PSlib.eps Catalogue, preliminary and abridged version

— use of PostScript libraries —

Abstract
A selection of PostScript definitions collected in my `PSlib.eps` library and documented as e-book catalogue is presented. Now and then variant pictures have been included from `pic.dat` which comes with `Blue.tex`. Old Metafont codes have been included which may be useful for MetaPost programmers. Variants of pictures enriched by postprocessing in Photoshop show other possibilities. Escher's doughnut is a teaser which has to be done in MetaPost. Next to `PSlib.eps` comes the file `PDFsfromPSlib`, which contains the pictures in `.pdf` format. The complete `PSlib.eps`, `PDFsfromPSlib` as well as the catalogue as e-book, will be released on occasion of NTG's 25th lustrum which will be celebrated in the fall of 2014, on `www.ntg.nl`. A prerelease will be offered to the GUST's file server. The (static) library for TEX alone pictures, `pic.dat`, packaged with `Blue.tex`, will be redistributed as well.

Keywords
Apollonius, Bluebook.eps, Cantor, CD-DVD label, Deubert, Escher, flowcharts, fractals, Gabo, Julia, Koch, Lancaster, Lauwerier, length Bézier curve, Lévy, Lindenmayer, Mandelbrot, Mondriaan, Op Art, orthogonal circles, Photoshop, pie chart, pi decimals, PostScript, projection, PSlib.eps, Pythagoras Tree, smiley, Soto, stars, (plain) TEX, text along path, tiling, toroid, SoC (Separation of Concerns), Schrofer, Vasarely. Yin Yang

Contents

Introduction

Gutenberg organized in Toulouse in the 90-ies a meeting themed Graphics and TEX. Later, especially after MetaPost had been released in the public domain, GUST paid attention to graphics. Piotr&Piotr developped `PSView`. LATEX came with its picture environment and Knuth c.s. developed the gkp-macros with the picture environment functionality for use in plain TEX. `Blue.tex` appeared with its library `pic.dat` of TEX alone pictures, biased by the gkp-macros. On the EuroTEX92 in Prague Karel Horak showed his math pictures created by Metafont.[1] Later Timothy van Zandt released

1. Karel works nowadays in MetaPost, with `.eps` and `SVG` output.

PSTricks, which uses PostScript as the graphics engine and LaTeX's picture environment as user interface. Herbert Vo maintains and extends PSTricks. PostScript was there all the time, since 1985 to be precise, but ...was apparently overlooked for graphics, except by Don Lancaster, who came with his PSSecrets and Gonzo collection.[2] PostScript was in use as (abstract) page description language for the upcoming laser printers.

In relation to TeX, we could in the mid-90-ies include .eps pictures in documents via the psfig command, to be processed by DVIPS.[3] Jackowski developed MFtoEPS, to transform Metafont pictures into PostScript for inclusion in TeX documents. Alas, pdfTeX does not allow direct inclusion of .eps files, a retrograde. Happily, ConTeXt not only allows inclusion of .eps files, but also works interactively with MetaPost.[4]

Of my ≈25 years of involvement with, and using, TeX, I spent ≈5 years on developing Blue.tex and ≈20 years on creating and collecting pictures. Lauwerier's BASIC pictures, especially his fractals, I have been translated into PostScript. As an aside I exercised HTML for my WWW-site.

PSlib.eps was introduced at the EuroTeX2009. The library facilitates programming in PostScript. Hundreds of codes for the pictures are stored in a ordered way, facilitating its use. The catalogue is the accompanying documentation,[5] where next to the code the result of the code has been shown. It contains Adobe's Blue book defs as starting point. Much more, what I needed, has been added. The collection has been tested by example. I don't claim that it has been thoroughly tested nor is the collection robust, although the numerical concept robustness does not apply to graphics, in general, it occurs. Since EuroTeX2012 it has grown substantially. Older PostScript codes with local dictionaries have been added. For the 25th lustrum meeting of NTG, in the fall of 2014, I decided to make the library more easily available, and come up with a catalogue, such that the desired code for the wanted graphics can be more easily spotted and used, hopefully. Numerici have their program libraries, we should have our catalogues for pictures, to start with Adobe's Blue book collection and my PSlib.eps extensions. I consider catalogues for graphics useful, because we lack a taxonomy, the world of pictures is too diverse. May the catalogue contribute to the "literature" of PostScript graphics programming. May PostScript start a second glorious graphics life. Hang on and enjoy!

Why PSlib.eps?

The idea of a library of pictures I already realized in Blue.tex in the 90-ies. The collection is called pic.dat and will also be redistributed along with PSlib.eps. It contains graphics by TeX alone via the macros of Graham-Knuth-Pasternak, the gkp-macros.

PostScript is a powerful language for graphics. But ... it is not enough. Extended with defs as given in Adobe's Blue book makes the use of PostScript feasible. Reality has it that Adobe's Bluebook is apparently not enough for programming in PostScript. Why?

My answer is

□ there are not enough Blue book defs, there are hardly no contributions from the community
□ the viewer and convertor Acrobat Pro is not in the public domain; GhostView and PSView appeared later

2. Gonzo utilities facilitate using PostScript as formatter for texts. Don does not use local dictionaries for library robustness.
3. My papers of that time can't be processed by pdfTeX. -;(. They have to be adapted and the pictures must be converted into .pdf, .jpg, ...
4. An example where TeX-Graphics interaction occurs naturally is in typesetting crosswords. But, ...a crossword is a simple graphic, so we can do with TeX alone.
5. Physical thick, logical thin!

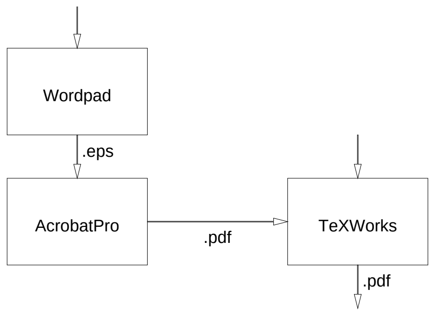
- □ the defs have not been made easily available, there was no PD PostScript library file on the WWW
- □ MetaPost is of higher level, with its inheritance of the beautiful Metafont language, which is more in use than PostScript, within the TEX community, Knuth included
- □ Adobe has abandoned Type 1 fonts in favour for OTF, and seems no longer to support the `run` command, which I use for library inclusion[6]
- □ and, indeed … PostScript is unusual, low-level, difficult and error-prone to program in, alas. But … if you do, the rewards are more than enough. We should finish up our programming of pictures by collecting the pictures and their codes into a library and document it, in for example the accompanying catalogue.

MetaPost has an explicit datastructure for paths which comes with nice operators such as `cutbefore`, `cutafter`, `intersectionpoint`, `interpath`, and `intersectiontimes`, which I used in the MetaPost code for Escher's doughnut in order to cope elegantly with the hidden lines. PostScript has the `arc` operator, for circular arcs. PostScript has no explicit paths. Nor has it a picture datastructure. PostScript has the powerful `flattenpath` which delivers an arbitrary path as a broken line, of which the points can be used by `pathforall`. By these tools the length of a path can be (approximately) calculated by Blue book's `pathlength`, for example. The absence of the path and picture datastructures in PostScript is a big lack. We will concentrate on graphical shapes and leave the glorious laserwriter history for what it is. Having said that, let us move on and see what kind of graphics can be written in PostScript.

However, … 20 years after the birth of MetaPost, I still don't have a pleasing MetaPost working environment on my PC.[7] The TEXlive DVD does not provide me such an IDE, except as integrated part of ConTEXt. Besides, much graphics can be programmed in PostScript, as is demonstrated in my `PSlib.eps`. A notational exception is the Escher doughnut, and ilks, with its hidden lines.

In my `PSlib.eps` catalogue the Blue book `defs` have been made digitally available and extended by local dictionaries. This collection has been extended by `defs` of my own and some collected from the WWW.[8] A couple of hundreds of `defs` emerged over the past 20 years. It has been strived after that `defs` can be read, understood, reused and adapted to your circumstances. Simplicity of use is paramount. The accompanying `PDF`s have also been collected and can be reused as such. Some graphics programs are no longer relevant, for example Adobe's program for a pie chart, because of special and easy to use packages, not in the least the wealth of LATEX packages.

```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Flowchart TeX processing and PS inclusion
%%BoundingBox: -101 -31 102 221
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
%%EndProlog
flowchartTeXandPS   showpage
%%EOF
```



6. Your old Acrobat Pro 7 is a treasure for converting `.eps` into `.pdf` and viewing it. Cherish Acrobat Pro 7!
7. Troy Henderson has his MetaPost viewer, which can be accessed and used remotely. It is just a pity that he has not released a local, strand-alone version.
8. Operators given in Lancaster's PSSecrets are hard to read and yielded non-working results. For example his Archimedian spiral, where he did not use polar coordinates, is unnecessarily complicated and did not work. On the other hand his Fonts for Free are a gem.

The first 6 lines are standard overhead for the kind of file, `!PS` means PostScript, providing values for the `BoundingBox`, specifying the library and its loading via `run`, cropping of the graphics to the `BoundingBox`, via the `BeginSetup` and `EndSetup` duo, `BeginProlog` and `EndProlog` enclose the needed `defs` (for example the library) and finally the invoke of the `def`: `flowchartTeXandPS`.

For the invoke one just has to known the name of the library `def` and its parameters. The parameters are documented in the code. The code is as extra supplied in the library contributions in this catalogue. Adding the library definitions to the example of use makes the example look more complex, but ... has been done on purpose to facilitate adaptability, sacrificing simple look-and-feel. However, this approach supports stand-alone use.

Processing the graphics separately and include the resulting `.pdf`, or converted `.jpg` which is more efficient in TEXworks, into your AnyTEX(, or Word, or ...) document, adheres the Separation of Concerns principle. A wider audience than the TEXcommunity, such as PostScript users, users of Word, ..., is served.

`PSTricks` based packages look nice and easy, but they suffer intrinsically from the deficiency of LATEX's picture environment, as 'graphical language,' such as the limited orientation of lines. But, ... the advantage is that the user is hidden from PostScript, does not have to know PostScript.

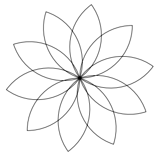
Why PostScript?

The use of a programming language has two aspects: the richness and power of the language proper and the ease of working in an IDE.

PostScript language The language was designed by Adobe, and introduced in 1985, as a device-independent page description language with powerful graphics capabilities, with initially $\approx$ 400 operators — i.e. built-in procedures of the system dictionary — in PostScript level 1, with substantial more in PostScript level 3. The extensive graphics capabilities are embedded in the framework of a general-purpose programming language. The language includes a conventional set of data types, such as numbers, booleans, arrays and strings; control primitives, such as conditionals, loops and procedures; and some unusual features, such as dictionaries, font transformation matrix, next to higher-level structures, such as patterns and forms. These features enable application programmers to define higher-level operations that closely match the needs of the application and then to generate commands that invoke those higher-level operations. The LRM 3 is the defining document for the syntax and semantics of the language, the imaging model, and the effects of the graphics operators.

Powerful concepts

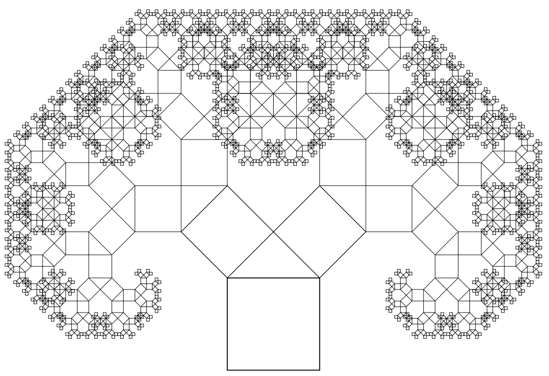
□ A user space which can be altered: "the coordinate system's origin may be *translated*, moved to any point in user space; the axes may be *rotated* to any orientation; the axes may be *scaled* to any degree desired; the scaling may be different in the x and y directions." Reflection and skewing are also supported. My favourite illustrations of the US concept are a stylistic flower and the recursive programming of the Pythagorean tree, which is all about drawing a square in repeatedly transformed US.



← Stylistic Flower

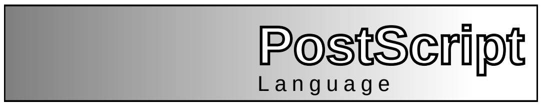
```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -26 -26 26 26
%%BeginSetup
%%EndSetup
/r 18 def
10 {0 r r 270 360 arc
      r 0 r   90 180 arc
     36 rotate} bind repeat
stroke showpage
```

Pythagorean Tree →
BachoTEX 2011 pearl



Another nice example of the usefulness of transforming US is to create a path of an ellipse by the use of the `arc` operator, for example `1 2 scale 0 0 25 0 360 arc` (Courtesy the Blue Book, but … be aware of the fact that the pen width has been scaled too).
To translate the centre of the coordinate system, default in the left lower corner of the current page, was the first thing I used to do. No longer necessary for my stand-alone illustrations in an EPSF program, which begin with the 4 lines as given in the stylistic flower example.

□ The colour spaces, which notion was introduced in PostScript 2 of 1991, and elaborated upon in 1997 in PostScript 3, with among others much more efficient shading functionality. In PostScript 1 there were already the concepts colour mode and half-tones, with operators `setrgbcolor` and `setgray`, which were generalized in level 2 into `setcolorspace` with `setcolor`. The chapter headings of the level 1 Red and Blue Books reflect the gradients, or smooth shading, functionality.



Inspired by The Green Book p139
Much can be learned from this example
which was state of the art in 1985
Level 3 features rich colour shading

