Tracking @stemxcomet: Teaching Programming to Blind Students via 3D Printing, Crisis Management, and Twitter

Shaun K. Kane
Department of Information Systems
UMBC
Baltimore, MD 21201
skane@umbc.edu

Jeffrey P. Bigham
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15217
jbigham@cs.cmu.edu

ABSTRACT
Introductory programming activities for students often include graphical user interfaces or other visual media that are inaccessible to students with visual impairments. Digital fabrication techniques such as 3D printing offer an opportunity for students to write programs that produce tactile objects, providing an accessible way of exploring program output. This paper describes the planning and execution of a four-day computer science education workshop in which blind and visually impaired students wrote Ruby programs to analyze data from Twitter regarding a fictional ecological crisis. Students then wrote code to produce accessible tactile visualizations of that data. This paper describes outcomes from our workshop and suggests future directions for integrating data analysis and 3D printing into programming instruction for blind students.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education, curriculum.*

Keywords
Education, programming, accessibility, visual impairments, crisis informatics, 3D printing, fabrication.

1. INTRODUCTION
Advances in computer accessibility over the past few decades have resulted in many computing devices and applications that are accessible to blind and visually impaired individuals. However, computer programming tools still present significant accessibility challenges to blind people, and blind people are currently underrepresented in computer science [8]. This effect is likely due, at least in part, to a lack of compelling accessible instructional materials and tools for learning how to program [7].

Developing a supportive environment in which blind students can learn to program presents several challenges. First, the programming tools must be accessible to the student and must work with the assistive technology that he or she uses, *e.g.,* a screen reader, screen magnifier, or refreshable Braille display. Second, the student must be provided with programming tasks that hold their interest and provide encouraging feedback. Many common programming exercises that have been used to entice younger programmers, such as GUI programming or creating graphical games, are inaccessible to blind students. Thus, finding alternative introductory programming activities that are both interesting and accessible to blind students could encourage more blind students to learn to program.

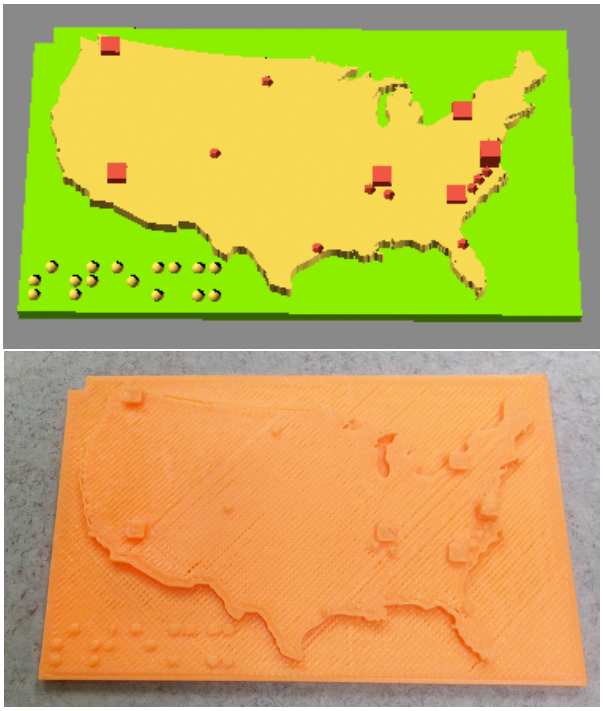

Figure 1. Blind high school students in our programming workshop analyzed Twitter data and visualized them using 3D printing. Top: 3D model generated from Twitter data, using our support libraries. Bottom: 3D-printed tactile graphic.

While prior attempts to improve programming accessibility for blind students have often focused on providing improved audio feedback (*e.g.,* [14,16]), there has been relatively little exploration of how tactile feedback could help blind students understand their programs or make programming more compelling. The increasing availability of digital fabrication or "making" technologies, such as 3D printers, presents an opportunity to connect student programming to the creation of physical, tactile objects that could be accessible to blind and visually impaired students.

In this paper, we describe the preparation and implementation of a four-day computer science workshop in which blind high school students explored Ruby programming by accessing social

SIGCSE '14, March 5-8 2014, Atlanta, GA, USA

ACM 978-1-4503-2605-6/14/03$15.00.
http://dx.doi.org/10.1145/2538862.2538975

networking data via the Twitter API, manually explored that data via their code, and visualized that data via an interactive iPad application and 3D-printed tactile models (Figure 1). We then report the results of this activity and identify additional opportunities for integrating data analysis and 3D printing into accessible computer science programs.

