Main content

Chapter 13

Simulation and Modeling

Chapter Introduction

r studying this chapter, you will be able to:

Describe the purpose of modeling in science
List the benefits of a computational model over a physical model
Explain the trade-off between accuracy and complexity in models
Define different types of simulation models, including discrete and continuous, deterministic and stochastic
Describe how a discrete event simulation works
Explain the purpose of scientific visualization

List some common methods of scientific visualization
Change font sizeMain content

Chapter Contents

13.1 Introduction

The computational devices of the 19th and early 20th centuries were used to solve important mathematical and scientific problems of the day. We saw this in the historical review of computing in Chapter 1: Charles Babbage's Difference Engine evaluated polynomial functions; Herman Hollerith's punched-card machines carried out a statistical analysis of the 1890 census; ENIAC computed artillery ballistic tables; and

Alan Turing's Colossus cranked away at Bletchley Park, breaking the "unbreakable" German Enigma code. The users of these early computing devices were almost exclusively mathematicians, physicists, and engineers.

Today, there is hardly a field of study or aspect of society—from art to zoology, business to entertainment—that has not been profoundly changed by information technology and telecommunications. Now we use computers in many "nonscientific" ways, such as buying and selling products (a topic we investigate in Chapter 14), playing games (the focus of Chapter 16), surfing the web, listening to music, and sharing photos and videos with friends.

However, the physical, mathematical, engineering, and economic sciences are still some of the largest users of computing and information technology. In this chapter, we investigate perhaps the single most important scientific use of computing—computational modeling. This application is having a major impact on many quantitative fields, including chemistry, biology, medicine, meteorology, ecology, geography, and economics.

13.2 Computational Modeling

13.2.1 Introduction to Systems and Models

The *scientific method* entails observing the behavior of a system and formulating a hypothesis that tries to understand and explain that behavior. We then design and carry out experiments on the system to either prove or disprove the validity of our hypothesis. This is the fundamental way of gaining new scientific knowledge and understanding.

Scientists often work with a model of a system rather than experimenting on the "real thing." Models were discussed in Chapter 12, where the Turing machine was presented as a model of a computing agent. A **model** is a representation of the system being studied, which we claim behaves much like the original. If that claim is valid, then we can experiment on the model and use the results to understand and explain the behavior of the actual system. For example, *physical models* (small-scale replicas) have been in use for many years, and we are all familiar with the idea of testing a model airplane in a wind tunnel to understand how the full-sized aircraft would behave. Small-scale mockups of a building give architects a sense of how the full-scale structure will function in its surroundings.

In this chapter, we are not interested in physical models but in **computational models**, also called **simulation models**. In a computer simulation, a physical system is not modeled as a small-scale replica but as a set of mathematical equations and/or algorithmic procedures that capture the fundamental properties and behaviors of a system. This computational model is then translated into a computer program written in one of the high-level languages described in Chapters 9 and 10 and executed on the Von Neumann computer described in Chapters 4 and 5.

Why construct a simulation model? Why not simply study the system itself, or a physical replica of the system? There are many reasons:

- *Existence*—The system might not yet exist, so it is not possible to experiment directly on the actual system. In fact, we might be using a model to help us with the construction of the system.
- *Physical realization*—The system is not constructed from entities that can be represented by physical objects. For example, it may be a social system (such as welfare policies or labor practices) that can only be simulated on a computer.
- *Safety*—It might be too dangerous to experiment on the actual system or a physical replica. For example, you would not want to try out a new monetary policy that could economically devastate a population or build a nuclear reactor using a new and unproven technology.
- *Speed of construction*—It might take too much time to construct a physical model. Sometimes it is faster to design and build a computer simulation.
- *Time scale*—Some physical systems change too slowly or too quickly. For example, an elementary particle in a high-speed accelerator may decompose in seconds. At the other end of the time scale, some ecosystems take thousands of years to react to a modification. A simulation can easily model fractions of a second or billions of years because time is simply a parameter in an equation.
- *Ethical behavior*—Some physical models have serious moral and ethical consequences, perhaps the best known being the use of animals in medical research. In this case, a computational model could eliminate a great deal of suffering.
- *Ease of modification*—If we are not happy with the design of a physical model, we need to construct a brand-new one. In a simulation, we only need to change some numerical parameters and rerun the existing model.

This last advantage—ease of modification—makes computational modeling a particularly attractive tool for designing totally new systems. We initialize the model, observe its response, and if we are not satisfied, modify the parameters and run the model again. We repeat this process over and over, always trying to improve system performance. Only when we think we have created the best design possible would we actually build it. This “interactive” approach to design, called **computational steering**, is usually infeasible using physical models, as it would take too much time and require too many rebuilds. This interactive design methodology is diagrammed in Figure 13.1.

Figure 13.1 Using a simulation in an interactive design environment

Computational models are therefore an excellent way to design new systems and to study and improve the behavior of existing systems. Virtually every branch of science and engineering makes use of models, and it is not unusual today to see chemists, biologists, economists, and physicians conducting fundamental research at their computer screens rather than in the laboratory.

Computational models often use advanced mathematical techniques that are far beyond the scope of this text (and solving them often requires the large-scale parallel computers described in Section 5.4). Therefore, in the following sections we often must rely on rather simple examples, far simpler than the models you will encounter in the real world. However, even these simple examples illustrate the enormous power and capabilities of computational modeling.

