right)^2}$$

$$G^{23}_{i,j,k} = \sqrt{\left(\frac{P6 - P5}{h}\right)^2 + dW^2 + \left(\frac{P56 - W0}{0.5h}\right)^2}$$

$$G^{24}_{i,j,k} = \sqrt{\left(\frac{W0 - P15}{0.5h}\right)^2 + dW^2 + \left(\frac{P5 - P1}{h}\right)^2}$$

Now that we can calculate $\left|\nabla u_T^{n-1}\right|_\varepsilon$ from (3.16) we will determine the values v_{pq}. As mentioned earlier $v_{pq} = h^2_{pq} v \cdot n$ and with $v = -\nabla d$ by evaluating for the six directions, we get

$$vt_{i,j,k} = -h\left(d_{i,j,k+1} - d_{i,j,k}\right), \quad vb_{i,j,k} = h\left(d_{i,j,k} - d_{i,j,k-1}\right)$$

$$vn_{i,j,k} = -h\left(d_{i+1,j,k} - d_{i,j,k}\right), \quad vs_{i,j,k} = h\left(d_{i,j,k} - d_{i-1,j,k}\right)$$

$$ve_{i,j,k} = -h\left(d_{i,j+1,k} - d_{i,j,k}\right), \quad vw_{i,j,k} = h\left(d_{i,j,k} - d_{i,j-1,k}\right)$$

Then we can construct the coefficients

$$b_{i,j,k} = \frac{\tau}{h^3}\left(min\left(vb_{i,j,k}, 0\right) - \delta\, M_{i,j,k} \frac{h}{4} \sum_{l=1}^{4} \frac{1}{\sqrt{\varepsilon^2 + \left(G^l_{i,j,k}\right)^2}}\right)$$

$$t_{i,j,k} = \frac{\tau}{h^3}\left(min\left(vt_{i,j,k}, 0\right) - \delta\, M_{i,j,k} \frac{h}{4} \sum_{l=5}^{8} \frac{1}{\sqrt{\varepsilon^2 + \left(G^l_{i,j,k}\right)^2}}\right)$$

$$n_{i,j,k} = \frac{\tau}{h^3}\left(min\left(vn_{i,j,k}, 0\right) - \delta\, M_{i,j,k} \frac{h}{4} \sum_{l=9}^{12} \frac{1}{\sqrt{\varepsilon^2 + \left(G^l_{i,j,k}\right)^2}}\right)$$

$$s_{i,j,k} = \frac{\tau}{h^3}\left(min\left(vs_{i,j,k}, 0\right) - \delta\, M_{i,j,k} \frac{h}{4} \sum_{l=13}^{16} \frac{1}{\sqrt{\varepsilon^2 + \left(G^l_{i,j,k}\right)^2}}\right)$$

$$e_{i,j,k} = \frac{\tau}{h^3}\left(min\left(ve_{i,j,k}, 0\right) - \delta\, M_{i,j,k} \frac{h}{4} \sum_{l=17}^{20} \frac{1}{\sqrt{\varepsilon^2 + \left(G^l_{i,j,k}\right)^2}}\right)$$

$$w_{i,j,k} = \frac{\tau}{h^3}\left(min\left(vw_{i,j,k}, 0\right) - \delta\, M_{i,j,k} \frac{h}{4} \sum_{l=21}^{24} \frac{1}{\sqrt{\varepsilon^2 + \left(G^l_{i,j,k}\right)^2}}\right)$$

where $M_{i,j,k}$ is

$$M_{i,j,k} = \sqrt{\varepsilon^2 + \left(\frac{1}{24}\sum_{l=1}^{24} G^l_{i,j,k}\right)^2}$$

and we define the diagonal coefficients as

$$c_{i,j,k} = 1 - b_{i,j,k} - t_{i,j,k} - n_{i,j,k} - w_{i,j,k} - s_{i,j,k} - e_{i,j,k}$$

so we can define for DF node corresponding to (i, j, k) the equation

$$\begin{aligned} &c_{i,j,k}u^n_{i,j,k} + b_{i,j,k}u^n_{i,j,k-1} + t_{i,j,k}u^n_{i,j,k+1} + n_{i,j,k}u^n_{i+1,j,k} \\ &+s_{i,j,k}u^n_{i-1,j,k} + e_{i,j,k}u^n_{i,j+1,k} + w_{i,j,k}u^n_{i,j-1,k} = u^{n-1}_{i,j+1,k} \end{aligned} \tag{3.22}$$

