Thinking Of Mathematics —An Essay On Eyes-free Computing

T. V. Raman
Google Research
http://emacspeak.sf.net/raman
raman@google.com

August 24, 2011

Abstract

This essay outlines some of my experiences as a mathematician who cannot see. Note that I transitioned to being a Computer Scientist during Graduate School. However I strongly believe in the edict "Once a mathematician, always a mathematician!" —my training in mathematics continues to influence the way I think.

I've been unable to see since the age of 14, which means that I've studied and practiced mathematics predominantly in an eyes-free environment. This essay is my first conscious attempt at asking the question "What is involved in doing mathematics when you cannot see?" I hope that some of the experiences outlined here will prove insightful to mathematicians at large. At its heart, mathematics is about understanding the underlying structure inherent in a given area of interest —and where no such structure exists —to define the minimal structure that is needed to make forward progress.

The general perception that mathematics might be hard to do in an eyes-free environment probably traces itself to the common view of mathematics as a field where one performs copious calculations on paper. I'll illustrate some of the habits and abilities one evolves over time to compensate for the lack of ready access to *scratch memory* provided by pencil and paper when working in an eyes-free environment. In this essay, I hope to demonstrate that mathematics in its essence is something far bigger. By being bigger than "calculations on paper", not being able to see rarely if ever proves an

obstacle when it comes to doing mathematics; the challenges one needs to overcome are primarily centered around gaining access to mathematical material, and communicating ones insights with fellow mathematicians. Thus, a large portion of this essay focuses on solutions to the challenges inherent in mathematical communication.

1 Creativity

The phases involved in the creative process were first described by German physiologist Herman Helmholtz in the late nineteenth century. He identified three stages of creativity:

- *saturation*,
- *incubation* and
- *illumination*

—see Promoting Creativity. These three stages have since been augmented with the additional step of *verification* by the scientific community.

When I started working on this essay, I found it useful to ask what impact if any my not being able to see had had on each of these stages within the creative process when studying or doing mathematics. This introductory section briefly summarizes my answers to this question —the remaining sections present a detailed analysis based on my experience of working on specific problems.

Saturation At this stage, one focuses on gaining a good grasp of the problem context. Given that a lot of mathematical literature is only available in print, this stage can be a challenge, especially when it comes to higher-level mathematics. But to do higher-level mathematics, one first needs to do elementary mathematics, and I believe that it is even more important to find the right kind of help when one is beginning to learn. I believe I gained a significant advantage here by virtue of having an elder brother who was a highly motivated teacher. The challenge of gaining access to higher-level mathematics might appear to be the more complex of the two; however I believe that proper access at the introductory level is far more critical, since good access at this stage ensures that a student with the necessary mathematical aptitude remains within the field to go on and solve the challenges that lie beyond.

Incubation Having absorbed the relevant material, this stage involves trying different approaches to making forward progress. In my experience, not being able to see has little or or no negative impact at this stage; in fact it might actually be an advantage since one has fewer distractions.

Illumination Not being able to see should not have any negatives at this stage. In practice, I have often found that I fall into the hole of *false* illumination *i.e.,* concluding that I have solved the problem when I have not fully done so. This can often be attributed to failures within the *saturation* step *e.g.,* missing a key portion of the problem statement, or pursuing an incorrect approach that has been tried and dismissed by others in the past. As the world of mathematics goes digital, I believe that tools like Google Scholar will serve to level the playing field in this regard.

Verification The verification step is closely connected with being able to reliably communicate mathematical ideas with ones peers. At this stage, it's important to be able to communicate with other mathematicians —traditionally, this meant writing with a piece of chalk. The move to electronic communication, and the invention of mathematical notation like TEX comes to the rescue here. The TEX notation *scales* —TEX keeps writing simple math simple, while being capable of encoding complex material. Thanks to TEX, I can communicate mathematical ideas via email —most mathematicians can read TEX math —I can also produce beautifully typeset mathematics when I have something more significant to convey.