13.2.2 Computational Models, Accuracy, and Errors

Legend says that in the late 16th century, the famed scientist Galileo Galilei dropped two balls from the top of the Tower of Pisa—a massive iron cannonball and a lighter wooden one—to disprove the Aristotelian theory, which predicted that heavy objects would fall faster than light ones. When Galileo dropped the two balls, they hit the ground at the same time, exactly as he had hypothesized. Whether this event actually took place (and there is considerable debate), it is an excellent example of scientific experimentation using a physical system, in this case two balls of different weight, a high platform, and the Earth below.

Today, we do not need to climb the Tower of Pisa because there is a well-known mathematical model that describes the behavior of a falling mass acted upon only by the force of gravity:

This equation says that if a mass in free fall has an initial velocity meters/sec at time 0, then at time *t* it will have fallen a distance of *d* meters. The factor *g* is the acceleration due to gravity, which is assumed to be 9.8 meters/ everywhere on the Earth's surface. (Notice that the object's mass is not part of the equation. This is exactly what Galileo was trying to demonstrate.)
Using this model, we can reproduce aspects of Galileo's experiment without having to travel to Italy. For example, we can determine the time when the two balls Galileo dropped from the 56-meter-high Tower of Pisa would have hit the ground, assuming that their initial velocity was 0.0:

This simple example shows the beauty and simplicity of computational models. Such models can provide quick answers to questions without the cumbersome setup often required of physical experiments. This model is also easy to modify. For example, if we want to know how long it would take for those same two balls to hit the ground when dropped from a height of 150 meters, rather than 56, we simply reset *d* to 150 and solve the same equation:

To use a physical model, Galileo would have had to scour the 16th-century world to find a 150-meter-high tower. (He would have had to travel 1,740 kilometers to Lincoln Cathedral in England, then the tallest building in the world at 159 meters.)

Unfortunately, computer modeling is not quite as simple as we have just described, and there are a number of issues that must be addressed and solved to make this technique workable.

The first issue is achieving the proper balance between *accuracy* and *complexity*. Our model must be an accurate representation of the physical system, but at the same time, it must be simple enough to implement as a program or a set of equations and solve on a computer in a reasonable amount of time. Often this balance is not easy to achieve, as most real-world systems are acted upon by a large number of external factors. We need to decide which of those factors are important enough to be included in our model and which can safely be omitted without jeopardizing the validity of our conclusions.

For example, the model of a falling body given earlier is inaccurate because it does not account for the effects of air resistance. (It is only an appropriate model if the object is falling in a vacuum.) Whereas the effect of air resistance on a cannonball is minimal, imagine dropping a feather! The model would produce inaccurate results, and our conclusions about how the system behaves would be totally wrong. It is obvious that we need to incorporate the effects of air resistance into our model if we have any hope of producing worthwhile and useful results.*

Our model also assumes that the Earth is a perfect sphere and that the acceleration due to gravity is constant everywhere along its surface. That assumption is not quite true. The Earth is a “slightly squashed” sphere with a radius of 6,378 km at the equator and 6,357 km at the poles. This means the acceleration due to gravity is a tiny bit greater at the North and South Poles (9.83 m/) than at the equator (9.78 m/), because the poles are 21 km closer to the center of the Earth. It also changes from the top of Mount Everest to the depths of Death Valley. Is this something for which we should account? Is this effect important when constructing a model of a freely falling body? In this case, probably not—because the miniscule error resulting from this approximation will almost certainly not affect our conclusions.

This is how computational models are built. We include the truly important factors that act upon our system so that our model is an accurate representation but omit the unimportant factors that add little to our understanding and only make the model harder to build and solve. As you might imagine, identifying these factors and distinguishing the important from the unimportant can be a daunting task.

Another problem with building simulations is that we may not know, in a mathematical sense, exactly how to describe certain types of systems and behaviors. The gravitational model given earlier is an example of a continuous model. In a **continuous model**, we write out a set of explicit mathematical equations that describes the behavior of a system as a continuous function of time *t*. These equations are then solved on a computer system to produce the desired results. Unfortunately, there are many systems that cannot be modeled using precise mathematical equations because researchers have not yet discovered exactly what those equations should be. Simply put, science is not yet sufficiently knowledgeable about how some systems function to characterize their behavior using explicit mathematical formulae.

In some cases, what makes these systems difficult to model is that they contain **stochastic components**, that is, parts of the system that display *random behavior*, much like the throw of the dice or the drawing of a playing card. In these cases, we cannot say with mathematical certainty what will happen to our system because it is the very essence of randomness that we can never know exactly which event will occur next. An example of this is a model of a business in which customers walk into the store at random times. In these cases, we need to build models that use *statistical approximations* rather than exact equations. We will present one such example in the following section.

In summary, computational modeling is a powerful but complex technique for designing and studying systems. However, building a good computational model can be a difficult task that requires us to capture all the important factors that influence the behavior of a system. If we are able to successfully build such a model, then we have at our disposal a powerful tool for studying the behavior of that system. This is how a good deal of quantitative research is done today. Simulation is also an interesting area of study within computer science itself. Researchers in this field create new techniques, both algorithms and special-purpose languages, that allow users to design and implement computer models more quickly and easily.

