REC-F for Flexagons

Harold V. McIntosh
Departamento de Aplicación de Microcomputadoras,
Instituto de Ciencias, Universidad Autónoma de Puebla,
Apartado postal 461, 72000 Puebla, Puebla, México.

September 26, 2000

Abstract

REC-F is one of a series of specialized REC programs; in this case one which provides assistance in designing flexagons. Flexagons were popularized by Martin Gardner in *Scientific American* in the early 1960's. It is still possible to read his descriptions inasmuch as the original articles have since been reprinted in various collections of his articles. Basically, a flexagon is a fractal derived from a stack of polygons which can be folded back and forth, preferably coming to rest in a plane; so they resemble dragon curves and Lindenmayer L-systems. The purpose of REC-F is to draw the frieze of polygons which will eventually be folded up to make a given flexagon.

1 Introduction

Three ingredients are required to understand this program. Since it will be used for designing flexagons, a basic familiarity with flexagon design and construction is required. Given that the design is accomplished by using REC, an understanding of that language is evidently required. Finally, the program is implemented in Objective C running under the NeXTSTEP operating system, to take advantage of its excellent visual interface and other attractive features.

1.1 The theory of flexagons

A sketch of their history and instructions for constructing them are most conveniently found amongst Martin Gardner's articles. The original presentation [1] dating from December of 1956, introduced triangle based flexagons, attributing their development to a group of graduate students at Princeton University in the late 1930's. The article was reprinted in his first puzzle collection [2], which is still on sale.

His first *Scientific American* article was followed in May of 1958 by a presentation of similar objects based on squares [3], duly reprinted in a second puzzle collection [4], also currently available. A third offering, oriented more toward "flexatubes" than flexagons, appears in a still later collection [5].

A definitive treatment of flexagons can be found in the RIAS Technical Report [7] of 1962 prepared by Anthony Conrad and Daniel Hartline, two students from Towson High School (near Baltimore, Maryland). Conrad's shorter summary [6] still covers the essential elements.

Practically the only article which delves into a mathematical analysis of flexagons rather than giving a simple description of one of them is the analysis published by C. O. Oakley and R. J. Wisner in *American Mathematical Monthly* in 1957 [8]. Beyond that there are a dozen or so articles scattered through mathematical and recreational publications.

Since the Technical Report was never formally published and RIAS itself has been changed, there are only a few copies in existence, two of which have been deposited in Marhematical Recreations collections[1], and one of which has been optically scanned and placed on our web page[2]. By and large, secondary references seem to have been lost, but those items which did get published can still be found in back issues of the corresponding periodicals. There is also a summary of recent activity [9].

1.2 The programming language REC

The programming language REC was derived from LISP over a period of time, and found to be useful for minicomputers such as Digital Equipment's PDP-8, or early microprocessors such as the IMSAI or Polymorphic 88, given its extreme conciseness. That, in fact, is the reason why it is still preserved as an interface to programs written in other languages running with other operating systems.

REC itself is strictly a control structure, using parentheses to group sequences of instructions which are supposed to be executed in order — in other words, programs. The instructions themselves, which are subdivided into operators and predicates, are not part of the control structure, but are always defined separately to create specialized variants on REC.

Besides the use of parentheses for aggrupation, the two punctuation marks, colon and semicolon, are used for repetition and termination, respectively; they were somewhat borrowed from musical notation. Spaces, tabs, all the `C` "white space," are used for cosmetic purposes and must be ignored.

There is also a mechanism for defining subroutines; definitions and their associated symbols alternate between a balanced pair of curly brackets, terminating in a main program which remains nameless. Definitions so introduced are valid only within their braces, permitting symbols to be reused over and over again on different levels and in different places. A subroutine is actually executed by prefacing its name with an "at sign," as in `@x`.

The distinction between operators and predicates is somewhat artificial, made mostly for convenience; a predicate which is always `true` is an operator. The values `true` and `false` of a predicate govern the sequence of operations, `true` meaning that the text of the program continues to be read in order, consecutively from left to right.

But `false` makes use of the punctuation, implying a skip to the next colon, semicolon, (or right parenthesis, in their absence) at the same parenthesis level. A colon means repetition starting back at the left parenthesis, a semicolon gives a `true` termination, realized by going forward to the closing right parenthesis, Arriving at a right parenthesis without the benefit of punctuation gives a `false` termination, but more generally inverts the action of predicates enclosed in such an interval.

This convention allows negating a predicate: `(p)` behaves oppositely from `p`. The other boolean combinations of predicates are easily written; "p and q" translates into `(pq;)`, "p or q" into `(p;q;)`. Moreover, all programs are predicates, even the main program, thereby

[1] Strens collection: see www.ucalgary.ca/library/SpecColl/strens.htm
[2] http//delta.cs.cinvestav.mx/~mcintosh

reporting their results to the next higher program level, the one in which they occur as a subroutine.

Having a specialized REC for flexagon design requires programming the operations used in the process, which includes generating the base polygon and then drawing a succession of reflections in prescribed sides. Although flexagons can be elaborated recursively to any degree, it is rare that as many as three levels would be used before the physical structure becomes impossible to manipulate. Consequently no terminating predicate has been included; just a specified finite polygon strip will be drawn.

REC-F uses the following ten operators; only the counter `!n!` is a predicate.

Xk - create a k-gon

C - draw the polygon

o - offset the polygon

n - reflect in the previous edge

N - reflect in the next edge

Rk - reflect in edge k

O - read a polygon off the stack

P - push the polygon onto the stack

p - pop the polygon from the stack

!n! - (!n! ... ;) repeat ... n times

For example, `(X5 C;)` would create and display a regular pentagon.

There are several documents available explaining REC [10], including a read.me file in the rec subproject of each of the REC files.

1.3 The NeXTSTEP operating system

One of the principal advantages of the NeXTSTEP operating system is its use of the PostScript language for all input and output. *All* includes displaying information on the monitor, reading and writing files from disk, the expected task of printing files on a printer, and even includes transmitting them by modem. Amongst the agreeable conveniences is not having to struggle with a screen dump to save information which was originally presented visually.

The appearance which NeXTSTEP presents to the user is a *File Viewer* which is a window filled with either icons or text, as the user prefers. Moving a cursor by mouse (or sometimes keyboard arrows) indicates a choice of file, each one of which responds in its own manner to clicking or dragging. If it is a subdirectory, additional details are shown; if it is a text file, a page will be opened for editing, while programs will simply be loaded and executed.

Besides the *TextView* window, almost always overlaid as soon as programs begin to execute, there is room around the margins for icons reminding the user of programs and text files which have been held in abeyance, for frequently used programs, and the like. Some space is reserved in the upper left hand corner for the operational menu of the program being executed.

In addition to the mere operating system, which loads and executes programs and interchanges information between different locations, there is an extensive collection of service programs, ranging from language compilers to spelling checkers and even a copy of the Bible. Two of the more important utilities are the *Program Manager* and the *Interface Builder*.

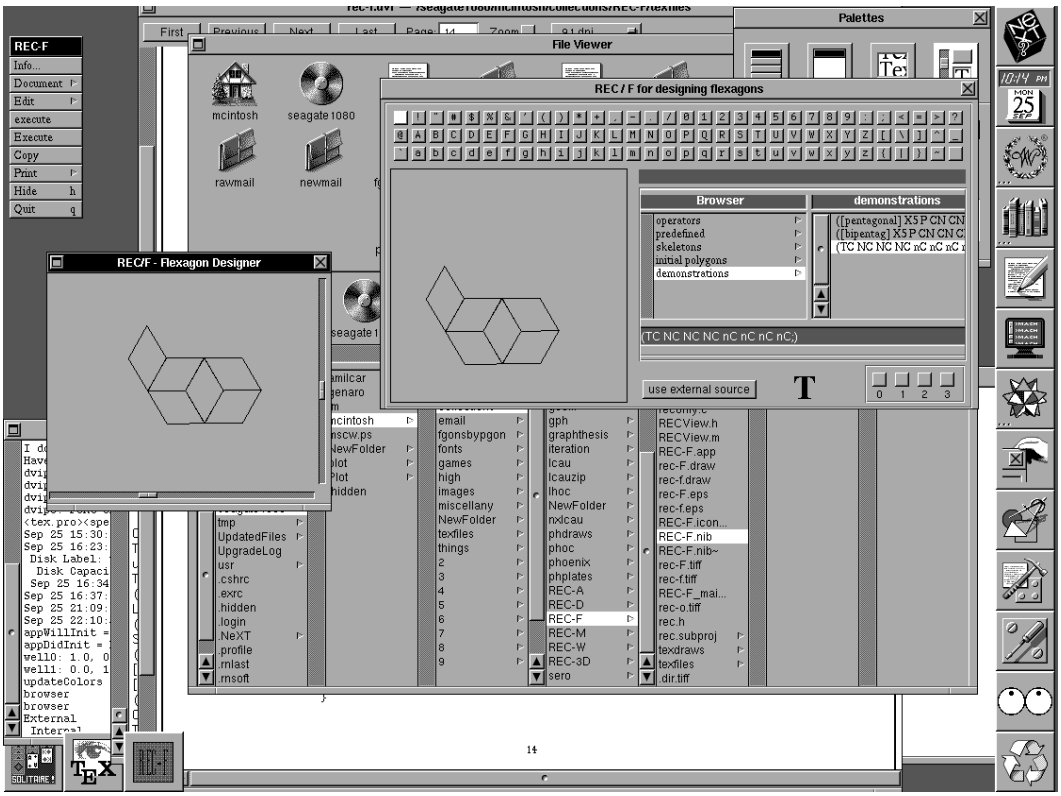

Figure 1: A REC-F program, running against a background consisting of the File Viewer and some other programs.