2 First Experience —A Mental Calendar

I was 15 and couldn't see any more; I realized I couldn't look at a print calendar to find out the day of the week. I also had time on my hands . . .

I decided that an interesting problem to solve would be to figure out how to tell the day of the week given the date. I had always been good with numbers, and at the time I was 15 years old and felt I knew enough mathematics to be able to work out the solution. What's more, the possibility of being able to tell the day given any date seemed far more useful than being able to look at the print calendar for any given year.

Saturation

In this case, grasping the problem at hand did not require extensive reading —a lucky coincidence since I had no means of reading any material that I might have needed. The facts needed for gaining insight into the problem were ready to hand. After all, even though I couldn't look at a print calendar, I always knew what day of the week it was *today* —moreover, by counting either backwards or forwards, I could also figure out the day of the week for *nearby* dates.

Once I started down this path, the answer to the first question I asked myself

> What is the reason for the 1st of every month not being the same day of the week?

became readily apparent —given two dates, the change in the day of the week is given by taking the number of intervening days, and obtaining the remainder modulo7.As an example, January 1 1980 (the year I did this) was a Tuesday; I could conjecture that February 1, 1980 would be a Friday, and readily confirm that my understanding was correct because I could check the following facts:

- January 1, 1980 was a Tuesday.
- February 1, 1980 was a Friday.
- January has 31 days, and $31\%7 = 3$.

Thus, the *saturation* phase did not present an obstacle; what's more, being forced to count forward/backwards while understanding the problem meant I had a leg up with entering the *incubation* phase.

Incubation

The incubation phase did not last very long in this instance. Given that I had already started down the path of using modulo 7 arithmetic, I already had the essence of the algorithm at my fingertips —given a date, do the following:

- Compute the number of intervening days d from today.
- Compute $d\%7$.
- Use this offset from *today* to obtain the answer.

Illumination

Though the above steps do give the answer, computing d —the number of intervening days between January 1 and any given date —is cumbersome. I therefore spent some time computing the day of the week for *convenient* dates *e.g.,* January 1. As I did this, I found computing the number of intervening days between January 1 and any date in the same year cumbersome. But by then I had sufficient insight into the problem to realize that I could apply the previous technique to compute the day for the first of each month. So most of the incubation phase was spent getting better at modulo 7 arithmetic, and deciding what bits I would need to remember as opposed to computing on the fly.

I eventually arrived at the following:

- A year has 365 days. $365\%7 = 1$, and so January 1 moves by one day each year (and by 2 for a leap year.
- Compute a list of 12 numbers that each give the remainder $d\%7$ at the beginning of each month.
- Compute the day of the week for January 1 at the turn of each century to make calculations easy.

So here is what I still use to look at the calendar. Let Sunday be 0. The table of offsets for the 12 months (assuming February has 28 days) is obtained by computing $d\%7$ for each month where d is the number of intervening days between the first of the month and January 1. Thus, the offset for January is 0. The offset for month m is given by

$$\text{offset}_m = \text{offset}_{m-1} + (D(m-1)\%7)$$

Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
31	28	31	30	31	30	31	31	30	31	30	31
0	3	3	6	1	4	6	2	5	0	3	5

Given a date (m, d, y), compute $w = M_7 + d_7 + y_7$, where:

- m_7 is the offset for the month obtained by looking up the month in the table of offsets.
- $d_7 = d\%7$.

- Let $yy = y - 1900$ then $y_7 = (y + y/4)\%7$. The offset for the year 1900 is 0 since January 1 that year was a Monday —the offsets at the turn of the century cycle through $(6, 4, 2, 0)$.

Notice that the above computation adds in the extra day for leap years for all dates. Since the extra day is in fact added in at the end of February, the w computed above needs to be decremented by 1 when working with dates in January and February of a leap year.

