G3 – A Language for Typesetting Three Dimensional Graphics

Andrei G. Yaneff
Trevor I. Fenner

Department of Computer Science,
Birkbeck College,
University of London,
Malet Street,
London, WC1E 7HX,
England
ubacw65@cs.bbk.ac.uk

ABSTRACT

A language for typesetting three-dimensional graphics as an integral part of a document preparation system is presented. g3 can handle a variety of 3D objects such as polylines, polygons, solids of revolution and extrusion, as well as parametrically defined curves and surfaces. Arbitrary 3D views can be specified. The system can be used to typeset 3D illustrations of hierarchically specified graphical objects and for 3D function plots.

1. Introduction

UNIX provides a comprehensive set of text formatting and processing tools. The basic formatter is troff (and nroff) for normal text formatting, tbl for table preparation, eqn for typesetting mathematical formulae, pic for creating simple line drawings and grap for generating two dimensional function plots. These are very popular and are used extensively; however, many users need the inclusion of 3D drawings and function plots in their documentation. The most popular approach for doing this is to create the required figure using a 3D CAD or drawing program, and then incorporate it into the troff document as a bitmap file. For example, in our department a patched troff (psroff) can be used to import encapsulated PostScript files into the text. A more natural and considerably less expensive approach would be to have a preprocessor for creating 3D figures, which could be used in the same manner as pic or grap.

This paper describes g3 – a language for typesetting 3D graphics currently under development in this department. Eventually it will be used for generating 3D function and data plots, as well as the creation of complex 3D figures to be typeset in conjunction with the standard UNIX document preparation tools.

2. Design Criteria

Identifying the exact needs of a user of a 3D graphics typesetting language is difficult. Some possible applications are 3D function plots and data charts, architectural drawings, molecular models, etc. With this in mind, there are a number of points that must be considered in the design of a 3D graphics system in the context of UNIX document preparation. Some of these are:

(a) 3D graphics language
The graphics language should allow the specification of arbitrary views and support geometric transformations. Control of the appearance or “quality” of 3D objects should also be possible.

(b) Picture structure
The picture structures supported in most 3D CAD systems are either 2-level or N-level picture hierarchies. Although a 2-level hierarchy is easier to implement, an N-level hierarchy offers certain advantages:

(i) It enables the user to specify complex 3D figures in a consistent and modular way;

(ii) It facilitates the production of documents which require an illustration of a figure as a whole and then wish show components separately, or emphasize them with respect to the rest of the figure, in subsequent illustrations.
3. The g3 Language

g3 is built around a general-purpose 3D graphics programming language. It allows the specification of arbitrary views and supports a comprehensive set of 3D graphics drawing functions. The formats of the graphical types are in most cases self-explanatory and are given below. Reserved words are shown in bold and non-terminal symbols are shown in italics.

Similar to grap, g3 is used as a preprocessor to pic:

```
g3 file [] [graph] [pic] [tbl] [eqn] [troff]
```

Output of g3 is at present pic source code. The g3 commands appear between .G3 and .GE instructions. To facilitate mixing of 3D figures with 2D drawings .PS can also be used instead of .GE. The following example illustrates the use of the .G3 and .GE pair:

```
.G3
pl is polyline ((0,0,0), (1,1,0), (2,1,0), (3,0,0))
p1
p2 is polyline
  ((1,1.5,-1), (1,2,-2), (2,2.5,-3), (2,3.0,-4),
   (3,3.5,-5), (3,4,-6), (4,4.5,-7), (4,5,-8))
translate 1 1 0
p2
viewport (-1,-1,-1)
window (-5,9,-5,9)
projection parallel (1,1,1)
.GE
...
.G3
translate (-5,0,0)
extrude pl p2
.GE
...
.G3
newpicture
....
.GE
```

This example illustrates the point that a picture is retained throughout the document until it is explicitly cleared by a newpicture command (in the third .G3 call). This can be used to provide various views of an object and to add new elements to it for subsequent illustrations. Figure 1 shows the picture as specified at the end of the second g3 block.

In the sequel we use the following non-terminals:

(a) `expr` is a real valued expression.

(b) `xyz_expr` is an expression of the type (`expr`, `expr`, `expr`). These can be combined in vector expressions using the standard scalar multiplication, addition, subtraction and cross product operations. It is also used to specify 3D points.

(c) `interval` is a pair of the type (`expr_min`, `expr_max`).

(d) `point_list = xyz_expr | xyz_expr, point_list`

(e) `point_set = (point_list) | (point_list), point_set` — but is restricted to be a rectangular matrix of points.

