Chapter 6: Visual Communication

This chapter will introduce you to some basic concepts in communicating with an audience through computer graphics. It will discuss the use of shapes, color, lighting, viewing, labels and legends, and motion, present issues in creating meaningful interaction and taking cultural differences into account in communication, and describe several techniques for presenting higher-dimensional information to the user. These discussions use a number of examples, and the chapter shows you how to implement many of these in OpenGL. This is something of an eclectic collection of information but it will give you valuable background on communication that can help you create more effective computer graphics programs.

When you have completed this chapter, you should be able to see the difference between effective and ineffective graphics presentations and be able to identify many of the graphical techniques that make a graphic presentation effective. You should also be able to implement these techniques in your own graphics programming. In order to benefit from this chapter, you need an understanding of the basic concepts of communication, of visual vocabularies for different kinds of users, and of shaping information with the knowledge of the audience and with certain goals. You also need enough computer graphics skills to understand how these communications are created and to create them yourself.

Introduction

Computer graphics has achieved remarkable things in communicating information to specialists, to informed communities, and to the public at large. This is different from the entertainment areas where computer graphics gets a lot of press because it has the goal of helping the user of the interactive system or the viewer of a well-developed presentation to have a deeper understanding of a complex topic. The range of subjects for this communication include cosmology, in showing how fundamental structures in the universe work; archaeology and anthropology, in showing the ways earlier human groups laid out their structures and cultures; biology and chemistry, in seeing the way electrostatic forces and molecular structures lead to molecular bonding; mathematics, in considering the behavior of highly unstable differential equations; or meteorology, in examining the way global forces such as the temperatures of ocean currents or the depth of the ozone layer affect the weather.

While the importance of visual communication and its associated visual vocabularies has been known by artists, designers, and film directors for a long time, its role in the use of computing in the sciences was highlighted in the 1987 report on Visualization in Scientific Computing [ViSC]. That report noted the importance of computer graphics in engaging the human brain’s extraordinary ability to create insight from images. That report noted that Richard Hamming’s 1962 quote, “The purpose if computing is insight, not numbers,” is particularly applicable when the computing can create images that lead to a deeper and more subtle insight into complex subjects than is possible without images. Indeed, for the student using these notes, we would paraphrase Hamming and say that our purpose for computer graphics is information, not images.

The process of making images—in particular, of making attractive and interesting images with computer graphics using powerful machines and a capable graphics API—is relatively easy. The
difficult part of effective computer graphics is the task of understanding your problem and developing ways to present the information that describes your problem so you can make images that communicate with your audience. This short section talks about this task and gives some principles and examples that we hope can start you thinking about this question, but it is a significant task to develop real skill in communicating by means of images. This chapter is relatively early in the overall presentation of graphics primarily to remind you that the main reason we use graphics is to communicate with others, and to help you keep that communication in mind as you learn about making images with computing. Some of the techniques we talk about here will not be covered until later in the notes, but they are not terribly complex, so you should be able to make sense of what the techniques mean even before you have learned how to make them work.

There are several key concepts in the area of communicating effectively through your images. In this chapter we will discuss several techniques and will consider their effect upon communication, but you must realize that this is only an introduction to the topic. Highly-skilled communicators are constantly inventing new ways to present specific information to specific audiences, creating in effect new visual vocabularies for these audiences. We do not try to give you the last answer in visual communication; instead, we are trying to get you to think about the information content of your images and about how you can communicate that to your particular audience. Only a great deal of experience and observation will make you genuinely skilled in this area.

In addition to effective images, visual communication can also include designing interactions so that they give effective support for the visual effects controlled by the interactions. Motion, selection, and control of graphical processes are all reflected in the images presented to the user, so we will discuss some ways you can design the way a user interacts with your programs to support effective and comfortable work.

*General issues in visual communication*

There are some general points in communicating with your audience that are so important that we want to highlight them here, before we begin looking at the details of communicating with your audience, and discuss some of the issues that are involved in carrying them out.

*Use appropriate representation for your information* so that your audience will be able to get the most meaning from your images. Sometimes this representation can use color, or sometimes it can use geometry or shapes. Sometimes it will use highly symbolic or synthetic images while sometimes it will use highly naturalistic images. Sometimes it will present the relationships between things instead of the things themselves. Sometimes it will use purely two-dimensional representations, sometimes three-dimensional images but with the third dimension used only for impact, and sometimes three-dimensional images with the third dimension a critical part of the presentation. In fact, there are an enormous number of ways to create representations of information, and the best way to know what works for your audience is probably to observe the way they are used to seeing things and ask them what makes sense for them, probably by showing them many examples of options and alternatives. Do not assume that you can know
what they should use, however, because you probably think differently from people in their field and are probably not the one who needs to get the information from the images.

**Keep your images focused** on just the information that is needed to understand the things you are trying to communicate. In creating the focus you need, remember that simple images create focus by eliminating extraneous or distracting content. Don’t create images that are “eye candy” and simply look good; don’t create images that suggest relationships or information that are not in the information. For example, when you represent experimental data with geometric figures, use flat shading instead of smooth shading and use only the resolution your data supports because creating higher resolution with smooth interpolation processes, because using smooth shading or smooth interpolation suggests that you know more than your data supports. The fundamental principle is to be very careful not to distort the truth of your information in order to create a more attractive image.

**Use appropriate presentation levels for your information.** There is a very useful concept that may help you determine how much effort you put into polishing your images. This concept suggests that there are three levels of information presentation: for yourself (personal), for your colleagues or collaborators (peer), and for an audience when you want to make an impression (presentation). Most of the time when you're trying to understand something yourself, you can use very simple images because you know what you are trying to show with them; you need not polish the images but can simply include essential material. When you are sharing your work with your colleagues who have an idea of what you’re working on but who don’t have the depth of knowledge in the particular problem you’re addressing, you might want a bit higher quality or perhaps a simple legend to help them see your point, but you don't need to spend a lot of time polishing your work. But when you are creating a public presentation such as a scientific paper or a grant proposal (think of how you would get a point across to a Congressional committee, for example!) you will need to make your work as highly-polished as you can. So your work will sometimes be simple and low-resolution, with very sketchy images; sometimes smoother and with a little thought to how your look at things, perhaps with a little simple animation or with some interaction to let people play with your ideas; and sometimes fully developed, with very smooth animation and high-resolution images, with great care taken to make the maximum impact in the minimum time.

**Use appropriate forms for your information.** There is a very useful categorization of information (or data) into interval data, ordinal data, and nominal data. Interval data is data that is associated with a meaningful number such as speed, weight, or count. This data is meaningful in individual cases and has a natural representation in real numbers that you can use for your graphical presentation. Ordinal data is data that can be compared with other similar data but that is not necessarily meaningful in itself. Thus the educational level of an individual can be compared with that of another individual, but the most you can usually say with any meaning is that one person has more (or less) education than another. Ordinal data can be represented by size or by color in color ramps that are discussed below. Note that for a color ramp, a viewer can tell which color is higher or lower than another on a color legend but is usually not able to get a careful numeric value from the color. Nominal data is data that describes something with no ordering or numerical meaning; an example might be hair color, with descriptions “red”, “brown”, “blonde”, or “gray”. Nominal data can be shown by shapes or individual distinct
colors. When you consider your display for a problem, you need to understand what kind of data or information you are working with so you can use an appropriate form for that kind of data.

**Be very careful to be accurate with your display.** If you have only scattered data for a topic, show only the data you have; do not use techniques such as smooth geometry or smooth coloring to suggest that information is known between the data points. Recognize that a simpler, more accurate representation is going to give your user better information for understanding the situation than a fancier representation. If you use simple numerical techniques, such as difference equations, to model a behavior or to determine the motions in your geometry, say this in a legend or title instead of implying that a more exact solution is presented, so that the modeling you present is not taken as exact. In general, try very hard not to lie with your presentation, whether that lie should be an artifact of your modeling or coding or is an attempt to spin your data to support a particular point of view.

**Understand and respect the cultural context of your audience.** When you create images to communicate with an audience, that audience can only understand the images in their own context. This context comes as part of their culture, which might be a professional culture (engineering, medicine, high-energy physics, publishing, education, management ...), a social culture (small-town, major urban, agricultural, rock, ...), a geographical culture (North American, western European, Chinese, Japanese, ...), an ethnic culture (Native American, Zulu, Chicano, ...), or a religious culture (Buddhist, Roman Catholic, fundamentalist Protestant, ...). In each of these you will find some symbols or other visual representations that have a meaning that might be significantly different from the meanings of these same things in other cultural contexts. You must make a careful effort to ensure that your images have the meaning you intend for their audience and not some accidental message caused by your failure to understand the context in which the images are received.

