ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2013, Vol.42, No.1

Surface Reconstruction from Partially Structured Noisy Cloud of Points Using B-Splines

Raimundas Liutkevičius, Andrius Davidsonas

Department of Systems Analysis, Vytautas Magnus University
Vileikos st. 8, LT – 44404, Kaunas Lithuania
e-mail: r.liutkevicius@live.vdu.lt, andrius@live.vdu.lt

crossref http://dx.doi.org/10.5755/j01.itc.42.1.1391

Abstract. This paper presents a new approach how to reconstruct a parametric surface from a partially structured and noisy cloud of points representing surface that has a centre-line, such that all perpendicular rays to that line intersects with a surface not more than once. Presented algorithm analyses partially structured cloud of points, generated by point based 3D scanner and calculates parameters to build a non-uniform B-spline 3D mesh.

Keywords: 3D surface reconstruction; Non-uniform B-spline; Parametric surfaces; 3D scanning and visualization; Point cloud approximation.

1. Introduction

Digital 3D models are widely used in scientific data visualization, entertainment industry, orthotics and prosthetics (medicine), quality control and inspection, industrial design, documentation of cultural artefacts and many other areas. There are many techniques to create digital 3D models and one of those techniques is a 3D model generation from a cloud of points, acquired from 3D scanners. An output of a point based digital 3D scanner is a set of surface coordinates that are used as a raw data either for a further pre-processing or a direct synthesis of a 3D model. To generate a 3D geometric model from a cloud of points most often triangulation algorithms like Cocone or Crust are used because of their ability to process unstructured cloud of points (acquired coordinates from a 3D scanner) [12]. The triangulation methods work well when the cloud of points is dense enough and is not noisy, because a surface is reconstructed by combining neighbour points into triangles. The quality of the result surface very much depends on the distance between and the position of the points in a cloud. Another important thing to note about the triangulation algorithms is that additional data pre-processing, like triangle mesh simplification or refinement, smoothing, reorientation of faces, filling of the holes and other repairs is usually needed in order to increase the quality and decrease complexity of the reconstructed surface.

An alternative to triangulation algorithms are algorithms that use parametric curves for the reconstruction of a surface from coordinates. Such algorithms gained attention for several advantages: parametric surfaces are the exact analytical representations, have the potential of the three dimensional shape editing, parametric models require less data to describe an object, have easily adjustable level of detail (LOD), and are convertible to triangulated one, but not vice versa [14].

This paper presents an algorithm for creating a parametric surface from a partially structured and noisy cloud of points. An algorithm calculates coefficients of a non-uniform B-spline surface, i.e. fits a B-spline surface to a cloud of points.

2. Source data and definitions

In this paper a cloud of points that represents a surface of an object is acquired using an experimental 3D scanner [10]. This 3D scanner rotates an object in small steps and measures a distance to its surface at different levels (see Figure 1).

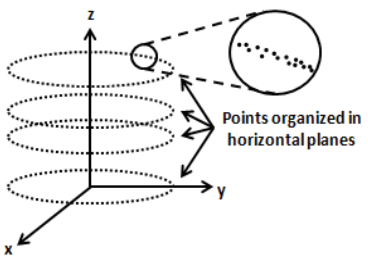

Figure 1. Structure of the scanned data

Measured data are stored in a cylindrical coordinate system representation and are converted to the Cartesian space during a pre-processing step. Conversion is required, because linear interpolation, approximation, filtration of flat surface areas in cylindrical coordinate system would cause undesired curvature (arcs, etc.) on the final surface. Also data may be acquired in Cartesian coordinate system if other 3D scanners are used.

All points in the same level have common coordinate z and form a layer. Distribution of points in the same layer and between layers can be arbitrary.

A presented algorithm is capable of reconstructing a 3D surface, which has a centre-line, such that all perpendicular rays coming out from that line intersect with the surface not more than once.

Surface points are defined for a scanned surface as:

$$\boldsymbol{\mathcal{P}}_{i,j}\left(x_{i,j}, y_{i,j}, z_{i,j}\right), \quad (1)$$

$$z_{i,j} = z_{i,j+1}, 1 \le j \le \beta_i, 1 \le i \le \gamma\ .$$

Here x, y, and z represent point's coordinates in the Cartesian coordinate system, γ is a number of layers and β_i is a number of points in a single layer i.

In this paper a presented algorithm converts a cloud of points into a parametric non-uniform B-spline surface that is defined as:

$$\boldsymbol{P}(u,v) = \sum_{c=0}^{s}\sum_{d=0}^{t} \boldsymbol{N}_{c,p}(u)\boldsymbol{N}_{d,q}(v)\boldsymbol{P}_{c,d} \quad (2)$$

$$u_k \le u \le u_{k+1},\ v_l \le v \le v_{l+1}.$$

Here p is a surface degree in u direction; q is a surface degree in v direction; $\boldsymbol{U} = \{u_k | 0 \le k \le s + p + 1\}$ is a B-spline's knot vector in u direction; $\boldsymbol{V} = \{v_l | 0 \le l \le t + q + 1\}$ is a B-spline's knot vector in v direction; $\boldsymbol{N}_{c,p}(u)$ and $\boldsymbol{N}_{d,q}(v)$ are B-spline's blending functions of degree p and q, respectively; $\boldsymbol{P}_{c,d}$ are B-spline's control (de-Boor) points; u and v are parametrization intervals (infinite set of points in a predefined range) and should not be confused with single values. The parameters $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{N}_{c,p}(u)$, $\boldsymbol{N}_{d,q}(v)$ and $\boldsymbol{P}_{c,d}$ are described in more detail in the following sections.

3. Representation of partially structured cloud of points into a parametric surface

3.1. Overview of an algorithm

This paper describes a new approach how to compute unknown parameters $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{N}_{c,p}(u)$, $\boldsymbol{N}_{d,q}(v)$ and $\boldsymbol{P}_{c,d}$ of a parametric B-spline surface $\boldsymbol{P}(u,v)$, described in (2) so that it approximates a cloud of points $\boldsymbol{\mathcal{P}}$.