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 200 36
/DataString 256 string def
/oshow {true charpath stroke} def
/H20 {/Helvetica-Bold 20 selectfont} def
/H8 {/Helvetica 8 selectfont} def
%%EndProlog
0 0 moveto
gsave 175 36 scale
0 1 127 {DataString exch dup 128 add put}bind for
128 1 8 [128 0 0 1 0 0] {DataString} image
grestore
0 0 200 36 rectstroke 1 setgray
H20 95 14 moveto  (PostScript) show 0 setgray
    95 14 moveto     (PostScript) oshow
H8  95  4 moveto 2 0 (Language)   ashow
showpage
%%EOF
```

The number of pages of the PostScript level 3 LRM has increased to 912p, while the number of pages of the level 1 LRM is 321p. See Appendix A of the LRM 3 for the ways the PostScript language has been extended with new operators and other features over time. The language versions are upward compatible.

- ☐ Handy and useful optimizations, such as `rectstroke` `rectfill` `userpath` `select-font` …
  but … don't use `store` in library procedures instead of `def`, with variables which **are intended** to have a local scope. Use a dictionary local to the library procedure for variables.
- ☐ The graphics state, with commonly used operators `gsave` `grestore` … .
- ☐ The current page, the abstract page, the raison d'être concept behind PostScript, to be rendered by the interpreter in the rendering device (printer, screen, …), commanded by the `showpage` operator.
- ☐ Variability of fonts by the font transformation matrix as argument of `makefont`, with scalability, mirrored fonts (as in XƎLATEX) smallcaps, and outlines as obvious examples.
- ☐ Stacks `operand dictionary graphics state execution`
- ☐ Immediately evaluated names, i.e. `//name` is immediately replaced by its value; useful for named constants.
- ☐ `bind` operator which looks up values of the **operator** names in the procedure operand and replaces these by their values, the so-called early binding, and returns the modified procedure. It enhances efficiency and robustness against (unintended) change of used operators, especially in library procedures. Optimize loop bodies too by `bind`.
- ☐ Idiom recognition feature of Level 3. A functionality added to the bind operator, which can be switched on/off by setting the user parameter `IdiomRecognition`. Bind can find and replace commonly occurring operators, called *idioms* by operators of higher quality and/or better performance. For example PostScript level 2 shading operators are replaced by PostScript level 3 improvements, silently behind the scenes, with new snazzy codes.
- ☐ `execform` for repeatedly placing a graphic, e.g. a logo, a fill-in form, … efficiently (since level 2).

Operator groups, with a few operators named from each group:

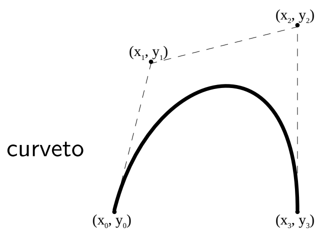
- ☐ operand stack `pop exch dup ...`
- ☐ arithmetic and math `add div ... srand`
- ☐ array `array ... getinterval`
- ☐ dictionary `dict ...dictstack`
- ☐ string `string ... run ... token`
- ☐ relation, boolean, and bitwise `eq ... bitshift`
- ☐ type, attribute, and conversion `type ... cvs`
- ☐ file `file = == ... echo`
- ☐ file inclusion `run`
- ☐ virtual memory `save restore ...`
- ☐ miscellaneous `bind null usertime version`
- ☐ graphics state `gsave grestore ... currenttransfer`
- ☐ coordinate system and matrix `matrix ... invertmatrix`
- ☐ path constructing `newpath`, `moveto lineto curveto arcto ... clip eoclip ... charpath ...`
  Only one path is possible, although by juggling with several graphics states one may maintain several collateral paths
- ☐ painting to the current page of
  – paths `stroke` paints lines, `fill eofill ...` fills an area
  – strings `show ...`
  – a sampled image `image shshow ...`
- ☐ device setup and output `showpage ...`
- ☐ character and fonts `findfont scalefont setfont` (or level 2 onward optimized concatenation of the 3 into `selectfont`) `makefont` (transforms more general than `scalefont`) `... kshow ... cshow ...`

□ font cache `setcharwidth ...`
□ errors `dictfull ...VMerror`.

Overwhelming, isn't it?

Let us pick out a few, which I use most of the time, apart from the arithmetic, math and relation operators, whose use we are already familiar with from our favourite programming language.

`def`	associates names with procedures or values available on the operand stack, and stores the associated pair in the current dictionary
`run`	includes the file given in parenthesis
`moveto`	creates the starting point of an (internal) path
`stroke`	and ilks, to paint the path to the current page
`lineto`	adds a line to the current path
`curveto`	adds a curve to the current path
`arc, arcn`	adds a circular arc to the current path
`image`	to render the (bitmap) image onto the current page
`gsave`	pushes the current graphics state on the gs-stack and creates a new current one
`grestore`	pops the (top) graphics state off the gs-stack and makes it the current graphics state, en passant obseleting the graphics state in use
`makefont`	is used by Don Lancaster (and undoubtedly by others) for creating a variety of fonts from the available ones. He calls his collection "Fonts for Free". I love his embossed variant
`kshow`	I used kerning show in my BachoTEX 2010 pearl for the typesetting of π-decimals along a spiral
`forall`	handy for creating concise code, used for example in my BachoTEX 2010 pearl.



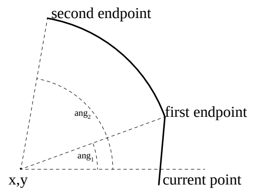
The language (version 3) is stable and flexible, also called extensible, meaning one can add procedures. It is the industry page description standard language. Programs are interpreted, line by line, not compiled, which at the time was important because of small memories. It is maintained by Adobe — the stewards of PostScript — and already with us for more than 25 years! Interpreters are generally provided by 3^{rd} parties, especially the interpreters which come with printers.

For graphics `lineto`, `curveto` and `arc`, and ilks, is what makes the drawing. Let us assume a_0 is the current point.

a_1 `lineto` adds a straight line to the path.
A point $z(t)$, t∈[0,1] on the line is given by $z(t) = (1-t)a_0 + ta_1$.

a_1 a_2 a_3 `curveto` adds a B-cubic to the path with end point a_3 and control points a_1 and a_2.
A point $z(t)$, t∈[0,1] on the B-cubic is given by $z(t) = \sum_{k=0}^{3}(1-t)^{3-k}t^k a_k$.

x y r ang$_1$ ang$_2$ `arc` appends a counterclockwise arc of a circle to the current path, possibly preceded by a straight line segment. The arc has (x,y) as center, r as radius, ang$_1$ the angle of a vector from (x,y) of length r to the first endpoint of the arc, and ang$_2$ the angle of a vector from (x,y) of length r to the second endpoint of the arc.



Knuth, in the Metafont book, paid much attention to a path through a set of points. A smooth curve through a set of points is not available in PostScript, as such.

The Red, Green and Blue books — Reference Manual, PostScript Language Program Design, respectively Tutorial and Cookbook — are published by Addison-Wesley and also available for free on the WWW, even the level 1 and 2 (774p) LRM's. The Blue Book, which was aimed at to set a standard for effective PostScript programming, contains examples of procedures, such as `oshow outsidecircletext insidecircletext pathtext pathlength printposter DrawPieChart centerdash smallcaps` and `arrow`, to name but a few.[9] The Red Book is indispensable. The Green Book is about software engineering in PostScript, not only to get the programs to work, but to create correct, readable, efficient, maintainable and robust PostScript programs. The underlying goal is to develop a printer driver. There is also an Adobe White Book about Type 1 fonts, also for free on the WWW. Mnemonics: the RGB-collection of Adobe :-). PostScript programs, in ASCII, are generally generated by programs, hardly self-written. They facilitate exchange of (stand alone EPSF) picture descriptions, and of course (the pages of) a complete publication. The structuring conventions of Appendix C of the Red book level 1 have grown out into an Adobe Technical note #5001, 109p. Nowadays, illustrations are easily exchanged in `.pdf`, and everybody can view them everywhere, because of the ubiquitous, free Acrobat reader. The Mathematica reader is also freely available, and facilitates the exchange of Mathematica `notebooks`. The exchange of ASCII PostScript goes without problems and is useful.

Although a powerful graphical language, PostScript is considered low-levelish by the TEX community at large. They favour John Hobby's preprocessor MetaPost, Knuth included, with exceptions: Taco Hoekwater, me … . Taco includes PostScript on-the-fly in `escrito`, his PostScript compatible interpreter in Lua. He is also on the MetaPost maintenance team and works on extensions of MetaPost. At BachoTEX 2010 he announced the release of MetaPost2.000.

"PostScript is underestimated and underused," to quote Taco Hoekwater.

"I agree with him … I'm not saying he is right ;-)," well … he is.
IMHO, a little bit of PostScript does not harm. On the contrary, you will benefit from the general-purpose programming language framework, the imaging model, or you may extend your knowledge about the interactive system for controlling raster output devices, but … self-study is dangerous. What we need is a teacher à la John Deubert who thoroughly understand the PostScript concepts. John in his Acumen Journal pays attention to among others the relation of PostScript to `PDF` and `XPS`, and gives many nice, good and useful examples, clearly explained line by line.

From the LRM: "PDF lacks the general-purpose programming language framework of the PostScript language. A PDF document is a static data structure that is designed for efficient random access and includes navigational information suitable for interactive viewing."
Finally, and in contrast with TEX, a mnemonic

All what you type in PostScript are

- ☐ comments after `%`
- ☐ numbers
- ☐ operators
- ☐ names to be looked up in the dictionaries, and at last
- ☐ strings which contain text.

From the LRM

"The interpreter manipulates entities called PostScript objects. Some objects are data, such as numbers, boolean values, strings, and arrays. Other objects are elements of programs to be executed, such as names, operators, and procedures."

In TEX the source is the text of the publication, interspersed with as few markup commands as possible, at least that is what Knuth aims at in his minimal markup style, which I love and practice too.

So …

`add def ...` are names to be looked up in the dictionaries

`( < / [ ...` are operators:

9. The procedure texts and the examples are available on the WWW as a UNIX shell archive. I have assembled the procedures into a `BlueBook.eps` library, which is system independent and also incorporated in `PSlib.eps`.

- `(` starts a string, where all is interpreted as text, with `\` as escape character; a TEXies niche
- `<` starts a hexadecimal string, consisting of the "digits" in the hexadecimal system `0,1, ... 9, a, b, c, d, e, f`, commonly used in the executable array, ie, procedure, as argument for the `image` operator
- `/` starts a literal name
- `[` starts an array, which may contain heterogeneous elements, in contrast with other languages.

Be aware of the difference between `name` and `/name`. The first is a `name` to be looked up in the dictionaries, while the slash in the second starts a *literal* name, which is only pushed on the operand stack, without execution. Unlike other programming languages such as PASCAL there are no reserved words.

Comments start with `%` Comments are used for structuring. Special comments are

- `%!` the start of a PostScript program with `%!PS-Adobe-3.0 EPSF-3.0` the complete line for illustrations to be encapsulated
- `%%` at the beginning of a line starts structural information about the PostScript program, as explained in Appendix C of the LRM version 1, or see ATN #5002; syntax `%%<keyword>: parameter values`.

EPSF Encapsulated PostScript File format Looking more closely at the `.eps`, which resulted from `.mp`, I found that header comments of a PostScript program starting with the error-prone mumbo jumbo

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: llx lly urx ury
%%BeginSetup
%%Endsetup
```

yields the image cropped to the supplied `BoundingBox: llx lly urx ury`, centred on the page, when processed by Acrobat Pro 7.1.[10] Explicit translation of the origin via `... translate` is no longer necessary in an EPSF with a BB around the origin. Very Handy! Not only very handy ... also with better results than my old way via selecting, copying and the reuse of the copy from the clipboard. The dimensions of the BoundingBox can be requested for in the program, to create a perfect cropped illustration.

From Adobe Technical Note #5001 PostScript Language Documents Structuring Convention Specification, the following about the keyword `EPSF-3.0`

... `EPSF-3.0` states that the file is an Encapsulated PostScript Format, which is primarily a PostScript language file that produces an illustration. The EPS format is designed to facilitate including these illustrations in other documents. ...

Appendix H of the LRM version 2 details on EPSF, which by the way is not included in the LRM 3.

Contents of the library

`PSlib.eps` contains constants, `defs` for operations, `defs` for pictures, and dictionaries which are coupled to the `defs`.

Constants are for example

```
□ /sqrt2 1.414213562 def
  /sqrt3 1.732050808 def
  /sqrt5 2.236067977 def
```

10. An undocumented feature? Non-universal?

```
  /pi 3.14159265358979 def
  /reus 2 30 exp  dup 1 sub add def
```
i.e. 2.14748e+09.

□ font abbreviations, such as `/H14pt /Helvetica 14 selectfont def`
□ colour presettings, such as `/lightred{1 0.3 0 setrgbcolor}`.

Definitions for operations, the utilities

Definitions for 2D pictures For example the flowcharts, Escher, Soto, Schrofer, …

Definitions for 2D pictures as projection of 3D emulated objects For example a sphere, pictures of the emulations of Gabo's constructions.

```
/linearconstructionno1{linearconstructionno1dict begin
 /phi 30 def /theta 5 def /scalingfactor 50 def
reversevideo 1 setgray%white
3  setlinewidth frame        stroke
2  setlinewidth approxellipsestroke %sampleellipsestroke
.5 setlinewidth dostringing  stroke
annotation
end}def
```

Definitions for dictionaries The dictionaries contain auxiliaries which are declared, initialized and kept local. For example the dictionary for `linearconstructionno1` has the components[11]

```
/linearconstructionno1dict 50 dict def

linearconstructionno1dict begin
/reversevideo {-130 -135 260 270 rectfill} def %Mimics BoundingBox
%Data
/origin { 0 0 } def
/a0f  { .1 s  -2 s  -2 s ptp  }def%y constant
/a1f  { .6 s  -2 s  -1 s ptp  }def
/a2f  { .6 s  -2 s   1 s ptp  }def
/a3f  { .2 s  -2 s   2 s ptp  }def
...

/sampleellipsestroke{gsave -45 rotate .6 1.2 scale%kind of ellipse
...
stroke grestore}def
%
/approxellipsestroke{gsave -45 rotate .6 1.2 scale%kind of ellips
...
grestore} def

/frame{a0f moveto  a1f  a2f  a3f curveto
...} def

/dostringing{0.02 .02 .5001{/t exch def
...
closepath }for } bind def

.1 setlinewidth stroke

/annotation{-32 -125  moveto ...}
 ..} def
end%linearconstructionno1dict
```

11. Much of the contents had been omitted here because it is about structural elements.

Structure of each listed code contribution

The approach is similar as adopted in Adobe's Blue book. Of each code included in the library, `PSlib.eps` cq `Bluebook.eps` the example of use is listed. The code is also included. The library codes used are mentioned in the code now and then. At the right of the code the resulting picture is shown. Variants are included.

Utilities

Just a few utilities have been selected and discussed.

Length. A too simple operator? Not so.

One must circumvent intermediate (numerical) overflow, and... use the stack only. Moreover, the name `length` is already in use as a so-called polymorphic operator — takes different kinds of arguments — also called an overloaded operator in Ada, for example.

$$\begin{aligned}|(x,y)| &= \sqrt{x^2+y^2}\\ &= |y|\sqrt{1+(x/y)^2} \qquad \text{numerically better if} \quad |y|\geq|x|\\ &= |x|\sqrt{1+(y/x)^2} \qquad \text{numerically better if} \quad |x|\geq|y|\end{aligned}$$

```
/size %x y ==> sqrt(x^2+y^2)
      %not robust against 0 0 on the stack
{abs dup  3 -1 roll abs dup 3 1 roll % y x y x
le {size}                            % y <= x
   {exch dup 3 1 roll                % y x y
     div                             % y x/y
     dup  mul 1 add sqrt mul
    }ifelse
}def
```

The operator is related to the polar coordinates (r,ϕ) of a point (x,y). In PostScript the angle ϕ can be obtained via the `atan` operator; for the size r one has to provide an operator, length, oneself.

Overloading length[12]

```
%!PS Overloading length. Taco Hoekwater April 2010
/PSlength {length} bind def % save old meaning
/lengthdict 5 dict def
lengthdict /arraytype   {PSlength} put
lengthdict /dicttype    {PSlength} put
lengthdict /stringtype  {PSlength} put
lengthdict /integertype {size} put
lengthdict /realtype    {size} put
/length { lengthdict begin dup type exec end} def
```

```
%Test
(whatever) length pstack pop
[1 2 3] length pstack pop
3 dict length pstack pop
3 4 length pstack pop
```

pathlength is part of Adobe's Blue book P5 p143, Centered dash patterns. The procedure `pathlength` computes the length of any given path. It does so by first flattening the path with the `flattenpath` operator. `flattenpath` converts any `curveto` and `arc` segments in a path to a series of `lineto` segments. Then the procedure `pathforall` is used to access each segment in the path, find its length and add the length to a total. A PostScript programming paradigm.

12. Courtesy Taco Hoekwater

Mean of two points The problem is to calculate .5[p_1, p_2]. Too trivial?

I found it worthwhile to communicate, because the operator makes use of the stack only. Stack-oriented PostScript, different from the PostScript I used in my `Just a little bit of PostScript`, of old.

```
/mean%p0 p1 on stack ==> .5[p0, p1] i.e. the coordinates of the mean
 {exch  4 -1 roll add .5 mul 3  1 roll add .5 mul}def
```

More general is the calculation of a point on the line between the two points. The `mediation` operator: t[p_1, p_2], t$\in[0,1]$.

```
/mediation {% a b t ==> c. c is weighted average of a and b;
            % c = a * (1-t) + b * t
dup 1 exch sub 4 -1 roll mul
3 1 roll mul add
} bind def
```

Rotation of a point Despite PostScript's functionality to rotate user space, I needed an operator to rotate just points. I chose conform the PostScript tradition that a positive angle rotates counterclockwise.

x y angle `rot` x' y', rotates the point x y over the angle `angle` in degrees, counterclockwise in the PostScript tradition

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

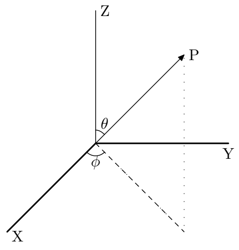
```
/rotdict 6 dict def
/rot %x y phi: point and angle of rotation (counterclockwise)
     %         ==> x y coordinates of point after rotation
{rotdict begin
 /phi exch def /y exch def /x exch def
 /xaux x phi cos mul y phi sin mul sub def
 /y     x phi sin mul  y phi cos mul add def
 /x xaux def
 x y
 end
 } def
```

Projection of a point, operator ptp. For projections I specify the graphics in 3D and project the data onto the plane by the operator `ptp` with viewing angles as parameters.[13]

□ num_1 num_2 num_3 ptp num_1 num_2, projects the 3D point on the plane with viewing angles ϕ and θ

The projection formula, with ϕ and θ viewing angles, reads

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} -\cos\phi & \sin\phi & \\ -\sin\phi\sin\theta & -\cos\phi\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$



```
/ptp{% point x y z ==> x' y'
      % use: /pair { x y z ptp } def
      % parameters: a, b viewing angles
   0 begin
   /z exch def/y exch def/x exch def
   x neg a cos mul y a sin mul add
   x neg a sin mul b sin mul y neg a cos mul b sin mul add
```

13. The projection I also coded in MetaPost.

```
   z b cos mul add
   end}def
/ptp load 0 3 dict put
```

Indispensible. A practical variant with fixed viewing angles, `ptpf`, is part of the `PSlib.eps` library too.

dotsandnames I consider it convenient to mark points in a picture with their names. I decided to supply the names of the points as a string in an array. Each name, a1 for example, is a definition which contains the coordinates of the point a1.

```
/dotsandnames{%[ str,..., str]==>
%str is a name, which stands for a pair, such as in pair moveto
{dup cvn load exec moveto (.) H15pt setfont centershow
                               H10pt setfont show}forall
}def
```

A nice use of `forall` and of using the contents of the name and the name itself. `dotandnames` is used in the emulations of Gabo's constructions. A PostScript programming paradigm.

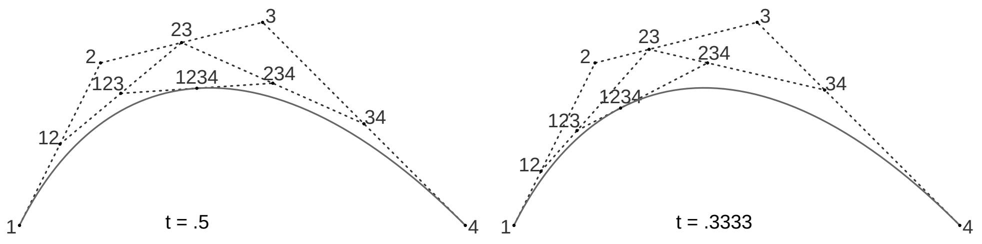
tonSpline yields the coordinates of a point on a spline given by the points (a0,a1,a2,a3)[14] as function of the time parameter $t \in [0,1]$: $z(t) = \sum_{k=0}^{3}(1-t)^{3-k}t^k a_k$. Algorithmically,[15]

$$\begin{aligned} z(t) =&(1-t)\Bigg((1-t)\Big((1-t)a_0 + t\,a_1\Big) + t\,\Big((1-t)a_1 + t\,a_2\Big)\Bigg) \\ &+t\Bigg((1-t)\Big((1-t)a_1 + t\,a_2\Big) + t\,\Big((1-t)a_2 + t\,a_3\Big)\Bigg) \end{aligned}$$

The (vector) value $z(t) = a_{0123}$, of the B-cubic characterized by a_0, a_1, a_2, a_3 for the value of the (time) variable t can algorithmically also be described as

$$\begin{array}{l} a_0 \\ a_1 \\ a_2 \\ a_3 \end{array} \rightarrow \begin{array}{l} a_{01} = a_0\,(1-t) + a_1\,t \\ a_{12} = a_1\,(1-t) + a_2\,t \\ a_{23} = a_2\,(1-t) + a_3\,t \end{array} \rightarrow \begin{array}{l} a_{012} = a_{01}\,(1-t) + a_{12}\,t \\ a_{123} = a_{12}\,(1-t) + a_{23}\,t \end{array} \rightarrow a_{0123} = a_{012}\,(1-t) + a_{123}\,t$$

Graphically, Knuth displayed in the Metafont book p13 the determination of a point on a cubic spline.



The PostScript operator in `PSlib.eps` has been written by Piotr Strzelczyk.

Intersection of 2 straight lines. This functionality was needed when I imitated GUST's batteship logo in PostScript. Under the hood 2 linear equations in 2 unknowns have to be solved. The operator is called `intersect`.

14. It is interesting to see the control points back as coefficients in the Bernsteĭn polynomial representation.
15. Named after de Casteljau.

```
/intersect {%p1 p2 p3 p4 -> x y
makecoef 7 3 roll  makecoef  solveit
}def
```

The library contains also an operator for solving 3 linear equations in 3 unknowns.

Fonts for free Don Lancaster in his PSsecrets gives many font variations.

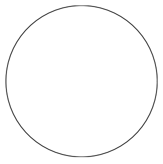
shadow

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Shadow font, Don Lancaster, 1990
%%BoundingBox: -1 -25 180 30
%%BeginSetup
%%EndSetup
.8 setgray  /msg (Free font) def
/Palatino-Bold findfont [40 0 32 -30 0 0]
  makefont setfont
 0 0 moveto msg show%shadow
 0 setgray /Palatino-Bold 40 selectfont
 0 0 moveto msg show
 showpage
 %%EOF
```

BoundingBox calculation by PostScript.
PostScript provides pathbbox for calculation of the size of the surrounding rectangle of a path. Together with setpagedevice this may be used to crop a picture on the fly.

```
%!PS-Adobe-3.0
%%Title: One-pass cropping, LRM 2
/one-passcroppingdict 6 dict def
/one-passcropping{one-passcroppingdict begin %example
/Times-Roman 30 selectfont
0 0 moveto (StarLines) false charpath
                 flattenpath pathbbox
/ury exch def /urx exch def /lly exch def /llx exch def
/w urx llx sub cvi def /h ury lly sub cvi def
<</PageSize [w h]>>setpagedevice
newpath
/rays{120{0 0 moveto 108 0 lineto 1.5 rotate
     }repeat stroke}def
0 1  moveto (StarLines) true charpath clip
newpath 50 -15 translate rays showpage
end}def
```

For a picture PostScript prompted me values, which I inserted in the BoundingBox structure command.



