Approaching Asymptote

Michael Doob and Jim Hefferon

Asymptote is a relatively new tool to make graphics that is integrated with the TEX family. On its website[1] its developers, Andy Hammerlindl, John Bowman, and Tom Prince, characterize it as "a powerful descriptive vector graphics language inspired by METAPOST that features robust floating-point numerics, automatic picture sizing, native three-dimensional graphics, and a C++/Java-like syntax enhanced with high-order functions." It is free software, released under the GNU General Public License.

Those developers have described Asymptote's advanced capabilities and algorithms in several papers (for instance, [1], [3], and [4]) and presentations (see [2]). The comprehensive manual and additional documentation is on the website. Also check out Philippe Ivaldi's wonderful gallery and tutorial.[2]

We are not developers, we are users (specifically, mathematical users). This is a gentle introduction aimed at people who need to produce mathematically-oriented graphics, and who may find that this system fits their needs and how they think.

We will first briefly compare Asymptote with METAPOST, since that program may be familiar to readers. We will then introduce capabilities that are basic to Asymptote by using some figures, chosen both to be close to what a person might produce in everyday work and to illustrate the power of the system. We will finish by comparing this system with two others that have many of the same capabilities and are widely used in the TEX community, Ti*k*Z and PSTricks.

1 Tangent: Predecessor systems

METAFONT is a programming language written by Donald Knuth to define the fonts for TEX. METAPOST[3] is an extension of METAFONT targeted at PostScript output. It remains widely used for graphics for mathematics and related fields, and it is under active development.

METAPOST produces two-dimensional line art of very high quality. A number of innovative algorithms are built in. Its output is vector-based, not raster — that is, not dot-by-dot — so making a graphic larger or smaller is smooth. One of its key strengths is solving systems of equations. For instance, the language allows you to easily find the intersection of two lines, so if you label an intersection and later adjust one line a bit then the label moves with the intersection. METAPOST is integrated with TEX so that, for instance, you can produce labels that use the LATEX style of the target document.

The most important point about METAPOST (and Asymptote also) is that it is not mouse-driven. Instead, you write a program to produce the output. Below we will discuss some advantages of this — how this allows us to construct figures in a way that fits with our training — but one disadvantage is that because the syntax of the METAPOST language is unlike most other languages, users can find awkward switching from programming in more everyday languages to programming in this one.

2 Meeting Asymptote: Two dimensions

Asymptote has essentially all of the capabilities of METAPOST. Its syntax is Java-like. Asymptote has a more extensive built-in function set than METAPOST. And it comes with many add-on modules (METAPOST is relatively weak in this area). In the next section we will highlight one add-on for drawing in three dimensions but first we focus how Asymptote does at METAPOST's strength, two-dimensional line art. Since Asymptote develops on those capabilities it too makes those drawings with ease.

For a first taste we will walk through making an elementary school star, as shown. The file `star.asy` contains this Asymptote code.

```
size(.5inch);
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;
draw(star);
```

For the first line, Asymptote expects that you typically want to specify the size of the graphic, and we've specified that its width is a half inch. In the second through fifth lines we construct the path with five line segments (as with METAPOST the `--` operator connects points with line segments while `..` makes Bezier curves) and close the path with `cycle`. Finally we draw that path to the output picture (the default line width is 1 PostScript point).

We compile that code to a graphic by running it through Asymptote. You should be able to try this also because there are versions for GNU/Linux, Mac, or Windows. We use the executable that comes with Ubuntu Linux but the commands below should work anywhere. The command line

```
$ asy star
```

produces the output file `star.eps` in Encapsulated PostScript. You can get just about any graphics format, such as PDF.

```
$ asy -fpdf star.asy
```

[1] `http://asymptote.sourceforge.net`
[2] `http://www.piprime.fr/asymptote`
[3] `http://www.tug.org/metapost.html`

LaTeX can use this output with the regular `graphicx` command

```
\includegraphics{star.pdf}
```

or you can instead embed the Asymptote code inside your LaTeX file and the graphic will automatically be inserted; see the documentation for that.

With that first taste of the system we can begin to go further. Asymptote has four basic operations that we will illustrate with four small listings.

The `draw` command is first. This `.asy` file

```
size(35pt);
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;
draw(star,linewidth(2)+lightblue+beveljoin);
```

draws this star.

