

Text Objects

George V. Reilly

Department of Computer Science
Brown University

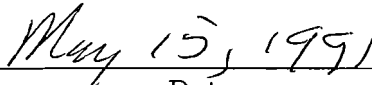
Submitted in partial fulfillment of the requirements for the
Degree of Master of Science in the Department of Computer Science
at Brown University

May 1991

This research project by George V. Reilly is accepted in its present form
by the Department of Computer Science at Brown University
in partial fulfillment of the requirements for the Degree of Master of Science.



Professor Andries van Dam
Advisor



Date

Contents

1	Introduction	1
2	Text Objects	2
2.1	Font Formats	2
2.2	Choosing a Font Format	3
2.3	Representations of Characters in 3D	3
2.4	Choosing Representations for Text Objects	4
2.5	Converting a Character to a TRIPobject	4
3	Text Layout	5
3.1	Two-Dimensional Virtual Page	5
3.2	Pros and Cons of T _E X as a Formatting Language	7
3.3	Interactive Text Editor	8
3.4	Three Dimensions: Arbitrary Positions and Orientations	8
4	Incorporation into BAGS	8
4.1	Adding New Fonts and Representations	9
4.2	Code Organization	9
5	Conclusions	10
5.1	Future Work	10
5.2	Summary	11

1 Introduction

Digital typography is the technology of using computers for the design, preparation, and presentation of documents, in which the graphical elements are organized, positioned, and themselves created under digital control. — Richard Rubinstein
[Rub88]

Digital typography has been a moderately active area of research in computer science. Work has been done in text formatting, page-description languages, WYSIWIG editors, writing tools, digital typefaces, the particular problems of low-resolution devices, and rasterization of letterforms. Almost all of this work has been directed towards two-dimensional text, that is, text on a sheet of paper or a computer screen. Although some work has been done on three-dimensional typography, as evinced by the ‘flying logos’ so beloved of advertisers, almost nothing has been published about it.

This report describes a research project which investigated three-dimensional text objects. An implementation exists within BAGS, the Brown Animation Generation System [ZCv⁺91, Str88], and arbitrary text can be rendered.

Two key issues were addressed: (1) taking data describing a character and converting that data into an object that BAGS can render, and (2) specifying the position and orientation of characters in a scene. (BAGS already handles the time-varying attributes.)

This report describes various kinds of font data, possible representations for text objects, how text objects can be positioned in space, the choices and tradeoffs that were made in my implementation, and presents my conclusions.

2 Text Objects

Before considering the nature of the text objects and how they might be represented, we must first consider the data that describes characters. There are several possible sources of font data. The limitations of the data used determines the nature of the text objects in BAGS; e.g., the Hershey fonts are made up of points connected by straight-line segments and clearly cannot be rendered as smooth curves (although curves could be fitted to the data).

2.1 Font Formats

The following list of font formats is by no means complete.

Bitstream: Bitstream fonts define characters as outlines. Bitstream supply data in three different formats.

Definitive Outlines: Characters are defined as a series of lines and circle arcs on a very high-resolution coordinate system [Bit84].

Compressed Outlines: These are derived from Definitive Outlines, but are stored in a very compact form [Bit87b].

Bézier Outlines: Character outlines are described by lines and cubic Bézier curves [Bit87a]. Again, the data is derived from the Definitive Outlines.

Hershey: The Hershey fonts [Her67, Her72] were developed for vector plotters and consist of points connected by straight-line segments.

POSTSCRIPT: POSTSCRIPT fonts come in two varieties: type 1 and type 3 [Ado85a, Ado85b, Ado90]. Type 1 fonts are described by POSTSCRIPT programs written using a very limited subset of the POSTSCRIPT language (essentially straight-line segments and Bézier curves), while type 3 font programs can use nearly any POSTSCRIPT construct.

TrueType: Another outline format, TrueType was developed by Apple and Microsoft [Gla90] in an effort to grab a share of the font market from Adobe.

METAFONT: Knuth's METAFONT is a sophisticated program for drawing character glyphs [Knu86c, Knu86d, Knu86e]. Each character is described by a series of pen strokes, drawn by pens with nibs of various shapes.

2.2 Choosing a Font Format

The primary requirement for font data used in generating three-dimensional text objects is that the characters be scaleable. This immediately precludes using bitmaps. The characters should also look good at large magnifications which rules out the (polygonal) Hershey characters. Finally, the character data should be in a simple, easy-to-manipulate format, which excludes POSTSCRIPT type 3 fonts and METAFONT fonts.