**Make your interactions reflect familiar and comfortable relationships between action and effect.** If your users are used to a particular kind of controls, simulate those controls and make the actions given by a control mimic the behavior that the control provides in the user’s world. The controls you have available are probably familiar from various applications that most people have used, so you can begin by looking at these examples. Some of the controls would naturally be presented in their own control panel; some would be applied in the image itself.

Let’s start by looking at some of the control panel vocabulary for interactions. If you want a user to select one among several distinct options in a program, it is natural to make that selection from a radio-button list — a list of the options with a button by each, with each button displaying whether it has been selected, and with a selection on one button canceling any selection on any other button. If you want a user to select zero or more options from a list of options that are not mutually exclusive, then you can use an ordinary button for each, with each button being selected independently of the others. If you want to set a value for a parameter from a continuous range of possible values, then you can use a slider, dial, or thumbwheel to select the value (and you should display the value of the device as it is changed so the user will know what value has been selected). Finally, if you want your user to enter some text (for example, the name of a color if you are using a color naming system instead of RGB values, or the name of a
file the program is to use), then you can use a text box to allow text entry. Examples of these controls are displayed in the chapter below on interaction.

If you want the controls to be applied directly in the scene, however, then there is another vocabulary for that kind of interaction. This vocabulary depends on the understanding that the scene represents a certain kind of space, usually a three-dimensional space containing models of physical objects, and that behaviors can be specified in terms of that space. Here we will find object selection by clicking a mouse on a scene and identifying the object that the mouse click has identified, for example. We will also find that we can specify rotations in both latitude and longitude by clicking the mouse in the scene and holding down the button while moving the mouse vertically or horizontally, respectively. We will also find that we can zoom into a scene by specifying that a mouse motion as seen above should be interpreted as a one-dimensional zoom instead of as a two-dimensional rotation. And finally, any of these behaviors could be replaced by using the keyboard to convey the same information, taking advantage of the very large number of degrees of freedom represented by all the keys, of the semantic strength of identifying actions with words and words with their leading characters, and of any familiar key patterns such as the cursor controls from text editors or motion controls from older games.

In all, recognizing that interaction is another form of language and that there are vocabularies from users’ backgrounds provides a form of communication that is often visual and that can be built upon to create effective interactive programs.

This chapter makes a number of points about visual communication and computer graphics. These include:

- Shape is a powerful tool that needs to be used carefully to present accurate ideas,
- Color offers many choices that are critical to effective images, and more than anything else, color is how you guide your audience to important parts of your images and convey the exact information you need to give them,
- The choice between naturalistic or artificial shapes and colors is important in communicating appropriate information to your audience,
- There are techniques of color and motion that can convey up to five dimensions of information to your audience, but you need to think carefully about how you encode different kinds of information into each dimension
- An audience may need to see your scene from the right viewpoint and have the right extra information with the image in order to understand the ideas you are presenting,
- Modern graphics APIs and computers have made animated images much simpler to create, and it can be very helpful to take advantage of motion to make your images work better,
- Your images not only can move, but you can create images that allow your audience to interact with the information you are presenting and so can explore the ideas themselves, and
- Your audience will always see your images in the context of their culture, so you must develop an understanding of that culture as part of designing your work.

Together these make up the design issues you must keep in mind as you create your images. They may not all apply in any single piece of work, some take a good bit of work that is outside computer graphics, and many of them take some experience before you can apply them confidently and skillfully. But if you understand their importance, you will create much more effective work.
Shape

Shape is the fundamental part of any image, because with an API-based approach to computer graphics, all our images are built from modeling, and modeling is based on creating geometric objects that have shape. Visual communication begins with images, and thus begins with shapes.

As you saw in the chapter on modeling, there are all sorts of shapes and all sorts of arrangements of those shapes available to you as you create images. You may use simple shapes, emphasizing a basic simplicity in the ideas you are communicating. There is a visual clarity to an image built of simple (or at least apparently simple) shapes; the image will seem uncluttered and it will be easy for your viewer to see what you are presenting.

Shapes are not used arbitrarily, of course. Sometimes your images will describe physical objects, and you will want to create shapes that represent those objects, either by being an accurate version of the objects’ shapes or by representing the objects in a recognizable but simplified way. These shapes may be smooth, if you know your theoretical or data values change smoothly across the display, or they may be coarse and irregular if you have only discrete data and do not know that it is correct to display it smoothly. Sometimes your images will represent concepts that are not embodied in physical objects, and in those cases you will want to create abstract shapes that give your viewer something to represent the ideas you are presenting. These shapes, then, can carry ideas through their position and their properties such as size, shape, color, or motion.

Be careful about the cultural context of shapes you might use as symbols. If you need to use a simple shape to show the position of a data point, for example, you could use a circle, a square, a cross, a pentagram, or a hexagram. A moment’s reflection would suggest that the latter three shapes have cultural associations that may or may not be appropriate for your use. In general, be sensitive to this aspect of shapes and recognize that a choice that seems simple to you may have a strong impact on someone else.

For several examples where we will be considering the use of shapes to represent information and comparing that with the use of color, we will use Coulomb’s law of electrostatic potential. This law states that at each point of a plane having several point charges, the potential at the point is the sum of the potentials at the point from each of the point charges, and the potential at any point is the point charge divided by the square of the distance between the point and the charge, or:

\[
P(x,y) = \sum \frac{Q_i}{\sqrt{(x-x_i)^2+(y-y_i)^2}}
\]

where each charge \(Q_i\) is positioned at point \((x_i, y_i)\). This example is discussed in more detail in the chapter on science applications, but we will consider some representations of the problem as examples of visual presentation options.
If we present the effect of Coulomb’s law on a rectangular surface with three fixed point charges, one positive and two negative, we could create the graph of a function of two variables over a domain of the real plane by presenting a surface in three dimensions. In Figure 6.1, we show such a presentation of the function by presenting its 3D surface graph purely as a surface in a fairly traditional way, with an emphasis of the shape of the surface itself. If the emphasis is on the surface itself, this might be a good way to present the graph, because as you can see, the lighting shows the shape well. This is a smooth shape because we are presenting a theory that operates continuously across the space, and the lighting helps to show that shape. The only lie we are telling with this surface is that the surface is bounded, when in fact there are discontinuities at the points where the fixed charges lie because at these points a denominator in the equation is zero. This view shows two minima and one maximum for the surface, corresponding to two negatively charged points and one positively charged point. The viewpoint that is used allows us to see these three points and shows us a fairly good representation of the shape, but another view might hide the smaller of the minima or details of the region between the two minima; here the viewpoint is critical to getting the best understanding of the surface.

![Figure 6.1: traditional surface model presented with three lights to show its shape](image)

There is a potential problem with this kind of representation, depending on the experience of the viewer. The viewer must be familiar with surface representations of functions because he or she must understand that there is no actual surface present; instead there is a value that is represented as the height. So this is not necessarily a good representation for novice users or for use when there is no other discussion of the situation being presented.

**Comparing shape and color codings**

In the next section of this chapter we will discuss the use of color in communicating with your audience, and will spend some time talking about representing information in terms of color. The three-part Figure 6.2 illustrates three different ways to represent temperatures for the heat diffusion program introduced in the first chapter. In this problem, heat is added to a metal bar at two points and is taken from the bar at another. The images produced by the program use higher bars and redder values to represent higher temperatures, while lower bars and bluer values represent lower temperatures. The figure shows the state of the temperatures in the bar using both geometric and color encoding (center), using only geometric coding (left), and using only
color encoding (right). Note that each of the encodings has its value, but that they emphasize different things. In particular, the height encoding tends to imply that the bar itself actually has a different geometry than a simple rectangular bar, so as we discussed above it might confuse someone who is not used to the substitution of geometry for numeric values. The color-only encoding, however, seems easier for a novice to understand because we are used to color coding for heat (think of hot and cold water faucets with different colored inlays) and metal changes color to read when it is sufficiently heated. Thus the way we encode information may depend on the experience of our users and on the conventions they are accustomed to in understanding information. The color encoding in this case follows a traditional blue for cold and red for hot that is common in western culture. Other encodings are found in other groups and could be used in this program; for example, some branches of engineering commonly use a full rainbow code for colors from magenta for cold, through blue to green and yellow, finally ending with red for hot.

