Gabo's Torsion

—and some more—

Abstract
Gabo's Torsion is emulated in EPSF, Encapsulated PostScript File format. Gabo's constructive art, Math, Computer Graphics and the use of PostScript are touched upon. Whether PostScript is a suitable language for projection and drawing 3D objects on paper is experienced. An introduction to PostScript aimed at EPSF use, in a nutshell, is included. How to obtain cropped pictures along with the conversion to .pdf is mentioned. An interesting observation is made: Bézier cubics, specified by begin point, the control points and the end point, are invariant under (oblique parallel) projection, which allows to project B-cubics efficiently. The efficient projection of (approximated) circles and ellipses has been addressed. For the evaluation of B-cubics de Casteljau's algorithm is used. Emulations in EPSF of Gabo's Linear Construction in Space No 1 and 2, of one of his Spheric Themes, and his Linear Construction Suspended, are also included. For the MetaFont aficionados my interactive version of old is also included.

Keywords
2.5D, art, AFII, ASCII, astroid, Bernstein polynomials, BoundingBox, Bézier cubic, de Casteljau algorithm, ConTEXt, cropping on-the-fly, EPSF, Gabo, hyperboloid, METAFONT, MetaPost, minimal encapsulated PostScript, plain TeX, projection, PSlib, stringed surface, TEXworks

Abbreviations
3D (3 Dimensional), ASCII (American Standard Code for Information Interchange), AFII (Association for Font Information Interchange), ATN (Adobe Technical Note), BB (BoundingBox), CAD (Computer Aided Design), DVD (Digital Versatile Disk), EPSF (Encapsulated PostScript File format), GS (GhostScript), IDE (Integrated Development Environment), IMHO (In My Honest Opinion), LL (LanguageLevel), LRM (Language Reference Manual), PDF (Portable Document Format), PS (PostScript), PSlib (my PostScript library), MF (METAFONT), MP (MetaPost), MS (MicroSoft), PC (Personal Computer), RF (Reference Manual), US (User Space), WWW (WorldWide Web), XPS (XML Paper Specification of MS, a functional subset of PDF), XML (eXtensible Markup Language).

Introduction

Nearly 50 years ago, while on a physics students excursion in the UK, I visited the Tate Gallery. I was captivated by LINEAR CONSTRUCTION IN SPACE No 2 by Naum Gabo.[1] Much later, in the mid 90s, I emulated this object in MetaFont, and Jos Winnink processed my adapted MetaFont code for MetaPost. Earlier, in the 80s, I made a perspex emulation and also did Gabo's TORSION in triplex. The tedious stringing was done together with my youngest daughter. On my WWW of old I even had a quasi-animation of LINEAR CONSTRUCTION IN SPACE No 2. For the EuroTEX&ConTEXt 2009 Jos and I had to repeat the processing of the .mp files, because the .eps files were lost, after 15 years.

Lauwerier(1987) treats the projection of polyhedra, crystals, toroid, sphere with meridians, and ... in BASIC. The present work goes beyond Lauwerier in the sense that projection of planar curves, with circles and ellipses as special cases, is done via B-cubics, in time-proven EPSF.

Of late, I had the inspiration to emulate Gabo's TORSION, see below, in EPSF directly.

I have no access to CAD/CAM software nor Mathematica nor ... to emulate 3D, but ... happily

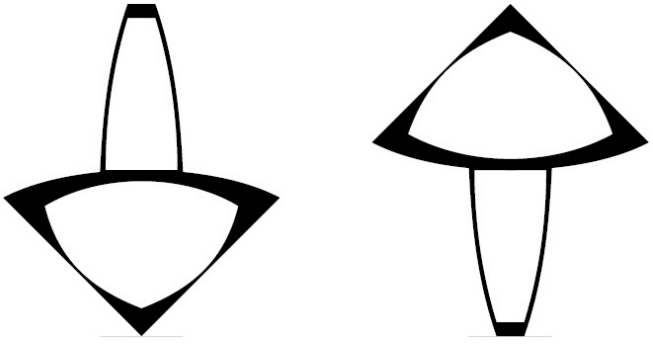
EPSF is a TEXie's graphics companion

For completeness I have included in the Appendices emulations in EPSF of Gabo's LINEAR CONSTRUCTIONS IN SPACE No 1 and 2, as well as one of his `Spheric Themes`, and on the nick LINEAR CONSTRUCTION SUSPENDED.

Analysis of the Torsion object

The construction, a stringed metal frame, consists essentially of 2 identical rectangular isosceles triangles, with a curved hypotenuse and curved inner sides, on top of each other, with the upper triangle turned upside down and rotated over 90°.

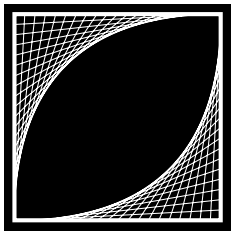
Frame. In the accompanying picture at the left the lower triangle with on top what I call a cap. At the right the upper triangle with the mirrored cap under it, which I call a cup.



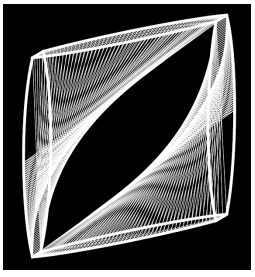
Stringing. The inner curves of the lower triangle are connected by lines to the inner curves of the upper triangle, the strings of the object. Cap and cup have horizontal strings. The stringing yields what I call stringed surfaces.

Stringed surfaces. intrigue me because the impression of a curved surface is obtained from straight lines, which only require 1D information: the boundaries of the frame.

My youngest daughter at primary school in the 70s made already a sort of Gabo in 2D, branches of an astroid, which I emulated.



← 2D stringed surface
with spurious envelope
Gabo's emulated 2.5D →
LINEAR CONSTRUCTION No 1



As appetizer of the use of EPSF the program of the `2D stringed surface` is given, which reflects the essentials and structure of the codes for the emulations of the TORSION, the LINEAR CONSTRUCTION, the SPERICAL THEME, and LINEAR CONSTRUCTION SUSPENDED objects.

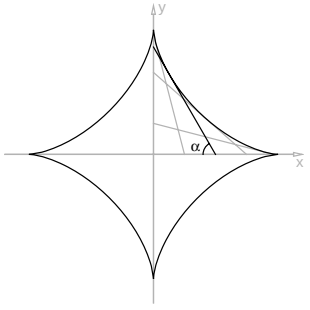
```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -55 -55 55 55
%%BeginSetup
%%EndSetup
%%Title: 2D stringed surface (envelopes are branches of astroid)
%%Creator: Kees van der Laan, kisa1@xs4all.nl
%%CreationDate: Februari 2011
%%EndComments
%%BeginProlog
/reversevideo{-55 -55 110 110 rectfill}def %(Black) Background; Mimics BoundingBox
%
/framestroke{-50 -50 100 100 rectstroke}def
%
/dostringing{0.05 .05 1.01{%for
    /incr exch s mul 2 mul def
    s neg           s neg incr add     moveto
    s neg incr add  s                  lineto
    s neg incr add  s neg              moveto
    s               s neg incr add     lineto
}for}bind def %end dostringing
%%EndProlog
%
%---Program--- the script
%
reversevideo 1 setgray      %black background and white lines, further on
1  setlinejoin 1.415 setmiterlimit 1 setlinecap
2  setlinewidth framestroke %paint frame to the current page
.1 setlinewidth dostringing stroke %paint strings to the current page
showpage %send the current page to the raster device, printer...
%%EOF
```

Note. Because I made use of `rectstroke` in the `framestroke` procedure the `setlinewidth` must precede `framestroke`. In all emulation programs rounded line joins and line caps were asked for and spikes were suppressed by the settings as given in the example program.

Equation of the envelope? Let us spend a little time on refreshing our analytic geometry.

> If a family of curves $f(x,y,\alpha) = 0$, parameterized by α, is tangent to a plane curve E then the curve E is the envelope of the family of curves.

An envelope E is characterized by $\begin{pmatrix} f(x,y,\alpha) \\ f_\alpha(x,y,\alpha) \end{pmatrix} = 0$. Suppose our family of curves are the straight lines of unit length intercepted by the x and y axis, which make an angle α with the x axis: $\frac{x}{\cos\alpha} + \frac{y}{\sin\alpha} = 1$.



$$\begin{pmatrix} f(x,y,\alpha) \\ f_\alpha(x,y,\alpha) \end{pmatrix} = 0 \quad \rightarrow$$

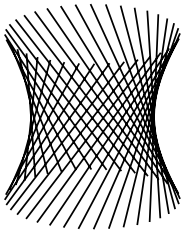
$$\begin{pmatrix} \frac{1}{\cos\alpha} & \frac{1}{\sin\alpha} \\ \frac{\sin\alpha}{\cos^2\alpha} & -\frac{\cos\alpha}{\sin^2\alpha} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \rightarrow$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos^3\alpha \\ \sin^3\alpha \end{pmatrix}$$

Elimination $\cos\alpha$ and $\sin\alpha$ in the equation for the line $\frac{x}{\cos\alpha} + \frac{y}{\sin\alpha} = 1$ yields $x^{2/3} + y^{2/3} = 1$, an astroid, also known as hypocycloid with 4 branches.[2]

My daughter's construction was made differently, not based on a 'unit length intercepted by the x and y axis'. But ... a similar reasoning as above yields the equation $\sqrt{x} + \sqrt{y} = 1$; I'll call her construction: astroid alike.

A stringed surface with known equation. If we rotate a line l around the z axis we obtain a stringed surface. It is a stringed surface because another way to construct it, is starting from 2 horizontal circular frames and connect points of the circumferences by straight lines.



Let l be given by $\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1-t \\ t \\ 2t-1 \end{pmatrix} \quad t \in [0,1]$

Squaring and eliminating t yields

$2(x^2 + y^2) - z^2 = 1$, a hyperboloid

Courtesy: Lauwerier(1987)

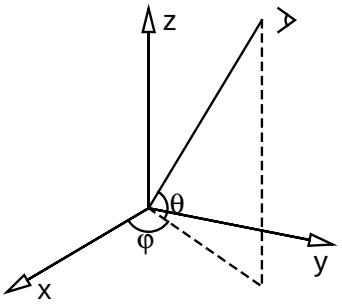
Gabo's stringed surfaces are too complex to be described in equations.

Projection

At high school I learned in the stereometry class to draw projections in the spirit of the drawing in the MetaFont book p113 ex13.7, and as detailed in Lauwerier(1987) Ch3.

Nowadays, because of the ubiquitous PCs, emulation of a 3D object is generally done by (oblique parallel) projection, i.e. the object is viewed under (φ, θ), the azimuth and inclination (Azimuth φ is the angle between the *x* axis and the view direction. Inclination θ is the angle between the view direction and the *xy* plane. In the picture φ and θ are positive.)

Moving the view direction over φ is the same as the object rotating over $-\varphi$, be aware of confusing the two. Trivial, ubiquitous examples of projection are photographs. Lauwerier makes the comparison: the shadow of a frame of wire, as e.g. in a sun dial. The projection plane is orthogonal to the view direction.



right-screw
coordinate system
with view direction φ,θ
in the main octant

In order to project a 3D curve the brute force method is to sample the curve and connect the projected samples by `linetos`. This is done in the SPHERICAL THEME emulation, see Appendix 5. In the LINEAR CONSTRUCTION IN SPACE No 2 emulation, see Appendix 4, the symmetry of the frame is used. In general when we approximate a curve by B-cubics we can project more efficiently.

Properties of (oblique parallel) projection

- straight lines remain straight lines (2 points on the line are sufficient for projecting the line) with preserved ratios
- circles become ellipses (for efficient projection see Appendix 1)
- Bézier cubics (as used in PostScript) are 'invariant' (see Appendix 1)
- angles are **not** preserved, i.e. parallel projection is not conformal
- shape of curves are in general **not** preserved; projections are done by projections of B-cubic approximations or by projection of sample points.

For drawing 3D objects in PostScript I discern the following spaces

- PostScript's Device Space and User Space
- the mathematical 3D User Space, the data.

3D data are projected on 2D PostScript US with the paths constructed in the projection plane. The result I call a 2.5D image.

The projection formula (see Appendix 0 or Lauwerier(1987) p48 (3.7)) reads

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -\sin\phi & \cos\phi & 0 \\ -\cos\phi\sin\theta & -\sin\phi\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

with $(x, y, z) \in \mathbb{R}^3, (u, v) \in$ Projection plane.

For the accompanying illustration of the coordinate system the view direction $\phi = 35^\circ$ $\theta = 20^\circ$ was used.

A feeling for the formula can be obtained by imagining special cases, such as: $\phi = 0$, $\theta = 0$: view of yz plane, i.e. (1,2,3) → (u,v) = (2,3), etc.

