Playing with Axodraw

Abstract
This paper shows some of the features of Axodraw. It puts emphasis on applications, not only in the field of physics, but also in completely unrelated fields like mathematical tiling constructions, fashion patterns or the design of sudokus.

Introduction

Question: what do the figures 1–5 have in common?
The answer to the above question is: all were made with the help of Axodraw.

A first version of Axodraw was created in 1992. At the time LaTeX did not have any sophisticated graphical capabilities beyond lines and the creation of fonts that might contain half circles and line segments. Attempts at drawing Feynman diagrams resulted either in very ugly graphics or the inclusion of an external file for each individual diagram/picture which had to be made by a separate drawing package. The advent of the NeXT computer with a screen that ran display postscript, combined with the possibilities to include postscript commands in TeX and LaTeX

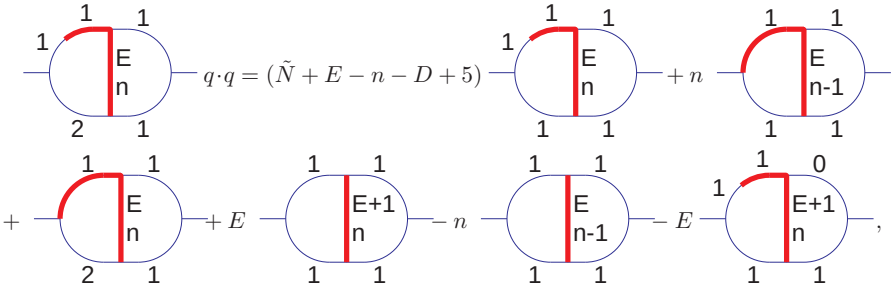


Figure 1.

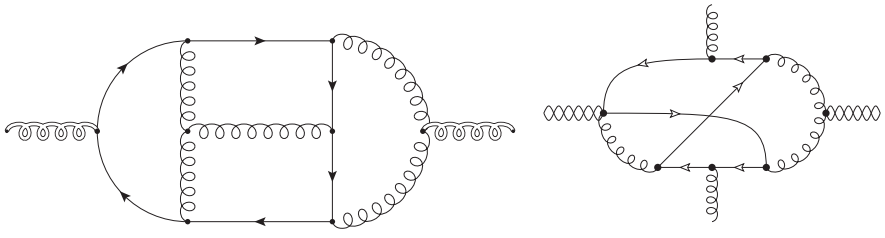


Figure 2.

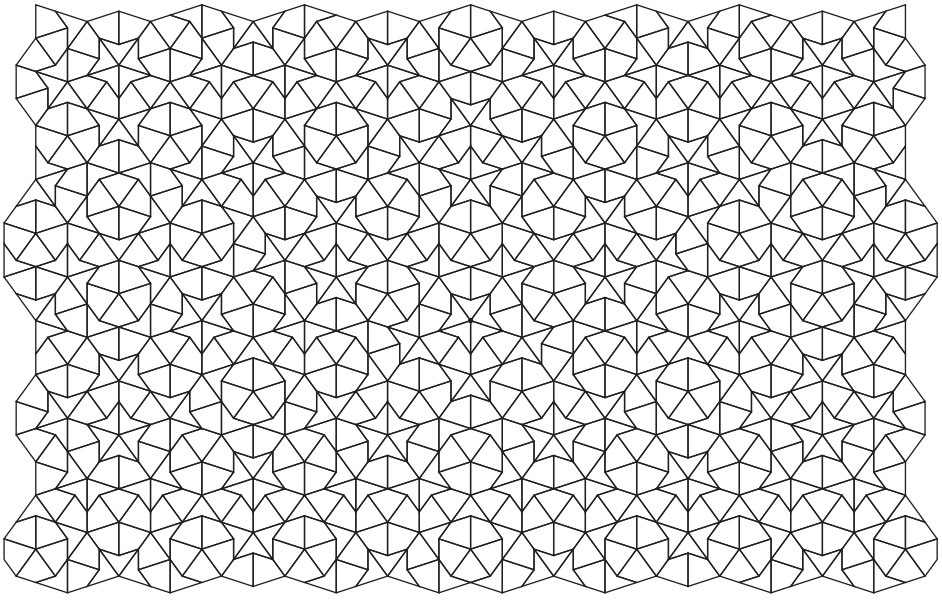


Figure 3.