The *Program Manager* mostly checks dates on files, to be sure that the source code for a program is not more recent than the object code (Beware the Y2K Bug!). To do so, it maintains lists of all kinds of associated material such as header files, icon images, sounds, and what not. The *Interface Builder* permits *Ugh! Grunt! (point) (point) ...* programming, in the sense that windows can be created, stocked with assorted artifacts, and have them interconnected, all through the suitable interpretation of mouse movements. Much programming can be accomplished in those terms, at least during the stages of initial layout.

Going beyond simple tinkering, one needs to use the language Objective C which is a mixture of C^{++} and SmallTalk. The latter uses *Methods*, which are statements with the syntax `[destination operation xxx :argument1 xxx :argument2 ...]`, where `xxx` denotes arbitrary intercalcated text, usually taking the form of helpful comments or comprising pieces of the body of a simple declaration in which the arguments are embedded. The `destination` is an *Object*, which is really a fancy name for a program to be executed[3]. SmallTalk envisioned

[3] *Objects* are actually programs packaged together with data and data structures, independecizing them from one another.

parallel programming, in which several independent programs could communicate with one another by exchanging messages coordinating their activities.

A familiarity with Objective C will be necessary to follow the program listings which are about to be described. In the process it cannot be avoided noticing that NeXTSTEP already includes a vast number of predefined methods destined to manage the user interface. They are all located in a directory, together with supporting information, which can be consulted online, or selectively printed out for further enlightenment.

Figure 2: Main window for REC-F.nib, showing the composition of REC-F.

2 Program Appearance

A user of REC-F will deal mostly with two windows, the main window and the auxiliary display window. There may be other panels and windows; for example the program menu makes its presence known every time the program is executed, and is so ubiquitous that it may not be even be counted amongst the program's windows. The main window of the Interface Builder, which customarily sits in the lower left hand corner of the screen as the interfaces are built, exhibits all the panels, windows, and accessible Objects, as shown for REC-F in Figure 2.

The main window, labelled "execute" in Figure 2, comes with two alternative forms, one of which was shown in Figure 4. The difference between them is that the browser panel can be exchanged for a text panel, into which programs can be read from disk storage, or composed on the spot. Traffic goes both ways, because the contents of the window can be transferred to the disk, either as a new file or as a correction to the existing file. The text handling variant is shown if Figure 3. The button at the bottom of the panel, displaying either the title "use external source," to load the text insert, or "load internal source" to load the browser insert, governs the transition between the two forms.

2.1 Text windows

There are some standard procedures for incorporating text windows in a view of one's own, which are probably best obtained by copying them from a program which already has them, making appropriate changes in file names, storage arrays, and options. Lacking examples or desiring better information, the book of Garfinkel and Mahoney [12] can be consulted. The tricky aspect of the process is that files have to be opened on at least two levels, first in the MACH or C level, and then again on the NeXTSTEP level; correspondingly they must be closed in stages, running along in reverse order.

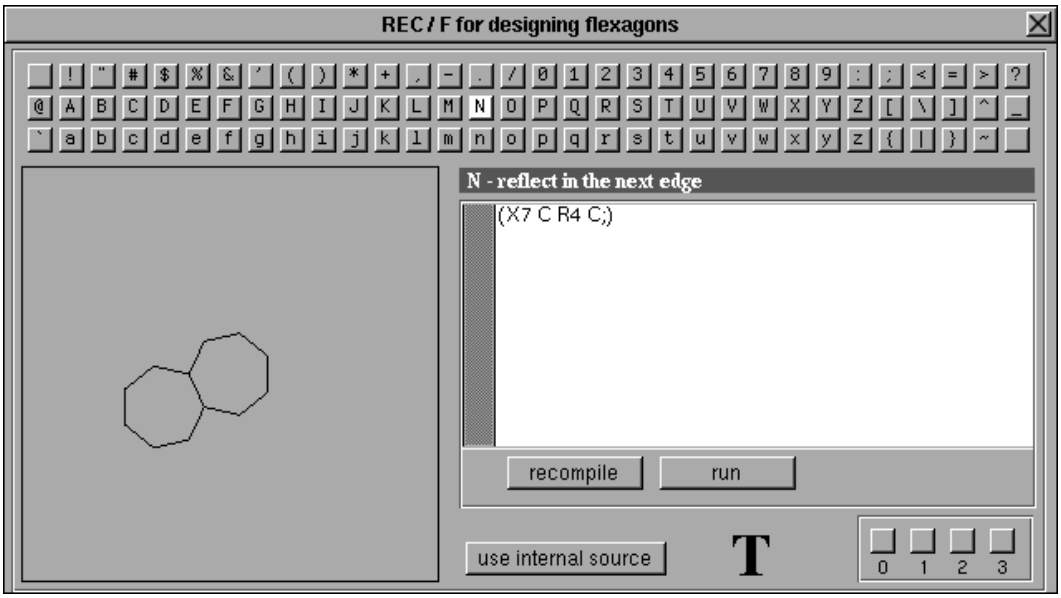

Figure 3: The REC-F main window, which contains the REC directory, a program definition area, and a viewer for the resulting construction.

Any file manipulation should be connected to the item "Document" in the program's menu, which can be seen in Figure 6, so as to preserve the uniformity in appearance of NeXTSTEP programs.

All text fields are automatically connected to an editor. Besides performing the expected functions of inserting, deleting, and searching, they can share information with other pro-

grams through a buffer called the pasteboard. It communicates with editors running in other windows or even from different programs. Besides responding to the keyboard, editing can be accomplished by selecting options in the "Edit" item on the main menu, once having wiped the editing I-beam cursor over a portion of text.

If the "Services" item has been incorporated in a program menu (which has not been done in REC-F), it is possible to have still more elaborate communication between programs, passing selected text either as data or as program instructions.

Of course, there is no reason that a REC program cannot be placed in some file using the editor the regular way, and subsequently executed. The advantage in editing from the REC program itself is that the code can be tested immediately, and saved only after it is performing satisfactorily.

2.2 Browsers

Browsers are another aspect of the operating system which can be incorporated in application programs at the programmer's desire; the procedure is much the same as for coupling text fields to files; namely to copy a browser which is already working, or to look it up in a textbook. The new browser program has to supply the text (or even icons) that the browser will display, which is accomplished by delegation. A delegate can be associated with an object by mouse dragging in the interface builder, or set up through programming.

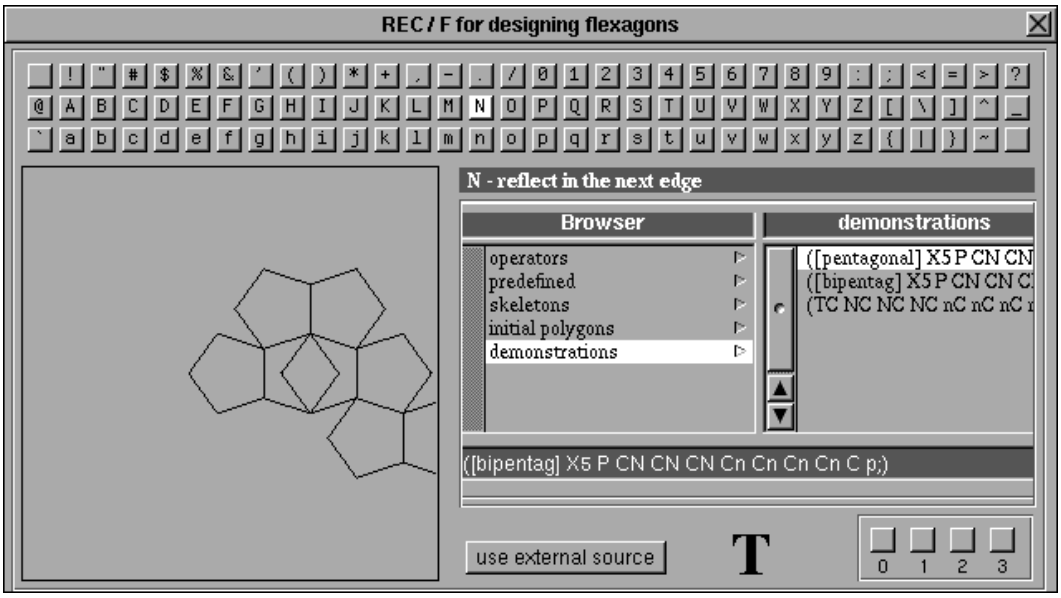

Figure 4: The REC-F main window, which contains the REC directory, a program definition area, and a viewer for the resulting construction.

A delegate is expected to provide methods which could have been incorporated in the delegating program, but for some reason, weren't. The principal reason is that there was no way of forseeing, when the master program was written, exactly what the methods were

supposed to do. After all, each application will have its own items to display, and its own reactions to their selection; such information is best incorporated in the application program itself.

Looking around the REC-F main window, two other text fields can be seen. One, which is alwas present, is used for holding error messages and comments, such as displaying the definition of REC operators. Although there is no requirement within the language to do so, it is customary to use single letters (or ASCII characters) for the operators because of their convenience and ease of remembering. Of course, such a principle only works when there are enough letters to go around, which is one of the reason for tailoring individual REC programs instead of concocting a universal or general purpose REC.

The other text field accompanies the browser to hold REC programs after their selection by a double mouse click. They can be edited, and executed by a carriage return, but the field is not connected to the filing mechanism. Nevertheless, copying and pasting between the big text panel and the browser line is always possible, so the text field in the browser is not completely isolated.

The browser in the main window of REC-F is stocked with information of various types. The first item copies the header file which identifies the operators and predicates in REC-F, a variant of which has to be included in all REC programs. It supplements the button panel for REC definitions, because it is easier to scan a list of possibilities than to press the buttons one by one. On the other hand, for quick recollection or to identify an unfamiliar letter in a displayed program, the buttons are handier.

The remaining items carry either the coordinates of some frequently used basic polygons or instructions for developing particular strips. Since the counter only admits constants as its argument, some editing may be required to adapt a generic "`!n!`" to the given polygon.

