Behavior Research Methods, Instruments, & Computers
1990, 22 (5), 453-465

— COMPUTER TECHNOLOGY —

A multipurpose software package for editing two-dimensional animated images

JEAN LORENCEAU
Laboratoire de Psychologie Expérimentale
Université R. Descartes, EHESS, CNRS, Paris, France

and

REMI HUMBERT
Unité Informatique Sciences Humaines
Université R. Descartes, EHESS, CNRS, Paris, France

In this paper, we describe a software package, LEDA, for editing two-dimensional images and films. It is written in Turbo-C and was first conceived to work with a high-resolution graphics card (Adage PG90/10, 2,048×1,023×8 bits) on an IBM PC/AT or compatible computer. The program is intended for managing images and films used in the fields of visual psychophysics, electrophysiology, and so forth.

Software packages intended to drive currently available graphics displays are not always well adapted to the needs of researchers who use visual stimuli to perform a variety of experiments. Consequently, a great deal of time is spent developing programs able to build the images or films needed for each experiment. It is common that, for each series of experiments, new programs must be written, even if parts of older programs can be reused. Sometimes, a new application requires a new data structure, since old ones are too specific, and each procedure must be modified accordingly. This increasingly large number of potential modifications increases the probability of bugs or program errors. Additional problems with this approach are the rapid increase in the number of programs together with their specific graphic images, the lack of comments in program code, the absence of compatibility, and so forth. This scenario represents one extreme; a number of researchers and engineers, on the other hand, have developed subtle methods and modular applications with few of the aforementioned disadvantages.

We have developed a multipurpose program that allows the rapid construction of many different images and films, while taking into account the constraints of an experimental approach. Our goal was to create software that would allow editing of most of the stimuli needed to perform a variety of experiments in our laboratory. These include studies on subjective contours, apparent motion with isoluminant stimuli, motion viewed through apertures, shape recognition, and texture discrimination. The software also had to be versatile enough so that the user could change any parameters of interest easily and rapidly.

We wanted the program to draw images quickly, to build a large variety of images, to be easily modifiable, and to use as little memory as possible. Moreover, we wanted a user-friendly program that anyone could use without prior knowledge of programming. In addition, the program had to be adaptable to different graphics displays.

In this paper, we describe the program LEDA, which we designed to meet the criteria outlined above. It is currently used on an AT-compatible microcomputer with a math coprocessor and a high-resolution graphics display (Adage PG90/10, 2,048×1,024×8 bits). The images are displayed on a monitor (Sony GDM 1950) refreshed at 60 Hz. To ensure maximum machine-independence, the program is written in Turbo-C and is distributed in different modules, each one intended for specific applications.

First, we will describe the data structure and the different modules that constitute the body of the program. Next, we will describe the different functions that are available and how they can be used to build different kinds of images. (More details on the editing process itself are provided in Appendix A.) Finally, we will consider some potential extensions as well as limitations of the software.

GENERAL STRUCTURE: MODULES

To ensure easy maintenance, the program is distributed in 13 modules, so that one can find, modify, or add functions that pertain to specific applications.

Four modules deal with low-level applications, such as the management of windows, control of keyboard inputs,

We wish to thank Margaret Shiffrar for her comments and help in improving an earlier draft of the text. This work was supported by Grant DRET 88/114. Correspondence should be addressed to Jean Lorenceau, Laboratoire de Psychologie Expérimentale, Université R. Descartes, EHESS, associé au CNRS, 28 rue Serpente, Paris 75006, France.

menus, and cursor control. They are grouped into a library. Four other modules manage operations on the data. One deals with elements (e.g., allocation of memory), one deals with images, and two manage parts of images (block) and a complementary buffer (clipboard). Two modules are used for mathematical operations on the data. The first one contains low-level functions such as calculating the x- and y-coordinates of lines from their length and orientation, or calculating the value of a sinusoid from its amplitude, pulse, and phase. The second one contains procedures that allow for more complex operations on the data. One module contains the functions used to save or load images, change the directory, and so forth. The procedures used to drive the graphics card are in a separate module, so that modifications only at this level are needed to adapt the program to different displays. (Additional information for adapting the software is provided in Appendix B.) Finally, the main program, LEDA, organizes the links between the modules.

STRUCTURES AND DATA: IMPLEMENTATION

To have a fast drawing program, we decided to directly manage *elements*, which correspond to the primitives generally implemented on graphics cards: dots, lines or polylines, circles, ellipses, polygons, and text. In addition, the software manages *sprites* (i.e., parts of the bit map) in order to display elements corresponding to smoothly varying gray levels, such as gabor functions, or shapes that do not correspond directly to the simple combination of primitives. Fractal elements are also available.

Each image or film is a file of such elements. Elements are embodied in a *structure*, which is larger than the information needed to draw an element. A large structure allows various manipulations and also permits modification of the program for future applications.

An element can be defined by a *head*, a *body*, and *limbs* (Figure 1). The head is a list of numbers, the first of which defines the position of the element in the list, so that each element can be easily found. Three additional fields contain the number of the image, the column, and the line to which the element belongs (these values are set to zero if no image is defined). Two pointers indicate the following and preceding element, as in double-chained lists.

The body is the list of the characteristics shared by all the elements: the geometry of a given element, coded with a number (this choice is intended to provide easy and rapid inputs with the numeric keyboard, and it allows for basic calculation and sorting), the color and the mode (the mode is used to select the combination—AND, OR, XOR, etc.—of an element with the background), the angle of the element relative to the horizontal axis (when relevant for the chosen geometry), and the location on the screen. Two additional fields are used to store the radius of a circle and the number of sides of polygons. Finally, a pointer is used to point to the limbs, if necessary.

The limbs are fields (arrays of x, y integers) used to encode the characteristics of a given geometry, such as the coordinates of the vertices of a polygon. The size of the limbs is variable; it depends on the geometry itself (no limbs are necessary for a dot). At any time, any field of the element's structure can be changed. The facilities for these changes are detailed below.

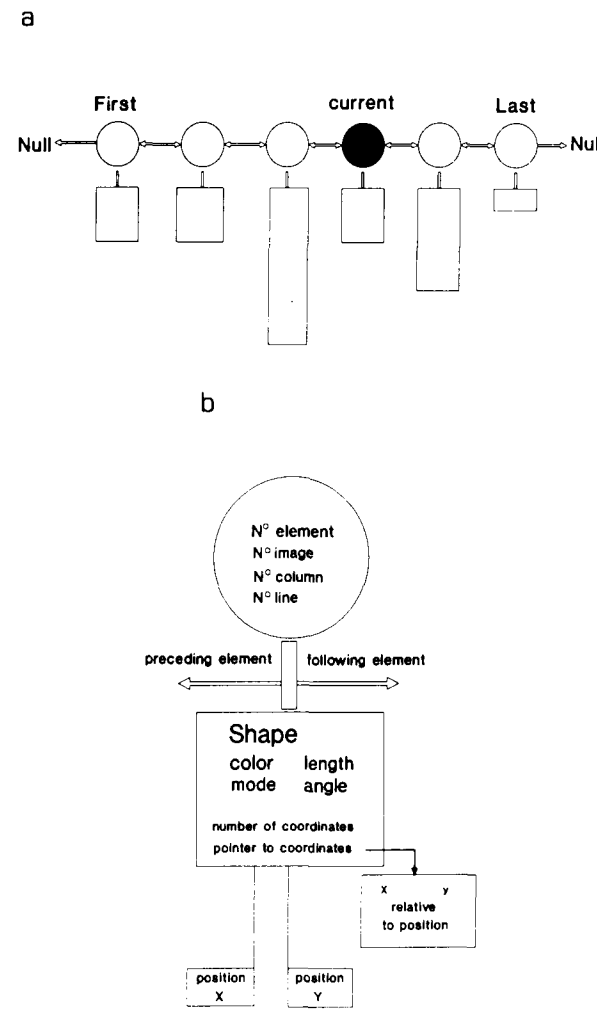

Figure 1. (a) Structure of the double-chained list. (b) Data structure for a single element: addresses and pointers of the one element and fields of the structure available for the user.

