Computer Art and Creative Tool Making

by Yuan-Lin Mao
B.A., Brandeis University
May 1982

Submitted to the Department of Architecture in partial fulfillment of the requirements for the degree of Master of Science in Visual Studies at the Massachusetts Institute of Technology

June 1985

[illegible]

Signature of Author, Yuan-Lin Mao, Department of Architecture, May 13, 1985

[illegible]

Certified by Muriel Cooper, ~~Associate~~ Professor of Visual Studies

[illegible]

Certified by Nicholas Negroponte, Chairman, Departmental Committee for Graduate Students

Table of Contents

Computer Art and Creative Tool Making

by Yuan-Lin Mao

Submitted to the Department of Architecture on May 13, 1985 in partial fulfillment of the requirements for the degree of Master of Science in Visual Studies at the Massachusetts Institute of Technology

Abstract

A digital paint package has been developed which places attention on the design of personal "brush" patterns. The user generates an image by iterating these pattern modules on the raster display. During the application of a pattern, it can grow, shrink, and change in opacity level under the user's control. This method of digitally creating images was developed in light of the problem of representing visual characteristics effectively while "painting" with a computer graphics system. Allowing the user to design personal brush patterns, which can be stored in a library of patterns, and to make marks by repeating them, expands the potential visual qualities of the image as demonstrated by sample images included in this thesis. Software functions are provided for creating and editing patterns through a menu of selections. These functions treat individual shapes and colors in a pattern as separate entities that can be manipulated. Shapes can be manipulated individually, or as a selected group. The manipulation functions include the following: move, copy, scale, and delete. Software functions are also provided for the editing of color components. One method allows a color's red, green, and blue components to be adjusted. And the other allows its hue, lightness, and saturation levels to be adjusted.

Thesis Supervisor: Muriel Cooper
Associate Professor of Visual Studies

Introduction:

With the development of computer graphics in recent years, yet another medium for visual communication is being provided to artists. The general term "computer graphics" describes a wide range of applications and techniques for generating images by means of a computer. This thesis concentrates on the application of computer graphics in the area of digital painting and addresses the problem of representing visual phenomena effectively.

Digital painting, just as the traditional sense of painting, is the interactive process in which the artist makes marks to a surface in order to render an image. Unlike traditional paint media, the necessary supplies, brush, pigment, and painting surface, are not material substances which are governed by physical rules, but are products of digital information stored in the computer's memory, and are governed by rules that a computer scientist prescribes. Though the prescription of rules for controlling the medium is a favorable attribute of using the computer tool, digital painting shares with traditional paint media the age-old problem of visual representation, in addition to the device problems peculiar to the computer system. Digital painting, as defined in this thesis, incorporates an electronic input and pointing device, such as a mouse or a stylus, which behaves as the mark making tool, and a table of digital values stored in the computer's memory which define the possible colors that can be assigned to pixels in a raster image. This table of values is analogous to an artist's palette of colors. Painting, in this context, is the interactive process by which the user assigns color values from the color table to pixels in the raster image by selecting the locations of the pixels with the input device.

This method of "painting" is not adequate for the rendering of visually interesting or complex images, if the user is not provided with additional software tools that operate on a selection of pixels that the user is pointing at with the electronic mouse or stylus. Without these tools, the user must indicate color values for each and every pixel in the image, which is a trying task for a medium or high resolution image of over 200,000 pixels. Software tools which allow the user to apply marks to the image, that suggest the visual characteristics of physical media, such as charcoal or watercolor, have been developed in some digital paint systems. These tools do increase the potential for a user to "paint" visually satisfying images, however the potential of the digital medium should be explored beyond the production of tools which simulate physical media.

Because the digital paint medium is developed from software which instruct its behavior, routines can be developed which allow the user to define the characteristics of the mark being applied. This would expand the variety of marks or patterns to "paint" an image with. In addition, the user is contributing to the image quality, not only by deciding the composition of the image, but also by defining the characteristics of personal patterns which build the image. In traditional media, the artist's only personal contribution to mark making is the manner in which he/she controls the implement that applies the pigment to the picture through gesture. The quality of the mark applied is a product of the user's gesture, the surface of the image, and the quality of the material which provides pigment, such as oil paint or charcoal. In digital painting, though the gesture of the artist may not be possible to communicate through the input device, the user can personalize his/her image by defining the characteristics of tools, if effective editing software are provided. Focusing on a particular class of tools for the user to create, and providing effective editing software has guided the research of this thesis.

The rendering of visual characteristics, particularly of nature, has been a non-trivial problem throughout the history of painting. This has been a contributing factor to the development of various schemata and styles amongst different cultures and people through the ages in attempts to describe the world. Identifying the visual elements of an object's appearance prior to the rendering of it onto an image surface, can be observed in the rigid methods of Chinese brush painting, as well as the experimental methods of the Impressionists. For rendering objects, the goal of the methods developed by the Chinese painters was to convey the essence of what is being visually described in a minimum of brush strokes and colors. The impressionists, however, juxtaposed dabs of high-toned primary colors of varying intensities as the means to render what they considered the unifying elements of a picture, the interplay of light and colors.

In digital painting, considering the problem of representing visual phenomena poses the question of what tools and software should be developed for a paint system which would allow users to generate images effectively. Unlike traditional media, many graphics systems do not have mark making devices that allow the user to control the applied mark through the gesture or pressure he/she applies to the tool. The system that this thesis project was developed on is one such example. An optical mouse with three buttons for user input is the interactive device that the user employs to apply pigment to the image. The software and hardware associated with this device allows only its location and status of its buttons to be read. Though this is a limiting factor of systems which do not have gesture or pressure sensitive devices, computers overall are efficient at tedious tasks such as repetition of instructions and managing a large memory of digital information.

In light of the problem of representing visual phenomena in digital painting, and attributes that a computer graphics system possesses, the digital paint package developed here focuses on the design of pattern modules by the user. These patterns are the basic components which are later iterated by the user to produce images. During the application of a pattern, it can grow shrink, and change in opacity level under the user's control.

This manner of building an image by the repetition of similar parts of various scales can be seen in other methodologies of picture rendering, though their parts are not defined as patterns, per se. This method of rendering images can be compared with the traditional methods of Chinese painting. Here, the applied characteristics of the brush stroke to the paper through gesture control form the "pattern" of repetition. In Chinese painting, an artist must first master the prescribed brush strokes for representing an object before he/she attempts to render it on the image surface. Leonardo DaVinci had also practiced repetition of forms. For example, in his illustrative studies on the impact of water falling on water, whirlpools are drawn in similar swirling lines, that suggest its form, as smaller eddies and whirlpools contained in it are drawn. Here, the swirling motions being suggested are the elements of repetition. It is no coincidence that the repetition of parts have been incorporated by various methods of representing natural phenomena. If we examine the visual characteristics of natural phenomena, we can notice that many of them are built on visual elements which are similar to each other. If we look at a flower, for example, we notice that it is composed of many petals which are similar to each other.

This thesis demonstrates that repetition of sub-images whose visual characteristics are defined by the user is an adequate method for controlling the visual qualities such as tone, contrast, color, and patterning in local areas of the image, which expands the type of images that the user can produce through digital painting. This methodology can be expanded in the future to allow the user to define potentially all attributes of the digital painting environment, such as suggested image surface quality, and the outcome between the "contact" of the digital "brush" and the digital "surface".

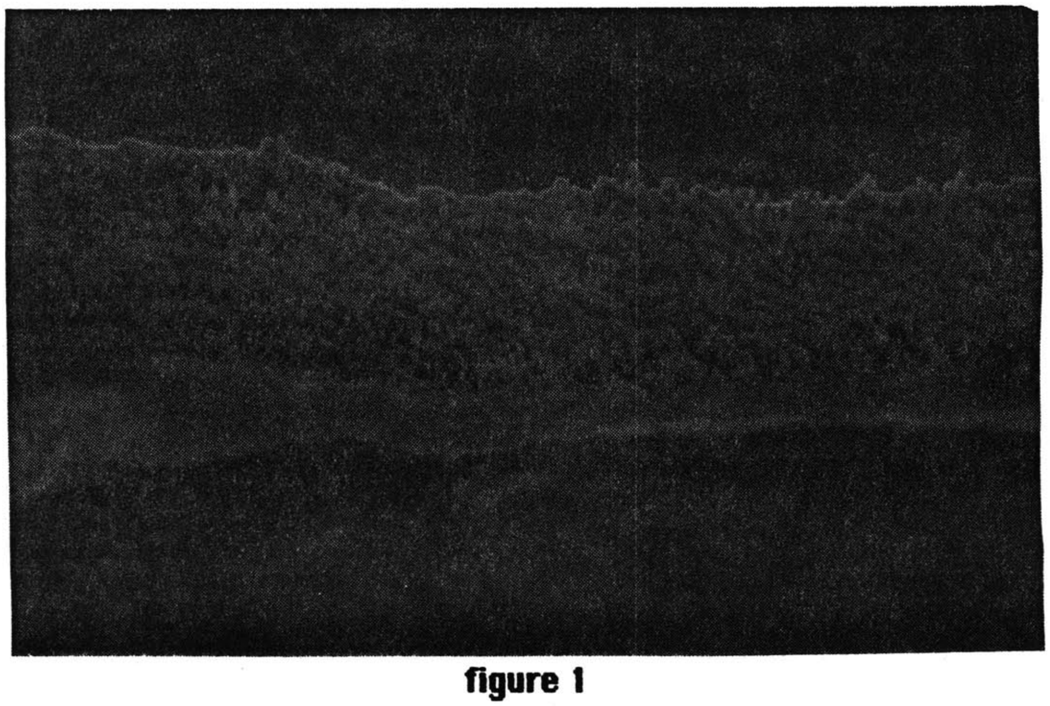


figure 1

Chapter 1: Digital Painting With Predesigned Personal Brushes

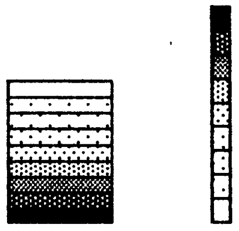
Background of the Project

The design and implementation of the software for the project developed from a culmination of observations that I experienced as a painter using traditional paint media (oils, pastels, and charcoal), and as an artist and computer scientist using routines which generated brush patterns on a computer graphics system.

While painting with traditional media, the realistic rendering of objects and people fascinated me. During the act of painting any object, such as an apple, a table, or a person, I noticed visual attributes about the object in local areas of its surface which I was unaware of or took for granted in my everyday life. I felt that the whole object could only be rendered faithfully if I satisfactorily rendered its parts. Besides rendering the color, luminosities, and shadows of particular areas of the object's surface, the object's texture also had to be dealt with. The rendering of different textures each involved particular instruments and techniques, ranging from a variety of charcoal densities and thicknesses, brush qualities and sizes, manual gestures and pressures to the application tool, and materials for applying or manipulating the pigment onto the surface, such as paper towels, palette knives, and my palm and finger tips.

From my experience with programs on computer graphics systems which allow the user to "paint" images, the tools provided were limiting in respect to the images I could generate with the brush patterns that were available. One program provided brush shapes of three rectangular sizes. A brush shape could only contain one color, however its opacity could be adjusted prior to its use. Another program provided an alternative method of choosing the brush. It allowed the user to specify a rectangular area of the image on the screen to use as a brush pattern. The limitations of these digital painting routines included the static application of the same size pattern at the same opacity level during the process of applying a continuous sequence of marks to the image, and the selection of brush shapes and patterns that the user could specify, in addition to the lack of tactile interaction which I had become accustomed to with traditional media.

An ancient expression in Chinese painting, "wu pi" (not having brush), appropriately describes the limitations I was experiencing. It describes the occasion when an artist knows how to outline the object in question, but does not know the strokes for modeling it. In terms of my digital painting experience, the patterns were not there. In reaction to the limitations described above, I designed and implemented programs which generated interesting patterns for digital painting. One such program allowed the user to indicate the rectangular size of the pattern and to indicate the extreme colors for the top line of the pattern and the bottom line of the pattern. The program, then, interpolated these colors to generate the adjacent horizontal lines which lay between the top line and the bottom line of the pattern. The colors of these lines differed incrementally from the color of the bottom line to the color of the uppermost line. The following are example brushes:



Later additions to the program included a means for the user to control the change in opacity level and size of the pattern as he/she is iterating the pattern onto the image. The user controlled these changes by clicking buttons on the mouse.

In using this "shaded" brush program, I experienced an important observation which directed my digital painting exploration from that point on. The iteration of the patterns over and over onto the image produced a great feeling of depth in the image which I had not experienced in either traditional media or in digital painting. (See **figures 2** and **3**)

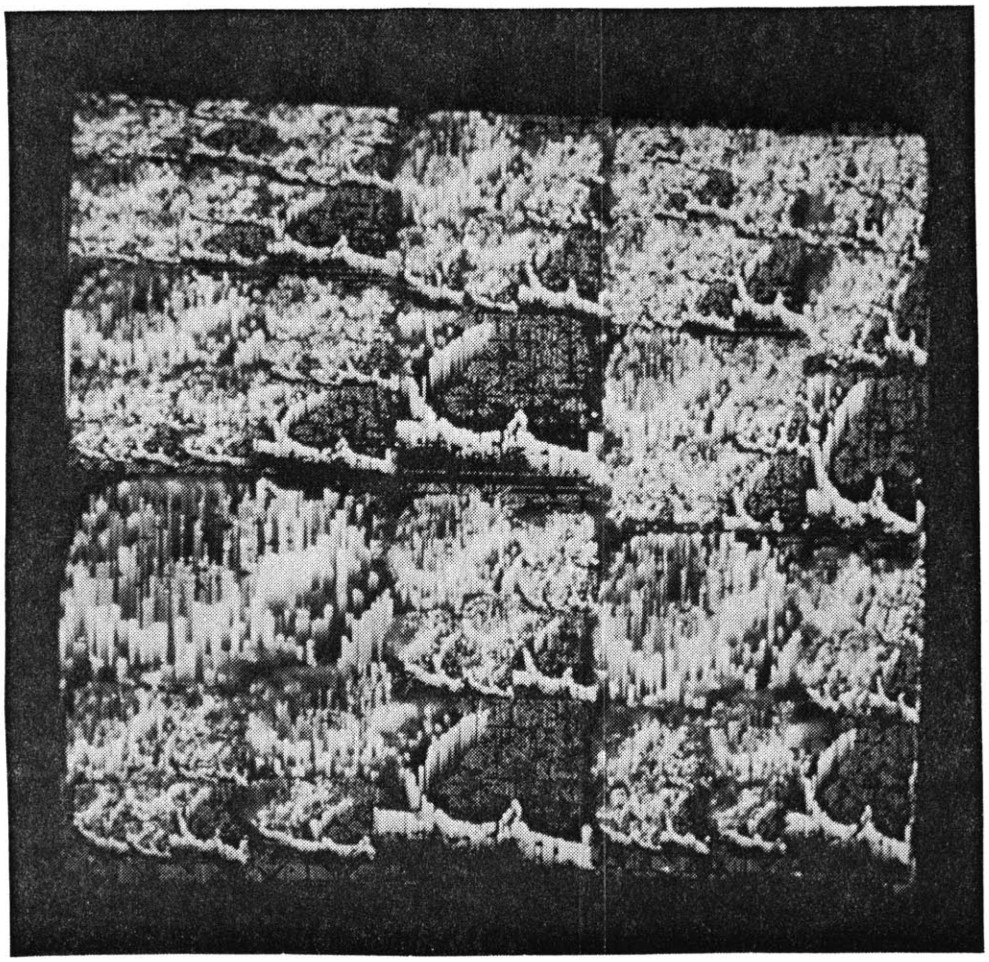


figure 2

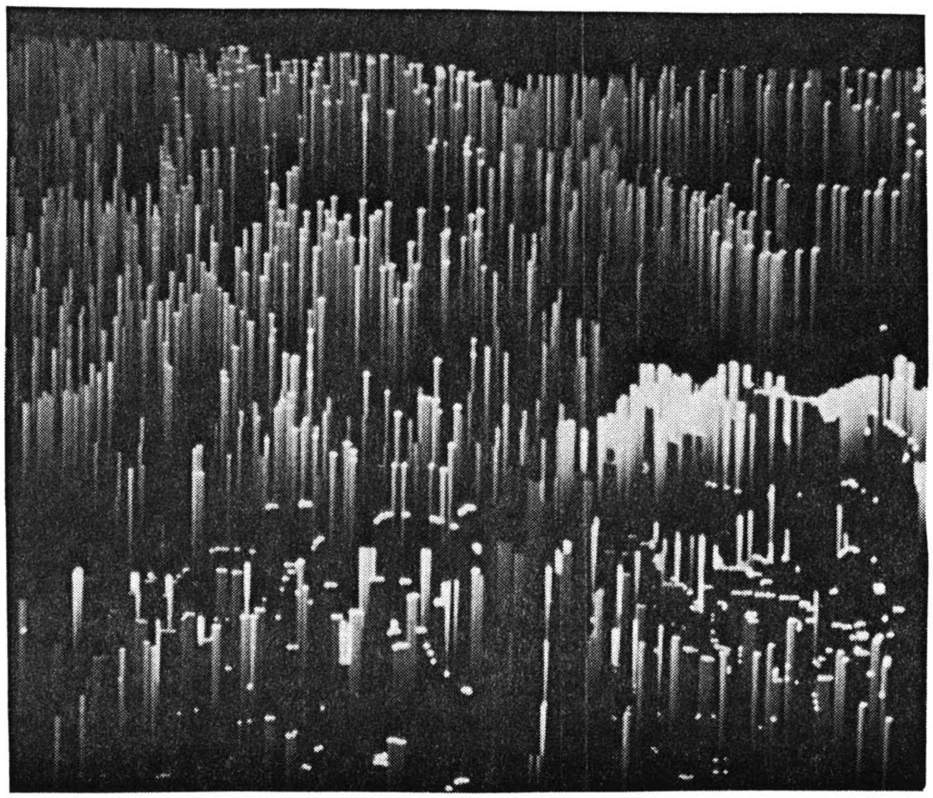


figure 3

In addition to this observation, I noticed that selecting a portion of the image generated by this type of digital brush, and scaling and copying it over parts of the image helped render objects which I was satisfied with as being complete and faithful to my intentions. It appeared that the repetition of patterns in various scales could suggest realistic qualities of an object. The variable factors of colors in the pattern and size of the pattern provided enough control for the user to suggest a wide variety of visual phenomena. The pattern's colors and shape in conjunction with the user's application of the pattern onto the image surface seemed to be sound components for suggesting an image of realistic visual qualities, or an interesting abstract image that demonstrates aesthetic qualities of digitally painted images which could not be easily rendered in traditional media. Being able to adjust the colors contained in the brush patterns allowed the user to determine the tonal, textural, and contrast qualities of the image.

This method of repeating patterns of various scales immediately from a click of a button is an advantageous quality that the digital medium provides. This is difficult or impossible to execute using traditional paint media.

The project developed and described here demonstrates that predesigning patterns and applying them at various scales and opacity levels allows a digital painter to generate a variety of image qualities which are tedious to produce in traditional media as well as in other existing digital paint systems. Unlike the "shaded" brush program described above, this package allows users to design patterns of various shapes and colors and to set up a library of patterns to choose from during the actual picture making process. The means for allowing the user to design patterns takes advantage of the computer's capability for recording a vast amount of information, and its efficiency in evaluating and processing stored information. The software developed allows the user to treat each constituent shape and color of the pattern as individual items that can be manipulated. This allows patterns to be quickly and easily generated and modified.

Breaking Down the Desired Product Into Easy to Render Parts

-Example Images and Brushes

Obviously, a different problem now arises for rendering images with the method of predesigning its constituent parts. Here, a naive user of this method is in a situation of "wu pi" (not having brush). Whereas the artist using the method of Chinese painting is aware of a vocabulary of brush strokes to master, the artist using this method of digital painting must concentrate on designing an appropriate pattern that will yield the faithful production of the image he/she has in mind.

The artist must now abstract an adequate set of characteristics which make up the desired object to render. For example, in order to generate the texture of the bark of the tree in **figure 4,** I assumed that designing **pattern 1** with the configuration of squiggly fat lines of different tones of brown would be an adequate set of shapes, which, when scaled down and iterated would suggest the appearance of bark.

In generating the quality of the figure's clothing, I decided that since the clothing has folds and wrinkles in it, a pattern composed of solid circles of varying degrees of beige should be designed (**pattern 2**). The outer rings of the pattern are specified as darker tones of beige in order to suggest shadows of the folds, and the inner rings are lighter to suggest highlights of the material. The technique of suggesting texture through highlights and shadows is widely known by artists of all paint media. The significant characteristic of generating the clothing with this type of "inking" method in this paint package is that both shadow and highlight are rendered at the same time, and there is a continuous flow of pigment.

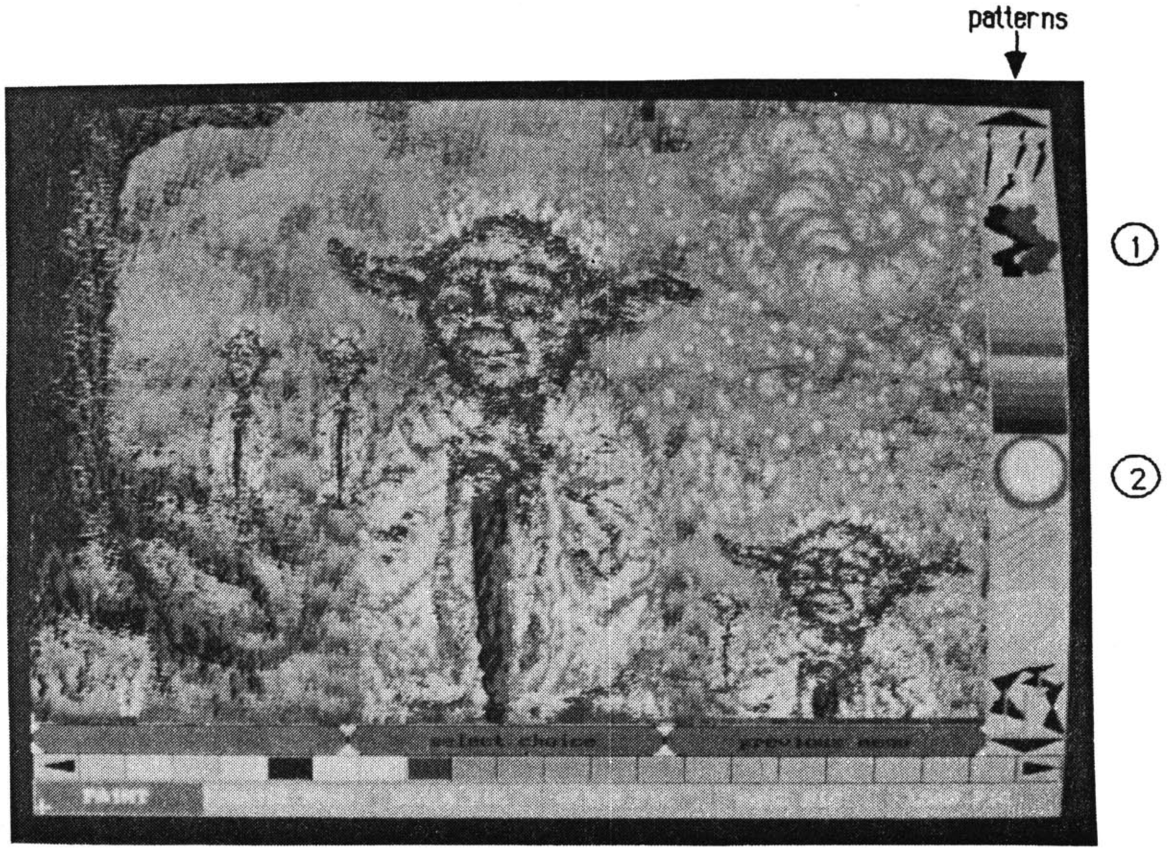


figure 4

When iterating the pattern using the "Dynamic Draw" routine in the package, the user is not constrained to repeating a static pattern. The user controls whether the pattern is shrinking, growing, or staying the same size, and whether it is becoming more opaque, more transparent, or staying at the same opacity level. With this in mind, I was able to "paint" the tree in **figure 5** by iterating a pattern of lines (**pattern 3**), which represented needles of the tree, and controlling its change in size. Notice that the needle patterns at the tips of the "branches" are smaller than the ones at the top of the tree.

