The Art of Digital Publishing: A foundation of combined standards to support the future of publishing

Daniel Lynch

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-268
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-268.html

December 18, 2012

The Art of Digital Publishing

A foundation of combined standards to support the future of publishing

Dan Lynch

UC Berkeley

EECS Department

D@nLynch.com

1st of December 2012

Contents

Abstract

Scientific content increasingly relies on the presentation and authoring of complex multimedia diagrams and figures, sometimes interactive, to convey information in a non-textual way. Wikis and user-generated hyper-linked content have both been very successful in the case for text—this is what we aim to do for mathematical diagrams. Many professors in higher education who write textbooks know TeX, however, they don't often know how to program the Web. The future of building interactive user interfaces should lie not in the hands of programmers, but in the hands of the expert of a given field—the goal of this project is to supply math, physics, and engineering professors with a platform to express mathematical concepts to students to provide immersive learning environments. Ideally, this projects serves twofold: First, in closing the gap for non-web-technical authors to express ideas and concepts through Web technology without the knowledge of coding or user interface design, by mapping a typesetting language to interactive programming. Second, in providing deep, educational experiences for our youth to engage more in the sciences, and begin to use exploration and creativity in learning through interactive textbooks. The loose structure and nature of user interface design poses a problem for documenting science and related interfaces in a consistent manner. TeX provides us with some "laws" to obey in order to design the output of a text and graphical language around. Hence, we can attempt to create a synthesis of a structured user interface specification (TeX) and a structured functional specification (HTML5) to provide a publishing platform for the current and next generation. The Art is where we can blend these two standards bodies; higher levels of abstraction allow people to express their ideas without having to worry about the mechanisms by which the technology is rendering their works. It is in these environments when people can express themselves freely.

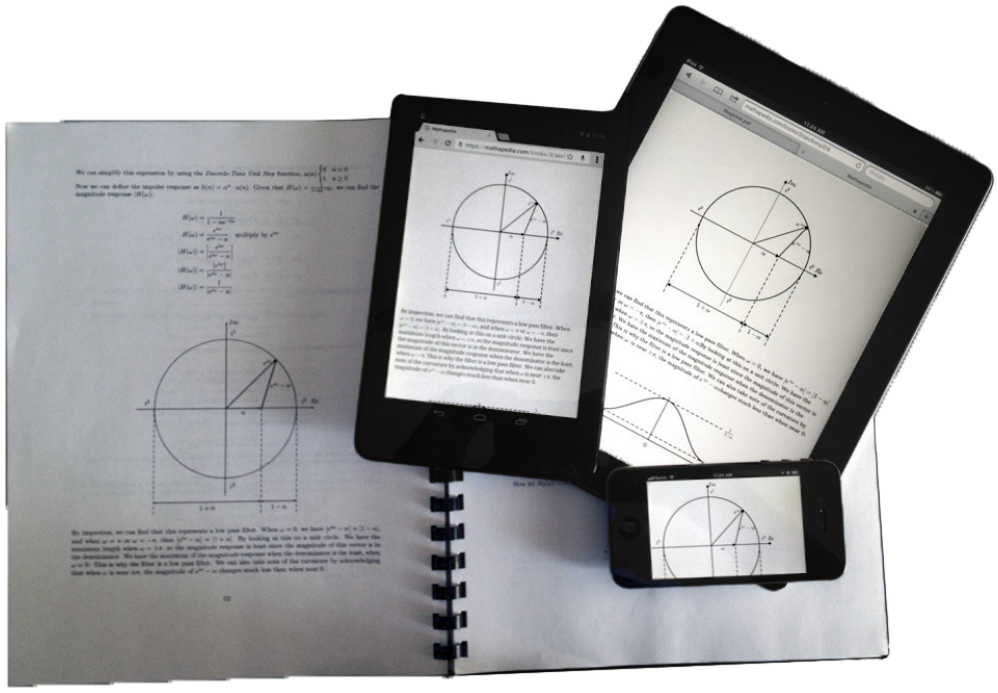
1 Preface

1.1 A note about this transmission of information

This paper serves as the documentation, philosophy, and implementation of a web-based typesetting platform called LaTeX2HTML5

It's hard to say whether this is a document, paper, or web page because it's intended to be viewed on either paper or a computer, on a phone or a tablet. The concept is to communicate an idea, and to transmit information to the reader in a consistent manner, no matter which method the reader decides to consume that information (although parts are more interactive online).

You seem to be reading the *static* version of this text. You can access the online version here (where most of the diagrams are interactive): https://mathapedia.com/books/31



It's almost impossible to go any further without the mention of Donald Knuth, arguably one of the most influential computer scientists alive today. Knuth created TeX, which is the *standard de facto* for academic writing. In was using this typesetting language in which he wrote "The Art of Computer Programming", which is one of the largest texts on computer science available. In one of the many volumes, he writes:

> Several programs that implement the algorithms in this book can be found on the diskette that comes with the book, as well as on the WorldWideWeb. The programs are two kinds: some Maple programs and some Mathematica programs. It should be noted at once that both the individual

> programs and the packages in their entirety will continue to evolve after the publication of this book (Knuth, 1973, 205).

The dichotomy of the function of an algorithm and its description interesting—the reader must go to a web site and download the functionality after reading. In today's world, we don't need diskettes with programs, we have the ability to run interpreted programs in a web browser—today, a digital version of his book could now implement the algorithms described, and perhaps as mentioned, this could be next next evolution of his text.

In this paper I present a web-based typesetting language for building interactive mathematical user interfaces and learning environments using TEX as the syntax for authorship.

2 Introduction

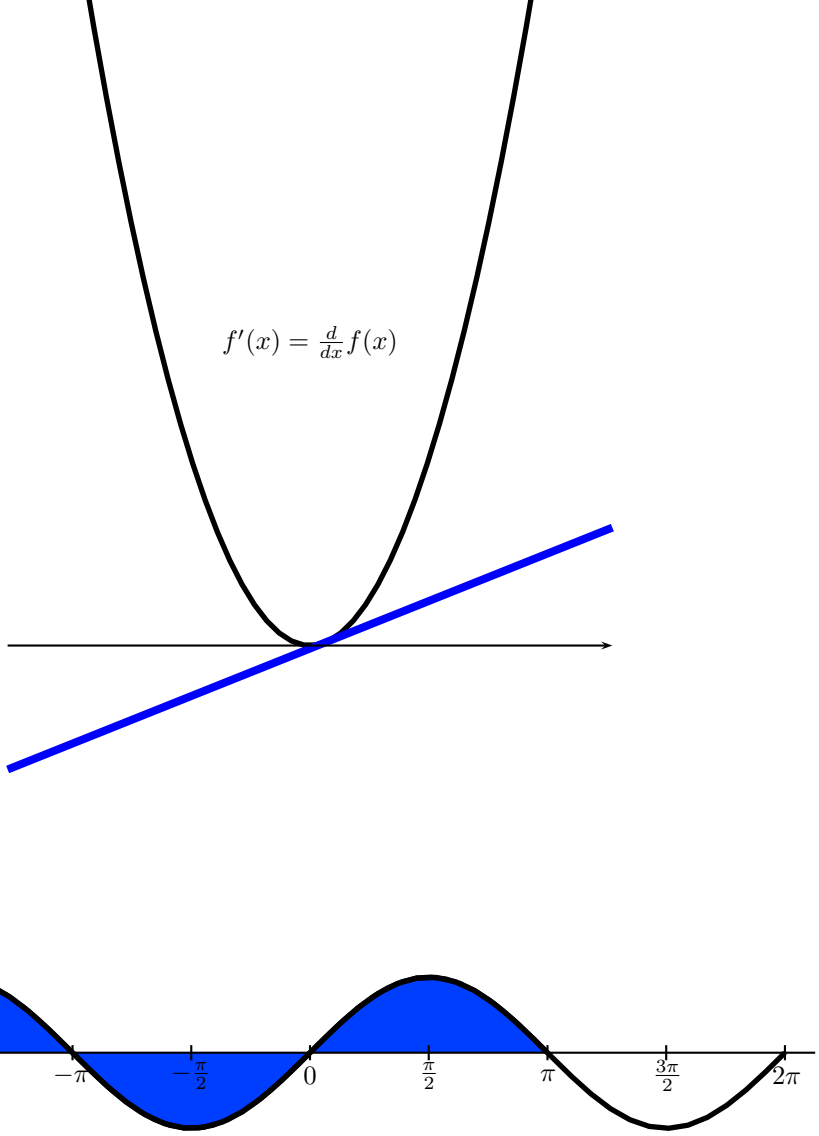
2.1 Education

Technology is growing faster than the rate at which the educational system is evolving, and the distance between these two vital functions of our society needs to be reduced. Utilizing computer science we can provide optimized learning methods to match this proliferating technology and accelerate education.

Gleaning knowledge from aggregate arts and sciences, the creation of supplemental learning tools, used by teachers and students, through poly-sensory experiences, can build solid foundations as a precursor to our future's successful and meaningful experiences in higher education.

New learning methods will accelerate education in a way that provides higher standards so that it catches up to technology. These new horizons will empower the future pioneers of science to explore new frontiers progressing from the technology revolution and information exchange era we live in today.

Imagine if middle school students were introduced to calculus concepts through interactive dimensions that provide an instantaneous understanding of a derivative or integral:



This changes the meaning of what $\frac{d}{dx} f(x)$ and $\int_a^b f(x)dx$ mean to a student, and these neurological footprints in our youth can be beneficial to our future. This ingrained solid foundation will allow for the affirmation of knowledge when it is learned in higher education, activating and reinforcing neurological connections, as opposed to learning important information for the first time; this type of preparation will metamorphose education into an edifying enlightenment.

Ideally, this projects serves twofold: First, in closing the gap for non-web-technical authors to express ideas and concepts through Web technology without the knowledge of coding or user interface design by mapping a typesetting language to interactive programming. Second, in providing deep, educational experiences for our youth to engage more in the sciences, and begin to use exploration and creativity in learning through interactive textbooks.

2.2 Publishing and Standards

A huge problem an author is faced with upon writing a digital textbook these days, is that in creating such digital learning content, there is no standard format. There exists not a cooperation, but a competition of

numerous attempts at forming "eBook" and digital textbook formats by myriad corporations. Are today's authors equipped for the numerous publishing platforms?

I argue we should throw out all this noise and use what we already have. There exists a standard, in fact two—HTML5 and TEX, and they aren't going anywhere. In fact, both happen to be the standard for their respective fields (Web and Academics).

Just as music notation is what enables an orchestra to play in a synchronized manner, standards can enable us to document science in a harmonious way.

HTML5 has proven itself as evolving from connected static documents to a viable computing platform for interactive applications and resources. TEX has also lasted the last 30+ years as the standard for publishing academic text.

Many professors in higher education who write textbooks know TEX, however, they don't often know how to program the Web. The future of building interactive user interfaces should lie not in the hands of programmers, but in the hands of the expert of a given field—the goal of this project is to supply math, physics, and engineering professors with a platform to express mathematical concepts to students to provide immersive learning environments.

I claim we can use existing standards to achieve the rapid adoption of utilizing technology to help education and solve the race to finding a viable digital publishing format.

2.3 The Art of Communicating Ideas

It's hard to see how the divergence of publishing formats affects us because authors represent a small percentage of our population. Science is good for society, and people need to share information—organizing education and learning can improve the articulation of scientific ideas. Can authors fully express ideas and concepts through existing publishing channels?

> In order to understand the phenomena surrounding a new technology, we must open the question of *design*—the interaction between understanding and creation. In speaking here of design, we are not restricting our concert to the methodology of conscious design. We address the broader question of how society engenders inventions whose existence in turn alters that society (Winograd and Flores, 1986, 4)

A new technology for publishing should facilitate *the transmission of ideas*—a professor doesn't just write a book, they are impacting many students across the globe.

Overtime, through *increased levels of abstraction*, like TEX → LATEX, authors have been able to publish more freely, because they can adopt and learn the standard typesetting languages. The distance between their intentions and articulations remains somewhat small. However, if an author must use a new interface

for publishing, this increases the distance between the expression of their ideas and the actual concepts at hand.

The hope is that by accelerating the authorship of academic work digitally, as well as increasing the interactivity and immersion of the resulting documents, both publishing and learning are positively influenced.

2.4 The Web and Typesetting

Many attempts and standards have been introduced to standardize the transmission and display of scientific text, formulae, and diagrams across the web. These include, but are not limited to MathML, SVG, and the canvas HTML5 element. The issue with these is that although they are supported on most browsers, the creation of content within these constructs can be verbose or require programming.

New tools have produced great value, namely MathJAX, which has solved rendering of mathematical formulae online. However, MathJAX is *only* focused on the math, not typesetting or generating graphics.

Tools such as `latex2html` have been released for converting typeset documents to HTML, however, they require a terminal, are not real-time, and do not render interactive graphics and diagrams. This solution also generates several static pages on a user's system, which still requires an upload for them to be made useful.

Since the successes of typesetting and word-processing, authoring the web through various forms of HTML has grown into myriad expressions. Today thousands of web frameworks exist, at a time where the common Web browser is a viable computing platform, and through programming we can provide real-time authoring of interactive mathematics across multiple devices with poly-sensory interactivity.

The Web has an infinite number of expressions, although it has strict functional specification. The general idea of this project is to combine two standards; LaTeX for how things should "look", and HTML5 as the standards for how things should "function".

The gap that should be filled is simple. Static HTML pages are not enough—diagrams need to be generated, and if interactivity is required, then dynamic computations need to made, which requires a dynamic language such as JavaScript. However, the authorship of HTML and JavaScript is not easy, thus using an existing standard de facto language like LaTeX for generating interactive documents can optimize the authorship of interactive material to help accelerate education.

Donald Knuth, who was the first man to call programming an Art, successfully paved the way for us to document science in a consistent and beautiful way through TeX. His creation was the result of frustration in producing mathematical books digitally, and to enable the authorship of **beautiful** books, "Gentle Reader: This is a handbook about TeX, a new typesetting system intended for the creation of beautiful books—and especially for books that contain a lot of mathematics" (Knuth, 1986, 5).

This text is about LaTeX2HTML5, a new web-based typesetting system intended for creation of beautiful

interactive books—and especially for books that contain of lot of mathematics. Donald Knuth, in his book "The Art of Computer Programming", said: "Science is what we understand well enough to explain to a computer. Art is everything else we do". It is the Art that allows us to bridge the gap for those that are interested, and that is the expression at which this project strives.

3 The Evolution of Ideas and Text

3.1 From Ideograms to the Alphabet

We've been on the planet Earth thousands upon thousands of years with the ability and need to communicate. Spoken word eventually became written word, and the genealogy of letters, text, and publishing can show us how the transmission of information has shaped our lives and launched publishing history down its current path.

Using text and images dates back to our primates who painted on caves—hieroglyphs were the standard for "writing" for over 3,000 years during the fourth millennium BC, which marked the beginning age of writing (McDermott, 2001, 12). The first systems of picture writing date back to ages "prior even to the taming of any domestic animal, or the cultivation of cereals; earler, so far as we know, than the construction of any kind of human habitation" (Taylor, 2003, 17).

We know there are at least five systems of "picture writing" that were each formed independently of one another, the Egyptian, the Cuneiform, the Chinese, the Mexican, and the Hittite (Taylor, 2003, 14). Through studying the history of these systems, we have seem the development of alphabets, starting from

graphics that represent ideas and concepts called an "ideogram":

> The history of the cuneiform writing also illustrates with great completeness the successive stages through which writing tends to pass; the primitive ideograms developing themselves, through verbal phonograms, into syllabic signs, until finally the alphabetic stage is reached (Taylor, 2003, 40)

This paved the way for us to have a standard for communication as a civilization—this event spawned a new conduit for ideas.

3.2 Age of Scribes to the Printing Press

Before Johannes Gutenberg's invention of the printing press in the late fifteenth century, books were created during the age of the scribes, where people copied books manually by hand, making it slow and expensive (Tames, 2006, 4). The shift from script to print was one of the most important events of our civilization (Eisenstein, 1979, 5).

Gutenberg's idea was to combine *two existing technologies*, printing and the press. Printing was done using carved wooden blocks, and the press was used to crush grapes to extract juice (Tames, 2006, 4). Another advancement was to use pieces of metal instead of carved wooden blocks so they could be continued to be used (Tames, 2006, 4).

Gutenberg had spent over 20 years and his life savings into building the printing press (Rees, 2006, 10), and never received a penny for his invention. It wasn't until the 16^{th} century that he was given credit as the inventor (Rees, 2006, 15), however, his impact on the world has not gone unseen since. In 1453, Gutenberg began working on a three volume bible. At that time, one bible would take four years to complete, but with the advent of the printing press, it could be produced 90 times faster. This meant that books were not only for the wealthy, but for everyone, which ultimately began the movement for the democratization of knowledge (Tames, 2006, 5).

"After the advent of printing however, the transmission of written information became much more efficient. It was not only the craftsman outside universities who profited from the new opportunities to teach himself. Of equal importance was the chance extended to bright undergraduates to reach beyond their teachers' grasp" (Eisenstein, 1979, 66).

3.3 Evolution of Text and Literacy

People became literate with the help of the alphabet and the printing press. One was a standard, the other was a process of dissemination. For the case of learning through text, alphabets and languages provide syntax and rules for the ways characters combine to map to ideas in the same way that more advanced systems of picture writing synthesized simple ideograms to facilitate transmission of more complex ideas.

Although intellectuals in 13^{th} century Europe had read Aristotle, Euclid, Ptolemy, Cicero, Galen, Hippocrates, it wasn't until the Protestant Reformation that was led by Martin Luther, when his translation of the New Testament when written in the vernacular in 1529. Books contained alphabet and syllable tables in the vernacular by the late 16^{th} century, with intention of enabling children to learn (Ellsworth, 1994, 41).

The first press was established in the the mid-17^{th} century in Cambridge, Massachusetts, and textbooks were published within the original Colonies. The American textbook today is a product of the later half of the 19^{th} century when the school was a profitable market and the transportation infrastructure allowed it (Ellsworth, 1994, 41). The authorship of textbooks became more important over time as school and literacy popularized, and more children and people were learning, which further increased the amount of educated people in the population (Ellsworth, 1994, 9).

3.4 The Information Age

As society's aggregate technological knowledge progressed, developments in science enabled the production of complex systems, which allowed for the advancement of hardware and software in the 1930's through the 1950's. This created a foundation for the formation of the personal computer in the 1960's and 1970's.

