Преглед НЦД 4 (2004), 70–77

Žarko Mijajlović, **Boško Jovanović**,
Filip Marić, Miroslav Marić
(Matematički fakultet, Beograd)

MATLAB TOOLBOX FOR ANALYSIS OF 3D IMAGES

Abstract: Here we describe a MATLAB toolbox for analysis of 3D images obtained using a 3D scanner. Toolbox consists of two main parts: various filters for improving the image quality and a module for surface reconstruction from an unorganized set of points.

Introduction

At the Faculty of mathematics in Belgrade a project on analysis of 3D images now is running the second year. During the first year we dealt with recognition of linear surfaces, and we published the results in the paper [4]. 3D images are obtained using a 3D scanner for collecting space coordinates of points on the scanned surface. One of the aims of the project is to build a software for analysis of so obtained 3D images. For this purpose we have chosen MATLAB, the language and software package developed by *MathWorks* [5]. It allows us to solve in comfortable way some computational problems, especially those with matrix and vector formulations. Our software implementation of selected algorithms is organized in a toolbox, a specific MATLAB unit that extends the MATLAB environment. Possible applications in digitization include production of 3D images of various artifacts.

Acquisition of scanned data

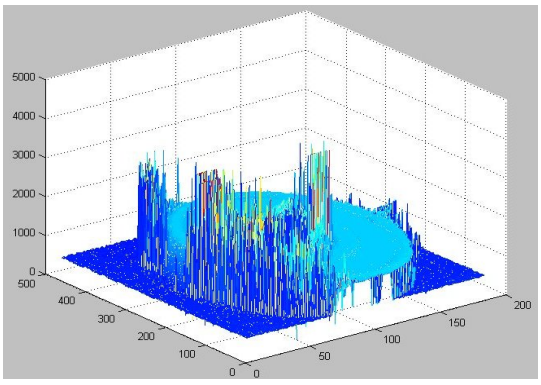

Figure 1: A scanned object - an image obtained by scanning

We have at our disposal a mechanical platform, a device which moves objects that are being scanned.

The scanner is adjusted to measure the distance from points of the object beneath it in regular time intervals. The speed of the platform can be adjusted, but it stays constant during scanning, in order to achieve the constant spatial density of points of the scanned surface. First spatial coordinates of contour points are collected and afterwards the collected data in the matrix form (*contour matrix*) are imported into MATLAB.

Since the scanned data may contain some errors, we have developed a set of filters for processing them. The object shown on Fig. 1 is a thin solid cylindrical disk. The scanning errors are shown as spikes.

Filters

This module consists of various filters. Filters should be applied on raw data before continuing the processing, as of recognition or determining of the type of the surface. Even if we were dealing with data in $\mathbb{R}^3$, we designed these filters starting from common filters for use on two-dimensional images that are described in [3].

Histogram. This is rather simple, but very useful and frequently used module. The histogram (Fig. 2) represents the distribution of the number of points in the z-axis zones. The interval $[\min z, \max z]$, where $\min z$, and $\max z$ are the minimal

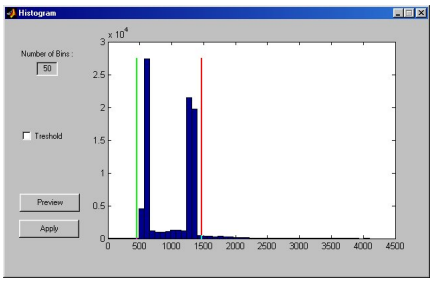

Figure 2: Histogram

and maximal z coordinates of the scanned points, is divided into a number of equally spaced bins, and the number of points that fall into each bin is determined. Under the assumption of the spatial coherence of the surface, bins that contain a small number of points usually contain only outliers and errors of the scanning process. So the user may remove all points belonging to bins with the number of points below a certain threshold value. This module also allows unconditional removing points at selected zones of raw data. Points are removed by inserting a MATLAB NaN (*not a number*) value into their position in the contour matrix.