Verification

Verification in this instance turned into a fun party trick —turns out that most people remember the day of the week for significant days in their life, and what's more, always ask you to verify their special day. They are also quick to tell you when you're wrong, which means you get an opportunity to both verify the algorithm, as well as checking your own ability to apply it successfully.

3 Solving The Rubik's Cube

> I learnt Braille when I was 17. The Rubik's Cube came to India around the same time, and marking a Rubik's Cube with Braille dots was the first useful thing I did with my newly acquired ability.

Saturation

I first heard of the Rubik's Cube on the BBC's Science In Action —growing up, the BBC World Service was something I listened to all the time. It took a few more months before the cube arrived in India —sometime in early 1982 which is also when I learnt Braille.

My initial exposure to the cube came via observing everyone around struggling to solve it and failing —this in itself was sufficiently motivating to want to solve it. I started by placing my finger on a given facelet of an unmarked cube and observing how it moved as I turned the various faces. At this stage, not being able to see the colors on the cube probably saved me from getting just as confused as everyone else around me. It also gave me the somewhat false impression that this wasn't going to be very difficult —a lucky misconception —because it also meant that I did not get discouraged early on.

As I began to discover patterns of movement on the cube, I felt the need to remember some of these patterns so that I could connect different things I discovered on the cube. I also discovered that it was important to hold the cube in a fixed orientation —otherwise I found myself getting very confused. I soon started referring to the bottom face as 1, the front face as 2 and the left face as 3. Whenever I discovered a sequence of face turns that I could predictably apply to move a facelet from a given position to another without disturbing its neighbours, I started associating a string of digits to that sequence. Notice that as a notation this was in fact incomplete —it fails to record the direction (clockwise vs counter-clockwise) that a face is being turned. But it was a key step in the saturation stage as I came to grips with the puzzle.

Still working with an unmarked cube, I also discovered the difference between edge cubelets (pieces that have two colors on them) vs corner cubelets (pieces that have three colors). I also realized that the face centers did not move at all, and were responsible for determining the color of a face. Since I was doing this by carefully holding a fingertip on a chosen piece, I also arrived at the useful insight that it was important to solve the cube a layer at a time —and *not* a face at a time.

Incubation

The incubation stage in this instance lasted almost a week from memory. Placing a fingertip on a piece and observing its motion was a good means of gaining insight into the puzzle. But as I made progress in that direction, things also got more and more confusing since I was now trying to do more complex steps. After doing complex hand contortions where I tried to track multiple pieces by placing different fingers on the pieces to track while attempting to turn cube faces, I realized that I needed tactile markers on the cube faces to make further progress.

At this point, my brother marked a Rubik's Cube with Braille stickers. We left the white face blank, and stuck small squares of relatively thick polythene marked with different Braille symbols on the 5 remaining faces.

I now got to experience first-hand why everyone else around me was so confused by the cube. But my earlier explorations during the saturation face had helped me build up an intuitive feel for how things worked. Needless to say, I was able to correct many of my prior misconceptions. I also fixed up my primitive notational system for remembering commonly occurring sequences —I attached $\pm$ to the digits to denote the direction in which a face was being turned.

At this point, I believe I had achieved parity with respect to attempting to solve the cube *i.e.,* not being able to see did not matter any more. From here on, my arriving at the solution went through the same sequence of effort, frustration and eventual success that everyone goes through when confronted with the cube.

Illumination

A week after marking the cube with Braille dots, I had an end-to-end solution that I believed worked. Given a randomly mixed cube, I went about it by:

- Start with the blank (white) face —identified by its center —on top.
- Move the edge cubelets of the top layer into position.
- Move the corner cubelets of this layer into position.
- Move the edge cubelets that made up the second layer into position.
- Move the corner cubelets on the bottom layer into their slots.
- Move the edge cubelets in the bottom layer into position.
- Orient the edge and corner cubelets correctly on the bottom layer.

