Automated Tactile Graphics Translation: In the Field

Chandrika Jayant, Matt Renzelmann, Dana Wen,
Satria Krisnandi, Richard Ladner, Dan Comden
University of Washington
Box 352350
Seattle, WA 98195-2350 USA
+1 206 616 1630
cjayant@cs.washington.edu

ABSTRACT

We address the practical problem of automating the process of translating figures from mathematics, science, and engineering textbooks to a tactile form suitable for blind students. The Tactile Graphics Assistant (TGA) and accompanying workflow is described. Components of the TGA that identify text and replace it with Braille use machine learning, computational geometry, and optimization algorithms. We followed through with the ideas in our 2005 paper by creating a more detailed workflow, translating actual images, and analyzing the translation time. Our experience in translating more than 2,300 figures from 4 textbooks demonstrates that figures can be translated in ten minutes or less of human time on average. We describe our experience with training tactile graphics specialists to use the new TGA technology.

Categories and Subject Descriptors

K.4.2 [**Social Issues**]: Assistive technologies for persons with disabilities

General Terms

Human Factors

Keywords

Tactile graphics, Braille, user study, image processing, machine learning, disability, accessibility

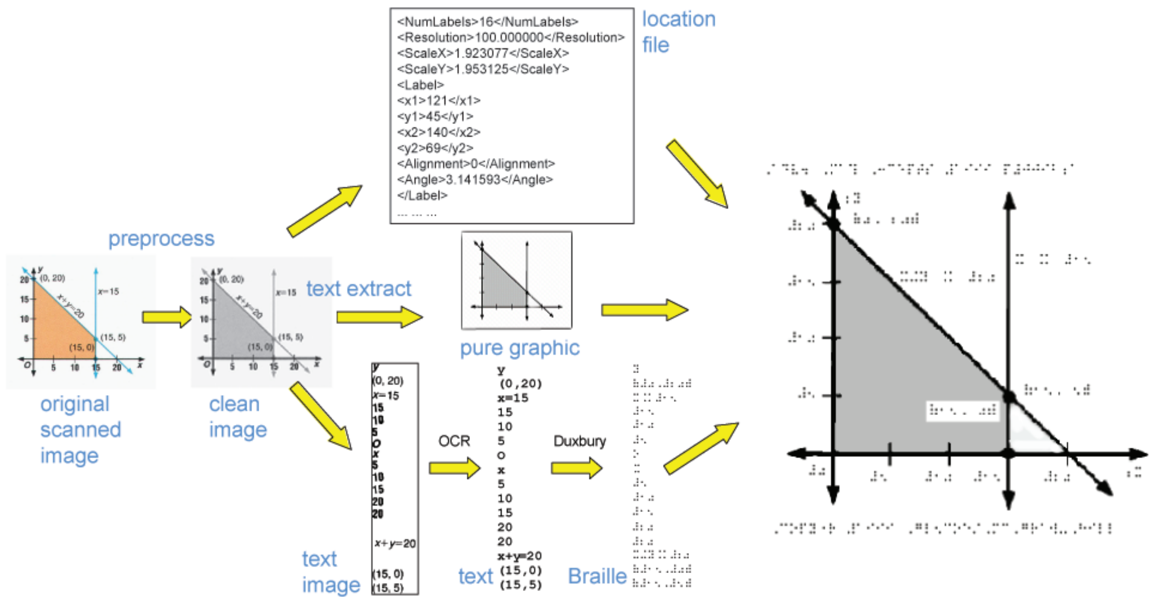
1. INTRODUCTION

A major impediment to the success of blind students in Science, Technology, Engineering, and Mathematics (STEM) fields is access to figures in textbooks. Often the figures in these books are not available in any accessible format. In some cases, important figures will be described orally by the teacher. In other cases, tactile graphics are made of select images from a book whose text is already being translated into Braille, but this is a more expensive option. Traditionally, tactile graphics are thermoform (raised, heated plastic), or swell paper (special paper the expands on darkly written material when heated), or made with crafts tools like string, textured materials, and glue. In the past few years printers, such as the Tiger Embosser, have been developed to print tactile graphics from digital images. The Tactile Graphics Project at the University of Washington has developed techniques to automate the translation of images from math and science books to an embossed tactile format.

There are 37 million blind people worldwide, and 1.4 million of those people are children below the age of 15 (World Health Organization 2004 [1]). It is estimated that in 1998 approximately 93,600 visually impaired or blind students were being served in special education programs in the United States. Most of these blind students would benefit from improved access to figures in textbooks.

