1

BASICS

It is rather amazing that a finite rectangular array of colored dots (called *pixels* as an abbreviation of picture elements) is sufficient to display the nearly limitless collection of images we recognize as realistically or symbolically representing portions of our world. The power of combinatorics helps us to explain the situation (millions of possible colors for each pixel in the large display array), but we can hardly conceive of all the images we have already seen let alone those that are yet to be seen. From this reductionist viewpoint, the whole idea of computer graphics is to set the right pixels to the right color. Easier said than done. Yes, a plain red square is easy, but one that looks like it is made of bricks is tougher, and one that includes a human face taxes the best of known algorithms.

Of course, the computer graphics enterprise includes any and all manipulations of images. We can start from scratch and produce a photo-realistic image of a new airliner or perhaps construct a landscape design complete with a variety of plants. Maybe the challenge is to translate CAT (computerized axial tomography) scan data into an image of the brain or correct the color balance in a photo being readied for publication. To bring some order to the very long list of possibilities, it is helpful to consider two main categories: either we are generating images, or we are processing existing images. Both require mathematical tools, but the first category encompasses the broad mathematical approaches necessary to understand three-dimensional descriptions of objects and their interactions with light. The second category starts with an image and draws on the mathematics of transformations and filters necessary to convert it into a more useful visual representation. In this survey of mathematical tools that are

Mathematical Structures for Computer Graphics, First Edition. Steven J. Janke.

useful in computer graphics, we will focus on the first category where we can start with the basics of mathematical descriptions and work through the generation and manipulation of objects in space.

1.1 GRAPHICS PIPELINE

As we examine the steps necessary to produce a new image on the computer screen, we are tracing what is often called the *graphics pipeline*. The pipeline analogy is intended to highlight the stages we go through both in designing images and in processing them on the computer to produce the final properly colored array of pixels on the display screen. As one frame is being completed, the next is making its way down the pipeline. Most modern hardware includes the main microprocessor (central processing unit, CPU), the graphics microprocessor (graphics processing unit, GPU), and various associated memory banks. The CPU and GPU work in parallel, as the CPU supplies descriptions of objects to the GPU which in turn processes the descriptions to determine which pixels on the screen need to be turned on. The exact order of all the required steps depends on the hardware and on the graphics software we use. However, we can make a more general description of the pipeline to enumerate the stages of image generation and set the context for understanding the associated mathematics. Our pipeline then looks like this:

1. *Modeling*. We need a mathematical description of objects, background, and light sources as well as a description of their placement in a scene. For more primitive objects such as buildings which are more or less constructed out of simple plane surfaces, the description includes a list of vertices and a list showing which vertices determine individual faces. For curved surfaces, we may attempt an accurate description (e.g., a sphere) or rely on an approximation with small flat triangles. These descriptions are, of course, just the beginning, as we need also to know the details of how the objects are placed in a scene and how light will interact with them. Mathematically, a geometric description including vertices and faces (surfaces) forms the kernel of our model, but certainly if the object is a tree or if there is fog affecting the lighting, the description may well require a deeper extension of the standard high school geometry. This modeling stage can be done with design software, allowing artists to manipulate the scene to reach the desired effect.
2. *Transformation*. Building a scene requires positioning objects relative to each other and includes rotation, scaling, and translation. Transformations reposition an object and convert its coordinate descriptions appropriately. Then, to view the scene, imagine a camera placed somewhere in space looking in a particular direction. (Alternatively, imagine your eye positioned in space looking at the scene.) Another transformation adjusts the mathematical descriptions so that they are relative to the camera position.
3. *Visibility*. Depending on where the camera is, we may not see the entire scene. Rather, some parts are outside the field of view and consequently can be ignored

when generating the image. Even those objects within the field of view usually have some surfaces that are facing away from the camera and need not be considered. Some objects in the scene may be only partially visible since they fall both inside and outside the field of view. These objects are clipped (often after projection) so that only the relevant pieces continue down the graphics pipeline. Dealing with only the visible portions of a scene improves the efficiency of image generation because the calculations necessary to determine pixel color are computationally expensive.