On the occasion of the joint EuroTEX&ConTEXt 2009, I programmed the projection in PostScript as procedure with name `ptp`, mnemonics pointtopair.

```
/ptp{% point x y z ==> u v, the projected point
       % example of use: /pair { x y z  ptp } def  %with deferred execution
       % parameters: phi, theta: the viewing angles azimuth and inclination
       % xyz coordinate axes: x to you, y right, z up, right-screw
   ptpdict%push ptpdict on the operand stack
   begin  %move the ptpdict dictionary from the operand stack to the d-stack
```

```
   /z exch def/y exch def/x exch def
   x phi sin mul neg               y phi cos mul add
   x phi cos mul theta sin mul neg y phi sin mul theta sin mul sub
                                                    z theta cos mul add
   end% pop the current dictionary, ptpdict, from the d-stack
} bind def
/ptpdict 3 dict def % create dictionary with name ptpdict
```

The use of `ptpdict` is paramount with such common (local) variable names, like `x`, `y` and `z`. The 'global' parameters ϕ, θ must be initialized in the dictionary, to avoid mix up and let them behave as locals to the procedure `ptp`.

Hidden lines.
A side effect in projection are lines which should not be visible, the so called hidden lines. I use one colour throughout with the pleasing result that I don't have to worry about hidden lines. In Gabo's transparent perspex objects everything is visible.

In the current object one might handle hidden lines by a judiciously chosen printing order of the elements, with the triangles split longitudinally along the axis. This works for $\phi \in [0{,}90°]$, $\theta \in [-90°{,}90°]$.

Emulation

My first emulation of TORSION was in 3D with the frame of triplex in the late 80s. Two sides are broken after so many years.

My emulation of LINEAR CONSTRUCTION IN SPACE No 2, in perspex stringed by nylon filament, has not stand time, which also has happened to some of Gabo's objects. The MF emulation and the reverse video picture is published in MAPS 16-28; the picture is also included in the LaTeX Graphics companion. I improved the MetaFont emulation on the occasion of the joint EuroTeX&ConTeXt 2009, by thinner strings and showing all the strings, also the no longer 'blurring' hidden ones.

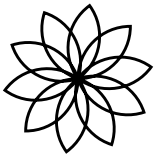
Why in PostScript?
The use of a programming language has two aspects: the richness and power of the language proper and the ease of working in an IDE.

PostScript language. The language was designed by Adobe, and introduced in 1985, as a device-independent page description language with powerful graphics capabilities, ≈ 400 operators — i.e. built-in procedures of the system dictionary — in PostScript level 1, with substantial more in PS 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, 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 version 3 is the defining document for the syntax and semantics of the language, the imaging model, and the effects of the graphics operators.

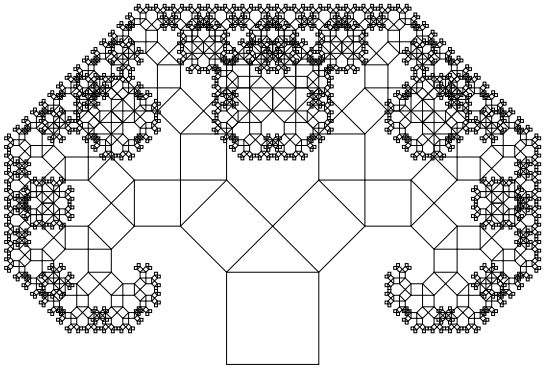
Powerful concepts:

- 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 Pythagoras 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
```



Pythagoras 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 level 2 of 1991, and elaborated upon in 1997 in PostScript level 3, with among others much more efficient shading functionality. In PostScript level 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
Note, however that
`%%BeginSetup`
`%%EndSetup`
are not necessary???
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` `selectfont` ...
  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 used more than once.
- 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 XƎTEX) 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`[3]
  — dictionary `dict ...dictstack`
  — string `string ... run ... token`

- relation, boolean, and bitwise `eq ... bitshift`
- type, attribute, and conversion `type ... cvs`
- file `file = == ... echo`
- 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 concatination 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
`moveto`	creates the starting point of an (internal) path
`stroke`	and ilks, to paint the path to the current page
`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, also used in my BachoTEX 2010 pearl.

The language 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 3rd parties, especially the interpreters which come with your printer.

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 printposter DrawPieChart centerdash smallcaps` and `arrow`, to

name but a few.[4] 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 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 is useful.

Although a powerful graphical language, PostScript is considered low-levelish by the TEX community at large. They favour John Hobby's preprocessor MP, 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 MetaPost 2.000.

> '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 understands 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:

- `(` 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

```
%!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.[5] 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.

PostScript IDE. On the WWW I found the nonfree `PSAlter's workbench`, a one-window PostScript IDE. In the PostScript FAQs it is mentioned that they provide visual debugging.

I use an editor for creating the source and transform .eps into .pdf by Acrobat Pro 7.1, with 2 windows open:

- the editor
- the map where the PostScript file resides (right mouse click the .eps file yields a pop-up menu with option convert to PDF, which converts the .eps into .pdf and opens the resulting .pdf in my Acrobat Pro 7.1; the %stdout is written to the messsages.log file in the map of Distiller.)

For me Acrobat is an excellent present day interpreter for daily use on my PC. In the mid 90s I used the (black and white) Apple Laser writer.
Those who favour Ghostscript might find the overview http://www.ghostscript.com/doc/7.07/Readme.htm, interesting. GSview does not allow the use of the run operator, apparently, which I use to include my library file.

I tried in TEXworks the preferences option and added Acrobat Pro. Opening in the edit window .eps and processing it by pushing the newly created Acrobat button did open the Acrobat window, yes, but ... empty, without the results. What did I do wrong?

A standing wish: Jonathan Kew, or somebody else, do come up with a nice PostScript...MP IDE similar to the TEXworks IDE, with its handy jumping from a line in the edit window to the corresponding place in the pdf window, and vice versa, its dictionaries (to be added by the user), and ... its promising scripting facilities.

I'm not aware of a MF, respectively MP, pleasing IDE.[6] But ... TEXies can choose for ConTEXt processing in TEXworks, with embedded MetaPost processing on the fly.

The main reasons for me to program in PostScript are:

- I like the powerful graphics facilities
- it is well-suited for my purposes, and ...
- I want to experience once more whether it is more difficult to write in my use of PostScript, which I call minimal PostScript, than in MF or MP, in analogy to minimal markup in TEX versus e.g. the use of LaTEX.[7]

For stringed surfaces emulation PostScript lacks MF's, and MP's inherited, point of operator. No problem, I added the procedure tOnSpline, see later.

A definitely nice feature of MetaFont, respectively MetaPost, I came across while working on this note, is the flexibility of the path specification. In PostScript only B-cubics can be added to the current path, obeying the rigid specification of control points. B-cubics are not provided for by just specifying a set of points lying on the polynomial. But ... with a little bit of Math ... maybe, sometime... someday...

The advantage of working in PostScript directly is that viewing PostScript is a 1-step process, while MP, the TEXworld's PostScript preprocessor, requires 2-steps. Moreover, the resulting .eps from MP is less intelligible and verbose. For example LINEAR CONSTRUCTION IN SPACE No 2 in MP, took 120 lines with the resulting PostScript 800+ lines (mainly caused by unwinded loops). I expect the hand-written PostScript to take an odd 100 lines, roughly of equal size and just as intelligible as the MP version, see Appendix 4.

PostScript output from Adobe Illustrator, which I have used for converting .jpg into .eps, is really unintelligible: lots of preceding obscure definitions, but ... one can use the 'meat' in it as encapsulated PostScript. I'll come back on the matter another time. For the impatient, see Acumen J. Nov 2004, or better still just use Photoshop, or ... for the conversion.

The super reduction in processing steps, is provided for in Hans Hagen's ConTEXt, where MP pictures are processed on the fly, while TEXing. The only thing I miss

is that he does not explain how he accomplished his tricks, for example shading and transparency. At the BachoTEX2010 Taco Hoekwater showed the inclusion of PostScript in LuaTEX, exciting. Note that inclusion of `.eps` in pdfTEX is not allowed.

My minimal use of EPSF. In the program, given in Appendix 2, I made use of the PostScript operators
`closepath curveto def eofill exch fill lineto moveto run setlinewidth setdash showpage translate`
next to the self-written procedures
`s ptp tOnSpline`.
Not so much isn't it?

Choice of curves

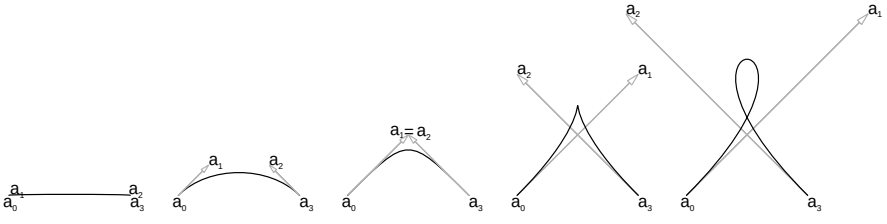
For each hypotenuse, for each of the inner sides of the triangles and for each of the legs of the cap and cup, I use just one B-cubic. The (control) points which characterize the spline have to be chosen, judiciously.

Splines

In PostScript a spline, a Bézier cubic, is characterized by the begin point a_0, the control points a_1 and a_2, also called handles, and the end point a_3. In Java one can drag the handles and watch the effects, interactively. Nice.

> *Splines are the important 20^{th} century's Math time-dependent functions, comparable to the 19^{th} century's Fourier series Math for approximations.*

The control point `a1` lies on the tangent to the spline in a0, and the control point `a2` lies on the tangent to the spline in a3. The points `a0` and `a3` and the angles of the tangents to the spline at these points are not enough to describe the spline uniquely: the size of the handles also matters, as explained in Manning(1972).[8] The control points stand for the angle of the tangents and the size of the handles and therefore determine uniquely the spline.



With `a0` as `currentpoint` a B-cubic, characterized by `a0, a1, a2, a3`, is appended to PostScript's (internal) `path` by `a1 a2 a3 curveto`.

Mathematical formula of a spline. Splines as used in PostScript, and in MP, are Bézier cubics. (Adobe in the Red Book of 1985 mentions along with the operator `curveto` that Bézier cubics are added to the `currentpath`).

These 3^{rd} degree polynomials are a linear combination of 3^{rd} degree Bernstein basis polynomials, which were discovered in 1912 by Bernstein. A n^{th} degree Bernstein basis polynomial reads $B_{\nu n}(t) = \binom{n}{\nu} t^{\nu}(1-t)^{n-\nu}$, $\nu = 0,1,...,n$.

A Bézier cubic — a linear combination of 3^{rd} degree Bernstein basis polynomials — with begin point a_0, control points a_1, a_2, and end point a_3 reads

$$\begin{aligned} z(t) =& (1-t)^3 a_0 + 3(1-t)^2 t\, a_1 + 3(1-t) t^2\, a_2 + t^3 a_3 \\ & \text{with} \quad z, a_0, a_1, a_2, a_3 \in \mathbb{R}^2,\ t \in [0,1] \end{aligned} \tag{1}$$

The common representation for evaluation, apart from the Horner scheme, reads

$$z(t) = A\, t^3 + B\, t^2 + C\, t + D \quad \text{with} \quad z,A,B,C,D \in \mathbb{R}^2,\ t \in [0,1] \tag{2}$$

The coefficients A,B,C,D are linear in a_0,a_1,a_2,a_3

$$\begin{aligned} A &= \ a_3 - 3a_2 + 3a_1 - a_0 \\ B &= 3a_2 - 6a_1 + 3a_0 \\ C &= 3a_1 - 3a_0 \\ D &= \ a_0. \end{aligned}$$

The above is implemented in `tOnSplineclassic`, which yields the point of the spline, characterized by `a0, a1, a2, a3`, for the time variable `t` by the Horner scheme.[9] PostScript only paints the paths or regions to the current page, it does not provide an operator for evaluating the B-cubic.

De Casteljau's algorithm. Bogusław Jackowski drew my attention to this algorithm for evaluating B-cubics, which is (more generally) discussed on `http://en.wikipedia.org/wiki/De_Casteljau's_algorithm`.
Formula (1) can be written in the de Casteljau representation

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

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

$$\begin{matrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{matrix} \rightarrow \begin{matrix} a_{01} = a_0\,(1-t) + a_1\,t \\ a_{12} = a_1\,(1-t) + a_2\,t \\ a_{23} = a_2\,(1-t) + a_3\,t \end{matrix} \rightarrow$$

$$\rightarrow \begin{matrix} a_{012} = a_{01}\,(1-t) + a_{12}\,t \\ a_{123} = a_{12}\,(1-t) + a_{23}\,t \end{matrix} \rightarrow a_{0123} = a_{012}\,(1-t) + a_{123}\,t$$

Implementation of de Casteljau's algorithm by Bogusław Jackowski and Piotr Strzelczyk for evaluating a spline at point t.