When we collect the equations for all DF nodes and take into account Neumann boundary conditions we get the linear system which we have to solve. For the solution of this system we choose the SOR (Successive Over Relaxation) iterative method. We start the iterations by setting $u^n_{i,j,k} = u^{n-1}_{i,j,k}$, then in every iteration $l = 1, ...$ we use the following two step procedure:

$$\begin{aligned} Y = (&u^{n(0)}_{i,j,k} - b_{i,j,k}u^{n(l)}_{i,j,k-1} - t_{i,j,k}u^{n(l-1)}_{i,j,k+1} - n_{i,j,k}u^{n(l-1)}_{i+1,j,k} \\ &-s_{i,j,k}u^{n(l)}_{i-1,j,k} - e_{i,j,k}u^{n(l-1)}_{i,j+1,k} - w_{i,j,k}u^{n(l)}_{i,j-1,k})/c_{i,j,k} \end{aligned}$$

$$u^{n(l)}_{i,j,k} = u^{n(l-1)}_{i,j,k} + \omega\left(Y - u^{n(l-1)}_{i,j,k}\right)$$

We define squared L_2 norm of residuum at current iteration by

$$\begin{aligned} R_l = \sum_{i,j,k}(&c_{i,j,k}u^{n(l)}_{i,j,k} + b_{i,j,k}u^{n(l)}_{i,j,k-1} + t_{i,j,k}u^{n(l)}_{i,j,k+1} + n_{i,j,k}u^{n(l)}_{i+1,j,k} \\ &+s_{i,j,k}u^{n(l)}_{i-1,j,k} + e_{i,j,k}u^{n(l)}_{i,j+1,k} + w_{i,j,k}u^{n(l)}_{i,j-1,k} - u^{n(0)}_{i,j,k})^2 \end{aligned}$$

The iterative process is stopped if $R^l < TOL$.

3.4 Calculation of the initial condition

As mentioned before, this method needs an initial condition $u^0(x)$, which will be deformed to get the solution, that is the final form of the model. Theoretically any initial surface that contains the point cloud data set could be used, but an optimal initial guess is crucial for the efficiency of the method. We can find this optimal surface by identifying all the points for which the value of the distance function is greater or equal to a parameter β. For simplicity let us call these points, exterior points. To find all these points we will use the following algorithm:

- Mark all points on the borders of the grid as exterior and add them to the set E.
- For every point in the set E check all neighboring points in the grid.
- If the neighboring point isn't an exterior point and his distance from the point cloud is greater or equal to β add it to end of set E and mark as exterior.
- Continue until you get to the last point of E.

When we found all the exterior point we set $u^0(x)$ to be equal 0 at all exterior point and 1 at all the other points. With this approach we can find an initial surface close to the final shape as seen on the Figure 4.

The right choice of the parameter β is very important for finding the optimal surface. Theoretically we want to get an initial surface as close to the point cloud as possible, so we would choose a very small β. Our choice could lead to a problem seen on Figure 5, which is a direct result of the discontinuity in our data set. Because β was too small our algorithm couldn't find a continuous surface. The right choice of β depends on the density of the data end and the selected grid size.

