* * * * * * * * * * * *

Letters et alia

* * * * * * * * * * * *

To the Editor:

I wonder how much serious interest there is amongst TUGboat readers in radically simplifying and speeding up the capture and presentation of complex mathematical text. I have designed an unexpected 'mutation' of the standard Qwerty keyboard which, in mock-up form, allowed—with generous allowance for making and correcting mistakes—simulating the capture of a standard book page full of equations with double levels sub/super-scripts, fractions, differentials and integrals, etc., in a few seconds over twelve minutes. Commercial typesetters estimate at least thirty minutes would be needed even on the most sophisticated equipment presently available. I have also found new and unexpected principles for displaying several typesizes and typefaces in an immediately and exactly recognisable way (without using graphics) that help increase error awareness. The speed of the keyboard comes from its physical design, rather than other 'tricks'; people quite inexperienced in typesetting have remarked how easy it is to use and set complex work that would have completely puzzled them beforehand—it is good for use by relatively unskilled people.

I would like to set up production for several kinds of units if there is sufficient interest of a serious nature. However, until the advent of TEX the worldwide interest in special maths terminals has been minuscule; it is now uncertain, but probably greater. That is why I would like to hear of interest from TUGboat readers.

There are several useful units in mind. An "Idiot Keyboard" for those well versed in TEX, that merely generates editable code and/or TEX code as a direct output; a "Semi-Intelligent Keyboard", basically an 'Idiot Keyboard' with a video output and able to interact with a host; an "Intelligent Keyboard", with a monitor and two floppy discs, able to read editable code and generate TEX code from that, and assistance facilities (e.g., spelling as well), also interacting with a host; a "Realistic Display Maths Terminal", having the performance of an 'Intelligent Keyboard', but giving realistic display of maths, that is also fully editable on screen (no coding shown) and showing typefaces and sizes in an immediately recognisable way, also assistance facilities and interaction with a host; a "T&M Host" also having a "Realistic Position Maths Display" (graphics) and a proofing printer. A 'Realistic Position Maths Display' terminal would allow any maths office typist (or perhaps any other) easily to set the most complex material and automatically generate and edit TEX code, for example.

I would very much like to hear from people who have a real interest in such units and/or systems. The nature of the response to this enquiry will very much determine whether there is significant interest. Items would be expected to give substantial savings in salaries of dedicated programmers, and compare well with currently available equipment.

J. M. Cole
17 St Mary's Mount
Leyburn
North Yorkshire DL8 5JB, England

* * * * * * * * * * * *

FORMATTING A BOOK WITH TEX: EXPERIENCES AND OBSERVATIONS

Michael Sannella
MIT Laboratory for Computer Science

In January of 1980 I was hired by Professors Harold Abelson and Andrea diSessa of MIT to format a book they had written: *Turtle Geometry, The Computer as a Medium for Exploring Mathematics.* They had used a computer to write and edit the book—the text was stored on-line—and they wanted me to use TEX to produce the final formatted copy. In part, they hoped that using a computer formatter would be cheaper than traditional typesetting. However, they were also interested in the experiment of publishing a computer-formatted book, and curious about the ways that computer formatting systems could change the relationship between authors and publishers, giving an author more control over a book.

When I was hired, I didn't know anything about TEX, or about book publishing. In the process of formatting the book (which took more than a year) I learned a lot about TEX, about the problems associated with book-quality formatting, and about the interaction of computer formatters with the world of book publishing. This article is an attempt to record my experiences formatting *Turtle Geometry* and some thoughts I have had concerning computer formatters in general. I hope that information will be useful to other people who are trying to format large documents, such as books, using computer formatters. I will try to talk about general problems I encountered rather than about the details of how I fixed each problem. Also, I will try to be as simple as possible—you won't need to know anything about TEX to read this article.