The `draw` command has many options. For instance, here we are drawing with a pen 2 points wide, in blue, and with line segments joined in a bevel.

Asymptote's second basic operation, `fill`, colors in a closed path.

```
size(35pt);
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;
fill(star,lightblue);
```

Here we `fill` the star with blue.

Asymptote's third basic operation is `clip`. It omits from a shape the part that does not fit in the clip-to area. For instance, we can drop the parts of the star that lie outside a circle.

```
size(35pt);
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;
fill(star,lightblue);
clip(scale(0.618)*unitcircle);
```

(Comparing this star to the prior one shows that after clipping Asymptote has expanded the shape to fit in the declared width.)

The fourth basic operation, `label`, brings in TeX text.

```
size(35pt);
path star;
for(int i=0; i < 5; ++i)
  star=star--dir(90+144i);
star=star--cycle;
draw(star,linewidth(2)+lightblue);
label("\footnotesize $0$",point(star,0),NE);
```

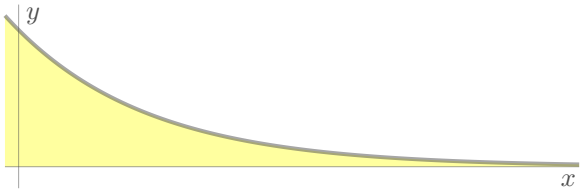
The label incorporates a LaTeX command to produce a subscript-sized label

that is northeast of the path's starting point (the initial point on the initial line segment is point 0 of the path `star`, the initial point on the following line segment is the path's point 1, etc.).

Asymptote embeds these four core capabilities inside of a powerful language with a familiar syntax.

We will illustrate with some examples of jobs that are typical for our work. For each we will first describe the figure and then examine the code listing in more detail.

The next graphic began life as an illustration for a calculus lecture.



For us, the natural way to produce this is not to use a mouse to try to get a good approximation of the true picture. Instead, we want to define the function $f(x) = e^{-x}$ and from that have the computer generate the graph and fill the area below it.

That thinking goes a long way toward writing the code, shown below. The key line is the fourth one where, after importing modules and declaring the horizontal size (*TUGboat* has columns that are about three inches wide), we define the function `f`. With that function, Asymptote makes the path that is the function's graph, fills the area between that graph and the x-axis, and finally draws the curve.

```
include "jh.asy";
import graph;
size(3inch);
real f(real x) {return exp(-x);}
real xmin=-0.1, xmax=4.1;
real ymin=-0.1, ymax=1/exp(-.1);
path g=graph(f,xmin,xmax,operator ..);
path c=buildcycle(g,
                  (xmax,f(xmax))--(xmax,0),
```

```
                (xmax,0)--(xmin,0),
                (xmin,0)--(xmin,ymax));
fill(c,THINPEN+FILLCOLOR+opacity(0.75));
xaxis("\footnotesize $x$",ymin,xmax,AXISPEN,
      above=true);
yaxis(Label("\footnotesize $y$",align=E),AXISPEN,
      above=true);
draw(g,FCNPEN);
```

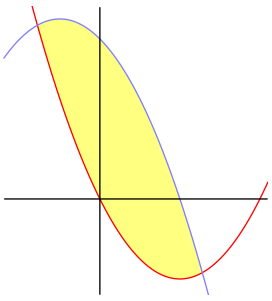
There are two things to note about this listing. The first is that to get the area to be filled we must close in the left and right sides. The `buildcycle` command creates the cyclic curve from the given sequence of bounding curves; the left and right sides are just line segments created with the `--` operator.

The second thing to note is that this code imports the standard Asymptote module `graph` to bring in the commands `graph`, `xaxis`, and `yaxis`. This is one of the add-on modules mentioned above to help with common tasks. This listing also has `include "jh.asy"` to bring in a local `.asy` file. It contains some of the authors' own commands and defines some constants to give a set of graphics a more uniform look. Here is `jh.asy`.

```
import fontsize; defaultpen(fontsize(9.24994pt));
import texcolors;
pen FILLCOLOR=lightyellow;

pen THINPEN=squarecap + linewidth(0.4pt);
pen AXISPEN=THINPEN + gray(0.3)
            +opacity(.5,"Normal");
pen FCNPEN=squarecap +linewidth(1.5pt) + gray(0.3)
            +opacity(.5,"Normal");
```

The prior example is written in a style where we declare what we want and Asymptote figures out how to draw it. The advantage of this over using a mouse-based painting program is like the advantage of separation of appearance from content that we get when using a system built on TEX rather than a word processor. The next example also illustrates this writing style.