4. *Projection*. Once we position the camera (or our eye), the actual image of the scene is produced on a two-dimensional surface in the camera (or on our retina). The three-dimensional scene is projected onto a two-dimensional screen. It is somewhat easier to imagine that there is a window placed just in front of our eye or camera and the scene itself is painted on this window. Positions of points in the scene, including how far away from the window they are, determine where on the window they are projected. Done correctly, the projection preserves the perspective in the scene and adds realism. If the dimensions of the window do not match the display screen, yet another transformation is necessary to convert window coordinates to display coordinates.
5. *Rasterize*. Mathematical descriptions of objects are usually continuous, allowing line segments, for example, to have an infinite number of points. At some stage of the graphics pipeline, the continuous line must be approximated by a finite set of screen pixels. This process requires some care to avoid distorting the line and introducing unintended artifacts, but then we have a finite set of pixels rather than an infinite set of points. The task of determining the colors of the pixels is now manageable and, if done appropriately, can maintain the illusion of a continuous image.
6. *Shading*. Light determines the exact shade of color that an object reflects, and that light depends on the position of light sources, their intensity, and their color. Some objects may be casting shadows and others may actually be reflecting light onto the rest of the scene. The geometry of light rays is essential for making these shading calculations. Positioning light sources and determining the material properties of objects occurs early in the pipeline, but it is late in the process when color calculations are actually completed for individual pixels.
7. *Texturing*. Describing the surface of an object as mathematically planar indicates that it is flat and smooth. Yet surfaces can be rough or covered with patterns of color. For example, in a computer game, the walls of a room may well be made of stone, so it becomes important to determine how light reflects off stone. In the texturing stage, this type of surface detail is added somewhat artificially by either copying the detail from existing images (e.g., taking a picture of real stone) or generating it mathematically with functional descriptions (e.g., some function describing how bumpy stone really is). To determine final colors for pixels, textures need to be generated and mapped to individual pixels.

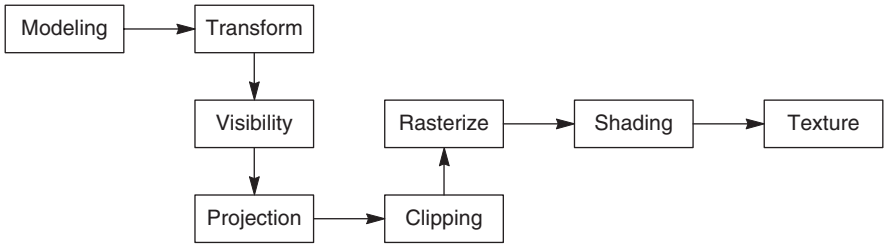

Figure 1.1 Graphics pipeline

This is the general notion of a graphics pipeline. The first stage, modeling, is often done interactively and builds the contents of the image, while all the rest of the stages taken together *render* the image on the screen. Depending on the hardware and software, the order of the rendering stages may be slightly permuted, but the scope of the process indicates the range of mathematical tools we need to explore (Figure 1.1).

1.2 MATHEMATICAL DESCRIPTIONS

To generate an image, we need to mathematically describe a scene. For a simple object such as a cube, we can list the position of its vertices and then list which vertices anchor each face. This is easy enough, but the positions of the vertices depend on the coordinate system we are using and it is not obvious which one we should use. Putting the origin of our coordinate system at the center of the cube makes describing the vertex positions easier, but there may be other objects in the scene which are good candidates for the origin. The answer may well be to use many coordinate systems and develop means of combining them into an all-inclusive scene.

If next to the cube there is a more organic object such as a flower, then we have an added difficulty because a flower is probably not well described by giving vertices and faces. There could be such a description, but there is also a special system of symbols (L-system) that was designed to capture the way plants grow, making a description both easier and more succinct.

The cube may be made of wood, making the faces more bumpy than smooth and making light reflect in a way that shows the grain. The scene description is now moving further and further from a simple list of vertices, and we will need additional means of describing the detail.

The larger goals are to make the mathematical description as simple as possible, as easy as possible to alter during the design process, and as independent of any fixed coordinate system as possible. Then the resulting computer code will be general and flexible.

To draw an object on the computer screen, we need to identify which pixels to light up (and what color to make them). Since most computer monitors are rectangular, locating a pixel usually means specifying its horizontal and vertical position in the rectangular array of pixels which make up the entire screen. This seems simple

enough, but our common descriptions of objects are not usually in terms of the horizontal and vertical distances to each point. A cartoon character's head might be described as "oval, with a button nose, and beady eyes." And a tree may be "conical with short drooping branches covered with folded, heart-shaped leaves." To draw these objects on the screen, there is no escape from determining the horizontal and vertical positions of appropriate pixels, but the challenge for the graphics programmer is to find ways of describing the objects that are between the intuitive common way and the hard-core quantitative way that lists the horizontal and vertical positions of all the points. Unfortunately, qualitative descriptions of objects are not easily incorporated into computer programs, so it makes sense, at least at first, to concentrate on more quantitative descriptions.