Figure 6.2: three encodings of the same information: temperature in a bar, encoded only through geometry (top left), only through color (bottom right), and through both (center)

Color

Color is one of the most important tools you have in creating effective communications with computer graphics. It enriches the image and attracts the eye, and it gives you more tools to use in making your points with your audience. As we will see later in this chapter, it can even serve to give you an additional dimension for your images. However, if it is misused, color can work against effective images, so you must be careful about using it; your goal is to create a color scheme that makes apparent sense to the viewer, even if it is completely artificial. In this section we describe many approaches to the question of using appropriate color to help you think about the meaning of color and how you can use it to create strong and effective communication.
Emphasis colors

Your overall image will include whatever information you want to present to the viewer, including context information and various kinds of details that depend on your application. As you design your image, however, you may well want to draw the viewer’s attention to specific points in order to show specific things about the content of the display.

There are many ways to draw attention to a specific feature of your scene, but one effective technique is to use a strong, contrasting color for that feature. Such a color will probably be bright and clear, and will be chosen to stand out from the other colors in the overall scene. If you want to have this kind of emphasis, you need to do two things: to design the scene with somewhat muted colors, so that a bright color can stand out from it, and to choose one or more emphasis colors that contrast strongly to the overall colors and are quickly attractive to the eye.

As an example of this, Figure 6.3 shows a surface with a number of control points, and one of the control points is highlighted. Here the background, pool top and bottom, and normal control points are in rather muted colors, but the highlighted control point is in red and can easily be identified as different.

![Figure 6.3: an image with one item highlighted for emphasis](image)

Background colors

Images include more than just the objects that are being emphasized. They also include a background color that fills the display space in any area not containing your working image. Background colors should be colors that recede in a user’s perception so that the objects to be emphasized stand out against them. In general, a good background color is a dark or neutral color, but black is usually a poor choice because anything that is dark will fade into it. White can be a good background color, because it is certainly neutral although it is not dark. However, just like black, if there are objects that are too light, they may now show up well against white.
If you look at professional photographers and videographers, you will note that they don’t use a constant-color background. They use a background with a central highlight that focuses attention on the subject. You should try to use a brighter spot at the center of your background, or possibly a brighter slash of light through the background (usually lower left to upper right) to pull the eye to the center of the image where you’ll put your critical content. This idea, and others from photo and video professionals, can help you focus your viewer’s attention on the critical part of your images.

**Color deficiencies in audience**

As you work with color, you must keep in mind that a significant portion of your audience will have difficulties distinguishing certain color combinations. Between 8 and 10 per cent of Caucasian males have a color-vision deficiency; this number is about 4 per cent for non-Caucasian males and only about 0.5 per cent for females. These persons confuse colors, whatever kind of display is used, but most can distinguish colors that differ in luminance, even if they cannot distinguish some differences in chroma. Recall that the luminance of a color is given by the luminance equation:

\[ \text{luminance} = 0.30*\text{red} + 0.59*\text{green} + 0.11*\text{blue} \]

(where red, green, and blue have the usual convention of lying between zero and one). If your audience will include significant numbers of Caucasian males, you should try to be sure that elements of your image that your audience needs to distinguish are presented with colors having different luminance, not just different chroma. We will visit this idea further when we talk about color ramps below.

**Naturalistic color**

If you are working with images of actual objects, you will often want to make those objects seem as realistic as possible to the user. When you do, you have the full capability of your API’s modeling, lighting, and shading tools to draw on to create appropriately colored images. You can also apply appropriate texture mapping to give an extra air of realism to your display. These are all discussed in separate chapters in these notes, so we will not discuss them further here.

**Pseudocolor and color ramps**

Color can mean much more than the actual color of objects. One of the very important uses of color is in serving as a representation of a parameter of the objects being displayed. This parameter could be temperature, velocity, distance, or almost anything you could think of as represented by a number. The value of this numerical parameter can be translated to a color, and then the display of the object in this color carries the information that the object actually has the value represented by the color. This color that represents another value is called a *pseudocolor*.

Pseudocolor separates the concepts of the shape of an object and the colors of the object. It does not try to create a realistic image of the object but applies colors to show some other properties of the object. In Figure 6.4, a house is shown at left in its normal view, and at right with the heat it emits. While this particular pair of images was created with thermal imaging, the principle of showing a property along with the shape is the key to pseudocolor imaging.
This kind of representation of values by colors is managed by creating color ramps, which are one-dimensional sequences of colors with a mapping of any value between 0.0 and 1.0 into a particular color. Color ramps provide the linkage of color to numeric value, so they must be chosen carefully to help the user understand the numeric values the colors display. These ramps may use colors that are customary in the field whose content is being displayed, so there are aspects of cultural context for the ramps. They may be developed to show changes smoothly or to have strong boundaries, depending on the meaning they are to convey. There is something of an art to deciding on the relation between colors and values for a particular application, and you are encouraged to design your applications so that you can change color ramps easily, allowing you to experiment with different ramps and the different meanings they can convey.

Implementing color ramps

Implementing color ramps is straightforward and is independent of your graphics API. We include some sample code here to show how two color ramps were created so you can adapt these ramps (or similar ones) to your projects. Each assumes that the numerical values have been scaled to this range and returns an array of three numbers that represents the RGB color that corresponds to that value according to the particular representation is uses. This code is independent of the graphics API you are using, so long as the API uses the RGB color model. The first ramp provides color in a rainbow sequence, red-orange-yellow-green-blue-violet. A particularly useful kind of color ramp uses colors that vary uniformly in luminance as the ramp’s values range in [0.0, 1.0]. Recalling from the discussion above that luminance is given by

\[ \text{luminance} = 0.30 \times \text{red} + 0.59 \times \text{green} + 0.11 \times \text{blue} \]

we may use any ramp that is linear in these values. The second ramp has uniform luminance and runs from black through red through yellow to white. Other uniform-luminance color sequences are also possible, and are discussed in the exercises for this chapter.

```c
void calcRainbow(float yval)
{/n   if (yval < 0.2) // purple to blue ramp
       [myColor[0]=0.5*(1.0-yval/0.2);myColor[1]=0.0;
       myColor[2]=0.5+(0.5*yval/0.2);return;]
   if ((yval >= 0.2) && (yval < 0.40)) // blue to cyan ramp
       [myColor[0]=0.0;myColor[1]=(yval-0.2)*5.0;myColor[2]=1.0;return;]
   if ((yval >= 0.40) && (yval < 0.6)) // cyan to green ramp
       [myColor[0]=0.0;myColor[1]=1.0;myColor[2]=(0.6-yval)*5.0;return;]
```
if ((yval >= 0.6) && (yval < 0.8)) // green to yellow ramp
    {myColor[0]=(yval-0.6)*5.0;myColor[1]=1.0;myColor[2]=0.0;return;}
if (yval >= 0.8) // yellow to red ramp^  
    {myColor[0]=1.0;myColor[1]=(1.0-yval)*5.0;myColor[2]=0.0;}
return;
}

void calcLuminance(float yval)
{
    if (yval < 0.30)
        {myColor[0]=yval/0.3;myColor[1]=0.0;myColor[2]=0.0;return;}
    if ((yval>=0.30) && (yval < 0.89))
        {myColor[0]=1.0;myColor[1]=(yval-0.3)/0.59;myColor[2]=0.0;return;}
    if (yval>=0.89)
        {myColor[0]=1.0;myColor[1]=1.0;myColor[2]=(yval-0.89)/0.11;}
    return;
}

The color ramp that is used in the thermal imaging of Figure 6.4 is different from either of these, because in the interval from 0 to 13.9 (degrees Celsius) it runs from black (lowest value) through dark blue to magenta, then to red and on to yellow, and finally to white. This could be implemented by segmenting the range from 0 to about 6 as a ramp from black to blue, then adding red and reducing blue from 6 to 9, then adding yellow to about 12, and then adding blue on up to 13.9. Here blue is used for the low values because blue is a customary color for cold or cool things, and red is added as blue is decreased because redder colors are associated with warmer things. Finally the red is moved to yellow by adding green, and yellow is moved to white by adding blue, because as metals are heated their colors move from red to yellow and finally to white (as in “white-hot”). Thus the color ramp uses a customary representation of temperatures in Western cultures.

