HZ BOOKS
华章教育

全美经典
学习指导系列

计算机图形学习题与解答

（英文版·第2版）

Computer Graphics Second Edition

The perfect aid for higher grades!

Covers Computer Graphics in 2D and 3D—supplements any class text

Simplifies all aspects of creating digital graphics

Over 350 solved problems step-by-step

Ideal for independent study!

Explains new techniques in shadowing and photo-realism

ZhiGang Xiang
y Plastock 著

全球销售超过3000万册！

机械工业出版社
China Machine Press

全美经典
学习指导系列

计算机图形学习题与解答

（英文版·第2版）

COMPUTER GRAPHICS, SECOND EDITION

（美）Zhigang Xiang Roy Plastock 著

机械工业出版社
China Machine Press

ZhiGang Xiang & Roy Plastock: Computer Graphics, 2E (ISBN 0-07-135781-5).

本书版权登记号：图字：01-2002-2181

图书在版编目（CIP）数据

计算机图形学习题与解答：第2版 /（美）翔（Xiang, Z.G.），（美）普拉斯托克（Plastock, R.）著. -北京：机械工业出版社，2002.8

（全美经典学习指导系列）

书名原文：Computer, Sraphics, SecondEdition

ISBN 7-111-10417-X

Ⅰ.计… Ⅱ.①翔… ②普… Ⅲ.计算机图形学-习题-英文 Ⅳ.TP391-44

中国版本图书馆CIP数据核字（2002）第038501号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：华 章

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2002年8月第1版第1次印刷

787mm × 1092mm 1/16 · 22.5印张

印 数：0 001-3 000册

定 价：35.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

PREFACE

We live in a world full of scientific and technological advances. In recent years it has become quite difficult not to notice the proliferation of something called computer graphics. Almost every computer system is set up to allow the user to interact with the system through a graphical user interface, where information on the display screen is conveyed in both textual and graphical forms. Movies and video games are popular showcases of the latest technology for people, both young and old. Watching the TV for a while, the likelihood is that you will see the magic touch of computer graphics in a commercial.

This book is both a self-contained text and a valuable study aid on the fundamental principles of computer graphics. It takes a goal-oriented approach to discuss the important concepts, the underlying mathematics, and the algorithmic aspects of the computerized image synthesis process. It contains hundreds of solved problems that help reinforce one's understanding of the field and exemplify effective problem-solving techniques.

Although the primary audience are college students taking a computer graphics course in a computer science or computer engineering program, any educated person with a desire to look into the inner workings of computer graphics should be able to learn from this concise introduction. The recommended prerequisites are some working knowledge of a computer system, the equivalent of one or two semesters of programming, a basic understanding of data structures and algorithms, and a basic knowledge of linear algebra and analytical geometry.

The field of computer graphics is characterized by rapid changes in how the technology is used in everyday applications and by constant evolution of graphics systems. The life span of graphics hardware seems to be getting shorter and shorter. An industry standard for computer graphics often becomes obsolete before it is finalized. A programming language that is a popular vehicle for graphics applications when a student begins his or her college study is likely to be on its way out by the time he or she graduates.

In this book we try to cover the key ingredients of computer graphics that tend to have a lasting value (only in relative terms, of course). Instead of compiling highly equipment-specific or computing environment-specific information, we strive to provide a good explanation of the fundamental concepts and the relationship between them. We discuss subject matters in the overall framework of computer graphics and emphasize mathematical and/or algorithmic solutions. Algorithms are presented in pseudo-code rather than a particular programming language. Examples are given with specifics to the extent that they can be easily made into working versions on a particular computer system.

We believe that this approach brings unique benefit to a diverse group of readers. First, the book can be read by itself as a general introduction to computer graphics for people who want technical substance but not the burden of implementational overhead. Second, it can be used by instructors and students as a resource book to supplement any comprehensive primary text. Third, it may serve as a stepping-stone for practitioners who want something that is more understandable than their graphics system's programmer's manuals.