> The next revolution in the mass production of machines was software. The late 1970s saw the rise of the personal computer, a device capable of behaving as any machinetypewriter, adding machine, filing cabinet, arcade game—when given the right instructions. Manufacturing a "machine" was now just a matter of printing its instructions onto a disk, and production costs plummeted (Victor, 2006, 42).

What started out as a mission to seek a military defense system, founded what is known as computer graphics and the Internet today. Around the same time, the aerospace industry developed the first plotting software and CAD system, and academia created user interfaces to input graphical data into computers. Eventually, technology advanced, and software that at one point was proprietary to engineering firms and the military, became available to anyone with a personal computer.

3.5 Computer Graphics and Networks

The design of systems has been greatly influenced by computers. For instance, the development of automated design and drafting came about in the 1950's and 1960's within the military and US Air Force. In the search for new defense systems, an innovative system was created that used a visual display to transmit vital, spatial information regarding a possible enemy threat. Since then the technology has continuously evolved at an accelerated rate, and spawned the fields of Computer Graphics (CG) and Computer-aided Design (CAD). Computer-aided manufacturing (CAM) was a well-developed industry which used computers to aid in production of metal parts. CAD was not more than an idea until CAM catalyzed its existence. Although CAD systems had been developed, graphical input data was cumbersome until Ivan Sutherland's sketchpad was developed in academia. These systems were almost all proprietary until AutoDesk brought CAD to the home market and onto the personal computer, streamlining the design process.

Halfway through the 20^{th} century, the United States was faced with two major international struggles: in June of 1948 the Berlin Blockade occurred, a major crisis of the Cold War; just two years later, the Korean war broke out in June of 1950. During these times it became imperative for the US to build an efficient military defense (Redmond and Smith, 2000, 1). A new type of defense gave promise on April 20^{th}, 1951, when a team of electrical engineers from MIT and radar engineers from Air Force's Cambridge Research Laboratories demonstrated how computers could be used to defend the continent from an airborne attack:

> For the first time, radar data on an approaching "enemy" aircraft had been sent over telephone lines and entered into an electronic digital computer, which had almost instantaneously calculated and posted instructions directing the pilot of a "defending" aircraft to his target (Redmond and Smith, 2000, 1).

The Semi-Automatic Ground Environment (SAGE) computer was used by the US Air Force.
(source: http://www.computermuseum.li/Testpage/IBM-SAGE-computer.gif)

This system was the world's first to use a computer to generate graphics. The US Air Force used this command and control system known as Semi-Automatic Ground Environment (SAGE) to display radar data and relevant information on an interactive *cathode-ray tube*. Information that traveled over a network between computers was then processed in real-time, enabling Air Force personnel to detect intruding aircraft and respond accordingly (Bozdoc, 2003, 1). SAGE was a system that later spawned what became the foundation for the rapidly-growing fields of computer networking and computer graphics.

3.6 Computer-Aided Design

The Boeing 707 aircraft began flight in 1954, and by 1958 flew commercially in the airline service industry, which then helped Boeing to design its now famous 727 aircraft. This meant that in succeeding years more and more industries would require large amounts of computing power (Sanders, 2008, 130). During that time the industry turned from using computers explicitly for design, to using them for numerical calculations and in manufacturing. Then in 1956, the Boeing Aerospace department acquired its first Numerical Computing (NC) machines, which where then used to produce fairly complex aluminum parts. By 1960, for the aerospace industry, these NC machines were "a part of life" (Sanders, 2008, 131).

At this time, the industrialized world was acquiring large super-computers, even though there was not a way to generate *renderings*. Then an engineer named of Norman Sanders who was at Boeing in 1959 describes how renderings were created: "All design drawings were made by hand; the definition of every motorcar, aircraft, ship, and mousetrap consisted of lines drawn on paper" (Sanders, 2008, 129). This meant that with every drawing created by hand, even minor changes incurred labor-intensive re-drawing; some changes required that a print be completely re-drawn from scratch. At this time computers were used only as an intermediary doing calculations for an engineer which were then drafted by hand.

Input was entered into computers via punch cards, and then at night would generate output via a printer. This data could then be used by an engineer to create a draft, a procedure that had obvious inefficiencies: "[t]here were cases of engineers spending three months drawing curves resulting from a single night's computer run" (Sanders, 2008, 130). This cumbersome method compelled Sanders to question whether or not computers could perform the task of drawing continuous lines and curves. But soon, with the help of Art Dietrich, Boeing sold their first software used to generate computer plots with printers. These plots, however, were far from accurate, and the Boeing drafting department refused to use them for design projects. To make matters worse, at this time Boeing would not invest any money in this type of research and development, because it wasn't efficient enough for their own use. In addition, no independent companies would engage in this venture because no market existed for it (Sanders, 2008, 132). However, the

world of *computer-aided manufacturing* (CAM) came to the rescue. Computers had already been used to operate milling machines that accurately cut metal in three dimensions. After a stroke of insight, in 1961, a Boeing design team replaced the cutter head of these machines with a small diamond scribe, which allowed these machines to plot in two dimensions on aluminum sheets. And "[p]resto! The computer had drawn the world's first accurate lines by machine" (Sanders, 2008, 132). However, as in any major engineering firm, the accuracy of this system required proof. So, in order to prove the competency of these drawings for the design of the Boeing 727, the project manager required that Norman's team generate 19 renderings of the Boeing 707's wing using this technology:

> We trucked the inscribed sheets of aluminum from the factory to the engineering building and for a month or so engineers on their hands and knees examined the lines with microscopes ... The chief engineer stood on his feet and said simply that the design lines that we had developed had been under the microscope for several weeks and were the most accurate lines ever drawn—by anybody, anywhere, at any time ... That is the closest I believe anyone every came to the birth of *Computer-aided Design* (Sanders, 2008, 132).

The success of this technology bound the Boeing 727 project to the computer not only for the manufacturing process, but for the design process as well: "We computed the definition of the outer surface of the 727 and stored it inside the computer, making all recourse to the definition via a computer run as opposed to an engineer looking at drawings using a magnifying glass" (Sanders, 2008, 130). To confirm the value of these developments, the Boeing 727 was completed on time and on budget, which "would not have happened had we not used CAD" (Sanders, 2008, 133).

3.7 Human Computer Interaction

Despite these breakthroughs, the industry had still not developed a practical means to input graphics, or any sort of graphical communication system for designing with computers. Then, in 1960, Ivan Sutherland used the TX-2 computer produced at MIT's Lincoln Laboratory to produce a system known as "Sketchpad", which is considered to be the first step towards the creation of the CAD industry (Bozdoc, 2003, 1).

This system was designed to be a "man-machine graphical communication system" that eliminates cumbersome input methods, such as manually typing coordinates and commands (Sutherland, 1988, 501). Instead of a complex, typed syntax, an intuitive "light pen" allows a user to interact immediately with Sketchpad by touching the pen to a monitor. Ronald Baecker recently wrote an article on human-computer interaction, and discusses the influential role that Sketchpad system had in the 1960's: "Sketchpad demonstrated the potential for effective computer-aided sketching and design through innovative concepts" (Baecker, 2008, 23). This potential inspired many other pioneering ideas within interactive computer graphics, and provided a working prototype for CAD applications.

Ivan Sutherland using the TX-2 computer to demonstrate "Sketchpad".

(source: http://design.osu.edu/carlson/history/images/ivan-sutherland.jpg)

3.8 The Internet

Joseph Carl Robnett Licklider, was a visionary and Internet pioneer whose contributions to computer science are great—a computer scientist from MIT who worked on SAGE, and later saw the value in connecting various communities together (Bisson, 2007, 1), is known for contributing his ideas to the formation of the Internet (Griffin, 1999a, 1).

At a time where computers were run using holes in punch cards, "Lick", as he asked to be called (Taylor, 1990, 5), had a better idea for how computers should work. In 1960 he wrote "Man-Computer Symbiosis", and talked about real-time communication between man and machine, and also modern topics that still remain somewhat unsolved such as speech recognition—he was ahead of his time.

It was in 1962 when J.C.R. was asked to take command of the Behavioral Sciences and Command and Control departments of the Pentagon's Advanced Research Projects Agency (ARPA) (Griffin, 1999a, 1), where his role was to manage a department to fund researchers to build new technology and teams. Lick had at his disposal a funding amount that at that time was greater than any other funded computer science projects (Taylor, 1990, 5).

Although Lick left ARPA, where he coined the name "ARPANET" (Griffin, 1999a, 1), it was shortly after

he left when ARPANET connected four computers together in 1969 (Bisson, 2007, 1). The chief architect of ARPANET, Larry Roberts, years later claimed that Lick was responsible for the ideas of the Internet, even though Lick wasn't present at ARPANET during the time it was implemented:

> The vision was really Lick's originally. None of us can really claim to have seen that before him norcan anybody in the world. Lick saw this vision in the early sixties. He didn't have a clue how to build it. He didn't have any idea how to make this happen. But he knew it was important, so he sat down with me and really convinced me that it was important and convinced me into making it happen (Griffin, 1999a, 1).

It is clear that Lick was an Internet Visionary. Written in 1968, "The Computer as a Communication Device" was another influential paper that Lick wrote a paper that envisioned the future of digital communication, in which he envisioned "[i]n a few years, men will be able to communicate more effectively through a machine than face to face" (Licklider et al., 1968, 26). Lick also wrote about shared computer programs and hinted at what today is know as the "cloud", and even online communities: "[b]ut let us be optimistic. What will on-line interactive communities be like?" (Licklider et al., 1968, 37).

3.9 TCP/IP

Before the World Wide Web came into existence, the Internet needed a way for computers to speak to one another. This is where Vinton Cerf, known to some as "the father of the Internet" (Cerf, 2009, 1), brought us *Transmission Control Protocol* (TCP) and *Internet Protocol* (IP), the standardized protocols that enables any two computers to communicate.

It was October 29th, 1969 when the first transmission was sent across the ARPANET between Stanford and UCLA (Cerf, 2009, 1), and by December that year the first four packet-switching nodes were up and running. The four nodes were located at Stanford Research Institute, UCLA, UCSB, and University of Utah (Cerf, 2009, 1).

Robert Kahn came to UCLA to help test the ARPANET, and with Cerf found that the ARPANET needed to handle transmission failures. Kahn specifically "foresaw the need to link together packet-switched networks with different designs so that any computers could communicate freely, no matter what the communication path" (Cerf, 2009, 1).

Then by 1973, Cerf and Kahn came up with the basis for transmission control protocol (TCP), and in May of 1974, they co-authored a paper "A Protocol For Packet Network Intercommunication", describing the protocol, which the basis our our Internet today (Cerf, 2009, 1):

> In particular, we have described a simple but very powerful and flexible protocol which provides for variation in individual network packet sizes, transmission failures, sequencing, flow

> control, and the creation and destruction of process-to-process associations. (Cerf and Kahn, 2005, 647).

Kahn and Cerf created the protocol on top of the technology that had been created at ARPANET. Their technology provided a platform upon which any computers that ran on different software could communicate. It also provided a mechanism to handle failures, which is extremely important due to instabile nature of computer networks. The point here is that they created a protocol, or on some level, a standard for computers to speak to each other. It is the standard that enabled the World Wide Web possible.

3.10 HTTP/HTML

After TCP/IP provided a standardized protocol for computers to communicate, *HyperText Markup Language* (HTML) provided a language to describe documents that can be transported via *HyperText Transfer Protocol* (HTTP). The beauty of these documents was that they could link to one another (Griffin, 1999b, 1).

> Neither the ARPANET nor the Internet and its constituent networks were purpose-built for particular applications. This design philosophy has allowed these systems to support a broad array of new techniques and applications including packetized voice, streaming video and, of course, the myriad applications built atop the World Wide Web, under the leadership of Tim Berners-Lee, first at CERN, the European particle-physics laboratory near Geneva, now at the Massachusetts Institute of Technology in Cambridge (Cerf, 2009, 1).

Tim Berners-Lee had been contracted to work at CERN, a particle physics lab. There he had created a program that used HyperText "for his personal use to help him remember connections between various people and projects at the lab" (Griffin, 1999b, 1).

A few years later, Tim took two *existing* technologies, HyperText and TCP/IP, and combined them to provide a way for people to share documents that can link to one another. He built what we know today as a web browser and a web server, which communicate via HTTP, which is the standard protocol for the Web today. This technology enabled people to share documents that could connect to one other. The first web browser he called "WorldWideWeb" (Griffin, 1999b, 1).

3.11 Web Browsers

Shortly after the first text-based browsers were in use, in 1992 Marc Andreesen and his co-worker Eric Bina started to build a browser called "MOSAIC". This was the first popular web browser, and a major factor was that it could display images inside of a Web page whereas other browsers could only show images as a separate file. (Griffin, 1999c, 1).

MOSAIC was the first web browser to embed images inside of documents.

(source: http://www.cedmagic.com/history/mosaic-1993.jpg)

This development led to what became known as "Netscape", which became the most widely used Web browser of its time. (Griffin, 1999c, 1). Within a matter of months, millions of people around the world began to use the Internet.

3.12 Evolution

As a civilization, the development of language and alphabets allowed us to communicate. After the printing press, a revolution in mass production of media and books advanced our education system and drove literacy rates up. As our accomplishments in technology expanded, the information age was born, and new digital tools accelerated the fields of design and engineering which impels great scientific fascination in its own right. Computer-aided design allows us to build the homes and communities in which we live. Computer graphics enabled us to have a view into computer processes that represent things in the physical world. HCI has provided us with many tools and perspectives for thinking about how we should interact with computers. We've reached the point in our society where the Web is an integral part of many peoples everyday lives.

Each of these events required adopting a system, or a definitive and agreed upon standard to enable such advancement. Combinations of ideograms progressed into alphabets, which formed from utilizing repeating characters which helped spawn languages. The printing press was built by combining two existing processes used successfully for other means. The Internet and family of technologies as we know it today

is a manifestation of a confluence of standards: TCP/IP provided a language for computers to speak. HTML provided a way for documents to speak to one another, and HTTP provided a way for a browser to communicate with a server. Each of these technologies has played a role in providing us the technological infrastructure of computers and systems as we know it today.

The various parts of history in computer science seem to converge on the presumption that utilizing standards for communication systems does indeed have positive consequences for the people utilizing those systems.

4 The Human Element

4.1 What Humans do with Computers

We simply cannot doubt our consciousness. We know qualia—internally accessed qualities that make up the phenomenal character of experience. Understanding the nature of consciousness is the Philosophy of Mind's underlying impetus for sagacious study of unanswered questions. At the crux of these questions lies the Mind-Body problem: What is the relation between the mind and the body? Is our existence completely physical, completely mental, or a confluence of both realms?

The personal computer was made with the vision of extending and improving the human mind and human condition: "the ideal of the then-hypothetical personal computer was a brain supplement, enhancing human memory and amplifying human reasoning" (Victor, 2006, 7). If computers can bestow such raw augmentation such that software we write can act as extensions of our minds, then code is a manifestation of ideas that can interact!

> One of the major themes of the past century has been the growing replacement of human thought by computer programs. Whole areas of business, scientific, medical, and governmental activities are now computerized, including sectors that we humans had thought belonged exclusively to us. (Knuth, 1973, 13).

Dr. Knuth delineates the fact that our lives are becoming enveloped by technology. Undoubtedly, you probably use a computer or smart device on a daily basis. Here are some thought-provoking questions for you:

1. does the computer expand your mind?
2. does the computer extend your physical life?

Your consciousness is the sum of the neurological patterns and circuitry that embody the knowledge you've accumulated over your lifetime. Computers facilitate the transmission of ideas and concepts that are

more complex than words. If software exists as representations of our consciousness, then through software we can express our ideas to others in ways that words may not have been able to otherwise.

> Every computer program is a model, hatched in the mind, of a real or mental process. These processes, arising from human experience and thought, are huge in number, intricate in detail, and at any time only partially understood. They are modeled to our permanent satisfaction rarely by our computer programs. Thus even though our programs are carefully handcrafted discrete collections of symbols, mosaics of interlocking functions, they continually evolve: we change them as our perception of the model deepens, enlarges, generalizes until the model ultimately attains a metastable place within still another model with which we struggle. The source of the exhilaration associated with computer programming is the continual unfolding within the mind and on the computer of mechanisms expressed as programs and the explosion of perception they generate. If art interprets our dreams, the computer executes them in the guise of programs! (Abelson and Sussman, 1996, 10)

4.2 The Human Visual System

The transmission of information is not limited to text or language—the human visual system has capabilities that allow us to absorb large amounts of information in parallel (Victor, 2006, 9). This channel for information thus implies that images are an important way for us to learn about our environment.

"To the thinking soul images serve as if they were contents of perception ... [t]hat is why the soul never thinks without an image" (Aristotle, 1855, 60). This quote from Aristotle describes a theory for how humans process information; simply, thought is not possible without images in his view. Humans harness an innate ability to process visual information—visualizations enable high-bandwidth communication of ideas between humans (Kamat, 2009, 309).

Beyond images are *diagrams*—a structured representation of a concept or object with graphical components laid out in a simplified way. The properties of diagrams present information in a manner that allows the human visual system to process the data in parallel (Smith and Ferguson, 2003, 2). There exists systems where a diagram greatly improves transmission, or may even be required, for example, in representing a data structure in a programming language such as Lisp (Smith and Ferguson, 2003, 67), or a the representation of a dynamical system in physics. To try and describe an environment diagram in a computer science class without a blackboard would be more than difficult.

> [P]roblem solvers in domains like physics and engineering make extensive use of diagrams, a form of pictures, in problem solving, and many distinguished scientists and mathematicians (e.g., Einstein, Hadamard) have denied that they "think in words" (Larkin and Simon, 1987, 2)

If images and diagrams are so important for the transmission of information, then it should be an important focus for the creation of academic works and documentation.

4.3 Design of Learning Systems

When approaching the design of a system designed to author content for learning, we have to best attempt to allow the author to freely express their ideas. The author should be able to communicate their vision with clarity and a way that allows for impact.

It has been shown that students learn more information through Web-based exercises than traditional tools (Mendicino et al., 2009, 12), so how can we build these systems? Previous attempts at studying the design of learning systems has proven its difficulty:

> It is difficult to establish a framework to symbolize and express learning contents design because the design of the target phenomenon, i.e. learning, is so abstract (Hayashi and Ikeda, 2004, 294).

Learning is too abstract! They key is to realize that *it is too abstract to abstract* into a graphical user interface. The author of a learning system should use a generalized framework to support the authorship of learning concepts, and thus, a programming language, or language of the sorts should be used directly by the author.