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

/tOnSpline{{% Purpose: t on spline a0,a1,a2,a3 ==> x y
% implementation of the De Casteljau's algorithm
%
% t: value in [0,1]
% a0 a1 a2 a3: point pairs which characterize the spline
%
% in MF lingo: given pairs a0, a1, a2, a3, and a real number t, 0<=t<=1;
%              we want to compute
%    t[ t[ t[a0,a1],t[a1,a2]],t[t[a1,a2],t[a2,a3]]]
% ==>
```

```
% x(t) y(t)
tOnSplinedict % look up the name and push the dictionary on the operand stack
begin % and move tOnSplinedict dictionary from the operand stack to the d-stack
/a3y exch def /a3x exch def
/a2y exch def /a2x exch def
/a1y exch def /a1x exch def
/a0y exch def /a0x exch def
/t exch def
/a01x a0x a1x t mediation def    /a01y a0y a1y t mediation def
/a12x a1x a2x t mediation def    /a12y a1y a2y t mediation def
/a23x a2x a3x t mediation def    /a23y a2y a3y t mediation def
/a012x a01x a12x t mediation def /a012y a01y a12y t mediation def
/a123x a12x a23x t mediation def /a123y a12y a23y t mediation def
   a012x a123x t mediation       a012y a123y t mediation
end% pop the tOnSplinedict dictionary off the d-stack
}def
/tOnSplinedict 20 dict def%create dictionary with name tOnSplinedict
```

Note. The created variables are 'local', i.e. only made available in the dictionary `tOnSplinedict`, which is popped off the d-stack on leaving the procedure. All important for library procedures.

In MP splines can also be characterized by the begin point, end point, and the tangents at these points (with a wired-in assumption of the smoothness). Even the concept tension, inherited from MF, is implemented such that wired-in smoothness can be overruled.

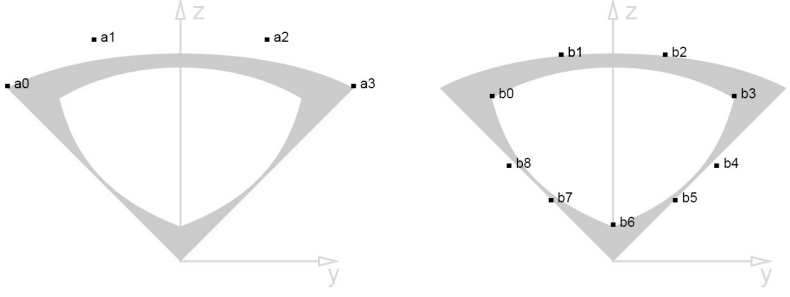
Classically a 3rd degree scalar polynomial

$$P_3(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

is specified by 4 points of the polynomial, or by 2 points and 2 derivative values. The coefficients of the polynomial, a_0,a_1,a_2,a_3, are obtained by solving a 4 x 4 system of linear equations. The latter can be done in MetaPost (and MetaFont) on the fly, albeit without paying attention to numerical stability, i.e. without pivoting. I have not written a numerical stable 4 x 4 linear equation solver in PostScript ... yet; 3 x 3, and 2 x 2, yes; they are included in my PSlib library.

Choice of the data

Choose as origin the centre of the foot. Locate the lower triangle in the yz plane. For the height of the object I decided on 60 pts (nearly an inch), meaning $\mathrm{top} = (0,0,60)$, which entails that the height of the triangle is 30 pts. Below at left the points which characterize the outer hypotenuse, at right the points which characterize the inner curves.



For the length of the legs of the triangles I took $25\sqrt{2}$ pts, meaning $a_0 = (0,-25,25)$, and $a_3 = (0,25,25)$, i.e. $-a_y = a_{3y} = 25$.

For the y coordinates of the control points a_1 and a_2, I chose the values -12.5 pts, respectively 12.5 pts. The z coordinates of a_1 and a_2 follow from the equation (2) and that the hypotenuse should touch (0,0,30)

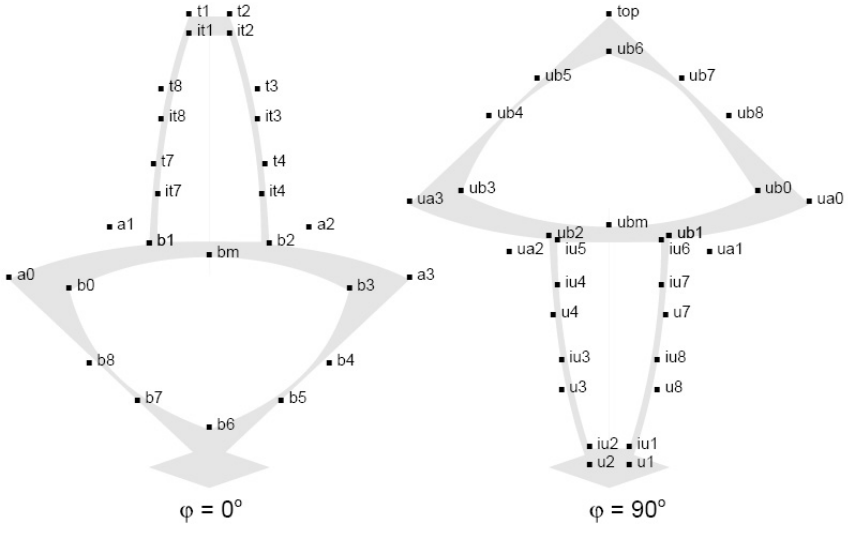
$$z(.5) = \frac{A_z}{8} + \frac{B_z}{4} + \frac{C_z}{2} + D_z \quad \text{with} \quad z(.5) = 30$$

which yields

$$a_{1z} = a_{2z} = (4z(.5) - a_z)/3 = 31\tfrac{2}{3} \rightarrow a_1 = (0, -12.5, 31\tfrac{2}{3}), a_2 = (0, 12.5, 31\tfrac{2}{3}).$$

The data points for the inner triangle, $b_0, \ldots b_8$, must be chosen such that the curves are pleasing and symmetric around the axis of the object; not critical. The data points for the upper triangle are rotated and mirrored values of those of the lower triangle.

I chose for cap and cup data point values which yield curves which looked pleasing to me. Visually, the data points are given in the figure below (for the values see the program in Appendix 2)



By the way, the dots and names above are printed by the following, interesting, mean and lean PostScript code, where the names of the projected points are supplied as strings in an array on the stack. The strings are used as such by show and converted to the name of the data points, which they represent.

```
/dotsandnames{%[ str,..., str]==>
%in the str a name, which stands for a pair, such as in pair moveto
%centershow is my centred show, H10pt stands for Helvetica 10 pts
%already scaled
{dup cvn load exec moveto (.) H10pt setfont centershow
                              H12pt setfont show}forall
}def
%invoke
[(a0) (a1) (a2) (a3) (top)... ] dotsandnames
```

Explanation line by line: In a `forall` loop each string is duplicated (remember that each string in the array is put on the stack in turn, in the order as given in the array, which is different from other languages, where parallel execution of the elements of the `forall` loop is allowed, be aware!) after which one copy is converted into a name, loaded and executed to yield the coordinates of the data point for the positioning, and the dot is (centered) painted by `centershow` to the current page; next in the loop the duplicated string is painted by `show` to the current page. Neat!

But ... sometimes the names can better be shown left, as in the frame of LINEAR CONSTRUCTION IN SPACE No 1, see Appendix 3.

Implementation I chose for each projected point a definition, which for `top`, for example, reads

```
/top { 0 0 60 s ptp} def% s=scaling, analogue to the use of 60 inch
```

The advantage is mean and lean correct code with deferred execution. The disadvantage is that each point is evaluated each and every time when needed. For my toy problems, on nowadays PCs, this is not relevant; clear, intelligible, and correct code is more important than execution speed.

A more efficient, but verbose, version reads

```
0 0 60 s ptp /topy exch def /topx exch def   /top { topx topy } def
```

The points named by `a0, a1, a2,...` in the program denote the projected spacial data a_0,a_1,a_2,... . The (projected) triangles are called `lowertorsiontriangle` and `uppertorsiontriangle` and implemented as follows

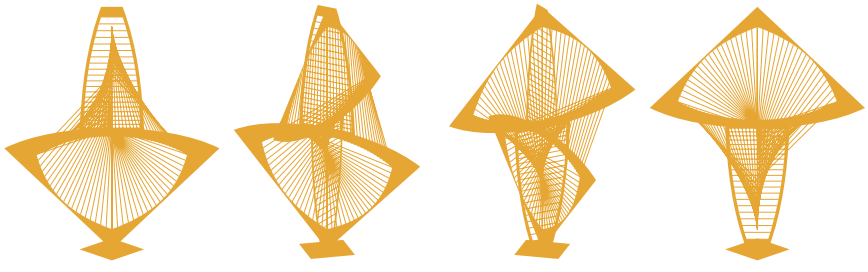
```
/lowertorsiontriangle{origin moveto  %outer path
       a0 lineto
       a1 a2 a3 curveto closepath
       b6  moveto                    %inner path
       b7 b8 b0 curveto
       b1 b2 b3 curveto
       b4 b5 b6 curveto closepath
} def
```

Stringing

Connecting points of the splines by straight lines yields the stringing, the stringed surfaces. The points on the spline are obtained by invokes of `tOnSpline`, see the procedure `dostringing` in the program given in Appendix 2.

Results

Below the emulations seen under azimuths 0°, 30°, 60°, 90°, and inclination 20°



Variations

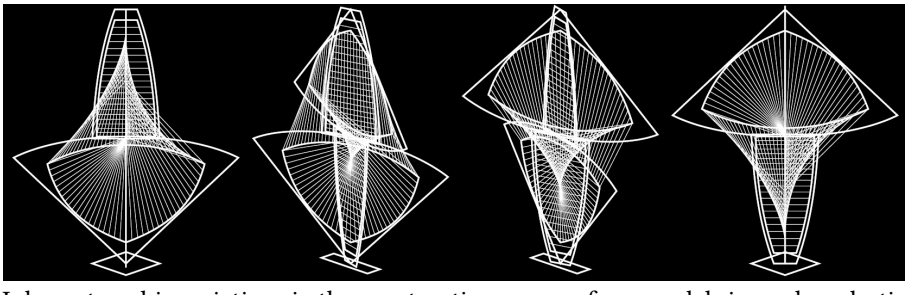
Gabo himself constructed variations, where the sides of the triangles are connected to the outer sides of cap, respectively cup. This can be coded easily, but I expect it to blur the picture by too many lines.

On closer inspection I noticed that in the centre Gabo had a horizontal metal square width sides of 10 cm or so. The hypotenuses were split in the middle and connected to the corners of the square.

Other emulations may use circular arcs as inner sides of the triangles (actually this was the case in the real thing).

On the other hand, one may vary by using curves instead of lines for the sides of the triangles.

Below the results of the outline version in reverse video, a fibre figure, which I consider beyond Gabo, because in the emulations I don't have to worry about construction necessities, although they much resemble his LINEAR CONSTRUCTIONs in perspex.



Inherent are his variations in the construction process from models in card or plastic to real objects made from wood, metal, perspex, plastic, ... of varying size.

Gabo's ultimate variation on the Torsion theme is his fountain in front of the Thomas hospital in London, opposite the Big Ben on the other bank of the river Thames.

In this kinetic art fountain the 'stringed' surfaces change, because the strings are water jets squirted under slowly varying pressure. The revolting time is 10 minutes.

In the beginning of this century I got an idea how to emulate this fountain (material: white curved plastic electricity tubes and a rotating lawn sprinkler) for my quarter circle garden pond.

I guess that the construction principle differs: my lawn sprinkler revolves on a water bed and due to the action=reaction principle.

Conclusions

Though PostScript lacks an explicit path data structure the TORSION object was easily emulated. The points of a Bézier cubic are calculated by the de Casteljau algorithm implemented by Bogusław Jackowski and Piotr Strzelczyk. Such an operator is not provided for in PostScript.

The use of ready-made EPSF procedures is not difficult at all, as can be witnessed from the included programs; to write robust EPSF procedures is a different story. A library of correct, intelligible EPSF procedures is what we need.

Bézier cubics are invariant under (oblique parallel) projection, which is according to Bogusław Jackowski obvious.

To program illustrations in PostScript is no more difficult than to program in MP, or MF, but ... one has to understand the concepts, thoroughly. For me the use of PostScript is easier than the use of MP.

Starting an EPSF program with

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox ...
%%Beginsetup
%%Endsetup
```

yields `.pdf` pictures, cropped to the specified BoundingBox, centred.