Main content

Chapter Contents

13.2.3 An Example of Model Building

As we mentioned at the end of the previous section, there are many ways to build a model, but most of them require mathematical techniques that are far beyond the scope of this text. In this section, we construct a model using a method that is relatively easy to understand and does not require a lot of complex mathematics. It is called *discrete event simulation*, and it is a very popular and widely used technique for building computational models.

In **discrete event simulation**, we do not model time as continuous, like the falling body model in the last section, but as *discrete*. That is, we model the behavior of a system only at an explicit and finite set of times. The moments we model are those times when an event takes place, an **event** being any activity that changes the state of our system. For example, if we are modeling the operation of a department store, an event might be a new customer entering the store, a customer purchasing an item, or a customer departing the building.

When we process an event, we change the state of the simulated system in the same way that the actual system would change if this event had occurred in real life. In the case of a department store, this might mean that when a customer arrives we add one to the number of customers currently inside the store or, if a customer buys an item, we decrease the number of these items on the shelf. Furthermore, the processing of one event can cause new events to occur at some time in the future. For example, a customer coming into a store creates a later event related to that customer leaving the store. When we are finished processing one event, we move on to the next, skipping those times in between when nothing is happening—that is, when there are no events scheduled to occur.

Figure 13.2(a) shows system S and three events scheduled to occur within system S: event at time 9:00, event at time 9:04, and event at time 9:10. Because is the event currently being processed, the variable current time, which functions like a “simulation clock,” has the value 9:00. Let’s assume that causes a new event, , to be created and scheduled for time 9:17. We add this new event to the list of all scheduled events. When we are finished processing event , we remove it from the list and determine the next event scheduled to occur in system S, in this case . We move *current time* ahead to 9:04, skipping over the time period 9:01–9:03, because nothing of interest happens, and begin processing . The new list of events scheduled for system S is shown in Figure 13.2(b).

Figure 13.2 Example of simulated events

We repeat this sequence—process an event, remove it from the list, add newly created events to the list, move on to the next event—as long as desired. The variable *current time* keeps advancing as we process the events in strict time order. Typically the simulation is terminated when *current time* reaches some upper bound. For example, in a department store we might choose to run the model until closing time. When the simulation is complete, the program displays a set of results that characterizes the system's behavior and allows the user to examine these results at his or her leisure.

Let's apply this modeling technique to an actual problem. Assume that you have been hired as a consultant by the owner of a new fast-food restaurant, McBurgers, currently under construction. The owner wants to determine the proper number of checkout stations that will be needed in the new store. This is an important business decision because if there are too few checkout stations, the lines will get long and customers will become irritated and leave. If there are too many checkout stations, money will be wasted paying for unnecessary construction costs, equipment, and personnel. You could just make an educated guess at the optimal number but, because you took a computer science class in college, you decide the best way to advise your client is to construct a simulation model of the new restaurant and use this model to determine the optimal number of servers.

The system being simulated is shown in Figure 13.3. Customers enter the restaurant and wait in a single line for service. If any of the *N* servers is available, where *N* is an input value provided by the user, the first customer in line goes to that station, places an order, waits until the order is processed, pays, and departs. During that time, the server is busy and cannot help anyone else. When the server is finished with a customer, he or she can immediately begin serving the next person, if someone is in line. If no one is waiting, then the server waits until a new customer arrives.

Figure 13.3 System to be modeled

To create this model, we must first identify the events that can change the state of our system and thus need to be included in the model. In this example, there are only two:

- (1)

  a new customer arriving, and
- (2)

  an existing customer departing after receiving his or her food and paying.

An arrival changes the system because either the waiting line grows longer by one, or, if there is no waiting line, an idle cashier becomes busy. A departure changes the system because the cashier serving that customer either begins serving a new customer or becomes idle because no one is in line.

For each of these two events, we must design an algorithm that describes exactly what happens to our system when that event occurs. Figure 13.4 shows the algorithm for the new customer arrival event.

Figure 13.4 New customer arrival algorithm

Let’s examine this algorithm in detail. When a new customer arrives, we record the current simulation time. The arrival time of each new customer is stored in a variable associated with that specific customer until that individual is served and departs. As we mentioned earlier, when the simulation is finished we want to display a set of results that allows the user to determine how well the system has performed. The total time a customer spends in the restaurant (waiting time 1 service time) is a good example of this type of result. If this value is large, we are not doing a good job serving customers, and we need to increase the number of servers so customers don’t wait so long. A key part of any simulation model is collecting important data about the system so that we can understand and analyze its performance.

The next operation in the new customer arrival algorithm of Figure 13.4 is to determine if there is an idle server. If not, the customer goes to the end of the waiting line (no special treatment here at McBurgers), and the length of the waiting line is increased by 1. If there is an idle server, then the customer goes directly to that server, who is then marked as busy. (*Note*: If more than one server is free, the customer can go to any one because our model assumes that all servers are identical. We could also construct a model in which not all servers are identical and some provide a special service.)

