Technical Sciences, 2020, 23(2), 115–130

UNIWERSYTET WARMIŃSKO-MAZURSKI W OLSZTYNIE

Quarterly peer-reviewed scientific journal
ISSN 1505-4675
e-ISSN 2083-4527
TECHNICAL SCIENCES
Homepage: www.uwm.edu.pl/techsci/

DOI: https://doi.org/10.31648/ts.5644

GEOMETRY EXTRACTION FROM GCODE FILES DESTINED FOR 3D PRINTERS

***Wojciech Kiński*[1], *Wojciech Sobieski*[2]**

[1]ORCID: 0000-0003-4973-7604
Department of Mechanical Engineering and Fundamentals of Machine Design
University of Warmia and Mazury, Olsztyn
[2]ORCID: 0000-0003-1434-5520
Department of Mechanical Engineering and Fundamentals of Machine Design
University of Warmia and Mazury, Olsztyn

Received 01 July 2020, accepted 22 July 2020, available online 22 July 2020.

Key words: GCODE, STL, additive manufacturing, 3D printers, reverse engineering.

Abstract

The paper presents a method of conversion of GCODE files designed for additive manufacturing in 3D printers to a format which may be conveniently visualized. In the investigations three different 3D models were created: a) shell model (a casing); b) solid model (a gear); c) model with curvilinear elements (a screw). All these models were converted to GCODE files. Next the reverse engineering was applied and GCODE files were converted to points sets. These points represent particular locations of the print head. In the developed algorithm the linear interpolation was added to obtain intermediate points between locations of the print head for longer sections. The final part shows an attempt of applying Poisson Surface Reconstruction in order to obtain the original geometry. The main motivation to develop a new software resulted from the observation that sometimes the original solid model is no longer available, while there is a need to change some geometry details or settings before production stage.

Correspondence: Wojciech Sobieski, Katedra Mechaniki i Podstaw Konstrukcji Maszyn, Wydział Nauk Technicznych, Uniwersytet Warmińsko-Mazurski, ul. M. Oczapowskiego 11, 10-957 Olsztyn, phone +48 (89) 523 32 40, e-mail: wojciech.sobieski@uwm.edu.pl.

Introduction

Nowadays the additive manufacturing (3D printing) are very popular and their importance grows every year. They are used in many areas of mechanical engineering (SHAHI 2016), material engineering (SZEBÉNYI et al. 2017), civil engineering (SHATORNAYA et al. 2017, TAY et al. 2017), architecture (MATHUR 2016), chemical sciences (PARRA-CABRERA et al. 2018), electronics (XU et al. 2016), medicine (HANGGE et al. 2018, TAPPA et al. 2017), food industry (GODOI et al. 2016, PITAYACHAVAL et al. 2018) and others. To obtain any printed object, in the first place its geometry, considering the limitations of the additive manufacturing technology, must be defined. To reach this aim, any software destined for preparing 3D geometry (in the STL file format) may be used. At this stage the assumptions related to the material used must be taken into account, too. It is also very important to note that virtual geometry cannot be printed directly with the use of a 3D printer. The geometry must be first converted to a set of instructions which will be sequentially read and executed by the printer (Fig. 1).

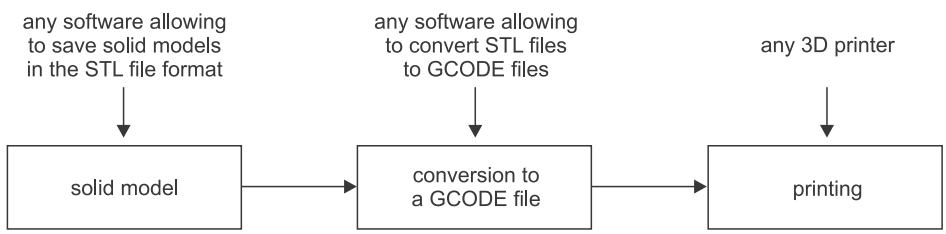

Fig. 1. Data flow from a solid model to the finished product

In particular, the movement of the print head and the material feeding speed must be specified. Here, a specific programming language, the so-called GCODE, is usually used (also RS-274; KRAMER et al. 2000). GCODE is a standardized command language that is used to operatenumerically controlled devices (CNC) (ISO 6983-1:2009). The main motivation to develop a new software resulted from own practical experience. It turns out that sometimes the original 3D solid model is no longer available, but a need arises to change some details of the geometry or settings of the printing process. Two options are possible in such a case: creation of the solid model from the beginning or converting the existing GCODE file to a format, which may be modify. The hereby paper is a try to apply the second approach. The possibility to save the time needed to prepare a second solid model is the main advantage of the idea presented.

The methodology applied covers four steps:

– preparing exemplary geometries in the STL format;
– converting STL files to GCODE files;

– decoding GCODE files to obtain a points set representing the subsequent coordinates of positioning the print head;

– surface reconstructing on the basis of the points set.