2. RELATED WORK

A common problem with mainstream programming tools and integrated development environments (IDEs) is that they rely on visual information to convey structure and provide helpful hints [2]. This problem is not new, and researchers have explored various techniques for making programming tools more accessible to blind and visually impaired users.

One approach has been to create new programming languages that are easier to read and write using a screen reader. APL [12] is an audio programming language designed by and for blind programmers. Quorum (formerly Hop) is a programming language that was designed to be easy for both blind and sighted novice programmers [16]. Custom programming languages can help to overcome some of the accessibility challenges present in mainstream programming languages, but also have limitations: knowledge of a specialized programming language might not transfer to mainstream programming tasks, and specialized programming languages may restrict the programmer to specific functionality, libraries, or development tools. Other projects have produced accessible variations of existing programming tools, such as Visual Basic [14] and LEGO Mindstorms [5].

Researchers have also explored how mainstream languages and IDEs can be made more accessible. Emacspeak [10] is a screen reader for the Emacs text editor that was designed to support several text editing tasks, including programming tasks. Smith *et al.* [15] developed a plugin for the Eclipse IDE that provided sound feedback when blind programmers traversed code hierarchies. PLUMB EXTRA^3 [4] is a supplementary educational tool that allows blind programming students to explore data structures spatially using a talking graph exploration tool.

Finally, some researchers have explored curricula for introducing existing programming tools to blind students. Ludi and Reichlmayr [6] have used introductory robotics kits, such as Mindstorms, to teach computer science concepts to blind and visually impaired students in grades 7–12. Bigham *et al.* [2] conducted an introductory programming workshop in which blind high school students created instant messenger chat bots in C#. Our current approach is similar to this prior work, in that we have focused on teaching blind students using existing programming technologies (Ruby, the Twitter API, and OpenSCAD [9]), but provide supporting libraries to enable students to create working programs faster.

3. PROGRAMMING WORKSHOP

Over the past year, our research group has explored the use of 3D printers to create accessible technology such as tactile graphics [3]. As part of this ongoing work, we have explored how 3D design and printing can be used to help teach basic programming concepts to blind students. This section describes our preparation and implementation of a four-day introductory programming workshop for blind high school students.

3.1 Workshop Setup

The workshop took place over four days in July and August 2013, as part of the computer science track of the National Federation of the Blind's STEM-X program. STEM-X (formerly known as the NFB Youth Slam) is a week-long summer science camp for blind and visually impaired youth [7]. In 2013, STEM-X had fifty attendees from across the United States. STEM-X students chose one disciplinary track (computer science, chemistry, engineering, robotics, or aerospace science) to explore during the week. For four days, students spent half the day working with instructors in their track, and half the day participating in science enrichment and social activities. On the final day, students reconvened to show off their work to their peers.

Students in all subject tracks worked together to solve a shared problem: the (fictional) impact of Comet ISON. Students in the aerospace track built a working hovercraft, students in the chemistry track studied the science behind desalinization plants, and students in the robotics track built robots to help find people in trouble. Students in the computer science track monitored comet sightings using social media and wrote programs to produce tactile visualizations of the predicted impact zone. At the end of each day, students came together to share status updates about the impending comet impact, and to discuss next steps.

The computer science track featured 9 students (3 female) from grades 8 to 12. Students varied greatly in their prior computer and programming experience, and also varied in their visual abilities and preferred assistive technologies. The computer science track had 6 instructors: 2 faculty members, 2 graduate students, and 2 undergraduate students. The instructors were also assisted by 2 mentors, who helped students with any general issues that came up, including basic computer or screen reader problems. The high instructor-to-student ratio ensured that students did not have to wait for help for very long if they became stuck, which was especially beneficial as most students were programming novices. However, students were able to make independent progress even when instructors were unavailable, suggesting that this workshop could work well with fewer instructors.