Verification

After claiming to have solved the cube, and reliably solving it a few times, I got my first rude surprise —someone mixed it up and I found I reached a configuration for which I had not worked out a reliable sequence of moves. Thus, in this case verification was iterative.

The problem was that I had missed the possibility that one can place a pair of cubes in the bottom layer in their home position and still have the other pair swapped. By this time, I was sufficiently well-versed in the secrets of the cube to derive the sequence of moves needed to dig myself out of this hole.

After finally learning to solve the cube, I spent the subsequent months deriving different shortcuts that allowed me to move multiple pieces into position in parallel —eventually I could solve the cube on average in under 30 seconds. I even relaxed one of my early constraints —having to hold the cube in a fixed orientation —when I solved the cube one-handed. Solving a Braille cube one-handed is an interesting challenge because I was using the same hand to both feel the Braille markers and turn the cube faces. Doing this requires physically orienting the cube in your hand so that you can turn a given face; this means that your mental model of the cube needs to account at each stage for a completely re-oriented cube.

Insights

Discovering the algorithm for computing the day given a date as well as solving the cube did not require doing mathematics in the traditional sense. Neither required me to access significant amounts of existing mathematical literature. When I had arrived at the solution, communicating the result was also easy. Thus, the two steps where not being able to see could potentially get in the way were in fact easy. At the same time, not being able to see helped me focus on the problem to a greater extent than I otherwise might have; additionally, I might not have been as motivated if I had been able to see.

4 Devising An Efficient Braille Notation For Math

I spent a year attempting to chase down the Nemeth Braille code for writing Math. I finally gave up and created my own in the summer of 1983 . . .

Saturation

During my first year of school after learning Braille, I tried in vain to chase down the Nemeth code for writing math in Braille. This was 1982 in India, *i.e.*, before the global availability of email. After mailing out many letters with nothing to show for it, I spent the summer of 1983 designing a math notation that I could use for taking class notes in Braille. Here are some of the requirements that went into this design:

Succinct The notation had to be sufficiently succinct to enable me to write at classroom speeds using a pocket slate.

Extensible The system had to be extensible to enable me to invent notation *on the fly* as I encountered new concepts and their accompanying mathematical notation in class.

Speed The notation had to be sufficiently concise to enable someone like myself who had learnt Braille late in life to read fast. This meant minimizing the number of dots it took to write —something that meshed well with the goal of succinctness —fewer dots are both quicker to read and write!

Incubation

I experienced first-hand what it meant to study mathematics without access to a good reading/writing system during the 1982 academic year. I was in the 11th grade, and relied exclusively on my brother to read me textbooks. I gave written exams with a *writer* —a student who would read me the question and write down the answer as I dictated. This necessarily forced me to practice solving problems a step at a time and dictating each step to the writer.

Solving problems in this mode is distinct from solving math problems mentally. Notice that the presence of the writer meant that I did have access to *scratch* memory of sorts. Dictating each step to the writer, having it read back to confirm that it had been written correctly, and proceeding to the next step meant that I needed to use *mental* calculation only for individual steps. This actually required a fair bit of practice since it was always tempting to try to solve problems in calculus or trigonometry end-to-end and then dictate the solution. In practice this is both error-prone and unnecessary —especially given an exam system that focuses on the student's ability to finish a given number of problems in the allotted time.

Illumination

The abilities I developed during the academic year 1982–1983 served me well on two counts. The ability to break down a problem and work through it systematically is more broadly useful than when taking timed exams with a writer. Secondly, my experience that year helped me implicitly understand the design requirements enumerated earlier for my ideal Braille notation for mathematics. I spent the summer of 1983 designing such a system and continued to use the result throughout my student career.