Calculating parameters for a B-Spline surface equation directly from a cloud of points is a difficult task. In order to perform this task, knot vectors for a surface have to be defined. It appears that in a generated data cloud there are no two perpendicular directions for which selected knot values could meet Shoenberg-Whitney conditions [13]. Those conditions are necessary and sufficient for the approximation of cloud of points with a B-spline to be possible.

To overcome the problem, an approach to split a cloud of points into several slices was proposed, implemented and results presented to demonstrate its advantage over the known algorithms.

In the presented approach a cloud of points is split into three different matrices $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$, where each corresponding entry in matrices contains a corresponding coordinate of a point. These matrices then are used as a data source to calculate B-Spline parameters for each coordinate separately from equations:

$$\boldsymbol{P}^{(x)}(u,v) = \sum_{c=0}^{s}\sum_{d=0}^{t} \boldsymbol{N}_{c,p}(u)\boldsymbol{N}_{d,q}(v)\boldsymbol{P}_{c,d}^{(x)} \quad (3)$$

$$\boldsymbol{P}^{(y)}(u,v) = \sum_{c=0}^{s}\sum_{d=0}^{t} \boldsymbol{N}_{c,p}(u)\boldsymbol{N}_{d,q}(v)\boldsymbol{P}_{c,d}^{(y)} \quad (4)$$

$$u_k \le u \le u_{k+1},\ v_l \le v \le v_{l+1}.$$

B-Spline parameters for $\boldsymbol{Z}$ coordinate values are derived from the already calculated parameters, as described in Subsection 3.5.

A process to calculate unknown parameters of equations (3) and (4) is divided into four steps that are explained in details in the next subsections (see Figure 2).

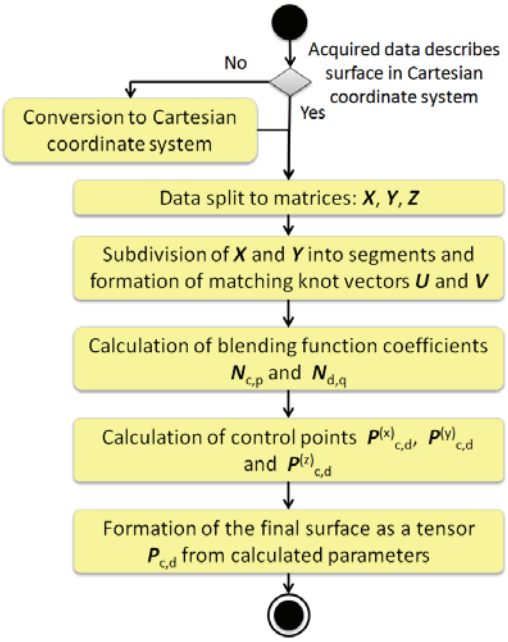

Figure 2. The process of parametric surface creation

The first step is a pre-processing step where input data are converted from cylindrical to Cartesian coordinate system. Then, during the second step, knot vectors $\boldsymbol{U}$ and $\boldsymbol{V}$ have to be defined. This is done by subdividing pre-processed data into estimated number of segments towards two perpendicular directions u and v, so that Shoenberg-Whitney conditions are met [13]. These segments correspond to B-spline local segments or knot intervals. Endpoints of segments are

used to create corresponding knot vectors $\boldsymbol{U}$ and $\boldsymbol{V}$. During the third step B-spline blending functions $\boldsymbol{N}_{c,p}(u)$ and $\boldsymbol{N}_{d,q}(v)$ are calculated from knot vectors and data projections in u and v directions, using Cox-deBoor recurrence [5]. In the last, forth step, B-spline control points $\boldsymbol{P}_{c,d}$ are calculated based on blending functions and other calculated data by applying optimization in the least square sense.

When parameters for equations (3) and (4) are calculated, they are combined to form parameters for a parametric surface (2) that approximates input data $\boldsymbol{\mathcal{P}}$. Then a parametric surface can be displayed or saved for the further use.

3.2. Pre-processing of input data

Experimental 3D scanner stores acquired points in the cylindrical coordinate system [10]. Pre-processing step is necessary to convert those points into the Cartesian representation.

Experimental 3D scanner, used for data acquisition, operates in a non-ideal environment; it has specific mechanical characteristics, limited accuracy and is affected by electrical noise. As a result, at different levels measurements might occur at different positions and if the measurements are stored in a matrix $\boldsymbol{C}$ as given below in (5), the matrix will most likely contain some number of empty elements r_{ij} [10]. To decrease the size of the matrix $\boldsymbol{C}$, acquired data in columns are quantized into 8000 intervals in a range of (0° - 360°) with a step size equal to 0,045°. Step size value here is a smallest detectable step size of a stepper motor, converted into angle and divided by 2. The number of rows in the matrix $\boldsymbol{C}$ depends on the number of stops a laser sensor makes in a vertical direction (Figure 1). A distance between layers in this case varied in a range [1,25-1,55] mm. An absolute vertical position of a layer is stored in a first column of the matrix C. The structure of the matrix $\boldsymbol{C}$ is:

$$\boldsymbol{C} = \begin{pmatrix} 0 & \theta_1 & \theta_2 & \dots & \theta_j & \dots & \theta_n \\ \tau_1 & r_{1,1} & r_{1,2} & \dots & r_{1,j} & \dots & r_{1,n} \\ \tau_2 & r_{2,1} & r_{2,2} & \dots & r_{2,j} & \dots & r_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \tau_i & r_{i,1} & r_{i,2} & \dots & r_{i,j} & \dots & r_{i,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \tau_m & r_{m,1} & r_{m,2} & \dots & r_{m,j} & \dots & r_{m,n} \end{pmatrix} \quad (5)$$

here θ, r, τ are angle, radius and height respectively, m is the number of layers and n is the number of quantization intervals in every layer i (Figure 3).

