Paper ID #44616

Converting Text Into 3D Printable Braille

Dax Amburgy, Ohio Northern University College of Engineering

I am a Junior Computer Science major with a concentration in Cybersecurity.

Dr. John K. Estell, Ohio Northern University

An active member of ASEE for over 30 years, Dr. John K. Estell was elected in 2016 as a Fellow of ASEE in recognition of the breadth, richness, and quality of his contributions to the betterment of engineering education. Estell currently serves as chair of ASEE's IT Committee; he previously served on the ASEE Board of Directors as the Vice President of Professional Interest Councils and as the Chair of Professional Interest Council III. He has held multiple ASEE leadership positions within the First-Year Programs (FPD) and Computers in Education (CoED) divisions, and with the Ad Hoc Committee on Interdivisional Cooperation, Interdivisional Town Hall Planning Committee, ASEE Active, and the Commission on Diversity, Equity, and Inclusion. Estell has received multiple ASEE Annual Conference Best Paper awards from the Computers in Education, First-Year Programs, and Design in Engineering Education Divisions. He has also been recognized by ASEE as the recipient of the 2005 Merl K. Miller Award and by the Kern Entrepreneurial Engineering Network (KEEN) with the 2018 ASEE Best Card Award. Estell received the First-Year Programs Division's Distinguished Service Award in 2019 and the 2022 Computers in Education Division Service Award.

Estell currently serves as an ABET Commissioner and as a subcommittee chair on ABET's Accreditation Council Training Committee. He was previously a Member-At-Large on the Computing Accreditation Commission Executive Committee and a Program Evaluator for both computer engineering and computer science. Estell is well-known for his significant contributions on streamlining student outcomes assessment processes and has been an invited presenter at the ABET Symposium on multiple occasions. He was named an ABET Fellow in 2021. Estell is also a founding member and current Vice President of The Pledge of the Computing Professional, an organization dedicated to the promotion of ethics in the computing professions.

Estell is Professor of Computer Engineering and Computer Science at Ohio Northern University, where he currently teaches first-year programming and user interface design courses, and serves on the college's Capstone Design Committee. Much of his research involves design education pedagogy, including formative assessment of client-student interactions, modeling sources of engineering design constraints, and applying the entrepreneurial mindset to first-year programming projects through student engagement in educational software development. Estell earned his BS in Computer Science and Engineering degree from The University of Toledo and both his MS and PhD degrees in computer science from the University of Illinois at Urbana-Champaign.

Converting Text Into 3D Printable Braille

Dax Amburgy
Electrical & Computer Engineering and Computer Science Department
Ohio Northern University
Ada, Ohio 45810
Email: d-amburgy@onu.edu

Introduction

Language is an important part of the transfer of information. For people who experience visual impairment, reading signs in buildings and public locations can be challenging. Common examples are restroom signs, room numbers, and memorial signs. For students on university campuses, these signs can be very common. There are two current methods of helping members of society with such impairments. The first method is using audio. Stations are set up with buttons that people can press to hear an audio recording of the message that needs to be transmitted. The other method is a written code of 2x3 matrix raised dot characters that the user can feel by running their finger over. This code, shown in figure 1, is known as braille.