The first edition of this book has served its audience well for over a decade. I would like to salute and thank my coauthors for their invaluable groundwork. The current version represents a significant revision to the original, with several chapters replaced to cover new topics, and the remaining material updated throughout the rest of the book. I hope that it can serve our future audience as well for years to come.

Thank you for choosing our book. May you find it stimulating and rewarding.

ZHIGANG XIANG

CONTENTS

CHAPTER 1

Introduction

Computer graphics is generally regarded as a branch of computer science that deals with the theory and technology for computerized image synthesis. A computer-generated image can depict a scene as simple as the outline of a triangle on a uniform background and as complex as a magnificent dinosaur in a tropical forest. But how do these things become part of the picture? What makes drawing on a computer different from sketching with a pen or photographing with a camera? In this chapter we will introduce some important concepts and outline the relationship among these concepts. The goal of such a mini-survey of the field of computer graphics is to enable us to appreciate the various answers to these questions that we will detail in the rest of the book not only in their own right but also in the context of the overall framework.

1.1 A MINI-SURVEY

First let's consider drawing the outline of a triangle (see Fig. 1-1). In real life this would begin with a decision in our mind regarding such geometric characteristics as the type and size of the triangle, followed by our action to move a pen across a piece of paper. In computer graphics terminology, what we have envisioned is called the object definition, which defines the triangle in an abstract space of our choosing. This space is continuous and is called the *object space*. Our action to draw maps the imaginary object into a triangle on paper, which constitutes a continuous display surface in another space called the *image space*. This mapping action is further influenced by our choice regarding such factors as the location and orientation of the triangle. In other words, we may place the triangle in the middle of the paper, or we may draw it near the upper left corner. We may have the sharp corner of the triangle pointing to the right, or we may have it pointing to the left.

A comparable process takes place when a computer is used to produce the picture. The major computational steps involved in the process give rise to several important areas of computer graphics. The area that attends to the need to define objects, such as the triangle, in an efficient and effective manner is called geometric representation. In our example we can place a two-dimensional Cartesian coordinate system into the object space. The triangle can then be represented by the x and y coordinates of its three vertices, with the understanding that the computer system will connect the first and second vertices with a line segment, the second and third vertices with another line segment, and the third and first with yet another line segment.

The next area of computer graphics that deals with the placement of the triangle is called transformation. Here we use matrices to realize the mapping of the triangle to its final destination in the image space. We can set up the transformation matrix to control the location and orientation of the displayed triangle. We can even enlarge or reduce its size. Furthermore, by using multiple settings for the

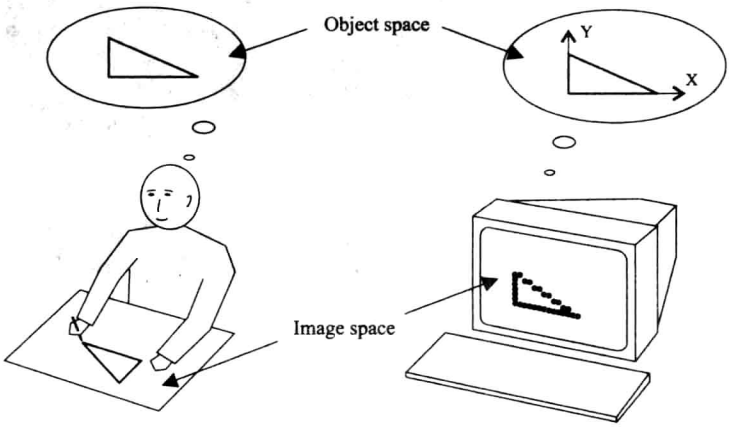

Fig. 1-1 Drawing a triangle.

transformation matrix, we can instruct the computer to display several triangles of varying size and orientation at different locations, all from the same model in the object space.