Thus, of the formats described above, we are left with the outline formats: the three Bitstream formats, POSTSCRIPT type 1 fonts, and TrueType.

The current implementation in BAGS uses Bitstream Compressed Outline fonts, primarily because we have a demonstration tape from Bitstream containing sample fonts—Swiss (Helvetica), Dutch (Times), and Kanji—in Compressed and Bézier Outline formats.

Bitstream sell several hundred different typefaces, so there is little point in adding support for TrueType or POSTSCRIPT type 1 fonts.

2.3 Representations of Characters in 3D

There are several possible representations for text objects: polyhedra—TRIPobjects (TRIangular Polyhedra) [Hua90] in BAGS terminology, CSG's [Hub90] of prisms [Kaj83] (called translational sweeps in [FvDFH90, Chapter 12]), and surface patches ([Far90], [BBB87], and [FvDFH90, Chapter 11]).

TRIPobjects: It is conceptually straightforward to polygonize an outline font and extrude the polygon into a three-dimensional shape.

CSG's of prisms: A prism is a closed planar curve extruded along a normal vector. Clearly an extruded three-dimensional character is a prism. CSG's are needed for characters that have two or more parts (e.g., an 'i' and its dot would be represented by the union of the two prisms) and characters that have holes (e.g., a 'b' would be the difference of the

prism described by the outer outline and that described by the inner outline).

Surface patches: A three-dimensional character can also be described as a collection of surface patches. As both Bitstream and Adobe type 1 fonts can be described by Bézier curves, conversion to Bézier bicubic patches should be straightforward.

Some of these representations can lead to having characters with beveled edges, or characters with arbitrary profiles between the front and back faces. Prisms cannot represent such character shapes, although polyhedra, surface patches, and ducts certainly could.

2.4 Choosing Representations for Text Objects

BAGS object classes must be able to present at least two versions of themselves to renderers—a polyhedral form for the polygonal modellers and a ray-intersection form, although strictly speaking, the polyhedral form can always be used instead of the ray-intersection form.

Text objects are currently implemented as TRIPobjects in BAGS. There are several reasons for this: (a) the interactive polygonal modellers require that object classes should provide a TRIPobject representation of themselves; (b) prisms have not yet been implemented in the new system, and the surface patch modeller is still under development; and (c) I expected it would be straightforward to polygonize character outlines. I was wrong, as I first had to enhance the TRI triangularization package so that it could triangulate polygons with holes, and this turned out to be much more difficult than I expected.

2.5 Converting a Character to a TRIPobject

To convert a Bitstream Compressed Outline character into a TRIPobject, we first build a tree of outline curves (e.g., a 'B' has one node at the root and two children; an 'i' has two nodes at the root; a kanji character might have as many as ten nodes in its curve tree. This hierarchical structure is needed by the triangulation routine). Each curve is then polygonized at a user-controllable resolution. (As a special case, if the resolution is zero, the

character is converted into a rectangle whose width and height is that of the character. This is useful for testing purposes as the rectangles will render quickly.) This collection of hierarchical polygonized curves is passed to the triangularization routine which considers it all to be one polygon with holes and islands.

The back face of the TRIPobject is built by copying the triangles of the front face in reverse order (so that the normals point outwards) and shifting their z -coordinates. Finally, the front face is connected to the back face, by circling around the interior and exterior curves and connecting vertices to their 'opposite numbers' and their neighbours..

3 Text Layout

Text can be thought of as one-, two-, or three-dimensional. A word, a short phrase, and a flying logo are all one-dimensional. One-dimensional text objects can be positioned like any other objects. A page of text, a caption, and a slide are two-dimensional.

There are two models of laying out text to be considered: the simple, rectilinear, two-dimensional *virtual page* model, and the arbitrary-position, arbitrary-orientation, three-dimensional model.

3.1 Two-Dimensional Virtual Page

To lay out text on a two-dimensional virtual page, some sort of text-formatting language is needed. (A WYSIWIG editor is arguably a simple, imperative text formatter.) Either a new language must be written from scratch or an existing one can be adapted.

