Using *Asymptote* like METAPOST

Jim Hefferon

TEX Users Group Annual Conference 2023-July

Abstract

Asymptote is a descriptive vector graphics language for technical drawing that fits very well with TEX, LATEX, and friends. It deserves to be more widely known.

One appealing thing is that it is in part based on algorithms from METAFONT and METAPOST but it extends those to three dimensions. I'll discuss a couple of workflow things that a beginner to this system who is coming from METAPOST might like, in particular using a single source file to output many related graphics.

Features

Asymptote is a powerful descriptive vector graphics language. It provides a natural coordinate-based framework for technical drawing. LaTeX typesets the text and equations.

- Mathematically oriented. Inspired by METAPOST but with a more standard C++-like programming syntax and IEEE floating points. Fully generalizes METAPOST's path construction algorithms to three dimensions.

Features

Asymptote is a powerful descriptive vector graphics language. It provides a natural coordinate-based framework for technical drawing. LaTeX typesets the text and equations.

- Mathematically oriented. Inspired by METAPOST but with a more standard C++-like programming syntax and IEEE floating points. Fully generalizes METAPOST's path construction algorithms to three dimensions.
- Generates high-quality PostScript, OpenGL, PDF, SVG, WebGL, V3D, and PRC vector graphics, as well as 3D vector WebGL graphics for HTML files and 3D vector PRC graphics for PDF files.

Features

Asymptote is a powerful descriptive vector graphics language. It provides a natural coordinate-based framework for technical drawing. LaTeX typesets the text and equations.

- Mathematically oriented. Inspired by METAPOST but with a more standard C++-like programming syntax and IEEE floating points. Fully generalizes METAPOST's path construction algorithms to three dimensions.
- Generates high-quality PostScript, OpenGL, PDF, SVG, WebGL, V3D, and PRC vector graphics, as well as 3D vector WebGL graphics for HTML files and 3D vector PRC graphics for PDF files.
- Uses the simplex method and deferred drawing to solve overall size constraint issues between fixed-sized objects, such as labels and arrowheads, and objects that should scale with figure size.

Features

Asymptote is a powerful descriptive vector graphics language. It provides a natural coordinate-based framework for technical drawing. LaTeX typesets the text and equations.

- Mathematically oriented. Inspired by METAPOST but with a more standard C++-like programming syntax and IEEE floating points. Fully generalizes METAPOST's path construction algorithms to three dimensions.
- Generates high-quality PostScript, OpenGL, PDF, SVG, WebGL, V3D, and PRC vector graphics, as well as 3D vector WebGL graphics for HTML files and 3D vector PRC graphics for PDF files.
- Uses the simplex method and deferred drawing to solve overall size constraint issues between fixed-sized objects, such as labels and arrowheads, and objects that should scale with figure size.
- Under continuing development. Runs on UNIX, MacOS, and Windows.

You can even run it in your browser without installing it, using the Asymptote Web Application.

Asymptote compared with Ti*k*Z

Ti*k*Z is very widely used and *Asymptote* has a much smaller user base.

I have not used Ti*k*Z much, and have not used it lately at all. But I compared the two when I was starting my latest project, some years ago. Personally I found that *Asymptote* had some relevant technical advantages, including native 3D graphics. I also found Ti*k*Z harder to program in. Obviously, YMMV.

Asymptote compared with Ti*k*Z

Ti*k*Z is very widely used and *Asymptote* has a much smaller user base.

I have not used Ti*k*Z much, and have not used it lately at all. But I compared the two when I was starting my latest project, some years ago. Personally I found that *Asymptote* had some relevant technical advantages, including native 3D graphics. I also found Ti*k*Z harder to program in. Obviously, YMMV.

Another difference, relevant here, is that the basic paradigm in Ti*k*Z is that your figures are in your document, generated when the document is generated. In *Asymptote* the paradigm is that they are generated outside the document. (Yes, you can generate stand-alone in TikZ and yes, you can include *Asymptote* code in a document.)

In the past I had a bad experience while using PS Tricks when the LaTeX world switched to pdfLaTeX. It left me with a preference for a dependence chain that is shorter and broader over one that is taller and thinner. So I wanted to generate the figures independently from the documents. Again, YMMV.

Working in METAPOST

I wrote a book years ago using METAPOST. For a person who cannot draw, being able to, for instance, tell the computer to put a label equidistant between two points is a comfort.

The basic structure is that one file holds many figures. This will output a graphic into a file numbered 1 and another graphic into a file numbered 2.

```
beginfig (1)
  -- figure drawing commands in here --
endfig;

beginfig (2)
  -- second figure drawing commands here --
endfig;
```

If these are drawings for Calculus then you might, for instance, at the top of the file declare `VECTOR_THICKNESS=0.8pt` and use that in lots of the figures. So putting all the sources in the same file is convenient.

