Using X11 to Demonstrate Visual Effects

by

Frankie Kim-Tak Sun

An essay
presented to the University of Waterloo
in fulfillment of the
essay requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, 1989

Author's Declaration

I hereby declare that I am the sole author of this essay.

I authorize the University of Waterloo to lend this essay to other institutions or individuals for the purpose of scholarly research.

_____

I further authorize the University of Waterloo to reproduce this essay by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

Record of Use

The University of Waterloo requires the signatures of all persons using or photocopying this essay. Please sign below, and give address and date.

Acknowledgements

My sincere thanks go to my supervisors, Kellogg S. Booth and William B. Cowan, for their invaluable help and guidance in completing this essay, and to Scott Flinn, who was kind enough to be the student reader for this essay. Last but not least, I would like to take this opportunity to express my deep gratitude to my family and friends for their constant encouragement, without which I would never have been able to finish this essay.

Financial support for the research reported in this essay was provided by operating, strategic, and infrastructure grants from the Natural Sciences and Engineering Research Council of Canada, and by Digital Equipment of Canada.

Abstract

We describe six different variants of visual effects reported in the vision literature. We begin with an introduction to the X Window System Version 11 (X11), which is the environment in which the demonstration of the visual effects takes place. For each visual effect, we discuss how it appears to an observer and the program that is used for its demonstration. The visual effects are then discussed in terms of what they teach about the display of information within a windowed environment and the tools that the visual effects themselves provide for improving that environment. Next, specific implementation techniques used in the demonstration programs are discussed. Finally, the performance of X11 is analyzed in terms of our implementation experience and suggestions are made for future window system architectures based on our experience.

Contents

1 Introduction 1

1.1 Overview 1

1.2 X Window System Version 11 2

1.2.1 Objectives of X11 3

1.2.2 X11 Structure 4

1.2.3 X11 Concepts 4

1.2.4 User Interface 11

1.3 Conclusion 11

2 Visual Effects 12

2.1 Background 12

2.2 Contrast 14

2.2.1 Black/White Contrast 14

2.2.2 Yellow/Gray Contrast 16

2.3 Equiluminance 16

2.3.1 Nulling of Apparent Motion 16

2.3.2 Pinwheel 18
2.4 Growth and Decay of Sensation 23
2.4.1 Benham's Disc 23
2.5 Gestalt Organization 26
2.5.1 Interwindow Interference 31

3 Demonstration Programs **33**

3.1 Overview 33
3.1.1 Windows 33
3.1.2 User Interface 34
3.2 Black/White Contrast 34
3.2.1 Main Window 36
3.2.2 Control Panel 36
3.2.3 Results 36
3.3 Yellow/Gray Contrast 39
3.3.1 Main Window 39
3.3.2 Control Panel 39
3.3.3 Results 39
3.4 Nulling of Apparent Motion 42
3.4.1 Main Window 42
3.4.2 Control Panel 42
3.4.3 Status Window 43

3.4.4 Results 43
3.5 Pinwheel 45
3.5.1 Main Window 46
3.5.2 Control Panel 46
3.5.3 Status Window 46
3.5.4 Results 50
3.6 Benham's Disc 51
3.6.1 Main Window 51
3.6.2 Control Panel 51
3.6.3 Status Window 54
3.6.4 Results 54
3.7 Interwindow Interference 55
3.7.1 Window One and Main Window 55
3.7.2 Control Panel 55
3.7.3 Status Window 57
3.7.4 Results 57
3.8 Significance of Visual Effects 59

4 Implementation **62**

4.1 Structure of Programs 62
4.2 Lookup Table Animation 64
4.3 Compression of Lookup Table Updates 65

4.4 Use of Plural Graphics Primitives 67
4.5 Window Border Manipulation 68
4.6 Dragging 68
4.7 Portability 71

5 Problems and Solutions **72**

5.1 Screen Refresh Synchronization 72
5.2 Drawing Speed 73
5.3 Number of Color Cells and Number of Planes 74
5.4 Interruption of Animation 74
5.5 Processing for Lexical and Syntactic Events 76
5.6 Temporal Control over Animation 76
5.7 Workstation Display Artifacts 77

A Location of Programs **80**

List of Tables

2.1 How the Visual Effects Are Grouped 13

List of Figures

1.1 Visual Effect vs. Non-Visual Effect 2

2.1 Black/White Contrast 15

2.2 Yellow/Gray Contrast 17

2.3 Color Bar A 19

2.4 Color Bar B 20

2.5 Color Bar C 21

2.6 Color Bar D 22

2.7 Pinwheel 24

2.8 Benham's Disc 25

2.9 Small Closed Region 27

2.10 Closeness 28

2.11 Closedness 29

2.12 Simplicity 29

2.13 Symmetry 30

2.14 Good Continuation 30

2.15 Interwindow Interference 32

3.1 Typical Screen Layout of Demonstration 35

3.2 Main Window for Black/White Contrast 37

3.3 Control Panel for Black/White Contrast 38

3.4 Main Window for Yellow/Gray Contrast 40

3.5 Control Panel for Yellow/Gray Contrast 41

3.6 Main Window for Nulling of Apparent Motion 43

3.7 Control Panel for Nulling of Apparent Motion 44

3.8 Status Window for Nulling of Apparent Motion 45

3.9 Main Window for Pinwheel 47

3.10 Sine Waves vs. Square Waves 48

3.11 Control Panel for Pinwheel 49

3.12 Status Window for Pinwheel 50

3.13 Main Window for Benham's Disc 52

3.14 Control Panel for Benham's Disc 53

3.15 Status Window for Benham's Disc 54

3.16 Window One and Main Window for Interwindow Interference 56

3.17 Control Panel for Interwindow Interference 58

3.18 Status Window for Interwindow Interference 59

4.1 Relationship Between Physical and Logical Sectors 65

4.2 Compression of Changes to Benham's Disc upon Rotation 66

5.1 Movement of Lines in Pinwheel 78

5.2 Red Leakage 79

Chapter 1

Introduction

1.1 Overview

In this essay, we discuss the results of using the X Window System Version 11 (X11) to display images which we call visual effects. We define visual effects as optical illusions which are false images produced by our visual system and/or our mind. As an example, two lines of equal length can be placed so that one appears longer than the other (Figure 1.1 (a)), which is a visual effect. It is not a visual effect if the two lines appear the way that they actually are (Figure 1.1 (b)). We believe that visual effects influence the way that information is perceived on a computer display and that an understanding of the parameters of real-time human performance is necessary for knowing how information is perceived on a computer display.

The visual effects described in this essay are displayed by programs which run on workstations in the X11 environment. We refer to these programs as *demonstration programs*. By the same token, we refer to the execution of such programs as *demonstrations*. The demonstration programs have been designed for two purposes.

1. To examine the suitability of X11 for supporting interactive applications un-

In (a), the two lines are actually equal in length but the one on top appears shorter, constituting a visual effect. In (b), the two lines which are equal in length actually do appear to be so.

Figure 1.1: Visual Effect vs. Non-Visual Effect

der the assumption that the visual effects being studied are related to display tasks that occur in real applications and the assumption that those tasks pose a reasonable performance measure for X11.

2. To test the hypothesis that common visual effects occur in windowed workstation environments, to test the hypothesis that the visual effects influence the way that information is perceived, and to provide a testbed for analyzing techniques that ameliorate the consequences of those effects.

1.2 X Window System Version 11

All of the visual effect demonstration programs described in this essay were developed at the Computer Graphics Laboratory at the University of Waterloo on Digital Equipment Corporation (DEC) VAXstations (II/GPX, 3200, etc.) running the X Window System Version 11 (X11) [11, 15]. X11 is a descendent of the W Window System [21] and the X Window System Version 10 (X10) [21]. Released in 1987, X11

has quickly become a *de facto* standard on many workstations. In this chapter, we present only enough X11 fundamentals for the reader to understand the implementation of the demonstration programs. Further details on X11 can be obtained in the X Window System Protocol, Version 11 [19] and other references [21].

1.2.1 Objectives of X11

X11 was developed which such goals as *network-transparency* and *portability* in mind. Network-transparency allows an X11 application to run on a remote CPU which is connected to a workstation and to use that workstation for input and output. This allows for flexibility in choosing whichever CPU is most suitable for the application while using the workstation for display. We tried this option out at the Computer Graphics Laboratory, where we have a DEC VAX 8600 mainframe connected via Ethernet to DEC VAXstations running X11. The results obtained under this arrangement are not as good as we can get by running the demonstrations locally on a workstation, although the difference in performance is not great. Neither arrangement is as good as what we had hoped for, especially for the programs which produce animated sequences (discussed in Section 5.4).

Secondly, X11 provides portability. This allows an X11 application to run on workstations of different makes and hardware architectures provided that they all support the *X11 network protocol*. In fact, portability was one of the main reasons behind the development of the X Window System at the Massachusetts Institute of Technology (MIT) in 1984. The portability of the demonstration programs will be discussed in Chapter 4.

1.2.2 X11 Structure

X11 has a layered architecture with the *base window system* at its foundation. Using the X11 network protocol the base window system is able to interact with the low-level programming interface called *Xlib*, which is a C language subroutine package. We used Xlib to create the demonstration programs in this essay. Higher level *X Toolkits* are available to hide some of the low-level intricacies of Xlib and to provide a more uniform 'look and feel' for different applications for which a consistent style is desired.

1.2.3 X11 Concepts

X11 is based on the *client-server model*. The *X11 server* is a program which resides in a workstation. *Clients* are application programs. Communication between the server and clients is in the form of *messages* which flow between them. The format and interpretation of these messages are specified in the X11 network protocol, which is the formal definition of the X11 window system.

In order for the client-server communication to take place, a client must first establish a *display connection* to the X11 server. The display connection is a logical network circuit between the client and the server and hides from the user the intricacies of the client-server connection, be it remote or local, thus accomplishing the objective of being network-transparent. There are basically two types of messages which pass through the display connection. First, we have *requests*, which are messages originating from the application destined for the X11 server. Requests can be *one-way* or *round-trip*. One-way requests are buffered to reduce communication overhead and are used by applications which want the X11 server to complete some task, such as drawing a line on the screen. Round-trip requests are used to obtain information from the server, such as getting the current cursor position on the screen.

The application blocks while it waits for the reply from the server. As we will see later, round-trip requests can be quite detrimental to the performance of programs which produce animated sequences.

The other type of message is the *event*. Events are of different types and are sent by the X11 server to an application as a means of notifying the latter of user-induced actions such as key presses, and application-induced changes to the workstation display. Application programs solicit the types of event that they want for a window by setting the appropriate bits in the event mask for the window. Events are placed in an event queue upon receipt by an application program in the order they arrive and await processing by an application. The usual way of dealing with events is for an application to read an event off the event queue (thereby removing it from the queue) and, depending on the type of event it is, process it accordingly. The above procedure is then repeated until the application encounters the event signaling the termination of execution.

Another important idea in X11 is that of *resources*. They are useful entities that an application manipulates, e.g. *windows, graphics contexts* (*GC's*), *fonts, color maps, pixmaps* and the *cursor*. Resources reside in the workstation. Some of them, like graphics contexts, require little memory, with many of them being used by an application at the same time. Other resources, like the color map, are very limited in their nature, and have to be used accordingly. For the demonstration programs to be described in this essay, the most widely used resources are windows, graphics contexts and color maps.

Windows

Windows are rectangular screen areas used for input and/or output operations. They follow the *desktop metaphor* [12], which allows us to think of windows as pieces of

paper lying on a desk surface, subject to certain restrictions, while at the same offering some flexibilities that objects on real desktops do not share.

Often an application program uses more than one window at the same time. To keep track of this potentially large number of windows, X11 organizes the windows as a hierarchy. At the top of this hierarchy is the *root window*, which covers the entire screen area. The root window can have child windows, each of which can in turn have its own child windows. There can be more than one window at the same level except at the top level. So the root window may have two child windows, which are considered siblings. An application must specify an existing parent window for every window that it creates. Thus, all application windows are descendents of the root window.

The first step in displaying a window on the screen is to create it. This is followed by *mapping*, which actually puts the window on the screen. However, under the X11 desktop metaphor, a mapped window may be obscured either fully or partially by other windows 'on top' of it, in which case the application or the user might want to move the window around to make it completely *viewable*. Note that the stacking order is independent of the parent/child tree.

The *characteristics* of a window, specified at creation time, cannot be changed during its lifetime and include its *class* (InputOutput or InputOnly), *depth* (number of bits per pixel) and *visual characteristics* (how pixel values are turned into colors on the screen). In addition, a window has a number of *attributes*, which can be changed by the application after the initial specification. Attributes are of the following types: *input, appearance, gravity* and *backing store*.

The origin of a window is its upper-right corner. Each window coordinate corresponds to a pixel on the screen. Before an application can create a window, it must specify the origin of the new window in terms of the coordinate system of its parent.

This information, along with the width, height and border width, make up the *geometry* of the new window. A window's *configuration* refers to its geometry plus its stacking order relative to its siblings.

X11 allows for a special client application, called a window manager, to organize and rearrange windows of other applications. Usually, an application uses the appropriate Xlib calls to give the window manager *hints* on things like constraints on the size of a window and a label for its icon. Also, the window manager usually allows an application's window to be moved and resized by the user, and an application should be designed to accept such actions and respond suitably.

The demonstration programs make extensive use of windows. The *main window* contains the actual images that make up a visual effect. Another window, which we call the *control panel*, contains controls for manipulating the contents of the main window. Some demonstration programs also display a *status window* which shows the values for some parameters relevant to the visual effect.

Graphics

The parameters for any graphics drawing into a window are set in a *graphics context (GC)*. A graphics context contains rendering attributes such as foreground and background colors and a line width. It provides for a convenient way to specify the drawing attributes for an image and reuse the same collection of attributes later for another image without having to re-specify them.

X11 offers geometric graphics primitives for *points, lines, rectangles, arcs* and *shapes*. An application specifies the screen coordinates of such images in terms of pixels, and the X11 server computes which pixels to draw into along the way. Besides windows, X11 allows drawing into *pixmaps*, which are invisible, off-screen raster areas. All windows and pixmaps are regarded as *drawables* in X11, and any drawing primitive

simply specifies the identifier for the drawable where the drawing should go.

Every demonstration program in this essay makes extensive use of the X11 graphics primitives for drawing into the main window images which, when manipulated, give the visual effect that we are seeking. Drawing of the graphics primitives goes through a five-stage *graphics pipeline* in X11. The five stages are:

1. **Pixel Selection** — X11 computes which pixels must be altered in the window in order to display the primitive object.
2. **Patterning** — X11 applies to the pixels selected in stage one foreground and background colors as specified in the graphics context.
3. **GC Clipping** (Optional) — X11 optionally *clips* or discards all pixels outside a region specified in the graphics context.
4. **Window Clipping** — X11 clips all pixels which lie outside the boundaries of the window or pixmap into which an application draws.
5. **Raster Output** — X11 combines, bitwise for every unclipped pixel, colors generated by the graphics primitive with the colors already present. Usually the new colors simply replace the old ones.

Color

X11's color capabilities are used extensively in the demonstration programs, although we are more or less limited to the *visual class* offered by the workstations. Color is specified in terms of a *pixel value*, which is an index to a *color cell* in a *color map*. The actual contents of a color cell determine what color it gives. The number of *planes* determines the size of the color map. For example, an 8-plane workstation has 8-bit pixel values and thus at most 256 color cells in a color map. The more planes a

workstation has, the more color cells are accessible. The X11 VAXstations that are used for the development of the demonstration programs have at most 8 planes.

A visual class is a strategy for translating a pixel value into RGB triplets. There are six visual classes under X11:

- **PseudoColor** — Each pixel value is an index to a color cell in a color map. For example, an 8-plane workstation can simultaneously display at most $2^8 = 256$ colors from a gamut of 256^3 (256 intensities each for R, G and B) possible colors. PseudoColor requires a relatively small number of bits/planes while still allowing a rich gamut. The demonstration programs in this essay run on workstations which support this visual class.

- **DirectColor** — Each pixel value is three separate indices to the R, G and B channels of a color map. For example, a 24-plane (8 planes each for R, G and B) workstation can display any color out of a gamut of 256^3 possible color combinations at a time. DirectColor requires many more planes than PseudoColor to be effective, and are thus expensive. Images which require a large number of colors at a time, such as smoothly shaded colored objects, will find DirectColor much more desirable than PseudoColor.

- **GrayScale** — Each pixel value is an index to a color cell, except that only one of the R, G and B channels of the color cell is used. It is assumed that the R, G, and B values are equal for the color cell. GrayScale is used on multiplane workstations which have monochrome displays.

- **StaticColor** — Like PseudoColor, except that the contents of a color cell cannot be changed.

- **TrueColor** – Like DirectColor, except that the contents of a color cell cannot be changed.

1.2.4 User Interface

Besides the keyboard, X11 workstations support a *pointer device*, usually a mouse. The mouse is used to control the position of the tracking cursor on the screen. This, along with the buttons that come with the mouse, allow the user to interact with an application in a wide variety of ways, such as clicking, double-clicking and dragging of an object on the screen. Although X11 provides an application with the mechanism for this array of interaction techniques, it is the application's responsibility to come up with the appropriate interaction policy.

Input from the keyboard and the pointer device is in the form of events. An application should solicit and interpret such events as necessary in order to provide a good user interface.

1.3 Conclusion

We have presented some fundamental concepts of the X Window System Version 11. This knowledge is helpful in understanding the implementation of the demonstration programs that display the visual effects.

In the next chapter, we will discuss the different types of visual effects that the demonstration programs are to produce.

- **StaticGray** – Like GrayScale, except that the contents of a color cell cannot be changed.

X11 also provides three strategies for dealing with color maps. They are:

- **Shared Color Map** — X11 returns pixel values for colors with symbolic names or specific RGB values. Most of these color cells are preallocated and fixed with their RGB values, allowing more than one application to display the same color at the same time. However, an application can also specify the RGB values for a color and have X11 allocate a color cell for it as long as the color map is not full. The shared color map strategy is thus a method for conserving limited color resources.

- **Standard Color Map** — X11 provides a number of standard color maps which contain color cells with preloaded R, G and B primary values. Associated with each standard color map is a data structure containing information which an application needs in order to compute a pixel value given an RGB triplet. This strategy is useful for the display of smoothly shaded objects and is mostly implemented under the DirectColor or TrueColor visual classes.