The first part of this article deals with the experiences I had formatting *Turtle Geometry*. Interacting with the people at MIT Press, learning TEX, and using TEX to format the book all presented problems. Many problems stemmed from the fact that I was using TEX, an extremely complex formatting program. However, all of my problems couldn't be blamed on TEX. *Turtle Geometry* was a very large, very complex document, with innumerable difficult figures, equations, etc. It would have been difficult to format this book using any method. Eventually, I dealt with all of these difficulties, and produced an extremely good-looking book. (*Turtle Geometry, The Computer as a Medium for Exploring Mathematics*, MIT Press, June 1981)

Since I finished formatting *Turtle Geometry* I have been thinking about computer text formatters. In the second part of this article, I discuss some of my ideas on the design of text formatting systems. My experiences formatting *Turtle Geometry* give me an interesting perspective for looking at this area. First, I talk about the problem of designing formatting languages which give the user the correct level of control over the formatting process. Then, I discuss the problem of integrating high-level aesthetic formatting goals into a computer formatting system. I don't claim to have any great answers to these problems—I just want to point out some problems that future text formatters will have to deal with.

I would appreciate receiving any comments that anyone may have concerning this article. I can be contacted be sending U.S. mail to:

Michael Sannella
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

Or (for those of you with access to the ARPAnet) by sending computer mail to MJS at net site MIT-AI.

Formatting *Turtle Geometry*

I worked on this project part-time from January, 1980, through March, 1981. My job was defined very simply: to produce a camera-ready copy of the book, using TEX. Aside from that, nothing was specified. I was confident that this goal was achievable, since Knuth had designed TEX specifically to format his books. However, I knew that I had a lot to learn about producing book-quality copy. During the fifteen months I worked on this project, I had a good opportunity to observe the interaction between the two worlds of computer text formatting and book publishing. Computer formatters are being used increasingly for large commercial formatting projects, so it is necessary to investigate how they can be used most effectively. I hope that this r of my experiences formatting *Turtle Geometr* be useful to those people actively investigatin area.

Dealing With MIT Press

One of the first things I did after being to format *Turtle Geometry* was to talk wit people at MIT Press (the publishers of the and hear their views on the project. The editor and the book designer were not com scientists—they were mainly interested in pr ing a book. However, they were intrigued b idea of using a computer formatting system, in of traditional typesetting, and were willing to ticipate in the experiment of formatting the book by computer. They made a few things first, the quality of *Turtle Geometry* must not as a result of doing the formatting by comput the past, a few people have published books (m computer science textbooks) formatted using p tive text formatters and low quality printers, looked horrible, and were impossible to read. MIT Press was not willing to publish a book of low quality. I was expected to produce comp generated output fine enough so that, when i photographed and printed, the book it wou virtually indistinguishable from a book types ing the normal methods. They were not willi sacrifice quality for the sake of our experime second condition was that they did not want t more work into this book then they would hav was typeset normally. There would be little r to use the computer if it took more work on part. Within these restrictions, however, they willing to be flexible. They were interested in we could do using a computer formatting syst

The book designer gave me a s specifications, written on a standard form. Th cluded such information as the page dimensior placement of headings and page numbers, the to be used with different types of text, etc. specifications had been chosen so as to mak book as beautiful and readable as possible. designers use their experience with book forma and a long-cultivated sense of aesthetics, to a combination of design specifications such th total effect is pleasing to the eye, as well propriate for the type of book being published is the art of book designing.

Turtle Geometry is a textbook dealing mathematics, physics, and computer science. tains mathematical equations, computer pr listings, and figures, interspersed with para of text. The book designer decided how

of these objects should be formatted, so that the book would be readable, and would have the right "style." Each of these objects seemed to have fairly simple formatting—there were no fancy footnotes or other objects requiring complicated formatting—so I didn't anticipate any problems with the formatting of individual objects. However, the book as a whole was sure to present some problems, because there were literally hundreds of figures and equations that had to be formatted. I hoped that using a computer formatting system would allow me to cope with this problem better. In retrospect, formatting individual equations and figures didn't present many problems. The hard problems arose when I had to deal with the formatting of the book as a whole. In part this is because TEX can handle low-level details beautifully, but cannot deal directly with higher-level formatting concepts. I will discuss this problem more thoroughly later.

