PHY 1200: Physics for Video Games

Notes, Labs, and Programs

Aaron Titus

High Point University

Spring 2016

http://physics.highpoint.edu/∼atitus/

Contents

1 Coordinates

Cartesian Coordinate System

To specify the location of an object, we use a coordinate system. The one shown in Figure 1.1 is a two-dimensional (2-D) Cartesian coordinate system with the $+x$ direction defined to the right and the $+y$ direction defined to be upward, toward the top of the page.

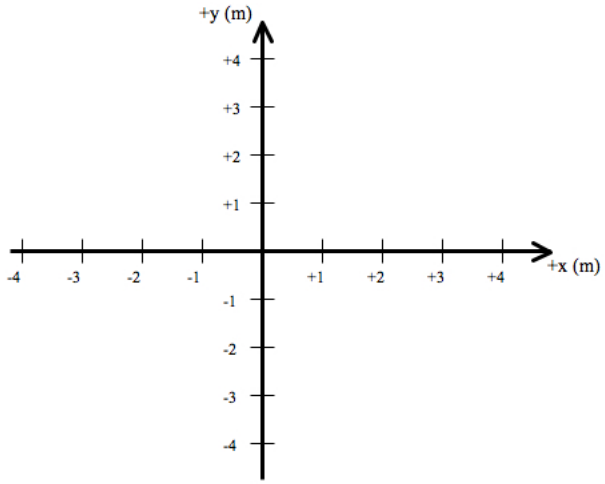

Figure 1.1: A 2-D Cartesian coordinate system.

A three-dimensional (3-D) coordinate system with the $+z$ axis defined to be outward toward you, perpendicular to the page, is shown in Figure 1.2.

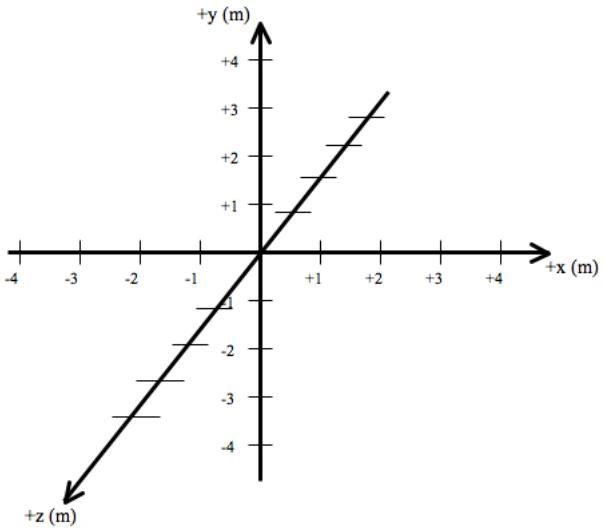

Figure 1.2: A 3-D Cartesian coordinate system.

A coordinate system is defined by:

1. an origin
2. a scale (determined by the tick marks, numbers, and units)
3. an orientation; the direction of the +x, +y, and +z directions, respectively.

The orientation is described verbally by saying something like "The +x direction is toward the right of the origin." Or, "The +y direction is upward toward the top of the page."

Coordinates

A point on a Cartesian coordinate system is designated by a pair of numbers in 2-D and a triplet of numbers in 3-D. On a 2-D coordinate system, the pair of numbers represents the (x, y) coordinates of the point.

On the coordinate system in Figure 1.3, the red dot is at the location $(+1, +3)$ meters. And the blue dot is at the location $(+4, -4)$ meters. Thus, we say that the x-position of the red dot is +1 m, and the y-position of the red dot is +3 m. Likewise, the x-position of the blue dot is +4 m, and the y-position of the blue dot is -4 m.

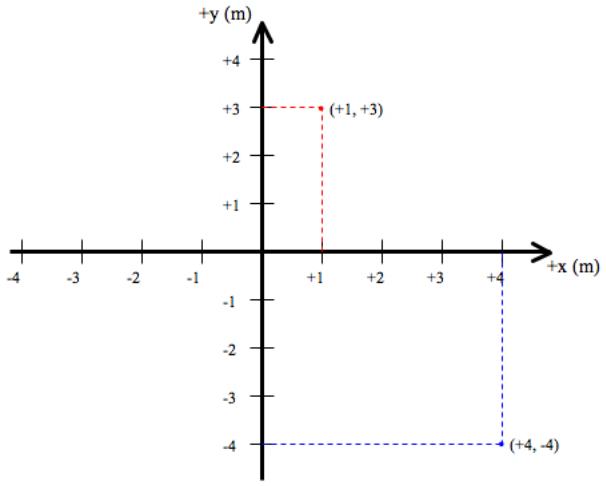

Figure 1.3: Coordinates of certain points on a coordinate system.