In our previous paper [15], we described current work practices in the tactile graphics field. We described the Tactile Graphics Assistant (TGA) that helps automate the process of translating figures to a tactile form. At that time the TGA was implemented and tested in the lab but not in the field. In the past two years, we have translated thousands of figures from 4 books, created a detailed and comprehensive workflow, and trained tactile graphics specialists how to use the TGA and accompanying workflow. The workflow takes advantage of the batch processing capabilities of modern image processing, drawing software, optical character recognition (OCR), automated Braille translation, and the TGA to accelerate the process of translating all the figures in a given textbook in several weeks rather than months of person power. In addition, we have added new TGA features including automated label placement and text recognition using machine learning techniques.

In the current paper we describe work that has occurred in the past two years. After reviewing related work in Section 2, we describe the new and refined Tactile Graphics workflow in Section 3. In Section 4 we describe the new features of the Tactile Graphics Assistant (TGA) including the use of machine learning and optimization to improve label placement. In Section 5 we describe other issues that have come up with the new workflow, such as dealing with math recognition and translation, visual to tactual simplification and alteration, and angled text in images. In Section 6 we describe our experience with the textbooks we have translated. In Section 7 we describe our first training workshop with practitioners. We conclude in Section 8.



```
<NumLabels>16</NumLabels>
<Resolution>100.000000</Resolution>
<ScaleX>1.923077</ScaleX>
<ScaleY>1.953125</ScaleY>
<Label>
<x1>121</x1>
<y1>45</y1>
<x2>140</x2>
<y2>69</y2>
<Alignment>0</Alignment>
<Angle>3.141593</Angle>
</Label>
... ... ...
```

Figure 1: Overview of our Tactile Graphics Production

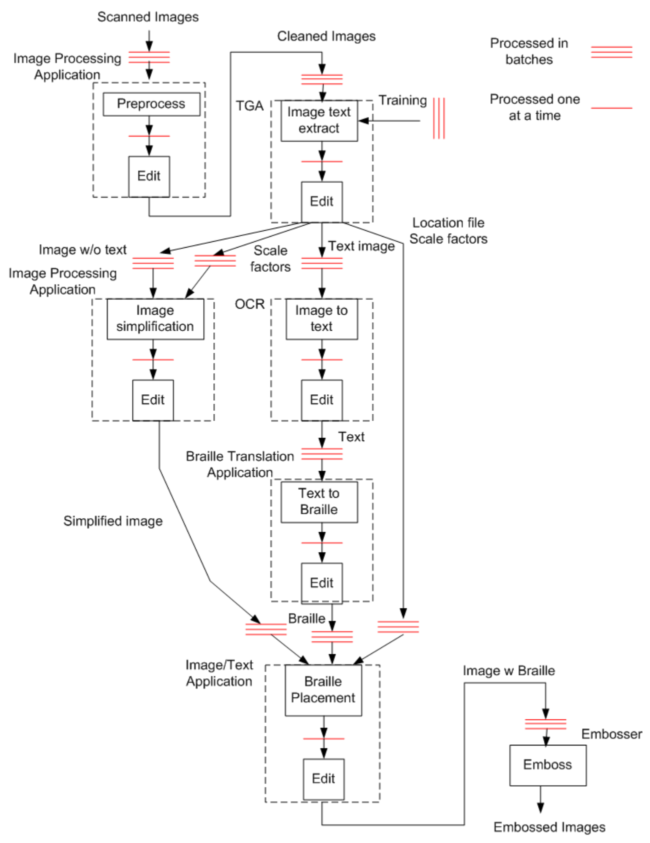


Figure 2: Workflow as of May 2007

2. RELATED WORK

Most work on the automation of tactile graphics has been concerned primarily with image processing, especially with forms of edge detection and image segmentation ([11, 5, 19]). This field, while very important, does not deal with the relationship between the text and graphics within an image. In our work, we are automating image processing combined with text extraction. As far as we know, there has not been much work done on OCR for images combined with text. We will discuss the reason that regular OCR cannot work for our text extraction, in Section 3.4.

In [14], work is done on automating information extraction from vector graphics images. However, many images are not yet in this format, and getting digital files from the publishers is no easy task. Also, there will always be older books to translate, which must be scanned. Some work has been done on examining the workflow of the translation of images to a tactile format, including the G2T Graphics to Tactile Project, which uses a semiautomatic image processing tool in conjunction with existing drawing tools [5]. This system also does not deal with text in images. Automatically produced tactile maps are made using geographical information systems and MATLAB along with available embossing and engraving technologies in [16], an excellent example of automation being used in the tactile graphics field, but only in creation and not translation.

3. TACTILE GRAPHICS WORKFLOW

Figure 1 gives an overview of the tactile graphics process that we have developed, while Figure 2 provides a detailed view of the workflow. In the overview, the original image is scanned, then preprocessed to clean it up. The TGA then finds and removes the text, creating three objects: the image without text, the text as an image, and the location file, an XML file, that describes the original locations of the text found in the image. The text image is then processed with OCR and Braille translation. The final step is to merge the

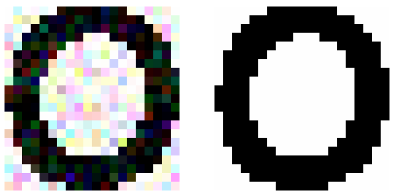


Figure 3: Noise introduced by scanning.

text without an image, the location file, and the Braille into a new document suitable for embossing.