LEARNING TEX

After talking with the book designer, I had to learn about TEX. I knew that Donald Knuth at Stanford University had developed TEX in order to format his series of books, *The Art of Computer Programming*. Supposedly, TEX was a book-quality typesetting system particularly optimized for formatting mathematics. So I picked up a copy of Knuth's TEX manual and proceeded to read it.

My first impression was that TEX was very complicated. Knuth's manual tries to present the language "gently," a little at a time, so that the reader is not overwhelmed, but it only succeeds in being confusing. This style only served to make it difficult to use the manual as a quick reference. Eventually, I wound up reading the manual cover-to-cover three times, until I had a good idea about how the whole TEX system worked.

TEX can be used to produce very high-quality formatted output on a high-resolution printer. However, this comes at a price. The TEX user has to specify exactly how he wants to format his document using a complicated formatting language. TEX is an improvement over many other formatters in that it uses some of the vocabulary and concepts of printing and typesetting, but it still requires the user to specify the formatting in excruciating detail. In general, I would characterize TEX by saying that its "output" is good, but its "input" is bad. It is possible to produce beautifully-formatted copy, but the specifications necessary to achieve this are very complex.

Rather than explicitly specifying the exact formatting for every object in a document, TEX provides a macro facility which allows one to encapsulate and name a sequence of commonly-used formatting specifications, and to refer to them by using the macro name. TEX's macros are implemented using simple text substitution—when you reference a macro, TEX acts as if the macro text were substituted for the macro reference. Macros can be defined to take arguments, which are inserted into the text of the macro in the same way that the macro is inserted into the text file. It was obvious that I should use macros to specify all of the different objects that would appear in *Turtle Geometry*. I began creating a file of macros. Initially, they were very simple, and I was able to get them working very quickly. I gradually refined them, changing the macro definitions until they fulfilled the specifications for the book.

I had a lot of problems using macros because of the way that they are implemented in TEX. In most cases, simple text-substitution worked well, but sometimes I wished that it were possible to write "smarter" macros. One problem was that the exact effect of a set of formatting commands depends on where they occur. A TEX macro has no way of detecting where it has been placed, so you can get very strange results by putting a macro in the wrong place. A similar problem occurred because the arguments to a macro are blindly inserted into the macro text, and there is no way to check whether they are appropriate. If the arguments are not of the correct form, the macro may do something quite unexpected. (I give an example of this second problem below, in the section on formatting figures.) Both of these problems happen as a result of people making mistakes—using macros incorrectly—so it could be claimed that this is not a deficiency of TEX. However, this situation influences the way that people use TEX. In order to use a macro correctly, you need to know exactly where it can be used, and the precise form of all of its arguments. Therefore, all macros need to be supplied with extensive documentation concerning their use. This is the same documentation problem that people have to deal with when maintaining programs written in computer languages, raised to a higher degree. Most computer languages allow programs to be somewhat isolated from each other, with only minimal communication between them—this allows one to consider the effects of one program apart from the others. Macros in TEX get much of their meaning from the context within which they are inserted. A macro is meaningless by itself. Consequently, this makes the tough problem of documentation even tougher. While I was developing the macros for *Turtle Geometry*, I found that I had to write exten-

sive notes to remind myself exactly where and how to use my macros. In spite of this, I made a lot of mistakes which were difficult to discover and to fix. Luckily, the macros would not have to be used by anyone else. It would not have been easy to write macros that anyone could use. In general, I would not want anyone else to use my macros Unless they understood them completely, it would just be too easy to make a mistake. Because of this problem, I predict that it will be difficult to set up "libraries" of commonly-used TEX macros, in an effort to prevent duplication of effort. Instead, most people using TEX will have to start from scratch, and write their own macros.

One unfortunate characteristic of TEX is that it is difficult to debug your formatting specifications if there is some type of error. TEX has a large set of error messages, but these only signal when you specify formatting illegally. Most of the time, TEX formats your document without complaining, and there is something strange in the output. What can you do? Usually you can figure out where something went wrong, but sometimes this is very difficult. This problem is compounded by the use of macros, which may do unpredictable things if you use them incorrectly. This makes TEX very hard to use.