```
%!PS-Adobe-3.0
%%Title: One-pass cropping, LRM 2.3
%%BoundingBox: -50 -50 150 150
%%BeginSetup
%%EndSetup
(c:\\PSlib\\PSlib.eps) run
/one-passcroppingdict 6 dict def
/one-passcropping{one-passcroppingdict begin %example
50 50 100 0 360 arc flattenpath pathbbox
/ury exch def /urx exch def /lly exch def /llx exch def
ury showobject urx showobject lly showobject llx showobject stroke
showpage end}def
one-passcropping
%%EOF
```

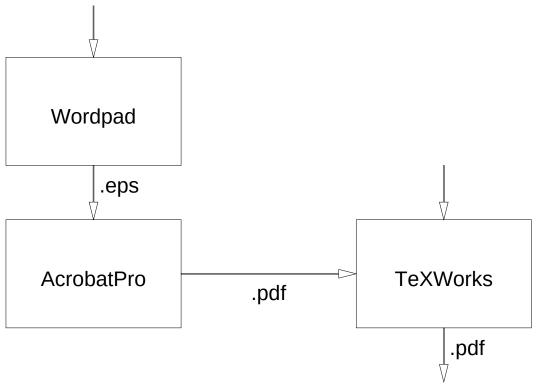
In practice I do not use this facility. I estimate the `BoundingBox` of a picture, reasonably. In the example it is a trifle.

2D pictures

Merging .eps pictures in .tex documents: flowchartTeXandPS

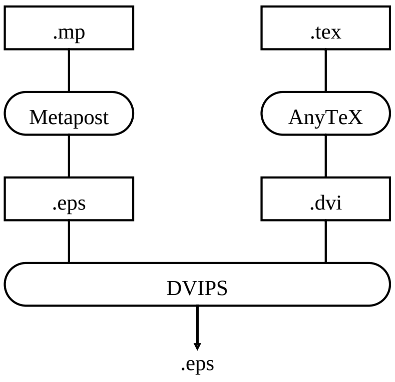
`.eps` pictures as such can't be included when processing with the TEX-engine pdfTEX, which I use in TEXworks. They first have to be converted into `.pdf`. They are more efficiently processed by TEXWorks when a priori converted into `.eps`.

MetaPost users prefer Hobby's `boxes` package for structure diagrams or flowcharts, I presume. I consider the use of straight PostScript equally simple or equally difficult, depending on your expertise.



```
%!PS-Adobe-3.0  EPSF-3.0
%%BoundingBox: -51 -63 252 153
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/flowchartTeXandPSdict 15 dict def
flowchartTeXandPSdict
begin%local auxiliaries
/s 50 def /t s .61803 mul def
/2s s s add def /2t t t add def
/3s 2s s add def /3t 2t t add def
/4s 2s 2s add def
/-s s neg def /-t t neg def
/-2s 2s neg def /-2t 2t neg def
/-3s -2s -s add def /-3t -2t -t add def
/Courier 12 selectfont
end
/flowchartTeXandPS%uses: arrow, centershow from PSlib.eps
{flowchartTeXandPSdict begin
gsave
-s -t 2s 2t rectstroke 0 -7 moveto (AcrobatPro) centershow %Acrobat Pro 7 box
0 2t 0 t  .5 5 10 arrow stroke                            %.eps downarrow
0 1.5 t mul moveto ( .eps) show                           %text right of downarrow
0 3t translate
0 2t 0 t .5 5 10 arrow stroke
-s -t 2s 2t rectstroke 0 -7 moveto (Wordpad) centershow    %WordPad box
grestore 4s 0 translate
-s -t 2s 2t rectstroke 0 -7 moveto (TeXWorks) centershow   %TeXWorks box
-3s 0 -s 0 .5 5 10 arrow stroke                           %graphics inclusion arrow
-2s -15 moveto ( .pdf) centershow                         %subcript below arrow
-3s 0 -s 0 .5 5 10 arrow stroke                           %graphics inclusion arrow
0 -t 0 -2t .5 5 10 arrow stroke 0 -1.5 t mul moveto ( .pdf) show %result down arrow
end} def
%%End Prolog
flowchartTeXandPS        showpage                          %invoke
%%EOF
```

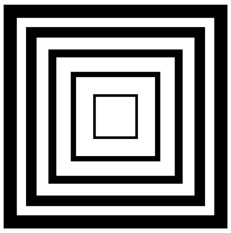
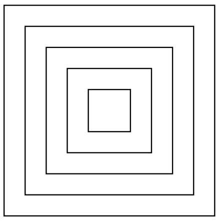
Compared with the flowchart as given in *Just a lttle bit of PostScript*[16] the inclusion of pictures produced by PostScript in TEX documents has become simpler, thanks to pdfTEX and TEXworks. For the flowchart `diamondstroke` and `ovalstroke` have been written in analogy with PostScript's `rectstroke`, and of course they have been added to `PSlib.eps`. One can easily imagine packages where the elements can be dragged and dropped on a grid to build flowcharts. For simple flowcharts the PostScript operators will do.



16. GKP picture lost alas after 20 years, so I programmed it anew in PostScript.

Nest of squares. Blue book, P2 p131

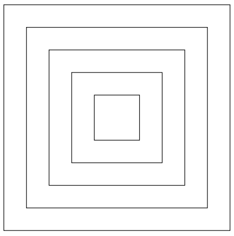
A triviality? Not so in PostScript, because the scaling of user space also scales the `linewidth`.[17] The example shows how to program à la PostScript with invariant `linewidth`. Below, I also included the old procedural way of programming a nest of squares, which circumvents the problem.

```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Nest of squares Adobe
%%BoundingBox: 80 335 525 529
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/nestofsquares% used from library: centersquare
{gsave 2.5 inch 6 inch translate 1 16 div setlinewidth
  1 1 5{ gsave  .5 mul inch dup scale
           centersquare stroke
         grestore} for
grestore
gsave
  6 inch 6 inch translate 1 setlinewidth
  /cmtx matrix currentmatrix def%store current matrix
  1 1 5{ gsave .5 mul inch dup scale
           centersquare
           cmtx setmatrix %reuse stored matrix
           stroke%invariant linewidth
         grestore} for
grestore}def
%%EndProlog
nestofsquares
showpage
%%EOF
```

Interesting use has been made of `currentmatrix` and `setmatrix`. A PostScript programming paradigm.

Variant nest of squares



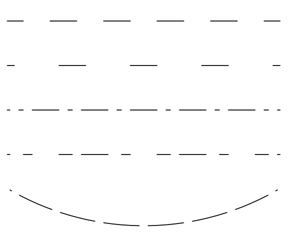
```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Nest of squares, variant
%%BoundingBox: -181 -181 181 181
%%BeginSetup
%%EndSetup
1 1 5{ 36 mul /s exch def /-s s neg def /2s s s add def
          -s -s 2s 2s rectstroke
       } for
showpage
%%EOF
```

Straight forward procedural approach, with `linewidth` invariant.

17. Sometimes very useful, for example in Fractal Arrow, Pythagoras trees, or the Lévy fractal... .

Centered Dash Patterns. Blue book P5 p143

In illustrations dashed lines occur frequently. Adobe provides nice variations.



```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Centered Dash Patterns. Blue book P5 p143
%%BoundingBox: 71 255 379 510
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/centerdashdict 9 dict def
/centerdash%uses pathlength from the library
  {centerdashdict begin /pattern exch def
    /pathlen pathlength def
    /patternlength 0 def
    pattern { patternlength add /patternlength exch def } forall
    pattern length 2 mod 0 ne
      { /patternlength patternlength 2 mul def } if
    /first pattern 0 get def
    /last patternlength first sub def
    /n pathlen last sub cvi patternlength idiv def
    /endpart pathlen patternlength n mul sub
       last sub 2 div def
    /offset first endpart sub def
    pattern offset setdash
  end } def
%%EndProlog
newpath 72 500 moveto 378 500 lineto   [30] centerdash stroke
newpath 72 450 moveto 378 450 lineto   [30 50] centerdash stroke
newpath 72 400 moveto 378 400 lineto   [30 10 5 10] centerdash stroke
newpath 72 350 moveto 378 350 lineto   [30 15 10] centerdash stroke
newpath 225 570 300 240 300 arc        [40 10] centerdash stroke
showpage
%%EOF
```

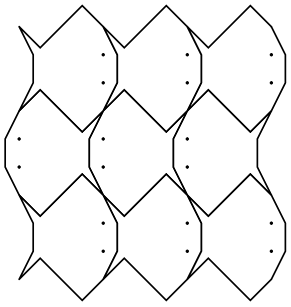
Interesting is the use of `flattenpath` in `pathlength` for calculating the length of a path, be it a Bézier cubic (or a circulare arc), or a line, respectively an arbitrary combination.

```
/pathlengthdict 7 dict def
/pathlength{pathlengthdict begin%Bluebook P5p143
   flattenpath /dist 0 def
      { /yfirst exch def /xfirst exch def
        /ymoveto yfirst def /xmoveto xfirst def }
      { /ynext exch def /xnext exch def
        /dist dist ynext yfirst sub dup mul
          xnext xfirst sub dup mul add sqrt add def
        /yfirst ynext def /xfirst xnext def }
      {}%path has been flattened, so no curveto or arc anymore
      { /ynext ymoveto def /xnext xmoveto def
        /dist dist ynext yfirst sub dup mul
          xnext xfirst sub dup mul add sqrt add def
        /yfirst ynext def /xfirst xnext def }
      pathforall
      dist
end} def
```

Escher fishes

The example has been included because a little deformed 3x3 tile can yield creative results.

```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Escherfishes, cgl 1997
%%BoundingBox: -85 -90 85 90
%%BeginSetup
%%EndSetup
%%BeginProlog
(C:\\PSlib\\PSlib.eps) run
/escherfishesdict 12 dict def
/escherfishes{escherfishesdict begin %tilLxxii
0 setlinejoin 1 setlinecap
/a 50 def          /ma a neg def
/ha .5  a mul def  /mha ha neg def
/qa .5 ha mul def  /mqa qa neg def
/d .3333 a mul def /dd .5 d mul def
/Courier findfont 25 scalefont setfont
/tile{gsave
  2{mha ha moveto
  mha dd add  ha d sub lineto
  mha dd add mha d add lineto
  mha mha lineto
  a 0 translate}repeat stroke
  grestore
  gsave
  2{ha ha moveto
    qa ha qa add lineto
    mqa qa lineto
    mha ha lineto stroke
    gsave ha  ha d sub translate
    0 0 moveto 0 0 1 0 360 arc fill
    grestore
    1 -1 scale}repeat
  grestore
}def
ma a a{/i exch def
ma a a{/j exch def
   gsave j i translate
         tile grestore
}for -1 1 scale}for
end}def
%%EndProlog
escherfishes showpage
%%EOF
```



Escher Alhambra tile

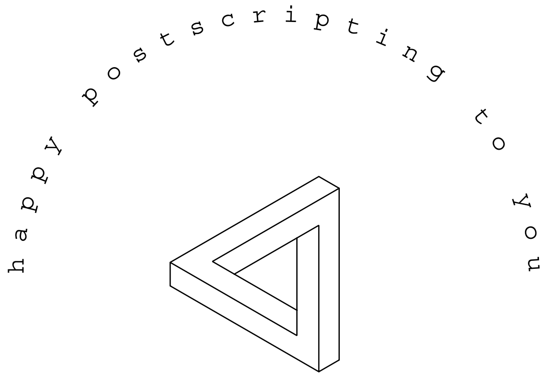
My old Metafont example has been included because it is a nice tile, and it has been processed by Troy Henderson's remote MP Previewer.

```
beginfig(0) size:=3; path p[]; picture pic[];
draw (-10size,5size)--(0,5size)--(0,-5size)--(10size,-5size);
draw (5size,10size)--(5size,0)--(-5size,0)--(-5size,-10size);
pic1:=currentpicture;
clearit;
%flower
p1= (origin--(3size,0)) shifted (size,0) rotated 22.5;
p2= point 1of p1{point 1 of p1 -origin}..{down}(5size,0);
p3= p2 reflectedabout(point 1 of p1, origin);
draw p1..p2;draw p3;
p4= (p1..p2)rotated 45;
p5= p3 rotated 45;
draw p4; draw p5;
draw (-5size,10size)--(-2.5size,10size)--(10size,-2.5size)--(10size,-10size);
addto currentpicture also currentpicture rotated90;
addto currentpicture also currentpicture rotated180;
draw fullcircle scaled 2size;
draw unitsquare scaled 10.5size shifted (-5.25size, -5.25size);
pic2:=currentpicture;
%
addto currentpicture also pic1 shifted (20size,0);
addto currentpicture also pic1 shifted (0,20size);
addto currentpicture also pic2 shifted (20size,20size);
addto currentpicture also currentpicture shifted (40size,0);
addto currentpicture also currentpicture shifted (0,40size);
pickup pencircle scaled 5;
draw unitsquare scaled 80size shifted (-10size,-10size);
endfig
end
```

Seal

A nice use of kshow.[18] Interesting is the Escher triangle. Once the rotation symmetry has been discovered the programming becomes a trifle. The text along an implicit path, via kshow, is also nice and a PostScript programming paradigm.

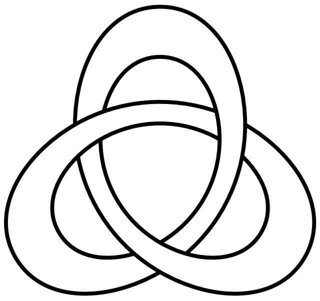
```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Seal
%%BoundingBox: -110 -60 110 110
%%Beginsetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/sealdict 3 dict def
/seal{sealdict begin
/Courier 10 selectfont /r 100 def
/text (happy postscripting to you) def
gsave 89.9 rotate
0 r moveto
{-7.04 rotate 0 r moveto} text kshow
grestore
.75 dup scale eschertriangle
end}def
seal showpage
%%EOF
```



18. What puzzles me is that rotation over 89.9 and 90 differ so much in appearance???

Escher doughnut

The doughnut was programmed in declarative Metafont. On occasion of EuroTEX2012 the code was processed by MetaPost and a PostScript version emerged by editing the PostScript code. How to program this doughnut from scratch in PostScript is not clear to me. What makes it hard are the hidden lines.

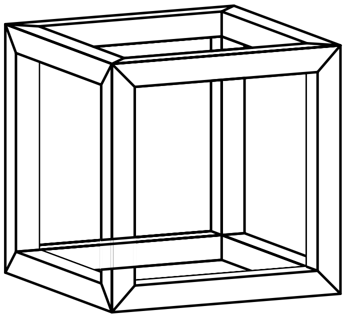


```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Escher doughnut
%%BoundingBox: -40 -32 40 45
%%Creator: MetaPost and JJW, CGL. Doughnut
%%BeginSetup
%%EndSetUp
3{-21.6507 12.5 moveto
-21.6507 27.74551 -13.78212 42.5003
    0 42.5003 curveto
13.78212 42.5003 21.6507 27.74551
   21.6507 12.5 curveto
21.6507 -0.24506 16.04897 -12.19249
   6.58395 -20.3313 curveto
%
-14.3152 15.86746 moveto
-12.43301 23.58928 -7.45113 29.75021
   0 29.75021 curveto
9.64748 29.75021 15.15549 19.42186
   15.15549 8.75 curveto
15.15549 -2.07904 9.37823 -12.08548
   0 -17.5 curveto
120 rotate
}repeat
stroke showpage
%%EOF
```

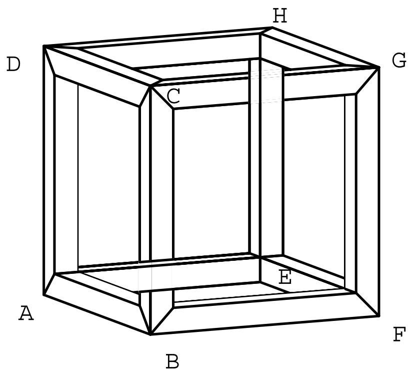
Postprocessed by Photoshop.

Escher’s impossible cubes

The example has been included because I have not seen Escher impossible cubes programmed.