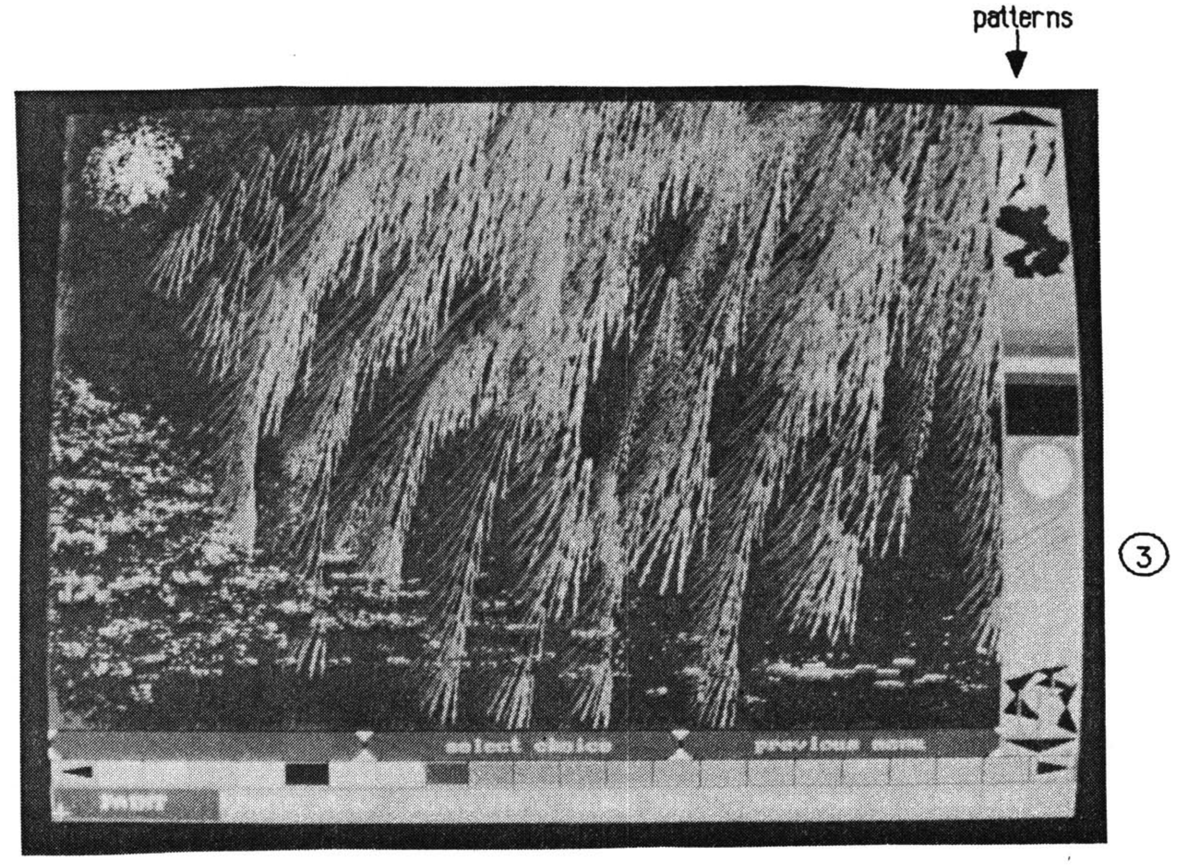


figure 5

Figure 6 represents an aerial view of clouds, trees, and a beach. The trees were generated by **pattern 4**, the clouds were generated by **pattern 5** and the water was generated by **pattern 6**.

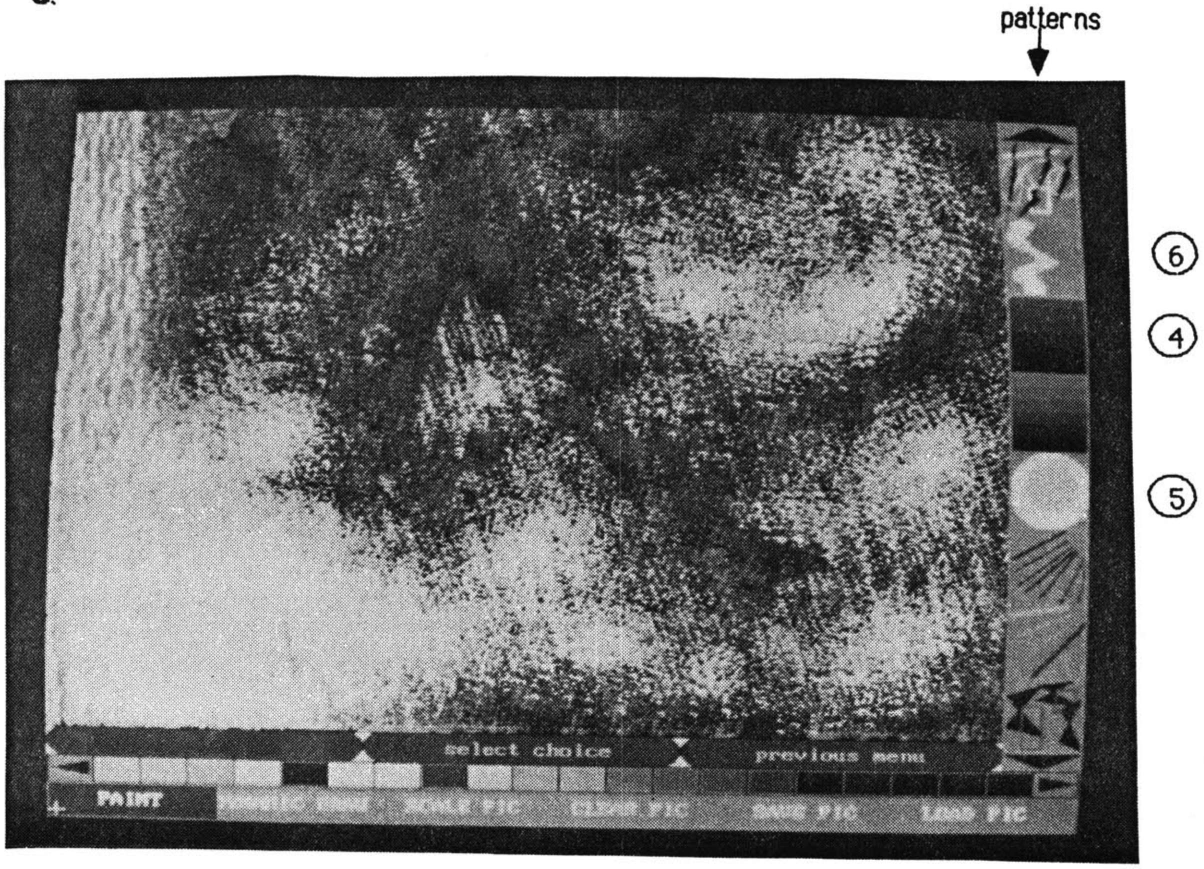


figure 6

It is not too difficult to decide on the patterns to design, since we are all able to identify various visual phenomena in reality. We have stored in our memory for each category of things or members of categories, which we have encountered earlier, a set of visual characteristics which aid us in recognizing the type of object. In generating a particular visual quality of reality, the user should concentrate on designing an adequate set of shapes from the characteristics stored in his/her knowledge about the particular visual quality.

Summary of Digital Painting and Brush Design Package

This project was developed in light of the following concerns: 1. allowing the user to easily generate a vast number of visual effects while painting with a digital brush pattern, 2. allowing the user to easily create and edit personal brush patterns, 3. providing the user the ability to control the change in size and opacity of a brush pattern during its application to the image, 4. implementing methods of applying brush marks to the image which are unique to the computer.

Software developed for creating brush patterns include rubberband rectangle, polygon, circle, and line routines, a freehand draw routine, and a routine which allows the user to select a group of shapes from any pattern on the work surface and to iterate the group a number of times onto the current pattern being edited. Rectangles, polygons, and circles can be either filled or hollow. For hollow shapes, the user selects the line width of the edges. The user also selects the line width for the freehand draw and the rubberband line routines. The user selects the color of a new shape from the color bar.

Software developed for modifying brush patterns include routines to move, copy, scale, and delete individual shapes or groups of shapes selected by the user. The user can also place shapes or groups of shapes in front of or behind other shapes. The user can also edit colors in the color table in order to change the color(s) of a pattern. Two methods of editing colors are provided. One allows a color's red, green, and blue values to be modified, and the other allows it's hue, lightness, and saturation values to be modified.

Software developed for generating images include the static and dynamic paint routines, and a routine which allows a selected rectangular area of the picture to be scaled and copied onto another portion of the image. While using either the static or dynamic paint routines, the user has the option to paint by iterating the brush pattern on top of all pattern marks on the picture, or to paint underneath certain colors that are in the picture. In dynamic painting, the user first specifies the limiting sizes that the brush pattern can grow or shrink to, and the change in transparency or opacity factor. While dynamically painting, the user has control as to whether the brush is shrinking, growing, staying the same size, becoming more transparent, becoming more opaque, or staying at the same opacity level while he/she is painting with it.

Software developed for the user interface consists of subroutines which set up a hierarchical menu of commands, read commands from that menu, set up and read items from a scrolling menu of mini-images of brush patterns, interact with the cursor icon and the mouse, set up and read color selections from a scrolling color table, display pop_up menus, and set up windows on the work surface in which brush patterns are created and edited in. Since the mouse is used a great deal by the user for communicating with the package, there is a mouse legend displayed on the screen continuously indicating the current functions of the three mouse buttons.

The User Interface

The design of the user interface was a major concern in this project. The user interface is the means in which the user communicates with the paint package. Since this system possesses only one display screen, menus are displayed in the same surface in which the picture is generated. The paint package consists of two modes of interaction, **EDIT** and **USE.** In order to use the limited space efficiently, the hierarchical command selection menu is displayed at the bottom edge of the screen, and displays only one level of the menu at a time. Immediately above this area is the color bar, which displays 20 of the 220 colors that the user can select or edit. This is a disadvantage to the user to only be able to see a small selection of the color palette, however it does use the area efficiently. Button arrows are provided at either side of the color bar to allow for scrolling to the left or right in order to display other colors in the palette. The library of patterns which the user has stored onto disk is displayed vertically at the rightmost edge of the screen. The images displayed in this menu are icons of the actual patterns. A pattern is scaled to fit into the rectangle slot representing it. Button arrows are provided at the top and bottom of this menu to allow for up and down scrolling in order to display other patterns in the library. The rest of the surface, which is a rectangular area functions as the work surface in the **EDIT** mode where up to 10 patterns can be created and edited simultaneously. In the **USE** mode, this area is the surface in which the image is generated from the iteration of patterns.

Except for commands which prompt the user to input the name of an image or color table, the user interacts with all other command operations solely with the optical mouse and its buttons. To provide for a more continuous flow of actions, the pattern and color editing functions, as well as functions which allow the user to create shapes, do not return the user to the command selection menu after the operation is complete, but allows him/her to continue applying the selected operation until he/she indicates otherwise.

The process of editing patterns and colors in the palette is simple and direct. Shapes behave as individual entities. They can be manipulated individually or as a selected group. If the user moves an item or group of items, any shapes that it covers are immediately revealed. Shapes are selected by pointing the cursor on them and clicking a button on the mouse. These are then the active shapes for the current operation. The user is also able to UNDO the current operation, and the pattern will be immediately displayed as it appeared prior to the operation.

A positive feature of the color bar is that whenever the cursor is placed in there for the user to make a color selection, he/she has the option to edit a color prior to its selection. For editing colors, two methods are provided to the user. One allows the user to modify the red, green, and blue color components of the color by adjusting sliders which represent each component. The other allows the user to modify the hue, lightness, and saturation levels of the color. A dial is provided for the hue adjustment, in which the user can continuously reposition the pointer on the dial to adjust the hue value. Sliders are provided for lightness and saturation adjustments. In both methods the scalars can be continuously adjusted until the user is satisfied with the current color that is displayed.

Though the package functions with a one screen system, the design of the display does not slow the application much. In fact, it is a positive feature in that the user focuses all his/her attention on one working environment. Using a multi-screen system would mean that the user would have to constantly shift his/her attention from one screen to another, which breaks up a smooth application of tasks. The disadvantage of a one screen system, however, is that the user can only see a limited set of items in the menus at a time.

Package Outline

I. Components of User Interface

A. Hierarchical Command Selection Menu

B. Scrolling Brush Image Menu

C. Brush Editing Windows

D. Work Surface
1. edit mode - area where editing windows exist
2. paint mode - area where image is painted

E. Scrolling Color Table

F. Pop Up Menus

G. Cursor Routines

H. Mouse Legend

II. Components of Brush Design Mode

A. Window Management
1. New Brush - select or create a window to design a brush in.
2. Copy Brush - make a copy of a window and its enclosed image.
3. Move Brush - move a window.
4. Clear Brush - erase a window's enclosed image.
5. Delete Brush - delete a window and its image from the work surface.
6. Test Brush - create a temporary window to test the use of a brush pattern.

B. Creating Shapes
   1. Freehand draw
   2. Box
   3. Polygon
   4. Circle
   5. Line
   6. Copy Shapes - select a shape or group of shapes from a window (another or current brush editing window) and repeat the pattern in the current editing window.

C. Color Table Management
   1. Edit Colors - edit color items in the color table.
   2. Blend Colors - interpolate between two colors in the color table.
   3. Load Colors - load a color table which has been saved on disk.
   4. Save Colors - save the current color table onto disk.

D. Shape/Object Manipulation
   1. Move Item/Group
   2. Copy Item/Group
   3. Scale Item/Group
   4. Delete Items

E. Brush Images on Disk
   1. Save Brush - save a brush image onto disk.
   2. Erase Brush - erase a brush image from disk.

III. Components of Paint Mode

A. The Image
   1. Paint - repeat a brush pattern onto the picture surface
   2. Dynamic Paint - repeat a brush image that can grow, shrink, become more transparent, and become more opaque onto the picture surface.

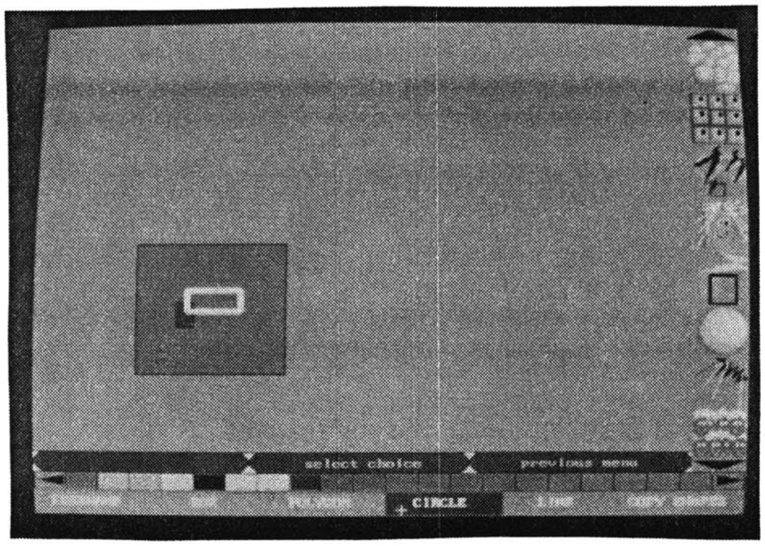
3. Clear Pic - clear the picture surface to a selected color.
4. Save Pic - save the picture onto disk.
5. Load Pic - load a picture from disk.

B. Color Management
1. Fill- fill a specified region of the image with a color selected from the color table.
2. Edit Colors
3. Blend Colors
4. Load Colors
5. Save colors

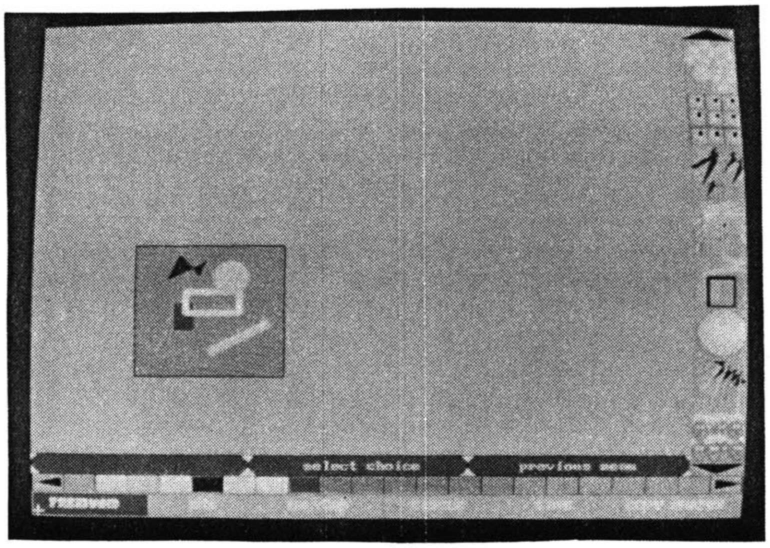
Scenario of Pattern Editing Functions

Creating Shapes:

filled box and hollow box:

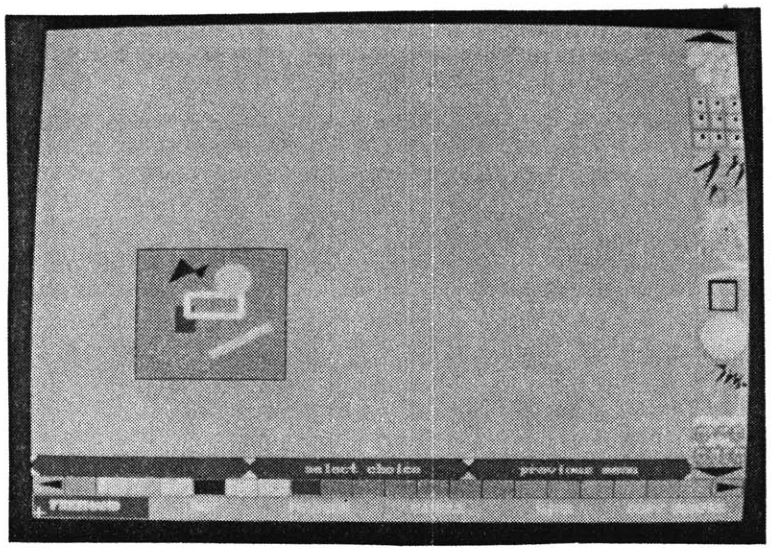


some more shapes:

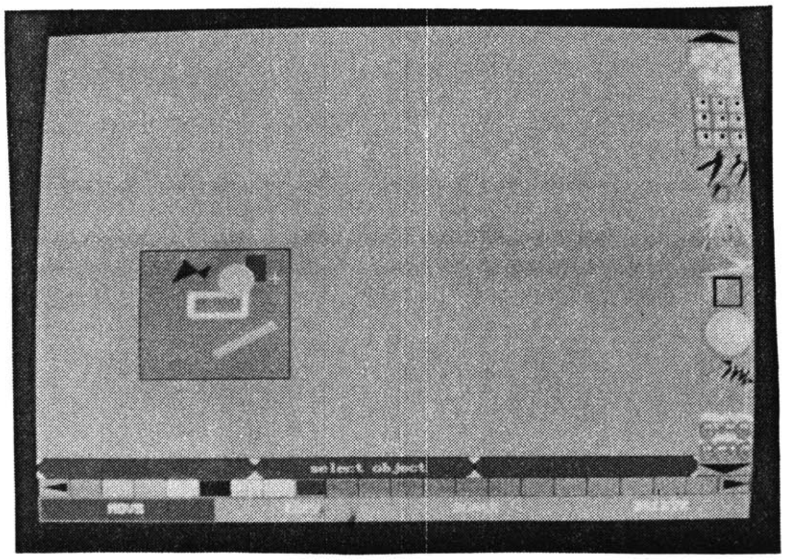


Moving an Individual Item:

before:

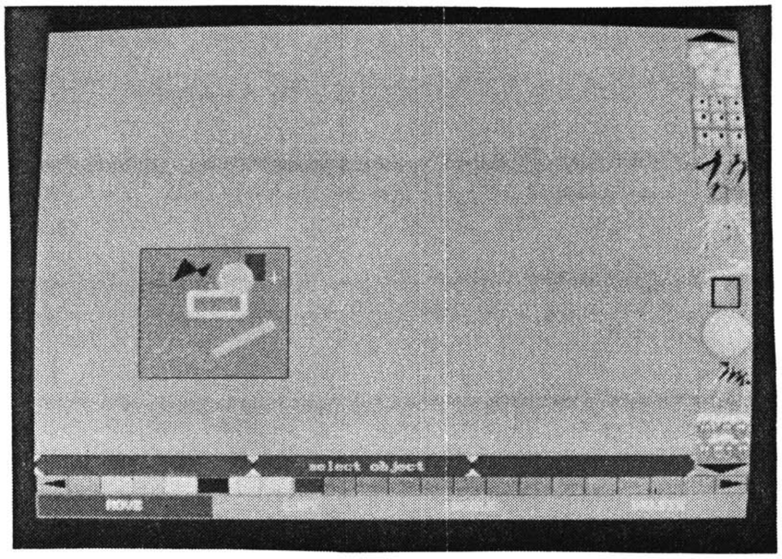


after:

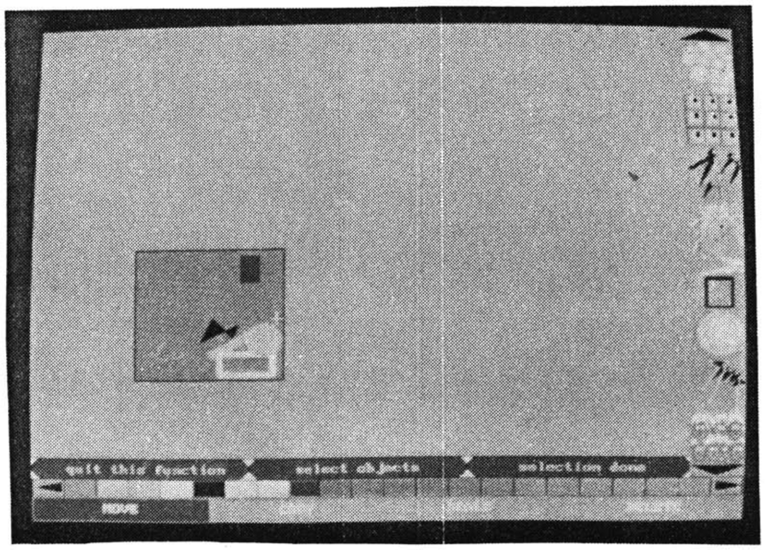


Moving a Group of Items:

before:

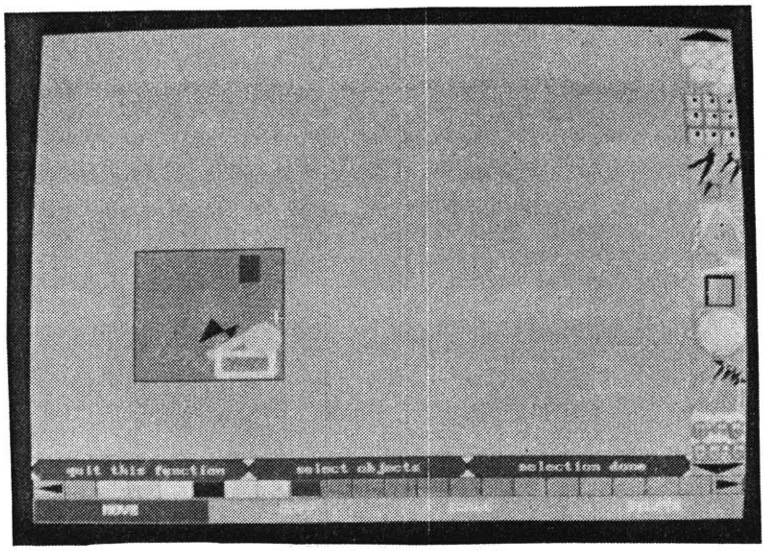


after:

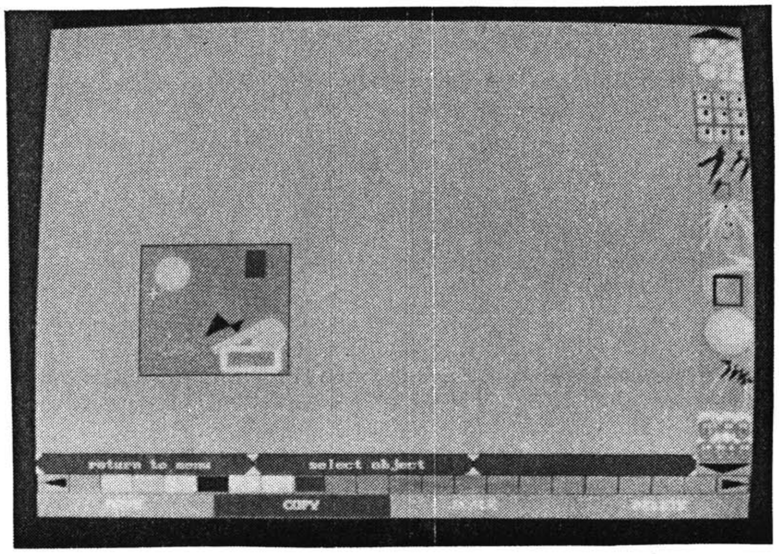


Copying an Individual Item:

before:

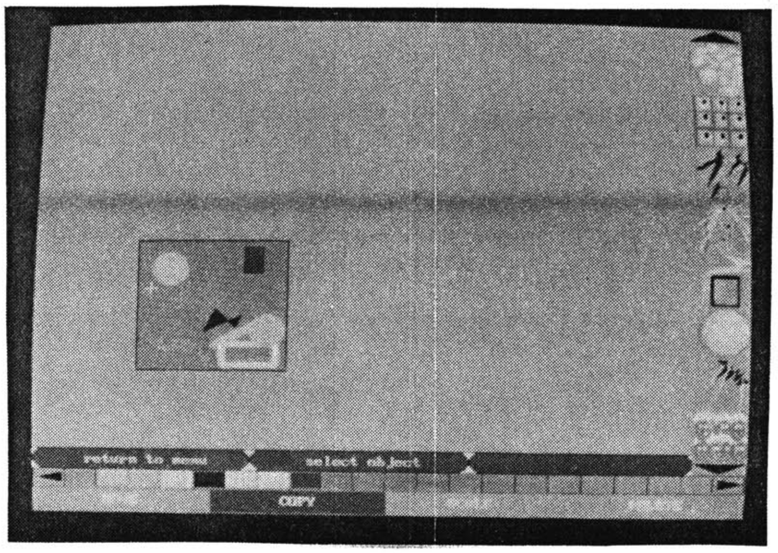


after:

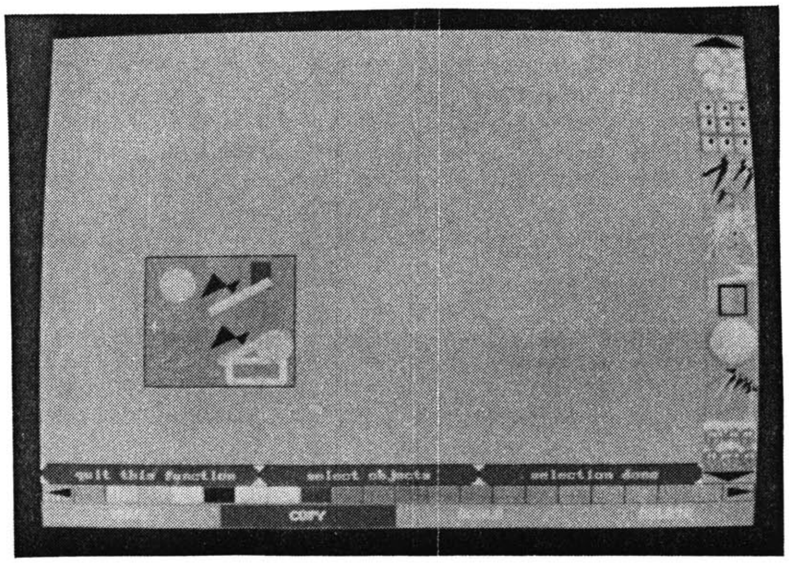


Copying a Group of Items:

before:

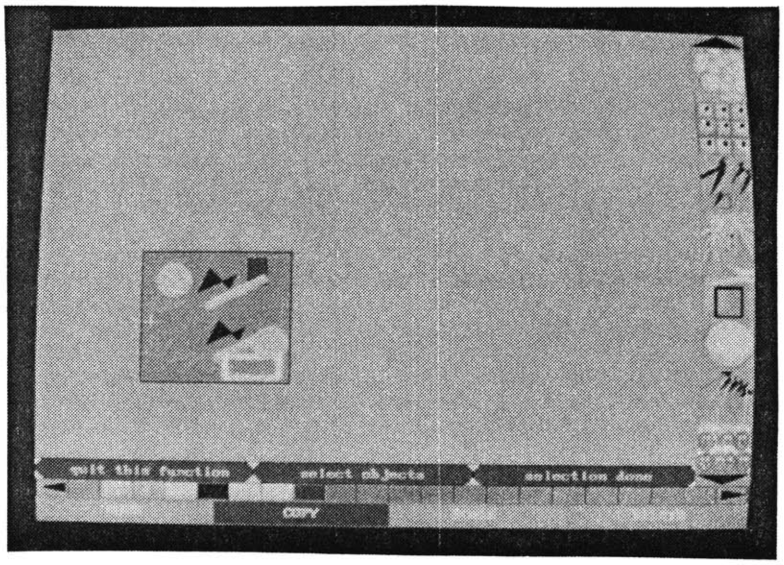


after:

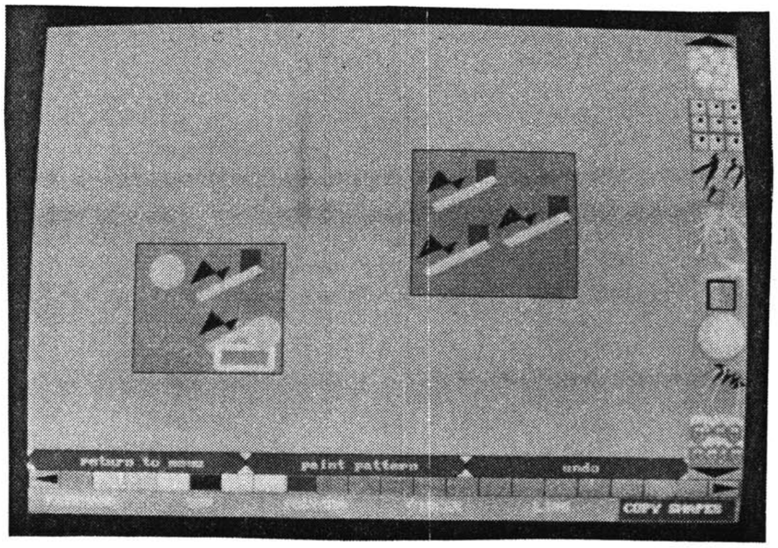


Copy Group into Another Window:

before:

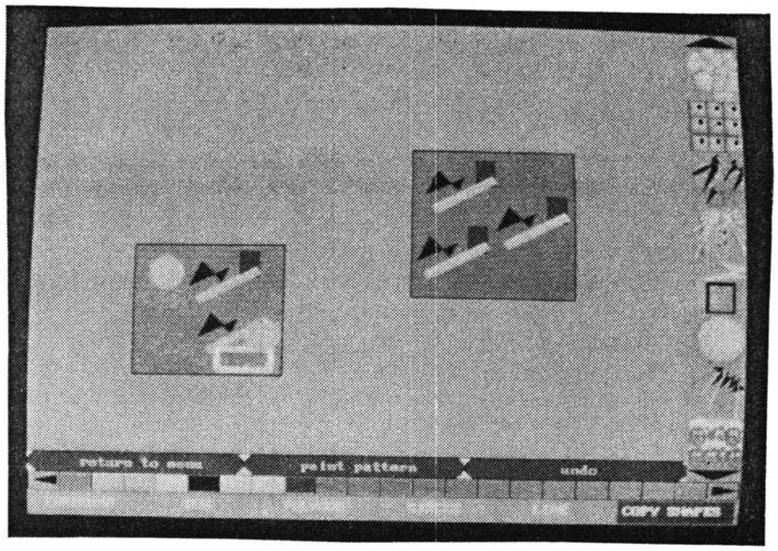


after:

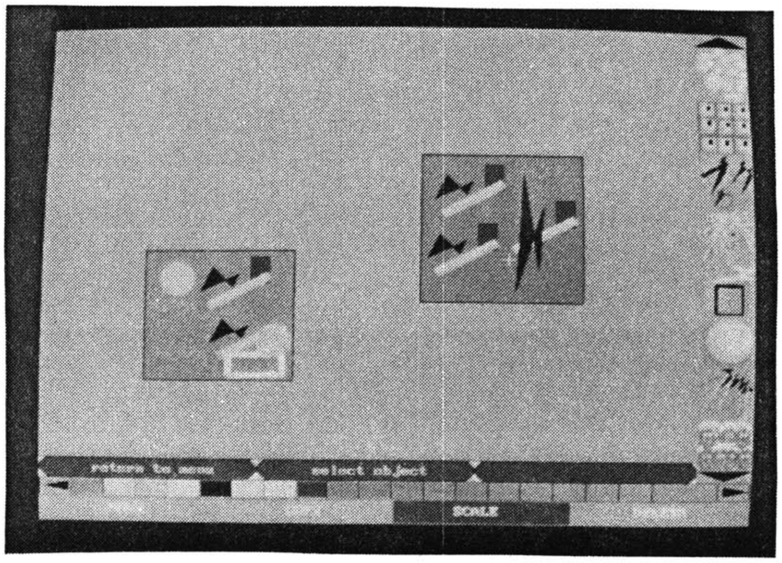


Scaling an Item:

before:

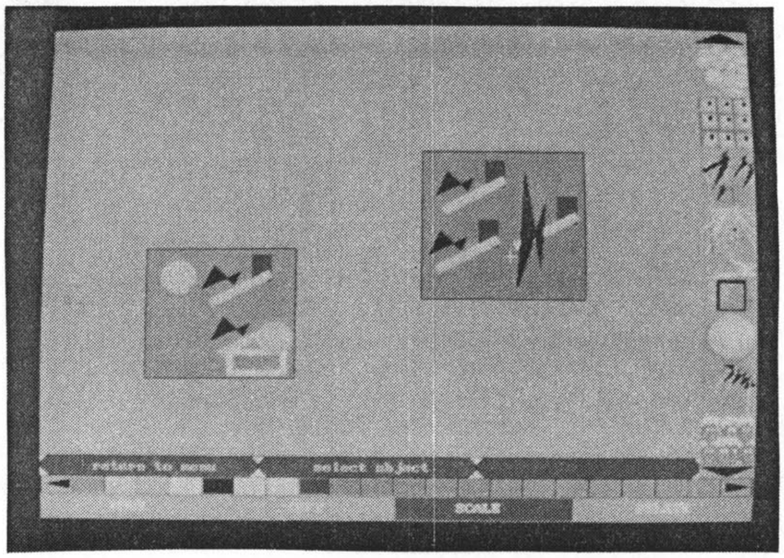


after:

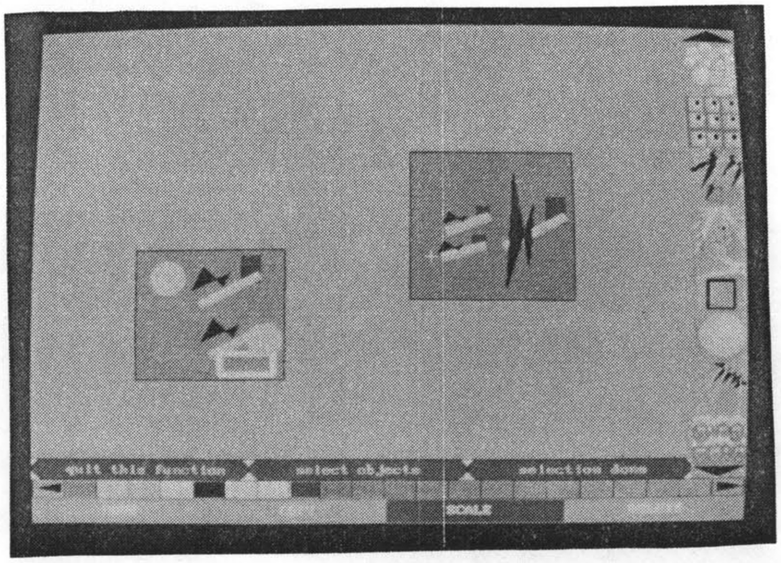


Scaling a Group of Items:

before:

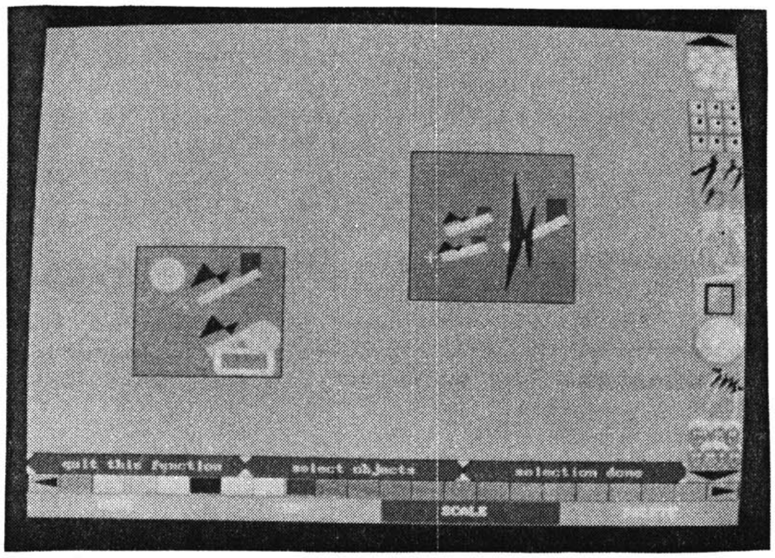


after:

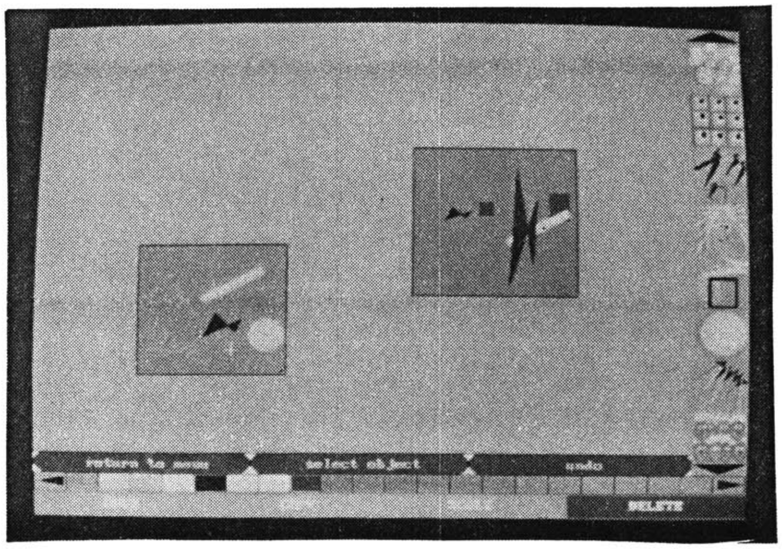


Deleting Items:

before:

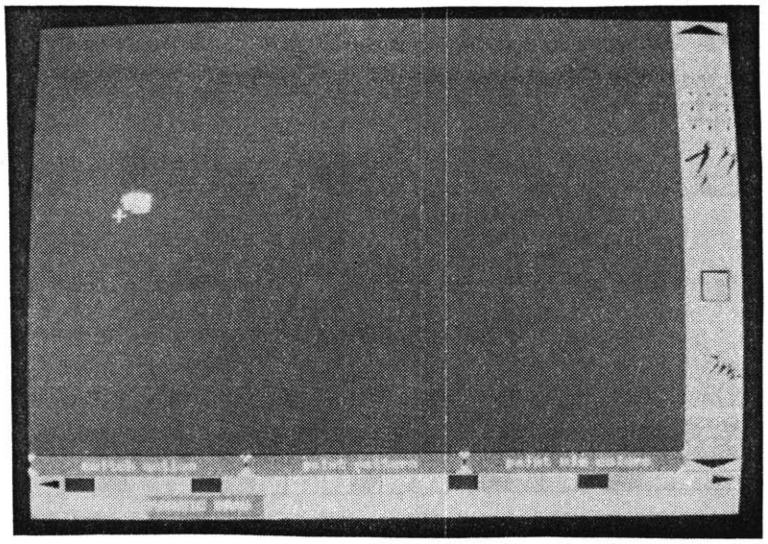


after:

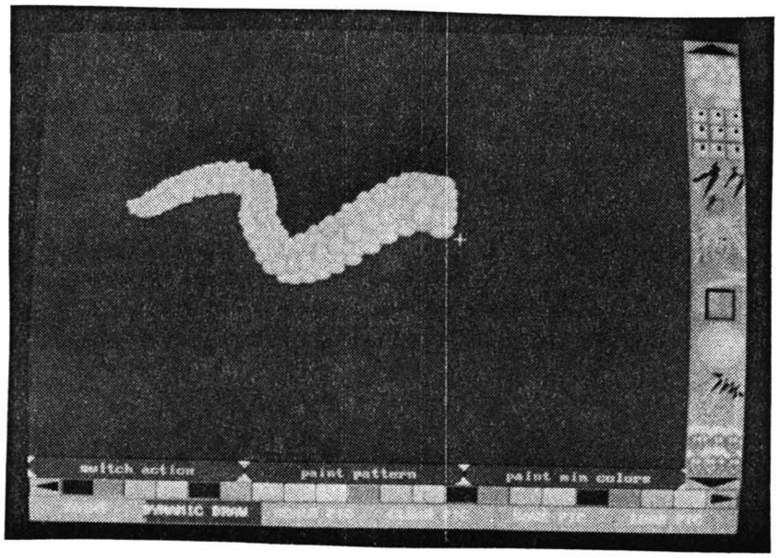


Scenario of Dynamic Painting

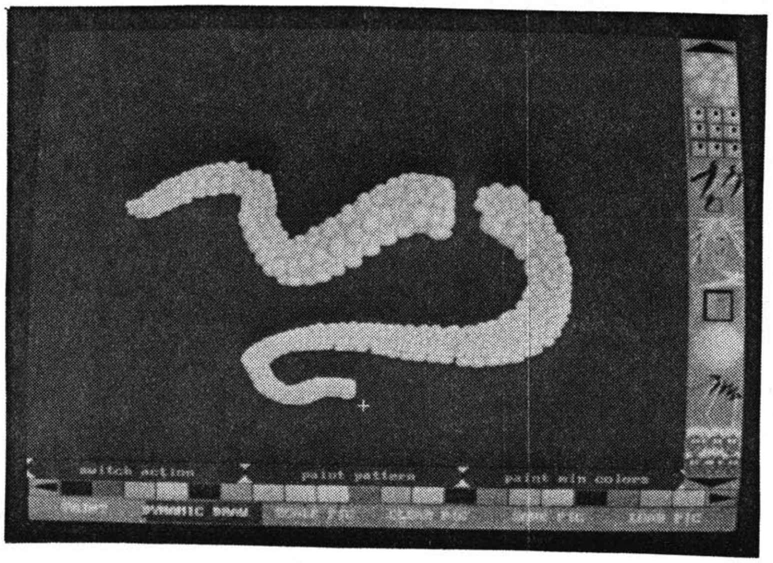
Initial Marks:



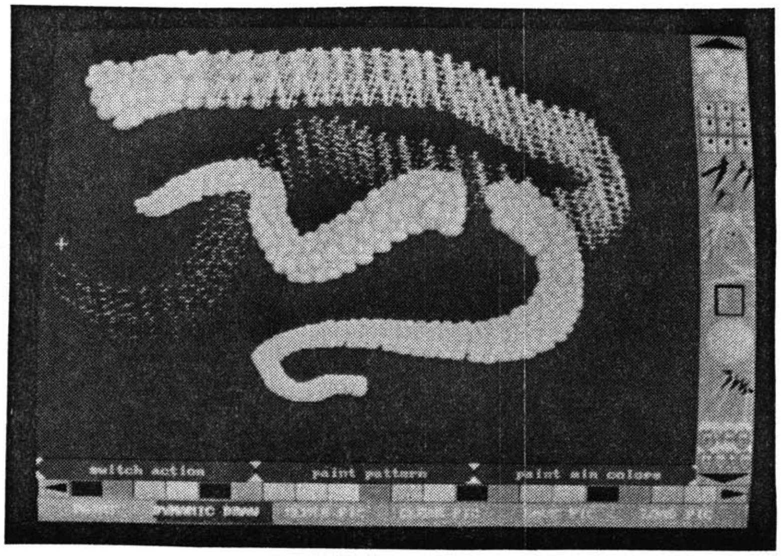
Brush Pattern Growing:



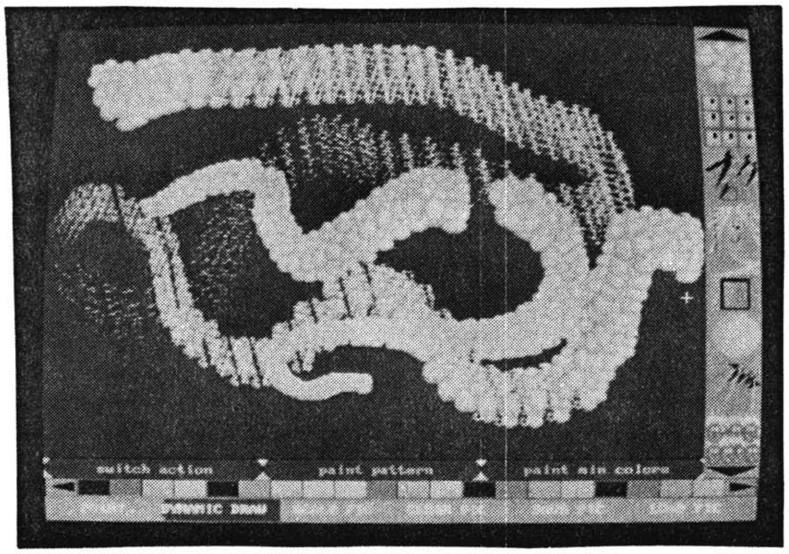
Brush Shrinking:



Brush Staying the Same Size and Becoming More Transparent:



Brush Growing and Becoming More Opaque:



Chapter 2: Comparison to Related Methods of Visual Representation

Introduction

Methods for rendering objects of nature range from the experimental to the rigid practices. This can be seen in both the traditional media of painting and the electronic medium of computer graphics. A scientific exploration into the visual attributes of nature has been a contributing factor in the development of methods by various artists. During the Renaissance period, Leonardo DaVinci, for example, had made many scientific studies in attempts to understand nature, and the outcome of his studies can be seen in many sketches and works of art that he produced. The Impressionist painters of the late 19th century rejected conventional painting procedures of their time and explored new methods of effectively rendering reality from their own direct observation of nature. The research of optics by scientists at that time had a great influence on their painting methodology. A painter of that period, John Constable, made the following remark:

> "Painting is a science, and should be pursued as an enquiry into the laws of nature. Why, then, may not landscape painting be considered as a branch of natural philosophy, of which pictures are but the experiments?"[1]

In the medium of computer graphics, research into the analytic description of nature has directly affected the formulation of algorithms, which instruct the generation of images by the computer. In raster images, ray tracing algorithms, which calculate the color value of each pixel in the object being rendered from the directions and intensities of specified light sources and the surface quality of the object is comparable to the methodologies of the Impressionist and the Neo-Impressionist painters, who considered the interplay of light and colors to be the unifying elements of a picture. Another class of algorithms for generating images with computers have evolved from a mathematical analysis of nature known as fractal geometry. Fractal geometry describes aspects of nature as being self-similar, that is, varying details of a phenomenon are of different scales, but are all geometrically similar to the whole.