In the first case, a double click on the selected polygon will set up the coordinates, in the second, the double click will move the REC program to the nearby text field for editing and execution.

A final component of the main window is a "Custom View" in which the polygons comprising the flexagon are actually drawn, with the help of PostScript functions within the RECView Object.

2.3 Auxiliary window

Drawing pictures brings up a series of problems, not all of which are evident at first. Scale and centering are among the most obvious, easily compensated by including provisions for moving the figure and changing its size. Conflicts can still remain, because excessive compression may make the figure difficult to read, or because the final result has to be presented on a sheet of paper of given size. The next remedy is to divide a figure into panels, or use scrolling, at the price of increased programming complexity.

Rotation and reflection bring up new challenges, although some foresight can accomodate them as well. The simplest solution to figure revision is to introduce homogeneous coordinates, and subject all drawing operations to a projective transformation. Of course that implies matrix multiplication, which may or may not have a deleterious effect on performance. Computers are now sufficiently rapid that simple drawings can receive a fair amount of preprocessing while producing results compatible with human reaction time, which means about twenty frames per second.

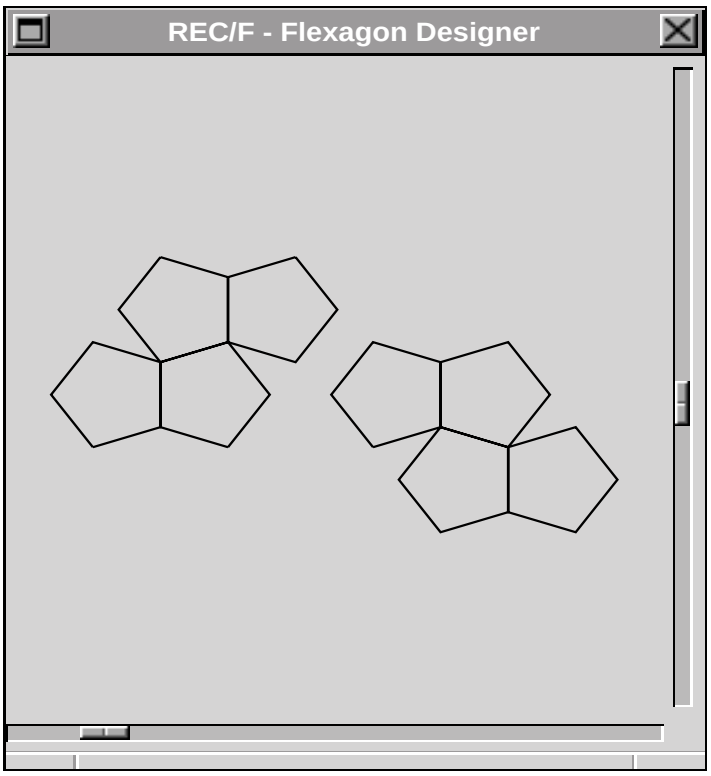

Figure 5: The auxiliary window can be enlarged or reduced by using the window sizer on its bottom margin. The two sliders position the image horizontally and vertically so that it can be kept on the page and centered.

While projective facilities are easy enough to incorporate in newly written programs, they are noticeably lacking in such service programs as Draw with which the new programs may still want to interact.

For the moment, REC-F provides two parallel views, one in the main window, the other in a window of its own. The latter, shown in Figure 5, allows some of the desirable adjustments, and contains an image which may be copied into the pasteboard by invoking the "Copy" button on the main menu, then transferred into Draw by pasting.

Whether two views, one fixed on a general purpose panel, and one adjustable, are really necessary or convenient is something which may be decided in future versions of the program. For the sake of having a program description to consult, we have taken the program as it exists. No doubt it should be considered but a beta version, subject to further modification and adjustment.

2.4 Program menu

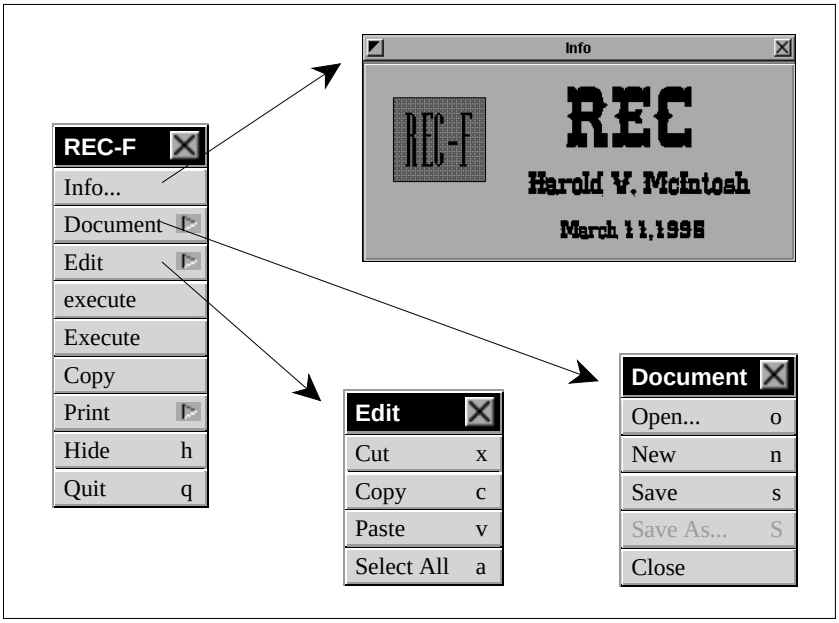

Figure 6: A program's menu will almost always begin with the items Info, Document, and Edit, and end with Hide and Quit. Print would be included as appropriate, usually near the end. Other items vary with the individual program.

Program execution in NeXTSTEP involves double clicking on an item in the FileViewer followed by use of the loaded program's main menu, by custom and tradition situated at the upper left hand corner of the monitor screen. It follows that designing a program would involve setting up a menu for the program, stocking it with commands, and writing (or drawing) the code to implement them. Figure 6 shows the menu for REC-F, with the submenus "Document" and "Edit" exposed for further inspection. The "Info" category is the place to insert the author's name and affiliation, possible copyright notices, and to the degree that seems appropriate or that good programming practice demands, descriptive information concerning the program and how to use it.

3 The REC Header and Program Listing

Besides other headers, such as `rec.h` which serves the REC compiler, and headers germane to a particular application, one more should be provided which is devoted exclusively to defining the REC operators and predicates which are going to be used. Additionally, the operators and predicates themselves must be defined, more likely in C than not; keeping them all together in one place facilitates program maintainance.

3.1 REC header

The header file seems excessively long when only a handful of definitions are actually stated, but in keeping with the tradition of using single ASCII characters as symbols, conserving the full list helps browsers and inspectors work uniformly.

Moreover, insertions or deletions to the table are confined to the locations already alloted for them. Such systematics often reduce error and increase programming convenience. In fact, for some of the more common variants, such as floating point numbers, comments in the form of quoted strings, or frequently used operators, including both forms in the table with the unused line neutralized by making it into a comment can reduce confusion and save programming effort.