(f) `curve` is similar to `point_list`, since all curves are eventually approximated by a sequence of connected line segments, which in turn can be represented by a sequence of points. A `curve` can be specified either by an explicit `point_list` or parametrically (see Section 3.2).
Arithmetic expressions can be assigned to variables, which can be used in other expressions in the usual way:

\[
\text{variable} = \text{expr} \mid \text{interval} \mid \text{xyz}_\text{expr}
\]

3.1. 3D Output Primitives

As in most 3D graphics programming languages, at the lowest level lines, markers, polygons and text are supported. The respective commands for these are

- **lineto** \text{xyz}_\text{expr}
- **moveto** \text{xyz}_\text{expr}
  - moves the current position with/without drawing a line.
- **polyline** \text{point}_\text{list}
  - a set of connected line segments defined by a point sequence (\text{point}_\text{list}).
- **polygon** \text{point}_\text{list}
  - a single closed polygon.
- **polymarker** \text{marker}
  - a predefined marker (usually two-dimensional, e.g. an arrow head) placed at the current position.
- **text** \text{string}
  - a string placed at the current position.
- **grid** \text{point}_\text{set}
  - a set of polygons generated by connecting the adjacent points of the \text{point}_\text{set}

3.2. Curves and Surfaces

Curves and surfaces can be of two types:
(a) predefined – generated from a set of data points using a selected interpolation technique;
(b) parametrically defined.

Parametric curves and surfaces can be defined in much the same way as one finds them in mathematical texts, i.e. resembling functions with parameter lists and bodies, with given ranges for the parameters. All ranges are specified as optional \text{intervals}, with default values of (0,1).

\[
\text{curve}(\text{variable}).\text{xyz}_\text{expr}[\mid \text{interval}]
\]
  - the curve is generated by evaluating \text{xyz}_\text{expr} for values of \text{variable} in the \text{interval} (= (\text{expr}_{\text{variable}_\text{min}}, \text{expr}_{\text{variable}_\text{max}})).
surface(variable_0,variable_1). xyz_expr(:{interval_0, interval_1})

-- the surface is generated by evaluating xyz_expr for values of variable_i in interval_i.

The type of facets generated by the surface command can be selected with the facets command:

facets number

-- the allowed values for number of facets are 3 for triangular and 4 for rectangular facets.

The system supports two predefined types: bezier and spline curves/surfaces. These can be requested in a similar fashion with the reserved words bezier or spline placed after the curve or surface commands:

curve bezier | spline point_list [:interval]

surface bezier | spline point_set [:{interval_0,interval_1}]

For the predefined curves/surfaces no variables are specified, i.e. they are implicit and the optional intervals apply to them. The default (0,1) will cause the entire curve/surface to be generated.

3.3. Surfaces of Extrusion and Revolution

g3 allows the creation of surfaces of extrusion and revolution.

revolve curve_outline([(angle_start] angle_end])

-- the outline curve is revolved around a specified vector from angle_start through angle_end. The initial zero angle is the present position of the outline curve. The defaults are 0 and 2π for angle_start and angle_end, respectively.

To specify an arbitrary axis of rotation for subsequent revolve operations, we use:

revaxis xyz_expr_direction [xyz_expr_start]

-- xyz_expr_direction is essentially a direction vector and xyz_expr_start denotes the starting position of that vector. The default axis of revolution is the y-axis (revolve(0,1,0)(0,0,0)).

Extrusion in CAD systems is done along a vector. The extrusion here is a slight generalization of this – it may be done along any curve.

extrude curve_outline curve_extrusion [expr_scale] [interval_extrusion]

-- the scaling expression is applied to the outline at each slice of the resulting surface. The default scaling is 1. The interval_extrusion is the section of curve_extrusion along which the outline is extruded. The default is (0,1), i.e. the total length of the curve.

An outline is extruded by having one of its points following the extrusion curve precisely. Selection of this principle point is done with the command:

epoint expr

-- expr must be in the range (0,1) – denoting the first and last points of the curve, respectively. The default is the first point of the outline curve (epoint0).

For both revolve and extrude, the outline curves may be non-planar. It is, however, recommended that all points of curve_outline are in the same plane, although sometimes this may be inconvenient to specify and check.

The following example produces the picture in Figure 2.