FORMATTING THE TEXT OF *Turtle Geometry*

Once I learned how to use TEX and developed the formatting macros, I began to edit the text of the book, inserting the macro invocations where appropriate. The book was first written and printed using a primitive text formatter called TJ6, so the text files (one per chapter) contained text interspersed with TJ6 commands. All of the TJ6 commands had to be removed, and TEX commands had to be inserted. I eventually deleted all of the TJ6 commands using a text editor, then scanned the straight text which remained, and inserted TEX commands as needed.

Most of the time formatting the text was no problem—the job was tedious, but not difficult. Occasionally I came across an object that had to be formatted somewhat strangely, and I had to do a lot of thinking and experimentation to make TEX do what I wanted. However, with most objects all I had to do was to insert the appropriate macro invocations, and TEX would format them correctly.

Using TEX's macro facility to specify formatting turned out to be very useful when I was not exactly sure how something should be formatted. For example, *Turtle Geometry* contained many vector equations. The authors and the book designer were not exactly sure how they wanted to represent vector variables. So I defined a macro called \vect, and used it to specify all vector variables. I printed out a section of the book several defining \vect a different way each time. By ing that single macro definition, I could repres vectors in an entire section by letters with on top of them, or by boldface characters. Th way to judge which notation was better was how it looked in print. Using macros allowed experiment with very little effort.

FORMATTING FIGURES

Of all of the objects in *Turtle Geometr* hardest to format were the figures. MIT Pre the authors had decided that the figures wo drawn by a technical illustrator, and paste the camera-ready copy of the formatted docu Therefore, spaces had to be left for the dra Formatting a figure proved more complicated simply leaving a specified amount of space on a however. The book designer had specified single figure should contain a caption, and bitrary number of subfigures, each with an op label, organized one or two or three in a row the page. I had to develop macros to allow construct complex figure boxes.

One problem I ran into was determining e how much space to leave for a particular figu I put in too much space, the drawing woul strange surrounded by emptiness. If I didn't enough space, on the other hand, it would possible to paste the drawing into the final c the document. Given a drawing, I measured give its height as an argument to a figure-gene macro. In most cases, this produced a good-l figure, with just enough blank space to paste appropriate drawing. However, there were number of figures which didn't have labels f subfigures. Because of the way that TEX macr implemented, there is no way for a macro to when it is given a null argument. When given label argument, my figure formatting macros s located space for a label, which resulted in a with too much space. Eventually, I solved this lem by subtracting an appropriate amount fr size I gave as an argument to the figure ma compensate for the extra space added for th existent label. This was a rather inelegant so In retrospect, it would have been better for have defined separate macros for figures with and figures without labels. The best solution, had had the capability, would have been to a figure macro that could detect when it is g null label, and format the figure accordingly could be greatly improved by extending its facility so that a macro could test its argume

PAGE BREAKING AND HIGH-LEVEL FORMATTING

As I have mentioned before, I didn't encounter many problems with formatting most of the individual objects in the book. To format a figure, or a program listing, or an equation, I just had to use the appropriate macro from the set I developed. Admittedly, constructing and debugging that set of macros was a slow and painful process, but I only had to do that once. The real problems began after I had formatted the individual objects and was trying to bring them all together. The book designer at MIT Press had given me certain aesthetic rules to follow when formatting pages. Unfortunately, TEX was not designed to deal with such high-level formatting concepts. As an example, consider the placement of figures within a document. Each figure is referred to at a specific point within the text. The readability of a document is improved if figures are placed so as to minimize the amount of page-flipping a reader has to go through to associate references with figures. For this reason, the book designer at MIT Press specified that a figure should be placed at the top of the page containing its reference, if possible. If that was not possible, the next best choice would be to have the figure on the facing page, to the left or to the right. If that too was not possible, then the figure should be on the next possible page later on in the document. Unfortunately, TEX cannot deal with figure placement in these terms. For one thing, there is no notion of "facing pages" in TEX. This is a serious failing, since commercial publishers design document formatting in terms of how each "spread" (facing left and right pages) looks. More generally, TEX does not give the user very much control over which page a figure is placed upon.