1.3 POSITION

In the geometry of Euclid, there are no coordinates. Instead, geometric objects are compared to each other in order to understand their features; lines are compared to other lines, and triangles to triangles. This approach is not sufficient for computer graphics because we eventually need the absolute position of an object in order to determine which pixels on the screen to turn on. In the seventeenth century, Descartes, the philosopher and mathematician, made attempts to connect algebra and geometry, and although he did not develop coordinate systems as we know them now, we still refer to the rectangular coordinate system as the *Cartesian* coordinate system. This is the default coordinate system for computer graphics and the one we are all familiar with from high school (Figure 1.2).

In two dimensions, we have two perpendicular axes, the horizontal one labeled x and the vertical one labeled y. They cross at a unique origin labeled 0. Each is a number line increasing either to the right or up. Any point in the plane has a coordinate representation which is a pair of numbers (x, y) indicating how far to go horizontally (negative distance indicates left of the origin) and then how far to go vertically (negative here means down from the origin). Note that this is a unique representation; any pair of numbers determines exactly one point in the plane and any point determines exactly one pair of numbers.

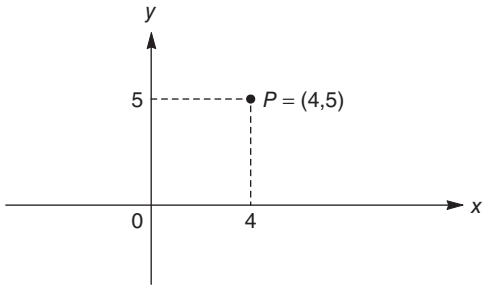

Figure 1.2 2D Cartesian coordinate system

In computer graphics, the fundamental task is to locate points in a scene, and to make that process easier and perhaps more intuitive, we define a new mathematical object called a *vector*.

Definition 1.1 *A two-dimensional vector is an object representing a displacement in the plane. It has a length and a direction.*

A vector is intended to describe how to get from one point to another. If our vector has a length of five units and is pointing to the right, then it represents moving from an arbitrary point to a point five units to the right. It is important to note that the vector is not positioned at any particular place in the plane. It represents displacement, not position. Once we apply the displacement to a point, we reach another point that is positioned in the plane. Visually, vectors are represented as arrows; they have length and direction. As we will soon see, a convenient way of describing a two-dimensional vector mathematically is to give two numbers indicating its displacement in the horizontal and vertical directions. Often the representative arrow is drawn with its tail at the origin, say, and its head (the end with the arrow) positioned to show the displacement. This can be a little confusing because the vector is really not positioned at any particular point in the plane; setting the tail at the origin is just a default approach to representing the vector visually (Figure 1.3).

We now can give a slightly different perspective on the standard Cartesian coordinate system. To describe a two-dimensional (2D) coordinate system, we specify a unique origin and two vectors. For the standard system, these vectors both have the same unit length and are perpendicular to each other. The vector with direction along the positive x-axis is usually referred to as $\vec{i}$, and the one in the direction of the positive y-axis is denoted as $\vec{j}$. Describing any point in the plane is now a matter of indicating how many unit steps in the direction of $\vec{i}$ and how many in the direction of $\vec{j}$ we need to take to reach the point. For example, the point $(4, 1.5)$ is the point we reach when starting at the origin, then taking 4 unit steps in the x-direction, and finally 1.5 unit steps in the y-direction. As you may have guessed, since $\vec{i}$ and $\vec{j}$ have unit length, the algebra of vectors allows us to represent the point as $4\vec{i} + 1.5\vec{j}$. This will prove to be useful in making geometric calculations (Figure 1.4).

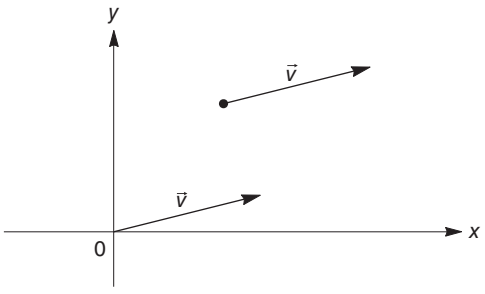

Figure 1.3 Vectors indicating displacement

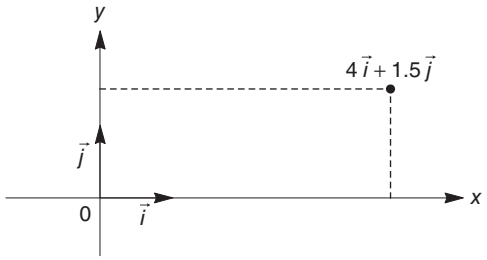

Figure 1.4 Vectors $\vec{i}$ and $\vec{j}$

Without much effort, we can move to three-dimensional space by adding a third axis labeled z perpendicular to the x- and y-axes. With our vector perspective, we have added a new vector, often called $\vec{k}$. Now each point in space is represented by a unique triple of numbers, or in terms of vectors as a combination of the three vectors $\vec{i}$, $\vec{j}$, and $\vec{k}$.