Using color ramps

A color ramp represents a one-dimensional space whose values are colors, not numbers. It sets up a relationship between any number between 0.0 and 1.0 and a unique color determined by the way the ramp is coded. As we saw above, the color is used as a surrogate that makes a numerical value visible, so when you want to show an object that has a particular value, you display it in the color that represents that value through the color ramp. You can use this color as an absolute color in a display that does not use lighting, or as a material color in a display that uses lighting; once the color has replaced the numeric value, you can treat the colored model in any way you wish.

At the end of the discussion on shapes, we described an example that showed how shapes or colors could be used to encode information. That example used a simple blue-to-red color change without any explicit use of color ramps. A combination of colors and shapes can also be used with other kinds of examples, so let us consider the Coulomb’s law example with the surface shown in Figure 6.1. There the electrostatic potential in a rectangle was shown as a pure surface, but if we consider the use of color ramps to show different values, we get the figures shown in Figure 6.5, with the left-hand image showing the surface with the rainbow color ramp and the right-hand image showing a uniform luminance ramp. You should look at these two images carefully to see whether you can figure out the numeric value of the electrostatic
potential at a given point in each. In an actual application, you would not simply display the colored surface but would also include a legend as in Figure 6.4 that identified colors with numeric values; this is discussed below.

Figure 6.5: electrostatic potential surface model presented with “rainbow” color ramp to emphasize the extreme values (left) and with a uniform luminance distribution of colors (right)

However, perhaps a combination of color and shape such as these might not be the only way to approach the problem. If we think of the surface and the color ramp as different representations of the same space, perhaps it would be useful to think of them as separate displays that are linked in a spatial way. The example of Figure 6.6 below shows one way that might be done, with a lighted surface as one approach and a pseudocolor approach as another, displayed together. Here the rainbow color ramp is used.

Figure 6.6: a pseudocolor plane with the lighted surface

To light or not to light

If you are developing an image that represents actual objects and uses naturalistic colors, you will almost certainly want to use lighting and shading to make that image as realistic as you can.
The representation of actual objects is enhanced by making the objects seem realistic. If the things you display in your scene do not represent actual objects, however, and particularly if your colors are also synthetic and represent some property, then you need to think carefully about whether you should use lighting and shading for your image.

While lighting a scene will make the shapes stand out, there is a danger that issue the colors in the resulting shaded image will not be accurate in representing the values of the data for the scene. If your light definitions for lighting use the same color for ambient and diffuse lighting (especially if that color is white), and if your material color is set by a color ramp that maintains relatively constant luminance and simply changes chroma, there is probably less danger of getting colors that are misinterpreted; the shading provided by the lighting model will change brightness but not color. This is shown by the shaded relief map of South Africa in Figure 6.7, where the colors represent the height (altitude) of the land but the image is shaded as though there were a light low in the eastern sky.

Other instances where lighting might be useful could be where spheres of different sizes and colors were used to represent values, and you could use lighting to emphasize the shape of the spheres.

![Figure 6.7: a lighted, false-color map of South Africa](image)

**Higher dimensions**

Surfaces and colorings as described above work well when you are thinking of processes or functions that operate in 2D space. Here you can associate the information at each point with a third dimension and associate that with height or with a color at the point. However, when you get into processes in 3D space, when you think of processes that produce 2D information in 2D
space, or when you get into any other areas where you exceed the ability to illustrate information in 3D space, you must find other ways to describe your information.

Perhaps the simplest higher-dimensional situation is to consider a process or function that operates in 3D space and has a simple real value. This could be a process that produces a value at each point in space, such as temperature. There are two simple ways to look at such a situation. The first asks “for what points in space does this function have a constant value?” This leads to what are called isosurfaces in the space, and there are complex algorithms for finding isosurfaces or volume data or of functions of three variables. The left-hand part of Figure 6.8 shows a simple approach to the problem, where the space is divided into a number of small cubic cells and the function is evaluated at each vertex on each cell. If the cell has some vertices where the value of the function is larger than the constant value and some vertices where the function is smaller, the continuity of the function assures that the function assumes the constant value somewhere in that cell and a sphere is drawn in each such cell. The second way to look at the situation asks for the values of the function in some 2D subset of the 3D space, typically a plane. For this, we can pass a plane through the 3D space, measure the values of the function in that plane, and plot those values as colors on the plane displayed in space. The right-hand part of Figure 6.8 shows an example of such a plane-in-space display for a function $f(x, y, z) = x*y*z$ that is hyperbolic in all three of the $x$, $y$, and $z$ components in space. The pseudocolor coding is the uniform luminance ramp described above.

A different approach is to consider functions with a two-dimensional domain and with a two-dimensional range, and to try to find ways to display this information, which is essentially four-dimensional, to your audience. Two examples of this higher-dimension situation are vector-valued functions on a rectangular real space, or complex-valued functions of a single complex variable. Figure 6.9 presents these two examples: a system of two first-order differential equations of two variables (left) and a complex-valued function of a complex variable (right). The domain is the standard rectangular region of two-dimensional space, and we have taken the approach of encoding the range in two parts based on considering each value as a vector with a length and a direction. We encode the magnitude of the vector or complex number as a pseudocolor with the uniform color ramp as described above, and the direction of the vector or
complex number as a fixed-length vector in the appropriate direction. In the top row we use a relatively coarse resolution of the domain space, while in the bottom row we use a much finer resolution. Note that even as we increase the resolution of the mesh on which we evaluate the functions, we keep the resolution of the vector display about the same. The top row of Figure 6.9 uses a 20x20 grid for the color presentation, while the bottom row of the figure uses a 200x200 grid. This allows the color in the finer resolution grid to be smoother, which is appropriate for a continuous function. However, the vector display mesh is kept at 20x20 for both rows because this is about as fine a resolution as a user can differentiate on a standard screen.

![Figure 6.9: two visualizations: a function of a complex variable, left; a differential equation, right](image)

The displays in Figure 6.9 that combine color and shape are fundamentally 2D images, with the domain of the functions given by the display window and the range of the functions represented by the color of the domain and the direction of the vector. There have been similar visualizations where the range had dimension higher than two, and the technique for these is often to replace the vector by an object having more information [NCSA work reference]. An example is shown in Figure 6.10. Such objects, called glyphs, are extremely abstract constructions and need to be designed carefully, because they combine shape and color information in very complex ways. However, they can be effective in carrying a great deal of information, particularly when the entire process being visualized is dynamic and is presented as an animation with the glyphs changing with time.
Of course, there are other techniques for working with higher-dimensional concepts. One of these is to extend the concept of projection. We understand the projection from three-dimensional eye space to two-dimensional viewing space that we associate with standard 3D graphics, but it is possible to think about projections from spaces of four or more dimensions into three-dimensional space, where they can be manipulated in familiar ways. An example of this is the image of Figure 6.11, a image of a hypercube (four-dimensional cube). This particular image comes from an example where the four-dimensional cube is rotating in four-dimensional space and is then projected into three-space.

Dimensions

We think of computer graphics as being 2D or 3D, because we think of creating images in either a plane or in three-space. But in fact, we have more than three dimensions we can use for images, and understanding this can help us select from options that can make our images more effective.

Three of the dimensions are obvious: the three coordinates of ordinary space. However, we can create motion over time, so we have some control over a fourth dimension of time. And we have seen how we can use color to represent a numeric value, so that allows us to think of color as yet another dimension, our fifth. Let’s consider how we might use these dimensions to represent a
particular problem: representing the temperature of an object. Recall that early in this chapter we used an example of this type (see Figure 6.2) to compare geometric and color temperature encoding.

There are several contexts for this problem, so let’s start with the simplest: the temperature along a wire. In this example, there is one dimension (length) in the physical object, and one more in the temperature at each point. There is no “temperature” dimensions, but we can represent temperature by another dimension: number or color. If we use number, we could create a graph whose horizontal dimension is length and whose vertical dimension is height; this kind of graph is very familiar in textbooks in mathematics and the sciences. If we use color, we have a line that shows different colors at different points on the line, a very different representation but one that looks quite a bit like an actual heated wire. These two representations are shown in Figure 6.12 for a wire that is initially at a uniform heat but has a cold heat sink applied to both ends.