3.2 Programming Tools

Choosing an environment for introducing students to computer programming can be challenging, especially for blind students, who might have difficulties working with standard programming tools. Thus, much of our preparation work focused on choosing appropriate programming tools and developing supporting library code for students to work with. Our primary goals in choosing programming tools were to create an environment that would be easy to start programming in, to use real programming languages if possible, and to enable students to try out several programming concepts over the course of the workshop.

We chose Ruby as the programming language for this workshop, as it had a number of advantages over competing languages. First, Ruby is a mainstream programming language that is available on many systems, and which offers many standard libraries. Second, Ruby syntax is comprised largely of text symbols (*e.g.,* "if", "then", "end"), and contains relatively few non-alphanumeric symbols that may be difficult to recognize via a screen reader. Third, Ruby provides an interactive interpreter, *irb*, that enables students to learn using a "read-eval-print loop" [13]. We considered using the Python programming language, which also satisfies these criteria, but Python uses whitespace to delimit code blocks, which could be confusing to navigate with a screen reader.

Since students only spent about 16 hours total in the workshop, they would be unable to make sophisticated programs completely on their own. To enable these novice students to interact with compelling applications, we created several library functions that

students could call to interact with more advanced features. These functions were grouped into four major categories:

- **Twitter:** a wrapper for the Twitter API that enabled students to log in, search tweets, and post tweets;
- **Geocoding:** functions for reverse-geocoding tweets that contained location data;
- **Data visualization:** functions that added geocoded tweets to an accessible map visualization, which could be viewed interactively on an iOS-based device using VoiceOver;
- **3D printing:** functions that assembled geocoded tweets into a 3D model of a tactile map, which could be 3D printed.

Figure 2 shows example Ruby code that illustrates some of the functions made available in the support libraries.

```
require 'twit' # load an external library
startup() # load twitter credentials
point = getCoordinatesForTweet('#stemxcomet',0)
loc = reverseGeocode(point) # get location of tweet
tweets = searchByKeyword('#barelyseeit') # search
tweets.each { |tweet| puts tweet } # looping
interactiveMap('#onthehorizon') # generate map
print3D('#stemxcomet) # generate 3d model
post('Programming is fun!') # post to twitter
```

Figure 2. Sample code (written by the authors) using the APIs provided to students. By the end of the workshop, students had experience using the Twitter API, working with variables and functions, looping, and generating visualizations.

Students used Apple MacBook Pro laptops with the built-in VoiceOver screen reader [1] to read and write code during the workshop. Students wrote code using Apple's TextEdit text editor, and using the *irb* interpreter prompt running in Apple's Terminal program. The majority of students used screen reader software during the workshop, while some students used screen magnification software or refreshable Braille displays (Figure 3).

Figure 3. Students used a variety of assistive technologies to write code. This student used both a screen reader and a refreshable Braille display. Photo credit: National Federation of the Blind. Used with permission.

3.3 Dataset

Students participating in each of the STEM-X tracks worked together to solve a shared problem: tracking the trajectory of the Comet ISON, predicting its impact spot, and directing support resources where necessary. Students in the computer science track were tasked with monitoring sightings of Comet ISON via posts on Twitter and determining the comet's likely point of impact.

We chose this task because social networks are often used to monitor crisis events in the real world, and because tracking mentions of Comet ISON on Twitter would require students to gain understanding of Ruby, the Twitter API, and visualizing data. However, since Comet ISON was not actually at risk of striking Earth, there were no posts on Twitter discussing it. Thus, we recruited volunteers across the United States to post geotagged tweets describing their view of the comet. We created a custom web application that enabled volunteers to log in using their Twitter account, and to post a pre-written tweet describing their comet sighting. These pre-written tweets required little effort from volunteers, and used the volunteer's own geographic location and Twitter handle, thus creating a geographically diverse data set. Tweets generated by the application were addressed to the Twitter user @stemxcomet (to prevent followers unaware of the project from seeing them), and featured one of several hashtags determined by the volunteer's location: tweets near the expected impact site were tagged #rightonme, somewhat distant tweets were tagged #onthehorizon, tweets from further away were tagged #barelyseeit, and tweets from the furthest location were tagged #notvisible. In total, volunteers generated 19 geotagged tweets from locations over the United States, and from several international locations (Figure 4).