The nature of the Cartesian coordinate system depends on the direction of the unit vectors. The standard two-dimensional system has vector $\vec{i}$ pointing to the right along the positive x-axis and the perpendicular vector $\vec{j}$ pointing up. However, we might also let $\vec{j}$ point down. This is actually the default coordinate system for the computer screen when using some standard programming languages. An easy transformation can get us back to the standard system with $\vec{j}$ pointing up, and usually this is desirable. As we will see later, we could pick the two vectors for a system so that they are not perpendicular. These systems may prove useful in describing some objects, so we will have to develop transformations that allow us to easily move between all these various possibilities.

In three dimensions, there is one, often troubling, complication. There are two geometrically different ways to add a third vector and this time the mathematical consequences are not trivial. Basically, the third vector $\vec{k}$ could point in the direction designated in Figure 1.5 or in the opposite direction. To standardize, we designate a *right-handed* system to be one where if we position our right hand with the fingers pointing in the direction of $\vec{i}$ and adjust so that when we curl our fingers they point

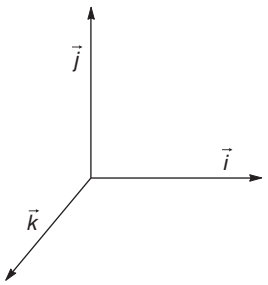

Figure 1.5 Right-handed coordinate system

in the direction of $\vec{j}$, then our thumb points in the direction of $\vec{k}$. Figure 1.5 shows a right-handed coordinate system. It is important to distinguish right-handed from left-handed systems in order to keep track of whether vectors point out of or into an object.

1.4 DISTANCE

Now that we have set up a coordinate system, we turn to the fundamental problem of determining the distance between two points. This is a job for the venerable Pythagorean theorem, named after an itinerant teacher of ancient Greece who led his devoted followers through a wide range of ideas drawn from topics as diverse as number theory and vegetarian diets. He is credited for the famous theorem about right triangles, but the result was undoubtedly known much earlier by at least Babylonian scholars if not others [1].

Theorem 1.1 (Pythagorean Theorem). *In a right triangle, the sum of squares of the two legs equals the square of the hypotenuse.*

Proof Sketch. First remember that a right triangle is one with a 90° angle. Figure 1.6 shows four identical right triangles with legs a and b. They are arranged in a large square on the left and then rearranged in the same large square on the right. On the left, the area not taken up by triangles is equal to c^2, the area of the labeled square in the middle. On the right, the area outside the triangles is in two pieces equaling $a^2 + b^2$. Hence the Pythagorean theorem: $a^2 + b^2 = c^2$. □

This result allows us to calculate the distance between any two points on the screen or in an arbitrary plane. Simply, the distance is the length of the hypotenuse of a right triangle. The two points are the opposite corners of a rectangle with sides parallel to the vectors $\vec{i}$ and $\vec{j}$ which determine the coordinate system. The distance between the corners of the rectangle is the length of the hypotenuse of a right triangle. The legs of the right triangle are easy to find by taking differences of the Cartesian coordinates for the two points. If one point has coordinates (x_1, y_1) and the other (x_2, y_2), then the

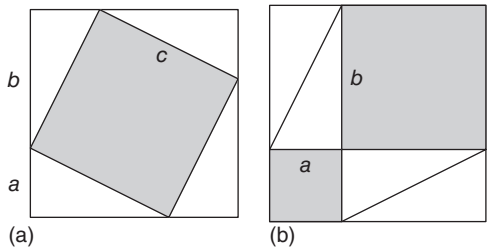

Figure 1.6 (a,b) Visual proof of the Pythagorean theorem

Pythagorean theorem gives

$$Distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

In three dimensions, the distance between points is not much harder to find once we visualize two right triangles. Suppose the two points are labeled P_1 and P_2. This time, they can be thought of as the opposite corners of a rectangular box where the faces of the box are parallel to the three coordinate planes (Figure 1.7). The distance between the points is the length of the hypotenuse of the right triangle $\Delta P_1 Q P_2$. The leg QP_2 is just one edge of the box, called a in the figure. The leg P_1Q is a diagonal of one face of the box and hence the hypotenuse of triangle $\Delta P_1 RQ$. By the Pythagorean theorem,