But having worked with METAPOST, I was aware of some warts. For me the two biggest are lack of any real 3D abilities, and that programming in the language can be … quirky.

So when I saw the new *Asymptote* system I was eager to try it. It has been very good.

I will describe two adjustments that may help a person coming from METAPOST and that took me some time to dope out.

Adjustment One: Multiple figures per file

The basic paradigm in *Asymptote* is to have one figure per input file. Here is the skeleton to put multiple figures in a file.

This outputs the file `test000.pdf` containing the graphic with a diagonal red line.

```
string OUTPUT_FN = "test%03d";

// =============
picture pic;
int picnum = 0;
unitsize(pic, 1cm);

draw(pic, (0,0)--(1,1), red);
shipout(format(OUTPUT_FN,picnum), pic, format="pdf");
```

Points

- The `OUTPUT_FN` gives all the file names the same structure. So if you have lots of graphics (my current book has more than 2000) then it is easier to work with them. Of course, the `"test%03d"` causes the picture number to be formatted as a three decimal place integer. I find that two decimal places in the file name is uncomfortably tight.
- I write `picnum = 0` and `picnum = 1`, etc., rather than `picnum = picnum+1`. When you go back into a file with eighty pictures looking to fix a bug in the fifty third, you want this.
- Because of the multiple outputs, lots of commands need a `pic` argument. That code shows it in the `draw(...)` command. If you leave it out, that line gets drawn somewhere (that is, there is no error) but not in the graphic file where you are looking for it. The `pic` is also in the `unitsize(...)` command.

Adjustment Two: Style files

This is related to the prior adjustment. As I mentioned, one way that having multiple outputs from a single input file is helpful is because uniformity. You can have some parameters such as line thickness or font or font size, and describe them in just the one file.

But the same applies across multiple files. It is good to have a single file `style.asy` that gets input into every *Asymptote* source file.

For instance, here is part of my style file for the most recent book.

```
import fontsize;
defaultpen(fontsize(9.24994pt));
import texcolors;
// colors Tech Office
pen darkgrey_color=rgb("595241");  // hex string 89 82 65
pen lightgrey_color=rgb("E0D4BE");  // tan 224 212 190
pen white_color=rgb("FFFFFF");
pen lightblue_color=rgb("ACCFCC");  // 172, 207, 204
pen red_color=rgb("8A0917");          //  138,09,23

pen highlightcolor=red_color;
pen backgroundcolor=lightblue_color;
pen boldcolor=darkgrey_color;
pen lightcolor=lightgrey_color;
pen verylightcolor=white_color;
```

I use the more abstract variables such as `highlightcolor` to make graphics.

Where to put it? The simple case

You can bring in a file with the `import filename` command. If that file is in the list of directories searched by the *Asymptote* system then you are good.

For instance, *Asymptote* has a standard file called `settings` that you often want to bring in.

```
import settings;
settings.outformat="pdf";
```

The searched list is what you'd think it would be: (1) the current directory; (2) one or more directories given by the environment variable `ASYMPTOTE_DIR`; (3) the directory given by `ASYMPTOTE_HOME` (or else `.asy` in the user's home directory); (4) the *Asymptote* system directory; (5) the *Asymptote* examples directory, such as `/usr/local/share/doc/asymptote/examples`.

A more complex case

I provide the files for my book in a git repository (`https://gitlab.com/jim.hefferon/toc`). I've had bad experience depending on people knowing how to get working, such as setting a environment variable. When these folks are unable, they sometimes write me. That means time and sometimes, aggravation. Is there a way to import from a file that lowers this bar?

A more complex case

I provide the files for my book in a git repository (`https://gitlab.com/jim.hefferon/toc`). I've had bad experience depending on people knowing how to get working, such as setting a environment variable. When these folks are unable, they sometimes write me. That means time and sometimes, aggravation. Is there a way to import from a file that lowers this bar?

I am more able to depend on people naming the download directory. Even if they don't read the instructions they almost always use either `computing/`, or `toc/`, or `toc-master/` (if they download the .zip file rather than clone the repo).

I wrote a routine to look through the path of the current file for those three strings, and then searching based on that.

```
string get_repo_path() {
  string current_dir = cd(""); // return current dir
  int project_part_of_path_dex_computing =
      rfind(current_dir, "/computing/");
  int project_part_of_path_dex_toc =
      rfind(current_dir, "/toc/");
  int project_part_of_path_dex_toc_master =
      rfind(current_dir, "/toc-master/");
  // There are eight cases
  //              computing>=0  toc>=0  toc-master>=0
  //    case 0:       F           F          F
  //    case 1:       F           F          T
  //    case 2:       F           T          F
  //    case 3:       F           T          T
  //    case 4:       T           F          F
  //    case 5:       T           F          T
  //    case 6:       T           T          F
  //    case 7:       T           T          T
```

Ending

Asymptote does a great job drawing technical graphics. Adding some METAPOST-like workflow makes it even better.