To show the area between two curves

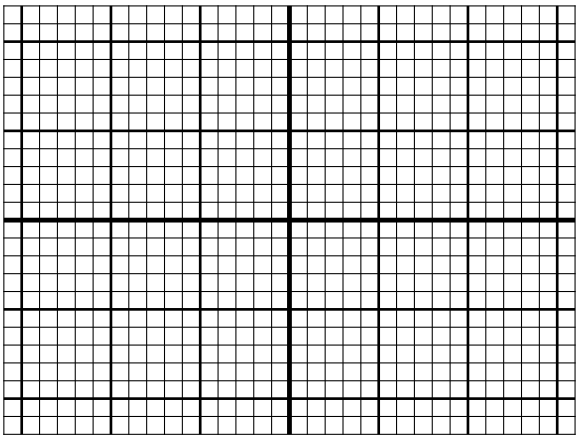


our inclination is to define the two functions and then ask Asymptote to use those definitions to generate the paths, find the intersections, and then construct and fill the desired area.

```
import graph;
size(0,1.5inch);

real p_up(real x) {return x^2-2*x+0;}
real p_down(real x) {return -x^2-x+2;}
real xmin=-1.2, xmax=2.1;
real ymin=-1.2, ymax=2.4;
path g_up=graph(p_up,xmin,xmax,operator ..);
path g_down=graph(p_down,xmin,xmax,operator ..);
pair ipoints []=intersectionpoints(g_up,g_down);
path c=
   graph(p_up,ipoints[0].x,ipoints[1].x,
         operator ..)
   --
   graph(p_down,ipoints[1].x,ipoints[0].x,
         operator ..)
   -- cycle;
fill(c,lightyellow);
draw(g_up,red);
draw(g_down,lightblue);
xaxis(above=true); yaxis(above=true);
path clippath = (xmin,ymin)--(xmax,ymin)
  --(xmax,ymax)--(xmin,ymax)
  --cycle;
clip(clippath);
```

In detail the code is much like the prior example. After importing the module, we set the graphic size (we set the height to be 1.5 inches; setting the width to 0 has the system find the natural width). We then define the functions `p_up` and `p_down`. We ask Asymptote to make paths that are the two graphs with given left and right endpoints, find where those two intersect, and construct the closed curve `c` between the two (`ipoints.x` gives the first component of `ipoints`). Asymptote fills the area inside, draws the two graph paths, and finally clips the ends of the parabolas that extend too far away from the part of the picture that we want to show.

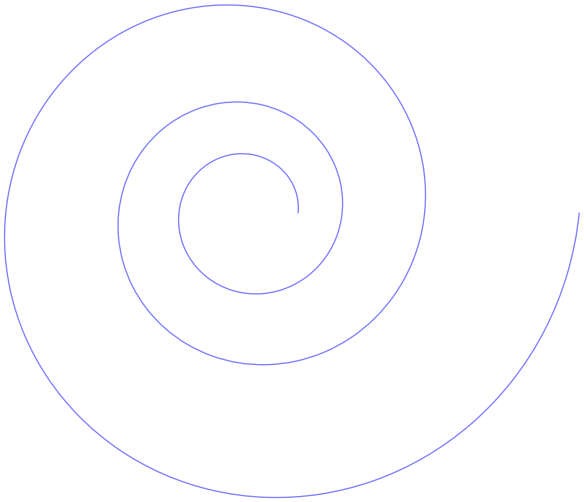
Asymptote includes a full suite of powerful programming constructs. The next illustration is rectilinear graph paper with three different line thicknesses.



