Bijlage 11

DVIview, a new previewer

Gilbert van den Dobbelsteen
email gilbert@login.iaf.nl

abstract

DVIVIEW is a new viewer for the Windows platform. Key features: virtual fonts, rotated and colored text and performance. This article focuses on the development process and hilights some features of the software.

Introduction

About a year ago I met some other TEX users who complained there was no decent DVI viewer available for Microsoft Windows. The development for dviwin stopped, and most of all, dviwin didn't do virtual fonts. So I wondered 'how difficult can it be to develop a viewer?'. At that time I truely believed it couldn't be that difficult. So I waited a few months, thought things over and started programming in December 1997. I planned to release the first beta within six months. The beta will be released to a few people who have lots of experience using TEX, and lots of experience in crashing viewers (big grin here). This is needed since I only create small documents using TEX.

After the beta-test a final release will be made. At the time of this writing (april 1998) the viewer is in alpha test (no no, not beta yet) and I am making progress. I hope the beta is released when you read this.

A new viewer

Why another DVI viewer one might ask? There are several products and all have there advantages and draw-backs. Probably the best viewer available is Y&Y's DVIWINDO. I've seen it compared it with mine, and I must confess Y&Y has a very good product here. It only costs money. You can't buy the viewer, you must buy their complete TEX system (which is good I've been told). The best feature of DVIWINDO is the re-encoding on the fly. I don't know how they do that, because it's not simple.

The main drive for me to develop such a beast was the challenge and of course there isn't a suitable free viewer. The DVI file format is pretty straight forward and excellent when sending data to the printer. So I thought 'this is pretty easy to interpret' let's do something nice with it. After studying the Windows API (Application Programmers Interface) I still thought it would be a fairly easy job.

So you might ask 'tell me, what does this new viewer bring me'. At the moment I can only say 'nothing you haven't got'. But I hope to include lots of functionality in a single product so you won't have to use another viewer.

Developing this program came with a few problems:

- I never wrote a program for the Microsoft Windows environment. I am a reasonably experienced C/C++ software engineer, and my work deals mostly with embedded applications[1].
- I am quite new to TEX. I knew almost nothing about DVI files and nothing about processing them.
- How well will it perform. I hate slow programs and Windows programs tend to be slow, memory hungry, and always contain severe bugs.

Development process

How to develop such a program? The easiest way is to use Delphi, C++builder, or MS Visual C++, generate a frame-work and fill in the gaps. This was not my intention, as I've seen dozens of programs built this way and the result is almost always bad performance. So I wanted no luxurious development environment or supurb foundation classes. This resulted in good old ANSI C.

Since I almost forgot how to do things in ANSI C (I use C++ most of the time) I looked up some CWEB sources. A splendid source of information is the Stanford GraphBase (also from D.E.K.). Though most of the algorithms are beyond my understanding, it provides good examples of ANSI C programming. Especially for programs that are a bit larger than 'Hello world'.

I started with linked lists and some basic windows programming. A friend borrowed me his copy of Programming Windows 3.1 by Petzold[2]. I asked my employer if I could use the Borland C compiler, since we had a copy which has never been used in any project.

That's how things got started. After muddling around with Windows[3] I decided how to proceed. Since developing a user interface in ANSI C is a lot of work I kept it

1. Embedded applications: software things you see all around you but don't notice, like remote controls, cordless phones, your dish-washer and off-course your internet aware watch.

2. If you intend to program for Windows in ANSI C this is one of the best books available.

3. Ok, I am a stubborn guy, but Windows does a better job which is almost impossible to believe.

clean and simple. So there are not a lot of dialog boxes and menu's. Currently some things can be customized by menus and dialog boxes, other features must be customized by editing the ini file. I am working on that so don't be afraid. Programming dialog boxes for data-structures is straight forward but a lot of work.

I decided to use ATM (Adobe Type Manager) for rasterizing. I have no knowledge of PK fonts and currently I don't need them. The Computer Modern and AMS fonts are freely available in type1 format and ATM does a reasonable job here. Printing DVI files is also easier this way.