At this point most readers may have already been wondering about the crucial difference between the triangle drawn on paper and the triangle displayed on the computer monitor (an exaggerated version of what you would see on a real monitor). The former has its vertices connected by smooth edges, whereas the latter is not exactly a line-drawing. The fundamental reason here is that the image space in computer graphics is, generally speaking, not continuous. It consists of a set of discrete pixels, i.e., picture elements, that are arranged in a row-and-column fashion. Hence a horizontal or vertical line segment becomes a group of adjacent pixels in a row or column, respectively, and a slanted line segment becomes something that resembles a staircase. The area of computer graphics that is responsible for converting a continuous figure, such as a line segment, into its discrete approximation is called scan conversion.

The distortion introduced by the conversion from continuous space to discrete space is referred to as the aliasing effect of the conversion. While reducing the size of individual pixels should make the distortion less noticeable, we do so at a significant cost in terms of computational resources. For instance, if we cut each pixel by half in both the horizontal and the vertical direction we would need four times the number of pixels in order to keep the physical dimension of the picture constant. This would translate into, among other things, four times the memory requirement for storing the image. Exploring other ways to alleviate the negative impact of the aliasing effect is the focus of another area of computer graphics called anti-aliasing.

Putting together what we have so far leads to a simplified graphics pipeline (see Fig. 1-2), which exemplifies the architecture of a typical graphics system. At the start of the pipeline, we have primitive objects represented in some application-dependent data structures. For example, the coordinates of the vertices of a triangle, viz., (x_1, y_1), (x_2, y_2), and (x_3, y_3), can be easily stored in a 3×2 array. The graphics system first performs transformation on the original data according to user-specified parameters, and then carries out scan conversion with or without anti-aliasing to put the picture on the screen. The coordinate system in the middle box in Fig. 1-2 serves as an intermediary between the object coordinate system on the

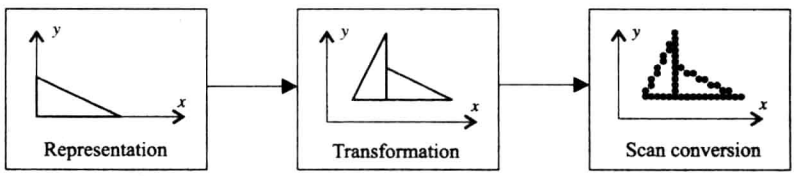

Fig. 1-2 A simple graphics pipeline.

left and the image or device coordinate system on the right. It is called the world coordinate system, representing where we place transformed objects to compose the picture we want to draw. The example in the box shows two triangles: the one on the right is a scaled copy of the original that is moved up and to the right, the one on the left is another scaled copy of the original that is rotated 90° counterclockwise around the origin of the coordinate system and then moved up and to the right in the same way.

In a typical implementation of the graphics pipeline we would write our application program in a host programming language and call library subroutines to perform graphics operations. Some subroutines are used to prescribe, among other things, transformation parameters. Others are used to draw, i.e., to feed original data into the pipeline so current system settings are automatically applied to shape the end product coming out of the pipeline, which is the picture on the screen.

Having looked at the key ingredients of what is called two-dimensional graphics, we now turn our attention to three-dimensional graphics. With the addition of a third dimension one should notice the profound distinction between an object and its picture. Figure 1-3 shows several possible ways to draw a cubic object, but none of the drawings even come close to being the object itself. The drawings simply represent projections of the three-dimensional object onto a two-dimensional display surface. This means that besides three-dimensional representation and transformation, we have an additional area of computer graphics that covers projection methods.

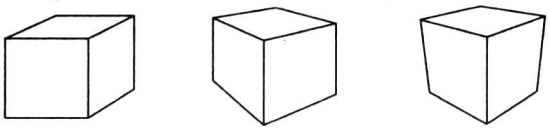

Fig. 1-3 Several ways to depict a cube.

Did you notice that each drawing in Fig. 1-3 shows only three sides of the cubic object? Being a solid three-dimensional object the cube has six plane surfaces. However, we depict it as if we were looking at it in real life. We only draw the surfaces that are visible to us. Surfaces that are obscured from our eyesight are not shown. The area of computer graphics that deals with this computational task is called hidden surface removal. Adding projection and hidden surface removal to our simple graphics pipeline, right after transformation but before scan conversion, results in a prototype for three-dimensional graphics.