Now we must determine how much time is required to service this customer. This is a good example of what we termed a *stochastic*, or *random*, component of a simulation model. Exactly what a customer orders and how much time it takes to fill that order are random quantities whose exact values can never be known with certainty in advance. However, even though it behaves in a random fashion, it is possible that this value, called in Figure 13.4, follows a pattern called a **statistical distribution**, a mathematical equation specifying the probability that a random variable takes on a certain value. If we know this pattern, then the computer can generate a sequence of random numbers that follows this pattern, and this sequence will accurately model the time it takes to serve customers in real life.
How can we discover this pattern? One way is to know something about the statistical distribution of quantities that behave in a similar way. For example, if we know something about the distribution of service times for customers in a bank or a grocery store, then this information might help us understand the pattern of service times at our hamburger stand. Another way is to observe and collect data from an actual system similar to ours. For example, we could go to other fast-food restaurants and measure exactly how long it takes them to service their customers. If these restaurants are similar to ours, then the McBurgers owner might be able to discover from this data the statistical distribution of the variable .
There are other ways to work with statistical distributions, but we will leave this complex topic to courses in statistics. In this example, we simply assume that the statistical distribution for the customer service time, , has been discovered and is shown in the graph in Figure 13.5.

Figure 13.5 Statistical distribution of customer service time

The graph in Figure 13.5 states that 5% of the time a customer is served in less than 1 minute; 15% of the time it takes 1–2 minutes; 40% of the time it takes 2–3 minutes; 30% of the time it takes 3–4 minutes; and, finally, 10% of the time it takes 4–5 minutes. It never requires more than 5 minutes to serve a customer. We can model this distribution using the algorithm shown in Figure 13.6.

Figure 13.6 Algorithm for generating random numbers that follow the distribution given in Figure 13.5

First, we generate a random integer *v* that takes on one of the values 1, 2, 3, ..., 100 with equal likelihood. This is called a **uniform random number**. We now ask if *v* is between 1 and 5. Because there are five numbers in this range, and there were 100 numbers that could originally have been generated, the answer to this question is yes 5% of the time. This is the same percent of time that customers spend from 0 to 1 minute being served. Therefore, we generate another uniform random value, this time a real number between 0.0 and 1.0, which is the value of , the customer service time.
If the original random value *v* is not between 1 and 5, we ask if it is between 6 and 20. There are 15 integers in this range, so the answer to this question is yes 15% of the time, exactly the fraction of time that customers spend 1–2 minutes being served. If the answer is yes, we generate a value that is in the range 1.0 to 2.0. This process is repeated for all possible values of service time.
Once the value of has been generated, we use this value to determine exactly when this customer leaves the store (*current time* +) as well as to update the total amount of time the server has spent serving customers. This last computation allows us to determine the percentage of time during the day that each server was busy.
The value assigned to using the algorithm of Figure 13.6 exactly matches the statistical distribution graph shown in Figure 13.5. If this graph is an accurate representation of customer service time, then our model is an accurate depiction of what happens in the real world. However, if the graph of Figure 13.5 is not an accurate representation of customer service time, then our model is incorrect and will produce wrong answers. This is a good example of the well-known computer science dictum **garbage in, garbage out**—the results you get from a simulation model are only as good as the data and the assumptions put into the model.
We can now specify how to handle the second type of event contained in our model, customer departures. The algorithm to handle a customer leaving the restaurant is given in Figure 13.7.

Figure 13.7 Algorithm for customer departure event

When a customer is ready to leave, we determine the total time this customer spent in the restaurant. The variable *current time* represents the time now, which is the time of this customer's departure. We recorded the time this customer first arrived on Line 2 of Figure 13.4, and we can retrieve the contents of the variable storing that information. The difference between these two numbers is the total time this customer spent in the restaurant, including both waiting time and service time. We use this result, averaged over all customers, to determine if we are providing an adequate level of service.

If there is another customer in line, the server begins serving that customer in exactly the same way as described earlier. If no one is waiting, then the server becomes idle and has nothing to do until a new customer arrives. (We don't want this to happen too often because then the restaurant owner will be paying the salary of someone with little to do.)

We have now described the two main events that change our system: someone arriving at the restaurant and someone leaving the restaurant. The only thing left is to initialize our parameters and get the model started. To initialize the model, we must do the following four things:

- Set the current time to 0.0 (we begin our simulation at time 0).
- Set the waiting line size to 0 (no one is in line when the doors open).
- Get a value for *N*, the number of servers, and make them all idle.
- Determine the total number of customers to be served and exactly when they will arrive.

The last value—customer arrival times—are like the service times discussed earlier in that they are stochastic, or random, values that cannot be known in advance. We cannot possibly know exactly when each new customer will walk in the front door. However, if we can determine the statistical distribution of the time interval between the arrival of any two customers, then we can generate a set of random intervals, called , that will allow us to model customer arrivals. It would also allow us to examine what would happen if our restaurant became more (or less) busy. We could simulate that effect by simply decreasing (or increasing) the average value of the time interval between customers.

Assume we have a graph like Figure 13.5 that specifies the statistical distribution of the time interval that elapses between the arrivals of two successive customers. (That is, it might say something like 10% of the time two customers arrive within 0–15 seconds of each other, 20% of the time they arrive within 15–30 seconds of each other, and so on.) We schedule our first customer to arrive at time 0.0, just as the doors open. We then use an algorithm like the one in Figure 13.6 to generate a random value that matches the distribution of interarrival times. Call this value . This represents the amount of time that will elapse until the next customer arrives. Because the first customer arrived at time 0.0, we schedule the next one to arrive at . We repeat this for as many customers as desired, scheduling each one to arrive at time units after the previous one. Our sequence of customer arrivals will look something like this:

The main program to run our McBurgers simulation model is given in Figure 13.8. It allows the user to provide two inputs: *M*, the total number of customers to model, and *N*, the number of servers. Each one of the *M* customer arrivals is handled by the arrival algorithm of Figure 13.4. Each arrival event generates a customer departure event that is handled by the departure algorithm of Figure 13.7. This simulation does not terminate at a specific point in time but, instead, when there are no more events to be processed—that is, when every one of the *M* customers scheduled to arrive has been served and has departed.