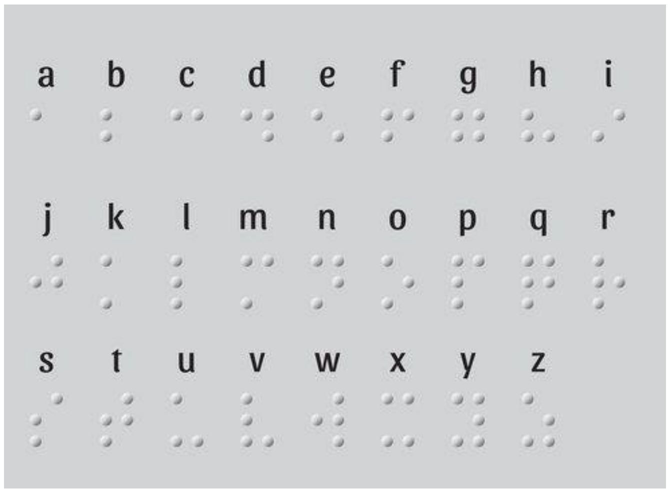
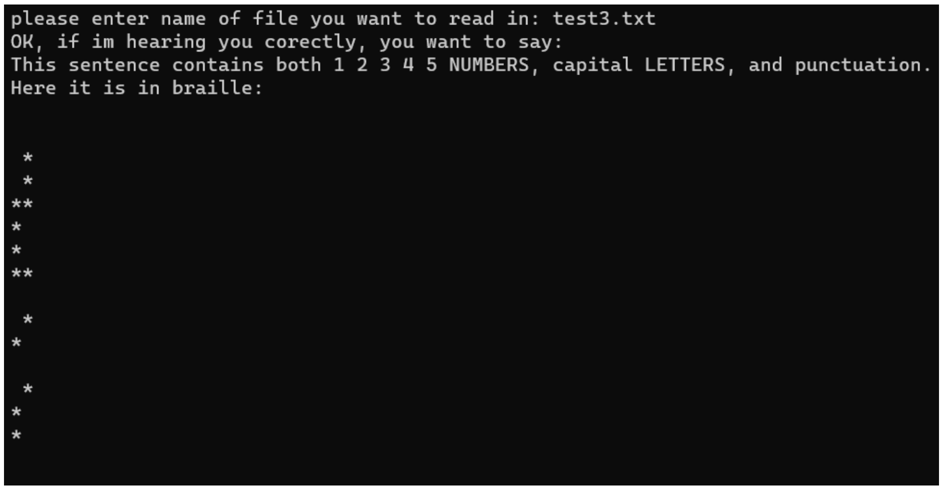


Figure 1: An example of the braille alphabet

While braille books and permanent signs fill the need of identifying unchanging information, there is currently not a method of developing braille signs for frequently changing information. If an office building has employees who often switch desks or a university has professors who switch classrooms, the braille signage identifying who is currently utilizing that space would need to be more flexible. This leaves the question: how can we provide the visually impaired with braille translations of text in venues where signage is regularly updated? The following research aims to create an answer to this question.

Methodology

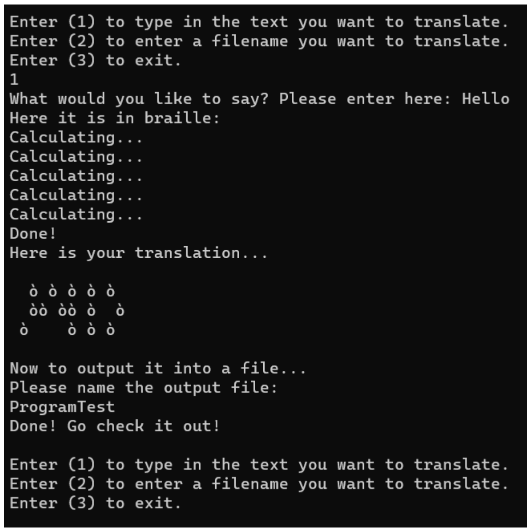
To solve this problem, the first question that must be answered is how a user can translate written words to braille. From a computer science perspective, a character in braille appears to be a 2x3 array data structure. This means that if 26 2x3 arrays can be generated (one for each character of the English language), they can be programmatically substituted for each character in a word, or each char in a string. In the first version of the Text to Braille program, the user could enter the name of a text file of any string of characters, and its exact representation was printed to the console in “Braille”. This was 2x3 matrices of the ‘*’ symbol. It could handle any sentence that included capital letters, punctuation, numbers, and contractions. When this initial algorithm translated each character, it output the matrix to the console and then generated a new line to write to. This caused the translation to be vertical instead of horizontal as seen in figure 2, which allowed for easy verification that each character was correct.



```
please enter name of file you want to read in: test3.txt
OK, if im hearing you corectly, you want to say:
This sentence contains both 1 2 3 4 5 NUMBERS, capital LETTERS, and punctuation.
Here it is in braille:


 *
 *
**
*
*
**

 *
*

 *
*
*
```

Figure 2: Vertical text translation

For the next iteration of the code, the goal was to render the braille translation horizontally. To achieve this, each matrix had to be split into three lines. Then, each line from that matrix was stored into one of three string global variables. These three strings could then be displayed and cleared for the next line of text. In version 2 of the program, users could select a file that they wanted translated, or they could type out directed into the console what they would like to say. This can be seen in figure 3.



```
Enter (1) to type in the text you want to translate.
Enter (2) to enter a filename you want to translate.
Enter (3) to exit.
1
What would you like to say? Please enter here: Hello
Here it is in braille:
Calculating...
Calculating...
Calculating...
Calculating...
Calculating...
Done!
Here is your translation...

  ò ò ò ò ò
  òò òò ò  ò
 ò    ò ò ò

Now to output it into a file...
Please name the output file:
ProgramTest
Done! Go check it out!

Enter (1) to type in the text you want to translate.
Enter (2) to enter a filename you want to translate.
Enter (3) to exit.
```

Figure 3: Translating the word “Hello” from the command line

This version of the program was also updated to use the Unicode bullet point "•" rather than the asterisk "*" to improve the consistency of displaying the braille characters. However, as seen in figure 3, this character occasionally does not display correctly in certain text editors. This error is inconsequential to answering the next question; how can someone with a visual impairment read this digitally translated text?