With respect to following rigid methods, which do not place the artist in a position of scientific enquiry, the style of Chinese painting is a prime example of rendering images with predefined brush strokes. In the schools of Chinese painting, the objective is for the artist to convey the essence of an object in a minimum of brush strokes. Before attempting to render an object of nature, the painter must first master the brush strokes for the painting of the object, which were developed and standardized by

Chinese painters in the past.

The methodologies described here share a common strategy. Though the styles and images rendered by each differ procedurally and visually, each places primary concern on rendering the characteristics of the fundamental parts as a means to generating an image. In Chinese painting and fractal-based rendering of images, as well as the technique explored in this thesis, it is the repetition of similar patterns or "strokes" at various scales which suggests the quality of the final visual phenomenon.

Chinese Painting

The style of Chinese painting through its history has placed significance on the materials and methodologies employed for the rendering of images. The materials used, brush, ink, inkstone, and paper, are refered to as the four treasures. Brushes, which are simple in appearance, are flexible for the marking of a variety of brush strokes, however they are very difficult to use. Brushes come in a variety of shapes, sizes, and pliability, and are constructed from a variety of animal hairs, such as wolf, goat, and rabbit hairs. These characteristics influence the quality of brush strokes that a brush can help produce.

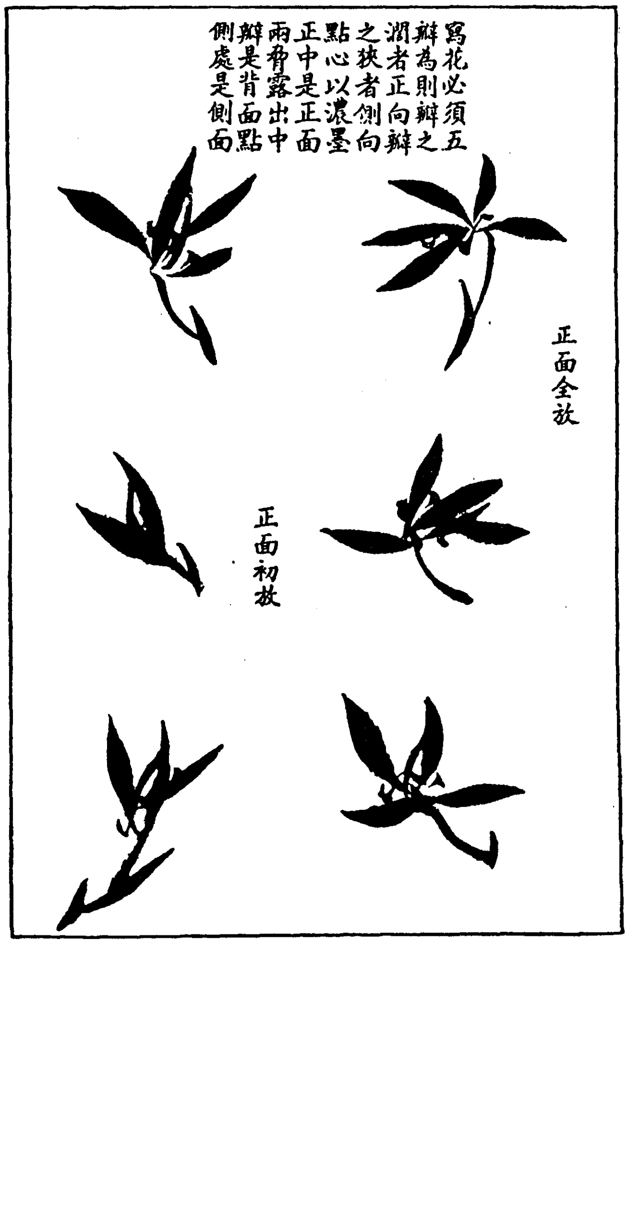
Vocabularies of brush strokes had been developed and standardized by painters in various schools of Chinese painting. Painters had to master the rendering of particular brush strokes before they could attempt to paint a picture. The brush strokes used in Chinese painting are similar to the strokes used in Chinese calligraphy. There is a legend that the origin of Chinese painting and calligraphy was not created by man, but originated in the language of Heaven, and the brush strokes, which were abstractions of the language, were brought to man by a divine sea-tortoise. Calligraphy used the abstract lines to develop the written form of Chinese language, while painting transformed these lines to represent visual essences of nature. In brief, brush strokes used in painting described the following marks: spread out hemp fibers, entangled hemp fibers, sesame seeds, big ax cuts, small ax cuts, closed heads or thunderheads, raindrops, eddy or whirlpool, veins of a lotus leaf, lumps of alum, skull bones, wrinkles on a devil's face, raveled rope, brushwood, hair of cattle, horse's teeth.[2]

In Chinese painting, the characteristics of brushes and ink do not allow for artists to correct, change, or trace over a stroke without damaging effects to the image. This means that manual coordination of the brush and mastering of strokes is essential.

Here is an example of rules for applying brush strokes to render an orchid:

> "In painting the orchid the first stroke is (the arc-shaped stroke) called *pieh* The brush should be handled through the wrist with agility. Brushstrokes should not be equal in length. When the leaves grow in bunches, crisscrossing, bending, and drooping, they should have *shih* (style and structural integration). Bending over or facing up, each has a special aspect; distinction, moreover, should be made between forms in the foreground and those in the background. There should be a variety of ink tones. Flowers and more leaves should be added, also the sheath that covers the base. The flowers should first be drawn in light ink; soft and pliant, they are supported by their stems. Differences between the inside and outside of petals should be shown, each form being delicate. The stem should be wrapped in fine young leaf. Flowers gain distinction when their stamens are dotted with dark ink. In full bloom a flower stands erect with face upwards. The mood is that of a fair and happy day. When flowers are painted in the breeze, they should seem to be weighted with dew. Buds are closed as though they were firmly holding fragrance. The five petals of the flower should not be arranged like fingers of an outstretched hand. They should be like fingers, though with one or two curled and one or two straight. The stem of the marsh orchid should be strong and upright, its leaf strong and vibrant, the leaves spread out on all sides more than the ordinary orchid. Flowers hang from tips of the stems; their subtle fragrance can be conveyed by the movement of the wrist. Through brush and ink it is possible to transmit their essence."[3]

(see **figures 7, 8, and 9** for a process of rendering the orchid)[4]



寫花必須五瓣為則瓣之潤者正向瓣之狹者側向點心以濃墨正中是正面兩脅露出中瓣是背面點側處是側面

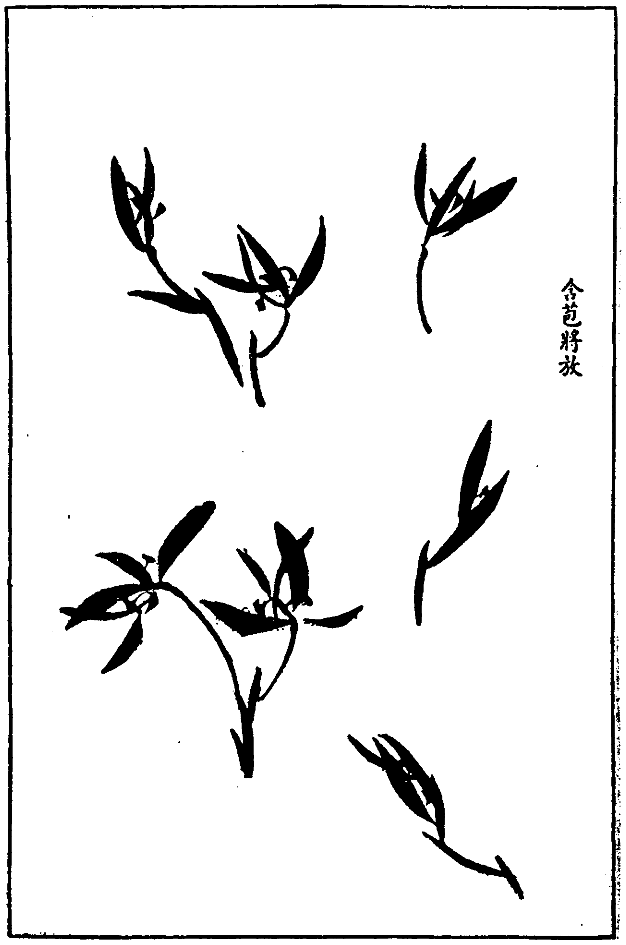
20 In the drawing of flowers, each should have five petals. The larger petals are straight and broad, the smaller and narrower petals curl. Use dark tones of ink to dot the stamens. When the stamens are in the center among the petals, the flower is facing front. When shown on either side of the middle petal, as though in its armpits, the flower is being viewed from the back. When the stamens are dotted in on the side, the flower is being seen from the side.

Left column:
Flowers front view and in bud.

Right column:
Flowers front view.

figure 7

21

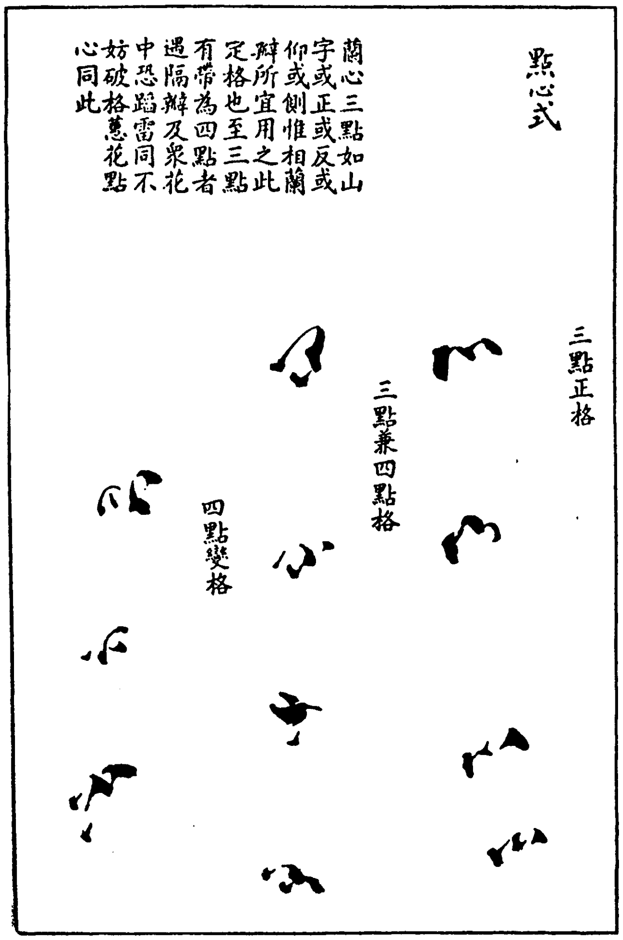


Buds beginning to open.

(Flowers at the same stage.) [1]

1. As in original edition.

figure 8



點心式

蘭心三點如山字或正或反或仰或側惟相蘭瓣所宜用之此定格也至三點有帶為四點者遇隔瓣及衆花中恐蹈雷同不妨破格蕙花點心同此

22 **Examples of dotting the heart**

The form of the three dots (of the stamens) in the heart of the orchid is like 山 (*shan*). Whether the flower is straight, turned over, leaning to one side, or facing upward, the dotting of the stamens in these various positions depends on the position of the petals. This is a fixed principle. In addition to the three dots of stamens, a fourth is sometimes added, because petals are often mingled and, among a group of flowers, one should avoid being repetitious. This does not violate the rules. The same kind of dotting is used for the marsh orchid.

Right column:
Correct forms of the three dots.

Mid column:
Examples of the three dots with a fourth added.

Left column:
More examples of four dots.

figure 9

In contrast to the techniques of Chinese painting, which contains a vocabulary of brush strokes for the artist to master, the methodology developed in this thesis requires the user to develop his/her own selection of brush patterns to render images with. In addition, no manual gesture is involved for manipulating the pattern's application with this method. The size change and amount of pigment applied for sequences of the pattern is consciously controlled by the user through button interactions of the mouse input device. In Chinese painting, the artist determines the particular brushstroke to render by selecting an appropriate type of brush, and by controlling the motion and pressure of the hand and wrist. In the methodology developed in my system, the artist determines the visual characteristics of the "brush" pattern prior to its use. Here the pattern is a pre-made sub-image of the whole to be generated through iteration of the pattern.

Other Modular Methods of Generating Images

The building of a visual phenomena from parts which are geometrically similar to the whole can be demonstrated by artists of traditional media, as well as by artists employing algorithms based on fractal geometry in computer graphics systems.

Leonardo DaVinci in the 15th century had made countless studies on the motion of water and air. DaVinci, in addition to his artistic pursuit, was also an engineer, physicist, and cosmologist. With respect to his studies on water, he produced a vast amount of illustrations, some of which possess an aesthetic quality of the typical sense, and others which appear more like scientific diagrams. He had studied the impact of water falling on water and this is illustrated in some of his drawings. Examining some of these pictures, we can sense parts which are similar to the whole. In his picture, Impact of Water on Water (1507-9), shown in **figure 10**[5], we can notice the suggestion of swirling whirlpools and eddies within the whole of the swirling pool which is being agitated by a cascade. The swirling lines suggesting the whirlpools and eddies appear similar in form to each other and to the whole of the phenomenon being illustrated.

In computer graphics, the application of fractal geometry to algorithms which instruct the rendering of visual phenomena produce images whose parts, at any detail of observation is geometrically similar to each other and to the whole. A position of research in fractal geometry is the study of phenomena whose dimensions cannot be adequately measured by Euclidean geometry, but can be determined by the rules of fractal geometry. Algorithms based on these measurements can produce striking textures which suggest realistic qualities. The regularity of texture production can be controlled, and for the images representing phenomena, which are generated by algorithms with stochastic variables, the random application of their similar parts in conjunction with the formulas which describe them, demonstrates visually that these methods are adequate for rendering the irregular appearances of nature. (See **figure 11**)[6].

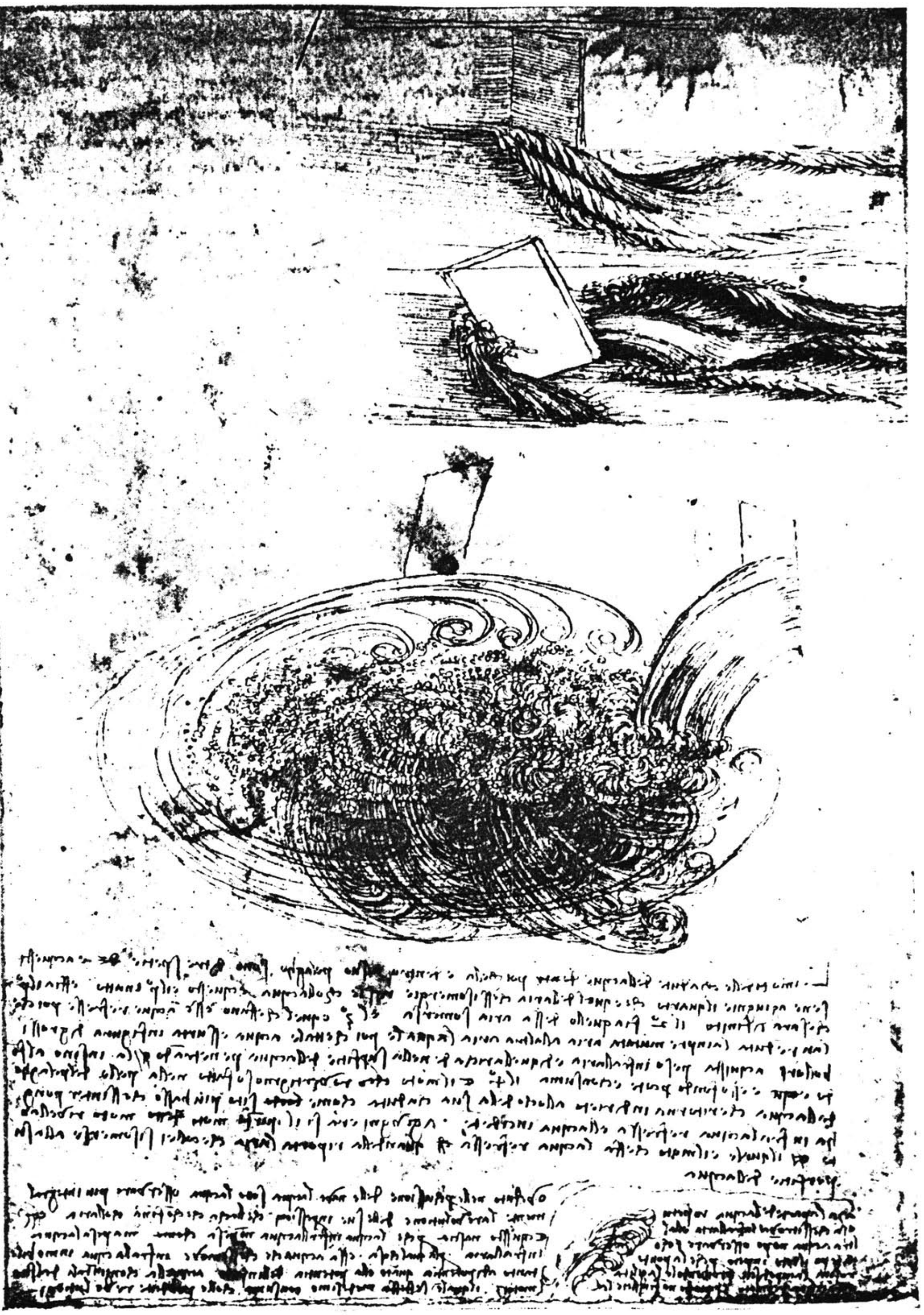


figure 10

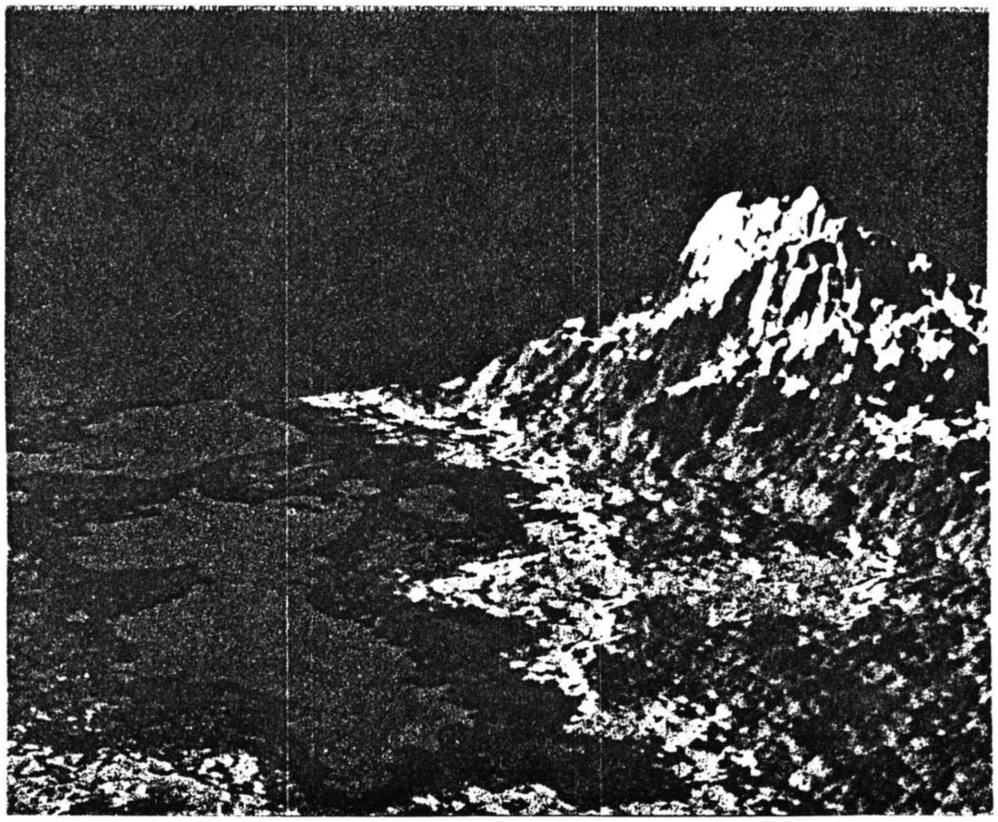


figure 11

The methodology developed in this thesis has overlapping traits with the stochastic modeling of texture. In pictures generated using these tools, patterns are repeated and concatenated at various sizes and pigment saturation levels to suggest an overall texture. In the images described by fractal-based algorithms, they also contain a repetition of patterns at various scales, however, the characteristics of the patterns are similar to the texture of the whole and they are constructed on patterns which are also similar to them and the whole, and this can be observed to the smallest detail of texture which can be rendered in the image. In the methodology described in this thesis, it is not the computer which determines the position of constituent marks, but the user by positioning the pattern at each application of it.