Figure 13.8 **The main algorithm of our simulation model**

The last issue that we must address is how to implement the second-to-last line of Figure 13.8, the one that reads, "Print out a set of data that describes the behavior of the system." Looking back at Figure 13.1, we see that one of the responsibilities of a simulation is to "collect data describing its behavior." Our model must collect data that accurately measures the performance of this McBurgers restaurant so that the new owner can configure it in a profitable manner *before* it is built. Therefore, we need to determine what data are required to meet this need. Often this cannot be done by the persons building the model because they are computer scientists and may be unfamiliar with this application area. Instead, it is the *users* of a model who can best determine what data should be collected and displayed. In this case, the user is the restaurant owner. Thus, model building is often a cooperative effort between technical specialists in the area of software development and those knowledgeable about the unique characteristics of the system being modeled.

Let's assume that we have talked to the restaurant owner and determined that the information he or she most needs to know is the following:

- The average time that a customer spends in the restaurant, including both waiting in line and getting served
- The maximum length of the waiting line
- The percentage of time that servers are busy serving customers

From this data, the owner should be able to determine whether the system is functioning well. For example, if our model determines that a server is busy only 10% of the time (about 48 minutes in an 8-hour workday), we can probably reduce the number of servers without affecting service, saving a good deal in salary costs. On the other hand, if the average time that a customer spends in the restaurant is 1 hour or there are times when there are 100 people in line, then we had better increase the number of servers if we want to avoid bankruptcy (or riots)!

This model will likely be used in the interactive design approach first diagrammed in Figure 13.1. The owner will enter his or her best estimate for the arrival time and service time distributions and then select a value for *N*, the number of servers. The computer will run the simulation, processing all *M* customers, and then print the results, perhaps something like the following:

With only two servers, our customers waited on average more than one hour to be served, there were dozens of people in line, and both servers were busy every second of the day—not very good performance! The owner would certainly try to improve on this performance, perhaps by having 6 servers, rather than only 2. He or she resets the parameter *N* to 6 and reruns the model, which now produces the following:

Now the owner may have erred too far in the other direction. Our customers are being well served, waiting only a couple of minutes, and the line is tiny, never having more than a single person. However, on the average our six servers are busy only 43% of the time—meaning they are idle about 4.5 hours during an 8-hour workday. Could we provide the same high level of service to our customers with fewer servers? To answer this question, the owner might try rerunning the model with , 4, or 5, a compromise value between these two extremes. The owner may also want to study how well this number of servers works when the restaurant gets slightly more or less busy. This is how a simulation model is used—run it repeatedly under different assumptions, examine the results, and use these results to reconfigure the system being modeled so its performance is enhanced.

This completes the development of our McBurgers simulation but is not the end of its usefulness. In the next Laboratory Experience, you are going to "play" with this model by selecting a range of values for customer arrival and service times. You then take on the role of the McBurgers owner and determine the optimal number of servers to use for the selected configuration. Working with a simulation in an interactive design environment demonstrates the enormous power and capabilities of computational models.

The restaurant modeled in this section is about as simple a system as we could present, yet it still took about 10 pages to describe its design.

Practice Problems

In the McBurgers new customer arrival algorithm, describe the consequences of accidentally omitting the instruction “Mark that server is now busy.”
Answer

Without this instruction, server is serving a newly arrived customer but is still marked as idle; the next newly arrived customer could try to go to that server.
In the McBurgers customer departure algorithm, describe the consequences of accidentally omitting the instruction “Mark this server as idle.”

Answer

Without this instruction, server has finished serving a customer and there are no customers waiting in line, but the next newly arrived customer will not try to be served by .
Suppose we try to simplify our model by assuming that every customer requires exactly two minutes of service to complete his or her purchase. How do you think this would affect the conclusions that we could draw from our model?

Answer

It most likely would make the conclusions less valid and much less usable. The idea of every customer taking the same amount of time is unrealistic. Therefore, making this assumption would cause our model to be a very poor abstraction of the real system. If your assumptions are wrong, then there is a far greater likelihood that your conclusions will be wrong as well. (We called this “garbage in, garbage out” in the text.)

Are there other parameters that you might have included if you were building a model of a fast-food restaurant?

Answer

There are many other possibilities, but here are a few:

a.

The possibility of mechanical breakdowns. The model could be designed to include the occasional breakdown of a critical component (e.g., the French fryer) to study how the system responds to these types of unexpected events.

b.

Rather than say no customer ever takes more than 5 minutes to be served, the model could be designed to allow for an occasional massive order—someone purchasing 100 hamburgers—to see what happens to our waiting times when this unusual event occurs.

c.

Some servers could be used only for special orders, rather than have all servers be identical.

A computational model of “El Niño” Pacific Ocean currents, the human heart, or a strand of DNA would certainly be far more complex than the simulation of a hamburger joint! Real-world models are mathematically intricate, highly detailed, and difficult to build. However, if we are able to build such a model or if we have access to such a model, then we have a powerful tool that can significantly enhance our ability to do high-quality research and design.