When a file is stored, each field of the element's structure is recorded instead of the usual bitmap (bitmap recording is available, however). The size of the file is thus adjusted to the size of the image, so that only the memory that is needed is occupied (e.g., a circle uses 48 bytes). Since all the information is defined at the element level, only the elements' list is stored.

As with elements, images are embodied in a double-chained list. Each image is characterized by its number in the list, as well as the number of lines and columns in it. Four other fields are used. Two of them are pointers to the following and preceding images. The two remaining fields are pointers to the first and last element of the image (Figure 2). At any time, images can be modified and defined again with different values. For instance, one image defined as a matrix of 10 columns and 10 lines can

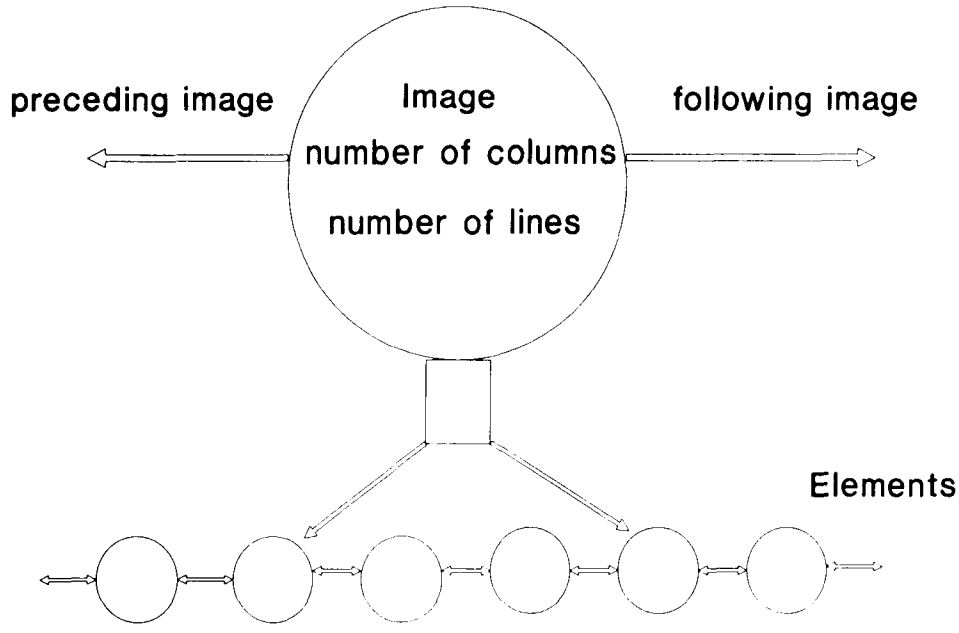

Figure 2. Data structure for the images. Fields for the definition of images, pointers between images, and pointers toward the elements in the list corresponding to the first and last element of one image.

be changed into 10 images defined as matrices of 10 columns and 1 line. The modifications change the segmentation of the list and not the locations of the elements on the screen, since a particular function is used to assign positions on the screen to the elements belonging to the same image. The various allowable manipulations are described below.

Note that the images' list is not recorded on disk, since it is rebuilt from the elements' list when a file is loaded to memory.

In addition to images, two volatile segmentations can be used to manipulate separate parts of the elements' list. *Motifs* are pieces of the elements' list defined by a starting and an ending element, independently of the actual image contents. *Blocks* are parts of an image (or images) defined by the starting and ending numbers of the images, the columns, and the lines. A block allows the selection of a subpart of an image through a film. The elements belonging to a block or to a motif can be copied in a buffer, moved on the screen, or erased. The elements stored in the buffer may be inserted anywhere in the list.

A *film* consists of a number of images that can be displayed sequentially with a given offset delay and a given interstimulus interval. Several other functions, such as panning the frame buffer or dynamic bit-plane segmentation, intended to display a film, are described below. It should be noted that a film may be used to store stimuli corresponding to a block of trials, to a moving stimulus, or to both.

PROGRAM AND FUNCTIONS

The structure of the program is depicted in Figure 3. Activity is controlled through a switching procedure toward basic functions. When the program is entered, a user can choose from a number of functions, including: editing elements (to assign values to the different fields), saving a list of elements, loading a list, clearing the current list from memory, changing directories, or exiting.

In the edit mode, four levels are accessible: edition of elements, edition of images, manipulation of blocks, and display functions. Within each of these levels, specific functions are used to define, modify, or display the contents or the structure of an element's list.

In the edit mode, the status of each of the different variables that define a list is shown: the number and attributes of the current element, the number and characteristics of the current image, the total number of elements and images, the size of the current block, and the size of the available memory (Figure 4). The different choices corresponding to the current level can be selected from a menu. Submenus are displayed in windows according to keyboard inputs.

At the element level, one can edit a list (element by element or all at once), add, insert, copy, or erase one or more elements, and use arrows to move within a file. (In addition, an element can be selected by its number and then edited.) One can define or modify an element (a geometry) or access a specific field of this element (its color, its angle, etc.). When defined (i.e., when the geometry is specified), the current element is always displayed on the graphics monitor, at the center of the screen if x- and y-coordinates have not been defined (they are initialized to zero), or else at the requested position. With the arrows, an element can be displaced pixel by pixel. Plus and minus signs are used to adjust the step size of displacement. An element can be rotated by an adjustable increment around its center. A switch allows the simul-