If we want to consider how the temperature on the wire changes over time, we add a third dimension to the question because time is another dimension. Again, we can ask how we would represent that third dimension, and again we have two different answers: as a third numerical dimension, or by animating the original image in time. But since the original image has two choices, we actually have four possible answers for this problem:

1. numerical curve for the temperature at any point in time, extended with numerical values for the third dimension to produce a three-dimensional surface; in this surface, slices parallel to the original curve are the temperatures at any time, while slices perpendicular to the original curve are temperatures at a particular point over time,

2. numerical curve for the temperature at any point in time, extended by animation so the curve changes over time,

3. colored line for the temperature at any point in time, extended with numerical values for the third dimension to produce a colored plane; in this plane, slices parallel to the original curve are the temperatures at any time, while slices perpendicular to the original curve are temperatures at a particular point over time,
4. colored line for the temperature at any point in time, extended by animation so the colors change over time.

Figure 6.13 shows two of these representations; unfortunately the media used for these notes does not permit showing animations, so the options that include animation are not shown.

![Image 1](image1.png) ![Image 2](image2.png)

**Figure 6.13: change in temperature over time from solutions 1 and 3 above**

This problem can be extended to temperatures in a 2D space, and the images in Figure 6.2 show two representations of the problem with the third dimension being temperature: in one the third dimension is height, and in the other the third dimension is color. Note that you can double up on these representations, and in fact the middle image in Figure 6.2 shows a dual encoding of the third dimension. To take this further to consider changes over time, we need a fourth dimension, and time is probably the best one; this would be presented by animating one of the representations from the figure over time.

If we go one step further, we would consider temperature in a 3D space. Here our choices of the fourth dimension are very limited, because time simply does not work differently for different points in the space. Thus we would probably use color as the representation of temperature, our fourth dimension, and we would have to use some kind of higher-dimensional approach to viewing the scene (e.g. slices or equitemperature surfaces). To see how the temperatures would change over time, we would almost certainly animate the display.

So we have many different ways to represent and show various dimensions, and you should look at your options in designing a scene.

**Image context**

As we discussed when we talked about the context for color, an image is never viewed separately from the rest of the viewer’s experience. It is always seen within his or her overall experience and expectations, and must be presented with an understanding of how that experience affects the user’s perception. In this section we will talk about a few additional things that help set an image’s context for the viewer.
Choosing an appropriate view

When you create a representation of information for an audience, you must focus their attention on the content that you want them to see. If you want them to see some detail in context, you might want to start with a broad image and then zoom into the image to see the detail. If you want them to see how a particular portion of the image works, you might want to have that part fixed in the audience’s view while the rest of your model can move around. If you want them to see the entire model from all possible viewpoints, you might want to move the eye around the model, either under user control or through an animated viewpoint. If you want the audience to follow a particular path or object that moves through the model, then you can create a moving viewpoint in the model. If you want them to see internal structure of your model, you can create clipping planes that move through the model and allow the audience to see internal details, or you can vary the way the colors blend to make the areas in front of your structure more and more transparent so the audience can see through them. But you should be very conscious of how your audience will see the images so you can be sure that they see what you need them to see.

Legends to help communicate your encodings

Always be careful to help your audience understand the information you are presenting with your images. Always provide appropriate legends and other textual material to help your audience understand the content of your displays. If you use pseudocolor, present scales that can help a viewer interpret the color information. This allows people to understand the relationships provided by your color information and to understand the context of your problem, and is an important part of the distinction between pretty pictures and genuine information. Creating images without scales or legends is one of the key ways to create misleading visualizations.

The particular example we present here is discussed at more length in the first science applications chapter. It models the spread of a contagious disease through a diffusion process, and our primary interest is the color ramp that is used to represent the numbers. This color ramp is, in fact, the uniform heat ramp introduced earlier in this chapter, with evenly-changing luminance that gets higher (so the colors get lighter) as the values gets higher.

Labels to help communicate your problem

An image alone only makes up part of the idea of using images to present information. Information needs to be put into context to help create real understanding, so we must give our audience a context to help them understand the concept being presented in the image and to see how to decode any use of color or other symbolism we use to represent content. Figure 6.14 shows an image with a label in the main viewport (a note that this image is about the spread of disease) and a legend in a separate viewport to the right of the main display (a note that says what the color means and how to interpret the color as a number). The label puts the image in a general context, and as the results of this simulation (a simulation of the spread of a disease in a geographic region with a barrier) are presented in the main viewport, the legend to the right of the screen helps the viewer understand the meaning of the rising and falling bars in the main figure as the figure is animated and the disease spreads from a single initial infection point.
Figure 6.14: an example of figure with a label and a legend to allow the figure to be interpreted

Another form of a label could be some text placed on a billboard that is located in the scene in such a way as to stay out of the important part of the scene. Creating billboards is described in the chapter on texture mapping, but they can give you the effect of floating text in a scene. If this is combined with a line or arrow from the billboard to the particular object that is being described, it can be a very effective way to highlight an object as animation or interaction moves the scene around.

**Motion**

The ability of a modern graphics API to show motion is a powerful communication tool. Whether the motion is created through animation or through interaction, it allows you to tell a story about your subject that can change through time or can be explored by each member of your audience individually. Presenting a time-changing display is particularly appropriate if you are considering a dynamic situation, which is a common theme in the sciences and in other areas of study. Some phenomena are simply invisible without motion; an example would be detecting a planet among the night stars, where only the different motion of the planet sets it apart to the unassisted eye.

When you create motion, you need to consider exactly what is going to move and the pace at which it is to move. Sometimes you will want to hold most of your scene fixed and have only part of it in motion, which could emphasize the way the moving part changes in relation to the rest of the scene; sometimes you will want to have everything moving so you can describe the overall motion present in the scene. If you are using animation to create this motion, it may be appropriate to use a time parameter in your modeling so that you may simply update the time and redisplay the model. In this case, of course, you will want to be sure that all the parts of your model use time the same way so that the motion is equally paced throughout your scene. If you are using interaction to create the motion, this could get more complicated because you may
want to allow your viewer to push some things harder than others, but you again need to create a consistent model of the behavior you will be showing.

The nature of today’s computing makes animation an interesting challenge. With faster systems and a growing hardware support for graphics, the time to render a scene keeps decreasing so you get faster and faster frame rates for pure animation. While we wouldn’t want to stop or slow this trend, it does mean that we run the risk of creating a real-time online animation that can come to run so fast that it’s difficult to understand. Most operating systems have a timed pause function that can be called from your program. You may want to design your animation to have a specific frame rate and use the pause function or a similar system utility to help you maintain the frame rate you want. Of course, this isn’t an issue with interactive motion, because the human is always the slowest part of the system and will serve to control his or her own frame rate.

Because sometimes you may want to produce particularly high-quality presentations for the general public or for an audience such as a funding agency, you may need to think about other aspects of a presentation. One of these is certainly sound; in the public presentation world, you will never see an animation without a sound track. Current graphics APIs do not often include sound capability, but we expect that this will come soon and you should think about the sound that would be used with your work. This could be a recorded voice-over, sound effects that emphasize the motion that is being displayed, or a music track—or all three. If you’re going to use video hardcopy for your presentation, you need to consider this now; if you’re only going to be doing online work, you need to think about this for the future.

**Leaving traces of motion**

When you convey information about a moving geometry to your audience, you are likely to use an animation. However, in order that your viewer can see not only the moving parts but also see how these parts have moved, you might want to leave something in the frame to show where the parts were in previous frames. One way to think of this is to say that you want to show your viewer a *trace* of the motion.

![Figure 6.15: two kinds of traces of moving objects](image)

Figure 6.15: two kinds of traces of moving objects
There are two standard ways you can show motion traces. The first is to show some sort of trail of previous positions of your objects. This can be handled rather easily by creating a set of lines or similar geometric objects that show previous positions for each object that is being traced. This trace should have limited length (unless you want to show a global history, which is really a different visualization) and can use techniques such as reduced alpha values to show the history of the object’s position. Figure 6.15 shows two examples of such traces; the left-hand image uses a sequence of cylinders connecting the previous positions with the cylinders colored by the object color with reducing alpha values, while the right-hand image shows a simple line trace of a single particle illustrating a random walk situation.