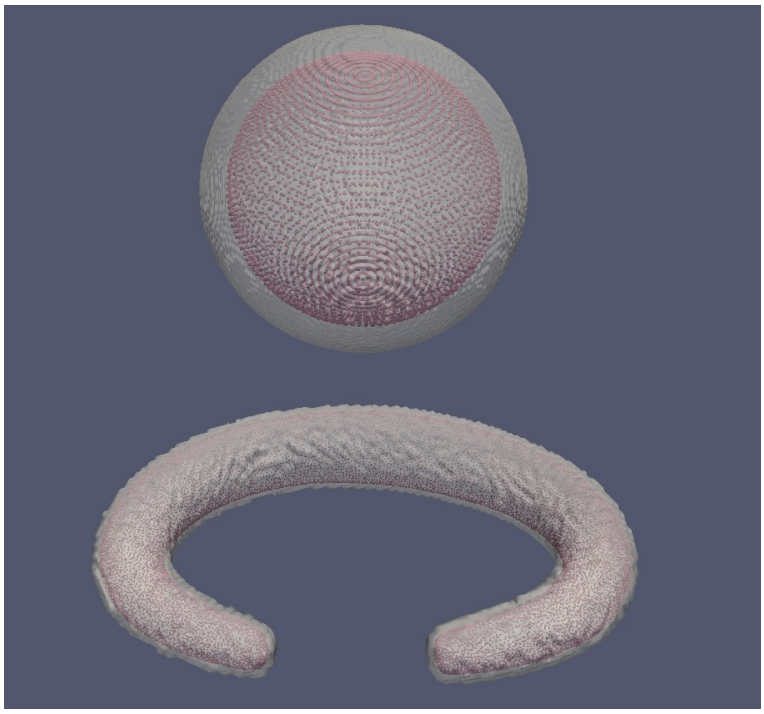

Figure 4: Examples for the initial condtition used in our method

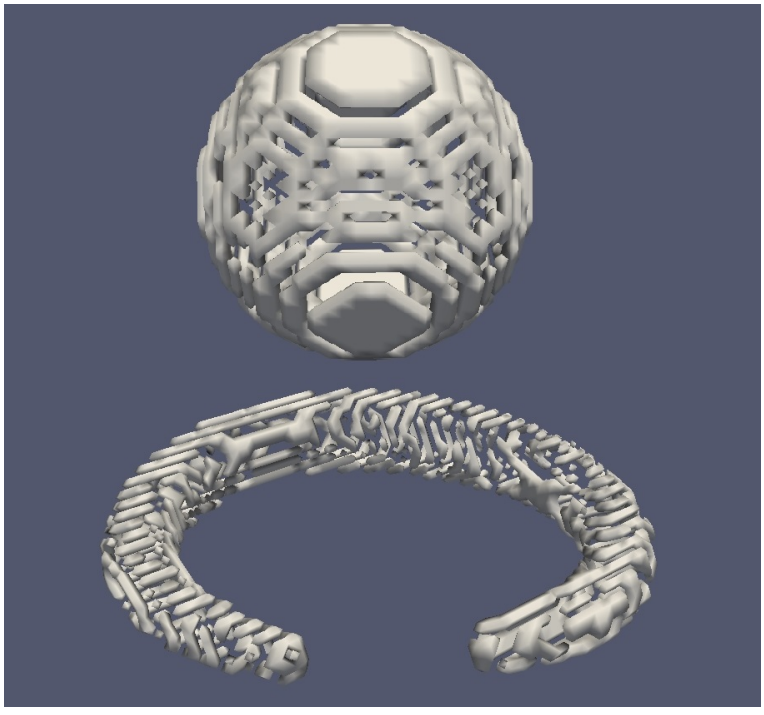

Figure 5: Initial condtition calculated with wrong choice of β

4 Numerical results

After we implemented the method in the programming language C we tested it on representative examples as well as real data. In this section we present some of our results. These examples are a good display of the quality of our method.

Figure 6 and 7 illustrate test examples. These were used for the verification of the correct behavior of our method during the implementation phase. The point cloud data was generated with the corresponding parametric equations of the objects. These representative examples where created on a grid containing 80^3 cells. We can see that for these tests with such a sparse grid we already got good results.

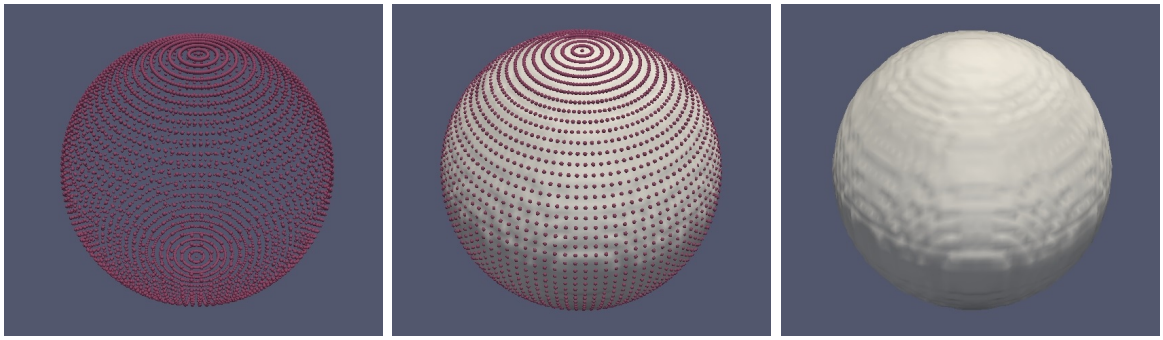

Figure 6: First test object. On the left we see the point cloud data, in the middle the point cloud with the final model and on the right the final model only.