The Braille notation I developed had the following features:

Phonetic To meet the succinctness requirement, I first created myself a phonetic shorthand in Braille. A Braille cell has 6 dots arranged in a 3×2 matrix to give 63 distinct symbols. The shorthand I devised dropped all vowels, and used different symbols for commonly occurring syllables. Further, the position of a symbol within a word changed the syllable that it denoted. As an example, the Braille character obtained by using the two dots in the bottom row of the 3×2 matrix denoted the syllable *pr* at the front of a word; it denoted the syllable *cy* at the end of a word. I therefore wrote the word *pricy* by repeating this character. Writing the word *pricy* in standard

Braille would take 5 characters comprised of $(4, 4, 2, 2, 5) = 17$ dots. In comparison my shorthand could encode the same word with 2 characters that used $(2, 2) = 4$ dots.

Math Escapes Braille is essentially a linear writing system —later, I came to appreciate that computer-based writing systems like $\TeX$ are also linear. To encode two-dimensional math notation, I assigned particular Braille symbols to denote the start of subscripts or superscripts —I later came to recognize these as escape sequences when I learnt computer science. I used parentheses to group sub-expressions. I used a special symbol to precede English letters to denote that the symbol was the Greek equivalent —as a computer scientist, I now recognize this as placing a symbol in a new namespace.

Verification

I started using the resulting Braille codes in class during the academic year 1983–84 and to my surprise it worked very well. I had been *taught* to write with my right-hand when I was very young. I suspect that if left to my own devices, I would have written with my left-hand —since I could see with my left eye. When I learnt Braille, this too turned into an advantage —I started reading Braille with my left hand. The ability to write right-handed while being able to read with my left-hand meant that I could read and write in parallel —something that proved quite useful when doing mathematics.

Both the Braille writing system I devised —as well as the divide and conquer strategy for problem solving during exams *scaled* well. I invented new symbols as I learnt more mathematics and encountered newer notation; but the underlying notational system never changed. When I encountered new concepts and associated new notation in class, I always asked what the visual notation was, and then invented a Braille notation that best matched. This also made it easy for me to remember the visual notation (which I would need to know when dictating my exams). As an example, when I first encountered group theory, the notation for group $\langle G, +\rangle$ was described to me in class as

G, $+$ enclosed in angle brackets

I automatically chose the Braille symbols I had previously used for $<$ and $>$ as a new pair of delimiters. For the record, I always sat in front of the class and insisted that every instructor spoke as they wrote on the blackboard —further, I was never shy if I heard the squeak of a chalk without an accompanying utterance from the person at the board.

Similarly, the strategy of solving written problems a step at a time and using the previously written step to provide scratch memory also scaled from high-school calculus to college-level mathematics. As an example, I took a class on Linear Programming in college and still have unpleasant memories of having to solve *transportation* problems using the Vogel's Approximation Method in timed exams. These experiences gave me two key insights that have served me well:

- Specific mathematical techniques *e.g.,* differentiation, integration or the Simplex method are algorithms.
- To truly appreciate an algorithm and understand how it works, one needs to be able to *run* it by hand on a representative set of examples.
- In mathematics, this translates to being able to differentiate or integrate a given expression.
- The latter requires a set of *semi-mechanical* steps and this is where one uses aids such as *scratch* memory provided by pencil and paper —something for which I needed to compensate.
- However, a true understanding of the underlying algorithm is far more important than any specific technique that one might devise for *running* the algorithm on specific instances.

5 Learning To Program

Asked to program a game in CS 101, I expressed the game as a recurrence equation and solved it —it made for a very short program that *always* won the game . . .

Saturation

I was asked to program the following two-person nimm-type game for a final class assignment. Here are the rules:

- The game starts with n sticks on the table, with each player taking turns to pick up k sticks.

- The first player can pick up at most $n-1$ sticks.
- Assume that a player picks k sticks at a given turn. Then his opponent can pick up at most $2k$ sticks.
- The player who picks up the last set of sticks wins.

We had been taught the technique of searching through game trees in class and were expected to use this to complete the assignment. To provide some additional context, CS 101 students at IIT were assigned limited amounts of computer time —typically 60 minutes slots between the hours of 10pm and 5am. I used to program by taking along a student to read the display for me.