Their conclusion is a result of the fact that what they developed was simply *their own expression* of a system to design learning systems. For the last century, teachers and books have proven to work as a "learning environment". Any graphical interface is simply creating more distance between the idea of what needs to be expressed and the actual articulation. The existing instructional systems are working—if it's not broken, don't fix it. Language is what teachers and academics should be using to create books.

4.4 Some Principles of Design

Perhaps we've all played the game "telephone" when we were kids. By the end of the transmission, the sentence was completely changed form its original form! When building a system to facilitate the conduction of ideas between humans, it is important to cause as little distortion as possible—an important concept in design known as *direct manipulation* mitigates this issue.

> Here the critical issues involve minimizing the effort required to bridge the gulf between the user's goals and the way they must be specified to the system (Hutchins et al., 1985, 318).

Imagine our world sometime in the future, and digital textbooks are ubiquitous. However, the mathematicians and physicists who author these textbooks must outsource the design of the diagrams and interactivity to programmers and engineers. Is it possible for the goals of the author to met with accuracy?

> Alternately, a designer can draw a series of mockups, snapshots of how the graphic should look for various data sets, and present these to an engineer along with a verbal description of what they mean. The engineer, who is skilled in manipulating textual abstractions, then implements the behavior with a programming language. This results in ridiculously large feedback loops– seeing the effect of a change might take a day instead of a second. It involves coordination and communication between at least two people, and requires that the designer justify herself—she must convince the engineer and possibly layers of management that each change is worth the engineers time. This is no environment for creative exploration (Victor, 2006, 43).

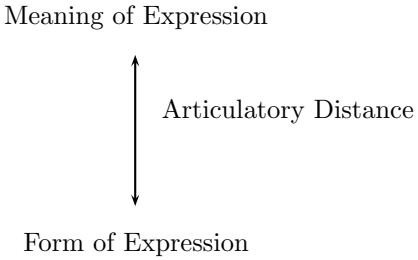
As Bret Victor's argument is laid out, this solution poses many issues, and ultimately people should render their own ideas if they want them to be expressed as close as possible to the representation in their minds.

> Historically, most interfaces have been built on the conversation metaphor. There is power in the abstractions that language provides (we discuss some of this later), but the implicit role of interface as an intermediary to a hidden world denies the user direct engagement with the objects of interest. Instead, the user is in direct contact with linguistic structures, structures that can be interpreted as referring to the objects of interest, but that are not those objects themselves. (Hutchins et al., 1985, 319)

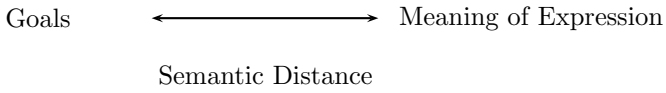
A famous paper describes to main important points regarding *semantic distance* (Hutchins et al., 1985, 321):

1. "Is it possible to say what one wants to say in this language?"
2. "Can the things of interest be said concisely?"

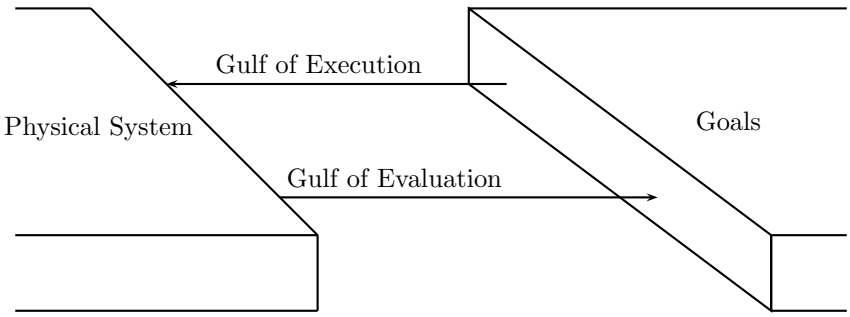
Below are graphics re-interpreted and reproduced version of the graphic from the paper:



"Semantic distance concerns the relation of the meaning of an expression in the interface language to what the user wants to say" (Hutchins et al., 1985, 321). The diagram below describes *semantic distance*,



The gulfs of execution and evaluation. Each gulf is unidirectional: The gulf of execution goes from goals to system state; the gulf of evaluation goes from system state to goals (Hutchins et al., 1985, 319).



We need to minimize the distance between the concepts in science and math and the articulation of these ideas, and especially these expressions on the Web and interactive devices. As technology advances, the technical knowledge required to produce interactivity is domain expertise to that of an engineer—to minimize the distance means enabling the professor or academician to express interactive ideas through an expressive language that is close as possible to the actual content they are describing.

4.5 Ethnographic Design

"[E]thnographers are interested in gaining an insider's view of a situation" (Blomberg et al., 2003, 967). We can use *ethnographic design* to build a system around the existing behavior of a "user", so that they can use the system in their natural state, and it adapts to their natural behavior.

When looking to build a digital system for expressing diagrams in the realm of mathematics, how can we adapt to their existing behavior? Given than the goal of this project is to support academics, using TEX as a basis makes sense from an ethnographic standpoint, given that it has been around for over 30 years and still remains the *de facto standard*. Now, not all of the academic population knows TEX, however, not all people are authors, either. It only takes one good book to propagate information to thousands of minds.

Going forward with a digital system using TEX as its syntax, it's of paramount importance to understand not only the language of TEX and its syntax, but also how it is structured. This can provide a technology that is engaging and productive for authors.

This system was genuinely designed to optimize development of structured documents in TEX. The structure was developed from my experience in authoring TEX. This project is just as much a project for other TEX people as it is for myself. I strive to be a part of my community, and I am a strong believer that we can use technology for learning. For example, I wrote a database application during my undergraduate years at Berkeley to categorize and store my entire lecture notes and output a structured version in TEX. I used this program to write a series of useful academic material—a 55-page reference manual for CS61A (The Structure and Interpretation of Programming Languages), a 35-page review for Math 55 (Discrete Mathematics), lecture notes for Math 54 (Linear Algebra), final and midterm papers for a series of liberal arts courses, and later used it to author over 300 pages of signal processing documentation for EE20 and EE120 in the EECS department.

Admittedly, much of this product was built while authoring textbooks within the platform itself (including *this* text). It was a symbiotic process where I stuck to using TEX as the standard, and even if it made programming more painful, I forced myself to implement it in a way that made sense for typesetting. In fact, the original diagrams were much more interactive in the Objective-C and JavaScript versions of the authorship interface than this current implementation, but they required the knowledge of programming! I believe the success of this platform solely lies on the adherence to TEX as the syntax for expression. My goal with this platform is to deliver tools that will accelerate education from both directions—the professors teaching, and the students learning—so that it stays in step with technology.

5 Standards and Languages

5.1 Ability to Express Ideas through Technology

Many of us think our thoughts using a language of some sort—there is usually some voice in our minds. Language in some ways, makes us who we are. Some even argue in the world of cognitive science that language is the foundation of our consciousness.

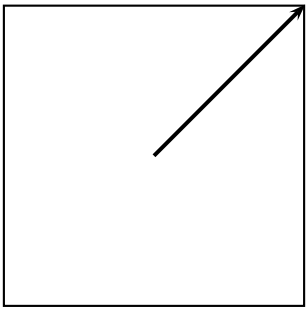
An author who has in their minds representations of intelligent concepts should be able to freely express herself through language with free association—digital expressions of these ideas in some cases requires total control of the computer and all of its processes.

The vision behind the personal computer was that any person could have full command of the functions of their device. I think this vision has come true to some degree, but not fully when it comes to creating graphics, especially mathematical diagrams online.

Does the common mathematician or professor have the ability to express concepts through web technology? The Web has its own language, and the goal of this project is to help blur the lines between what authoring the mathematical Web should be like and typesetting beautiful Math.

If you know LaTeX, then get ready to author interactive diagrams in real-time (try using mouse or touch to interact with diagrams).

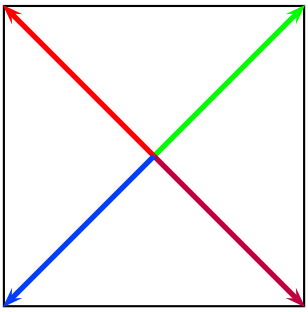
What matters most is minimizing the distance between our expression and our execution of an idea. For example, I can describe a vector at $(0, 0)$ and initial value of the head at $(2, 2)$ that will follow a user touch or mouse event. This will produce the following interaction (you seem to be reading the .pdf, so it will not be interactive! this page is interactive at this url: https://mathapedia.com/books/31/sections/155/405):



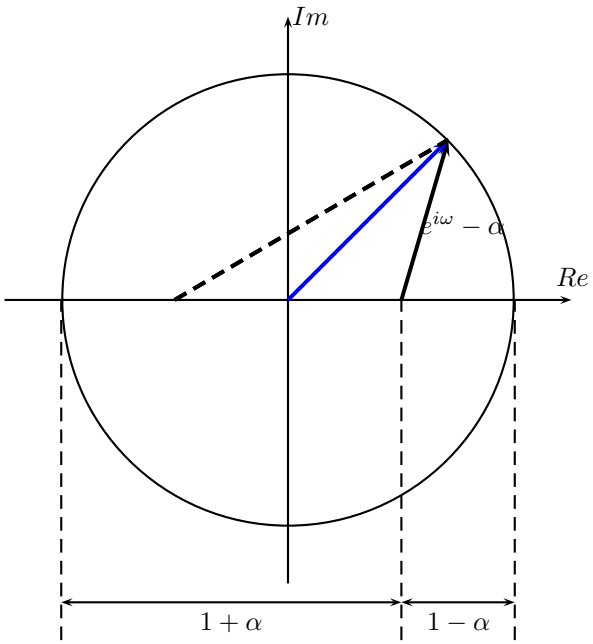
This was as easy as using this TeX, which many math professors could understand.

```
\begin{pspicture}(-2,-2)(2,2)
\psframe(-2,-2)(2,2)
\userline[linewidth=1.5 pt]{->}(0,0)(2,2)
\end{pspicture}
```

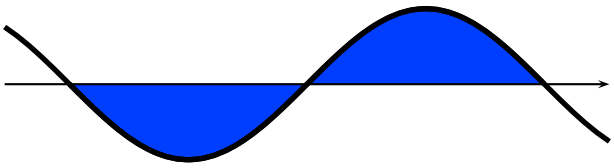
You can start to add more lines, if you wish:



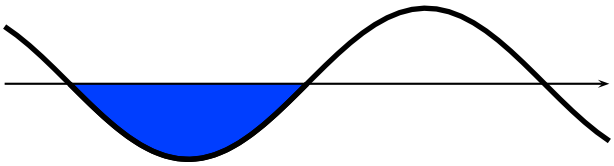
I can also draw a more complex version, and start to make more useful diagrams to describe vectors:



Think about the student learning a new concept. Here is an example of teaching integration using "area under the curve", for example, you can represent $\int_a^b f(x)dx$ as:



The next example allows you to move the height of the graph and also integrate over a moving interval.



5.2 TeX: Art and Technology

In 1978, Donald Knuth started to work on TEX, which has revolutionized digital typesetting. He was working on his book "The Art of Computer Programming" when he was frustrated with the results and was "shocked at the poor quality" (Hefferon, 2012, 1).

I want to take a moment to bring to light two things. First, the pronunciation of TEX (and therefore LATEX), as it is an often debated in many university classrooms and libraries, and second, the origin of the word $\tau\epsilon\chi$, because you can begin to understand the vision and purpose of Donald Knuth's creation:

> English words like 'technology' stem from a Greek root beginning with the letters $\tau\epsilon\chi$...; and this same Greek word means art as well as technology. Hence the name TEX, which is an uppercase form of $\tau\epsilon\chi$.
>
> Insiders pronounce the of TEX as a Greek chi, not as an 'x', so that TEX rhymes with the word blecchhh. Its the 'ch' sound in Scottish words like loch or German words like ach; its a Spanish 'j' and a Russian 'kh'. When you say it correctly to your computer, the terminal may become slightly moist.
>
> The purpose of this pronunciation exercise is to remind you that TEX is primarily concerned with high-quality technical manuscripts: Its emphasis is on art and technology, as in the underlying Greek word (Knuth, 1986, 1)

His purpose was to create *beautiful* documents for expressing technical concepts and ideas using TEX, and the origins of TEX are in both *art and technology.*

Today, and for the last 30 years, it's been the de facto standard for documenting the sciences. Most scientists and academicians know TEX, and use it to publish there academic work and papers (Hefferon, 2012, 1).

5.3 LaTeX: TeX evolved

It is widely known in the academic world that TEX is the *standard de facto* language for authoring academic documents.

LATEX is a typesetting language that evolved from TEX, and provides a higher-level abstraction so the author can focus on writing.

You can do things like **this** and *this*.

```
You can do things like {\bf this} and \emph{this}.
```

Mathematics has great uses, even for the non-math major. Math develops analytical and critical thinking which benefit lifelong problem solving. These skills are of paramount importance in our society, regardless of field or profession—the ability to break any situation down into its elements, and determine how each variable is related.

Learning math is heavily depending on mathematical text, and thanks to MathJAX, we can render LATEX equations in the browser. Here is an example

```
$$\int_0^\infty H(\omega)e^{i\omega x} dx$$
```

which produces

$$\int_0^\infty H(\omega)e^{i\omega x}dx$$

more info on TEXsupport here (read carefully!) http://docs.mathjax.org/en/latest/tex.html

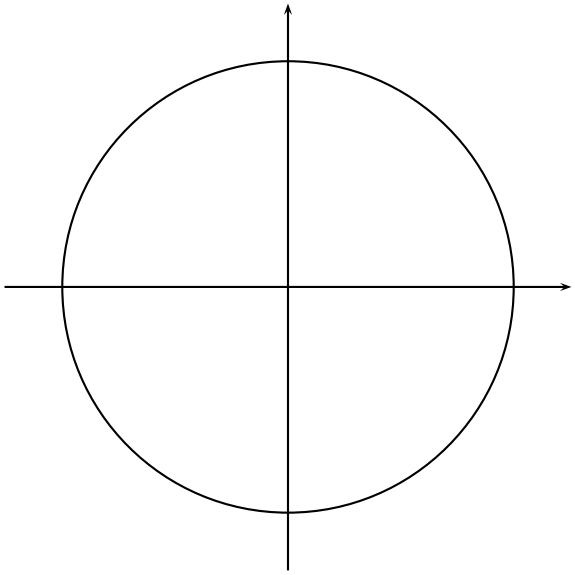
5.4 PSTricks: Graphics and Diagrams

Often scientists need to convey information in a non-textual way to express certain ideas that otherwise would be difficult to communicate. Using TEX alone would be difficult, and out of that need packages like TikZ and PSTricks came to life.

For the Web, there are many solutions, but none that provide client-side rendering of diagrams for packages like PSTricks in an analogy to how MathJAX has done for LATEX.

Because LATEX is so popular, PSTricks became wide-spread and many academics use it to produce diagrams in academic text. Also, many programs generate PSTricks as well. Note that this platform could have supported TiKZ, but I chose PSTricks because of the difference in verbosity, and wanted to use a higher-level of abstraction. TiKZ support would be a great addition in the future!

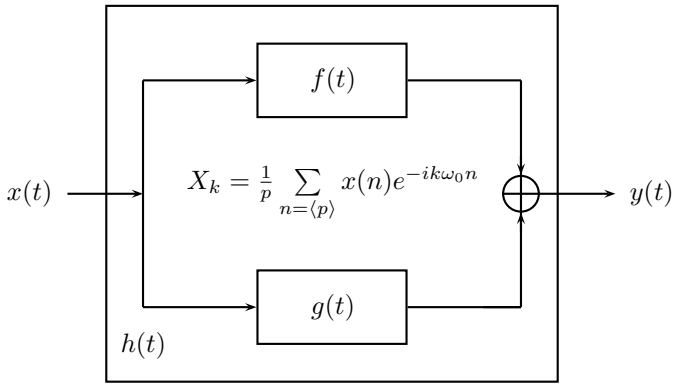
Let's get to the point. The core of PSTricks is graphics!



which can be produced using the following TEX:

```
\begin{center}
\begin{pspicture}(-5,-5)(5,5)
\psline{->}(0,-3.75)(0,3.75)
\psline{->}(-3.75,0)(3.75,0)
\pscircle(0,0){ 3 }
\end{pspicture}
\end{center}
```

But more importantly, and probably the best part of using such a package is that you also get to mix beautiful mathematics with a consistent look for your diagrams. This can be done using *rput*:



which can be produced with the following LaTeX (hint: try to copy and paste it into the sandbox):

```
\begin{center}
\begin{pspicture}(0,-3)(8,3)
\rput(0,0){$x(t)$}
\rput(4,1.5){$f(t)$}
\rput(4,-1.5){$g(t)$}
\rput(8.2,0){$y(t)$}
\rput(1.5,-2){$h(t)$}
\psframe(1,-2.5)(7,2.5)
\psframe(3,1)(5,2)
\psframe(3,-1)(5,-2)
\rput(4,0){$X_k = \frac{1}{p} \sum \limits_{n=\langle p\rangle}x(n)e^{-ik\omega_0n}$}
\psline{->}(0.5,0)(1.5,0)
\psline{->}(1.5,1.5)(3,1.5)
\psline{->}(1.5,-1.5)(3,-1.5)
\psline{->}(6.5,1.5)(6.5,0.25)
\psline{->}(6.5,-1.5)(6.5,-0.25)
\psline{->}(6.75,0)(7.75,0)
\psline(1.5,-1.5)(1.5,1.5)
\psline(5,1.5)(6.5,1.5)
\psline(5,-1.5)(6.5,-1.5)
\psline(6,-1.5)(6.5,-1.5)
\pscircle(6.5,0){0.25}
\psline(6.25,0)(6.75,0)
\psline(6.5,0.5)(6.5,-0.5)
```

```
\end{pspicture}
\end{center}
```

5.5 HTML5: The Living Standard

It can be argued that the most prevalent programming language to date is JavaScript, given the fact that billions of web pages exist, and as we approach the closing of the HTML5 standards in 2014, HTML is becoming the computing platform for the future.

When Tim Berners-Lee created the "WorldWideWeb" he was also thinking about the political and corporate issues that could potentially harm the web, and he helped form the World Wide Web Consortium (W3C), whose purpose is to "lead the Web to its full potential" (Griffin, 1999b, 1).

The W3C works to create an open standard and specification and keep moving the Web forward in a democratic way, adding new features that it believes are conducive to the achieving the Web's full potential. Soon we will see full access to the device sensors and actuators on mobile devices, although these specifications are actually recommendations. Soon the lines between what we call Web vs. Native will begin to blur.