This header file is also the place to declare all the REC functions which it introduces — at least the compiling and executing functions — if not all their satellites. Those can probably be accounted for in the REC implementation file with less overall effort.

```
/* rectbl.h -- G. Cisneros, 4.91; rev. 2.10.91 */
/* Mandatory declarations */

# include "rec.h"

int r_lpar();        /* compilation of a left parenthesis            */
int r_rpar();        /* compilation of a right parenthesis           */
int r_colon();       /* compilation of a colon                       */
int r_semicol();     /* compilation of a semicolon                   */
int r_code();        /* compilation of a simple operator             */
int r_oper1();       /* compilation of an operator with an ASCII arg */
int r_pred();        /* compilation of a simple predicate            */
int r_pred1();       /* compilation of a predicate with an ASCII arg */
int r_noopc();       /* compilation of no-op                         */
int r_comment();     /* compilation of a (bracket enclosed) comment  */
int r_ubrack();      /* compilation of an unbalanced right bracket   */
int r_lbrace();      /* compilation of a (brace enclosed) program    */
int r_ubrace();      /* compilation of an unbalanced right brace     */

int r_call();        /* execution of predicate @x (call subr. x)     */
int r_quit();        /* execution of operator _ (exit to O.S.)       */
int r_xbrace();      /* execution of brace-enclosed program          */

/* Optional: Counter predicate */

 int r_ctrc();        /* compilation of a counter */
 int r_ctrx();        /* execution of a counter */

/* Optional declarations, needed only in versions using quotes and
   numbers (see cnum.c and cquo.c); r_ld, r_ldch and r_xstr must
   be provided by user */

// int r_dquote(); /* compile doubly quoted string into symbol table        */
// int r_squote(); /* compile singly quoted char into the program array     */
// int r_cmin();   /* compile '-' as simple operator, call r_cnum if followed
```

```
//                      by digit or period */
// int r_cnum();   /* compile a number (WORD, LONG or REAL) into sym. table */

// int r_ld();        /* execution of compiled numbers               */
// int r_ldch();       /* execution of apostrophe (single quote op.) */
// int r_xstr();       /* execution of quoted strings                 */

// int r_cdblp(), r_ldblp();

/* Declarations of user-provided execution subroutines. */
/* functions have type void, predicates get type int.   */

void ropla(), ropln(), roplo(), roplp();

void ropua(), ropuc(), ropun(), ropuo(), ropup(), ropur(), ropux();

/* Table of compiling/executing definitions, three items per line */

struct fptbl dtbl[] = {
        r_noopc,      FALSE,    " space (separator)",      /* [blank] */
        r_ctrc,    r_ctrx,    " !n! count to n   ",             /* ! */
//        r_dquote,    r_xstr,    "",               /* " quoted string */
        r_noopc,    FALSE,    "",               /* " quoted string */
        r_noopc,    FALSE,    " # -  ",                         /* # */
        r_cdblp,    r_ldblp," $xxx.x$ - floating point number ", /* $ */
        r_noopc,    FALSE,    " % -         ",                  /* % */
        r_noopc,    FALSE,    " &           ",                  /* & */
//        r_squote,    r_ldch,    " quoted character",            /* ' */
        r_noopc,    FALSE,    "         ",          /* '  quoted char */
        r_lpar,     FALSE,    " ( - start of expression",       /* ( */
        r_rpar,     FALSE,    " ) - end of expression",         /* ) */
        r_noopc,    FALSE,    " * ",      /* * */
        r_noopc,    FALSE,    " + ",      /* + */
        r_noopc,    FALSE,    " , ",      /* , separator */
//        r_cmin,    FALSE,    " - ",      /* - */
        r_noopc,    FALSE,    " - ",      /* - */
//        r_cnum,    r_ld,    " , ",      /* . f.p. number */
        r_noopc,    FALSE,    " . ",      /* . */
        r_noopc,    FALSE,    " / ",      /* / */
//        r_cnum,    r_ld,    " 0 ",      /* 0 number */
//        r_cnum,    r_ld,    " 1 ",      /* 1 number */
//        r_cnum,    r_ld,    " 2 ",      /* 2 number */
//        r_cnum,    r_ld,    " 3 ",      /* 3 number */
//        r_cnum,    r_ld,    " 4 ",      /* 4 number */
//        r_cnum,    r_ld,    " 5 ",      /* 5 number */
//        r_cnum,    r_ld,    " 6 ",      /* 6 number */
//        r_cnum,    r_ld,    " 7 ",      /* 7 number */
//        r_cnum,    r_ld,    " 8 ",      /* 8 number */
//        r_cnum,    r_ld,    " 9 ",      /* 9 number */
        r_pred,     rprze,    " 0 - switch ",      /* 0 */
        r_pred,     rpron,    " 1 - switch ",      /* 1 */
```

```
r_pred,     rprtw,    " 2 - switch ",      /* 2 */
r_pred,     rprth,    " 3 - switch ",      /* 3 */
r_noopc,    FALSE,    " 4 ",      /* 4 number */
r_noopc,    FALSE,    " 5 ",      /* 5 number */
r_noopc,    FALSE,    " 6 ",      /* 6 number */
r_noopc,    FALSE,    " 7 ",      /* 7 number */
r_noopc,    FALSE,    " 8 ",      /* 8 number */
r_noopc,    FALSE,    " 9 ",      /* 9 number */
r_colon,    FALSE,    " : - repeat from opening lparen",/* : iterate */
r_semicol,  FALSE,    " ; - T exit at rparen", /* ; true termination */
r_noopc,    FALSE,    " <           ",           /* < */
r_noopc,    FALSE,    " =           ",           /* = */
r_noopc,    FALSE,    " >           ",           /* > */
r_noopc,    FALSE,    " ? -         ",           /* ? */
r_pred1,    r_call,   " @x - call subroutine x", /* @ */
r_noopc,    FALSE,    " A -          ",          /* A */
r_noopc,    FALSE,    " B -          ",          /* B */
r_code,     ropuc,    " C - draw the polygon ",  /* C */
r_noopc,    FALSE,    " D -          ",          /* D */
r_noopc,    FALSE,    " E -          ",          /* E */
r_noopc,    FALSE,    " F -          ",          /* F */
r_noopc,    FALSE,    " G            ",          /* G */
r_noopc,    FALSE,    " H -          ",          /* H */
r_noopc,    FALSE,    " I -          ",          /* I */
r_noopc,    FALSE,    " J -          ",          /* J */
r_noopc,    FALSE,    " K -          ",          /* K */
r_noopc,    FALSE,    " L -          ",          /* L */
r_noopc,    FALSE,    " M -          ",                          /* M */
r_code,     ropun,    " N - reflect in next edge",               /* N */
r_code,     ropuo,    " O - read top of polygon stack",          /* O */
r_code,     ropup,    " P - push onto the polygon stack ",       /* P */
r_noopc,    FALSE,    " Q -        ",                            /* Q */
r_oper1,    ropur,    " Rn - reflect in edge n",                 /* R */
r_noopc,    FALSE,    " S -        ",                            /* S */
r_noopc,    FALSE,    " T -        ",                            /* T */
r_noopc,    FALSE,    " U -        ",                            /* U */
r_noopc,    FALSE,    " V          ",                            /* V */
r_noopc,    FALSE,    " W          ",                            /* W */
r_oper1,    ropux,    " X - Xn puts regular n-gon at origin", /* X */
r_noopc,    FALSE,    " Y -         ",                          /* Y */
r_noopc,    FALSE,    " Z -         ",                          /* Z */
r_comment,  FALSE,    " [ - [begin comment]", /* [ */
r_noopc,    FALSE,    "        ",             /* \ */
r_ubrack,   FALSE,    " ] - [end comment] ",  /* ] */
r_noopc,    FALSE,    " ^ -        ",         /* ^ */
r_code,     r_quit,   " _ - abandon ship",    /* _ */
r_noopc,    FALSE,    "             ",        /* ` */
r_noopc,    FALSE,    " a -         ",        /* a */
r_noopc,    FALSE,    " b -         ",        /* b */
r_noopc,    FALSE,    " c -         ",        /* c */
r_noopc,    FALSE,    " d -         ",        /* d */
```

```
        r_noopc,    FALSE,    " e -          ",        /* e */
        r_noopc,    FALSE,    " f -          ",        /* f */
        r_noopc,    FALSE,    " g -          ",        /* g */
        r_noopc,    FALSE,    " h -          ",        /* h */
        r_noopc,    FALSE,    " i -          ",        /* i */
        r_noopc,    FALSE,    " j -          ",        /* j */
        r_noopc,    FALSE,    " k -          ",        /* k */
        r_noopc,    FALSE,    " l -          ",        /* l */
        r_noopc,    FALSE,    " m -          ",        /* m */
        r_code,     ropln,    " n - reflect in last edge  ", /* n */
        r_code,     roplo,    " o - offset the polygon    ", /* o */
        r_code,     roplp,    " p - pop the polygon stack ", /* p */
        r_noopc,    FALSE,    " q -          ",        /* q */
        r_noopc,    FALSE,    " r -          ",        /* r */
        r_noopc,    FALSE,    " s -          ",        /* s */
        r_noopc,    FALSE,    " t -          ",        /* t */
        r_noopc,    FALSE,    " u -          ",        /* u */
        r_noopc,    FALSE,    " v -          ",        /* v */
        r_noopc,    FALSE,    " w -          ",        /* w */
        r_noopc,    FALSE,    " x -          ",        /* x */
        r_noopc,    FALSE,    " y -          ",        /* y */
        r_noopc,    FALSE,    " z -          ",        /* z */
        r_lbrace,   r_xbrace, " { - start of program",/* { */
        r_noopc,    FALSE,    " | -    ",             /* | */
        r_ubrace,   FALSE,    " } - end of program",  /* } */
        r_noopc,    FALSE,    " ~.- "                 /* ~ */
};
```

3.2 REC-F data flow

Figure 7 shows the data flow in REC/F, which is not very extensive. There is essentially one array, of ample dimension, which will hold the coordinates of any polygon which is likely to be used. The coordinates are stored in projective form, because that is also the way that the coefficients of lines will be stored and maintains duality. This array acts as an accumulator, which can be modified, to and from which data can be transported, and which is the reference for figure drawing.

There is also a pushdown list, not much used by REC/F, in which data can be stored or read, and eventually removed as it is recovered. Finally, there is a graphing area, in which the polygons are drawn, one by one. The auxiliary area copies the first, but changing the position of its image reflects back in the small view because there is an additional coordinate adjustment. This latter will probably get replaced by a full projective transformation of the plane, being the most convenient place to overcome the limitations on reflecting and rotating inherent in Draw and other programs.

There are some operators which generate polygons, the most important of which is `Xn` for generating a regular n-gon with a vertical rightmost edge. It is the only one formally listed in the `rectbl` header, but as time goes on and interest centers on particular irregular polygons, more will undoubtedly be added.

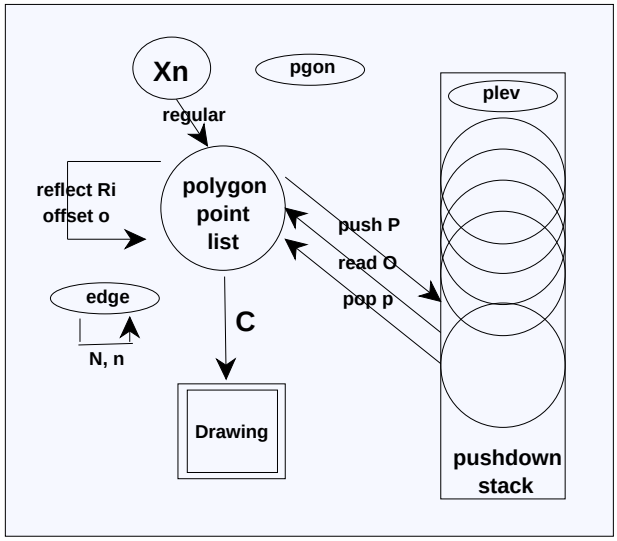

Figure 7: Data movements caused by the REC operators.