Currently in production are devices known as braille readers [4]. One specific model known as the Orbit Reader 20 costs $700. This can be a very expensive solution, and also requires the device to be always present at the location of where the translation needs to be. If there are multiple locations of translations, then many of these devices are required. These devices could also be easily damaged if not properly taken care of and then would need to be replaced further increasing costs. In recent years, however, a new technology has risen in popularity and abundance that can prove to be an easier and more cost-effective option. This technology is 3D printing. If the translation could be 3D printed, then plastic signs could be displayed where the translation is needed. These signs can be cheaply and quickly produced, and if damaged would not be hard to replace. But how can digital text be converted into a physical sign?

The first step in answering this question was to analyze the literature on 3D printing 2D objects. The first solution found was if it was possible to 3D print ASCII text [1]. By using a 3D printing technique known as a lithophane, an ASCII text art piece was able to be 3D printed. This technique works by taking the darker parts of the image and creating more 3D printed layers in those spots so that when light shines behind the print the negative light will reveal the image. While the 3D printer was not able to print at a high enough resolution to correctly display the characters, it was still a viable solution to the problem. By using an online .TXT to .JPG converter, the braille translation could be converted into an image as seen in figure 4. This image could then be used inside of CURA 3D printing software to be converted into a lithophane.

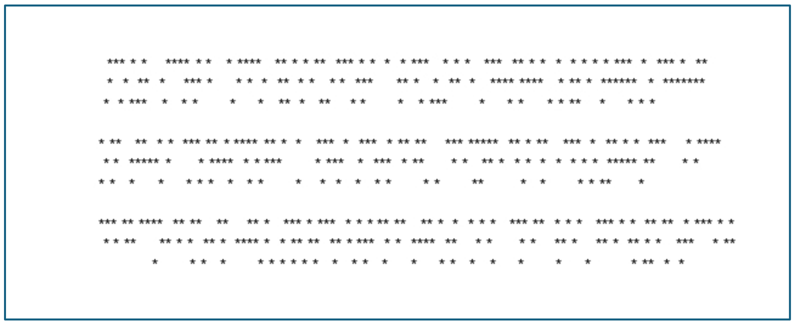


Figure 4: The first paragraph of the Gettysburg Address using asterisks

When using asterisks, many of the online converters worked well, however the 3D prints were very muddled and did not display each raised dot correctly. While the sizing of each bullet was close, certain bullets were misaligned from how they would be normally displayed. This was due to the conversion programs interpreting what it thinks is present, which can be seen in figure 5. Once the program was switched to Unicode bullet points, the 3D prints became much clearer. This can be seen in figure 6.