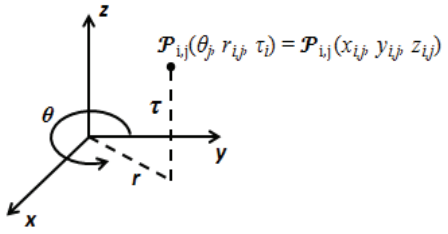

Figure 3. Relationship between cylindrical and Cartesian coordinate systems

A conversion from the cylindrical to the Cartesian coordinate system is straightforward:

$$\begin{cases} x_{i,j} = r_{i,j} \times \cos\theta_j, \\ y_{i,j} = r_{i,j} \times \sin\theta_j, \\ z_{i,j} = \tau_i. \end{cases} \quad (6)$$

After the conversion, object's surface is represented by $m \times n$ number of points, including empty ones. Conversion results are stored in three separate matrices $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$, where each row corresponds to a layer data in a cloud of points:

$$\boldsymbol{X} = (x_{i,j}), \boldsymbol{Y} = (y_{i,j}), \boldsymbol{Z} = (z_{i,j}) \quad (7)$$

$$z_{i,j} = z_{i,j+1}, 1 \le i \le m, 1 \le j \le n.$$

Matrices $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ are sparse because matrix (5) consists of empty entries.

The sparseness is the most undesirable property of a data set and needs to be corrected to proceed. The problem can be solved by inserting data into sparse areas using, for example interpolation between neighbor data points. If the input data contains a considerable amount of noise, it is recommended to apply data filtering before performing data correction. Otherwise noise can lead to undesirable deformations on the reconstructed surface. Recommendations on the selection of filters can be found in [4].

Data sparseness can be eliminated using a linear interpolation algorithm. In this paper a slightly modified algorithm was used:

```
procedure interpolate(X, Y, θ)
// in: X – x coordinates of surface points
// in: Y – y coordinates of surface points
// in: θ – position angle from matrix (5)
// out: X, Y
begin
  for each row i in matrix X do
    p = index of last non empty element in row i;
    r = index of first non empty element in row i;
    for each column j in matrix X do
      if X_{i,j} is not empty then
        p = j;
        r = index of next non empty element in row i;
        if r is empty then
          r = index of first non empty element in row i;
        end if
      else
        if θ_p > θ_r then qp = θ_p − 360;
        else qp = θ_p; end if
        if θ_j > θ_r then qj = θ_j − 360;
        else qj = θ_j; end if
        k = (qj − qp)/(θ_r − qp);
        X_{i,j} = X_{i,p} × (1 − k) + X_{i,r} × k;
        Y_{i,j} = Y_{i,p} × (1 − k) + Y_{i,r} × k;
      end if
    end for
  end for
end.
```

Further steps in the algorithm will be performed with pre-processed matrices ***X*** and ***Y***.

3.3. Calculation of knot vectors

A B-spline surface is defined by a set of curves that intersect and form surface patches. To create such a structure, matrices ***X*** and ***Y*** are subdivided into estimated number of segments: *r* number of segments in a row direction *u* and *w* number of segments in a column direction *v* (Figure 4).

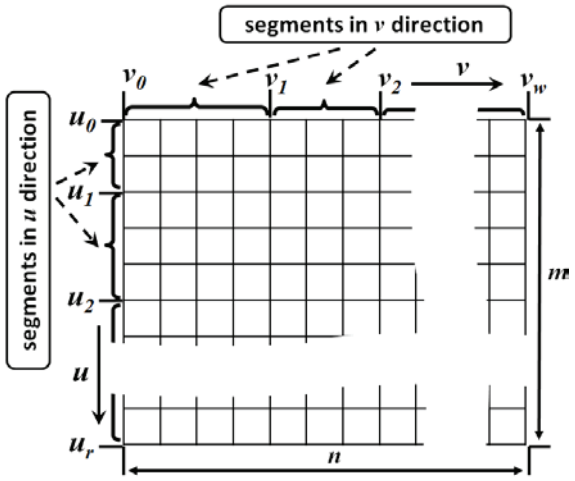

Figure 4. Segmentation of matrices. The same schema is used to create segments in matrices ***X*** and ***Y***

These segments are called B-spline local segments or knot intervals. The number of segments and the number of data points in them depend on input data distribution in coordinate space and a degree of a surface in both directions *u* and *v*. In B-spline theory, a cubic spline is the most common type, sufficient to approximate most types of shapes, so cubic B-spline surface i.e. of order 4 (degree = 3) is used through this paper too.

Input data spacing in row direction *u* is directly related to z coordinate of each data layer. To maintain this information and use it in calculations, a vector ***H*** is defined as:

$$\boldsymbol{H} = (h_1, h_2, \dots, h_i, \dots, h_m) = (z_{1,1}, z_{2,1}, \dots, z_{m,1}). \quad (8)$$

To represent input data spacing in column direction *v* a vector ***A*** is defined as:

$$\boldsymbol{A} = (a_1, a_2, \dots, a_j, \dots, a_n) = (1,2, \dots, n). \quad (9)$$

Because data points in *v* direction were interpolated using linear interpolation, spacing of points is considered to be uniform and is encoded as a sequence 1,2,...,n in the vector A. Segments of B-splines are estimated from the vectors ***H*** and ***A*** rather than values in matrices ***X*** and ***Y***.

Segments have to be selected according to the following requirements:

$$u_k \le h_i \le u_{k+1},\quad 1 \le i \le m, 0 \le k \le r, 1 \le r \le m - p, \quad (10)$$

$$v_l \le a_j \le v_{l+1},\quad 1 \le j \le n, 0 \le l \le w, 1 \le w \le n - q. \quad (11)$$

Here *r* and *w* are numbers of segments in directions *u* and *v*, respectively; *p* and *q* are B-spline degrees in directions *u* and *v*, respectively; u_k and v_l are B-spline knots, which match with position values of segment endpoints in corresponding directions *u* and *v*. These requirements ensure that Shoenberg-Whitney conditions are met i.e. it is possible to create a valid parametric surface that approximates a given data set [13].