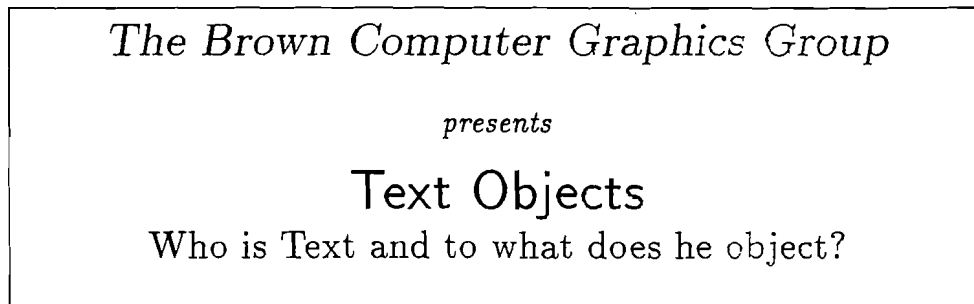
The complexity and sophistication of existing text-formatting languages indicates that writing a new text formatter would be a difficult and time-consuming task.

Thus, I decided to use \TeX ¹ as the formatting language for the BAGS text objects. \TeX outputs dvi (DeVice Independent) files, which contain directions describing where characters should be placed on a page.

¹ \TeX is a sophisticated, programmable text formatter that comes in two major flavors: 'plain' \TeX [Knu86a] and \LaTeX [Lam86]. (This report was written using \LaTeX .) \TeX is sometimes used as a synonym for plain \TeX .

The characters in a BAGS text object are taken from either a literal string or a dvi file.

For example: to generate the following virtual page



we would say something like this in the `Flesh` script:

```
foo:  rep           0 = text
      resolution    0 = [20]
      scale         0 = [2,2,2]
      color         0 = [1,0,0]
      textstuff     0 = [ "", "bar.dvi", TEXT_DVI ]
      ;
```

where 'bar.tex' (the \LaTeX file from which 'bar.dvi' is generated) looks like:

```
\documentstyle[bagstext]{article}
\begin{document}
  \begin{center}
    {\Large\sl The Brown Computer Graphics Group}\\
    \ \ \
    {\it           presents}\\
    \ \ \
    {\LARGE\sf           Text Objects}\\
    \ \ \
    {\large Who is Text and to what does he object?}\\
  \end{center}
\end{document}
```

The material between the `\begin{center}` and the `\end{center}` is the source for the above example. The `bagstext` style in the `\documentstyle` line sets up the correct set of fonts for use with BAGS text objects.

There are two variants of `textstuff`:

```
textstuff [(font), (string), TEXT_LITERAL]
textstuff [(page-specifier), (dvi-filename), TEXT_DVI]
```

The first form is used to produce literal strings; the second reads a page from a dvi file.

3.2 Pros and Cons of \TeX as a Formatting Language

Advantages of using \TeX and \LaTeX include:

- The pages can be previewed and quickly debugged using any of the several standard dvi previewers. The edit-compile-run cycle for a small \LaTeX file is considerably shorter than that for a `Flesh` script.
- Switching fonts, weights, styles, etc. is trivial (if suitable font metric data is set up).
- The user has control over leading (space between lines), formatting, and justification.
- There is built-in pairwise kerning (adjusting the spacing between pairs of letters, such as moving 'AV' closer together to form 'AV') and ligatures (converting pairs of letters such as 'f' and 'i' to 'fi').
- Specifying attributes of the text that make no sense in more conventional \LaTeX environments (e.g., laying a string along a duct) can be achieved by writing a few macros which make use of \LaTeX 's `\special` primitive.
- \TeX has its own powerful macro language.

Disadvantages of using $\text{T}_{\text{E}}\text{X}$ and $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ include:

- $\text{T}_{\text{E}}\text{X}$ is not interactive.
- $\text{T}_{\text{E}}\text{X}$ is not easy to program (as distinct from using a canned set of macros such as $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$.)
- Track kerning (the overall spacing between letters) cannot be easily adjusted.
- $\text{T}_{\text{E}}\text{X}$ places boxes (lists of characters, rules, and boxes) in horizontal rows across a page. It cannot run text along non-horizontal paths or wrap text around non-rectangular figures.

3.3 Interactive Text Editor

A WYSIWIG editor that interacts with the modellers would certainly be easier to use than writing *Flesh* scripts containing references to *dvi* files, although $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ is quite straightforward to use for simple purposes.

Time did not permit the implementation of a suitable text editor.

3.4 Three Dimensions: Arbitrary Positions and Orientations

To place text at arbitrary positions with arbitrary orientations, the virtual page can be mapped onto a patch surface. The position of a string in 3-space would be determined by the (x, y, z) value of the patch surface at the (u, v) value to which the coordinates of the string on the virtual page mapped. The orientation of the string would be determined by the frame (tangent, normal, and binormal) of the point (u, v) on the patch surface.

