Gerber File Parsing for Conversion to Bitmap Image–The VINCI7D Case Study

Article *in* IEEE Access · June 2022

DOI: 10.1109/ACCESS.2022.3187042

5 authors, including:

Ricardo B. Sousa
Institute for Systems and Computer Engineering, Technology and Science (INESC …
31 PUBLICATIONS **116** CITATIONS
SEE PROFILE

Cláudia D. Rocha
Institute for Systems and Computer Engineering, Technology and Science (INESC …
16 PUBLICATIONS **30** CITATIONS
SEE PROFILE

Hélio Mendonça
University of Porto
26 PUBLICATIONS **69** CITATIONS
SEE PROFILE

A. Paulo Moreira
University of Porto
321 PUBLICATIONS **4,242** CITATIONS
SEE PROFILE

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 May 2022, accepted 20 June 2022, date of publication 29 June 2022, date of current version 6 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3187042

APPLIED RESEARCH

Gerber File Parsing for Conversion to Bitmap Image–The VINCI7D Case Study

RICARDO B. SOUSA[1,2], CLÁUDIA ROCHA[2], HÉLIO SOUSA MENDONÇA[1,2], ANTÓNIO PAULO MOREIRA[1,2], AND MANUEL F. SILVA[2,3]

[1]Electrical Engineering Department, Faculty of Engineering, University of Porto, 4200-465 Porto, Portugal
[2]INESC TEC—Institute for Systems and Computer Engineering, Technology and Science, 4200-465 Porto, Portugal
[3]ISEP—School of Engineering of the Porto Polytechnic, 4200-072 Porto, Portugal

Corresponding author: Ricardo B. Sousa (up201503004@fe.up.pt)

This work was supported by the National Funds through the Portuguese Funding Agency, FCT—Fundação para a Ciência e a Tecnologia, under Project LA/P/0063/2020.

ABSTRACT The technological market is increasingly evolving as evidenced by the innovative and streamlined manufacturing processes. Printed Circuit Boards (PCB) are widely employed in the electronics fabrication industry, resorting to the Gerber open standard format to transfer the manufacturing data. The Gerber format describes not only metadata related to the manufacturing process but also the PCB image. To be able to map the electronic circuit pattern to be printed, a parser to convert Gerber files into a bitmap image is required. The current literature as well as available Gerber viewers and libraries showed limitations mainly in the Gerber format support, focusing only on a subset of commands. In this work, the development of a recursive descent approach for parsing Gerber files is described, outlining its interpretation and the renderization of 2D bitmap images. All the defined commands in the specification based on Gerber X2 generation were successfully rendered, unlike the tested commercial parsers used in the experiments. Moreover, the obtained results were comparable to those parsers regarding the commands they can execute as well as the ground-truth, emphasizing the accuracy of the proposed approach. Its top-down and recursive architecture allows easy integration with other software regardless of the platform, highlighting its potential inclusion and integration in the production of electronic circuits.

INDEX TERMS Parser, Gerber format, recursive descent, printed circuit boards.

I. INTRODUCTION

Industrial companies are facing a huge global competition in today's market, which exacerbates the need to constantly innovate to stay competitive and ahead of their competitors. The production of electronic circuits has been typically performed resorting to the development of a Printed Circuit Board (PCB) in which the electronic components are soldered. In the last couple of years, a new approach has been considered and studies are under development for its implementation: the use of functional inks and the direct printing of the electronic circuit on a substrate, which allows to adapt the technology to more complex surfaces, increase the versatility of using electronic circuits in different applications, and make the process more agile [1]–[3]. The objective of the VINCI7D project is the development of a robotic printing solution to be used for printing thin-film devices on 3D polymeric structures [4]. This solution encompasses multi-ink printheads (compatible with decorative and functional inks) integrated into a robotic arm, using the required communication protocols with the printheads, combined with a software solution for mapping the circuits to print onto the substrate. The end goal is using this solution for printing electroluminescent and piezoelectric devices, as well as capacitive sensors, directly onto 3D polymeric structures (envisioned as parts for the interior of automobiles), to create structures that are capable of emitting light, detecting the presence or touch of the user, and providing haptic feedback [4]. The printing of these circuits is commonly performed using ink-jet printers and the circuit schematics can be passed to the printer in different

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.

formats. In the case of the VINCI7D project, the circuits to be printed are supplied as Gerber files and transformed into bitmap images for generating the pattern to be printed by the inkjet printheads. This approach created the need to use a parser for converting the Gerber files into bitmap images.

According to Eurocircuits [5], 95 of 100 PCB designs transferred, worldwide, from the designer to the manufacturer, correspond to Gerber files. Up to now, the Gerber format holds four generations from X0 to X3. Gerber X0, also known as Gerber RS274D (specification published in 1980), was revoked most due to the limitation of the physical aperture wheel that limits the size, shape, and number of apertures (e.g., a wide variety of mainly rectangular pad sizes). To overcome this restriction, in 1998, the Gerber X1 (Gerber RS274X, Extended Gerber or Gerber X) was released providing information of the fabrication image for each layer, as well as embedded apertures and their custom definition. However, there were no standards to define the depiction of each file/layer or the model of each pad, being the information in an additional document or drawing. Therefore, Gerber X2 emerged in 2014, keeping the data from Gerber X1 but adding new imaging functions such as aperture block, load rotation and load scaling, and bare board attributes which contributed to define the file and the function of an object (e.g. a pad). The added attributes offered new information for fabrication, without adding image data. To be able to assemble the data, the Gerber X3 specification emerged in 2020 reusing the format and attribute syntax of Gerber X2 but adding new content – new attributes relative to the components [5]–[7].

The goal of this work is to develop a parser compliant with, at least, the Gerber X2 specification using a recursive descent approach [8]. This parser will be used for rendering 2D bitmaps described in Gerber files equivalent to electronic circuits within the scope of the VINCI7D project. The parser must be compatible with the Windows Operating System (OS) due to later integration of the parser with other modules to be developed in the project. Even so, the parsing architecture is this work is independent of the OS or programming language used for its implementation. Given the focus on the rendered image from a Gerber file, the parser is not required to be fully compliant with Gerber X3 due to the latter not adding imaging data to the Gerber file.

A. RELATED WORK

1) GERBER VIEWERS

Currently, there are several Gerber viewers in the market whose tools allow to visualize, inspect, measure and/or edit the contents of the file. PCBWay and JLCPCB Gerber Viewers are online platforms that support the Gerber X1 format [9], [10]. These online platforms are unable to set the mm/px (millimeter per pixel) resolution of the viewport. PCBWay does not allow to export the project into a bitmap format. JLCPCB requires registration and does not have a mechanism to define the desired export format. GerbMagic also supports only the Gerber X1 format [11]. This application allows to define the DPI (dots per inch), export files such as bitmaps, and it can be used as a standalone library (GerbLib). While FAB 3000 [12] and ZofzPCB [13] support Gerber X1, GerbView from Software Companions supports the Gerber X2 specification [14]. These three softwares follow the freemium business model. FAB 3000 allows bitmap format export including the definition of DPI or DPM (dots per millimeter). ZofzPCB does not allow to export or to define the resolution of the PNG format. GerbView exports the workspace to formats such as TIFF, JPEG, PNG and BMP, and allows to define the DPI in the file export. CUPRUM supports Gerber X2 format and is only available for Mac OS X [15]. Finally, the Reference Gerber Viewer, an online platform created by the company Ucamco that produced the Gerber specification, supports the Gerber X3 format [16]. Even though the platform allows to visualize the project and measure the distance between elements (in millimeters), the functionality to export to the image or bitmap format is not enabled and it is not possible to specify the resolution.

2) GERBER LIBRARIES

Furthermore, there are libraries which hold a collection of tools to handle the parsing of PCB Gerber files. GerbLib is a dynamic-link library (DLL) which converts a RS274X Gerber file format to PDF, Postscript, TIFF, BMP or RID files, being possible to set the resolution value, and relies on GerbMagic application [17]. GBR_RIP converts a Gerber file RS274X format into a raster bitmap (BMP or TIFF). The library is not open source, and to download it, some information is required from the Artwork company such as program license strings or installation password. Moreover, to modify the program, a charge is considered relying on the number of licenses, and the rights retain at Artwork [18]. Libpger is a dynamic-link library referenced for the Mac OS X platform which allows to parse a Gerber RS274X file [19]. GerberParser from Jules Stringer is an open-source C++ library for Gerber and drill files parsing. The parsing of some commands is not implemented yet, such as aperture block, step and repeat, and macro apertures [20]. The development of the Gerber and drill file parser in the repository hosted at [21] is deprecated, having been moved to the tracespace one written in the JS language [22]. A viewer based on the tracespace repository can be found at [23], which exports the workspace to SVG format. Gerbview from KiCad supports Gerber RS274X format, even though it does not provide a standalone library for parsing Gerber files [24]. KiCad provides a 3D bitmap visualization, being able to change its size in mm, inches and DPI, which exports the file to PNG and JPEG format. However, it does not allow to export a Gerber file to a bitmap format [25].

3) SCIENTIFIC LITERATURE

Looking at the literature, only a couple of related works were found. Qi *et al.* present a method for parsing the information of Gerber files, reading it as an ASCII code format file, and

converting it to output images [26]. In their work, the authors also introduce a solution to solve the problem of data loss after the conversion from Gerber file data coordinates to pixel coordinates, but the extension of the work is limited since they only address the analysis and drawing of five graphic output primitives (namely, ellipse, arc, sector, regular polygon and round rectangle). More recently, Fan *et al.* also developed a parser for converting Gerber files into bitmap images, with the purpose of obtaining a standard image to be used in Automatic Optical Inspection machines [27]. However, this work does not consider the more complex structures of the Gerber format such as aperture block, step and repeat statements, and macro apertures.

The current solutions available to the end-user for parsing Gerber files have limitations in supporting at least the Gerber X2 specification format. PCBWay, JLCPCB, GerbMagic/GerbLib, FAB 3000, ZofzPCB, GBR_RIP, Libpger, GerberParse, and Gerbview from KiCad only support the Gerber X1 specification. Even though the GerbView from Software Companions states that supports Gerber X2 files, the results presented in this work will demonstrate that it is incapable of rendering aperture blocks, a specific command to the Gerber X2 format. The ones that support all the commands of Gerber X2, the CUPRUM and Reference Gerber Viewer solutions, focus on the viewing capabilities and not on the export settings. For example, these viewers do not support exporting the renderization result for an image with a desired millimeter per pixel resolution. Moreover, to the best of the authors' knowledge, the current literature does not have any work on a parsing architecture for Gerber files compatible with all commands of the Gerber format that generate image data.

B. CONTRIBUTIONS

Therefore, this work intends to address the gaps previously identified for parsing and rendering Gerber files. When comparing the proposed parsing architecture with other works available in the literature, it is concluded that its scope is larger. This work addresses all Gerber structures, including Gerber macros, a more complex structure of the Gerber format and which has not been the focus of other authors, while also allowing to define the image resolution. Although the parser developed within the scope of VINCI7D must be compatible with the Windows OS, the parsing architecture and methodology in terms of interpreting Gerber files and rendering the equivalent 2D bitmap images are agnostic of the OS and programming language. Unlike other existent applications and libraries, this work is independent of the implementation technologies facilitating integration with the remaining modules of the VINCI7D system or even other systems. Bearing these ideas in mind, this paper presents the development of a parser able to convert Gerber files [28] into bitmaps according to a recursive descent approach [8], using the C++ language for implementing the parser and the OpenCV [29] library for renderization. The bitmap files will then be the basis for the printing of the electronic circuits.

The main contributions of the presented work are the following ones:

- recursive descent parsing approach to render 2D images from Gerber files that, to the best of the authors' knowledge, is not used in any other previous works in the literature or in other Gerber viewers and libraries;
- architecture of a Gerber parser to streamline future integration in other applications, allowing the replication of the proposed approach;
- interpretation and renderization methodology compatible with all commands of the Gerber X2 specification format that output image data;
- theoretical and practical rationale providing specific experimental results of the comparison between the proposed approach and its relation with other Gerber viewers and libraries.

The remainder of this paper is organized as follows. Section II introduces the Gerber format specification with practical illustrations of the commands portrayed in the paper. Section III presents the parsing architecture for processing files and the methodology to process arithmetic expressions required for defining macro variables. Section IV details the renderization approach implemented for each Gerber command. Section V presents the experimental results obtained in this work. Finally, Section VI presents the conclusions and discusses features that may be interesting for future work.

II. GERBER FORMAT SPECIFICATION

The Gerber format is an open standard in the electronics fabrication industry of PCB. A Gerber file comprises a sequence of command instructions to define graphic objects in a vector format that can be rendered as 2D binary images. The detailed information of the Gerber commands is documented in the revision 2021.04 of the format specification [28]. In the following topics, the Gerber commands applied in this work are highlighted with a brief explanation of each one of them, following the syntax applied in the referred revision.

A. COORDINATE COMMANDS (FS AND MO)

Gerber files are modeled by 2D coordinate frames. In this work, the convention followed makes the X and Y axes point to the right and upwards, respectively. For interpreting correctly the parameters and coordinates defined in a file, the Gerber format defines the following two commands:

- Unit (MO): unit of the coordinates and parameters data (either millimeters or inches);
- Format Specification (FS): specifies the format of X and Y coordinate data (defined as integer numbers in a Gerber file).

Table 1 shows an example in which the file unit chosen is mm and the coordinate unit is 10^{-6} mm.

B. APERTURE TRANSFORMATIONS (LP, LM, LR, LS)

In terms of 2D shapes, the Gerber format defines a graphic object called aperture that can have either simple or complex

TABLE 1. Coordinate commands syntax. Cmd stands for Command.