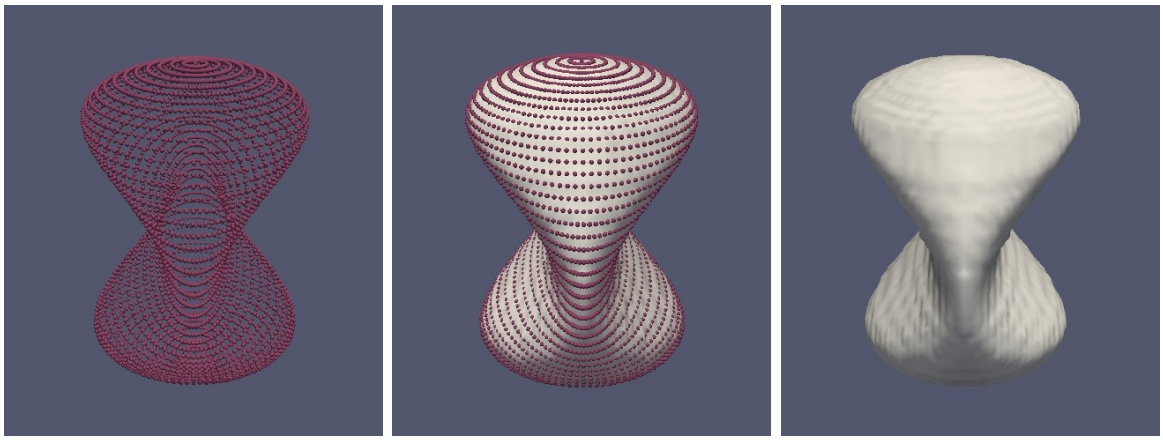

Figure 7: Second test object. On the left we see the point cloud data, in the middle the point cloud with the final model and on the right the final model only.

On Figure 8 and 9 we can see real life data. These items where archaeological finds and the point cloud scans were provided by Jana Haličková from the Monuments Board

of the Slovak republic to which we express our great thanks. On Figure 8 we can see a bracelet. The model was calculated on a grid with 160^3 cells. On Figure 9 we can see a sealer. The model was calculated on a grid with 320^3 cells. This model has a very interesting surface structure, which confirms the accuracy of the method. Figure 11 shows an angel statue with the numerical results calculated on a 400^3 grid.