The number of segments *r* and *w* depends on a priori knowledge about the input data and the desired approximation properties, and should be selected individually. A common methodology on how to select *r* and *w* was not developed as it was not a scope of this research. In an ideal case, a quantity of segments in each direction should be selected so that curves of selected degrees *p* and *q* are flexible enough to accurately approximate data that fall into successive segments. In practice, there should be at least one segment defined in each direction, but usually the number of segments should be greater than one in order to create a good surface approximation. On the other hand, if too many data points fall into a single segment, smoothing effect arise and a final surface might be distorted. If noise presents in input data, smoothing effect might be desired and it can be controlled by changing the number and the size of segments. In this paper 100 segments were chosen in the row direction *u* and 100 segments - in the column direction *v* in order to define knot vectors.

Knots u_k and v_l in corresponding directions *u* and *v* can be calculated using predefined functions as segmentation templates and selected depending on variation of values in vectors ***A*** and ***H***. Examples of such functions can be a linear function, a Gaussian function, or other. If distances between values in vectors ***A*** and ***H*** are random, a use of linear function is suggested. In the presented algorithm a linear function was used to define knots (see algorithm calcKnots below).

Knot vectors ***U*** and ***V*** are created from knots u_k and v_l. In order to force B-spline curves to begin at the first control point and to end at the last control point, knot vectors were modified by duplicating the first and the last knots *p* and *q* times, respectively. As a result, the knot vector ***U*** is:

$$\boldsymbol{U} = (u_0, \dots, u_0, u_1, u_2, \dots, u_{r-1}, u_r, \dots, u_r) \quad (12)$$

and vector ***V*** is:

$$\boldsymbol{V} = (v_0, \dots, v_0, v_1, v_2, \dots, v_{w-1}, v_w, \dots, v_w). \quad (13)$$

Knot duplication makes B-spline surface to interpolate endpoints of input data. A resulting parametric surface becomes non-uniform. Our implementation of knot vectors calculation algorithm is presented below:

```
procedure calcKnots(H, A, r, w, p, q)
// in: H – see equation (8)
// in: A – see equation (9)
// in: r – number of segments in u direction
// in: w – number of segments in v direction
// out: U, V – knot vectors
begin
    m = length(H);
    for i = 0 to (r + 2 × p) do
        if i < p then
            U_{i+1} = H_1;
        else if i > r + p – 2 then
            U_{i+1} = H_m;
        else
            U_{i+1} = H_1 + (i - p) × (H_m – H_1) / m – 1;
        end if
    end for
    n = length(A);
    for j = 0 to (w + 2 × q) do
        if j < p then
            V_{j+1} = A_1;
        else if j > w + q – 2 then
            V_{j+1} = A_n;
        else
            V_{j+1} = A_1 + (j - q) × (A_m – A_1) / n – 1;
        end if
    end for
end.
```

The same knot vectors are used to approximate data in both matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$.

3.4. Calculation of B-spline blending functions

B-spline's are constructed using basis functions, which are blended together and transformed by control points in order to achieve a desired shape. These functions are named blending functions and according to (2)-(4) are encoded in parameters $\boldsymbol{N}_{c,p}(u)$ and $\boldsymbol{N}_{d,q}(v)$. At this point relation between parametrization intervals u and v in expressions (3)-(4) and values h_i and a_j in expressions (10)-(11) can be noticed. As a result, blending functions can be calculated using parameters $h_i \in u$, and $a_j \in v$ and be denoted as $\boldsymbol{N}_{c,p}(h_i)$ and $\boldsymbol{N}_{d,q}(a_j)$.

B-spline blending functions $\boldsymbol{N}_{c,p}(h_i)$ and $\boldsymbol{N}_{d,q}(a_j)$ are defined by p+1 and q+1 non-zero coefficients respectively. Coefficients must be selected to satisfy the following requirements:

$$\begin{aligned} &\boldsymbol{N}_{c,p}(h_i) \geq 0, \quad \boldsymbol{N}_{d,q}(a_j) \geq 0, \\ &\textstyle\sum_{c=0}^{s} \boldsymbol{N}_{c,p}(h_i) = 1, \quad \sum_{d=0}^{t} \boldsymbol{N}_{d,q}(a_j) = 1, \\ &u_k \leq h_i \leq u_{k+1}, 1 \leq i \leq m, \\ &v_l \leq a_j \leq v_{l+1}, 1 \leq j \leq n. \end{aligned} \tag{14}$$

Here $\boldsymbol{U} = \{u_k | 0 \leq k \leq s + p + 1\}$ and $\boldsymbol{V} = \{v_l | 0 \leq l \leq t + q + 1\}$ are B-spline's knot vectors.

Coefficients of blending functions $\boldsymbol{N}_{c,p}(h_i)$ and $\boldsymbol{N}_{d,q}(a_j)$ in this paper are calculated using Cox-de Boor recurrences [5]:

$$\begin{aligned} &\text{Initialise } \boldsymbol{N}_{c,0}(h_i) = \begin{cases} 1, & h_i \in [u_k, u_{k+1}) \\ 0, & \text{otherwise} \end{cases}, \\ &\boldsymbol{N}_{d,0}(a_j) = \begin{cases} 1, a_j \in [v_l, v_{l+1}) \\ 0, \text{otherwise} \end{cases}, \\ &\boldsymbol{N}_{c,p}(h_i) = (h_i - u_k)\frac{\boldsymbol{N}_{c,p-1}(h_i)}{u_{k+p} - u_k} + \\ &+(u_{k+p+1} - h_i)\frac{\boldsymbol{N}_{c+1,p-1}(h_i)}{u_{k+p+1} - u_{k+1}}, \end{aligned} \tag{15}$$

$$\begin{aligned} &\boldsymbol{N}_{d,q}(a_j) = (a_j - v_l)\frac{\boldsymbol{N}_{d,q-1}(a_j)}{v_{l+q} - v_l} + \\ &+(v_{l+q+1} - a_j)\frac{\boldsymbol{N}_{d+1,q-1}(a_j)}{v_{l+q+1} - v_{l+1}}. \end{aligned} \tag{16}$$

The same two sets of parameters $\boldsymbol{N}_{c,p}(h_i)$ and $\boldsymbol{N}_{d,q}(a_j)$ are used to approximate data in matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$, because the knot vectors $\boldsymbol{U}$, $\boldsymbol{V}$ and the parameters in vectors $\boldsymbol{H}$, $\boldsymbol{A}$ for both matrices are also the same. Matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ are defined in (7).