Cmd	Syntax	Example	Description
MO	'%'('MO'('MM'|'IN'))'*%' MM – metric (millimeter) IN – imperial (inch)	%MOMM*%	Sets the file unit to mm
FS	'%'('FS''LA''X'digits 'Y'digits)'*%'	%FSLAX36Y36*%	3: maximum number of integer digits; 6: number of decimal digits

TABLE 2. Aperture Transformations syntax. Cmd stands for Command.

Cmd	Syntax	Example	Description
LP	'%'('LP'('C'|'D'))'*%' C - Clear polarity D - Dark polarity	%LPD*%	Sets the polarity to dark
LM	'%'('LM'('N'|'XY'|'Y'|'X')) '*%' N - No mirroring X,Y,XY - Mirroring along the X, Y or XY axes	%LMN*%	Sets object mirroring to no mirroring
LR	'%'('LR'decimal)'*%'	%LR5.0*%	Sets object rotation to 5 degrees counterclockwise
LS	'%'('LS'decimal)'*%'	%LS0.7*%	Sets object scaling to 70%

TABLE 3. Standard Aperture Template syntax. Cmd stands for command.

Cmd	Syntax	Example	Description
C	'C',diameter 'X'hole_diameter	%ADD11C,0.7X0.45*%	Creates an aperture number 11: a circle with diameter 0.7 and hole diameter 0.45
R	'R','x_size'X'y_size'X'hole_diameter	%ADD17R,0.033X0.019X 0.017*%	Creates an aperture number 17: a rectangle with sides of 0.033 and 0.019 and a 0.017 diameter round hole
O	'O','x_size'X'y_size'X'hole_diameter	%ADD32O,0.052X0.028X 0.014*%	Creates an aperture number 32: an obround with sizes of 0.052 and 0.028 and a 0.014 diameter round hole
P	'P','outer_diameter'X'vertices'X' rotation'X'hole_diameter	%ADD21P,.080X6X0.0X 0.012*%	Creates an aperture number 21: a polygon with 0.080 outer diameter, 6 vertices and a 0.012 diameter round hole

TABLE 4. Opening and closing AB commands syntax. Cmds stands for Commands.

Cmd	Syntax	Example	Description
AB_{open}	'%'('AB'aperture_ident)'*%'	%ABD102*%	Opens the definition of aperture D102
AB_{close}	'%'('AB')'*%'	%AB*%	Closes the AB statement

shapes. The commands load polarity (LP), load mirroring (LM), load rotation (LR) and load scale (LS) can transform apertures by updating the graphics state – set of parameters that affect the renderization of graphic objects – of the Gerber file. LP sets the polarity of graphic objects, which can be clear or dark. LM, LR and LS affect the shape of the aperture by mirroring, rotating or scaling the objects, respectively. The syntax of each referred command can be seen in Table 2.

C. STANDARD APERTURE (C, R, O, P)

The simplest shapes described in the Gerber specification are the standard apertures, which can have the following templates and their respective parameters:

- circle (C): its diameter and the hole diameter;
- rectangle (R): X and Y directions' sizes and the hole diameter;
- obround (O): the same parameters as R, but it has rounded corners on the sides with the smaller direction's size;
- polygon (P): diameter of its circumscribing circle, number of vertices, rotation angle, and the hole diameter.

All standard apertures can have a round hole defined by its diameter. However, the hole diameter is optional when defining the template's parameters. Table 3 shows the syntax of the standard templates as well as examples of the respective apertures.

D. APERTURE DEFINITION (AD)

Each aperture is created by the AD command defining its identification number and template, and the parameters that parameterize the template itself. The syntax of an AD command can be illustrated as follows:

$$AD = \text{`}\%\text{'}(\text{`}AD\text{'}aperture\_ident\ template\_call)\text{`}*\%\text{'},$$

being the *aperture_ident* explained in Section II-E and the *template_call* comprised by the aperture template (standard or macro aperture) and the respective parameters.

E. SET CURRENT APERTURE (Dnn)

The D*nn* Gerber command determines that the current aperture saved in the graphics state should be updated to the aperture with the identification number *nn* (integer greater than 9, due to the 00-09 numbers being reserved in the Gerber specification).

F. APERTURE BLOCK (AB)

Another type of apertures defined in the Gerber format specification is the aperture block declared by an AB statement. These apertures do not represent an aperture's template. An AB statement consists of an ordered set of graphics objects defined by their respective Gerber commands. Any aperture transformation can affect block apertures when flashing them (replication of the aperture at specific coordinates). Indeed, each graphic object of a block aperture has its own polarity. If the polarity of AB is dark when flashed, the graphic objects of AB maintain their original polarity. If not, the objects' polarities are toggled. The AB statement is comprised by the opening and closing operations, as illustrated in Table 4, which delimit the Gerber commands that compose the statement.

G. APERTURE MACRO (AM)

Furthermore, the Gerber format defines also macro apertures. First, their templates are created by the AM command. These templates can have any shape (composing primitive shapes) or parametrization (using macro variables to define sizes or other parameters), and are defined by their name. The AM command syntax is described by the following expression,

where N is the unique name of the macro, C is the primitive shape code and P its parameters, I is an integer positive, and E is an arithmetic expression for the macro variable '$'$I$':

$$AM = \text{'}\%AM\text{'}N\text{'}*\text{'} \mid \begin{Bmatrix} C\text{'}, \text{'}\{P\text{'}, \text{'}P\}* \\ \text{'}\$\text{'}I\text{'} = \text{'}E \end{Bmatrix} + \text{'}*\%\text{'}.$$

The basic primitive shapes defined in the Gerber specification, using a primitive code (the C positive integer), and their required parameters are the following ones:

- circle (1): exposure (0-off/1-on), diameter, center point coordinates, and rotation angle;
- vector line (20): exposure, width, coordinates of the start and end points, and rotation angle (same as a rectangle);
- center line (21): exposure, width, height, center point coordinates, and rotation angle (similar to a vector line);
- outline (4): exposure, number of vertices, vertices coordinates, and rotation angle (same as an irregular polygon);
- polygon (5): exposure, number of vertices, center point coordinates, diameter, and rotation angle;
- moiré (6): exposure, center point coordinates, diameter, thickness, gap, maximum number of rings, crosshair thickness and length, and rotation angle (revision 2020.09 [30]);
- thermal (7): exposure, center point coordinates, outer and inner diameter, gap, and rotation angle.

The default rotation angle of the macro primitives is 0° being this parameter an optional one. A macro template can be a composition of one or more primitive shapes. After a template is defined by an AM command, the AD can instantiate it into a macro aperture.

H. OPERATIONS (D01, D02, D03)

A Gerber file can also contain the so-called operations consisting of coordinate data followed by an operation code. The Gerber format defines three operations, whose syntax is represented in Table 5:

- interpolate (D01): create linear or circular segment between the current point (saved in the graphics state) and the one defined in D01, and updates the current point to the latter;
- move (D02): update current point saved in the graphics state to the one defined in the D02's coordinate data;
- flash (D03): flash of the current aperture (identifiable by its identification number D*nn* and also saved in the graphics state) at the specified coordinates.

I. INTERPOLATION STATE COMMANDS (G01,G02,G03,G74,G75)

For the D01 interpolate command, at least the segment's type (linear or circular) must be known when processing a Gerber file. So, the Gerber format defines the G01 command to enable the linear interpolation mode in the graphics state.

TABLE 5. Operations commands syntax. x_c and y_c stand for X and Y coordinate, and x_o and y_o for the offset in X and Y axis, respectively.

Cmd	Syntax	Example	Description
$D01_{lin.}$	(['X'x_c]['Y 'y_c] 'D01') '*'	X100D01*	Creates a straight-line segment by interpolating from the current point to the new one (100, 0)
$D01_{circ.}$	(['X 'x_c]['Y'y_c]'D01"I'x_o 'J'y_o)'*'	X1Y3I2J0D01*	Creates a circular segment by interpolating from the current point to the new one (1,3) with center offset (2,0)
D02	(['X'x_c]['Y 'y_c] 'D02') '*'	X150Y2875D02*	Moves the current point to the new one (150,2875)
D03	(['X'x_c]['Y 'y_c] 'D03') '*'	X0Y0D03*	Creates a flash object by replicating the current aperture at the new coordinates (0,0)

TABLE 6. SR commands syntax. Cmds, p_int and dec stand for Commands, positive integer and decimal, respectively.

Cmd	Syntax	Example	Description
SR_{open}	'%'('SR"X'p_int 'Y'p_int 'I'dec 'J'dec) '*%'	%SRX4Y115.0J0.0%	Opens an SR statement by replicating a set of graphical objects, 4 times along the X axis with a step distance of 5.0 units
SR_{close}	'%'('SR') '*%'	%SR*%	Closes the SR statement and the block is replicated according to the parameters in the opening SR command

The G02 and G03 commands enable the clockwise (CW) and counterclockwise (CCW) circular interpolation modes, respectively. A circular segment can be restricted to a single quadrant (the absolute angle of the arc being equal to or lower than 90°) or not having any restrictions using the G74 and G75 commands to enable the single and multi quadrant modes, respectively. However, it should be noted that the parser must access the current graphics state of the Gerber file prior to generating the D01's graphics object. Hence, the intended modes for a D01 operation should be defined before D01.

J. REGION STATEMENT (G36/G37)

The region statement command is composed by a stream of commands defining one or more contour segments of a graphics object. The initial point of a contour is set by a D02 and the other points are defined by D01 operations.

K. STEP & REPEAT (SR)

The SR command, defined by a stream of commands, replicates a set of graphics objects without replicating their corresponding commands. The command's parameters are the number of repeats along the X and Y axes, and the step distances between elements along each axis. The SR statement is comprised by the opening and closing operations, as illustrated in Table 6.

III. INTERPRETATION

In terms of processing Gerber files, the format specification already defines a block diagram for dealing with this processing, assuming that there are no attributes (addition

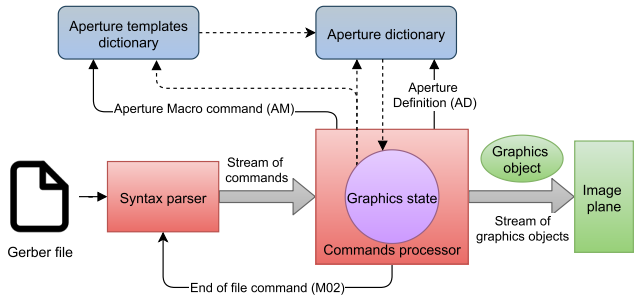


FIGURE 1. Gerber file processing schematic. Adapted from [28].

of meta-information to a Gerber file) defined in the file. An adaptation of this diagram is presented in Figure 1. Given that Gerber attributes do not add image data to the file renderization, they were not considered in this work.

Analyzing Figure 1, the syntax parser would be responsible for reading a Gerber file and producing the resulting list of commands for the commands processor, which transforms it into a stream of graphics objects. All the created objects are overlaid on the image plane following the order of their creation. As already mentioned in Section II, the graphics state contains information of the current aperture and point (when processing the commands' list), in which format and units the coordinate and parameters data are defined in the file, and which modes (aperture transformations and interpolation ones) are enabled. The commands processor is responsible for updating this state by executing the commands that implicitly or explicitly modify the graphics state. Furthermore, the commands processor generates the macro templates and the ones required for standard apertures, which are added to the aperture templates dictionary. This dictionary is a collection of all the available templates to instantiate an aperture. The apertures based on templates are created through the execution of an AD command receiving the values of the template's parameters and added into the apertures dictionary afterward. When the commands processor executes a D*nn* command, the processor updates the current aperture by searching the aperture that has the same identification number as the one specified by D*nn*. The M02 command defines only the end of the file, interrupting the syntax parser and, therefore, stopping the graphical objects generation [28]. In the following subsections, the proposed parser methodology for processing Gerber files into a bitmap image will be portrayed including how to interpret and evaluate arithmetic expressions.

A. PARSER ARCHITECTURE

The methodology for processing Gerber files illustrated in Figure 1 is the basis for the parser implemented in this work. The parser implements the architecture defined by a simplified class diagram illustrated in Figure 2. This architecture follows a recursive descent parsing approach that uses the grammar of the Gerber format to decide the parsing steps. The recursive approach allows to process and rendering Gerber commands depending on their context [8]. An example is rendering a standard aperture inside an aperture block statement versus in the root level of the Gerber file. The former renderization depends not only on the graphics state before initiating the AB statement but also on the one changed inside the statement itself. In contrast, rendering the single standard aperture only depends on the current graphics state. So, it is logical to use a recursive descent approach for parsing Gerber files due to its top-down and recursive characteristics. Even though the parser was implemented in C++ for the Windows OS, the architecture for parsing Gerber files into a bitmap image proposed in this work is agnostic to code language and operating system.

In the diagram of classes shown in Figure 2, it is clear that the central class is the Parser itself. This class is responsible for executing the interpretation and renderization of each Gerber file. Consequently, the Syntax Parser class was defined, which is responsible for retrieving the commands from the file as strings. The commands are retrieved when the method of the Parser for interpretation of a Gerber file is called. This method is responsible for creating all objects that represent Gerber commands, interpreting the strings given by the Syntax Parser, and defining the parameters of each command. For generalization purposes, an abstract class was defined, the Command class, to represent a generic Gerber command. The Parser class must have a list of Commands' objects for calling the render method to generate graphics objects and add them to the image. This approach resembles the interpreter design pattern, where the abstract class works as an interface and the child classes are the ones that implement the renderization and creation of graphics objects, being specific for each type of Gerber command. The commands' parameters are passed by the interpretation method to the constructor of each class specific to a Gerber command.