Another formatting rule I was given was that the pages should all be of the same length, if possible, but a page could be a line short or a line long, if that would help the formatting of the page. In TEX, the height of a page is set to a specific distance using a `\vsize` command. Page breaks are generated to fit as closely as possible to that limit, without going over. There is no provision in TEX for changing this number dynamically, on a page by page basis, subservient to other formatting needs. Within TEX all formatting decisions have equal weight, which sometimes unnecessarily restricts the possibilities for formatting something, and reduces TEX's power.

If I only had to deal with a small, simple document, the aesthetic goals mentioned above would not be very important. It is not difficult to place a figure in a small document consisting of a few pages, and TEX would probably position it correctly, using its simple figure-placement algorithms. However, *Turtle Geometry* was a very large document, with many figures and equations. When you have a large number of figures in a document, placing these figures becomes a much more complicated problem, and it is harder for TEX to place them correctly. It can't deal with the aesthetic concepts that come into play with large documents. The first time I formatted *Turtle Geometry* using TEX, it formatted the individual figures and equations correctly but the total effect was horrendous. The book was unreadable—of much lower quality than you would get using traditional human typesetting.

I had to find some way to correct this situation. First, I tried coercing TEX into doing the correct page formatting by adding formatting commands at places where it broke pages and placed figures incorrectly. This proved to be very difficult. The problem was that I could easily prevent TEX from breaking a page at a particular bad place, but it was harder to ensure that it would break it at a good place. I was never sure of the effect of adding any particular formatting command until I saw the text reformatted. Coercing TEX involved endless experimentation, just to get around the inadequacies of TEX's page formatting algorithm. I realized quickly that this method was unsuitable. I was spending my time fighting TEX, trying to merge its page formatting with my aesthetics, rather than worrying about the formatting of the book.

Rather than having both TEX and myself handle the page breaking and figure placement, I decided to simplify things by doing all of the page breaking by myself. To be exact, I inserted an `\eject` command every place that I wanted a page break, and specified to TEX that it shouldn't do any page breaks itself. So that I would not have to figure out by myself where all of the page breaks should go, I developed a macro called `\bookmark` which I inserted into each file containing a chapter of the book. The `\bookmark` macro told TEX that all page breaks before the point where it occurs will be handled by explicit `\eject` commands, but that TEX should attempt to generate page breaks from that point on. I eventually developed a routine for handling the page breaks in a chapter: I would insert a few `\eject` commands, move the `\bookmark` macro call forward to the end of my `\ejects`, and run the file through TEX. Seeing how TEX would format the next few pages helped me decide where I wanted to insert page breaks. Often, I didn't have to do anything more than insert `\eject` macros where TEX had broken the page before, but most of the time I wanted to move figures around, and change the

exact placement of the page breaks. It was necessary to do this work a few pages at a time, because each formatting change I made could send effects percolating through the rest of the document in unpredictable ways.

By doing the page breaking and figure placement myself, I was able to optimize the page formatting with respect to aesthetic rules that I could not express with TEX. Consequently, I produced a book of very high quality.

LOOKING BACK

Formatting *Turtle Geometry* was a long, difficult job. Everything did not go as smoothly as it may seem from what I have said above. One problem was that the macro development, the figure formatting, and the editing of the actual text of the book were all happening at the same time. It was very difficult to coordinate these separate activities. Another problem, which is sure to occur whenever a document is available on a computer system, was that of "freezing" the text of the document. When it is easy to change the text, authors are very reluctant to finish editing. There is always a great temptation to fix one more error. At some point, however, it is necessary to decide that the text will not change any more.

A typical example of the organizational problems I had to deal with was that associated with putting the figures in the book. It took a long time to get the figures drawn and checked. Until the drawings were in final form, I couldn't measure them and give these measurements to the figure macros. Therefore I couldn't do the page formatting, since the sizes of the figures critically affected the location of page breaks. To cope with this situation, I worked on each chapter separately, so that at any one time different chapters could be at different stages of production—the authors might still be editing the text of chapter 5 while I was sizing the figures for chapter 1, and formatting its pages.