3.5. Calculation of B-spline control points

The shape of B-spline surface $\boldsymbol{P}(u,v)$ is formed using control (de-Boor) points $\boldsymbol{P}_{c,d}$ and blending functions $\boldsymbol{N}_{c,p}(u)$ and $\boldsymbol{N}_{d,q}(v)$, equation (2). Control points normally do not belong to B-Spline surface, but this property depends on knot vectors. In the proposed algorithm knot vectors are designed so that the control points at the edges of a surface match with the surface points.

As knot vectors and blending functions are calculated using parameters $h_i \in u, a_j \in v$, equation (2) can be expressed as:

$$\boldsymbol{P}(h_i, a_j) = \textstyle\sum_{c=0}^{s}\sum_{d=0}^{t} \boldsymbol{N}_{c,p}(h_i)\boldsymbol{N}_{d,q}(a_j)\boldsymbol{P}_{c,d}, \tag{17}$$

$$u_k \leq h_i \leq u_{k+1}, v_l \leq a_j \leq v_{l+1},$$

$$1 \leq i \leq m, 1 \leq j \leq n.$$

Here $\boldsymbol{U} = \{u_k | 0 \leq k \leq s + p + 1\}$ and $\boldsymbol{V} = \{v_l | 0 \leq l \leq t + q + 1\}$ are B-spline knot vectors.

Since $\boldsymbol{N}_{c,p}(u)$ in (17) does not depend on the index d, the equation can be rewritten as:

$$\boldsymbol{P}(h_i, a_j) = \textstyle\sum_{c=0}^{s} \boldsymbol{N}_{c,p}(h_i)\left(\sum_{d=0}^{t} \boldsymbol{N}_{d,q}(a_j)\boldsymbol{P}_{c,d}\right). \tag{18}$$

Then inner sum can be denoted as $\boldsymbol{Q}_{c,j}$:

$$\boldsymbol{P}(h_i, a_j) = \textstyle\sum_{c=0}^{s} \boldsymbol{N}_{c,p}(h_i)\boldsymbol{Q}_{c,j}, \tag{19}$$

$$\boldsymbol{Q}_{c,j} = \textstyle\sum_{d=0}^{t} \boldsymbol{N}_{d,q}(a_j)\boldsymbol{P}_{c,d}. \tag{20}$$

Equation (19) hides n over-determined systems of linear equations with $s \times n$ unknown coefficients $\boldsymbol{Q}_{c,j}$. Assuming that parameters $\boldsymbol{N}_{c,p}(h_i)$ are stored in a matrix $\boldsymbol{N}_{i,c}$, the unknown coefficients can be calculated by minimizing a sum:

$$\delta_j^2 = \min_{Q_{c,j}} \sum_{i=1}^{m} \left| \boldsymbol{D}_{i,j} - \boldsymbol{N}_{i,c} \boldsymbol{Q}_{c,j} \right|^2, \quad (21)$$

$$\boldsymbol{N}_{i,c} = \begin{pmatrix} N_{1,1} & N_{1,2} & \cdots & N_{1,s} \\ N_{2,1} & N_{2,2} & \cdots & N_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ N_{m,1} & N_{m,2} & \cdots & N_{m,s} \end{pmatrix},$$

$$\boldsymbol{Q}_{c,j} = \begin{pmatrix} Q_{1,j} \\ Q_{2,j} \\ \vdots \\ Q_{s,j} \end{pmatrix}, \boldsymbol{D}_{i,j} = \begin{pmatrix} D_{1,j} \\ D_{2,j} \\ \vdots \\ D_{m,j} \end{pmatrix},$$

$0 \le c \le s, 1 \le j \le n.$

Here $|\cdot|$ is the length of vector, $\boldsymbol{D}_{i,j}$ stores the acquired data coordinates $\boldsymbol{X}$ or $\boldsymbol{Y}$.

Equation (20) hides s over-determined systems of linear equations with $s \times t$ unknown coefficients $\boldsymbol{P}_{c,d}$. Assuming that the parameters $\boldsymbol{N}_{d,q}(a_j)$ are stored in the matrix $\boldsymbol{N}_{j,d}$, the unknown coefficients can be calculated by minimizing a sum:

$$\varphi_c^2 = \min_{P_{c,d}} \sum_{j=1}^{n} \left| \boldsymbol{Q}_{c,j}^T - \boldsymbol{N}_{j,d} \boldsymbol{P}_{c,d}^T \right|^2, \quad (22)$$

$$\boldsymbol{N}_{j,d} = \begin{pmatrix} N_{1,1} & N_{1,2} & \cdots & N_{1,t} \\ N_{2,1} & N_{2,2} & \cdots & N_{2,t} \\ \vdots & \vdots & \ddots & \vdots \\ N_{n,1} & N_{n,2} & \cdots & N_{n,t} \end{pmatrix},$$

$$\boldsymbol{P}_{c,d} = \begin{pmatrix} P_{1,c} \\ P_{2,c} \\ \vdots \\ P_{t,c} \end{pmatrix}^T, \boldsymbol{Q}_{c,j} = \begin{pmatrix} D_{1,c} \\ D_{2,c} \\ \vdots \\ D_{n,c} \end{pmatrix}^T,$$

$0 \le c \le s, 0 \le d \le t.$

All over-determined systems of linear equations can be solved separately using least squares or other algorithms.