```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Imp Cube in projection. cgl 2009
%%BoundingBox: -20 -30 150 120
%%Beginsetup
%%EndSetup
(C:\\PSlib\\PSlib.eps) run
/ptp{/z exch def/y exch def/x exch def
   x neg a cos mul y a sin mul add
   x neg a sin mul b sin mul y neg a cos mul b sin mul add
     z b cos mul add}def
/a 25 def /b 10 def%projection parameters
/r 100 def /mr r neg def /hr r 2 div def%size
/d 10 def  /md d neg def  /rmd r d sub def  /dmr rmd neg def  %d<<r
1 setlinewidth 1 setlinejoin /Courier 10 selectfont
/a1 {0 0 0 ptp} def
/a2 {0 d 0 ptp} def
/a3 {0 d d ptp} def
/a4 {0 0 d ptp} def
/a5 {md 0 0 ptp} def
/a6 {md d 0 ptp} def
/a7 {md d d ptp} def
/a8 {md 0 d ptp} def
%a1 moveto -10 -10 rmoveto (A) show
/b1 {0  rmd  0 ptp} def
/b2 {0  r    0 ptp} def
/b3 {0  r    d ptp} def
/b4 {0  rmd  d ptp} def
/b5 {md rmd  0 ptp} def
/b6 {md r    0 ptp} def
/b7 {md r    d ptp} def
/b8 {md rmd  d ptp} def
%b1 moveto 10 -15 rmoveto (B) show
/c1 {0  rmd  rmd ptp} def
/c2 {0  r    rmd ptp} def
/c3 {0  r    r   ptp} def
/c4 {0  rmd  r   ptp} def
/c5 {md rmd  rmd ptp} def
/c6 {md r    rmd ptp} def
/c7 {md r    r   ptp} def
/c8 {md rmd  r   ptp} def
%c2 moveto 6 3 rmoveto (C) show
/d1 {0  0 rmd ptp} def
/d2 {0  d rmd ptp} def
/d3 {0  d r   ptp} def
/d4 {0  0 r   ptp} def
/d5 {md 0 rmd ptp} def
/d6 {md d rmd ptp} def
/d7 {md d r   ptp} def
/d8 {md 0 r   ptp} def
```

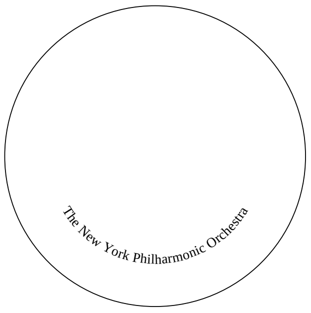


```
%d1 moveto -15 0 rmoveto (D) show
/e1 {dmr 0 0 ptp} def
/e2 {dmr d 0 ptp} def
/e3 {dmr d d ptp} def
/e4 {dmr 0 d ptp} def
/e5 {mr 0 0 ptp} def
/e6 {mr d 0 ptp} def
/e7 {mr d d ptp} def
/e8 {mr 0 d ptp} def
%e5 moveto 2 -3 rmoveto (E) show
/f1 {dmr rmd  0 ptp} def
/f2 {dmr r    0 ptp} def
/f3 {dmr r    d ptp} def
/f4 {dmr rmd  d ptp} def
/f5 {mr  rmd  0 ptp} def
/f6 {mr  r    0 ptp} def
/f7 {mr  r    d ptp} def
/f8 {mr  rmd  d ptp} def
%f6 moveto 5 -10 rmoveto (F) show
/g1 {dmr rmd  rmd ptp} def
/g2 {dmr r    rmd ptp} def
/g3 {dmr r    r   ptp} def
/g4 {dmr rmd  r   ptp} def
/g5 {mr  rmd  rmd ptp} def
/g6 {mr  r    rmd ptp} def
/g7 {mr  r    r   ptp} def
/g8 {mr  rmd  r   ptp} def
%g7 moveto 5  0 rmoveto (G) show
/h1 {dmr 0 rmd ptp} def
/h2 {dmr d rmd ptp} def
/h3 {dmr d r   ptp} def
/h4 {dmr 0 r   ptp} def
/h5 {mr 0 rmd ptp} def
/h6 {mr d rmd ptp} def
/h7 {mr d r   ptp} def
/h8 {mr 0 r   ptp} def
%h8 moveto 0 2 rmoveto (H) show
%AEHD
/AEHDo {a1 moveto e5 lineto h8 lineto d4 lineto closepath}def
/AEHDoi{a8 moveto e4 lineto h1 lineto d5 lineto closepath}def
/AEHDii{a7 moveto e3 lineto h2 lineto d6 lineto closepath}def
%EFGH
/EFGHo {e5 moveto f6 lineto g7 lineto h8 lineto closepath}def
/EFGHoi{e7 moveto f8 lineto g1 lineto h6 lineto closepath}def
/EFGHii{e3 moveto f4 lineto g1 lineto h2 lineto closepath}def
%ABCD
/ABCDo{a1 moveto b2 lineto c3 lineto d4 lineto closepath}def
/ABCDoi{a3 moveto b4 lineto c1 lineto d2 lineto closepath}def
%BFGC
/BFGCo{b2 moveto f6 lineto g7 lineto c3 lineto closepath}def
/BFGCoi{b7 moveto f3 lineto g2 lineto c6 lineto closepath}def
%CGHD
/CGHDii{c3 moveto g7 lineto h8 lineto d4 lineto closepath}def
/CGHDoi{c8 moveto g4 lineto h3 lineto d7 lineto closepath}def
/CGHDo {c3 moveto g7 lineto h8 lineto d4 lineto closepath}def
```

```
%ABFE
/ABFE0i{a6 moveto b5 lineto f1 lineto e2 lineto closepath}def
/ABFEii{a7 moveto b8 lineto f4 lineto e3 lineto closepath}def
%Snijpunten ivm clipping window
/uc1{h2 d6 d7 c8 intersect}def
/uc2{h2 g1 c8 g4 intersect}def
/uc3{c6 g2 g1 f4 intersect}def
/uc4{f4 b8 b7 c6 intersect}def
/uc5{d6 a7 d2 c1 intersect}def
/uc6{a7 b8 b4 c1 intersect}def
%3 windows as clipping path
/windows{newpath %
a7 moveto uc6 lineto c1 lineto uc5 lineto closepath %ABCDoi
c6 moveto uc3 lineto f4 lineto uc4 lineto closepath %BFGCoi
c8 moveto uc1 lineto h2 lineto uc2 lineto closepath %CGHDoi
}def
%Draw Front and top
ABCDo ABCDoi BFGCo BFGCoi CGHDo CGHDoi stroke
%upper what is in sight
h8 moveto h3 lineto h2 lineto
d2 moveto d4 lineto d7 lineto
g4 moveto g7 lineto g2 lineto
c1 moveto c3 lineto c8 lineto
c3 moveto c6 lineto
h2 moveto uc1 lineto
h2 moveto uc2 lineto stroke
% einde achterbovenkant in zicht, lower corners
a1 moveto a3 lineto a7 lineto
b4 moveto b2 lineto b7 lineto
e4 moveto e3 lineto e2 lineto
e3 moveto e7 lineto
f4 moveto f3 lineto f6 lineto stroke
```

Adobe's Blue book CD label, part1

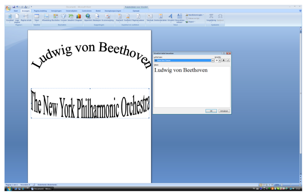
The PostScript def has been published in Adobe's Blue book. The `insidecircletext def` demonstrates how to process a string character by character, a PostScript programming paradigm.



Insidecircletext prints the text in a counter-clockwise fashion with its baseline along the circumference, on the inside of a circle.

```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Blue book CD Label Program 10 p167
%%BoundingBox: -166 -166 166 166
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\Bluebook.eps) run
/circtextdict 19 dict def
circtextdict begin /pi 3.1415923 def
/findhalfangle{ stringwidth pop 2 div
         2 xradius mul pi mul div 360 mul} def
/outsideplacechar{ /char exch def
       /halfangle char findhalfangle def
       gsave halfangle neg rotate
         radius 0 translate
         -90 rotate
         char stringwidth pop 2 div neg 0 moveto
         char show
       grestore
       halfangle 2 mul neg rotate} def
/insideplacechar{ /char exch def
       /halfangle char findhalfangle def
       gsave  halfangle rotate
         radius 0 translate 90 rotate
         char stringwidth pop 2 div neg 0 moveto
         char show
       grestore
       halfangle 2 mul rotate} def
end%circletextdict

/insidecircletext{%parameters: text pointsize centerangle radius
circtextdict begin  /radius exch def/centerangle exch def
          /ptsize exch def/str exch def
         /xradius radius ptsize 3 div sub def
         gsave centerangle str findhalfangle sub rotate
           str{ /charcode exch def
                  ( ) dup 0 charcode put insideplacechar
              } forall
         grestore
end} def
%%EndProlog
/Times-Roman 15 selectfont (The New York Philharmonic Orchestra) 15 270 118
insidecircletext showpage
%%EOF
```

Circular Text by Photoshop

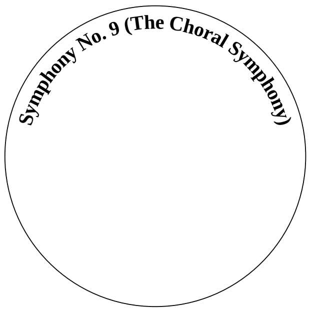


Circular Text by Word

Blue book

Adobe’s Blue book CD label, part2

The PostScript def has been published in Adobe’s Blue book. The `outsidecircletext def` demonstrates how to process a string character by character, a PostScript programming paradigm.

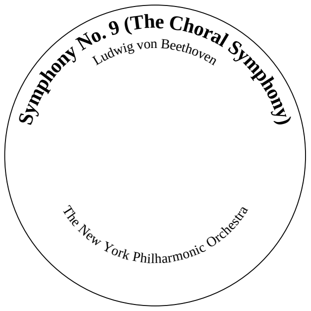


Outsidecircletext prints the text in a clockwise fashion with its baseline along the circumference, on the outside of a circle.

```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Blue  book, Program 10 p167
%%BoundingBox: -166 -166 166 166
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\Bluebook.eps) run
/circtextdict 19 dict def
circtextdict begin/pi 3.1415923 def
/findhalfangle{ stringwidth pop 2 div
         2 xradius mul pi mul div 360 mul} def
/outsideplacechar{ /char exch def
       /halfangle char findhalfangle def
       gsave halfangle neg rotate
         radius 0 translate -90 rotate
         char stringwidth pop 2 div neg 0 moveto
         char show
       grestore
       halfangle 2 mul neg rotate} def
/insideplacechar
{ /char exch def
       /halfangle char findhalfangle def
       gsave halfangle rotate
         radius 0 translate
         90 rotate
         char stringwidth pop 2 div neg 0 moveto
         char show
       grestore
       halfangle 2 mul rotate} def
end%circtextdict

/outsidecircletext%parameters: text pointsize centerangle radius
 {circtextdict begin
   /radius exch def /centerangle exch def /ptsize exch def /str exch def
         /xradius radius ptsize 4 div add def
         gsave centerangle str findhalfangle add rotate
           str{ /charcode exch def
                  ( ) dup 0 charcode put outsideplacechar
              } forall
         grestore
       end} def
%%EndProlog
0 0 165 0 360 arc stroke
/Times-Bold 22 selectfont (Symphony No. 9 (The Choral Symphony)) 22 90 140
outsidecircletext showpage
%%EOF
```

Adobe's Blue book CD label, complete

The PostScript def has been published in Adobe's Blue book.


Symphony No. 9 (The Choral Symphony)
Ludwig von Beethoven
The New York Philharmonic Orchestra

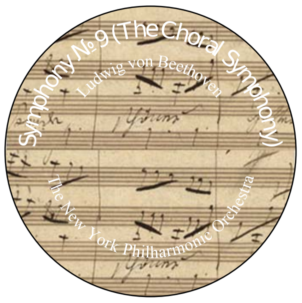
```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Blue Book CD Label Program 10 p167
%%BoundingBox: -166 -166 166 166
%%BeginProlog
(C:\\PSlib\\Bluebook.eps) run
/circtextdict 19 dict def
circtextdict begin /pi 3.1415923 def
/findhalfangle{ stringwidth pop 2 div
         2 xradius mul pi mul div 360 mul} def
/outsideplacechar{ /char exch def
       /halfangle char findhalfangle def
       gsave halfangle neg rotate
         radius 0 translate -90 rotate
         char stringwidth pop 2 div neg 0 moveto
         char show
       grestore halfangle 2 mul neg rotate} def
/insideplacechar{ /char exch def
       /halfangle char findhalfangle def
       gsave  halfangle rotate
         radius 0 translate 90 rotate
         char stringwidth pop 2 div neg 0 moveto
         char show
       grestore halfangle 2 mul rotate} def
end%circtextdict
/outsidecircletext{ circtextdict begin
        /radius exch def /centerangle exch def /ptsize exch def /str exch def
         /xradius radius ptsize 4 div add def
         gsave centerangle str findhalfangle add rotate
           str{ /charcode exch def
                  ( ) dup 0 charcode put outsideplacechar
                } forall
         grestore end} def
/insidecircletext{ circtextdict begin
        /radius exch def /centerangle exch def /ptsize exch def /str exch def
         /xradius radius ptsize 3 div sub def
         gsave centerangle str findhalfangle sub rotate
           str{ /charcode exch def
                  ( ) dup 0 charcode put insideplacechar
                } forall
         grestore end} def
 %%EndProlog
/cddvdlabel{0 0 165 0 360 arc stroke
/Times-Bold 22 selectfont (Symphony No. 9 (The Choral Symphony)) 22 90 140
outsidecircletext
/Times-Roman 15 selectfont
   (Ludwig von Beethoven) 15 90 118 outsidecircletext
   (The New York Philharmonic Orchestra) 15 270 118 insidecircletext
}def
 /EndProlog
cddvdlabel
%%EOF
```

Blue book

Adobe's Blue book CD label, enriched by background notes

The PostScript def has been published in *CD and DVD labels.* A simplified version has been incorporated. `partituur9e.eps` had to be placed in the c-directory!?!

```
%!PS-Adobe-3.0
%%Title: Blue Book label with score illustration added
%%Creator: Kees van der Laan, kisa1@xs4all.nl, 2011
%%BoundingBox: -175 -175 175 175
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/toplabel{/mm {72 25.4 div mul} def
0 0 60 mm 0 360 arc
20 mm 0 moveto
0 0 20 mm 0 360 arc
0 -20 mm moveto 0 20 mm lineto
-20 mm 0 moveto 20 mm 0 lineto
}def%toplabel
%
%---Program--- the script
%
3 setlinewidth toplabel gsave stroke grestore eoclip
gsave
-170 -170 translate
(c:\\partituur9e.eps) run %put the partituur clipped to the label
grestore
%annotations as prompted by Program 10 of Adobe's Blue Book
1 setgray     %annotations in white
/size 27 def
/Times-Roman size selectfont
(Symphony No. 9 (The Choral Symphony)) size  90 135 outsidecircletext
/size 20 def
/Times-Roman size selectfont
(Ludwig von Beethoven)                 size  90 118 outsidecircletext
(The New York Philharmonic Orchestra)  size 270 118 insidecircletext
showpage
%%EOF
```


Symphony № 9 (The Choral Symphony)
Ludwig von Beethoven
The New York Philharmonic Orchestra

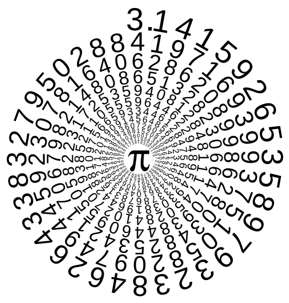
In MS Word and Nero (Version 10 with LightScribe) one can create labels interactively. LightScribe, a HP technique to burn labels on special discs as well, allows only for black and white labels.

My wife, not at all a TEXie, designs labels in Photoshop and prints them on prefab glued paper by the tool CDFACE1.6 (Media labeling software templates for: CDs, DVDs, jewel cases, envelopes, floppy discs, audio cassettes, dat tapes, zip discs). The special tool CDFACE can handle all, no PhotoShop is needed.

Willi Egger has shown, MAPS 2009, how to use ConTEXt for the purpose.

Pi decimals

Released as BachoTEX2012 Programming Pearl. The spiral path is implicit!

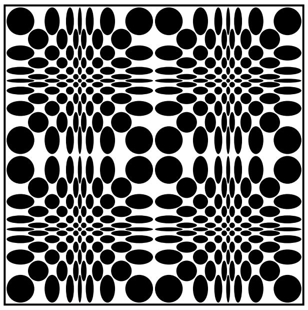


```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Pi-decimals along a  Spiral cgl 2010, 2012
%%BoundingBox: -80 -100 100 90
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
%%Endprolog
/Symbol 26 selectfont
1 -18 moveto (p) show%p denotes pi in the symbol font
/Helvetica 20 selectfont
0 70 moveto (3) show 1 0 rmoveto (.) show -2 0 rmoveto
-10 rotate .995 dup scale
%
{pop pop -10 rotate 3 0 rmoveto .995 dup scale}
(3.141592653589793238462643383279502884197169399375\
1058209749445923078164062862089986280348253421170\
6798214808651328230664709384460955058223172535940\
8128481117450284102701938521105559644622948954930\
3819644288109756659334461284756482337867831652712\
0190914564856692346034861045432664821339360726024\
9141273724587006606315588174881520920962829254091\
7153643678925903600113305305488204665213841469519\
4151160943305727036575959195309218611738193261179...) kshow
showpage
%%EOF
```

Vo in his *PSTricks — Graphics and PostScript for TEX and LATEX, p294* shows a variant π-decimals in PSTricks in LATEX.

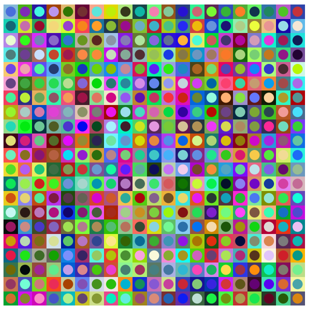
The original picture from the CWI calendar of 1972

Schrofer’s Op Art



```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Schrofer’s OpArt
%%BoundingBox: -195 -195 570 570
%%BeginSetup
%%endSetup
%%BeginProlog
/s 5 def %BB for 1 with s=5
/drawgc{gsave
   r c translate
   r abs 5 div s add
   c abs 5 div s add scale
   0 1 moveto
   0 0 1 0 360 arc
   fill
grestore}def
/schrofer{/flipflop true def
/indices [30 21 14 9 5 2 0 -2 -5 -9 -14 -21 -30] def
indices{/r exch s mul def
   gsave
   indices{/c exch s mul def
      flipflop{drawgc}if
      /flipflop flipflop not def}forall
   grestore
   }forall
closepath 5 setlinewidth stroke
}def
%%EndProlog
gsave
2{gsave
  2{schrofer
    375 0 translate}repeat
  grestore
  0 375 translate
 }repeat
grestore
5 setlinewidth -190 dup 755 dup rectstroke%border
%%EOF
```

Vasarely's Op Art: random coloured squares with circles



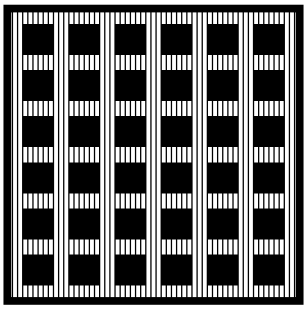
```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Vasarely random coloured Squares. CGL 2007
%%BoundingBox: -211 -211 211 211
%%BeginProlog
/vasarelyrandomdict 6 dict def
vasarelyrandomdict begin
/r 20 def              %side square
/R r 10 mul def        %bound loops
/r2 r 2 div def        %r/2
/ri r2 2 sqrt div def %radius inner circle
/s 2 31 exp 1 sub def %scaling random  numbers, reus
/square
{rand s div rand s div rand s div setrgbcolor
 r2 neg dup r dup rectfill

 rand s div rand s div rand s div setrgbcolor
 0 0 ri 0 360 arc fill} def
end%dict
/vasarelyrandom{vasarelyrandomdict begin 1951 srand
R neg r R{/i exch def
R neg r R{/j exch def
          gsave i j translate square grestore
         }for
         }for
end}def
%%EndProlog
vasarelyrandom showpage
%%EOF
```

In Metafont the following Vasarely's I programmed 20 years ago. Interesting is the use of interpath, similar as used in the heart figure in the Metafont book p134. interpath is not available in PostScript, because we don't have explicit paths. Below I have imitated interpath functionality in PostScript for this simple example. No path data structure nor picture datastructure. Just transformation of User Space. Compare the code with the MetaPost code given in Recreational use of TEX&Co. Simpler to read isn't it? The code of the right picture resembles much the code of Schrofer and is supplied in PSlib.eps under the name vasarelybloks.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Vasarely lines. CGL 2014
%%BoundingBox: -221 -211 221 211
/pattern{newpath
/h 210 def /k h 3 div def /d 30 def
0 10 200{/x exch def
  x d add 0 moveto x d add k  x h k sub %control points
  x h curveto /d d 1 sub def
  }for
stroke}def
2{2{pattern 1 -1 scale pattern -1 1 scale
   }repeat 90 rotate}repeat
2 setlinewidth -210 -210 420 420 rectstroke showpage
%%EOF
```