The detailed view of the workflow (Figure 2) has a box for each software package that is used and emphasizes the batch processing to achieve high throughput. Image processing software, with batch processing capability, can process all the scanned images at once. Our TGA, with proper training, can find and remove all the text from images as a batch process. Notice that each box has an editing sub-box where someone has to check and possibly correct the results from the batch processing. These editing steps are where almost all the human time is spent during the workflow.

One question that we are always asked is why we cannot use modern OCR software to find and remove all the text in images. This simple answer is that current OCR software is not up to the task [17]. OCR software may recognize a lot of the text in the image, but it also recognizes many parts of the graphics as text when it is not (false positives). The user must parse through the OCR results to get the correct results out of it, and then must know where to place that text back into the image. This takes an extraordinary amount of time to edit! By employing machine learning, the TGA exploits the consistent text style found in figures in textbooks to accurately find text in images. We found that OCR software does a fine job on the text alone once it is removed from the figure.

In the subsections below we describe in more detail the parts of the detailed workflow. The software we use in our workflow was chosen because of their processing abilities, scripting abilities, and reputation.

3.1 Preprocessing

The first step of the workflow is to obtain and preprocess the image files. Most often these images are not available from the publisher in a digital format, but must be scanned. Scanning introduces noise into the images, even at a high resolution (see Figure 3). Noise is a problem for the TGA because the text finding algorithm uses color as a feature in identifying connected components that may be individual characters. Figure 3 illustrates the problem. Even though the "o" on the left side of Figure 3 looks mostly black, it is actually a vast array of colors that make the "o" a large number of connected components rather than one of a single color. Imaging processing software, such as Adobe Photoshop, has the ability to apply a threshold to change multiple colors into one color. This can be achieved for an entire set of images by using a Photoshop script.

3.2 Image Classification

The detailed workflow, Figure 2 does not show this step, but images in textbooks often fall into a number of classes, such as line graphs, diagrams, and bar charts, that can be processed as separate groups.

Different image classes might require different treatment in terms of preprocessing, training for the TGA, and other special processing that may be needed. After the image files are all obtained from a book, we put them in a few different classes manually, so that all figures in the same class are handled as a batch. An example of a class might be a set of bar charts where the colors in the bars of the the charts are transformed into textures suitable for a blind person. Another example of a class might be a set of figures that use a lot of math that require special processing in the Braille translation step.

3.3 Tactile Graphics Assistant

The next step of the workflow is to automatically extract text from the image using the TGA software. First, the user must manually train on a few representative images so the software can learn what are characters and what are not. Often the training set is only a few images, with the batch being many times larger. For example, in one of our textbooks there was a batch of 600 figures. We trained on fewer than 5 images, taking just a few minutes. Almost all the characters in the remaining figures in the batch were found automatically. A relatively small amount of time was needed to find those characters not identified by the TGA. Details of the TGA are covered in Section 4.

The TGA creates three output files for each image processed (see Figure 1). The original image with the text removed is saved as a bitmap, the extracted text labels are saved as another bitmap, and an XML file, called the location file, is saved with the locations of all the text labels in the original image (so that the Braille will be placed back in the right place in the final step of the workflow).

3.4 OCR and Braille Translation

In the OCR step, the text image produced by the TGA is processed to actually identify the text image as text. Notice in Figure 1 that one of the equations in the figure is at an angle. The TGA rotates this text automatically so that the OCR software can identify it. Standard OCR engines, such as Omnipage or FineReader, are, for the most part, capable of identifying text from text images provided as a batch. An exception is math where the math OCR InftyReader [18] can be used for better results. If InftyReader is not used, the user must make sure to add appropriate math tags to convert the recognized text into the LaTex format, before sending it to the Braille translator for final conversion into Nemeth Braille. Math issues are discussed more in Section 5. Again, the user must manually make corrections to validate the OCR results. Once the OCR results are edited, the files must be saved as text files for later translation to Braille. An annoying detail that happens in this step is to check that the number of lines of OCR text matches the number of lines of text identified by the TGA and stored in the location file. This can be verified in a batch way using a Perl script.

After the text files are generated, they must be translated to Braille using a Braille translator, such as Duxbury or Braille2000. Again, the files can be translated in batch, but must be verified manually by a Braille specialist, and saved as text files. The same Perl script can be used to make sure the number of lines of the Braille still matches the number of lines stored in the location file.