There exist myriad disparate languages out there to create "native" apps, however soon all functionality will be accessible within HTML5 as well. HTML5 is the democracy of programming. HTML is a survivor like Lisp was: "Using Lisp we restrict or limit not what we may program, but only the notation for our program descriptions" (Abelson and Sussman, 1996, 10). The Abelson and Sussman book is considered by some to be one of the most important books in computer programming. Lisp could not limit what is possible to program in the context of a computer programming book that covers every possible software idea, mostly because Lisp provided access to higher-order functions that can be described by lambda calculus, and functional and object-oriented programming patterns. JavaScript is just as flexible as Lisp, and also has access to the document object model (DOM), which is the user interface, or representation of elements on a Web page.

JavaScript, which is a part of the HTML5 specification, is in the family of *evented languages*, which " have been successful because they map well to the direct manipulation graphical user interface" (Myers et al., 2000, 6). JavaScript is inherently tied to the DOM (unless you are using node.js), which provides a way to express a user interface along with functionality. Other languages (python, etc) are still stuck in the terminal, while HTML5 is readily available to the non-technical person!

In general, the WWW has lowered the barrier for authorship and consumption on the Web (Myers et al., 2000, 9). However, with the growing complexity of the Web, things like CSS and JavaScript are not necessarily something that ordinary people understand.

5.6 From Digital to Paper

Many people in the vast conduits of the interwebs talk about creating cross-platform frameworks. What happened to paper? Is it not a valid "device" anymore?

> THE POSTSCRIPT LANGUAGE is a simple interpretive programming language with powerful graphics capabilities. Its primary application is to describe the appearance of text, graphical shapes, and sampled images on printed or displayed pages (Incorporated, 1999, 8)

In other words, *PostScript* is the language that your printer speaks (although your display can, too). PostScript is responsible for revolutionizing digital publishing, mostly because it provided an integrous communication channel between a computer and a printer. Therefore, since its introduction in 1985, it has become the industry page description language standard (Incorporated, 1999, 8).

Note that *Portable Document Format* (PDF) and PostScript can be converted to one another interchangeably, except that a PDF is a static document, whereas PostScript has an underlying general purpose programming environment (Incorporated, 1999, 9).

PostScript is a very, very low-level language that you would never want to write yourself, luckily we have things like TEX out there to help us! Embracing TEX is so important for the digital world because it allows us to conform to the standard for page description language—to be truly cross-platform, a system should work with every device, including paper!

5.7 The Convergence of Standards

The Web is now becoming an even more global language than English, thus, a new form of Literacy is beginning to exist. The amount we of time we spend consuming information is probably less on paper and more digital as time progresses, however, not everyone can author Web-based content.

TEX has been the academic publishing standard for the last 30 years, and HTML has been the standard since its birth over 20 years ago. The synthesis of the two languages provides a road map for how to build a robust mathematical platform that has the potential to enable many new authors to enter digital publishing.

The Web has myriad number of expressions which have been enabled by the creation of CSS3, and it is also this flexibility that causes the enormous eruption of attempts at creating "eBooks"—but it does have a *strict functional specification*. TEX is somewhat limited in design, yet provides us a bridge to PostScript, which enables the production of not only beautiful digital pages, but also print. If we use TEX and PSTricks as a standard for how things should "look", and HTML for how things should "function", then we can construct a system with full functional and design specifications, already complete. Now it's just a matter of implementation.

6 The Mathematical Web

6.1 Rendering Math

The rendering of online mathematics has been a struggle to say the least. There have been advancements made in recent years, but for some professors, the Internet still remains a poor fit for mathematics:

> It has been an extremely frustrating experience teaching math online... [t]his aspect of my internet course is the most discouraging. I find myself spending an exorbitant amount of time responding to students mathematically [because the infrastructure does not support it] and not enough time teaching. If this doesn't change, it will eventually be the reason why I give up teaching mathematics on the internet. (Smith and Ferguson, 2003, 2)

6.2 Server Side Rendering

In the beginning there was server-side rendering, providing online math as a Web service. That meant that the LaTeX was sent from the client to the server, where image files were generated and then served back to the client for display within an `img` tag in an HTML document.

I attempted this quite some years ago with some success. I utilized the "textogif" command on a server and enabled real-time AJAX communication to dynamically serve generated images. The results were similar to that of MathTRAN, another server-side rendering solution.

A major downside of this is that the server has to generate files and serve them, and the client must make HTTP requests for every single equation, which is very inefficient.

Another issue with MathTRAN was that the syntax was verbose. You had to set an HTML element's attribute to the LaTeX code you wanted rendered, so it was a very disconnected editing experience, since you had to combine HTML and TeX.

MathTRAN was some sort of TeX and HTML:

```
<p>The equation of the unit circle <img alt="tex:x^2+y^2=1" /> and
  a related trignometric identity <img alt="tex:\sin^2\theta +
  \cos^2\theta = 1" />.  </p>
```

This type of authorship interface cannot put an author into a "flow". HTML, as much as I love it, is starting to resemble a sort of assembly language.

6.3 Equation Editors

In addition to TEX rendering, there are also the equation editors that exist out there, which provide a graphical user interface for generating equations. Although the market is potentially larger, because fewer professors know LATEX than those who do, it is typically difficult to modify or copy the equations that are generated, since they are graphical (Smith and Ferguson, 2003, 6).

6.4 MathML

The W3C came up with a recommendation called MathML, which is a form of XML used to describe mathematical symbols online. Unfortunately, support for MathML is low and it is extremely verbose.

For example, here is an equation $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, that is rendered in TEX like so:

```
x=\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
```

and rendered in MathML like this (source wikipedia):

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
    <eq></eq>
    <ci>x</ci>
    <apply>
      <frac></frac>
      <apply>
        <csymbol definitionURL="http://www.example.com/mathops/multiops.html#plusminus">
          <mo></mo>
        </csymbol>
        <apply>
          <minus></minus>
          <ci>b</ci>
        </apply>
        <apply>
          <power></power>
          <apply>
            <minus></minus>
            <apply>
              <power></power>
              <ci>b</ci>
```

```
            <cn>2</cn>
          </apply>
          <apply>
            <times></times>
            <cn>4</cn>
            <ci>a</ci>
            <ci>c</ci>
          </apply>
        </apply>
        <cn>0.5</cn>
      </apply>
    </apply>
    <apply>
      <times></times>
      <cn>2</cn>
      <ci>a</ci>
    </apply>
  </apply>
</apply>
</math>
```

Pretty obvious why it didn't really catch on. However, probably a bigger problem with MathML is that it cannot be used to create diagrams (Smith and Ferguson, 2003, 686).

6.5 Other formats

Many people resorted to using third-party plug-ins in browsers such as Java applets within web pages to make them interactive or mathematical. These have numerous problems. The first is that the browser blocks these plug-ins these days as a security feature, and the install process is not always intuitive or successful.

Other formats exist that are more promising, such as Wolfram's computable document format, however, you still have to download it! We need to look at Adobe's flash as a metaphor here. It's dead. Browsers are ready, are you?

6.6 MathJAX

Organizations including the American Mathematical Society, Design Science, Inc., and the Society for Industrial and Applied Mathematics have provided major funding to produce what is known as MathJax. It

is a way to create beautiful math on all browsers. It is a simple JavaScript library that you can include on your site, and you can type LaTeX and it will handle all the rendering for you.

MathJax, however, only does math (although it does it extremely well!). There still needs to be additional work done to create diagrams and also general typesetting via TeX syntax online.

6.7 Diagrams

So there are a number of JavaScript libraries out there that render amazing graphics. However, not a single one does for diagrams what MathJax does for math formulae.

Even if they make it easy, things like D3, or paper.js, people that don't code the Web simply won't use even the simplest JavaScript libraries. That's why it's important to take an enthographic approach to design.

In the end, there are no solutions for blending mathematics and diagrams together using TeX online. Seems as though instructors out there are still waiting for something that works:

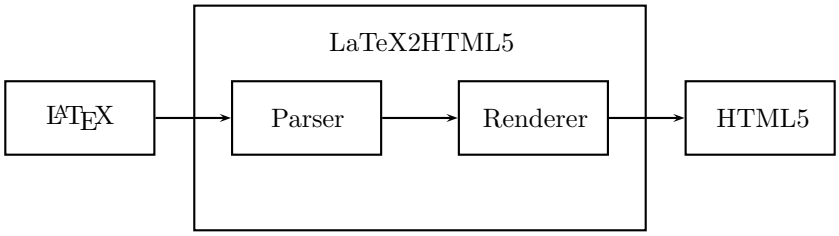
"Either way, since mathematics is emphasized as an educational priority for many developed countries, one would think the longstanding woes of online mathematics instructors would shortly be solved. However, like the long overdue artificial intelligence (AI) revolution which never arrives, most online mathematics instructors still wait for a simple and convenient way to communicate two-way with their students in the very language of mathematics." (Smith and Ferguson, 2003, 6)

7 LaTeX2HTML5

7.1 How it works

The primary concern of this platform is rendering on the Web, TeX already handles rendering to PDF and paper. The primary duties include controlling the DOM and DOM events, and also managing and multiplexing environments into their respective rendering hierarchies

The platform calls MathJAX on the pieces it wants to, but not on all parts, and also prepares TeX items for MathJAX that it would not normally render without some modification.



The parser generates the various environments after parsing the TEX, and the renderer manages views that generate HTML, CSS, and JavaScript to create interactive content on the fly.

This is done completely in the browser, no server-side rendering is required. The server is purely used to store LATEX in a database.

7.2 Expressions and Parsing

The code was designed to have extendability, and uses a basic parsing and evaluation model. Every command has a hash that references a regular expression in the `Expressions` object defined in `expressions.js`, and a function in the `Functions` object also defined in `expressions.js` for what to do once the expression matches.

To add new commands, you can simply add a new regular expression inside the `Expressions` object, and a match function in `Functions` object of the same hash name. If you defined a new command to be evaluated within a PSTricks environment, then you should also add a graphing function inside of `psgraph.js`. If you want to add a new environment, then you can add a regular expression to the `Delimiters` object in `expressions.js` and add a new block inside of the `renderer.js` file's `beforeRender` method.

The parser located in `parse.js` chops and dices all text into an array of lines using the newline character as the delimiter. This is a very simple model for parsing, as it enabled focus to be on the commands instead of writing a full grammar. Writing a full parser would be ideal for future versions to be able to properly parse multi-line environments that use opening braces '' and closing braces '' on separate lines.

After the parser creates an array of text it passes all the lines through the defined regular expressions, first parsing any text-based expressions primarily for TEX commands, and then parses more advanced expressions such as PSTricks variables and expressions as they are defined within their respective environments.

The parser instantiates and manages environments during parsing, and creates new environments packing them with state when any delimiters are detected, which are also defined as regular expressions inside of `expressions.js`.

Each environment has its own execution context, and a hash of matched objects that map to TEX

commands. It also stores the intial settings, which can be modified during evaluation. When the match functions are called, they are given the context using the JavaScript `Function.call` passing the context to be evaluated for `this` object. The reason for this is to properly address coordinate transformations, which requires access to the current environment's variables and context and is at the core of the platform.

Certain expressions can modify the current environment variables and settings, so that during evaluation, the graphics and text can update accordingly. Some commands that modify the environment are `pspicture` and `psset`, because they define the size of pictures and units or dimensions.

7.3 Rendering and Graphing

The renderer contains a set of view objects that inject themselves with subviews that are rendered in specific ways depending on the environment. After the parser is complete, it then passes the completed objects into the renderer's main view, called `TEX` located inside of `renderer.js`.

Upon initialization of the `TEX` view, it stores the current scroll position of the document, to enable flawless real-time editing upon re-render, which we will discuss more in detail later. Once the `TEX` view's `render` method is called, this triggers the `beforeRender` method, which multiplexes the injection of various views depending on the environment, and passes in the appropriate contexts.

It is within these subviews where the magic happens. The view associated with a `verbatim` environment will wrap the text in `pre` tags, for example, and the view associated with the `enumerate` environment will iterate over the items and generate the HTML elements `ul` and `li` to render lists.

The views have access to their representation in the DOM via a variable `this.el`. The math environment will simply make a call to MathJax in its `afterRender` method. This method means that `this.el` exists in the DOM, thanks to the layout manager. The view specifically calls the MathJax `Typeset` method for *only* that DOM element. This controls the rendering so that we do not attempt to render the entire page, which we do not want. It is this fine grained control that enables the system to function as expected.

The more advanced view is probably for the PSTricks environment, however. Upon initialization, the PSTricks view will store a reference to the environment and execution context that was passed in. It then creates a new SVG element based on these parameters, and then registers touch and mouse events that call a user event function to trigger re-rendering of various graphical elements when the user interacts with the SVG.

In the PSTricks view's `afterRender` method, it iterates over all objects that are not `rput` first, rendering those using the graphing function that matches the hash inside of the `psgraph.js` file. For example, a `psline` expression that may be defined in the `Expressions` object inside of `expressions.js` corresponds to a `psline` function defined inside of `psgraph.js`. So the corresponding graph function is called, and the context of the environment is passed along, as well as the reference to the SVG element.

The various functions within the `Functions` object inside of `psgraph.js` contain the drawing commands for the respective PSTricks commands. Most employ turtle drawing to create `svg:path` elements using notation such as:

```
M 0 150 L 0 3 4 5 6 6
```

What this means is move to position `(0,150)`, then draw a line to `(0,3)`, then draw a line to `(4,5)`, and so on. Then the associated styles and settings are applied, for example, if there is a fillcolor or a stroke width specified.

Before the `afterRender` method is complete, the `rput` elements are "absolutely relatively" positioned on top of the `this.el` which is enabled because it's parent's `width` is set, and it s parent's `position` is set to `relative`. The `rput` method defined in `psgraph.js` generates a new `div` element, and sets the CSS properties to be `position:absolute`, and `top` and `left` properties mapped `this.x` and `this.y`, which are the transformed coordinates specified form the original `rput` command written in text. The second to last step is calling the MathJax `Typeset` method on the DOM element for math rendering.

Because `rput`'s definition is that it is centered at the specified position, and we require the MathJax rendering to be complete to get the width of the resulting div element, we use a `setTimeout` of 0 milliseconds, which is a trick to get access to the DOM element during the next repaint of the browser. Think of it as an `afterAfterRender` method. If this code could be included in the MathJax library, perhaps this step would be unnecessary. It is in this critical step where we get the width and height of the div, and offset the `top` and `left` CSS parameters by half of the height and width respectively.

In totality, this system of managing the rendering of DOM and SVG elements enables us to fully control the rendering of math and any related graphics or diagrams, so that they can be rendered on the Web, and match those graphics that are output to PostScript, PDF, or paper.

7.4 Coordinate Systems

A core component of the platform is coordinate transformations. TEX has its own coordinate system, as well as the `pspicture` and `psset` commands, which can modify the coordinate systems. The Web has a system for describing the DOM elements on the page, and even the SVG element that we use to render graphics has its own local coordinate system for describing the graphical elements that it contains. Touch and mouse events from user input are associated with these coordinate systems, and it is the harmonious transformation between the various coordinate systems that enables the flawless rendering of mathematical diagrams and figures.

A change of coordinates is required to map PSTricks code to the SVG coordinate system, as well as inverse transformations to go from SVG back to PSTricks in the case of user mouse or touch events.

If you recall, we store an execution context within each environment. This also contains references to the scale of the current environment, which can be manipulated using `pspicture` and `psset`.

The transformations are as follows:

Let X be the transformation function that maps a x-coordinate v from PSTricks $\rightarrow$ SVG:

$$X(v) = (w - (x_1 - v))\, x_{unit}$$

Let X^{-1} be the transformation function that maps a x-coordinate v from SVG $\rightarrow$ PSTricks:

$$X^{-1}(v) = \frac{v}{x_{unit} - w + x_1}$$

Let Y be the transformation function that maps a y-coordinate v from PSTricks $\rightarrow$ SVG:

$$Y(v) = (y_1 - v)\, y_{unit}$$

Let Y^{-1} be the transformation function that maps a y-coordinate v from SVG $\rightarrow$ PSTricks:

$$Y^{-1}(v) = y_1 - \frac{v}{y_{unit}}$$

The values `this.x1`, `this.y1`, `this.xunit`, `this.yunit`, `this.w`, `this.h` are associated with every PSTricks environment. The x_{unit}, which corresponds to `this.xunit` is the amount the diagram gets scaled

in the x-axis. The same true for y_{unit} in the y-axis.

Note that if an author uses the `psplot` environment with any interactivity, we must first apply an inverse transformation, apply the function, and then apply the transformation function to get the accurate values.

The two X transformation functions utilize `this.w` which represents the width of the graphic which is determined by `psset` and `pspicture`. However, because of the CSS positioning, the height is irrelevant in these calculations.

7.5 Privacy and DRM

As soon as information is digitized today, it becomes a sequence of bits that by the nature of the operating system can be copied very easily. To address the digital rights issues associated with digital publishing, this platform has taken a few steps.

Every page is dynamically generated after the first HTML page is served to the client. The JavaScript that is included in the HTML page generates a series of interface components on the client side to render the page. If somebody tried to view the source of a page, they would not get any information about the book, whatsoever. They also do not get any access to the LaTeX.

You would have to know quite a bit about networks to get access to the source of the book. Every page is sent to a client via *Secure Sockets Layer* (SSL), a cryptographic protocol, over a Secure HTTP connection (HTTPS).

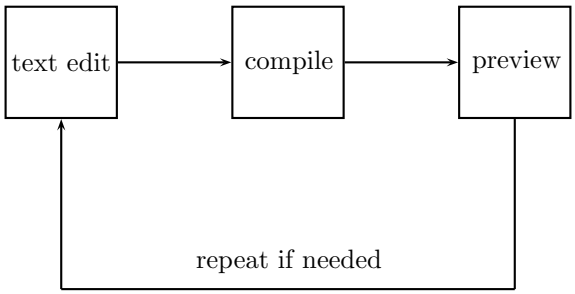
Last but not least, authors can make their writing public or private. Private books will only be viewable by the author themselves, or by any authenticated users that have permissions. Readership and authorship permissions are stored in the database, and enable an author to create a white-list of readers or collaborative authors.

7.6 Optimizing the Editing Flow

Often, rendering graphics is a very disconnected process from the editing of the TeX source. First, the author must write or edit a .tex file, that contains the source to generate the graphic, usually describing the digram using PSTricks or TikZ. The author then runs a series of commands to generate a PostScript or PDF file, which they can then open to preview the results. For example, to render a latex document often requires use of terminal and you may use these commands:

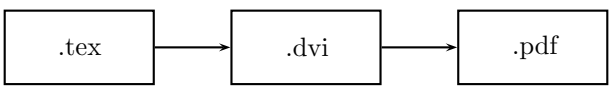
```
latex paper.tex
dvipdfm paper.dvi
xpdf paper.pdf
```

That required *three* steps in order to view the actual graphic. But that is not the only step required by the user. They also have to context switch between the text editor and the terminal and the graphics display program. The process then becomes more fragmented. Here is a graphic where each arrow represents a context switch:

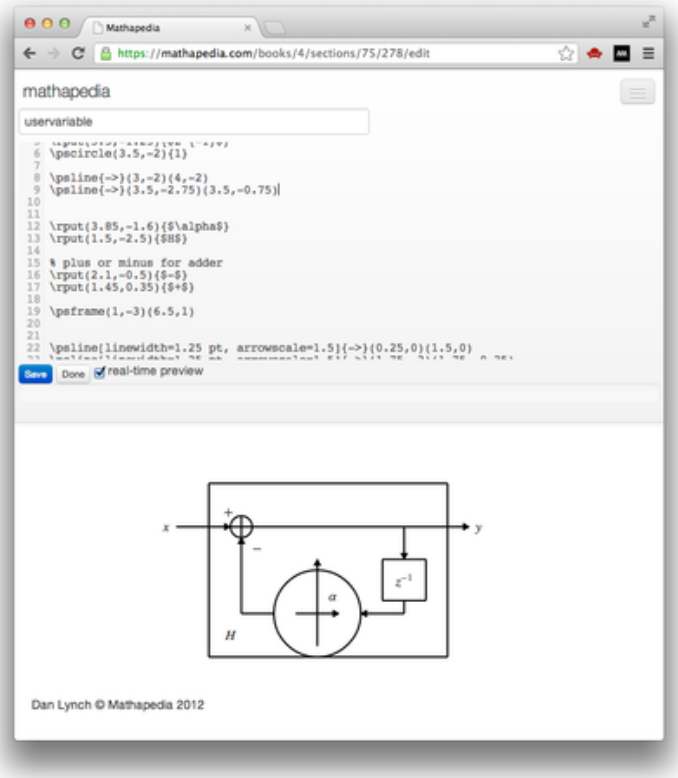


The user has a series of steps within these contexts:

1. switch to text editor
2. edit the .tex
3. switch to terminal
4. run `latex` command on .tex file
5. run `dvipdfm` command on generated .dvi file
6. open preview the pdf
7. if the graphic is incorrect, repeat



This process takes anywhere from seconds to minutes. If the user must make a change, they have to repeat the entire process which can be frustrating. How can we minimize amount of time and space it takes to render a document? Enable real-time editing and preview.



This platform provides an editing experience where the text editor and preview are on the same page, hence, updates appear in real-time within the same context. The real-time authoring aspects allow a user to get instantaneous feedback while creating these diagrams, and making changes becomes simple and the results are displayed right away.

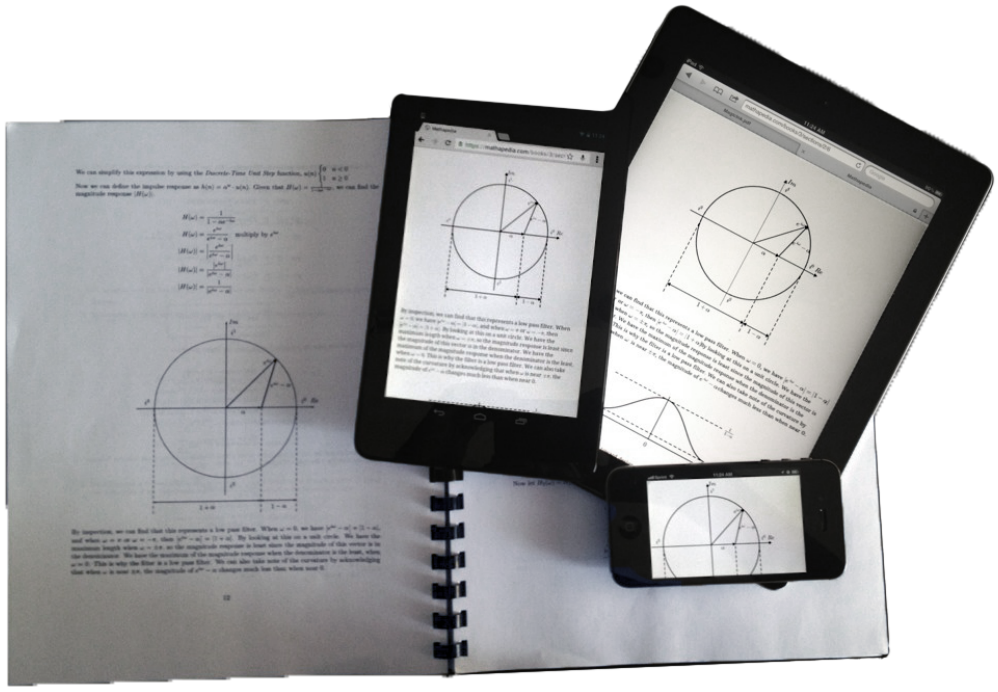
The user interface continuously evolved. At first, there was a text box that would scroll off the page as you scrolled down the preview content. It make it extremely hard to edit continuously. Then I created a fixed position text area, but then every re-render initially caused the page preview to scroll to the top of the page, so to mitigate this, at the initialization of a `TEX` view, the platform stores a reference to the scroll height and repositions the window accordingly, so the user doesn't have to re-scroll the window to the point of interest. In fact, the experience is seamless now. You can scroll through the text and the preview independently on the same page while editing in real-time.

Another issue was using `keyup` events to trigger a re-render. Every time the user would type in the text editor, the page would re-render sometimes causing a lag for larger pages. A way to fix this was to use throttling. The framework stores a reference to an id that points to a callback function supplied by a `setInterval` function that has a value of 500 milliseconds. Basically the system then only calls the function every half second, and only then. If a new `keyup` event occurs, it only adds a callback if there isn't a reference to one already. This ensures a streamlined editing experience for the author.

7.7 Results

The purpose is to create a *beautiful* pages, whether on paper or the Web

Using two standards, HTML5 and TEX, we were able to produce a consistent look and feel across all devices, including paper:



8 TeX Implementation

8.1 structure

8.1.1 sections and subsections

As an avid LATEX author, I've mapped the data structure of a typical document to the database structure that the books are based upon.

This can be viewed as a feature or a flaw, but the user of this particular system doesn't have to type `section` or `subsection` commands. Users can use the UI to partition their work.

When the user outputs a .tex file, the `section, subsection, and subsubsection` commands are automatically put into the .tex file for the user, if they decide they want to export their book.

The philosophy is to quickly generate the structure of a paper, and then just focus on the sections. Then an author can manage a larger paper consisting of hundreds of sections if needed.

This also played into the navigation quite well. The pages on mobile devices initially caused a JavaScript execution timeout, where the script was running too long and the operating system cut the script off for security reasons. Thus, dividing pages into sections also enabled a good construct for rendering portions of a book onto smaller mobile and touch devices.

8.2 elements

8.2.1 accents, styles, and typeface

Directly from "The TeXbook":

> switches to the normal "roman" typeface
>
> switches to a slanted roman typeface
>
> switches to italic style
>
> switches to a typewriter-like face
>
> switches to an extended boldface style
>
> (Knuth, 1986, 13)

Example. You can **see** what it's *like* to use `different` types of *styles* within your words

```
You can {\bf see} what it's {\sl like} to use {\tt different} types of {\it styles} within your {\rm wo
```

8.2.2 quotes

Quotes begin with two ‘ symbols, and end with two ’ symbols.

Example. Quotes are often uses in literary work, and can be "added to text" very easily.

```
Quotes are often uses in literary work, and can be ‘‘added to text’’ very easily.
```

8.2.3 hyphens and dashes

for a hyphen, type a hyphen (-)

> for an en-dash, type two hyphens (–)
>
> for an em-dash, type three hyphens (—)
>
> for a minus sign, type a hyphen in mathematics mode (−)
>
> (Knuth, 1986, 4)
>
> This can be created using the following TEX:

```
for a hyphen, type a hyphen (-)
for an en-dash, type two hyphens (--)
for an em-dash, type three hyphens (---)
for a minus sign, type a hyphen in mathematics mode ($-$)
```

8.3 math

8.3.1 the beauty of mathematics

Math has provided us a method for which we can **derive** through proofs and create theorems, and corollaries or even Laws.

This section describes some details about what you can do with MathJAX and LaTeX2HTML5!

$$\iint_D dx\,dy$$

$$\int\int_D dx\,dy$$

$$\frac{x+y^2}{k+1+2}$$

$$\binom{n}{k}$$

$$\sum_{n=1}^{m}$$

$$\int_0^{\frac{\pi}{2}}$$

$$\int_{-\infty}^{+\infty}$$

8.3.2 proof

You can write a proof using `proof` environment.

The platform will automatically apply the Q.E.D symbol at the end of your proof, and also make a nice bold **Proof** at the beginning of the proof. In case you are curious, Q.E.D. comes from Latin, which means that that was asked to be demonstrated, or *quod erat demonstrandum*. The mathematical symbol looks like this: □.

You can write a proof using this notation:

```
\begin{proof}
 % some typsetting + math goes here
\end{proof}
```

Here is an example that proves that multiplication in the time domain is the product of a scalar and the convolution in the frequency domain. In other words, prove $f(t)g(t) \quad \underset{\longleftrightarrow}{\mathcal{F}} \quad \frac{1}{2\pi}(F * G)(\omega)$

Proof.

$$\begin{aligned}
\mathcal{F}^{-1}\left\{(F*G)(\omega)\right\} &= \frac{1}{2\pi}\int_{-\infty}^{\infty}(F*G)(\omega)e^{i\omega t}d\omega \\
\mathcal{F}^{-1}\left\{(F*G)(\omega)\right\} &= \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}F(W)G(\omega-W)dWe^{i\omega t}d\omega \\
\mathcal{F}^{-1}\left\{(F*G)(\omega)\right\} &= \frac{1}{2\pi}\int_{-\infty}^{\infty}F(W)\int_{-\infty}^{\infty}G(\omega-W)e^{i\omega t}d\omega dW \\
\mathcal{F}^{-1}\left\{(F*G)(\omega)\right\} &= \frac{1}{2\pi}\int_{-\infty}^{\infty}F(W)\int_{-\infty}^{\infty}G(\Omega)e^{i(\Omega+W)t}d\Omega dW \\
\mathcal{F}^{-1}\left\{(F*G)(\omega)\right\} &= \frac{1}{2\pi}\int_{-\infty}^{\infty}F(W)\left(\int_{-\infty}^{\infty}G(\Omega)e^{i\Omega t}d\Omega\right)e^{iWt}dW \\
\mathcal{F}^{-1}\left\{(F*G)(\omega)\right\} &= \frac{1}{2\pi}\int_{-\infty}^{\infty}F(W)e^{iWt}dW\left(\int_{-\infty}^{\infty}G(\Omega)e^{i\Omega t}d\Omega\right) \\
\mathcal{F}^{-1}\left\{(F*G)(\omega)\right\} &= \frac{1}{2\pi}\left(2\pi f(t)\right)\left(2\pi g(t)\right) \\
\mathcal{F}^{-1}\left\{(F*G)(\omega)\right\} &= 2\pi f(t)g(t) \\
\mathcal{F}^{-1}\left\{\frac{1}{2\pi}(F*G)(\omega)\right\} &= f(t)g(t)
\end{aligned}$$

Which shows that multiplication in the time domain is the product of a scalar and the convolution in the frequency domain.

□

8.3.3 example

Sometimes it is useful to include examples within text, and for this the `example` environment works well.

You can add an example with the following syntax:

```
\begin{example}
 % some math + typesetting here
\end{example}
```

Here is an example (no pun intended):

Example. Consider the causal signal $x(t) = e^{-3t}u(t)$. What is the Laplace Transform for $x(t+T)$?

$$y(t) = x(t+T)$$
$$y(t) = e^{-3(t+T)}u(t+T)$$

Note that the signal is no longer causal, although it is still right-sided. We know the transform for $x(t)$ and can utilize the shifting property:

$$x(t) \quad \underset{\longleftrightarrow}{\mathcal{L}} \quad \hat{X}(s) = \frac{1}{3+s}$$
$$x(t+T) \quad \underset{\longleftrightarrow}{\mathcal{L}} \quad e^{sT}\hat{X}(s) = \frac{e^{sT}}{3+s}$$

The *RoC* is the same in this case, except that it excludes ∞:

$$RoC(y) : \{s \mid -3 < \Re e\{s\} < \infty\}$$

8.3.4 definition

Definition 1. If the function $x \in X$ is defined as $x : A \to B$, and A is uncountable, then we say that the signal x is a *Continuous-Time Signal* (a.k.a. CT Signal).

Tip: You can use `nicebox` to make it stand out if you like:

Definition 2. The function f is a *one-to-one correspondence*, or a *bijection*, if it is both one-to-one and onto.

8.3.5 claim

Claim. *if a signal is real-valued, then its Fourier transform is conjugate symmetric. In other words,*

$$x(n) \in \mathbb{R}\ \forall n \Rightarrow X(\omega) = X^*(\omega)$$

8.3.6 theorems and corollaries

The platform also supports corollary and theorem environments. You can define them like so:

```
\begin{corollary}
```

```
 corallary goes here!
\end{corollary}

\begin{theorem}
 theorem goes here!
\end{theorem}
```

Theorem 1. The Chinese Remainder Theorem *Let* $m_1, m_2, \cdots, m_n \in \mathbb{Z}$ *such that* $\gcd(m_i, m_j) = 1$ *where* $i \neq j$ *Let* $a_1, \cdots, a_n$ *are arbitrary. This system of congruences*

$$x \equiv a_1 \bmod m_1$$

$$x \equiv a_2 \bmod m_2$$

$$\cdots$$

$$x \equiv a_n \bmod m_n$$

has a solution modulo $m = \prod_{i=1}^{n} m_i$*. Basically we can find a value* x *that is a solution by the formula*

$$x = a_1 M_1 y_1 + a_2 M_2 y_2 + \cdots a_n M_n y_n$$

where $0 \leq x < m$*, and all other solutions are congruent modulo* m*.*

8.3.7 problems and solutions

The platform also supports `problem` and `solution` environments:

```
\begin{problem}
Given the complex number $z = e^{i\frac{2\pi}{3}}$, determine $\sqrt[4]{z}$ . Be mindful of how many fou
of them.
\end{problem}

\begin{solution}
\begin{align*}
z &= e^{i\frac{2\pi}{3}} \\
z &= e^{i(\frac{2\pi}{3} + 2\pi k) } \mbox{ for some } k \in \Z \\
\sqrt[4]{z} &= e^{i(\frac{2\pi}{3} + 2\pi k) \frac{1}{4}} \\
\sqrt[4]{z} &= e^{i(\frac{\pi}{6} + \frac{\pi}{2} k) \frac{1}{4}}
```

```
\end{align*}
So we can determine that within the range [$0,2\pi$), we find that $\sqrt[4]{z} = e^{i\frac{\pi}{6}}, e
\end{solution}
```

Problem. Given the complex number $z = e^{i\frac{2\pi}{3}}$, determine $\sqrt[4]{z}$. Be mindful of how many fourth roots z has and identify each of them.

Solution.

$$z = e^{i\frac{2\pi}{3}}$$
$$z = e^{i(\frac{2\pi}{3}+2\pi k)} \text{ for some } k \in \mathbb{Z}$$
$$\sqrt[4]{z} = e^{i(\frac{2\pi}{3}+2\pi k)\frac{1}{4}}$$
$$\sqrt[4]{z} = e^{i(\frac{\pi}{6}+\frac{\pi}{2}k)\frac{1}{4}}$$

So we can determine that within the range $[0, 2\pi)$, we find that $\sqrt[4]{z} = e^{i\frac{\pi}{6}}, e^{i\frac{2\pi}{3}}, e^{i\frac{7\pi}{6}}, e^{i\frac{5\pi}{3}}$

8.4 Environments

8.4.1 verbatim

This is probably one of the most important environments to implement! This is because it's required to show syntax to the reader who wishes to learn about TEX and LATEX.

```
 begin{verbatim}
  some \LaTeX\ here will not get parsed!
 end{verbatim}
```

which looks like this when rendered:

```
some \LaTeX\ here will not get parsed!
```

Example. This method based on the Chinese Remainder Theorem, is used to factor a large number N.

Pick a random integer for a starting number = $y_0 = z_0$ Then recursively apply the function $f(x) = x^2 + a \bmod N$ twice to z, and once to y, where a is the random remainder to start with.

$$y_1 = f(y_0)$$

$$z_1 = f(f(y_0))$$

We use this process to generate remainders until we hit a loop and one of them is divisible by p, where $N = pq$. which has a good chance of being achieved by $\sqrt{p}$ factors.

We test this at each iteration by taking the greatest common divisor of the difference between the values and the number you are trying to factor.

$$\gcd(|y_k - x_k|, N)$$

If the gcd 1, then find x_k and y_k and recursively apply the algorithm. If you do not get a one, then the number is a factor of N.

An algorithm written in Scheme emulates this procedure:

```
(define (pollards n y z fn)
        (let* ((n-y (fn y))
                  (n-z (fn (fn z)))
                  (g (gcd (abs (- n-y n-z)) n)))
            (if (= g 1) (pollards n n-y n-z fn) g)))
```

Here is an example of a program written in an assembly language called MIPS:

Example. This emulates `*x = *y` where `x` and `y` are pointers

```
        .data
x:      .word 4
y:      .word 5

        .text
main:
    la      $s0, x
    la      $s1, y
    lw      $t1, 0($s1) # read y into temp var
    sw      $t1, 0($s0) # store into x
    lw      $t1, 0($s0) # read from x into temp var
    move    $a0, $t1
    li $v0, 1
    syscall
    li      $v0, 10
    syscall
```

8.4.2 math

Thanks to MathJax, we can render LaTeX equations in the browser. Here is an example

Using "$" symbols, you can create a math environement. Also using the `align` environment also works (thanks to MathJAX).

You can produce math *within* sentences.

We can see that differentiation of a rational function will produce a rational Z-transform. If you recall, the set of signals that produce rational Z-transforms are obtained from, and only from, linear combinations of $\alpha^n u(n)$, $\alpha^n u(-n)$, $n^k \alpha^n u(n)$, and $n^k \alpha^n u(-n)$, where $k \in \mathbb{N}$, and shifted versions thereof.