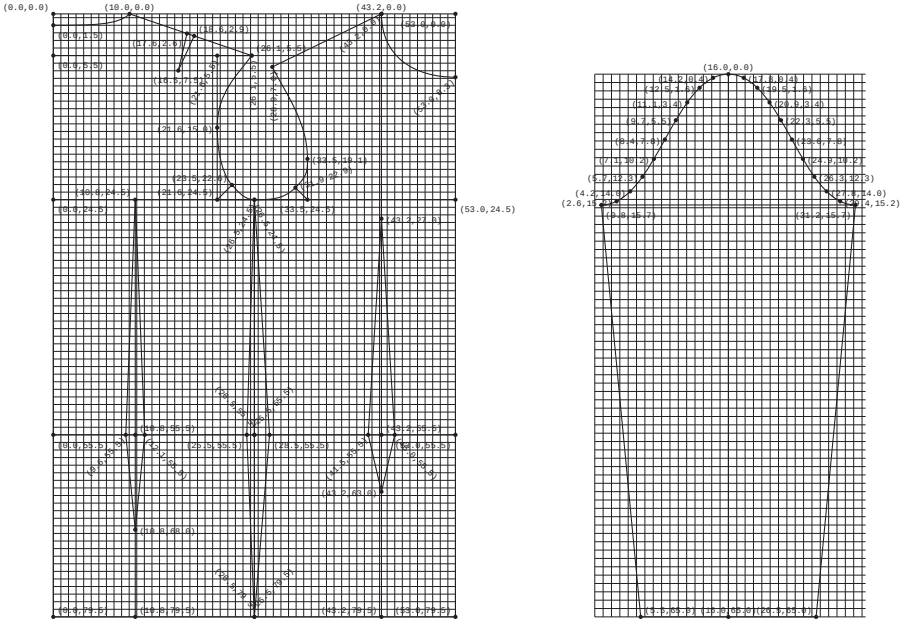


Figure 4.

files made it possible to design a style file that contained many graphical primitives inside the LaTEX picture environment. This gave much better graphics and also allowed all diagrams to be part of the LaTEX source file. At first this was for private use, but on demand it was made publicly available and published in CPC [1].

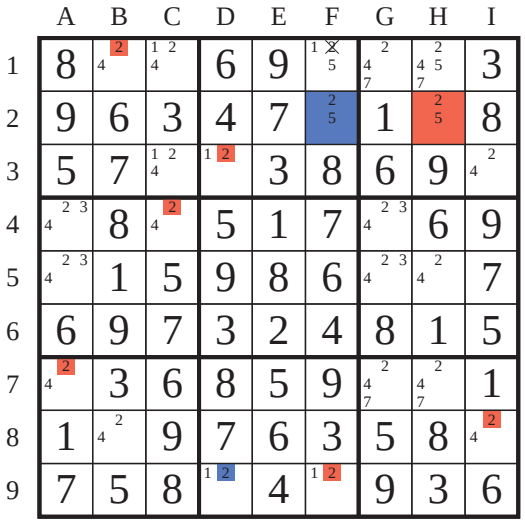


Figure 5.

Version 1

Version 1 worked with the use of postscript specials. Such specials pass the code inside them on to the dvips program that converts the dvi code to postscript. Hence the flow of translation is

```
latex file
dvips file -o
```

and the result would be file.ps with the postscript code inside the specials now inside the .ps file. Here is an example of such a special

```
\special{! /bbox{
%
%   Draws a blanked out box x1,y1,x2,y2
%
    gsw p2 p1
    gsave
    1 setgray abox fill
    grestore
    abox stroke
    grestore
} def }
```

and the LaTeX interface for this function:

```
\def\BBox(#1,#2)(#3,#4){
%
%   Draws a box with the left bottom at (x1,y1) and the right top
%   at (x2,y2). The box is blanked out.
%
\put(\axoxoff,\axoyoff){\special{"\axocolor #1 \axoxo add #2 \axoyo
 add #3 \axoxo add #4 \axoyo add \axowidth \axoscale bbox showpage}}
}
```

The variables here are offsets, color, linewidth and a scale factor. All postscript functions are in a preamble which, together with the LaTeX interfaces, are inside the file Axodraw.sty.