13.3 Running the Model and Visualizing Results

The McBurgers restaurant model developed in Section 13.2.3 is much simpler than most real-world computer models for two reasons. First, it is computationally small. The process of running it and producing results does not require much in the way of hardware resources. For example, assume that we model customers, a reasonable daily value for a large restaurant. Each customer generates one arrival event (Figure 13.4) and one departure event (Figure 13.7), for a total of 2,000 events that must be processed by the computer before the simulation is completed and the results displayed. The execution of 2,000 events is a miniscule amount of work that could be handled by even the smallest laptop or tablet in just a few seconds or, more likely, fractions of a second. Most real-world models require much more computational work to produce their results.

For example, the U.S. Department of Energy’s National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory has developed a powerful climate system model. Using this model, simulating one year of global climatic change requires about computations—one quintillion operations. A single Von Neumann machine could not handle this almost unimaginably large amount of work. A typical laptop computer executes roughly 5 billion instructions per second. At this rate, completing one year of simulated time in the model would require about six-and-a-half years of real time—we would not get our results until the actual time period being simulated had long since passed! Even a high-speed desktop system that executes 50 billion instructions per second would require over seven months of nonstop computing to produce a single result.

Massive models like this one can be executed only on the large-scale parallel machines described in Chapter 5. For example, the NERSC climate model was executed on a massively parallel Cray XC40 supercomputer containing 11,000 separate processors, and with a peak computation rate of about 10 petaflops, or floating point operations per second. At this rate, one year of climatic change can be modeled in less than two minutes!

This latter example is far more typical of the amount of computational work required by large, real-world simulations. It is not unusual for a model to perform , , , or more operations just to produce a single result—amounts far beyond the capabilities of individual single-, dual-, or quad-processor machines. The increasing interest in building complex computational models is one of the main reasons behind the development of larger and more powerful supercomputers. For example, the massively parallel Chinese Sunway TaihuLight supercomputer mentioned in Chapters 3 and 5, which runs at a rate

of 93 petaflops, will be executing computational models in the areas of climate change, neural imaging, and nuclear weapons testing.
The second reason why the McBurgers model in Section 13.2.3 is so unrealistic is that it produces only a tiny amount of output. After each run is complete, the model generates only three lines of output, such as those shown below and in the previous section:

Because the number of servers in a restaurant might range from one up to a few dozen, the total volume of output this model would ever produce is about 20–60 lines, less than a single page. With such a small amount of output, our model can display its results using a simple text format, as shown in the lines above. A user will have no difficulty reading and interpreting this output.

Unfortunately, most simulations do not produce a few dozen lines of output, but rather tens or hundreds of thousands of lines, perhaps even millions. For example, assume the NERSC climate model described earlier displayed the temperature, humidity, barometric pressure, wind velocity, and wind direction at 50-mile intervals over the entire surface of the Earth for every simulated day the model is run. After one year of simulated time, it will have produced roughly 500 million data values—about 10 million pages of output! If these values were displayed as text, it would overwhelm its users, who wouldn't have a clue how to deal with this mountain of paper.

Text, when it is presented in such large amounts, does not lend itself to easy interpretation or understanding. The field of **scientific visualization** addresses the issue of how to visualize large volumes of scientific data in a way that highlights its important characteristics and simplifies its interpretation and analysis. This is an enormously important part of computational modeling because without it we would be able to construct models and execute them, but we would not be able to interpret their results.

The term *scientific visualization* is often treated as synonymous with the related term **computer graphics**, but there is an important difference. The field of computer graphics is concerned with the technical issues involved in information display. That is, it deals with the actual algorithms for rendering a screen image—light sources, shadows, hidden surfaces, shading, contours, and perspective. (We will be discussing these operations in Section 16.2.) Scientific visualization, on the other hand, is concerned with how to display a large data set in a way that maximizes its clarity and user comprehension. It is concerned with issues such as *data presentation*—determining the optimal format for presenting data; *data extraction*—determining which values are important and should be included and which values can be omitted; and *data manipulation*—converting the data to other forms or to different units that make the information easier to understand and interpret. Once we have decided exactly how we want to display the data, then scientific visualization software typically uses a computer graphics package to display an image on a screen or printer.

For example, assume we have built a computer model of the ocean tides at some point along the coast. Our model predicts the height of the tide every 30 seconds in a 24-hour day, based on such factors as lunar phase, water depth, wind speed, and wind direction. If this information were printed as text, it might look something like the following:

Time	*Height (feet)*
12:00:00 A.M.	43.78
12:00:30 A.M.	43.81
12:01:00 A.M.	43.84
12:01:30 A.M.	43.88
12:02:00 A.M.	43.92
12:02:30 A.M.	43.97
.	.
.	.
.	.
11:57:00 P.M.	45.08
11:57:30 P.M.	45.04
11:58:00 P.M.	45.01
11:58:30 P.M.	44.99
11:59:00 P.M.	44.97
11:59:30 P.M.	44.95

There are 2,880 lines of output, which at 60 lines per page would produce almost 50 printed pages. Trying to extract meaning or locate significant features from these long columns of numbers would certainly be a formidable, not to mention boring, task.

What if, instead, we displayed these two columns of values as a two-dimensional graph of time versus height? The output could also include a horizontal line showing the average water height during this 24-hour period. This latter value is not part of the original output but can easily be computed from these values and included in the output—an example of a data manipulation carried out to enhance data interpretation. Now the output of our model might look something like the graph in Figure 13.9.