Furthermore, the Gerber commands that define a stream of graphics objects (AB, SR, and Region Statement) are a composition of Command ones. This composition is processed recursively due to the graphic objects being dependent on the context in which they are defined and also on the order of the commands presented in the Gerber file. In the specific case of region statements, it is considered that a contour is a set of Gerber commands, even though this class will only contain Move and Interpolate objects. This approach was implemented for generalizing the contours processing. However, the Move and Interpolate classes have specific render methods that, instead of generating graphics objects, only retrieve the points that define a contour. For simplification purposes of the diagram, the primitives of macro templates (Macro Define) are not represented. However, it was created a specific class for each one of these. Given that macro templates have a specific format not used by any other Gerber command, it was defined that macro apertures are a composition of macro objects. The latter are the abstract class that represents all the macro primitives defined in the format.

Overall, in this work, the Gerber file processing was separated into two phases: interpretation and renderization. The former phase processes each Gerber command and retrieves

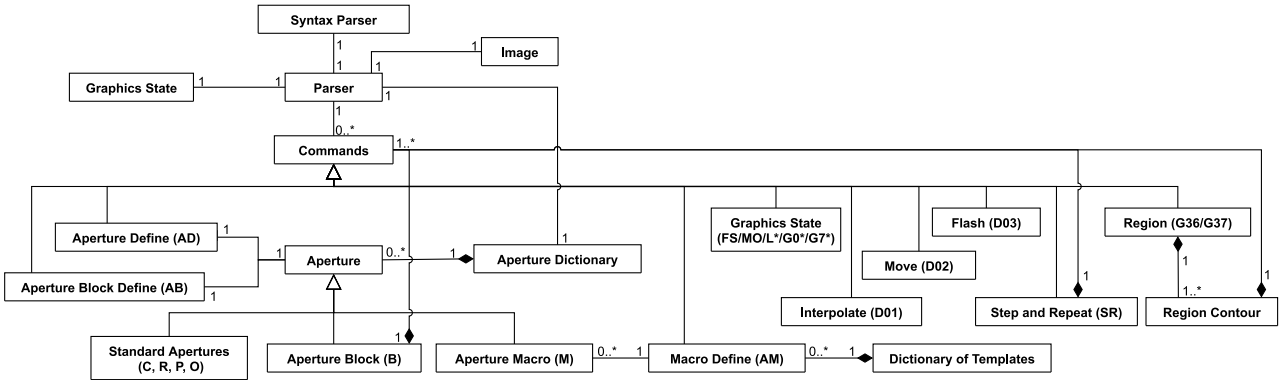


FIGURE 2. Architecture of the implemented parser.

all the information given by it. The latter adds a graphic object resulting from interpreting each command of the Gerber file to the image. Since the interpretation phase is only processing strings and retrieving the information given by the command accordingly to the Gerber format specification [28], this work focus on the renderization phase. Even so, next, it is given details on how to interpret arithmetic expressions required for processing definitions of macro templates. These details are important due to the renderization of macro apertures depending on the correct parameterization of the corresponding templates.

B. ARITHMETIC EXPRESSIONS

In Gerber files, arithmetic expressions are used not only to define macro variables but also to define parameters of macro primitives. These expressions follow a infix-based notation where the operator is between the related operands, being easily readable and understandable for a human. It can comprise brackets and arithmetic operators such as addition, subtraction, multiplication and division, and unary plus and minus ($+_u$ and $-_u$, respectively), which requires the machine to know how to process each command and the respective order. For that purpose, the postfix expression format was adopted in this work. Thus, brackets are not required to process a postfix expression, and the operators follow the operands, being the sequence of operations held [31].

An example of a notation commonly used by humans to express an arithmetic operation is the following:

$$(-A + B)/(C * D - E) - (F - G)$$

whose infix equivalent comes as:

$$(((-A) + B)/((C * D) - E)) - (F - G)$$

and the postfix representation[1] is expressed as:

$$A -_u B + CD * E - /FG - -.$$

[1]An explanation of the differences between infix, prefix and postfix notations can be found in https://runestone.academy/ns/books/published/pythonds/BasicDS/InfixPrefixandPostfixExpressions.html

Given the possibility of having macro variables (defined by its ID and arithmetic expression that could depend on other macro variables) and/or arithmetic expressions for the macro primitives' parameters, the developed parser implements a class for interpreting arithmetic expressions (even if the variables/parameters are only scalars). Due to the infix notation (the one used in a Gerber file) being ambiguous [31], the class converts all arithmetic expressions to a postfix notation. Algorithm 1 defines the interpretation steps implemented in the parser for converting an infix into a postfix notation.

Algorithm 2 formulates the evaluation algorithm of postfix arithmetic expressions. These expressions will be only evaluated upon rendering macro apertures. When instantiating a macro aperture from an existent template, the parameters defined in the AD Gerber command are equivalent to macro variables. These variables can formulate arithmetic expressions to parameterize the primitives of a macro template. So, the arithmetic expressions inside an AM statement that represents a macro template need to be evaluated in every instantiation of the template due to the possibility of having different values for each aperture of the same template.

IV. RENDERIZATION

This section focuses on the Gerber files renderization phase. First, the management of graphic objects is discussed (e.g., automatic resize of the image when rendering these objects) by the class responsible for rendering a bitmap image. Then, all the Gerber commands that add a graphic object to the bitmap image, or define transformations affected to other commands, are analyzed in terms of their renderization implementation. Commands such as updating the graphics state (FS, MO) or the move operation (D02) are not analyzed since they do not add any graphic object to the image plane, even though they were implemented in the parser.

A. MANAGEMENT OF GRAPHIC OBJECTS

The class `GerberImage` (equivalent to the class Image in Figure 2) is responsible for adding and managing the graphic objects rendered from a Gerber file. This class has the following members:

Algorithm 1: ConvertInfix2PostfixExpression

input : *InfixExp*
output: *PostfixExp*

OperatorStack = []
PostfixExp = []
foreach *element* e_k *(operand or operator) of InfixExp* **do**
  if e_k *is an operand* **then**
    PostfixExp.push(e_k)
  else
    if $k = 0$ *or* $e_{k-1} = ($ *or* $e_{k-1} = *$ *or* $e_{k-1} = /$ **then**
      if $e_k = +$ **then** e_k = kUnaryAdd
      else if $e_k = -$ **then** e_k = kUnarySub
    if $e_k = ($ **then** *OperatorStack*.push(e_k)
    else if $e_k =)$ **then**
      while *OperatorStack.size* > *0* **do**
        if *OperatorStack.back* = *(* **then** break
        PostfixExp.push(*OperatorStack*.back)
        OperatorStack.pop()
      OperatorStack.pop()
    else
      while *OperatorStack.size* > *0* **do**
        if *Precedence level of* e_k *greater than OperatorStack.back* **then** break
        PostfixExp.push(*OperatorStack*.back)
        OperatorStack.pop()
      OperatorStack.push(e_k)
while *OperatorStack.size* > *0* **do**
  PostfixExp.push(*OperatorStack*.back)
  OperatorStack.pop()

Algorithm 2: EvaluatePostfixExpression

input : *PostfixExp*, *MacroVariables*
return: computed value of the arithmetic expression

OperandStack = []
foreach *element* e_k *(operand or operator) of PostfixExp* **do**
  if e_k *is an operand* **then**
    if e_k *is a macro variable* **then**
      OperandStack.push(*MacroVariables*.GetValue(e_k))
    else
      OperandStack.push(e_k)
  else
    if e_k *is an kUnarySub* **then**
      nop= $-$*OperandStack*.back
      OperandStack.pop()
      OperandStack.push_back(nop)
    else
      op2= *OperandStack*.back
      OperandStack.pop()
      op1= *OperandStack*.back
      OperandStack.pop()
      if $e_k = +$ **then** nop = op1 + op2
      else if $e_k = -$ **then** nop = op1 − op2
      else if $e_k = *$ **then** nop = op1 · op2
      else if $e_k = /$ **then** nop = op1 / op2
      OperandStack.push_back(nop)
return *OperandStack*.back

- `image`: `cv::Mat` object from the OpenCV [29] library that represents a 2D binary (bitmap) image;
- `origin` (mm): 2D decimal point equivalent to the origin of the image (pixel point with coordinates $(0, 0)$) relative to the Gerber file coordinate frame;
- `res` (mm/px): image resolution set for rendering the stream of graphic objects from a Gerber file.

In terms of generating the graphic objects, the OpenCV [29] library offers drawing methods,[2] such as adding circles, ellipses, or polygons, useful for rendering the objects defined in the Gerber file. Also, note that the bitmap image (represented by the `image` member variable of `GerberImage`) is a `cv::Mat` object from the OpenCV library. Furthermore, the parser must consider two coordinate frames: the one in which all Gerber commands are defined relative to ($\{X^G_{\text{mm/in}}, Y^G_{\text{mm/in}}\}$, in millimeters or inches) and the image one ($\{X^I_{\text{px}}, Y^I_{\text{px}}\}$, in pixels). The parser needs to transform coordinates data from the Gerber to the image frames in order to render the graphic objects correctly. First, a parameter or a point defined in a Gerber file can be in millimeters or in inches (see MO Gerber command explained in Section II-A).

[2]https://docs.opencv.org/4.5.3/d6/d6e/group__imgproc__draw.html

Given the mm/in conversion ratio, all parameters and coordinates defined in inches are multiplied by 25.4 mm/in to only consider the millimeter metric in the `GerberImage` class. Next, `origin` represents the translation transformation between the image and the Gerber coordinate frames but still in millimeters. Finally, the decimal point defined in the image frame is converted into a pixel point. This conversion, represented by the member variable `res`, defines the resolution of the rendered bitmap image set by the user. Equation (1) defines the transformation of a point $P_{\text{mm/in}}$ from the Gerber to the image frame.

$$P^I_{\text{px}} = \begin{cases} (P_{\text{mm}} - \texttt{origin})\,/\texttt{res} & \text{if MO = MM} \\ (25.4 \cdot P_{\text{in}} - \texttt{origin})\,/\texttt{res} & \text{if MO = IN} \end{cases} \quad (1)$$

The raw `cv::Mat` object defines the X and the Y axes as the column (0..width − 1) and row indexes (0..height − 1), respectively. Also, the origin of the `cv::Mat` is on the top-left corner of its image. Even though the parser did not change the way of accessing a `cv::Mat` object when rendering the Gerber file, the image was flipped around the x-axis (without changing the image variable of the `GerberImage` class) using `cv::flip` when saving the image in the file system. Figure 3 illustrates the raw image of a `cv::Mat` object and the flip operation used when saving it. The flip

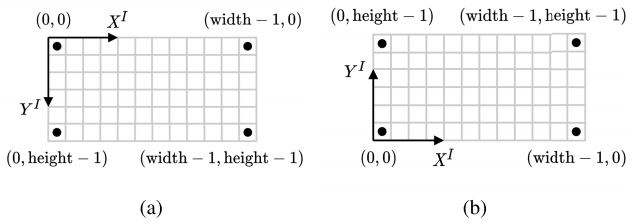


FIGURE 3. Vertical flip operation on the `cv::Mat` object: (a) raw image; (b) flipped image around the x-axis.

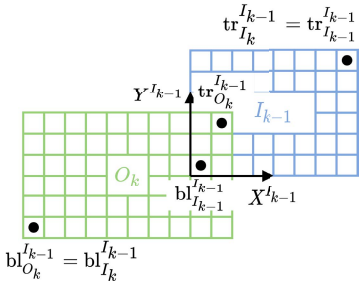


FIGURE 4. Dynamic resize of a image I_{k-1} when adding the graphic object O_k.

operation is only performed after the Gerber file's renderization is finished.

Finally, the class `GerberImage` is also responsible for changing the `image`'s size over rendering each Gerber command. When adding the k graphic object to the image I_{k-1} (where $k = 1, \ldots, N$ assuming that a Gerber file defines N graphic objects), the image's bottom-left and top-right limits (points $\mathrm{bl}_{I_{k-1}}^{I_{k-1}} = (0, 0)$ and $\mathrm{tr}_{I_{k-1}}^{I_{k-1}} = (\mathrm{width}_{k-1}, \mathrm{height}_{k-1} - 1)$ relative to I_{k-1}, respectively) are compared to the ones of the object (points $\mathrm{bl}_{O_k}^{I_{k-1}}$ and $\mathrm{tr}_{O_k}^{I_{k-1}}$). Note that the limits $\mathrm{bl}_{O_k}^{I_{k-1}}$ and $\mathrm{tr}_{O_k}^{I_{k-1}}$ coordinates can have negative values, as illustrated in Figure 4. Next, the expected limits of the image when adding the k object (points $\mathrm{bl}_{I_k}^{I_{k-1}}$ and $\mathrm{tr}_{I_k}^{I_{k-1}}$, respectively) are computed as in (2). After saving the image I_{k-1} in a temporary variable for not losing data, the limits $\mathrm{bl}_{I_k}^{I_{k-1}}$ and $\mathrm{tr}_{I_k}^{I_{k-1}}$ are converted into width and height for increasing the image's size. Also, if the X and/or Y coordinates of $\mathrm{bl}_{I_k}^{I_{k-1}}$ are negative, it means that the same coordinate of `origin` must be subtracted the same negative value in millimeters (to correct the `origin` of the image with new size) as formulated in (3). Finally, the temporary image is added to the image (not to lose any data) considering the new `origin` translation transformation, and then, the graphic object is added to the image. The proposed approach avoids using unnecessary memory (in terms of unused pixels).

$$\begin{aligned} \mathrm{bl}_{I_k}^{I_{k-1}} &= \min\left(\mathrm{bl}_{I_{k-1}}^{I_{k-1}}, \mathrm{bl}_{O_k}^{I_{k-1}}\right) \\ \mathrm{tr}_{I_k}^{I_{k-1}} &= \max\left(\mathrm{tr}_{I_{k-1}}^{I_{k-1}}, \mathrm{tr}_{O_k}^{I_{k-1}}\right) \end{aligned} \tag{2}$$

$$\mathtt{origin}_k = \mathtt{origin}_{k-1} + \min\left(\mathrm{bl}_{I_k}^{I_{k-1}}, [0\ 0]^T\right) \cdot \mathtt{res} \tag{3}$$

B. APERTURE TRANSFORMATIONS (LP, LM, LR, LS)

The `GerberImage` class defines dark polarity by white pixels (intensity = 255) and clear polarity as black ones (intensity = 0). These definitions allow the use of addition and multiplication pixel-wise operations to add a dark and a clear object to an image, respectively. Considering that the area of a clear object O_k is defined by pixels with an intensity of 0 and the other pixels as 255, the multiplication (in OpenCV, `cv::multiply`) of the object O_k with the corresponding area in the image I_{k-1} (using its updated size) will be the following one: maintain the image's pixel values unless the corresponding ones on the object are 0 setting the later ones also as 0. Consequently, the clear graphic object O_k clears the underlying objects present in the image as intended and defined in the Gerber specification. For adding (function `cv::add`) a dark graphic object O_k (its pixels set as 255 while the other ones as 0), the object's area on the image I_{k-1} is set as 255 while the other image's pixels maintain their value. Note that OpenCV [29] saturates the pixel's intensity at 255, just as intended for the addition operation (e.g., if both pixels in the image and object were 255, the arithmetic addition would be 510).