It should be noted that there are publications related to data conversion from STL files to GCODE files (so-called slicing) (GUERRERO-DE-MIERA et al. 2015, HU 2017, TOPÇU et al. 2011) or printing quality (BAUMANN et al. 2016, WANG et al. 2014) as well as articles on reverse engineering dealing with the use of 3D printers and scanners (BARONIO et al. 2016, DÚBRAVČÍK, KENDER 2012, ESLAMI 2017). However, among them the papers related to decoding of the GCODE language are very rare. Apparently, to the best of our knowledge, there is only one paper alluding to this issue (BAUMANN et al. 2017). In this paper a reconstruction algorithm is briefly presented. The Authors use similar methodology to ours, but the tools applied by them are different and they illustrate the algorithm with only one example. In particular, the software for slicing a STL model was different (Slic3r, v. 1.2.9 and the main programming language was Python). We tried to compare our code with the software developed by them, but it turned out, that the webpage indicated by the Authors in the paper (BAUMANN et al. 2017) is not available.

Materials and Methods

Materials

The object of analysis is a set of three files in the GCODE format which are intended for the execution of selected geometric objects with the use of 3D printing technique. Three basic types of shapes were taken into account (i.e. flat, solid and curved) and the most important instructions used in source files for 3D printers.

GCODE programming language

The GCODE language is a standardized command language that is very oft used to operate numerically controlled (CNC) devices (ISO 6983-1:2009). The source code in GCODE language consists of a series of individual instructions which, depending on the device, are able to make details by subtractive (e.g. CNC milling machine) or additive (3D printer) manufacturing. The GCODE commands (Tab. 1) can be created by using the following methods:

– creating a list of commands in a text editor;

– preparing the code using CAM software;

– entering commands directly via the control panel of the device.

In practice, due to the complexity of the geometry of the processed model, it is mainly the software dedicated to a given device class that is used. In the context of the paper important is, that programs "cutting" three-dimensional objects mainly use the commands of simple linear motions. They do not use G2 and G3 commands that use circular motion. The cutting software treats each arc as a set of straight sections.

Table 1

Examples of GCODE language commands used in 3D printers

Command	Description	Sample
G0	Fast movement	G0 X12
G1	Work movement by linear interpolation	G1 X90.6 Y13.8 E22.4
G2	Circular motion in a clockwise direction	G2 X90.6 Y13.8 I5 J10 E22.4
G3	Movement around the circle counter-clockwise	G3 X40.6 Y11.7 I5 J10 E72.4
G28	Axis reset	G28

Source: based on HABRAT (2007).

Poisson Surface Reconstruction

There are two groups of surface reconstruction methods based on a points set: explicit methods, in which the surface must pass through all points of the set, and implicit ones in which the surface is approximated on the basis of on the points locations (NORLANDER 2017). However, due to the drawbacks of the explicit methods (which are usually based on the Delaunay algorithm) such as high sensitivity to data quality and high demand for computing power, there were developed implicit methods, such as MarchingCubes (LORENSEN, CLINE 1987) or Poisson Surface Reconstruction (KAZHDAN et al. 2006), that are most commonly used since they are more resistant and faster.

The Poisson Surface Reconstruction is based on calculating the so-called indicator function X (a scalar function), whose value is one (inside the surface) and zero (outside the surface). If this function is known, then the reconstructed surface may be obtained by extracting an appropriate iso-surface. In the Poisson method an oriented points set can be treated as a sampling of the gradient of the indicator function. In the consequence, the discrete points set may be used to define a continuous vector field $\vec{V}$ representing the gradient field of the indicator function. Solving the indicator function X then amounts to finding the scalar function whose gradient best matches $\vec{V}$. To reach this aim, the Poisson equation must be solved:

$$\Delta X = \nabla \cdot \vec{V} \tag{1}$$

where:

$\vec{V}$ – the smoothed normal field.

Results and Discussion

Solid models

The aim of the first stage of investigations is to create a set of models of three-dimensional objects, taking into account the most common geometric features of elements made on 3D printers. For testing, one thin-wall structure (casing – Fig. 2*a*), one full structure (gear – Fig. 2*b*) and one structure containing curved sections (screw – Fig. 2*c*) were prepared. The geometries were created in the SolidWorks program and then saved in the form of STL files. The dimensions of objects in the context of this study are irrelevant.

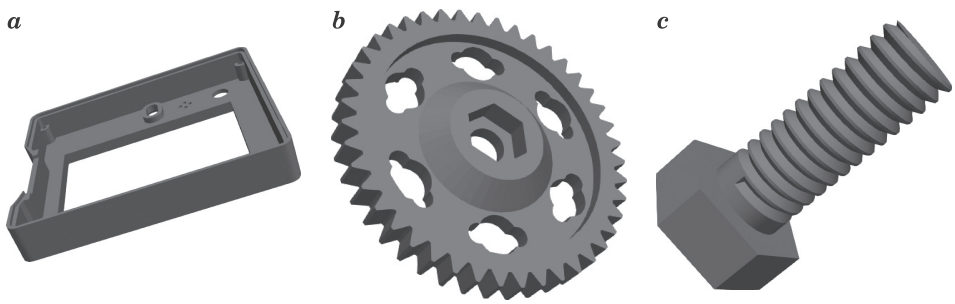

Fig. 2. 3D models used in investigations; *a* – casing, *b* – gear, *c* – screw

Generation of sample GCODE files

Having prepared the geometry of selected objects in the form of STL files, in the second stage of the investigations three GCODE files were created – one for each geometry. These files were prepared with the use of the MatterControl program (MatterControl Home Page 2019). This is a free Open Source application that is used to construct 3D objects for 3D printing. The preparation of the GCODE file requires a number of parameters characterizing a specific device as well as the properties of the 3D printing process itself, such as the layer height or wall thickness. A typical GCODE source file contains the following sections:

– header – consists of several lines of comments (always beginning with a semicolon) informing about the version of the software used, the wall thickness of the model, the width of the extruded path and the diameter of the beam of the material being fed;

– pre-processing data – consists of commands informing about the type and location of the centre of the coordinate system used and of commands used to prepare the print head and working platform (e.g. heating up to the set temperature);

– work area – consists of rows which specify the next positions of the print head and the current material feeding speeds;

– print completion area – consists of commands to be made after printing the model, such as axis homing, turning off the print head heaters and the work platform;

– footer – consists of lines informing about print parameters, such as material feeding speed, type and density of the model filling and many others.

```
; Generated with MatterSlice 1.0
; filamentDiameter = 1.75
; extrusionWidth = 0.4
; firstLayerExtrusionWidth = 0.52
; layerThickness = 0.2
; firstLayerThickness = 0.3
; automatic settings before start_gcode
G21 ; set units to millimeters
M107 ; fan off
T0 ; set the active extruder to 0
; settings from start_gcode
G28 X0;
G28 Y0;
G1 X82 Y82 F5000;
G28 Z0;
G1 Z5 F5000;
; automatic settings after start_gcode
T0 ; set the active extruder to 0
G90 ; use absolute coordinates
G92 E0 ; reset the expected extruder position
M82 ; use absolute distance for extrusion
; Layer count: 45
; Layer Change GCode
; LAYER:0
M400
M107
G0 F7800 X0 Y0 Z0.3
G0 X45.711 Y45.733
G0 X58.561 Y56.444
G0 X59.362 Y57.337
G0 X60.343 Y57.86
G0 X63.469 Y65.796
G0 X63.223 Y66.21
G0 X62.918 Y66.229
G0 X62.087 Y66.421
```

Fig. 3. Fragment of the GCODE language code for printing a model of a gear

Typically, for one specific printing device, the information necessary for the preparation and completion of printing is entered once (Tab. 2). This information is then duplicated for the subsequently generated GCODE files. In the context of the current article, it is important to emphasize that individual sections do not have any markers defining their scope. The beginnings and ends of the section

are determined by analysing the instructions of the following lines. An initial part of a source file taken from one of the examples described in the further part of the paper is shown in Figure 3. In this figure, the work area begins with the 27th line of code (highlighted).

Table 2

Parameters characterizing the generated GCODE files

Name	Value
Layer height	0.3 [mm]
Infill	20 [%]
Type of infill	grid an angle 45
Permiters	2
The number of bottom layers completely filled	4
The number of top layers completely filled	4

A characteristic feature of GCODE files is that full objects are usually printed in the form of a shell filled with a lattice structure (with different shapes of such structures). This saves the material and significantly speeds up the printing process. An example of a model of a gear with a truss type fill is shown in Figure 4 (mid-model view).

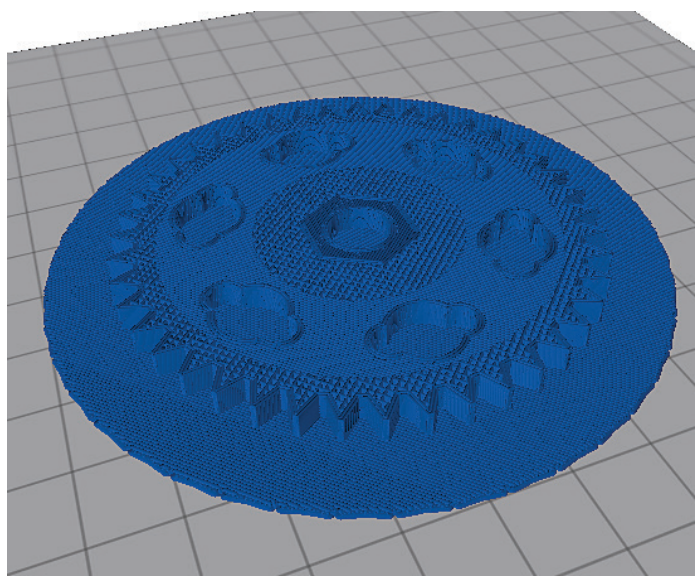

Fig. 4. View of the gear model in the MatterControl program

Geometry extraction

Currently available computer programs allow the visualization of GCODE files (jView 2020, NC Viewer 2020). However, they show usually not the geometry of the three-dimensional object but the trajectory of the printing head. For this reason they are not enough functional, if the User want re-select individual 3D printing parameters (e.g. specify the type and type of filling or change the height of the model layer).