3.3 REC operators and predicates

The only thing complicated amongst the REC definitions is the scheme for reflecting a polygon in one of its edges. To begin with, the formula for a line between two points (x_1, y_1) and (x_2, y_2) in a plane is conveniently written in in the form

$$(x_2 - x_1)y \quad = \quad (y_2 - y_1)x + (y_1 x_2 - y_2 x_1).$$

Matrixwise, the formula for reflecting the point (x_0, y_0) in the line $ax + by + c = 0$ is

$$\begin{bmatrix} x_0' \\ y_0' \end{bmatrix} \quad = \quad \frac{1}{a^2 + b^2} \left(\begin{bmatrix} b^2 - a^2 & -2ab \\ -2ab & a^2 - b^2 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} - \begin{bmatrix} 2ac \\ 2bc \end{bmatrix} \right).$$

These two formulas have been transcribed into C and coded into the function `refen(n)`, which consults the list of vertices of the current polygon to get edge n to perform the reflection.

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*            definition of REC operators and predicates                   */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

int rec(z) char *z; {if (rec_c(z,rprg,PLEN,dtbl)==1) rec_x(rprg);}

void ropln() {edge--; refen((edge+pgon) % pgon);}   /* reflect in last edge */
```

```
void roplo() {int i;       /* offset the polygon */
    for (i=0; i<pgon; i++)
        pointbase[i][0]=pointbase[i][0]/pointbase[i][2]+20.0;
        pointbase[i][1]=pointbase[i][1]/pointbase[i][2];
        pointbase[i][2]=1.0;
}

void roplp() {                                          /* pop fpolygon */
        if (plev<0) return;
        roplo();
        plev--;
}

void ropuc() {int i;                                   /* draw polygon */
    PSsetgray(NX_BLACK);
    PSmoveto(pointbase[pgon-1][0]/pointbase[pgon-1][2]+exoff,
            pointbase[pgon-1][1]/pointbase[pgon-1][2]+wyoff);
    for (i=0; i<pgon; i++) {
            PSlineto(pointbase[i][0]/pointbase[i][2]+exoff,
            pointbase[i][1]/pointbase[i][2]+wyoff);}
}

void ropuk() {int i, j;                              /* turn 60 deg right */
    pgon = 3;
    edge = 0;
    for (i=0; i<pgon; i++) for (j=0; j<3; j++) pointbase[i][j] = isosrt[i][j];
}

void ropul() {int i, j;                              /* turn 60 deg right */
    pgon = 3;
    edge = 0;
    for (i=0; i<pgon; i++) for (j=0; j<3; j++) pointbase[i][j] = halftr[i][j];
}

void ropun() {edge++; refen(edge % pgon);}          /* reflect in next edge */

void ropuo() {int i, j;                               /* read top  polygon */
    for (i=0; i<pgon; i++) for (j=0; j<3; j++)
        pointbase[i][j]=polystack[plev][i][j];
}

void ropup() {int i, j;                                    /* push polygon */
        plev++;
        if (plev>=HMAX) return;
    for (i=0; i<pgon; i++) for (j=0; j<3; j++)
        polystack[plev][i][j]=pointbase[i][j];
}

void refen(nn) int nn; {                              /* reflect in edge n */
    int i, m, n;
```

```
    double aa, bb, cc, x, y, ex1, ex2, wy1, wy2, rr, m11, m12, m21, m22;
        n = nn % pgon;
        m = (n + 1) % pgon;
        ex1 = pointbase[n][0]/pointbase[n][2];
        ex2 = pointbase[m][0]/pointbase[m][2];
        wy1 = pointbase[n][1]/pointbase[n][2];
        wy2 = pointbase[m][1]/pointbase[m][2];
        aa = wy1 - wy2;
        bb = ex2 - ex1;
        cc = wy2 * ex1 - wy1 * ex2;
        rr = aa * aa + bb * bb;
        m11 =  bb * bb - aa * aa;
        m12 = -2.0 * aa * bb;
        m21 =  m12;
        m22 = -m11;
        for (i=0; i<pgon; i++) {
            x = (m11 * pointbase[i][0] + m12 * pointbase[i][1])
                /pointbase[i][2];
            y = (m21 * pointbase[i][0] + m22 * pointbase[i][1])
                /pointbase[i][2];
            pointbase[i][0] = (x - 2.0 * aa * cc)/rr;
            pointbase[i][1] = (y - 2.0 * bb * cc)/rr;
            pointbase[i][2] = 1.0;
            }
}

void ropur() {int n;                                    /* reflect in edge n */
        n = *r_pc++ - '0';
        if (n < 0 || n > pgon-1) return; else {edge=n; refen(n);};
}

void roput() {int i, j;                                          /* rhombus */
    pgon = 4;
    edge = 0;
    for (i=0; i<pgon; i++) for (j=0; j<3; j++) pointbase[i][j]=rhombus[i][j];
}

void ropuu() {int i, j;                                        /* snubsquare */
    pgon = 5;
    edge = 0;
    for (i=0; i<pgon; i++) for (j=0; j<3; j++) pointbase[i][j]=snubsq[i][j];
}

void ropux() {int c, i; double th, dth;                   /* regular polygon */
        c = *r_pc++ - '0';
        if (c < 1 || c > 12) return;
        pgon = c;
        edge = pgon-1;
        dth = (2 * PI)/((double)(pgon));
        for (i=0, th=-0.5*dth; i<pgon; i++, th+=dth) {
                pointbase[i][0] = cos(th);
```

```
                pointbase[i][1] = sin(th);
                pointbase[i][2] = 1.0/scale;
                }
}
```

4 RECView.h Header Listing

Although the headers for a methods file can contain all the things that ordinary headers do, they are mostly given over to listing objects and method prototypes for their associated file. If the Interface Builder has been used without the benefit of a header file, there is an option called "unparsing" which will create one. Conversely, if the text of the header file has been changed, the `.nib` file can be updated by using "unparse."

It is worth noting that there is no convenient way to supply several arguments to a method which is going to be connected graphically in the Interface Builder. But methods are not supposed to look at each other's data anyway; they should only transmit the data or pointers to the data in response to a request. To know who made a request, and thus to whom to send the reply, a method can volunteer its own name which the called program can recognize as an argument called `:sender`, at which time it knows how to ask for further services or data. That is why `:sender` is the only argument of most of the methods in the prototype list.

```
/*
 * RECView.h -- Interface file for the RECView class
 *
 * You may freely copy, distribute, and reuse the code in this example.
 * NeXT disclaims any warranty of any kind, expressed or implied, as to
 * its fitness for any particular use, or even that it has a use at all.
 *
 */

#import <appkit/appkit.h>

@interface RECView:View {

        id                browser;

        id                exsli;
        id                wysli;
        id                bigView;
        id                bigWindow;
        id                infoPanel;
        id                parPanel;

        id                comment;

        id                lilView;
        id                lilWindow;
        id                execWin;
        id                execLine;
        id                value;
```

```
        id              keyWindow;
        id              menuCopy;
        id              menuPrint;

      char              *filename;
      char              **filenames;

        id              windows;
        id              externalRECBox;
        id              internalRECBox;
        id              theBigBox;
        id              theLilBox;
        id              xtrnWin;

}

- ascii:sender;
- recMatrix:sender;

- rexsli:sender;
- rwysli:sender;

- run:sender;
- recompile:sender;
- gho:sender;
- setOption:sender;
- windowWillClose:sender;
- loadRECPanel:sender;
- loadInternal:sender;
- loadExternal:sender;
- loadLowerPaneltwo:thisRuleBox;
- nhul:sender;
- browserDoubleClick:sender;

- smartPrintBigWin:sender;
- openBigWin:sender;

- open:sender;
- setFilename:(const char *)aFilename;
- save:sender;
- saveAs:sender;

- copyREC:sender;
- copySomeView:theView;

- initFrame:(const NXRect *)frameRect;
- drawSelf:(const NXRect *)rects :(int)rectCount;
- sizeTo:(NXCoord)width :(NXCoord)height;
- mouseDown:(NXEvent *)theEvent;
```

```
@end

/* end RECView.h */
```

5 RECView.m Program Listing

REC-F is short enough that a single method file, RECView.m and its associated header, RECView.h, are sufficient to contain the program. Nevertheless the use of such artifacts as the browser and text windows, in addition to the main Window and View controls, and the C implementation program for the REC operators, divide the program into several natural parts, each of which can be described separately.

5.1 Declarations

The declarations shown below contain several extraneous elements, and are only included for consistency with the rest of the program, in case the definition of terms used elsewhere is needed.

```
/*
 * RECView.m -- Implementation file for the RECView class
 *
 * You may freely copy, distribute, and reuse the code in this example.
 * NeXT disclaims any warranty of any kind, expressed or implied, as to
 * its fitness for any particular use, or that it even has a use.
 *
 */

#import "rec.h"
#import "recdtbl.h"
#import "RECView.h"
#import <NXCType.h>
#import <math.h>
#import <stdlib.h>
#import <string.h>

# define   KK 2
# define CLEN 300 /* buffer length for rec source        */
# define PLEN 900 /* buffer length for rec object        */
# define HMAX  25 /* maximum depth pushdown stack        */
# define PMAX  12 /* maximum polygon accepted by program */
# define  NMU   5 /* number of items in first column     */
# define   NY   8 /* number of demonstration programs    */
# define  NIP  12 /* number of initial polygons          */
# define  NSK   4 /* number of prototype skeletons       */
# define PI 3.14159