The code is a simple illustration of looping. We won't expand on it since readers who have programmed in mainstream languages such as Java or Python will recognize it right away.

```
size(3inch);
pen thinpen=(linewidth(.4)+extendcap+miterjoin);
pen mediumpen=(linewidth(1)+squarecap);
pen thickpen=(linewidth(1.8)+squarecap);
int xmin=-16, xmax=16;
int ymin=-12, ymax=12;
// draw horizontals
for (int k=xmin; k<=xmax; ++k) {
    if (k==0) draw((k,ymin)--(k,ymax),thickpen);
    else if (k%5 ==0)
      draw((k,ymin)--(k,ymax),mediumpen);
    else draw((k,ymin)--(k,ymax),thinpen);
    }
// draw verticals
for (int k=ymin; k<=ymax; ++k) {
    if (k==0) draw((xmin,k)--(xmax,k),thickpen);
    else if (k%5 ==0)
      draw((xmin,k)--(xmax,k),mediumpen);
    else draw((xmin,k)--(xmax,k),thinpen);
    }
```

The final two-dimensional graphic has a polar flavor; it's a logarithmic spiral.



The source shows how easily we can define and combine functions, and again illustrates how the language fits with how a person with mathematical training thinks.

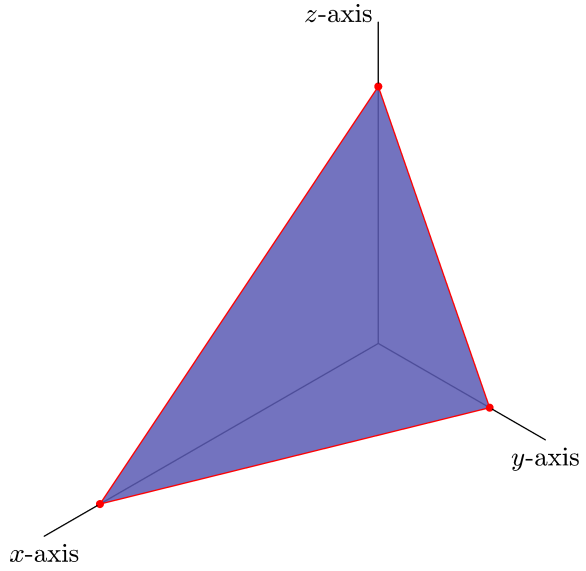
```
import graph;
size(3inch);
real pi=2*acos(0);
real a=0.5, b=0.1; // parameters for the shape
// polar to cartesian
real x(real t) { return a*exp(b*t)*cos(t);}
real y(real t) { return a*exp(b*t)*sin(t);}

pair f(real t) { return (x(t),y(t));}
draw(graph(x, y, 0, 6*pi, operator ..),lightblue);
```

In particular, note that the function `f` returns a pair of reals, where the coordinate functions were defined earlier in the listing.

3 In the limit: Three dimensions

In addition to its capabilities in two dimensions, Asymptote comes with modules targeted at three-dimensional graphics. These develop some of METAPOST's ideas from two dimensions. For instance, a core capability of METAPOST (and originally in METAFONT) is to produce useful and good-looking curves without requiring that the author completely specify those curves. Asymptote extends this to 3D.

Our first graphic is a straightforward image, another one that might appear in a calculus lecture.



Because it is a three-dimensional graph, we will import a different module, but the thinking behind the code is similar to what we've done earlier. In addition to drawing the axes, we declare where the plane intersects each axis, make a surface connecting those three points, and have Asymptote draw and fill the surface.

```
size(3inch);
import settings;
settings.render=10;
settings.maxtile=(50,50);
import graph3;
currentprojection=orthographic(2,2,2);
currentlight=(9,3,4);
// where plane intercepts x, y, and z axes
triple intercepts=(5,2,4);
path3 P = (intercepts.x,0,0)
         --(0,intercepts.y,0)
         --(0,0,intercepts.z)
         --cycle;
draw(surface(P), lightblue+opacity(.85));
draw(P,red);
```

```
dot(P,red);
axes3("\footnotesize $x$-axis",
      "\footnotesize $y$-axis",
      "\footnotesize $z$-axis",
      (0,0,0),(6,3,5));
```

The listing imports the 3D module `graph3`. This gives us a command to draw axes in three dimensions and also lets us define the projection to be orthographic (projection lines are orthogonal to the plane to which the image is projected, that is, the projection is from infinity) and to define the location of the light source.