- **Private Color Cells** — An application sends requests to X11 for the allocation of color cells. The application can subsequently load RGB triplets into the color cells and reference them for the display of colors. The contents of these color cells can be changed any time by an application, with any such change reflected on the screen immediately. By having different objects on the screen reference different private color cells, and by carefully changing the contents of these color cells dynamically, an application is able to achieve *lookup table animation* [18, 22, 23]. The demonstration programs in this essay which produce animated sequences are made possible using this technique (discussed in Sections 4.2 and 4.3).

Chapter 2

Visual Effects

2.1 Background

X11 offers a wide variety of graphics primitives which are suitable for the reproduction of two-dimensional images. Additionally, X11 allows for interaction techniques using a mouse. We would like to challenge X11 with the reproduction and manipulation of some visual effects that are representative of display tasks that occur in many workstation-based applications.

Visual effects abound in our everyday life. They range from those which are routinely exploited in commercial advertising to ones of which nobody is yet aware. In recent years, with the popularity of the television medium and computers, we have gotten more and more exposure to the cathode ray tube (CRT) display, be it a common television screen or a sophisticated computer workstation display. Moreover, with the availability of color on many of these CRT's, we are constantly in need of more information on how to effectively make use of their capabilities in order to convey ideas as clearly as possible. Careful study and understanding of visual effects could help us determine which images to avoid and how.

	Static	Dynamic
Contrast	Black/White Contrast Yellow/Gray Contrast	
Equiluminance		Nulling of Apparent Motion Pinwheel
Growth and Decay of Sensation		Benham's Disc
Gestalt Organization	Interwindow Interference	

Table 2.1: How the Visual Effects Are Grouped

The visual effects that are reproduced in the demonstration programs are of four types: *contrast, equiluminance, growth and decay of sensation* and *Gestalt organization*. Each visual effect is either *static* or *dynamic*. A static visual effect is defined as one which is produced by an image which does not change over time without user control. If a visual effect is produced by an image which changes over time without user control, then it is considered dynamic. The types of visual effects that we concentrate on in this essay cover a fairly range of display tasks, and we have chosen from each type one or two typical examples. Table 2.1 shows what visual effects are demonstrated and where they stand with respect to the two grouping schemes.

In this chapter, we will discuss each visual effect in its ideal form without implementation details. For each visual effect, we will describe what an observer sees, and how it works. Then in the next chapter, we will see how each visual effect is implemented by a demonstration program.

2.2 Contrast

The contrast effects that areas of different colors have on one another are probably some of the most common visual effects in our daily life [1, 7, 17]. We have chosen to concentrate on one type of contrast, namely *simultaneous contrast*, meaning effects stemming from areas contiguous in space and simultaneous in time, as opposed to *successive contrast*, meaning effects from areas which have the same physical location but are displayed one after another. Simultaneous contrast helps us to make out more clearly areas of different luminance (brightness) and chromaticities (colors).

Simultaneous contrast is explained by *lateral inhibition* caused by center-surround receptive field in the ganglion cells of the retina [8, 9, 14]. Simply put, lateral inhibition causes the retina regions illuminated by different patches of light to have inhibitory effects on one another. Two other related causes of simultaneous contrast include *induction* and *adaptation* [8]. Induction is the visual system's way of enhancing our perception of edges. Adaptation, on the other hand, is the method by which our visual system adjusts the gain of our vision in environments of different brightness, for example, when we step out of a movie theater into bright sunshine. The interaction between these two factors along with other lesser known factors give rise to a wide variety of visual effects. In this essay, we will look at two of these, a well-known black/white contrast effect and a yellow/gray contrast effect.

2.2.1 Black/White Contrast

The effect [1] that we try to show here takes place when we have two patches of the same gray, one of which is surrounded by a relatively large black area and the other surrounded by a relatively large white area (Figure 2.1). Looking at both gray patches, one notices that the gray patch surrounded by the black area appears quite a bit brighter than the one surrounded by the white area.

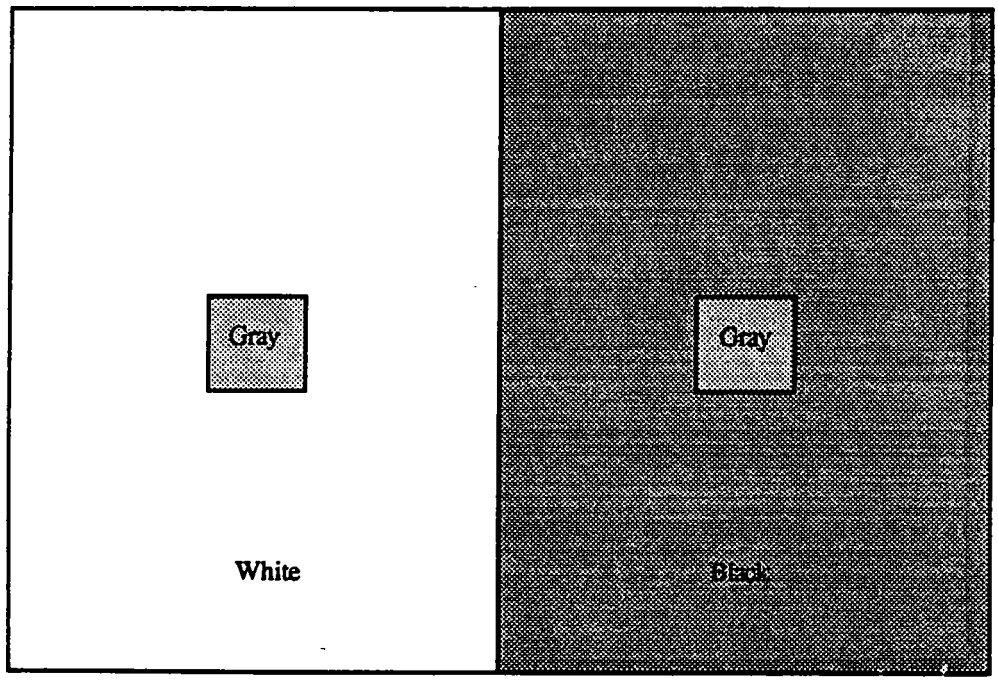


Figure 2.1: Black/White Contrast

2.2.2 Yellow/Gray Contrast

A color derivative of the black/white contrast is the yellow/gray contrast [1]. We have two X-shaped areas connected to each other at the top by a strip of the same width. The X on the left lies within a yellow area, while the X on the right lies within a gray area. The two X's, including their connecting strip, all have the same color, which is an equal mixture of the surrounding yellow and gray (Figure 2.2). By adjusting the relative luminance (but not the chromaticity) of the yellow and gray so that they have the same brightness perceptually, we get a *reversed ground* effect, in which each X takes on the color of the opposite surrounding. This effect demonstrates that simultaneous contrast is not confined to luminance, but exists for purely chromatic contrasts as well.

2.3 Equiluminance

We try to demonstrate two different visual effects which deal with the *equiluminance* of two colors, in this case red and green. Equiluminance is defined as two or more colors having the same perceived brightness. First, using the correct animation sequence, we can null the apparent motion of rectangles in a color bar sequence. In a related visual effect, we see what a rotating pinwheel looks like when portions of it are at equiluminance.

2.3.1 Nulling of Apparent Motion

The apparent motion takes place in a color bar made up of rectangles [2, 6, 13]. Two such rectangles of the same color adjacent to each other make up a square in the color bar. The color bar has four frames which are displayed one after another at the same physical spot. We will look at four different versions of the color bar, all

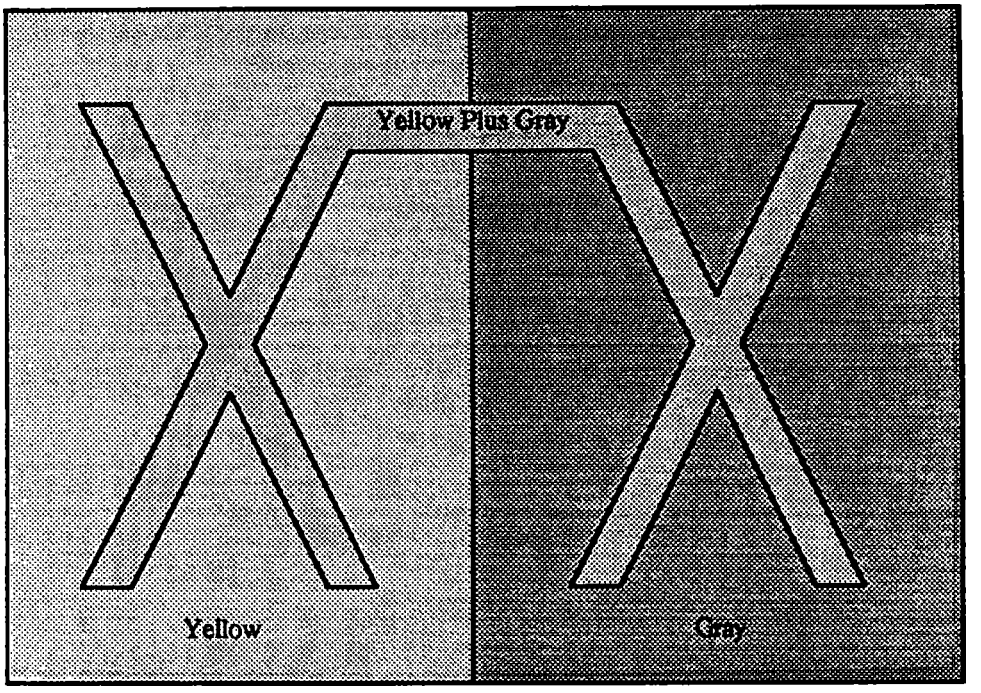


Figure 2.2: Yellow/Gray Contrast

of which have the same first and third frames. We start off with color bar A (Figure 2.3). If we display this color bar, we should clearly see an apparent motion toward the left. Now consider color bar B (Figure 2.4). If we display this color bar, we should see an apparent motion toward the right. Now consider color bar C (Figure 2.5) in which the second and fourth frames are just gray. With this color bar, we have equal evidence to support leftward and rightward motion. We therefore conclude that there is only flickering — black squares becoming white squares and vice versa. Thus, the apparent motion as seen in color bars A and B are nulled in color bar C. In fact, we could get rid of the second and fourth frames of color bar C and still get the same effect. Now consider the last scenario — color bar D, and suppose that for the second and fourth frames the green is much darker than the red. We have essentially the color equivalent of color bar A in this case. The result of this sequence is an apparent motion toward the left. If, on the other hand, the green is much brighter than the red for the second and fourth frames, we should perceive an apparent motion toward the right. This is equivalent to color bar B. The nulling of apparent motion occurs in color bar D when the red and green in the second and fourth frames are equiluminous, thus equivalent to the gray in color bar C.

2.3.2 Pinwheel

A special case of the equiluminance effect discussed above is that exhibited by the rotating pinwheel (Figure 2.7). When the red and green portions of the rotating pinwheel are equiluminous, by looking at the center of the pinwheel, one should get the distinct impression that the outer red/green portion of the pinwheel is slowing down or is even not rotating at all relative to the inner white/black portion. The explanation for this phenomenon is that looking at the center of the rotating pinwheel results in the red/green portion being seen by our peripheral vision. While the periphery is good at detecting motion that has a luminance component (different brightness), it