Reduction of the number of points. The surface is often scanned on a too dense grid. This makes manipulation with a scanned image slow and difficult. This module allows a selection of appropriate set of representatives that preserve the geometric nature of the surface. It is very suitable if one wants fast preliminary processing of data. The algorithm divides the contour matrix to a number of rectangular blocks, and then replaces each block with a single point. This point represents the block and it can be the average of points contained in the block, the median (as shown on Fig. 3), or it can be obtained by some other linear statistics algorithm.

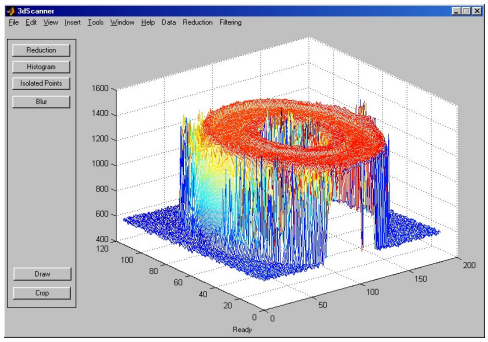

Figure 3: A scanned object - after performing density reduction

Isolation filters. These filters are used for removing isolated points (in $\mathbb{R}^3$), based on various criteria. For example, if it is given a neighborhood, the software determines the number of neighboring points for each point of the scanned image. If this number is under a specified threshold, the point is considered isolated and removed from the contour matrix. The simplified but useful version of this filter, considers the point isolated if its projection on XOY plane is isolated point in $\mathbb{R}^2$ sense.

Bluring filters. This group of filters consists of assorted functions for smoothing data. For each point a neighboring block is determined, and a new value is generated based on certain criteria. The filters include:

- **Averaging** blur – a new value is the average of values in a given block.
- **Gaussian** blur – gaussian linear filter. For each block, a weighted average is computed. The weights are approximation of two dimensional gaussian distribution with dispersion σ, so that the original value and the closest neighbors in the block have most influence on the filtered value.
- **Median** blur – a nonlinear filter which computes the median of points in a selected block.

The effect of these filters is shown on Fig. 4.

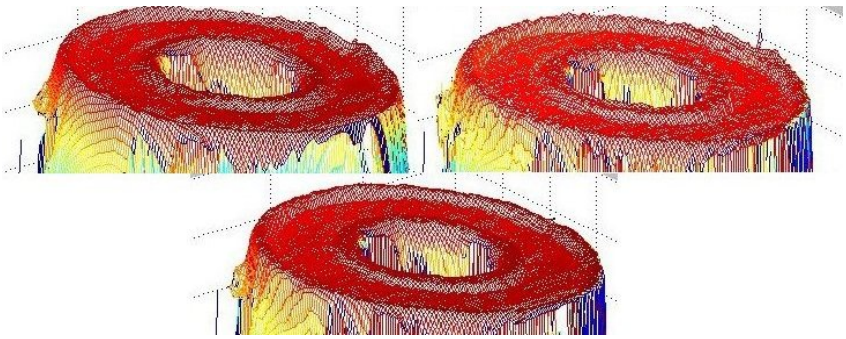

Figure 4: The effect of some blurring filters - top-left: averaging, top-right: gaussian, bottom: median

Edge detection. The Canny edge detection algorithm [6] is implemented. It behaves fine, even if it is originally proposed for flat images.

Surface reconstruction from a contour set

The goal of this module is stated as follows: *Given a set* $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^3$*, assumed to lie near an unknown surface* U*, create a surface model (triangular mesh) approximating* U*.*

Surface reconstruction algorithms are usually designed to exploit additional knowledge in specific problem instances (eg. topological type of the surface, structure of the data, orientation information, the fact that the data is noise-free etc). However, we decided to take a more general approach. We used a method proposed in [2], for automatic construction a triangular mesh from a set of points X where

- X is an unorganized, noisy sample of an unknown surface U.
- The unknown surface U can have (almost) arbitrary topological type, and may contain tangent plane discontinuities such as creases and corners.
- No other information about contour points are assumed.