In order to reproduce the geometry of an object for 3D printing based on a GCODE file, the coordinates of the subsequent positions of the print head must be known. Other parameters, such as filler temperature or feed speed, do not matter, but they can be useful for locating characteristic code fragments. The basis of the algorithm used in this investigation is to search for and then store the X, Y and Z coordinate values in the subsequent lines of code. To realize this, subsequent lines of the source code are read (as individual text variables), and then by means of text functions of a given programming language, the existence of specific sequences of characters in these lines is checked. The search starts with a line containing the "LAYER: 0" sequence (denoting the beginning of printing of the first layer) and ends with the line of the "M104 S0" sequence (which means switching off the heating of the print head, and consequently the end of the printing process). The numbers of these lines are denoted by variables n_{start} and n_{stop}, respectively. It is important to note that the Z coordinate of the first layer (denoted by Z0 symbol) may be sometimes defined before the n_{start} line. Therefore, it should be always checked if this is the case and remember the initial value of this coordinate.

Scanning a single line of the code, understood as a multi-character text variable, includes the following steps:

– checking if the coordinate mark (X, Y or Z) is present in the line;

– finding and saving the characters located between the coordinate mark and the nearest space or between the coordinate mark and the end of the line;

– converting characters into numbers and saving the result.

It is important to make sure that each line of the code is checked several times in order to find the current coordinate values X, Y and Z one by one. If the coordinate mark does not exist in the line being scanned, then the current coordinate value is taken as the previously read value. This means that the print head moves parallel to one of the Cartesian coordinate axes. In such cases, there is no need to duplicate the same coordinate, so that the files have a smaller volume. The values of read variables can be saved directly to a file or can be stored in indexed variables. The size of such variables can be determined on the basis of the knowledge of n_{start} and n_{stop}.

A Fortran (GNU Fortran Home Page 2019) code was written to decode GCODE files, using internal SCAN and INDEX text functions. The first function allows

to determine the position of a single character in a text string (e.g. character *X*), the second function determines the position of a particular character subsystem (e.g. the "LAYER: 0" sequence). The resulting data, i.e. the coordinates of all points found, were saved in the form of ASC and VTK files (The Visualization Toolkit 2019). The code that is intended to read the *X* coordinate value in a given line of the GCODE file has the following form:

```
!if the line in the GCODE file begins from G0 or G1:
if((index(line,'G0')/=0).or.(index(line,'G1')/=0))then
 !find position of X sign (0 means that this sign do not exists in this line):
posX=scan(line,'X')
 !if the X sign exists:
 if(posX>0)then
 !find the position of a space occurring after the X sign:
posNull=scan(line(PosX:),char(32))
 !save all signs from the X sign to a next space located in the same line:
lineX=line(posX:posX+posNull-1)
 !extract the numbers occurring after the sign X:
 write(tmp,fmt='(A)')lineX(2:)
 !assign the read out number to the current X coordinate:
 read(tmp,*)x(lp)
 !remember the last known value of the X coordinate:
xw=x(lp)
 !if the X sign do not exists:
 else
 !assign the previous value to the current X coordinate:
 x(lp)=xw
endif
endif
```

The meaning of variables is as follows: "line" – a character variable containing the current line of the GCODE file; "posX" – an integer variable containing the position of X sign in the variable "line"; "posNull" – an integer variable containing the position of a space in the variable "line" (here the space located after X sign); "lineX" – a character variable containing a substring of the variable "line" (here X sign together with numbers occurring after this sign); "lp" – counter of points in the model; "x(lp)" – a float variable containing the current value of the X coordinate; "xw" – a float variable containing the last remembered value of the *X* variable.

In addition to the three basic GCODE files, during the development of the algorithm, other variants of these files were also tested. The variants were obtained with different settings of the MatterControl program. Depending

on the settings, the movement of the print head can be described with instructions G0 or G1 (see Table 1). Another difference appears in the length and content of the GCODE file header or footer, in particular the method and location of the Z0 coordinate. The program has been written to take into account as many possible differences as possible.

In Figures 5*a*, 6*a* and 7*a* there are examples of visualization of subsequent positions of the print head obtained after decoding the selected GCODE files. Visualizations were made in the ParaView (ParaView Home Page 2019) with the use of the automatically generated VTK files. The overall shape of the object is visible, but some walls are clearly missing. This effect results from the fact that if the print head moves along a straight line (regardless of its length), only the coordinates of the beginning and end of this line are saved in the GCODE file. To improve the reproduction quality of the original geometry with a pointsset, it is preferable to insert intermediate points. In the developed program, in relation to straight lines, this stage consists of the following steps:

– length calculation of the section between the current (i) and previous ($i-1$) location of the print head:

$$l = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (z_i - z_{i-1})^2};$$

– calculation of the number of intermediate points:

$$n_l = \mathrm{NINT}\left(\frac{l}{dl}\right),$$

where dl is the maximum spatial step defined by the user;

– calculation of increments of individual coordinates:

$$dx = \frac{x_i - x_{i-1}}{n_l}, dy = \frac{y_i - y_{i-1}}{n_l}, dz = \frac{z_i - z_{i-1}}{n_l}.$$

In the last step the coordinates of intermediate points are calculated:

$$x_j = x_{i-1} \cdot j \cdot dx\,,\ y_j = y_{i-1} \cdot j \cdot dy,\ z_j = z_{i-1} \cdot j \cdot dz\,,$$

where j denotes the sequence number of the intermediate point, whereby $j \in \langle 1, n_l \rangle$.