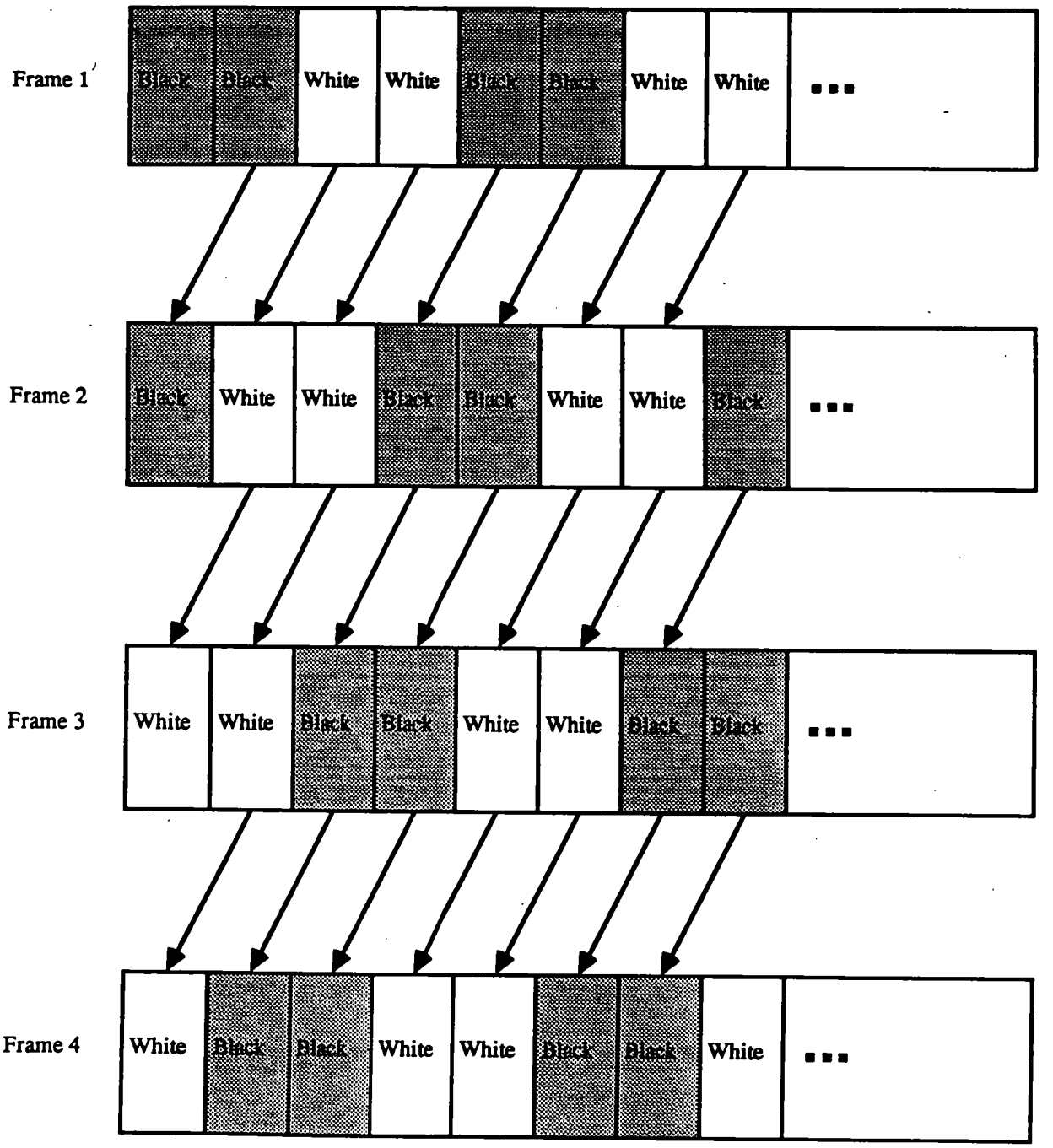


Figure 2.3: Color Bar A

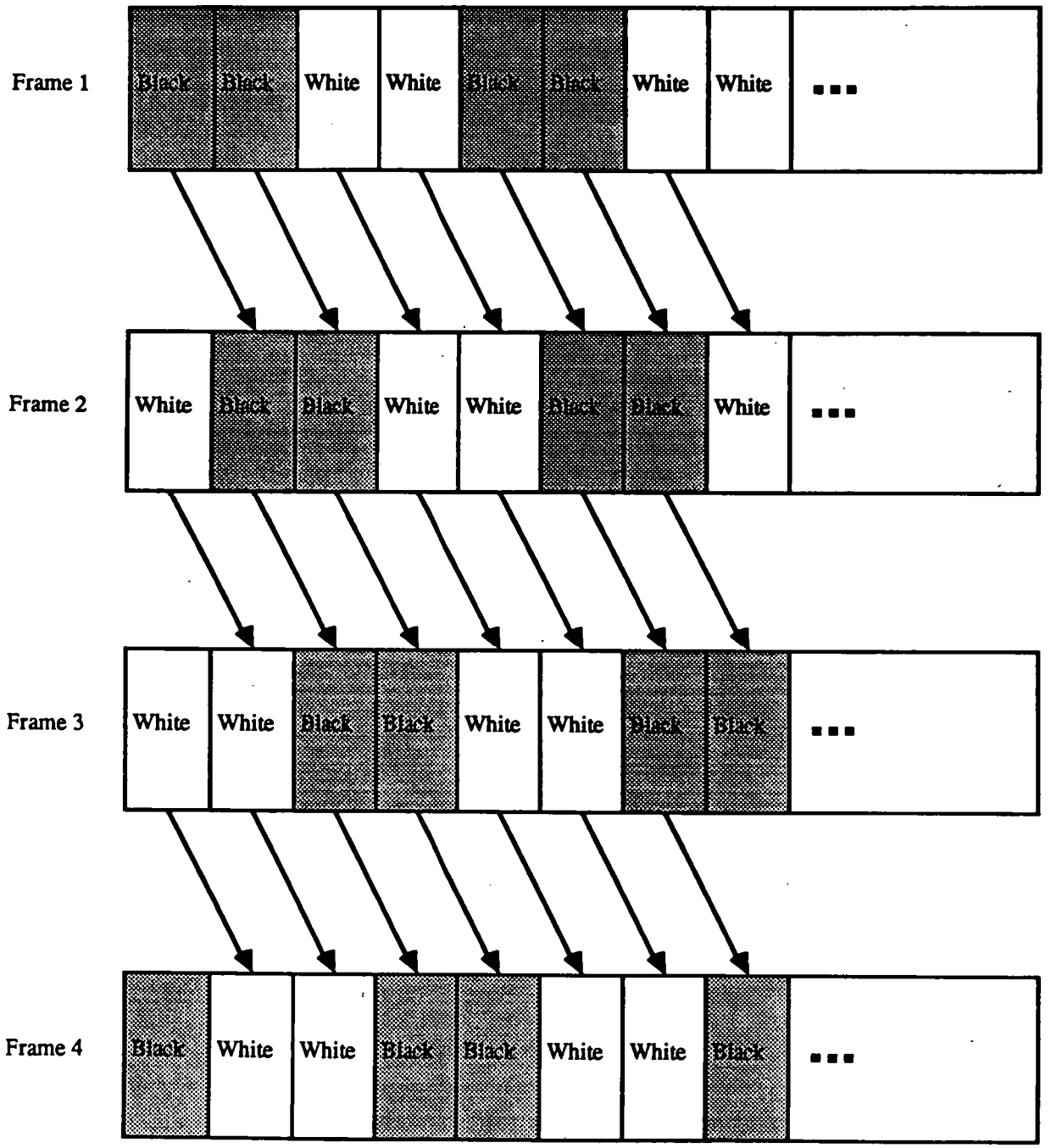


Figure 2.4: Color Bar B

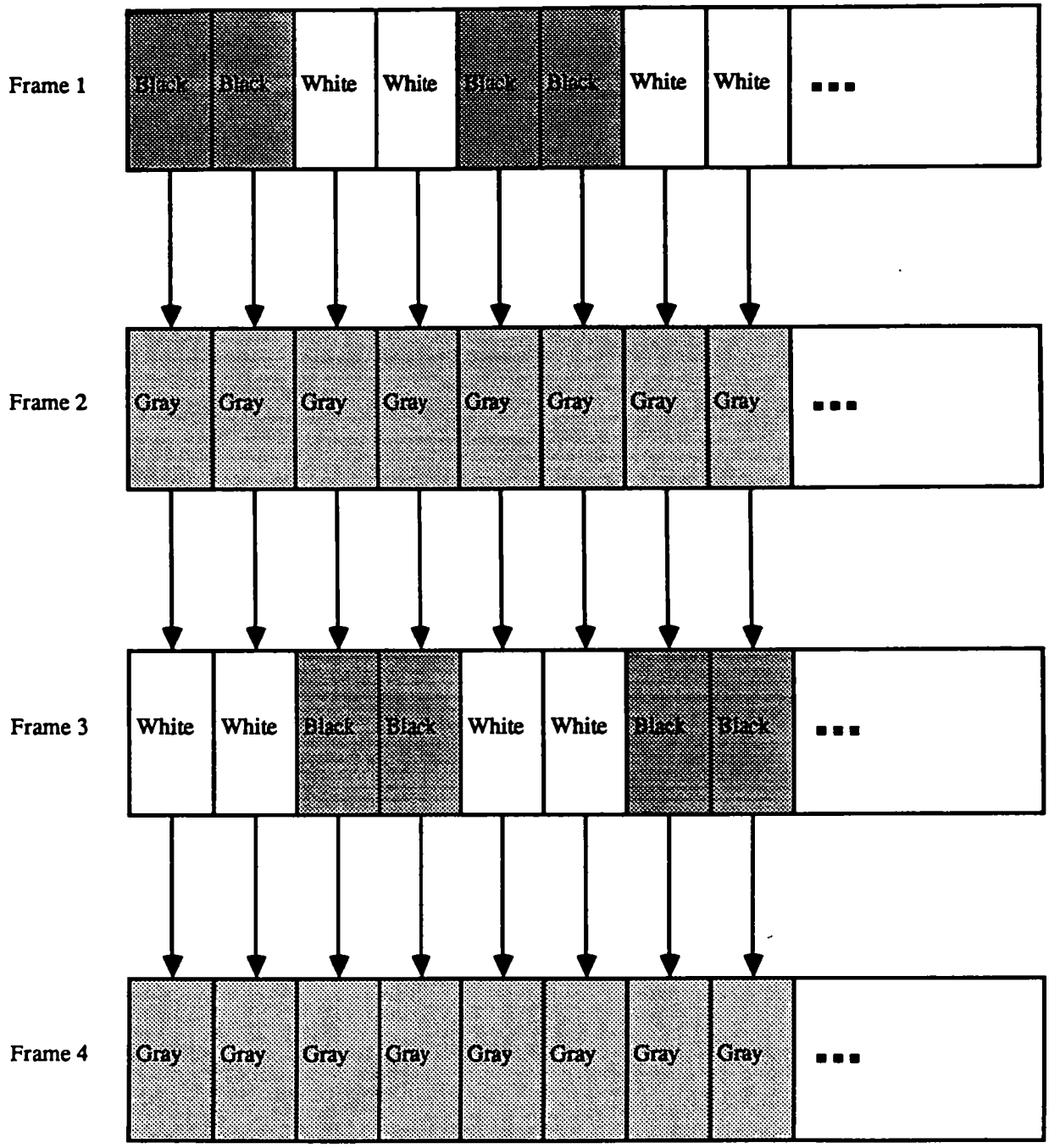


Figure 2.5: Color Bar C

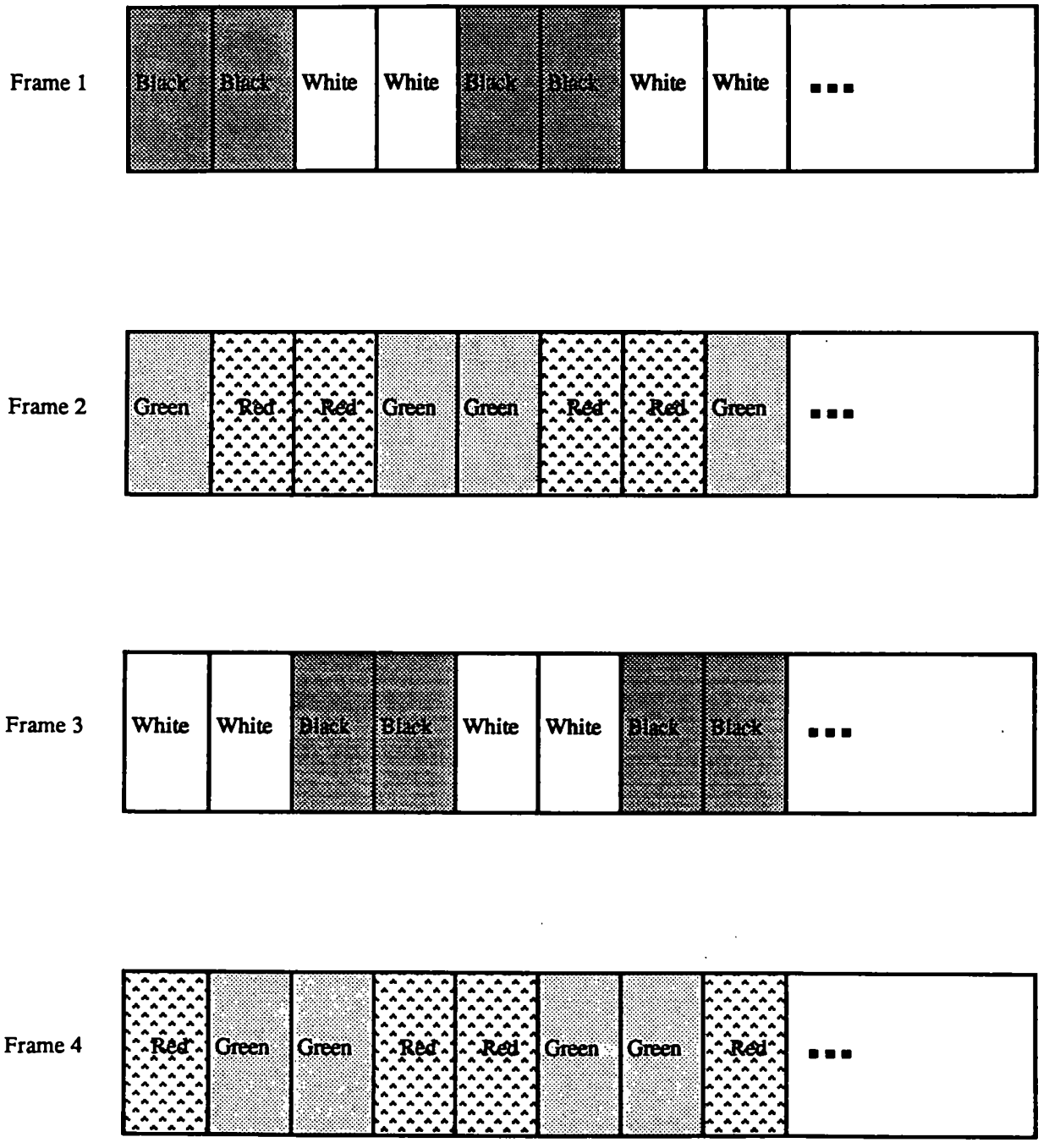


Figure 2.6: Color Bar D

is relatively poor at detecting the rotation of the red/green portion of the pinwheel when the red and the green have the same perceived brightness. This means that the perception of the outer rotation relies on the chromaticity difference between the red and green. However, the perception of color is highly localized in the fovea, with significantly less color acuity in the periphery. In contrast, rotation of the inner white/black portion is easily detected by our foveal vision because of a luminance component.

2.4 Growth and Decay of Sensation

This category of visual effects deals with illusions that arise as a result of *after-images* [1]. We see after-images after looking intently at bright objects. These after-images are of the same size and form as the original object, but vary in color. This is related to the persistence of vision, which begins when we first look at an object, at which time our visual sensation of that object grows. This sensation gradually levels off and begins to decline or decay when we take our vision off that object. One interesting example of this type of visual effect is Benham's Disc.

2.4.1 Benham's Disc

Benham's disc [1, 14, 17] is a disc divided into two equal portions, one white and one black, by a diameter. The white portion contains several groups of black concentric arcs (Figure 2.8). Upon rotation, one is able to see the different groups of arcs in different colors. The color of a group is dependent on the direction of rotation and its position relative to the black portion of the disc. For example, in clockwise rotation, group A will be reddish, while groups B and C will be bluish and greenish respectively. When the direction of rotation is counterclockwise, the colors of A and C will be reversed.

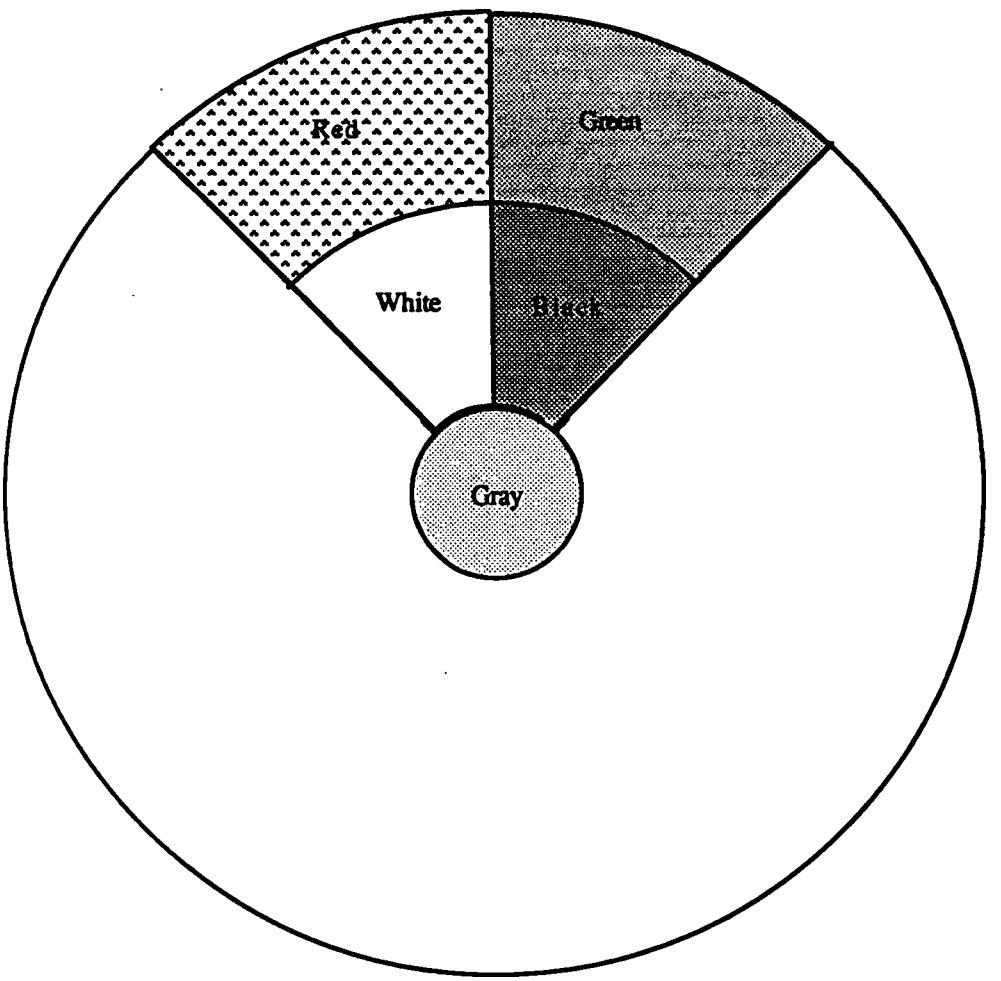


Figure 2.7: Pinwheel

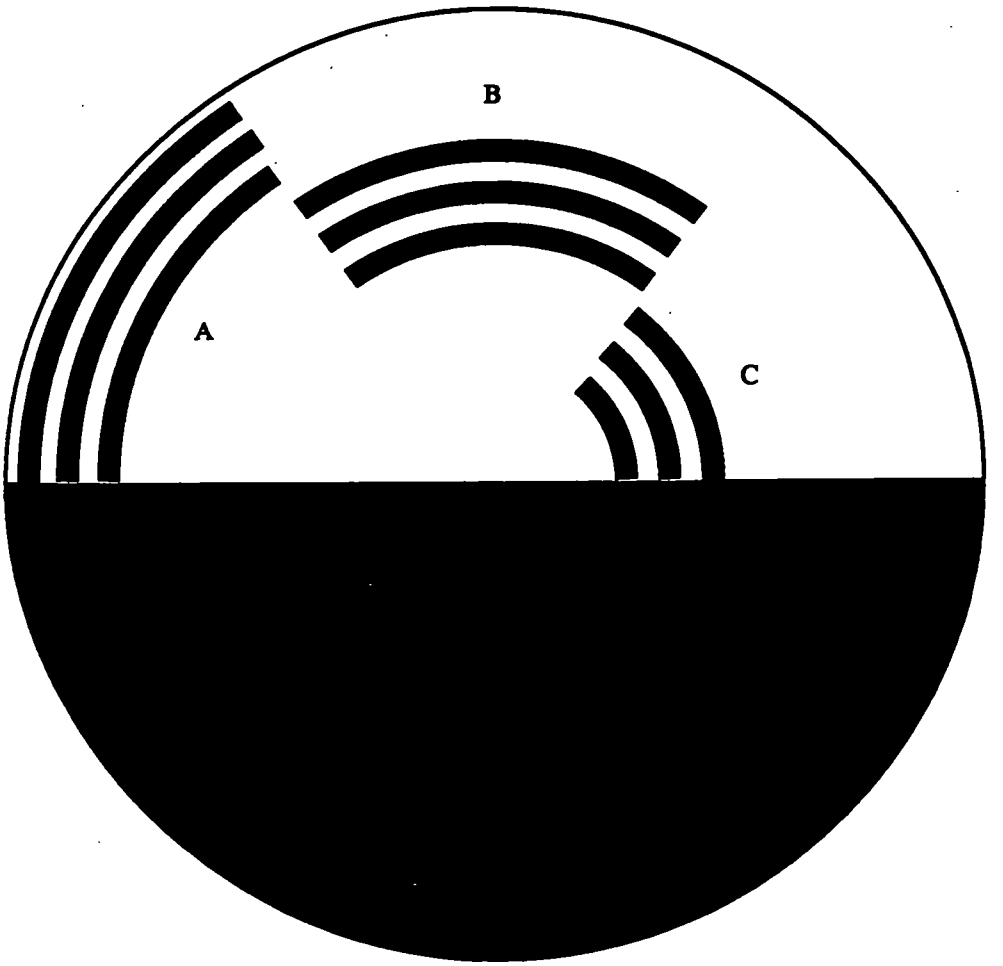


Figure 2.8: Benham's Disc

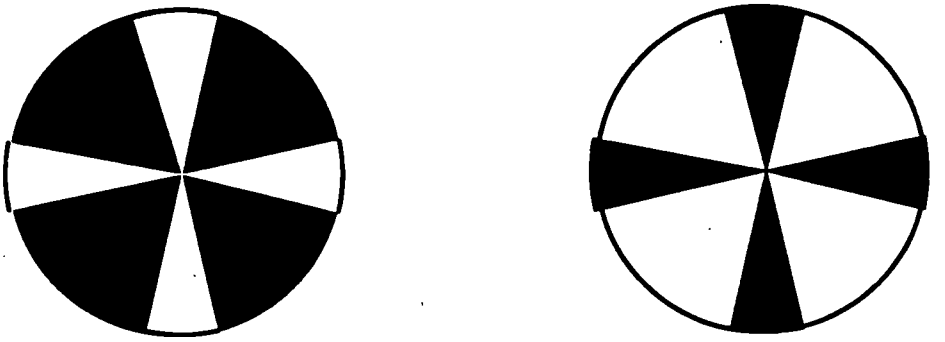
We still lack a detailed explanation for this visual effect, which takes place under monochromatic light as well [14]. It was suggested early on that the differences in the growth and decay rates of the various color sensations reflected from the white stimulus plays a part in the formation of this visual effect [17]. Moreover, it is believed that the time-varying activity produced in the optic nerves upon rotation of the Benham's disc is similar to that produced by the perception of real colored lights [14].

2.5 Gestalt Organization

This class of visual effect is based on the *Gestalt Laws of Organization* [10, 20]. These laws tell us that the following properties of and between different objects are conducive to our grouping them together:

- Small Closed Region (Figure 2.9)
- Closeness (Figure 2.10)
- Closedness (Figure 2.11)
- Simplicity (Figure 2.12)
- Symmetry (Figure 2.13)
- Good Continuation (Figure 2.14)

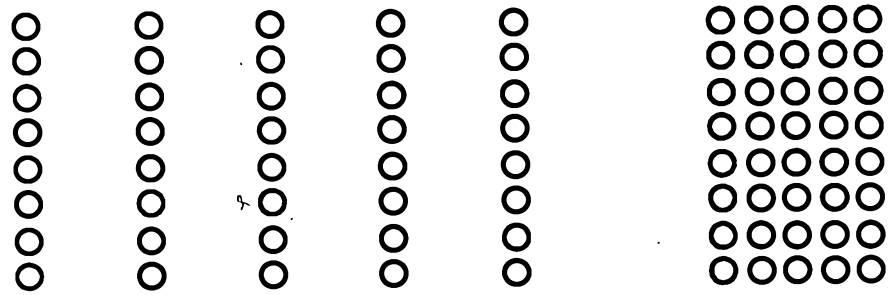
Gestalt organization visual effects are different from the other types of visual effects that we have discussed thus far — they are formed in the human brain rather than in the human visual system [10]. Because of their high-level nature, it is certainly possible that the Gestalt Laws of Organization are not universal but are a product of human experience, and as such, are subject to change over time and from person to person, and should be treated accordingly.