3.5 Resizing and Image Simplification

The original image files that have the text extracted must be resized, usually to 10 × 10 inches, which fits nicely on a standard 11 × 11 inch Braille page. The batch script that does this also modifies the location file to take into account the larger size. The TGA records the scaling factors in the location file so that later placement of the Braille is done to scale. There is also an option in our TGA software to preserve the aspect ratio of the original image, and still fit it onto the 10 × 10 inch page.

These image files might need to be simplified or changed for better tactile perception. For example, lines might need to be thickened, interfering lines removed, legends resized, and colors changed to textures. These changes can be done in image processing software, such as Adobe Photoshop, either manually or with scripts. We discuss this in more detail in Section 5.

3.6 Final Output and Printing

The final step is to take the XML location files, the Braille text files, and the resized (and sometimes simplified) image files that have the text extracted, and combine them into the final product (see Figure 1). This is done with an Adobe Illustrator script as a batch. Manual editing must be done to adjust the location of the Braille labels in the final image. Label placement will be discussed more in later sections. Another part of the step is to add information, such as figure number and page number in the original textbook, into the image. This can also be done as a batch using a script. Once this step is completed each original figure is stored as an Illustrator file ready for printing.

For printing we use the Tiger Embosser from ViewPlus. This embosser can print on the standard 11 × 11 inch paper, and can also print larger sizes up to 16 inches wide and up to 50 inches long. The printouts have 20 dots per inch resolution. Any embosser that can print Illustrator files can be used.

4. TACTILE GRAPHICS ASSISTANT

The TGA software has undergone some major improvements in the past two years, including an implementation of better character recognition after training, consistent justification of Braille labels in the final image, and better Braille label placement.

4.1 Character Training

It was useful to apply a formal machine learning technique to improve character recognition accuracy further and simplify the user's task by eliminating the need to fine tune parameters to the character-finding algorithm in the original TGA [17]. Since the support vector machine (SVM) is well suited to classification problems, it is a natural technique to apply [3]. Support vector machines are a set of related supervised learning methods used for classification and regression. The TGA uses the svmLight package [12].

It is important to note that in this part of the process, we are finding what parts of the original image are characters-not recognizing what each particular character is. That is up to the OCR system later in the workflow. Characters are typically connected components of one color (or set of similar colors) in an image (see Figure 4). In the training phase of the TGA, the user manually selects characters in the image.

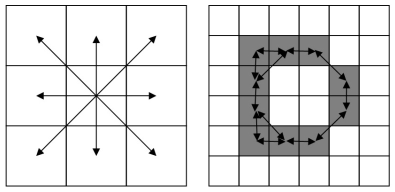


Figure 4: Definition of "connected."

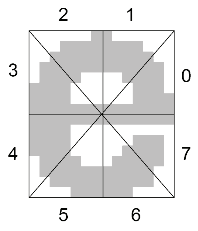


Figure 5: Radial features of a connected component.

Training is both on positive and negative examples; characters that are selected are positive examples, and all other connected components are negative examples of characters. Features used to classify characters are height, width, area, pixel color and what we call radial density. The bounding box of each connected component is subdivided into a number k of pie shaped pieces around by rotating a radial line around the center of mass of the character. See Figure 5 for an example where $k = 8$. The radial density consists of k features, each of which is the percentage of pixels in each pie piece that are part of the character. Thus, there are a total of $k + 4$ features to train for. The choice of $k = 5$, or 9 features, achieves excellent results for the textbooks we translated.

Evaluation of the character-finding algorithm using the features mentioned above is shown in [17]. Three line graph figures from [9] were used for training, with a total database of 475 character connected components. The algorithm was run on 25 figures (line graphs, bar charts, and diagrams), with a total of 16145 connected components, 6895 of which were character connected components. There were 92 false positives and 17 false negatives, yielding a 0.68% character recognition error rate.

4.2 Label Finding

The custom label finding algorithm was discussed in the 2005 paper and has not changed [15]. Training is done in the same way, with the user selecting only positive examples of text labels in this case. By definition a text label is a segment of text that will be recognized as one line by the OCR and subsequently translated to Braille as a single line. The reason this label step must be done is two-fold. First of all, the OCR expects to recognize lines of text, not individual letters. Second, Braille translators expect lines of text, not individual letters because Braille has contractions, that contract sequences of letters or words into smaller number of Braille characters. That is, Braille, or more properly Grade 2 Braille, is compressed for faster reading.

Labels are manually chosen on the training set by the user, and using a minimum spanning tree algorithm connecting the centroids of all characters in the image, the software determines the labels in the rest of the batch. The custom algorithm uses characteristics of labels such as the angle of the line of best fit, the mean squared error of the line of best fit, and the spacing between the adjacent characters [17].

Evaluation of the label-training algorithm is also shown in more detail in [17], using the same 3 figures as in the evaluation for character-finding. These training figures contained 126 character labels. The algorithm was then run on the same 25 figures (line graphs, bar charts, and diagrams). There were 824 character labels in these figures that resulted. There were 8 mis-grouped connected components, 4 false positives, and 1 false negative.