```
We can see that differentiation of a rational function will produce a rational Z-transform.
If you recall, the set of signals that produce rational Z-transforms are obtained from, and only from, l
$\alpha^n u(n)$, $\alpha^n u(-n)$, $n^k \alpha^n u(n)$, and $n^k \alpha^n u(-n)$, where $k \in \N$, and
```

Using two "$" symbols makes a centered math environment.

```
$$\int_0^\infty H(\omega)e^{i\omega x} dx$$
```

which produces

$$\int_0^\infty H(\omega)e^{i\omega x}dx$$

more info on TeXsupport here (read carefully!) http://docs.mathjax.org/en/latest/tex.html

$$\hat{G}(z) = \sum_{n\in\mathbb{Z}} g(n)z^{-n}$$
$$\frac{d}{dz}\hat{G}(z) = -\sum_{n\in\mathbb{Z}} ng(n)z^{-n-1}$$
$$\frac{d}{dz}\hat{G}(z) = -z^{-1}\sum_{n\in\mathbb{Z}} ng(n)z^{-n}$$

8.4.3 nicebox

You can create "nice" boxes by using the `nicebox` environment, which produces something like this:

$$\frac{\delta}{\delta u}\int_{birth}^{death} f(life)du = \text{your life}$$

8.4.4 quotation

Using the `quotation` environment, you can do quotation blocks!

```
\begin{quotation}
a very long quote here... \cite[10]{sicp}
\end{quotation}
```

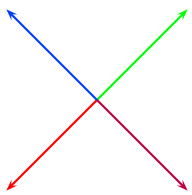
Here is an example of what it looks like (a great quote from Abelson and Sussman):

> Our traffic with the subject matter of this book involves us with three foci of phenomena: the human mind, collections of computer programs, and the computer. Every computer program is a model, hatched in the mind, of a real or mental process. These processes, arising from human experience and thought, are huge in number, intricate in detail, and at any time only partially understood. They are modeled to our permanent satisfaction rarely by our computer programs. Thus even though our programs are carefully handcrafted discrete collections of symbols, mosaics of interlocking functions, they continually evolve: we change them as our perception of the model deepens, enlarges, generalizes until the model ultimately attains a metastable place within still another model with which we struggle. The source of the exhilaration associated with computer programming is the continual unfolding within the mind and on the computer of mechanisms expressed as programs and the explosion of perception they generate. If art interprets our dreams, the computer executes them in the guise of programs! (Abelson and Sussman, 1996, 10)

8.5 PSTricks

8.5.1 basic graphics parameters

You can specify `linewidth, linecolor, fillstyle, fillcolor` on most PSTricks objects. For example:

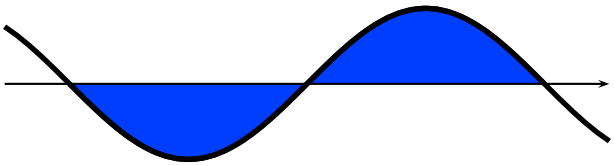


Which can be created with the following source:

```
\begin{pspicture}(-2,-2)(2,2)
\psline[linecolor=green]{->}(0,0)(1.2,1.2)
```

```
\psline[linecolor=red]{->}(0,0)(-1.2,-1.2)
\psline[linecolor=purple]{->}(0,0)(1.2,-1.2)
\psline[linecolor=lightblue]{->}(0,0)(-1.2,1.2)
\end{pspicture}
```

Fill modes can be useful to show things like integration and area:

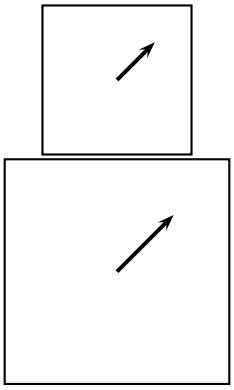


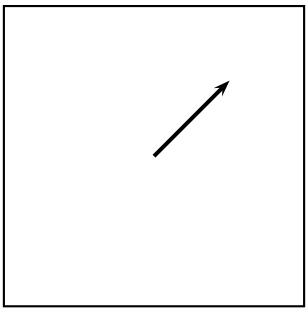
Which can be created with the following source:

```
\begin{pspicture}(-4,-3)(4,3)
\psplot[algebraic,linewidth=2pt,fillstyle=solid, fillcolor=lightblue]{-1.5}{1.5}{sin(x)}
\psplot[algebraic,linewidth=2pt]{-4}{4}{sin(x)}
\psline{->}(-4,0)(4,0)
\end{pspicture}
```

8.5.2 units

You can use `psset` to set units before a graphic is rendered.

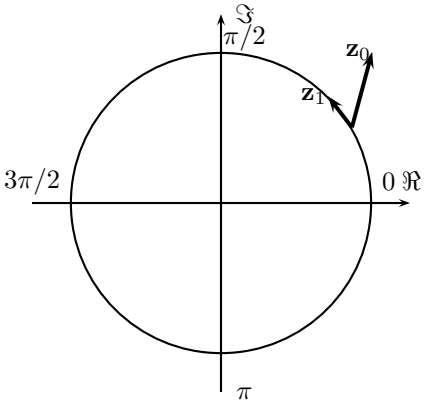




The way you do this is by supplying `psset` before the `pspicture` environment.

8.5.3 arrows

Arrows are much more complicated that at first glance. If you use turtle style drawing to draw everything on the page, how to you render an arrowhead on a given line?



After much frustration, I found something promising, and note that the hope for this platform is to benefit people like this: mathforum.org

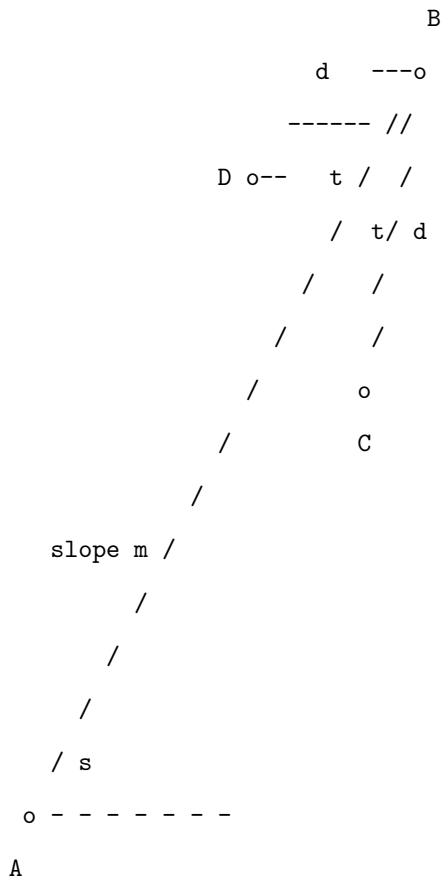
Ironically, below is a snippet from the link above, where a person asked Dr. Math a question about arrows, and the responses were all in ASCII text! Imagine if Dr. Math had a mathematical diagramming system at his disposal!

```
Hi, Johan.

Now that I've given you a chance to work this problem out, I want to
let you know what I came up with, because I recognize it may be
difficult for you to do on your own. It's an interesting problem.

We have a line segment from A(x0,y0) to B(x1,y1), and want to draw
```

segments BC and BD of length d, at a t degree angle on either side of the segment at (x1,y1):

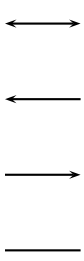


The solution was this following formula:

$$x = x1 - [(x1 - x0)\,cos(t) - (y1 - y0)\,sin(t)]\,\frac{d}{l}$$
$$y = y1 - [(y1 - y0)\,cos(t) + (x1 - x0)\,sin(t)]\,\frac{d}{l}$$

This enables us to use pslines and have arrows!

Here are four example lines with and without arrows:



Which can be produced with the following TEX:

```
\begin{pspicture}(-3,-3)(3,3)
\psline{<->}(0,3)(1,3)
\psline{<-}(0,2)(1,2)
\psline{->}(0,1)(1,1)
\psline(0,0)(1,0)
\end{pspicture}
```

8.6 PSTricks elements

8.6.1 pstrick docs

All decisions on how to implement the PSTricks commands came from Timothy Van Zandt's PSTricks User Guide which can be found here

8.6.2 pspicture

The very first step in creating a graphic to use the `pspicture` command:

```
\begin{pspicture}(x0,y0)(x1,y1)
 % pstricks goes here
\end{pspicture}
```

This will create a picture object where the lower left-hand corner is at `(x0,y0)` and upper right-hand corner is at `(x1,y1)` (Zandt, 2007, 40).

8.6.3 psframe

`psframe` will simply create a box.

```
\begin{pspicture}(-1,-1)(2,2)
```

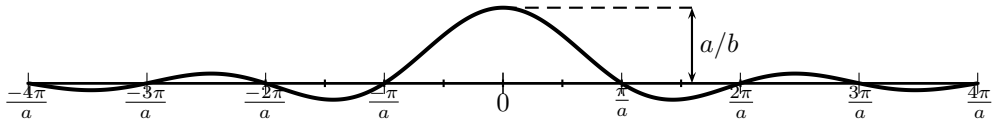
```
\psframe(0,0)(1,1)
\end{pspicture}
```

This will create a box where the lower left-hand corner is at `(x0,y0)` and upper right-hand corner is at `(x1,y1)`:

8.6.4 psplot

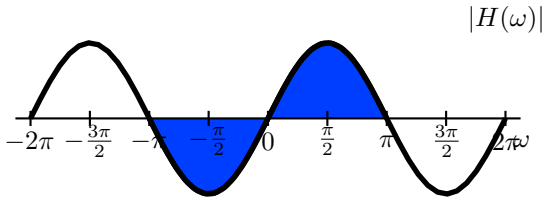
The `psplot` command is powerful. This allows you to create graphs.

An important note: if you want your work to work on *both* Web and paper, you must specify `algebraic` in the parameters. This platform actually uses JavaScript syntax for evaluating the formulae, which is almost identical to the syntax used normally.



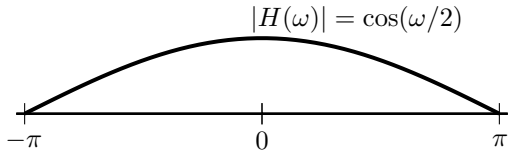

```
\begin{pspicture}(-13,-1.25)(13,3)
\psplot[algebraic,linewidth=1.5pt,plotpoints=1000]{-12.56}{12.56}{2*sin(x)/x}
\psaxes[showorigin=false,labels=none, Dx=3.14](0,0)(-12.6,0)(12.6,0)
\rput(5.7, 1){$a/b$}
\psline[linestyle=dashed](0,2)(5,2)
\psline{<->}(5,0)(5,2)
\rput(3.14, -0.5){$\frac{\pi}{a}$}
\rput(6.28, -0.5){$\frac{2\pi}{a}$}
\rput(9.42, -0.5){$\frac{3\pi}{a}$}
\rput(12.56, -0.5){$\frac{4\pi}{a}$}
\rput(-3.14, -0.5){$\frac{-\pi}{a}$}
\rput(-6.28, -0.5){$\frac{-2\pi}{a}$}
\rput(-9.42, -0.5){$\frac{-3\pi}{a}$}
\rput(-12.56, -0.5){$\frac{-4\pi}{a}$}
\rput(0, -0.5){$0$}
```

```
\psline(-6.2831853, -0.1)(-6.2831853, 0.1)
\psline(-4.712388975, -0.1)(-4.712388975, 0.1)
\psline(-3.14159265, -0.1)(-3.14159265, 0.1)
\psline(-1.570796325, -0.1)(-1.570796325, 0.1)
\psline(0, -0.1)(0, 0.1)
\psline(1.570796325, -0.1)(1.570796325, 0.1)
\psline(3.14159265, -0.1)(3.14159265, 0.1)
\psline(4.712388975, -0.1)(4.712388975, 0.1)
\psline(6.2831853, -0.1)(6.2831853, 0.1)
\end{pspicture}
```

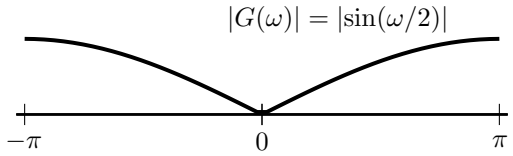



```
\begin{pspicture}(-8,-2)(8,2)
\psplot[algebraic,linewidth=2pt,fillstyle=solid, fillcolor=lightblue]{-3.14}{3.14}{sin(x)}
\psplot[algebraic,linewidth=2pt]{-6.2831}{6.2831}{sin(x)}
\rput(6.2831853,1.3){$\left|H(\omega)\right|$}
\psline(-6.6831853, 0)(6.6831853, 0)
\rput(6.6831853,-0.3){$\omega$}
\psline(-6.2831853, -0.1)(-6.2831853, 0.1)
\rput(-6.2831853, -0.3){$-2\pi$}
\psline(-4.712388975, -0.1)(-4.712388975, 0.1)
\rput(-4.712388975, -0.3){$-\frac{3\pi}{2}$}
\psline(-3.14159265, -0.1)(-3.14159265, 0.1)
\rput(-3.14159265, -0.3){$-\pi$}
\psline(-1.570796325, -0.1)(-1.570796325, 0.1)
\rput(-1.570796325, -0.3){$-\frac{\pi}{2}$}
\psline(0, -0.1)(0, 0.1)
\rput(0, -0.3){$0$}
\psline(1.570796325, -0.1)(1.570796325, 0.1)
\rput(1.570796325, -0.3){$\frac{\pi}{2}$}
```

```
\psline(3.14159265, -0.1)(3.14159265, 0.1)
\rput(3.14159265, -0.3){$\pi$}
\psline(4.712388975, -0.1)(4.712388975, 0.1)
\rput(4.712388975, -0.3){$\frac{3\pi}{2}$}
\psline(6.2831853, -0.1)(6.2831853, 0.1)
\rput(6.2831853, -0.3){$2\pi$}
\end{pspicture}
```




```
\begin{pspicture}(-3.5,-3.25)(3.5,2)
\psplot[algebraic,linewidth=1.5pt]{-3.14}{3.14}{cos(x/2)}
\psaxes[showorigin=false,labels=none, Dx=3.14](0,0)(-3.25,0)(3.25,0)
\rput(3.14, -0.35){$\pi$}
\rput(-3.14, -0.35){$-\pi$}
\rput(0, -0.35){$0$}
\rput(1.25,1.25 ){$\left| H(\omega) \right| =\cos(\omega/2)$}
\end{pspicture}
```

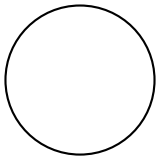



```
\begin{pspicture}(-3.5,-1.25)(3.5,2)
\psplot[algebraic,linewidth=1.5pt]{-3.14}{3.14}{abs(sin(x/2))}
\psaxes[showorigin=false,labels=none, Dx=3.14](0,0)(-3.25,0)(3.25,0)
\rput(3.14, -0.35){$\pi$}
\rput(-3.14, -0.35){$-\pi$}
```

```
\rput(0, -0.35){$0$}
\rput(1,1.25 ){$\left| G(\omega) \right| =\left| \sin(\omega/2) \right|$}
\end{pspicture}
```

8.6.5 pscircle

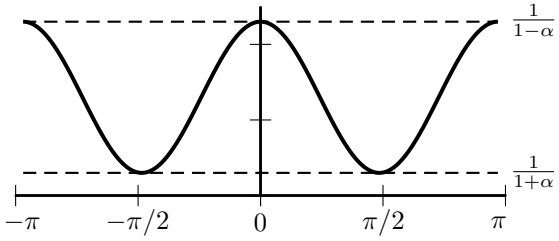
Circles are simple, just specify an origin and a radius:



```
\begin{pspicture}(-2,-2)(2,2)
\pscircle(0,0){ 1 }
\end{pspicture}
```

8.6.6 psaxes

The `psaxes` command is somewhat unfinished, but it does generate the x and y axes:



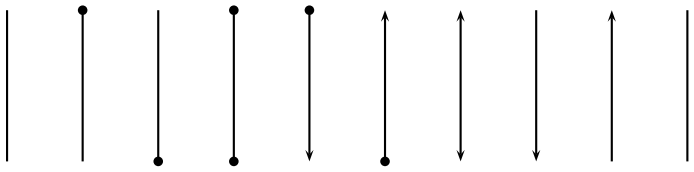

```
\begin{pspicture}(-3.5,-1)(3.75,3.5)
\psplot[algebraic,linewidth=1.5pt,plotpoints=1000]{-3.14}{3.14}{cos(4*x/2)+1.3}
\psaxes[showorigin=false,labels=none, Dx=1.62](0,0)(-3.25,0)(3.25,2.5)
\psline[linestyle=dashed](-3.14,0.3)(3.14,0.3)
\psline[linestyle=dashed](-3.14,2.3)(3.14,2.3)
\rput(3.6,2.3){$\frac{1}{1-\alpha}$}
\rput(3.6,0.3){$\frac{1}{1+\alpha}$}
\rput(3.14, -0.35){$\pi$}
```

```
\rput(1.62, -0.35){$\pi/2$}
\rput(-1.62, -0.35){$-\pi/2$}
\rput(-3.14, -0.35){$-\pi$}
\rput(0, -0.35){$0$}
\end{pspicture}
```

8.6.7 psline

The `psline` command can be used in a multitude of ways.

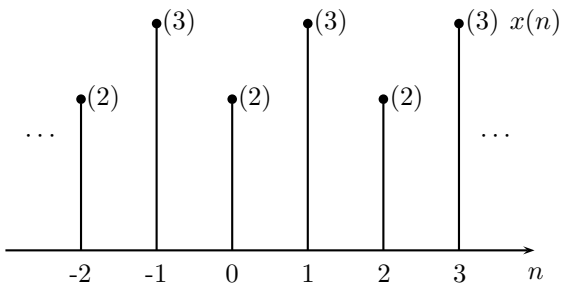
There is a special option on `psline` that allows you to specify dots and arrows.



Which can be produced with the following source:

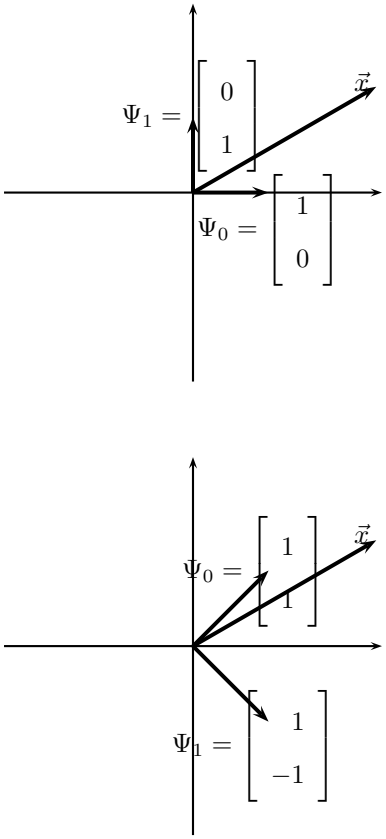
```
\begin{pspicture}(-2.5,-1)(7.6,4)
\psline(-2,0)(-2,2)
\psline{-*}(-1,0)(-1,2)
\psline{*-}(0,0)(0,2)
\psline{*-*}(1,0)(1,2)
\psline{<-*}(2,0)(2,2)
\psline{*->}(3,0)(3,2)
\psline{<->}(4,0)(4,2)
\psline{<-}(5,0)(5,2)
\psline{->}(6,0)(6,2)
\psline(7,0)(7,2)
\end{pspicture}
```

They can be useful for creating kronecker delta plots:




```
\begin{pspicture}(-3,-1)(4,4)
\rput(-2.5,1.5){$\cdots$}
\rput(3.5,1.5){$\cdots$}
\psline{-*}(-2,0)(-2,2)
\rput(-1.7,2){(2)}
\rput(-2,-0.3){-2}
\psline{-*}(-1,0)(-1,3)
\rput(-0.7,3){(3)}
\rput(-1,-0.3){-1}
\psline{-*}(0,0)(0,2)
\rput(0.3,2){(2)}
\rput(0,-0.3){0}
\psline{-*}(1,0)(1,3)
\rput(1.3,3){(3)}
\rput(1,-0.3){1}
\psline{-*}(2,0)(2,2)
\rput(2.3,2){(2)}
\rput(2,-0.3){2}
\psline{-*}(3,0)(3,3)
\rput(3.3,3){(3)}
\rput(3,-0.3){3}
\psline{->}(-3,0)(4,0)
\rput(4,-0.3){ $n$ }
\rput(4,3){ $x(n)$ }
\end{pspicture}
```

Other uses include creating representations of vectors:

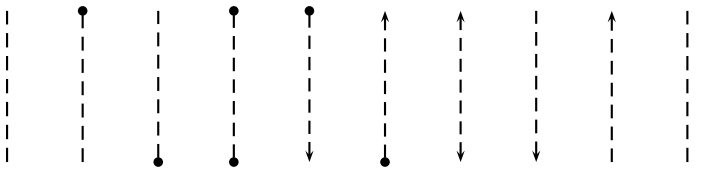


Which can be created with the following source:

```
\begin{pspicture}(-3,-6)(3,6)
\psline{->}(0,-6)(0,-1)
\psline{->}(-2.5,-3.5)(2.5,-3.5)
\rput(0.8,-4.8){$\Psi_1 = \left[ \begin{array}{r} 1 \\ -1 \\ \end{array} \right]$}
\psline[linewidth=1.5 pt]{->}(0,-3.5)(1,-4.5)
\rput(2.22487174976714,-2){$\vec{x}$}
\psline[linewidth=1.5 pt]{->}(0,-3.5)(2.42487174976714,-2.10000107243568)
\rput(0.8,-2.5){$\Psi_0 = \left[ \begin{array}{r} 1 \\ 1 \\ \end{array} \right]$}
\psline[linewidth=1.5 pt]{->}(0,-3.5)(1,-2.5)
\psline{->}(0,0)(0,5)
\psline{->}(-2.5,2.5)(2.5,2.5)
\rput(2.22487174976714,3.95){$\vec{x}$}
\psline[linewidth=1.5 pt]{->}(0,2.5)(2.42487174976714,3.89999892756432)
\rput(1,2){$\Psi_0 = \left[ \begin{array}{r} 1 \\ 0 \\ \end{array} \right]$}
\psline[linewidth=1.5 pt]{->}(0,2.5)(0,3.5)
```

```
\rput(0,3.5){$\Psi_1 = \left[ \begin{array}{r} 0 \\ 1 \\ \end{array} \right]$}
\psline[linewidth=1.5 pt]{->}(0,2.5)(1,2.5)
\end{pspicture}
```

You can also use `linestyle=dashed` in the options.

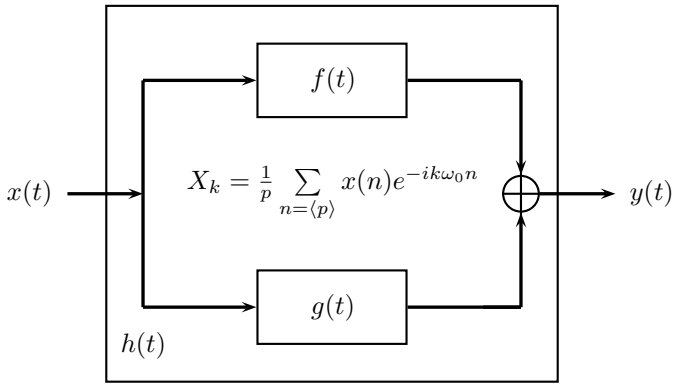


Which can be produced with the following source:

```
\begin{pspicture}(-2.5,-1)(7.6,4)
\psline[linestyle=dashed](-2,0)(-2,2)
\psline[linestyle=dashed]{-*}(-1,0)(-1,2)
\psline[linestyle=dashed]{*-}(0,0)(0,2)
\psline[linestyle=dashed]{*-*}(1,0)(1,2)
\psline[linestyle=dashed]{<-*}(2,0)(2,2)
\psline[linestyle=dashed]{*->}(3,0)(3,2)
\psline[linestyle=dashed]{<->}(4,0)(4,2)
\psline[linestyle=dashed]{<-}(5,0)(5,2)
\psline[linestyle=dashed]{->}(6,0)(6,2)
\psline[linestyle=dashed](7,0)(7,2)
\end{pspicture}
```

8.6.8 rput - place math anywhere

Probably the best part of using PSTricks is that you can mix both graphics and mathematics:



which can be produced with the following code:

```
\rput(0,0){$x(t)$}
\rput(4,1.5){$f(t)$}
\rput(4,-1.5){$g(t)$}
\rput(8.2,0){$y(t)$}
\rput(1.5,-2){$h(t)$}
\psframe(1,-2.5)(7,2.5)
\psframe(3,1)(5,2)
\psframe(3,-1)(5,-2)
\rput(4,0){$X_k = \frac{1}{p} \sum \limits_{n=\langle p\rangle}x(n)e^{-ik\omega_0n}$}
\psline[linewidth=1.25 pt]{->}(0.5,0)(1.5,0)
\psline[linewidth=1.25 pt]{->}(1.5,1.5)(3,1.5)
\psline[linewidth=1.25 pt]{->}(1.5,-1.5)(3,-1.5)
\psline[linewidth=1.25 pt]{->}(6.5,1.5)(6.5,0.25)
\psline[linewidth=1.25 pt]{->}(6.5,-1.5)(6.5,-0.25)
\psline[linewidth=1.25 pt]{->}(6.75,0)(7.75,0)
\psline[linewidth=1.25 pt](1.5,-1.5)(1.5,1.5)
\psline[linewidth=1.25 pt](5,1.5)(6.5,1.5)
\psline[linewidth=1.25 pt](5,-1.5)(6.5,-1.5)
\psline[linewidth=1.25 pt](6,-1.5)(6.5,-1.5)
\pscircle(6.5,0){0.25}
\psline(6.25,0)(6.75,0)
\psline(6.5,0.5)(6.5,-0.5)
```

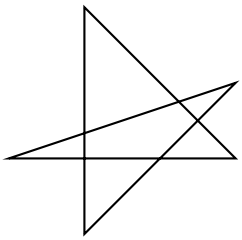
8.6.9 pspolygon

The `pspolygon` is a great way to create arbitrary shapes!

```
\pspolygon(x0,y0)(x1,y1)(x2,y2) ... (xn,yn)
```

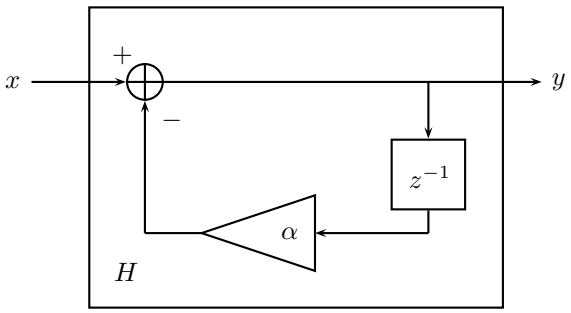
You can provide an array of coordinates that will render a polygon (Zandt, 2007, 10)

You can add an aribitray number of coordinates onto the end of the command, and produce any shape you'd like!



which can be produced with the following code:

```
\begin{pspicture}(-3,-3)(3,3)
\pspolygon(-1,-1)(-1,2)(1,0)(-2,0)(1,1)
\end{pspicture}
```



Which can be produced with the following TEX:

```
\begin{pspicture}(0,-3.5)(8,2)
\rput(0,0){$x$}
\rput(7.2,0){$y$}
```

```
\rput(5.5,-1.25){$z^{-1}$}
\rput(3.65,-2){$\alpha$}
\rput(1.5,-2.5){$H$}
\rput(2.1,-0.5){$-$}
\rput(1.45,0.35){$+$}
\psframe(1,-3)(6.5,1)
\pspolygon(4,-1.5)(4,-2.5)(2.5,-2)
\psline{->}(0.25,0)(1.5,0)
\psline{->}(1.75,-2)(1.75,-0.25)
\psline{->}(5.5,-2)(4,-2)
\psline(2.5,-2)(1.75,-2)
\psline{->}(2,0)(7,0)
\psline(5.5,-1.7)(5.5,-2)
\psline{->}(5.5,0)(5.5,-0.75)
\psframe(5, -1.7)(6,-0.75)
\pscircle(1.75,0){0.25}
\psline(1.5,0)(2,0)
\psline(1.75,0.25)(1.75,-0.25)
\end{pspicture}
```

8.7 Backwards Compatability

8.7.1 Rendering issues

Because of the Web, there are a few inherent issues that we cannot really get around directly. HTML uses brackets to enclose HTML tags, '<' and '>', therefore, it is **not** a good idea to use them in your mathematics.

MathJax addresses this on their website here if you'd like to read more about it. The main reason is that the browser has access to the text before MathJax has a chance to render it. Now in the case of this platform, we actually *do* have access to the math before the DOM does, however, to keep consistent with the MathJax community, we've also decided to use the `lt` and `gt` commands to replace less than and greater than symbols.

$$a < b$$

$$b > a$$

which can be produced using the following LaTeX:

```
\begin{align*}
a \lt b \\
b \gt a
\end{align*}
```

8.7.2 Online and Offline environments

Also there are two environments to encourage keeping your work both online (Web) and offline (PDF): `print` and `interactive`.

```
begin{print}
stuff for your PDF version goes here
end{print}

begin{interactive}
stuff for your Web version goes here
end{interactive}
```

We have provided, however, our .sty files for some sort of backwards and forwards compatibility here.

8.7.3 Printing your work

Download the .tex, save it as `paper.tex`:

Download the .bib, save it as `sources.bib`:

Then download the .sty files here.

Now, a number of different commands work for this, `latex, pdflatex, xelatex`, but here is what I've gotten the best results with:

```
xelatex -shell-escape paper.tex
bibtex paper
xelatex -shell-escape paper.tex
xelatex -shell-escape paper.tex
```

8.8 extended interactivity

8.8.1 slider

We have provided some useful elements to expose basic interactive functionality for an end user. The first of these is the `slider` command.

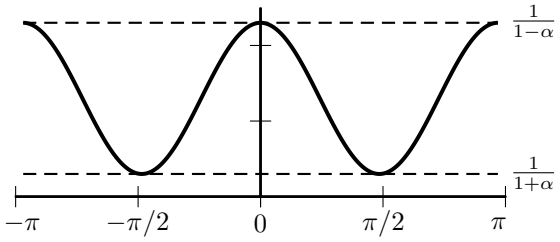
Arguments for the `slider` command are:

```
\slider{min}{max}{variable}{latex}{default}
```

The min and max values are for specifying the minimum and maximum of the range of the slider.

The `latex` argument is what to display next to the slider to indicate to the end user what variable the slider is changing. Finally, the most important is the `variable` argument. The `variable` specifies the variable that is changed based on the values of the slider, and can be used in the equations of `psplot` commands.

Note that using these "extended" features are not backwards compatible, a.k.a they don't work on paper!



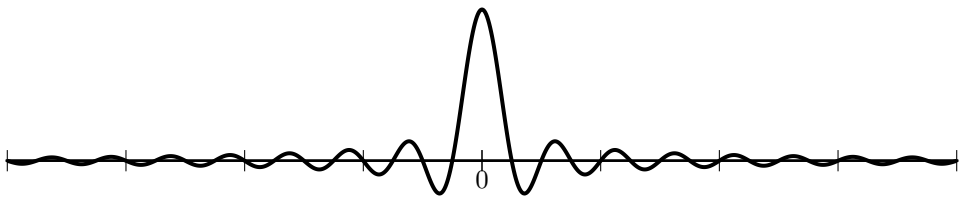
Which can be created using the following TEX:

```
\begin{pspicture}(-3.5,-1)(3.75,3.5)

\slider{1}{8}{n}{$N$}{4}
```

```
\psplot[algebraic,linewidth=1.5pt,plotpoints=1000]{-3.14}{3.14}{cos(n*x/2)+1.3}
\psaxes[showorigin=false,labels=none, Dx=1.62](0,0)(-3.25,0)(3.25,2.5)
\psline[linestyle=dashed](-3.14,0.3)(3.14,0.3)
\psline[linestyle=dashed](-3.14,2.3)(3.14,2.3)
\rput(3.6,2.3){$\frac{1}{1-\alpha}$}
\rput(3.6,0.3){$\frac{1}{1+\alpha}$}
\rput(3.14, -0.35){$\pi$}
\rput(1.62, -0.35){$\pi/2$}
\rput(-1.62, -0.35){$-\pi/2$}
\rput(-3.14, -0.35){$-\pi$}
\rput(0, -0.35){$0$}
\end{pspicture}
```

You can also add more than one slider (variable):



which can be created with the following:

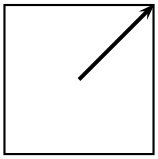
```
\begin{pspicture}(-13,-5)(13,10)
\slider{1}{8}{a}{amplitude}{4}
\slider{1}{8}{n}{frequency}{4}
\psplot[algebraic,linewidth=1.5pt,plotpoints=1000]{-12.56}{12.56}{a*sin(n*x)/(n*x)}
\psaxes[showorigin=false,labels=none, Dx=3.14](0,0)(-12.6,0)(12.6,0)
\rput(0, -0.5){$0$}
\end{pspicture}
```

8.8.2 userline

`userline` is just like `psline`, except you can have extra arguments on the end to represent a user action. Suppose you'd like a vector to follow the user's cursor or finger on a computer or touch device. You can simply do

```
\userline[linewidth=1.5 pt]{->}(0,0)(2,2)
```

This places a vector at $(0, 0)$ and initial value of the head at $(2, 2)$. This will produce the following interaction



This can be produced with the following source:

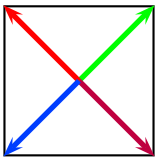
```
\begin{pspicture}(-2,-2)(2,2)
\psframe(-2,-2)(2,2)
\userline[linewidth=1.5 pt]{->}(0,0)(2,2)
\end{pspicture}
```

If you specify more arguments, you can create functions for the head and and tail of the vector, which each takes the current x and y position of the users finger or cursor and they move.

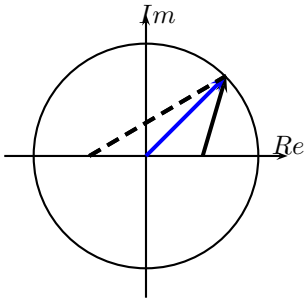
2 extra arguments provide functions for the head, 4 extra arguments allows you to control both and tail

```
\userline[linewidth=2pt,linecolor=green]{->}(0,0)(2,2){-x}{-y}
\userline[linewidth=2pt,linecolor=red]{->}(0,0)(2,2){0}{y}
\userline[linewidth=2pt,linecolor=purple]{->}(0,0)(2,2){-x}{cos(y)}
\userline[linewidth=2pt,linecolor=lightblue]{->}(0,0)(2,2)(sin(x)}{-y}
```

and produce the following interaction



Then you can have fun with math!




```
\begin{pspicture}(-5,-5)(5,5)
\rput(0.3,3.75){ $Im$ }
\psline{->}(0,-3.75)(0,3.75)
\rput(3.75,0.3){ $Re$ }
\psline{->}(-3.75,0)(3.75,0)
\pscircle(0,0){ 3 }

\userline[linewidth=1.5 pt]{->}(1.500,0.000)(2.121,2.121)
\userline[linewidth=1.5 pt,linecolor=blue]{->}(0,0.000)(2.121,2.121){(x>0) ? 3 * cos( atan(-y/x) ) : -3
\userline[linewidth=1.5 pt,linestyle=dashed](-1.500,0.000)(2.121,2.121){x}{0}{x}{y}
\userline[linewidth=1.5 pt,linestyle=dashed](-1.500,0.000)(2.121,2.121){0}{y}{x}{y}
\end{pspicture}
```

8.8.3 uservariable

The `uservariable` command represents a variable that changes with user interaction such as touch or mouse events over the graphic.

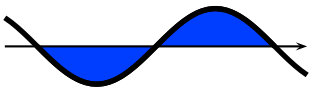
```
\uservariable{variable}(x1,y1){f(x,y)}
```

`variable` is the variable that you can use in `psplot` and `userline` parameters and equations. Eventually it should be something you can use everywhere, but for the scope of this project, we kept it limited.

`x1` and `y1` are the initial values of `x` and `y` for the function $f(x, y)$ that is that last argument for the command. If there are no user interactions, the function is evaluated with the specified inital values.

On any user event, such as a touch or mouse event, the function $f(x, y)$ is evaluated and any dependent commands, namely `psplot` and `userline` are immediately notified and re-rendered with updated execution contexts.

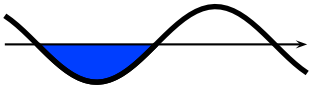
Here is an example of teaching somebody integration, for example, you can represent $\int_a^b f(x)dx$ as:



which can be produced with the following TEX:

```
\begin{pspicture}(-4,-3)(4,3)
\uservariable{alpha}(0,0){x}
\psplot[algebraic,linewidth=2pt,fillstyle=solid, fillcolor=lightblue]{-4}{alpha}{sin(x)}
\psplot[algebraic,linewidth=2pt]{-4}{4}{sin(x)}
\psline{->}(-4,0)(4,0)
\end{pspicture}
```

The next example allows you to move the height of the graph and also demonstrates how to specify an interval:



which can be produced with the following source:

```
\begin{pspicture}(-4,-3)(4,3)
\uservariable{alpha}(0,0){x}
\uservariable{beta}(0,0){y}
\psplot[algebraic,linewidth=2pt,fillstyle=solid, fillcolor=lightblue]{alpha-3}{alpha}{beta + sin(x)}
\psplot[algebraic,linewidth=2pt]{-4}{4}{beta + sin(x)}
\psline{->}(-4,0)(4,0)
\end{pspicture}
```

8.9 additional and proposed elements

8.9.1 img

the `img` command allows you to include an image.

can be produced with the following source:

```
\img{https://c328740.ssl.cf1.rackcdn.com/mathjax/badge/badge-square-3.png}
```

8.9.2 youtube

Sometimes the need to display videos is appropriate. For this, we provided a `youtube` command. This only works online, and takes the youtube video id as an argument.