Now let's follow up on the idea that we want to produce a picture of an object in real-life fashion. This presents a great challenge for computer graphics, since there is an extremely effective way to produce such a picture: photography. In order to generate a picture that is photo-realistic, i.e., that looks as good as a photograph, we need to explore how a camera and nature work together to produce a snapshot.

When a camera is used to photograph a real-life object illuminated by a light source, light energy coming out of the light source gets reflected from the object surface through the camera lens onto the negative, forming an image of the object. Generally, the part of the object that is closer to the light source should appear brighter in the picture than the part that is further away, and the part of the object that is facing away from the light source should appear relatively dark. Figure 1-4 shows a computer-generated

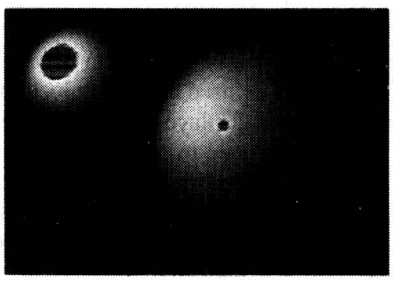

Fig. 1-4 Two shaded spheres.

image that depicts two spherical objects illuminated by a light source that is located somewhere between the spheres and the "camera" at about the ten to eleven o'clock position. Although both spheres have gradual shadings, the bright spot on the large sphere looks like a reflection of the light source and hence suggests a difference in their reflectance property (the large sphere being shinier than the small one). The mathematical formulae that mimic this type of optical phenomenon are referred to as local illumination models, for the energy coming directly from the light source to a particular object surface is not a full account of the energy arriving at that surface. Light energy is also reflected from one object surface to another, and it can go through a transparent or translucent object and continue on to other places. Computational methods that strive to provide a more accurate account of light transport than local illumination models are referred to as global illumination models.

Now take a closer look at Fig. 1-4. The two objects seem to have super-smooth surfaces. What are they made of? How can they be so perfect? Do you see many physical objects around you that exhibit such surface characteristics? Furthermore, it looks like the small sphere is positioned between the light source and the large sphere. Shouldn't we see its shadow on the large sphere? In computer graphics the surface shading variations that distinguish a wood surface from a marble surface or other types of surface are referred to as surface textures. There are various techniques to add surface textures to objects to make them look more realistic. On the other hand, the computational task to include shadows in a picture is called shadow generation.

Before moving on to prepare for a closer look at each of the subject areas we have introduced in this mini-survey, we want to briefly discuss a couple of allied fields of computer science that also deal with graphical information.

Image Processing

The key element that distinguishes image processing (or digital image processing) from computer graphics is that image processing generally begins with images in the image space and performs pixel-based operations on them to produce new images that exhibit certain desired features. For example, we may reset each pixel in the image displayed on the monitor screen in Fig. 1-1 to its complementary color (e.g., black to white and white to black), turning a dark triangle on a white background to a white triangle on a dark background, or vice versa. While each of these two fields has its own focus and strength, they also overlap and complement each other. In fact, stunning visual effects are often achieved by using a combination of computer graphics and image processing techniques.

Computer–Human Interaction

While the main focus of computer graphics is the production of images, the field of computer–human interaction promotes effective communication between man and machine. The two fields join forces when it comes to such areas as graphical user interfaces. There are many kinds of physical devices that can be attached to a computer for the purpose of interaction, starting with the keyboard and the mouse. Each physical device can often be programmed to deliver the function of various logical devices (e.g., Locator, Choice—see below). For example, a mouse can be used to specify locations in the image space (acting as a Locator device). In this case a cursor is often displayed as visual feedback to allow the user see the locations being specified. A mouse can also be used to select an item in a pull-down or pop-up manual (acting as a Choice device). In this case it is the identification of the selected manual item that counts and the item is often highlighted as a whole (the absolute location of the cursor is essentially irrelevant). From these we can see that a physical device may be used in different ways and information can be conveyed to the user in different graphical forms. The key challenge is to design interactive protocols that make effective use of devices and graphics in a way that is user-friendly—easy, intuitive, efficient, etc.