As for the LM command, the mirroring transformation mirrors the aperture around its origin (which may not be its geometric center) along the X (LM_X), Y (LM_Y), or along both axes (LM_{XY}). In the case of an interest point P of an aperture (e.g., a vertex of a polygon), the transformations LM_X, LM_Y, and LM_{XY} are defined as in (4), (5), and (6), respectively.

$$P_{LM_X} = LM_X(P) = [-P_X\ P_Y]^T \tag{4}$$

$$P_{LM_Y} = LM_Y(P) = [P_X\ -P_Y]^T \tag{5}$$

$$P_{LM_{XY}} = LM_{XY}(P) = [-P_X\ -P_Y]^T \tag{6}$$

Moreover, the LR transformation defines the rotation angle of the aperture when flashing it on the image. Similar to the mirroring transformation, the aperture is rotated around its origin. Given an interest point P of an aperture, (7) formulates the LR_α transformation that rotates the point P by a certain angle α (rad). Also, the LR transformation is always performed after the mirroring one when flashing an aperture.

$$P_{LR_\alpha} = LR_\alpha(P) = R(\alpha) \cdot P = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \cdot P \tag{7}$$

Lastly, the LS command defines the scale factor used when flashing an aperture. Similar to both LM and LR transformations, the aperture is scaled centered on its origin. Equation (8) defines the scaling transformation for an aperture's interest point P by a scale factor of s (decimal greater than 0).

$$P_{LS_s} = LS_s(P) = P \cdot s \tag{8}$$

C. STANDARD APERTURES (C, O, P, R)

Figure 5 illustrates the renderization of the four shapes that represent the standard apertures: C, O, P and R, respectively. When a standard aperture is flashed, the respective shape

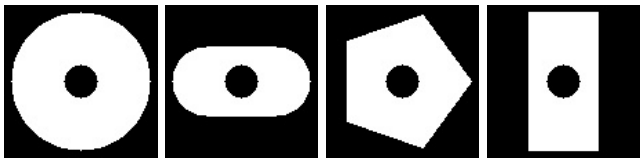


FIGURE 5. Renderization examples of standard apertures with a hole: (a) C; (b) O; (c) P; (d) R.

creates a temporary image containing only the aperture. If the current polarity is dark, the aperture's pixel area is set as 255 and the other pixels as 0, and vice versa if the current polarity is clear. This approach was implemented for rendering every single graphic object of a Gerber file allowing the use of the addition and multiplication pixel-wise operations for adding the graphic objects to the image (see Section IV-A). The graphic object is only added after all the renderization operations required for rendering the aperture are finished (e.g., the possibility of a standard aperture to have a transparent hole).

In terms of how to render the shapes, the circle and polygon standard apertures were rendered using the functions `cv::circle` and `cv::fillPoly` from OpenCV, respectively. In the case of the regular polygon shape, its vertices ($V^A_{P,i}$, in pixels, relative to the aperture's local coordinate frame $\{X^A, Y^A\}$ centered on the aperture's center) depend on the specified outer diameter ($d_{P,outer}$, in pixels) and the number of vertices ($\#V_P$, where $i = 0, \ldots, \#V_P - 1$). These vertices are defined in (9). The rectangle was considered in the implementation as an irregular polygon of 4 vertices (top-left $V^A_{R,\text{tl}}$, top-right $V^A_{R,\text{tr}}$, bottom-left $V^A_{R,\text{bl}}$, and bottom-right $V^A_{R,\text{br}}$ vertices, in pixels). Equation (10) defines these 4 vertices depending on the rectangle's width (w_R, in pixels) and height (h_R, in pixels). As for the obround aperture, it can be viewed as a composition of three different primitive shapes (but consider as a single graphic object): two circles and one rectangle. The diameter (d_O, in pixels) of the circle equals the minimum of the specified width (w_O, in pixels) and height (h_O, in pixels) for the obround. The rectangle's vertices ($V^A_{O,\text{tl}}$, $V^A_{O,\text{tr}}$, $V^A_{O,\text{bl}}$, and $V^A_{O,\text{br}}$) change depending on if the obround's width (or height) is equal to the circles' radius or not. Considering that, (11) defines the vertices of the obround's rectangle.

$$V^A_{P,i} = \left[\frac{d_{P,outer}}{2}\cdot\cos\left(\frac{2\pi i}{\#V_P}\right) \quad \frac{d_{P,outer}}{2}\cdot\sin\left(\frac{2\pi i}{\#V_P}\right)\right]^T \tag{9}$$

$$\begin{bmatrix} {V^A_{R,\text{tl}}}^T \\ {V^A_{R,\text{tr}}}^T \\ {V^A_{R,\text{bl}}}^T \\ {V^A_{R,\text{br}}}^T \end{bmatrix} = \frac{1}{2}\cdot\begin{bmatrix} [-w_R \quad h_R] \\ [w_R \quad h_R] \\ [-w_R \quad -h_R] \\ [-w_R \quad h_R] \end{bmatrix} \tag{10}$$

$$\begin{bmatrix} {V^A_{O,\text{tl}}}^T \\ {V^A_{O,\text{tr}}}^T \\ {V^A_{O,\text{bl}}}^T \\ {V^A_{O,\text{br}}}^T \end{bmatrix} = \frac{1}{2}\cdot\begin{bmatrix} [-w_O \quad h_O - d_O] \\ [w_O \quad h_O - d_O] \\ [-w_O \quad -h_O + d_O] \\ [w_O \quad -h_O + d_O] \end{bmatrix}, \quad \text{if } d_O = w_O$$

$$\begin{bmatrix} {V^A_{O,\text{tl}}}^T \\ {V^A_{O,\text{tr}}}^T \\ {V^A_{O,\text{bl}}}^T \\ {V^A_{O,\text{br}}}^T \end{bmatrix} = \frac{1}{2}\cdot\begin{bmatrix} [-w_O + d_O \quad h_O] \\ [w_O - d_O \quad h_O] \\ [-w_O + d_O \quad -h_O] \\ [w_O - d_O \quad -h_O] \end{bmatrix}, \quad \text{if } d_O = h_O \tag{11}$$

When rendering a standard aperture outside a block aperture, the mirroring transformation (LM) can be performed on the polygon standard aperture. This transformation only has a difference in the renderization output when mirroring along the Y-axis on polygons with odd numbers of vertices. Mirroring circles, obrounds, or rectangles does not produce any difference because these shapes are symmetric along both X and Y axes. In contrast, LM can have renderization differences on the standard apertures if these are defined inside a block aperture. For instance, the standard aperture inside the block has non zero rotation and the respective block aperture has a 0° rotation. If the block aperture is mirrored, the condition of the primitive apertures being symmetric along the X and/or Y axes is no longer valid.

As for rotating the apertures, the parser rotates their points of interest (e.g., the vertices of a rectangle and/or the center points of the circle that can compose an obround aperture) to avoid the deformation of the rendered image (see (7)). This method of rotation is illustrated in Figure 6a comparing it to rotating the temporary image using the `cv::warpAffine` OpenCV function (with the nearest interpolation for the image to remain binary) illustrated in Figure 6b in which the aperture is first rendered. The former leads to no deformation, while the latter has convexity defects when the the aperture was rotated in 30° or 60° (red circles on the bottom-right and top-left obrounds in Figure 6b, respectively). If the primitive is inside a block aperture, the desired rotation for the primitive must be the addition of the rotation defined in the local graphics state of the block aperture with the rotation set on the current graphics state.

D. APERTURE BLOCKS (AB)

The renderization of a block aperture is accomplished by calling the render method of each Gerber command defined in the block's command stream. An example is presented in Figure 7 illustrating the renderization of a block aperture affected by the LM, LR and LS transformations. Similar to the standard apertures, if a mirror or a rotation transformation affects an AB command, all points of interest of the commands are mirrored or rotated, respectively. Figure 7 shows that the implemented method does not deform the block aperture (when affected by aperture transformations).

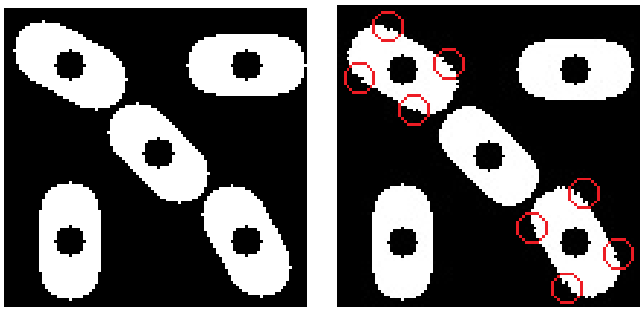


FIGURE 6. Example of rotating obround apertures: (a) rotation of the points of interest; (b) `cv::warpAffine` for rotating each aperture's temporary image (deformation appeared on the top-left and bottom-right apertures).

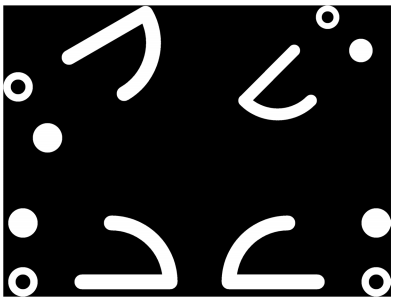


FIGURE 7. Renderization example of a block aperture (top-left: LM_Y and $LR_{30°}$; top-right: LM_{XY}, $LR_{45°}$, and $LS_{0.8}$; bottom-left: original block aperture; bottom-right: LM_X).

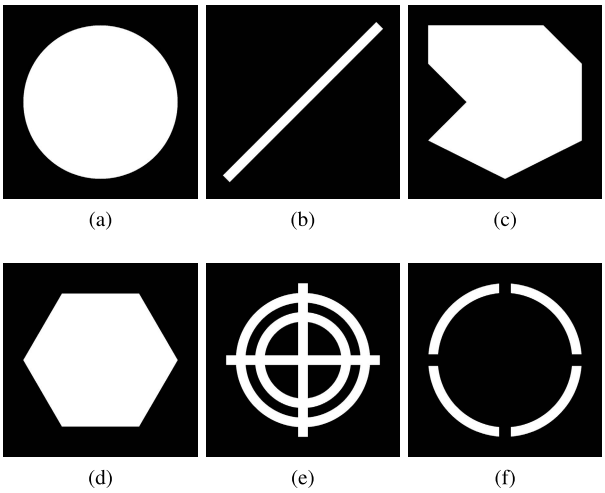


FIGURE 8. Renderization of macro primitives: (a) circle; (b) vector line or center line (different parameterization, same renderization output); (c) outline (irregular polygon); (d) polygon; (e) moiré; (f) thermal.

E. MACRO APERTURES (AM)

In terms of renderization, the definition of macro templates using the AM Gerber command does not generate any graphic object. However, this command defines a template that parametrizes a macro aperture and also implements a render method given the value of the macro aperture's parameters. Indeed, when flashing a macro aperture, the parser first accesses the dictionary of macro templates to find the respective template. This dictionary was formulated upon the interpretation phase of a Gerber file. Next, it is created a temporary image for rendering the macro aperture. The renderization is executed by the render method of the macro's template taking as input the parameters defined by the AD command. As for macro primitives, the render method adds the respective graphic object to the temporary image. After rendering all the commands defined in the macro template, the temporary image is added to the image member variable of the `GerberImage` class.

Examples for rendered macro primitives are illustrated in Figure 8. The implementation in the parser for rendering macro apertures defined these primitives by one or more primitive shapes, as follows:

- circle (1): a single circle (`cv::circle`);
- vector line (20): a line defined by its width and start and end points, rendered exactly as a rectangle standard aperture (a single rectangle after converting it to a line defined by its center, width, height and rotation);
- center line (21): rendered exactly as a rectangle standard aperture (the line is already defined by its center, width, height and rotation);
- outline (4): a single irregular polygon (`cv::fillPoly`);
- polygon (5): rendered exactly as a polygon standard aperture;
- moiré (6): one or more circles with holes and two rectangles (`cv::fillPoly`);
- thermal (7): one circle with an hole and two rectangles with opposite polarity to create the 4 gaps.