(a) (b)

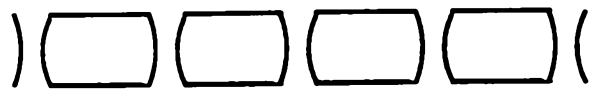
In (a), we see a white cross on a black background. In (b), we see a black cross on a white background. We don't see a fat black cross on a white background in (a) and we don't see a fat white cross on a blackground in (b). This is because a thin cross forms a small closed region.

Figure 2.9: Small Closed Region



In (a), we tend to see the circles as separate rows or columns. In (b), we tend to to see the circles as a rectangular entity. The reason is that in (b), the circles are much closer together.

Figure 2.10: Closeness

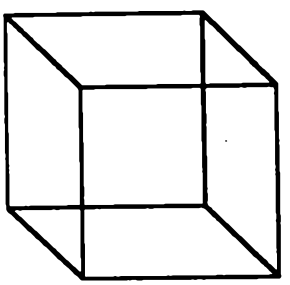


(a)

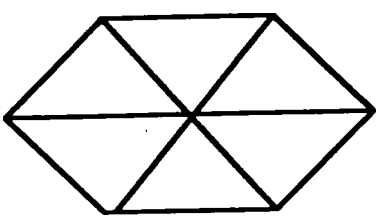
(b)

In (a), we tend to see four television screens. In (b), we tend to see four apple cores.

Figure 2.11: Closedness



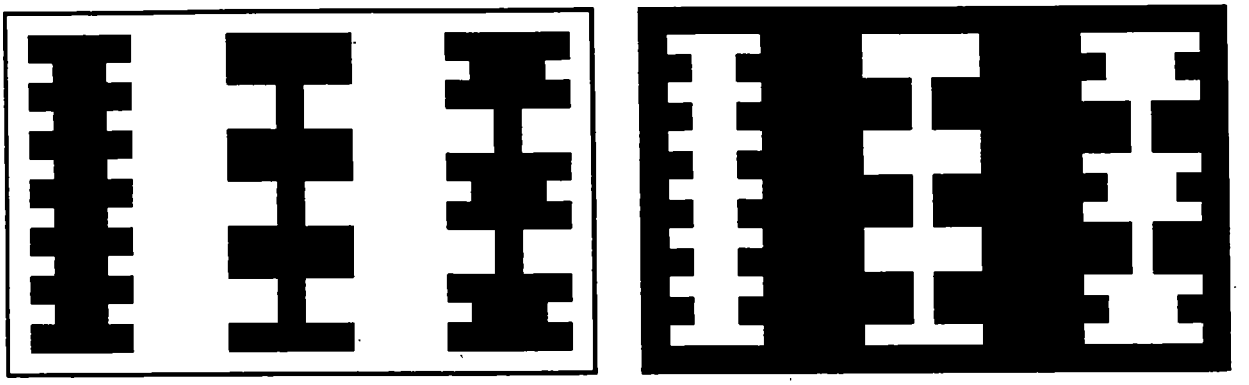
(a)



(b)

Both (a) and (b) are equally accurate (or inaccurate) views of the same wireframe cube, yet we are able to perceive (a) as a cube easier because it is simpler.

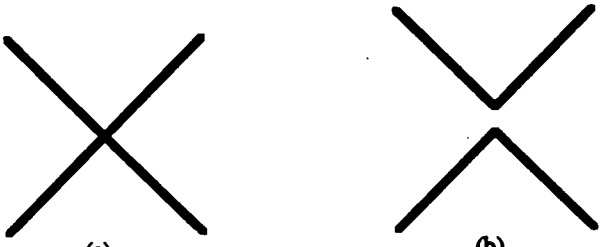
Figure 2.12: Simplicity



(a) (b)

In (a), we tend to see black columns on a white background. In (b), we tend to see white columns on a black background. The reason behind this is that the edges that make up a column are always symmetrical.

Figure 2.13: Symmetry



(a) (b)

We tend to think of (a) as two straight lines crossing each other. However, it is equally possible that the cross in (a) is made up of the two lines in (b), which do not have good continuation.

Figure 2.14: Good Continuation

2.5.1 Interwindow Interference

Suppose we have two windows side by side. If we have a line in each of these two windows in such a way that they are more or less of the same slope, width and color, and that they are aligned, there is a tendency for us to conclude incorrectly that it is really one single line that we are seeing passing underneath the borders of the two windows (Figure 2.15). This ideal Gestalt scenario meets all the influential properties described above except closedness.

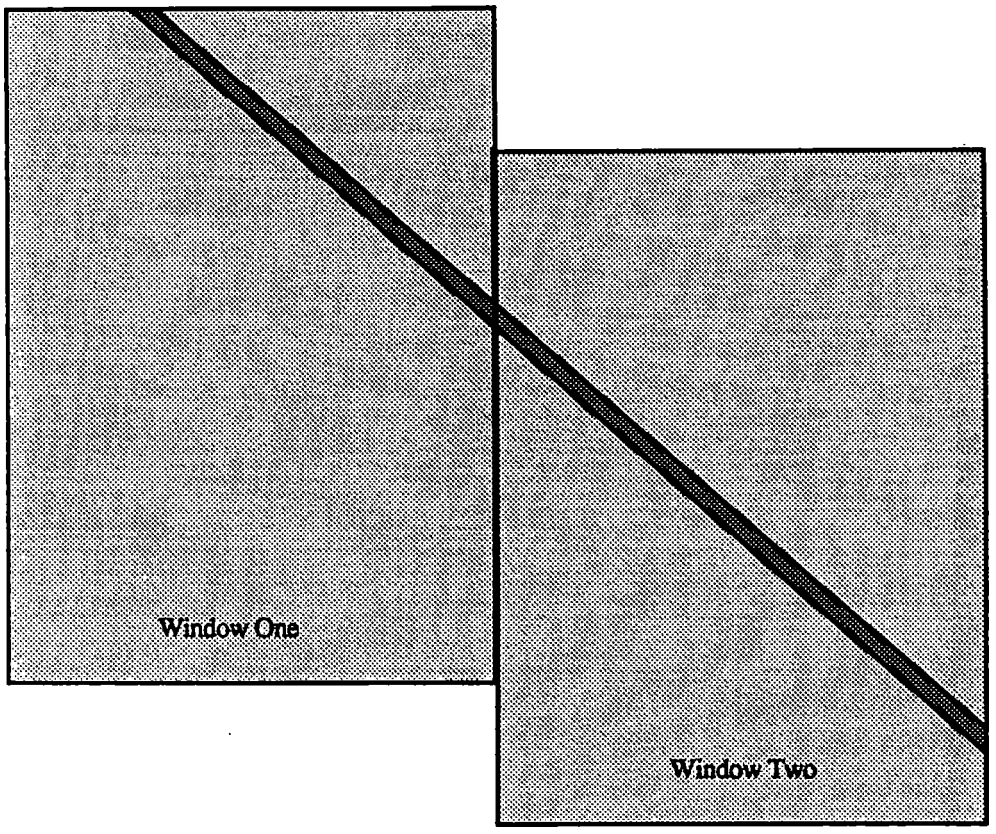


Figure 2.15: Interwindow Interference

Chapter 3

Demonstration Programs

3.1 Overview

Every visual effect described in the last chapter is implemented by a demonstration program. These demonstration programs run on X11 workstations which support the PseudoColor strategy. In addition, each program requires a certain number of private color cells to be allocated by the X11 server. The programs are all written in C code and they make use of X11's capabilities by issuing the appropriate Xlib calls.

In this chapter, we describe for each visual effect its demonstration program in terms of the windows it puts on the screen and what the intended visual effect actually looks like to the observer. To facilitate this discussion, we should take note of certain shared features among the demonstration programs.

3.1.1 Windows

First, the images which make up the visual effect reside in the *main window*, which, unless otherwise stated, is 820 pixels x 820 pixels (277 mm x 277 mm), and which

at a normal viewing distance of 60 cm, subtends a visual angle of 26 degrees. All demonstration programs display a second window called the *control panel*, which is 165 pixels (56 mm) wide. The control panel is the user interface which allows the user to manipulate the various parameters which are relevant to the reproduction of a visual effect. Some of the demonstration programs also display a third window which we refer to as the *status window* (165 pixels or 56 mm wide). The status window allows the user to determine at a glance the state of some relevant parameters. A typical layout of the screen with all three windows present is shown in Figure 3.1.

3.1.2 User Interface

All the demonstration programs are interactive. Once execution begins, the user interacts with the program via the control panel using a mouse attached to the workstation. All mouse buttons are treated equivalently. The controls in the control panel are of three types: *sliders, regular buttons*, and *extended buttons*. The user manipulates a slider by dragging its arrow pointer. Regular buttons provide only one option at a time, and when clicked on, carry out the action as labeled. An extended button is different from a regular button in that it offers more than one option at a time. The current option, chosen by a mouse click, is indicated by an arrow pointer. The extended button is used for the speed control.

3.2 Black/White Contrast

This static visual effect is implemented using a main window and a control panel.

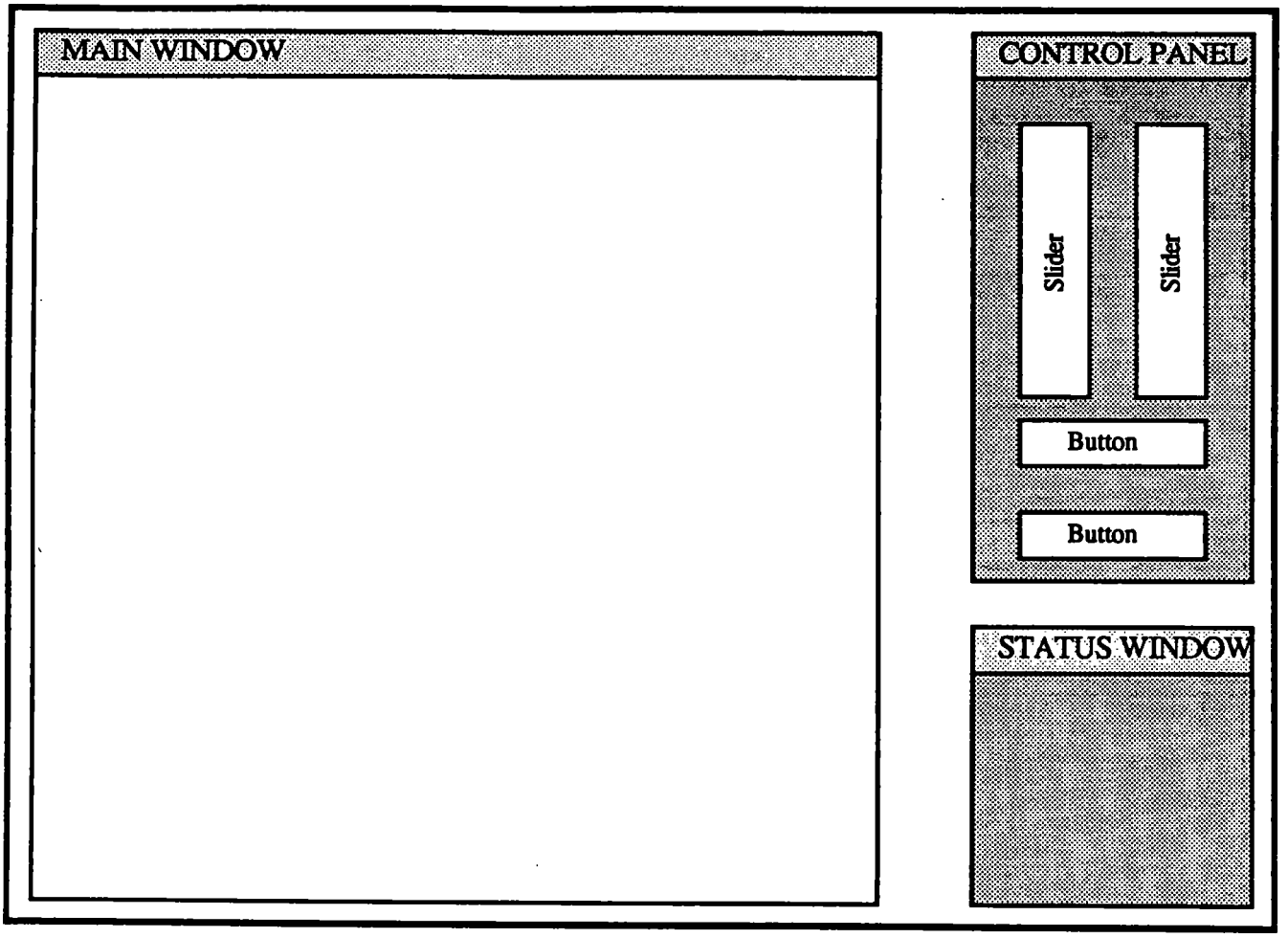


Figure 3.1: Typical Screen Layout of Demonstration

3.2.1 Main Window

The main window is divided up into two equal portions, left and right. The left portion is white in color (full RGB intensity), while the other is black (zero RGB intensity). Each portion contains a considerably smaller square (100 pixels or 34 mm wide, subtending a visual angle of 3.25 degrees) in the middle, the one on the left being fixed with a gray color somewhere in the range between black and white, the one on the right initially given a white color (Figure 3.2).

3.2.2 Control Panel

The control panel (Figure 3.3) consists of a slider for varying the gray intensity of the small right square in the main window within the complete range of full black to full white. The *surround* button allows the user to toggle the main window between a default *normal* state and a *comparison* state. In the regular state, the main window is as shown in Figure 3.2. In the comparison state, the surrounds are replaced by the colors of the small squares. The *quit* button allows the user to terminate execution of the program.

3.2.3 Results

The objective here is to match the brightness of the square on the right with that of the small square on the left under the influence of the black and white surrounds. The surround button in the control panel allows the user to tell at once how different the two colors really are even though they may look the same with the surrounds present. Conversely, the user can match the two colors with the surrounds off, and see how much they differ with the surrounds back on. And despite the fact that the visual effect which is reproduced is quite widely known, people seem to be more convinced

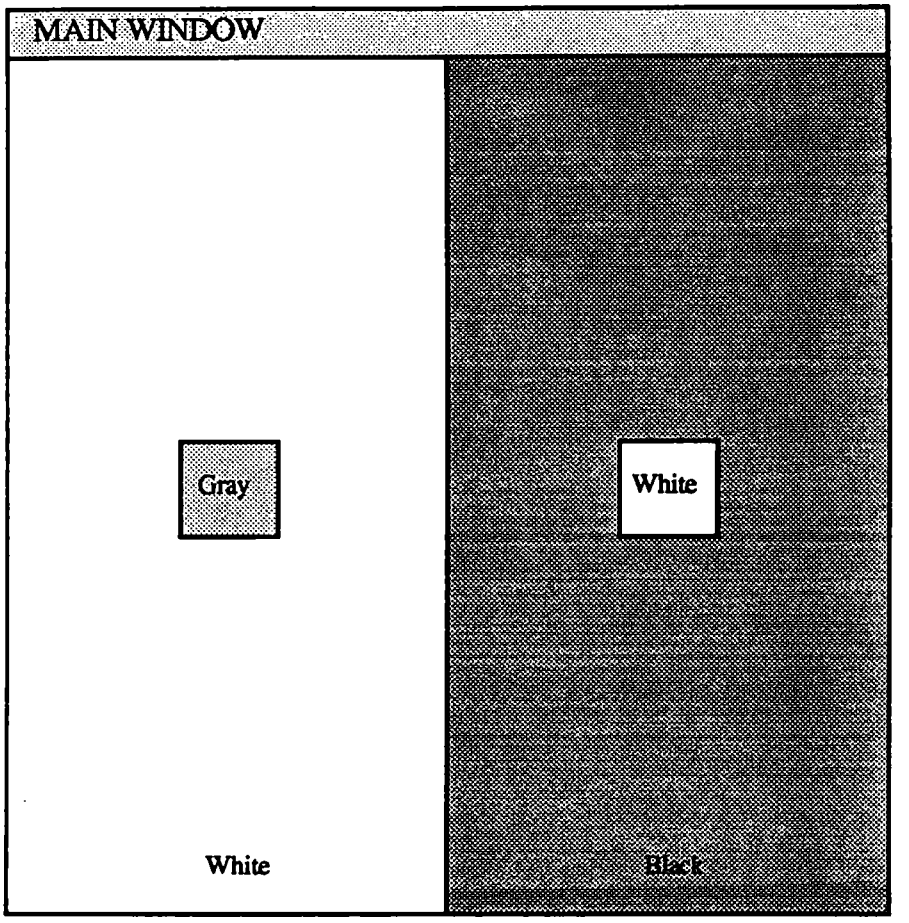


Figure 3.2: Main Window for Black/White Contrast

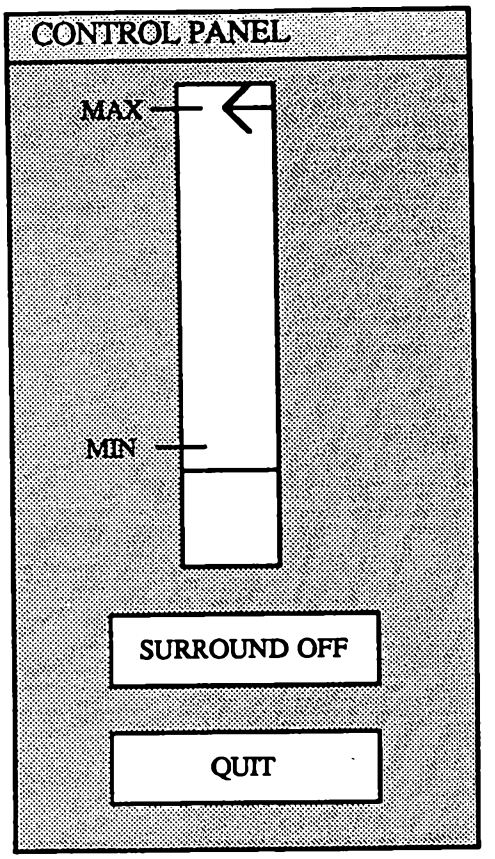


Figure 3.3: Control Panel for Black/White Contrast

than ever of its presence after using the demonstration program and are genuinely impressed with how it affects luminance judgement.