**Motion blurring**

In many contexts such as sports and high-speed action, we are accustomed to seeing moving objects blurred in a frame. One way to show motion, then, is to create images having a blurred image of those things that are moving, and a crisp image of those things that are in fixed positions. This can be done in several ways, but a standard approach is to use an *accumulation buffer* [ref], a technique that allows you to composite several images of a scene, each taken at a slightly different time. Those objects that are moving will be shown at different positions, so they will seem blurred; those objects that are fixed will be shown at the same position, so they will be seen as crisp. Many graphics APIs provide the accumulation buffer tool. An example of motion blur is seen in Figure 6.16 below.

![Example of motion blur](image)

Figure 6.16: a moving mechanism shown with one part fixed and the rest blurred from motion

**Interactions**

Many graphics applications provide the user the ability to interact with the images they create in order to understand the concepts displayed by the graphics. This interaction is a communication between the user and the program, so it comes under the general theme of this chapter. An interactive application needs to consider the way the user and application communicate about interactions so that the user can make the most productive use of this capability.
One common kind of application is viewing a scene or an object from a number of different viewpoints. When you do this, you are allowing the user to move around the scene and to zoom in or out of the scene. Another way to think of moving around the scene, however, is to rotate the scene in world space while you hold your eye fixed. In either case, you are determining a rotation around a fixed point, either for your eye point or for the scene, with your eye staying the same distance from the scene. This is, in effect, changing the latitude and longitude of the eye point, and it is straightforward to see this as moving in a vertical (latitude) or horizontal (longitude) direction on a sphere that encloses the scene. A natural control for this might be using the mouse in the window, so that when the mouse button is held down, the vertical motion of the mouse is translated into changes in latitude and the horizontal motion of the mouse is translated into changes in longitude. This mouse use is common in applications, and will likely be familiar to an experienced user. The other control for viewing a scene is zooming into or out of the scene, which is a one-dimensional motion. If you have additional buttons on the mouse, you might want to model this with a horizontal mouse motion with a different button pressed, although it might be confusing to have the same action with different buttons treated differently. Another approach would be to use a keyboard action, such as the f and b keys, move the user forward and back in the scene. This is simple to implement and works well for a single-language application, though it might not work as well if you were to use it for an application that would be used with languages other than English; however, if you were to allow other key pairs to work in addition to f and b, it might still work.

Another kind of application involves working with an individual part of an image through selection and manipulation of the part. Here the communication issues are showing that the part may be selected, creating the actual selection, and providing natural ways to manipulate the part. Showing that something may be selected is often done by changing the cursor shape when it is over an item that is selectable, and this can be implemented by using a passive mouse motion and setting a new cursor shape if your graphics API allows it. If the cursor shape cannot be changed, then perhaps the object’s display can be subtly different for selectable things than non-selectable things, or perhaps a label or legend can say what is selectable. The actual selection will probably be done with a mouse click, probably with the left button unless there is a good reason for another choice, and the manipulations will be chosen to fit the functions needed. We have described the use of a highlight color to show that something was selected, and above we talked about ways to provide 1D and 2D manipulations. Menus can be used to provide more complex kinds of manipulation, and we would encourage you to think about providing shortcuts in addition to menus to aid the expert user.

While we commonly think of interaction in terms of the common mouse and keyboard inputs, we need to recognize that many of the kinds of interaction we use involve six degrees of freedom (for example, X-Y-Z positioning and roll-pitch-yaw motion). There are devices available for high-end graphics systems that provide direct control of six degrees of freedom, and we expect even simple graphics APIs to provide device drivers for these soon. So do not limit your thinking about interaction to the mouse and keyboard, but be prepared to add more sophisticated devices to your toolkit as they become available.
Cultural context of the audience

As we described above, when members of your audience try to understand an image you have created to communicate an idea to them, they will do so within their individual culture. The culture of your audience is a complex issue, involving many different parts—professional cultures, social cultures, geographic cultures, and many others, as we noted above. If someone has to learn how to understand your image (for example, if your image has unique features that the audience has to figure out, or if your figure uses features that mean something different than they would in the audience’s culture), then your image will be less effective. You must learn how to express your ideas in the audience’s cultural context.

In order to communicate with your audience using images, you must understand the visual vocabularies of your audience’s culture. You must understand the nature of the symbols used in the culture, the color schemes familiar to the culture and their meanings, and the way graphic design is used in the culture. For example, while we may be familiar with the spare and open design of traditional Japanese life, the present culture of Japan may be more accurately represented by the crowded, banner-laden Japanese Web sites, which are similar to Japanese newspapers and magazines. If you are reaching a Japanese audience, you will have to choose which of these two approaches you would use in laying out your image.

Most of the work you will do will probably be oriented towards a professional group than a cultural group, however. Thus you will need to understand the use of images to represent concepts in physics, chemistry, biology, or engineering rather than the use of images in Japanese or another ethnic or religious culture. To do this, you will need to know how physicists, chemists, biologists, or engineers are accustomed to using images, and what they may assume your images would mean.

How can you find out what the visual manifestations of a particular culture are? You can research the content of the culture you’re reaching by reading the magazines, newspapers, or professional publications of the group. You can develop a bank of images that are commonly found in this audience culture by extracting images from these sources, and you can then test the role of the images in that culture with a panel of experts in the culture—persons deeply familiar with the culture, such as persons from an ethnic or religious culture or professionals from a professional culture. In a similar way, you can develop standard layouts, color schemes, and symbol sets for the culture that have been reviewed and shaped by expert panels. When you are finished, you will have a collection of images, layouts, symbols, and color sets—that is, a design vocabulary—for your target audience and can be comfortable that you have at least the basis for effective communication with that group.

There are references on the meaning of color in various contexts that could be useful to you, especially [THO]. The table below gives one look at this, and the reference contains other ideas about color meanings in context. You are encouraged to look at this further if you need a starting point for cultural issues in color.
Table 6.17: Associations of Color by Profession

<table>
<thead>
<tr>
<th>Color</th>
<th>Process Control Engineers</th>
<th>Financial Managers</th>
<th>Health Care Professionals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>Cold</td>
<td>Corporate</td>
<td>Death</td>
</tr>
<tr>
<td></td>
<td>Water</td>
<td>Reliable</td>
<td></td>
</tr>
<tr>
<td>Turquoise (Cyan)</td>
<td>Steam</td>
<td>Cool</td>
<td>Oxygen deficient</td>
</tr>
<tr>
<td>Green</td>
<td>Nominal</td>
<td>Profitable</td>
<td>Infected</td>
</tr>
<tr>
<td></td>
<td>Safe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Yellow</td>
<td>Caution</td>
<td>Important</td>
<td>Jaundiced</td>
</tr>
<tr>
<td>Red</td>
<td>Danger</td>
<td>Unprofitable</td>
<td>Healthy</td>
</tr>
<tr>
<td>Purple</td>
<td>Hot</td>
<td>Wealthy</td>
<td>Cause for concern</td>
</tr>
<tr>
<td></td>
<td>Radioactive</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Finally, you should also remember that color is not seen by itself; it is always seen in the context of other colors in the scene, and this context-sensitive color can lead to some surprises. An extensive description of color in context is found in [BRO]. As a very simple example of this context, in Figure 6.18, the image consists of a sequence of black squares, separated by gray lines with a bright white spot at each junction. So why do you only see the white spot you’re looking at, when the white spots away from the one you’re looking at appear gray or black?

Figure 6.18: a figure that shows artifacts that aren’t there
Accuracy

The concept of accuracy is pretty obvious when you are creating images to communicate ideas to your audience. You want to present them with images that represent the information that you have in a way that helps them understand that information clearly and accurately. You must respect your data and not try to go beyond what it can give you. Your goal is not to present pretty pictures; it is to present the data or theory you have as accurately as possible. Of course, making attractive images is useful when you can, even if your goal is accuracy—especially when you are creating images that will be used for public presentation.

Probably the most important point here is to work hard to understand what the data or theory you are working with really tells you, and what it doesn’t. For data presentation, the issues are fairly straightforward. If you have data that is organized on a regular mesh, then it is straightforward to use polygons based on the mesh to present that data. If your data is not organized regularly, then you may have to create some sort of representation of the data based on polygons for most graphics APIs to work with. If you do not know *a priori* that there is a smooth variation to the population from which you get your data, then you need to present the data as discrete values and not use smooth interpolations or smooth shading to make your data look smoother. If your data values are spatially distributed and there is no reason to assume anything about the values of the data between your samples, then the best approach might be to display the data in the sampled space as simple shapes at the sample locations that have different sizes and/or colors to show the values at those positions.