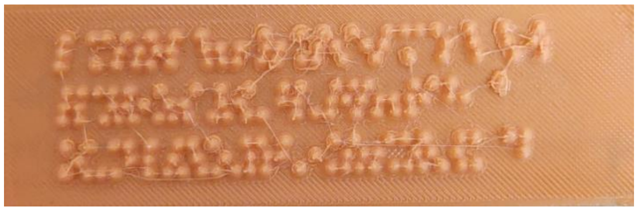


Figure 5: First test print using asterisks

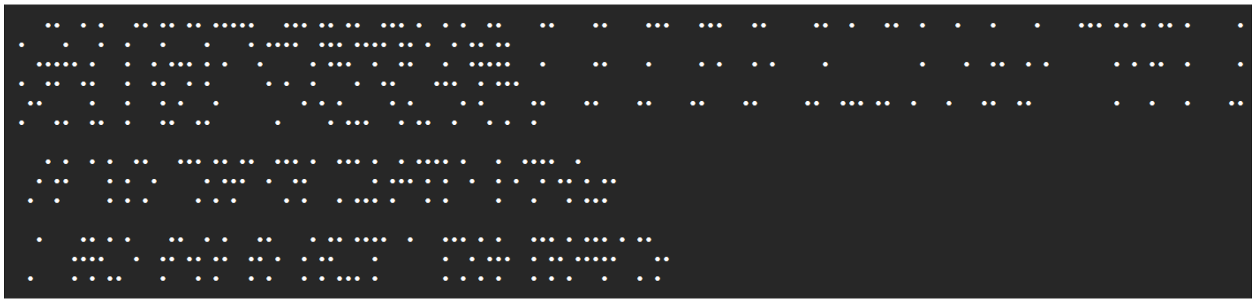
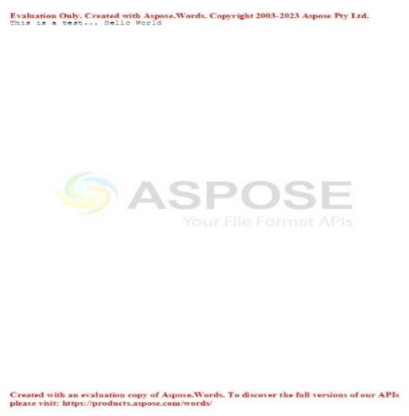


Figure 6: Translation using "•"

The perfect pipeline for converting text to 3D printed braille had been discovered. However, this process required 2 online conversions that a user would have to seek out in order to create these signs. The user must manually switch from .TXT – a text file, to .JPG – an image format, then from .JPG to .STL – a 3D model file, and finally from .STL to .GCODE – a processed 3D print file that can be printed from. The other important thing to consider was the user needed to make sure that all of the print settings are correctly selected every time the .JPG was imported into CURA for conversion into .STL. This leads to the final question; How can this pipeline be simplified for the user?

At this point, the current version of the code was written in the C++ programming language. Therefore, the goal was to find a solution in C++ to convert .TXT files to .JPG files. The first solution discovered was a C++ library known as Aspose.Cells [2]. This library, however, created a watermark when used that would not allow for clean .JPG to .STL conversion. This can be seen in figure 7.


Evaluation Only. Created with Aspose.Words. Copyright 2003-2023 Aspose Pty Ltd.

```
This is a test... Hello World
```

Created with an evaluation copy of Aspose.Words. To discover the full versions of our APIs please visit: https://products.aspose.com/words/

Figure 7: Aspose watermark example

In the process of searching for an adequate .TXT to .JPG converter, a python library was discovered that worked extremely well. This library was the Python Image Library (PIL). At the same time, a library for converting .JPG to .STL was also discovered for python [5]. This led to the decision to convert the entire text to braille script from C++ to python. Not only would this allow for the addition of these two libraries, but it would also make the code more approachable to a wider audience of programmers, especially since it is becoming more common for beginner programmers to learn python. With the addition of both libraries, users can now enter one file name or string, and it will instantly convert what they want into the 3D printable .STL format. An example of the final 3D printable file can be seen in figure 8.

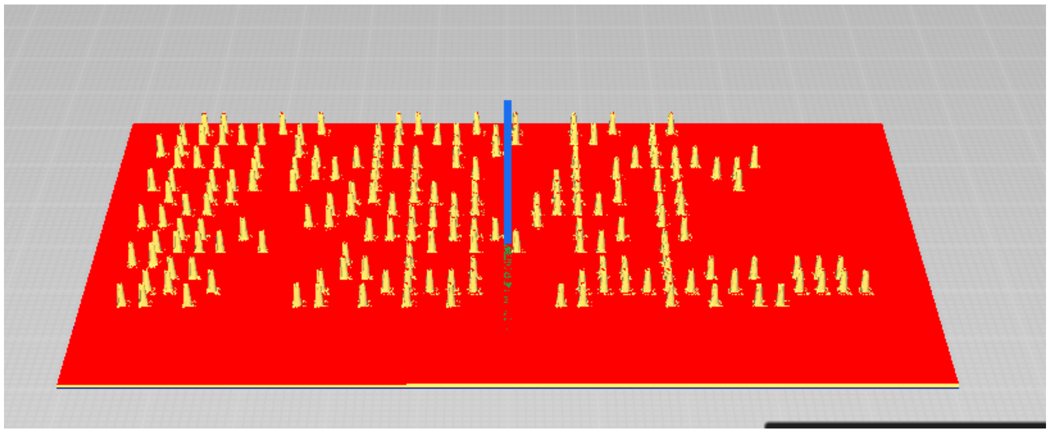


Figure 8: .STL imported into CURA from Python V3 version of code

One major issue that was occurring with this new code was that PIL had a variable which permanently set the size of the output .JPG file. This would mean that the image could cut off part of the translation. In order to solve this problem, a mathematical proof was developed to dynamically change the size of the image based on the number of characters in a specific line. The library measured the size of the image in pixels, so the first step was to figure out the exact size of one singular braille dot ("•") in pixels. PIL featured a function that calculated the X and Y coordinates of two opposing corners of a character, allowing for the calculations of the X and Y dimensions:

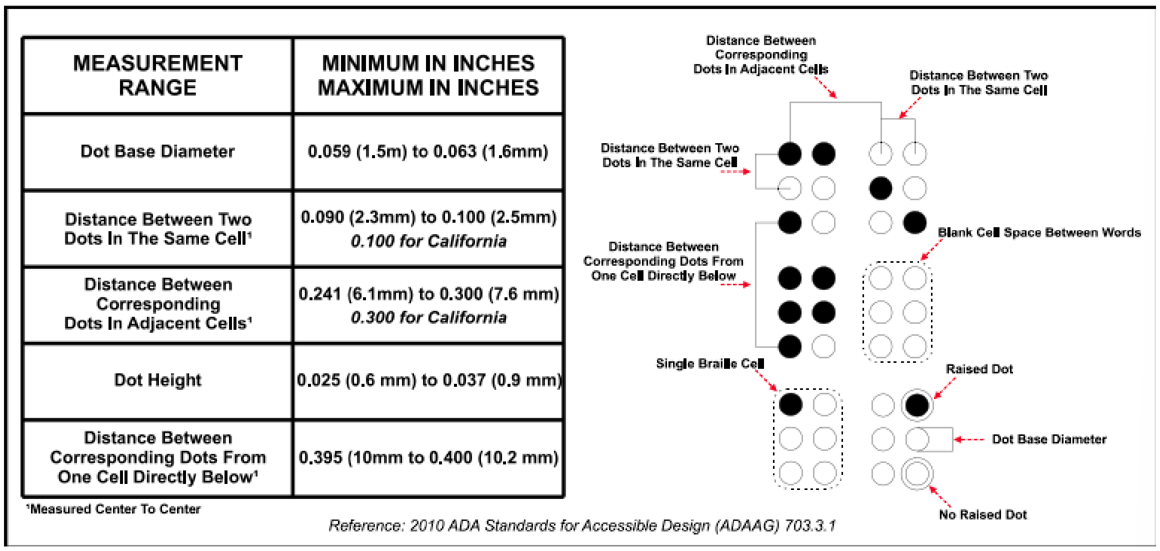
$$((Xvalue \times 3) \times \#characters)$$

The number of characters is for the longest line on the sign. The X value is multiplied by 3 so that it counts two dots and the proceeding space between the current and next braille characters. For the Y dimension, the calculation is:

$$Ceil(\frac{((Yvalue \times 3) \times \#lines)}{2})$$

The Y value is multiplied by 3 to get the height of an individual braille character, then multiplied by the number of total lines in the entire text. These calculations create the perfect amount of space for the text with minimal overhang. The less overhang, the faster they can be printed. The final file would then need to be imported into CURA so it can be converted to .GCODE and printed.

With a solution for creating 3D printed braille signs, it became increasingly important to think about the stakeholders of the project. The next question to answer is: Can someone who reads braille read these signs? In order to answer this question, it is important to determine what makes braille signage readable. The American Disability Association (ADA) has a set of standards for all braille signs [3]. These standards contain sizing ranges that create an achievable goal for the 3D printed signs. According to the literature, the distance between two dots in the same cell of six dots should be between 2.3mm and 2.5mm, the height of a single dot should be between 0.6mm and 0.9mm, and the diameter of a single dot should be 1.5mm to 1.6mm. This information can all be found in figure 9.