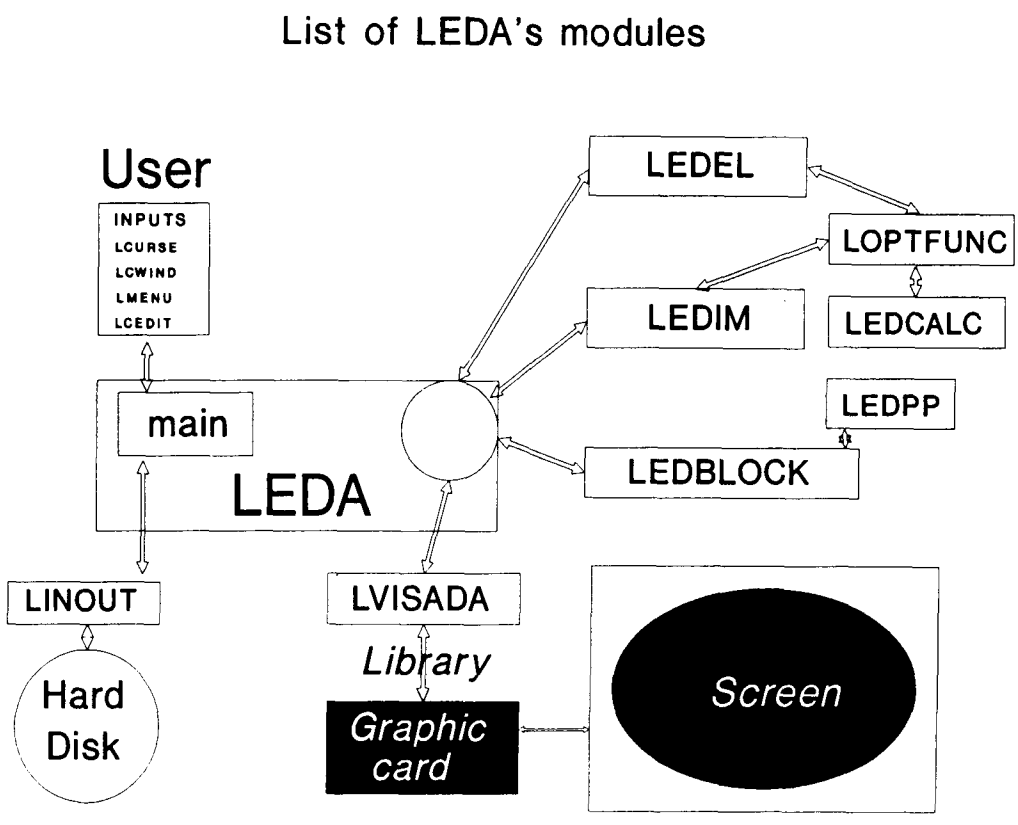

Figure 3. Modular conception of the software package: modules for keyboard inputs and management of windows, management of elements, images, mathematical operations, block, and complementary buffer. Display functions are grouped in a single module, so that only modifications at this level are necessary when adapting the software to a different graphics library.

taneous presentation of several elements as one moves through the list, so that one can control and adjust the relative positions of different elements.

At this element level, one can copy the contents of a motif of a given size (number of elements) from a starting point to an end point. The contents of a motif, defined by its size and the number of the first element, can be elements as a whole or a single field of elements. For instance, four colors assigned to four different elements may be defined according to a color motif that can be applied to other elements of different shapes, orientations, and so forth. In the same manner, three elements defined as a triangle, a disk, and a line may be used as a motif of three geometries copied within boundaries without changing the color of the modified elements.

Mathematical functions (linear, triangle, exponential, square, gaussian, sinusoid, etc.) can be applied to modify a specific field of one part of the list and/or one type of geometry. For instance, one can linearly increase the size of all the lines between given limits of a file containing disks and polygons. This option works in the following manner: for each element within the defined boundaries, a value, k, is regularly incremented and used to calculate a function whose parameters have been input at the keyboard. On request, the result of the calculus, y, can replace, be added to, multiply with, or divide, the value of the element's attribute to be changed. The resulting value is then assigned to the attribute's field.

The image level is designed to edit, move, modify, insert, or add images to the image's list. The arrow keys are used to move back and forth in the images' list; the current image is always displayed on the graphics monitor. A switch permits the viewing of successive images, which are then superimposed. When the switch is on, all but the current image are cleared.

If the number of elements in the list is not large enough to build the desired images, additional elements are automatically allocated and initialized to zero (the "copy a motif" option can fill them in).

The structure of an image is a matrix of columns and lines. Film editing requires the definition of the number of images together with the number of columns and lines. Images are defined one by one or all at once and can be changed at any time.

Once the images are defined, the positions of the elements belonging to the same image can be defined. For this purpose, one must define the position of the first element of the image, the distance between columns, and the distance between lines. It is also possible to use increments to modify the interelement, intercolumn, interline, and interimage intervals. This permits one to edit different types of periodic structures very rapidly (checker-

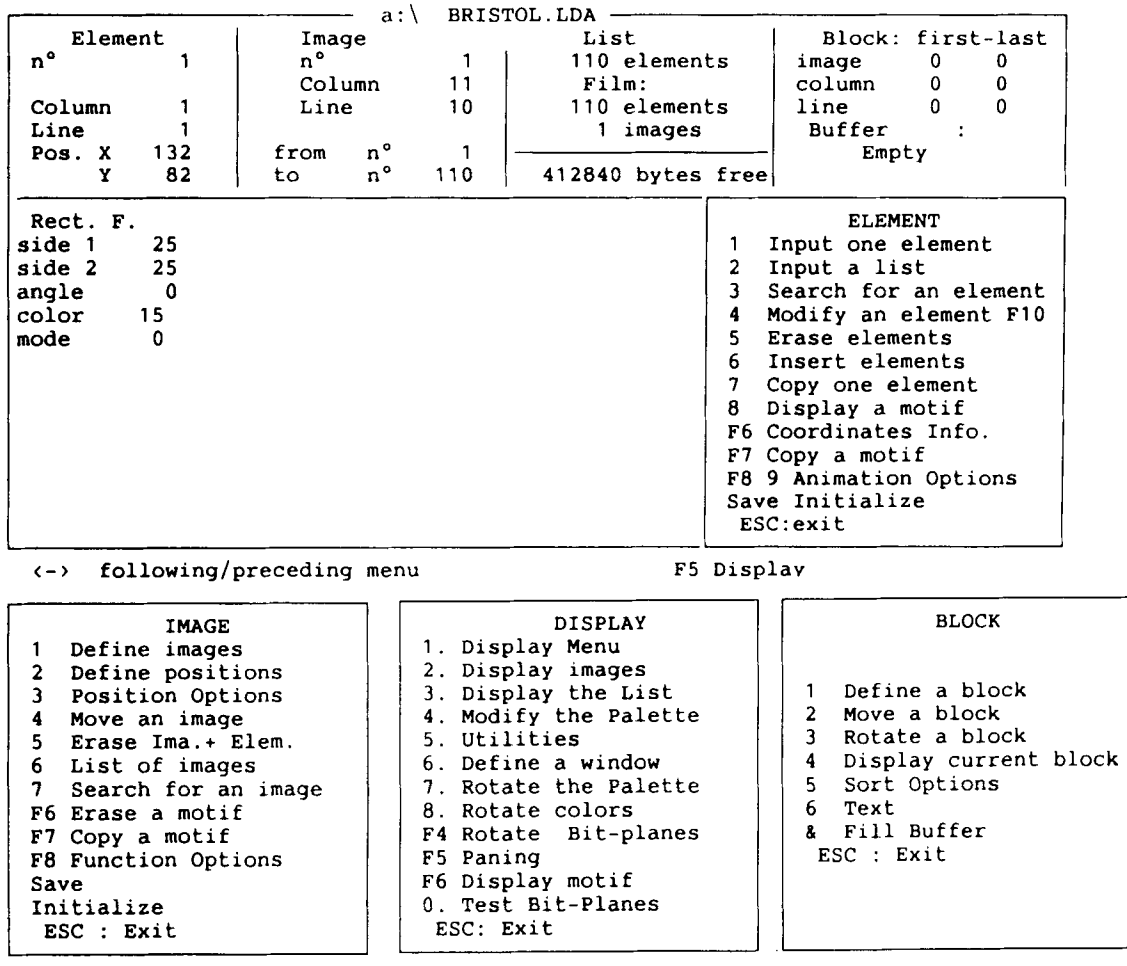

Figure 4. Copy of the control screen while in the edit mode. The menus for managing images, blocks, and display functions are also shown. Depending on the user's choice, submenus are displayed in the main window. However, information about the status of the file remains visible.

board, square-wave gratings, etc.). In addition, an image can be moved in any direction with the arrow keys. (Steps of displacement are adjustable with the plus and minus keys.)