1.2 WHAT'S AHEAD

We hope that our brief flight over the landscape of the graphics kingdom has given you a good impression of some of the important landmarks and made you eager to further your exploration. The following chapters are dedicated to the various subject areas of computer graphics. Each chapter begins with the necessary background information (e.g., context and terminology) and a summary account of the material to be discussed in subsequent sections.

We strive to provide clear explanation and inter-subject continuity in our presentation. Illustrative examples are used freely to substantiate discussion on abstract concepts. While the primary mission of this book is to offer a relatively well-focused introduction to the fundamental theory and underlying technology, significant variations in such matters as basic definitions and implementation protocols are presented in order to have a reasonably broad coverage of the field. In addition, interesting applications are introduced as early as possible to highlight the usefulness of the graphics technology and to encourage those who are eager to engage in hands-on practice.

Algorithms and programming examples are given in pseudo-code that resembles the C programming language, which shares similar syntax and basic constructs with other widely used languages such as C++ and Java. We hope that the relative simplicity of the C-style code presents little grammatical difficulty and hence makes it easy for you to focus your attention on the technical substance of the code.

There are numerous solved problems at the end of each chapter to help reinforce the theoretical discussion. Some of the problems represent computation steps that are omitted in the text and are particularly valuable for those looking for further details and additional explanation. Other problems may provide new information that supplements the main discussion in the text.

CHAPTER 2

Image Representation

A digital image, or image for short, is composed of discrete pixels or picture elements. These pixels are arranged in a row-and-column fashion to form a rectangular picture area, sometimes referred to as a raster. Clearly the total number of pixels in an image is a function of the size of the image and the number of pixels per unit length (e.g. inch) in the horizontal as well as the vertical direction. This number of pixels per unit length is referred to as the resolution of the image. Thus a 3×2 inch image at a resolution of 300 pixels per inch would have a total of 540,000 pixels.

Frequently image size is given as the total number of pixels in the horizontal direction times the total number of pixels in the vertical direction (e.g., 512×512, 640×480, or 1024×768). Although this convention makes it relatively straightforward to gauge the total number of pixels in an image, it does not specify the size of the image or its resolution, as defined in the paragraph above. A 640×480 image would measure $6\frac{2}{3}$ inches by 5 inches when presented (e.g., displayed or printed) at 96 pixels per inch. On the other hand, it would measure 1.6 inches by 1.2 inches at 400 pixels per inch.

The ratio of an image's width to its height, measured in unit length or number of pixels, is referred to as its aspect ratio. Both a 2×2 inch image and a 512×512 image have an aspect ratio of 1/1, whereas both a $6 \times 4\frac{1}{2}$ inch image and a 1024×768 image have an aspect ratio of 4/3.

Individual pixels in an image can be referenced by their coordinates. Typically the pixel at the lower left corner of an image is considered to be at the origin $(0, 0)$ of a pixel coordinate system. Thus the pixel at the lower right corner of a 640×480 image would have coordinates $(639, 0)$, whereas the pixel at the upper right corner would have coordinates $(639, 479)$.

The task of composing an image on a computer is essentially a matter of setting pixel values. The collective effects of the pixels taking on different color attributes give us what we see as a picture. In this chapter we first introduce the basics of the most prevailing color specification method in computer graphics (Sect. 2.1). We then discuss the representation of images using direct coding of pixel colors (Sect. 2.2) versus using the lookup-table approach (Sect. 2.3). Following a discussion of the working principles of two representative image presentation devices, the display monitor (Sect. 2.4) and the printer (Sect. 2.5), we examine image files as the primary means of image storage and transmission (Sect. 2.6). We then take a look at some of the most primitive graphics operations, which primarily deal with setting the color attributes of pixels (Sect. 2.7). Finally, to illustrate the construction of beautiful images directly in the discrete image space, we introduce the mathematical background and detail the algorithmic aspects of visualizing the Mandelbrot set (Sect. 2.8).