.G3
translate(1,0,0)
postrotate (0,90,0)

outline is curve bezier
   (1,0,0),(1.5,0.5,0),(0.5,2.5,0),
   (0.5,3.5,0),(1,3.5,0),(1.2,3.5,0)

outline

slice 1/12
revolve outline
window(-3,3,-1.5,4.5)
projection parallel (3,2,1)
vpn(-3,-2,-1)
.GE
3.4. Geometric Transformations

Each graphical object generated using the above functions is subjected to transformation by the CTM (Current Transformation Matrix) whenever instances of the object occur. The operations affecting the CTM are rotation, translation and scaling. These can be applied in relative and absolute modes, so that geometric transformations can be applied in any order, e.g. translate and then rotate, or rotate first and then translate, etc. The commands for the absolute transformations are:

\begin{verbatim}
translate xyz_expr
rotate xyz_expr
scale xyz_expr
\end{verbatim}

In addition to these, there are commands for premultiplication and postmultiplication of the CTM for relative transformations:

\begin{verbatim}
pretranslate, prerotate, prescale, posttranslate, postrotate and postscale.
\end{verbatim}

The system maintains a stack of transformation matrices. It is used for temporary storage of the CTM, as well as for swapping the ITM (Instance Transformation Matrix) of graphical objects (see Section 3.5) and manipulation of the viewing matrix (see Section 3.6). The commands are

\begin{verbatim}
push ctm | view | label
\end{verbatim}

and

\begin{verbatim}
pop ctm | view | label
\end{verbatim}

For \texttt{view} see Section 3.6, and for \texttt{label} see the following subsection.

3.5. Naming Objects and Macro Definitions

In order to avoid repetition in the creation of a 3D figure, a suitable scheme for naming objects is provided. For example, to draw a parametric unit sphere we can use:

\begin{verbatim}
surface (u, v) . (cos(u) * cos(v), sin(u) * cos(v), sin(v)) : ((0, 2 * PI), (0, 2 * PI))
\end{verbatim}

This, however, is quite unsatisfactory if we have several such spheres in the picture, i.e. this code would have to be repeated several times. A more convenient approach is to give a name to that surface and create instances of it whenever its name is given, as follows:
Figure 3: The two instances of the unit_sphere

.G3
unit_sphere is surface
(u,v).((cos(u)*cos(v),
sin(u)*cos(v),
sin(v)):(0,2*PI),(0,2*PI))
translate (2,2,0)
unit_sphere
translate (0,3,4)
unit_sphere

projection parallel (1,2,3)
vpn (-1,-2,-3)
window (-3,4,-3,4)
display hidden
axes on
.GE

This will create two instances of the unit sphere, with centres on the x-y and y-z planes, respectively (see Figure 3). It should be noted that the first line does not display anything – it is only a definition and is used to compute the 3D data for the sphere. To actually display the unit sphere its name must be given. Instances of symbols copy the already computed 3D data after it is transformed by the CTM.

In order to be able to access instances of an object in subsequent operations, a form of labelling similar to that in pic is used:

translate (2,2,0)
first_sphere: unit_sphere
translate (0,3,4)
second_sphere: unit_sphere

Only labelled instances of objects can be manipulated in subsequent operations. Unlabelled instances become part of the higher symbol definition level (see Section 3.6) and can not be accessed in subsequent operations. Only graphical objects may be labelled.

An ITM is attached to every instance of an object. The ITM is a copy of the CTM prior to the call which instantiates an object.

A simple macro definition scheme is also provided. Instead of specifying a unit sphere we can define the general type sphere as follows:

define sphere as surface
(u,v).($1*cos(u)*cos(v),
$1*sin(u)*cos(v),
$1*sin(v)):(0,2*PI),(0,2*PI))

endif
and then specify the unit sphere as:

    unit_sphere is sphere(1)

We can also specify any sphere in a similar way:

    any_sphere is sphere(radius)

where radius is an expression or previously initialized variable.

Another example of a macro definition is:

    define sphere1 as
    translate ($2,3,4)
    surface (u,v).($1*cos(u)*cos(v),
                $1*sin(u)*cos(v),
                $1*sin(v)) : ((0,2*PI),(0,2*PI))
    endif

This macro, however, cannot be used in a statement of the type:

    unit_sphere is sphere1(1,5,5,5)

because names may only be given to objects and not to arbitrary command sequences. Placing a label in a macro definition will result in an error if the macro is used more than once in the same level of the hierarchy (see below).

3.6. Structuring 3D Objects

Tree hierarchical structure is chosen for two main reasons:

(a) to enable the user to define the 3D figure in a consistent and modular way;
(b) because in many documents requiring 3D figures it is not uncommon to display the figure as a whole and then modify some components in subsequent illustrations.

A general hierarchy is created using symbols. A symbol can contain instances of other symbols or drawing functions. It is defined as

    symbol(statement_list)

where statement_list may contain calls to other previously named symbols, graphical primitives and other g3 functions.

Like objects, symbols may be named and instances of symbols may be labelled.

Having defined a general symbol hierarchy, there are a number of operations that can be performed on it. An instance of an object, a symbol or any of its components may be made invisible, emphasized or fainter. The commands are:

    visibility on|off(label_list)
    emphasis on|off(label_list)

where label_list is a list of label names. To access various levels in a hierarchy, label_0 . label_1 .... label_n-1 may be used.

As with the primitives, an instance of a symbol is displayed when either its name is given or it is specified without a name.