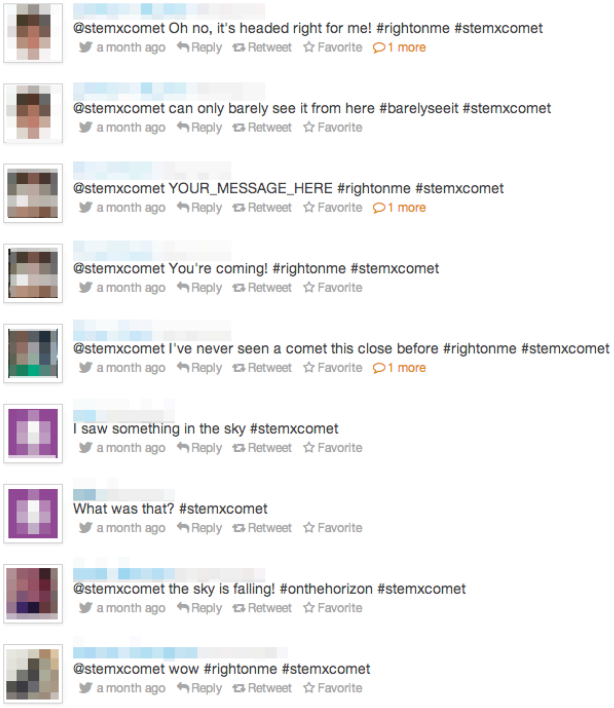

Figure 4. Twitter users posted sightings of Comet ISON as it approached the Earth in the fictional STEM-X narrative. Tweets posted close to the expected impact site were tagged differently than tweets posted from further away, enabling students to predict the impact site from Twitter data.

3.4 Workshop Curriculum

The programming workshop took place over four days, and met for four hours each day during this time. The majority of workshop time was spent working on programming activities. Each day also featured a guest speaker, who called in to discuss

their work with the group. Guest speakers included a professor and crisis informatics researcher, a blind software engineer, and a blind computer science graduate student.

Because students began with different levels of background knowledge, we expected that they would proceed through the workshop activities at their own pace. Thus, students generally worked individually on the activities, and sought guidance from instructors or their peers if they became stuck. All workshop activities were written out as a step-by-step tutorial and posted on a single web page. The instructors set an approximate schedule for the workshop, as described below, and worked with students to help them complete each day's tasks. The major curriculum tasks are described below.

3.4.1 Introduction to Mac, VoiceOver, and Ruby

On the first day, students were introduced to the Macintosh computers and VoiceOver screen reader, opening the terminal, and running terminal commands. Students also created user accounts on Twitter and posted their first tweets from those accounts.

Once students became comfortable navigating with VoiceOver, they were directed to the tutorial web site, and began writing some basic Ruby code using the *irb* interpreter, including evaluating arithmetic operations and printing strings.

3.4.2 Interacting with the Twitter API

Once they had become comfortable using the Ruby interpreter, students downloaded the support libraries written by the instructors and loaded them as a module in Ruby. Because the Twitter API requires each user to be authenticated, students were required to locate their account's OAuth tokens and add them to the library file. Then, students used the libraries to log into Twitter and post a tweet via Ruby, and to begin to search tweets by hashtags. Students performed search queries on various Twitter hashtags using the *searchByKeyword* function, and inspected the returned post objects using *irb*.

3.4.3 Geocoding

Students were introduced to the @stemxcomet tweets generated by our volunteers. Students first used the *searchByKeyword* function to manually explore each of the hashtags from the generated dataset. Students were then introduced to the concept of geotagging and GPS data, and used the provided *getCoordinatesForTweet* and *reverseGeocode* functions to retrieve location data from geotagged tweets. Students were asked to explore the various tagged tweets to see if they could figure out where the comet might strike.

3.4.4 Exploring and Visualizing Data

Following the geocoding exercise, students were introduced to functions provided by the support libraries for visualizing aggregate tweet data. Two forms of visualization were provided. First, students could use the *interactiveMap* function and pass a list of tweets to a web service, designed by the authors, that plotted the tweet coordinates on a United States map (Figure 5). The function returned a URL to the generated map, which could be loaded in an iOS web browser and explored using VoiceOver: touching anywhere on the screen would speak out the number of tweets in that region of the map, while performing directional swipe gestures would read through each of the geotagged regions.

Figure 5. Tweets tagged #stemxcomet were generated and given location data across the United States. Geotagged tweets could be explored on a map using a screen reader.