We also suppose the following assumptions and constraints

- We do require for U to be a manifold (i.e. not self-intersecting surface).
- A noiseless sample $\{y_1, \ldots, y_n\}$ of U is said to be ρ-dense if each sphere with center in U and radius ρ contains at least one sampled point. A sample is said to be δ-noisy if each point x_i is of the form $x_i = y_i + e_i$, where $y_i \in U$ and $||e_i|| < \delta$. We assume that our sample is δ-noisy, ρ-dense. Therefore, a point $p \in \mathbb{R}^3$ cannot be a point of U if $d(p, X) > \rho + \delta$, where $d(p, X) = \inf_{q \in X} d(p, q)$, $d(p, q) = ||p - q||$.
- Features of U that are small compared to ρ or δ are obviously not recoverable. While it is acceptable to leave out such details, it is critical that the topological type of U is inferred correctly. Thus, we assume that no two "sheets" of U come closer together than $\rho + 3\delta$.

The algorithm overview. The main idea of the algorithm is to compute the *signed distance* from each point of $\mathbb{R}^3$ to the surface U. The signed distance is given by $d_U(p) = \pm d(p, U)$, where the sign depends on which side of the surface p lies. Knowing that, we can represent U as $\{p \in \mathbb{R}^3 : d_U(p) = 0\}$. Since U is unknown we have to approximate it. The key idea is to assign an oriented tangent plane approximation to each data point. The major obstacle is the finding globally coherent tangent plane orientation for the surface. Once achieving that, we could approximate $d_U(p)$ with the function $\tilde{d}_U : D \to \mathbb{R}$ which is defined in a region D near data points. It also represents a distance from a point to the nearest tangent plane approximation of U. Using this, and the well-known marching cubes algorithm [1], we form a cube grid, calculate $\tilde{d}_U$ for each vertex of the grid and finally extract the contour of U. Fig 5. shows different phases during the reconstruction of a torus-shaped surface.

Tangent plane estimation

A tangent plane of U at data point x_i is estimated and represented with a center c_i and the normal vector n_i. It is settled by finding a *least-squares approximation* plane for a group of points that lie in $\rho + \delta$ neighborhood of x_i which we denote by $\text{Nbhd}(x_i) = \{x \in X : d(x, x_i) < \rho + \delta\}$.

The brute force algorithm for finding this neighborhood takes $O(n^2)$ time, where n is the number of nodes. We improved this by adopting a simple space partitioning scheme. We divided the space into cubes and for each cube we store the pointers to the points that lie in it.

The c_i is taken to be the centroid of the neighborhood and n_i is determined using the *principal component analysis*. That means that the normal vector n_i is taken to be the eigenvector that belongs to the smallest-modulus eigenvalue of the covariance matrix of vectors from $\text{Nbhd}(x_i)$.

Tangent plane orientation

Using the mentioned technique, we determined tangent plane normals, but not their orientation. In order to achieve the second goal, we propose the following idea: traverse all normals constructed so far, starting from a fixed point and change the orientation of normals that face the wrong direction. The traversal should avoid sharp edges and corners as much as possible.

The first step is to choose one fixed orientation. We do this by choosing, for example, the highest center of a tangent plane c_i and orienting the corresponding normal towards above (i.e. $n_z > 0$). Now we have to propagate this orientation to all remaining normals.

For smooth surfaces, if two centers of tangent planes are geometrically close, corresponding normals should be nearly parallel i.e. for the scalar product of normals we have $n_i \cdot n_j \approx \pm 1$. If the dot product is close to -1, we should flip the normal. The problem that may arise is that U might have sharp edges and corners. If $|n_i \cdot n_j|$ is near 1 the situation is clear, but if it is not, we are dealing with ambiguity which should be avoided. We can reformulate this as a graph optimization problem. We assign a graph vertex to each tangent plane and connect the vertices whose centers are geometrically close i.e. where $d(c_i, c_j) < \rho + \delta$. In order to avoid a lot of new parameters, we use $\rho + \delta$ wherever it makes sense. To improve the speed, we adopt indexing scheme for tangent plane centers based on spatial partitioning, similar to the one described above. We assign a weight factor $1 - |n_i \cdot n_j|$ to each edge, so clear situations, where the surface is smooth, are favored in the graph traversal. The algorithm is based on construction of the *Minimal Cost Spanning Tree*. The greedy approach is used to overcome the NP-hard complexity of the algorithm that computes the most coherent orientation of all.