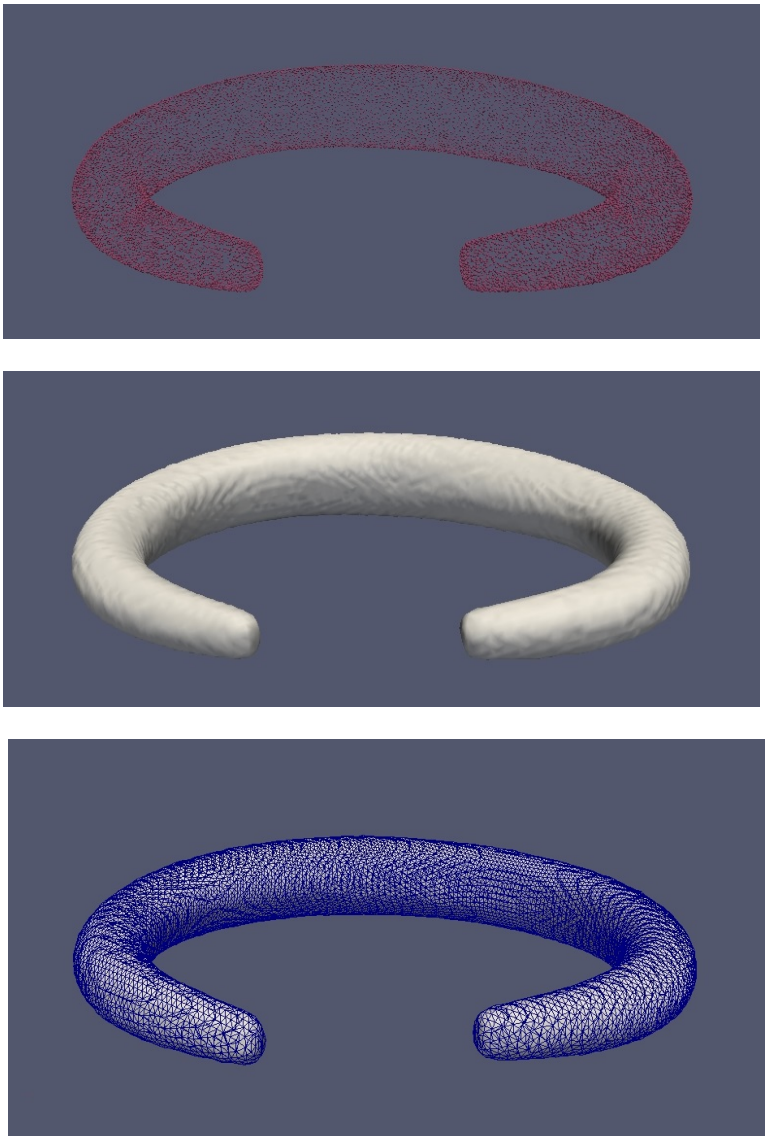

Figure 8: Archaeological finds: bracelet. On the top we see the point cloud data, in the middle our final result and on the bottom the final result with triangulated surface.

Figure 9: Archaeological finds: sealer. On the left we see the point cloud data, in the middle and the right the final result from different viewpoints.

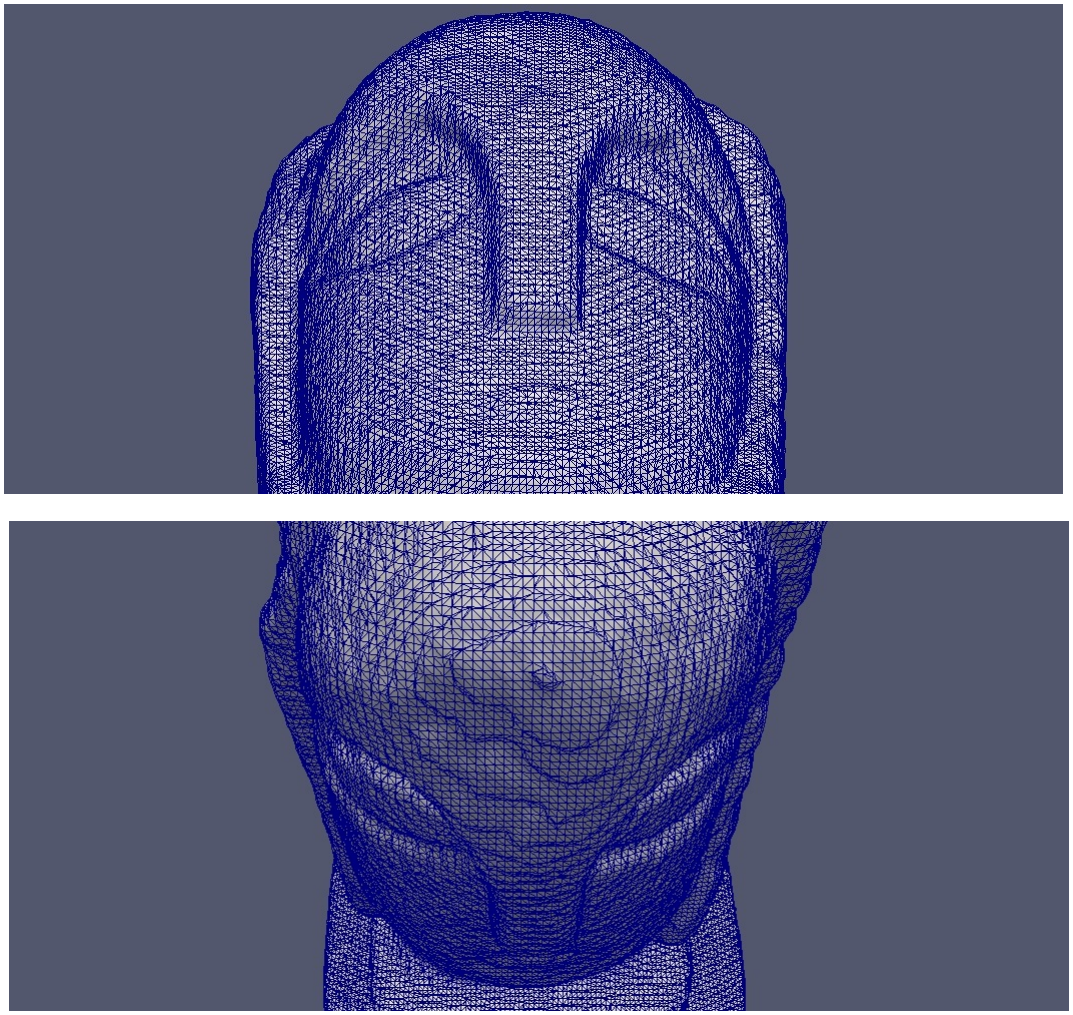

Figure 10: Details of the sealer with triangulated surface.