4.3 Text Rotation

The character and label finding algorithms work well with angled text as illustrated in Figure 6(b). Unfortunately, OCR does not work well with angled text so it must be rotated to be horizontal as in Figure 6(c). The angle of the text can be approximated by computing the perpendicular least square fit of the character pixels in the label, that is, the affine line which minimizes the perpendicular squared distance from the line to the pixels [10]. Once this line is found, the angle of rotation can be computed. One difficulty arises when the text is vertical; there may be ambiguity as to whether the text should be rotated left or right to be horizontal. Occasionally, the wrong decision is made, and the text ends up upside down in the image text file.

An example of an image where this capability is needed is shown in Figure 6 (image from [2]).

4.4 Alignment and Justification

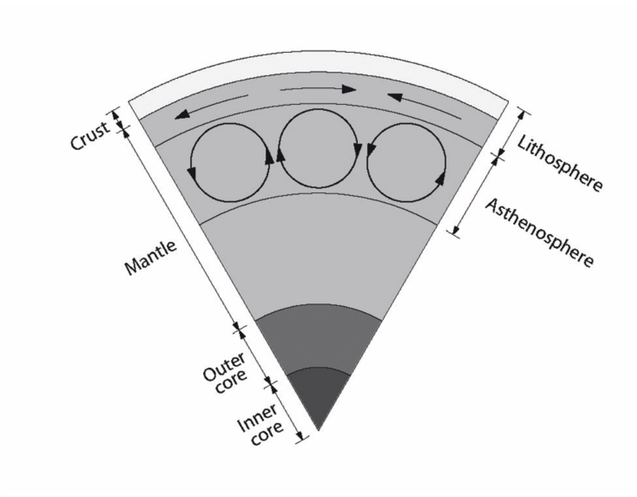
Because the Braille labels are typically a different size and aspect ratio than the original labels there is question about where they should be placed in the output image. A Braille label could be left justified, right justified, or centered relative to the original label. For example, labels on the y-axis of a graph should likely be right justified, while the labels on a legend should be left justified. To determine the justification a variant of the well known plane-sweep algorithm from computational geometry is used [4]. We have not changed this since our 2005 paper.

4.5 Automated Label Placement

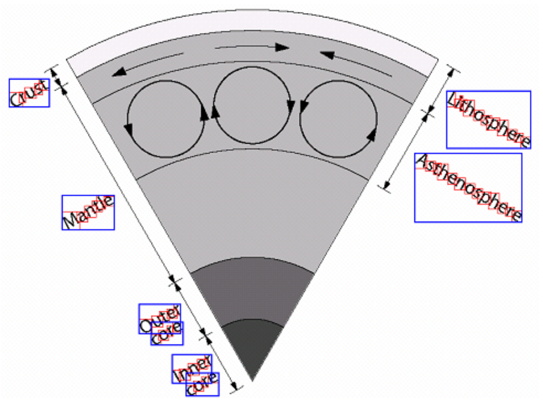
Braille labels will be of different size than the original text labels because Braille characters have a fixed height and width, and because of Braille contractions. In addition, Braille must be horizontal even though the original label may be at an angle. Calculating the exact length of a Braille label is impossible without knowing the results of the OCR and Braille translation. Although we are able to calculate, with a high degree of accuracy, the justification for a label, there is still the problem of making sure that Braille labels do not overlap the pixels of the image and each other.

In order to improve the label placement we have developed a new algorithm to place labels better. In part, the motivation for developing this new algorithm is the data we collected in three translated textbooks ([2, 8, 9]) using only the justification algorithms for placement (see Figure 7). We noted that almost a fourth of the time is spent in adjusting the labels in the final step before printing. Given that this was a major bottleneck, we focused our attention on improving label placement.

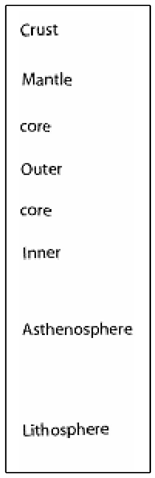
The new label placement algorithm calculates a locally optimal location for each Braille label. Angled labels are



(a) Original Image.



(b) Image with with TGA labels.



Crust

Mantle

core

Outer

core

Inner

Asthenosphere

Lithosphere

(c) Image of text rotated by TGA.

Figure 6: Handling of Angled Text.