An interesting and possibly enlightening methodology that should be researched in the future is a synthesis of the technique described in this thesis with fractal-based algorithms that describe visual characteristics. One possibility is to allow the user to edit patterns by adjusting parameters of a function that describes forms in terms of fractal geometry. Then the form of the pattern, in addition to being repeated on the image surface, can contribute to the overall form of the visual structure being built. Here, the user indicates the locations where the pattern is to be repeated, however the computer manipulates the overall form based on the parameters selected for the fractal function.

Chapter 3: Program Design and Implementation

Introduction

The digital painting and brush pattern design package is a menu-driven system. The software developed can be divided into supporting six major components of the system. These are the user interface of the package, the creation and direct manipulation of shapes by the user, the production of an image by the user, color table management and editing of individual color items, storage and retrieval of image and color table data to and from disk, and miscellaneous utility functions.

The main subroutine which drives the package first calls on initialization routines which set up the graphics display screen, the initial default color table, the hierarchical command menu, menu of patterns previously saved on disk, mouse legend display, and the speed and boundary parameters of the mouse. Then its sole purpose is to continuously read selections that the user makes from the hierarchical command menu and to call on the appropriate subroutine to execute the selected command.

This chapter describes the design and implementation of significant software pertinent to the main concerns of this thesis. Software concerned with manipulation of individual shapes as well as a group of shapes, and software concerned with dynamic iteration of patterns will be the main focus of discussion. A high resolution display processing unit with an invisible as well as a visible frame buffer and operations for bit-boundary block transfer of pixel values are requisites for the implementation of the routines described here.

System Environment

This package was developed on an IBM XT personal computer with 512 k of core memory. The display unit includes a high resolution red, green, blue color monitor, experimental graphics board which supports an 8-bit frame buffer with a visible area of 640 x 480 pixels and an invisible area of 640 x 336 pixels, and graphics support routines. Up to 256 different colors can be displayed at a time from a set of 16,777,216 possible colors. The value of a pixel ranges from 0 to 255 and is an index to the color lookup table of 256 items. Each item in the color lookup table is composed of red, green, and blue values, each of which are represented by eight bits of memory. The primary input device is an optical mouse with three buttons. This package was written in Lattice C on the DOS version 2.0 operating system.

Programming Support Routines

These routines were provided with the graphics system and are written in microcode. The operations are entirely performed in the image domain on areas refered to as pixel maps. These are rectangular volumes defined by the programmer in the image space. The image space includes the visible and invisible areas of the frame buffer and also areas of host memory which the programmer may allocate to hold additional image data. Each pixel map in image space is defined by the x,y, and z coordinates of its origin in that space, and the extent of its width, height, and depth. The depth of the pixel map (0 - 7 bit planes) determines the range of values that can be assigned to each pixel in the map. Each pixel value indexes to a color item in the color lookup table. This package uses pixel maps which are all defined to be 8-bits deep in order to allow pixels to index any of the 256 color items in the color lookup table.

Defining pixel maps and applying bit-boundary block transfer (BitBlt) operations to them has been a very effective and powerful technique throughout all the routines which comprise this package. BitBlt operations perform on each pixel in the source and destination pixel maps, replacing the destination pixel with the value that results from it and the source pixel in its corresponding relative position.

The invisible area of the frame buffer which is 640 x 336 pixels has proven to be a useful area for temporary storage of sub-images, and intermediate graphics operations. Pop up menus, cursor routines, painting routines, and routines which allow the user to directly manipulate shape objects are some of the major routines which could not have been smoothly implemented without this region of the frame buffer.

The following list describes programming support routines which were provided:

- set up the initial pixel maps which describe the visible and invisible areas of the frame buffer.
- clear pixels of a given pixel map to a single pixel value.
- define a pixel map in host memory.
- define a pixel map inside an existing pixel map.
- load a new or modified color lookup table.
- calculate the red, green, and blue components of a color which has been described in hue, lightness, and saturation values.
- calcuate the hue, lightness, and saturation components of a color which has been described in red, green, and blue values.
- set a pixel value to be the current transparency pixel value.
- BitBlt, copying the source pixel map to the destination pixel map.
- BitBlt, replacing only the pixels in the destination pixel map whose values are higher than their corresponding pixels in the source pixel map.
- BitBlt, copying the source pixel values excluding the transparency pixel value into the destination area.
- read a pixel value from a given pixel map.
- assign a value to a pixel in a given pixel map.
- draw a line of a specified color and width between two

given endpoints in a given pixel map.

- draw a series of lines of a specified color between given vertices in a given pixel map.
- display a solid polygon of a specified color given a set of vertices in a given pixel map.
- draw an unfilled circle of a specified color in a given pixel map.
- draw a filled circle of a specified color in a given pixel map.
- write a string of characters of a given color into a given pixel map.

Description of Significant Software

Reserved Colors in the Color Table:

220 colors are provided to the user of the package to use and edit. The remaining colors in the color table are reserved for use in various parts of the package. Color items indexed 5, 6, and 7 in the color table are reserved for displaying red, green, and blue respectively. Color item 1 is reserved for displaying the background color of the package display. Color item 2 is reserved for displaying the boundary color of the command selection menu and the brush pattern menu. Color item 3 is reserved for text color. Color item 4 is reserved for the highlight color of slots in the command selection menu. Color

item 8 is the current erase color. Color item 252 displays grey. Color item 255 displays black. Color item 251 displays white. Color item 253 is the current rubberband color for the routines which allow the user to create shapes interactively. It displays either white or black, depending on the color of the background. Color items 240 to 249 are reserved for the background/transparency color values of patterns that the user generates.

Manipulation of Individual Shapes and Groups of Shapes:

These subroutines are divided into five parts:

1. subroutines which record descriptive values into two structures for each shape created.
2. subroutines which evaluate information in the data structures in order to redraw a shape at a specified location.
3. subroutines associated with a user-selected shape, or user-selected group.
4. subroutines associated with repositioning a shape or group of shapes.
5. subroutines which execute the shape manipulation functions: move, copy, scale, delete.

Data Structures:

The following type of record is created for each new shape produced by the interactive functions which generate freehand, rectangle polygon, circle, and line elements. Upon the creation of each, values are recorded in this type of record in order to describe the shape. Coordinate values are recorded to lie within the local coordinate space of each shape. This method of description allows for efficient representation of a shape which may be displayed in several locations in various sizes in the actual space of the image. Here, the attributes of the shape is described in only one record, yet can be addressed by any number of items from a linked list data structure, which describes its actual location relative to the image space (this data structure is described later). From calculating the amount of memory needed for a data structure of this type and of the type which addresses it, one can easily discern that this method is more memory efficient than a list of structures which define each shape, without concern that some are duplicates or scaled versions of other shapes in the image.

```
struct objects {
        int shape_type;
        int width;
        int height;
        int number_of_times_being_displayed;
        union {
                struct {
                        int number_of_vertices;
                        int vertices[20][2];  /* 20 vertices max */
                        int color;
                        int width_of_edges_if_hollow;
                ) polygon;

                struct {
                        int lower_x;
                        int lower_y;
                        int upper_x;
                        int upper_y;
                        int color;
                        int width_of_edges_if_hollow;
                } box_or_line;

                struct {
                        int color;
                        int center_x;
                        int center_y;
                        int radius;
                } circle;

                struct {
                        int transparency/background color;
                        int name[10];
                } freehand;
        } type;
};
```

The following data structure is a symmetrically linked list of records which describe the actual location of a shape relative to the image space (pixel map containing the particular pattern). Each pattern image is represented by an independent linked list of this type. Each record in the list also contains a parameter which describes the scale type of the shape being addressed, a pointer to the address of the object record (described above), a pointer to the preceding record in the list, and a pointer to the succeeding record in the list. If a record has either no preceding element or no succeeding element in the list, the respective pointer is set to the NULL value. A record is created or updated when a shape has been either newly positioned, repositioned, copied, or scaled. A record is removed from the list and the list reconnected when a shape is deleted from the image space. By re-addressing pointers, appropriate records in the list are reorganized in the instance that a shape is repositioned either behind or in front of a designated shape. Though it is not always visually perceptible, such as when the locations of shapes in the x-y plane do not intersect, the records in the list are sorted so that each succeeding record describes a shape which lies on top of the one described by the preceding record.

```
struct attach {
        int lower_x_in_image_space;
        int lower_y_in_image_space;
        int upper_x_in_image_space;
        int upper_y_in_image_space;
        int scale_type;
        struct attach *previous_record;
        struct objects *shape_pointer;
        struct attach *next_record;
    };
```

The following are each pattern's pointers to the first item and last item of the symmetrically linked list associated with it. These are used to access the linked list from either end.

```
    struct attach *first_item_pointer [NUMBER_OF_PATTERNS];
    struct attach *last_item_pointer [NUMBER_OF_PATTERNS];
```

The following are arrays of pointers which address particular items in a pattern's linked list. These arrays, together, manage shapes which have been grouped together for an operation.

```
    struct attach *mark_the_items [20];
    struct attach *group_the_items [20];
```

Scenario of Flow of Control:

sample pattern:

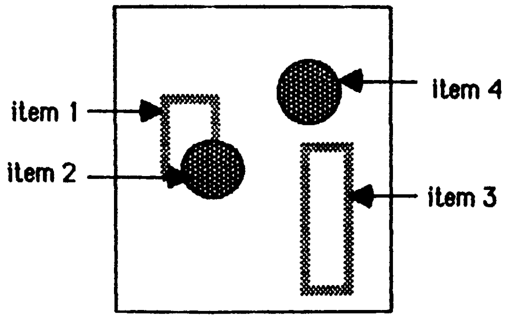
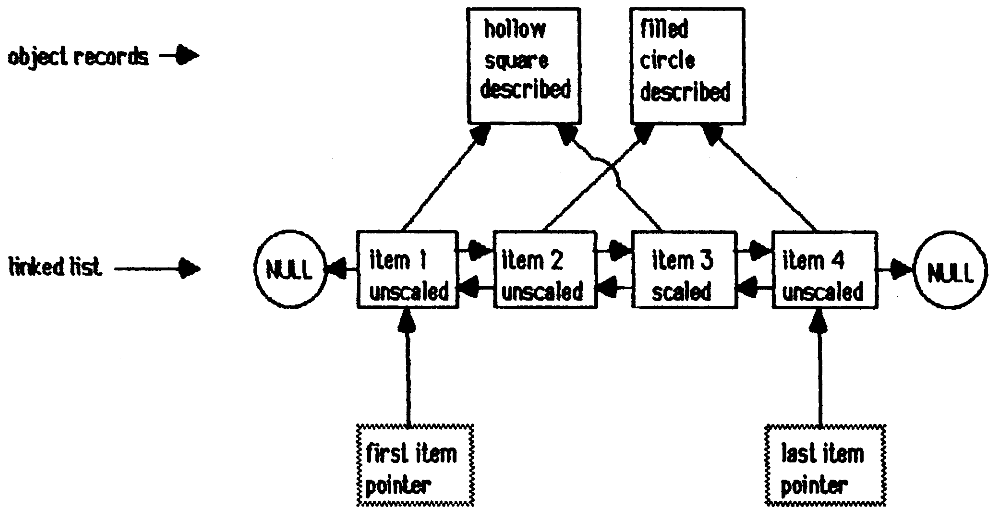
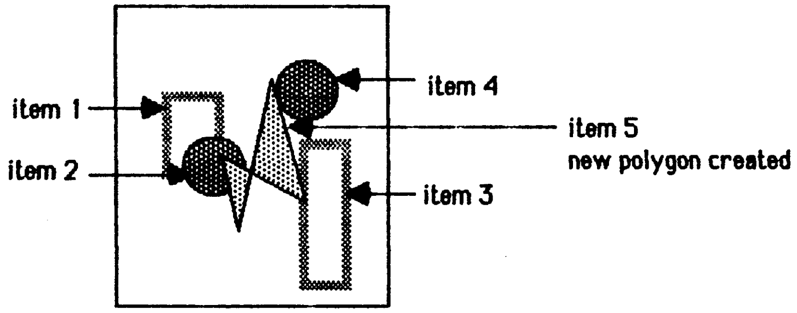


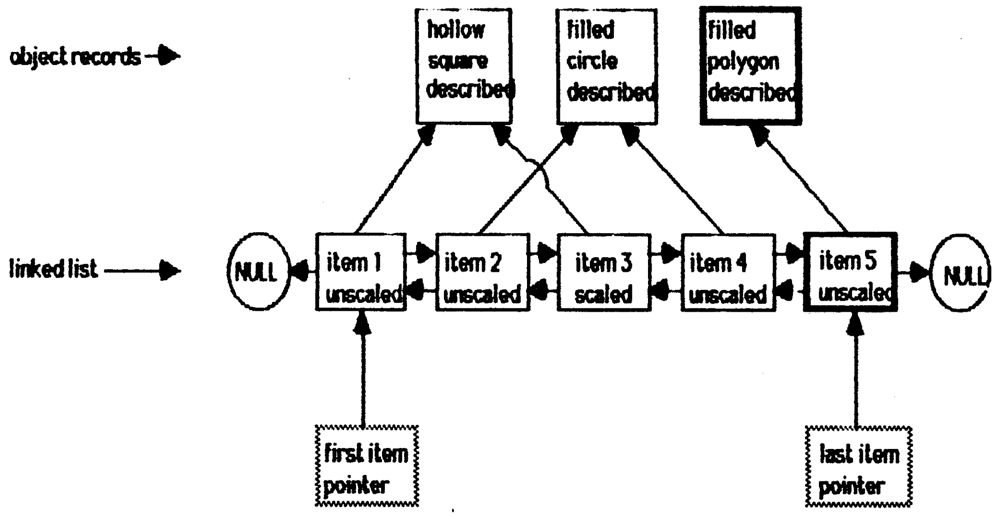
diagram of associated data structures:



1. New polygon created



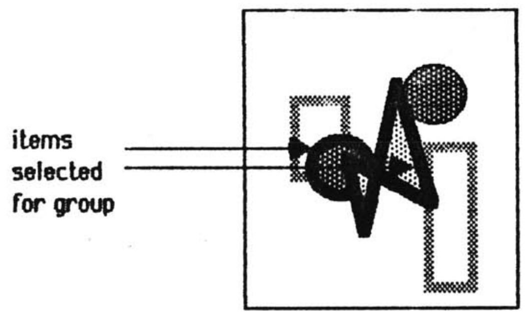
Call **new_polygon** routine. Pass it the number of vertices, coordinates of vertices, color, type of polygon (filled or hollow), and width of edges if it is hollow. **New_polygon** creates a new record for the shape and sets appropriate values in the record. This routine calls **new_shape_attached** which adds a new record to the end of the pattern's linked list. **New_shape_attached** also assigns to the record the lower x, lower y, upper x, and upper y coordinates of the area that the shape occupies, a value indicating that the new shape is not scaled, and the proper addresses of the pointers. It updates **last_item_pointer** to address this new record in the list.

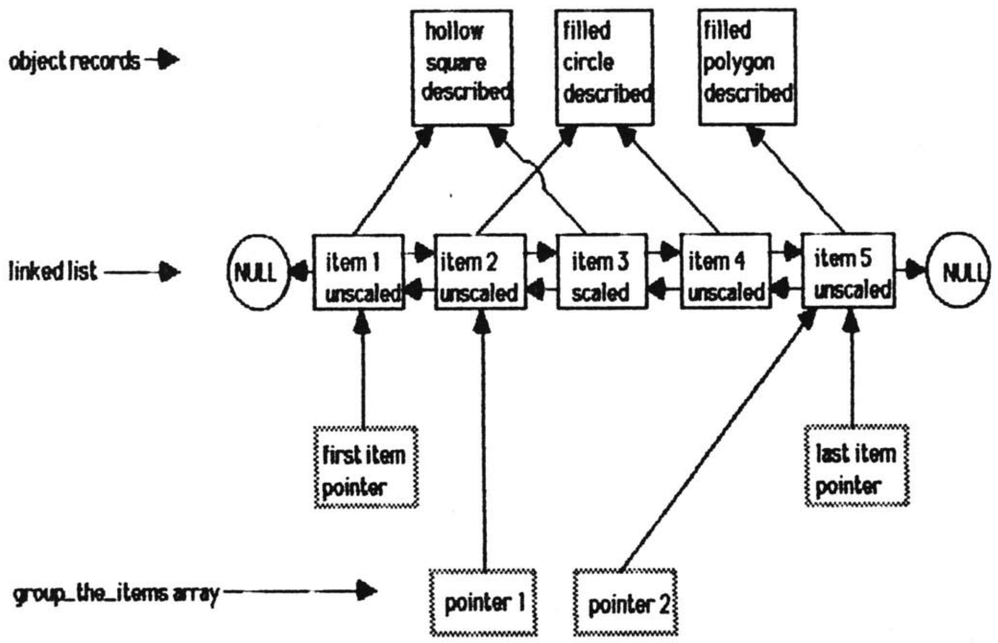


2. The user wants to select a group of items to copy into another area of the pattern.

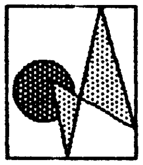
Copy_group routine is executed. It calls **select_group** routine to read the items that the user selects for a group. **Select_group** marks these items by assigning their addresses to ***mark_the_items** pointers. After the group is selected, **order_the_items** is called. This routine uses **mark_the_items**

pointers to order the selected items into **group_the_items** array, where items of lower indices lie behind items of higher indices.

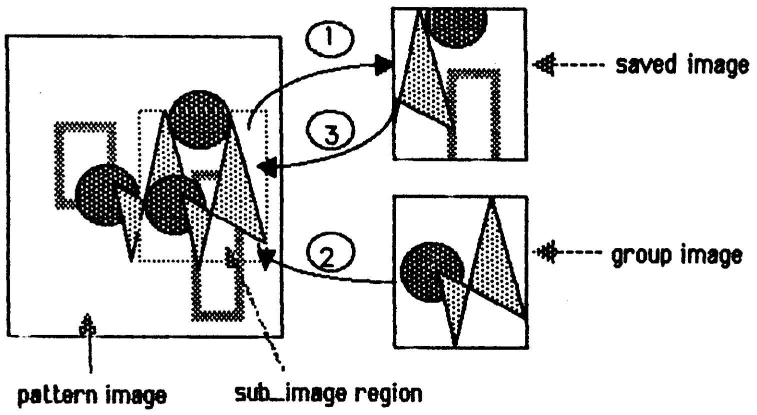




Prepare_group routine is called. It draws the group into a pixel map in the <u>invisible</u> frame buffer. This temporary image behaves like a cursor in the pattern space. It is repeatedly copied and erased in the pattern space, following the current location of the mouse. The lower left corner of the group image in the pattern space, at any instance, coincides with the location of the mouse.



Designate a temporary area in the invisible frame buffer to save the portion of the pattern image which is currently being replaced by the group pattern. As the group image is moved around the pattern space, the saved sub_image is copied back into its proper space, and the new area that the group image draws over is copied into this temporary area. This process continues until the user has selected a location in the pattern space for the group of shapes.



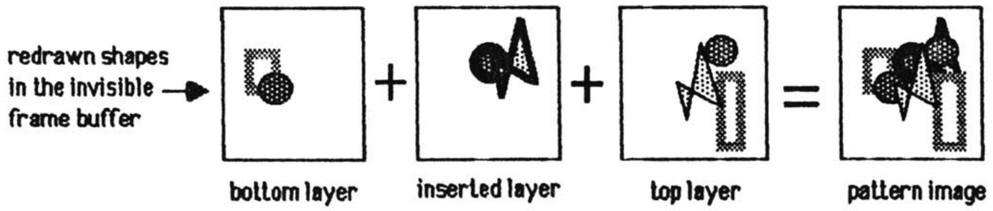
(1) copy sub_image to saved image area.

(2) copy group to sub_image location .

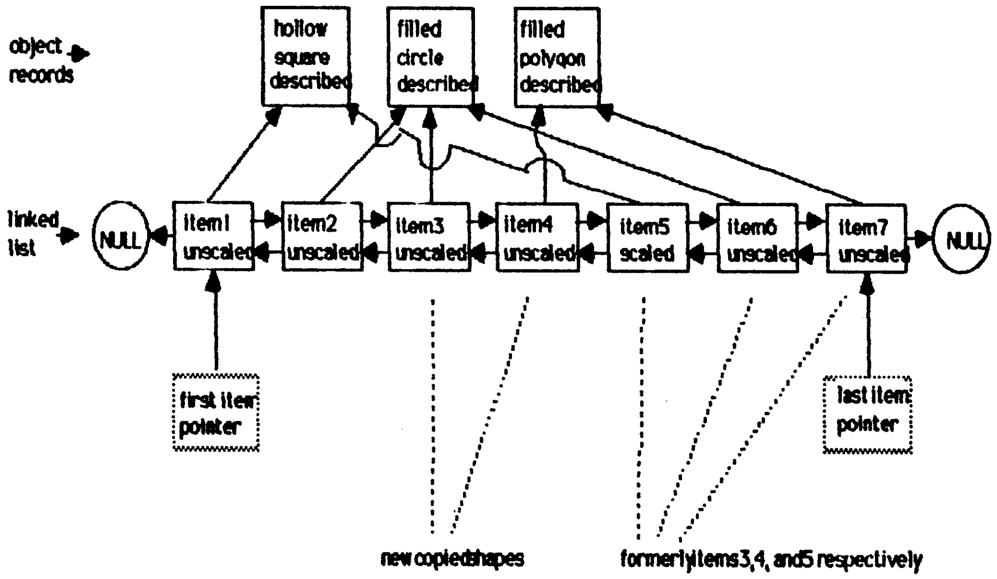
(3) copy saved image back to its proper location.

The user has decided to place this group behind a shape. A temporary pointer is assigned the address of the record in the linked list which is associated with the selected shape that the group will be placed immediately behind of. All the shapes lying behind the selected shape are redrawn in the invisible frame buffer and then copied together into the pattern space. This method of copying the collection of shapes at the same time rather than redrawing each shape into the pattern space is more appealing to the eye. Redrawing each shape elicits a sense of disorder and may unnecessarily prompt the user to wonder why his/her pattern image is being disrupted even though it is for a short instance. Copying the collection of shapes is quick and minimally, if at all, noticeable to the eye. In this way, the mystery of how the group of shapes is placed between other shapes is not revealed to the user.