The handling of hidden lines is circumvented.

The use of `pdf optimizer` of Acrobat Pro 7.1 resulted in a ≈ 30 times smaller `.pdf` file of this note.[10]

In TEXworks' editor window I use the monotype Lucida console font, where all characters have the same width (it is default in MS Wordpad) in order to keep verbatim texts vertically aligned, but it did not work as expected; 16 pts is a convenient magnification for me. Bookman Old Style (gave good results with respect to alignment in verbatims) and Baskervile Old Face are also monotype and pleasant for the eyes. The pdf viewer used by TEXworks is not so good as Acrobat: sometimes annotations in my illustrations don't show up.

Never throw data away, even after 15 years they may still be of value; beware when you replace your computer.

TEXing this document required little, minimal plain TEX markup. Interesting, although not suitable for full-automatic typesetting with its shrinking and stretching, its automatic splitting into pages, ... is the inclusion of a picture next to the verbatim listing. (A practice borrowed from the Blue Book. I did not use Phil Taylor's `\parshape` preprocessor for flowing text around illustrations, presented at BachoTEX 2009, because I was only after placing illustrations in the open space at the right of (narrow) verbatim listings.) A way how to do this is essentially explained in

the TEXbook on p389, which I implemented in the macros `\insertjpg`, respectively `\insertpdf`. The TEXnique I used earlier in my plain TEX Turtle Graphics macros, which are not suited for graphics in TEX: EPSF is much better and more general for the description of (single) illustrations, to be included in AnyTEX documents. I used the wrong tool for my graphics: TEX as American screwdriver, to paraphrase A. Perlis.

When I think I'm finished it turns invariably out that I'm only halfway.

I don't expect you, kind reader — who does not suffer of the disease of our times: lack of time — to come up with a Gabo of your own (not that difficult just design a frame and do the stringing), but ... you may profit from the projection technique, the examples of EPSF programming, the communicated experiences, and hopefully you have become familiar with some of Gabo's works and last but not least enjoy the illustrations.

Æsthetics and effectiveness of the message, cultural contexts? My message: PostScript can be used gracefully, æsthetically, and effectively to emulate a class of Gabo's objects, and undoubtedly some more. TEXies should catch up.

Wouldn't it be nice to have holographic 3D projections of Gabo's constructions?

Afterthoughts

The projected frame resembles a font character. Would the program, or me in writing the program, have benefited from 'subscripting' more in the spirit of MF's general `suffix`? I might have used a_i, inner a, instead of `b` and subscripted it by numbers: i.e. have used variables `a0i, a1i,...` instead of `b0, b1, ....` I did do this for cap and cup and followed Knuth's convention in LINEAR CONSTRUCTION IN SPACE No 1 to begin with, see Appendix 3.

A standing wish An IDE for PostScript, and MP, in the spirit of TEXworks I would welcome. Why not distribute it on the TEXlive DVD?

Daydreaming If only I could handle gradients in PostScript gracefully, I might emulate Gabo's CONSTRUCTION HEAD No 3 (Head in a Corner Niche, which I watched in MOMA, NY) although it has a much different character than the stringed surfaces. Gradients I do for the moment by the PostScript level 1 technique. In PS3 really efficient shading gradients are accounted for. Quite substantial, looks complex. Work to do!

Further reading

Instead of a bibliography a selection which I experienced useful:

- Adobe's Red, Green and Blue books, an absolute must.
  `http://www-cdf.fnal.gov/offline/PostScript/`
  From the preface of the Green Book: PostScript Language Program Design is intended to provide a solid background for developing software in the PostScript language — not just getting a program to work, but thoroughly designing it from top to bottom ...
  ... The sample programs contained in each chapter are intended to be directly usable. They are examples, but they are not trivial ones. Each program has been

carefully designed and debugged and should provide an excellent foundation for an industrial-strength printer **driver** ...
Disclaimer: For EPSF usage not so appropriate.

- Adobe PostScript 3 Fonts Set ... come standard with 136 distinctive and stylish fonts, including those packaged with the leading operating systems...
Disclaimer: I did not succeed in finding these fonts in Acrobat Pro 7.1.
Puzzling message to `%stdout`*: Helvetica not found, using Font Substitution. Font cannot be embedded.*
- Digital Acumen Journal: `http://www.acumentraining.com`, by John Teubert, for free. Top!
- Adobe Technical Note #5002, Encapsulated PostScript File Format Specification `http://partners.adobe.com/public/developer/en/ps/5002.EPSF_Spec.pdf`
- Don Lancaster's PostScript use: `http://www.tinaja.com`, for free. Informative.
- PostScript FAQ `http://en.wikipedia.org/wiki/Wikibooks:PostScript_FAQ`

On the other hand, I myself have published in MAPS a few articles on, cq related to, PostScript:

- Just a little bit of PostScript, MAPS 17, contains nice, if I may say so, examples you don't see that often.
- Stars around I & Stars around II, MAPS 18, after Jackowski's MetaFont lectures. Jacko's example of the creation of the 2 character `OK` font is much in the spirit of the creation of an analytic font in PostScript as given in the Blue Book Program 20.
- Tiling in PostScript and MetaFont—Escher's wink MAPS 19.
- TEX education—a neglected approach MAPS 39. Presented at the EuroTEX&ConTEXt 2009 and rehearsed at the BachoTEX 2010.
- Circle Inversions MAPS 40, where I introduced among others my `PSlib`.
- à la Mondriaan MAPS 41. Presented at BachoTEX 2010.
- Programming Pearls 2010: π-decimals along a spiral and `The mouse's tail and Alice's tale'.
- Programming Pearls 2011: Pythagoras Tree.

For the Math:

- Lauwerier, H.A.(1987):[11]Meetkunde met de Microcomputer. Epsilon 8. (Projection, hidden lines, projection of crystals, inside-outside, sphere with meridians, toroid ... . However ... no projection of B-cubics. How to project circles and ellipses efficiently was developed while working on this note.)
- Manning J.R.(1972): Continuity conditions for spline curves. Computer Journal, 17,2, p181-186.

For those who want to pass by (La)TEX and do it all in PostScript might find the following interesting
`http://www.cappella.demon.co.uk/bookpdfs/pracpost.pdf`
NTG provides a printing on demand service `http://www.boekplan.nl` for among others copies of MAPS articles.

Acknowledgements

Thank you Naum Gabo for your inspiring art, Hans Lauwerier for your lectures on projection and some more, Adobe for your good old PostScript and `Acrobat` to view it, Don Knuth for your stable plain TEX, Jonathan Kew for the TEXworks IDE, Hàn Thế Thành for your pdfTEX, Bogusław Jackowski for your thorough review, your communicating de Casteljau's algorithm and sending me a copy of Manning's paper,

Piotr Strzelczyk for the implementation of de Casteljau's algorithm, Jos Winnink for comments on an early version of this paper, Henk Jansen for stressing the point to coddle data, which might still be of use after ≈15+ years, Wim Wilhelm for prompting the formula for the curve on the tennis ball, MAPS editors, especially Frans Goddijn, for improving my use of English, and Taco Hoekwater for suggestions and procrusting this note into MAPS format.

Notes

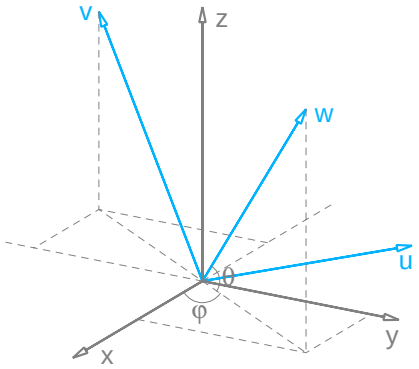
1. Naum Gabo, 1890–1977. Born Naum Borisovich Pevsner. Bryansk. Russian Constructivist. An excellent book about him and his works: Naum Gabo 60 years of Constructivism. Prestel-Verlag 1985, which appeared on the occasion of the retrospective exhibition with the same name at the Dallas Museum of Art, the Art Gallery of Ontario, the Guggenheim Museum NY, the Akademie der Künste Berlin, the Kunstsammlung Nordrhein-Westfalen, the Tate Gallery London. Wikipedia contains a short biography.
2. Courtesy: Courant, R (1936) Differential and Integral calculus II, p175. The astroid was already discussed by J. Bernouilli in 1691.
3. Note that there is no automatic type conversion. For example the index of an array `a` must be of type integer eventually converted by `cvi: a <real expression> cvi <value> put`. This also holds for the integer operands of `idiv`.
4. 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.
5. An undocumented feature? Non-universal?
6. On my PowerMac and Mac Classic of ≈ 1992 (still alive, mind you!), I could view MF pictures on the screen in Blue Sky's MF. I don't have that functionality on my PC. No longer relevant because the resulting bitmaps are outdated.
7. Obeying the famous 80-20% adage: 20%, or less, of the needed energy with 80%, or more, of the results. It is true that LaTeX comes with a wealth of packages. John Hobby has donated the powerful `boxes` for flowcharts, and `graph` for drawing graphs in MP.
8. This is different from a scalar (3^{rd} degree) polynomial $P_3(x)$, where 2 values and 2 derivatives determine the polynomial uniquely.
9. Well-known time representations of curves are the Lissajous figures defined by (A sin(at+δ), B cos(bt)).
10. Courtesy Péter Szabó, EuroTeX 2009.
11. For Lauwerier's biography see the general site about Dutch mathematicians: `http://bwnw.cwi-incubator.nl/cgi-bin/uncgi/alf`

My case rests, have fun and all the best.

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

Appendix 0 Projection formula

Let us choose a Cartesian orthonormal (projection) coordinate system uvw, with the w axis in the direction (ϕ,θ), the v axis in the wz plane orthogonal to the w axis. Because the v and w axes are in the plane wz the u axis is in the xy plane, with direction according to the right-screw rule.



uvw coordinate system
with $\mathbf{e}_u, \mathbf{e}_v, \mathbf{e}_w$ unit vectors
$\mathbf{e}_w$ in the direction (ϕ,θ) (Puvw)
$\mathbf{e}_v$ in the zw plane, $\mathbf{e}_v \perp \mathbf{e}_w$
$\mathbf{e}_u$ in the xy plane, $\mathbf{e}_u \times \mathbf{e}_v = \mathbf{e}_w$

A point $\mathbf{P}$ can be described in the coordinate systems as

$$\mathbf{P} = x\mathbf{e}_x + y\mathbf{e}_y + z\mathbf{e}_z = u\mathbf{e}_u + v\mathbf{e}_v + w\mathbf{e}_w$$

If the unit vectors $\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z$ are projected in the uvw coordinate system we obtain

$$\begin{aligned}\mathbf{P} = {} & x(a_{11}\mathbf{e}_u + a_{21}\mathbf{e}_v + a_{31}\mathbf{e}_w) + \\ & y(a_{12}\mathbf{e}_u + a_{22}\mathbf{e}_v + a_{32}\mathbf{e}_w) + \\ & z(a_{13}\mathbf{e}_u + a_{23}\mathbf{e}_v + a_{33}\mathbf{e}_w) \\ = {} & u\mathbf{e}_u + v\mathbf{e}_v + w\mathbf{e}_w\end{aligned}$$

Rearranging terms and equating coefficients yields the change of coordinates: $(x,y,z) \to (u,v,w)$, which in matrix notation reads

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Deleting the w component yields the projection coordinates in the uv plane, the projection plane. Substituting the values for a_{ij} in terms of ϕ and θ yields the projection

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -\sin\phi & \cos\phi & 0 \\ -\sin\phi\cos\theta & -\sin\phi\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Properties

- because of the orthonormal coordinate systems the coefficients of the matrix A obey the relations

$$\sum_{k=1}^{3} a_{ik}a_{jk} = \delta_{ij} \begin{cases} 0 \text{ if } i \neq j \\ 1 \text{ if } i = j \end{cases}$$

□ because a similar reasoning can be applied to transforming the uvw system into the xyz system, the inverse matrix equals the transposed matrix, i.e. $A^{-1}=A^t$.