Students could also call the *print3D* function from their Ruby code. This function passed tweet data to a second web service, which used OpenSCAD to generate a 3D tactile graphic based on the map. The tactile graphic is a credit-card-sized (85mm × 54mm × 5mm) version of a United States map. Geotagged tweets were shown on the map as bars: higher bars indicated more geotagged tweets in that region (Figure 1). During the workshop, the *print3D* function did not directly print the tactile graphic, but instead saved a printable stereolithography (*.stl*) file. Because each tactile graphic required approximately one hour to print, they were printed overnight and returned the next day. Tactile graphics were printed on a MakerBot Replicator 3D printer using ABS plastic.

3.4.5 3D Printing Demonstration

On the final day, the instructors brought one of the 3D printers to the workshop, and introduced it to the students. The instructors provided a verbal overview of the printer and its functions, and then began a demonstration print. While printing, students were able to examine the printer and to touch its non-moving parts. The instructors answered students' many questions about the printer, including how it worked and what could or could not be printed. Finally, each student was given a copy of the US map tactile graphic to take home. The tactile graphic was marked with the tweet locations from the volunteer dataset, the location of each student's hometown, and the word "STEMX" in 3D-printed Braille.

4. INSIGHTS FROM THE WORKSHOP

Overall, we considered the workshop to be successful. While not every student completed every activity, every student spent several days developing his or her programming skills. Spending four days teaching high school students to program in Ruby also provided valuable insights about the suitability of Ruby for blind programmers, about teaching beginning programmers to explore and visualize data, and about keeping students engaged through an intensive programming course.

4.1 Using Ruby for Blind Programming

Generally, students in our workshop were successful at writing Ruby programs, using the terminal to launch Ruby programs, and using *irb* to test code. There were, however, several usability issues relating to the interaction between Ruby and screen readers that created minor challenges.

Keeping track of scope: As found in prior explorations of blind programming (*e.g.,* [2]), our novice programmers sometimes had difficulty keeping track of their current program scope, and entered code in invalid locations. Ruby uses the keyword "end" to denote the end of a code block, such as a function, conditional statement, or loop. However, it is sometimes difficult to determine which block the "end" keyword is referring to without visual feedback. We thus encouraged students to add commends indicating the ends of blocks, and included such comments in our own example code, as shown in Figure 6.

```
def reverseGeocode(point)
   loc = Geocoder.search("#{point[0]},#{point[1]}")
   city = loc[0].city
   state = loc[0].state
   country = loc[0].country
   return "#{city},#{state},#{country}"
end
# end of function
```

Figure 6. In our sample code, we added comments to the end of function blocks to help students keep track of scope.

Conflicts between screen reader and console commands: The VoiceOver screen reader uses a combination of modifier keys (*Control* and *Option)* as a prefix for most commands. For example, *Control-Option-H* opens the VoiceOver help menu. However, VoiceOver also overrides the behavior of some keys, such as the arrow keys, which caused confusion when students attempted to use the arrow keys to browse the terminal or navigate through text files. VoiceOver provides alternative modes for accessing overridden keys, but students were not always familiar with these modes, and sometimes forgot to activate them.

Pronunciation: Ruby uses identifiable English words for many of its commands, and uses few unpronounceable symbols. However, some Ruby terms were consistently mispronounced. For example, the Ruby interpreter program, *irb,* is pronounced as "urb" by VoiceOver, and was frequently misspelled by students when using the terminal. Knowledgeable students knew to switch VoiceOver to read character-by-character, and could correctly copy the word, but not all VoiceOver users may know how to handle this type of error. Ruby also pronounced some characters in ways that could be confusing to a novice programmer. For example, the character "-" may be read and pronounced differently depending on its context: it may be "minus" in an arithmetic expression, "dash" when used as a command line parameter, and "negative" when used as a unary operator. By default, VoiceOver does not consider these distinctions, and universally refers to the "-" character as "minus", which could be confusing.

Typically, all of these problems were identified when the programming exercise was tested using a screen reader, but since the instructors were not everyday screen reader users, they sometimes missed these errors. Testing all programs and written content using a screen reader would help to identify and eliminate these types of errors before students encounter them. Fortunately, the specific problems mentioned above were all solved during the workshop, either by altering the written instructions or by advising students on how to change their screen reader settings.