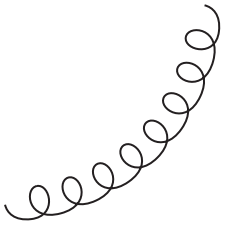
There are of course much more complicated functions, like a gluon on a circle segment:

```
\special{! /gluearc{
%
%  Draws a gluon on an arcsegment
%  x_center,y_center,radius,stat_angle,end_angle,gluon_radius,num
%  in which num is the number of windings of the gluon.
%  Method:
%  1:  compute length of arc.
%  2:  generate gluon in x and y as if the arc is a straight line
%  3:  x' = (radius+y)*cos(x*const)
%      y' = (radius+y)*sin(x*const)
%
   gsw /num ed /ampi ed /arcend ed /arcstart ed /radius ed
%
%  When arcend comes before arcstart we have a problem. The solution
%  is to flip the order and change the sign on ampi
%
   arcend arcstart lt {
       /ampi ampi -1 mul def
       /arcstart arcend /arcend arcstart def def
   } if
%
   translate                              % move to center of circle
   arcstart rotate                          % segment starts at zero
   /darc arcend arcstart sub def             % argsegment
%
  /dr darc 180 div 3.141592 mul radius mul def  % length of segment.
%
   /const darc dr div def                    % conversion constant
%
   /num num 0.5 sub round def
  /inc dr num 2 mul 2 add div def       % increment per half winding
%
   /amp8 ampi 0.9 mul def
   /amp1 radius ampi add def
   /amp2 radius ampi sub def
   /amp3 radius ampi 2 div add def
   /amp4 amp1 inc amp8 add const mul cos div def
   /amp5 amp2 amp8 const mul cos div def
   /amp6 amp1 inc 0.6 mul amp8 add const mul cos div def
   /amp7 amp1 inc 0.9 mul const mul cos div def
   amp8 0 lt {/amp8 amp8 neg def} if
%
   /x1 inc 2 mul def
%
   newpath
       radius 0 moveto
%
       inc 0.1 mul const mul dup cos amp3 mul exch sin amp3 mul
       inc 0.5 mul const mul dup cos amp7 mul exch sin amp7 mul
       inc 1.4 mul const mul dup cos amp1 mul exch sin amp1 mul
           curveto
       x1 amp8 add const mul dup cos amp6 mul exch sin amp6 mul
```

```
       x1 amp8 add const mul dup cos amp5 mul exch sin amp5 mul
       x1 const mul dup cos amp2 mul exch sin amp2 mul
           curveto
%
       2 1 num {
           pop
          x1 amp8 sub const mul dup cos amp5 mul exch sin amp5 mul
          x1 amp8 sub const mul dup cos amp4 mul exch sin amp4 mul
           x1 inc add const mul dup cos amp1 mul exch sin amp1 mul
               curveto
           /x1 x1 inc dup add add def
          x1 amp8 add const mul dup cos amp4 mul exch sin amp4 mul
          x1 amp8 add const mul dup cos amp5 mul exch sin amp5 mul
           x1 const mul dup cos amp2 mul exch sin amp2 mul
               curveto
       } for
%
       x1 amp8 sub const mul dup cos amp5 mul exch sin amp5 mul
       x1 amp8 sub const mul dup cos amp6 mul exch sin amp6 mul
      x1 inc 0.6 mul add const mul dup cos amp1 mul exch sin amp1 mul
           curveto
      x1 inc 1.5 mul add const mul dup cos amp7 mul exch sin amp7 mul
      dr inc 0.1 mul sub const mul dup cos amp3 mul exch sin amp3 mul
       dr const mul dup cos radius mul exch sin radius mul
       curveto
   stroke
%
   grestore
} def }
```

Its use is with:

```
\begin{center}
\SetScale{1.5}
\begin{axopicture}(70,70)
\GluonArc(10,60)(50,270,360){4}{8}
\end{axopicture}
\end{center}
```



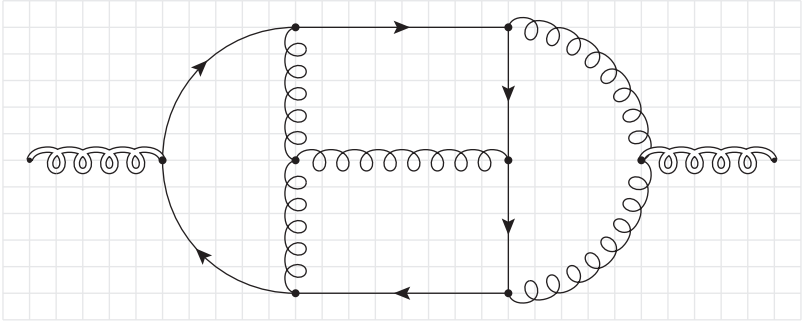
Version 2

A big weakness of Axodraw was that one needed to figure out the coordinates. D.Binosi en L.Theussl solved that problem by creating an interactive Java program named JAxodraw [2] in which one can select graphical elements and position them with the mouse. When the figure is complete one can let it generate the proper Axodraw code. It is still used very much.