I also decided to do my own scaling. Windows can't be trusted on that. I've seen numerous applications with 1 pixel round-off errors when scrolling and that's not what I wanted. Since I had to scale anyway (DVI units are way too large for Windows to handle) I do the complete job myself. All scaling is done through fixed-point arithmetic. No floating point is used in calculating positions of characters. This results in great accuracy although the current results leave some wishes.

Fast viewing

The main problem is how to get things fast. TEX can position characters anywhere in the DVI file and you're never sure what you're up against. Furthermore, re-organizing characters (e.g. drawing them in a different order than TEX did put them in the DVI file) is risky business.

The easiest way to display a DVI file is to draw it character by character. This approach *always* works, and you are sure the characters are positioned correctly. But drawing each character separately takes *a lot* of overhead. Why? Because you have to call a Windows function for each single character. Calling Windows functions involves calling overhead and processing overhead for the drawing engine inside windows. So the trick here is to avoid calling a GDI function for each single character.

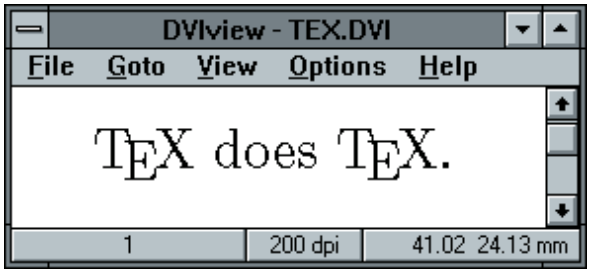

Figure 1. TEX demo (original)

I needed a function which could draw several characters in one call, but where *I* could determine the spacing between the characters. Fortunately Windows provides the function ExtTextOut. It takes a string, a position and an array of distances between the characters in the string. After doing several experiments with this function I decided this was the way to go.

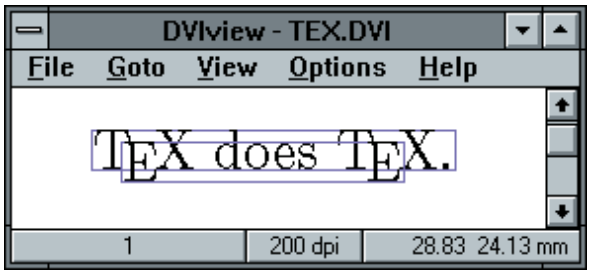

Figure 2. TEX demo (with bounding boxes)

Now take a look at the figures with this article. The first one (figure 1) shows a typical text string. The second (figure 2) shows the boxes calculated by DVIVIEW. The rectangles around the text show the actual internal data-structures in the program. Each rectangle results in exactly *one* call to the ExtTextOut function. So this piece will result in two calls, one for the string TX does TX and one for EE. Thus as long as things don't get too bad in respect to positioning, the program optimizes the output pretty well.

Gathering characters

Processing DVI files is similar to what DVItype does. In fact almost everything in the processing functions comes from DVItype[4].

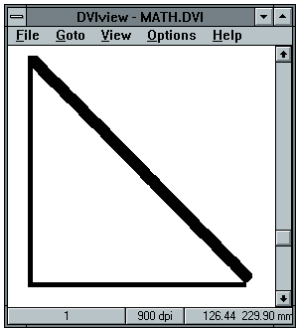

Figure 3. Excerpt from math.dvi (orignal)

When processing a DVI file I keep track of the position of the characters. DVIVIEW gathers characters which have the same vertical position so they can be written in one function call. This gathering process has some side effects. Currently the program has ten rectangular structures

4. I am quiet sure that DVItype does a good job here, since it was meant as a reference for people developing DVI processing software.

to keep track of vertical positions. This is a reasonable amount for normal documents. However when the same technique is applied to documents containing a lot of math (especially display math) the optimizations don't come out too well. See figure 3 and 4 for a example of math.dvi made by Kees van der Laan. As you can see several things are not correct here. First of all the sloped lines are drawn using dots. Although this is a valid TEXnique it isn't very fast for displaying. Also, when looked at high resolutions (1200 dpi in the figure) things are not completely correct. This originates from the DVI file.