4.2 3D Printing in Introductory CS

Students were clearly excited by the use of the 3D printer during the workshop. When the 3D printer was demonstrated, students paid careful attention and asked questions. Students were eager to touch the printer and observe its mechanics. We printed tactile graphics for each student, and most students were excited to take the tactile graphics home as a souvenir. When students had the opportunity to test both interactive touch screen graphics (presented on the iPad) and tactile graphics, students clearly seemed more interested in the tactile graphics. It is unclear whether students preferred the tactile graphics because they were unfamiliar, because they were accessible, or for some other reason, but even some students who were less enthusiastic about their programming activities were intrigued by the 3D printer hardware and its output.

We also found that students who did not read Braille were eager to explore the Braille printed on the tactile graphics. As Braille literacy has declined in recent decades, and because Braille literacy seems closely related to employment [11], using 3D-printed objects to motivate Braille learning presents an exciting opportunity for future work.

In general, it seemed clear that the 3D printer was a valuable addition to the computer programming workshop curriculum. However, there were some challenges in using the 3D printer in the classroom. First, the printer is quite slow: the tactile graphics, which were approximately the size of a credit card (85mm × 54mm × 5mm) each took approximately one hour to print on the MakerBot Replicator printer. Faster printing settings are available, but result in a more brittle object. As a result of the slow print time, we were not able to print each student's tactile graphics in class, but instead collected data at the second-to-last meeting, printed the tactile graphics overnight, and delivered them at the final meeting. Furthermore, even with the default settings used, the printed tactile graphics could be quite brittle. Some parts of the tactile graphics would easily wear or break off, including fine details such as the 3D-printed Braille. While the prints made with the current printer were usable, there remains room for improving the quality and durability of the 3D-printed tactile graphics, especially since tactile graphics are likely to be handled frequently.

4.3 Screen Readers as Musical Instruments

One of the most intriguing outcomes from this workshop was completely unplanned, and was instead the result of creative procrastination from the students. Throughout the week, several of the students took breaks from programming and decided to have fun with the VoiceOver screen reader. These students began adjusting the settings of VoiceOver's speech output (speech rate, voice, *etc.*) to make interesting noise, *e.g.,* clicks, beeps, and speech sounds. Over the course of the week, these sounds became more musical as students took longer breaks and invested more time in their music production.

Given the students' interest in experimenting with the screen reader, we set aside time on the second-to-last day to allow students to "perform" using their instruments. Nearly all of the students were engaged in this performance, and those who did not make music with their screen readers sometimes drummed on the table to contribute. The final performance was recorded, and was shared with students from the other STEM-X tracks on the final meeting day. In general, students seemed quite excited by their performance, and were eager to share the recording with their peers.

5. OPPORTUNITIES FOR FUTURE WORK

Our experiences in preparing and conducting this workshop suggest numerous possibilities for future work. First, the libraries that we created to support workshop participants could be extended into reusable tools to support both blind and sighted

programmers. Although tools exist for programmatically generating tactile graphics (*e.g.,* VizTouch [3] and OpenSCAD [9]), these tools are not tightly integrated with standard programming languages. Universal tools for creating tactile visualizations of program output could benefit both blind and sighted programmers.

Second, although the progress made during the workshop was comparable to similar workshops with students at this ability level (*e.g.,* [2]), this workshop covered only a small subset of basic programming topics. Our approach could be extended to cover computing topics in more depth, using the provided libraries as scaffolding that is replaced with the student's own code over time. This approach may even be used as the foundation of a universally designed introductory programming course.

Third, while we did not conduct a formal evaluation of Ruby for blind programming, our informal testing during this workshop showed that Ruby was generally usable when programming with a screen reader. Ruby is an open source and mainstream programming language, and has significant potential as an introductory language for blind programmers. A more formal comparison between Ruby, other commercial programming languages, and specialized programming languages could identify opportunities and challenges to using Ruby as an introductory language for blind programmers.

Finally, while it was not part of our formal curriculum, all of our students were excited by the impromptu screen reader music session. A programming workshop in which blind students were trained to create interactive musical instruments might encourage musically inclined students to try programming. Our goal in including the tactile graphic production using 3D printing was to engage students by producing something "real." Unfortunately, this technology is still slow, whereas the product of music could be created in real time and therefore may be preferred.