When John Collins was writing a book on field theory, he needed more graphical primitives. At first this resulted in a second version of JAxodraw [3], but in the end it was not sufficient.

Eventually John and I have made a second version of Axodraw in which many of the user remarks were addressed. For instance the coordinate problem was greatly improved by a grid function which is used during the design of a picture and commented out when the picture is complete.

```
\begin{center}
\begin{axopicture}(300,120)
\AxoGrid(0,0)(10,10)(30,12){LightGray}{0.5}
\Line[arrow](110,110)(190,110)
\Line[arrow](190,10)(110,10)
\Arc[arrow,clockwise](110,60)(50,270,180)
\Arc[arrow,clockwise](110,60)(50,180,90)
\GluonArc(190,60)(50,270,360){4}{8}
\GluonArc(190,60)(50,0,90){4}{8}
\Gluon(110,110)(110,60){-4}{5}
\Gluon(110,60)(110,10){-4}{5}
\Gluon[double](10,60)(60,60){4}{4}
\Gluon[double](240,60)(290,60){4}{4}
\Line[arrow](190,110)(190,60)
\Line[arrow](190,60)(190,10)
\Gluon(110,60)(190,60){4}{8}
\Vertex(110,110){1.5}
\Vertex(110,10){1.5}
\Vertex(60,60){1.5}
\Vertex(240,60){1.5}
\Vertex(190,110){1.5}
\Vertex(190,10){1.5}
\Vertex(110,60){1.5}
\Vertex(190,60){1.5}
\Vertex(10,60){1}
\Vertex(290,60){1}
\end{axopicture}
\end{center}
```



One of the wishes from my side was the capability to use pdflatex because nowadays the pdf file format is used much more than the postscript format. This is far from trivial, because pdf is not a language in which one can compute things and in addition its manual is about 1000 pages. The eventual solution is a separate program, axohelp, that generates the pdf instructions. It is used in a way that is similar to the use of makeindex:

```
pdflatex file
axohelp file
pdflatex file
```

The first run of pdflatex generates a file file.ax1 which contains a list of all Axodraw instructions that need to be translated. The running of axohelp creates a file file.ax2 that contains the original file.ax1 statements and their pdf translation. When pdflatex is run again, it sees the .ax2 file and as long as all Axodraw statements agree with the statements in the .ax2 file pdflatex can use the translation. If there is disagreement, because the input pictures have been changed, there will be the advise to run axohelp again. Hence, when no pictures have been changed and there is already a .ax2 file one can suffice with running pdflatex just once.

Hence, if in a file.ax1 the graphical object 287 would look like

```
[287] Polygon "(-163.4721,-118.7694)(-139.6218,-126.5188)
           (-148.7318,-113.9800)(-148.7318,-98.4812)" 1.000000;
```

in file.ax2 it would become:

```
\axo@setObject{287}%
{Polygon "(-163.4721,-118.7694)(-139.6218,-126.5188)
         (-148.7318,-113.9800)(-148.7318,-98.4812)" 1.000000;}%
{ 1 w -163.472 -118.769 m -139.622 -126.519 l -148.732 -113.98 l
  -148.732 -98.481 l h S}
```

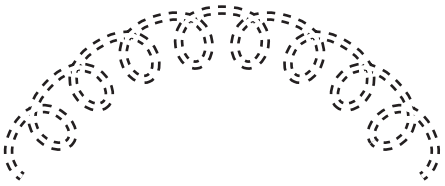
Among the new features are Bezier curves and customizable arrows that can be put at a percentage of the length of a line. This gives interesting problems like how to put a properly alligned arrow on a Bezier curve at a given fraction of its length.


```
\Bezier[arrow,arrowpos=0.20](10,10)(20,60)(80,10)(90,40)
```

To do this well you need to:

1. Compute the length of a Bezier curve.
2. Compute the length of part of a Bezier curve.
3. Iterate until you are at the proper percentage.
4. Compute the derivative at the given point.

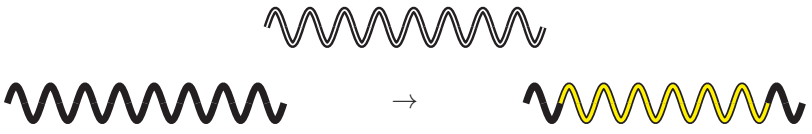
Another little problem: How to draw a double line or spiral?



```
\begin{center}
\SetScale{2.0}
\begin{axopicture})100,50)(0,0)
\DashDoubleGluonArc(45,0)(40,20,160){5}{8}{1.3}{1.5}
\end{axopicture})100,50)(0,0)
\end{center}
```

Some of these features take quite some calculations. This is easy in the C sources of axohelp, but it is also possible in postscript. In LaTeX and in pdf files this is not possible.