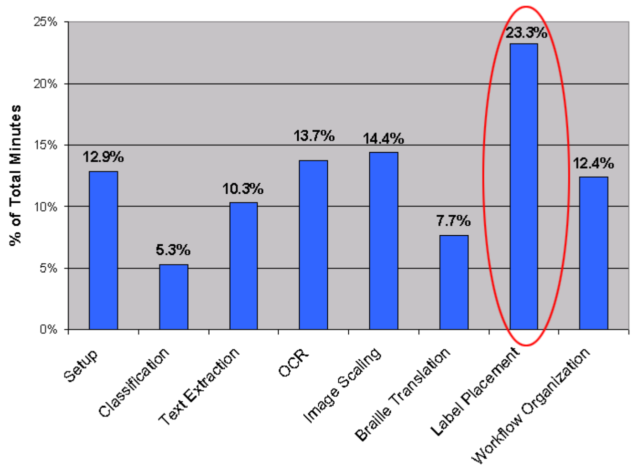


Figure 7: Label Placement as Part of Overall Image Translation Tasks.

rotated to become horizontal. Each label is given a score which indicates how far it is from it original location and how much it overlaps the image and other labels. The goal of the algorithm is to move the label to new locations to minimize the sum of scores for all the labels. Labels are placed in a priority queue based on their scores, and the label with the highest score are moved a small distance to reduce its score. It is then reinserted into the priority queue. The score reductions and label re-insertions are repeated until no improvement can be made. A label's score is calculated as $aL + bP + cD$ where L is the number of pixels overlapping other labels, P is the number of pixels overlapping the image, D is the distance in pixels from of the label from it original position, and a, b and c are tuning constants. In addition, for each label we keep track of the number of times it has been moved. If that number exceeds a specified constant we set the score to zero so that it will not be moved in the future. Labels are moved in "steps" of 5-10 pixels. A label is moved in 8 directions to find a better position. Once final label locations are reached, the new label locations are saved in the location file.

We are still in the process of evaluating the new label placement algorithm and making sure it is compatible with the justification algorithm.

5. OTHER ISSUES

In the best of all worlds one could follow the workflow as described above to achieve results we would like. Unfortunately, there are a number of special cases that arise that have us deviate from the workflow. Although this slowed down the work it also made it more interesting because more problem solving was needed. Many issues can come up, most of which are particular image simplification or clarification problems. Shading in the background may need to be removed, grid lines taken out, parts of the image enlarged. With more experience, more issues will come up, some of which could be handled by image processing scripts in Photoshop. Below we describe two common deviations from the workflow that we experienced, math and legends.

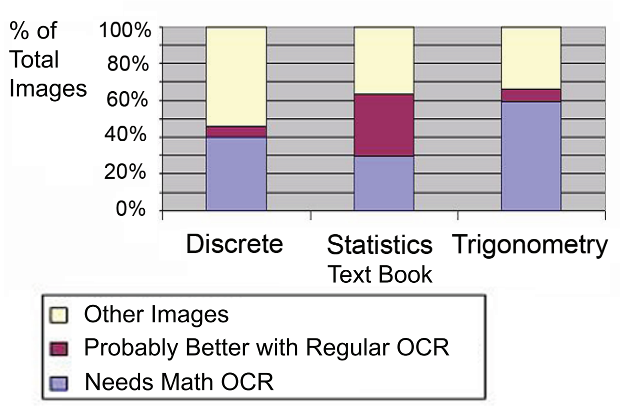


Figure 8: Percentage of Images with Math Text

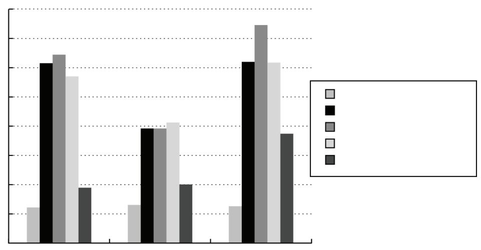
5.1 Math

There is a large amount of math present in many textbook images, particularly those in the STEM fields. Figure 8 shows the percentage of images with math text in 3 books we investigated [13, 6, 7], one of which we had already fully translated. The other 2 books were just used for the math experiments and not fully translated. Regular OCR systems do not perform very well on math recognition. InftyReader [18] is a specific math-OCR system that works well on most images of math. However, with our workflow, an added problem arises because we are recognizing images of text labels, rather than full pages of text out of books or papers. InftyReader always does well on math, but falls short on recognizing regular non-math text properly with these text images from the TGA. In a preliminary study a system that combined the output of math OCR and regular OCR achieved the best results. The output of InftyReader can be either LaTex or MathML. Fortunately, InftyReader has a WYSIWYG math editor so that errors can be corrected without directly editing the LaTex or MathML files.

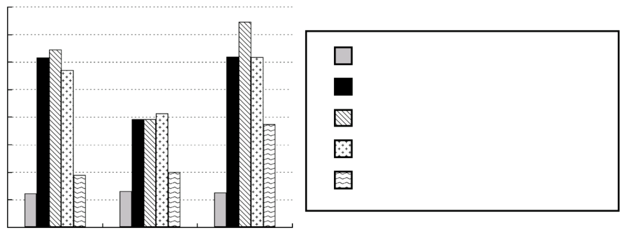
Once the math (as part of the text) is extracted with the TGA, and the math is recognized by an OCR system, it must be translated to Nemeth, a standard for math Braille. Translating from LaTex and MathML to Nemeth can be done using Duxbury or Braille2000, although both are limited in their coverage of the two math markup languages.