6. CONCLUSION

Learning to program still presents many accessibility challenges for blind and visually impaired people. One major opportunity is to identify introductory programming experiences that are compelling to novice programmers, but that are also accessible to programmers with varied abilities. We argue that combining data analysis tools with visualization tools, and with the fabrication of tactile graphic-based visualizations, presents an ideal environment to teach programming to blind students.

Our results from a four-day workshop show that blind students were motivated to learn about 3D printing technologies, and to use their programming skills to create 3D-printed artifacts. We also found that Ruby and its interactive interpreter offer a sufficient, if not perfect, environment for teaching blind students to program. We hope that this work will motivate the development of software tools and curricula to support blind programming students in the process of exploring, analyzing, and visualizing data.

7. ACKNOWLEDGMENTS

We thank the National Federation of the Blind for organizing the STEM-X 2013 event. We also thank Patrick Carrington, Ben Gershowitz, Skye Horbrook, Karim Said, Md. Iftekhar Tanveer, and Clayonna Wheat for their help in running the workshop.

8. REFERENCES

[1] Apple. 2013. Accessibility: iOS: VoiceOver. Retrieved September 6, 2013 from http://www.apple.com/accessibility/ios/voiceover

[2] Bigham, J.P., Aller, M.B., Brudvik, J.T., Leung, J.O., Yazzolino, L.A. and Ladner, R.E. 2008. Inspiring blind high school students to pursue computer science with instant messaging chatbots. *ACM SIGCSE Bulletin*, 40, 1 (Feb. 2008), 449–453.

[3] Brown, C. and Hurst, A. 2012. VizTouch: automatically generated tactile visualizations of coordinate spaces. *Proceedings of TEI '12*, ACM Press, 131–138.

[4] Calder, M., Cohen, R.F., Lanzoni, J., Landry, N. and Skaff, J. 2007. Teaching data structures to students who are blind. *Proceedings of ITiCSE '07*, ACM Press, 87–90.

[5] Ludi, S., Abadi, M., Fujiki, Y., Sankaran, P. and Herzberg, S. 2010. JBrick: accessible Lego Mindstorm programming tool for users who are visually impaired. *Proceedings of ASSETS '10*, ACM Press, 271–272.

[6] Ludi, S. and Reichlmayr, T. 2011. The use of robotics to promote computing to pre-college students with visual impairments. *ACM Transactions on Computing Education*, 11, 3, 1–20.

[7] National Federation of the Blind. 2006. National Center for Blind Youth in Science. Retrieved September 6, 2013 from http://www.blindscience.org/ncybs-concept-paper

[8] National Science Foundation. 2013. Women, Minorities, and Persons with Disabilities in Science and Engineering: 2013. Special Report NSF 13-304. Retrieved December 3, 2013 from http://www.nsf.gov/statistics/wmpd/

[9] OpenSCAD. 2013. OpenSCAD–The Programmers Solid 3D CAD Modeller. Retrieved September 6, 2013 from http://www.openscad.org

[10] Raman, T.V. 1996. Emacspeak—a speech interface. *Proceedings of CHI '96*, ACM Press, 66–71.

[11] Ryles, R. 1996. The impact of Braille reading skills on employment, income, education, and reading habits. *Journal of Visual Impairment and Blindness,* 90, 219–226.

[12] Sánchez, J. and Aguayo, F. 2006. APL: Audio programming language for blind learners. *Computers Helping People with Special Needs*, Springer, 1334–1341.

[13] Sandewall, E. 1978. Programming in an interactive environment: the "Lisp" experience. *ACM Computing Surveys*, 10, 1, 35–71.

[14] Siegfried, R.M. 2006. Visual programming and the blind: the challenge and the opportunity. *ACM SIGCSE Bulletin*, 38, 1, 275–278.

[15] Smith, A.C., Cook, J.S., Francioni, J.M., Hossain, A., Anwar, M. and Rahman, M.F. 2004. Nonvisual tool for navigating hierarchical structures. *Proceedings of ASSETS '04,* ACM Press, 133–139.

[16] Stefik, A.M., Hundhausen, C. and Smith, D. 2011. On the design of an educational infrastructure for the blind and visually impaired in computer science. *Proceedings of SIGCSE '11*, ACM Press, 571–576.