3.3 Yellow/Gray Contrast

This static visual effect is implemented with a main window and a control panel.

3.3.1 Main Window

The main window (Figure 3.4) consists of two adjacent X's connected by a strip of the same width at the top. The X on the left sits in an area initially of a full yellow intensity. The X on the right sits in an area initially of a middle gray intensity. The X's and their connector are of the same color, which is an equal mixture of the surrounding yellow and gray.

3.3.2 Control Panel

Two sliders are provided in the control panel (Figure 3.5). One controls the yellow intensity of the left half of the main window. The other controls the gray intensity of the right half of the main window. Since the X's and their connecting strip get their color from the yellow and gray surrounds, any manipulation of the sliders is reflected in the X's and their connecting strip as well. The *quit* button allows the user to exit the program.

3.3.3 Results

The objective of this demonstration is to adjust the brightness of the yellow and gray so that when they are of the same brightness perceptually (equiluminous), each of the X's will be the color of the background on the opposite side.

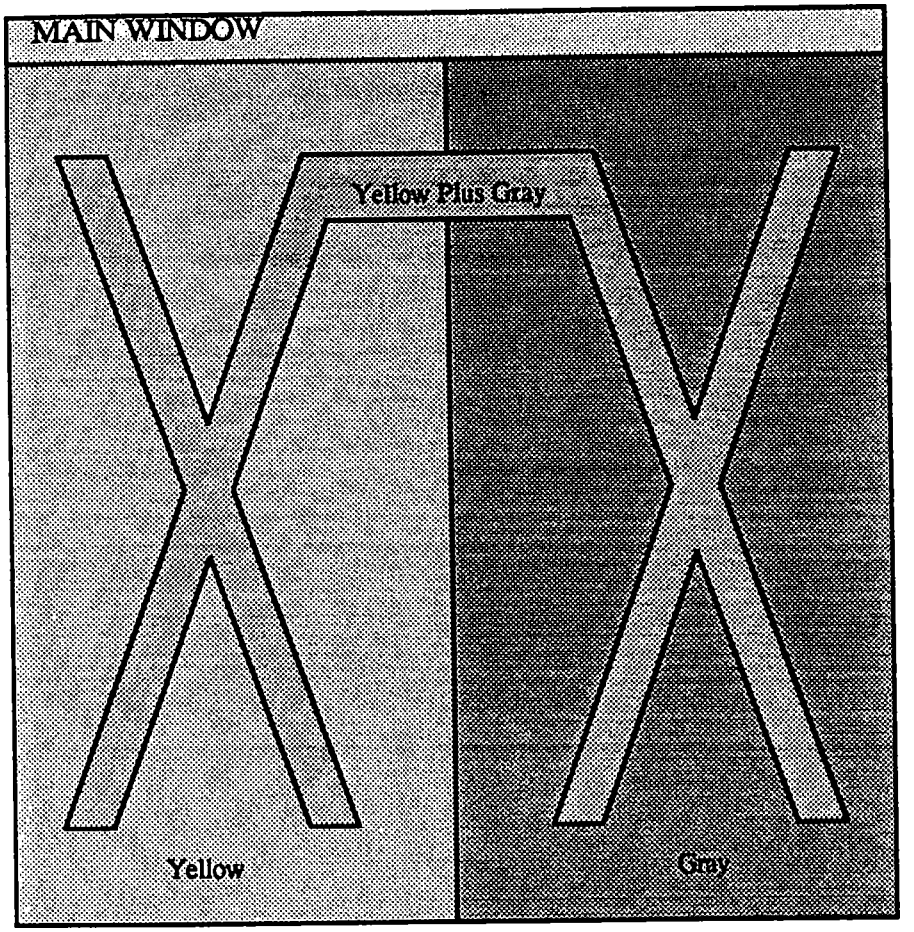


Figure 3.4: Main Window for Yellow/Gray Contrast

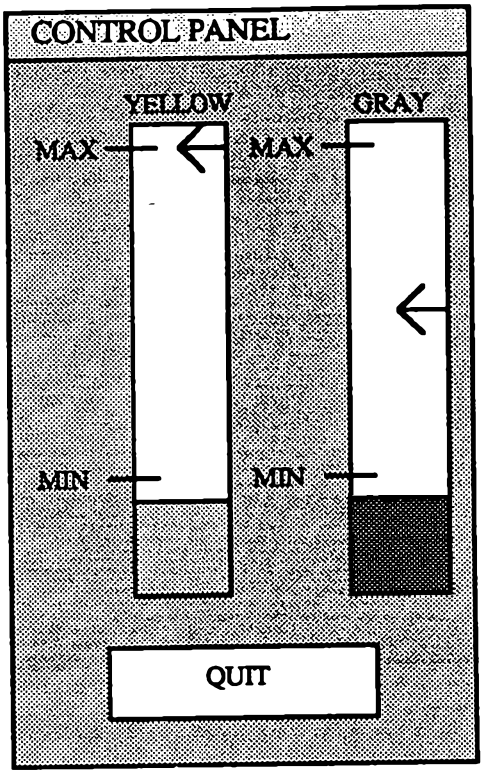


Figure 3.5: Control Panel for Yellow/Gray Contrast

Not everyone can catch the desired visual effect. In most cases, the user needs to view the main window for a while, typically 15 to 20 seconds, before he is able to see the effect. Of course, the difficulty may be a result of the user's looking at the control panel while adjusting the yellow and gray intensities, which could easily cancel the desired visual effect.

As an extension of this visual effect, we were able to get similar results by substituting other colors for yellow.

3.4 Nulling of Apparent Motion

The demonstration program for the nulling of apparent motion requires a main window, a control panel and a status window.

3.4.1 Main Window

The color bar which demonstrates the nulling of apparent motion resides in a main window (Figure 3.6) which is 1000 pixels x 280 pixels (338 mm x 95 mm, subtending a visual angle of 31.5 degrees). The four frames that the color bar cycles through are shown in Figure 2.6, each rectangle being 5 pixels x 10 pixels (1.7 mm x 3.4 mm, subtending a visual angle of 0.3 degrees).

3.4.2 Control Panel

The control panel (Figure 3.7) allows the user to adjust the intensities of the red and green rectangles of the second and fourth frames of the sequence using two sliders. The speed control (an extended button) is used to control how fast the demonstration program cycles through the four frames of the color bar. The *step* button is only active

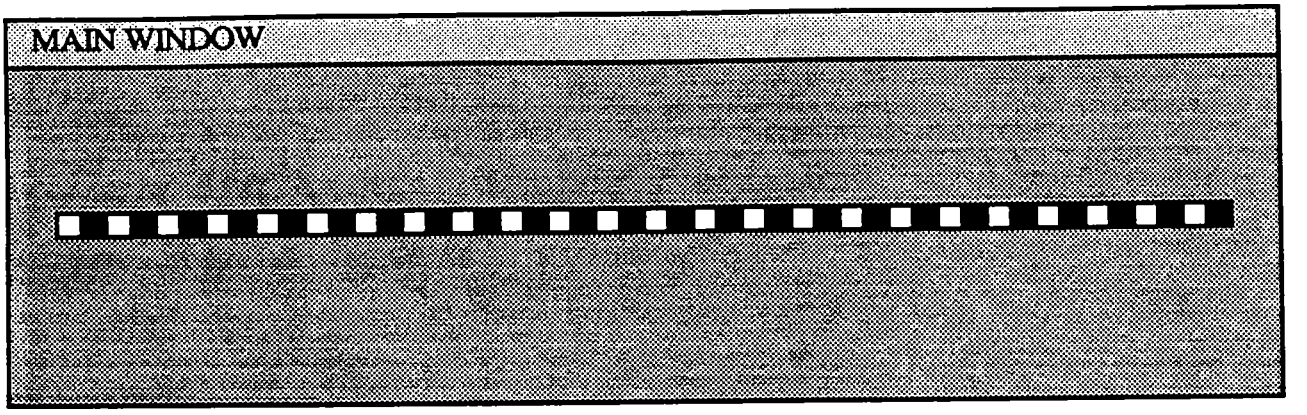


Figure 3.6: Main Window for Nulling of Apparent Motion

at a speed of zero, and enables the user to advance one frame of the color bar. The *quit* button allows the user to terminate execution of the program.

3.4.3 Status Window

The status window (Figure 3.8) provides the following information: red intensity, green intensity and the ratio between the two. Since we are interested in the correlation between the perceived motion of the rectangles in the color bar and the levels of the red and green intensities, the information displayed in the status window should be useful in helping us determine when a certain perceived motion should occur.

3.4.4 Results

During the demonstration, the user can detect the motion of the rectangles quite easily. Choosing a higher cycling speed may well accentuate any perceived motion, although how this parameter affects the determination of equiluminance is still under

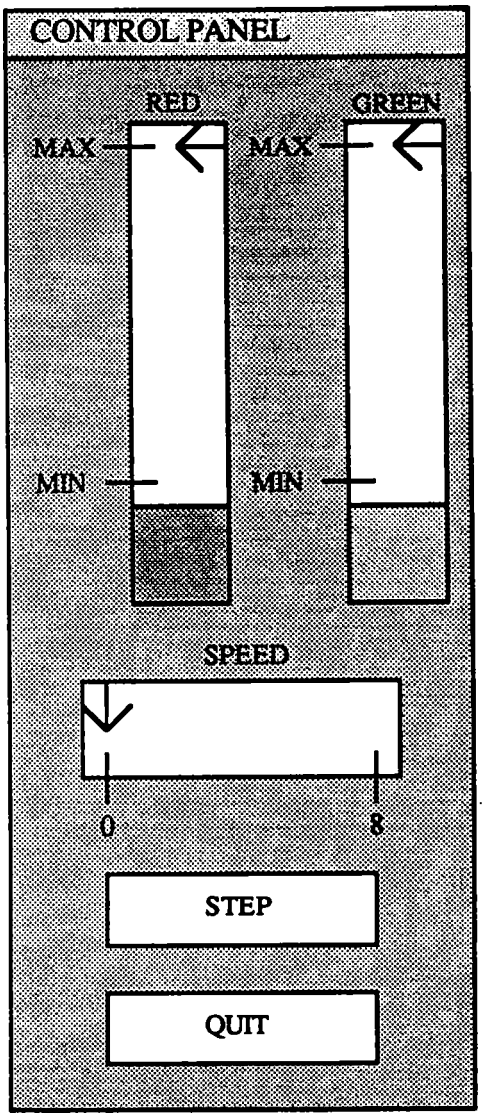


Figure 3.7: Control Panel for Nulling of Apparent Motion

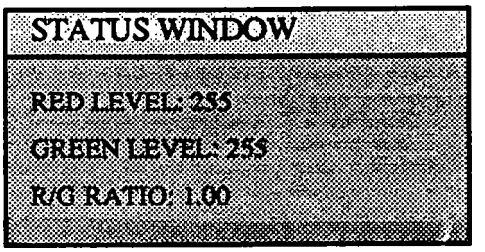


Figure 3.8: Status Window for Nulling of Apparent Motion

debate [2]. The actual R/G ratio for the successful nulling of apparent motion depends on the calibration of individual workstation displays. But after running this program on several of the workstations available, we determined that the calibration of their displays is close enough for us to conclude that at an R/G ratio of about 1.6, we are able to null the apparent motion (i.e. achieve equiluminance for the red and green rectangles). At this point, we only see some intense flickering from the color bar, especially at the higher cycling speeds. At any R/G ratio below 1.6, a rightward motion is detected. Similarly, a leftward motion is detected at any R/G ratio above 1.6.

3.5 Pinwheel

The demonstration program which displays the pinwheel allows the user to specify the number of sectors desired (an even number between 6 and 360 inclusive). If the user does not enter a desired number on the command line prior to execution, then the default number of 20 is used. A main window, a control panel and a status window are displayed on the screen.

3.5.1 Main Window

The pinwheel that is displayed in the main window (Figure 3.9) is generally the same as the one described in the last chapter. One noticeable difference in our implementation is that the pinwheel is painted in sine waves rather than the more conventional square waves (Figure 3.10), as experience shows that the former is more effective for this task. One interesting result of this is that the white/black inner portion of the pinwheel actually resembles a three-dimensional rendering of shaded white cones suspended in dark space, because of the similarity between the sinusoidal color intensities and those used in a shading model.

3.5.2 Control Panel

At the top of the control panel (Figure 3.11) are two sliders for controlling the intensities of the red and green of the outer portion of the pinwheel. The speed control allows the user to choose a desirable rotation speed, given in degrees of rotation per frame. The next button allows the user to toggle between clockwise and counterclockwise rotation. Another button allows the user to stop or resume rotation. Finally, there is the *quit* button.

3.5.3 Status Window

The status window (Figure 3.12) shows the following information: number of sectors, number of rotations per second, maximum red intensity, maximum green intensity and the ratio between the two.

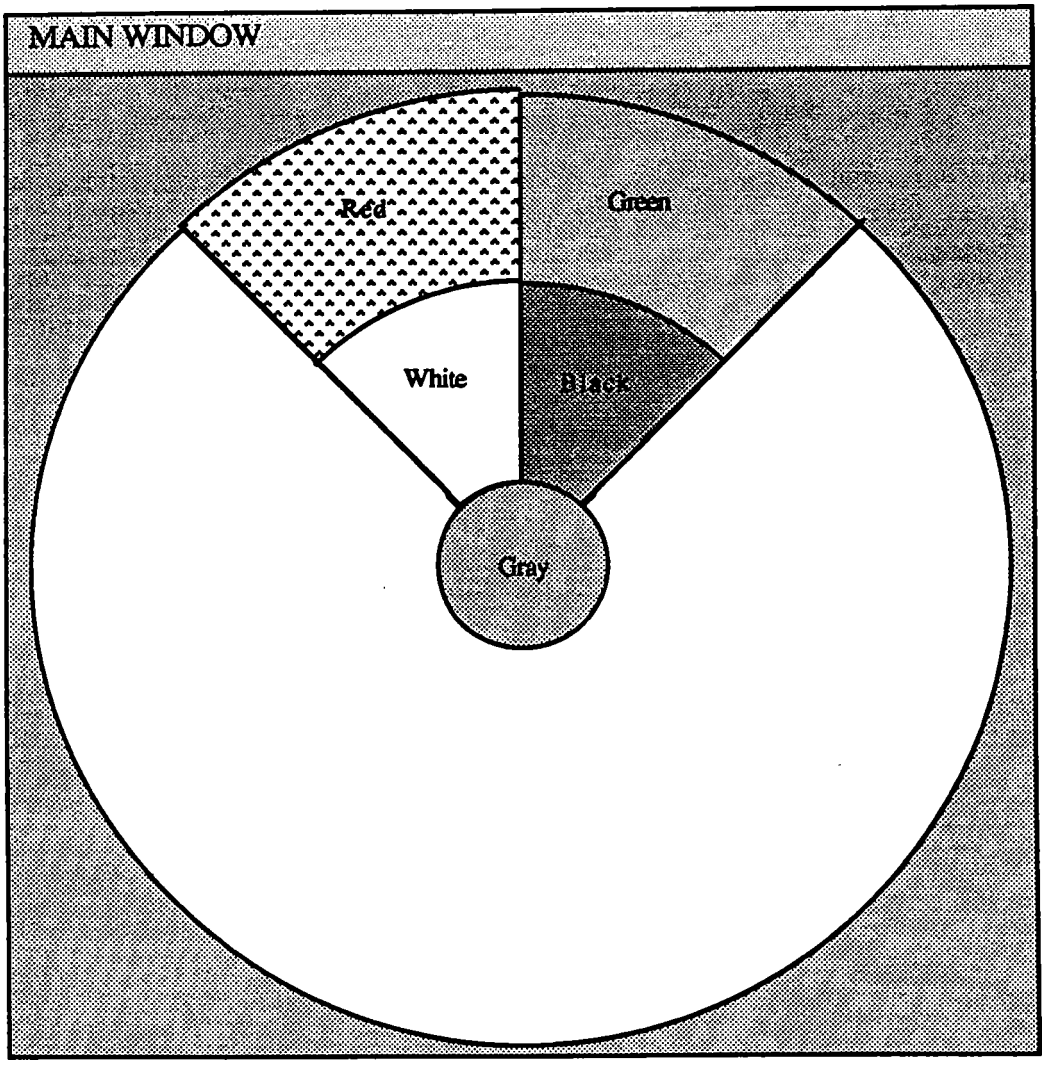


Figure 3.9: Main Window for Pinwheel

Suppose the pinwheel is divided up into 20 alternating red/white and green/black sectors. Each logical sector therefore covers 18 degrees of the pinwheel. The following graphs depict the differences in the color intensities between sine waves and square waves in two adjacent logical sectors.