Figure 11: Angel statue. On the left we see the original object, on the right the result of reconstruction by our method.

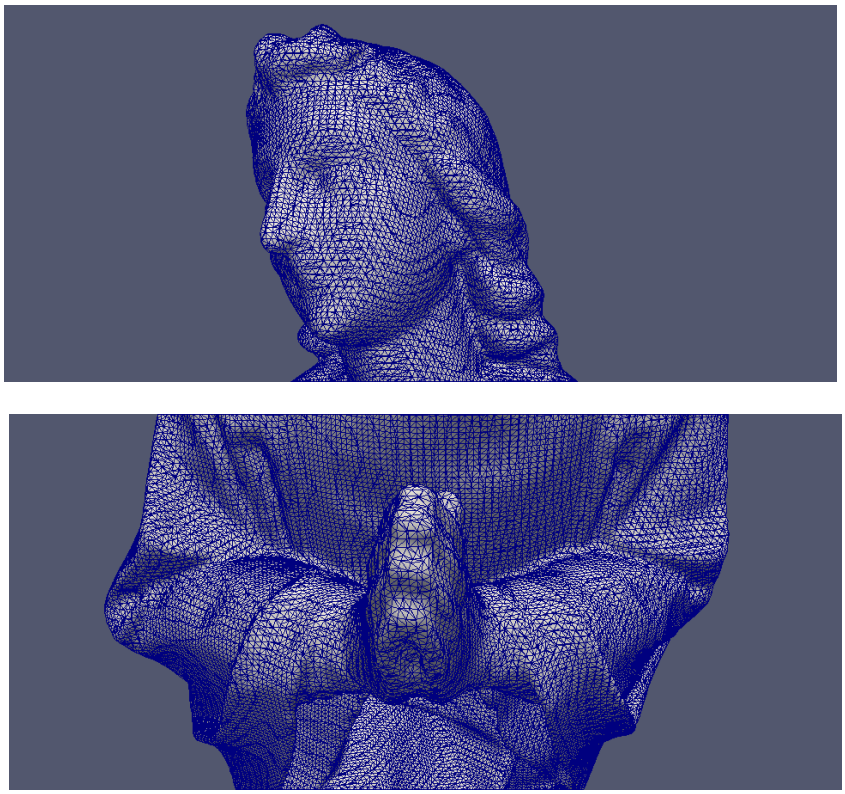

Figure 12: Details of the angel statue with triangulated surface.

We also tested our method on data sets with noise. In the point cloud data of the sealer and bracelet we added artificial noise by changing the coordinates of 100 random points. Thanks to the curvature part of equation (2.1) this kind of noise has no effect on our final model. We can observe that fact in Figures 13 and 14.

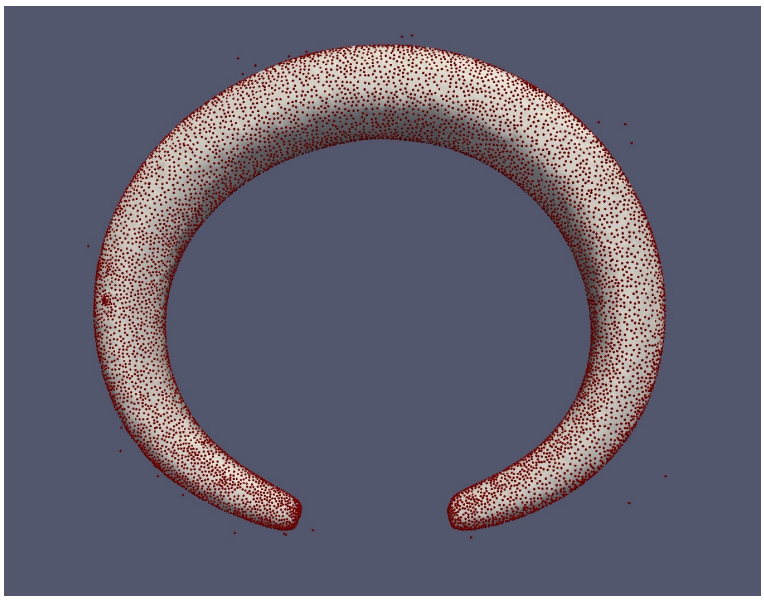

Figure 13: Bracelet point cloud data with noise, visualized with the final result.