Figure 13.9Using a two-dimensional graph to display output

Using the graph in Figure 13.9, it is a lot quicker and easier to identify the interesting features of the model's output. For example,

- There appear to be two high tides and two low tides during this 24-hour time period.
- The high tide is about 8 feet above the average water level, whereas the low tide is about 8 feet below the average water level.

It is possible to extract the same information from a textual representation of the output, but it would take much more time. Interpreting the graph of Figure 13.9 is a great deal easier than working directly with the raw data. The use of visualizations becomes more and more important as the amount of output increases and grows more complex. For example, what if in addition to tidal height our model also predicted the water temperature and displayed its value every 30 seconds. Now the raw data produced by the model might look like this:

Time	*Height (feet)*	*Temperature (°C)*
12:00:00 A.M.	43.78	15.03
12:00:30 A.M.	43.81	15.02
12:01:00 A.M.	43.84	15.01
12:01:30 A.M.	43.88	14.99
12:02:00 A.M.	43.92	14.97
12:02:30 A.M.	43.97	14.94
.	.	.
.	.	.
.	.	.
11:57:00 P.M.	45.08	14.95
11:57:30 P.M.	45.04	14.98
11:58:00 P.M.	45.01	15.00
11:58:30 P.M.	44.99	15.01
11:59:00 P.M.	44.97	15.03
11:59:30 P.M.	44.95	15.05

Now there are almost 6,000 numbers, and our task has become even more difficult as we try to understand both the behavior of the *two* variables, height and temperature, as well as any possible relationship between the two. Working directly with the raw data generated by the model is cumbersome. However, if the value of both variables were presented on a single graph, as shown in Figure 13.10, the interpretation would be much easier.

Figure 13.10 Using a two-dimensional graph to display and compare two data values

Looking at Figure 13.10, we quickly observe that water temperature seems to move in the opposite direction as the tide, but delayed by a few minutes. That is, water temperature reaches its minimum value shortly after the tidal height has reached its maximum value, and vice versa. This is exactly the type of information that could be of help to a researcher. Without the graphical visualization in Figure 13.10, we may have overlooked this important relationship.

The graphs in Figures 13.9 and 13.10 are both two-dimensional, but many real-world models study the behavior of three-dimensional objects, for example, an airplane wing, a gas cloud, or the Earth’s surface. The results produced by these models are also three-dimensional, such as the spatial coordinates of a point on that airplane wing or on a gas molecule. Therefore, it is common for the output of a computational model to be displayed as a three-dimensional image rather than the two-dimensional graphs shown earlier. For example, Figure 13.11 shows output from a model of a portion of the Earth's surface overlaid with colors that represent the intensity of a forest fire at a particular moment in time. The hottest areas are shown in yellow and red, while cooler areas are displayed in blue and green. Such three-dimensional digital elevations make it easy to locate important topographical features, such as mountains, valleys, and rivers and areas where we should place our maximum effort. This type of output would be extremely useful when, for example, planning the movement of equipment to fight the fire or directing airplanes on where to drop fire retardant chemicals. Given the same information in a textual format, it would take a far, far greater amount time to extract the identical information.

Figure 13.11 Three-dimensional image of the Earth's surface with overlay showing status of a forest fire

Source: NASA

As a second example, suppose that medical researchers are using a simulation model to study the behavior of the chemical compound methyl nitrite, , a potential carcinogen found in our air and drinking water. Assume that their molecular model produces the following textual output:

		Location			
Molecule Number	**Element**	**x**	**y**	**z**	**bonded To**
1	O	1.7	1.0	0.0	3, 4
2	O	3.0	0.0	0.0	3
3	N	2.6	0.3	1.0	1, 2
4	C	0.0	0.0	0.0	1, 5, 6, 7
5	H	•0.5	0.5	0.5	4
6	H	0.5	0.5	0.5	4
7	H	•0.5	•0.5	0.5	4

This is an accurate textual description of a methyl nitrite molecule. The output specifies the seven atoms in the molecule, the spatial (*x, y, z*) coordinates of the center of each atom in the molecule, and the identity of all other atoms to which this one has a chemical bond. This is all the information required to understand the structure of this molecule. However, most of us would find it hard to form a mental image of what this molecule actually looks like using just this table.

What if, instead, our model took this textual description of methyl nitrite and used it to create and display the three-dimensional image of Figure 13.12?

Figure 13.12 Three-Dimensional Model of A Methyl Nitrite Molecule

It is certainly a lot easier to work with the visualization in Figure 13.12 than with the original textual description. For example, if our model changed the shape or structure of this molecule, say by simulating a chemical reaction or the breaking of a chemical bond, we would be able to observe this change on our computer screen, significantly increasing our understanding of exactly what is happening. In the table-based representation, we would only see changing numerical values without any clue as to what these changes represent chemically or structurally.

The image in Figure 13.12 makes use of two other features found in many visualizations—color and scale. These characteristics allow us to display information in a way that makes the image more understandable by someone looking at the diagram. In this example, color represents the element type—blue for hydrogen, yellow for carbon, purple for oxygen, and red for nitrogen. The relative size of each sphere represents the relative size of each of the atoms.

The clever use of visual enhancements such as color and scale can make an enormous difference in how easy or hard it is to interpret the output of a computer model. For example, the image displayed in Figure 13.13 models the dispersion and height of tsunami waves following a hypothetical earthquake near Japan.