2.1 THE RGB COLOR MODEL

Color is a complex, interdisciplinary subject spanning from physics to psychology. In this section we only introduce the basics of the most widely used color representation method in computer graphics. We will have additional discussion later in another chapter.

Figure 2-1 shows a color coordinate system with three primary colors: R (red), G (green), and B (blue). Each primary color can take on an intensity value ranging from 0 (off—lowest) to 1 (on—highest). Mixing these three primary colors at different intensity levels produces a variety of colors. The collection of all the colors obtainable by such a linear combination of red, green, and blue forms the cube-shaped RGB color space. The corner of the RGB color cube that is at the origin of the coordinate system corresponds to black, whereas the corner of the cube that is diagonally opposite to the origin represents white. The diagonal line connecting black and white corresponds to all the gray colors between black and white. It is called the gray axis.

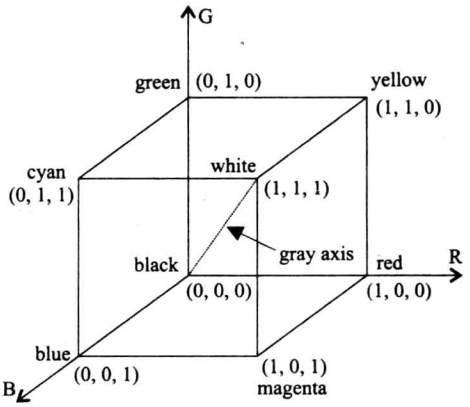

Fig. 2-1 The RGB color space.

Given this RGB color model an arbitrary color within the cubic color space can be specified by its color coordinates: (r, g, b). For example, we have $(0, 0, 0)$ for black, $(1, 1, 1)$ for white, $(1, 1, 0)$ for yellow, etc. A gray color at $(0.7, 0.7, 0.7)$ has an intensity halfway between one at $(0.9, 0.9, 0.9)$ and one at $(0.5, 0.5, 0.5)$.

Color specification using the RGB model is an additive process. We begin with black and add on the appropriate primary components to yield a desired color. This closely matches the working principles of the display monitor (see Sect. 2.4). On the other hand, there is a complementary color model, called the CMY color model, that defines colors using a subtractive process, which closely matches the working principles of the printer (see Sect. 2.5).

In the CMY model we begin with white and take away the appropriate primary components to yield a desired color. For example, if we subtract red from white, what remains consists of green and blue, which is cyan. Looking at this from another perspective, we can use the amount of cyan, the complementary color of red, to control the amount of red, which is equal to one minus the amount of cyan. Figure 2-2 shows a coordinate system using the three primaries' complementary colors: C (cyan), M (magenta), and Y (yellow). The corner of the CMY color cube that is at $(0, 0, 0)$ corresponds to white, whereas the corner of the cube that is at $(1, 1, 1)$ represents black (no red, no green, no blue). The following formulas summarize the conversion between the two color models:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} C \\ M \\ Y \end{pmatrix} \qquad \begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

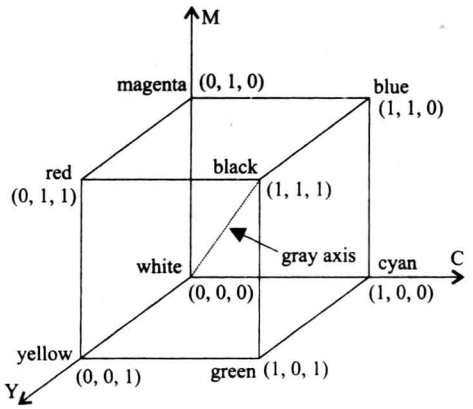

Fig. 2-2 The CMY color space.