Incubation

I was highly motivated to devise a solution that would not require me to go multiple times in the middle of the night to the computer room to implement a solution. I like sleeping early and well, and finding willing volunteers in the middle of the night is not easy. The problem description was simple, and not being able to see was not a shortcoming at the saturation stage. It turned out to be a significant advantage during the incubation phase. While the rest of my peers begged and borrowed additional computer time to implement a game-tree based solution, I spent my time thinking about the problem in the relative comfort of my dorm room.

Analyzing the game, I simulated it for small values of n and discovered the following:

- The game is meaningless for $n = 1$.
- If $n = 2$ the first player loses.
- If $n = 3$, first player loses.
- If you play two successive games (n_1, n_2), where the first player is guaranteed to lose each game, then the first player can be forced to lose for $n = n_1 + n_2$.

As I jotted down the numbers $(1, 2, 3, 2 + 3 = 5)$, and observed above facts, I spotted the Fibonacci sequence. I initially conjectured that the first player would always lose if n was a Fibonacci number. This then indicated a possible winning strategy; if n is a Fibonacci number, ask the opponent to go first; if not, play first and pick k sticks such that $n - k$ is the closest Fibonacci number.

Illumination And Verification

There remained but one twist to complete the solution —the rule that said you could pick at most $2k$ sticks at each turn. Consider $n = 12$. This is not a Fibonacci number; however naïvelyapplying the strategy would suggest picking $k = 4$ sticks to leave a remainder of 8 sticks —this allows the second player to win. So I went back to the initial strategy of decomposing larger games into smaller ones. Given $n = 12$, consider it as a pair of games $(4, 8)$. You go first, and win the game for $n = 4$ by forcing the opponent to the closest Fibonacci number 3. When this game for $n = 4$ concludes, the opponent is left to start the next game with $n = 8$ which is guaranteed to lose.

My final submission consisted of a 40 line Fortran program accompanied by a two-page proof that I typed out on a portable typewriter. Given the size of the program, it only took me one 30 minute session in the computer room to finish the assignment.

Looking back, I believe not being able to see gave me a significant advantage over my fellow students in this instance. Communicating the solution was an interesting challenge, since I did not have access to or know anything about mathematical typesetting. The proof I wrote up was therefore mostly in plain English. But the implementation that went along with the proof was the clinching argument —the program beat everyone who played against it.

6 ASTER —Speaking Mathematics

I obtained my first talking computer during the second semester of Grad School at Cornell and learnt (LA)TEX. Then, I found that the computer couldn't speak the math I was writing . . .

I took CS 681, the graduate class on algorithms at Cornell in the fall semester of 1990. The instructor, Dexter Kozen, was using lecture notes typeset in (LA)TEX. Since I had a computer that could talk, I asked him for the (LA)TEX sources. After listening to the speech synthesizer speak (LA)TEX code for a few days, I decided that I could make it do far better —this eventually led to the work on audio formatting and my PhD thesis entitled ASTER —Audio System For Technical Readings.

Saturation

In a sense, the saturation stage for this problem had begun long before. Over time, I had learnt to recruit enthusiastic student volunteers to read math material for me, and this required training readers in efficiently speaking complex mathematics. I conceived the idea of getting the computer to speak (LA)TEX documents at the same time that I learnt (LA)TEX —a key step in the saturation stage.

I learnt TEX by reading the raw sources for the TEX book; later, I obtained the (LA)TEX sources to the LATEX book from Leslie Lamport at DEC Research. This was a case of jumping in at the deep end —reading the TEX sources to the TEX book is not the easiest way to learn TEX. However, the TEX sources were readily available, and once I had overcome the initial hurdle of listening to the TEX markup, I learnt TEX to a far deeper level than I otherwise might have. This might not have been necessary if all I had needed to do was to author TEX documents; but a full understanding of the TEX machinery served me well when it came to implementing a system that consumed (LA)TEX documents to produce rich auditory renderings.