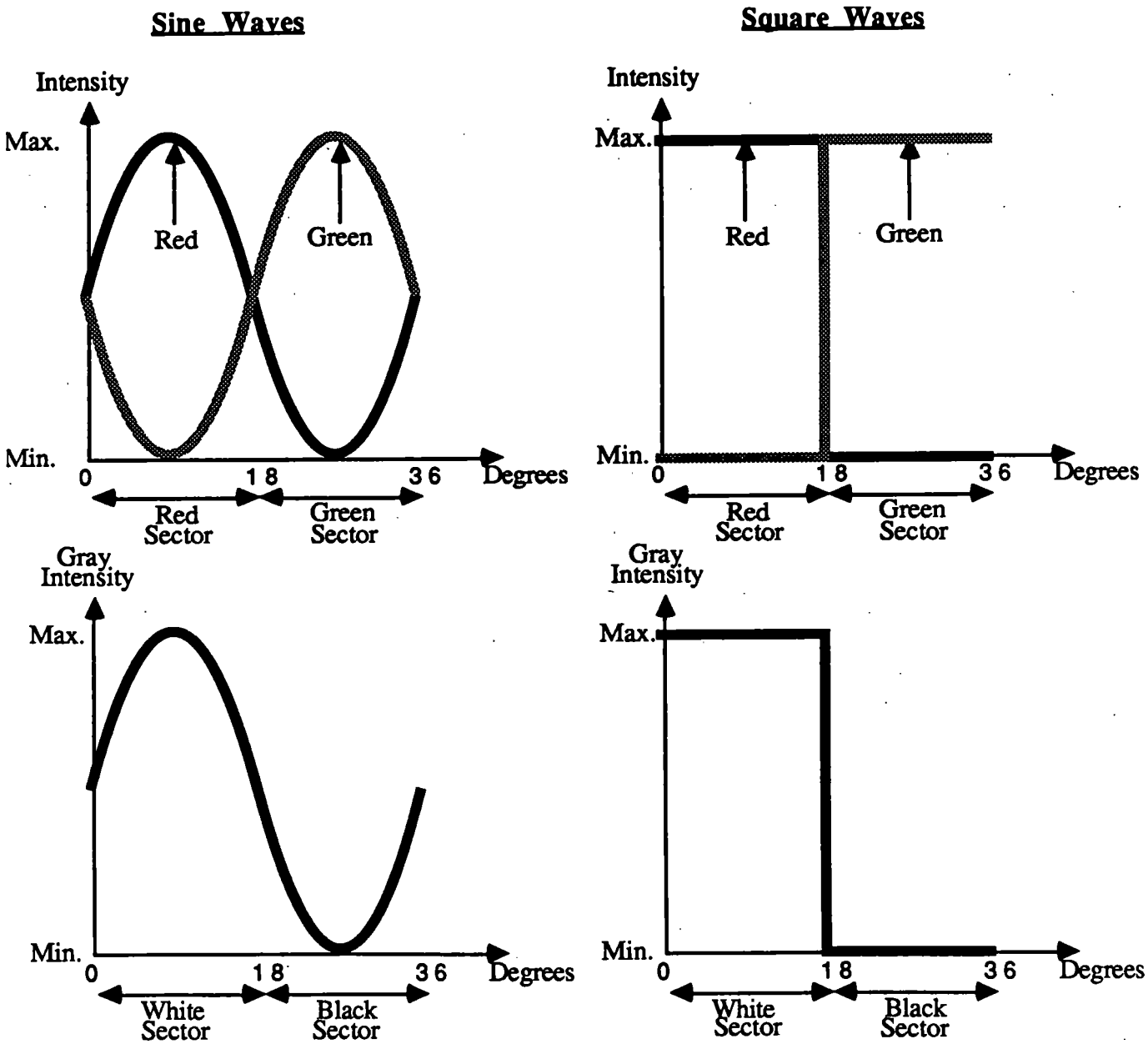


Figure 3.10: Sine Waves vs. Square Waves

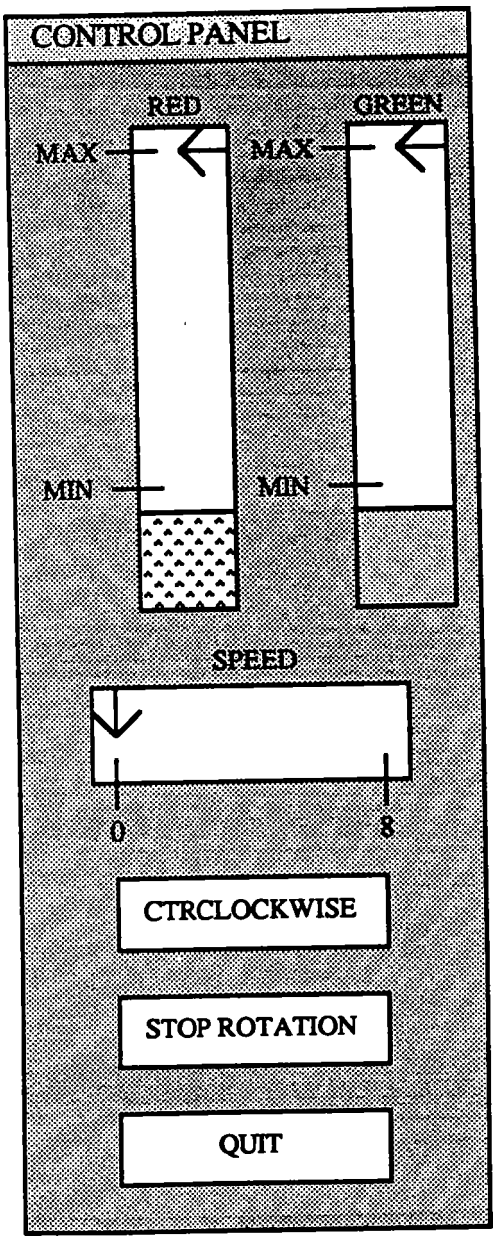


Figure 3.11: Control Panel for Pinwheel

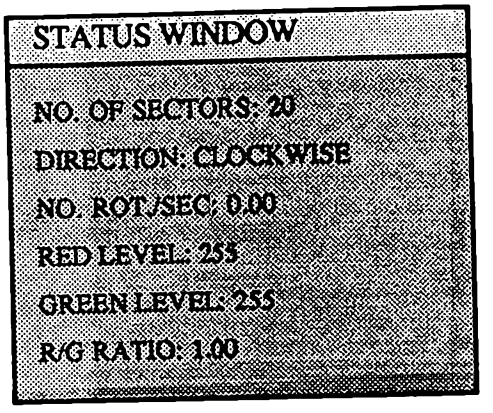


Figure 3.12: Status Window for Pinwheel

3.5.4 Results

Although the user is able to specify the number of sectors for the pinwheel, the default number of 20 is quite satisfactory for our purposes. However, whether any user of this demonstration actually sees the desired apparent cessation of rotation is not clear. Still, at a speed of around 0.5 rotations per second and by setting the red and green intensities so that they give a R/G ratio of about 1.6, most users are able to see an apparent slowing down of rotation of the red/green portion of the pinwheel relative to its white/black inner portion. Some users of this demonstration program thought they saw no rotation of the red/green portion relative to the white. The direction of rotation seems to have little or no effect on a user's ability to see this visual phenomenon. On the other hand, any user of this demonstration program will notice the actual sporadic slowing-down or even stopping of the rotating pinwheel, which is a phenomenon attributable to the irregular update rate for frames (discussed in Section 5.4). This problem causes distraction for the user of this demonstration program, and certainly adds to the difficulty in seeing this subtle visual effect.

3.6 Benham's Disc

The Benham's Disc demonstration program uses a main window, a control panel and a status window.

3.6.1 Main Window

Our version of Benham's Disc which resides in the main window (Figure 3.13) is implemented using 60 physical sectors, which means any change has to be in multiples of 6 degrees. This is a result of the irregular pattern of the disc and of the limitation of the specific implementation of X11. This problem is discussed in more detail in Chapter 5.

3.6.2 Control Panel

It is an aim of this demonstration program to see what happens when we slide the middle group of arcs left or right, and what happens if we vary its length. The control panel (Figure 3.14) allows the user to apply these and other changes to Benham's Disc. At the top of the control panel is a display which shows the position as well as the length of the middle group of arcs. Below this display are buttons for sliding the middle group of arcs left and right, and buttons for increasing and decreasing the length of the arcs within. The speed control is an extended button for setting the speed in multiples of 6 degrees of rotation per frame. One button each is also provided for controlling the direction of rotation, and for stopping/resuming rotation. The last button in the control panel is the *quit* button.

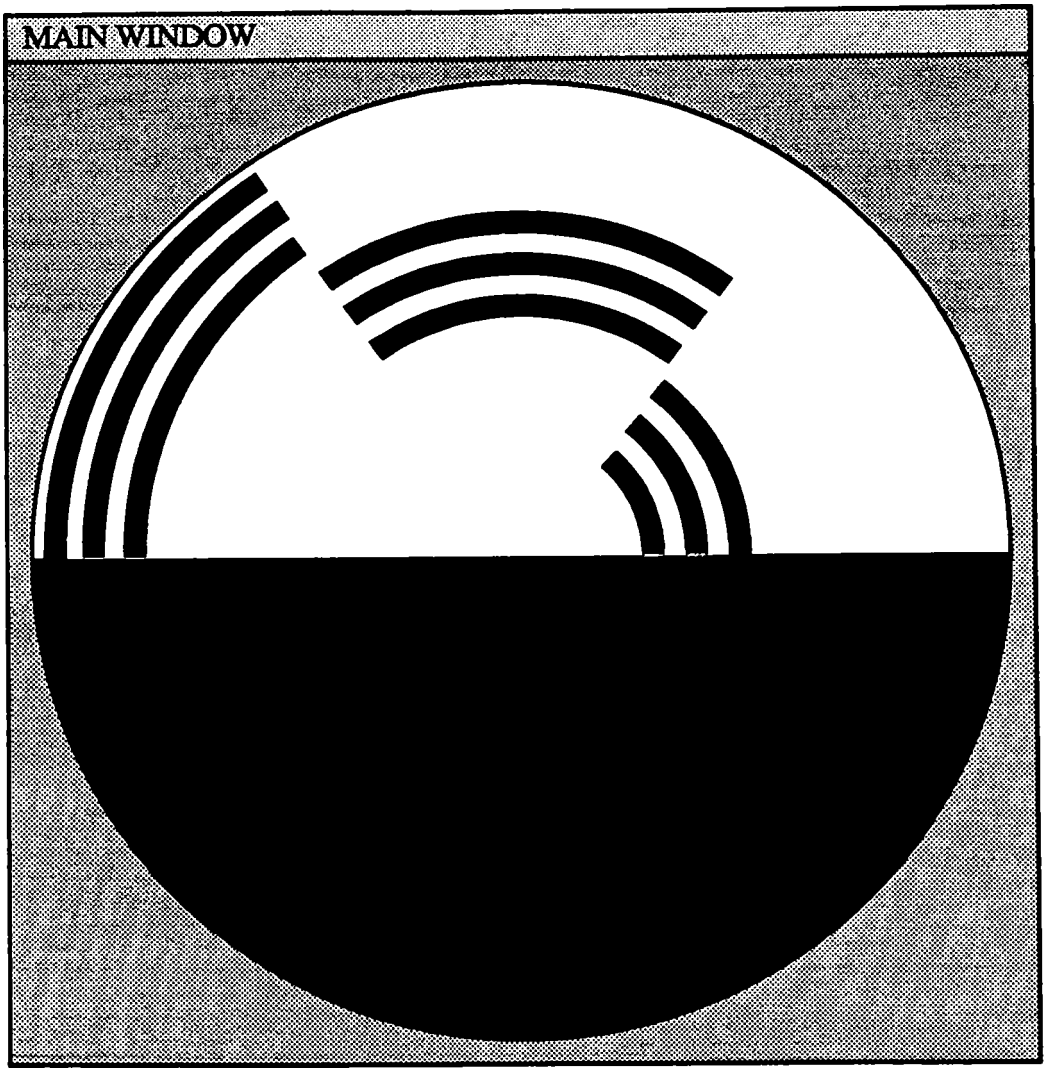


Figure 3.13: Main Window for Benham's Disc

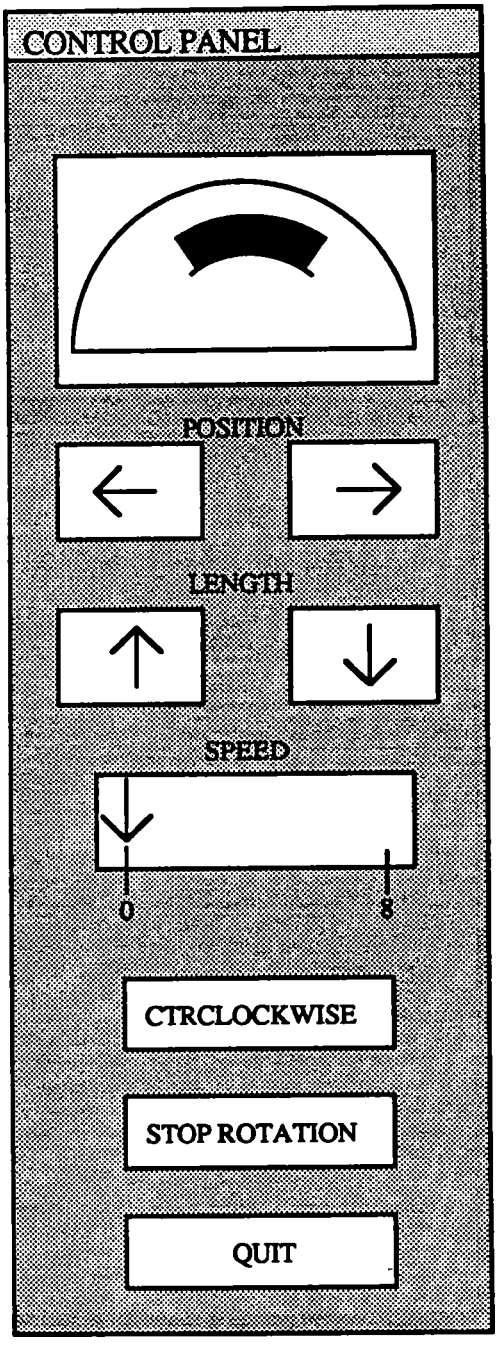


Figure 3.14: Control Panel for Benham's Disc

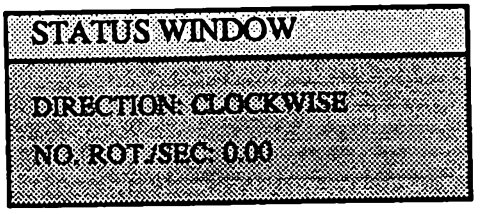


Figure 3.15: Status Window for Benham's Disc

3.6.3 Status Window

The status window (Figure 3.15) shows the direction of rotation and the number of rotations per second.

3.6.4 Results

All the colors for this visual effect are seen as expected: red outermost, blue in the middle and green innermost upon clockwise rotation. Changing the speed of rotation has little effect, except that at least 4 rotations per second are required before the colors are seen. The colors tend to become more vivid as the rotation speed goes up. Sliding the middle group right will gradually change its color from blue to green, while sliding it left will result in a gradual shift toward red. A longer middle group gives a darker color, while shortening the group gives a brighter color. Reversing the direction of rotation reverses the colors of the left and right groups, while the color change of the middle group depends on its position. It can be concluded that the color a particular group assumes depends on its length and its position relative to the border between the black and white semi-circles. The radius of the concentric circle on which a group lies is irrelevant. As in the pinwheel demonstration, the problem of the irregular frame update rate once again results in the non-uniform rotation of

Benham's Disc in this demonstration. This problem will be discussed in Section 5.4.

3.7 Interwindow Interference

The demonstration program for this visual effect puts up a *window one*, a main window, a control panel and a status window.

3.7.1 Window One and Main Window

Since this visual effect takes place across two windows, merely having a main window is not sufficient to display the effect. Therefore, the demonstration program puts up another window called window one, which is the main window's partner in this visual effect (Figure 3.16). Window one is 400 pixels x 700 pixels (135 mm x 237 mm), while the main window is 1000 pixels x 1000 pixels (338 mm x 338 mm). Initially, both windows have the same gray background, a one-pixel wide black border, and each has a line with the same slope and of the same red color within. The two windows are initially positioned in such a way that the two red lines are aligned. Finally, a title bar comes with the main window only, thus allowing the user to move it around the screen by dragging.

3.7.2 Control Panel

Note that the control panel (Figure 3.16) only controls the main window and not window one, which is fixed in position as well as in color and border width. At the top of the control panel are two sliders for controlling the intensity of the gray background and the intensity of the red foreground respectively, both of the main window. A third slider is also provided to vary the border width of the main window between 0 and 160 pixels. The *restore* button restores the main window to its initial

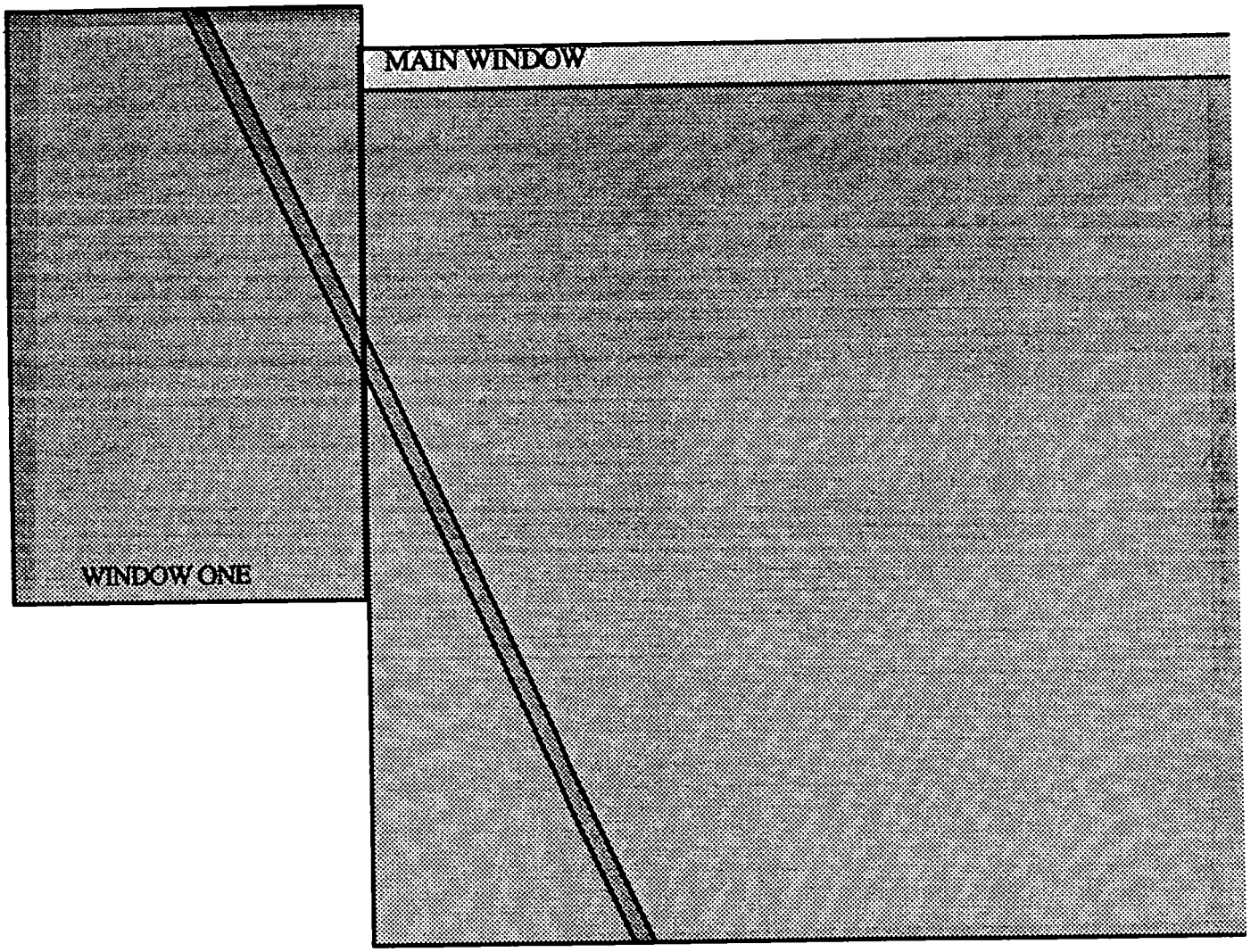


Figure 3.16: Window One and Main Window for Interwindow Interference

appearance and position. As always, the control panel would not be complete without the *quit* button.