In general you don't draw a double line by drawing two lines, shifted from each other as in JAxodraw, but by first drawing a fat line in the foreground color and then a thinner line in the background color on top:



```
\begin{center}
\SetScale{1.3}
\begin{axopicture}(250,15)(0,0)
\Photon[width=2](10,8)(90,8){5}{8}
\Text(125,8)[]{$\rightarrow$}
\Photon[width=2](160,8)(240,8){5}{8}
\SetColor{Yellow}
\Photon[width=1](170,8)(230,8){5}{6}
\end{axopicture}
\end{center}
```

Another problem in Axodraw is how to draw a sine wave, a spiral or, in pdf, a circle? It can be approximated by a very large number of very small lines, but this is incredibly slow, specially back in 1992. The solution is to use approximations in terms of Bezier curves which are present both in postscript and pdf. A sine wave is drawn with the use of 180 degree segments (from 90 to 270 degrees and from 270 to 90 degrees) with special segments for the endpoints. If this is done well, you need to magnify it very much to see the difference with a proper sine wave. This can also be done with a spiral. The gluon has special endpoints to keep everything properly centered. This is a feature that shows immediately whether a picture has been made with Axodraw.

Postscript has circles, but pdf does not. Hence also circles and circle segments have to be approximated with Bezier curves.

The postscript output and the pdflatex output are, to my knowledge, identical, except for one detail:

In postscript there are real postscript fonts and those do not exist as such in the pdf version. There do exist LaTeX versions of these fonts, and those are used in the pdflatex version, but there is a problem with that the zero height is interpreted differently in postscript than in LaTeX and pdf. Because of this there can be a small vertical displacement between the two versions. A good example is the character q. In Postscript the zero is at the bottom.

About axohelp: the program is set up in in a way that makes it relatively easy to prepare files for other formats if those would become popular in the future. The language in which it is written is C, because this is available on all computers, and also because I am rather familiar with it. If the project would have to be redone, the language Rust might be a consideration, due to its draconic error checking.

An example of some code in axohelp:

```
static double BzK;

void BezierCircle(double r,char *action)
{
    outpos += sprintf(outpos,
        " %12.3f 0 m %12.3f %12.3f %12.3f %12.3f 0 %12.3f c\n",
        -r,-r,r*BzK,-r*BzK,r,r);
    outpos += sprintf(outpos,
        " %12.3f %12.3f %12.3f %12.3f %12.3f 0 c\n",
        r*BzK,r,r,r*BzK,r);
    outpos += sprintf(outpos,
        " %12.3f %12.3f %12.3f %12.3f 0 %12.3f c\n",
        r,-r*BzK,r*BzK,-r,-r);
    outpos += sprintf(outpos,
        " %12.3f %12.3f %12.3f %12.3f %12.3f 0 c %s\n",
        -r*BzK,-r,-r,-r*BzK,-r,action);
}
```

with $\mathrm{BzK} = \frac{4}{3}(\sqrt{2}-1)$, or

```
void Polygon(double *args,int num,int type)
{
    int i;
    MoveTo(args[0],args[1]);
    args += 2;
    for ( i = 1; i < num; i++, args += 2 ) {
        LineTo(args[0],args[1]);
    }
    if ( type == 0 ) { CloseAndStroke; }
    else if ( type ==  1 ) { CloseAndFill; }
}
```