I spent the summer of 1991 at Xerox PARC as a summer intern in the Electronic Documents Lab. This proved to be an excellent environment to absorb the background knowledge in document understanding and electronic documents that I would need to implement the final system. Looking back, I believe this experience helped me work around some of the lack of access to relevant literature that might have otherwise held me back in building the right system.

Incubation

The system I built was primarily motivated by my own desire to read math publications. This meant that I had a significant leg up during the incubation phase —being the system's primary customer meant that the feedback loop between conceiving, implementing and testing out different ideas was extremely tight. In 1990, I published an early write-up in TugBoat on the predecessor to ASTER — a simple SED script called TEXTALK that transformed (LA)TEX documents. An interesting side-benefit of this write-up was that it put me in touch with the TEX community and in particular Barbara Beaton of the AMS; she provided me access to AMS Bulletins in (LA)TEX that I used as input to early prototypes of ASTER.

By the middle of 1992, I was well into the incubation phase and ready to implement ASTER. I chose Common Lisp Object System (CLOS) as the implementation language. I was still learning to program in the large, and ASTER was the first significant software system that I implemented. This meant that I was "learning on the job" and needed access to the relevant reference material.

By then, I had discovered that I could email authors asking for access to the markup sources of books I really needed. Usually, a brief description of the project I was working on, followed by a letter from the publisher granting the author permission to give me the files was all that was needed. usually I received (LaTeX)TeX files which also became input for the system I was building. Here are some of the programming books I used in this form:

- Structure And Interpretation Of Computer Programs (SICP) by Abelsen and Sussman.
- Paradigms of AI Programming by Peter Norvig
- The Common Lisp ANSI specification

I augmented these with excellent online support from Usenet group *comp.lang.lisp*.

Illumination

In summer of 1992, I picked David Gries as my Phd adviser. After I described the system I was trying to build, he thought about it and said

> First design a language in which you can describe how you want documents to be spoken.

It took me a week to appreciate the import of this suggestion —but looking back, this was the final breakthrough that made ASTER a workable solution. The rest was relatively easy:

- I had the necessary test material in the form of electronic books.
- I had the resources I needed to teach myself to program.
- I had the end-user (myself) to test the system on.

Verification

AsTeR was implemented during the calendar year 1992 and by early 1993, I had started testing the efficacy of the auditory renderings by having people in the CS department at Cornell write down equations as they heard it spoken by the system. I declared the system complete once it reached a level where users correctly wrote down what they heard. The final test was to collect a set of mutually ambiguous examples, have these rendered by AsTeR, and compare the result to the recording produced by a trained reader from Recordings For the Blind —Chuck Romine of the Oakridge Tennessee Labs volunteered to produce a suitable cassette tape.

I built AsTeR as a tool for reading mathematical documents. But how about tools for writing mathematics? Despite What You See Is What You Get (WYSIWYG) systems being the present rage, I believe that (LA)TeX markup is still the best option for writing mathematics when you cannot see. Here are some reasons why:

- The results of (LA)TeX markup are predictable. As someone who cannot see the visual output, I never want to risk getting what I didn't see by using a WYSIWYG system.
- (LA)TeX linearizes the two-dimensional math notation. It is hard to understand that linear notation when you are also trying to understand the underlying mathematics being communicated; this is why mathematicians format their (LA)TeX documents to produce good visual copy.
- AsTeR audio formatted (LA)TeX so that the listener could focus on the math being communicated.
- When you are writing down something you already understand, the linearized (LA)TeX does not prove as significant a cognitive burden, and given the drawbacks in WYSIWYG systems mentioned earlier, (LA)TeX remains an attractive authoring solution.