I made many mistakes, and had to do some things several times. I was learning what to do, and developing new methods for doing things, while I was formatting the book. Everything would have been much better if all of the people working on *Turtle Geometry* had sat down before the work was started, and developed a routine for producing the book. We could have avoided a lot of problems, and produced a better book with less effort, if we had been more organized.

Thoughts on Text Formatters

Current computer text formatting systems suffer from many deficiencies, which become increasingly apparent as people attempt to use the more and more situations. Some of these lems are due to formatting languages that c cope with the aesthetic ideas that book desi have developed from long experience with docu production. Other problems result because fo ters are designed with small documents in mind cannot deal with the formatting concerns of documents. I don't claim to have "the solution all of these formatting problems. However, I come up with a few ideas about how text forma could be improved.

FORMATTING LANGUAGES AND CONTROI

Computer-controlled printers deal with ments at a very low level. To a graphics-ori printer, a document is a set of commands specify the exact positions of individual dots, and characters on a page. Obviously, peop not want to have to specify the formattir their documents in such excruciating detail. stead, they use a computer text formatter, could be considered as a translator betwee abstract, high-level formatting language an specific detailed language of a printer. A fo ter allows its user to specify document forma in terms which are more abstract and general, as words and paragraphs and lines of text. ing these specifications, the formatter decides each character of the document should be plac each page.

An important characteristic of a formattin guage is the level of detail with which the user specify the formatting of a document. Suppose a formatting language deals only with high-leve matting concepts, and the user wishes to specif formatting of a particular object a little mo plicitly. This could happen if the formatter d format something exactly right automatically the user wants to fix this. Another possibility i the user may wish to create a one-time specia object, which needs to be formatted strangely the user is willing to take the extra time to spe in detail. (As an example of this, in *Turtle Geo* there were only two or three tables, and it was to specify them in detail, rather than devel general macros for formatting tables.) In this the only solution is for the user to try to the system into formatting correctly. This m volve tricking the formatter, making it treat objects like other objects, and trying to second- the formatting algorithm. This is very diffic do, it requires an intimate knowledge of the nal workings of the formatting system which users may have, and it is extremely inelegant

problem is really that the way such formatters are designed, there are simply some documents which cannot be produced. In the interest of making a formatter easier to use, the set of possible documents that could have been formatted with it has been constrained.

Now, let us consider the opposite extreme, where you have a very low-level formatting language, and you wish to do high-level formatting. In other words, you don't care exactly how your document looks, as long as it conforms to certain high-level formatting goals, such as having lines filled to a constant length, etc. Using a low-level formatting language, it should be possible to produce a document conforming to these high-level constraints, but you will be required to specify a lot of low-level details that you are not really interested in. Aside from the inconvenience of specifying the formatting in this much detail, this makes things very difficult if you wish to change your document later. The more detailed your specifications, the more work has to be done in order to change them. When you work with specifications at a low level of detail, you essentially have to calculate the formatting yourself, rather than letting a computer do it.

There is another, rather subtle, problem associated with formatting languages which use descriptions at a lower level than you need for a particular document. The greater the detail in which you specify the formatting of a document, the more you constrain the possible ways that it could be formatted. If you have to use a low-level language, you wind up making a lot of fairly arbitrary decisions about how something should be formatted at a low level, since you have to specify something. However, all of the formatting specifications are heeded by a computer formatting system, whether they are important to the user, or whether they were just specified arbitrarily. The more constrained a formatter is, the smaller the space of possible ways a document could be formatted, and the more likely that there will be no formatting which is acceptable. When a user is specifying how his document should be formatted, he should use the least possible amount of detail, while still mentioning those things important to him. In this way, the possibility is increased that the formatter can find an acceptable way to format his document.