After the collection of preceding shapes are displayed, the selected group is copied into the pattern space. And then the selected shape that the group lies behind and the collection of shapes that succeed (lie on top) of it are redrawn in the invisible frame buffer and copied together into the pattern space.



Once the location of the group has been determined, the linked list data structure of the pattern image is updated to include the new group of shapes.



Subroutines Which Record Descriptive Values into Data Structures for Each Shape Created:

Each of the following routines allocates memory to create a new record of the type **struct objects** and assigns values to this record which describe the new shape just created. **Shape_type** item in the record is assigned an integer value according to the following key:

1 = unfilled box

2 = filled box

3 = line

4 = unfilled polygon

5 = filled polygon

6 = unfilled circle

7 = filled circle

8 = freehand line

The value of **shape_type** determines which conditional part of the record is assigned values. The record contains four conditional parts, each of which is a sub-record: **polygon, box_or_line, circle, freehand.** **Shape_type** also indicates to subroutines concerned with this, the type of shape to redraw.

new_box(box_type, x1, y1, x2, y2, color, width, half_width). After this routine completes its computations, it calls the generic routine **new_box_or_line.**

new_line(x1, y1, x2, y2, color, width, half_width). After this routine completes its computations, it calls the generic routine **new_box_or_line.**

new_box_or_line (box_or_line_type, lower_x, lowery_y, upper_x, upper_y, x1, y1, x2, y2, color, width). This routine assigns the appropriate values to the new record of type **struct objects**. It then calls **new_shape_attached.**

new_polygon(polygon_type, number_of_vertices, array_of_coordinates_of_vertices, color, width, half_width). This routine assigns the appropriate values to the new record of type **struct objects**. It then calls **new_shape_attached.**

new_circle(circle_type, center_x, center_y, radius, color). This routine assigns the appropriate values to the new record of type **struct objects**. It then calls **new_shape_attached.**

new_freehand(**freehand_pattern, lower_x, lower_y, upper_x, upper_y, transparency/background_value**). This routine assigns the appropriate values to the new record of type **struct objects**. It also saves onto disk the image, **freehand _pattern**. The reason why a freehand drawn shape is recorded in this manner is that it is more memory efficient than recording all attribute values to describe it. Stored in the new record is the name of the temporary file in which the pattern is stored in. This routine then calls **new_shape_ attached.**

new_shape_attached(**lower_x, lower_y, upper_x, upper_y, object_pointer**). **Object_pointer**, which addresses the new record of type **struct objects,** is passed to this routine. This routine creates a new record of type **struct attach** which describes the actual location of the newly displayed shape in the pattern image space. The location that its pointer, ***shape_pointer**, addresses is assigned the value of **object_pointer**. This routine also assigns values to pointers that address the previous record in the linked list and the NULL value. It also assigns the appropriate pointer in the previous record to address this new one. Since this program records each newly generated shape as lying on top of all shapes in the pattern, this pattern's ***last_item_pointer** is updated to address this recent addition.

Subroutines Which Evaluate Information in the Data Structures in order to Redraw a Shape at a Specified Location:

draw_object(destination_space, pointer_to_record_in_linked_list, low_x_of_shape in destination space, low_y_of_shape in destination space). This routine first checks if the record in the linked list describes a scaled shape. If it does not, it checks the type of shape in the **struct objects** record that it addresses and calls the appropriate subroutine to redraw the shape. If the shape to be drawn is scaled, this routine calls **draw_scaled_shapes.**

draw_box(destination_space, low_x_of_shape in destination space, low_y_of_shape in destination space, pointer_to_struct_objects_record). This routine reads the parameters in the **struct objects** record to draw a rectangle relative to the coordinates passed to this routine. This is called <u>from</u> **draw_object.**

draw_polygon(destination_space, low_x_of_shape in destination space, low_y_of_shape in destination space, pointer_to_struct_objects_record). This routine reads the parameters in the **struct objects** record to draw a polygon relative to the coordinates passed to this routine. This is called <u>from</u> **draw_object.**

draw_circle(destination space, low_x_of_shape in destination space, low_y_of_shape in destination space, pointer_to_struct_objects_record). This routine reads the parameters in the **struct objects** record to draw a circle relative to the coordinates passed to this routine. This is called from **draw_object.**

draw_freehand(destination_space, low_x_of_shape in destination space, low_y_of_shape in destination space, pointer _to_struct_objects_record). This routine reads the parameters in the **struct objects** record to access the file on disk that the freehand pattern is temporarily being saved in. This pattern is copied relative to the coordinates passed to this routine. This is called from **draw_object.**

draw_line(destination_space, low_x_of_shape in destination space, low_y_of_shape in destination space, pointer_to_ struct_objects_record). This routine reads the parameters in the **struct objects** record to draw a line relative to the coordinates passed to this routine. This is called from **draw_object.**

draw_scale(destination_space, low_x_of_shape in destination space, low_y_of_shape in destination space, pointer_ to_struct_objects_ record). This routine checks if the scaled item is either a freehand drawn shape or a circle. If it is either of these shapes, there are not sufficient coordinates recorded, if at all, to use the matrix scale function. For a circle or a freehand shape, **robt_scale** is called. For a rectangle, line, or polygon shape, **math_scale**, which multiplies coordinates with a scale matrix, is called.

robt_scale(destination_space, low_x_of_shape in destination space, low_y_of_shape in destination space, pointer_

to_struct_objects_record, width_of_scaled_shape, height_of_scaled_shape). This routine scales the shape described in the **struct_objects** record into the location indicated by the coordinates, and the values for width_of_scaled_shape and height_of_scaled_shape passed to this routine. This routine first calls either **draw_freehand** or **draw_circle** to generate the unscaled shape into a source pixel map, then uses a scan-line based method of scaling the source image to generate the scaled image in the proper area of the destination space.

math_scale(destination_space, **pointer_to_struct_objects_record, height_scale_factor, width_scale_factor, center_x_of_shape, center_y_of_shape, new_center_x, new_center_y**). This routine first calls matrix routines which perform matrix multiplication operations to set up the correct scale matrix. Then it calls **trans_box_or_line** if the shape is either a rectangle or a line, or it calls **trans_polygon** if the shape is a polygon. The computed matrix is passed to the appropriate routine.

trans_box_or_line(destination_space, **scale_matrix, pointer_to_struct_objects_record**). This routine multiplies the coordinate pairs of the vertices/endpoints with **scale_matrix** to calculate the new location of these points in the scaled version of the shape. If the shape is a line or a hollow box, it connects the new points with edge(s) of the correct line_width and color as defined in the **struct_objects** record. If the shape is a filled rectangle, it connects the new points and fills the shape with the correct color.

trans_polygon(destination_space, scale_matrix, pointer_to_struct_objects_record). This routine multiplies the coordinate pairs of the vertices with **scale_matrix** to calculate the new location of these points in the scaled version of the shape. If the polygon is hollow, it connects the vertices with edges of the correct line_width and color as defined in the **struct_objects** record. If the shape is a filled polygon, it connects the vertices and fills the shape with the correct color.

Subroutines Associated with a User-Selected Shape or User- Selected Group:

prepare_surface(pointer_to_record_in_linked_list). This routine calls **draw_object** to redraw every shape composed in the pattern image, except the selected shape. It redraws them in a pixel map in the invisible frame buffer, and then BitBlts it to the pattern image area.

***find_selected_shape**(x, y). This function returns a pointer to the calling routine. The pointer addresses the **struct attach** record in the linked list that corresponds with the selected item. This routine searches backwards through the linked list until it finds the first record where the x, y coordinates, which are passed to this routine, lie within the bounding coordinates defined in the record. It then calls **verify** to check that the x, y coordinate is actually lying on the shape. Sometimes the x,y coordinate is contained within the hole of a hollow shape. Also, since the bounding coordinates represent a rectangular area, the x, y coordinate may be at a point that is not part of the shape. If **verify** returns a false value, this function keeps searching through the list until it finds the next record whose bounding coordinates encompass the selected point. If the whole list is searched and no shape is found, the NULL value is returned to the calling routine, else the pointer to the address of the determined item in the linked list is returned.

verify(pointer_to_possibly_found_item_in_the_linked_list, **boolean variable, x**, y). This routine temporarily draws the shape in question into the invisible frame buffer, and samples an area of 3 x 3 pixels around the (**x**, y) point. If none of the pixels sampled are of the shape's color, the **boolean variable** returns FALSE to the calling routine, else it returns TRUE.

select_group(button_value). For each selection of a shape for the group, this routine marks the shape by setting a pointer of the **mark_the_items** array equal to the pointer returned by **find_selected_shape.** It also draws a rectangle around the shape selected. There are two ways for the user to finish selecting. One, click a button to indicate that the selection is done, or two, click a different button to indicate that the selection has been terminated and the shapes should not be grouped together. **Button_value** returns to the calling procedure the value of the button was clicked.

prepare_group_surface(). This routine calls **draw_object** to redraw into a temporary pixel map in the invisible frame buffer those shapes that were not grouped and BitBlts this to the pattern image space. To check which shapes were selected, it calls **is_it_marked** to test each record in the linked list, starting with the first item, to see if a **mark_the_items** pointer is addressing it. If a pointer is addressing the record, it calls **push_object** in order to assign the next **group_the_items** pointer in the array to point at this record. **Group_the_items** array is automatically sorted from the shape lying at the bottom of the group to the shape lying on top of the group.

is_it_marked(pointer_to_the_record_in_question, boolean variable). This just checks to see if pointer_to_the_record_in_question is equal to any of the **mark_the_items** pointers. If it is, boolean_variable returns TRUE, else FALSE.

push_object(pointer_to_record_in_linked_list). This routine sets the next **group_the_items** pointer in the array equal to pointer_to_record_in_linked_list.

order_the_items(). This routine scans through the linked list from beginning to end calling **is_it_marked** to determine if a shape has been selected for a group, and if it has, it calls **push_object**.

Subroutines Associated With Repositioning a Shape or Group of Shapes:

place _after(pointer_to_record_in_linked_list, x_location, y_location, width_of_inserted_image, height_of_inserted_image, inserted _ image). This routine redraws into a pixel map in the invisible frame buffer all the shapes associated with the records preceding **pointer_to_record_in_linked_list**, and then the shape associated with **pointer_to_record_in_linked_list**. Then it BitBlts the **inserted_image** to this pixel map. Finally, it redraws all the shapes associated with the records succeeding **pointer_to_record_in_linked_list** into this area. Then it copies this pixel map to the pattern image space.

prepare_object(**pointer_to_record_in_linked_list, brush_ pattern_number, shape_image, saved sub_image).** This routine defines pixel maps in the invisible frame buffer for **shape_image and saved_sub_ image. Shape_image** contains the selected shape. This is the image which follows the mouse location in the pattern image space. **Saved_sub_image** is the space where the portion of the pattern currently being covered by **shape_image** is saved. When **shape_image** is moved to a new location, **saved_sub_image** is copied back to its proper place.

prepare_group(group_image, saved_sub_image, group_width, group_height). This routine defines pixel maps in the invisible frame buffer for **group_image and saved_sub_image.** Drawn into **group_image** is each of the shape of the selected group. This is the image which follows the mouse location in the pattern image space. **Saved_sub_image** is the space where the portion of the pattern currently being covered by **group_image** is saved. When **group_image** is moved to a new location, **saved_sub_image** is copied back to its proper place.

init_shape(s)_cursor(pattern_image, initial_x, initial_y, shape/

group_width, shape/group_height, shape/group_image, saved_sub_image, transparency_value). This routine places **shape/group_image** into an initial position in pattern_image. Prior to copying this to the initial location, it saves the to-be replaced portion of the **pattern_image** into **saved_sub_image).**

move_and_read_shape(s)_location(location_x, location_y, **button_state, shape/group_image, saved_sub_image).** This routine first reads the location of the mouse and calls **move_shape(s) cursor** to copy **shape/group_image** to the current location. Then it reads the state of the mouse buttons. It returns to the calling routine the value of **button_state** and the current location of the mouse.

move_shape(s)_cursor(location_x, location_y, **shape/group_image, saved_sub_image).** This routine first copies the contents of **saved_sub_image** back to its proper place in the pattern image. Then it copies the portion of the pattern image at the current location of the mouse to **saved_sub_image**, and replaces that portion with a copy of **shape/group_image.**

Subroutines Which Execute the Shapc Manipulation Functions, Move, Copy, Scale, and Delete:

move_item(). This routine is executed when a user selects to move a shape in the pattern. **Find_selected_shape** is called to determine the shape selected and to locate the associated records which describe it. If a shape is found to have been selected, **prepare_surface** and **prepare_object** are called. Then **init_**

shape(s)_cursor and **move_and_read_shape(s)_location** is called to allow the user to move the selected shape around the pattern space. Once a position for the shape is chosen, the user can decide to place it either behind or in front of a selected shape (note: its location on the x-y plane will not change). If the user decides on either of these options, **place_after** is called, and pointers of the relevant records in the linked list are repositioned to reflect this change. Upon the final positioning of the shape, the bounding coordinates of the record in the linked list associated with it are updated with the bounding coordinates of the new location.

move_group(). This routine is executed when a user wants to move a selected group in the pattern. **Select_group** is called to determine the shapes selected and to mark the associated records which describe them. If a group has been formed, **prepare_group _surface** and **prepare_group** are called. Then **init_shape(s)_ cursor** and **move_and_read_shape(s)_location** is called to allow the user to move the selected group around the pattern space. Once a position for the group has been chosen, the user can decide to place the group either behind or in front of a selected shape (note: the group's location on the x-y plane will not change). If the user decides on either of these options, **place_after** is called, and pointers of the relevant records in the linked list are repositioned to reflect this change. Upon the final positioning of the group, the bounding coordinates of each record in the linked list associated with the shapes involved are updated with the bounding coordinates of each shape's new location.

scale_item(). This routine is executed when the user wants to scale an individual shape. **Find_selected_shape** is called to determine the shape selected and to locate the associated records which describe it. If a shape is determined to be selected, a rubberband rectangle scalar is provided to the user to indicate the size and location of the scaled shape. Then the user decides whether or not the source shape is to be wiped from the pattern space. If not, a new record, which is to be attached to the linked list, is created. If the selected shape is either a polygon, box, or line, **math_scale** is called, else **robt_scale** is called. After the scaled shape is generated, the user can decide whether to place it behind or in front of a selected shape in the pattern. If the user decides on either of these options, **place_after** is called, and pointers of the relevant records in the linked list are repositioned to reflect the change. When the scaling and positioning of the shape is done, the parameters in the record associated with it are assigned new values. The bounding coordinates are assigned the extreme coordinates of the scaled shape's location. **Scale_type** is assigned the value of 2, which indicates to the drawing routines that the addressed shape is to be scaled into the location described by the bounding coordinates of the record.

scale_group(). This routine is executed when the user wants to scale a selected group. **Select_group** is called to determine the shapes selected and to locate the associated records which describe them. If a group was formed, a rubberband rectangle scalar is provided to the user to indicate the size and location of the scaled group. Then the user decides whether or not the source group is to be wiped from the pattern space. If not, new records, which are to be attached to the linked list, are created. Then each shape in the group is scaled relative to the scale size of the entire group. If the shape is either a polygon, box, or line, **math_scale** is called, else **robt_scale** is called. After the scaled group is generated, the user can decide whether to place it behind or in front of a selected shape in the pattern. If the user decides on either of these options, **place_after** is called, and pointers of the relevant records in the linked list are repositioned to reflect the change. When the scaling and positioning of the group is done, the parameters in each record associated with shapes in the scaled group are assigned new values. The bounding coordinates are assigned the extreme coordinates of the scaled shape's location. **Scale_type** is assigned the value of 2, which indicates to the drawing routines that the addressed shape is to be scaled into the location described by the bounding coordinates of the record.

copy_item(). This routine is executed when a user selects to copy a shape in the pattern. **Find_selected_shape** is called to determine the shape selected and to locate the associated records which describe it. If a shape is found to have been selected, **prepare_object** is called. Then **init_shape(s)_cursor** and **move_and_read_shape(s)_location** is called to allow the user to move the copied shape around the pattern space. A new record for this copied shape is created and attached to the linked list. Its record addresses the same **struct object** record as the source shape. Once a position for the shape is chosen, the user can decide to place it either behind or in front of a selected shape (note: its location on the x-y plane will not change). If the user decides on either of these options, **place_after** is called, and pointers of the relevant records in the linked list are repositioned to reflect this change. Upon the final positioning of the copied shape, the bounding coordinates of the record in the linked list associated with it are updated with the bounding coordinates of the new location.

copy_group(). This routine is executed when a user selects to copy a group of shapes in the pattern. **Select_group** is called to determine the shapes selected and to locate the associated records which describe them. If a group was formed, **prepare_group** is called. Then **init_shape(s)_cursor** and **move_and_read_shape(s) _location** is called to allow the user to move the copied group around the pattern space. A new record for each shape in the copied group is created and attached to the linked list. These records address the same **struct object** records that the records associated with the source group address. Once a position for the group is chosen, the user can decide to place it either behind or in front of a selected shape (note: its location on the x-y plane will not change). If the user decides on either of these options, **place_after** is called, and pointers of the relevant records in the linked list are repositioned to reflect this change. Upon the final positioning of the copied group, the bounding coordinates of their records in the linked list are updated with the bounding coordinates of the shapes' new locations.

delete_item(). This routine is executed when the user wants to delete shapes from the pattern. **Find_selected_shape** is called. If a shape was selected, the pointers of the records immediately preceding it and succeeding it in the linked list are re-addressed to point at each other. The memory allocated to the **struct_attach** record in the list is released into the memory pool. If the **struct_objects** record, which defines the shape, is not being addressed by other items in the linked list, it is also released to the memory pool.

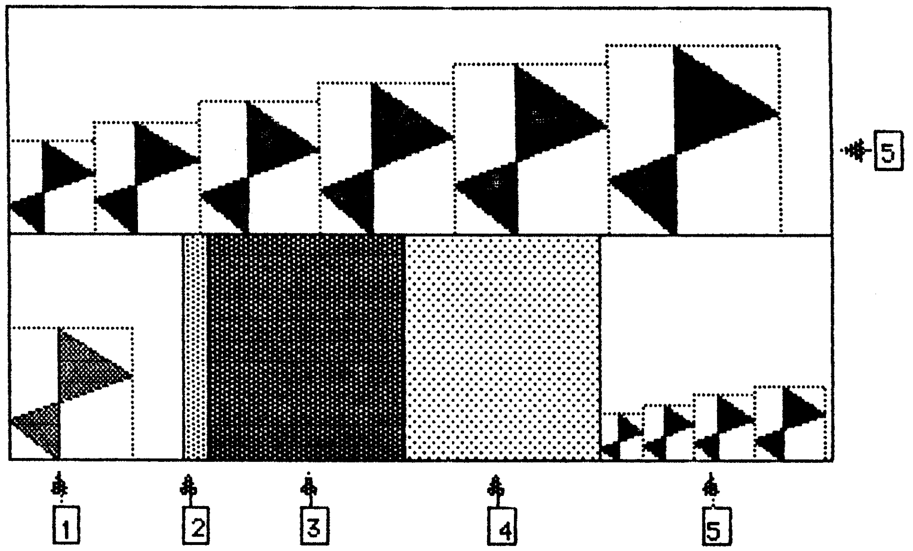
copy_shapes(). This routine is executed when a user wants to iterate a group of shapes from one pattern into another. **Select_group** is called to determine the shapes selected and to locate the associated records which describe them. If a group was formed, **prepare_group** is called. Then **init_shape(s)_cursor** and **move_and_read_shape(s) _location** is called to allow the user to move the copied group around the pattern space. The group is iterated each time the user clicks a button on the mouse. A new record for each copied shape is created and attached to the linked list. These records address the same **struct object** records that the records associated with the source group address. After the user is done iterating the shapes, the bounding coordinates of their records in the linked list are updated with the bounding coordinates of the shapes' new locations.

Dynamic Iteration of Patterns:

This algorithm is divided in the following manner:

1. prompt the user for the extreme sizes that a pattern can grow and shrink to.
2. prompt the user for the change in transparency and opacity factors. This factor determines the amount of change in opacity or transparency when the pattern is sequentially iterated.
3. interpolate and generate the patterns which occur between the two specified extreme sizes.
4. create the opacity and transparency veils which are merged with the current pattern being iterated.
5. apply the appropriate operations on the current pattern of the sequence as determined by the user's control of the pattern's application.

Invisible Frame Buffer During Dynamic Paint:



$\boxed{1}$ current mark being copied to screen. If the pattern is to be changing in size or opacity, this image is being redrawn continuously.

$\boxed{2}$ opacity strip of current pattern. If the pattern is becoming more transparent then each pixel in the veil is determined by the following operation:
veil pixel value = max (transparency pixel value, veil pixel value).
If the pattern is becoming more opaque then each pixel in the veil is determined by the following operation:
veil pixel value = min (opacity pixel value, veil pixel value).

$\boxed{3}$ transparency veil

$\boxed{4}$ opacity veil

$\boxed{5}$ interpolated brush patterns

Scenario and Description of the Algorithm:

1. The user selects a pattern from the brush pattern menu. The

pattern is retrieved from the disk and copied into an area of the invisible frame buffer.

2. Set the background color of the pattern as the transparency value, so that when BitBlt with the transparency option is performed on the pattern, these pixel values will not be copied into the destination image. The transparency values for patterns in this program have been selected to index color items greater than 240 in the color lookup table.
3. Prompt the user for the two extreme sizes that the pattern can grow and shrink to.
4. Prompt the user for the amount of change in opacity and transparency factor.
5. Determine the number of patterns, two which are of the extreme sizes and the ones that occur between them, that can be generated and stored sequentially in the invisible frame buffer.
6. From the number of patterns that can be generated and the differences in height and width of the two extreme sizes, calculate the difference in height and the difference in width between adjacent patterns of the sequence.
7. From the values determined in 5 and 6, above, generate the sequence of patterns into the invisible frame buffer.
8. Define a region for the transparency veil and a region for the opacity veil, both of which lie in the invisible frame buffer. Clear all the pixels in the transparency veil to the first item (0) in the color lookup table. Clear all the pixels in the opacity veil

to the transparency/background color of the pattern (a value greater than 240).

9. Define a region for the opacity strip, which later merges with a solid pattern to change its opacity or transparency. The width of this strip is $^1/_7$th the width of the opacity or transparency veil. The height is the same as both. When a pattern is to become more transparent, a strip of pixels of the transparency value is copied into here. When a pattern is to become more opaque, a strip of pixels of the first color item value (0) is copied into here.
10. Calculate the density of transparent and opaque pixels that should be assigned in the respective veils by multiplying the number of pixels contained in each veil with the change in transparency/opacity factor that the user selected.
11. With a random number generator determining the location of target pixels, which total the amount calculated in **10**, above, assign the transparency value (value greater than 240) to target pixels in the transparency veil, and assign the value of the first item (0) in the color lookup table to target pixels in the opacity veil.
12. Divide the veils into seven initial strips. The transparency and opacity veils are constantly divided into six or seven vertical strips. After a complete set of strips in a veil has been used, the location of the strips are shifted horizontally a difference of one pixel. This allows efficient change for a pattern becoming either

continuously more transparent or more opaque. In either case, strips of one veil is repeatedly being copied into the opacity strip of the pattern. Shifting the strips, when a set has been used, expands the range of opaqueness that the opacity strip can possess.