Before preparing the output files, the data should be also checked for any duplicate points. If they are found, such points should be removed.

The disadvantage of inserting intermediate points is a significant increase in the number of points representing the geometry of the object. For $dl = 0.5$ the total number of points for the casing, gear and screw increased from 52,926 to 186,910, from 64,447 to 233,981 and from 11,151 to 12,776, respectively. This is 3.532, 2.127 and 1.146 times more points, respectively. This condition

can lead to difficulties in visualizing such a large points set. The effect deepens with the decrease of *dl*. It should be added that difficulties in visualizing large sets of points (a long waiting time for a program to response) occurred only in the ParaView software. The MeshLab program (MeshLab Home Page 2019), described later in the article, worked much faster.

In Figures 5*b*, 6*b*, 7*b* the final results are visible. They were prepared in the ParaView software with the use of the "glyph" tool. This tool allows to draw a simple 3D object (arrow, cone, box, cylinder, line, sphere) in every point with a specified location. In the figures mentioned, cubes (Fig. 5 and 6) and spheres (Fig. 7) were used to prepare the visualisation. In such a way a set of points may be visualized as a sets of solids. Such a result may be saved, among others, in the STL file format. Unfortunately, such a file consists of many separate objects (cubes and spheres) and cannot be applied directly in the same way as the original STL file. All tries of merging these objects into one surface or solid body failed.

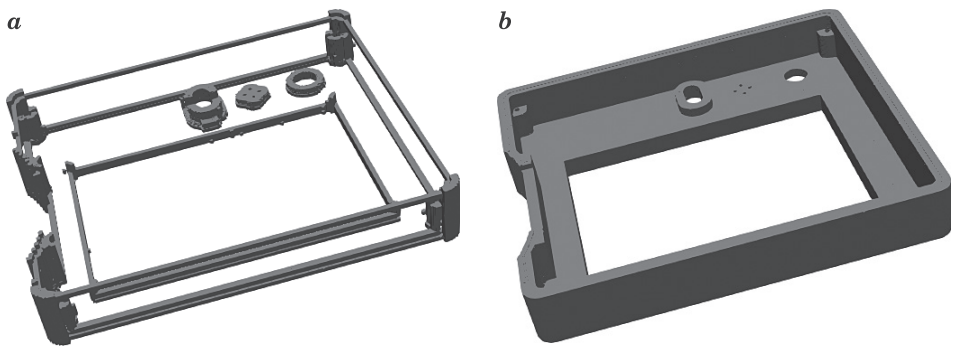

Fig. 5. Example of visualization of print head positions without interpolation (*a*) and with interpolation, at $d_l = 0.2$ (*b*)

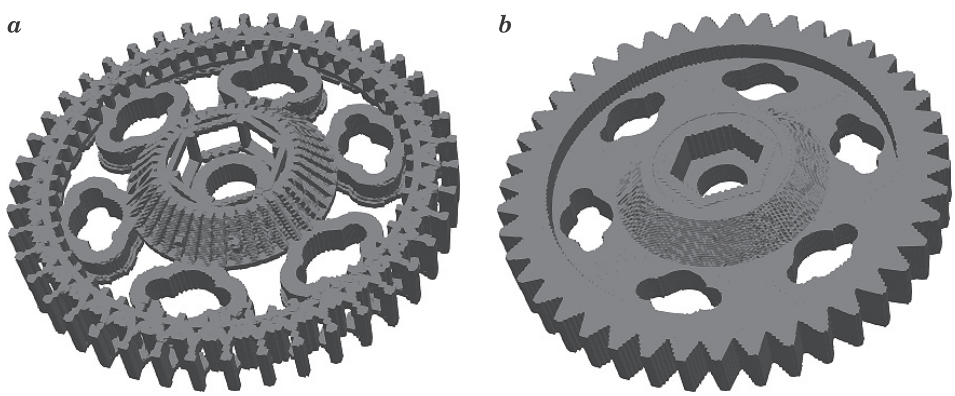

Fig. 6. Example of visualization of print head positions without interpolation (*a*) and with interpolation, at $d_l = 0.2$ (*b*)