Factorization of the projection matrix

$$\begin{pmatrix}1&0&0\\0&1&0\end{pmatrix}\begin{pmatrix}1&0&0\\0&\sin\theta&\cos\theta\\0&-\cos\theta&\sin\theta\end{pmatrix}\begin{pmatrix}-\sin\phi&\cos\phi&0\\-\cos\phi&-\sin\phi&0\\0&0&1\end{pmatrix}=\begin{pmatrix}-\sin\phi&\cos\phi&0\\-\cos\phi\sin\theta&-\sin\phi\sin\theta&\cos\theta\end{pmatrix}$$

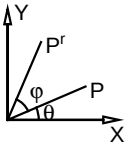
The left factor is rotation over $\theta-\frac{\pi}{2}$; the other factor

$$\begin{pmatrix}-\sin\phi&\cos\phi&0\\-\cos\phi&-\sin\phi&0\\0&0&1\end{pmatrix}=-\begin{pmatrix}\cos(\frac{\pi}{2}-\phi)&-\sin(\frac{\pi}{2}-\phi)&0\\\sin(\frac{\pi}{2}-\phi)&\cos(\frac{\pi}{2}-\phi)&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\0&0&-1\end{pmatrix}$$

is a composition of inversion of the z coordinate and rotation over $\frac{\pi}{2}-\phi$.

Background: rotation over ϕ in xy plane The coordinates (x',y') of a point $P_{r\,\theta}(x,y)=(r\cos\theta,r\sin\theta)$ rotated over ϕ read

$$\begin{pmatrix}x'\\y'\end{pmatrix}=\begin{pmatrix}r\cos(\phi+\theta)\\r\sin(\phi+\theta)\end{pmatrix}=\begin{pmatrix}r(\cos\phi\cos\theta-\sin\phi\sin\theta)\\r(\cos\phi\sin\theta+\sin\phi\cos\theta)\end{pmatrix}=\begin{pmatrix}\cos\phi&-\sin\phi\\\sin\phi&\cos\phi\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}$$



Lauwerier used the projection matrix $\begin{pmatrix}-39&52&0\\-20&-15&60\end{pmatrix}$ because he was after a pleasant projection, and his μ-computer was not powerful enough to facilitate animation, even in BASIC.

Appendix 1 Projection of curves

Invariance of Bézier cubics under projection

Observation A projection of a Bézier cubic can be obtained by projecting the begin point, the control points, and the end point, and use these to build the (projected) spline.

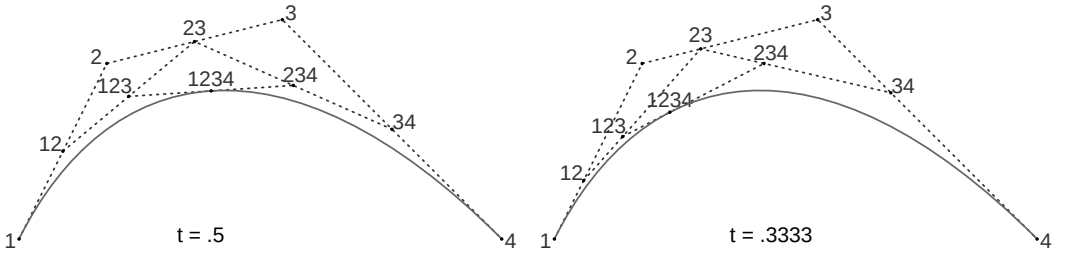
Proof A Bézier cubic is characterized by four points z_1,z_2,z_3,z_4, the begin point, the control points and the end point. A point on the curve, z_{1234} the third-order midpoint, is obtained by

$$\begin{matrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{matrix} \rightarrow \begin{matrix} z_{12} = \frac{1}{2}[z_1, z_2] \\ z_{23} = \frac{1}{2}[z_2, z_3] \\ z_{34} = \frac{1}{2}[z_3, z_4] \end{matrix} \rightarrow \begin{matrix} z_{123} = \frac{1}{2}[z_{12}, z_{23}] \\ z_{234} = \frac{1}{2}[z_{23}, z_{34}] \end{matrix} \rightarrow z_{1234} = \frac{1}{2}[z_{123}, z_{234}]$$

where $\frac{1}{2}[z_1,z_2]$ means the midpoint of the line through z_1 and z_2.

To get the remaining points of the curve, for eample for the time variable $t = \frac{1}{3}$, determined by z_1,z_2,z_3,z_4

repeat the same construction on z_1,z_{12},z_{123},z_{1234} and z_{1234},z_{234},z_{34},z_4, ad infinitum (Courtesy The MF book p13).



If we project the four initial points and construct the polynomial in the projection plane from the projected points, the same curve will result as if we had projected each point of the original polynomial, because the construction lines and their midpoints are invariant. qed.

Note the above shows that Knuth was already aware of what became known as the de Casteljau algorithm for evaluation of B-cubics.

Projection of circles and ellipses
A point on a circle obeys

$$(x,y) = (r\cos t, r\sin t) \quad t \in [0,2\pi] \quad \text{with} \quad x^2 + y^2 = r^2 \quad r \text{ the radius and } (0,0) \text{ the centre.}$$

The above circle is in PostScript constructed as path by `0 0 r 0 360 arc`.
A point on an ellipse obeys

$$(x,y) = (a\cos t, b\sin t) \quad t \in [0,2\pi] \quad \text{with} \quad \frac{x^2}{a} + \frac{y^2}{b} = 1 \quad \text{with } a \text{ and } b \text{ half the axes.}$$

From this we can understand that the an elliptical path can be obtained in PostScript by
`a b scale 0 0 1 0 360 arc` (Courtesy the Blue Book p55, MetaFontbook p123).

The projection (u,v) of a point (x, y, z) on a circle in 3D obeys

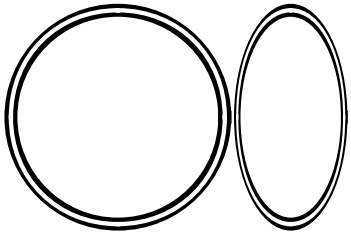
$$(u,v) = (-x\sin\phi + y\cos\phi,\ x\cos\phi\sin\theta - y\sin\phi\sin\theta + z\cos\theta)$$

which does not look like a composition of scaling and/or rotation, but actually it is, as communicated by Bogusław Jackowski

Projection of a circle or ellipse can be done by sampling and connecting the sample points in the projection plane by `linetos`, of course. But ... what if we project B-cubic approximations? (I did not find time yet to go for the approach suggested by Bogusław Jackowski: find the corresponding transformation matrix.)

Projection of approximated circles and ellipses Sampling of a circle for projection requires 360 samples, say, dependent on the wanted accuracy. What if we approximate the circle by 4 suitably chosen B-cubics?

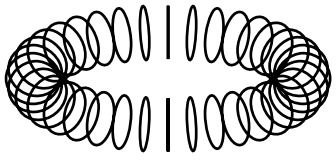
```
...
% .5 1 scale
0 0 100 0 360 arc 12 setlinewidth stroke %circle
4{-100 0 moveto -100 57 -57 100 0 100 curveto
  90 rotate}repeat% 4 B-cubics approximation of circle
1 setgray 4 setlinewidth stroke
...
```



The white approximation of the black circle, respectively ellipse, is good enough, for my projection purposes.

Why Handles of size 57? This value results from the requirement that the circle with radius 1 should pass through $\frac{1}{\sqrt{2}}(1,1)$, see below. It demonstrates that a user should have some knowledge of B-cubics.

Conclusion The projection of a circle, respectively ellipse, can be efficiently approximated by projection of 4 suitably chosen B-cubics. Instead of 360 sample points, say, only 12 points have to be projected. Not that relevant for my toy problems — I did not notice the difference in my LINEAR CONSTRUCTION IN SPACE No 1 & 2 — and nowadays PCs, but ... it is a factor 30, or so, more efficient, and no more cumbersome than the full sampling approach.



A nice example of repeatedly drawing projected circles is the toroid impression

A professional starts where an amateur ends In the MetaFont book p263 Knuth approximates the quarter circle by

```
quartercircle=(right{up}..(right+up)/sqrt2..up{left}) scaled .5
```

which is better, because he attacks the point of the greatest deviation by requiring that the spline should pass through $\frac{1}{\sqrt{2}}(1,1)$, unscaled. For the full circle (with unit diameter) he splices rotated copies of the quarter circle.

In PostScript we don't have such a path construction operator.

The requirement that the B-cubic, see Bézier formula (1), specified by $(1,0),(1,\delta),(\delta,1),(0,1)$ should pass for $t = .5$ through $\frac{1}{\sqrt{2}}(1,1)$ yields $\delta \approx .573$.

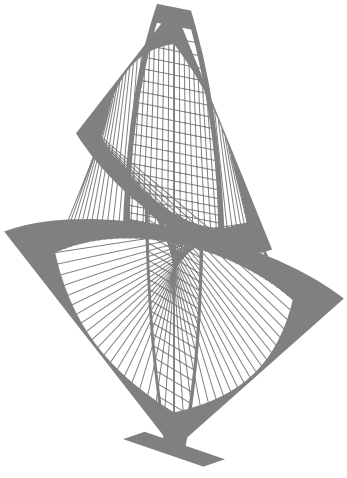
Appendix 2 TORSION emulation in EPSF

Conceptually the program, the script in PostScript lingo, is a handful of lines.

```
/theta 20 def  /phi 45 def  /scalingfactor 4 def
.5 setgray 0.5 setlinewidth %settings
foot fill
lowertorsiontriangle eofill cap eofill
uppertorsiontriangle eofill cup eofill
stringing stroke% paint the stringed surfaces to the current page
0 setgray annotations
showpage %print the current page
```

preceded by `%!PS-Adobe-3.0 EPSF-3.0`, DSC Comments, the Prolog (with the inclusion of the library, the ad hoc procedures cq data), and eventually followed by the Trailer part and closed by `%%EOF`.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Emulation of Naum Gabo Torsion
%%Creator: Kees van der Laan, kisa1@xs4all.nl
%%CreationDate: Februari 2011
%%BoundingBox: -35 -10 35 65
%%BeginSetup
%%EndSetup
(C:\\PSlib\\PSlib.eps) run %used from lib: ptp ,  tOnSpline, s (scaling) and default scalingfactor 1
%Specification of data in 3D and projections in 2D, dependent on phi, theta and scaling
%lowertorsiontriangle data
/a0 { 0 -25   s  25     s  ptp} def %outer points
/a1 { 0 -12.5 s  31.67  s  ptp} def
/a2 { 0  12.5 s  31.67  s  ptp} def
/a3 { 0  25   s  25     s  ptp} def
/origin { 0 0 } def
/b0 { 0 -17.5 s  23.5   s  ptp} def%inner points
/b1 { 0  -7.5 s  29.5   s  ptp} def
/bm { 0   0      28     s  ptp} def
/b2 { 0   7.5 s  29.5   s  ptp} def
/b3 { 0  17.5 s  23.5   s  ptp} def
/b4 { 0  15   s  13.5   s  ptp} def
/b5 { 0   9   s   8.5   s  ptp} def
/b6 { 0   0       5     s  ptp} def
/b7 { 0  -9   s   8.5   s  ptp} def
/b8 { 0 -15   s  13.5   s  ptp} def
%uppertorsiontriangle, 90 rotated, upside down data
/ua0 { -25   s 0  35    s  ptp} def%outer points
/ua1 { -12.5 s 0  28.34 s  ptp} def
/ua2 {  12.5 s 0  28.34 s  ptp} def
/ua3 {  25   s 0  35    s  ptp} def
/top { 0 0 60 s  ptp} def
/ub0 { -18.5 s 0  36.5 s   ptp} def%inner points of uframe
/ub1 {  -7.5 s 0  30.5 s   ptp} def
/ub2 {   7.5 s 0  30.5 s   ptp} def
/ub3 {  18.5 s 0  36.5 s   ptp} def
/ub4 {  15   s 0  46.5 s   ptp} def
/ub5 {   9   s 0  51.5 s   ptp} def
/ub6 {   0     0  55   s   ptp} def
/ub7 {  -9   s 0  51.5 s   ptp} def
/ub8 { -15   s 0  46.5 s   ptp} def
/ubm {   0     0  32   s   ptp} def
%cap outer
/t1 { 0 -2.5 s 60 s  ptp} def
/t2 { 0  2.5 s 60 s  ptp} def
/t3 { 0  6   s 50 s  ptp} def
/t4 { 0  7   s 40 s  ptp} def
```



φ = 30° θ = 20°