13. Now the user is ready to iterate the interpolated set of patterns onto the image surface. The intial states of the brush is as follows: a pattern of one of the two extreme sizes is the current pattern being applied, the brush is being applied with no size change and no opacity change.

14. The program repeats a **while loop** of instructions, which, first, interprets from the user how the current pattern is to be applied, and then applies the appropriate operations to access, modify, and apply the current pattern to the image.

    a. While the middle button or right button is being depressed by the user, the change in size state, and the change in opacity state is evaluated. The value of the parameters describing the change in size state and the change in opacity state can each have a value of -1, 0, or 1. of -1, 0, or 1. For the change in size, the state toggles between growing, staying the same size, and shrinking. For the change in opacity, the state toggles between becoming more transparent, staying at the same opacity level, and becoming more opaque.

    b. Each of the patterns in the interpolated sequence has a number associated with it. The first pattern has a value of 0, and the last pattern has the value of (number of patterns - 1). The current pattern in the sequence is accessed by its value. If the value of state of size change is either -1 or 1, this value is added to the value which indexes the pattern to be applied, in order to access the previous or next pattern in the sequence.

c. If the value of state of opacity change is -1, pixels of the transparency value from a strip in the transparency veil are copied into the brush pattern's opacity strip. If the value of state of opacity change is 1, pixels of the value 0 from a strip in the opacity veil are copied into the brush pattern's opacity strip.

d. If either of the above described states are not 0 (static), the current pattern (could be the same, previous, or next) is copied into the pixel map designated for the current pattern to be applied. Then the opacity strip is overlapped into this area. The strip is repeatedly copied next to each other until it covers the full width of the pattern.

e. The user indicates the current state of size change and opacity change, by clicking buttons on the mouse.

Conclusion:

The impetus for the research involved in this thesis is the situation of "wu pi" (not having brush) that can be experienced in digital paint systems. For the digital medium, I have altered the meaning of this ancient Chinese expression to describe the situation in which the artist knows visual characteristics of the image that he/she would like to render, but is not provided with adequate digital marking functions to effectively generate it. The digital paint package developed in this thesis project reduces the occurences of "wu pi". This is demonstrated in the sample images, which were digitally painted with this package.

Allowing the user to design personal brush pattern modules is an effective means to "painting" personal images of various visual qualities. Personal brush patterns, however, is only a particular class of tools that I have focused on. Future research should explore other aspects of the digital paint medium that the user should be able to define. If the user is provided more and more control of the medium, the occurences of "wu pi" may diminish completely, and in the digital medium which is governed by software, this could potentially happen. The following is a list of potential aspects of a digital paint system which the user should be able to define: 1. defining the fluidity of the digital "brush" and the fluidity of the image surface and the characteristic of the mark that is the product of the two (the mark does not have to simulate the result that would occur with physical materials), 2. defining transformations of the brush's physical features as it is being applied to the image surface (for example, change colors when it is applied over certain colors), 3. defining the characteristics of the image surface (for example, wavy, bumpy, curved, cracked, prickly, wet, etc.). The potential visual effects to digital images which a computer paint medium could provide is an issue which still needs to be further explored.

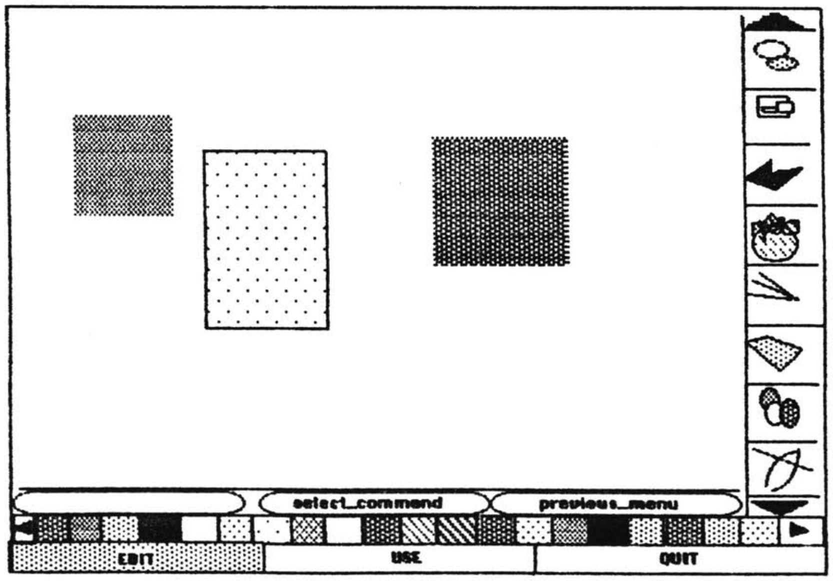
Appendix: User's Guide

Components of User Interface

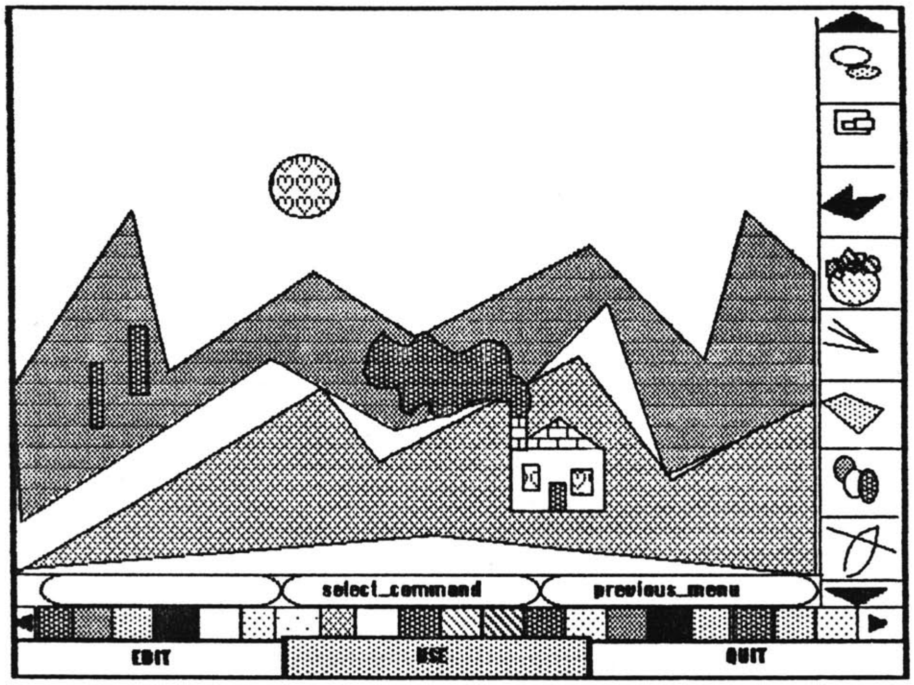
Screen Display

The display is divided into five major sections: hierarchical command selection menu, scrolling brush pattern menu, scrolling color bar, mouse legend, and work surface. In the edit brush pattern mode, the user creates windows in the work surface. Brush patterns are created and edited in these windows. In the use/paint mode, the work surface functions as the area in which the image is generated.

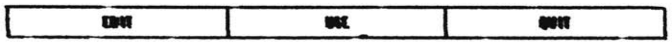
Edit Mode



Use Mode



Hierarchical Command Selection Menu



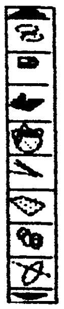
The command selection menu at any time displays a single level from a tree structure of commands. The user traverses the menu by either selecting a command which identifies a group of commands at the next level of the tree, or by depressing the right button on the mouse which brings the user back to the previous level. A command box is highlighted whenever the cursor lies inside of it in order to give visual feedback to the user of the current command item being touched. The user selects a command with the middle button of the mouse.

Hierarchical Menu Structure

```
LEVEL1               LEVEL 2               LEVEL 3

        ----------> WINDOWS --------> NEW BRUSH
        |                             COPY BRUSH
        |                             MOVE BRUSH
        |                             CLEAR BRUSH
        |                             DELETE BRUSH
        |                             TEST BRUSH
        |
        |
        ----------> SHAPE ----------> FREEHAND
        |                             BOX
        |                             POLYGON
        |                             CIRCLE
EDIT----|                             LINE
        |                             COPY SHAPES
        |
        |
        ----------> COLORS ---------> EDIT COLORS
        |                             BLEND COLORS
        |                             LOAD COLORS
        |                             SAVE COLORS
        |
        |
        ----------> TRANSFORM ------> MOVE
        |                             COPY
        |                             SCALE
        |                             DELETE
        |
        |
        ----------> DISK -----------> SAVE BRUSH
                                      ERASE BRUSH
```

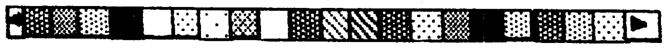
```
LEVEL 1              LEVEL 2                   LEVEL 3

        ----------> IMAGE -----------> PAINT
        |                               DYNAMIC PAINT
        |                               SCALE PIC
        |                               CLEAR PIC
US ---- |                               SAVE PIC
        |                               LOAD PIC
        |
        |
        ----------> COLORS ----------> FILL
                                        EDIT COLORS
                                        BLEND COLORS
                                        LOAD COLORS
                                        SAVE COLORS


QUIT
```

Scrolling Brush Image Menu



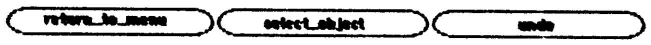
This menu displays on the screen at a time eight of the brush patterns saved on disk. An arrow button is located on the top of the menu list and another at the bottom to allow for scrolling of the menu. When the top button is selected, the menu scrolls up one item and when the bottom button is selected, it scrolls down one item. When an item is to be selected from this menu, the cursor is bound inside this menu until the user has made a selection.

Scrolling Color Bar



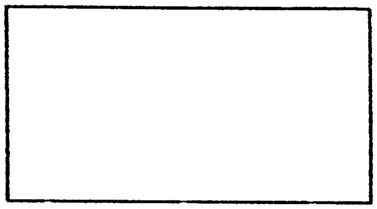
This menu displays on the screen at a time 20 of the 210 colors available to the user in the color table. Arrows are located to the right and left of the color bar to allow for scrolling of the menu. The user can select to scroll the menu either a color item at a time or 20 color items at a time. When an item is to be selected from the color bar, the cursor is bound inside it until the user has made a selection. While in the color bar, the user can edit any of the color slots in the color table. Two methods of editing colors are provided. One allows a color's red, green, and blue values to be modified, and the other allows it's hue, lightness, and saturation values to be modified. The user is also able to save a color table onto disk or load one from disk.

Mouse Legend



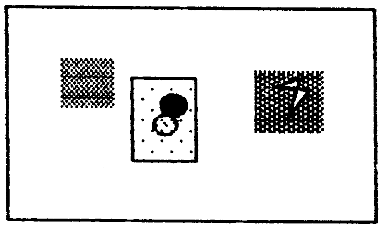
This legend is located directly above the color bar. It contains three oval shapes that correspond to the three buttons on the mouse. Displayed inside each oval is a description of it's button's function at that time. This legend is continuously displayed in order to alleviate the confusion that a user may experience when keeping track of button functions which vary depending on the application.

Work Surface



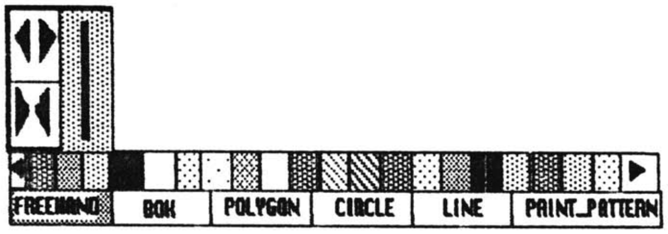
This is the large area on the screen in which the user does most of his/her work. In the edit mode, the user creates and edits brush patterns in windows on this surface. In the use/paint mode, the image is generated on this surface. When the user has edited patterns, then generates a picture in the use/paint mode, and then goes back to the edit mode, the windows and patterns reappear on the surface in the same state that the user had left them. When leaving the use/paint mode and entering the edit mode, if the user has generated a picture, the program prompts the user to choose whether the picture should be saved or not, since the image will be replace by the editing windows.

Brush Editing Windows



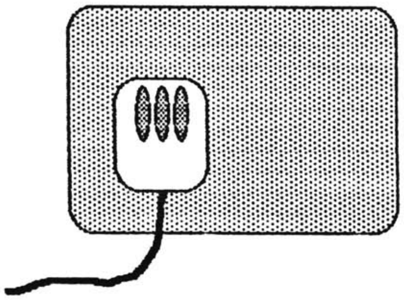
The user creates and edits brush patterns inside these windows. The user is provided functions to manage these windows in the work surface. Windows can be moved, copied, cleared to a single color, or deleted from the work surface. At any time, only one of the windows is active for editing. To work in a different window, the user either selects a point in an existing window on the screen, or selects a point on the work surface not currently being occupied by a window. When the latter case occurs, the program allows the user to create a new window by using a rubberband box.

Pop Up Menus



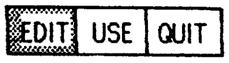
Pop up menus are displayed in most cases where a function needs the user either to set certain parameters or to make a decision from a selection of options that the function can act on. The instances that a user would need to input through the keyboard is when he/she is naming a color table to be saved or loaded, and when he/she is naming an image to be saved or loaded from disk.

Mouse



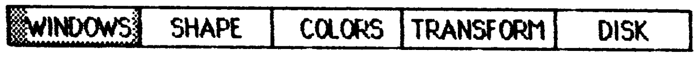
The primary means of user input to the package is through an optical mouse with three buttons. A "+" shaped cursor is constantly displayed on the screen at the current location of the mouse. The cursor is always bounded in the current area of the screen display in which the user is either working or making a selection. For example, the cursor is bounded to the work surface area when the user is painting a picture, and is bounded to a pop_up menu when the user has to make a selection.

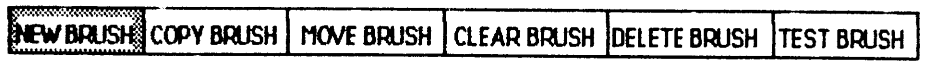
Components of Brush Design Mode



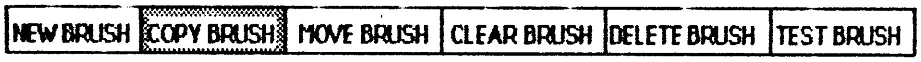
This mode provides utilities to create and edit brush patterns. Each pattern is created and edited in its own window. If there are no windows on the work surface when the user selects a function to create a shape, he/she will be forced to create one before the selected function is executed. Each window has a background color that the user selects. A window's background is not part of the brush pattern. Assigning a color to the window's background merely allows the user to see how a brush pattern looks on a potential image background color. The user makes all item selections by pointing the cursor at the location of the item on the display screen and depressing the middle button on the mouse.

Window Management

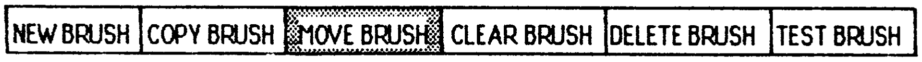




NEW BRUSH allows a user to select the current window to work in. The user points at a location on the work surface with the cursor and clicks the MIDDLE BUTTON on the mouse. If there is a window at this location, it becomes the current window to work in and the user is returned to the command menu. If there is not a window here, the user creates a new window with a rubberband rectangle, using the selected point as one corner of the window. The user drags the diagonal corner of the rectangle with the cursor and clicks the MIDDLE BUTTON when he/she is satisfied with the size of the window. Then the cursor is popped into the color bar and constrained there until the user selects a background color for the new window (see section on color bar described in **COLOR TABLE ROUTINES** for description of user's options in the color bar). At any time in the work surface the user can quit the function by clicking the RIGHT BUTTON. This will return the user to the command menu. If this occurs when a new window is being created, the window will disappear. All operations which create new shapes or allow the user to manipulate existing shapes will constrain the user to the current editing window.



COPY BRUSH allows the user to select an existing window and copy it and its contents. The user selects the source window by placing the cursor inside of it and clicking the MIDDLE BUTTON. After a window has been chosen, a copy of it appears and the user drags this window by moving the mouse until he/she has located a place for it on the work surface. The user clicks the MIDDLE BUTTON when he/she has decided on the new location of the copy. The copied window now becomes the current window to edit in. At any time in the work surface, the user can click the RIGHT BUTTON to quit this function.



MOVE BRUSH allows the user to select an existing window and move it and its contents. The user selects the window by placing the cursor inside of it and clicking the MIDDLE

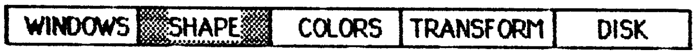
BUTTON. After a window has been chosen, the user drags it by moving the mouse until he/she has selected a new location by clicking the MIDDLE BUTTON again. At any time in the work surface, the user can click the RIGHT BUTTON to quit this function.

CLEAR BRUSH allows the user to erase the contents of an existing window and to select a new background color for it from the color bar (see section on color bar described in **COLOR TABLE ROUTINES** for description of user's options in the color bar). The user selects the window by placing the cursor over it and clicking the MIDDLE BUTTON. Clicking the RIGHT BUTTON, instead, will quit this function.

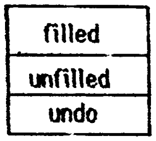
DELETE BRUSH allows the user to delete a window and its contents from the work surface. The user selects the window by placing the cursor over it and clicking the MIDDLE BUTTON. Clicking the RIGHT BUTTON, instead, will quit this function.

TEST BRUSH allows the user to test paint a selected brush from the work surface. First the user selects a brush pattern by placing the cursor over its window and clicking the MIDDLE BUTTON. Then he/she creates a temporary test window on the work surface by using a rubberband rectangle. The test window may cover any area on the work surface. When the user is done with this routine, whatever items the test window covered will reappear on the work surface unchanged. Diagonal corners of the temporary window are selected by clicking the MIDDLE BUTTON at each location. The user then selects a color from the color bar to clear the test window with (see section on color bar described in **COLOR TABLE ROUTINES** for description of user's options in the color bar). Then the cursor is constrained inside the test window. Depressing the MIDDLE BUTTON will iterate the brush pattern at the current location of the cursor. The user can drag the pattern by keeping the MIDDLE BUTTON depressed and moving the mouse. Depressing the RIGHT BUTTON allows the user to iterate or drag the pattern under certain colors. Colors in the pattern which have higher color table index values than the colors of the image at the current location of the pattern, do not replace those colors in the image. This gives the illusion of painting under a certain layer of colors. The user clicks the LEFT BUTTON when he/she is done. The test window disappears, and the user is returned to the command menu.

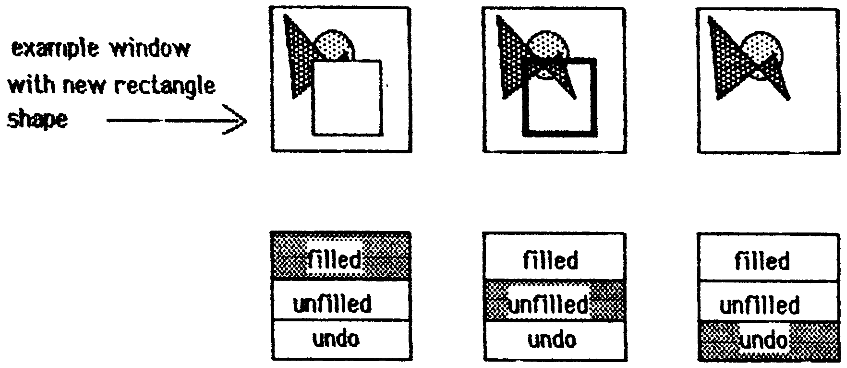
Creating Shapes

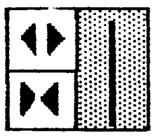


POP UPs are automatically displayed and the cursor constrained in it when the routine needs the user to make a decision.



FILLED/UNFILLED/UNDO POP UP allows the user to indicate the outcome of the shape just created. The user is able to see the options before he/she makes a decision. When the cursor lies in the "filled" button, the shape becomes filled with a default color. When it lies in the "unfilled" button, the outline of the shape is displayed, and when it lies in the "undo" button the new shape disappears. The user makes a choice by placing the cursor in one of the buttons and clicking the MIDDLE BUTTON on the mouse.





LINE THICKNESS POP UP allows the user to adjust the thickness of a line. Placing the cursor in the opened arrows button, and clicking the MIDDLE BUTTON thickens the line. Placing the cursor in the closed arrows button, and clicking the MIDDLE BUTTON slims the line. As the user is adjusting the line width, the line in the right-hand box displays the current line thickness. A click of the LEFT BUTTON finalizes the line width and returns the user back to the routine. FREEHAND, LINE, unfilled BOX and unfilled POLYGON use this pop up.

FREEHAND allows the user to hand draw a shape. The user first selects a color for the pen from the color bar (see section on color bar described in **COLOR TABLE ROUTINES** for description of user's options in the color bar). He/she then selects the line width for the pen using the LINE THICKNESS POP UP. The user keeps the MIDDLE BUTTON depressed when drawing. A click of the LEFT BUTTON completes the routine and returns the user back to the command menu. A click of the RIGHT BUTTON erases the just drawn shape and returns the user back to the command menu.

BOX allows the user to create a rectangle shape using a rubberband rectangle. The user selects two diagonal corners by locating each with the cursor and clicking the MIDDLE BUTTON. After the user selects the second corner, the FILLED/UNFILLED/UNDO POP UP appears. The user makes a selection. If he/she chooses UNFILLED, the LINE THICKNESS POP UP appears and he/she adjusts the thickness for the edges of the rectangle. The user selects a color for the filled or unfilled rectangle from the color bar. The user is then returned to the brush window and the routine is ready for the user to create another rectangle if he/she chooses to. A click of the LEFT BUTTON returns the user to the command menu.

POLYGON allows the user to create a polygon shape by using a series of connected rubberband lines. The user selects vertices by locating each with the cursor and clicking the MIDDLE BUTTON. A rubberband line becomes fixed between the current vertex selected and the previous vertex selected. The current vertex selected is an endpoint for the next rubberband line. When the user is done selecting vertices, he/she clicks the LEFT BUTTON. The routine automatically connects the last vertex with the first. Then the FILLED/UNFILLED/UNDO POP UP appears. The user makes a selection. If he/she chooses UNFILLED, the LINE THICKNESS POP UP appears and he/she adjusts the thickness for the edges of the polygon. The user selects a color for the filled or unfilled polygon from the color bar. The user is then returned to the brush window and the routine is ready for the user to create another polygon if he/she chooses to. A click of the LEFT BUTTON returns the user to the command menu.