Expressions in equations (21) and (22) are minimized twice: the first time – to find control points $\boldsymbol{P}_{c,d}^{(x)}$ and to approximate the data in the matrix $\boldsymbol{X}$, and the second time – to find control points $\boldsymbol{P}_{c,d}^{(y)}$ and to approximate the data in matrix $\boldsymbol{Y}$. Both matrices are defined in (7).

In the experiments described in this paper, a minimization problem is solved using QR decomposition. In case of large input data (over 200 000 points), Householder transformations are used to perform QR decomposition [7].

A final parametric surface $\boldsymbol{P}(u,v)$ will contain $c \times d$ control points:

$$\boldsymbol{P}_{c,d} = \left(\boldsymbol{P}_{c,d}^{(x)}, \boldsymbol{P}_{c,d}^{(y)}, \boldsymbol{P}_{c,d}^{(z)} \right), \quad (23)$$

$0 \le c \le s, 0 \le d \le t.$

The third component $\boldsymbol{P}_{c,d}^{(z)}$ can be calculated using the following equation:

$$\boldsymbol{P}_{c,d}^{(z)} = \frac{(\boldsymbol{U}_{c+1} + \cdots + \boldsymbol{U}_{c+p})}{p}. \quad (24)$$

Here $\boldsymbol{U}$ is the knot vector in u direction, and p is the degree of a surface in u direction.

4. Storing parametric surface

In order to restore a parametric surface it is enough to store the values of two knot vectors in both u and v directions (Section 3.3) and control points $\boldsymbol{P}_{c,d}$ (Section 3.5). Most of the 3D APIs accept those parameters in their surface reconstruction functions.

As an alternative, control points $\boldsymbol{P}_{c,d}$ and sets of parameters $\boldsymbol{N}_{c,p}(u)$ and $\boldsymbol{N}_{d,q}(v)$ can be saved (Section 3.4). Then there would not be necessary to recalculate coefficients of blending functions when loading a surface from a hard disk drive, but this approach has several disadvantages: 1) it takes at least 3 times more storage space on a hard disk drive; 2) collocation matrices are usually not accepted in well-known 3D API's, such as OpenGL, etc., and 3) GPU based visualisation of a parametric surface, including generation of collocation matrices from knot's, is usually done faster than providing already predefined collocation matrices so saving of parameters of blending functions does not add any value.

STEP, IGES and others file formats can be considered for saving and describing parametric surface in more standardized way. Although today's CAD system's have their own methods for representing and exchanging data. As a result data representation and exchange issues may arise. Some recommendations how to minimize bad consequences of the data exchange using STEP is provided in [11].

5. Implementation

The presented surface reconstruction algorithm is part of the 3D surface reconstruction system (see Figure 5).

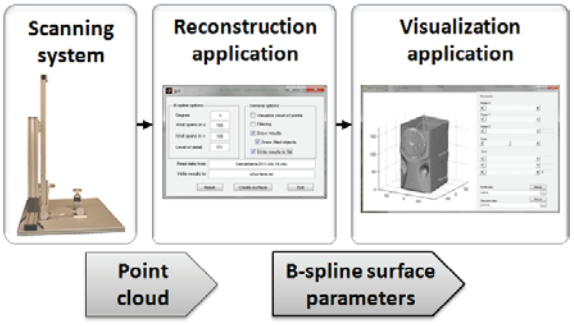

Figure 5. 3D surface reconstruction system

The system consists of a custom built 3D scanner and software, consisting of the hardware control, surface reconstruction and visualization algorithms. C# language was used to write hardware control software. The purpose of the software is to send commands to the scanner, acquire measurements and store them in a single matrix (5) in a comma separated values (CSV) format. This file is later used by the presented surface reconstruction algorithm as an input data. The reconstruction algorithm was implemented using MATLAB, writing custom code. For the calculation of B-spline control points' values the

systems of linear equations has to be solved. MATLAB implementation of QR algorithm was used (function qr) for this task. qr function in Matlab uses the LAPACK [9] routines DGEQP3, DORMQR and DTRTRS (see MATLAB documentation). The presented surface reconstruction algorithm calculates B-spline surface parameters and outputs them into a text file. This file then is used as an input data in the visualization step. Model visualization part was implemented using C# language and Tao Framework - OpenGL wrapper for .NET.

6. Experimental results

For the experiments a statue (Figure 6 (a)) was scanned using a point based laser scanner [10]. As a result, a set of 144 450 data points in 136 layers was acquired and saved in a sparse data matrix (5). This set of points was used as input data for testing different surface reconstruction algorithms. The result of the presented surface reconstruction algorithm is shown in Figure 6 (g-h) and was compared to the neighbour triangulation algorithm (f) described in [4] and the other three well known 3D surface reconstruction algorithms: Delaunay, Crust and Tight Cocone, Figure 6 (b-e). The performance and other statistical results were also compared and provided in a Table 1.

The processing time in Table 1 includes average time required to read the data, to calculate 3D surface and to save the results on storage disk using a PC with 2.4GHz Core 2 Duo CPU and 4GB memory. 3D surface was visualized using OpenGL library and hardware accelerated graphics enabled. Triangulation results were stored in a plain text object file format (OFF). This has to be taken into account when comparing sizes of output file presented in Table 1.

Figure 6b shows a result of simple Delaunay triangulation algorithm. This algorithm gave the worst visual result. Data points in a resulting data structure were not connected correctly and a large number of triangles were positioned incorrectly too. A triangle mesh was created using CGAL implementation of Delaunay triangulation [2] and a natural neighbor approach.