```
/t5 { b2 } def
/t6 { b1 } def
/t7 { 0 -7   s 40 s  ptp} def
/t8 { 0 -6   s 50 s  ptp} def
%cap inner
/it1 { 0 -2.5 s 57.5 s  ptp} def
/it2 { 0  2.5 s 57.5 s  ptp} def
/it3 { 0  6   s 46   s  ptp} def
/it4 { 0  6.5 s 36   s  ptp} def
/it5 { 0  6.5 s 30   s  ptp} def
/it6 { 0 -6.5 s 30   s  ptp} def
/it7 { 0 -6.5 s 36   s  ptp} def
/it8 { 0 -6   s 46   s  ptp} def
%cup outer; mirrored vertically around z=30 and rotated
/u1 { -2.5 s 0   0    ptp} def
/u2 {  2.5 s 0   0    ptp} def
/u3 {  6   s 0  10 s  ptp} def
/u4 {  7   s 0  20 s  ptp} def
/u5 { ub2 } def
/u6 { ub1 } def
/u7 { -7 s  0  20 s  ptp} def
/u8 { -6 s  0  10 s  ptp} def
%cup inner
/iu1 { -2.5 s 0  2.5 s ptp} def
/iu2 {  2.5 s 0  2.5 s ptp} def
/iu3 {  6   s 0 14   s ptp} def
/iu4 {  6.5 s 0 24   s ptp} def
/iu5 {  6.5 s 0 30   s ptp} def
/iu6 { -6.5 s 0 30   s ptp} def
/iu7 { -6.5 s 0 24   s ptp} def
/iu8 { -6   s 0 14   s ptp} def
%frame, formally in x- and y-components of the points
/lowertorsiontriangle{origin moveto %outer path
       a0 lineto
       a1 a2 a3 curveto closepath
       b6  moveto                                %inner path
       b7 b8 b0 curveto
       b1 b2 b3 curveto
       b4 b5 b6 curveto closepath} def
/uppertorsiontriangle{top moveto %outer path
        ua0 lineto
        ua1 ua2 ua3 curveto closepath
        ub6 moveto                           %inner path
        ub7 ub8 ub0 curveto
        ub1 ub2 ub3 curveto
        ub4 ub5 ub6 curveto closepath} def
/foot{-7.5 s     0   0  ptp moveto
         0    -7.5 s 0  ptp lineto
       7.5 s     0   0  ptp lineto
         0     7.5 s 0  ptp lineto
      closepath} def
/cap{ t1 moveto t2 lineto %outer
        t3 t4 b2 curveto
        b1 lineto
       t7 t8 t1 curveto closepath
      it1 moveto it2 lineto %inner
      it3 it4 it5 curveto
      it6 lineto
      it7 it8 it1 curveto closepath
}def
/cup{ u1 moveto u2 lineto %outer
      u3 u4 u5 curveto
      u6 lineto
      u7 u8 u1 curveto closepath
     iu1 moveto iu2 lineto %inner
      iu3 iu4 iu5 curveto
      iu6 lineto
```

```
      iu7 iu8 iu1 curveto closepath
}def
%build the paths for stringed surfaces
/dostringing{ 0 0.05 1.01{/t exch def
  t b0  b8  b7  b6  tOnSpline moveto
  t 2 div ub0 ub1 ub2 ub3 tOnSpline lineto
%
  t 2 div b0  b1  b2  b3  tOnSpline moveto
  t ub3 ub4 ub5 ub6 tOnSpline lineto
%
  t b3  b4  b5  b6  tOnSpline moveto
  t 2 div ub3 ub2 ub1 ub0 tOnSpline lineto
%
  t 2 div .5 add b0  b1  b2  b3  tOnSpline moveto
  t ub6 ub7 ub8 ub0 tOnSpline lineto
%cap
  t it1  it8  it7   it6  tOnSpline moveto
  t it2  it3  it4   it5  tOnSpline lineto
%cup
  t iu1  iu8  iu7   iu6  tOnSpline moveto
  t iu2  iu3  iu4   iu5  tOnSpline lineto
}for} bind def %end dostringing
%
/annotations{-32 -25  moveto
 S12pt setfont (j) show ( = ) H12pt setfont show phi   (  ) cvs show %note the use of phi for its value
   gsave 0 5 rmoveto (o) H7pt setfont show grestore
8 0 rmoveto
 S12pt setfont (q) show ( = ) H12pt setfont show theta (  ) cvs show %note the use of theta for the value
         0 5 rmoveto (o) H7pt setfont show} def
%%EndProlog
%
%---Program--- the script
%
/theta 20 def  /phi 45 def  /scalingfactor 4 def
.5 setgray 0.5 setlinewidth %settings
foot fill
lowertorsiontriangle eofill cap eofill
uppertorsiontriangle eofill cup eofill
dostringing stroke% paint the stringed surfaces to the current page
0 setgray annotations
showpage
%%EOF
```

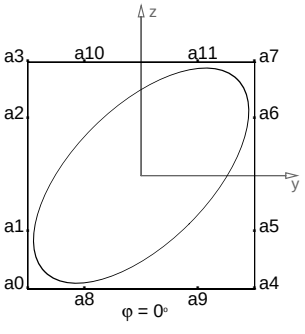
The annotations end all the emulation programs and are not repeated in the other listings, though they differ a little.

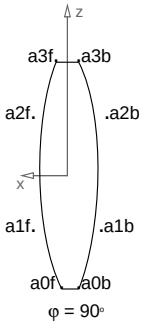
Appendix 3 Linear Construction in Space No 1 emulation in EPSF

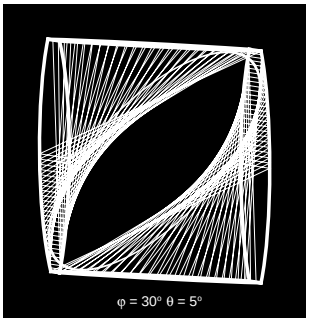
In one afternoon I rewrote the following program as emulation of LINEAR CONSTRUCTION IN SPACE No 1 from scratch in PostScript as EPSF, with subscripting conventions à la MetaFont for the points which determine the frame: the suffix `f` is a mnemonic for in **f**ront of the `yz` plane, the suffix `b` for **b**ehind the `yz` plane. The interior ellipse, which I neglected previously in the MetaFont code, has been accounted for (Gabo needed the interior slab for solidity. The elliptic hole for beauty?).

The program comprises 85 lines (roughly the same as the older `.mp`). The `.eps` result from the 1996 MP source takes 233 lines and is less intelligible.
The listing of the code is in the spirit of the Blue Book: the illustration along with the commented code. For the reader's convenience I have also included the frame with the data points, visually.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Emulation of Naum Gabo Linear Construction in Space No 1
%%BoundingBox: -130 -135 130 135
%%BeginSetup
%%EndSetup
%%Creator: Kees van der Laan
%%CreationDate: Februari 2011
%%DocumentFonts: Helvetica System
%%EndComments
(C:\\PSlib\\PSlib.eps) run
/reversevideo{-130 -135 260 270 rectfill}def%Mimics the BoundingBox
/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
%
/frame{
a0f moveto  a1f  a2f  a3f curveto
            a10f a11f a7f curveto
            a6f  a5f  a4f curveto
            a9f  a8f  a0f curveto closepath
```







```
a0b moveto  a1b  a2b  a3b curveto
            a10b a11b a7b curveto
            a6b  a5b  a4b curveto
            a9b  a8b  a0b curveto closepath
a0f moveto a0b lineto
a3f moveto a3b lineto
a4f moveto a4b lineto
a7f moveto a7b lineto
}def
%
/approxellipsestroke{gsave % project approximated ellipse (by 4 B-cubics)
-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
          closepath
stroke grestore} def
%
/dostringing{.02 .02 .5001{/t exch def
t       a0f a1f  a2f  a3f tOnSpline moveto
2 t mul a3b a10b a11b a7b tOnSpline lineto% note the cross-over
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 %end dostringing
%%EndProlog
%
%---Program--- the script
%
/phi  30 def /theta 5 def /scalingfactor 50 def
reversevideo 1 setgray %white further on
3  setlinewidth frame        stroke
2  setlinewidth approxellipsestroke
.5 setlinewidth dostringing  stroke
showpage
%%EOF
```

Gabo has made variations of the LINEAR CONSTRUCTION No 1, where the sides of the frame are varied.

Appendix 4 Linear Construction in Space No 2 emulation in EPSF

A MetaFont program has been published in MAPS in 1996, where use is made of `path p[]`, `dir`, `down`, `left`, `hide`, `point ... of`, `for ... endfor`, `xpart`, `ypart`, `drawoptions(withcolor white)`, `draw`, `fill ... reverse ... cycle withcolor black`. Still a nice program after so many years. I paid attention this time also to the elliptic cut outs in the perspex frame in the yz slab as well as in the xz slab.

I had difficulties in passing along the circumference of the frame in equidistant steps: after `flattenpath` {}{}{}{}`pathforall` did not give me an equidistant set of points on the stack, because of too much change in curvature.

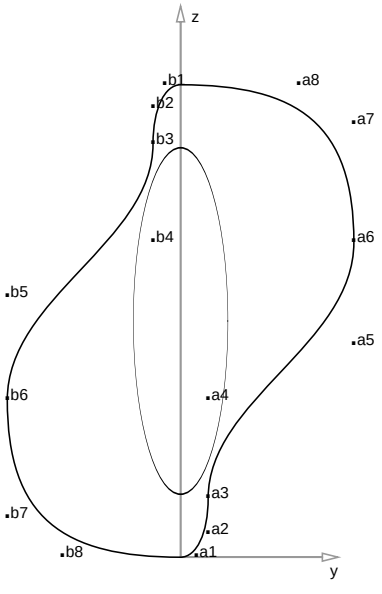
Moreover, MetaFont/MetaPost's part of the frame path

```
p1:= (0, 3size){right}..{down}(1.1size, 1.75size)..
     (.175size, .375size)..{left} origin;
```

I could not precisely transcribe into PostScript. How to mimic the directions `left`, `down` and ilks? Of course in the direction of the tangents of the curve at the point, but how far, taking into account scaling? Furthermore, in PostScript you can't apparently supply a point without the control points for drawing a B-cubic through the point.

The Frame of the object consists of a BarbaPapa-like shape in perspex, with an elliptic cutout, intersected with a copy rotated over 90° along the vertical axis.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Linear Construction in Space No 2, Frame in yz plane
%%Creator: Kees van der Laan, kisa1@xs4all.nl
%%CreationDate: Februari 2011
%%BoundingBox: -115 -15 125 355
%%BeginSetup
%%EndSetup
%%DocumentFonts: Helvetica
%%Pages: 0
%%EndComments
(C:\\PSlib\\Pslib.eps) run % used: origin arrow dotsandnames s  H12pt
/y-axis { origin  1 s    0   .1 5 10 arrow } def
/z-axis { origin    0  3.5 s   .1 5 10 arrow } def
 1 s    0    moveto -5 -12 rmoveto (y)  H10pt setfont show
   0  3.5 s  moveto  7 -10 rmoveto (z)                show
%yz plane
/a0 {0         0       } def
/a1 { .1    s  0       } def
/a2 { .175  s   .15  s } def
/a3 { .175  s   .375 s } def
/a4 { .175  s  1     s } def
/a5 { 1.1   s  1.35  s } def
/a6 { 1.1   s  2     s } def
/a7 { 1.1   s  2.75  s } def
/a8 {  .75  s  3     s } def
/a9 { 0        3     s } def
%mirrored and upside down, -yz plane
/b0 { 0       3     s } def
/b1 { -.1   s 3     s } def
/b2 { -.175 s 2.85  s } def
/b3 { -.175 s 2.625 s } def
/b4 { -.175 s 2     s } def
```