This awaits a working patch modeller.

4 Incorporation into BAGS

Text objects have been added to BAGS and they work with the new system. Adding text objects entailed:

1. Generating TRIPobjects from the Bitstream characters (and incidentally making the TRI triangulation package work with polygons which contain holes and islands).
2. Text Layout:
 - (a) Building a dvi parser which takes a page of text and generates text objects on a plane in a scene.
 - (b) Writing a \LaTeX style file ('bagstext.sty') which sets up the appropriate fonts and \backslash specials (currently none).
 - (c) Generating tfm² files and bitmaps of characters for previewing.
3. Integrating the text objects with the new system.

4.1 Adding New Fonts and Representations

Adding a new Bitstream Compressed Outline font to BAGS should be the work of a few minutes—an entry needs to be added to the internal list of fonts (in the `TYPO_open_font()` routine), and two shellscripts need to be run to generate a tfm file and the bitmapped fonts for the previewer.

Adding a font in a different format would entail writing a back end to generate outline curves in the internal format used by the text object routines.

Adding a new representation (prisms, surface patches, etc.) for text objects is more difficult, but not excessively so. A routine to take the outline curves and convert them into the new form must be written. This routine should be modelled upon the existing routine which polygonizes and extrudes text objects.

4.2 Code Organization

The code exists in two packages within BAGS: `TEXT` and `TYPO`. The `TEXT` package is just the object class routines needed by the new system. `TYPO` contains all of the interesting code for converting Bitstream characters into

²tfm= \TeX Font Metrics. A tfm file contains the width, height, and depth of each character, kerning information, and instructions on how to construct ligatures. \TeX needs this data so it can lay characters out on a page. The dvi parser also needs this data, as does the routine which turns a literal string into a TRIPobject.

TRIPobjects and for parsing dvi files. There are three subdirectories within the TYPO 'src' directory: 'bitstream', 'dvi', and 'test', with the obvious distribution of functionality. The data files—Bitstream Compressed Outline fonts, tfm files, bitmap fonts, and 'bagstext.sty'—can be found in TYPO's 'data' directory.

See the 'typo(3B)' manual page for details on generating dvi files.

5 Conclusions

There is no One True Way to generate text objects. Choices and tradeoffs had to be made.

Bitstream Compressed Outline fonts were used because (a) they are scaleable and high quality, and (b) we had a sample set of data at hand.

Text objects were converted into TRIPobjects because (a) we needed them to be TRIPobjects for the polygonal modellers, and (b) no other set of primitives (prisms or patches) is currently available. The TRIPobjects ray trace well when they are generated at a reasonably high resolution, but should produce better-looking results and require less (ray-tracing) computation if the text objects were represented by prisms or patches.

The characters in a text object can be specified by either a literal string or the contents of a dvi file. A literal string is sufficient for a one-dimensional short phrase, but inadequate for a two-dimensional page. Generating a typeset page of text is an inherently complex task and one that \TeX already does very well, far better than any text formatter that one could hope to build in a short amount of time.

5.1 Future Work

In descending order of importance:

- Obtain a proper set of fonts. The sample data is sufficient for a proof-of-concept, but the lack of vowels and other characters makes the fonts worthless for real use.
- Implement a ray-intersection form for the text objects (prisms or patches). This will give better-looking results (curves will be smooth) and will render faster, and use less memory.

- Write a simple WYSIWIG editor for the artists.
- Map a virtual page to a patch surface.
- Special effects such as bevelled edges, tubular characters, etc.

5.2 Summary

The current implementation is functionally adequate for most uses: slides, captions, and flying logos. Using \TeX as a front end provides a powerful means for laying out text, but it is not interactive, so the less technically inclined artists will find it harder to use than a WYSIWIG editor.

Aesthetically, $\text{\text{TRIPobjects}}$ are more-or-less satisfactory when generated at high resolution, but a ray-intersection form would be superior.

For more advanced uses, mapping a page onto a patch surface probably provides all the necessary functionality. It is hard to say how easy this would be to use in practice.