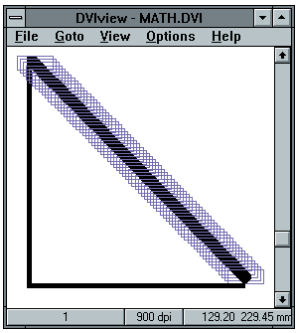

Figure 4. Excerpt from math.dvi (with bounding boxes)

Take a look at figure 5 and 6. The parentheses are gathered into a group, and the string $edx\pi$ is gathered here. This figure also shows another peculiar thing of TEX. The bounding box of the integral sign looks incorrect. This is not exactly true, but TEX uses the bounding box of characters to calculate where to put the limits. You can clearly see that the positioning of the limits is correct (horizontally and vertically). The upper limit is placed to right by using the italics correction. Is this clever or not?

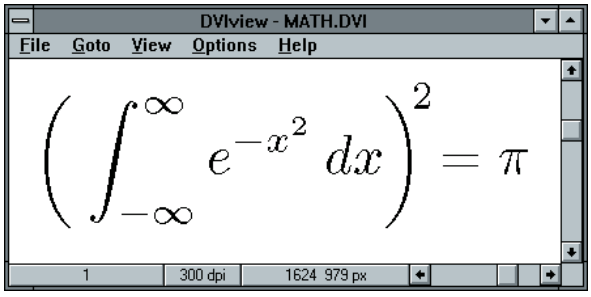

Figure 5. Excerpt from math.dvi

For viewing this leads to all kind of problems which are not there when you create a simple DVI printer driver. For example, the program uses the bounding box to determine what to update on the screen. So if you carefully scroll the window you would get incorrect results. This is shown in figure 7. This is easily solved by redrawing the entire screen but it would be nicer if the program could detect this.

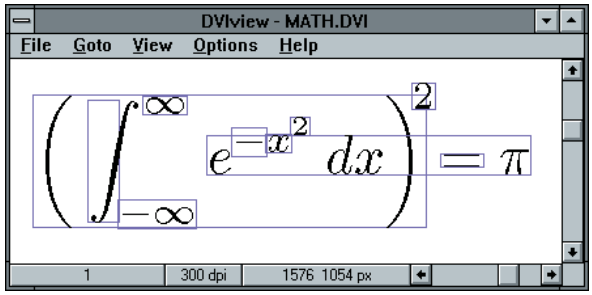

Figure 6. Excerpt from math.dvi (with bounding boxes)

There are many math characters which have inproper bounding boxes. This is due to the fact that TEX uses the bounding box information to position accents, limits or extensions (for ellipses). A solution for DVIVIEW would be to let Windows calculate the actual bounding box. I'll probably implement that sometime. For now it doesn't bother me too much.

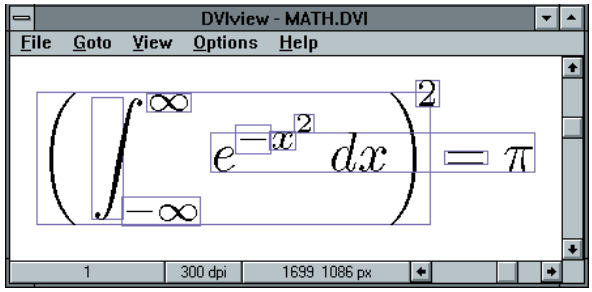

Figure 7. Excerpt from math.dvi (with bounding boxes)

The character gathering technique I use has a disadvantage: The characters are drawn in a different order than they originally had in the DVI file. Is this bad? Well I didn't encounter any problems yet, but that's because the rectangular arrays are flushed on some conditions:

- When a rule (rectangular black area) is typeset. It is possible to obscure some text by placing a rule on top of it. This has particular disadvantages when drawing mathematics. Many mathematics commands use rules to extend characters. For example `\sqrt` is a good example.
- Some `\special` commands. Especially the drawing commands. You could draw an oval box obscuring some

text. The text must be drawn first, so all the gathered information should be flushed into the drawing-list. This also holds for \specials involving figures.
- When using different colors to create shadowed text. You could draw a drop-shadow for a character string. If things are misinterpreted, the shadow is drawn too late.