Figure 13.13 Visualization of Projected Tsunami Wave Heights

Source: NOAA Center for Tsunami Research

In this example, color indicates projected wave heights across the Pacific Ocean. The largest wave heights, shown in purple, are expected near the earthquake epicenter off the Japanese coast. Progressively smaller waves are indicated in red, orange, and yellow. Using images like Figure 13.13, it is easy to see the areas likely to be most impacted, information that helps relief organizations determine where the greatest assistance is needed. If, instead of these color-coded images, we were given only page after page of numerical values, it would take much longer to extract this vital information. Here is another example, along with the forest fire status model of Figure 13.11, in which enhancing the comprehension of a model's output is not just for convenience but for saving lives!

Finally, we mention one of the most powerful and useful forms of scientific visualization—**image animation**. In many models, time (whether continuous or discrete) is one of the key variables, and we want to observe how the model's output changes over time. This could be the case, for example, with the forest fire model discussed earlier. The image in Figure 13.11 is a picture of the fire at one discrete instant in time. That may be of value, but what might be of even greater interest is how the fire moves and disperses as a function of time. Some questions we could answer using this time-varying model are: How long does it take for the hottest areas (yellow and red) to dissipate completely? What areas change more rapidly from hot to cool?

To answer these and similar questions, we need to generate not one image like Figure 13.11, but many, with each image showing the state of the system at a slightly later point in time. If we generate a sufficient number of these images, then we can display them rapidly in sequence, producing a visual animation of the system's behavior over time.

Obviously we cannot show an animation in this book, but Figure 13.14 shows two images (out of 365) from a program that models the total amount of ozone present in the Earth's atmosphere over a one-year period. The model computes the ozone levels for each day of the year and displays the results graphically, with green and blue representing acceptable ozone levels and red representing a dangerously low level. These 365 images can be displayed in sequence to produce a "movie" showing how the ozone level changes throughout the year.

Figure 13.14 Use of animation to model ozone layers in the atmosphere

Source: Lloyd A. Treinish/IBM Thomas J. Watson Research Center

The amount of output needed to produce these 365 images was probably in the range of tens or hundreds of billions of data values. If this volume of data were displayed as text, a user would be overwhelmed, and the truly important characteristics of the data would be buried deep within this mass of numbers, much like the proverbial "needle in a haystack." However, using the visualization techniques highlighted in this section—two- and three-dimensional graphics, color, scale, and animation—key features of the data, such as the presence of a significant ozone hole (the red area) over the Antarctic on day 292, can be quickly and easily located.

This is precisely why scientific visualization techniques are so important. Their goal is to take a massive data set and present it in a way that is more informative and more understandable for the user of that data. Without this understanding, there would be no reason to build computational models in the first place.

Main content

Chapter Contents

13.4 Conclusion

Computational modeling is a fascinating and highly complex subject, and one that will become even more important in the coming years as computers increase in power and researchers gain experience in designing and building models.

Constructing models of complex systems requires a deep understanding of both mathematics and statistics so, as we have mentioned a number of times, they can be rather difficult to build. However, even if you are not directly involved in building computer models, it is quite likely that you will be working with these types of models in your research, development, or design work. Simulation is affecting many fields of study. For example, in this chapter we looked at models drawn from physics (the falling body equations), economics (the McBurgers simulation), chemistry (the molecular model of methyl nitrite), cartography (a map of the Earth's surface), meteorology (tides, climatic changes), and ecology (forest fire dispersion, ozone depletion). We could just as easily have selected examples from the fields of medicine, geology, biology, pharmacology, or urban planning. For those who work in scientific or quantitative fields like these, computational modeling is rapidly becoming one of the most important tools available to the researcher. It is also a vehicle for amusing and entertaining us through the creation of simulated fantasy worlds and alien planets where we can relax, explore, and play. We'll discuss this exciting role of simulation in Chapter 16.

The Mother of All Computations!

In May 2014, a research group from the Physics department of MIT reported in the journal *Nature* that they had completed the most detailed and realistic computer simulation ever built to study the formation of the universe. The model, called *Illustris*, tracked the evolution of the cosmos from a few hundred thousand years after the Big Bang up to the current time, 13.8 billion years later. Illustris uses equations drawn from physics, astronomy, hydrodynamics, chemistry, and mathematics to study a cube of simulated space that is 350 light years on each side and that contains tens of thousands of galaxies. The model, executed on supercomputers in France and Germany, simulated the expansion of galaxies, the gravitational pull of matter, the formation of stars and black holes, and the motions of cosmic gases.

It took five years for the software teams in the United States, France, and Germany to design, program, and test Illustris, and it required three months of nonstop parallel computing using two supercomputers with 8,000 processors each to execute the approximately 1021 (one sextillion, or one billion trillion) operations needed to simulate 14 billion years of galactic evolution. Researchers estimated that if this model were run on a high-perforamce desktop computer, it would have taken about 1,500 years to complete. You can read more about this enormous simulation project at www.illustris-project.org/about/

Even though simulation is an important scientific application, you are probably more familiar with the many uses of computers in the commercial sector—paying bills online, remotely accessing financial data, and buying and selling products on the web. These commercial applications, often grouped together under the generic term *electronic commerce*, or *ecommerce*, will be discussed at length in Chapter 14.