3.7.3 Status Window

The status window (Figure 3.18) shows the border width of the main window. In addition, it shows two quantities labeled X and Y. X refers to the horizontal distance between the left border of the main window and window one. Y refers to the vertical distance by which the main window should be moved in order for the lines in the two windows to be colinear. Both values are expressed in number of pixels.

3.7.4 Results

The objective of this demonstration is to show that by changing the position of the main window and/or its appearance, we can correct the erroneous impression that the two lines are one line passing underneath the borders of window one and the main window. Based on informal comparisons conducted for this demonstration, we have come up with the following conclusions. Of all the methods provided to achieve this goal, repositioning the main window seems the most effective. To be really useful, the repositioning should be two-fold: distance the main window from window one, and break the colinearity of the two lines. These two aims are represented by the values of X and Y respectively in the status window. Of course, manipulating the background and foreground colors of the main window and its border width helps to resolve the ambiguity between the two lines, but their roles seem supplementary to the repositioning option. A detailed experiment could be carried out to determine the real effectiveness of each resolving method.

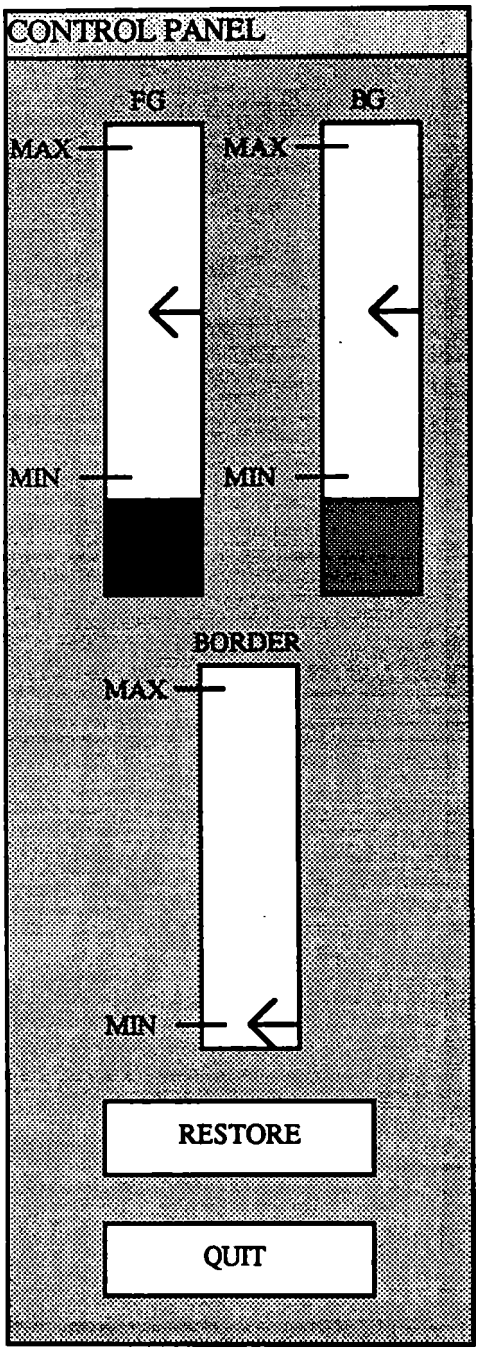


Figure 3.17: Control Panel for Interwindow Interference

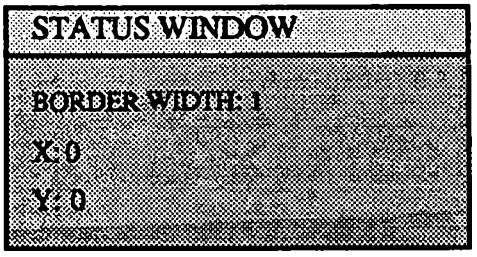


Figure 3.18: Status Window for Interwindow Interference

3.8 Significance of Visual Effects

While it is true that the various demonstration programs allow us to test the capabilities of X11, the visual effects that are produced in the demonstrations are noteworthy in their own right. Through the reproduction of these visual effects, we have gained insight into their mechanisms and their implications for computer graphics.

From the two contrast visual effects , we learn that our perception of a color is strongly influenced by the color of its surrounding. Using the right combinations of colors, we can accentuate color differences of different areas, which helps us distinguish among these areas more easily. Conversely, under the influence of contrast, it is possible to equate erroneously one color with another. One application on the computer display in which both scenarios could take place is a weather radar map which depicts precipitation levels in a fixed area, with the various precipitation ranges indicated by different colors or even shades of the same color. Thus, while it may be easy for the observer of such a map to make out an area of uniform precipitation level with a contrasting background color, it can however be extremely difficult to determine the identities of the colors used when the precipitation levels are mixed

and are close to one another.

The equiluminance demonstration programs show us that it is difficult to tell between different colors when they have the same brightness, especially in the periphery. For instance, the trick of moving the mouse around to help in visually locating the cursor fails when the colors of the cursor and of the screen are at equiluminance.

The demonstration for the nulling of apparent motion in the color bar suggests other potential problems as well. Suppose there is on the screen a control panel which consists of various buttons arranged horizontally. Now if the buttons are designed to blink when they are active, one problem that could arise is the perception of apparent motion as seen in the color bar demonstration. Certainly, if this were to actually occur in this fictitious control panel, the user would be quite distracted from his principal task.

On the other hand, nulling of apparent motion is a highly effective technique for calibrating workstation displays, as equiluminance can usually be determined with an error of about one percent [2]. Calibration is important because many common techniques for improving image quality, such as antialiasing, dithering and elimination of Mach bands, depend on the calibration of the display system [3, 5]. As a calibration technique, nulling of apparent motion is particularly attractive over radiometry in two situations: when a display that is equiluminous for a specific observer is desired; and when radiometry is unavailable or too inconvenient to apply.

The pinwheel demonstration tells us how to induce the perception of motion in the periphery by making sure that there is a luminance change when displayed objects move and how to inhibit it by eliminating any luminance contrast when moving objects are not intended to draw attention to themselves. Thus if we want motion to be detected wherever it appears on the screen, there must always be a significant luminance change as a result of the motion, but if we want to restrict the perception

of motion to the current focal point, motion should result in only chromatic changes that do not affect the luminance.

Benham's disc is an interesting example of the growth and decay of sensation. This particular visual effect has little direct applicability in computer graphics, but it may provide clues for improving perception of continuous motion in animated computer graphics. Unintended derivatives of this effect can arise in animated graphics, for example, a rotating wheel with a hubcap which has complicated patterns.

It is also interesting to note how visual effects interact. Reducing one effect may increase another. There is an interaction between spatial and chromatic changes with time. An example of this is the inverse relationship between Benham's disc and the nulling of apparent motion: the former shows chromatic changes induced by spatial changes while the latter shows spatial changes induced by chromatic changes.

The interwindow interference demonstration illustrates another example of interaction among visual effects. In attempting to break the illusion a single line crossing underneath the boundary between the two windows, the foreground and background colors might be adjusted. Yet we know from our study of contrast phenomena that adjusting the foreground affects our perception of the background and that increasing the distance between the windows can make it more difficult to detect color differences and hence decrease the effectiveness of such differences in breaking the illusion of a single line.

Perhaps the interwindow interference demonstration can be extended to windows of text. Undoubtedly if we have two windows of text side by side, there is a possibility of intermixing the text from both windows while reading. Solutions such as the repositioning of either window, changing the foreground and background colors and border widths should work in this case also, and there are also other options such as using different spacing, fonts, styles, etc. for the text in two such windows.

Chapter 4

Implementation

From the development of the demonstration programs for the visual effects, we have identified various common techniques that should be useful to other X11 graphics programs as well. These techniques are as follows: structure of programs, lookup table animation, compression of lookup table updates, use of plural graphics primitives, window border manipulation, dragging and portability. We will now discuss each of these techniques in detail.

4.1 Structure of Programs

The demonstration programs all have the same basic two-part structure, i.e. an initialization part and an event loop.

Inside the initialization part, we establish the specifications of all windows used, allocate and set up private color cells and graphics contexts, solicit events for windows and have the windows mapped. This part is only executed once when a demonstration program is run, so the fact that it sometimes takes a while to complete for more complicated images, such as Benham's disc, is of little concern.

One interesting note about the initialization of X11 programs, both in real applications such as ours and in textbook examples, is that the code almost always resides in the main body of a program and does not exist as a subroutine. The likely reason is that the code usually requires the manipulation of so many X11 resources that it is too cumbersome to have it as an initialization subroutine by its own.

The other part of this program structure is the event loop. This is a loop which reads an event returned by the X11 server and deals with the event depending on its type. Of course, an event such as a *ButtonPress* (indicating a press of any mouse button) has to be further analyzed by an application program in order to determine what action it really stands for. In all cases, one such action is the *quit* option, which allows a user to exit the program. The types of event that are inserted in the event queue to be read subsequently and processed are stated in the initialization part of the program and are subject to the requirements of the visual effect being reproduced. It is essential that all programs which display windows solicit *XExpose* events. This is the type of event which the X11 server sends whenever parts of an application window are exposed, as a result of its being mapped, moved, deiconified or resized. Upon reading an XExpose event, the application redraws the newly exposed part of the window.

This basic event loop structure is modified to include a mini-loop at the top for the demonstration programs that produce the dynamic visual effects. The mini-loop checks for events waiting to be processed in the event queue and advances the animation by one frame when the event queue is empty. When the event queue is non-empty, the mini-loop is exited and execution of the program resumes at the event-identification stage of the outer event loop. The advantage of the mini-loop is that the outer loop does not block to wait for events to arrive and frame updates continue when there are no events to be processed.

4.2 Lookup Table Animation

The demonstration programs which produce animated sequences use *lookup table animation* [18, 22, 23]. The lookup table contains the colors of the image. By dynamically changing the entries (colors) in the lookup table, we can animate the image. In X11, the lookup table is a collection of private color cells which an application program requests the X11 server to allocate as part of initialization. These private color cells are given their initial colors by the application. Graphics contexts then determine which color cell indices to write into the frame buffer when the objects are drawn. With an object referencing a private color cell, we can change its color by changing the contents of the private color cell. The new color appears immediately upon the next screen refresh. We are thus able to achieve animation for the demonstration programs using this method by uniformly dividing up the images and have the different parts reference different private color cells.

Using this method, for the pinwheel and Benham's disc demonstration programs, we divide the circles into physical sectors of equal size and have each reference a different collection of private color cells (Figure 4.1). By knowing for the next frame which physical sector should acquire the colors of the previous physical sector, we are therefore able to achieve rotation of the circles. For the color bar sequence program, the color bar is divided up into rectangles, instead of sectors, each referencing a different lookup table entry (private color cell). Apparent motion is achieved by modifying the lookup table entries according to the predefined frame sequence.

Lookup table animation has some limitations, as we will discuss in the next chapter.

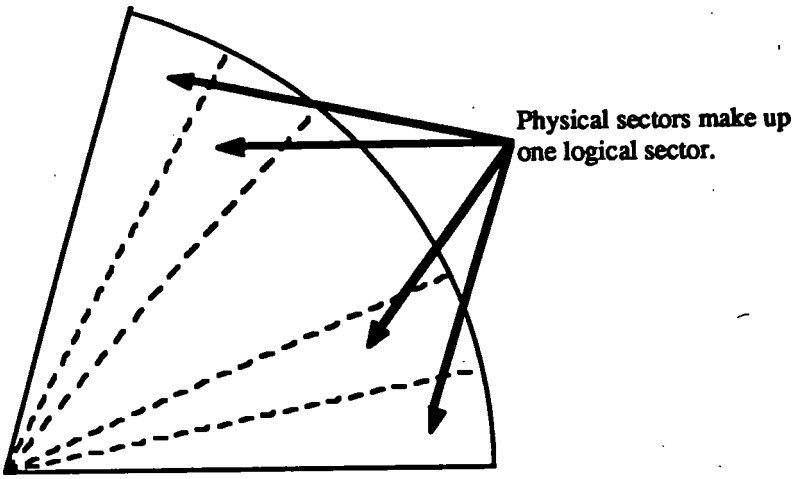


Figure 4.1: Relationship Between Physical and Logical Sectors

4.3 Compression of Lookup Table Updates

Benham's disc in the demonstration program is divided up into sixty six-degree physical sectors, each referencing its own collection of private color cells. Suppose we would like to rotate the disc clockwise by the smallest increment, i.e. six degrees. The straight-forward way to do this is to simply shift the colors of every physical sector on to the next, requiring lookup table updates for sixty sectors. A closer look at the pattern of Benham's disc reveals that many of the updates are superfluous. For example, for the black semi-circle on the disc, we only need to change the colors of the two physical sectors which border on the white semi-circle (Figure 4.2). The same idea applies to the three groups of concentric arcs. The savings offered as a result is essential, because the rotation of Benham's disc in another implementation which does not compress lookup table updates is found to be too slow for any colors to be seen.

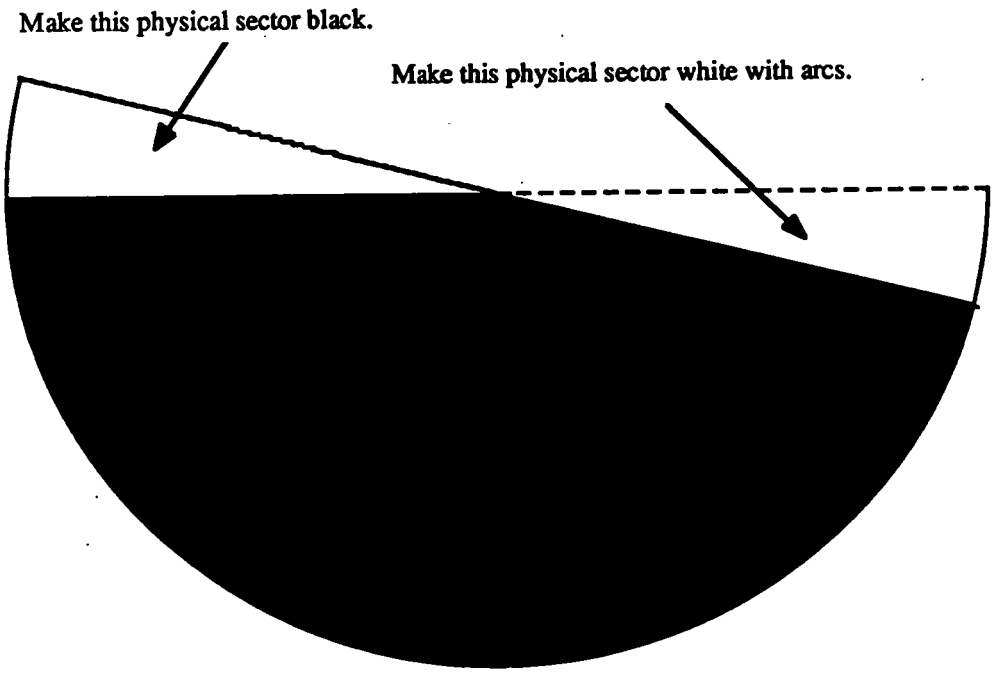


Figure 4.2: Compression of Changes to Benham's Disc upon Rotation

The above technique is not used in the pinwheel demonstration program, since it requires a lot less lookup table updates to achieve rotation. Recall that the pinwheel consists of alternating logical sectors of red/white and green/black. All the private color cells that we need are those for the physical sectors that make up two such logical sectors. The trick is to make sure that all logical sectors of the same color combination reference the same graphics contexts and thus the same private color cells.

4.4 Use of Plural Graphics Primitives

By plural graphics primitives in X11, we mean those primitives which carry out multiple instances of its basic action. Usually the multiple instances are in the form of a list of objects, such as a collection of private color cells, which is passed as an array parameter to the primitive along with the array size. Such primitives speed up execution considerably due to the reduced communication overhead compared to using their singular counterparts multiple times. For example, X11 makes available a primitive called *XStoreColor* which is used for changing the RGB components of a single private color cell. Its plural version, *XStoreColors*, can be used for changing the contents of a non-contiguous collection of private color cells. XStoreColors is used extensively in the demonstration programs for updating a frame within an animated sequence. The same idea applies to many of the graphics drawing primitives as well, such as those for lines and rectangles. It is an objective of the demonstration programs to make use of X11's plural primitives as frequently as is warranted.

4.5 Window Border Manipulation

In X11, the way a window appears and how it is manipulated depends on the window manager used. The demonstration programs run under *Tom's Window Manager* (TWM). The policies imposed by TWM are fairly representative of window managers that operate under X11 [21].

Under TWM, second-level windows (the root window being the only top-level window) come automatically with a title bar. This is convenient because the title bar allows us to use the window manager to move, resize and iconify the window. However, at this level, all X11 primitives that are supposed to operate on the window border have no effect. Thus, while second-level windows are fine for all other demonstration programs, we have to come up with something else for the interwindow interference demonstration program. The solution in this case is to create a second-level window that covers the whole screen. The regular windows (two side-by-side windows, the control panel and the status window) are created as children of this window and are thus on the third level of the window hierarchy. While this solves the window border manipulation problem, we have lost the useful title bars which come automatically with second-level windows. Therefore, in order to implement the 'move main window' option in this demonstration, we have to create another third-level window and use it as the title bar of the main window.

4.6 Dragging

Dragging an object on the screen requires that one position the cursor at the object using the mouse and move the mouse with any button pressed. Visual feedback showing the object moving along with the cursor should be present during the dragging. Dragging terminates when the mouse button is released. This is the technique used

in the demonstration programs for manipulating sliders in the control panel and for moving the main window around in the interwindow interference demonstration.

To implement this technique in X11, an application program needs to set the *ButtonMotionMask* bit in the event mask for the window where dragging is to take place. This set bit ensures that whenever the mouse moves in that window with any button pressed, the X11 server will notify the program by sending it a *MotionNotify* event. In response, the program finds out the new position of the cursor by looking up the x and y fields in the MotionNotify event structure and draws the dragged object at this location. One problem of this naive approach to implementing dragging in X11 is that the rate at which MotionNotify events are sent may inundate the program to such a degree that performance suffers. Fortunately, there are two solutions to this problem in X11.