Not every condition has to result in flushing the rectangular list, some intelligence could be built in. I am currently working on that but it's not always obvious what was meant by examining the DVI file.

I could for example check where things are drawn, and if some text is obscured or overlapped. Based on that I could generate things in the correct order. Since this is highly complicated and involves a lot of calculation, it will take time.

Current status

So how far am I? Well, not far. The first alphas are out and I locate bugs every day. The program is reasonably stable and doesn't generate too many General Protection Faults. I hope it will be bugfree some day. There's a lot on my wish-list and probably the program is never finished.

The performance is excellent. I compared the program with DVIWINDO and with xdviwin32. DVIview is two to three times faster. It is also a small program so there's hardly any startup time involved. As features will be added the performance will probably degrade, but I think I'll keep ahead of xdviwin32 and DVIWINDO.

The quality of the output is good. Ok, xdvwin32 does a better job because you can't beat anti-aliased PK fonts. Every time I see the results of PK-fonts I am astonished by the quality. We live in the late 90s and some 15 years ago Knuth set a standard which still has superior quality than currently available commercial stuff from Adobe. I sometimes wonder what these guys at Adobe are actually doing. Even with ATM4 (which does anti-aliasing) the quality of resolution specific font-rasterizing (like METAFONT does) is unsurpassed. Way to go Don.

Current features

What features are there in this program? The beta provides support for:

- Full DVI 2 support with almost no limitations. No font loading limits. I have previewed files with over 300 fonts in a single DVI file. Current TEX implementations cannot generate that many, so I guess it covers it.
- User interface: Similar to dviscr from Eberhard Mattes. I use this viewer for reference and I am pleased with this product. It's a good product.
- Dynamic reloading. The viewer keeps the file closed as much as possible and if the file is updated (e.g. re-run through TEX) it re-reads the DVI file and repositions to the same page.
- Caching. All tfm files are cached. So when you load another file the viewer will only load the tfm files which are not present. Already loaded tfm files are re-used.
- Customizable paper sizes. You can create your own list of favorite paper sizes. So American users will not be bothered by A4 like stuff, and the European users can zip out the Letter/Legal sizes.
- Drag and drop. Just drop a file on DVIview from Explorer.
- Customizable zoom factors. Set up your favorite zoom factors.

What features will be in the first release? Here's a list:

- virtual fonts. Since the program was designed with this in mind it isn't too difficult. The main problem here is font-encoding. You don't want virtual fonts sec, but you want to re-encode the font as well. Postscript supports font re-encoding and so should the viewer. ATM supports font encoding but it is not documented by Adobe. When I asked them they said it's ATM private and they will not publish the interface. I even asked the developer of DVIWINDO but he said he couldn't tell me that because it would harm Y&Y's commercial product[5]. He advised me to hack ATM so this will probably take a lot of time.
- Color support. I'll start with macros for CONTEXT and LATEX 2ε. color support will be similar to Rockiki's dvips.
- Transformated text. Each font can have its own transformation matrix. So you can slant, extend, rotate or skew text. This is a powerfull feature which is not found in viewers which use PK fonts. Furthermore, you can generate font definitions which are dvips alike. For example:

```
rptmro Times-Roman ".167 SlantFont"
```

  can easily be made available for DVIVIEW.
- Simple graphics commands. Needed to support some special features of the CONTEXT package. Things like drawing rounded rectangles and the like.
- Printing. Seems like a nice feature to have. You can print a hardcopy of your document to a Windows printer driver. You should even be able to print to the PDF writer, so generating pdf is a simple step.