References

- [Ado85a] Adobe Systems Incorporated. *POSTSCRIPT Language Reference Manual*. Addison-Wesley, 1985.
- [Ado85b] Adobe Systems Incorporated. *POSTSCRIPT Language Tutorial and Cookbook*. Addison-Wesley, 1985.
- [Ado90] Adobe Systems Incorporated. *Adobe Type 1 Font Format*. Addison-Wesley, 1990.
- [AH89] Jacques André and Roger D. Hersch, editors. *Raster Imaging and Digital Typography*. Cambridge University Press, October 1989.
- [BBB87] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1987.
- [Bit84] Bitstream. Bitstream guide to fonts: Outlines. [Confidential] Definitive Outlines, November 1984.
- [Bit87a] Bitstream. Bitstream guide to fonts: Bézier outlines. [Confidential], 1987.
- [Bit87b] Bitstream. Bitstream guide to fonts: Compressed outlines. [Confidential], June 1987.
- [Far90] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, second edition, 1990.
- [FK82] David R. Fuchs and Donald E. Knuth. Optimal font caching. Technical Report STAN-CS-82-901, Stanford University, March 1982.
- [FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, second edition, 1990.
- [Gla90] L. Brett Glass. Font wars. *Byte*, pages 289–295. August 1990.

- [Her67] Allen V. Hershey. Calligraphy for computers. Technical Report 2101, United States Naval Weapons Laboratory, Dahlgren, Virginia, August 1 1967.
- [Her72] Allen V. Hershey. A computer system for scientific typography. *Computer Graphics and Image Processing*, 1(4):373–385, December 1972.
- [Hob89] John D. Hobby. A METAFONT-like system with POSTSCRIPT output. *TUGboat*, 10(4):505–512, 1989. TUG Conference Proceedings.
- [Hua90] Nathan T. Huang. *A Guide to TRIP*. Brown University Computer Graphics Group, October 1990.
- [Hub90] Philip M. Hubbard. Constructive solid geometry for triangulated polyhedra. Technical Report CS-90-07, Brown University, September 1990.
- [Kaj83] James T. Kajiya. New techniques for ray tracing procedurally defined objects. *ACM Transactions on Graphics*, 2(3):161–181, July 1983. An earlier version of this paper appeared in *Proceedings of SIGGRAPH '83*, published as *Computer Graphics*, Vol. 17, No. 3.
- [Knu86a] Donald E. Knuth. *The T_EXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, 1986.
- [Knu86b] Donald E. Knuth. *T_EX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, 1986.
- [Knu86c] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, 1986.
- [Knu86d] Donald E. Knuth. *METAFONT: The Program*, volume D of *Computers and Typesetting*. Addison-Wesley, 1986.
- [Knu86e] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, 1986.
- [Lam86] Leslie Lamport. *L^AT_EX: A Document Preparation System*. Addison-Wesley, 1986.

- [Nai88] Avi Naiman. Generating and modeling grayscale characters. In [SIG88], pages 13-1—13-30, August 1988.
- [NF87] Avi Naiman and Alain Fournier. Rectangular convolution for fast filtering of characters. In *Proceedings of SIGGRAPH '87*, published as *Computer Graphics*, Vol. 21, No. 3, pages 229-239, July 1987. Reprinted in [SIG88].
- [PS83] Michael Plass and Maureen Stone. Curve-fitting with piecewise parametric cubics. In *Proceedings of SIGGRAPH '83*, published as *Computer Graphics*, Vol. 17, No. 3, pages 229-239, July 1983.
- [Rub88] Richard Rubinstein. *Digital Typography*. Addison-Wesley, 1988.
- [SIG87] SIGGRAPH '87 Course Notes, Volume 2. Documentation graphics, July 1987.
- [SIG88] SIGGRAPH '88 Course Notes, Volume 14. Digital typography, August 1988.
- [Str88] Paul S. Strauss. BAGS: The Brown Animation Generation System. Technical Report CS-88-22, Brown University, 1988.
- [War80] John E. Warnock. The display of characters using gray level sample arrays. In *Proceedings of SIGGRAPH '80*, published as *Computer Graphics*, Vol. 14, No. 3, pages 302-307, July 1980. Reprinted in [SIG88].
- [YB90] Shimon Yanai and Daniel M. Berry. Environment for translating METAFONT to POSTSCRIPT. *TUGboat*, 11(4):525-541, 1990.
- [ZCv+91] Robert C. Zeleznik, D. Brookshire Connor, Andries van Dam, Matthias M. Wloka, Daniel G. Aliaga, Nathan T. Huang, Philip M. Hubbard, Brian Knep, Henry E. Kaufman, and John F. Hughes. An object-oriented framework for the integration of interactive animation techniques. In *Proceedings of SIGGRAPH '91*, to be published as *Computer Graphics*, Vol. 25, No. 3, July 1991.