$$(P_1Q)^2 = b^2 + c^2.$$

Since $(P_1P_2)^2 = (P_1Q)^2 + (QP_2)^2$, we finally have

$$(P_1P_2)^2 = a^2 + b^2 + c^2$$

where a, b, and c are all edges of the rectangular box.

The Pythagorean theorem is essential to computer graphics. Dropping perpendiculars and forming right triangles is one of the most useful tools in the mathematics toolbox. Triangles are everywhere in graphics, and, in fact, they are central to all of geometry. Projections and visibility questions invariably involve drawing a triangle usually including the eye position as a vertex. Complicated objects are usually built from triangles because triangular faces are guaranteed to be planar; they can be drawn in a plane. (This is unlike quadrilateral faces which might be twisted so that the four vertices do not all lie in a plane.) So calculations with triangles are central to computer graphics and we rely both on the Pythagorean theorem and on a generalization that covers triangles of arbitrary angles. To reach this generalization, we use the cosine function (reviewed in Appendix A).

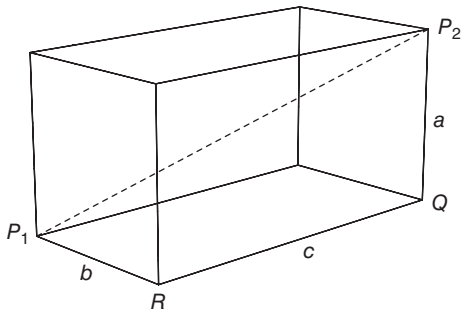

Figure 1.7 Distance in three dimensions

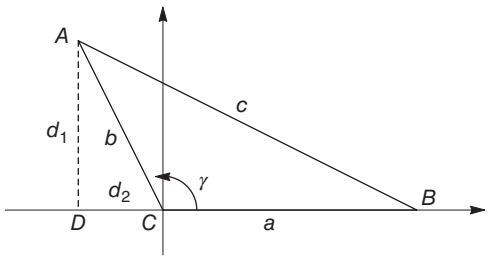

Figure 1.8 Law of cosines

Theorem 1.2 (Law of Cosines). *In a triangle with sides $a, b,$ and c, let the angle γ be opposite the side c. Then we have $c^2 = a^2 + b^2 - 2ab\cos\gamma$.*

Proof Sketch. Note that γ is an angle inside the triangle, so we know it is less than 180°. When $\gamma = 90°$, then $\cos\gamma = 0$ and we have a right triangle so the Pythagorean theorem applies and the law of cosines reduces to it. In general, we try making right triangles out of the original triangle to see why the law holds. There are actually two cases: $\gamma > 90°$ and $\gamma < 90°$. Figure 1.8 shows the first case.

In the figure, the side AD is perpendicular to the baseline CB. Then we have three triangles: the given triangle ΔABC and two right triangles ΔADC and ΔADB. With the lower case letters indicating lengths, apply the Pythagorean theorem to the triangle ΔADC to get

$$b^2 = d_1^2 + d_2^2.$$

Applying the Pythagorean theorem to the second right triangle, ΔADB gives

$$c^2 = d_1^2 + (d_2 + a)^2 = d_1^2 + d_2^2 + 2d_2a + a^2 = a^2 + b^2 + 2d_2a$$

Notice that $\angle ACD$ is the supplement of γ (i.e., they add to 180°). This means $\cos(\angle ACD) = -\cos\gamma$, and since $\cos(\angle ACD) = d_2/b$, we now have the result $c^2 = a^2 + b^2 - 2ab\cos\gamma$.