5. Ok, he's right, but I tried anyway

And what is planned to include:

- Facing pages view. Some people want this very badly.
- Colored fonts. Each used font can have it's own color. This is mainly for locating fonts in your DVI file (where did I use cmr10 at 12 pt, I can't see the difference between that one and cmr12).
- PK fonts. It seems TEX users can't live without them. Not all metafonts are available as type1 so PK fonts must be included anyway. Don't expect too much of this. Transformation matrices will definitely not work for PK fonts, or at least be limited.
- Hyper stuff. Clickable links and launching programs from the viewer. This is needed since I use this a lot myself.
- Custom paper sizes. Through specials you can give every page a different size than the default. It also includes paper color. Currently the viewer has already support for this, but I haven't worked out the \special interface yet.
- Graphics. Especially EPS graphics. The program should use GhostScript to render the EPS files. I hope to include decent TIFF support as well. Perhaps some other formats through the use of DLLs. This way one can easily extend the viewer with his or her own graphics format. The dviwin viewer does graphics support through DLLs so it seems logical to use that too.
- Inline METAPOST. No Ghostscript is needed for Metapost graphics, the viewer can draw this direct. METAPOST uses a simple subset of PS commands which is easy to interpret. Directly drawing mftoeps[6] output should also be possible as well as other simple PS output formats. This will save you from installing and maintaining Ghostscript. Besides that, direct drawing is probably faster than using GhostScript.
- Reader profiles. This is similar to Acrobat's article feature. The functionality Acrobat provides is a bit simplistic.
- Scripting language and input fields. You could actually write a program in TEX. I am not sure about the scripting environment. JavaScript seems a logical choice here. The main problem here is time to get a basic understanding of things. I know nothing about JavaScript, and how I could implement an interface to a Java virtual machine (or something like that).
- Clipboard support. You can mark a rectangular area and copy it to the clip-board as a Windows MetaFile. It would probably be nice to create an EPS file as well.
- Simple editing. Since the internal structures I use are suited for fast viewing (and definitely not editing) this will take a lot of time for me.
- Foreign language support. The user interface (menu's dialogs and the online help) can be customized for any language. The log file will always be English so I can understand what's in it.
- Custom printer interface. Interfacing directly to dvips and DVIPSONE. The viewer is not as accurate as these programs. Furthermore I don't want to implement all 200 features these programs provide. DVIPSONE does a far better job generating PS code than the Windows printer driver does.
- Gzip support. Just gzip your DVI-file. The viewer automatically unzips it. This is done through zlib.

When can you have it

I am sorry to say, but it will take some time. I hope to release it somewhere in October. Currently I spend five hours on the program every week. So that's not a lot, but it's all the time I have. I hope I can distribute the program through CTAN. Don't worry too much, I'll let you know when it is good enough to go public.

Will it cost you?

Yes it will, but not what you expect. No, I am not charging money here but you'll probably spend some time configuring the tool. As in good TEX tradition it comes with many options and they probably don't suit your personal needs. For emTEX users it should be fairly easy to set up since some settings are similar. There will be no registration fee, the program is absolutely free (and with no warranty). The first version will also include the full source code.

O yeah, for all you users who are getting tired of reconfiguring Ghostscript's fontmap file, be prepared. The viewer also uses a font map file and yes you must adapt it to your personal needs. You'll probably need to do this when you update the program (as in Ghostscript). I hate this too and I hope to find some solution for this some day. I will try to include all the fonts that are on the 4AllTEX cd, but you probably have to customize things. At least the complete Computer Modern family and the AMS fonts. Also some support for basic postscript fonts.

So that's it. I hope you're a bit enthousiastic about the viewer-to-be-born. Again I hope to have the features implemented as soon as possible but as it is with these freeware tools, there are no guarantees. I'll come back to it in the next MAPS.

Needless to say but there are many people on the background supporting and motivating me. Special thanks to: Hans Hagen, Taco Hoekwater and Erik Frambach.

6. mftoeps is a package from Jackowski to create eps files from metafont code.