```
\youtube{QYMLMUKJyFc}
```

8.10 BiBTeX

8.10.1 citations

This platform would offer academics no value if it couldn't support citations, hence the `cite` command has been added, with the ability to add BiBTeX references into the database.

The syntax right now is limited to this form:

```
\cite[pagenum]{name}
```

where `pagenum` must be a number, and `name` is the has in the name provided in the BiBTeX.

First, navigate to the main table of contents for the book you are editing, and you will see a button that says "Add Reference":

Click on that and you can enter a BiBTeX reference. Here is an example in use:

"In a single stroke, Engelbart experienced a complete vision of the information age. He saw hiself sitting in front of a large computer screen full of different symbols " (Markoff, 2005, 9)

```
@book{dormouse,
  author    = {John Markoff},
  title     = {What the Dormouse Said},
  publisher = {Penguin Group},
  year      = {2005}
}
```

When you are finished, you can download your .bib file:

9 JS Implementation

9.1 Frontend

Some may notice that this application is what they call a "single page application". That means that the entire page is loaded once, and after that, only small pieces of the page are re-rendered. The Web page performs more like an application than traditional Web pages.

A lot of work went into ensuring an extensible and modular frontend infrastructure was in place. Managing the hierarchy of view rendering with many complex components on a web page can get tricky. As the system grew, it's complexity did as well. It became very important to build a system to last, a system that enables the addition of new features and makes it easy to find bugs.

> Unfortunately, as programs get large and complicated, as they almost always do, the adequacy, consistency, and correctness of the specifications themselves become open to doubt, so that complete formal arguments of correctness seldom accompany large programs. Since large programs grow from small ones, it is crucial that we develop an arsenal of standard program structures of whose correctness we have become sure (Abelson and Sussman, 1996, 10)

The "arsenal" of programming structures includes utilizing a multitude of programming patterns such as: module pattern, facade pattern, mediator pattern, observer pattern, and model-view-controller to name a few. It was crucial for the success of this platform, which was achieved with the help of aggregate technologies, and I would like to list off a few:

1. MathJax: used to render math
2. Aura: used for message passing infrastructure to decouple the application enabling modular development
3. Backbone: provided an MVC and some communication between server and client
4. RequireJS: provided a way to dynamically include and manage JavaScript libraries
5. Backbone Layout Manager: managed the rendering of nested views, particular provided great interface for "beforeRender" and "afterRender"
6. Async.js: provided great interface for asynchronous control flow, facilitated development of application controller and data flow
7. CodeMirror: great, super simple code editor used to edit the TEX
8. D3: jQuery-like interface to manipulating SVG, didn't really use it for what it can really do, but was useful
9. bootstrap: provided CSS and JS for respsonsive design enabling display on multiple screen sizes
10. jQuery: required by Backbone, but also in general a good way to manipulate the DOM
11. underscore.js: a JavaScript utility belt that provides a lot of functionality while maintaining cross-browser gotchas
12. handlebars.js: a great library for rendering client side templates

9.2 End to End JavaScript

This entire website was built with one language, JavaScript. The main duties of the backend was to provide an *Application Programming Interface* (API) to the frontend. The data was stored in a MySQL database, and the server was built on top of node.js. The list of technologies that comprise the backend are:

1. nginx: a super fast, light-weight web server
2. node.js: a server-side JavaScript platform to build scalable applications
3. express.js: a web application framework built on top of node.js
4. jugglingdb: great library for handling MySQL connections through a standard ORM interface

10 Limitations and Improvements

10.1 A Sketch

Because I am only one person, I had to scope this project in many areas, and it is not a perfect system. There are many elements to create a mapping for, beyond the scope of this project, in the web to provide socket connections between devices, and also various recording and other uses of device sensors that could be expressed through typesetting.

Think of this project as the first sketch or proof of concept of a much bigger idea with much broader strokes—deeper in mathematics and even across other fields. The main point is to bring mathematicians closer to Web publishing that they are currently, by mapping typesetting to interactive programming:

> Another way to span the gulf is for the users to change their own conceptualization of the problem so that they come to think of it in the same terms as the system. In some sense, this means that the gulf is bridged by moving the user closer to the system (Hutchins et al., 1985, 328)

10.2 Additions

New features that could make the platform better would be the ability to easily upload images and datasets. I think it would be great if a researcher could visualize the data that they are writing about in real-time.

Also, there could be some post-processing of TEX to optimize some of the generated elements instead of requiring everything to be done at the client. This could improve some areas of performance and do syntax checking for PDF output, etc. It would also provide the ability to determine objects referenced in the bibliography better.

Figures aren't as structured as they could be, and tables don't exist yet. They definitely should be implemented.

The ability to allow users to upload their own macros. This can be very useful, espcially for porting a lot of older documentation that relies on macros to render properly without a major refactor.

Enabling the ability to add functions that generate PSTricks code in JavaScript. This way the platform can evolve with the authors, and abstract the TEX away over time as people become more programming literate.

From a Web standpoint, caching could be implemented, and a few other improvements as time permits.

11 Conclusion

11.1 What is the Assembly Language of Digital Publishing?

In 1984, Abelson and Sussman brought up an intriguing idea: "An exquisite engineering art has been developed balancing between multiplicity of function and density of devices. In any event, hardware always operates at a level more primitive than that at which we care to program" (Abelson and Sussman, 1996, 1). Today, software operates at a level more primitive than at which (some) care to program.

More specifically, aspects of digital publishing concerning things like the user interface are also at a level more primitive than at which some care to program. If we simply look at blogging platforms or Facebook, we can see that people like to "publish", but don't want to have to worry about creating the user interface. Especially with books, it is the *text* that is creative, the layout is expected.

TEX has already been the de facto standard for publishing academic work, so why not combine the standard in academia for authoring with the standard de facto for Web consumption (HTML)? This bridges the gap between academic publishing and cutting edge technology.

"Unlike programs, computers must obey the laws of physics" (Abelson and Sussman, 1996, 1). This quote suggests the loose structure and nature of the software developing environment, and user interface design is far more free from structure. TEX provides us with some "laws" to obey in order to design the output of a text and graphical language around. Hence, we can attempt to create a synthesis of a structured user interface specification (TEX) and a structured functional specification (HTML5) to provide a publishing platform for the current and next generation.

This project presents itself as a Sketch of what authoring the academic Web could be like, if we can take two standards bodies and combine them. It also opens up the door to cross more people over the Web authoring bridge, because not all LATEX authors know HTML5.

That's not to say people shouldn't learn to program, but some should still be able to express ideas digitally without having to worry about how to structure their ideas. That was the beauty of TEX and LATEX, they provided a means to create beautiful published documents. The Art is where we can blend these two standards bodies; contributions like MathJAX, Backbone, Aura, and others can help people express their ideas by *providing higher levels of abstraction so the authors don't have to worry about the mechanisms by which the technology is rendering their works.* It is in these environments when people can express themselves freely.

11.2 Interactive Text and Learning

Just as it became important to author textbooks near the end of the 19^{th} century, in the 21^{st} century we've seen it's become important to program. More and more students are learning to program, but not everyone

is a programmer, just like everyone wasn't an author in the 19^{th} century. Programmers are like the modern authors during an era when computer literacy is increasing. However, if we can use programming to enable some non-programmers to communicate technical ideas, then we can enable a larger population of "modern" authors.

What about *interactive* authorship?

History has told us about how the democratization of knowledge has been inhibited by literacy, and that it's been used over societies to exercise power and dominance over subpopulations (Collins and Blot, 2003, 8). How can educators make students literate if the ability to control machines is indeed the key to a new era of literacy of this century?

As society progresses today, the amount of people who work in front of computer are increasing (Collins and Blot, 2003, 1). It's becoming more important in the modern world to be computer literate in general. Programming languages are built on top of languages like English, and can be used to express extremely complex ideas. If a person learns to code, perhaps the ability to express their ideas more closely to the actual concept of it in their mind can increase, or at least the number of ways they can store an idea in their mind. The analogy for the current use of the alphabet becomes like an ideogram, and programming will emulate the combination of ideograms to represent more complexity.

We speak more than words these days through technology—it's Time that software is accessible to everyone through Language, and the Web is the next big step towards this movement. But we also have to understand that it takes time, and not everyone is an "author". But for those who are interested, how can we bridge the gap?

TEX is and has been the academic publishing standard, and HTML5 is the *lingua franca* of the Web. Within the unification of these two languages we can find a simple prescription—using TEX for how things should "look" and HTML5 for how things should "function". TEX is also closer to what most people understand as language, since it primarily uses text to write papers and books as opposed to programs—it is less abstact. Mapping a typesetting language to programming can help us build a robust online platform for documenting math and science, and one that is built to last.

Interactive experiences provide deeper learning because we use more parts of our brain, so authorship of these types of systems can catalyze the learning process. Students can learn concepts more effectively when presented with a digital textbook that provides interactive, mathematical diagrams, than compared to their dusty, scholastic counterparts.

When questioning whether or not computers are providing something positive in our lives—the computer is good. It *is* the future of the textbook. We *learn* from computers.

> Most of the time, a person sits down at her personal computer not to create, but to read, observe, study, explore, make cognitive connections, and ultimately come to an understanding.

> This person is not seeking to make her mark upon the world, but to rearrange her own neurons. The computer becomes a medium for asking questions, making comparisons, and drawing conclusions—that is, for learning (Victor, 2006, 7).

To accelerate education, we need to first provide immersive learning experiences, and also accelerate the authorship of such interactive material. We can do the later by providing a language that expresses interactivity in a syntax that is as close to resembling mathematics as possible, to enable the common math professor to express their ideas through digital medium that their students can engage with.

11.3 The Future of Education

The convergence of standards proves to be a viable platform for moving society forward. The result of such agreements is a conduit for human ideas.

The aggregate history of computer science, literacy, and language all seem to assemble the idea that taking advantage of *existing* standards is conducive to the democratization of knowledge through communication systems formed as a result of the collective system. Gutenberg's life work took existing technology of the press and printing to create a movable type system that revolutionized literacy and education. The infrastructure of the Internet and the agreed standard of TCP/IP enabled Tim Berners-Lee to write HTTP, spawning the World Wide Web, which has exploded information across our Earth and made us a more global society in general.

Ideograms have metamorphosed into alphabets and languages, which have transformed into written words in books and now travel through the the Internet as bits that make up 1's and 0's. The Information Age is here, and we can use computers for something positive: learning.

Computers have shown to be a positive part of our civilization, and history may be telling us that command of the computer is the future of literacy. In the future, end user programming will become more important. People will go from coding for others to coding for themselves—this is how we can *truly* minimize the gulf of execution.

The expert in a given field may not be able to fully express their ideas if they must hire a developer or programmer, thus providing an interface to creating interactive environments through a language that they already use can close this gap, enabling the most optimal experiences for students learning.

> Alternately, a designer can draw a series of mockups, snapshots of how the graphic should look for various data sets, and present these to an engineer along with a verbal description of what they mean. The engineer, who is skilled in manipulating textual abstractions, then implements the behavior with a programming language. This results in ridiculously large feedback loops– seeing the effect of a change might take a day instead of a second. It involves coordination and

> communication between at least two people, and requires that the designer justify herself—she must convince the engineer and possibly layers of management that each change is worth the engineers time. This is no environment for creative exploration (Victor, 2006, 43)

It's like letting somebody build a robot to do something for you, it's kind of clunky if you could just do it yourself! People should be writing their own software. Today's Web feeds and interfaces are the metastable relation between the next interfaces created by non-web-developers with written and spoken languages.

Until then, we can use current standards as a bridge until we reach new technological horizons. It is the Art that allows us to bridge the gap for those that are interested in TEX and LATEX, and that is the expression which this project strives.

This platform is not a silver bullet, but it's a first step toward looking at our history to see what worked—combining standards. There are systems for graphics creation that are more powerful, but not as powerful in the audience they can reach as a result of the interface they provide. It's important to indoctrinate our society towards a new literacy, and guide them using small steps, instead of taking giant leaps where only a portion of society is literate.

The future of education will be affected by the future of programming and technology—the work done in the late 1950's and early 1960's set the stage for contributions that have shaped the technology timeline today. *What we build today will be used by our children tomorrow.* As ubiquitous computing pushes forward (e.g. when printed electronics are embedded in milk cartons and a kid can buy a Popsicle from the ice cream truck with his glasses), we need to find what will push us further.

In the end, the crucial point is that computers push us far beyond what we were ever capable of before, and will continue to do so. Using technology to create *increased levels of abstraction* for the documentation of science can push education and society forward. Higher-level learning will result in a smarter society. Using knowledge from aggregate sciences in the creation of supplemental learning tools can build positive foundations for our future's successful and meaningful experiences in higher education.

References

Abelson, H. and Sussman, G. J. (1996). Structure and Interpretation of Computer Programs. 2nd edition, MIT Press, Cambridge, MA, USA.

Aristotle (1855). On the Soul.

Baecker, R. M. (2008). TIMELINES Themes in the early history of HCI—some unanswered questions. interactions *15*, 22–27.

Bisson, C. (2007). Open Systems, Formats, and Standards. Library Technology Reports *43*, 21 – 27.

Blomberg, J., Burrell, M. and Guest, G. (2003). The human-computer interaction handbook. chapter An ethnographic approach to design, pp. 964–986. L. Erlbaum Associates Inc. Hillsdale, NJ, USA.

Bozdoc, M. (2003). Resources and Information for professional designers. `http://mbinfo.mbdesign.net/CAD-History.htm`.

Cerf, V. (2009). The day the internet age began.

Cerf, V. G. and Kahn, R. E. (2005). A protocol for packet network intercommunication. Computer Communication Review *35*, 71–82.

Collins, J. and Blot, R. K. (2003). Literacy and Literacies: Texts, Power, and Identity. Cambridge University Press.

Eisenstein, E. L. (1979). The printing press as an agent of change. Cambridge University Press.

Ellsworth, N. J. (1994). Literacy: A Redefinition. Lawrence Erlbaum Associates.

Griffin, S. (1999a). J.C.R. Licklider. `http://www.ibiblio.org/pioneers/licklider.html`.

Griffin, S. (1999b). J.C.R. Licklider. `http://www.ibiblio.org/pioneers/lee.html`.

Griffin, S. (1999c). J.C.R. Licklider. `http://www.ibiblio.org/pioneers/andreesen.html`.

Hayashi, Y. and Ikeda, M. (2004). A design environment to articulate design intention of learning contents. International Journal of Continuing Engineering Education and Lifelong Learning *14*, 276+.

Hefferon, J. (2012). What are TeX, LaTeX, and friends? `http://www.ctan.org/what_is_tex.html`.

Hutchins, E. L., Hollan, J. D. and Norman, D. A. (1985). Direct Manipulation Interfaces. HumanComputer Interaction *1*, 311–338.

Incorporated, A. S. (1999). PostScript: Language Reference. `http://partners.adobe.com/public/developer/en/ps/PLRM.`

Kamat, V. R. (2009). Special Issue on Graphical Three-Dimensional Visualization in Architecture, Engineering, and Construction. Journal of Computing in Civil Engineering *23*, 309 – 310.

Knuth, D. E. (1973). The Art of Computer Programming, Volume I: Fundamental Algorithms, 2nd Edition. Addison-Wesley.

Knuth, D. E. (1986). The TeXbook. Addison-Wesley.

Larkin, J. H. and Simon, H. A. (1987). Why a Diagram is (Sometimes) Worth Ten Thousand Words. Cognitive Science *11*, 65–100.

Licklider, Robert, Licklider, J. C. R. and Taylor, R. W. (1968). The Computer as a Communication Device. Science and Technology *76*, 21–31.

Markoff, J. (2005). What the Dormouse Said. Penguin Group.

McDermott, B. (2001). Decoding Egyptian Hieroglyphs. Chronicle Books.

Mendicino, M., Razzaq, L. and Heffernan, N. T. (2009). A Comparison of Traditional Homework to Computer-Supported Homework. Journal of Research on Technology in Education *41*, 331–359.

Myers, B., Hudson, S. E. and Pausch, R. (2000). Past, present, and future of user interface software tools. ACM Trans. Comput.-Hum. Interact. *7*, 3–28.

Redmond, K. C. and Smith, T. M. (2000). From whirlwind to MITRE: the R & D story of the SAGE air defense computer. MIT Press.

Rees, F. (2006). Johannes Gutenberg: Inventor of the printing press. Compass Point Books.

Sanders, N. (2008). An industry perspective on the beginnings of CAD. SIGCSE Bulletin *40*, 128–134.

Smith, G. G. and Ferguson, D. (2003). Diagrams and math notation in e-learning: growing pains of a new generation. International Journal of Mathematical Education in Science and Technology *35*, 681+.

Sutherland, I. E. (1988). Sketchpad a man-machine graphical communication system. In 25 years of DAC: Papers on Twenty-five years of electronic design automation pp. 507–524, ACM, New York, NY, USA.

Tames, R. (2006). The Printing Press: A Breakthrough in Communication. Heinemann Library.

Taylor, I. (2003). History of the Alphabet: Semitic Alphabets Part 1. Kessinger Publishing.

Taylor, R. W. (1990). J.C.R. Licklider. `http://memex.org/licklider.pdf`.

Victor, B. (2006). Magic Ink: Information Software and the Graphical Interface. `http://worrydream.com/MagicInk/MagicInk.pdf`.

Winograd, T. and Flores, F. (1986). Understanding Computers and Cognition: A New Foundation for Design. Ablex, Norwood, NJ.

Zandt, T. V. (2007). PSTricks User Guide. http://mirror.unl.edu/ctan/graphics/pstricks/base/doc/pst-user.pdf.