In terms of postscript the corresponding routine is

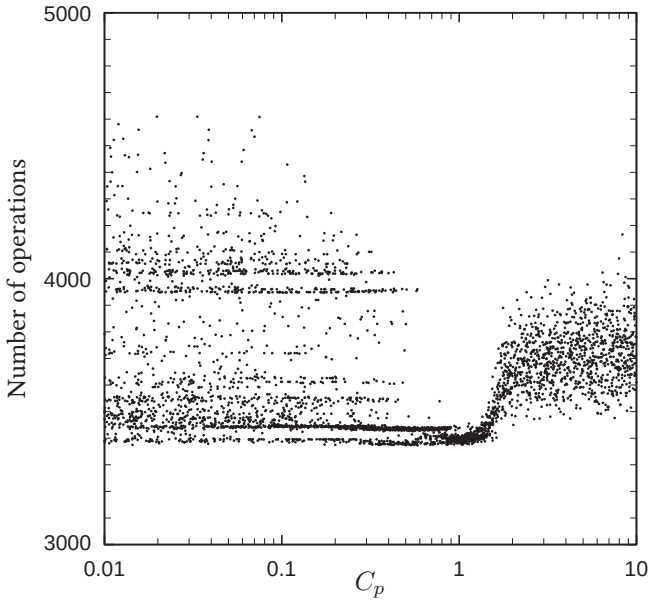
```
%   Incoming stack:
%   [array of x,y pairs] width scale
\special{! /polygon{
    gsw /points ed
    /ss points length 2 idiv 2 mul def
    ss 4 gt {
        newpath
        points 0 get points 1 get moveto
        0 2 ss 4 sub { /ii ed
```

```
            /x1 points ii 2 add get def
            /y1 points ii 3 add get def
            x1 y1 lineto
        } for
        closepath
        stroke
    } if
    grestore
} def }
```

It would have been not very complicated to have axohelp generate the postscript code as well, but much of the code existed already from version 1 and hence we left it that way. Hence the LaTeX binding:

```
%
% Draws a curve through the points in argument 1.
% The points are given as coordinates (x1,y1)(x2,y2)(x3,y3).....
% The curve is continous and continuous in its first and second
% derivatives. The method is linear interpolation of
%         quadratic curves.
%         Color name is argument 2.
%
\def\Polygon#1#2{%
  {%
    \SetColor{#2}%
    \ifcase\axo@pdfoutput
      \put(\axoxoff,\axoyoff){\AXOspecial{%
       [ \axoparray#1] \axowidth\space \axoscale\space polygon }}%
    \else
      \getaxohelp{Polygon}{"#1" \axowidth}%
      \put(\axoxoff,\axoyoff){\axo@pdfliteral{\contentspdf}}%
    \fi
  }%
  \ignorespaces
}
```

Axohelp can do something that TeX/LaTeX cannot do: read the complete .ax1 file and prepare the complete .ax2 file inside the memory and hence optimize it. TeX has only a limited capacity for such things. One can increase the upper-limit but even then there is a hard coded upper-upper-limit. In a first version of Axodraw2 it was programmed that the complete .ax2 file would be read at startup. This ran into trouble with the follwing type of plots:



There are 4000 points in this plot.

At the time we were preparing an article [4] that had about a dozen of these plots. Each point is drawn with a statement of the type

```
\Vertex( 68.08, 44.40){0.5}% 330
```

The system kept crashing and we were forced to translate the plots one by one and feed them in as .eps files. This is entirely against the Axodraw philosophy. In the end we decided to read the .ax2 file one line at a time. The loss in speed turned out to be rather small and it did solve the problem. It took however much more thinking.

About the examples

Back to the examples at the start.

The first example with the graphical formula is from a physics paper [5]. It is actually rather simple. If we take the code for the formula

```
\begin{eqnarray}
  \TAF(1,1,1,2,E,n,1)\ q\mydot q & = &
                (\tilde{N}+E-n-D+5)\TAF(1,1,1,1,E,n,1)
                +n\TAB(1,1,1,1,E,{n-1})
                \nonumber \\ &&
                +\TAB(1,1,1,2,E,n)
                +E\TAD(1,1,1,1,{E+1},n)
                -n\TAD(1,1,1,1,E,{n-1})
                -E\TAF(1,0,1,1,{E+1},n,1) \, ,
                \nonumber
\end{eqnarray}
```

there are a few macro's with parameters. One of these macro's is:

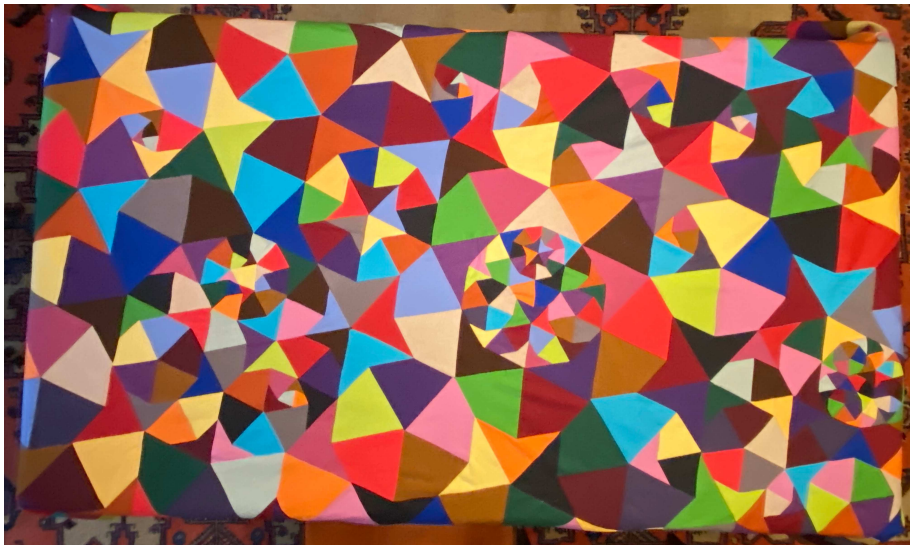
```
\def\TAB(#1,#2,#3,#4,#5,#6){
    \raisebox{-28.1pt}{
    \SetPFont{Helvetica}{14}
```

```
        \hspace{-22.5pt}
        \begin{axopicture}(90,59)(-15,-6)
        \SetScale{0.75}
        \SetColor{Blue}
        \CArc(40,35)(25,90,270) \CArc(60,35)(25,270,90)
       \Line(40,60)(60,60) \Line(40,10)(60,10) \Line(50,10)(50,60)
        \Line(0,35)(15,35) \Line(85,35)(100,35)
        \SetColor{Black}
        \PText(53,39)(0)[lb]{#5} \PText(53,36)(0)[lt]{#6}
        \PText(35,62)(0)[rb]{#1} \PText(65,62)(0)[lb]{#2}
        \PText(65,12)(0)[lt]{#3} \PText(35,12)(0)[rt]{#4}
        \SetColor{Red}
        \SetWidth{3}
                \Line(50,10)(50,60)
                \Vertex(50,60){1.3}
                \Line(40,60)(50,60)
                \CArc(40,35)(25,90,180)
        \SetWidth{0.5}
        \end{axopicture}
        \SetScale{1.0}
        \hspace{-7pt}
        }
    }
```

The Penrose file was generated by computer (of course). If not it can become quite difficult to make very big patterns. The computer generates the LaTeX file. Here is a little piece of it:

```
\begin{axopicture}{(560,400)(0,0)}
\SetWidth{1.000000}
\SetPFont{Courier}{182.03}
\SetOffset(360,160)
\Polygon{(-163.4721,-118.7694)(-139.6218,-126.5188)% \
         (-148.7318,-113.9800)(-148.7318,-98.4812)}{Black} % Dart
\Polygon{(-163.4721,-118.7694)(-187.3223,-126.5188)% \
         (-172.5821,-131.3082)(-163.4721,-143.8471)}{Black} % Dart
\Polygon{(-163.4721,-118.7694)(-163.4721,-143.8471)% \
         (-154.3621,-131.3082)(-139.6218,-126.5188)}{Black} % Dart
\Polygon{(-163.4721,-118.7694)(-148.7318,-98.4812)% \
         (-163.4721,-103.2706)(-178.2123,-98.4812)}{Black} % Dart
\Polygon{(-163.4721,-118.7694)(-178.2123,-98.4812)% \
         (-178.2123,-113.9800)(-187.3223,-126.5188)}{Black} % Dart
\Polygon{(-240.6531,-118.7694)(-216.8029,-126.5188)% \
         (-225.9128,-113.9800)(-225.9128,-98.4812)}{Black} % Dart
                .
                .
\Polygon{(288.3536,93.6918)(303.0939,73.4035)% \
         (312.2039,85.9424)(312.2039,101.4412)}{Black} % Kite
\Polygon{(38.5905,93.6918)(14.7403,101.4412)% \
         (14.7403,85.9424)(23.8503,73.4035)}{Black} % Kite
\Polygon{(38.5905,93.6918)(38.5905,118.7694)% \
         (23.8503,113.9800)(14.7403,101.4412)}{Black} % Kite
%\Polygon{(-328.2706,-205.1691)(328.2706,-205.1691)%\
%         (328.2706,205.1691)(-328.2706,205.1691)}{Black}
\Vertex(0,0){2}
\end{axopicture}
```

The generating C program takes about 350 lines and can make the pattern as big as you might want it. In this case it was a pattern for a table cloth of about 900 pieces of cotton in various colors. In the actual execution some local variations were made in the size of the pieces. The result became:



The example of the pattern for the t-shirt with long sleeves has an interesting problem. How do you make it such that the sleeve fits exactly with the body? Fashion designers use heuristic algorithms to draw lines that will fit more or less, but they can be off by a centimeter. There are special sewing techniques to correct for this. In our example the lines are drawn as the sum of a few Bezier curves and then the problem is that the sum of those (different) curves for the sleeves must be equal to the sum for the body. By making variations in the control points of the Bezier curves one can get the lines correct to small fractions of a millimeter. And the routines to calculate those lengths can be copied from axohelp, if the creation program is written in C at least.