//int  rec(char *z);
void refen(int n);
```

```
char   cstr[CLEN]; /* console rec program     */
Inst   rprg[PLEN]; /* rec program array       */
int    phil; /* length of cstr    */
int    plev; /* pushdown depth    */
char   idef[1][30] = { " (()) " };  //(z(!100!0r:;)1r2r(0r:;);) a" };
char   recrule[NY][CLEN] = {
" ([pentagonal] X5 P CN CN CN CN C p;)",
" ([bipentag] X5 P CN CN CN Cn Cn Cn Cn C p;)",
" (TC NC NC NC nC nC nC nC;)"
};
char   skeletons[NSK][CLEN] = {
" ([show polygon] 0 C;)",
" ([sector] 0 (! n ! C N:;);)",
" ([binary] 0 (! n-1 ! C N:;)(! n-1 ! C n :;);)",
" ([second] 0 (! n ! C N:;);)"
};

int    isErrm=FALSE;
int    drawBack=YES;
int    exSource=YES;
int    options[4]={FALSE,FALSE,FALSE,FALSE};
int    pgon=3, edge=0;
char   errm[50];
double scale=25.0;

char dfclr[KK][12] = {"1.0,0.0,0.0","0.0,1.0,0.0"};
struct inpoly {int order; double points[8][3]; char *title;};
struct inpoly inpollies[NIP] = {
3,{ { 0.0,  0.0, 0.25},
{ 5.0,  0.0, 0.25},
{ 0.0,  8.6, 0.25}},
"3 - 30-60-90 triangle",
3,{ { 0.0,  0.0, 0.25},
{10.0,  0.0, 0.25},
{ 0.0, 10.0, 0.25}},
"3 - 45-45-90 triangle",
3,{ { 0.0,  0.0, 0.25},
{ 5.0,  0.0, 0.25},
{ 0.0,  8.6, 0.25}},
"3 - skew triangle",
4,{ { 0.0,  0.0, 0.25},
{10.0,  0.0, 0.25},
{15.0,  8.7, 0.25},
{ 5.0,  8.7, 0.25}},
"4 - rhombus",
4,{ { 0.0,  0.0, 0.25},
{30.0,  0.0, 0.25},
{20.0, 10.0, 0.25},
{10.0, 10.0, 0.25}},
"4 - 45 degree trapezoid",
4,{ { 0.0,  0.0, 0.25},
```

```
{20.0,  0.0, 0.25},
{15.0,  8.6, 0.25},
{ 5.0,  8.6, 0.25}},
"4 - 60 degree trapezoid",
5,{ { 0.0,  0.0, 0.25},
{10.0,  0.0, 0.25},
{10.0,  5.0, 0.25},
{ 5.0, 10.0, 0.25},
{ 0.0, 10.0, 0.25}},
"5 - snub square",
5,{ { 0.0,  0.0, 0.5},
{10.0,  0.0, 0.5},
{10.0, 10.0, 0.5},
{ 5.0, 15.0, 0.5},
{ 0.0, 10.0, 0.5}},
"5 - 45 gable house",
5,{ { 0.0,  0.0, 0.5},
{10.0,  0.0, 0.5},
{10.0, 10.0, 0.5},
{ 5.0, 18.6, 0.5},
{ 0.0, 10.0, 0.5}},
"5 - 60 gable house",
6,{ { 0.0,  0.0, 0.5},
{30.0,  0.0, 0.5},
{30.0, 10.0, 0.5},
{20.0, 20.0, 0.5},
{10.0, 20.0, 0.5},
{ 0.0, 10.0, 0.5}},
"6 - barn",
};

double high, wide;
double eks, wye, angle, alpha;
double exoff=0.0, wyoff=0.0;
extern double r_dblpar;
double polystack[HMAX][PMAX][3];
double pointbase[PMAX][3],edgebase[PMAX][3];
double pointtraj[PMAX][3],edgetraj[PMAX][3];
double pointsnow[PMAX][3],edgesnow[PMAX][3];
double pointinit[3][3]={{0.0,0.0,1.0}, {0.0,10.0,1.0}, {10.0,0.0,1.0}};
double halftr[3][3]={ { 0.0,  0.0, 0.25},
{ 5.0,  0.0, 0.25},
{ 0.0,  8.6, 0.25}};
double isosrt[3][3]={ { 0.0,  0.0, 0.25},
{10.0,  0.0, 0.25},
{ 0.0, 10.0, 0.25}};
double rhombos[4][3]={ { 0.0,  0.0, 0.25},
{10.0,  0.0, 0.25},
{12.0, 08.0, 0.25},
{ 5.0,  7.0, 0.25}};
double rhombus[4][3]={ { 0.0,  0.0, 0.25},
```

```
{10.0,  0.0, 0.25},
{15.0,  8.7, 0.25},
{ 5.0,  8.7, 0.25}};
double snubsq[5][3]={ { 0.0,  0.0, 0.25},
{10.0,  0.0, 0.25},
{10.0,  5.0, 0.25},
{ 5.0, 10.0, 0.25},
{ 0.0, 10.0, 0.25}};
```

5.2 Initialization and termination

```
/* ---------------  A P P   I N I T I A L I Z A T I O N   ---------------- */

- appWillInit:sender {printf("appWillInit = REC-F\n"); return self;}

- appDidInit:sender {

  printf("appDidInit = REC-F for Flexagons\n");

  [browser getTitleFromPreviousColumn: YES];
  [browser setDoubleAction: @selector(copyAndRunREC:)];
  [browser setPath: "/demonstrations"];

  [self loadExternal: self];

  return self;
}

- appWillTerminate:sender {printf("goodbye\n"); return self;}
```

5.3 Opening and closing files

There is a fairly standard collection of methods which can be used to get files for a program, or to save them afterward. The main complication consists of nesting them within the operating system, but naming files and searching for them also requires some attention.

```
/* - - - - - - - - -  F I L E   A N D   D I R E C T O R Y  - - - - - - - - */

- setFilename:(const char *)aFilename {
if (filename) free(filename);
filename = malloc(strlen(aFilename)+1);
strcpy(filename, aFilename);
[window setTitleAsFilename:aFilename];
return self;
}

- open:sender {
  char *types[2] = {"rec",0};
  int  i, fd;
```

```
  NXStream *theStream;

  [[OpenPanel new] allowMultipleFiles: NO];
  [[OpenPanel new] chooseDirectories: NO];

  if ([ [OpenPanel new] runModalForTypes:types]) {
    [self setFilename: [[OpenPanel new] filename]];
    fd = open(filename, O_RDONLY, 06666);
    theStream = NXOpenFile(fd, NX_READONLY);
//      NXScanf(theStream,"%s",cstr);
      i=0;
      while (NXScanf(theStream,"%c",&cstr[i])!=EOF) i++;
      phil=strlen(cstr);
    NXClose(theStream);
    close(fd);
    [window setTitleAsFilename: filename];
    [window setDocEdited:NO];
  [execWin setSel: 0: [execWin textLength]];
  [execWin replaceSel: cstr];
  [execWin scrollSelToVisible];
  [execWin display];
    }

return self;
}

- saveAs:sender {
  id panel;
  const char *dir;
  char *file;

/* prompt user for the file name and save to that file */

  if (filename==0) {
    /* no filename; set up defaults */
    dir  = NXHomeDirectory();
    file = (char *)[window title];
    } else {
    file=rindex(filename,'/');
    if (file) {
      dir   = filename;
      *file = 0;
      file++;
      } else {
      dir  = filename;
      file = (char *)[window title];
      }
    }
  panel = [SavePanel new];
  [panel setRequiredFileType: ""];
  if ([panel runModalForDirectory: dir file: file]) {
```

```
    [self setFilename: [panel filename]];
    return [self save: sender];
    }
    return nil;  /* didn't save */
  }

- save:sender {
  int      fd;
  NXStream *theStream;

  if (filename==0) return [self saveAs: sender];
  [window setTitle: "Saving ..."];

  fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 06666);
  if (fd < 0) {
    NXRunAlertPanel(0, "Cannot save file: %s",0,0,0,strerror(errno));
    return self;
    }
  theStream = NXOpenFile(fd, NX_WRITEONLY);
  [execWin writeText: theStream];
  NXClose(theStream);
  close(fd);
  [window setTitleAsFilename: filename];
  [window setDocEdited:NO];
  return self;
}

- close:sender {const char *fname;
int         q;
  if ([window isDocEdited])
//    fname=filename ? filename : [sender title];
//    if (rindex(fname,'/')) fname = rindex(fname,'/') + 1;
    q = NXRunAlertPanel(
      "Save",
      "Save changes to %s?",
      "Save",
      "Don't Save",
      "Cancel",
      filename);
    if (q==1) {if (![self save:nil]) return nil;}
    if(q==-1) return nil;
//  [sender setDelegate: nil];
//  [proc free];
//  [self free];
  return self;
}

- windowWillClose:sender {
[self close: self];
printf("windowWillClose\n");
return self;
```

```
}
```

5.4 Copying and printing Views

There is a mechanism for printing selected views, which is more selective than the printing options which can be connected by mouse dragging in the Interface Builder. Likewise Views can be copied to a pasteboard, where they are available for use by other programs, such as Draw. The problem is to know which Window, or which View to draw, although in principle a separate button for every one of them could be placed on the program's menu.

By making RECView a delegate of a Window thich could be printed, it has an opportunity to respond to the announcement that the Window has become the key Window — in other words, the focus of attention of keyboard strokes and mouse movements and has been displayed in front of everything else. The response takes the form of connecting the one single copy button and the single print button to the View which is to be copied or printed.

To avoid the uncertainty of knowing what to print when some other window has taken priority, the "key resigned" can be used to deactivate the menu buttons. Of course, the substituting window can delegate RECView the responsibility of printing *it*, but otherwise the buttons in the menu remain disconnected.