Signed distance function. We already said that we can estimate U with $\{p \in \mathbb{R}^3 : \tilde{d}_U(p) = 0\}$, where $\tilde{d}_U(p)$ is the approximation for the signed distance function $d_U(p)$ which gives the distance of an arbitrary point $p \in \mathbb{R}^3$ from the surface U. Its sign depends on which side of the surface p lies. Once we had determined tangent planes, it is easy to calculate $\tilde{d}_U(p)$ for any point of space. First,

we determine which of the tangent planes is the best local linear approximation of U near p. We do this by finding the tangent plane τ_i whose center c_i is closest to p. If there is no such plane, we leave $\tilde{d}_U(p)$ undefined. The tangent plane τ_i has normal vector n_i for which we assume that it is oriented correctly at the previous stage. Knowing this, we calculate the distance from p to τ_i by

$$\tilde{d}_U(p) = (p - c_i) \cdot n_i$$

In this way we obtain a piecewise linear approximation that it is not continuous – the discontinuities appear on places where we "switch"from one tangent plane to another. Fortunately, this does not affect the algorithm, because we use the contouring algorithm that discretely samples values of $\tilde{d}_U$, and creates a continuous piecewise linear approximation of U.

Contour extraction. We used in this part the well known *marching cubes* algorithm. The space is divided into a rectangular grid that consists of cubes. The resolution of the final surface estimation depends directly of the size of the cubes so it can be specified as a parameter.

The value $\tilde{d}_U$ is calculated for every vertex in the grid. Depending on its sign, it can be estimated if the vertex is "in"or "out"the surface. If signs differ for vertices on the same edge of a cube, we can conclude that there must be a point on the edge where $\tilde{d}_U = 0$. Then we adopt that the point belongs to the approximation of U.

For each cube, there are 2^8 possible combinations for vertex position, but many of these are geometrically equivalent i.e. one can be transformed into another using rotations, symmetries, and changing sign of $\tilde{d}_U$ on the vertices. In fact, there are only 15 nonequivalent cases.

The final result, a triangular mesh, is stored in a *polymesh* data structure i.e. the information about the vertices is stored separately from the information about their connectivity.

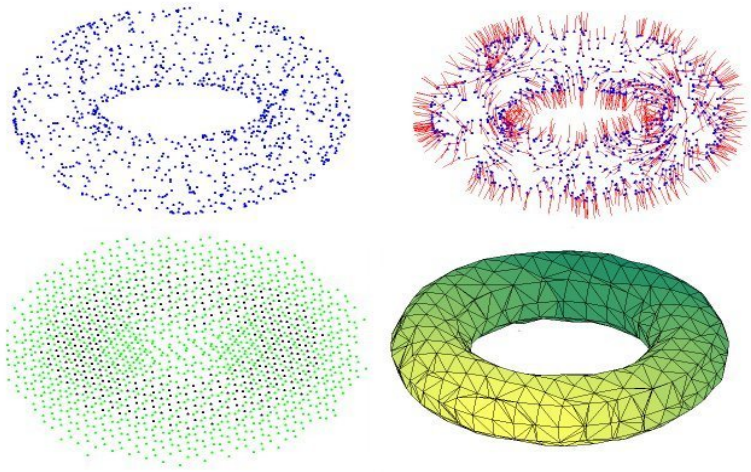

Figure 5: Various phases in surface reconstruction: 1. Raw data, 2. Oriented normals, 3. Interior/Exterior determination, 4. Reconstructed mesh