Note that the vector and the center lines define a line resorting to different parameters. A vector line requires its width (w_{vl}), and the start (S_{vl}) and end (E_{vl}) points, whereas the center line needs its height and width, and the center point. In this work, both primitives were considered equivalent to a solid rectangle standard aperture. Even though renderization of a center line is straightforward (has the same parameters as R apertures), (12) is needed for converting a vector line parameterization into a center line one, where $w_{\text{vl}\to\text{cl}}$, $h_{\text{vl}\to\text{cl}}$, and $\alpha_{\text{vl}\to\text{cl}}$ are the width, height, and rotation angle of the vector line's equivalent rectangle, respectively. The vector line's center point is the average of its start and end points.

$$\begin{aligned} w_{\text{vl}\to\text{cl}} &= \text{dist}\left(S_{\text{vl}}, E_{\text{vl}}\right) \\ h_{\text{vl}\to\text{cl}} &= w_{\text{vl}} \\ \alpha_{\text{vl}\to\text{cl}} &= \text{atan2}\left(E_{\text{vl},Y} - S_{\text{vl},Y}, E_{\text{vl},X} - S_{\text{vl},X}\right) \end{aligned} \tag{12}$$

F. OPERATIONS (D01, D03)

1) INTERPOLATION (D01)

The linear interpolation (D01 with G01 enabled) operation has the same parameters as a vector line (20):

- start (S_{D01}) and end (E_{D01}) points of the linear segment;

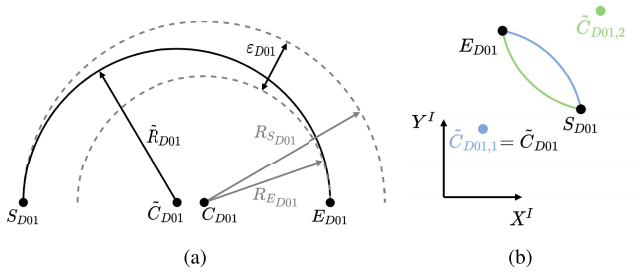


FIGURE 9. Circular interpolation operation: (a) deviation (ε_{D01}); (b) solutions to the corrected arc's center ($\tilde{C}_{D01}$).

- width equivalent to the diameter of the current circle standard aperture (only type of aperture allowed in the specification [28] to stroke a linear draw, i.e., the line has rounded endpoints defined by the circle).

Consequently, the same approach for rendering vector lines can be used for the D01 linear interpolation converting the D01 parameters to linear segment's center, width, height, and rotation (see (12)). Next, the same rendering procedure as for rectangle standard apertures and the macro primitive center lines is used to render the linear straight segment defined by D01 with G01 enabled.

In the case of the circular interpolation (D01 with G02/G03 enabled), this interpolation can have two orientations: clockwise (G02) or counterclockwise (G03). Also, the D01 operation defines not only the arc's start (S_{D01}) and end (E_{D01}) points but also two variables: i_{D01} and j_{D01} (parallel to the X and Y axes, respectively). These two variables define the arc's center (C_{D01}) and are interpreted as distances or offsets relative to S_{D01} depending on the quadrant mode enabled, as follows:

- single quadrant mode enabled (G74):
  - arc angle β_{D01} not allowed to extend more than 90° (i.e., $0° < \beta_{D01} \leq 90°$);
  - i_{D01} and j_{D01} interpreted as distances defining 4 possible candidates to the arc's center C_{D01}: $S_{D01} + [\pm i_{D01} \ \pm j_{D01}]^T$;
- multi quadrant mode enabled (G75):
  - arc can have an angle of $0° < \beta_{D01} \leq 360°$;
  - if $S_{D01} = E_{D01}$, the arc is a full circle of 360°;
  - i_{D01} and j_{D01} interpreted as offsets defining only 1 possible candidate to the arc's center C_{D01}: $S_{D01} + [i_{D01}\ j_{D01}]^T$.

Ideally, the distance from C_{D01} to S_{D01} is exactly equal to the distance to E_{D01}. However, it could not be possible to place C_{D01} exactly on the desired point, and also the Gerber file has a finite resolution. Figure 9a illustrates an example in which the C_{D01}'s definition leads to different start and end radius ($R_{S_{D01}}$ and $R_{E_{D01}}$, respectively). This difference is called the arc deviation (ε_{D01}) shown in Figure 9a and formulated in (13).

$$\varepsilon_{D01} = |\text{dist}\,(C_{D01}, E_{D01}) - \text{dist}\,(C_{D01}, S_{D01})| \tag{13}$$

Given that the Gerber specification accepts the existence of deviation (the specification does not specify a way to deal with this problem), the arc radius was defined as the average ($\tilde{R}_{D01}$) of $R_{S_{D01}}$ and $R_{E_{D01}}$ (see (14)). Then, the arc is restricted in terms of passing through S_{D01} and E_{D01}, as shown in Figure 9a and formulated in (15). The system of equations computes the unknown arc's center ($\tilde{C}_{D01}$). Note that, even though $\tilde{C}_{D01}$ can be shifted relative to C_{D01} (see Figure 9a), the proposed approach guarantees the arc's endpoints defined by the D01 command and reduces the deviation ideally to 0 (not considering rounding errors or the Gerber file's finite resolution).

$$\tilde{R}_{D01} = \left(R_{S_{D01}} + R_{E_{D01}}\right)/2 \tag{14}$$

$$\begin{cases} \left|S_{D01} - \tilde{C}_{D01}\right|^2 = \tilde{R}_{D01} \\ \left|E_{D01} - \tilde{C}_{D01}\right|^2 = \tilde{R}_{D01} \end{cases} \tag{15}$$

Assuming that D01 has already only 1 possible candidate to the arc's center (e.g., when multi quadrant mode is enabled), Figure 9b illustrates that (15) has 2 possible solutions for $\tilde{C}_{D01}$. These two solutions are formulated for different cases in (16) (where a_1, a_2, and a_3 are defined in (17), (18), and (19)) and (20) (for b_2 and b_3 formulated as in (21) and (22), and considering $b_1 = 1$) if $S_{D01,Y} \neq E_{D01,Y}$ or $S_{D01,Y} = E_{D01,Y}$, respectively. Parameters a_1, a_2, and a_3, and b_1, b_2, and b_3 are relative to two 2nd-order equations formulated from solving (15). The final value of $\tilde{C}_{D01}$ is formulated as the closest center to the original one (C_{D01}), as formulated in (23).

$$\tilde{C}_{D01,i} = \begin{bmatrix} \left(-a_2 \pm \sqrt{a_2^2 - 4\,a_1 a_3}\right)/(2\,a_1) \\ \frac{|S_{D01}|^2 - |E_{D01}|^2 + 2\tilde{C}_{D01,i,X}(E-S)_{D01,X}}{2\cdot(S_{D01,Y} - E_{D01,Y})} \end{bmatrix}$$
$$\text{if } S_{D01,Y} \neq E_{D01,Y}, \text{ with } i = 1(+), 2(-) \tag{16}$$

$$a_1 = 1 + \frac{\left(E_{D01,X} - S_{D01,X}\right)^2}{\left(S_{D01,Y} - E_{D01,Y}\right)^2} \tag{17}$$

$$a_2 = -2S_{D01,X} - 2\,S_{D01,Y} \cdot \frac{E_{D01,X} - S_{D01,X}}{S_{D01,Y} - E_{D01,Y}} + \frac{(|S_{D01}|^2 - |E_{D01}|^2)(E_{D01,X} - S_{D01,X})}{(S_{D01,Y} - E_{D01,Y})^2} \tag{18}$$

$$a_3 = S_{D01,X}^2 + S_{D01,Y}^2 - S_{D01,Y} \cdot \frac{|S_{D01}|^2 - |E_{D01}|^2}{S_{D01,Y} - E_{D01,Y}} + \frac{(|S_{D01}|^2 - |E_{D01}|^2)^2}{4 \cdot \left(S_{D01,Y} - E_{D01,Y}\right)^2} - \tilde{R}_{D01}^2 \tag{19}$$

$$\tilde{C}_{D01,i} = \begin{bmatrix} \left(S_{D01,X}^2 - E_{D01,X}^2\right)/\left(2 \cdot \left(S_{D01,X} - E_{D01,X}\right)\right) \\ \left(-b_2 \pm \sqrt{b_2^2 - 4\,b_1 b_3}\right)/(2\,b_1) \end{bmatrix}$$
$$\text{if } S_{D01,Y} = E_{D01,Y}, \text{ with } i = 1(+), 2(-) \tag{20}$$

$$b_2 = -2 \cdot S_{D01,Y} \tag{21}$$

$$b_3 = S_{D01,X}^2 + S_{D01,Y}^2$$

$$- S_{D01,X} \cdot \frac{S_{D01,X}^2 - E_{D01,X}^2}{S_{D01,X} - E_{D01,X}} + \frac{(S_{D01,X}^2 - E_{D01,X}^2)^2}{4 \cdot \left(S_{D01,X} - E_{D01,X}\right)^2} - \tilde{R}_{D01}^2 \quad (22)$$

$$\tilde{C}_{D01} = \underset{x \in \{\tilde{C}_{D01,1}, \tilde{C}_{D01,2}\}}{\arg\min} \{\text{dist}\,(x, C_{D01})\} \quad (23)$$

For single quadrant mode, it is required to select a center for the arc from the 4 possible candidates. First, after obtaining the parameters ($\tilde{C}_{D01}$, start angle – $\beta_{D01,S}$, and end angle – $\beta_{D01,E}$) of the arcs for each candidate, it is checked if the arc's angle (as in (24), where $\beta_{D01,CW} = -\beta_{D01,CCW}$) is between 0° and 90° depending if it is enabled the CCW or the CW directions. $\beta_{D01,S}$ and $\beta_{D01,E}$ are formulated in (25) and (27), respectively. From the remaining candidates, it is chosen the one that leads to a minimum deviation (ε_{D01}) relative to the original center (C_{D01}).

$$\beta_{D01,CCW} = \text{WrapToPi}\left(\beta_{D01,E} - \beta_{D01,S}\right) \quad (24)$$

$$\beta_{D01,S} = \text{atan2}\left(S_{D01,Y} - \tilde{C}_{D01,Y}, \quad (25)\right.$$

$$\left. S_{D01,X} - \tilde{C}_{D01,X}\right) \quad (26)$$

$$\beta_{D01,E} = \text{atan2}\left(E_{D01,Y} - \tilde{C}_{D01,Y}, \quad (27)\right.$$

$$\left. E_{D01,X} - \tilde{C}_{D01,X}\right) \quad (28)$$

The renderization of circular interpolations considers both orientation (G02/03) and quadrant (G74/75) modes. After obtaining the parameters of the respective arc, the function `cv::ellipse` from the OpenCV [29] library was used to render the arc. Both axes of the ellipse were set as the diameter of the arc ($2 \cdot \tilde{R}_{D01}$) for rendering the circular segment. Similar to the linear interpolation, the diameter of the current circle standard aperture defines the thickness of the ellipse's boundary. Normally, the start ($\beta_{D01,S,\texttt{cv::ellipse}}$) and end ($\beta_{D01,E,\texttt{cv::ellipse}}$) angles required for the function `cv::ellipse` are equal to the arc angles ($\beta_{D01,S}$ and $\beta_{D01,E}$, respectively). The latter ones are wrapped to the interval $[-\pi, \pi[$ rad. However, there is an exception when the arc passes through the angle's discontinuity ($-\pi$). This exception is due to `cv::ellipse` assuming that $\beta_{D01,E,\texttt{cv::ellipse}} > \beta_{D01,S,\texttt{cv::ellipse}}$. If not, the OpenCV function swaps the start angle with the end one, always rendering the arc in the CCW direction (from the start to the end). Thus, the following check was implemented:

- clockwise orientation enabled (G02):
  - if $(\min\{\beta_{D01,S}, \beta_{D01,E}\} \neq \beta_{D01,E}$ OR $\beta_{D01,S} = \beta_{D01,E})$
    then $\beta_{D01,S,\texttt{cv::ellipse}} = \beta_{D01,S} + 2\pi$;
- counterclockwise orientation enabled (G03):
  - if $(\min\{\beta_{D01,S}, \beta_{D01,E}\} \neq \beta_{D01,S}$ OR $\beta_{D01,S} = \beta_{D01,E})$
    then $\beta_{D01,E,\texttt{cv::ellipse}} = \beta_{D01,E} + 2\pi$.

The example shown for block apertures (Figure 7) has both linear and circular interpolation operations defined inside the aperture block. The parser was able to compute the correct linear and arc parameters, even when these graphic objects were defined inside a command stream such as the block aperture (and also subject to different aperture transformations). Furthermore, it is clear in Figure 7 that the rendered arcs pass exactly through one of the linear segments endpoints, just as defined in the respective Gerber file. So, this result also validates the approach of restricting the arc by (15).

FIGURE 10. Renderization example of region statements.

2) FLASH (D03)

When rendering the flash (D03) operation, first, the parser accesses the graphics state to get the current aperture. Next, the render method of the respective aperture is called to add the respective graphic object(s) to the rendered image (see sections IV-C, IV-D, and IV-E for further details on how to render apertures defined in the Gerber format specification [28]) at the point defined by the D03 operation. Finally, the current point saved in the graphics state is updated setting it to the D03's flashing point.

G. REGION STATEMENTS (G36/G37)

Figure 10 presents an example of rendering two region statements. First, it was added a region with a rectangular shape and dark polarity (white intensity). The contour of this region is defined by 4 linear segments delimited by their endpoints. Then, the other region (with some rounded corners using D01 command with G02/G03 enabled) was added to the image with clear polarity.

The renderization of a region statement calls (G36/37) the render methods of each contour defined in its statement. For generalization purposes, the developed parser considers each contour as an irregular polygon (see Section IV-E for details on rendering the macro primitive outline also equivalent to an irregular polygon). In the case of linear segments, only its end point is added to the list of vertices of the irregular polygon (assuming that the initial point of the contour is already on this list). As for circular segments, these are discretized (angular discretization resolution is set by the user) and converted from an arc into a polygon (using `cv::ellipse2Poly` of OpenCV [29]) to obtain the respective vertices. Similar to the linear segment, the first/starting point is ignored and the others are added to the contour's vertex list.