```
/b5 {-1.1   s 1.65  s } def
/b6 {-1.1   s 1     s } def
/b7 {-1.1   s  .25  s } def
/b8 {-.75   s 0     s } def
/b9 { 0       0       } def
%
/frame{
a0 moveto
a1 a2 a3 curveto
a4 a5 a6 curveto
a7 a8 a9 curveto
%b0 moveto
b1 b2 b3 curveto
b4 b5 b6 curveto
b7 b8 b9 curveto
closepath} def
%
/ellipse{gsave
   0 1.5 s translate
  .3 1.1 scale
   0 0 1 s 0 360 arc stroke
grestore}def
%%EndProlog
%
%Program
%
/scalingfactor 100 def
frame ellipse stroke
[(a1) (a2) (a3) (a4) (a5) (a6) (a7) (a8)
 (b1) (b2) (b3) (b4) (b5) (b6) (b7) (b8)] dotsandnames
y-axis z-axis  .6 setgray stroke
showpage
```

Stringing is more difficult.

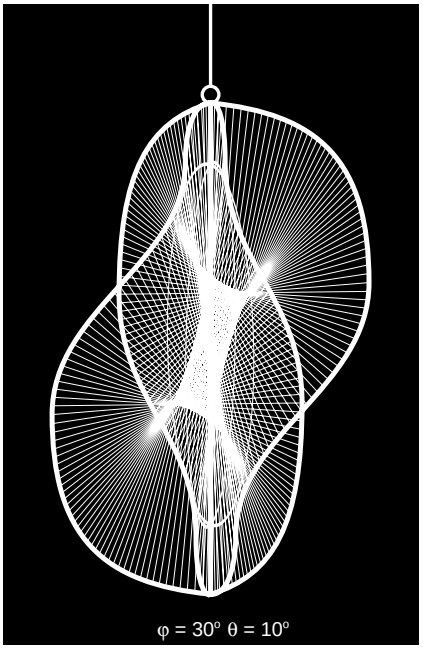
The origin is chosen at the foot of the object. The (unprojected) circumference in the first quadrant of the yz plane consists of three B-cubics, with varying curvature. Roughly equidistant points are sampled and stored on the stack. Each point is and projected and transformed into the x^-z quadrant, i.e. the plane through the negative x axis and the positive z axis, and also projected. Both projected points are connected by a line (wire). Schematically, for each point on the stack

$$(0,\ y,\ z)\texttt{ptp<-string->}(-y,\ 0,\ size-z)\texttt{ptp} \quad \text{with} \quad size = 3s, s = 100.$$

This is repeated for the y^-x^+ quadrant, and programmed with the sample points on the stack (too low-levelish, yes I know, but ... interesting; for the other quadrants the sample values on the stack are copied into an array, for easy multiple access, not difficult at all).

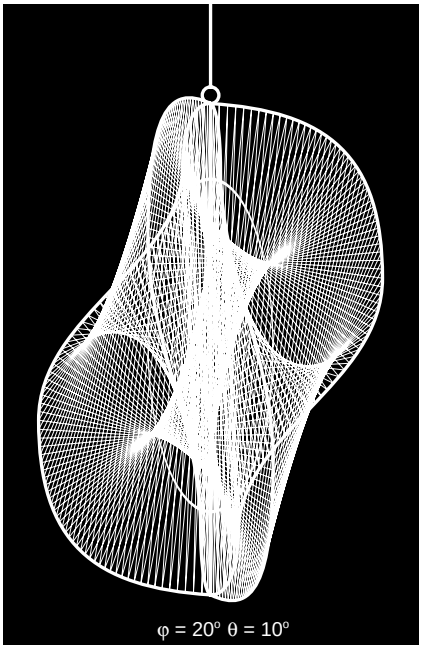
```
%!PS-Adobe-3.0 EPSF-3.0
%%Title: Emulation of Naum Gabo Linear Construction in Space No 2
%%Creator: Kees van der Laan, kisa1@xs4all.nl
%%CreationDate: Februari 2011
%%BoundingBox: -125 -30 125 355
%%ProgramFonts: Helvetica
%%BeginSetup
%%EndSetup
%%Pages: 0
%%EndComments
(C:\\PSlib\\Pslib.eps) run %used: origin dotsandnames s ptp tOnSpline
/reversevideo{-125 -30 250 385 rectfill}def%Mimics BoundingBox
/rope{0 355 moveto
      0 0 0 3 s ptp 5 add 5 90 450 arc}def %ring
%%EndProlog
%In yz plane data points for the frame
/a0 {0 0        0       ptp} def
```

```
/a1 {0  .1   s  0       ptp} def
/a2 {0  .175 s   .15  s ptp} def
/a3 {0  .175 s   .375 s ptp} def
/a4 {0  .175 s  1     s ptp} def
/a5 {0 1.1   s  1.35  s ptp} def
/a6 {0 1.1   s  2     s ptp} def
/a7 {0 1.1   s  2.75  s ptp} def
/a8 {0  .75  s  3     s ptp} def
/a9 {0  0       3     s ptp} def
%
%Roughly Equidistant Sampling of frame part in yz plane, with phi=0 and theta=0
/sampleframe{/phi 0 def /theta 0 def   %results on the operand stack
0 .175  .95{ a0 a1 a2 a3 tOnSpline }for
0.02 .028 .99{ a3 a4 a5 a6 tOnSpline }for
0  .025 1  { a6 a7 a8 a9 tOnSpline }for
}def
%rotated -xz plane data points for frame
/a0r { 0       0 0       ptp} def
/a1r {  .1   s 0 0       ptp} def
/a2r {  .175 s 0  .15  s ptp} def
/a3r {  .175 s 0  .375 s ptp} def
/a4r {  .175 s 0 1     s ptp} def
/a5r { 1.1   s 0 1.35  s ptp} def
/a6r { 1.1   s 0 2     s ptp} def
/a7r { 1.1   s 0 2.75  s ptp} def
/a8r {  .75  s 0 3     s ptp} def
/a9r { 0       0 3     s ptp} def
%mirrored and upside down, -yz plane data points for frame
/b0 {0  0       3     s ptp} def
/b1 {0  -.1   s 3     s ptp} def
/b2 {0  -.175 s 2.85  s ptp} def
/b3 {0  -.175 s 2.625 s ptp} def
/b4 {0  -.175 s 2     s ptp} def
/b5 {0 -1.1   s 1.65  s ptp} def
/b6 {0 -1.1   s 1     s ptp} def
/b7 {0 -1.1   s  .25  s ptp} def
/b8 {0 -.75   s 0       ptp} def
/b9 {0   0      0       ptp} def
%rotated -xz plane data points for frame
/b0r { 0       0 3     s ptp} def
/b1r { -.1   s 0 3     s ptp} def
/b2r { -.175 s 0 2.85  s ptp} def
/b3r { -.175 s 0 2.625 s ptp} def
/b4r { -.175 s 0 2     s ptp} def
/b5r {-1.1   s 0 1.65  s ptp} def
/b6r {-1.1   s 0 1     s ptp} def
/b7r {-1.1   s 0  .25  s ptp} def
/b8r { -.75  s 0 0       ptp} def
/b9r { 0       0 0       ptp} def
%
/frame{%executed with user projection angles
a0 moveto
a1 a2 a3 curveto
a4 a5 a6 curveto
a7 a8 a9 curveto
a0r moveto
a1r a2r a3r curveto
a4r a5r a6r curveto
a7r a8r a9r curveto
b0 moveto
b1 b2 b3 curveto
b4 b5 b6 curveto
b7 b8 b9 curveto
b0r moveto
b1r b2r b3r curveto
b4r b5r b6r curveto
```



```
b7r b8r b9r curveto
closepath
}def %end frame
%
/dostringing{count %83 number of samples x and y on the stack
/n exch def
n copy %for stringing 2 quadrants
%Stringing +yz<->-xz
n 2 div cvi {2 copy      % y z y z on stack
             0 3 1 roll  % y z 0 y z
             ptp         % y z projected point
             moveto      % y z
             exch neg exch% -y z
             3 s sub neg % -y 3s-z
             0 exch      % -y 0 3s-z
             ptp         % projected pair
             lineto
            }repeat
%Stringing +xz<->-yz
n 2 div cvi {2 copy      % y z y z on stack
             0 2 1 roll  % y z y 0 z
             ptp         % y z projected point
             moveto      % y z
             0 3 1 roll  % 0 y z
             exch neg exch% 0 -y z
             3 s sub neg % 0 -y 3s-z
             ptp         % projected pair
             lineto
            }repeat
%other quadrants
%stringing +xZ<->+yz
/np1 n 1 add def%n-1 last element of samples array
/cnt 0       def%first element of samples array
n 2 div cvi {/cnt cnt 2 add def
% in +yZ plane
0                       %x=0 inserted
samples n   cnt sub get %y value from samples array
samples np1 cnt sub get %z value
ptp moveto
% rotate to +xz plane
samples cnt 2 sub get %y value from samples array --> x
0                     %y=0 inserted
samples cnt 1 sub get %z value
ptp lineto
}repeat
%stringing -xz<->-yz
/np1 n 1 add def%n-1 last element of samples array
/cnt 0       def%first element of samples array
n 2 div cvi {
/cnt cnt 2 add def
% in +yZ plane
0                                  %x=0 inserted
samples n   cnt sub get neg        %y value from samples array
samples np1 cnt sub get 3 s sub neg %mirrored z value
ptp moveto
% rotate to +xz plane
samples cnt 2 sub get neg         %y value from samples array ->  x
0                                 %y=0 inserted
samples cnt 1 sub get 3 s sub neg %mirrored z value
ptp lineto
}repeat
} bind def %end dostringing
%
/yz-approxellipseNo2stroke{gsave%implicit in yz plane, phi, theta must be given
0 1.5 s translate
.3  1.1 scale%kind of ellipse
/a0 {0 -1   s  0     ptp} def
```