Applications in digitization

One of the most important goals in the field of digitization is presentation of cultural and scientific heritage in readable and useful form to the general and scientific auditorium. 3D scanning is another modern tool heavily based on computer technologies that may help to achieve this aim. Possible applications include production of 3D images of various artifacts, especially of the archeological origin. Compared with 2D images, 3D images give much more possibilities in detailed study of an artefact without destructive or offensive impacts. This includes the study of the shape, measurements along all three axes, and the choice of an arbitrary view on the object by rotating, zooming, moving around, putting textures, etc. It also gives opportunity for an examination of the artefact, dislocated in space and time. It provides a very simple way for producing of exact replica of the object, physical and virtual. Finally but not at least, 3D images give quite new dimension in visualization and esthetic comprehension of the (image) of the artefact.

As an illustration, we give a 3D image of a Roman mask ($3.4 \times 3.8 \times 1.8$cm). The image is produced by our software described in this article. Here are presented applications of assorted filters, and various views on the object. All views are derived from a single scanning. Images are shown on figures 6. and 7.

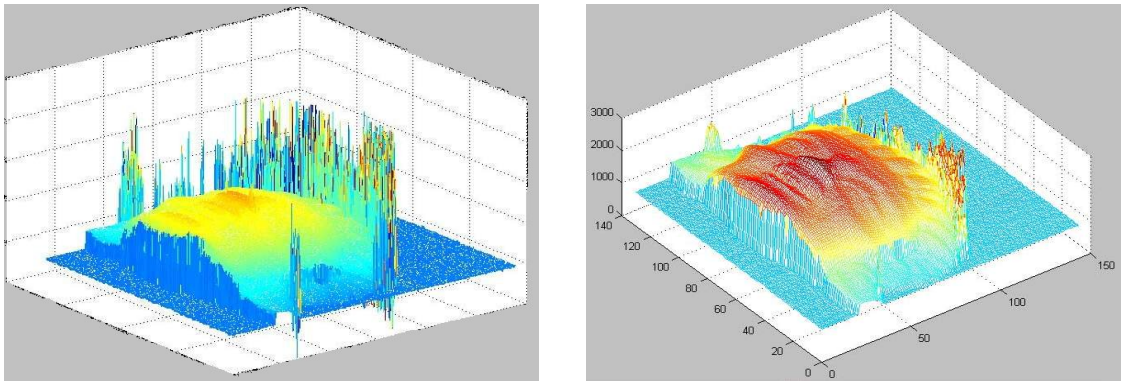

Figure 6: Raw data. Density reduction filter applied.

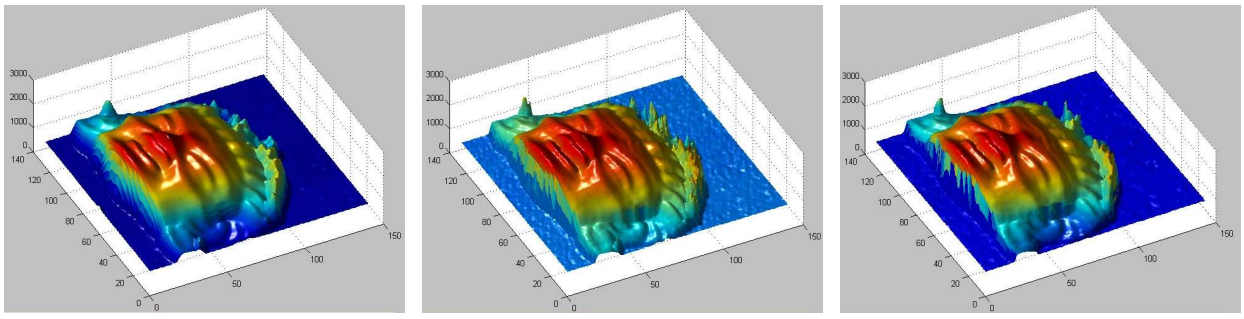

Figure 7: Averaging, Gaussian blur and Median filter applied

*

References

[1] William E. Lorensen and Harvey E. Cline, *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*, Computer Graphics (Proceedings of SIGGRAPH '87), Vol. 21, No. 4, pp. 163-169.