H. STEP & REPEAT (SR)

Lastly, given the number of repeats along the X and Y axes ($N_{SR,X}$ and $N_{SR,Y}$, respectively) and the step distances

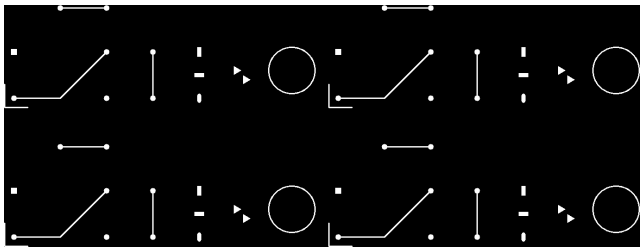


FIGURE 11. Renderization example of a step & repeat.

between elements (i_{SR} and j_{SR} along the X and Y axes, respectively), the parser executes the render method of each command of the SR command stream $N_{SR,X} \cdot N_{SR,Y}$ times. An offset O_{SR} is added when rendering all the graphic objects. For the u (where $u = 0, \ldots, N_{SR,X} - 1$) and v (where $v = 0, \ldots, N_{SR,Y} - 1$) replications along the X and Y axes, respectively, the offset to add when rendering each command of SR is formulated in (29). Figure 11 presents a renderization example of a step & repeat block that replicates a block of graphic objects 2 times along each X and Y axes.

$$O_{SR} = [i_{SR} \cdot u \; j_{SR} \cdot v]^T \tag{29}$$

V. RESULTS AND DISCUSSION

All the experimental results presented in this work are renderization of Gerber files into Tag Image Files (TIF). The latter is used instead of bitmaps (BMP) due to using lossless compression algorithms that allows reduced file sizes and preserved quality. For the results of the proposed parser, the Lempel–Ziv–Welch [32] (LZW) compression is used to save the TIF files. These reduced files facilitate sharing the results presented in this work. Indeed, all the examples shown in Section IV and the experimental results presented in this section are publicly available in a GitHub repository.[3] The repository contains the rendered images and the corresponding Gerber files. Additional documentation of the experimental methodology is provided in the repository.

Next, the Gerber viewers and libraries considered for the experiments in this work are introduced to the reader. After, two types of results are presented: accuracy evaluation of the renderized images and execution time of the parsing process. The former is relative to qualitative (visual inspection) and quantitative results (distance errors considering the desired millimeters per pixel resolution) retrieved from the renderized images. The latter evaluates the evolution of the execution time depending on the number of image objects present in the Gerber file of the proposed parsing architecture versus another parser.

A. TESTED GERBER VIEWERS AND LIBRARIES

According to the related Gerber information found in the literature as well as commercial online libraries and viewers, only a few fully support the Gerber X2 format specification, being able to handle additional Gerber commands compared to the previous X1 format, such as macro apertures, aperture blocks, and load rotation. As a result, the GerbView [14] (Software Companions) application is selected as an alternative to compare with the achieved results in terms of renderization accuracy due to stating that it is capable of interpreting Gerber X2 files. The standalone GerbLib [17] library available as a DLL file is assessed to evaluate the proposed approach's computational performance by having a processing time reference, while also being evaluated in terms of accuracy for Gerber X1 compatible files. In terms of ground-truth for the accuracy evaluation, the Reference Gerber Viewer [16] (Ucamco) will be used to retrieve the positions of certain image objects on the Gerber coordinate frame in millimeters. The distance between these known positions will be compared to the estimated ones based on the pixel distances and the desired resolution set for the renderization process.

1) GERBVIEW (SOFTWARE COMPANIONS)

The GerbView application supports the Gerber X2 format specification including interpreting the pad attributes. This application allows exporting the Gerber file into the following image formats: BMP, TIF, JPG, PNG, PDF, Cals Type 1 (CAL), HP-RTL (PLT), or WebP Image (WEBP). In terms of TIF files, the compression options available are the LZW, PackBits, or no compression. The LZW lossless compression algorithm was the one used to retrieve the renderization results with GerbView in this work. Although the resolution of the image file is set in DPI, the desired resolution in millimeters per pixel can be converted to DPI by (30). Furthermore, the option of selecting only the Gerber file and not the whole workspace fits the renderized image to only the image data contained in the Gerber file. This option is used for all the experiments with GerbView given that allows a direct comparison of the image size versus the Gerber file size given by the Reference Gerber Viewer, considering the resolution set on the renderization process. Lastly, the software has a free trial license of 30 days that allowed retrieving the experimental results presented in this work.

$$\mathrm{res}_{\mathrm{DPI}} = \frac{25.4}{\mathrm{res}_{\mathrm{mm/px}}} \tag{30}$$

2) GERBLIB (GERBMAGIC)

As for GerbLib, this standalone DLL library supports the Gerber X1 format, exporting Gerber files to the following formats: PDF, Postscript, TIF, BMP, or RID files. The renderization resolution in the case of outputting image files is set in DPI. Similarly to GerbView, the resolution units considered in this work, millimeters per pixel, are converted to DPI as in (30). Although margins can be added to the rendered image, all experimental results retrieved with GerbLib use zero margins and bottom-left alignment. The former allows a direct comparison between the image and the Gerber file size depending on the desired resolution. The latter is to avoid

[3] https://github.com/sousarbarb/V7DParser/

cutting the renderized image when there is image data outside the first quadrant of the Gerber coordinate frame. In terms of licensing, the GerbLib library allows 20 free renderization executions.

3) REFERENCE GERBER VIEWER (UCAMCO)

The developer of the Gerber format specification Ucamco has available the free online Reference Gerber Viewer to visualize Gerber files, among other formats. This viewer allows reliable visualization of Gerber files due to being compatible with the Gerber X3 specification and being developed by the creator of the Gerber format. Although the platform does not allow exporting the renderization output or defining the desired resolution, the Reference Gerber Viewer has useful tools to retrieve ground-truth data relative to the Gerber coordinate frame in millimeters. Some of these features are measuring distances and clearances, showing the attributes of the Gerber commands when hovering over objects (e.g., diameter of apertures or the flash center position), and computing the size of the file in millimeters.

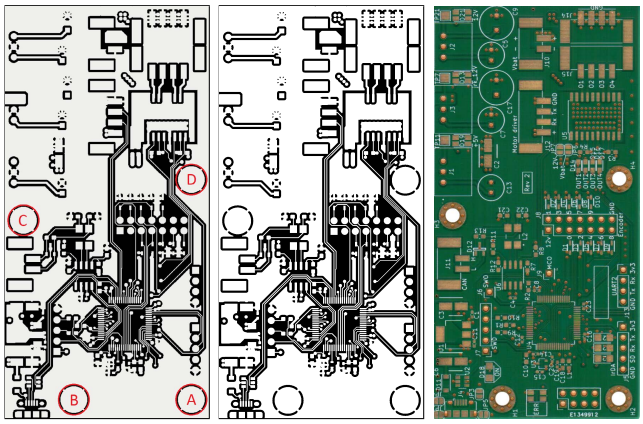
B. ACCURACY EVALUATION

First, the renderization accuracy was evaluated for the parsers tested in this work. Two different types of evaluations are analyzed here. The first is visual inspection of the renderization output. Given the vectorized renderization from a Gerber file generated on the Reference Gerber Viewer online platform, this renderization is compared to the ones generated by the developed parser, the GerbLib, and the GerbView. The other type of evaluation is a quantitative one. The size of the rendered image is compared to the real size computed by the Reference Gerber Viewer, when considering a certain resolution set upon the renderization process. The real size in millimeters can be retrieved from the characteristics upon the importation of the Gerber file computed on the Reference Gerber Viewer from Ucamco. Similarly, the distance between known points, such as centers of circle standard apertures upon their flash, are selected to compute a distance error. Equations (31) and (32) represent the sum of size and distance errors, respectively.

$$\varepsilon_{\text{size}} = |\varepsilon_{\text{size,width}}| + |\varepsilon_{\text{size,height}}| \tag{31}$$

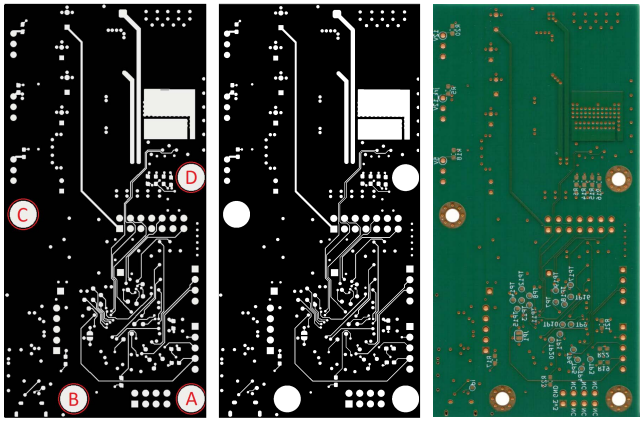
$$\varepsilon_{\text{dist}} = \sum_{i} \sum_{j} |\varepsilon_{\text{dist},ij}| \tag{32}$$

Two different Gerber files from a PCB board are used for the first tests. These files only contain Gerber commands compatible with the Gerber X1 format specification. Therefore, it is possible to compare three different parsers: the one proposed in this article represented by V7DParser in the results, GerbView from Software Companions, and the GerbLib dynamic library. Three resolutions are considered for the discussion: 0.050 mm/px, 0.010 mm/px, and 0.005 mm/px (equivalent to 508dpi, 2540dpi, and 5080dpi, respectively). The first resolution was chosen as half of the smaller element in the Gerber files. In both files, the smallest



(a) (b) (c)

FIGURE 12. Front view of the PCB: (a) ground-truth; (b) V7DParser; (c) photo.



(a) (b) (c)

FIGURE 13. Back view of the PCB: (a) ground-truth; (b) V7DParser; (c) flipped photo.

Gerber element is a circle standard aperture with a 0.1 mm diameter (aperture D46 in the first file, and D22 and D30 in the case of the second file). The other resolutions were selected as 5 times and 25 times greater than 0.050 mm/px (0.010 mm/px and 0.005 mm/px, respectively). Figures 12 and 13 illustrate the renderization results of the V7DParser compared to the ground-truth and a photo of the front and back views of the PCB, respectively. Figures 12a and 13a outline the circle apertures of which their center points (A, B, C, and D) will be used to retrieve distance data. Table 7 presents the quantitative results computed from the sizes of the renderized images and error distance data between points A, B, C, and D represented in the Figures for the two PCB Gerber files.

In terms of visual inspection, there was not noted any significant difference between the rendered images using either V7DParser, GerbView, GerbLib, or relative to the one given by the Reference Gerber Viewer. This observation is the main reason for not showing the three renderizations generated by

TABLE 7. Size, distances, error size and error distance sums, in mm, of the GerbView, GerbLib and V7DParser software regarding PCB front and back. Taking into consideration the ground-truth values given by the Reference Gerber Viewer, the lowest values of the absolute error module for the measured size and distance are shown in blue and bold.

	Resolution (mm/px)	Software	Size (mm)		ε_{size} (mm)	Distance (mm)						ε_{dist} (mm)
			Width	Height		AB	AC	AD	BC	BD	CD	
PCB Front	–	*ground-truth*	*98.934*	*48.934*	–	*28.00*	*58.73*	*52.75*	*44.64*	*59.72*	*41.17*	–
	0.05	GerbView	98.95	48.95	**0.03**	27.95	58.73	52.75	44.66	59.70	41.17	0.09
		GerbLib	98.90	48.90	0.07	28.00	58.73	52.75	44.64	59.72	41.17	**0.00**
		V7DParser	98.95	48.95	**0.03**	28.00	58.73	52.75	44.64	59.72	41.17	**0.00**
	0.01	GerbView	98.94	48.94	**0.01**	28.00	58.73	52.75	44.64	59.72	41.17	**0.00**
		GerbLib	98.93	48.93	**0.01**	28.00	58.73	52.75	44.64	59.72	41.17	**0.00**
		V7DParser	98.93	48.93	**0.01**	28.00	58.73	52.75	44.64	59.72	41.17	**0.00**
	0.005	GerbView	98.935	48.935	**0.002**	28.003	58.728	52.750	44.642	59.722	41.171	0.004
		GerbLib	98.930	48.930	0.008	28.000	58.728	52.750	44.643	59.721	41.171	**0.000**
		V7DParser	98.935	48.935	**0.002**	28.000	58.728	52.750	44.643	59.721	41.171	**0.000**
PCB Back	–	*ground-truth*	*98.225*	*47.55*	–	*28.00*	*59.46*	*52.75*	*45.61*	*59.72*	*40.95*	–
	0.05	GerbView	98.25	47.55	**0.03**	28.00	59.46	52.75	45.61	59.72	40.95	**0.00**
		GerbLib	98.20	47.50	0.08	28.00	59.45	52.75	45.60	59.72	40.92	0.05
		V7DParser	98.25	47.60	0.08	28.00	59.46	52.75	45.61	59.72	40.95	**0.00**
	0.01	GerbView	98.23	47.55	**0.01**	28.00	59.46	52.75	45.61	59.72	40.95	**0.00**
		GerbLib	98.22	47.54	0.02	28.00	59.46	52.75	45.61	59.72	40.95	**0.00**
		V7DParser	98.23	47.56	0.02	28.00	59.46	52.75	45.61	59.72	40.95	**0.00**
	0.005	GerbView	98.225	47.550	**0.000**	28.000	59.464	52.750	45.607	59.721	40.946	**0.000**
		GerbLib	98.225	47.545	0.005	28.000	59.464	52.750	45.609	59.723	40.946	0.005
		V7DParser	98.230	47.555	0.010	28.000	59.464	52.750	45.607	59.721	40.946	**0.000**

the three parsers, even though all rendered images are present in the public repository referred in this section.