On the other hand, when you display theoretical concepts, you may find that you can calculate the exact theoretical values of your displayed information at some points (or some times), but the behavior between these points or these times is based on operations that cannot readily be solved exactly. This is common, for example, when the theory leads to systems of differential equations that cannot be solved in closed form (that is, whose solutions are not exact expressions). Here you must pay careful attention to the numerical solutions for these operations to be sure that they give you enough accuracy to be meaningful and, in particular, that they do not diverge increasingly from accurate results as your solution points or times move farther from the known values. It can be very useful to have a good knowledge of numerical techniques such as you would get from a study of numerical analysis in order to develop accurate presentations of the theory.

Output media

It is one thing to create images that are effective on the computer screen; it can be quite different to create images that are effective when presented to your audience in other media. As part of your developing your communication, you need to understand what media will be used to get your images to your audience, and work towards those media. Print, online, video, digital video, or physical objects created with rapid prototyping tools all have different properties that need to be considered in making your image presentation effective. We cover most of the issues in output media in the later chapter of these notes on hardcopy, and you need to have a good understanding of media issues, usually by doing a lot of work in different media and studying the
properties of the media and how they are used by your audience, before you can be comfortable that you can adjust your work to various media and retain its effectiveness.

Implementing some of these ideas in OpenGL

Many of the techniques we have discussed here are fundamental graphics concepts and have been discussed in other chapters in these notes. Shapes are implemented with the basic modeling tools in OpenGL, colors are implemented with either the absolute color or color through the standard lighting model, and motion is implemented with parametrized transformations and the idle event callback. There is little to be added here to these discussions. Instead, we will focus on a few of the techniques that are not found in these other chapters.

Using color ramps

Because color ramps are used to allow color to represent the numeric value of some parameter, you will use a color ramp by setting the color of some geometric entity (for example, a triangle or quad) based on a calculation of the parameter’s value. Using the notion of a ramp developed earlier in this chapter, one would have a code segment such as that below to calculate the RGB components of myColor[] from a function calcRamp(float) and then set either a material color or an absolute color to the values of myColor. This looks like the code below for absolute colors on a grid of triangles, where the average height of the three vertices for a triangle is used to determine the color.

```c
for ( i=0; i<XSIZE-1; i++ )
    for ( j=0; j<YSIZE-1; j++ ){
        // first triangle in the quad
        glBegin(GL_POLYGON);
            zavg = (height[i][j]+height[i+1][j]+height[i+1][j+1])/3.0;
            calcRamp((zavg-ZMIN)/ZRANGE);
            glColor3f(myColor[0],myColor[1],myColor[2]);
        glEnd();
        // now give coordinates of triangle
    }
    // second triangle in the quad
    glBegin(GL_POLYGON);
        zavg = (height[i][j]+height[i][j+1]+height[i+1][j+1])/3.0;
        calcRamp((zavg-ZMIN)/ZRANGE);
        glColor3f(myColor[0],myColor[1],myColor[2]);
    glEnd();
}
```

As we said, if you were using a lighted model with materials, you would use the results of the color ramp calculation to set the appropriate material properties for your object.

Legends and labels

Each graphics API will likely have its own ways of handling text, and in this short section we will describe how this can be done in OpenGL. We will also show how to handle the color
legend in a separate viewport, which is probably the simplest way to deal with the legend’s graphic. This code was used in creating the image in Figure 6.14 above.

The text in the legend is handled by creating a handy function, `doRasterString(...)` that displays bitmapped characters, implemented with the GLUT `glutBitmapCharacter()` function. Note that we choose a 24-point Times Roman bitmapped font, but there are probably other sizes and styles of fonts available to you through your own version of GLUT, so you should check your system for other options.

```c
void doRasterString( float x, float y, float z, char *s)
{
    char c;
    glRasterPos3f(x,y,z);
    for ( ; (c = *s) != '\0'; s++)
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, c);
}
```

The rest of the code used to produce this legend is straightforward and is given below. The color of the text is set as an absolute color and lighting, if any, is disabled in order to have complete control over the presentation of the legend. Note that the `sprintf` function in C needs a character array as its target instead of a character pointer. This code could be part of the display callback function where it would be re-drawn.

```c
// draw the legend in its own viewport
glViewport((int)(5.*(float)winwide/7.),0,(int)(2.*(float)winwide/7.),
            winheight);
clrClor(GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT);
...
// set viewing parameters for the viewport
glPushMatrix();
   glEnable (GL_SMOOTH);
   glColor3f(1.,1.,1.);
   doRasterString(0.1, 4.8, 0., "Number Infected");
   sprintf(s,"%5.0f",MAXINFECT/MULTIPLIER);
   doRasterString(0.,4.4,0.,s);
// color is with the heat ramp, with cutoffs at 0.3 and 0.89
   glBegin(GL_QUADS);
   glColor3f(0.,0.,0.);
   glVertex3f(0.7, 0.1, 0.);
   colorRamp(0.3, &r, &g, &b);
   glVertex3f(1.7, 1.36, 0.);
   glVertex3f(0.7, 1.36, 0.);
   colorRamp(0.89, &r, &g, &b);
   glVertex3f(1.7, 4.105, 0.);
   glVertex3f(0.7, 4.105, 0.);
   glVertex3f(0.7, 4.105, 0.);
   glVertex3f(1.7, 4.105, 0.);

   glVertex3f(1.7, 0.1, 0.);
   glVertex3f(0.7, 4.105, 0.);
   glVertex3f(0.7, 4.105, 0.);
   glVertex3f(1.7, 4.105, 0.);
```
Legends are implemented in much the same way. You can simply create text that is to go into the image however your design dictates, and write that text to the screen with the same kind of text tools.

Creating traces

One way to create the trace of an object, as shown in Figure 6.15 above, is to create a sequence of cylinders that connect a certain number of previous positions of an object and fade out as the points get older. The code below does that, based on a global variable tails that maintains the last several positions of the object in an array list. Elements of the list describe the previous positions and directions of the object, as well as the color and length of each segment. The variable valid is used as the trace is initialized and not all segments of the trace are yet created.

typedef struct {  // hold properties of individual tail cylinders for bodies
    point4 color;
    point3 position;
    point3 direction;
    float length;
    int valid;
} tailstruct;

void draw_tail()
{
    int j;
    float angle;
    point3 rot_vect;
    point3 origin={0.0,0.0,0.0};
    point3 y_point={0.0,1.0,0.0};

    for(j=0; j<T_LENGTH; j++)
    {
        if(tails.list[j].valid) {
            glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,tails.list[j].color);
            // calculate angle to rotate cylinder so it points in right direction
            angle = angle*180/PI+90;
            // calculate vector perpendicular to direction vector and y axis
            // for the line to rotate around.
            normal(tails.list[j].direction, origin, y_point, rot_vect);
            glPushMatrix();
            // move tail segment to right location, rotate, and set length.
        }
    }
}
glTranslatef(tails.list[j].position[0], tails.list[j].position[1], tails.list[j].position[2]);
glRotatef(angle, rot_vect[0], rot_vect[1], rot_vect[2]);
glScalef(1.0, tails.list[j].length, 1.0);
// draw tail segment as cylinder with 12 slices
cylinder(radius/30., 12);
glPopMatrix();
}
}

In the other example of Figure 6.15, showing a random walk of a certain number of steps of a single particle, a similar but simpler kind of process is used because we do not try to fade out the individual steps of the trace. Instead, we merely retain a certain number of previous positions and draw a polyline that connects them in a contrasting color.

**Using the accumulation buffer**

The accumulation buffer is one of the buffers available in OpenGL to use with your rendering. This buffer holds floating-point values for RGBA colors and corresponds pixel-for-pixel with the frame buffer. The accumulation buffer holds values in the range [-1.0, 1.0], and if any operation on the buffer results in a value outside this range, its results are undefined (that is, the result may differ from system to system and is not reliable) so you should be careful when you define your operations. It is intended to be used to accumulate the weighted results of a number of display operations and has many applications that are beyond the scope of this chapter; anyone interested in advanced applications should consult the manuals and the literature on advanced OpenGL techniques.