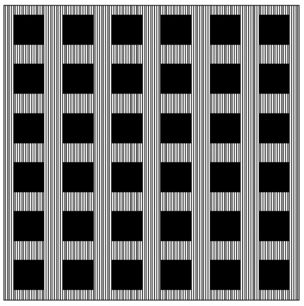
Soto’s Op Art

```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Soto, cgl 2012
%%BoundingBox: -1 -1 58 58
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/soto{%cgl 2012
gsave .25 setlinewidth
57{1 0 translate 0 0 moveto 0 57 lineto}repeat stroke
grestore
gsave 1.5 setlinewidth 0 0 57 57 rectstroke grestore
3 3 translate
6{gsave
  6{0 0 6 6 rectfill 9 0 translate}repeat
  grestore
  0 9 translate
 }repeat
}def
%%EndProlog
soto showpage
%%EOF
```



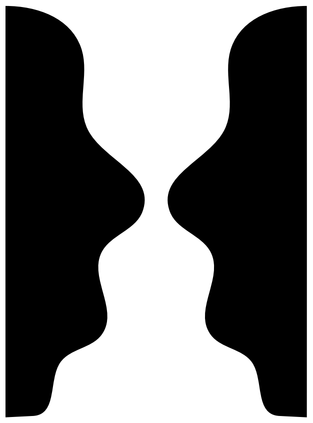
Earlier in TEX alone

```
%cgl feb 2010 tst Soto
\def\boxit#1{\vbox{\hrule\hbox{\vrule#1\vrule}\hrule}}
\newbox\cb  \newdimen\ul \ul=6ex \newdimen\size \size12\ul
\setbox\cb\vbox to2\ul{\vss%colored box with transparent boundary
   \hbox to2\ul{\hss\vrule height1.2\ul width1.2\ul \hss}%
        \vss}%

$$\boxit{\vbox{\offinterlineskip
\hbox{\xleaders\hbox to.5ex{\hss\vrule height\size\hss}\hskip\size}%
\kern-\size%setback
   \leaders\hbox{\leaders\copy\cb\hskip\size}\vskip\size}}$$
\bye
```



Candle or faces

PostScript lacks to draw a pleasing curve through given points. The controle points must be specified.



```
%!PS-Adobe-3.0  EPSF-3.0
%%BoundingBox: -234 -317 234 317
%%BeginSetup
%%EndSetup
newpath -231 -315 moveto
-230.996576 -315 -189.003419 -312.900171 -189 -312.9 curveto
-150.602108 -310.979948 -165.354069 -257.301583 -147 -231 curveto
-132.060075 -209.590923 -99.648395 -209.766718 -84 -189 curveto
-55.898607 -151.707122 -103.284993 -102.741264 -84 -63 curveto
-70.625453 -35.4386 -31.836956 -29.048744 -21 0 curveto
0.145155 56.680143 -81.874447 78.741461 -105 126 curveto
-124.104631 165.04153 -100.718717 211.542235 -115.5 252 curveto
-131.812491 296.648828 -181.870276 315 -231 315 curveto
-231.102539 315 -231.102539 -315 -231 -315 curveto closepath fill
newpath 231 -315 moveto
230.996576 -315 189.003419 -312.900171 189 -312.9 curveto
150.602108 -310.979948 165.354069 -257.301583 147 -231 curveto
132.060075 -209.590923 99.648395 -209.766718 84 -189 curveto
55.898607 -151.707122 103.284993 -102.741264 84 -63 curveto
70.625453 -35.4386 31.836956 -29.048744 21 0 curveto
-0.145155 56.680143 81.874447 78.741461 105 126 curveto
124.104631 165.04153 100.718717 211.542235 115.5 252 curveto
131.812491 296.648828 181.870276 315 231 315 curveto
231.102539 315 231.102539 -315 231 -315 curveto closepath fill
2 setlinewidth -232 -316 454 632 rectstroke
showpage
%%EOF
```

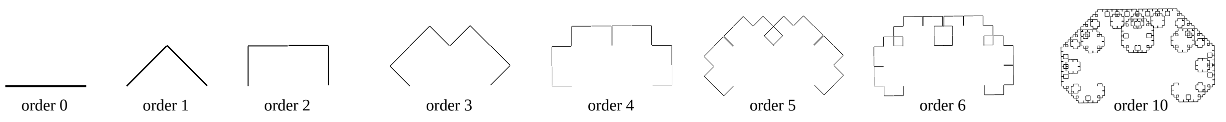
MetaPost code adapted from the old Metafont code.

```
beginfig(0);
size:=210; path p[];
p1=(-1.1size, -1.5size){right}---(-.9size,-1.49size)..(-.7size,-1.1size)..
  (-.4size,-.9size)..(-.4size, -.3size)..(-.1size,0)..(-.5size,.6size)..
  (-.55size,1.2size)...{left(-1.1size,1.5size)---cycle;
p2=p1 reflectedabout((0,-size), (0,size));
fill p1; fill p2;
draw (-1.2size, -1.6size)---(1.2size,-1.6size)---(1.2size,1.6size)---
  (-1.2size,1.6size)---cycle;
endfig;
```

The .eps code was obtained from MetaPost, with the frame added in PostScript.

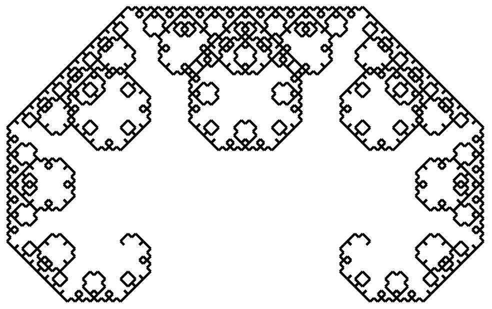
Lévy fractal

An approximation of the Lévy fractal is also called a C (broken) line of a certain order. The constructive definition of various orders of C lines starts with a straight line, let us call this line C_0. An isosceles triangle with angles 45°, 90° and 45° is built on this line as hypotenuse. The original line is then replaced by the other two sides of this triangle to obtain C_1. Next, the two new lines each form the base for another right-angled isosceles triangle, and are replaced by the other two sides of their respective triangle, to obtain C_2. After two steps, the broken line has taken the appearance of three sides of a rectangle of twice the length of the original line. At each subsequent stage, each segment in the C figure is replaced by the other two sides of a right-angled isosceles triangle built on it. Such a rewriting relates to a Lindenmayer system. After n stages the C line has length $2^{n/2}\times$ C_0: 2^n segments each of size $2^{-n/2}\times$ C_0.



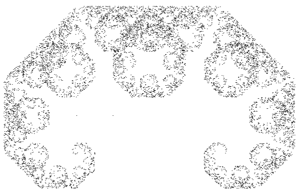
Production rule
$C_n = [R_{45} S_{(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}})} C_{n-1}] \oplus T_{\frac{s}{2}\frac{s}{2}} [R_{-45} S_{(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}})} C_{n-1}]$, with $C_0 =$ initial line, and C_n the Lévy C curve of order n, $\oplus$ splice operator, meaning add properly, i.e. $T_{(\frac{s}{2},\frac{s}{2})}$, [means store graphics state on the GS stack and open a new one,] means remove current graphics state off the GS stack and recall previous, R_{45} means rotate US 45° in the PS sense $S_{a,b}$ means scale US by a and b, in x- and y-direction $T_{a,b}$ means translate US by a and b, in x- and y-direction.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Levy fractals 0..4. cgl, 2011, kisa1@xs4all.nl
%%BoundingBox: -1 -1 363 65
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/levyC{%on stack: the order ==> C line
       %s = size of line segment (global)
dup 0 eq
{0 0 moveto s 0 lineto currentpoint stroke translate}%draw line
{1 sub %lower order on stack
  45 rotate levyC%-45 rotate%combine -45 twice into -90
 -90 rotate levyC  45 rotate
 1 add %adjust order on stack
}ifelse
}def
%%EndProlog
/s 20 def             0 levyC pop
s 2 div   0 translate 1 levyC pop
s         0 translate 2 levyC pop
%%EOF
```



Lévy fractal as Iterated Function System

$$\begin{pmatrix}x'\\y'\end{pmatrix} \overset{\mathrm{L}}{=} .5\begin{pmatrix}1&-1\\1&1\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}+.5\begin{pmatrix}-1\\1\end{pmatrix} \text{ and } \begin{pmatrix}x'\\y'\end{pmatrix} \overset{\mathrm{R}}{=} .5\begin{pmatrix}1&1\\-1&1\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}+.5\begin{pmatrix}1\\1\end{pmatrix}.$$

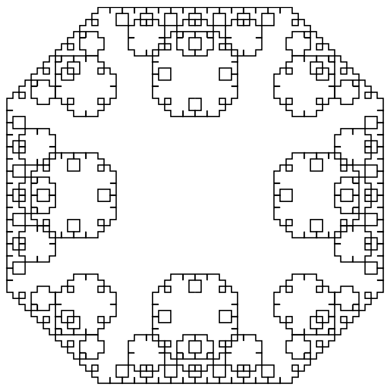


Associated with the Lévy fractal are 2 rotations with rotation centres for L: (-1,0) and for R: (1,0) and contraction $\sqrt{.5} \approx .7$. Amazing, isn't it! Laurier's BASIC program FRACMC2 and my conversion are given below.

```
REM ***naam: FRACMC2***
KMAX=60000
REM ***Coefficienten***
   A=.5: B=A: C=A : D=-A
   DET1=A*A+B*B : DET2=C*C+D*D
   Q=DET1/(DET1+DET2)
   X=1 : Y=0 : K=0 : KMAX=10000
   DO WHILE K<KMAX AND INKEYS$=""
      R=RND
      IF R<Q THEN
         U=A*X-B*Y-1+A :
           V=B*X+A*Y+B 'rotatie L
      ELSE
         U=C*X-D*Y-1-C :
           V=D*X+C*Y-D 'rotatie R
      ENDIF
      X=U : Y=V
      PSET (X,Y)
   LOOP : BEEP
END
```

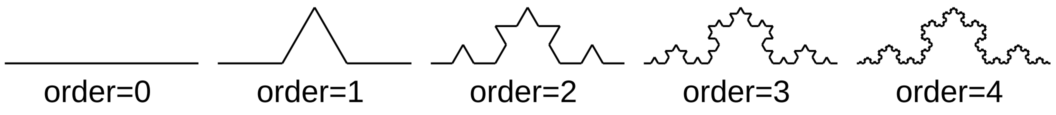
```
%!PS-Adobe-3.0 EPSF-3.0
%%Author: H.A. Lauwerier(1994): Spelen met Graphics en Fractals
%%BoundingBox: -203 -54 205 206
%%BeginSetup
%%EndSetup
%%BeginProlog
/Courier 7 selectfont
/x 0 def  /y 0 def /halfmaxint 2 30 exp  def
/a .5 def  /b a def /c a def /d a neg def
/det1 a dup mul b dup mul add def /det2 c dup mul d dup mul add def
/q det1 det1 det2 add div def
/printxy{x s  y s moveto (.) show}def
%%EndProlog
/s {100 mul}def 10 srand%10 is seed
10000{rand halfmaxint lt
      {/xnew a x mul b y mul sub 1 sub a add def
       /y b x mul a y mul add b add def           /x xnew def%rot L
      }{/xnew c x mul d y mul sub 1 add c sub def
        /y    d x mul c y mul add       d sub def /x xnew def%rot R
      }ifelse
      printxy}repeat
showpage
%%EOF
```

Łévy carpet. C_{10} spliced with its `-1 1 scaled` copy, a Lévy carpet.

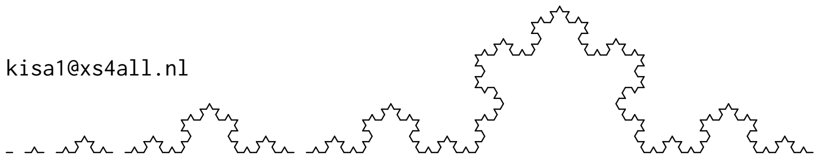


von Koch fractal

A von Koch broken line and a Lévy C line are related to a Lindenmayer system, also called a rewrite system. For the von Koch broken line the rewrite is: divide a line in 3 pieces and replace the middle piece by an equilateral triangle, with the base omitted. Repeat the process on the 4 line pieces to the required order. It is similar to the defining construction process of the Lévy fractal; the result conveys a different impression, however. Below $K_0 ... K_4$, scaled with increasing order (line thickness is scaled as well).



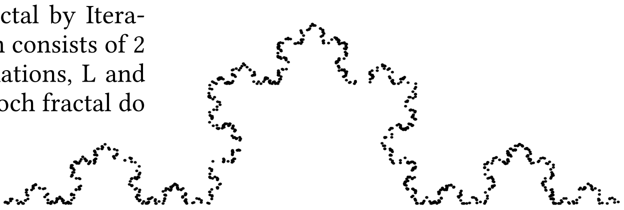
Lindemayer production rule enriched with PostScript concepts for the von Koch fractal
$K_n = [S_{\frac{1}{3}\frac{1}{3}} K_{n-1}] \oplus T_{\frac{s}{3}0}\, [S_{\frac{1}{3}\frac{1}{3}} R_{60} K_{n-1}] \oplus T_{\frac{s}{6}\,\frac{s\sqrt{3}}{6}}\, [S_{\frac{1}{3}\frac{1}{3}} R_{-60} K_{n-1}] \oplus T_{\frac{s}{6}\,\frac{-s\sqrt{3}}{6}}\, [S_{\frac{1}{3}\frac{1}{3}} K_{n-1}]$
with, K_0 the initial line, K_n the von Koch curve of order n, $\oplus$ splice operator, meaning add properly, i.e. translate, [open a new GS on the GS stack,] remove current graphics state from the GS stack and recall previous, R_{60} means rotate US 60° in the PS sense, $S_{a,b}$ means scale US by a and b, in x- and y-direction $T_{a,b}$ means translate US by a and b, in x- and y-direction.



```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: vonKoch fractal. cgl, kisa1@xs4all.nl
%%BoundingBox: -1 -1 650 120
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/vonKoch{%on stack order >=0; ==> von Koch
         %s = size of the line segment (global)
dup 0 eq
{0 0 moveto s 0 lineto currentpoint stroke translate}
{1 sub vonKoch %lower the order on the stack
   60 rotate vonKoch
 -120 rotate vonKoch
   60 rotate vonKoch
 1 add         %reset order
}ifelse}def
%%EndProlog
/s 5 def
0 vonKoch 10 0 translate
1 vonKoch 10 0 translate
2 vonKoch 10 0 translate
3 vonKoch 10 0 translate
4 vonKoch showpage
%%EOF
```

Von Koch-like fractal as Iterated Function System

Lauwerier(1994) in one of his exercises created a von Koch-like fractal by Iterated Function Systems, which consists of 2 contracted, affine transformations, L and R, which both for the von Koch fractal do contraction and mirroring.



$$\begin{pmatrix} x' \\ y' \end{pmatrix} \overset{\text{L}}{=} \begin{pmatrix} .5 & .289 \\ .289 & -.5 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} -.5 \\ .289 \end{pmatrix} \text{ and } \begin{pmatrix} x' \\ y' \end{pmatrix} \overset{\text{R}}{=} \begin{pmatrix} .5 & -.289 \\ -.289 & -.5 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} .5 \\ .289 \end{pmatrix}$$

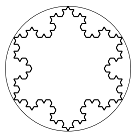
Associated with the von Koch fractal are 2 rotations with mirroring with centres for L: (-1,0) and for R: (1,0) and contraction $\sqrt{.5^2 + .289^2} \approx .58$. Amazing, isn't it! Laurier's BASIC program FRACMC4 and my transcription are given below. (MC is abbreviation for Monte Carlo, meaning alternate L and R by gambling.)

```
REM ***iteratief systeem,
REM    2 spiegelingen, FRACMC4***
REM ***coefficienten***
A=.5 : B=..289 : C=A : D=-B
DET1=A*A+B*B : DET2=C*C+D*D :
  Q=DET1/(DET1+DET2)
X=1 : Y=0 : K=0 : KMAX=1000
DO WHILE K<KMAX AND INKEY$=" "
   R=RND
   IF R<Q THEN
      X1=A*X+B*Y-1+A :
        Y1=B*X-A*Y+B 'spiegeling L
   ELSE
      X1=C*X+D*Y+1-C :
        Y1=D*X-C*Y-D 'spiegeling R
   END IF
   X=X1 : Y=Y1
   PSET (X,Y),10
   K=K+1
LOOP : BEEP
END
```

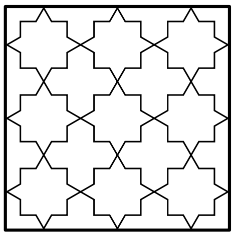
```
Other values of the parameters
a=.5 b=.5 c=.6667 d=0 %bebladerde tak
a=.5 b=.289 c=.5  d=-.289 %von Koch
a=.5 b=.5   c=.5  d=0     %kale tak
a=.5 b=.5   c=.6  d=-.2
a=0  b=.64  c=0   d=-.64  %tegelpatroon
```

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: fracmc4
%%Author: H.A. Lauwerier(1994): Spelen met Graphics en Fractals
%%BoundingBox: -100 -1 103 60
%%BeginSetup
%%EndSetup
%%BeginProlog
/Courier 7 selectfont
/x 1 def /y 0 def /halfmaxint 2 30 exp def/maxint 2 31 exp 1
sub
def
/a .5 def  /b .289 def /c a def /d b neg def
/det1 a dup mul b dup mul add def /det2 c dup mul d dup mul add
def
/q det1 det1 det2 add div def
/printxy {x s y s moveto (.) show}def
%%EndProlog
10 srand%10 is seed
/s {100 mul }def%scaling
1000{rand maxint div q lt
     {/xnew a x mul b y mul add 1 sub a add def
       /y b x mul a y mul sub b add def /x xnew def%mirror L
     }
     {/xnew c x mul d y mul add 1 add c sub def
      /y d x mul c y mul sub d sub def /x xnew def%mirror R
     }ifelse
     printxy
 }repeat
showpage
%%EOF
```

A von Koch island is a closed splicing of von Koch fractals; at right a von Koch tile (van der Laan(1997)).



```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: von Koch (triangular) island
%%...
/s 100 def
gsave .5 s mul dup neg exch translate 3 vonKochfractal pop
grestore
gsave .5 s mul dup translate
      -120 rotate 3 vonKochfractal pop grestore
gsave 0  -.366 s mul translate
      -240 rotate 3 vonKochfractal pop grestore
.001 setlinewidth 0 21 57.8 0 360 arc stroke
showpage
%%EOF
```

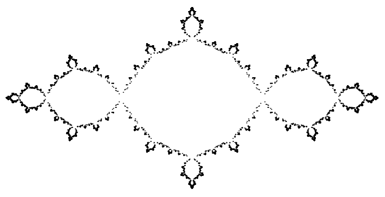


Julia fractals

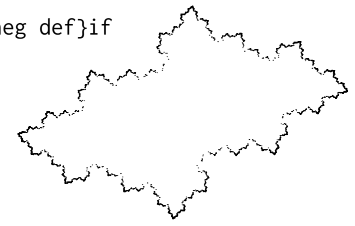
The Julia set for $z_i = z_{i-1}^2 + c, \quad i = 1, 2, 3, ...$ is a repeller; the Julia set for the inverse iteration is an attractor. JULIAMC implements inverse iteration. The parameters of the JULIAMC are: the real and imaginary part of the problem parameter c, i.e. a and b, and the number of points of the fractal to be generated, n. It is the simplest Julia fractal generator.

$$z_i = z_{i-1}^2 + c, \quad i = 1, 2, 3, ... \qquad \rightarrow$$

$$\text{Inverse:} \quad z_{i-1} = \pm\sqrt{z_i - c}, \quad i = n + 10, ... , 1, \quad |z_{n+10}| \leq 1.$$



```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -165 -85 165 85
%%BeginSetup
%%EndSetup
%%BeginProlog
(C:\\PSlib\\PSlib.eps) run
/JULIAMCdict 11 dict def  %local dictionary
/JULIAMC{%stack: a, b, maxk.
        %a+ib complex constant c,  maxk maximal iterations
        % ==> Julia set of z^2 + a + ib
JULIAMCdict begin%open the local dictionary
/kmax exch def /b exch def /a exch def  /Courier 7 selectfont
/nextpoint{/x1 x a sub 2 div def /y1 y b sub 2 div def
            x1 dup mul y1 dup mul add sqrt dup%R
           /y exch x1 sub sqrt y1 0 lt{neg}if def
           /x exch x1 add sqrt def
}bind def
/printxy{x s y s       moveto (.) centershow
    x s neg y s neg     moveto (.) centershow   %point symmetry
    b 0 eq y 0 ne and
       {x s y s neg moveto (.) centershow
        x s neg y s moveto (.) centershow}if%symmetry x- and y-axes
}bind def
/s {100 mul} def                        %scaling
/nrand rand 2147483647 div def          %random number in [0,1]
/x nrand def /y nrand def               %start values in [0,1]
10{nextpoint}repeat                     %discard 10 iterations
kmax{nextpoint
     rand 1073741823 gt {/x x neg def /y y neg def}if
     printxy
    }repeat
end}bind def
%%EndProlog
-1    0   5000 JULIAMC showpage  %SanMarco
-.59 -.34 5000 JULIAMC showpage  %cloud
%%EOF
```



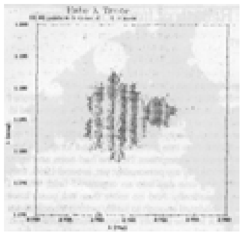
PSlib.eps contains moreover defs JULIABS, which implements the boundary scan method, JULIAF, which generates a filled fractal, JULIAP, which implements the pixel method, JULIAD, which is based on the distance formula.