Figures 6c and 6d show the results of the Crust triangulation algorithm [1]. The algorithm does a

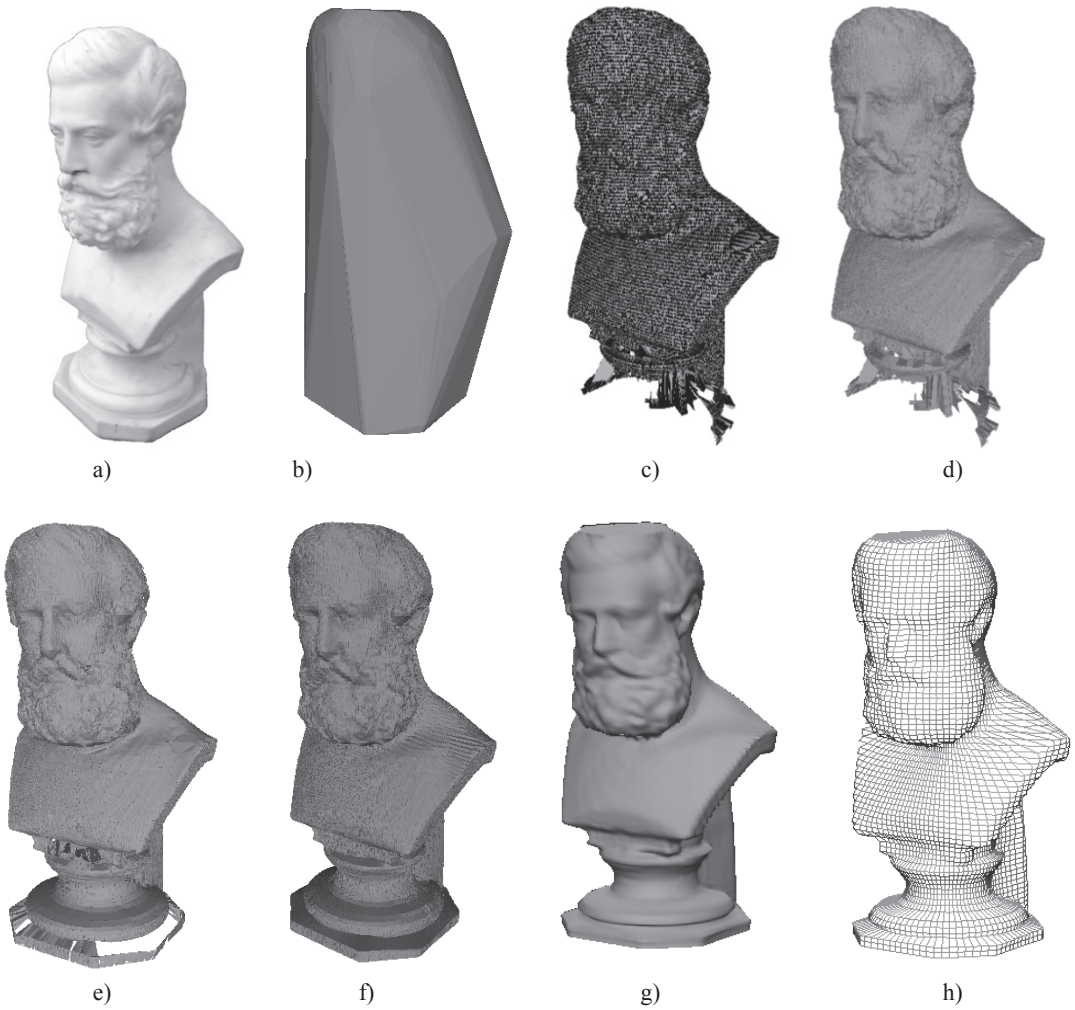

Figure 6. Results of surface reconstruction algorithms: (a) a test object – 18,2 cm height statue , (b) Delaunay triangulation [2], (c) Crust triangulation [1], (d) Crust triangulation with surface correction, (e) Tight Cocone triangulation [6] with surface correction, (f) nearest neighbour triangulation [4], (g) parametric surface, and (h) a frame of parametric surafce

Table 1. Experimental statistics of methods for surface reconstruction (average values)

Reconstruction method	Processing time (s)	Triangle / control points	ASCII file size (MB)
Delaunay triangulation (b)	7.8830	58 818	3.7
Crust triangulation (c)	133.2070	772 644	8.66
Tight Cocone (e)	699.506	751 254	11.1
Nearest neighbour triangulation (f)	18.3560	6 474 330	73.0
Parametric surface (g)	13.9620	10 609	0.238

manifold extraction in order to form a regular surface and uses a ball pivoting method to extract a manifold and to return outward's normal's orientation. Because of the noise and uneven distribution of the input data, the resulting surface is not water-tight (there are holes on the surface), have errors in manifold structure and the orientation of faces. Surface after reorientation of triangle faces is shown in figure 6d. The errors are best seen at the bottom part and near the ear on the head of the statue. Also, small noise covers the entire surface. The performance of the algorithm is the worst if compared to the other analyzed algorithms.

Figure 6e shows the result of the Tight Cocone algorithm. The algorithm is based on the cocone algorithm that uses a single Voronoi/Delaunay computation. It is supposed to be water-tight and is claimed to be scale and density independent [6]. In this algorithm, the number of holes is reduced by stitching them from existing Delaunay triangles and without inserting new points. The result is similar to the one produced by the Crust algorithm: the surface is not water-tight, has errors in the manifold structure and the orientation of faces. The algorithm was tested with several parameters. Only the best achieved result (after performing face reorientation) is presented in figure 6e. The Tight Cocone triangulation algorithm takes about 50 times more time and about 46 times more storage space for a surface reconstruction than using our parametric reconstruction algorithm (see Table 1). The code for the Tight Cocone, released for academic use, can be downloaded from [3].