```
/* - - - - - - - -  c o p y  a n d  p r i n t  v i e w s  - - - - - - */

- printREC:sender {[keyWindow smartPrintPSCode: sender]; return self;}

- windowDidBecomeKey:sender {
        keyWindow=sender;
        [menuCopy setTarget: self];
        [menuCopy setAction: @selector(copyREC:)];
        [menuCopy setEnabled: YES];
        [menuPrint setTarget: self];
        [menuPrint setAction: @selector(printREC:)];
        [menuPrint setEnabled: YES];
return self;
}

- windowDidResignKey:sender {
        [menuCopy setEnabled: NO];
        [menuPrint setEnabled: NO];
return self;
}

- copyREC:sender {
    drawBack=NO;

    if (keyWindow==bigWindow)
        [self copySomeView: bigView];
    if (keyWindow==infoPanel)
        [self copySomeView: [infoPanel contentView]];
    if (keyWindow==execWin)
        [self copySomeView: [execWin contentView]];
```

```
    if (keyWindow==parPanel)
        [self copySomeView: [parPanel contentView]];

    drawBack=YES;
return self;
}

- copySomeView:theView {
NXRect zbounds;
id     pBoard;
//  drawView=viewnum;
//  *draw=YES;
  pBoard = [Pasteboard new];
  [pBoard declareTypes: &NXPostScriptPboardType num: 1 owner: self];
  [theView getBounds: &zbounds];
  [theView writePSCodeInside: &zbounds to: pBoard];
//  *draw=NO;
return self;
}
```

5.5 Browser delegate methods

The program for the NeXTSTEP browser goes to considerable effort to create sliding panels, embody them in an attractive framework, fill them with information, and respond to the activities of the mouse. It still needs to be provided with information to display, and to be told how to respond. The program using the browser has to take charge, which it does by having been designated as a delegate. There are some methods which the browser supervisor expects to find, such as fillMatrix::: described below.

```
/*---------------- B R O W S E R  D E L E G A T E ------------------*/

- (int)browser:sender fillMatrix: matrix inColumn: (int)col {int  i, n;
    n=0;
    printf("browser\n");
    switch(col) {
      case 0:
        for(i=0; i<NMU; i++) {
          [matrix addRow];
          [[matrix cellAt: i: 0]  setLeaf : NO ];
          [[matrix cellAt: i: 0]  setLoaded : YES ];
          [[matrix cellAt: i: 0]  setEnabled : YES ];
          [[matrix cellAt: i: 0]  setEnabled : YES ];
          }
        [[matrix cellAt: 0: 0]  setStringValue: "operators"];
        [[matrix cellAt: 1: 0]  setStringValue: "predefined"];
        [[matrix cellAt: 2: 0]  setStringValue: "skeletons"];
        [[matrix cellAt: 3: 0]  setStringValue: "initial polygons"];
        [[matrix cellAt: 4: 0]  setStringValue: "demonstrations"];
        n=NMU;
        break;
```

```
case 1:
  if(strcmp([[sender selectedCell] stringValue],"demonstrations") == 0){
    for(i=0; i<NY; i++) {
      [matrix addRow];
      [[matrix cellAt: i: 0]  setStringValue: recrule[i]];
      [[matrix cellAt: i: 0]  setLeaf : YES ];
      [[matrix cellAt: i: 0]  setLoaded : YES ];
      [[matrix cellAt: i: 0]  setEnabled : YES ];
      }
    n=NY;}

  if(strcmp([[sender selectedCell] stringValue],"predefined") == 0){
    for(i=0; i<1; i++) {
      [matrix addRow];
      [[matrix cellAt: i: 0]  setStringValue: idef[i]];
      [[matrix cellAt: i: 0]  setLeaf : YES ];
      [[matrix cellAt: i: 0]  setLoaded : YES ];
      [[matrix cellAt: i: 0]  setEnabled : YES ];
      }
    n=1;}

  if(strcmp([[sender selectedCell] stringValue],"operators") == 0){
    for(i=0; i<95; i++) {
      [matrix addRow];
      [[matrix cellAt: i: 0]  setStringValue: dtbl[i].r_cmnt];
      [[matrix cellAt: i: 0]  setLeaf : YES ];
      [[matrix cellAt: i: 0]  setLoaded : YES ];
      [[matrix cellAt: i: 0]  setEnabled : YES ];
      }
    n=95;}
  if(strcmp([[sender selectedCell] stringValue],"initial polygons") == 0){
     for(i=0; i<NIP; i++) {
      [matrix addRow];
      [[matrix cellAt: i: 0]  setStringValue: inpollies[i].title];
      [[matrix cellAt: i: 0]  setLeaf : YES ];
      [[matrix cellAt: i: 0]  setLoaded : YES ];
      [[matrix cellAt: i: 0]  setEnabled : YES ];
     }
    n=NIP;}
  if(strcmp([[sender selectedCell] stringValue],"skeletons") == 0){
     for(i=0; i<NIP; i++) {
      [matrix addRow];
      [[matrix cellAt: i: 0]  setStringValue: skeletons[i]];
      [[matrix cellAt: i: 0]  setLeaf : YES ];
      [[matrix cellAt: i: 0]  setLoaded : YES ];
      [[matrix cellAt: i: 0]  setEnabled : YES ];
     }
    n=NSK;}

      break;
  default: n=0; break; }
```

```
      return n;
    }

/* to distract the single click in a browser double click */
- nhul: sender {return self;}

- browserDoubleClick:sender {int i, j, k;
   if (strcmp([browser titleOfColumn: [browser selectedColumn]],
        "initial polygons") == 0) {
        k=[[sender matrixInColumn: [sender selectedColumn]] selectedRow];
        pgon=inpollies[k].order;
        edge = 0;
        for (i=0; i<pgon; i++) for (j=0; j<3; j++)
                pointbase[i][j] = inpollies[k].points[i][j];
           [execLine setStringValue: "(CP;)"];
          strcpy(cstr,"(CP;)");}
   else if (strcmp([browser titleOfColumn: [browser selectedColumn]],
        "demonstrations") == 0) {
           [value setStringValue: " "];
           [execLine setStringValue: [sender stringValue]];
          strcpy(cstr,[sender stringValue]);
        }
   else if (strcmp([browser titleOfColumn: [browser selectedColumn]],
        "skeletons") == 0) {
           [value setStringValue: " "];
           [execLine setStringValue: [sender stringValue]];
        }
   [lilView display];
return self;
  }
```

5.6 View methods

There are several methods which the programmer must supply to create or modify the content of a View. Which ones are needed and their complexity depends on the intricacy of the planned view. The most essential methods are:

- - initFrame: sets up the overall characteristics of the View, such as its background color, the location and orientation of its coordinate system, and similar details.
- - drawSelf:: draws the View each time it is invoked. It is therefore responsible for the entire content of the View, which may depend on parameters which vary between successive invocations.
- - mouseDown: has an argument describing the interruption, which can be used as the basis for decisions about how to respond to the interruption.
- - sizeTo:: would be used to respond to a change in the size of the View; for example to ensure centering the origin of coordinates.

Beyond these essentials there are other possibilities, such as assigning a delegate.

```
/* = = = = = = = = = = = = = =   R E C V I E W   = = = = = = = = = = = = = */

- initFrame:(const NXRect *)frameRect
/*
 * Initializes the new RECView object.  First, an initFrame: message is sent
 * to super to initialize RECView as a View.  Next, the RECView sets its own
 * state -- that it is opaque and that the origin of its coordinate system
 * lies in the center of its area.
 */
{
        [super initFrame:frameRect];
        [self setOpaque:YES];
        [self translate:floor(frame.size.width/2) :floor(frame.size.height/2)];
        return self;
}

- drawSelf:(const NXRect *)rects :(int)rectCount
/*
 * Draws the RECView's background and axes.  If there are any points,
 * these are drawn too.
 */
{
        if (rects == NULL) return self;

//        PSsetgray(NX_WHITE);
        PSsetgray(NX_LTGRAY);
        if (drawBack) NXRectFill(&rects[0]);
        PSsetgray(NX_DKGRAY);
//        PSsetgray(NX_BLACK);

                    isErrm=FALSE;
                    [value setStringValue: " "];
                    if (rec(cstr)) [value setStringValue: "T"];
                        else [value setStringValue: "F"];
                    if (isErrm) {[comment setStringValue: errm]; isErrm=FALSE;}
                    PSstroke();

        return self;
}

- sizeTo:(NXCoord)width :(NXCoord)height
/*
 * Ensures that whenever the RECView is resized, the origin of its
 * coordinate system is repositioned to the center of its area.
 */
{
        [super sizeTo:width :height];
        [self setDrawOrigin:-floor(width/2) : -floor(height/2)];
```

```
        return self;
}

- mouseDown:(NXEvent *) theEvent
/*
 * Responds to a message the system sends whenever the user presses the
 * mouse button when the cursor is over the RECView. The RECView changes
 * the cursor to a cross-hairs image and then starts asking for mouse-dragged
 * or mouse-up events. As it receives mouse-dragged events, the RECView
 * updates the readOut text Cell with the cursor's coordinates. If the user
 * releases the mouse button while the cursor is over the RECView,the RECView
 * registers the point and then sends a message to its delegate notifying it
 * of the new point.
 */
{
        int        looping = YES, oldMask;
        NXPoint        aPoint;
        NXRect        plotRect;