The first solution controls the rate of delivery of MotionNotify events by setting the *PointerMotionHintMask* bit in addition to the ButtonMotionMask bit in the event mask for the window in which an object is to be dragged. As a result of these two set bits, the X11 server still returns MotionNotify events during the dragging, although the x and y fields in the event structure are no longer up-to-date. To get the new position of the cursor so that we can update the screen, we use the primitive *XQueryPointer*. This is where the delivery rate of MotionNotify events is controlled, because the X11 server will not send a new MotionNotify event before the primitive XQueryPointer is called. The number of MotionNotify events received and processed by a program is thus greatly reduced using this technique. However, even though this technique works on the workstations that we have used, there is no guarantee that the X11 server on all workstations will stop sending out MotionNotify events before a call to XQueryPointer [11].

Instead of controlling the rate of delivery of MotionNotify events, the second solution aims at processing only a fraction of all regular MotionNotify events received by

a program. Whenever a MotionNotify event has been read from the event queue, we use the primitive *XPeekEvent* to look at the next event in the queue, if any. If the next event happens to be another MotionNotify event, we just read it off the queue. The process is then repeated until the next event is of a different type or until the event queue is empty. This technique saves us the work of actually processing all but the last of a whole series of MotionNotify events. Note that this technique is possible only because the x and y fields in a MotionNotify event make an absolute coordinate rather than a relative one.

Even though it is suggested that the first solution is the more effective of the two [11], our tests indicate the opposite, probably because XQueryPointer is a two-way request, which blocks the program for an excessive amount of time, preventing frame updates from taking place. Hence, we have elected to adopt the second solution in the demonstration programs. We have also implemented dragging without either of the two techniques discussed above. The results that we obtained were, as expected, much worse than those obtained using either technique.

Dragging as implemented on the Macintosh is somewhat different. Once an object has been 'grabbed' by the mouse, the standard technique is to monopolize the CPU (on the Macintosh) while the mouse button is depressed, continuously updating the display with the new position of the object. This has the side effect of eliminating any animation the program might be performing at the time, unless the program is carefully crafted to multiplex the mouse polling with the animation. Our implementation of dragging in X11 is different in the sense that during dragging, animation is performed whenever the mouse is not in motion, because there are no MotionNotify events to process at that time.

4.7 Portability

The only issues that arise from the demonstration programs as far as their portability among X11 workstations is concerned are those dealing with color (discussed in Section 1.2.3). First, the demonstration programs are designed for use on workstations which support the PseudoColor visual class. As such, the two demonstration programs which display gray scale images only, namely the ones for black/white contrast and Benham's disc, will work on X11 workstations which run under the GrayScale visual class also.

The other issue has to do with the color map strategy used by the demonstration programs, namely the private color cells strategy. Under this strategy, the successful execution of the demonstration programs requires that the private color cells needed to draw the image be allocated by the X11 server. The problem occurs if the workstation runs out of private color cells for allocation, in which case the demonstration programs displays a diagnostic message on the screen and exits normally.

Chapter 5

Problems and Solutions

The demonstration programs which display various types of visual effects all face some common problems, especially when animation is involved. Not all the problems are associated with X11; some can more appropriately be attributed to its specific implementation.

5.1 Screen Refresh Synchronization

A problem with graphics systems occurs when we try to change the contents of a lookup table entry (private color cell in X11) in the middle of a screen refresh. One possible result is that part of the image will bear the old color, while the other part will display its new color. The demonstrations of the dynamic visual effects suggest that X11 synchronizes the vertical retrace with a lookup table change, thereby eliminating this potential problem. This suggestion is corroborated by a test program in which the whole screen gets its color from one lookup table entry, the contents of which are constantly switched between black and white. What we observe during the execution of this program is that the screen always displays one solid color — black or white.

Note that the screen would have appeared solid gray at a sufficiently high update speed, even with no synchronization. However, since the update rate from running this program is slow enough that only black or white, and not gray, is seen at a time, we conclude that this problem, which may well exist in other graphics systems, is not present in X11.

This is a mixed blessing, however, because we do not know how this is synchronized with the event loop, which means that an X11 client is unlikely to be able to guarantee that lookup table entries are in fact being changed in a precise temporal pattern.

We suggest that all window systems should provide, at least optionally, a method for updating lookup tables in synchronization with the vertical retrace and for delaying further processing of output requests until a specified number of vertical retraces have been performed. Early high performance line drawing systems had provisions for this type of synchronization to avoid burning out the CRT as the length of the display list changed. The same effect happens for raster displays (double-buffered display algorithms such as z-buffer [4] have this problem unless the update rate is oblivious to the contents of the image).

5.2 Drawing Speed

It is impossible to do effective real-time animated graphics by redrawing under X11 simply because the graphics primitives are not fast enough. The only way that satisfactory animation can be accomplished in X11 is by lookup table animation. This works for our animated demonstration programs to a certain degree. However, this method is only applicable to relatively simple images. Thus, anyone wishing to create complex animation sequences in real time should instead make use of dedicated graphics workstations such as the Silicon Graphics Iris workstation.

5.3 Number of Color Cells and Number of Planes

As described before, lookup table animation is made possible by the extensive use of private color cells. The number of private color cells available limits the size of the lookup table. As an 8-bit index, this number ranges from 254 to 14 on the X11 workstations at the Computer Graphics Laboratory. Even the maximum number of private color cells available in this environment is barely enough for some of our animated visual effect reproductions. For example, our implementation of Benham's disc has three groups of arcs. With the disc being divided up into 60 physical sectors, we need for each sector a private color cell for each group of arcs, and a fourth private color cell for the background, giving us a total requirement of 240 private color cells. It is thus impossible for our version of Benham's disc to add another group of arc, desirable as this may be.

Note that this problem does not lie with X11 itself. Rather, the problem can be traced to the number of planes a workstation has. For example, on a workstation with 12 planes, which as a consequence has 12-bit lookup table indices, we can expect to have access to at most 2^{12} lookup table entries.

5.4 Interruption of Animation

As pointed out earlier for the pinwheel and Benham's disc, both demonstration programs are plagued by the irregular frame update rate. Since the rotation of the circle in both images relies on the successive update of frames from the animation sequence, this problem makes itself quite visible as erratic rotation — slowing down or even stopping every now and then. There are two factors to consider for this problem: user-induced events and interference from system processes.

Recall that the basic structure of the demonstration programs which produce an-

imated sequences is that whenever an event is being processed, no frame updates take place. Such interruptions are relatively transient for mouse clicks, but the same cannot be said for dragging. Even with the compression of MotionNotify events in use, the interruption imposed by dragging on animated sequences is still quite damaging indeed. However, the problem caused by this factor is relatively minor when compared to that caused by the second factor.

Concerning system processes, our experience is that our lookup table animations had unpredictable performance even on an isolated workstation because of system processes pre-empting the X11 server. Multiprocessors may be the solution to this, since many workstations now have dedicated processors (e.g. the Ikonas) where the user/client may be able to specify actions that cannot be pre-empted by demands of the operating system. But most window systems do not provide a mechanism for specifying this type of behavior. A window server architecture that does support this behavior has been discussed elsewhere [15].

We have also tried running the demonstration programs remotely on the mainframe, which presents problems of its own. Interruption to frame updates on the mainframe is caused by time-sharing among different processes. And since the remote configuration still requires the X11 server for display on the workstation (which is interrupted by system processes locally), the results are, as expected, somewhat worse than what we get from running the programs locally.

The interruption of animation caused by the irregular frame update rate is very damaging, and makes the perception of some of the more subtle dynamic visual effects almost impossible.

5.5 Processing for Lexical and Syntactic Events

A problem that is related to the uneven performance of lookup table updates that was caused by interference from system processes and a lack of synchronization with the refresh cycle is the difficulty in providing simple lexical feedback in X11. Sliders require a significant amount of computation by the client. In X11 it appears to be necessary for a client to 'latch on' to the mouse for effective dragging of sliders. This seems inefficient.

NeWS provides a mechanism whereby actions can be performed locally on the server as a result of mouse actions. The same technique existed in the E&S PS 300 through function networks, whereby a dial or mouse could control the position of a visual slider. In many applications, satisfactory animation can be achieved by directly connecting a lookup table entry to mouse just as for a slider. Older displays featured a hardware cursor, precisely because the turnaround time to interact with an application program was too long for effective interaction. Similar requirements exist for rapid lexical and syntactic feedback where the necessity of invoking actions within the client is a bottleneck on the real-time performance of the window system.

5.6 Temporal Control over Animation

Demonstrations such as Benham's disc exist at the limits of practical animation tasks and thus provide a good measure of the performance of X11 and of the workstation displays themselves. The 60-cycle refresh rate is the limiting factor for this effect. Colors are just beginning to appear at the fastest we can rotate the disc. 60 Hz is the limit of the human visual system for temporal information, but to achieve this on a computer display, a 120Hz refresh rate is required for animated images to avoid aliasing artifacts. (With 60 Hz display, it is necessary to filter everything above 30

Hz to avoid aliasing. With a 120 Hz display, filtering only has to be performed above 60 Hz which is close to the cutoff of the human visual system).

Display processors such as the Adage/Ikonas and the Sun TAAC are capable of refresh rates higher than 60 Hz. They have been used successfully for animated images where the elimination or precise control of visual effects is important [16]. If we believe that animated images will be commonplace in future workstation applications, then some provision should be made for higher refresh rates. This is perhaps more important than the current efforts to increase the spatial resolution of workstation displays.

5.7 Workstation Display Artifacts

On the DEC VAXstations which we use to display the visual effects from our demonstrations, we are able to detect some undesirable artifacts on their displays. Two such examples are the movement of lines seen inside the gray circle at the center of the pinwheel under rotation (Figure 5.1) and the 'leakage' of redness from the line in the main window into a light background in the interwindow interference demonstration (Figure 5.2). All these artifacts that have been noticed suggest that there could well be others that exist but which have yet to be discovered. Thus, it may be that our reproduction of visual effects are affected adversely by such artifacts.

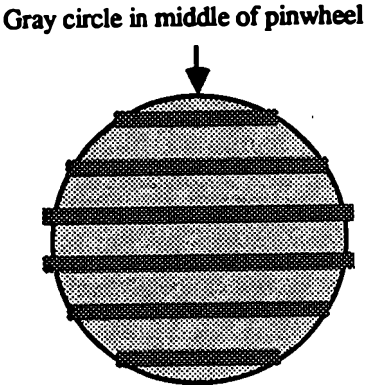


Figure 5.1: Movement of Lines in Pinwheel

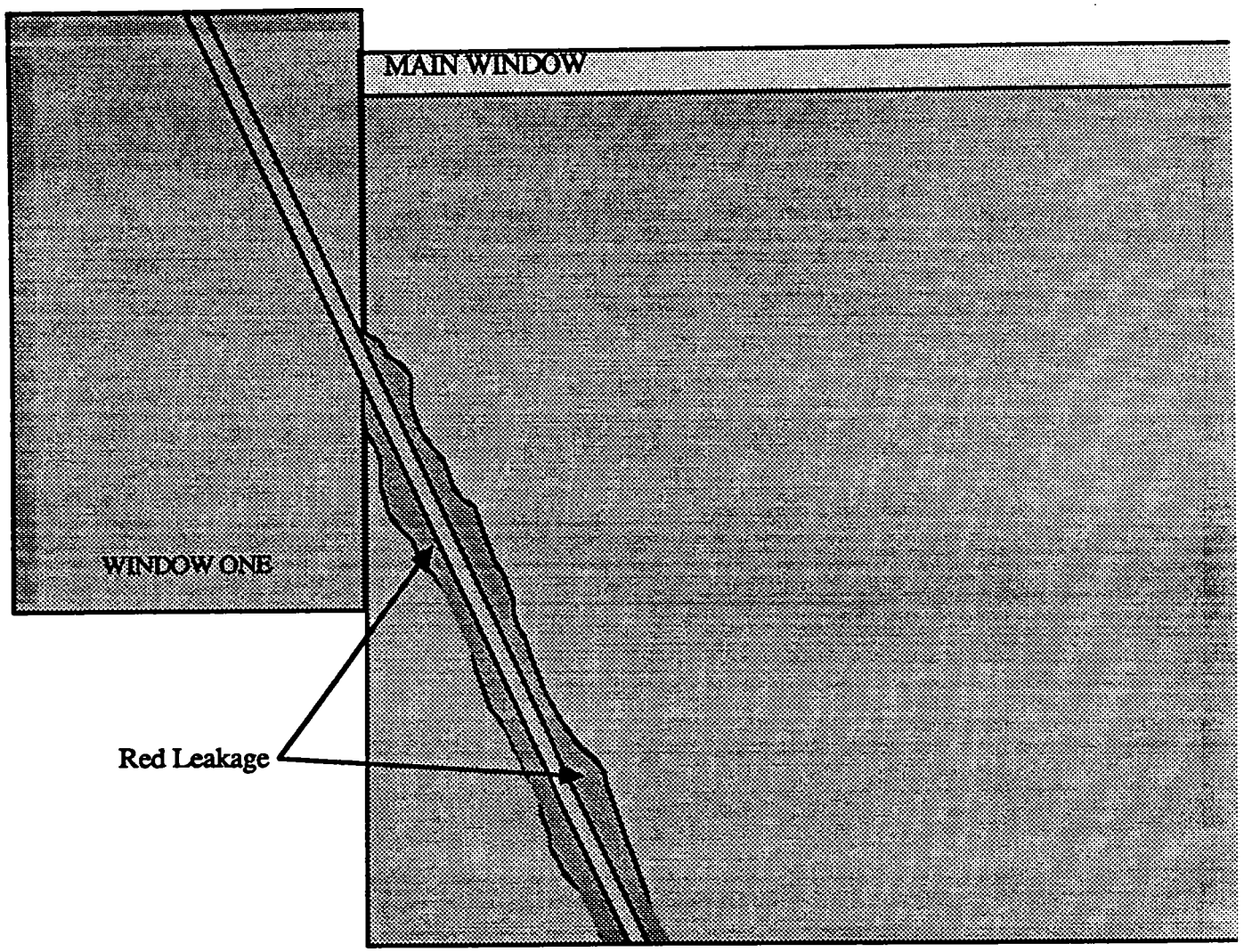


Figure 5.2: Red Leakage

Appendix A

Location of Programs

All files pertaining to the demonstration programs that are discussed in this essay reside in 'watcgl'.

The file /u/demos/cue_card/visual_effects contains explanation on what each demonstration program illustrates. This file can be displayed by executing the shell script /u/demos/visual_effects.

The file /u/demos/README/visual_effects contains explanation on how to run the demonstration programs.

The executables of the demonstration programs are contained in /u/demos/Visual_effects. The following names are used:

1. Black/White Contrast — bw
2. Yellow/Gray Contrast — yg
3. Nulling of Apparent Motion — cb
4. Pinwheel — pw
5. Benham's Disc — bd

6. Interwindow Interference — lines

7. Test Program — test

Bibliography

[1] Albers, Josef. *Interaction of Color.* New Haven, CT: Yale University Press, 1963.

[2] Antis, Stuart M., and Cavanagh, Patrick. A Minimum Motion Technique for Judging Equiluminance. *Color Vision.* New York, NY: Academic Press, 1983, pp. 155–166.

[3] Barros, J. and Fuchs, H. Generating Smooth 2-D Monocolor Line Drawings on Video Displays. *Computer Graphics,* **13**(2), pp. 260–269, August 1979.

[4] Booth, K. S., Forsey, D. R. and Paeth, A. W. Hardware Assistance for Z-Buffer Visible Surface Algorithms. *IEEE Computer Graphics and Applications,* **6**(11), pp. 31–39, November 1986.

[5] Catmull, E. A Tutorial on Compensation Tables. *Computer Graphics,* **13**(2), pp. 1–7.

[6] Cavanagh, Patrick, MacLeod Donald I. A., and Antis, Stuart M. Equiluminance: Spatial an Temporal Factors and the Contribution of Blue-sensitive Cones. *Journal of the Optical Society of America A,* **4**(8), pp. 1428–1438, August 1987.

[7] Cowan, William B. Computer Science 788 Courses Notes. University of Waterloo, 1989.

[8] Frisby, John P. *Seeing.* Oxford, UK: Oxford University Press, 1980.

[9] Goetz, Susan M. and Beatty, J. C. Color Principles and Experience for Computer Graphics. *Graphics Interface '82,* pp. 313–322, May 1982.

[10] Hochberg, Julian E. *Perception.* Englewood Cliffs, NJ: Prentice-Hall, 1978.

[11] Jones, Oliver. *Introduction to the X Window System.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

[12] Kaehler, C., Horn, B., Capps, S. *Macintosh.* Cupertino, CA: Apple Computer Inc., 1984.

[13] Kaiser, Peter A., Vimal, Ram L. P., Cowan, William B., and Hibino, Haruo. Nulling of Apparent Motion as a Method for Assessing Sensation Luminance: An Additivity test. *Color Research and Application,* **14**(4), pp. 187–191, August 1989.

[14] Kaufman, Lloyd. *Sight and Mind.* New York, NY: Oxford University Press, 1974.

[15] Kelley, Jeffrey V., Booth, K. S. and Wein, M. Design Experience with a Multiprocessor Window System Architecture. *Graphics Interface '89,* pp. 62–69, June 1989.

[16] Klassen, R. V. PhD Thesis, University of Waterloo, 1989.

[17] Luckiesh, M. *Visual Illusions: Their Causes, Characteristics and Applications.* New York, NY: Dover Publications, 1965.

[18] MacKay S. A. and Booth, K. S. Techniques for Frame Buffer Animation. *Graphics Interface '82,* pp. 213–220, May 1982.

[19] Massachusetts Institute of Technology. *X Window System Protocol, Version 11.* September 1987.

[20] Rock, Irvin. *An Introduction to Perception.* New York, NY: Macmillan Publishing, 1975.

[21] Scheifler, Robert W., Gettys, James, and Newman, Ron. *X Window System.* Bedford, MA: Digital Press, 1988.

[22] Shoup, Richard G. Color Table Animation. *Computer Graphics (Siggraph '79 Conference Proceedings),* **13**(2), pp. 8–13, August 1979.

[23] Sloan, K. R. Jr., and Brown, C. M. Color Map Techniques *Computer Graphics and Image Processing,* pp. 297–317, August 1979.