```
/a1 {0 -1   s  .55 s ptp} def
/a2 {0 -.55 s  1   s ptp} def
/a3 {0  0      1   s ptp} def
/a4 {0  .55 s  1   s ptp} def
/a5 {0  1   s  .55 s ptp} def
/a6 {0  1   s  0     ptp} def
/a7 {0  1   s -.55 s ptp} def
/a8 {0  .55 s -1   s ptp} def
/a9 {0  0     -1   s ptp} def
/a10{0 -.55 s -1   s ptp} def
/a11{0 -1   s -.55 s ptp} def
a0 moveto a1  a2  a3 curveto
          a4  a5  a6 curveto
          a7  a8  a9 curveto
          a10 a11 a0 curveto closepath stroke
grestore} def  %end yz-approxellipseNo2stroke
%
/xz-approxellipseNo2stroke{gsave%implicit in xz plane, phi, theta must be given
0 1.5 s translate
.3  1.1 scale%kind of ellipse
/a0 {-1   s 0  0     ptp} def
/a1 {-1   s 0  .55 s ptp} def
/a2 {-.55 s 0  1   s ptp} def
/a3 { 0     0  1   s ptp} def
/a4 { .55 s 0  1   s ptp} def
/a5 { 1   s 0  .55 s ptp} def
/a6 { 1   s 0  0     ptp} def
/a7 { 1   s 0 -.55 s ptp} def
/a8 { .55 s 0 -1   s ptp} def
/a9 { 0     0 -1   s ptp} def
/a10{-.55 s 0 -1   s ptp} def
/a11{-1   s 0 -.55 s ptp} def
a0 moveto a1  a2  a3 curveto
          a4  a5  a6 curveto
          a7  a8  a9 curveto
          a10 a11 a0 curveto closepath stroke
grestore} def  %end xz-approxellipseNo2stroke
%
%---Program--- the script
%
/scalingfactor 100 def
reversevideo 1 setgray %white lines, further on
sampleframe %with phi=theta=0!
/phi 30 def /theta 10 def% viewing angle
.5 setlinewidth dostringing stroke
3  setlinewidth frame       stroke
2  setlinewidth yz-approxellipseNo2stroke
                xz-approxellipseNo2stroke
                rope        stroke
showpage
%%EOF %Of my most complete Linear Construction in Space No 2 emulation
```



The .eps result from MetaPost comprises 800+ lines and is, alas, not intelligible. I can't even recognize the created path.

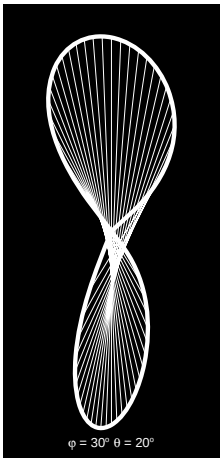
Note the flexible way of specifying a path in MetaFont/MetaPost. In PostScript we can just construct piecewise B-cubics. A general curve is approximated after projection of the sample points by `lineto's. Of course we can write procedures of our own, to imitate MetaPost's flexibility. Another difference with respect to `hidden lines' is that MetaPost has an `undraw` macro while in PostScript we must explicitly draw in the colour of the background.

Gabo has made LINEAR CONSTRUCTION IN SPACE No 2 in 1949, 1958, 1962-64, 1972-73, and 1976.

My 1996, simplified, pseudo-animated version was on my WWW of old, and is included below. It does not show all lines; nevertheless nice in its simplicity.

Variations

Gabo has made variations among others based on an octahedral frame. I made a variation with a twisted lemniscate frame —lemniscate in polar coordinates $r^2 = 2a^2 \cos 2\theta$ — in an hour or two. The octahedral variant would take another hour to emulate.



The architectural variation, he never realized, was a commission for the Esso Company NY: above each of 2 revolving entrance doors a `Linear Construction No 2` revolving at a slower speed than the doors.

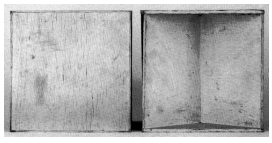
The wall between was a relief in itself: a set of perspective lines that seem to break the wall into a deeper space, and within these lines, a large construction of spiralling arcs around a vertical axis.

Appendix 5 Emulation of a Spheric Theme in EPSF

Spheric Themes demonstrate another way of how Gabo constructed (the idea of) surfaces: by planes orthogonal to the surface.

He called it the stereometric method of representing surfaces.

> Stereometry is a branch of mathematics concerned with the description of 3D objects and calculations related to these. For Gabo a cube is not represented by its 6 surfaces, but by top, bottom, and the intersecting diagonal planes, making the inside visible



Famous are his constructed head series, such as the earlier shown `Head in a Corner Niche`, and the `Head ...` on the book cover. In the Spheric Theme we are about to emulate both ways of representing surfaces: stereometric and stringed surfaces are integrated in one object.

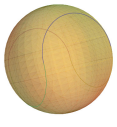
Gabo invented how to model his Spheric Theme curves:

> `The basic form was made by taking two identical flat pierced circles or broad rings and making a single cut in each along the line of a radius. The two discs are bent in a serpentine curve and butt-jointed to each other at both ends. The resulting figure fits exactly into a sphere and the outer edges of the discs form the interlocking curves like those that divide the pieces of felt covering the tennis ball.'

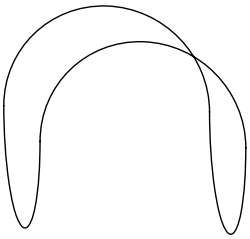
Frame

The outer circumference of the frame is the curve on the tennis ball, a little deformed. The four parts of the tennis ball curve are given by the formulas

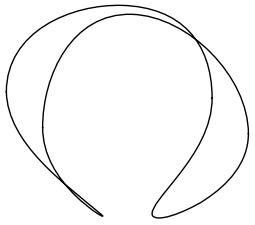
$$\begin{aligned}&\{(x,y,z)|(1,\sin t,\cos t)\}\\&\{(x,y,z)|(-1,\sin t,\cos t)\}\\&\{(x,y,z)|(-\sin t,1,-\cos t)\}\\&\{(x,y,z)|(-1,-\sin t,-\cos t)\}\end{aligned}\qquad t\in(-\frac{\pi}{2},\frac{\pi}{2})$$



I modified the tennis ball curve by the deformation $1\rightarrow(1-d)\sin^2 t+d$, $d=.25$, see figure below, which comes close to Gabo's. For the inner circumference I took the outer scaled by a factor 3.

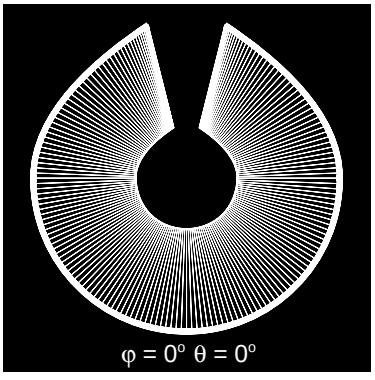


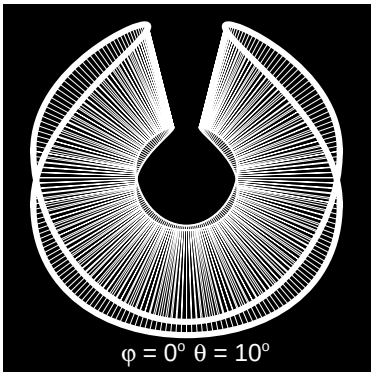
Curve Tennis ball $\varphi = \theta = 10^o$

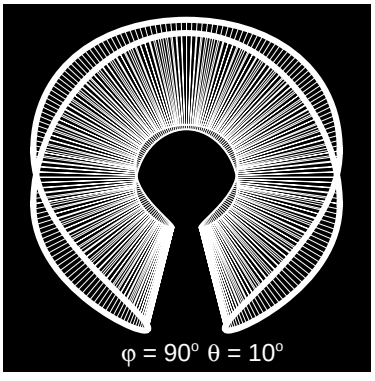


Mod: 1 -> .75sin^2(t)+.25, $\varphi = \theta = 10^o$

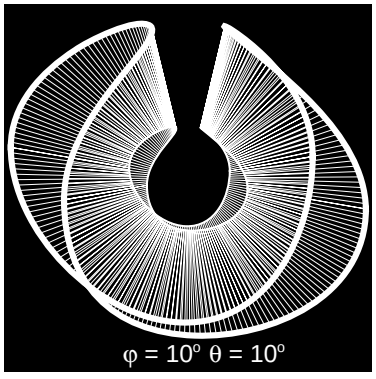
```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: -90 -90 90 90
%%BeginSetup
%%EndSetup
%%Title: Emulation of Naum Gabo Spheric Theme
%%Creator: Kees van der laan, kisa1@xs4all,nl.
%%CreationDate: feb 2011
%%DocumentFonts: Helvetica Symbol
%%EndComments
(C:\\PSlib\\PSlib.eps) run %used: s ptp
%
/reversevideo{-90 dup 180 dup rectfill}def %Mimics the BoundingBox
%
/sphericthemestroke{%
%%%part upper front x>0 {t|(ft, sin t, -cos t)}, f(-90)=1
1 s   -1 s    0 ptp moveto
-89 1 90{/t exch def
ft s   t sin s   t cos s neg ptp lineto
}for stroke
%%%part upper back x<0 {t|(-ft, sin t, -cos t)}
-1 s   -1 s   0  ptp moveto
-89 1 90{/t exch def
ft neg s   t sin s   t cos s neg  ptp lineto
}for stroke
%%%part right y>0 {t|(-sin t, ft,  cos t)}
1 s   1 s   0 ptp moveto
-89 1 90{/t exch def
t sin neg s   ft s   t cos  s    ptp lineto
}for stroke
%%%part left y<0 {t|(-sin t, -ft,  cos t)}
1 s   -1 s   0 ptp moveto
-89 1 90{/t exch def
t sin neg s    ft neg  s    t cos s   ptp lineto
}for stroke
} bind def %end sphericthemestroke
%
/dostringingstroke
{%%%part upper front x>0 {t|(ft, sin t, -cos t)}, f(-90)=1
/v {.333 mul} def /step 2 def% implies number of strings
-90  step 90{/t exch def
ft s     t sin s    t cos s   neg ptp moveto
ft s v   t sin s v  t cos s v neg ptp lineto
}for
stroke
%%%part upper back x<0 {t|(-ft, sin t, -cos t)}
-90 step 90{/t exch def
ft neg s    t sin s    t cos s   neg ptp moveto
ft neg s v  t sin s v  t cos s v neg ptp lineto
}for
stroke
%%%part right y>0 {t|(-sin t, ft,  cos t)}
```







```
-90 step 90{/t exch def
t sin neg s     ft s     t cos s     ptp moveto
t sin neg s v   ft s v   t cos s v   ptp lineto
}for
stroke
%%%part left y<0 {t|(-sin t, -ft,  cos t)}
-90 step 90{/t exch def
t sin neg s     ft neg  s     t cos s    ptp moveto
t sin neg s v   ft neg  s v   t cos s v  ptp lineto
}for
stroke
} bind def %end dostringingstroke
%%EndProlog
%
%---Program--- the script
%
reversevideo 1 setgray /d .25 def
/ft { 1 //d sub t sin dup mul mul //d add} def
/phi 90 def /theta 10 def
/scalingfactor 25 def sphericthemestroke
/scalingfactor 75 def sphericthemestroke
.1 setlinewidth dostringingstroke
showpage
%%EOF
```

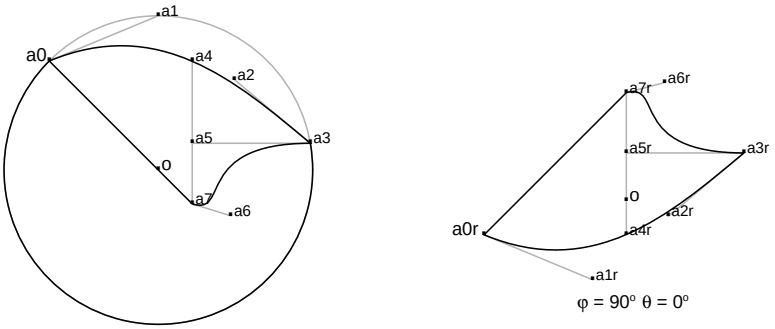


Appendix 6 Linear Construction in Space: Suspended

One of Gabo's favorites, which he used to display on retrospective exhibitions.

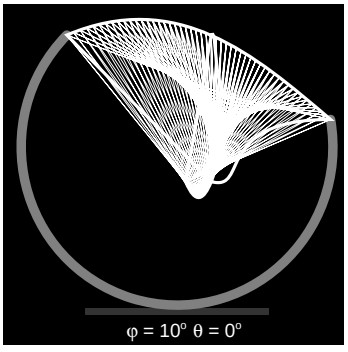
Frame The frame consists of 2 identical, `triangular' slabs of clear plastic orthogonal (rotated and mirrored) to each other, suspended on a metal arc with a wooden base.

What motivated Gabo to create just these triangular shapes? What were the restrictions? Did the balance of the object play a role, in choosing the rotation symmetry axes? I don't think so, just his feeling for beauty, I guess. The unbalance creates tension.



Left the frame part in the yz plane, right the (rotated and mirrored) frame part in the xz plane. The frame part in the yz plane is realized by 2 splines: `a0 moveto a1 a2 a3 curveto a5 a5 a7 curveto closepath`.

For the first B-cubic the data points are a_0,a_3 with control points a_1,a_2. For the second B-cubic the data points are a_3,a_7 with control points a_5,a_6; not critical. The axis of rotation is the line from a_4 to a_7. The suffix `r` denotes the mirrored and rotated triangle points.



I did not include the program, because the rotation required details, which I expect too boring for the casual reader.

Appendix 7 Torsion (interactive) MetaFont program of old

For the MetaFont aficionados: the program below still works on my PowerMac. Not so difficult to adapt for those fluent in MetaPost, I presume.

```
%Gabo's torsion example. December 1995, CGL
%
tracingstats:=proofing:=1;screenstrokes;
"Torsion from different viewpoints";
string yorn;
message "Gabo's Torsion.";
message "Do you wish simplest variant? (y or n):";
yorn:= readstring;
size=50;
def openit = openwindow currentwindow
   from origin to (screen_rows,screen_cols) at (-size,300)enddef;
pickup pencircle scaled .005pt;
pair aux[];
path  po[], pi[];
if yorn="n": d=.1size
        else: d=0 fi;
r=.2size;
for b= 20:%15step10until45:
for a= 0 step30until90:
```

```
currentpicture:=currentpicture shifted (2size,0);
%The following def must be included or a, b, must be supplied as arguments.
def ptp(expr x,y,z)=(-x*cosd a +y*sind a, -x*sind a *sind b -y*cosd a *sind
b+ z*cosd b)enddef;
po1:=ptp(0,-size,2d)--ptp(0,0,size+2d)--ptp(0,size,2d)&
     ptp(0, size,2d)..ptp(0,0,0)..ptp(0,-size,2d)..cycle;
aux0:=.5[ptp(0,-size,2d),ptp(0,0,size+2d)];
aux1:=.5[ptp(0,size,2d),ptp(0,0,size+2d)];
pi1:=ptp(0,-size+2.5d,2.5d)..controls aux0..
    ptp(0,0,size-d)..controls aux1..
    ptp(0,size-2.5d,2.5d)...
    ptp(0, size-2.5d,2.5d)...ptp(0,0,d)...ptp(0,-size+2.5d,2.5d)..cycle;
po2:=ptp(-size,0,-2d)--ptp(0,0,-size-2d)--ptp(size,0,-2d)&
     ptp(size,0,-2d)..ptp(0,0,0)..ptp(-size,0,-2d)..cycle;
aux3:=.5[ptp(-size,0,-2d),ptp(0,0,-size-2d)];
aux4:=.5[ptp(size,0,-2d),ptp(0,0,-size-2d)];
pi2:=ptp(-size+2.5d,0,-2.5d)..controls aux3..
    ptp(0,0,-size+d)..controls aux4..
    ptp(size-2.5d,0,-2.5d)...
    ptp(size-2.5d,0,-2.5d)...ptp(0,0,-d)...
    ptp(-size+2.5d,0,-2.5d)..cycle;
%Foot
po3:=ptp(r, 0,-size-2d)..ptp( .706r, -.706r,-size-2d)..
     ptp(0,-r,-size-2d)..ptp(-.706r, -.706r,-size-2d)..
     ptp(-r,0,-size-2d)..ptp(-.706r,  .706r,-size-2d)..
     ptp( 0,r,-size-2d)..ptp( .706r,  .706r,-size-2d)..cycle;
%
if yorn="n":fill po1; unfill pi1;fill po2; unfill pi2
            ;draw ptp(0,0,-size+d)--ptp(0,0,-size-2d)
            ;draw ptp(0,0,size-d)--ptp(0,0,size+2d)
        else:draw po1; draw po2; draw pi1; draw pi2 fi;
fill po3;
for k=0 upto 20:
 draw point .1k of pi1--point 5-.1k of  pi2;
endfor
for k=0 upto 20:
 draw point 3+.1k of pi1--point 2-.1k of  pi2;
endfor
showit;
endfor
endfor
end
```