For the second case where $\gamma < 90°$, we proceed just as above by drawing a new side and building new right triangles. The algebra is just a little different (Section 1.5). □

Example 1.1 (Triangles in 3D). Suppose we have three vertices in three dimensions complete with coordinates $A = (3, 5, 4), B = (6, 2, -3), C = (-4, 6, 3)$. Although we are in three dimensions, the triangle does lie in a single plane, so using the tools we developed we can completely describe it. Applying the Pythagorean theorem, first we get the lengths of all the sides:

$$|AB| = \sqrt{(3-6)^2 + (5-2)^2 + (4-(-3))^2} = \sqrt{67} \approx 8.185$$

$$|AC| = \sqrt{(3-(-4))^2 + (5-6)^2 + (4-3)^2} = \sqrt{51} \approx 7.141$$

$$|BC| = \sqrt{(6-(-4))^2 + (2-6)^2 + (-3-3)^2} = \sqrt{197} \approx 14.036$$

Comparing the squares of the lengths, we can determine whether each angle is larger or smaller than a right angle. For example, since $67 + 51 < 197$, we know that $\angle ABC$ must be larger than a right angle. That makes the other two smaller than a right angle because the sum of all angles must be 180° (or π radians).

Now, an application of the law of cosines gives us the actual angle:

$$197 = 67 + 51 - 2\sqrt{67}\sqrt{51}\cos(\angle BAC) \;\Rightarrow\; \cos(\angle BAC) \approx -0.676.$$

This indicates that $\angle BAC \approx 132.53^\circ$. The same procedure gives the other two angles: $\angle ACB \approx 25.46^\circ$ and $\angle ABC \approx 22.03^\circ$. □

Once we can completely describe the triangle, we should be able to determine whether a light ray hits it. This is a common problem that we will solve later by first finding where the light ray hits the plane of the triangle and then deciding whether the intersection point is inside or outside the triangle. In order to solve this problem, it helps to translate the tools we are using to the language of vectors, and this we do in the next chapter.

1.5 COMPLEMENTS AND DETAILS

1.5.1 Pythagorean Theorem Continued

No one knows how Pythagoras proved his theorem because even the basic facts of his life (about 500 BCE) are a bit sketchy. Since then, there have been many proofs devised including a somewhat complicated one given by Euclid in Proposition 47 of Book I of his Elements (about 300 BCE). Just a little later, the illustrated square on the left in Figure 1.6 appeared in a Chinese manuscript, and in 1876 a New England education journal published a proof apparently constructed by James A. Garfield who later became President of the United States. Most of the proofs involve constructing geometric figures in one way or another. (A more complete history of the theorem is given in [1].)

For a slightly more algebraic approach to the proof in Figure 1.6, notice that the area of the large square in the left half of the figure is $(a+b)^2$. Yet, this must be equal to the area of four triangles plus the area of the square in the middle (c^2).

$$\begin{aligned} (a+b)^2 &= 4\left(\frac{1}{2}ab\right) + c^2, \\ a^2 + 2ab + b^2 &= 2ab + c^2 \\ a^2 + b^2 &= c^2 \end{aligned}$$

One important number-theoretic consequence of the theorem emerges when we draw the right triangle with both legs equal to 1. Then the hypotenuse is $\sqrt{2}$ and this number was important to the Greeks because it was *incommensurable*. To us now, this means the number is irrational; it cannot be represented as the quotient of two integers. The discovery of irrational numbers was both progress and an annoyance to the Greeks.

Other right triangles are equally surprising. If the legs are 3 and 4, then the hypotenuse is 5. This set of three integers is called a *Pythagorean triple* and is used frequently by carpenters to quickly construct a right angle. There are infinitely many of these Pythagorean triples and a detailed theory surrounding them (see Exercises for further examples).

1.5.2 Law of Cosines Continued

To derive the law of cosines, we noted that, if we do not have a right triangle, there are two cases: one where $\gamma > 90°$, and one where $\gamma < 90°$. The first case was covered in Figure 1.8, so now we consider the second case.

When $\gamma < 90°$, our triangle looks like the one in Figure 1.9. We have constructed two right triangles by adding the perpendicular AD. The Pythagorean theorem says

$$c^2 = h^2 + a_2^2$$
$$b^2 = h^2 + a_1^2$$

Note that $a = a_1 + a_2$ and $a_1 = b\cos\gamma$. By adding a^2 to both sides of $b^2 = h^2 + a_1^2$, we have

$$\begin{aligned} b^2 + a^2 &= h^2 + a_1^2 + a^2 = h^2 + a_1^2 + a_1^2 + 2a_1a_2 + a_2^2 \\ &= c^2 + 2a_1(a_1 + a_2) \\ &= c^2 + 2(b\cos\gamma)a \end{aligned}$$

Rearranging just a little gives the law of cosines.