As an aside, the inventor of TeX Donald E Knuth mentioned to me when we met at the TUG95 conference that TeX math notation had itself been motivated by how mathematicians speak expressions when conversing with one another. Thus, `x_1 + x_2` closely matches what a mathematician would say:

x sub 1 plus x sub 2

A few years earlier, Brian Kernighan, the inventor of EQN told me that the notation he invented was informed to an extent by his experience of recording math books for Recordings For The Blind in Princeton. I believe that this is more than a coincidence —simple linearization of two-dimensional mathematical notation as embodied by systems like TEX will probably remain one of the most effective means for mathematicians to communicate math to the machine, since they appear to closely mirror how we think —at least at the linguistic level.

7 Zome Systems —Rediscovering Mathematics

> I discovered Zome Systems in 1999 —my only regret is that I did not have it when I was a student . . .

I started implementing ASTER because I found math publications difficult to read —in 1990 it was because listening to (LA)TEX markup took too much away from focusing on the mathematical content. When I had finished implementing ASTER in the fall of 1993, I had a system that spoke publications from AMS Bulletins very well —but I still couldn't understand them since I had lost touch with math.

I rediscovered many of the things I enjoyed about Mathematics after coming into contact with Zome Systems in late 1999 —a polyhedral building set that leverages the symmetries of the dodecahedron/icosahedron. I think Zome Systems makes a wonderful teaching aid for students —and even more so for students who cannot see.

Saturation

On the surface, Zome appears to be a very visual toy —the sticks are color coded. But like any well-designed system, Zome sports a high level of redundancy —the sticks are also shape-coded.

Understanding the geometry of the Zome ball during the saturation phase presented interesting challenges. The Zome ball has three types of holes —pentagonal, triangular and rectangular. The ball is therefore flatter at the pentagonal holes, and as a consequence, if you roll the ball between finger and thumb, it usually comes to rest with a pentagonal hole against the fingertip. Once I understood this, the next step was to fully populate a zome ball with sticks of a given color —blue (flat), yellow (triangular) and red (pentagonal) sticks go into the appropriately shaped holes. Fully populating a Zome ball with a given type of stick makes it easier to perceive the underlying geometry via touch since this effectively increases the resolution.

After buying my first Zome kit at Linuxworld 1999, I looked it up on the Web and found many Web sites describing its underlying geometry —primary among these being George Hart's site on Zome geometry. Later, he provided me online access to his book on this subject.

Incubation

During this phase, I built several fun models including many that I had studied in the context of abstract algebra and group theory. My permanent favorites among these are:

- The compound of 5 cubes.
- The compound of 5 tetrahedra.
- The compound of 5 rhombic dodecahedra.

In addition, the highly symmetric rhombic triacontahedron is a favorite that shows up in the context of all of the above models.

Illumination And Verification

After a couple of years of playing with Zome Systems, I put together a paper describing some of the things that could be learnt with my co-author Krishnamoorthy —see Visual Techniques For Computing Polyhedral Volumes. I authored this material (including all figures) in (LA)TEX. The figures were drawn using package *pstricks*. I worked with my co-author in (LA)TEX with email being the primary form of communication. This is a good example of (LA)TEX providing a common language and thereby bridgeing the communication gap that I would face if I still needed to use a piece of chalk to communicate my ideas.

8 Writing This Essay

To iterate is human, to recur divine . . .

To conclude, let us apply the overall framework of this article to the creation of this essay.

Saturation In a sense, one can consider all of the experiences described in this essay as forming part of the saturation phase.

Incubation Most of the incubation phase was spent asking myself the question "what does it mean to do math when you cannot see".

Illumination I had a good sense of the material I wanted to cover, but was still searching for a good framework within which to organize what I wanted to convey. At around this time, I was lucky to attend an excellent talk by Murray Gell-mand at Google on the topic of creativity. At the outset of his talk, he laid out the steps in the creative process. This was the final piece I needed; I decided to structure the essay in the form you are presently reading it.

Verification The verification step remains —it cannot be completed without readers' comments.