Since one often has to deal with different aperiodic structures, facilities for the positioning of elements are provided: One can fill in a pattern with the elements of the list, given the intervals x and y between elements and lines of elements. One can also copy a motif of positions within boundaries of the elements' list. For instance, if three or more elements have been placed at different locations, their relative positions can be assigned to any group of three or more elements. Absolute positions can be defined by fixing the amount of displacement on the x- and y-axes relative to the copied positions.

Random noise can be applied to an image in order to modify the positions of the elements by a random amount on the x- and y-axes separately. A similar randomization can be applied to each of the fields of the element's structure (color, length, etc.).

These different functions appear to be very powerful for building simple or complex images, simply by moving between positioning functions and image definitions. For instance, if several images are defined and positioned, a simple redefinition of these images as one single larger image will produce a much more complex structure, which in turn can be replicated and modified. The result will be a film of complex images.

LEDA can also be used to edit fractal images (Mandelbrot, 1982). The construction of fractal images requires the choice of a number of transformations of the element's list (see Barnsley & Sloan, 1988). Each transformation is defined by seven values, including the probability of applying the transformation onto the list. These transformations apply on x and y locations and are added to the current positions of the elements (for that reason, all the elements should be initialized to the same location before constructing a fractal image).

The block level is designed for manipulation of parts of images, such as the elements between two different columns, two different lines, and two different images. A block can be moved, rotated, or erased. At this block level (but also accessible from the element and the image level), part of the list (i.e., a block or a motif) can be

copied in a buffer. The contents of the buffer can be reinserted anywhere in the list. (This function corresponds to the copy-cut-paste functions available on Macintosh computers, but it should be noted that LEDA deals with elements rather than bit maps).

There are different animation procedures for computers (see Proffitt & Kaiser, 1986). One can draw the different images sequentially, with the constraint that the time between frames depends on the time needed to write the images. Other possibilities offer faster and more flexible animation. One is to write all the images in a different part of the frame buffer. The animation is then produced by panning the frame buffer with appropriate values in order to present the images within a window. This method has the constraint that the number and the size of the images cannot be too large. When several bit planes are separately accessible on the graphics display, and few colors are displayed simultaneously, one can write each image on one or several bit planes. Animation is obtained by showing the selected bit planes at a chosen rate. Dynamic changes in the look-up table (LUT) have also proven to be powerful tools for animation.

The choice of one or another procedure (or their combination) depends on the required animation rates, the complexity of the images, the number of colors to be displayed simultaneously, and/or the number of images that make a film. In apparent motion perception (Braddick, 1974; Cavanagh & Mather, 1989; Papathomas & Gorea, 1988), one often uses cyclic presentations of a reduced number of images. This allows the use of more powerful animation procedures (i.e., the faster rates).

In LEDA, the film level includes display functions and several other useful functions, such as the edition of colors (i.e., changes of the LUT), zooming, panning, testing of bit planes, and so forth.

The display menu permits one to look at a film—several images—at a chosen rate, with or without an interimage interval. The images are displayed one by one in sequence. This menu allows a control of what should appear during an experiment, although the rate is not optimized, since information, printed on the control screen, is time-consuming (moreover, the rate depends on the number and type of the elements within an image; filled elements are drawn more slowly than outlined elements).

LEDA also provides a cyclic display of the bit planes (on an 8-bit graphics card, each bit plane can be accessed separately for read or write operations). One can define one, two, four, or eight planes to produce the desired segmentation of the eight available bit planes. Animation is realized by cyclic presentation of the selected bit planes with an adjustable delay between frames. The rate of animation is then independent of the complexity of the image. Along the same lines, a cyclic modification of the LUT is available to produce high-rate animated images. Suppose, for instance, that one image corresponds to 100 lines displayed side by side, but that 99 lines have the color of the background (i.e., they are invisible). If the color of the remaining visible line is successively assigned to each of the lines, one will perceive an apparent motion of one single line. The rotation of the colors of the LUT available in the program can be realized within boundaries at a chosen rate. Moreover, different parts of the LUT can be rotated simultaneously, which allows different animation rates for different objects (e.g., two or more gratings can drift in different directions at different rates).

The LUT available with 8-bit planes has 256 entries (i.e., 256 colors can be displayed simultaneously). Each color is defined by the intensity of the red, green, and blue channel. LEDA proposes different predefined definitions of the LUT that can be chosen in a menu. In order to manipulate the LUT further and thus change the colors, several functions are available. One can change each color separately by adjusting the red, green, or blue channel. The values of calculated functions can also be applied to each channel separately or to the three channels at once. With these functions, it is easy to define different palettes. For instance, one can define a sinusoidal variation of the intensity of the red channel and a linear increase of the intensity of the green channel, while the intensity of the blue channel decreases. These functions can be applied within limits. The LUT defined in this way can be saved (or not), together with the images.

POTENTIAL EXTENSIONS OF LEDA

Although the current version of the software allows the construction of a large number of different images, several extensions are possible. One extension under development is the management of a mouse for positioning or editing elements or images.

In order to run different experiments with specific methods and procedures with the images edited with LEDA, a specific module intended for running experiments can be added to the existing ones. We developed such a module for running experiments in our laboratory (Gorea, Lorenceau, Bagot, & Papathomas, 1990; Lorenceau & Shiffrar, 1990). The experiment module allows the definition of the parameters and procedures that are needed for specific animations used in experiments, together with the definition of specific variables such as the name of the subject, the number of trials, and so forth. This strategy is very useful for adjusting the parameters of one experiment; it is economical, since all the functions used to load a film or display the elements are already written; and it is flexible, since only the experiment module needs to be modified for a new experiment.

The software manages text, so LEDA can be used in research concerned with reading. Each element can hold a letter, a word, or a sentence. The size of the letters, their color, and their position can be changed easily. Image editing could be used to display several letters, words, or sentences. The display facilities can be used to run experiments.

Another extension concerns the synthesis of images from frequency spectra. It is possible to use the fields of an element's structure to define frequency, intensity,

phase, location, and so forth. The program can allow for manipulations (e.g., filtering) of the current spectrum before it is sent to a Fourier synthesis module, in order to draw images.