[2] Hugues Hoppe, *Surface reconstruction from unogranized points*, University of Washington, doctoral dissertation 1994.

[3] Martin J. Turner, Jonathan M. Blackledge, Patrick R. Andrews, *Fractal Geometry in Digital Imaging*, Academic Press, 1998.

[4] Žarko Mijajlović, Dragan Urošević, *An algorithm for recognition of linear surfaces*, Proceeding "CFT Varna 2002", Workshop "Multiscale approximations (wavelets, splines, RBF) and applications to Image and Signal Processing"

[5] http://www.mathtools.net/MATLAB/index.html, date of last access: 15.11.2003.,

[6] http://www.aai.com/AAI/IUE/taskodoc/Canny/Canny.html, date of last access: 15.11.2003.

zarkom@eunet.yu, bosko@matf.bg.ac.yu, filip@matf.bg.ac.yu, maricm@matf.bg.ac.yu

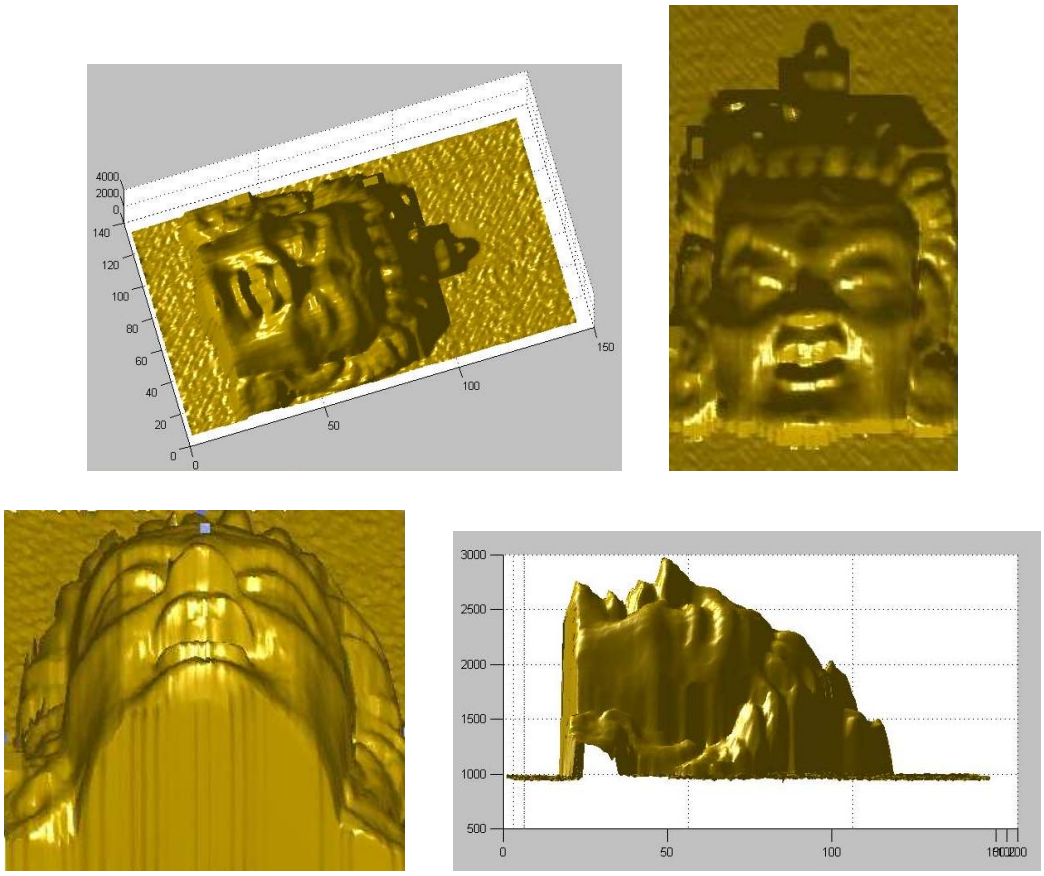

Figure 8: Final Result