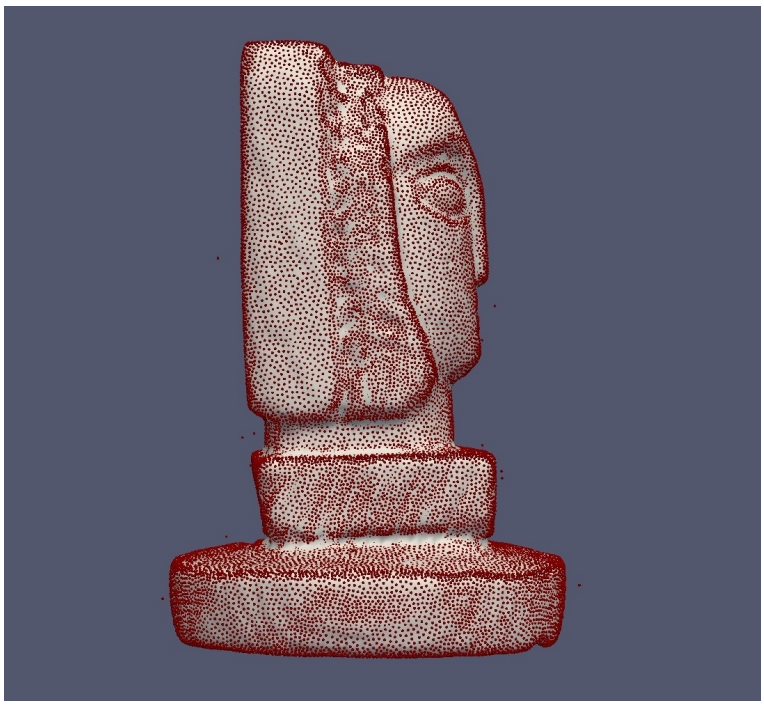

Figure 14: Sealer point cloud data with noise, visualized with the final result.

5 Computation acceleration

The part of our algorithm which consumes the most time during the computation is the construction and solution of the linear system of equations (3.17). To reduce this time we came up with the following idea. First we construct a band around the area between the initial surface and the point cloud data.To find the surface which we want to reconstruct it is sufficient to update the values on grid cells contained in such a band, thus we can calculate the SOR method only in this new subset of all grid cells. On Figure 15 we can see an example of this subset on the sealer mentioned in the previous chapter. For easier visualization we show this on a slice with the plane $x = 0$. Here the red line marks the point cloud data, the purple line the initial surface and the white lines the borders of the created band. In the background of the picture we show the values of the distance function like seen in previous examples.

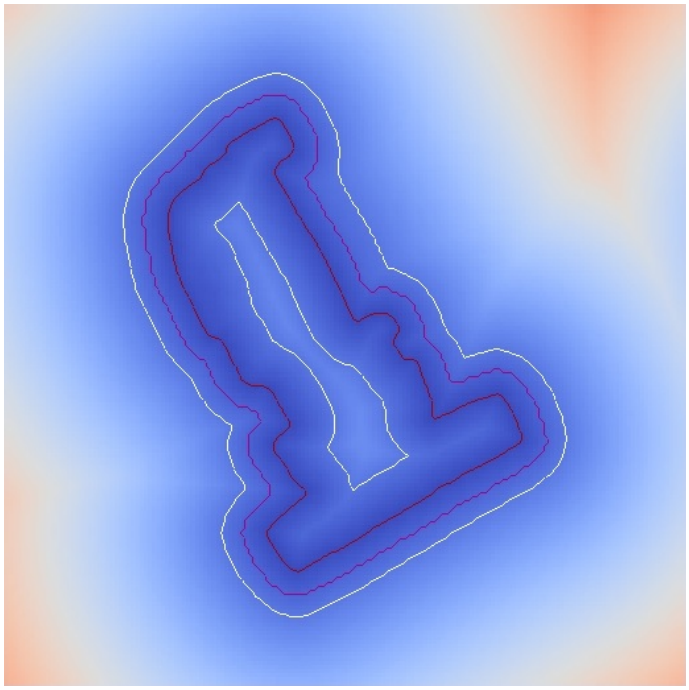

Figure 15: The slice of our new computational area on the plane $x = 0$.