LEDA could also be of use in other fields of research, such as psychoacoustics. For instance, the different fields of an element's structure could be used to define parameters such as the frequency, intensity, and phase of different sounds. Only the messages displayed on the control screen and the output module that are actually used to draw elements on a graphics display need be rewritten in order to fit the hardware specifications.

The software's limitations should also be mentioned. One is the limited number of elements and images that can be managed simultaneously in the current version. About 400K are available for the data (elements and images). Extended or expanded memory is not presently managed by the software. In addition, the matrix structure of images does not allow three-dimensional manipulations.

CONCLUSION

A general multipurpose software package for editing and animating images was developed to avoid the disadvantages that occur when a new program must be written for each new experiment. This approach is very powerful and saves a great deal of time. Several experiments that have been run with this software have required a minimum of additional programming, although it was sometimes easier to write an additional specific procedure than it was to edit the images with the tools available. However, a new procedure is easier to write when the data structure is already defined.

The adaptation of LEDA to different graphics displays is ensured by its modular design, although some of the graphics functions in the present version may not exist in other systems, and some functions available on more sophisticated systems cannot be implemented in LEDA. Thus, some time may be required for the user to adapt the program to work with different graphics displays. Although a version that runs on an AT with a single monitor is available for demonstration, the program should be used with two monitors for maximum flexibility and adaptability. Appendix B contains the procedure used to display an element on the screen. Procedures used to display a motif, an image, or a film are repetitive calls, within limits, to this "display element" procedure. Other modifications concern panning, zooming, writing a single color or the whole palette, and so forth. It should be noted that the procedures used in LEDA are available on most of the graphics cards currently available, running on an AT with MS-DOS 3.3. Adapting LEDA to run with different operating systems (e.g., UNIX) requires the change of low-level procedures for management of files, directories, and so forth.

If one is interested in working with LEDA, the program and documentation are available from the first author for $40 US. A version that runs on a PC (VGA display) will be provided for testing; the program requires an ADAGE PG90/10 graphics card.

REFERENCES

BARNSLEY, M. F., & SLOAN, A. D. (1988). A better way to compress images. *Byte*, **1**, 215-223.

BRADDICK, O. (1974). A short range process in apparent motion. *Vision Research*, **25**, 839-847.

CAVANAGH, P., & MATHER, G. (1989). Motion: The long and the short of it. *Spatial Vision*, **4**, 103-129.

GOREA, A., LORENCEAU, J., BAGOT, J. D., & PAPATHOMAS, T. V. (1990). Color-based motion perception may be stronger at equiluminant than under nonequiluminant conditions. *Investigative Ophthalmology & Visual Science*, **31**(Suppl. 2544-51), 518.

LORENCEAU, J., & SHIFFRAR, M. (1990). Motion viewed through several apertures: Non-rigid percepts. *Investigative Ophthalmology & Visual Science*, **31**(Suppl. 2557-61), 520.

MANDELBROT, B. (1982). *The fractal geometry of nature*. San Francisco: W. H. Freeman.

PAPATHOMAS, T. V., & GOREA, A. (1988). Simultaneous motion perception along multiple attributes: A new class of stimuli. *Behavior Research Methods, Instruments, & Computers*, **20**, 528-536.

PROFFITT, D. R., & KAISER, M. K. (1986). The use of computer graphics animation in motion perception research. *Behavior Research Methods, Instruments, & Computers*, **18**, 487-492.

APPENDIX A

To provide an example of the use of LEDA, we describe the steps needed to construct the Bristol wall illusion.

1. Enter LEDA and select the "edit" mode.
2. Select "Input one element":
   - define a filled rectangle and its characteristics:
   - sides, color, mode.
3. Select "input a list":
   - choose "all at once" option.
   - choose "copy last element."
   - enter the number of elements.
4. Use down arrow to display next element.
5. Select "modify an element" option:
   - change the white color to gray.
6. Select the "copy a motif" option:
   - enter: the size of the motif and its starting element.
     : the limits for copying.
   - select the element's fields to copy through the list.

APPENDIX A (Continued)

7. Use the right arrow to display the image menu.
8. Select the "define an image" option:
   - enter the number of images to define.
   - select the "all at once" option.
   - enter: the number of columns and lines.
9. Select the "define positions" option:
   - enter: x-, y-coordinates of the first element.
     : the separation between columns and lines.
     : the step increment on the x-axis.
     : parameters for "alternate step's sign" option.
10. The image is displayed on the screen.
11. Each step can be reselected to change colors, positions, and size or structure of images, or to apply a function on one field of the element's list. Animation procedures are available.

APPENDIX B

```
/* ---- Procedure used to display an element on the graphic screen ---- */
   void ON_elem(liste_EL *graf)
   {
   dsopm(graf→mod_disp);                    /* set the mode for combining element */
                                            /* with the background */
   dscol(graf→color);                       /* set the color of the element */

/* move to the center of the screen if x = y = 0 */
/* move to real x & y otherwise */
   if (graf→xposel == 0 && graf→yposel == 0) dsamove(CentX, CentY);
   else dsamove(graf→xposel,graf→yposel);

/* Selection of what procedure to call */
/* depending on the geometry of the element, coded by a number (0 to 15) */
/* Coordinates of the element are relative to X & Y location */
   switch (graf→geometry)
   {
/* undefined geometry */
   case 0: return;

/* Display a dot */
   case 1: dsdot();
           return;

/* Display a line whose coordinates are stored in an array of integer */
   case 2: dsrmove(graf→ar[0].x, graf→ar[0].y); /* makes a relative move */
           dsrline(graf→ar[1].x, graf→ar[1].y);  /* draw a relative line */
           return;

/* Display a circle whose radius is equal to 'len' */
   case 3: dscrcl(graf→len);
           return;

/* Display a disk. MOPIN is the mode for filling in */
   case 4: dscrcl(graf→len);                    /* Display a circle */
           dsrpaint(0, 0, graf→color, MOPIN);   /* makes a relative paint */
           return;

/* Display an outlined rectangle relative to XY location */
/* Length and height are stored in an array of integer */
   case 5: dsrrct(graf→ar[0].x, graf→ar[0].y);
           return;
```