MEASUREMENT RANGE	MINIMUM IN INCHES MAXIMUM IN INCHES
Dot Base Diameter	0.059 (1.5m) to 0.063 (1.6mm)
Distance Between Two Dots In The Same Cell¹	0.090 (2.3mm) to 0.100 (2.5mm) *0.100 for California*
Distance Between Corresponding Dots In Adjacent Cells¹	0.241 (6.1mm) to 0.300 (7.6 mm) *0.300 for California*
Dot Height	0.025 (0.6 mm) to 0.037 (0.9 mm)
Distance Between Corresponding Dots From One Cell Directly Below¹	0.395 (10mm to 0.400 (10.2 mm)

¹Measured Center To Center

Distance Between Corresponding Dots In Adjacent Cells
Distance Between Two Dots In The Same Cell
Distance Between Two Dots In The Same Cell
Blank Cell Space Between Words
Distance Between Corresponding Dots From One Cell Directly Below
Single Braille Cell
Raised Dot
Dot Base Diameter
No Raised Dot
Reference: 2010 ADA Standards for Accessible Design (ADAAG) 703.3.1

Figure 9: ADA braille standards [3]

The signs that were being printed were too tall and too wide as seen in figure 10. To ammend this, a .TXT translation was imported into Microsoft Word. By setting the font to the fixed width font consolas, all of the braille matricies became perfectly aligned. The next step was to change the vertical line width. By modifying the setting of spacing to before and after both being 0 pt and the line spacing to be multiple at 0.5, the .txt translation was getting closer to being real braille. For more accurate printing, the font was made bold. This was so that the algorithm that converted the .JPG to .STL had more data to work with. It uses an image analyzer to determine where the darkest parts of the image are and develop a raised location in the 3D model. It also increased the diameter of all the dots. The diameter could also be changed by increasing the font size. This was set to 15pt font. A multi-line print test directly from the Word document can be seen in figure 11. The dots were much more accurate to the actual braille sign.

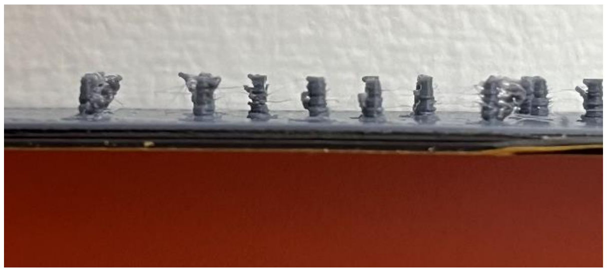


Figure 10: Side view of 3D printed sign before height change

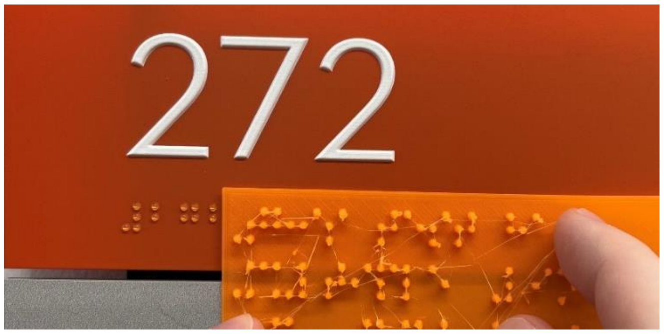


Figure 11: Line Spacing and Font Test

To decrease the height of the dots, the depth value needed to be changed in the .JPG to .STL conversion algorithm. This was changed from 3mm in initial testing to 0.9mm as per the standard. The text conversion algorithm was also updated to include a single character space between rows of text, as well as standardize the font size to 20pt font. The updated version of this algorithm can be seen in figures 13 and 14 showing the measurements using CURA meeting ADA standards. These optimizations and improvements also decreased the printing time of the signs from 50 minutes to 37 minutes due to the decrease in the number of layers that are needed to print to achieve the height of the dots.

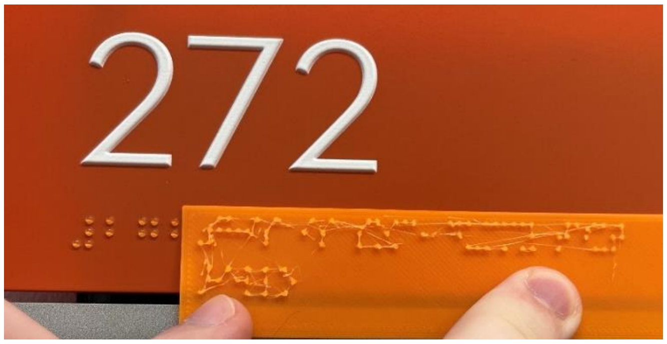
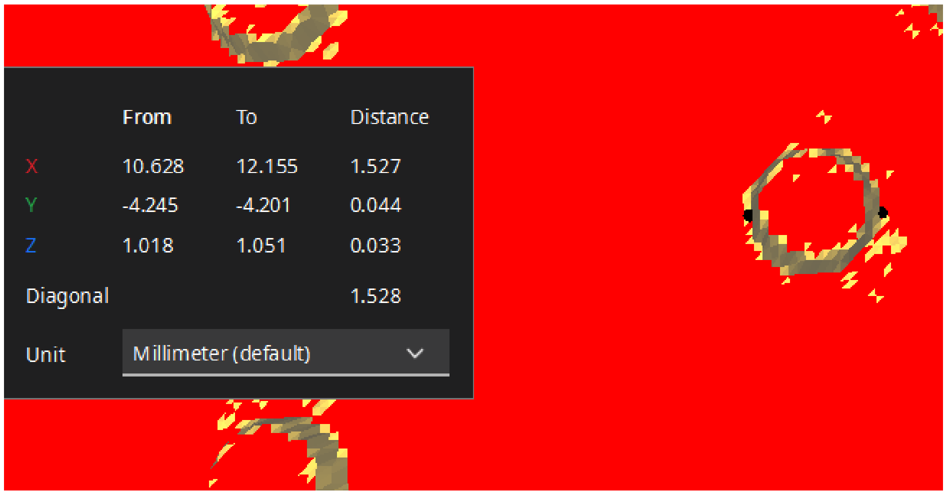