To find this area we adopted the algorithm mentioned in chapter 3.4, which was used to find the initial surface, to this task. To obtain an outer border for the band which contains the initial condition we chose a new parameter $\gamma = 2\beta$. With this additional parameter and the introduction of a new set denoted F the algorithm changes as follows.

- Mark all points on the borders of the grid as exterior and add them to the set E.
- For every point in the set E check all neighboring points in the grid.
- If the neighboring point isn't an exterior point and his distance from the point cloud is greater or equal to β add it to end of set E and mark as exterior.
- If the neighboring point isn't an exterior point and his distance from the point cloud is smaller or equal to γ add it to end of set F.
- Continue until you get to the last point of E.
- For every point in the set F check all neighboring points in the grid.
- If the neighboring point isn't an exterior point and his distance from the point cloud is smaller or equal to γ add it to end of set F.
- Continue until you get to the last point of F.

From the set F we can create an array consisting of values 0, for point not in the band, and 1, for points in the band. This will serve as a mask for the SOR method, thus in the calculation loops we can determine if it is necessary to calculate the new value or if we can skip to the next grid point.

We measured how much time we managed to save with this new approach on the real life data sets of the bracelet and sealer. The tests were executed on a personal notebook with a dual core processor and 4 GB of memory. Our results are listed in the tables 1 and 2. We tested the algorithm on grids containing 40^3, 80^3 and 160^3 grid cells. All tests were performed with the same parameter β and stopping criteria for the iterations.

In the second column of the tables we recorded the number of points contained by the band. This number depends on the size and form of the original object represented by the data set. In columns three and four we see the measured times for the original and optimized implementation. In the tests we achieved not only reduced times but also better convergence, so fewer time steps were needed. This led to calculations which were 20 to 60 times faster.

Visually we can't detect any difference between the models created by the two methods. We measured the mean value of squared differences and listed the obtained values in the third column. We can see that these values are in the tolerable range.

Number of grid cells	Points in band	CPU time (s) Original	CPU time (s) Optimized	Mean squared difference
40^3	4 636	4.269	0.261	8.90795e-7
80^3	37 640	34.247	1.677	2.27554e-8
160^3	304 456	895.68	13.385	1.92055e-8

Table 1: CPU times comparison for the bracelet data set

Number of grid cells	Points in band	CPU time (s) Original	CPU time (s) Optimized	Mean squared difference
40^3	6 075	13.914	0.537	1.75849e-6
80^3	48 710	88.673	3.470	4.38982e-8
160^3	392 185	2 051.402	72.846	9.36878e-9

Table 2: CPU times comparison for the sealer data set

6 Conclusions

In this work we presented our approach of surface reconstruction from point cloud data utilizing the level set method. We formulated the mathematical model, derived the time and spatial discretization and provided the reader with an exact description of the numerical solution. By implementing the method we could obtain several interesting results for numerical tests and real life data which we presented as examples in different chapters. Our results show that for smoother objects a sparse grid already shows good result, but for a model with more detail we need more grid points. With adjusting the SOR method to our needs we achieved significant reduction of the required computational time, thus making our method more suitable for real life application.

References

[1] J. Haličková, K. Mikula, *Level Set Method for Surface Reconstruction (LSMSR) and its Application in Survazing* . Journal of Surveying Engineering, submitted 2013

[2] H. K. Zhao, S. Osher, B. Merriman, M. Kang, *Implicit and Non-parametric Shape Reconstruction from Unorganized Points UsingVariational Level Set Method* . Computer Vision and Image Understanding Vol. 80 (2000) pp. 295-319

[3] H. K. Zhao, *A fast sweeping method for Eikonal Equations.* Mathematics of Computation (2004), pp. 603-627

[4] S. Corsaro, K. Mikula, A. Sarti, F. Sgallari, *Semi-Implicit Covolume Method In 3D Image Segmentation.* SIAM Journal on Scientific Computing, (2006), pp.2248-2265