APPENDIX B (Continued)

```
/* Display a filled rectangle relative to XY location */
/* Length and height are stored in an array of integer */
   case 6: dsrfrct(graf→ar[0].x, graf→ar[0].y);
           return;

/* Display a polyline of 'arsize' sides */
   case 9: dsrpll(graf→arsize,graf→ar);
           return;

/* Display an outlined polygon regular (7) or irregular (10) */
/* Makes a relative move from XY location to the first XY coordinates */
/* and draws a polygon of 'arsize' sides */
   case 7:
   case 10:dsrmove(-graf→ar[graf→arsize].x, -graf→ar[graf→arsize].y);
           dsrplg(graf→arsize-1,graf→ar);
           return;

/* Display a filled polygon regular (8) or irregular (11) */
/* Makes a relative move from XY location to the first XY coordinates */
/* draws a polygon of 'arsize' sides */
/* fills in from the XY location to the border */
   case 8:
   case 11:dsrmove(-graf→ar[graf→arsize].x, -graf→ar[graf→arsize].y);
           dsrplg(graf→arsize-1,graf→ar);
           dsrpaint(graf→ar[graf→arsize].x,graf→ar[graf→arsize].y,graf→color, MOPIN);
           return;

/* Display an outlined ellipse defined by the ratio of X/Y and its length */
   case 12:dselps(graf→ar[0].x,graf→ar[0].y,graf→len);
           return;

/* Display a filled ellipse defined by the ratio of X/Y and its length */
/* MOPIN defines the mode for painting */
   case 13:dselps(graf→ar[0].x,graf→ar[0].y,graf→len);
           dsrpaint(0, 0, graf→color, MOPIN);
           return;

/* Makes a call to a procedure that display a text */
/* ON_text transforms integers stored in an array into letters */
/* concatenates these letters and display the string */
   case 14:ON_text(graf);
           return;

/* Makes a call to a procedure that displays a sprite */
/* writ_sprite reads an array of colors */
/* draws a dot within a squared matrix */
   case 15:writ_sprite(graf, graf→mod_disp);
           return;

/* Makes a call to a procedure that draws a fractal element. */
   case 16:ON_frac(graf);                    /* plots colored dots only */
   }
}
```

Note—The ON_elem() procedure is called each time an element is displayed on the graphic screen. To adapt this procedure to another graphics card, one must change the calls to the graphics library—for example, change dsamove(x,y) dscrcl(radius), which are used here to draw a circle to circle(x,y, radius) if a VGA graphics card is to be used. Other modifications—for panning, zooming, writing the LUT, and so forth—are also needed. All these graphic functions are grouped in the same module. If only one screen is available for graphics and controls, switches between text and graphic modes should be inserted within the software.

APPENDIX C: Black and White Examples of Images Edited with LEDA

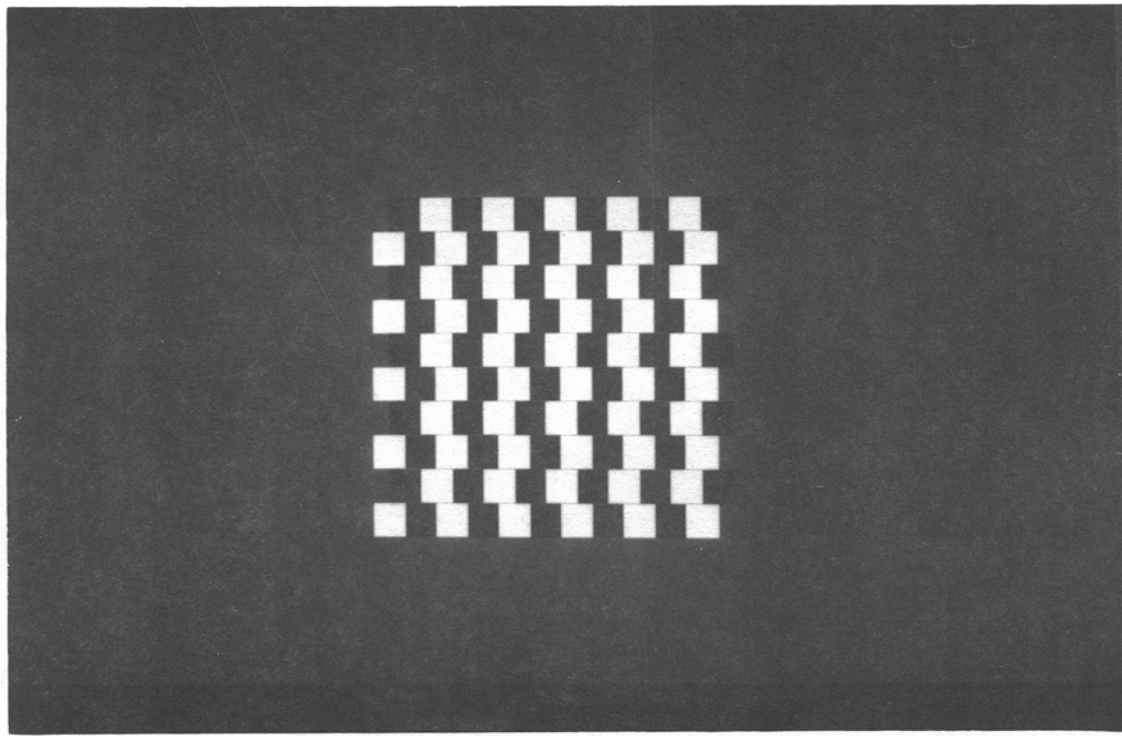

Image 1. Size in memory: 6,591 bytes. Editing process: 2min.

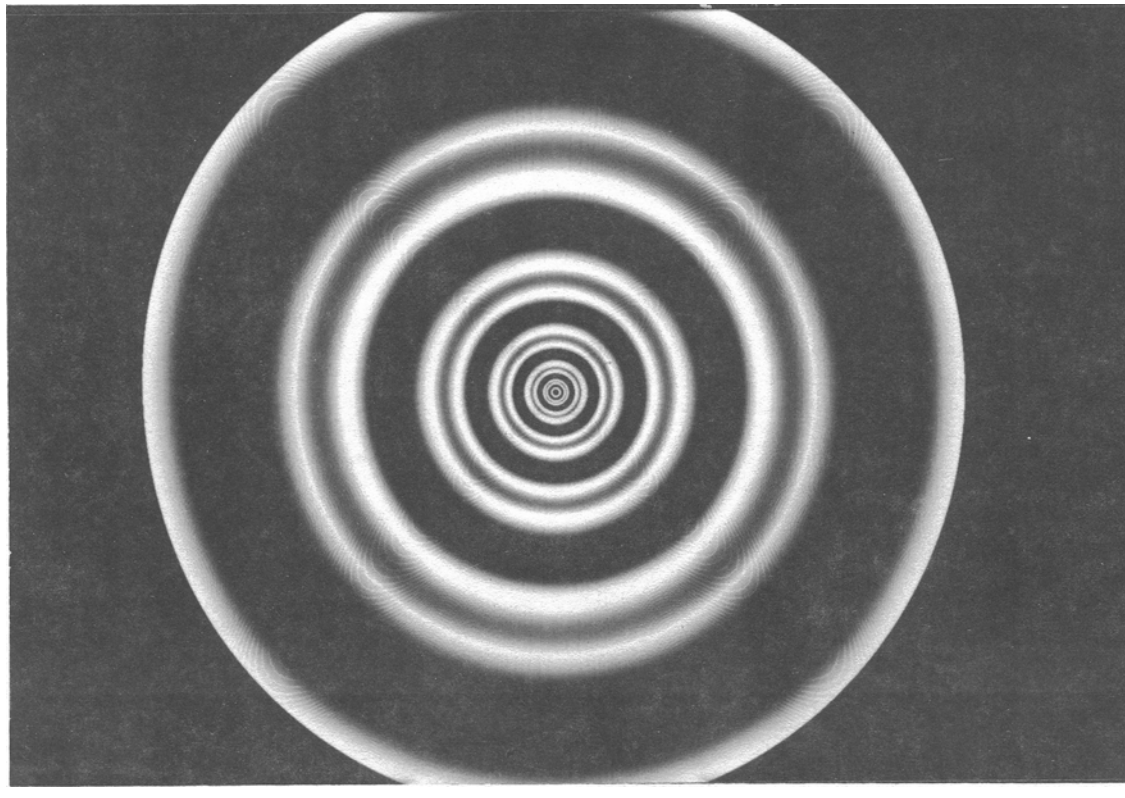

Image 2. Size in memory: 10,529 bytes. Editing process: 3min.