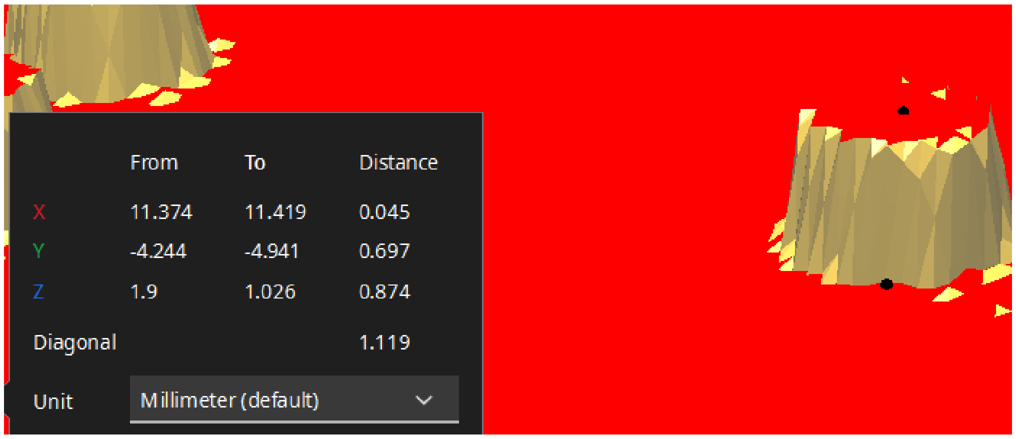
Figure 12: Closer ADA compliant sign



	From	To	Distance
X	10.628	12.155	1.527
Y	-4.245	-4.201	0.044
Z	1.018	1.051	0.033
Diagonal			1.528

Unit Millimeter (default)

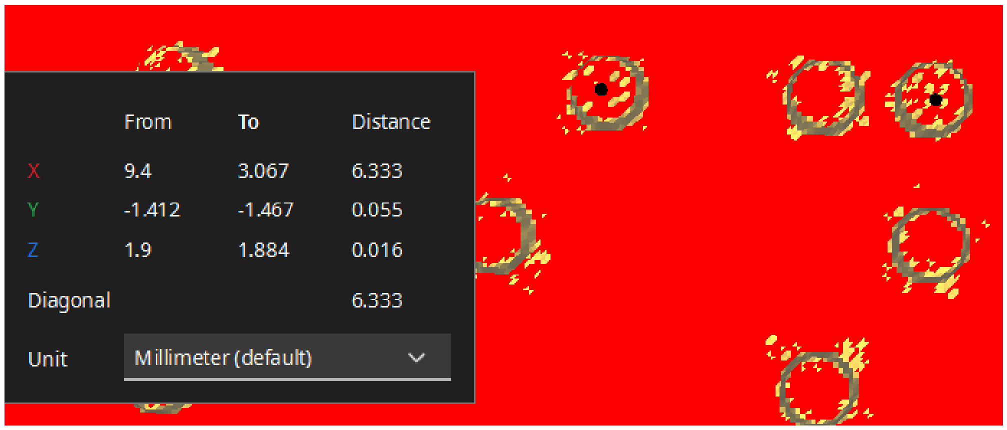
Figure 13: Width calculation using CURA



	From	To	Distance
X	11.374	11.419	0.045
Y	-4.244	-4.941	0.697
Z	1.9	1.026	0.874
Diagonal			1.119