Early text formatters were designed around the idea that documents were fairly simple, consisting of text organized into paragraphs, with an occasional heading or footnote. These programs would take the user's specifications, and format a document using rules internal to the formatter. It was possible to modify exactly how the formatting was done by specifying the values of certain parameters, but the formatting was essentially done by one "smart" program. These early formatters present a good example of formatting languages which work at too high a level for many formatting tasks. As long as the shape of your document conformed to the ideas explicit in the formatter's abstract paragraphs and words, it would be easy to to format your document. However, if your document contained something out of the ordinary that the program was not equipped to handle, there was little that you could do. No matter how much effort you were willing to expend to specify the detail, these early formatting languages wouldn't allow you to. The user simply had very little control over the formatting process. The shape of a document had to be specified in terms of the objects the formatter was built to deal with, then the user would just have to "throw it to the winds," and hope the formatter produced something acceptable.

TEX is similar to these early formatters, in that it deals with abstract ideas such as paragraphs and pages, and the user only has indirect control over how these objects are formatted. However, TEX also can deal with the idea of "boxes and glue," which allow a user to specify an object at a very low, but manageable, level. This facility is very useful for handling "special case" objects, such as complex figures, which would be too complicated for TEX to deal with as built-in objects. In essence, TEX allows the user to deal with formatting at two distinct levels of detail, or control. Either you can "throw your formatting to the wind" and let TEX take care of it all, or you can specify the formatting of the object in low-level detail. This is an improvement over the earlier text formatters, which gave you no choice, but it still proves inadequate in many situations. The problem is, a user sometimes want to have control over the formatting, without having to specify *all* of the formatting explicitly. For example, consider the problem I had while formatting the Turtle Geometry book with breaking pages, and placing figures. When I let TEX do all of the formatting by itself, it would do it incorrectly, in part because it doesn't consider some of the aesthetic considerations I am concerned with. For a while, I tried tricking TEX's formatting algorithms into doing the correct thing, but this involved going through all sorts of complicated contortions. Finally, I was forced to deal with the problem at a very low level, by specifying all of the page breaks and figure placements myself, explicitly. I was able to get what I wanted, but it required a lot of work

on my part. Another issue in this situation was that once I had done the page breaking by hand, that essentially "froze" that part of the document. If any changes had needed to be made to the document after I had done the page breaks, it would have been necessary for me to do all of that work over again. It would have been better if I had the ability in TEX to specify formatting at a higher level, while still retaining control over the formatting.

People who design text formatting systems need to worry about this issue. One way that formatters could be improved is by giving more control to the user, at different levels of detail, so that it is not necessary to specify document formatting at more than the minimum amount of specificity. Looking at TEX in particular, one way to improve it would be to introduce "smarter" boxes—objects that are not quite as explicit and low-level as boxes are in TEX. For example, you could have an object called a "paragraph box," which contains text that should be formatted as a paragraph. Eventually, this may not be formatted as a single rectangular box—the paragraph may eventually extend across several pages. Without explicitly specifying the exact formatting of every character in the paragraph, you could express parameters appropriate to the level of abstraction of a paragraph, such as line length, degree of raggedness, whether the paragraph can be broken, etc. To format any particular object in a document, simply express it in terms of a formatter object which deals with the correct level of abstraction.

One problem with implementing this scheme is that it could lead to a complicated formatting language with an unmanageably large number of objects defined. A way to deal with this is to only define a few objects, which can be used to specify formatting at differing levels of detail. For example, consider a "box" object that acts in the following way: given a set of objects as arguments, it formats them like a paragraph, using default width, raggedness, etc. values. If these values are given as parameters to the box, it will use them instead. You can imagine a whole set of parameters and defaults set up so that a single box object could be used for formatting a paragraph (at the most abstract level) or used exactly like a box in TEX (if everything, including vertical and horizontal sizes, are specified). A single object, with many parameters and defaults, could be used at many different levels of abstraction.

High-level Formatting Goals

One problem I have mentioned regarding current text formatters is that they do not deal with some of the aesthetic rules that book designers have developed after years of experience with form documents. I had to explicitly specify page b ing for *Turtle Geometry*, because TEX coul deal with some of these concepts. Let us exa this problem more closely. When breaking pag hand, I had to keep in mind several aesthetic siderations. First, figures had to be position be as near to their references in the text as po (on the same page, or on the facing page, or o next page). Second, I didn't want a page bre the middle of a computer program listing, if possible. Third, the length of each page shou nearly constant, plus or minus a line. In ce special situations, I had to worry about othe thetic considerations, such as not having an equ directly under a figure, but this didn't come u often. Mainly, I was concerned with choosing breaks, and positioning figures, so as to prod document optimized with respect to those thre thetic rules given above.