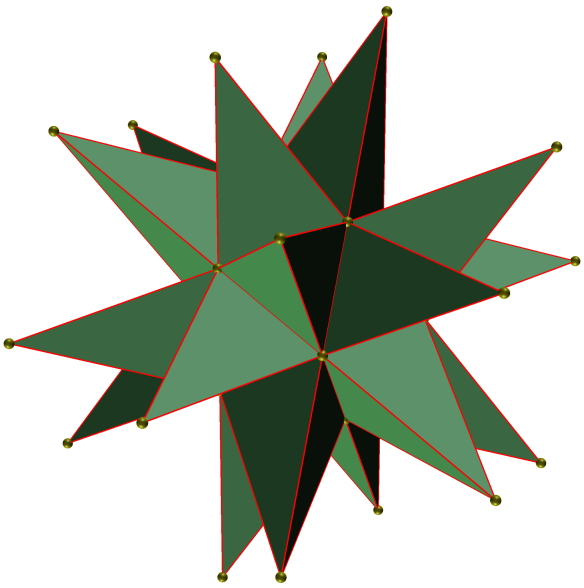
One more point about this graphic: we produced it as a `.png` file using this command line.

```
$ asy -fpng intercepts_plane
```

We chose this format to show well on the printed page; we used `.pdf` for the earlier figures but it doesn't show in our viewer because Asymptote's default behavior is to produce a figure that can be manipulated with the mouse — turned, or zoomed to — but that behavior relies on a viewer's capabilities, and some viewers lack that capability (*TUGboat*'s printed pages also don't have it!). Because of this change to the `.png` raster format we also changed Asymptote's defaults to adjust `render` for 10 pixels per big point to reduce jaggies and changed `maxtile` to get around a common bug in the graphics driver.

Our final two figures are less prosaic. We want to close with the message that Asymptote can indeed do some fancy things, often with very little code.

First, here is a stellated icosahedron.

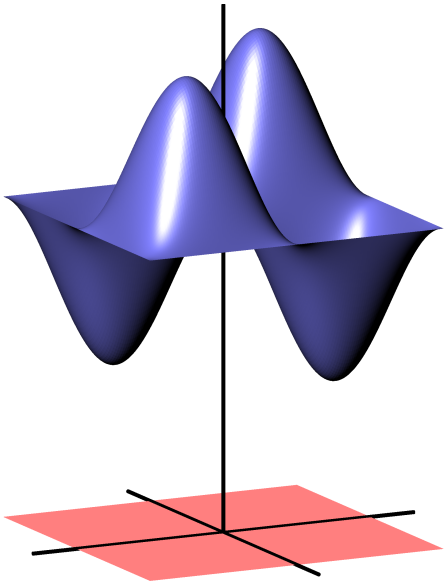


The construction is logical and easy to implement using Asymptote. First define the twelve points of the icosahedron, then use these points to define the twenty faces and, finally, create a function that will erect a pyramid on (i.e., stellate) each of these faces. Asymptote itself takes care of the projections and shading.

```
size(3inch);
import settings;
settings.render=10;
import three;
currentprojection=perspective(21,25,15);
currentlight=White;

real phi = (1+sqrt(5))/2;
// Vertices of the icosahedron are of the form
// (0, \pm 1, \pm\phi), (\pm\phi, 0, \pm 1),
// (\pm 1, \pm\phi, 0)
triple [] Pts = {
(  0,    1,  phi),
(  0,   -1,  phi),
( phi,   0,    1),
(  1,  phi,    0),
( -1,  phi,    0),
(-phi,   0,    1),
( phi,   0,   -1),
(  0,    1, -phi),
(-phi,   0,   -1),
( -1, -phi,    0),
(  1, -phi,    0),
(  0,   -1, -phi)
};

// Faces listed as triples (i,j,k) corresponding
// to the face through Pts[i], Pts[j] and Pts[k].
triple [] faces = {
// upper cap
(0,1,2), (0,2,3), (0,3,4), (0,4,5), (0,5,1),
// upper band
(11,6,7), (11,7,8), (11,8,9), (11,9,10),
(11,10,6),
// lower band
(10,1,2), (6,2,3), (7,3,4), (8,4,5), (9,5,1),
// lower cap
(3,6,7), (4,7,8), (5,8,9), (1,9,10), (2,10,6)
};
// draw the twelve vertices of the icosahedron
for (triple T: Pts)
  draw(shift(T)*scale3(.08)*unitsphere,
       yellow);
real t=3.0; // Scaling for stellation height
// Function to compute the stellation point
triple stell_point(triple u, triple v, triple w)
          {return t/3*(u+v+w);}

void stellate(triple Face) {
   int i=round(Face.x),
     j=round(Face.y),
     k=round(Face.z);
   triple S=stell_point(Pts[i], Pts[j], Pts[k]);
   draw(shift(S)*scale3(.08)*unitsphere,
        yellow);
   draw(S--Pts[i],red);
   draw(S--Pts[j],red);
   draw(S--Pts[k],red);
```

```
    draw(Pts[i]--Pts[j]--Pts[k]--cycle,red);
    draw(surface(S--Pts[i]--Pts[j]--cycle),
         lightgreen);
    draw(surface(S--Pts[i]--Pts[k]--cycle),
         lightgreen);
    draw(surface(S--Pts[j]--Pts[k]--cycle),
         lightgreen);
    }
for (triple Face: faces ) stellate (Face);
```