2.2 DIRECT CODING

Image representation is essentially the representation of pixel colors. Using direct coding we allocate a certain amount of storage space for each pixel to code its color. For example, we may allocate 3 bits for each pixel, with one bit for each primary color (see Fig. 2-3). This 3-bit representation allows each primary to vary independently between two intensity levels: 0 (off) or 1 (on). Hence each pixel can take on one of the eight colors that correspond to the corners of the RGB color cube.

bit 1: r	bit 2: g	bit 3: b	color name
0	0	0	black
0	0	1	blue
0	1	0	green
0	1	1	cyan
1	0	0	red
1	0	1	magenta
1	1	0	yellow
1	1	1	white

Fig. 2-3 Direct coding of colors using 3 bits.

A widely accepted industry standard uses 3 bytes, or 24 bits, per pixel, with one byte for each primary color. This way we allow each primary color to have 256 different intensity levels, corresponding to binary values from 00000000 to 11111111. Thus a pixel can take on a color from $256 \times 256 \times 256$ or 16.7 million possible choices. This 24-bit format is commonly referred to as the true color representation, for the difference between two colors that differ by one intensity level in one or more of the primaries is virtually undetectable under normal viewing conditions. Hence a more precise representation involving more bits is of little use in terms of perceived color accuracy.

A notable special case of direct coding is the representation of black-and-white (bilevel) and gray-scale images, where the three primaries always have the same value and hence need not be coded separately. A black-and-white image requires only one bit per pixel, with bit value 0 representing black and 1 representing white. A gray-scale image is typically coded with 8 bits per pixel to allow a total of 256 intensity or gray levels.

Although this direct coding method features simplicity and has supported a variety of applications, we can see a relatively high demand for storage space when it comes to the 24-bit standard. For example, a 1000×1000 true color image would take up three million bytes. Furthermore, even if every pixel in that

image had a different color, there would only be one million colors in the image. In many applications the number of colors that appear in any one particular image is much less. Therefore the 24-bit representation's ability to have 16.7 million different colors appear simultaneously in a single image seems to be somewhat overkill.

2.3 LOOKUP TABLE

Image representation using a lookup table can be viewed as a compromise between our desire to have a lower storage requirement and our need to support a reasonably sufficient number of simultaneous colors. In this approach pixel values do not code colors directly. Instead, they are addresses or indices into a table of color values. The color of a particular pixel is determined by the color value in the table entry that the value of the pixel references.

Figure 2-4 shows a lookup table with 256 entries. The entries have addresses 0 through 255. Each entry contains a 24-bit RGB color value. Pixel values are now 1-byte, or 8-bit, quantities. The color of a pixel whose value is i, where $0 \leq i \leq 255$, is determined by the color value in the table entry whose address is i. This 24-bit 256-entry lookup table representation is often referred to as the 8-bit format. It reduces the storage requirement of a 1000×1000 image to one million bytes plus 768 bytes for the color values in the lookup table. It allows 256 simultaneous colors that are chosen from 16.7 million possible colors.

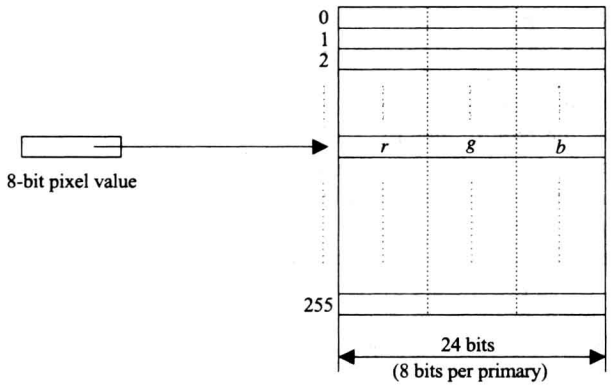

Fig. 2-4 A 24-bit 256-entry lookup table.

It is important to remember that, using the lookup table representation, an image is defined not only by its pixel values but also by the color values in the corresponding lookup table. Those color values form a *color map* for the image.

2.4 DISPLAY MONITOR

Among the numerous types of image presentation or output devices that convert digitally represented images into visually perceivable pictures is the display or video monitor.