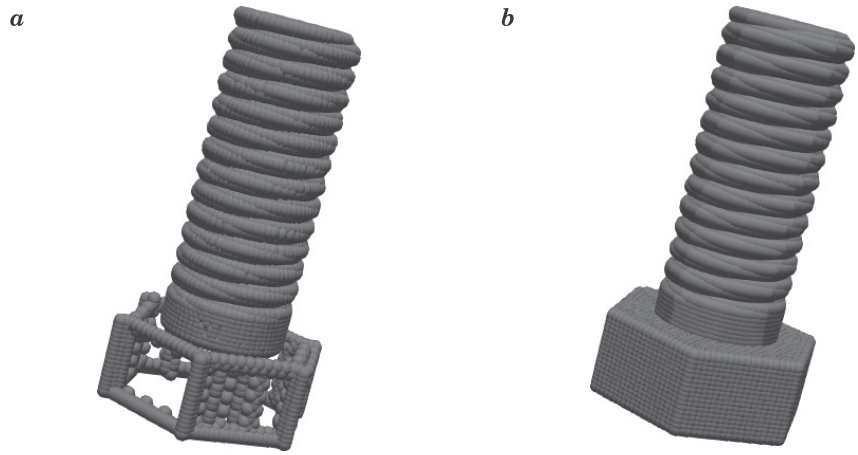

Fig.7. Example of visualization of print head positions without interpolation (*a*) and with interpolation, at $d_I = 0.5$ (*b*)

The results obtained from the extraction of geometry, in the version with interpolation of intermediate points, were also saved in the form of text files in the ASC format. This format consists of only three columns of numbers, containing the *X*, *Y* and *Z* coordinates of all the points of the set.

Poisson Surface Reconstruction

The previously mentioned MeshLab (MeshLab Home Page 2019) application was used to carry out the next stage of work. The choice of this particular application was dictated by the fact that it allows to load data in many different file formats, among others in the ACS format. What is more, the program can be used, for instance, to perform the earlier mentioned Poisson Surface Reconstruction.

In the case of dense points sets, such as those obtained in the previous stage, the local number of points should be reduced. It may be done by selecting an appropriate tool from the main menu: "Filters – Cleaning and Repairing – Merge Close Vertices". Now the normal vectors to the surface in all points of the current set may be calculated by selecting the following from the main menu: "Filters – Normal, Curvatures and Orientation – Compute normals for points sets". The last step is to select the Poisson Surface Reconstruction tool from the main menu. This is done as follows: "Filters – Remeshing, Simplification and Reconstruction – Screened Poisson Surface Reconstruction". It should be added that few constants have to be defined while using these tools. The best results were obtained for the parameters that were listed in Table 3.

Table 3

Surface reconstruction parameters in the MeshLab program

MeshLab Filter			Casing	Gear	Screw
Cleaning and Repairing	Merge Close Vertices	perc on	150	100	180
Normal, Curvatures and Orientation	Compute normals for points sets	Neighbour num	10	10	50
		Smooth Iteration	0	0	0
Remeshing, Simplification and Reconstruction	Screened Poisson Surface Reconstruction	Reconstruction Deepth	10	8	8
		Minimum Number of Samples	10	4	3
		Integration Weight	4	10	6

In Figure 8*a* there is the starting points set for the casing model, and to the Figure 8*b* the reconstructed 3D object is visible. Unfortunately, despite many attempts, it was not possible to reconstruct the surface in such a way that it would accurately represent the output objects. In particular, it was not possible to obtain STL files that could be later modified and reused in the printing process. Figures 9 and 10 show the reconstruction of the gear model and screw.

It is clearly visible, that the geometry visible in Figures 8, 9 and 10 do not correspond with the original 3D models (Fig. 2). It means, that the Poisson Surface Reconstruction is not a good method to obtain a file, which may be again used as the input file in printing process. The investigation should focus rather on the problem described in the previous section, consisting of merging many separate objects into one surface or solid body.