Chaos and Fractalus are packages, which create fractals with rich colour possibilities.

Mandelbrot’s fractal

The picture consists of a cardioid, some circular bulbs, hairy details and stretching out with an “antenna”. So nice to find a real-life application where a classical math contour, cardioid, pops up.

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox:  -210 -135 85 135
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/MANDELdict 20 dict def
/MANDEL%==>Mandelbrotfractal in coloured bands
{MANDELdict begin /Courier 12 selectfont
/colours [/white /blue /lightblue /green /lightgreen
          /red /lightred /brown /yellow] def
/step .01 def /sc {100 mul} def
-2.1 step .85{/a exch def /asc a sc def
   0 step 1.35 {/b exch def /bsc b sc def
     /u 4 a dup mul b dup mul add mul def
     /v u 2 a mul sub .25 add def
     u 8 a mul add -3.75 le           %exclude cardioid
     v v sqrt sub 2 a mul add .5 le or%exclude circle
     {/l 0 def}%inside white, do nothing
     {/x a def /y b def /k 0 def
         {%loop over k
          /z x def
          /x x dup mul y dup mul sub a add def
          /y 2 z mul y mul b add def
          /s x dup mul y dup mul add def
          /k k 1 add def
           s 100 gt k 50 eq or{exit}if
         }loop%k
       k 40 lt{/l k 8 mod def}{/l 0 def}ifelse
       colours l get cvx exec
       k 3 gt{asc bsc     moveto (.) show
              asc bsc neg moveto (.) show}if
     }ifelse
   }for%j
}for%i
end}bind def
%%Endprolog
MANDEL
%respectively
%MANDELzw         %see PSlib.eps
%MANDELzwcontour  %see PSlib.eps
showpage
%%EOF
```


Mandelbrot’s first M-fractal

Mandelbrot in 1980 answered the question:

For which values of c will the Julia fractal, J(c), be line-like and for which values dust-like?

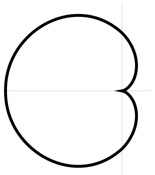
He was surprised, but … realized the relevance.

Mandelbrot also elaborated on the fractal dimension notion. The M-fractal curve, and surface, have fractal dimension D=2.

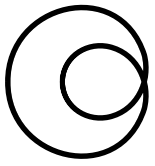
Form and size of M-fractal

M-cardiod

A Cardioid is defined in polar coordinates by $r = 2a(1 + \cos\theta)$, $\theta \in [0{,}2\pi]$. A version of the more general Limaçon, with parameter b=.25, $r = 2a(b + \cos\theta)$, $\theta \in [0{,}2\pi]$ is included at right.


Cardioid

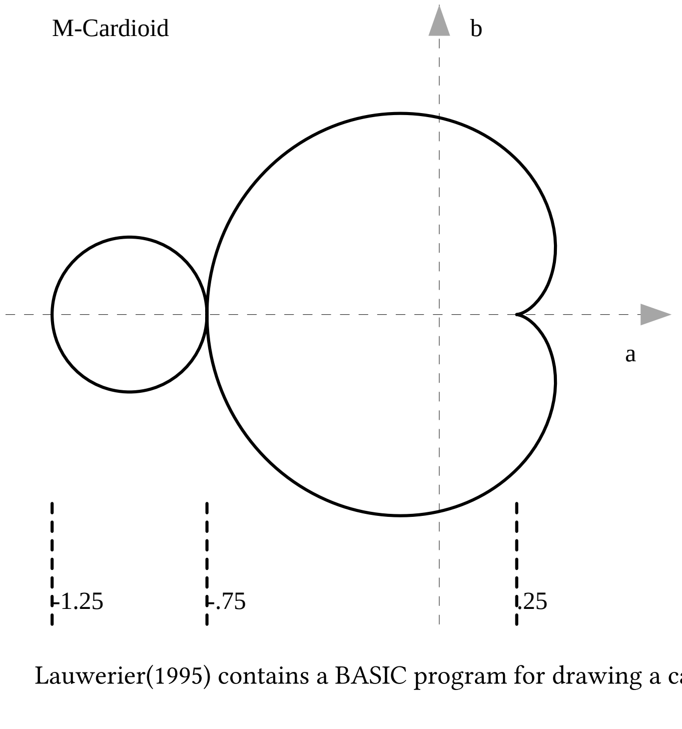
```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Cardioid in Polar Coordinates
%%BoundingBox: 0-30 -41 30
%%BeginSetup
%%EndSetup
/t 0 def /2a -20 def 2 2a mul 0 moveto
120{3 rotate /t t 3 add def
    2a 1 t cos add mul 0 lineto}repeat
stroke showpage
```


Limaçon

The equations for the cardioid in Cartesian coordinates, parametric in $\varphi \in [0, 2\pi]$, read

$$x = \cos(\varphi)/2 - \cos(2\varphi)/4$$
$$y = \sin(\varphi)/2 - \sin(2\varphi)/4$$

The cardioid has been drawn, with the main circle centre at (-1,0) and radius .25, next to it; see accompanying picture. The cardioid is of the same size as the M-fractal cardioid.
In order to have an idea of the scale in the M-fractal picture the relevant numbers are shown underneath the drawing. The a- and b-axis have been drawn dashed.
If $|4a^2 + 8a + b^2| < 3.75$ then (a,b) lies inside the cardioid.
If $|a + ib + 1| < .25$ then (a,b) lies inside the circle.



Lauwerier(1995) contains a BASIC program for drawing a cardiod, `CARDIO`.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: M-cardioid. cgl 2012
%%BoundingBox: -140 -105 80 100
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/m-cardioid{% Math cardoid of mandelbrot fractal
/Times-Roman 8 selectfont
/s {100 mul} def .01 setlinewidth [3] 0 setdash
-1.4 s 0 moveto .75 s 0 lineto
0 -1 s moveto 0 1 s lineto stroke
1 setlinewidth [] 0 setdash
.6 s -15 moveto (a) show
10 .9 s  moveto (b) show
.65 setgray%arrowheads coordinate axis
0 90 0 100 1 7 10 arrow fill %y-axis
65 0 75 0 1 7 10 arrow fill  %x-axis
0 setgray
.25 s 0 moveto
.1 .1 360.05{/phi exch def
    2 phi mul cos -4 div phi cos -2 div sub s
    2 phi mul sin 4 div phi sin 2 div sub s lineto
   }for
-.75 s 0 moveto
-1 s 0 .25 s 0 360 arc stroke
[3] 0 setdash
-1.25 s -1 s moveto -1.25 s -.6 s lineto stroke
-1.25 s -.95 s moveto (-1.25) show
-0.75 s -1 s moveto -0.75 s -.6 s lineto stroke
-0.75 s -.95 s moveto (-.75) show
0.25 s -1 s moveto 0.25 s -.6 s lineto stroke
0.25 s -.95 s moveto (.25) show
%
-1.25 s .9 s moveto (M-Cardioid) show
}bind def
%%EndProlog
m-cardioid showpage
%%EOF
```

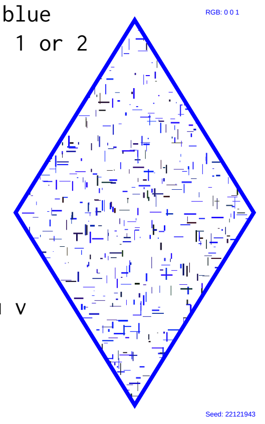
Wim W. Wilhelm communicated his compact specification for drawing the cardioid in Mathematica, using Cartesian coordinates.

```
ParametricPlot[{Cos(fi)/2-Cos(2 fi)/4, Sin(fi)/2-Sin(2 fi)/4}, (fi, 0, 2 pi)]
```

Mondriaan's line pieces.

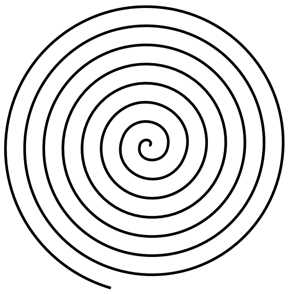
Since *à la Mondriaan* the PostScript program has been extended by a square and circle as frame. In the paper the MetaPost program as well as the PostScript program have been developed and discussed. Because of the seed the pictures are unique. It illustrates the use of PostScript's random number generator. The whole page is used.

```
%!PS-Adobe-3.0 EPSF-3.0
(c:\\PSlib\\PSlib.eps) run
/mondrian% 2014 extended by circle and square
%birthday: ddmmyyyy, a number as seed for srand
%three numbers from the closed interval [0, 1], for the rgb-color values: red green blue
%number for the kind of frame (0=rectanglere 1=Oval 2=Lozenge 3=square 4=circle): 0, 1 or 2
%==> generated Mondrian-alike
{0 begin gsave %savety for not changing the graphics state outside
  % used from the library: unifrmdef, maxrandom, ellipse
/form exch def
/b exch def /g exch def /r exch def /date exch def
date srand% start random generator with (birthday date) seed
100 50 translate
%wired-in parameters
/u 420 def /v u 1.618 mul def /hu u 2 div def /hv v 2 div def%BB of rectangle: 0 0 u v
/maxrandom 500 def /maxlength 20 def /maxwidth 3 def /eps 0.1 def
/hx  {u unifrmdev}            def
/hy  {v unifrmdev}           def
/l    {maxlength unifrmdev}def
/w  {maxwidth unifrmdev} def
/spread {2 unifrmdev mul }def                 %(0, 2)
/color{r 0 eq {eps}{r} ifelse spread g 0 eq {eps}{g} ifelse spread b 0 eq {eps}{b} ifelse spread} def
form 0 eq {/contour {0 0 moveto u 0 lineto u v lineto 0 v lineto closepath} def} if      %rectangle
form 1 eq {/contour {hu hv hu hv 0 360 ellipse} def} if    %oval
form 2 eq {/contour {hu 0 moveto u hv lineto hu v lineto 0 hv lineto closepath} def} if%lozenge
form 3 eq {/contour {0 0 moveto u 0 lineto u u lineto 0 u lineto closepath} def} if      %square
form 4 eq {/contour {hu hv hu hu 0 360 ellipse} def} if    %circle
 %oval
gsave contour clip%random pattern will only show up in (is clipped to) contour
maxrandom{%draw pattern in loop confined to contour
/xaux hx def /yaux hy def%position in (0, u) x (0, v) rectangle
/laux l 2 div def
xaux laux sub yaux moveto  xaux laux add yaux lineto w setlinewidth color setrgbcolor stroke%h-line
/xaux hx def /yaux hy def
/laux l 2 div def
xaux yaux laux sub moveto  xaux yaux laux add lineto w setlinewidth color setrgbcolor stroke%v-line
}repeat
grestore %end clipping path
contour 7 setlinewidth r g b setrgbcolor stroke%original color od choice
H12pt setfont  /nstr 8 string def %0 0 0 setrgbcolor
u 85 sub v 10 add moveto (RGB: ) show
  r nstr cvs show ( ) show  g nstr cvs show ( ) show b nstr cvs show
u 85 sub -20 moveto (Seed: ) show  date nstr cvs show
grestore end}def%end Mondrian
/mondrian load 0 26 dict put
22121943 0 0 1 2 mondrian showpage %example of use
%%EOF
```

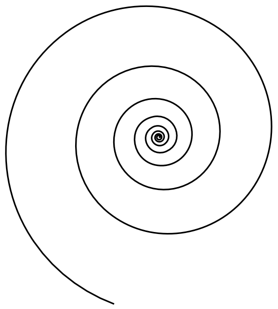


Spirals: Archimedian and Growth

The Archimedes Spiral is in polar coordinates defined by $r_\theta = k\theta, \quad 0 \leq \theta < \infty$.
The Growth Spiral is in polar coordinates defined by $\ln r_\theta = k\theta$ or $r_\theta = \mathrm{e}^{k\theta}, \quad -\infty < \theta < \infty$.



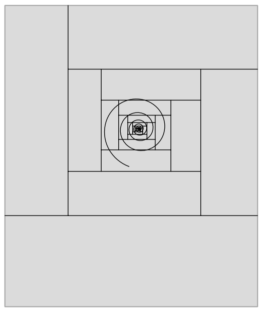
```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Archimedes Spiral
%%BoundingBox: -56 -59 54 55
%%BeginSetup
%%EndSetup
/archimedesspiral{0 0 moveto
/f .1 def /r 0 def
555{5 rotate /r r f add def
    r 0 lineto}repeat stroke
}bind def
%%EndProlog
archimedesspiral showpage
%%EOF
```



```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Growth Spiral
%%BoundingBox: -100 -110 75 86
%%BeginSetup
%%EndSetup
/growthspiralpc{1 0 moveto%off the 0
/f 2.718 .0085 exp def /r 1 def
555{5 rotate /r r f mul def
    r 0 lineto}repeat stroke
}bind def
%%EndProlog
growthspirapc
```

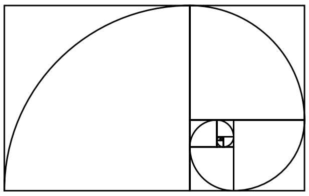
The Gyre Open font Type activity logo

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Imitation Gyre-logo, cgl 2012
%%BoundingBox: -2 -2 722 862
%%BeginSetup
%%EndSetup
%%BeginProlog
(C:\\PSlib\\PSlib.eps) run % contains growthspiral
%%EndProlog
/gyrelogodict 8 dict def
/gyrelogo{gyrelogodict begin
/ux 720 def /uy 860 def
gsave .86 setgray 0 0 ux uy rectfill
  3 setlinewidth .65 setgray 0 0 ux uy rectstroke%background
grestore
2 setlinewidth
/xl{ux 180 720 div mul}def /xu{ux xl .9 mul sub}def
/yl{uy 260 860 div mul}def /yu{uy yl .7 mul sub}def
7{xl yl moveto xl uy lineto 0 yl moveto  ux yl lineto %low
  xu yl moveto xu yu lineto xl yu moveto ux yu lineto %up
```



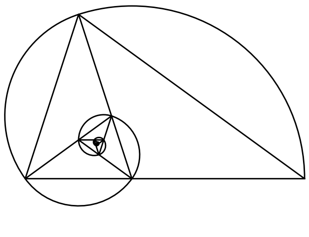
```
  stroke
  xl yl translate /ux xu xl sub def /uy yu yl sub def
}repeat
%growth spiral
ux uy translate growthspiralpc
end}def
gyrelogo showpage
%%EOF
```

Variant growth spiral



```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Growth spiral. cgl 1997
%%BoundingBox: -16 -22 201 126
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/growthspiraldict 6 dict def
/growthspiral{growthspiraldict begin %tilLiv
0 setlinejoin 1 setlinecap
/x 200 def /y .618 x mul def
/square{0 y lineto y y lineto y 0 lineto
  y 0 y 180 90 arcn
  y y translate}def
12{0 0 moveto square -90 rotate
  /aux x def /x y def /y aux y sub def
 }repeat stroke
end}def
%%EndProlog
growthspiral showpage
%%EOF
```

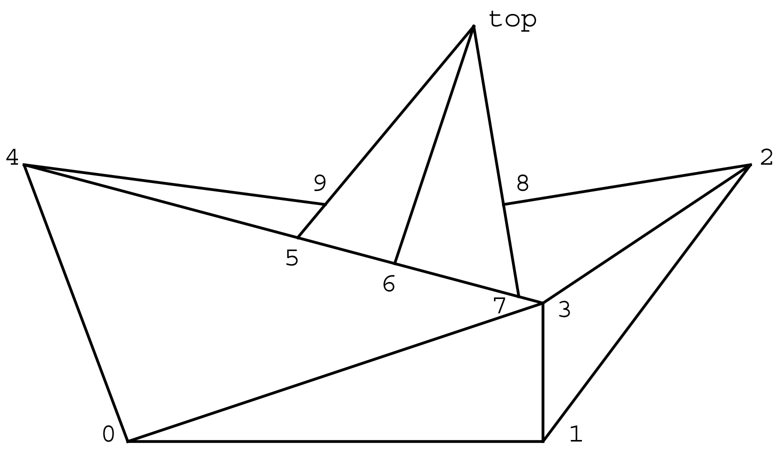
Variant growth spiral



```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Growth spiral II. cgl 1997
%%BoundingBox: -16 -22 201 126
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/growthspiralIIdict 3 dict def
/growthspiralII{growthspiralIIdict begin %tilLvi
0 setlinejoin 1 setlinecap
/x 200 def /xg{.618 x mul}def
/tri{x xg sub 0 moveto
     currentpoint translate
     0 0 moveto 0 0 xg 0 108 arc
     xg 0 lineto stroke
    108 rotate /x xg def
}def
12{tri}repeat
end}def
%%EndProlog
growthspiralII showpage
%%EOF
```

Gust battleship logo

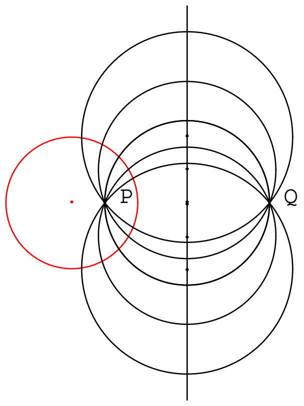
The programming is interesting because the intersection point of straight lines has to be calculated.