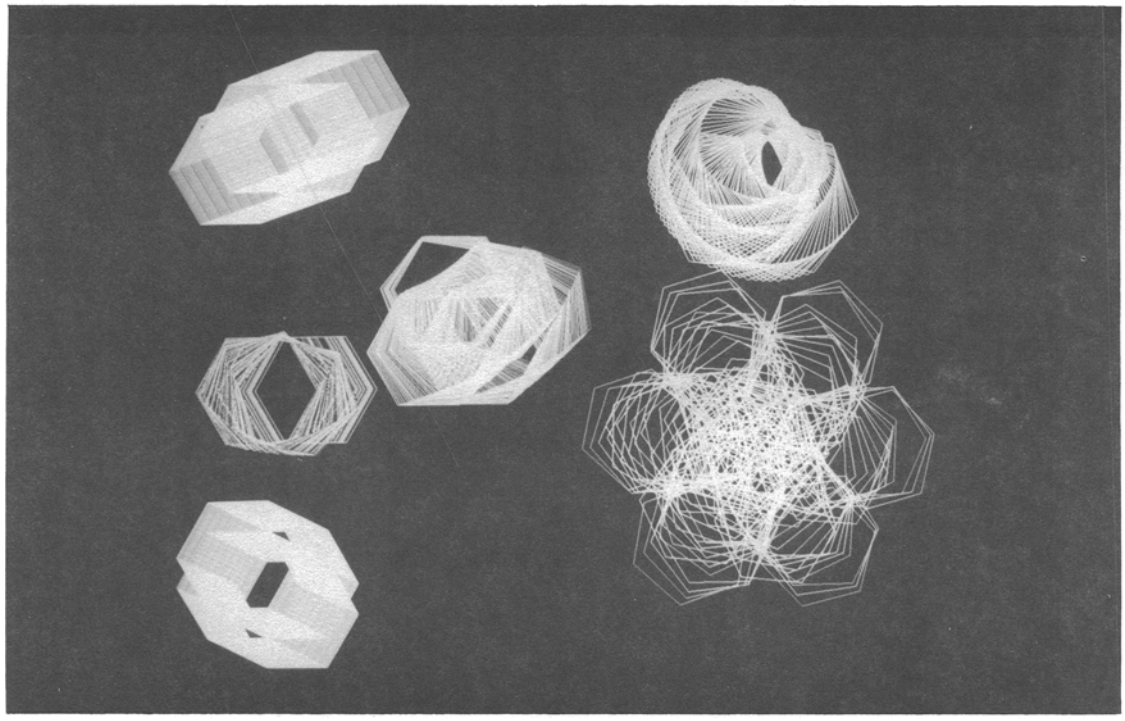

Image 3. Size in memory: 63,081 bytes. Editing process: 10 min.

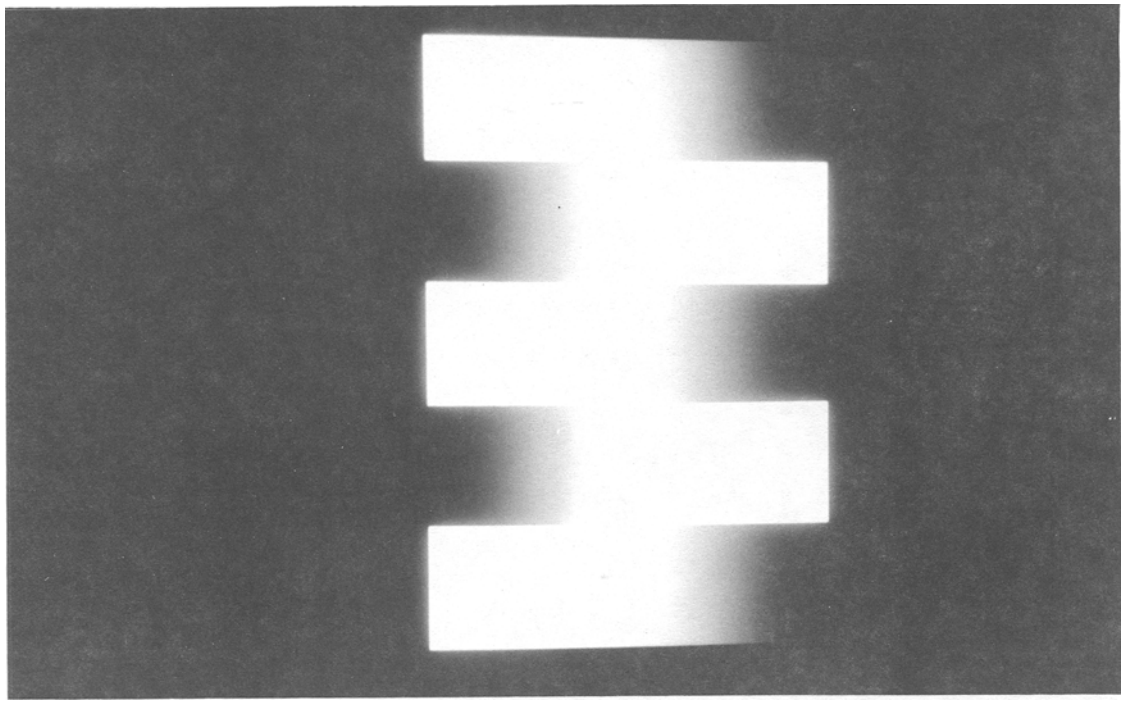

Image 4. Size in memory: 114,581 bytes. Editing process: 5 min.