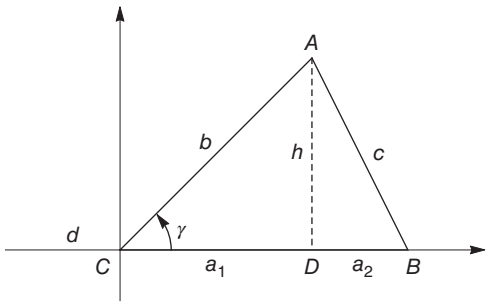

Figure 1.9 Law of cosines: $\gamma < 90°$

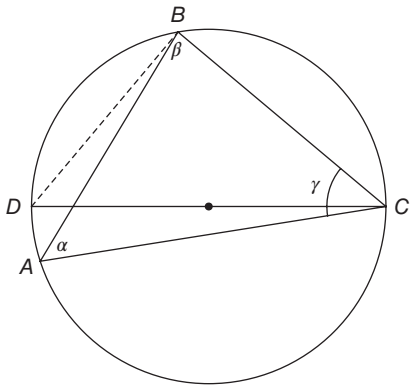

Figure 1.10 Law of sines

1.5.3 Law of Sines

Yes, there is a law of sines as well as a law of cosines.

Theorem 1.3 (Law of Sines). *In a triangle ΔABC, where the angles at the vertices are, respectively, $\alpha, \beta,$ and γ, and the sides opposite the vertices are $a, b,$ and c, respectively, we have*

$$\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c} = \frac{1}{d},$$

where d is the diameter of the circumcircle for the triangle.

Proof Sketch. Any triangle can be inscribed in a circle so that the three vertices are on the circle (Appendix A). Figure 1.10 shows one such arbitrary triangle, ΔABC. Draw diameter CD and dotted line DB. Since ΔCDB is inscribed in a semicircle, it is a right triangle. The sine of $\angle CDB$ is $\frac{a}{CD}$. But angle α equals angle $\angle CDB$ because they cut the same arc from the circle. Hence $\sin A = \frac{a}{CD}$, or $\frac{\sin A}{a} = \frac{1}{CD}$. The same argument for each angle shows the ratio of the sine to the side opposite is always the reciprocal of the diameter.

The diameter in Figure 1.10 cuts through the triangle. There is a second case where the diameter is outside the triangle. The argument changes only slightly (see Exercises). □

The common ratio of the sine to the side opposite is equal to the reciprocal of the diameter of the circumcircle for the triangle.

1.5.4 Numerical Calculations

One slight hitch for the graphics programmer is the fact that calculating the square root and trigonometric functions (e.g., sine, cosine, and tangent) takes time. Modern

processors incorporate floating point operations much more efficiently that they once did, but still, floating point arithmetic is slower than integer arithmetic. Making graphics programs run quickly requires attention to the length of calculations.

Consider the square root first. If the task is to simply compare distances, using the square of the distances works equally well. However, if the square root is actually needed, then often an approximation can work. To illustrate, suppose we need to calculate $\sqrt{x}$. Start with a guess, say g_0. Then x/g_0 should be g_0 if it is the square root. It probably is not, so take a next guess $g_1 = (g_0 + \frac{x}{g_0})/2$; this is the mean of the first guess and the quotient. Similarly, we can define successive guesses. For example, to find $\sqrt{120}$, let 10 be the first guess. Then $g_1 = 11$ and $g_2 = 10.95$. This last guess is accurate to two decimal places. (This algorithm for square root is actually Newton's method applied to the square root function.) Of course, this approximation is useful only if the time required to execute it is reasonably short.

Calculating the sine and cosine causes similar timing issues. One solution is to precalculate a table of common values and simply look up the answer when needed. For example, we could calculate the sine and cosine for all angles of radian measure $2\pi/n$ where $1 \leq n \leq 64$. If we need more accuracy, we can recall the Taylor series expansion (from calculus) of sine and cosine for small angles. The first few terms of these expansions (for radian measure) give

$$\sin(\theta) \approx \theta - \frac{\theta^3}{6} + \frac{\theta^5}{120}$$

$$\cos(\theta) \approx 1 - \frac{\theta^2}{2} + \frac{\theta^4}{24}$$

The Taylor series approximations are more accurate for small angles, so one scheme is to precalculate a table as before, let θ be the difference between the desired angle and the closest angle in the table (say α), approximate the sine or cosine of θ, and then use the addition formulas for sine and cosine to get $\sin(\alpha + \theta)$ or $\cos(\alpha + \theta)$.

1.6 EXERCISES

1. The standard Cartesian coordinate system has the vectors $\vec{i}, \vec{j}$, and $\vec{k}$ positioned to form a right-handed system. We can replace any or all of these vectors with one pointing in the opposite direction. This gives us a total of eight different coordinate systems. Determine which of these are right-handed systems.