        [crossCursor set];
           [self getBounds:&plotRect];

        oldMask = [window addToEventMask:NX_MOUSEDRAGGEDMASK];
        [self lockFocus];
        do {
                aPoint = theEvent->location;
                [window flushWindow];
                if (theEvent->type == NX_MOUSEUP) {
        /* on mouse-up, register point, inform delegate, and clean up state */
                        [window flushWindow];
                        looping = NO;
                }
           } while (looping &&
    (theEvent=[NXApp getNextEvent:NX_MOUSEUPMASK|NX_MOUSEDRAGGEDMASK]));
        [self unlockFocus];
        [window setEventMask:oldMask];
        [NXArrow set];
        return self;
}
```

6 A Case Study: the cis-Ternary Pentagonal Flexagon

Ternary flexagons are those whose map has three cycles. For symmetry reasons when using regular pentagons, there are only two ternary flexagons, which can be called cis-ternary and trans-ternary from the viewpoint of chemical nomenclature. In the cis- form, the two secondary rings are adjacent; in the trans- form, they are separated.

To get the plan of a flexagon, its map is first drawn, and the Tukey Triangle network set up; it may be more appropriate to call them Tukey pentagons in this case. The result is shown in Figure 8, below.

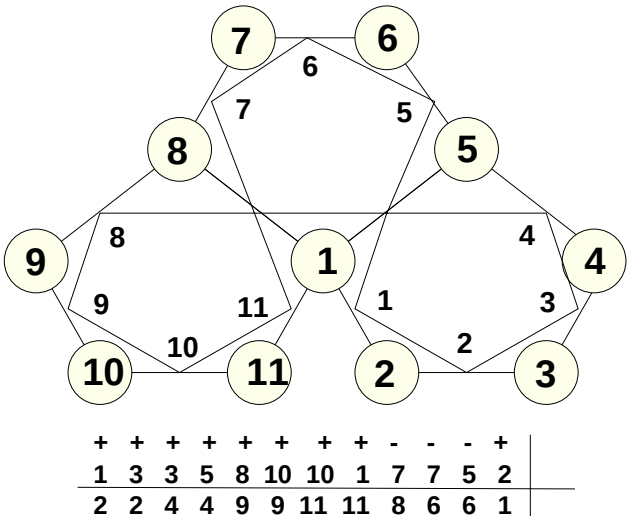

Figure 8: Map, with Tukey pentagons, of the cis-ternary pentagonal flexagon.

The essential element of Figure 8 is the sign sequence which describes the succession of edge reflections used to make the plan. They must be translated into REC in terms of `N`'s and `n`'s. In order to have the longest runs of like signs, the map was numbered in just the way shown in the figure, producing the sign sequence {+ + + + + + + + - - - }.

Using `X5` to create the original pentagon,

```
(X5 CN CN CN CN CN CN CN CN Cn Cn Cn ;),
```

would give a satisfactory plan, each `N` corresponding to a `+` and each `n` to a `-`. The resulting plan has a kink, with some of the pentagons overlapping others. Such conflicts can be resolved one way or another. Since preserving the hinging edges is what really matters, overlapping polygons can be cut apart and the resulting fragments used even when they are in tatters.

If that is impractical, or it is desired to preserve the full polygon, the plan can be interrupted and the continuation offset, by using the REC operator `o`, sometimes several times to get a good separation.

Using a counter to avoid repeating `CN` over and over (there is a crossover point before which repetition is shorter than counting), the program which is finally used is

```
([cis-ternary pentagonal] X5 N P (!8!CN :;)PCoo p (!4!  Cn:;) p;).
```

Actually, an extra count is given for each stretch, together with the push and pop operators. The reason is to have an extra polygon which can be glued to its equivalent in the next stretch to join the pieces together. Having repeated the bridging polygon, the program returns to where it left off before continuing. The exterior push-pop pair is actually unnecessary, but is useful in a library where one strip needs to be inserted into another without losing continuity.

Once a satisfactory plan has been created, it can be transferred to the Draw program for lettering and coloring.

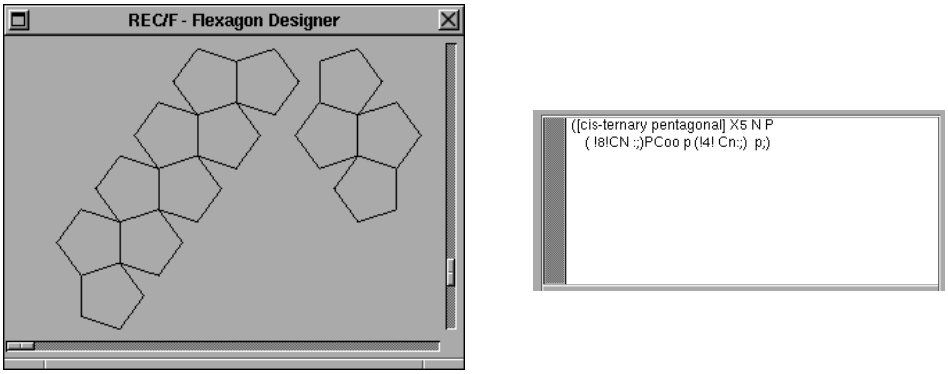

Figure 9: Panels used in drawing the cis-ternary pentagonal flexagon. The right panel is just the text field from the main window in "external" mode.

In this case, the plan is of odd length, so it is convenient to show the second upside down plan on the same page; in any event the pentagons are going to be rather small, and will be no smaller on account of this accomodation. Top and Bottom pages will have to be prepared anyway; one of the inconveniences of Draw is that it does not reflect images, maybe to avoid trying to reflect lettering.

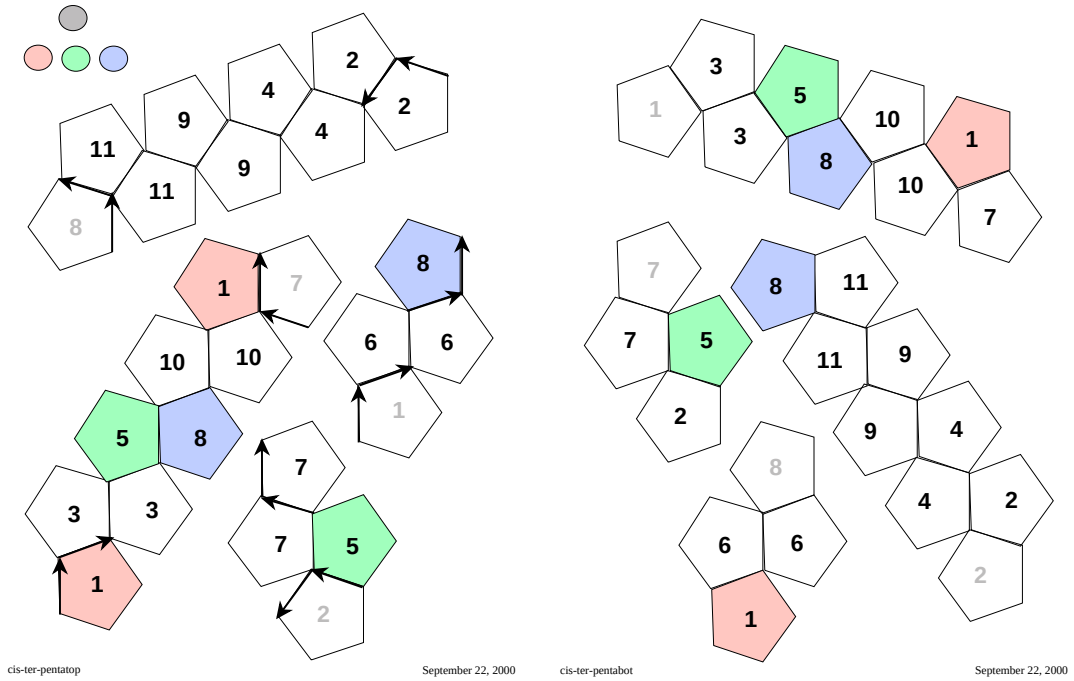

Figure 10: The cutout page from which the final flexagon may be cut, folded and pasted. Left: the top side. Right: the bottom side. Not all the pentagons have been colored; only those which are faces appearing in two different cycles. If they are visible while flexing the flexagon, the mountain-valley transition will switch cycles.

7 Acknowledgements and Disclaimer

The NeXT computer on and for which this program has been developed is an acquisition from a CONACYT grant, which must have been one of its more productive investments, given that the computer has been in daily use for many hours since its acquisition.

The program REC/F described in the document can hardly be expected to be error free. Moreover, if it is successful, it will be subject to user demands for improvements, corrections, and assorted changes. Not all of them will necessarily be reflected back in this manual, which is intended more to explain the program than to give a verbatim documentation.

References

[1] Martin Gardner, "Flexagons," *Scientific American*, December, 1956, pp. 162-166.

[2] Martin Gardner, Chapter 1: Hexaflexagons (pp. 1 - 14), *The Scientific American Book of Mathematical Puzzles & Diversions*, Simon and Schuster, New York, 1959 (ISBN 0-671-63652-9 Pbk.).

[3] Martin Gardner, "About tetraflexagons and tetraflexigation," *Scientific American*, May, 1958, pp. 122-6.

[4] Martin Gardner, Chapter 2: Tetraflexagons (pp. 24 - 31), *The* 2^{nd} *Scientific American Book of Mathematical Puzzles & Diversions*, Simon and Schuster, New York, 1961; republished The University of Chicago Press, Chicago, 1987 (ISBN 0-226-28253-8(pbk).

[5] Martin Gardner, Chapter 7: The Combinatorics of Paper Folding (pp. 60 - 73), *Wheels, Life, and Other Mathematical Amusements*, W. H. Freeman and Company, New York, 1983 (ISBN 0-7167-1589-9 Pbk.).

[6] A. S. Conrad, "The Flexagon," RIAS Miscellaneous Report, Baltimore, Maryland, 1960. (later renamed "The Theory of the Flexagon," RIAS Technical Report 60-24, 1960).

[7] Anthony S. Conrad and Daniel K. Hartline, *Flexagons*, RIAS Technical Report 62-11 Baltimore, Maryland, 1962.

[8] C. O. Oakley and R. J. Wisner, "Flexagons," *American Mathematical Monthly*, **64** 143-154 (1957).

[9] Harold V. McIntosh, "My Flexagon Experiences," University of Puebla, 2000.

[10] Harold V. McIntosh and Gerardo Cisneros, "The programming languages REC and Convert," *SIGPLAN Notices*, **25** 81-94 (July 1990).

[11] Gerardo Cisneros, "Configurable REC," *ACM SIGPLAN Notices* **29** May 1995 (11 pages)

[12] Simson L. Garfinkel and Michael K. Mahoney, *NeXTSTEP*^{TM} *Programming*, Springer Verlag, New York, 1953. ISBN 0-387-97884-4

[13] Adobe Systems, Incorporated, *PostScript Language Reference Manual*, Second Edition, Addison-Wesley Publishing Company, Inc., Reading, Massachussetts, 1990. ISBN 0-201-18127-4.