Certain problems arise when dealing with level aesthetic formatting goals. It is difficu explicitly specify such goals in a form that a puter can work with. Consider what a *huma* to go through to format a document while con ing these goals. It is seldom the case, given a t formatting situation with a lot of figures, th of these formatting goals are compatible. C they are contradictory. For example, suppos have a page with a computer program listing i middle, and a figure reference near the top o page. Suppose that if you place the figure a top of the page, this pushes the text on the down far enough so that the computer progra to be broken between this page and the next Here you have a conflict between the rule that figures on the same page as their references an rule that doesn't want computer program li to be broken. Exactly what I would do de on the exact situation. If the page in questio a left-hand one, it might be possible to mov figure to the facing right-hand page. Alternat the place where the program was to be broke only a few lines into the program, I might just that page short. (notice that in this second may be breaking the third aesthetic rule.) M an aesthetic decision of this sort involves find compromise between several contradictory goa

The problem of formatting a document respect to certain aesthetic formatting goals c reduced to the problem of considering two tions, where a part of a document is formatt two different ways, and deciding which is "be Usually there are only a few possibilities to co

in any formatting problem, and a computer could afford to try the various possibilities, and pick the "best" one. The key to this problem is obviously determining what the word "better" means with respect to a set of high-level aesthetic formatting goals. It is instructive to examine a mechanism that TEX has for dealing with this problem on a lower level. In order to optimally generate line breaks and page breaks within a paragraph, TEX associates an integer with each paragraph known as its "badness." Every line break generates a certain "penalty," another integer, which indicates how "bad" that break is. Line breaks have large penalties associated with them if the break requires hyphenating a word, or if a line is too long, among other things. Page breaks in a paragraph have large penalties associated with them if they generate widows (the first or last line in the paragraph alone on a separate page). The badness of a paragraph is calculated as the sum of all of the penalties in the paragraph. The formatting algorithms inside TEX are designed to make line and page breaks so as to minimize the total badness of a paragraph. This is a fairly flexible system—users can specify penalties explicitly at certain points to indicate places that are particularly good or bad to break at—and it formats individual paragraphs very well. However, this type of system would not be suitable for dealing with more complex formatting constraints. "Badness" is a simple, one-dimensional "score" which can be easily calculated and used for comparisons, but it is not powerful enough to express some of the more complex combinations of higher-level formatting goals. Consider the aesthetic goals given above, dealing with figure placement and computer program listings. You can't simply take a badness score with respect to one goal, and a badness score with respect to another, and sum them to get total badness. This is like adding apples and oranges. Aesthetic formatting goals combine in more complex ways than that. In a particular situation, it might be possible to represent a situation in terms of carefully-adjusted penalties, but this wouldn't work in general. You need a more complex, multidimensional, way of comparing two situations.

The fundamental problem with implementing high-level abstract formatting goals in a text formatting system is the problem of achieving a goal. Supposing that you have a way of unambiguously stating fairly complex aesthetic rules, how is the computer to make sure that they are achieved? Exactly what can the formatter do if the formatting it chooses disobeys one of the aesthetic rules? Consider the low-level decisions that a formatter makes when formatting a document. The decisions about exactly which word to break a line at, or exactly how much to stretch a line, are fairly arbitrary. Unless the user specifically specifies it, the formatter has a fair amount of leeway as to exactly how the document is formatted. Ideally, the low-level arbitrary decisions could be influenced by the high-level aesthetic rules. I admit that I don't know how this could be accomplished, but this is obviously what text formatters should work towards. As an intermediate step, formatting systems should be designed that can express the concepts of high-level aesthetic goals. Even if they couldn't use these goals to influence the formatting process, it would be worthwhile simply to flag violations of these rules, pointing out to the user places in the document where one or another of the rules are broken. This by itself would lead to better formatters, producing better-looking documents.