```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: GUST Battleship, 1996
%%BoundingBox: -41 -1 231 151
%%BeginSetup
%%EndSetup
%%BeginPrologue
(C:\\PSlib\\PSlib.eps) run
/battleshipdict 13 dict def
battleshipdict begin
/s 50 def 2 setlinejoin
/p0{0 0}def
/p1{3 s mul 0}def
/p2{4.5 s mul 2 s mul}def
/p3{3 s mul s}def
/p4{-.75 s mul 2 s mul}def
/top{2.5 s mul 3 s mul}def
/p5{p0 top p3 p4 intersect}def
/p6{p0 p1 mean top p3 p4 intersect}def
/p7{top p1 p3 p4 intersect}def
/p8{p2 p5 top p1 intersect}def
/p9{p8 dup 0 exch top p0 intersect}def
end
%
/battleship{battleshipdict begin
p0 moveto p1 lineto p2 lineto p3 lineto p0 lineto
p1 moveto p3 lineto p4 lineto p0 lineto
top moveto p5 lineto
top moveto p6 lineto
top moveto p7 lineto
p2 moveto p8 lineto
p4 moveto p9 lineto
stroke
end}def
%%EndPrologue
battleship showpage
```

Orthogonal circles II

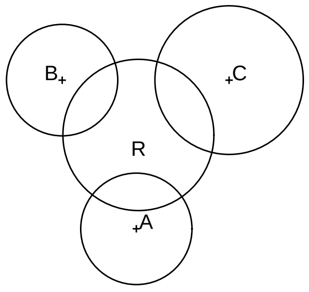
The problem is given a circle and a point P inside the circle construct a circle bundle through P which cut the original circle orthogonally. (Q is the inverse of P).



```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Orthogonal circle bundle. cgl 2014
%%BoundingBox: -105 -150 125 150
(c:\\PSlib\\PSlib.eps) run
/orthogonalcirclebundledict 11 dict def
/orthogonalcirclebundle{orthogonalcirclebundledict begin
/r 50 def /Courier 18 selectfont
/px -25 def/py 0 def
/qx r r mul  px neg div def /qy 0 def%inverse
gsave 1 0 0 setrgbcolor -50 0 r 0 360 arc stroke
-50 0 moveto (.)centershow  %(C) show
grestore
px py moveto (.) centershow ( P) show
qx qy moveto (.) centershow ( Q) show
/m px qx add 2 div def%middle
m 150 moveto m -150 lineto stroke%middleline
2{/r1 m px sub def
m 0 r1 0 360 arc stroke gsave m 0 moveto (.)centershow grestore
/r2 m px sub dup mul 625 add sqrt def
m 25 r2 0 360 arc stroke gsave m 25 moveto (.)centershow grestore
/r3 m px sub dup mul 2500 add sqrt def
m 50 r3 0 360 arc stroke gsave m 50 moveto (.)centershow grestore
1 -1 scale}repeat
end}def
orthogonalcirclebundle showpage
%%EOF
```

Orthogonal circles III. Radical circle

The problem is given three circles construct a circle which cuts the original circles orthogonally.

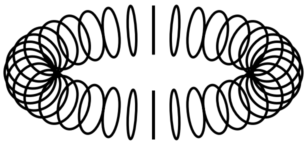


```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Radical circle. CGL april2010
%%BoundingBox: -190 -185 235 215
(C:\\PSlib\\PSlib.eps) run %PS library
/r 100 def /mr r neg def  H14pt setfont
/Ax 0 def  /Ay mr  def           /A {Ax Ay} def  /Ar .75 r mul def
/Bx mr def /By r  def             /B {Bx By} def   /Br .75 r mul def
/Cx 1.25 r mul def  /Cy r def   /C {Cx Cy} def  /Cr r def
A plus B plus C plus
newpath A Ar 0 360 arc stroke A moveto  2 0 rmoveto (A) show
newpath B Br 0 360 arc stroke  B moveto -12 0 rmoveto (B) show
newpath C Cr 0 360 arc stroke C moveto   2 0 rmoveto (C) show
Ax Ay Ar Bx By Br Cx Cy Cr radical
/radr exch def /rady exch def /radx exch def
newpath radx rady radr 0 360 arc stroke
radx rady plus
radx 3 add rady 5 sub moveto (Radical) show showpage
%%EOF
```

An ill-posed subproblem — touching point of 2 circles — has been reformulated in a well-posed problem.

Toroid

Lauwerier in *Meetkunde met de microcomputer* explained and demonstrated the picture.

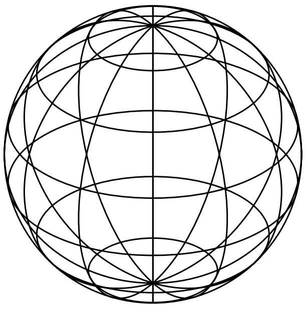


```
%!PS-Adobe-3.0  EPSF-3.0
%%BoundingBox: -61 -28 61 28
%%BeginProlog
(c:\\PSlib\\Pslib.eps) run
/toroiddict 16 dict def
toroiddict begin%locals
/R 50 def /r 10 def /phi 0 def /theta 20 def
/a1{R r sub t cos mul   R r sub t sin mul   0         ptp}def
/a2{R r sub t cos mul   R r sub t sin mul   .55 r mul ptp}def
/a3{R r sub .55 r mul add t cos mul R r sub .55 r mul add t sin mul r ptp}def
/a4{R t cos mul    R t sin mul              r         ptp}def
/a5{R .55 r mul add t cos mul R .55 r mul add t sin mul r ptp}def
/a6{R r add t cos mul   R r add t sin mul   .55 r mul ptp}def
/a7{R r add t cos mul   R r add t sin mul   0         ptp}def
/a8{R r add t cos mul   R r add t sin mul  -.55 r mul ptp}def
/a9{R .55 r mul add t cos mul R .55 r mul add t sin mul r neg  ptp}def
/a10{R  t cos mul       R  t sin mul         r neg  ptp}def
/a11{R .55 r mul sub t cos mul  R .55 r mul sub t sin mul r neg  ptp}def
/a12{R r sub t cos mul   R r sub t sin mul  -.55 r mul ptp}def
end
/toroid{toroiddict begin
%used from library: ptp
0 10 360{/t exch def a1 moveto a2 a3 a4 curveto a5 a6 a7 curveto
  a8 a9 a10 curveto a11 a12 a1 curveto closepath
}for stroke
end}def
%%EndProlog
toroid showpage
%%EOF
```

A direct translation of Lauwerier's BASIC code

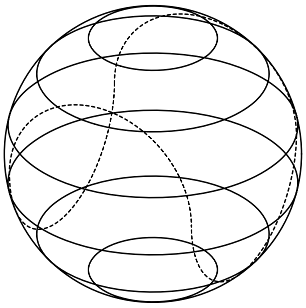
```
%!PS-Adobe-3.0 EPSF=3.0  %Torus, cgl Jan 07
%%BoundingBox: -300 -100 300 300
%%Creator: H Lauwerier. Meetkunde met de microcomputer. Adapted to PS cgl Jan2007
/PI 3.141593 def /C .8 def  /C1 .7071 1 C C mul sub sqrt mul def
/r1 235 def /r2 40 def
/ptp2{/z exch def/y exch def/x exch def %point to pair projection
   y x sub .7071 mul
   C z mul x y add C1 mul sub}def
/n 75 def 300 300 translate
0 1 n 1 sub{/k exch def
   /A k n div 360 mul cos def
   /B k n div 360 mul sin def
   0 10 360{/t exch def
           B neg r1 r2 t cos mul add mul %x stacked
           A     r1 r2 t cos mul add mul %y stacked
           r2 t sin  mul                 %z stacked
           ptp2                          %u,v stacked xyz consumed
           t 0 eq{moveto}{lineto}ifelse
   }for stroke %less memory requirement than stroke on the end
}for
showpage
```

Sphere with meridians and latitude circles



```
%%!PS-Adobe-3.0 EPSF-3.0
%%Title: Spere with meridans,latitude circles
%%Author: H A Lauwerier(1987): Meetkunde met de microcomputer, Epsilon
%%Transcriptor: Kees van der laan, kisa1@xs4all.nl,  2012
%%BoundingBox: -101 -102 101 102
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/spheremeridianslatitudesdict 25 dict def
/spheremeridianslatitudes{spheremeridianslatitudesdict begin
/m 6 def /n 6 def /r 100 def /-r r neg def
/latitudecircles{1 1 m{/i exch def /t i 180 mul m 1 add div def
 /x r t sin mul def /y 0 def /z r t cos mul def
 x y z ptp moveto
 1 1 100{/j exch def  /s j 180 mul 50 div def
        /x{r s cos t sin mul mul}def
        /y{r s sin t sin mul mul}def
        x y z ptp lineto
        }for %j
}for %i
stroke}def%latitudecircles
/meridians{1 1 n{/i exch def /s i 180 mul n div def
  /x{0}def /y{0}def /z{r}def
 x y z ptp moveto
 1 1 100{/j exch def  /t j 180 mul 50 div def
        /x{r s cos t sin mul mul}def
        /y{r s sin t sin mul mul}def
        /z{r t cos mul}def
        x y z ptp lineto
       }for %j
}for %i
stroke}def%meridians
r 0 moveto  0 0 r 0 360 arc stroke%circle in projection plane
latitudecircles
meridians
end} bind def
%%Endprolog
/phi 30 def /theta 30 def
spheremeridianslatitudes
showpage
%%EOF
```

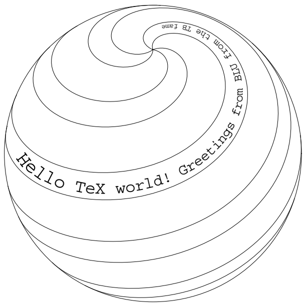
With tennisball curve dashed
(note scaling $\sqrt{2}$, because tennisball curve is for sphere with radius $\sqrt{2}$.)



```
%tenniscurve
/tennisball{/r r 2 sqrt div def /-r r neg def
%Tennisball part {t:(1, sin t, cos t)}
r -r 0 ptp moveto
-89 1 90{/t exch def
 r  t sin r mul  t cos r mul  ptp lineto
}for
%Tennisball part {t:(-1, sin t, cos t)}
-r   -r   0  ptp moveto
-89 1 90{/t exch def
-r   t sin r mul  t cos r mul  ptp lineto
}for
%Tennisball part {t:(-sin t, 1, -cos t)}
r   r   0 ptp moveto
-89 1 90{/t exch def
t sin neg r mul   r   t cos neg r mul    ptp lineto
}for
%Tennisball part {t:(-sin t, -1, -cos t)}
r   -r   0 ptp moveto
-89 1 90{/t exch def
t sin neg r mul    -r    t cos neg r mul   ptp lineto
}for
[2] 1 setdash stroke}def%tennisball
```

Sphere with spiral and text

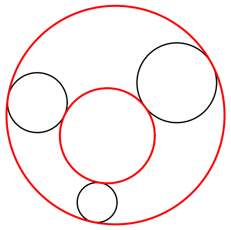
Interesting is the diminishing font.

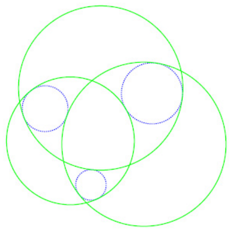


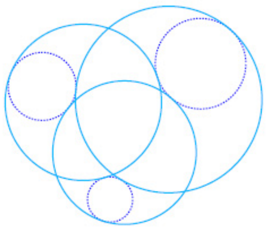
```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Spere with belts and text, kisa1@xs4all.nl, 2012
%%BoundingBox: -101 -102 101 102
%%BeginSetup
%%EndSetup
(C:\\PSlib\\PSlib.eps) run
/Courier 14 selectfont
pathtextdict begin /size 14 def
/linetoproc
  { /oldx newx def /oldy newy def
       /newy exch def /newx exch def
       /dx newx oldx sub def
       /dy newy oldy sub def
       /dist dx dup mul dy dup mul add sqrt def
       dist 0 ne
         { /dsx dx dist div ovr mul def
           /dsy dy dist div ovr mul def
           oldx dsx add oldy dsy add transform
           /cpy exch def /cpx exch def
           /pathdist pathdist dist add def
              {
/size size .08 sub def
/Courier findfont [size 0 0 size 0 0] makefont setfont
                 setdist pathdist le
                { charcount str length lt
                     {setchar} {exit} ifelse }
                { /ovr setdist pathdist sub def exit }
                ifelse
              } loop
      } if
  } def
end
/sphereandspiraldict 30 dict def
/sphereandspiral{sphereandspiraldict begin /r 100 def
/c .7 def /a2 1 1.41421 div def /b1 a2 1 c c mul sub sqrt mul def
/c1 c a2 mul def /c3 1 c c mul sub sqrt def
/spiral{-90 1 +90{/thetaj exch def
 /phij thetaj 4 mul phi0 add def%windings
 /x r phij cos thetaj cos mul mul def
 /y r phij sin thetaj cos mul mul def
 /z r thetaj sin mul def
 /w c1 x y add mul c3 z mul add def
 /u a2 y x sub mul def
 /v c z mul b1 x y add mul sub def
 w 0 lt{u v moveto}{u v lineto}ifelse
}for %thetaj
}def%spiral
gsave
0 c r mul moveto /phi0   0 def spiral
0 c r mul moveto /phi0  45 def spiral stroke
grestore
gsave
0 c r mul moveto /phi0 120 def spiral
0 c r mul moveto /phi0 165 def spiral stroke
grestore
gsave
0 c r mul moveto /phi0 240 def spiral stroke
0 c r mul moveto /phi0 285 def spiral stroke
0 c r mul moveto /phi0 275 def spiral
( Hello TeX world! Greetings from BLU from the TB fame) 20 pathtext
grestore
%circle in projection plane
r 0 moveto
1 1 100{/k exch def  /t k 180 mul 50 div def
        r t cos mul r t sin mul lineto
}for %k
stroke end} bind def
```

Apollonius problem

Given three disjunct circles find the circles touching the circles. In *Circle Inversion* the problem has been solved and all 8 solutions have been programmed in MetaPost and PostScript. A universal PostScript definition `apollonius` has emerged. The definition for a slight variant for the case when one given circle contains the two other ones is called `apollonius2`. The inversion method for solving Apollonius problem has also been explained and illustrated in the article.



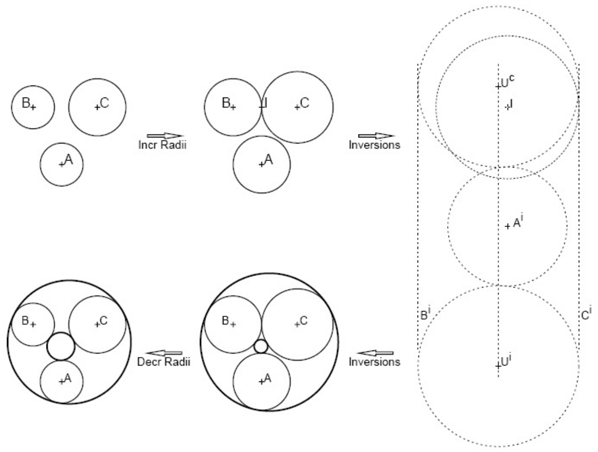




```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Apollonius. Three circles ->circum-/inscribed circle. cgl 2010
%%BoundingBox: -71 -50 100 121
(C:\\PSlib\\PSlib.eps) run
%given circles
/r 15 def /r1 r  def /x1 0 def /y1 r -2 mul def
/r2 r 1.5 mul def /x2 r -3 mul def  /y2 x2 neg def
/r3 r 2 mul def /x3 r 4 mul  def  /y3 x3 def
newpath%show the given circles
x1 y1 r1 0 360 arc stroke
x2 y2 r2 0 360 arc stroke
x3 y3 r3 0 360 arc stroke
%inscribed circle
x1 y1 r1
x2 y2 r2
x3 y3 r3  Apollonius /ri exch def /yi exch def /xi  exch def
red newpath xi yi ri  0 360 arc stroke%inscribed
%circumscribed circle
x1 y1 r1 neg
x2 y2 r2 neg
x3 y3 r3 neg Apollonius /rout exch def /yout exch def /xout exch def
newpath  xout yout rout 0 360 arc stroke%circumscribed
showpage
%%EOF
```

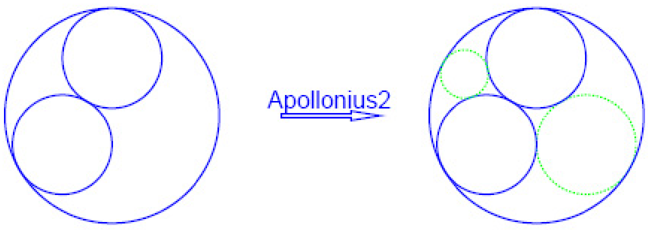
Note that the radius is specified negatively when the solution circle should contain the given circle. So the blue case has one negative radius specified and the green case two.

Below a picture of the inversion method.

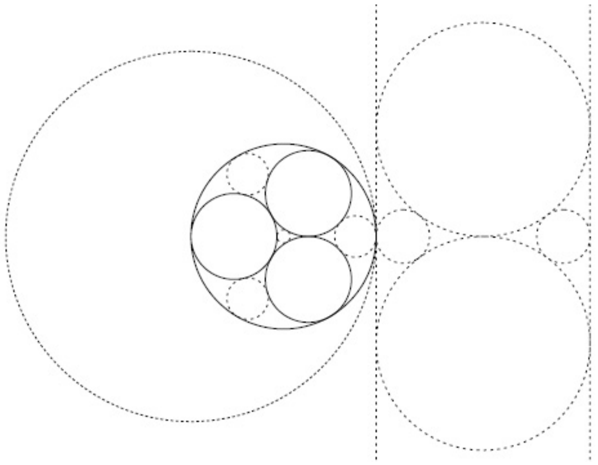


Apollonius problem, variant non-disjunct circles

Given three non-disjunct circles, one circle contains the other two, find the circles touching the circles. In *Circle Inversions* the problem has been solved and all 8 solutions have been programmed in MetaPost and PostScript. A universal PostScript definition apollonius2 for this case has emerged.

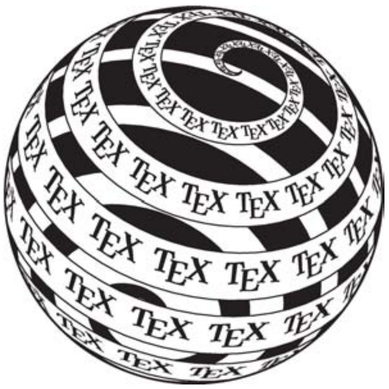