As is the case with other buffers, the accumulation buffer must be chosen when the OpenGL system is initialized, as in

```
glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE|GLUT_ACCUM|GLUT_DEPTH);
```

The accumulation buffer is used with the function `glAccum(mode, value)` that takes one of several possible symbolic constants for its mode, and with a floating-point number as its value. The available modes are

- **GL_ACCUM**: Gets RGBA values from the current read buffer (by default the FRONT buffer if you are using single buffering or the BACK buffer if double buffering, so you will probably not need to choose which buffer to use), converts them from integer to floating-point values, multiplies them by the value parameter, and adds the values to the content of the accumulation buffer. If the buffer has bit depth \( n \), then the integer conversion is accomplished by dividing each value from the read buffer by \( 2^n - 1 \).

- **GL_LOAD**: Operates similarly to **GL_ACCUM**, except that after the values are obtained from the read buffer, converted to floating point, and multiplied by value, they are written to the accumulation buffer, replacing any values already present.

- **GL_ADD**: Adds the value of value to each of the R, G, B, and A components of each pixel in the accumulation buffer and returns the result to its original location.

- **GL_MULT**:Multiplies each of the R, G, B, and A components of each pixel in the buffer by the value of value and returns the result to its original location.
GL_RETURN Returns the contents of the accumulation buffer to the read buffer after multiplying each of the RGBA components by value and scaling the result back to the appropriate integer value for the read buffer. If the buffer has bit depth n, then the scaling is accomplished by multiplying the result by $2^n - 1$ and clamped to the range $[0, 2^n - 1]$.

You will probably not need to use some of these operations to show the motion trace. If we want to accumulate the images of (say) 10 positions, we can draw the scene 10 times and accumulate the results of these multiple renderings with weights $2^{-i}$ for scene i, where scene 1 corresponds to the most recent position shown and scene 10 to the oldest position. This takes advantage of the fact that the sum

$$\sum_{i=1}^{10} 2^{-i}$$

is very close to 1.0, so we keep the maximum value of the accumulated results below 1.0 and create almost exactly the single-frame image if we have no motion at all. An example of code that accomplishes this is:

```c
// we assume that we have a time parameter t for the drawObjects(t) function and that we have defined an array times[10] that holds // the times for which the objects are to be drawn. This is an example // of what the manuals call time jittering; another example might be to // choose a set of random times, but this would not give us the time // trail we want for this example.
drawObjects(times[9]);
glAccum(GL_LOAD, 0.5)
for {i = 9; i > 0; i--} {
    glAccum(GL_MULT, 0.5);
    drawObjects(times[i-1]);
    glAccum(GL_ACCUM, 0.5);
}
glAccum(GL_RETURN, 1.0);
```

The array `times[]` is then updated in the `idle()` function so that each call to the `display()` function shows the object sequence after the next motion step.

A few things to note here are that we save a little time by loading the oldest image into the accumulation buffer instead of clearing the buffer before we draw it, we draw from the oldest to the newest image, we multiply the value of the accumulation buffer by 0.5 before we draw the next image, and we multiply the value of the new image by 0.5 as we accumulate it into the buffer. This accomplishes the successive reduction of the older images automatically.

There are other techniques one could find here, of course. One would be simply to take whatever image you had computed to date, bring it into the accumulation buffer with value 0.5, draw the new scene and accumulate it with weight 0.5, and return the scene with weight 1.0. This would be faster and would likely not show much difference from the approach above, but it does not show the possibilities of drawing a scene with various kinds of jittering, a useful advanced technique.
A word to the wise

Visual communication is much more complex than the material presented in this chapter, which has just tried to hit the high points of the subject to help you get started thinking about the issue. There are full courses in the subject in fields such as communication studies, and there are courses in the arts, cinematography, or videography that focus on just a small portion of this chapter. But if you learn to think about your communication from the beginning of your computer graphics studies, you will find it easier to learn to incorporate more sophisticated and advanced techniques as you develop your professional skills in the field.

Summary

This chapter has included a number of examples that illustrate many techniques in creating effective visual communications with computer graphics programming. These can be implemented with most graphics APIs, but in line with the focus of this book we have included information on doing this with OpenGL. The techniques we discuss are covered throughout this book including several later chapters, but we are putting this chapter immediately after the most basic graphics content so that you can understand the reasons for using these techniques.

Questions

1. Discuss some issues in choosing an appropriate viewpoint for presenting a scene so the viewer can get the best information from it. Consider issues such as whether key information in the scene is visible or whether some key issues may be difficult to distinguish from the viewpoint.

2. Several different kinds of color ramps were discussed in this chapter. Discuss some of the scientific issues that could make one or another color ramp preferable for a particular display. Include such things as whether changes in the ramp’s colors show changes in the value being shown, whether the ramp has discontinuities while the data is smooth, whether the ramp can readily be shown in a legend, and whether the ramp highlights particular values that are important for the viewer to understand.

3. In the previous question we asked you to discuss scientific issues in color ramps; in this question we ask you to discuss cultural issues in color ramps. Consider some places you see color ramps used, such as elevation maps in atlases, temperature maps in weather forecasts, or weather threat maps in news broadcasts. Discuss these color ramps in terms of the cultural issues in the things they display and see if you can identify other ramps that could represent the same information but have other cultural meanings.

4. We discussed using a surface with lights and lighting to show how a function varies in a region, and we also discussed using a surface whose color is determined by a color ramp to show the same information. Compare the information in the pseudocolor surface, where you can see the color at each point, with the information in the lighted surface, where lighting shows the variations in the surface. Is it worthwhile considering a presentation that includes a surface with both lighting and pseudocolor? How would you implement that idea?
5. How easy is it to determine a value from a color? If an image uses a color ramp and includes a legend that shows the relation between colors and values, can you match the color of a point in the image with a numeric value? Is there anything you could do to the color ramp or the legend in order to make it easier to match the color and value?

6. In the discussion of higher-dimension viewing, we showed a representation of a complex-valued function $f$ of a complex variable $z$, and showed the value of the function with the polar representation $(r, \theta)$ of the complex variable $f(z)$, with the direction $\theta$ as a vector and magnitude $r$ as a color. Discuss the effectiveness of that representation and consider whether the 20x20 grid of direction vectors that was presented is appropriate for the image. What would happen if you used a coarser grid (perhaps 10x10) for the directions? If you used a finer grid (perhaps 50x50) for the directions?

7. Designing interaction is a very complex subject and there is a whole discipline in computing, HCI (Human-Computer Interaction) that is devoted to it. However, we can think about interaction in terms of the relation between controls presented in an application and controls for the area of the application in terms that are familiar to a user from the problem outside computing. For each of the following interaction techniques, identify a problem area where you might find the technique in a context outside the computer.
   • A slider to control the value of a program parameter,
   • A dial to control the value of a program parameter,
   • A button to choose an option,
   • A set of radio buttons to choose only one of a set of options,
   • A mouse click to identify an item in an image,
   • A mouse drag to move an image or a selected item in an image,
   • A menu to make a selection from a list of options.
(See the chapter on events and interaction if you have not covered that already.)

**Exercises**

8. (Drawing) Visual communication is much broader than computer graphics, and skills in one part of this communication can lead to skills in others. As directed by your instructor, pick a view in your local environment and draw this view in a way that focuses on communicating the important parts of that view to others. When the drawings are done, share them with others and discuss what you were trying to communicate and how well the drawings do that.

9. We talked about color ramps whose values have luminance values that vary uniformly with the data being represented, and we gave an example of a ramp that ranged from black to red to yellow to white. Create other examples of uniform-luminance color ramps and compare the feeling of those color ramps as compared to the feeling of the very warm black-red-yellow-white ramp.
Experiments

10. Create an image that uses the black-red-yellow-white color ramp discussed above to implement its pseudocolor, and then change that color ramp to other ramps; try those discussed in the chapter as well as others you design yourself. Compare the feelings of the resulting images; you may find that changing the color ramp changes the overall sense of the image.

11. For the heat diffusion program illustrated in Figure 6.2, and using the code included in the introductory chapter, change the temperature-to-color function `setColor()` to use different color ramps, and examine how well the different ramps show the temperature changes across the bar.

12. Simulate color deficiencies in your viewer by modifying the colors you present in an image, and consider the effect on whether the image can be understood. For example, simulate red-green color blindness, one of the most common deficiencies, by changing all colors to have both red and green values equal to the average of the original red and green values. (You can do this in a color ramp by changing the ramp computation; you can do this in a lighted surface by changing the colors of the lights and materials.)

Projects