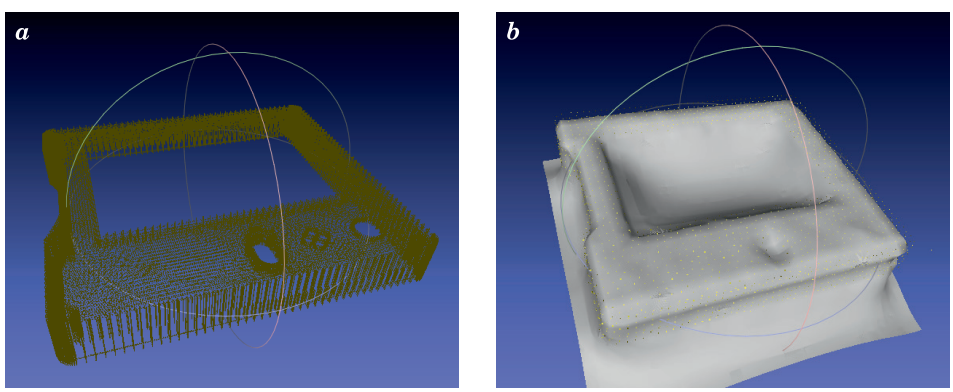

Fig. 8. Casingmodel; *a* – pointsset, *b* – reconstructed model

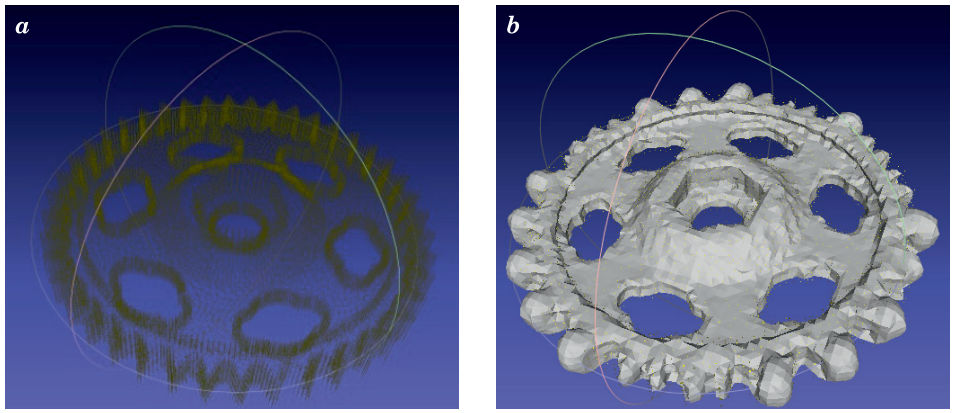

Fig. 9. Model of the gear in the form of the starting pointsset (*a*) and the reconstructed area (*b*)

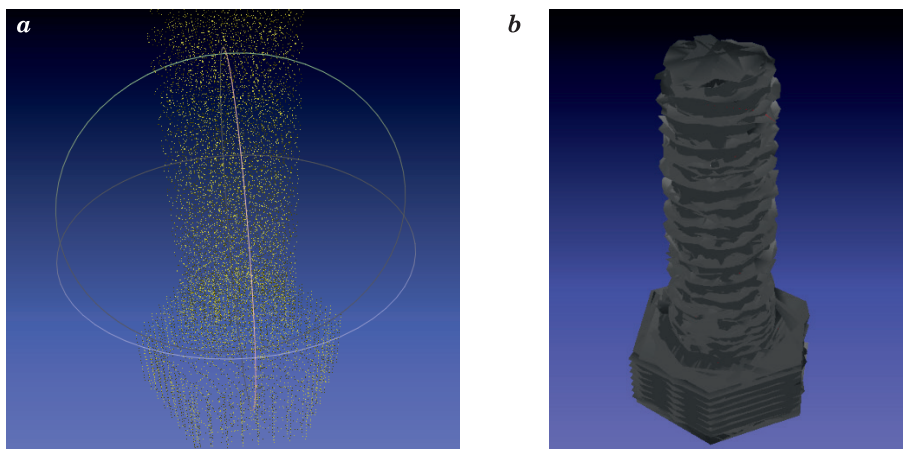

Fig. 10. Screw model; *a* – pointsset, *b* – reconstructed model

Summary

The main observations and final conclusions are as follows:

– It is possible to transform a geometry saved in a GCODE file to a points set. In turn, this points set may be visualised (and saved in the STL file format) by the use of appropriate software, e.g. ParaView.

– The movement of the print head must be interpolated to reconstruct filled structures.

– Duplicates of points in points sets should be detected and removed.

– The decoding process is hindered by the fact that from one geometry many differing GCODE files may be obtained (it depends on settings of the algorithm

converting the STL files to the GCODE files). This problem sometimes results in the situation when the points appear outside the original geometry. It is difficult to develop a universal algorithm to filter such points which could be applied to all possible variants of GCODE files.

– The quality of the reconstructed surface depends on the parameters introduced in the MeshLab program. There are no universal parameters that reproduce all models in the same way (each model has a different geometry and needs different parameters).

– On the current stage the algorithm described may be used only to view the geometry saved in a GCODE file. The main advantage is, that the source code of the developed software is very short and concise and may be easy modify. No access to the Internet is needed, too.

– Resulting files with coordinates of points belonging to a point cloud are usually very large in volume (in the order few hundred of MB). It causes that many software crashes during manipulation with such a files. This problem do not occurs in the developed software.

References

Baronio G., Harran S., Signoroni A. 2016. *A Critical Analysis of a Hand Orthosis Reverse Engineering and 3D Printing Process*. Applied Bionics and Biomechanics, 2016: 1-7, article ID 8347478.

Baumann F., Bugdayci H., Grunert J., Keller F., Roller D. 2016. *Influence of slicing tools on quality of 3D printed parts*. Computer-Aided Design & Applications, 13(1): 14-31.

Baumann F.W., Schuermann M., Odefey U., Pfeil M. 2017. *From GCODE to STL: Reconstruct Models from 3D Printing as a Service*. IOP Conf. Series: Materials Science and Engineering, 280: 012033.

Dúbravčík M., Kender Š. 2012. *Application of reverse engineering techniques in mechanics system services*. Procedia Engineering, 48: 96-104.

Eslami A.M. 2017. *Integrating Reverse Engineering and 3D Printing for the Manufacturing Process*. American Society for Engineering Education, Paper ID #18869.

GNU Fortran Home Page. 2018. https://gcc.gnu.org/fortran/ (access: 18.10.2018).

Godoi F.C., Prakash S., Bhandari B.R. 2016. *3D printing technologies applied for food design: Status and prospects*. Journal of Food Engineering, 179: 44-54.