5.2 Legends

Apart from preprocessing images to have them usable by the TGA, care must be put into how well the tactual representation of a visual medium will be perceived. One problem that came up had to do with the limits of tactual perception. The legend of a graph had a little square representing a color of a bar graph. Even when the image was enlarged to 10×10 inches, the square was too small for textures to be distinguished. The solution was to make the legend much larger in the first place (see the before and after in Figure 9). Patterns, that can serve as textures, can be created in Photoshop, and colors can be easily changed into these patterns using the color select and fill commands. To make the legend larger we manually cut the legend out of the image, made it bigger in Photoshop, and reinserted the legend into the image. This was done before using the TGA, and it dealt



(a) Original Image



(b) With Textures and Enlarged Legend

Figure 9: Visual to Tactual Issues (With Text and Braille Removed For Clarity)

perfectly well with the different text sizes. This is just one example of making sure the graphic actually makes sense to the blind person as a final product. Enlarging the legend was also an example of a task that had to be accomplished one figure at a time and could not be done as a batch.

6. TRANSLATED BOOKS

We have translated images from 4 textbooks. These include Computer Architecture: A Quantitative Approach [9] at 25 minutes per figure (a book with very complex graphs), Advanced Mathematical Concepts, Precalculus with Applications [8] at 6.3 minutes per figure, An Introduction to Modern Astrophysics [2] at 10.2 minutes per figure, and Discrete Mathematical Structures [13] at 8.8 minutes per figure. The images are available at `www.tactilegraphics.cs.washington.edu/books.html`. The workflow breakdown for 3 of these books is shown in Table 1. The computer architecture book was a trial run, so we did not record detailed workflow breakdown data for this book.

For all 3 books, the Illustrator part of the workflow was one of the top 2 most time-consuming tasks. These results were before the addition of the automated label placement algorithm to the TGA. We hope with further investigation and study into the label placement issue, the time spent in this step can be greatly reduced.

The TGA step does not seem to be a major bottleneck of the workflow at all. If good training images are chosen, there is minimal editing to be done at this text extraction step. Omnipage (OCR editing time) causes a large time-sink in the case of math text in images. The discrete math and astronomy books both had more complicated math text than the precalculus book, and they took more editing time because we used regular OCR software. This human time could be decreased by using InftyReader for math, as is discussed in Section 5, which is why we were looking into finding better ways to distinguish math and text in images.

The setup and classification steps are manual steps that don't easily translate to a batch form. Setup here consisted of scanning the books, and then manually cropping out the images from each scanned page in Photoshop. With digital images, this time is greatly reduced. Classification takes little time and seems worth the time it saves in the TGA step (better classes lead to better training and less editing time). With Duxbury (Braille translation step), the only change that we can make is using a different Braille translator, which might be more or less accurate. Either way, the user must manually verify the results before placing the Braille back into the images.

The workflow step consists of "thinking time." This includes figuring out what scripts to write, and how to classify, preprocess, and simplify images. This step varies greatly from book to book. However, the more books a tactile graphics specialist translates with our method, the more intuitive and less time consuming these issues will be.

All of the hours listed are human hours. Some processes might take longer to run, but the tactile graphics specialist can work on other books or batches while waiting for the processes to finish. These batch processes do not run for long on a computer with the right specifications, on the scale of seconds per image.

7. BRINGING THE PROCESS TO THE FIELD

In April, 2007, we gave a one day training session on the TGA and workflow at the National Braille Association Spring Professional Development Conference in in Colorado Springs. At our Accelerated Production of Tactile Graphics Workshop we had about 60 participants in the morning overview session and 30 participants for the hands-on afternoon session, with 15 stations enabled with the TGA and workflow software. We had participants that were mostly Braille transcribers and tactile graphics specialists. We had a very positive overall reaction to the software, and many people expressed interest in using the software in their real work environments. Some practical concerns came up, such as access to the Tiger Embosser, and getting the correct file formats of images. Since the training workshop there have been at least 20 downloads of the TGA by practitioners. One experienced tactile graphics specialist who was trained on the software in July 2007 will now train others and utilize the software as part of her job.

We will be having another workshop at the 10th Annual Accessing Higher Ground Conference (Accessible Media, Web and Technology Conference for Education, Businesses, and Web and Media Designers) in November 2007 in Boulder, Colorado and get more transcriber feedback.