CIRCLE allows the user to create a circle shape by selecting the center point and a point on the circumference. The user locates the center point with the cursor and clicks the MIDDLE BUTTON. He/she then moves the cursor, and the circle is continuously redrawn with the cursor lying on the circumference. The user sets the circle size by clicking the MIDDLE BUTTON. Then the FILLED/UNFILLED/UNDO POP UP appears. The user makes a selection. The user selects a color for the filled or unfilled circle from the color bar. The user is then returned to the brush window and the routine is ready for the user to create another circle if he/she chooses to. A click of the LEFT BUTTON returns the user to the command menu.

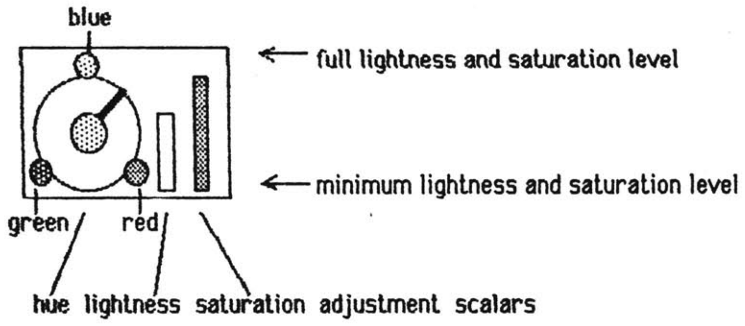
LINE allows the user to create a line by using a rubberband line. The user selects the two endpoints by placing the cursor at each location and clicking the MIDDLE BUTTON. Then the LINE THICKNESS POP UP appears and the user adjusts the thickness of the line. The user selects a color for the line from the color bar. The user is then returned to the brush window and the routine is ready for the user to create another line if he/she chooses to. A click of the LEFT BUTTON returns the user to the command menu.

COPY SHAPES allows the user to select a shape or group of shapes from a brush pattern window and iterate the item(s) into any brush pattern window. The user selects the source window by placing the cursor over the window and clicking the MIDDLE BUTTON. The cursor then selects the shapes by placing the cursor ontop of each shape and clicking the MIDDLE BUTTON. He/she clicks the RIGHT BUTTON when done selecting. Then the user selects the destination window by placing the cursor over the window and clicking the MIDDLE BUTTON. Each click of the MIDDLE BUTTON in the destination window creates a copy of the item(s) at the location of the cursor. The user clicks the LEFT BUTTON to return to the command menu. If the user wants to undo this current operation, he/she clicks the RIGHT BUTTON and the copied items will disappear. This will also return him/her to the command menu.

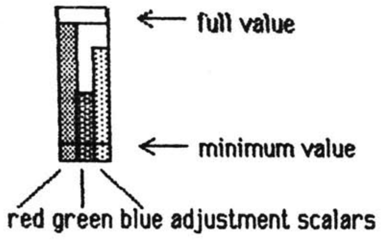
Color Table Routines

WINDOWS	SHAPE	COLORS	TRANSFORM	DISK

POP UPs

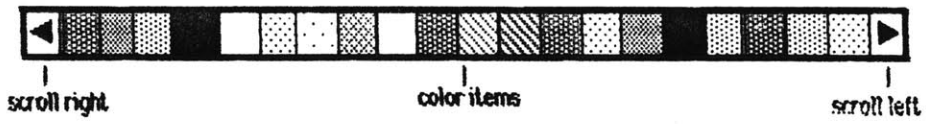

blue
full lightness and saturation level
minimum lightness and saturation level
green
red
hue lightness saturation adjustment scalars

HUE, LIGHTNESS, SATURATION COLOR ADJUSTMENT POP UP allows the user to edit a color item in the color table by modifying the hue, lightness and saturation values of the color. The user selects this method of editing by placing the cursor over the color to be modified and clicking the LEFT BUTTON. The pop up then appears and the cursor is constrained inside it. The user adjusts the hue scalar by placing the cursor along the hue dial and clicking the MIDDLE BUTTON. The pointer inside this dial will adjust itself to the new hue location. To adjust either the lightness or saturation levels, the user places the cursor at the respective bar and clicks the MIDDLE BUTTON. The top of the bar will adjust itself to the cursor location. As the scalars are being adjusted, the solid circle in the middle of the hue dial is constantly displaying the adjusted color. When the user is finished editing he/she clicks the LEFT BUTTON.



RED, GREEN, BLUE COLOR ADJUSTMENT POP UP allows the user to adjust a color item in the color table by modifying its red, green, and blue components. The user selects this method of editing by placing the cursor over the color to be modified and clicking the RIGHT BUTTON. The pop up then appears and the cursor is constrained inside it. To adjust the red, green, or blue values, the user places the cursor at the respective bar and clicks the MIDDLE BUTTON. The top of the bar will adjust itself to the cursor location. As the scalars are being adjusted, the color item in the color table is constantly displaying the adjusted color. When the user is finished editing he/she clicks the LEFT BUTTON.

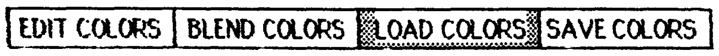
COLOR BAR



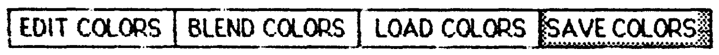
The cursor is constrained inside the color bar whenever a routine needs the user to select a color. There are 210 color items available to the user, however only 20 are displayed at a time. While in the color bar, the user can edit each color item using either of the methods described above. To select a color for a function, the user places the cursor over a color item and clicks the MIDDLE BUTTON. The user can also bring to view color items which are not currently displayed by selecting either of the arrow buttons. Placing the cursor over the left arrow button and clicking the MIDDLE BUTTON scrolls the items to the right and displays the next adjacent item to the left. Clicking the LEFT BUTTON over this arrow replaces the items in the color bar with the next 20 adjacent items to the left. Placing the cursor over the right arrow button and clicking the MIDDLE BUTTON scrolls the items to the left and displays the next adjacent item to the right. Clicking the RIGHT BUTTON over this arrow replaces the items in the color bar with the next 20 adjacent items to the right.

EDIT COLORS allows the user to access the color bar and edit any color items in it. He/she places the cursor over the color item to be modified and selects one of the color editing methods described above. To return to the command menu, the user clicks the MIDDLE BUTTON. (note: color items can always be edited when the cursor is constrained in the color bar. The cursor is placed in there whenever a routine needs the user to select a color. This routine gives the user a means to edit colors without having to select a function which needs a color value.)

BLEND COLORS interpolates between two selected color items in the color bar. The in between color items differ in even gradations from one selected color to the other selected color. The user selects each color item by placing the cursor over it and clicking the MIDDLE BUTTON. If the user modifies a color with one of the above methods, the modified color is automatically selected. The user is returned to the command menu when the interpolation is completed.

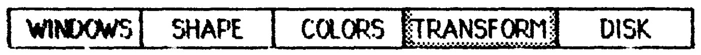


LOAD COLORS allows the user to completely replace the current active color table with one that was previously saved onto disk. When this command is selected, the PC monitor will prompt the user for the name of the color table to be loaded. The user enters the name through the keyboard.

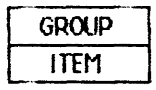


SAVE COLORS allows the user to save the complete color table onto disk. When this command is selected, the PC monitor will prompt the user for the name of the color table to be saved. The user enters the name through the keyboard.

Pattern Editing / Manipulating Shapes

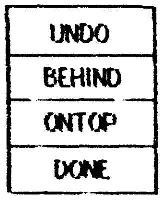


POP UPs

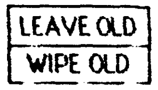


GROUP/ITEM POP UP prompts the user to indicate whether a single shape or group of shapes is to be manipulated. The user places the cursor over the desired choice and clicks the

MIDDLE BUTTON. If ITEM is selected, the user selects the shape by placing the cursor over it and clicking the MIDDLE BUTTON. If GROUP is selected, the user selects each shape by placing the cursor over it and clicking the MIDDLE BUTTON. To indicate that the group selection is done, he/she clicks the RIGHT BUTTON. Clicking the LEFT BUTTON will abort grouping and return the user back to the command menu.



UNDO/BEHIND/ONTOP/DONE POP UP appears after an item or group of items have been moved, copied, or scaled. Selecting UNDO puts the pattern back to its previous state. Selecting BEHIND allows the user to place the item or group of items behind a selected shape in the window. The user selects the shape with the MIDDLE BUTTON. Selecting ONTOP allows the user to place the item or group of items in front of a selected shape in the window. The user selects the shape with the MIDDLE BUTTON. This pop up keeps reappearing until the user selects DONE. Selecting DONE returns the user back to the window. All choices in the pop up are selected by placing the cursor over it and clicking the MIDDLE BUTTON.



LEAVE OLD/WIPE OLD POP UP appears after an item or group of items has been selected for scale. Selecting LEAVE OLD leaves the source item(s) in place. Selecting WIPE OLD erases the source item(s) from the window. The user places the cursor over the desired choice and clicks the MIDDLE

BUTTON.

MOVE	COPY	SCALE	DELETE

MOVE allows the user to relocate a shape or group of shapes inside its window. GROUP/ITEM POP UP appears. The user makes shape(s) selection accordingly. The user moves the shape(s) by moving the mouse. When he/she is satisfied with the new placement he/she clicks the MIDDLE BUTTON. The UNDO/BEHIND/ONTOP/DONE POPUP appears. The user makes a selection and then acts accordingly as described above. After the user has selected DONE, the cursor is placed inside the window and the routine is ready for the user to select more item(s) to move. At this point, a click of the LEFT BUTTON returns the user to the command menu.

MOVE	COPY	SCALE	DELETE

COPY allows the user to copy a shape or group of shapes inside the window. GROUP/ITEM POP UP appears. The user makes shape(s) selection accordingly. The user moves the copied shape(s) by moving the mouse. When he/she is satisfied with a location for the copied shape(s), he/she clicks the MIDDLE BUTTON. The UNDO/BEHIND/ONTOP /DONE POPUP appears. The user makes a selection and then acts accordingly as described above. After the user has selected DONE, the cursor is placed inside the window and the routine is ready for the user to select more item(s) to copy. At this point, a click of the LEFT BUTTON returns the user to the command menu.

MOVE	COPY	SCALE	DELETE

SCALE allows a user to pick a shape or group of shapes as source item(s) to be scaled. GROUP/ITEM POP UP appears. The user makes shape(s) selection accordingly. Then the user uses a rubberband rectangle to indicate the new size and location of the modified shapes. Then the LEAVE OLD/WIPE OLD POP UP appears. The user makes a selection. The source shape(s) is/are then scaled to fit the new size at the new location. The UNDO/BEHIND/ ONTOP /DONE POPUP appears. The user makes a selection and then acts accordingly as described above. After the user has selected DONE, the cursor is placed inside the window and the routine is ready for the user to select more item(s) to scale. At this point, a click of the LEFT BUTTON returns the user to the command menu.

DELETE allows the user to erase shapes from a window. The user places the cursor over the shape and clicks the MIDDLE BUTTON. Clicking the RIGHT BUTTON will put back the just deleted shape. Clicking the LEFT BUTTON will return the user back to the command menu.

Brush Images on Disk

SAVE BRUSH allows a user to save a brush pattern onto disk. The user places the cursor over the desired pattern to save in the work surface and clicks the MIDDLE BUTTON. If the user chooses to quit the function, instead, a click of the RIGHT BUTTON will return him/her back to the command menu. The cursor is then placed inside the scrolling brush menu (see section on **THE IMAGE** in **Components of Paint Mode** to get further detail about the scrolling brush menu). Here he/she indicates whether the pattern should replace an existing one on disk or be added to the end of the list. If the user decides the latter, he/she places the cursor over the WHITE SLOT at the bottom of the brush menu and clicks the MIDDLE BUTTON. If the user decides to replace a pattern, he/she places the cursor over the image of that pattern and clicks the MIDDLE BUTTON. The following pop up appears:

OVERWRITE
RESELECT

The user makes a selection with a click of the MIDDLE BUTTON. If RESELECT is chosen, the cursor is placed back inside the brush menu for the user to select another slot.

SAVE BRUSH	ERASE BRUSH

ERASE BRUSH allows the user to erase a brush pattern from disk. The cursor is placed inside the scrolling brush menu where the user selects the brush pattern to be erased by placing the cursor over its icon and clicking the MIDDLE BUTTON (see section on **THE IMAGE** in **Components of Paint Mode** to get further detail about the scrolling brush menu). Then the following pop up appears to make sure the user wants to erase this pattern from disk:

FORGET IT
DO IT

The user places the cursor over the desired choice and clicks the MIDDLE BUTTON.

Components of Paint Mode

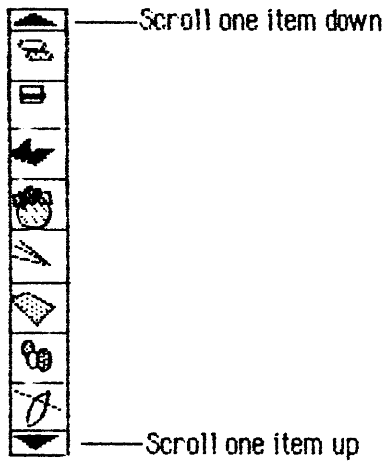
EDIT | USE | QUIT

This mode provides routines to generate images in the work surface.

The Image

IMAGE | COLORS

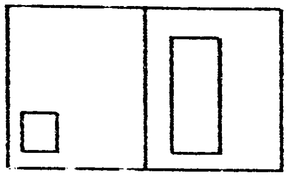
BRUSH IMAGE MENU

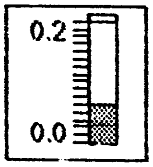


The cursor is constrained inside the brush image menu when a routine needs the user to select a brush pattern that has been saved on disk. The limit for the number of brush patterns that can be saved depends on the available memory on disk. This menu displays at a time eight of the saved brushes. To select a brush, the user places the cursor over its icon and clicks the MIDDLE BUTTON. While in this menu, the user can bring into view brushes which are not currently displayed by selecting either of the arrow buttons. Placing the cursor over the top arrow button and clicking the MIDDLE BUTTON scrolls the items down and displays the next item above. Placing the cursor over the the bottom arrow button and clicking the MIDDLE BUTTON scrolls the items up and displays the next item below.

PAINT allows the user to create an image by iterating a brush pattern over the work surface. The cursor is placed in the brush image menu where the user makes a selection as described above. Upon selection of the brush, the cursor is placed inside the work surface. To iterate the pattern ontop of all colors in the work surface, the user clicks the MIDDLE BUTTON. Dragging is possible if the user keeps the button depressed. To iterate the pattern "beneath" certain colors in the image, the user clicks or keeps depressed the RIGHT BUTTON. In this type of application only colors in the brush whose index into the color table are lower then the index of colors at the current location of the brush on the work surface are copied. A click of the LEFT BUTTON returns the user to the command menu.

DYNAMIC DRAW allows the user to create an image by iterating a brush pattern, which the user can control the change in size and opacity of, over the work surface. The cursor is placed in the brush image menu where the user makes a selection as described above. Then the following pop ups appear:





The user first adjusts the two extreme sizes that the brush can grow and shrink between using the pop up on the left. Two rectangles are provided to represent the sizes. The user places the cursor over either side of the pop up and adjusts a size by placing the cursor at a new location for the upper right corner of the rectangle and clicking the MIDDLE BUTTON. When the adjustments are done, the user clicks the LEFT BUTTON. Then the pop up on the right appears. Placing the cursor at a new level of the slider and clicking the MIDDLE BUTTON adjusts the change in transparency and opacity factor. When the adjustments are done, the user clicks the LEFT BUTTON. The cursor is then placed inside the work surface. To iterate the pattern ontop of all colors in the work surface, the user clicks the MIDDLE BUTTON. Dragging is possible if the user keeps the button depressed. To iterate the pattern "beneath" certain colors in the image, the user clicks or keeps depressed the RIGHT BUTTON (see PAINT for more detail). Clicking the LEFT BUTTON TWICE at various instances toggles between the brush growing, shrinking, or staying the same size during the next application of it. Clicking the LEFT BUTTON and then the MIDDLE BUTTON at various instances toggles between the brush becoming more opaque, becoming more transparent, or staying at the same opacity level during the next application of it. A click of the LEFT BUTTON and then a click of the RIGHT BUTTON returns the user to the command menu.

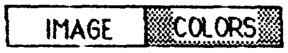
SCALE PIC allows the user to select a source area and a destination area in the image in which the portion of the image in the source area is copied and scaled into the destination area. The user first selects the source area using a rubberband rectangle. He/She selects two diagonal corners by placing the cursor over each location and clicking the MIDDLE BUTTON. This same process is done to select the destination area. Then the source image is copied and scaled into the destination area and the user returned to the command menu.

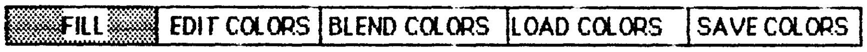
CLEAR PIC allows the user to clear the work surface to a single color. The cursor is constrained inside the color bar until the user selects a color (see section on color bar described in **COLOR TABLE ROUTINES** in **Components of Brush Design Mode** for description of user's options in the color bar).

SAVE PIC allows the user to save the current image in the work surface onto disk. When this command is selected, the PC monitor will prompt the user for the name of the image to be saved. The user enters the name through the keyboard.

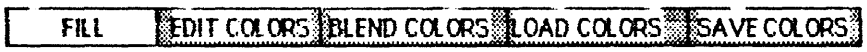
LOAD PIC allows the user to load onto the work surface an image which has been previously saved onto disk. When this command is selected, the PC monitor will prompt the user for the name of the image to be loaded. The user enters the name through the keyboard.

Color Table Routines





FILL allows the user to select a region of the image to fill with a different color. The user locates a point on the image with the cursor and clicks the MIDDLE BUTTON. Then he/she selects the fill color from the color bar. The routine will replace all the neighboring points which have the same color as the selected point with the fill color. It terminates when all adjacent points to the filled area are of a different color than the source point. The user can stop the fill before its done by clicking the RIGHT BUTTON. After a region has been filled, the cursor is placed inside the work surface so the user can continue to select more regions to fill. A click of the LEFT BUTTON returns the user to the command menu.



See **Color Table Routines** in **Components of Brush Design Mode** for descriptions.

Footnotes:

[1]Stephen Bann, Experimental Painting (New York, 1970), p. 10.

[2]Mai-Mai Sze, The Tao of Painting (Princeton, 1963), vol. 2, p. 27.

[3]Mai-Mai Sze, The Tao of Painting (Princeton, 1963), vol. 2, p. 328.

[4]Mai-Mai Sze, The Tao of Painting (Princeton, 1963), vol. 2, pp. 349-351.

[5]E. H. Gombrich, The Heritage of Apelles (Ithaca, 1976), plate 82.

[6]Benoit Mandelbrot, The Fractal Geometry of Nature (San Francisco, 1982), plate C13.

Bibliography:

Bann, Stephen, Experimental Painting. New York: Universe Books, 1970.

Baumgartner, Victor, Graphic Games: From Pattern to Composition. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1983.

Bearden, Romare and Carl Holty, The Painter's Mind. New York: Garland Publishing, Inc., 1981.

Benthall, Jonathan, Science and Technology in Art Today. New York: Praeger Publishers, 1972.

Brooks, Leonard, Oil Painting: Basic and New Techniques. New York: Van Nostrand Reinhold Company, 1971.

Dondis, Donis A., A Primer of Visual Literacy. Cambridge, Mass.: The M.I.T. Press, 1973.

Foley, James D. and Andries Van Dam, Fundamentals of Interactive Computer Graphics. Reading, Mass.: Addison-Wesley Publishing Company.

Fournier, Alain and Don Fussell, "Computer Rendering of Stochastic Models," Communications of the ACM, vol. 25, no. 6 (June, 1982), 371-384.

Gombrich, E. H., Art and Illusion. Princeton: Princeton University Press, 1972.

Gombrich, E. H., The Heritage of Apelles. Ithaca: Cornell University Press, 1976.

Gombrich, E. H., The Story of Art. Oxford: Phaidon Press Limited, 1972.

Hardwick, Martin, "Graphical Data Structures," Computer Graphics, vol. 15, no. 4 (December 1981), 376-404.

Lewis, John-Peter, "Texture Synthesis for Digital Painting," Computer Graphics, vol. 18, no. 3 (July, 1984), 245-252.

Lieberman, Henry, "How to Color in a Coloring Book," Computer Graphics, vol. 12, no. 3 (August, 1978), 111-116.

Mandelbrot, Benoit B., The Fractal Geometry of Nature. San Francisco: W. H. Freeman and Company, 1982.

Mandelbrot, Benoit. B., Fractals: Form, Chance, and Dimension. San Francisco: W. H. Freeman and Company, 1977.

McDermott, J., "Geometrical Forms Known as Fractals Find Sense in Chaos," Smithsonian, vol. 14, no. 9, 1983.

Newman, William M. and Robert F. Sproull, Principles of Interactive Computer Graphics. New York: McGraw-Hill Book Company, 1979.

Norton, A. "Generation and Display of Geometric Fractals in 3-D," Computer Graphics, vol. 16, no. 3 (July, 1982), 61-61.

Owen, Peter, Painting; The Appreciation of the Arts/5. London: Oxford University Press, 1973.

Pool, Phoebe, Impressionism. New York: Oxford University Press, 1979.

Rowley, George, Principles of Chinese Painting. Princeton: Princeton University Press, 1959.

Silbergeld, Jerome, Chinese Painting Style. Seattle: University of Washington Press, 1982.

Standish, Thomas A., Data Structure Techniques. Reading, Mass.: Addison-Wesley Publishing Company, 1980.

Sze, Mai-Mai, The Tao of Painting. Princeton: Princeton University Press, 1963.

Thompson, d'Arcy Wentworth, On Growth and Form. Cambridge: Cambridge University Press, 1952.

Whitted, Turner, "Anti-Aliased Line Drawing Using Brush Extrusion," Computer Graphics, vol. 17, no. 3 (July, 1983), 151-156.

Wolfram, Stephen, "Computer Software in Science and Mathematics," Scientific American, (Sept, 1984),. 188-203.

"ColorPaint, PCPaint, and PC Paintbrush." Popular Computing, (May, 1985), 98-102.

Acknowledgements:

I dedicate this thesis with love to my parents, Bertrand and Yu-Chen Mao, for their continuous support, guidance, and love, and credit to them all accomplishments that I have made or will make in my life.

I would like to thank my sisters, Pei-Pei, Sandi, and Lisa, for their support and humor, my friends, Steve Simon and Jan Cutro, for their companionship, as well as for looking after my well being during my toiling hours of work, and all my friends for their conversation, encouragement, and humor. I am grateful to Frank Chu, Scott Lichtman, and Dimitry Rtischev for their contribution to the software development of this thesis project, to Brian Swift for helping me select a thesis topic, and to my colleagues, Susan Wascher, Liz Rosenzweig, and Debra Adams for sharing and pulling through rough times together. I would like to thank Professor Muriel Cooper and Ron MacNeil for providing me the opportunity to merge my interests in art and computer science.

The research for this thesis project was partly supported by HELL GMBH of Kiel, Germany.