Guerrero-de-Miera A., Espinosa M.M., Domínguez M. 2015. *Bricking: A new slicing method to reduce warping*. Procedia Engineering, 132: 126-131.

Habrat W. 2007. *Obsługa i programowanie obrabiarek CNC. Poradnik operatora.* Wydawnictwo KaBe, Krosno.

Hangge P., Pershad Y., Witting A.A., Albadawi H., Oklu R. 2018. *Three-dimensional (3D) printing and its applications for aortic diseases*. Cardiovascular Diagnosis and Therapy, 8: 19-25.

Hu J. 2017. *Study on STL-Based Slicing Process for 3d Printing*. Proceedings of the 28th Annual International, Solid Freeform Fabrication Symposium – An Additive Manufacturing Conference. Austin TX, August 7-9, 11 p.

ISO 6983-1:2009: Automation systems and integration – Numerical control of machines -- Program format and definitions of address words – Part 1: Data format for positioning, line motion and contouring control systems. https://www.iso.org/standard/34608.html (access: 18.10.2018).

jView. 2020. http://www.jtronics.de/software/jview-simple-g-code-viewer/ (access: 1.04.2020).
KAZHDAN M., BOLITHO M., HOPPE H. 2006. *Poisson surface reconstruction*. Proceedings of the 4th Eurographics Symposium on Geometry Processing, Sardinia, Italy, p. 1-10.
KRAMER T.R., PROCTOR F.M., MESSINA E. 2000. *The NIST RS274NGC Interpreter – Version 3*. NISTIR 6556, August 17, p. 121.
LORENSEN W.E., CLINE H.E. 1987. *Marching cubes: a high resolution 3D surface construction algorithm*. ACM SIGGRAPH Computer Graphics, 21(4): 163-169.
MATHUR R. 2016. *3D Printing in Architecture*. International Journal of Innovative Science, Engineering & Technology, 3(7): 583-591.
MatterControl Home Page. 2019. https://www.matterhackers.com/ (access: 10.04.2019).
MeshLab Home Page. 2019. http://www.meshlab.net/ (access: 10.04.2019).
NC Viewer. 2020. https://ncviewer.com/ (access: 11.04.2020).
NORLANDER R. 2017. *Make it Complete: Surface Reconstruction Aided by Geometric Primitives*. http://liu.diva-portal.org/smash/get/diva2:1153573/FULLTEXT01.pdf.
ParaView Home Page. 2019. https://www.paraview.org/ (access: 10.04.2019).
PARRA-CABRERA C., ACHILLE C., KUHN S., AMELOOT R. 2018. *3D printing in chemical engineering and catalytic technology: structured catalysts, mixers and reactors*. Chemical Society Reviews, 1.
PITAYACHAVAL P., SANKLONG N., THONGRAK A. 2018. *A Review of 3D Food Printing Technology*. MATEC Web of Conferences, 213: 01012.
SHAHI B.S. 2016. *Advanced Manufacturing Techniques (3D Printing)*. International Journal of Mechanical And Production Engineering, 4(4): 16-23.
SHATORNAYA A.M., CHISLOVA M.M., DROZDETSKAYA M.A., PTUHINA I.S. 2017. *Efficiency of 3D printers in Civil Engineering*. Construction of Unique Buildings and Structures, 9(60): 22-30.
SZEBÉNYI G., CZIGÁNY T., MAGYAR B., KARGER-KOCSIS J. 2017. *3D printing-assisted interphase engineering of polymer composites: Concept and feasibility*. eXPRESS Polymer Letters, 11(7): 525-530.
TAPPA K, JAMMALAMADAKA U., BALLARD D.H., BRUNO T, ISRAEL M.R., VEMULA H., MEACHAM J.M., MILLS D.K., WOODARD P.K., WEISMAN J.A. 2017. *Medication eluting devices for the field of OBGYN (MEDOBGYN): 3D printed biodegradable hormone eluting constructs, a proof of concept study*. PLoSONE, 12(8): e0182929.
TAY Y.W.D., PANDA B., PAUL S.C., MOHAMED N.A.N., TAN M.J., LEONG K.F. 2017. *3D printing trends in building and construction industry: a review*. Virtual and Physical Prototyping, 12(3), 261-176.
TOPÇU O., TAŞCIOĞLU Y., ÜNVER H.Ö. 2011. *A Method for Slicing CAD Models in Binary STL Format*. 6th International Advanced Technologies Symposium (IATS'11), 16-18 May 2011, Elazığ, Turkey.
VTK – The Visualization Toolkit. https://www.vtk.org/ (access: 10.04.2019).
WANG W., CHAO H., TONG J., YANG Z., TONG X., LI H., LIU X., LIUY L. 2014. *Saliency-Preserving Slicing Optimization for Effective 3D Printing*. COMPUTER GRAPHICS forum, 33(5): 1-12.
XU Y., WU X., GUO X., KONG B., ZHANG M., QIAN X., MI S., SUN W. 2016. *The Boom in 3D-Printed Sensor Technology*. Sensors, 17: 37.