```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Rerich Circles. CGL 2010
%%BoundingBox: 40 240 370 360
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/r 25 def /y r 3 sqrt div dup add def  /x 0 def
/x1 x def /y1 y def /r1 r def
/ri y r sub def %center  (0,0)
/R y r add def %center  (0,0)
x y -120 rot    %rotate center
/y2 exch def   /x2 exch def
/y3 y2 def      /x3 x2 neg def

100 300 translate%left figure
 0 0 R 0 360 arc stroke
gsave 2{newpath x y r 0 360 arc stroke 120 rotate}repeat
grestore

85 0 translate newpath 0 0 50 0 2 5 15 arrow stroke
                    -9 +4 moveto H12pt setfont (Apollonius2) show

128 0 translate%right figure
newpath 0 0 R 0 360 arc stroke
gsave 2{newpath x y r 0 360 arc stroke 120 rotate}repeat
grestore

x1 y1 r
x3 y3 r
0 0 R neg   Apollonius2
/rsnd1 exch def /ysnd1 exch def /xsnd1 exch def
/rsnd2 exch def /ysnd2 exch def /xsnd2 exch def
newpath xsnd2 ysnd2 rsnd2 0 360 arc stroke
newpath xsnd1 ysnd1 rsnd1 0 360 arc stroke
showpage
%%EOF
```



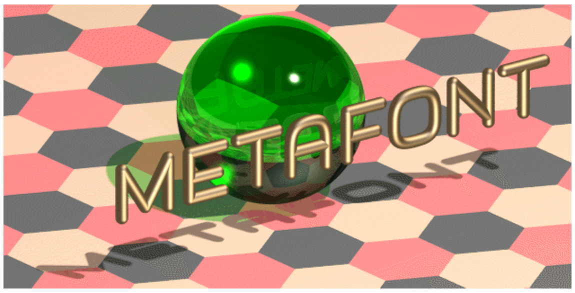
Tiling a circle by circles via inversion, tedious.

Jacko's Toruń98 logo.
The above is my best mimic in PostScript of Jackowski's Toruń98 dynamic masterpiece logo.



Metafont&TEX can be used to create beautiful artistic results with fonts.

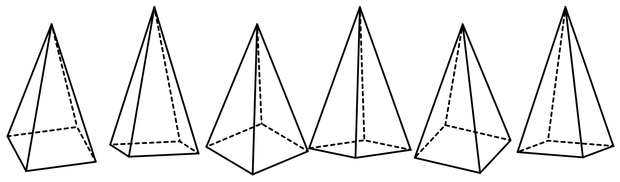
This has been shown in the 90-ies by Bogusłav Jackowski and Marek Ryćko. Nonscalability is not relevant for pieces of art.



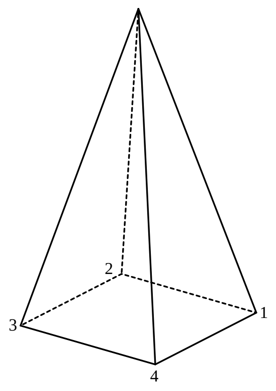
I could not reproduce these TEX-Metafont pictures in PostScript. Maybe the TEXnique has become outdated in view of the OpenTypeFonts activity. A challenge for the reader to realize this with OTFs?

Pyramids

Hobby showed in his MetaPost manual one full-page pyramid in MetaPost. Below are pyramids specified in 3D and projected with varying viewing angles.



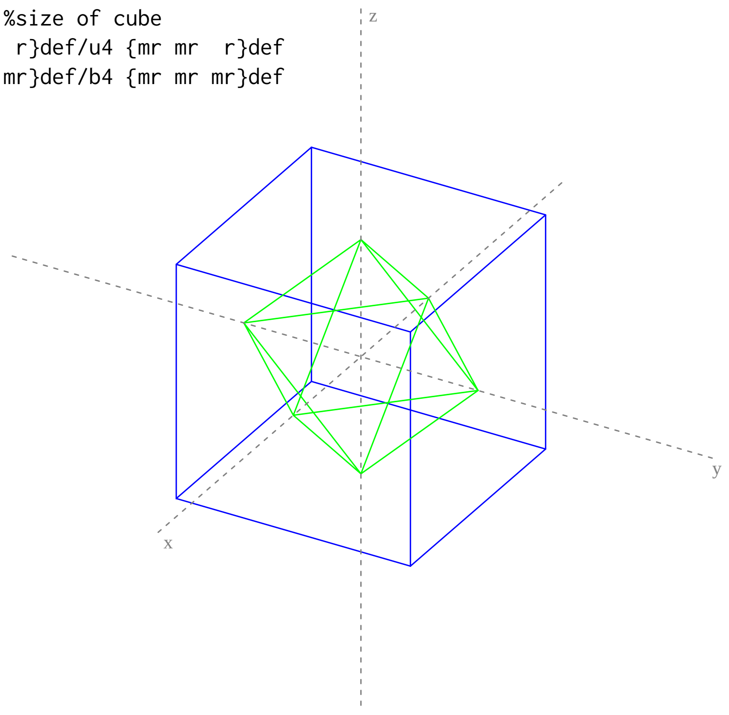
```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Pyramids in projection, cgl 2009
%%BoundingBox: -25 -20 315 85
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/pyramidsdict 9 dict def
pyramidsdict begin
/ptp{/z exch def/y exch def/x exch def
   x neg a cos mul y a sin mul add
   x neg a sin mul b sin mul y neg a cos mul b sin mul add
     z b cos mul add}def
/r 20 def /hr r 2 div def
1 setlinecap 1 setlinewidth
/z1{r neg r   0 ptp}def
/z2{r neg dup 0 ptp}def
/z3{r r neg   0 ptp}def
/z4{r r       0 ptp}def
/top{0 0 r 4 mul ptp}def
%
/pyramid{z1 moveto z2 lineto z3 lineto
   [2]1 setdash stroke
z3 moveto z4 lineto z1 lineto
   []0 setdash stroke
top moveto z1 lineto
top moveto z3 lineto
top moveto z4 lineto
%   -3 -10 rmoveto (4)show
stroke
top moveto z2 lineto
[2]1 setdash stroke
%  -10 0 rmoveto (2)show
} def  %end pyramid
end%pyramidsdic
/pyramids{pyramidsdict begin
15  25  65{/a exch def
30  -20 10{/b exch def
   pyramid 57 0 translate
}for}for
end}def
%%endProlog
pyramids  showpage
%%EOF
```



```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Pyramid in projection, cgl 1997
%%BoundingBox: -90 -40 90 190
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/pyramid{/ptp{/z exch def/y exch def/x exch def
                       x -.6 mul y .8 mul add
                        -4 x mul -3 y mul add
                                                12 z mul add 13 div}def
 /Times-Roman findfont 10 scalefont setfont
 /r 50 def /top{0 0 r 4 mul ptp}def
 /z1{r neg r  0 ptp}def  /z2{r neg dup 0 ptp}def /z3{r r neg 0 ptp}def /z4{r
r 0 ptp}def
 z1 moveto z2 lineto z3 lineto  [2]1 setdash stroke
 z3 moveto z4 lineto z1 lineto  []0  setdash stroke
 top moveto z1 lineto    2  -3 rmoveto(1)show
 top moveto z3 lineto    -7  -3 rmoveto(3)show
 top moveto z4 lineto   -3 -10 rmoveto(4)show  stroke
 top moveto z2 lineto
   -10  0 rmoveto(2)show
 [2]1 setdash stroke  }def
%%EndProlog
pyramid showpage
%%EOF
```

Octaeder in cube

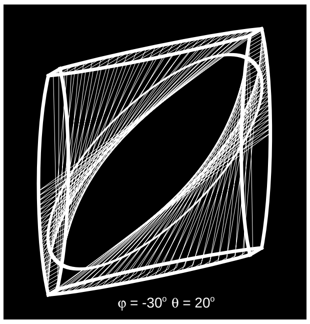
Essentially an old program. Nowadays I would use ptp in the definition of the points.

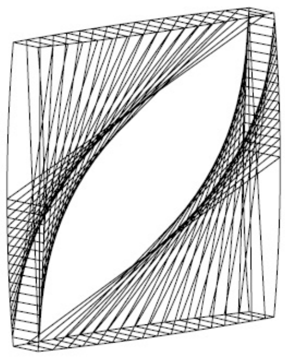


```
%!PS-Adobe-3.0  EPSF-3.0
%%Title: Cube-Oktaeder
%%BoundingBox: -280 -280 280 280
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\PSlib.eps) run
/cubedict 12 dict def
/cube{cubedict begin/r 100 def /mr r neg def %size of cube
/u1 { r mr  r}def/u2 { r  r  r}def/u3 {mr  r  r}def/u4 {mr mr  r}def
/b1 { r mr mr}def/b2 { r  r mr}def/b3 {mr  r mr}def/b4 {mr mr mr}def
u1 ptp moveto %u1%upper
u2 ptp lineto %u2
u3 ptp lineto %u3
u4 ptp lineto %u4
u1 ptp lineto %u1
b1 ptp moveto %b1 %lower
b2 ptp lineto %b2
b3 ptp lineto %b3
b4 ptp lineto %b4
b1 ptp lineto %b1
%edges (vertical)
u1 ptp moveto %u1
b1 ptp lineto %b1
u2 ptp moveto %u2
b2 ptp lineto %b2
u3 ptp moveto %u3
b3 ptp lineto %b3
u4 ptp moveto %u4
b4 ptp lineto %b4
end}def
/octaederdict 12 dict def
/octaeder{octaederdict begin
/r 100 def /mr r neg def  /2r r 2 mul def  /m2r 2r neg def %size of octaeder
top {0 0  r  }def/v4  {0 mr 0  }def /v1  {r 0  0  }def /v2  {0  r 0   }def
/v3  {mr 0  0 }def /nadir {0 0  mr}def
top ptp moveto %top
v4 ptp lineto
v1 ptp lineto %v1
v2 ptp lineto %v2
v3 ptp lineto %v3
v4 ptp lineto %v4
%
top ptp moveto %top
v1  ptp lineto %v1
top ptp moveto %top
v2  ptp lineto %v2
top ptp moveto %top
v3  ptp lineto %v3
%
nadir ptp moveto %nadir
v4 ptp lineto %v4
nadir ptp moveto
v1 ptp lineto %v1
nadir ptp moveto
v2 ptp lineto %v2
nadir ptp moveto
v3 ptp lineto %v3
}def
/cubeoktaederdict 5 dict def
/cubeoktaeder{cubeoktaederdict begin  /Times-Roman 14 selectfont
cube     0 0 1 setrgbcolor stroke
octaeder 0 1 0 setrgbcolor stroke
0.5 0.5 0.5 setrgbcolor
%x-y-z coordinate axis
/r r 3 mul def /mr r neg def
mr 0 0 ptp moveto r  0 0 ptp lineto 4 -12 rmoveto (x) show
0 mr 0 ptp moveto 0  r 0 ptp lineto 0 -12 rmoveto (y) show
0 0 mr ptp moveto 0 0  r ptp lineto 6 -12 rmoveto (z) show [3 5]6 setdash stroke
end}def
/phi 30 def/theta 30 def cubeoktaeder
showpage
%%EOF
```

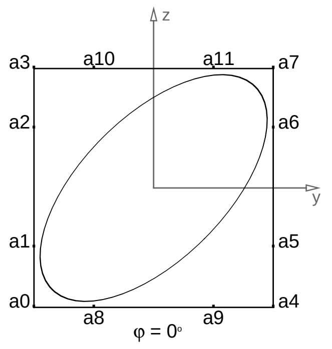
Emulation Gabo's linear construction no1

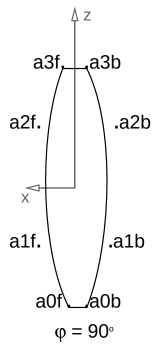
For more Gabo emulations, see Gabo's Torsion.



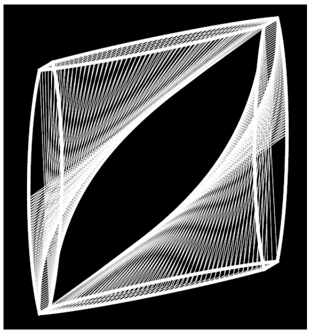


```
%!PS-Adobe-3.0  EPSF-3.0
%%BoundingBox: -61 -28 61 28
%%BeginSetup
%%EndSetup
%%BeginProlog
(c:\\PSlib\\Pslib.eps) run
/linearconstructionno1dict 50 dict def
linearconstructionno1dict begin
/reversevideo {-130 -135 260 270 rectfill} def %Mimics BoundingBox
%Data
/origin { 0 0 } def
/a0f  { .1 s  -2 s  -2 s ptp  }def%y constant
/a1f  { .6 s  -2 s  -1 s ptp  }def
/a2f  { .6 s  -2 s   1 s ptp  }def
/a3f  { .2 s  -2 s   2 s ptp  }def

/a4f  { .2 s   2 s  -2 s ptp  }def%y constant
/a5f  { .6 s   2 s  -1 s ptp  }def
/a6f  { .6 s   2 s   1 s ptp  }def
/a7f  { .2 s   2 s   2 s ptp  }def

/a8f  { .6 s   -1 s -2 s ptp  }def%z constant
/a9f  { .6 s    1 s -2 s ptp  }def

/a10f { .6 s   -1 s  2 s ptp  }def%z constant
/a11f { .6 s    1 s  2 s ptp  }def

/a0b  { -.2 s  -2 s -2 s ptp  }def%y constant
/a1b  { -.6 s  -2 s -1 s ptp  }def
/a2b  { -.7 s  -2 s  1 s ptp  }def
/a3b  { -.2 s  -2 s  2 s ptp  }def

/a4b  { -.2  s  2 s -2 s ptp  }def%y constant
/a5b  { -.6  s  2 s -1 s ptp  }def
/a6b  { -.6  s  2 s  1 s ptp  }def
/a7b  { -.2  s  2 s  2 s ptp  }def

/a8b  { -.6 s  -1 s -2 s ptp  }def%z constant
/a9b  { -.6 s   1 s -2 s ptp  }def

/a10b { -.6 s  -1 s  2 s ptp  }def%z constant
/a11b { -.6 s   1 s  2 s ptp  }def

/sampleellipsestroke{gsave
-45 rotate
.6 1.2 scale%kind of ellipse
1 1 360{/t exch def
2 s t cos mul 2 s t sin mul }for %sample the skewed ellipse
count /n exch def
0 3 1 roll  % 0 y z
ptp         % u v projected point
moveto
n 2 div 1 sub cvi {0 3 1 roll  % 0 y z
             ptp        % u v projected point
             lineto
            }repeat closepath
stroke grestore} def
%
/approxellipsestroke{gsave
-45 rotate
.6 1.2 scale%kind of ellips
/a0 {0 -2 s    0     ptp} def
/a1 {0 -2 s    1.1 s ptp} def
/a2 {0 -1.1 s  2 s   ptp} def
/a3 {0  0      2 s   ptp} def
/a4 {0  1.1 s  2 s   ptp} def
/a5 {0  2 s    1.1 s ptp} def
/a6 {0  2 s    0     ptp} def
/a7 {0  2 s   -1.1 s ptp} def
/a8 {0  1.1 s -2 s   ptp} def
/a9 {0  0     -2 s   ptp} def
/a10{0 -1.1 s -2 s   ptp} def
/a11{0 -2 s   -1.1 s ptp} def
a0 moveto a1  a2  a3 curveto
          a4  a5  a6 curveto
          a7  a8  a9 curveto
          a10 a11 a0 curveto stroke
grestore} def
/frame{
a0f moveto
  a1f  a2f  a3f curveto
  a10f a11f a7f curveto
  a6f  a5f  a4f curveto
  a9f  a8f  a0f curveto
a0b moveto
  a1b  a2b  a3b curveto
  a10b a11b a7b curveto
  a6b  a5b  a4b curveto
  a9b  a8b  a0b curveto
a0f moveto a0b lineto
a3f moveto a3b lineto
a4f moveto a4b lineto
a7f moveto a7b lineto
} def
/dostringing{
0.02 .02 .5001{/t exch def
t       a0f a1f  a2f  a3f tOnSpline moveto
2 t mul a3b a10b a11b a7b tOnSpline lineto
2 t mul a3f a10f a11f a7f tOnSpline lineto
t       a0b a1b  a2b  a3b tOnSpline lineto
closepath
t       a7f a6f  a5f  a4f tOnSpline moveto
2 t mul a4b a9b  a8b  a0b tOnSpline lineto
2 t mul a4f a9b  a8f  a0f tOnSpline lineto
t       a7b a6b  a5b  a4b tOnSpline lineto
closepath
}for } bind def
.1 setlinewidth stroke
/annotation
```





```
{-32 -125  moveto
 S12pt setfont (j) show ( = ) H12pt setfont show phi (   ) cvs show
   gsave 0 5 rmoveto (o) H7pt setfont show grestore
8 0 rmoveto
 S12pt setfont (q) show ( = ) H12pt setfont show theta (  ) cvs show
         0 5 rmoveto (o) H7pt setfont show
} def
end


/linearconstructionno1
{linearconstructionno1dict begin
 /phi 30 def /theta 5 def /scalingfactor 50 def
reversevideo 1 setgray%white
3  setlinewidth frame        stroke
2  setlinewidth approxellipsestroke %sampleellipsestroke
.5 setlinewidth dostringing  stroke
annotation
end}def
%%Endprolog
linearconstructionno1
%%EOF
```



Gabo's Torsion

Conclusions

The use of PostScript pictures in AllTEX documents has a history of 20 years already, to start with the use of DVIPS. With pdfTEX the use of direct PostScript inclusion was hampered. .eps has to be converted into .pdf. Adobe seems to have stopped the support of the run command for library inclusion, which is a big loss. It seems that the use of PostScript pictures is still possible in TEX documents, but the future does not look promissing. This holds also for MetaPost and PSTricks pictures, because MetaPost is just a preprocessor for PostScript and PSTricks uses PostScript under the hood. For the future we need a better and more stable graphics tool to cooperate with TEX&Co. On the other hand: is it too optimistic to expect that the PostScript programs will be read and used?

Acknowledgements

Thank you Adobe for your maintained, adapted to Language 3 since 1997, good old, industrial standard PostScript and `Acrobat Pro` (actually DISTILLER) to view it, Don Knuth for your stable plain TEX, Jonathan Kew for the TEXworks IDE,

Hàn Thế Thành for pdf(La)TEX,

Thank you Jos Winnink and Henk Jansen for proofing. MAPS editors for improving my use of English.

Kees van der Laan
Hunzeweg 57, 9893PB Garnwerd, Gr, NL
kisa1@xs4all.nl