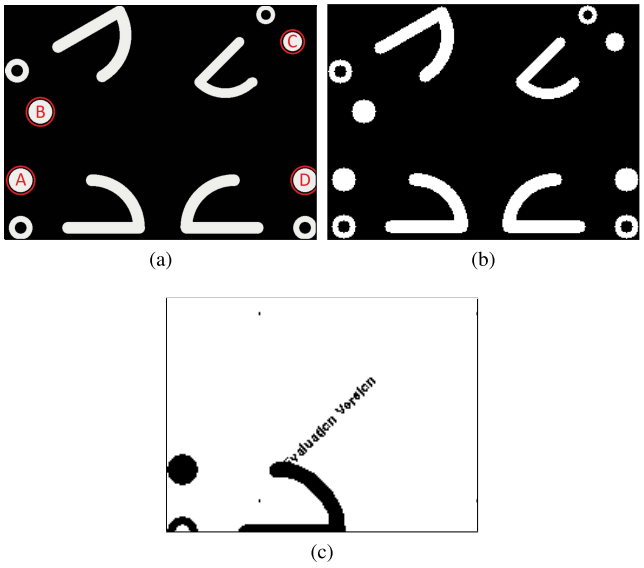
Analyzing the quantitative error results shown in Table 7, and taking the Reference Gerber Viewer as the reference, the quantitive errors indicate that the V7DParser accuracy is comparable to the other considered software highlighting its potential in the production of the electronic circuit. Assessing the sum of distance errors of the parsers regarding the distance between four points, GerbView and GerbLib have the highest error when compared with zero error from V7DParser. However, the values are not significant and are mostly due to pixelization, i.e., no more than 1 pixel due to resolution issues. This observation depends on the way the parsers deal with the pixelization of graphic objects. In the case of a circular aperture, its diameter is always an odd number of pixels for the V7DParser rendered images due to the implementation of `cv::circle`. This approach allows centering the circle in the closest pixel using (1). For GerbLib and GerbView, it was noted that they usually render circle aperture with an even number of pixels. The main disadvantage is the center of the circle not being a specific pixel but instead between four pixels. Even though the results for GerbView and GerbLib considered the 0.5 px values of the image coordinates to estimate the distance errors (expected to decrease them, instead of using the integer pixel values), the V7DParser accomplished the lower distance error. Concerning size errors, GerbLib has the highest value, on average for most of the considered cases, but once again the values are not considered significant given that some issues such as the pixelization or rounding measures play a role in this matter.

Next, to evaluate and compare the V7DParser software regarding the support of the Gerber X2 format specification, an experiment was performed using a Gerber file containing an aperture block. To the best of authors knowledge, the GerbView application is the only one that fulfills the requirements for this experiment, mainly, the support of the Gerber X2 format. Figure 14 presents the rendered images from the Reference Gerber Viewer (ground-truth), the proposed parsing approach (V7DParser), and the GerbView. Table 8 presents the quantitative error results for the 0.05 mm/px resolution. This resolution is equivalent to 10× greater than the smaller Gerber element in the file (i.e., the D11 circle aperture with a 0.5 mm diameter).

For this second experiment, GerbView did not render correctly the Gerber file, as shown in Figure 14. Although GerbView was able to render part of the first flashed aperture, the other ones that suffer rotations, mirroring and/or scaling transformations were not successfully rendered. In contrast, the proposed V7DParser was able to deal not only with the aperture block but also with all geometric transformations set on the graphics state. Furthermore, GerbLib was also tested, and, as was expected since it only supports Gerber X1, the standalone library was not able to produce the aperture block results. Consequently, Table 8 only presents the quantitative errors for the V7DParser considering the ground-truth data given by the Reference Gerber Viewer. Similar to the first

TABLE 8. Size, distances, error size and error distance sums, in mm, of the V7DParser software compared with the Reference Gerber Viewer regarding an aperture block case.

Resolution (mm/px)	Software	Size (mm)		ε_{size} (mm)	Distance (mm)						ε_{dist} (mm)
		Width	Height		AB	AC	AD	BC	BD	CD	
–	*ground-truth*	*13.165*	*9.880*	–	*3.00*	*12.88*	*12.00*	*11.05*	*11.53*	*5.87*	–
0.05	**V7DParser**	13.20	9.95	0.11	3.02	12.90	12.00	11.05	11.52	5.87	0.05



(a) (b) (c)

FIGURE 14. Aperture Block: (a) ground-truth; (b) V7DParser; (c) GerbView.

experiment for accuracy evaluation, the errors are mostly due to pixelization. All the single distance errors are lower than the renderization resolution. For the size measures, only the height measure has an error higher than the desired resolution, i.e., 0.07 mm. These results demonstrate, again, the high renderization accuracy of the proposed parsing approach while being compatible with the graphic objects defined in the Gerber X2 format specification.

C. COMPUTATIONAL PERFORMANCE

Although the focus of this work was not on the time performance of rendering Gerber files but on its accuracy, an execution time evaluation was performed by flashing a different number of circle standard apertures. These apertures have a diameter of 1 mm, and the renderization resolution was 0.002 mm/px. The latter is due to decreasing the influence of the application execution timing in the results by increasing the renderization time (higher resolutions require more execution time). The circle apertures are flashed along the line $x = y$ equally distant in both axes by 2 mm. Given that standard circle apertures are compatible with the Gerber X1 format specification, the execution time results of the proposed V7DParser are compared to the ones obtained using the GerbLib library. The number of experiments per Gerber file was 20 samples. Both parsers were tested in the same Windows virtual machine that had allocated 4 GB of memory and 1 virtual CPU. This virtual machine was run using Oracle VM VirtualBox on a laptop with Windows 10, an Intel Core i7-9750H CPU @ 2.60 GHz, and 12 GB of memory.

Two execution time experiments presented in this work are relative to flashing a different number of circle apertures: 1 to 40. The first experiment is based on flashing the apertures consecutively along the $x = y$ line starting on $(x, y) = (0, 0)$ mm spaced out evenly between each other by 2 mm. The second experiment differs from the first one by flashing firstly the most bottom-left and top-right apertures ($(x, y) = (0, 0)$ mm and $(78, 78)$ mm, respectively) of the 40 circle apertures considered in these experiments, and then flashing the other ones. This way the files from 2 up to 40 apertures have the same image area. Only the first file contains a single circle aperture having the image area occupied by its single aperture. The results of first and second experiments are presented in Figures 15 and 16, respectively.

Analyzing the results of the first experiment, both parsers have a polynomial complexity. This computational complexity can be due to increasing size of the image area or increasing the number of flashed apertures. Given the polynomial order of at least 2, it seems to indicate that the computational complexity of both parsers is dependent on increasing size of the image area due to increases in both X and Y directions between consecutive Gerber files. This indication will be cleared out in the second experiment. Even so, up to 11 circle apertures, the proposed parsing architecture has lower execution times than GerbLib, in terms of median values for the execution time. For files with more than 11 apertures, the GerbLib outperformed the V7DParser. One possible reason for the results obtained with the proposed parsing process is due to dynamically changing the size of the rendered image when rendering each object of the Gerber file, in contrast to GerbLib. The latter has a specific function, `GbxGetPageSize`, to compute the image size in inches before the image is rendered and saved. As seen in the results, the advantage of the GerbLib approach is a faster renderization process mostly due to preallocating the image size. However, a disadvantage of preallocation is the renderization process dependency on the interpretation of the Gerber file. For example, the proposed dynamic resize approach allows to interpret the Gerber files, change parameters on specific pads if necessary in runtime, and then rendering after these modifications. In the case of GerbLib, the modifications must be made directly in the file. Also, when using GerbLib, the parser must always interpret the file again.

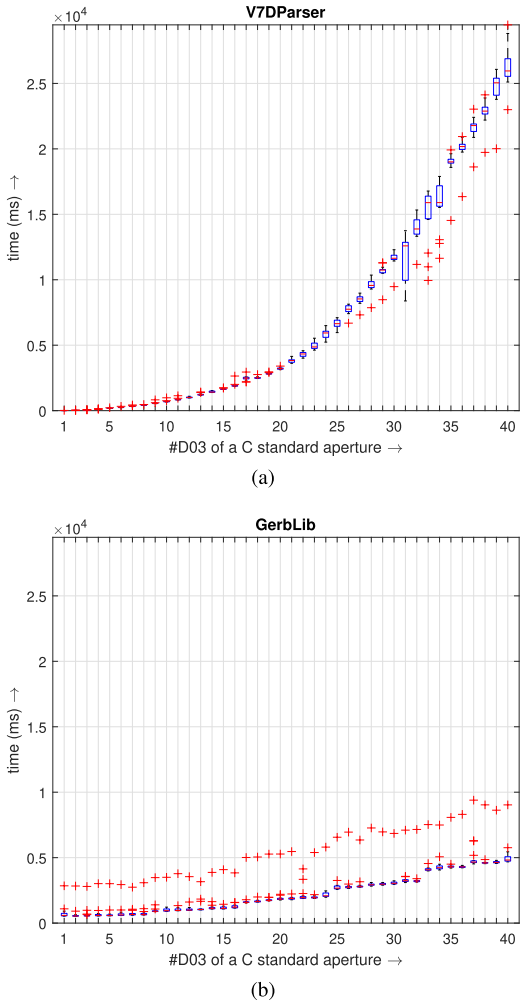


FIGURE 15. Execution time performance of V7DParser versus GerbLib parsers when rendering a Gerber file composed by an increasing number of C standard apertures (1–40) and increasing board area. (a) V7DParser; (b) GerbLib.

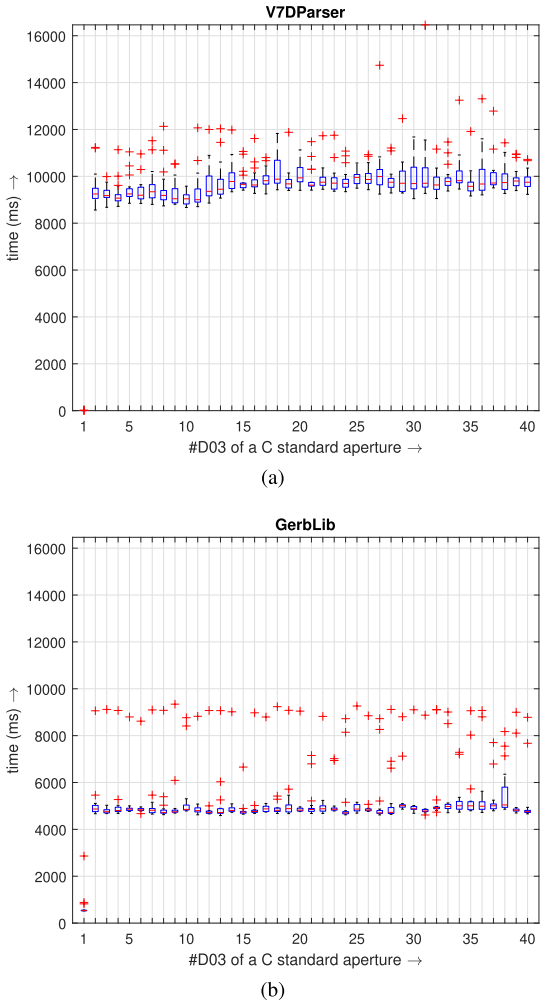


FIGURE 16. Execution time performance of V7DParser versus GerbLib parsers when rendering a Gerber file composed by an increasing number of C standard apertures (1–40) and static board area. (a) V7DParser; (b) GerbLib.

The goal of the second experiment is to clarify if the complexity is dependent on the image area or the number of graphic objects represented in the Gerber file. By having the same image area for the files with 2 up to 40 apertures, the image size remains constant and dynamic resizing is only executed between the first and second apertures (the latter when using V7DParser). Analyzing Figure 16, the execution time remains approximately constant over all files except for the first one, which the latter has a single circle aperture. These results indicate that the main factor for the execution time is the area of the rendered image. In the case of GerbLib, the image for #$D03 = 2..40$ has 39500 x 39500 px, exactly the same size as for #$D03 = 40$ in the first experiment. Indeed, the median execution time is approximately the same: 4820.87 ms versus 4758.34 ms. These results supports the statement of the executing time being highly dependent on the image size.

In the case of the proposed approach, the image size remains the same when rendering the Gerber files equivalent to #$D03 = 2..40$ in the second compared to #$D03 = 40$ in the first experiment (39501 x 39501 px). However, the median execution time decreased between the second and first experiments for #$D03 = 40$: 9747.98 ms versus 25954.02 ms, respectively. The main reason for this optimization is that dynamically resizing the image requires a reallocation of memory and the subsequent copy of the current image before rendering the next graphic object in the resized image. Indeed, the first experiment leads to resizing the image 40 times, given that the 40 flash operations increase the image by the same amount of times. The results obtained with the proposed parsing approach in the two experiments show that pre-allocating the image size before rendering Gerber commands improves the parser's computational performance. Even though the focus of this work was not on the computational performance, and dynamic resizing makes the interpretation phase independent of the renderization one, pre-allocating the image size should be the approach used, especially for higher resolutions and/or bigger image areas.

D. DISCUSSION

Overall, the proposed parsing architecture achieved similar accuracy results to the current solutions compared to the ground-truth data given by the Reference Gerber Viewer. The renderization errors obtained in the experiments for all parsers tested decrease with the increase of resolution. In addition to the errors being in most of the cases no more than 1 pixel, these results indicate that the main accuracy errors shown in Table 7 are due to the resolution itself, i.e., the limited millimeter per pixel resolution that results in pixelization of the vector data described in the Gerber file. Furthermore, this work addresses the ambiguities of the Gerber specification for rendering rotated apertures and circular interpolation arcs. The first ambiguity was addressed by first rotating the aperture's points of interest, and then rendering it. The second one was handled by rendering the circular arc to always pass through its initial endpoints defined in the Gerber file. These proposed approaches led to no deformation as illustrated in the renderization of the aperture block.