Finally, while the prior image is polyhedral, that is, the sides are flat, our closing example is a real surface in that it is curved.



As with the earlier listing, the code here merely defines a function `f` and a region over which the graph of that function will lie, and then asks Asymptote to produce the graph. (The `nx` value gives the mesh).

```
size(3inch);
import settings; settings.render=10;
import graph3;
currentprojection=orthographic(2,4,1);
currentlight=(5,4,4);
real pi=2*acos(0);
path3 P=(-1,-1,0)--(-1,1,0)--(1,1,0)--(1,-1,0)
       --cycle;
draw(surface(P),lightred,nolight);

real f(pair z)
   {return 2+sin(z.x*pi)*sin(z.y*pi);}
draw(surface(f,(-1,-1),(1,1),nx=128), lightblue);

pen axispen=(linewidth(1.5)+squarecap);
axes3((-1.3,-1.3,0),(1.3,1.3,3.3),axispen);
```

4 Convergence of technologies: Comparison with Ti*k*Z and PSTricks

The TEX community now has the luxury of choice among three very capable graphics systems, Asymptote, Ti*k*Z,[4] and PSTricks.[5] (We have left METAPOST off this list because at the moment three-dimensional graphics are an issue.) These three are similar. They are close in ability and very powerful, and all are under active development. All have extensive add-on sets that greatly increase their usefulness.

Any of the three can be a great choice for your projects. To some extent, which you choose will be dictated by which one has modules that fit your exact needs. It is also partly a matter of taste.

One point in favor of using Asymptote is that it is a stand-alone program. This may reduce the extent to which your document depends on the current software ecosystem because under natural development you make a stand-alone graphic. Put another way, Asymptote fits a bit better with the Unix philosophy of having a number of tools, each of which does one thing only, but does it well.

For us, a particularly appealing feature is that Asymptote lifts many METAPOST constructs from 2D to 3D. It also has additional advanced functions.

Finally, programming in Asymptote is in a style close to Java and C++, which you may find familiar. For us, graphics is something that we do only occasionally and so we must switch to this language from others. Having familiar constructs helps that switching.

References

[1] J. C. Bowman. Asymptote: Interactive TEX-aware 3D vector graphics. *TUGboat*, 31(2):203–205, 2010. `http://tug.org/TUGboat/tb31-2/tb98bowman.pdf`.

[2] John Bowman. Interactive TEX-aware 3D vector graphics. *TEX Users Group Annual Meeting*, 2010. `http://river-valley.tv/interactive-tex-aware-3d-vector-graphics`.

[3] John C. Bowman and Andy Hammerlindl. Asymptote: A vector graphics language. *TUGboat*, 29(2):288–294, 2008. `http://tug.org/TUGboat/tb29-2/tb92bowman.pdf`.

[4] John C. Bowman and Orest Shardt. Asymptote: Lifting TEX to three dimensions. *TUGboat*, 30(1):58–63, 2009. `http://tug.org/TUGboat/tb30-1/tb94bowman.pdf`.

◇ Michael Doob
University of Manitoba

◇ Jim Hefferon
Saint Michael's College
`jhefferon (at) smcvt dot edu`

[4] `http://sourceforge.net/projects/pgf`
[5] `http://tug.org/PSTricks`