2. Consider an isosceles right triangle. (This is one where both legs are equal.) Construct a square on each of the three sides. The Pythagorean theorem says that the sum of the areas of the two smaller squares equals the area of the larger square. By dividing each square into triangles equal to the initial triangle, establish the theorem in this special case.

3. For another proof of the Pythagorean theorem, consider Figure 1.11. Triangle ΔABC is a right triangle with the right angle at C. Each of the smaller triangles

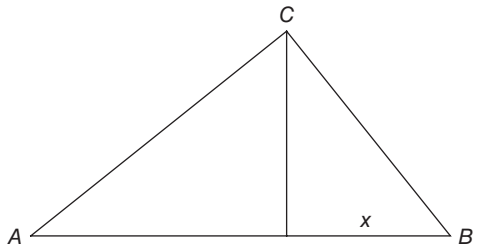

Figure 1.11 Alternate proof of Pythagorean theorem

is a right triangle and each is similar to ΔABC. This means that the ratio of sides in one triangle equals the ratio of sides in another. Find two of these equations which when added together give the Pythagorean theorem.

4. The vertices $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$ form a triangle. Find the perpendicular distance from the origin $(0, 0, 0)$ to this triangle using right triangles.

5. Find the three angles of the triangle with vertices $A = (-1, 1, 2)$, $B = (5, 3, 1)$, $C = (2, 6, -4)$.

6. The four vertices $(2 + \sqrt{2}, 2 + \sqrt{2}, 2)$, $(1 - 3\sqrt{2}, 1 + 3\sqrt{2}, 1)$, $(-6, -6, -2)$, and $(2, 1, -6)$ form four triangles in space. Determine which of the four, if any, are right triangles.

7. Given two points in the plane, where are all the points that are at the same distance from both these selected points? Given three points in the plane that are not on a line, where are all the points equidistant from all three?

8. Describe all points that are at a fixed distance from a solid square in the plane. (The distance from a point to the square is the minimum distance between the point and any point on the square.)

9. For some right triangles, the two legs and the hypotenuse are all integers. For example, sides 3, 4, 5 form a right triangle. We call the triple $(3, 4, 5)$ a Pythagorean triple. Of course, any multiple of these three numbers [such as $(6, 8, 10)$] also forms a Pythagorean triple. Find two Pythagorean triples that are not multiples of $(3, 4, 5)$ or of each other.

10. Pick two positive integers s and t such that one is odd, one is even, and $s > t$. Show that $x = 2st$, $y = s^2 - t^2$, and $z = s^2 + t^2$ form a Pythagorean triple as defined in the previous exercise. If, in addition, s and t do not have a common divisor greater than 1, the triple is said to be primitive and all primitive triples can be found in this way.

11. The vectors $\vec{i}$ and $\vec{j}$ define the two-dimensional coordinate system. Suppose we replace $\vec{j} = (0, 1)$ with the vector $w = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$. In this new coordinate system, what are the coordinates of the point with old coordinates $(2, 3)$?

12. If we use the vectors $2\vec{i}$ and $3\vec{j}$ to define a Cartesian coordinate system and we move the origin to the point $(-1, 6)$ in the original coordinate system, what are coordinates of the point with old coordinates $(4, 7)$? Give equations showing how to convert from old coordinates to new coordinates.

13. Referring to Figure 1.10, the diameter for the circle passes through the triangle. It could have passed outside the triangle. Complete the proof of the law of sines in this second case.

14. By drawing a perpendicular from the vertex A to the opposite side in a triangle, form two right triangles and show that $a = b\cos\gamma + c\cos\beta$. Then use the law of sines to show $\sin(\beta + \gamma) = \sin\beta\cos\gamma + \sin\gamma\sin\beta$.

1.6.1 Programming Exercises

1. Write a program displaying a right triangle along with squares drawn on each of the three sides in order to illustrate the Pythagorean theorem. Allow the user to dynamically change the shape of the right triangle.

2. The left diagram in Figure 1.6 has a square in the middle turned at an angle. We can replicate the same diagram inside this smaller square by drawing four more right triangles. To construct the new triangles, divide the side of the smaller square in the same ratio $(a : b)$ as the division on the side of the larger square. The process can be repeated many times to give an image of spiralling squares. Write a program to produce this image with as many spiralling squares as the user wishes. Also allow input for the ratio $(a : b)$. The key is to find the vertices of each smaller square.