8. CONCLUSION

Since our previous paper [15], we have improved the TGA's features and refined the overall Tactile Graphics workflow. In the TGA, we added the capabilities of automated label placement, used machine learning to recognize text characters in images, and dealt with angled text. We hope to

	Discrete Math			Precalculus			Astronomy	
	Minutes			Minutes			Minutes	
SetUp	425	10.30%		660	9.80%		1110	18.30%
Classification	245	5.90%		390	5.80%		270	4.40%
TGA	595	14.40%		570	8.40%		585	9.60%
Omnipage	714	17.30%		660	9.80%		945	15.60%
Photoshop	800	19.40%		975	14.40%		660	10.90%
Duxbury	225	5.50%		630	9.30%		450	7.40%
Illustrator	770	18.70%		1335	19.70%		1845	30.40%
Workflow	350	8.50%		1545	22.80%		210	3.50%
Total	4124	100.00%		6765	100.00%		6075	100.00%
	num figs	467		num figs	1080		num figs	598
	min/fig	8.8		min/fig	6.3		min/fig	10.2

Table 1: Workflow Breakdown for 3 Books

receive continued valuable feedback on our system as more tactile graphics specialists are trained.

A detailed workflow and extensive training manuals have been created, which include information on software and hardware requirements to use our system. These along with the TGA software are available for download at `http://tactilegraphics.cs.washington.edu`.

9. ACKNOWLEDGEMENTS

This research was funded by the National Science Foundation Grant No. IIS-0415273, the Washington Research Foundation, and the Boeing Professorship. Thanks to Adobe Systems Corporation for a software license grant and to ViewPlus Technologies Inc. for technical assistance. Thanks to Sangyun Hahn, Zach Lattin, Stuart Olsen, Carson Smith, and Cian Malone for feedback.

10. REFERENCES

[1] American foundation for the blind. `http://www.afb.org`.

[2] B. Carroll and D. Ostlie. *An Introduction to Modern Astrophysics.* Addison-Wesley, 1996.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: a library for support vector machines, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, 2001.

[5] D. Crombie, R. Lenoir, N. McKenzie, and G. Ioannidis. The bigger picture: Automated production tools for tactile graphics. In *Proceedings of ICCHP*, pages 713–720, 2004.

[6] J. Freund. *Statistics: A First Course.* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[7] I. M. Gelfand and M. Saul. *Trigonometry.* Birkhuser, Boston, 2001.

[8] B. Gordon-Holliday, L. Yunker, G. Vannatta, and F. Crosswhite. *Advanced Mathematical Concepts, Precalculus with Applications.* Glencoe/McGraw-Hill, 1999.

[9] J. Hennessy and D. Paterson. *Computer Architecture, A Quantitative Approach. 3rd Edition.* Morgan Kaufmann Publishers, San Francisco, 2003.

[10] J. Hollmen. Principal component analysis, 1996. `http://www.cis.hut.fi/~jhollmen/dippa/node30.html`.

[11] M. Horstmann, M. Lorenz, A. Watkowski, G. Ioannidis, O. Herzog, A. King, D. Evans, C. Hagen, C. Schlieder, A. Burn, N. King, H. Petrie, S. Dijkstra, and D. Crombie. Automated interpretation and accessible presentation of technical diagrams for blind people, 2004.

[12] T. Joachims. *Making large-Scale SVM Learning Practical.* MIT-Press, 1999.

[13] B. Kolman, R. Busby, and S. Ross. *Discrete Mathematical Structures.* Prentice Hall, 2003.

[14] S. Krufka and K. Barner. Automatic production of tactile graphics from scalable vector graphics. In *Proceedings of The Seventh International ACM SIGACCESS Conference on Computers and Accessibility, (Baltimore, MD, Oct 09 - Oct 12, 2005)*, pages 166 – 172, 2005.

[15] R. Ladner, M. Ivory, R. Rao, S. Burgstahler, D. Comden, S. Hahn, M. Renzelmann, S. Krisnandi, M. Ramasamy, B. Slabosky, A. Martin, A. Lacenski, S. Olsen, and D. Croce. Automating tactile graphics translation. In *Proceedings of The Seventh International ACM SIGACCESS Conference on Computers and Accessibility, (Baltimore, MD, Oct 09 - Oct 12, 2005)*, pages 50–57, 2005.

[16] J. Miele. Tactile map automated production (tmap): Using gis data to generate braille maps, 2004.

[17] M. Renzelmann. Text segmentation and grouping for tactile graphics, 2005. `http://www.cs.washington.edu/education/ugrad/current/bestseniortheses/R%enzelmann.pdf`.

[18] M. Suzuki, F. Tamari, R. Fukuda, S. Uchida, and T. Kanahori. Infty: an integrated ocr system for mathematical documents. In *DocEng '03: Proceedings of the 2003 ACM symposium on Document engineering*, pages 95–104, New York, NY, USA, 2003. ACM Press.

[19] T. Way and K. Barner. Automatic visual to tactile translation, part ii: Evaluation of the tactile image creation system. In *IEEE Transactions on Rehabilitation Engineering*, pages 95–105, 1997.