All together it does not take much effort to make a program that for given sizes creates a perfect pattern in terms of a LaTeX program with Axodraw statements.

In the postscript version of Axodraw it is not very difficult to add commands. This can be done with an extra style file, e.g. sudoku.sty for showing sudokus together with their step-by-step solution with explanations.

It is a bit more complicated with the pdflatex version. The LaTeX part is not very complicated. The problem is when calculations are needed, because then the axohelp.c file needs to be extended. By itself that is not very difficult, but then the modified axohelp needs to be used and not the version that comes with the TeX distribution. This makes it a bit less elegant. To avoid this the current paper has been translated in the old fashioned way:

```
latex paper
dvips paper -o
ps2pdf paper.ps
```

This gave however the problem that dvips cannot handle .jpg files and the paper has two photo's included. Fortunately these could be converted with the GIMP program into .eps files.

The code for the sudoku is given by:

```
\begin{center}\begin{picture}(200,200)(0,0)
\SetWidth{0.5 }
\MakeBoard(10,10,20)
\ColorCell(2,6){LightBlue}
\ColorCell(2,8){LightRed}
\PutSudoColor(8,9,2){LightRed}
\PutSudoColor(7,1,2){LightRed}
\PutSudoColor(4,3,2){LightRed}
\PutSudoColor(1,2,2){LightRed}
\PutSudoColor(3,4,2){LightRed}
\PutSudoColor(9,6,2){LightRed}
\PutSudoColor(9,4,2){LightBlue}
\SetCoordinates
\PutCell(1,1,8)
\PutAllow(1,2,4)
\PutAllow(1,3,1)
\PutAllow(1,3,2)
\PutAllow(1,3,4)
\PutCell(1,4,6)
\PutCell(1,5,9)
\PutAllow(1,6,1)
\PutScratch(1,6,2)
\PutAllow(1,6,5)
\PutAllow(1,7,2)
\PutAllow(1,7,4)
\PutAllow(1,7,7)
\PutAllow(1,8,2)
     .
     .
\PutCell(9,5,4)
\PutAllow(9,6,1)
\PutCell(9,7,9)
\PutCell(9,8,3)
\PutCell(9,9,6)
\MakeBoard(10,10,20)
\end{picture}\end{center}
```

and it was completely computer generated. It is an example from a sudoku book that first explains many tricks and then gives 500 sudokus that were generated by a Monte Carlo technique. When one does this, most sudokus are rather trivial. Hence the program generated many millions of them and the more interesting ones were selected. There were also still quite a few that could not be solved with the techniques explained in the book, indicating that maybe there will be a sequel with very difficult sudokus. The book can be found at

`http://www.nikhef.nl/~t68/sudokus/book.pdf`

The origin of the name

In 1976, during my graduate study in Stony Brook, we did our computer work on the CDC computer in Brookhaven. We worked by means of a telephone line and the fast modem was 300 baud. I had a number of kinematics routines for my calculations and I was told things would go much faster if I would make them into a library. Hence I got the CDC manual. It had an example and the name of the library in the example was mylib. That did not look very creative. Hence, how to call it then? I looked around and at my feet was my dog. And the name of the dog was Axo. And there

was the name: axolib. That library has served many years and featured in many calculations. I still use it in lectures about particle kinematics. There are some rather unique routines in it. Years later it was a small step to call the drawing package Axodraw, even though at that moment the dog had been dead already for 7 years.

References

[1] J.A.M. Vermaseren, Comput. Phys. Commun. **83** (1994) 45–58.

[2] D. Binosi and L. Theussl, Comput. Phys. Commun. **161** (2004) 76–86.

[3] D. Binosi, J. Collins, C. Kaufhold, L. Theussl, Comput. Phys. Commun. **180** (2009) 1709–1715.

[4] J. Kuipers, A. Plaat, J.A.M. Vermaseren and J.H. van den Herik, Comput. Phys. Commun. **184** (2013) 2391–2395.

[5] Sven Olaf Moch and J.A.M.Vermaseren, Nucl. Phys. B573 (2000) 853-907.

J.A.M. Vermaseren
Science Park 105, 1098 XG Amsterdam, The Netherlands
t68 at nikhef dot nl
28-Aug-2020