We first take a look at the working principle of a monochromatic display monitor, which consists mainly of a cathode ray tube (CRT) along with related control circuits. The CRT is a vacuum glass tube with the display screen at one end and connectors to the control circuits at the other (see Fig. 2-5). Coated on the inside of the display screen is a special material, called phosphor, which emits light for a period of time when hit by a beam of electrons. The color of the light and the time period vary from one type of

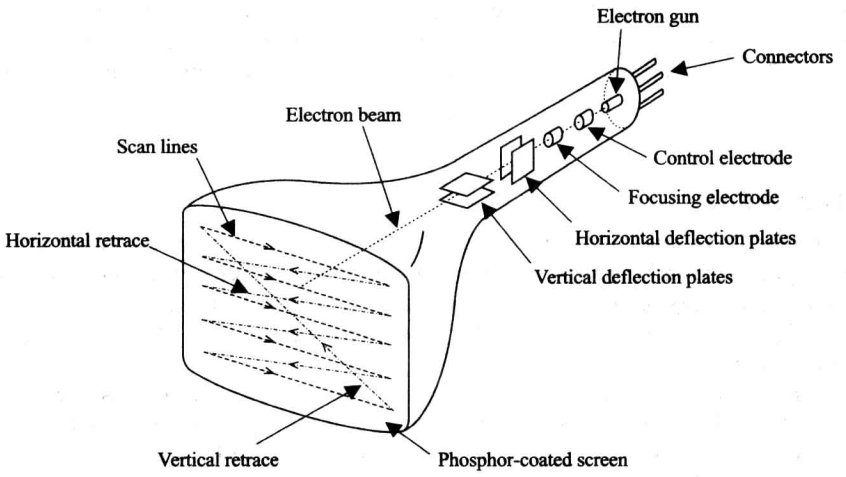

Fig. 2-5 Anatomy of a monochromatic CRT.

phosphor to another. The light given off by the phosphor during exposure to the electron beam is known as fluorescence, the continuing glow given off after the beam is removed is known as phosphorescence, and the duration of phosphorescence is known as the phosphor's persistence.

Opposite to the phosphor-coated screen is an electron gun that is heated to send out electrons. The electrons are regulated by the control electrode and forced by the focusing electrode into a narrow beam striking the phosphor coating at small spots. When this electron beam passes through the horizontal and vertical deflection plates, it is bent or deflected by the electric fields between the plates. The horizontal plates control the beam to scan from left to right and retrace from right to left. The vertical plates control the beam to go from the first scan line at the top to the last scan line at the bottom and retrace from the bottom back to the top. These actions are synchronized by the control circuits so that the electron beam strikes each and every pixel position in a scan line by scan line fashion. As an alternative to this electrostatic deflection method, some CRTs use magnetic deflection coils mounted on the outside of the glass envelope to bend the electron beam with magnetic fields.

The intensity of the light emitted by the phosphor coating is a function of the intensity of the electron beam. The control circuits shut off the electron beam during horizontal and vertical retraces. The intensity of the beam at a particular pixel position is determined by the intensity value of the corresponding pixel in the image being displayed.

The image being displayed is stored in a dedicated system memory area that is often referred to as the frame buffer or refresh buffer. The control circuits associated with the frame buffer generate proper video signals for the display monitor. The frequency at which the content of the frame buffer is sent to the display monitor is called the refreshing rate, which is typically 60 times or frames per second (60 Hz) or higher. A determining factor here is the need to avoid flicker, which occurs at lower refreshing rates when our visual system is unable to integrate the light impulses from the phosphor dots into a steady picture. The persistence of the monitor's phosphor, on the other hand, needs to be long enough for a frame to remain visible but short enough for it to fade before the next frame is displayed.

Some monitors use a technique called interlacing to "double" their refreshing rate. In this case only half of the scan lines in a frame is refreshed at a time, first the odd numbered lines, then the even numbered lines. Thus the screen is refreshed from top to bottom in half the time it would have taken to sweep across all the scan lines. Although this approach does not really increase the rate at which the entire screen is refreshed, it is quite effective in reducing flicker.