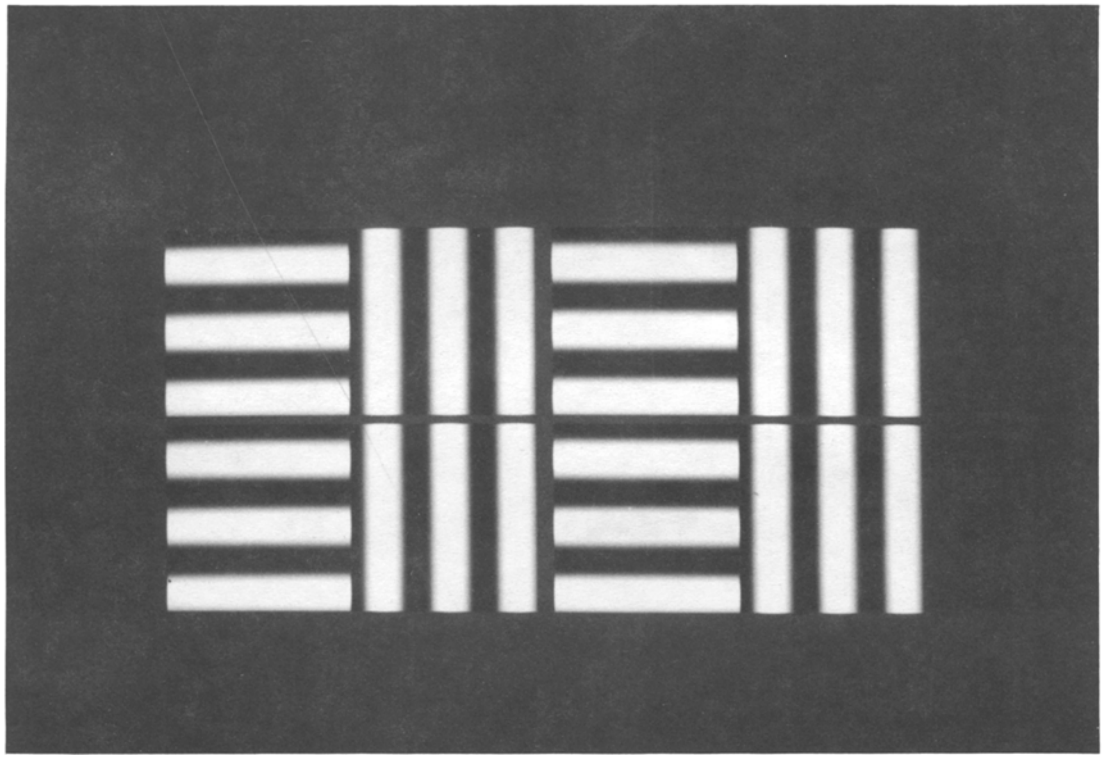

Image 5. Size in memory: 80,208 bytes. Editing process: 10 min.

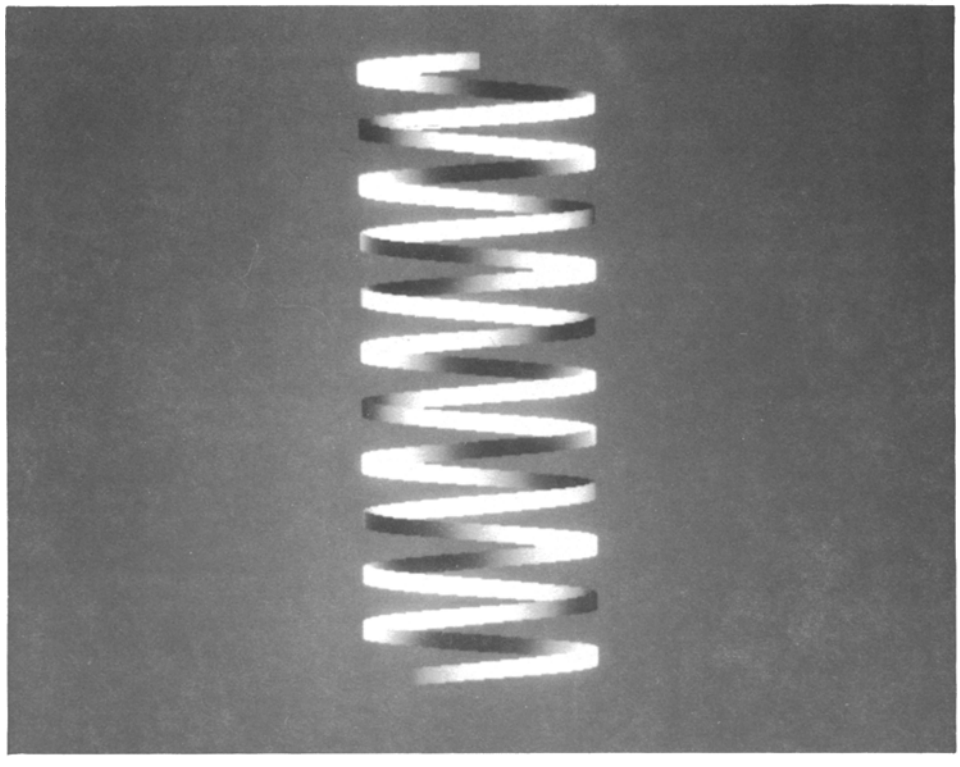

Image 6. Size in memory: 166,081 bytes. Editing process: 12 min.

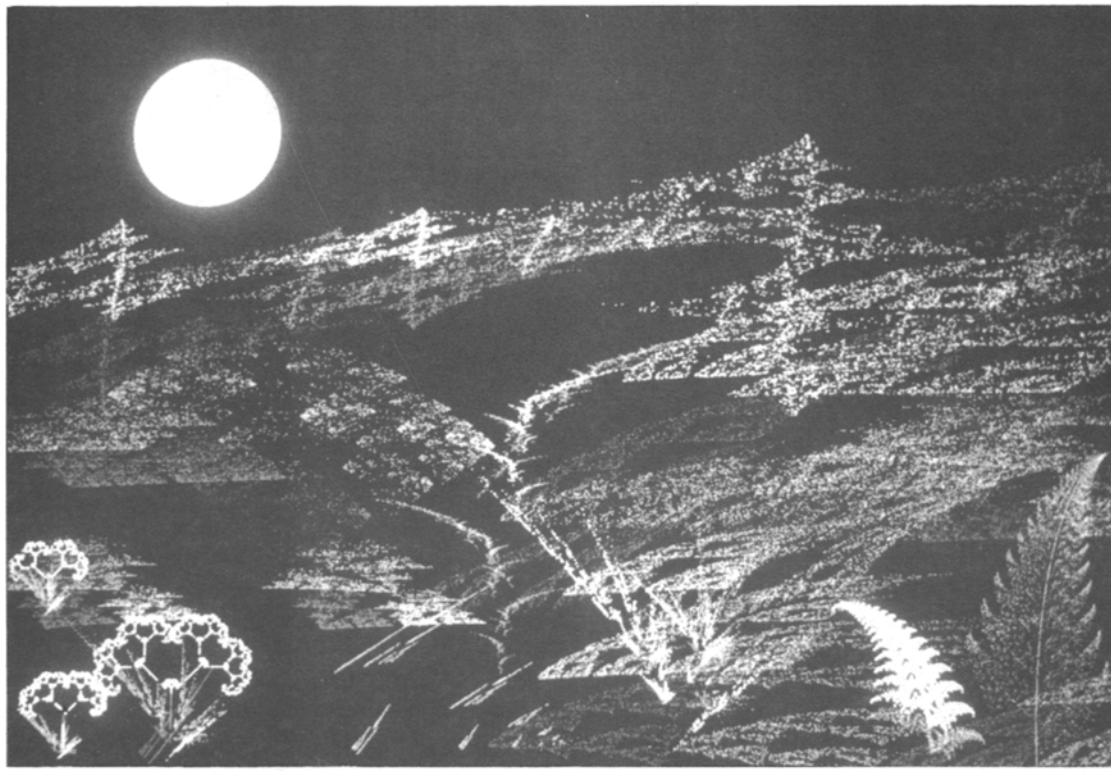

Image 7. Size in memory: 4,703 bytes. Editing process: 30 min.

(Manuscript received March 19, 1990;
revision accepted for publication July 30, 1990.)