A limitation of the proposed approach is the computational performance. The main cause is the dynamic resize affecting the computational performance, even though it facilitates the implementation of the interpretation and renderization processes while also making these two processes independent of each other. A possibility identified in the results discussed is estimating the rendered image size upon the file's interpretation, for allocating the image size a priori to the renderization process. This modification not only improves the computational time but also its complexity relative to the image size. Even so, the computational performance was not the main focus of this work but instead the renderization accuracy; so, the a priori estimation of the image size will be a future improvement on the current parsing architecture.

Finally, it should be highlighted the full support for all commands defined in the Gerber X2 specification that generate renderization data, unlike existing solutions. These commands include renderization of aperture blocks, macro apertures, interpretation of arithmetic expressions defined within a macro template definition, and load rotation. The support of the Gerber X3 format was not required for this work. The additional structures defined in this format are focused on defining new attributes for the fabrication process of a PCB, not on structures that add image data to the renderization output. Along with rendering Gerber X2 files, the proposed approach allows exporting in multiple formats including bitmaps due to the use of OpenCV for this purpose while defining a desired millimeter per pixel resolution.

VI. CONCLUSION AND FUTURE WORK

Nowadays, the electronics fabrication industry resorts to the Gerber open standard format to exchange PCB manufacturing specifications. The industry is also facing a global competition and new production methods are being devised day by day. Regarding the automotive industry, it is looking for innovations in its production processes and also in the design of automobile interiors. One possibility being developed is the printing, using functional and decorative inks, of sensors and actuators directly on automobile interior parts. Given this environment, the objective of the VINCI7D project consists of the development of a solution based on a fixed inkjet printhead and an industrial robot arm that manipulates a part with a 3D surface. This solution must be able to print electrical circuits directly on the surface. For this purpose, a parser for the Gerber language was developed, accepting Gerber files as input and bitmap images as outputs, to be supplied to the printhead.

To be best of the authors' knowledge, and according to the related Gerber information founded in the literature as well as into the commercial online libraries and viewers data, only a few fully support the Gerber X2 generation, being able to handle with Gerber commands such as macro apertures, aperture blocks and load rotation. The proposed work contemplates the Gerber X2 generation, taken into consideration the Gerber specification from Ucamco, and follows a recursive descent approach for parsing all the reviewed Gerber commands. The distinction between this parser and others already described in the literature resides in the fact that this one is able to convert all Gerber commands, including Gerber macros (a complex structure of the format not considered in other works), contrary to the others which are only able to support a subset of this format. GerbView (Software Companions) application was assessed as an alternative to compare with the V7DParser accuracy results. GerbLib, a standalone library which provides a DLL and that only supports Gerber X1, was assessed to evaluate the proposed approach's accuracy and computational performance by having a processing time reference. Regarding the accuracy evaluation, the obtained results from V7Parser were comparable with GerbViewer and GerbLib software, and with the used ground-truth (Reference Gerber Viewer – Ucamco). Concerning the aperture block analysis, only handled by the Gerber X2 generation, the results achieved from V7DParser were aligned with the reference; however, GerbView was not able to render the Gerber command, even though the application states that it is compatible with the Gerber X2 format. Regarding the time performance evaluation, the performed tests based on dynamic size and preallocating the image size led to the conclusion that pre-allocating the image size improves substantially the computational performance of the parser. Finally, given the accuracy evaluation results of the proposed parsing architecture, the V7DParser is a real option to the electronics fabrication industry. Furthermore, due to V7DParser top-down and recursive characteristics, its architecture allows an easy integration with other software regardless of the platform.

Nonetheless, further development is currently being carried out to improve the functionalities of the parser, including to improve the computational performance based on the current experimental results and to integrate this parser for projecting 2D Gerber image data onto 3D surfaces. Moreover, the work will be directed to the projection of the bitmap

images over 3D surfaces, and its printing over solid parts with complex geometries.

ACKNOWLEDGMENT

The authors would like to thank Rui Azevedo—Soluções de Acabamentos e Tampografia in the scope of the VINCI7D Project for its support and assistance in the development of this work.

REFERENCES

[1] A. Mohapatra, B. I. Morshed, S. Shamsir, and S. K. Islam, "Inkjet printed thin film electronic traces on paper for low-cost body-worn electronic patch sensors," in *Proc. IEEE 15th Int. Conf. Wearable Implant. Body Sensor Netw. (BSN)*, Mar. 2018, pp. 169–172, doi: 10.1109/BSN.2018.8329685.
[2] X. Feng, A. Scholz, M. B. Tahoori, and J. Aghassi-Hagmann, "An inkjet-printed full-wave rectifier for low-voltage operation using electrolyte-gated indium-oxide thin-film transistors," *IEEE Trans. Electron Devices*, vol. 67, no. 11, pp. 4918–4923, Nov. 2020, doi: 10.1109/TED.2020.3020288.
[3] T. Tilford, S. Stoyanov, J. Braun, J. C. Janhsen, M. K. Patel, and C. Bailey, "Comparative reliability of inkjet-printed electronics packaging," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 11, no. 2, pp. 351–362, Feb. 2021, doi: 10.1109/TCPMT.2021.3049952.
[4] Centre for Nanotechnology and Smart Materials (CeNTI), "Highly flexible displays for automotive interior applications," *OPE J.*, vol. 11, no. 36, pp. 12–13, Sep. 2021. [Online]. Available: https://www.coating-converting.com/epaper/c2com/193/epaper/9405/12/index.html
[5] Eurocircuits. *Gerber Formats*. Accessed: Oct. 7, 2021. [Online]. Available: https://www.eurocircuits.com/gerber-format/
[6] Ucamco. *Gerber Format*. Accessed: Sep. 8, 2021. [Online]. Available: https://www.ucamco.com/en/gerber
[7] I. R. Sinclair and J. Dunton, *Practical Electronics Handbook*, 6th ed. Amsterdam, The Netherlands: Elsevier, 2007. [Online]. Available: http://www.aeroelectric.com/Reference_Docs/Books/Practical_Electronics_%Handbook_6th_ed.pdf
[8] D. Grune and C. J. Jacobs, *Parsing Techniques* (Monographs in Computer Science), 2nd ed. New York, NY, USA: Springer, 2008, doi: 10.1007/978-0-387-68954-8.
[9] PCBWay. *Online Gerber Viewer PCB Prototype the Easy Way*. Accessed: Aug. 23, 2021. [Online]. Available: https://www.pcbway.com/project/OnlineGerberViewer.html
[10] JLCPCB. *PCB Order & Online PCB Quote & SMT Assembly Quote*. Accessed: Aug. 23, 2021. [Online]. Available: https://cart.jlcpcb.com/quote
[11] Bronzware. *Gerbmagic*. Accessed: Mar. 24, 2022. [Online]. Available: https://www.bronzware.com/GerbMagic/index.htm
[12] Numerical Innovations. *FAB 3000 Gerber Viewing, Editing, Panelization, DRC, DFM, Compare Nets, Build Centroids, SMT Stencil Pads, Convert DXF, Odb++ and More*. Accessed: Aug. 23, 2021. [Online]. Available: https://www.numericalinnovations.com/collections/fab-3000-gerber-cam
[13] R. Powierski. *ZofzPCB: Free 3D Gerber Viewer + Premium Step Export*. Accessed: Aug. 23, 2021. [Online]. Available: https://www.zofzpcb.com/
[14] S. Companions. *Gerbview*. Accessed: Aug. 24, 2021. [Online]. Available: https://www.gerbview.com/
[15] R. Poelstra. *Cuprum—The Gerber Viewer Built Exclusively for MAC*. Accessed: Aug. 23, 2021. [Online]. Available: https://www.wortum.com/cuprum/
[16] Ucamco. *Reference Gerber Viewer*. Accessed: Aug. 23, 2021. [Online]. Available: https://gerber-viewer.ucamco.com/
[17] GerbMagic. *Gerblib Functions*. Accessed: Aug. 23, 2021. [Online]. Available: https://www.bronzware.com/GerbMagic/functions.htm
[18] Artwork Conversion Software. *Gbr_Rip High Speed Multi-Threaded Gerber Rasterizer*. Accessed: Aug. 23, 2021. [Online]. Available: https://www.artwork.com/gerber/gbr_rip/index.htm
[19] P. Prokofyev. *Libpger*. Accessed: Aug. 23, 2021. [Online]. Available: https://github.com/integralpro/libpger.git
[20] J. Stringer. *Gerberparser*. Accessed: Aug. 23, 2021. [Online]. Available: https://github.com/JulesStringer/GerberParser.git
[21] M. Cousins. *Gerber-Parser*. Accessed: Aug. 23, 2021. [Online]. Available: https://github.com/mcous/gerber-parser.git
[22] Mike Cousins. *Tracespace*. Accessed: Aug. 23, 2021. [Online]. Available: https://github.com/tracespace/tracespace.git
[23] M. Cousins and Contributors. *Tracespace View*. Accessed: Aug. 23, 2021. [Online]. Available: https://tracespace.io/view/
[24] KiCad. *Gerbview*. Accessed: Aug. 23, 2021. [Online]. Available: https://gitlab.com/kicad/code/kicad/-/tree/master/gerbview
[25] KiCad. *KiCad EDA: A Cross Platform and Open Source Electronics Design Automation Suite*. Accessed: Aug. 23, 2021. [Online]. Available: https://www.kicad.org/
[26] M. Qi, Z. Wang, X. Wei, and A. Wang, "Efficient gerber file parsing and drawing," in *Proc. Int. Conf. Mach. Learn. Mach. Intell. (MLMI)*, 2018, pp. 13–17, doi: 10.1145/3278312.3278327.
[27] Y. Fan, S. Wu, X. Wu, and J. Yang, "Gerber file parsing and the implementation method of its conversion to bitmap image," *J. Phys., Conf. Ser.*, vol. 1820, no. 1, Mar. 2021, Art. no. 012160, doi: 10.1088/1742-6596/1820/1/012160.
[28] Ucamco. (Apr. 2021). *The Gerber Layer Format Specification*. Ucamco. [Online]. Available: https://www.ucamco.com/files/downloads/file_en/436/gerber-layer-format-%specification-revision-2021-04_en.pdf
[29] OpenCV. *Open Source Computer Vision Library (OpenCV)*. Accessed: Oct. 10, 2021. [Online]. Available: https://opencv.org/
[30] Ucamco. (Sep. 2020). *The Gerber Layer Format Specification*. Ucamco. [Online]. Available: https://www.ucamco.com/files/downloads/file_en/399/the-gerber-file-form%at-specification-revision-2020-09_en.pdf?df3a1d29deffdbc28f2712cb5a613ae6
[31] B. N. Miller and D. L. Ranum, *Problem Solving With Algorithms and Data Structures using Python*, 2nd ed. Portland, OR, USA: Franklin Beedle & Associates, 2013. [Online]. Available: https://runestone.academy/runestone/books/published/pythonds/BasicDS/In%fixPrefixandPostfixExpressions.html
[32] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, Jun. 1984.

RICARDO B. SOUSA received the Master of Science (M.Sc.) degree in electrical and computers engineering (ECE) from the Faculty of Engineering, University of Porto (FEUP), in 2020, where he is currently pursuing the Ph.D. degree in electrical and computer engineering. He has a Graduate Research Scholarship from FCT—Fundação para a Ciência e a Tecnologia, Centre for Robotics in Industry and Intelligent Systems, INESC TEC. Also, he is currently an Invited Assistant lecturing the courses software design and industrial informatics from the M.Sc. degree in ECE at FEUP. His research interests include robotics, sensor fusion, and localization and mapping for autonomous robots.

CLÁUDIA ROCHA received the M.Sc. degree in bioengineering in biomedical engineering from the Faculty of Engineering, University of Porto, Portugal, in 2016. She joined the Centre for Robotics in Industry and Intelligent Systems, INESC TEC, in 2016, as a Researcher, participating in various projects. Her research interests include robotics, automation, 3D modeling, and healthcare.

HÉLIO SOUSA MENDONÇA was born in Porto, Portugal, in 1968. He received the Graduate and M.Sc. degrees in electrical and computer engineering from the Faculty of Engineering, University of Porto (FEUP), in 1991 and 1994, respectively, and the Ph.D. degree, in 2004. Currently, he is an Assistant Professor at FEUP and a Researcher at the Centre for Robotics in Industry and Intelligent Systems (CRIIS), INESC TEC. His research interests include embedded systems, wireless sensor networks, and robotics.

ANTÓNIO PAULO MOREIRA received the Graduate degree in electrical engineering from the University of Oporto, in 1986, the M.Sc. degree in electrical engineering—systems from the University of Porto, in 1991, and the Ph.D. degree in electrical engineering, in 1998. He is currently an Associate Professor with tenure at the Faculty of Engineering, University of Porto and a Researcher and the Head of the Centre for Robotics in Industry and Intelligent Systems, INESC TEC. His research interests include process control and robotics.

MANUEL F. SILVA was born in April 1970. He received the B.Sc., M.Sc., and Ph.D. degrees in electrical and computer engineering from the Faculty of Engineering, University of Porto, in 1993, 1997, and 2005, respectively. He is currently a Coordinating Professor at the Department of Electrical Engineering, Instituto Superior de Engenharia do Porto (having supervised or co-supervised more than 60 M.Sc. theses and 80 graduation projects/internships) and a Senior Researcher at the Centre for Robotics in Industry and Intelligent Systems, INESC TEC. He is the author of more than 100 publications in international journals and conferences (https://www.researchgate.net/profile/Manuel-Silva-8) and has been involved in several research and development projects. He has also been actively involved in the organization of several international conferences, integrating the Management Team of the CLAWAR Association (https://clawar.org/) and the General Assembly Board of the Portuguese Robotics Society (of which he has been the President of the Steering Committee). His research interests include modeling, simulation, robotics, bio-inspired robotics, control and education in robotics, and control.