An IAM is attached to every instance of a symbol in the same way as with object instances.

3.7. Specifying a 3D View

The system allows the specification of an arbitrary view. This is done in much the same way as in the CORE and GKS-3D graphics systems. The commands necessary to specify a 3D view are:

    vrp xyz_expr
    - sets the view reference point
    vpn xyz_expr
    - sets the view plane normal
    vup xyz_expr
    - sets the view up vector
projection parallel|perspective $xyz\_expr$

- selects the projection: for parallel projection $xyz\_expr$ denotes the direction of projection; for perspective projections it is the centre of projection (eye point).

viewdist $expr$

- sets the view distance

window($expr_{u_{\min}}, expr_{u_{\max}}, expr_{v_{\min}}, expr_{v_{\max}}$)

- defines the window on the projection plane

Two additional commands are provided for clipping, although these may not be needed for most documentation purposes:

viewdepth $expr_{front\_plane}, expr_{back\_plane}$

- sets the back and front clipping plane, thus defining a 3D truncated viewing pyramid for perspective projections and a parallelepiped for parallel projections. The expressions are signed distances from the view reference point along the view plane normal.

and

depthclip on | off [on | off ]

- enables/disables clipping by the front and back planes.

Effects like zooming in and out and looking at objects from various points in any direction can be obtained by combining the viewing matrix with a suitable translation, rotation, and scaling matrix. The CTM mechanism (see Section 3.4) can be used to do this. The following can be used for this purpose:

```plaintext
call push ctm
push view
pop ctm

... transformations which change the CTM ...

push ctm
pop view
pop ctm
```

Subsequent changes to any of the viewing parameters will generate a new viewing matrix using their current values. The above transformation would no longer apply.

### 3.8. Miscellaneous Commands

#### 3.8.1. Controlling the Amount of Detail

The quality and the amount of detail required in the generation of a picture can be controlled in two ways:

(a) controlling the step size for computations for parametrically defined curves and surfaces;

(b) choosing a suitable contour grid of the picture to be shown.

The step size for computations of parametric expressions can be specified with the command:

```plaintext
calcstep $expr$
```

where $expr$ is between 0 and 1 (calcstep 1 would produce a straight line).

Quality of curves is essentially the length of the line segments used to approximate the curve, i.e. the smaller they are the smoother the appearance. Curve quality depends on calcstep. Surface quality, however, also depends on the number of contours to be displayed. Since parametric surfaces are usually specified in terms of two parameters, $u$ and $v$, say, we need suitable slices for both. We use a normalizing scheme, similar to intervals (defaulting to (0,1)), for this purpose:

```plaintext
call slice $expr_u, expr_v$
```

Both $expr_u$ and $expr_v$ take values in the range (0,1) and should not be less than the step size for the calculations (which was specified using calcstep). The value of $expr$ indicates that only contours close to the value of $N * expr$ for integers $N$ such that $0 \leq N * expr \leq 1$ are displayed.

All figures in this text have been done with $expr_u$ and $expr_v$ equal to the step size.
3.8.2. Display Modes
The system can render pictures using two display models: wire-frame and hidden line representations.

\texttt{display wire \mid hidden}

wire-frame is the default.

3.8.3. Axes
Axes can be displayed using the command:

\texttt{axes on \mid off \mid xyz\_expr}

where \texttt{xyz\_expr} represents the lengths of the respective axes. The axes may be labelled in a fashion similar to \texttt{grap}. Axes are switched off by default.

3.8.4. Attributes
Attributes such as line styles, e.g. solid, dashed, etc., are also available. Similar to most \texttt{g3} commands, they have effect on the current state of \texttt{g3} and only subsequent commands are affected by them.

4. Implementation
Originally \texttt{g3} was developed on a PC. At the time of writing the UNIX version of the system (on a MicroVAX under Ultrix) was not fully operational. The illustrations were computed on a PC and the data was translated (with some difficulty) into \texttt{pic} for inclusion in this text.

3D drawings are difficult to visualize mentally and for this a \texttt{g3p} utility is provided, which displays all \texttt{g3} pictures in a given document on a graphics terminal (other text is ignored).

5. Future Work and Possible Enhancements
The system is still in a state of development; various enhancements are planned, and other modifications and extensions will be incorporated in the light of experience, particularly to improve the user-friendliness of the language.

Currently only wire-frame and hidden line removal are supported; future work may include adding flat surface and Gouraud shading (possibly using PostScript for the output).

Work is also needed in terms of combining 3D pictures. In \texttt{pic} one navigates according to the relative positions of a figure, e.g. nw, w, s, etc., but it is tricky to assign directions in 3D space, especially when arbitrary views are used.

Bibliography
5. AT&T, “UNIX System V Documenter’s Workbench R2.0”, 1986