Unit Millimeter (default)

Figure 14: Height calculation using CURA



	From	To	Distance
X	9.4	3.067	6.333
Y	-1.412	-1.467	0.055
Z	1.9	1.884	0.016
Diagonal			6.333

Unit Millimeter (default)

Figure 15: Distance between corresponding dots

Throughout the testing process, a set of recommended settings was developed for the CURA 3D printing software. All printing tests were performed on the Ender 3 printer. Each sign was printed at a 0.12mm layer height at 100% infill. Ironing was enabled, as well as Wipe nozzle between layers and Wipe Z Hop in an attempt to minimize stringing between the dots. This, however, is hard to control due to the chemical properties of PLA+ plastic. Once it starts to receive moisture from the air, it is more likely to produce the stringing as seen in figure 12.

Applications

This technique of creating these braille signs can be applied to many aspects of college campus life, as well as other organizations. In the cafeteria, dishes that are rotated daily can have new signs generated the night before. In grocery stores these signs can be easily created for new products and be moved around. The focus of this project is the creation of professor name plates.

In many universities, professors do not teach in one singular classroom. They move from room to room. While these rooms may have permanent braille signage for the numerical identifier, they cannot have permanent names for the professors. By creating these name plates, students with visual disabilities can read the signs next to the door to make sure that the correct professor is teaching in that classroom. For the display of these signs, magnetic strips with adhesive on both sides would be applied to both the 3D print and the location that the sign will hang. This will make it easier to swap when a professor changes rooms. Figure 15 shows an example of what this would look like.

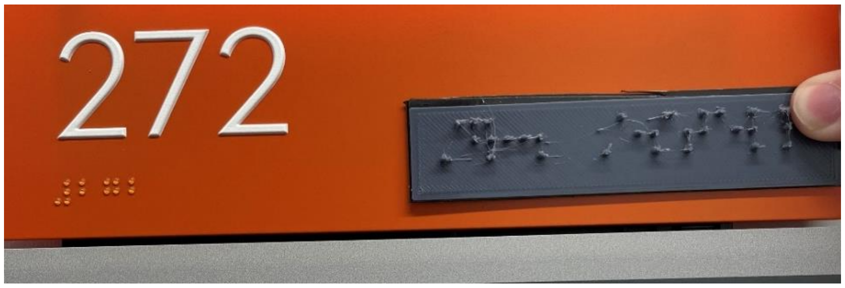


Figure 15: Example sign location

Conclusion

Future progress can be made to this project. The first improvement that can be made is that the conversion from .JPG to .STL adheres closer to the braille standards [3]. Currently, the final braille signs do not meet the standards of the Americans with Disabilities Act in terms of vertical dot spacing. The second improvement is that the text conversion algorithm only covers grade 1 braille. There are two grades of braille, with grade 2 being the most common. The first grade of braille covers just the alphabet and certain numbers and symbols, whereas grade 2 braille is more complex and has dot patterns that can combine letters and substitute for certain words. It is also more common in permanent public signage. This improvement would allow for the more efficient creation of signs, which would both increase the amount of potential text and further decrease the printing time. The current text limit is the dimensions of the 3D printer's bed; however, the algorithm cannot detect these limits. Therefore, further improvements to the algorithm could also include the constraints created by 3D printing technology. Human testing also has not yet been performed on this project. While visual indicators show that it is theoretically possible to read these signs, it has not yet been tested with people who have actual visual disabilities.

Bibliography

[1] J. Telling, "Can You 3D Print ASCII?," 3D Printing Nerd, 12 2 2022. [Online]. Available: https://www.youtube.com/watch?v=9km9I6_XBCY. [Accessed 1 27 2024].

[2] Aspose, "Convert TXT to JPEG in C++," Aspose, 2024. [Online]. [Accessed 27 1 2024].

[3] "ADA Braille Signs," Compliance Signs, 2019. [Online]. Available: https://www.compliancesigns.com/media/resource-bulletins/CRB-ADA-Braille.pdf. [Accessed 27 1 2024].

[4] "Orbit Reader 20," Orbit Research, 2024. [Online]. Available: https://www.orbitresearch.com/product/orbit-reader-20/. [Accessed 1 27 2024].

[5] D. Colbry, "Python Lithophane Generator," 2020. [Online]. Available: https://github.com/colbrydi/Lithophane. [Accessed 20 12 2023].