Figure 6f shows a result of simple nearest neighbour triangulation algorithm that creates tight triangulated surface and can be used to perform real time triangulation with partial data. The algorithm is based on RBF, triangle strips and predictable structure of cloud of points [4]. The algorithm gives better results than the previous triangulation algorithms. The recreated surface is water-tight and without noticable errors, processing time is relatively small. However, the algorithm creates a large number of triangles. This might be a problem to render and manipulate such a reconstructed surface. Also, the generated output is not very smooth if compared to the presented algorithm.

The result of the algorithm, presented in this paper, (Figure 6g) is constructed from cubic non-uniform B-spline curves that lay down horizontally and vertically as shown in Figure 6h. The surface contains 100 B-spline segments in each directions and the total of 103x103=10609 control points. 150 approximation steps in each direction in B-spline surface were made to draw the surface displayed in Figure 6g and 80 steps - for the surface in Figure 6h. The generated surface is watertight (without surface holes) and smooth compared to the results of the previously analyzed algorithms. Our algorithm needs less processing time to calculate surface's parameters than the Crust and the Tight Cocone algorithms and uses much less storage space to store output results than the analyzed triangulation algorithms (Table 1).

In order to evaluate an approximation accuracy of the presented algorithm in this experiment, vectors connecting input data points and corresponding points on the B-spline surface were calculated. The mean of vector lengths is 0.1377 mm, the standard deviation is 0.2385 mm. These numbers are influenced by the noise and measurement errors that are present in the input data and are filtered out in the B-spline surface.

7. Conclusions

Experimental results presented in this paper show that the proposed algorithm can be successfully used for the reconstruction of complex geometric surfaces from a partially structured cloud of points that does not fit into a data grid. The algorithm combines well-known parametric and optimisation algorithms with the coordinate-wise surface generation approach to create a non-uniform B-spline surface that can be described with 15 times less data if compared to triangulation surfaces.

The proposed strategy to split input data into separate groups by coordinates and process separately was shown to be working and can lead to expected results.

It was demonstrated that QR decomposition and Householder reflection algorithms can be effectively used for the calculation B-spline surface parameters. Linear interpolation, used in the pre-processing step to adjust input data is shown to be sufficient to solve points-to-surface problem when the input data set is unorganized. This technique simplifies the process of surface subdivision. It also provides information about the distribution of surface points.

It was also experimentally shown that the suggested algorithm has a small average processing time if compared to the analyzed triangulation algorithms, and can generate a high quality surface even from the poorly sampled and noisy data.

8. Further research

The presented algorithm is targeted to the reconstruction of surfaces, which have a centre-line such that all perpendicular rays to that line intersect with the surface not more than once. In general, it is possible to use the algorithm with the other types of objects. In that case, an object should be subdivided into parts, which satisfy Shoenberg-Whitney conditions [13]. Then parts of the object should be parameterised separately and joined together to form a final surface. The join can be performed using a direct degree elevation of NURBS curves [8] and an integration of curve segments.

In the presented algorithm a linear subdivision of acquired data into segments is used, but possibly better results could be achieved by performing density analysis of the cloud of points and adjusting segments accordingly.

Acknowledgments

The authors would like to thank Tomas Krilavičius and Kazimieras Padvelskis from Vytautas Magnus University for their valuable remarks and suggestions, and Tamal K. Dey from the department of CIS at Ohio State University for providing implementation of Tight Cocone algorithm.

References

[1] **N. Amenta, M. Bern, M. Kamvysselis**. A New Voronoi-Based Surface Reconstruction Algorithm. In: *SIGGRAPH 98*, pp. 415-421.

[2] CGAL-Computational Geometry Algorithms Library, 2011. Available at: http://www.cgal.org/.

[3] COCONE-Cocone Software for surface reconstruction and medial axis**.** Available at: http://www.cse.ohio-state.edu/~tamaldey/cocone.html**.**

[4] **A. Davidsonas, R. Liutkevičius.** A PLC Application for Data Scanning and Visualization. In: *Proceedings of International Conference "Electrical and Control Technologies–2008"*, Kaunas, 2008, pp. 144-149.

[5] **C. de Boor.** On calculating with B-splines. In: *Journal of Approximation Theory*, *6*, 1972, pp. 50–62.

[6] **T. K. Dey, S. Goswami.** Tight Cocone: A water tight surface reconstructor. In: *Proceedings of 8th ACM Symposium. Solid Modeling Appl.*, 2003, pp. 127-134.

[7] **G. H. Golub, C. F. Van Loan.** *Matrix Computations (3rd ed.)*, Johns Hopkins University Press, 1996.

[8] **K. Jankauskas, D. Rubliauskas.** Direct degree elevation of NURBS curves. In: *Information Technology and Control,* 2010, Vol. 39, No. 4, pp. 269-283.

[9] LAPACK–Linear Algebra Package. Version 3.3.1, 2011. Available at: http://www.netlib.org/lapack.

[10] **R. Liutkevičius, A. Davidsonas.** Positioning of stepper motors in graphical data acquisition system. In: *Proceedings of International Conference "Electrical and Control Technologies – 2009", Kaunas*, 2009, pp. 70-74.

[11] **G. Liutkus, A. Riškus, A. Tomkevičius, A. Lenkevičius.** Issues with exchange of presentation data among CAD systems. In: *Information Technology and Control*, 2012, Vol. 41, pp. 385-391.

[12] **V. Matukas, M. Paulinas, A. Ušinskas, R. Adaškevičius, R. Meškauskas, D. A. Valinčius.** Survey of Point Cloud Reconstruction Methods. In: *Proceedings of International Conference "Electrical and Control Technologies – 2008", Kaunas,* 2008, pp. 150-153.

[13] **J. Schoenberg, A. Whitney.** On Pólya frequency functions. III. The positivity of translation determinants with an application to the interpolation problem by spline curves. *Transactions of the American Mathematical Society*, 1953, Vol. 74, pp. 246-259.

[14] **A. Watt.** 3D Computer Graphics. Third Edition. *Addison-Wesley Publishing*, 2000.

Received March 2012.