The sign of the coordinate tells us the side of the origin that the point is on. Thus, $x = +1$ means that the point is on the right-side of the origin. If $x = -1$ then the point is on the left-side of the origin.

Likewise, $y = +3$ means that the point is above the origin, and $y = -4$ means that the point is below the origin.

Question: State in words the location of a point with a positive z-coordinate.
Answer: Using the coordinate system in Figure 1.2, a positive value of z means that the point is in front of the page (which we assume to be the x-y plane), toward you. For example if you are reading this page, then your eyes have positive z coordinates.

Computer Convention

Programming languages define the origin of the monitor to be at the top left corner. The $+x$ axis is to the right, and the $+y$ axis is downward, as shown in Figure 1.4.

The units, in computer graphics, are pixels. If the resolution of a monitor is is 1440×900, it means that the monitor displays 1440 pixels horizontally and 900 pixels vertically. Since the origin is in the top left corner, the coordinates of a single pixel on a monitor are always positive.

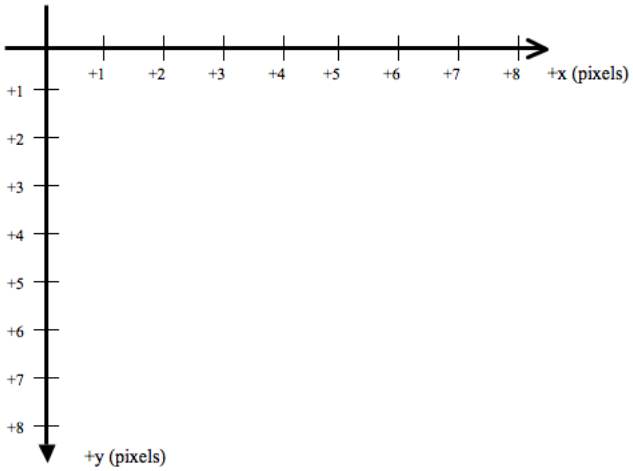

Figure 1.4: Convention for pixel coordinates on a computer monitor.

Example

Question: What are the coordinates of point A in Figure 1.5?
Answer: For point A, $x = +2$ m and $y = +3$ m. Thus, it's coordinates are $(+2, +3)$ m.

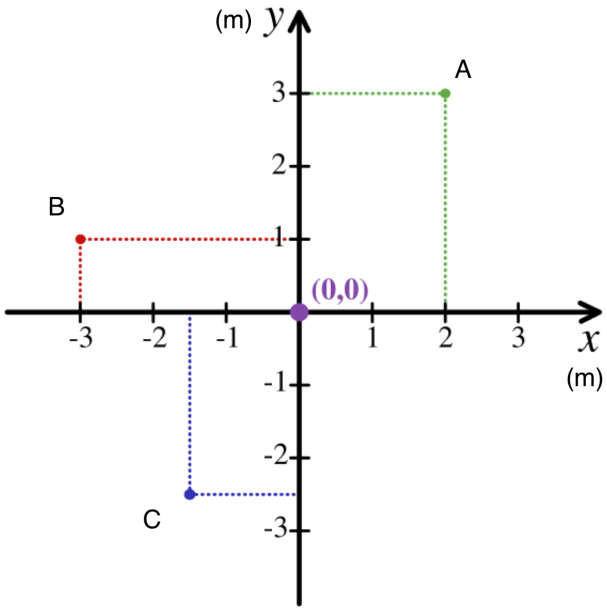

Figure 1.5: Three points on a Cartesian coordinate system.

Homework

1. What are the coordinates of points B and C in Figure 1.5?

2. A puck travels across the monitor in a computer game as shown in Figure 1.6. The top left corner of the image is the origin. Each line on the grid represents 10 pixels. The puck moves from the top, right side of the monitor to the bottom, left side of the monitor. What are the coordinates in pixels of each image of the puck?

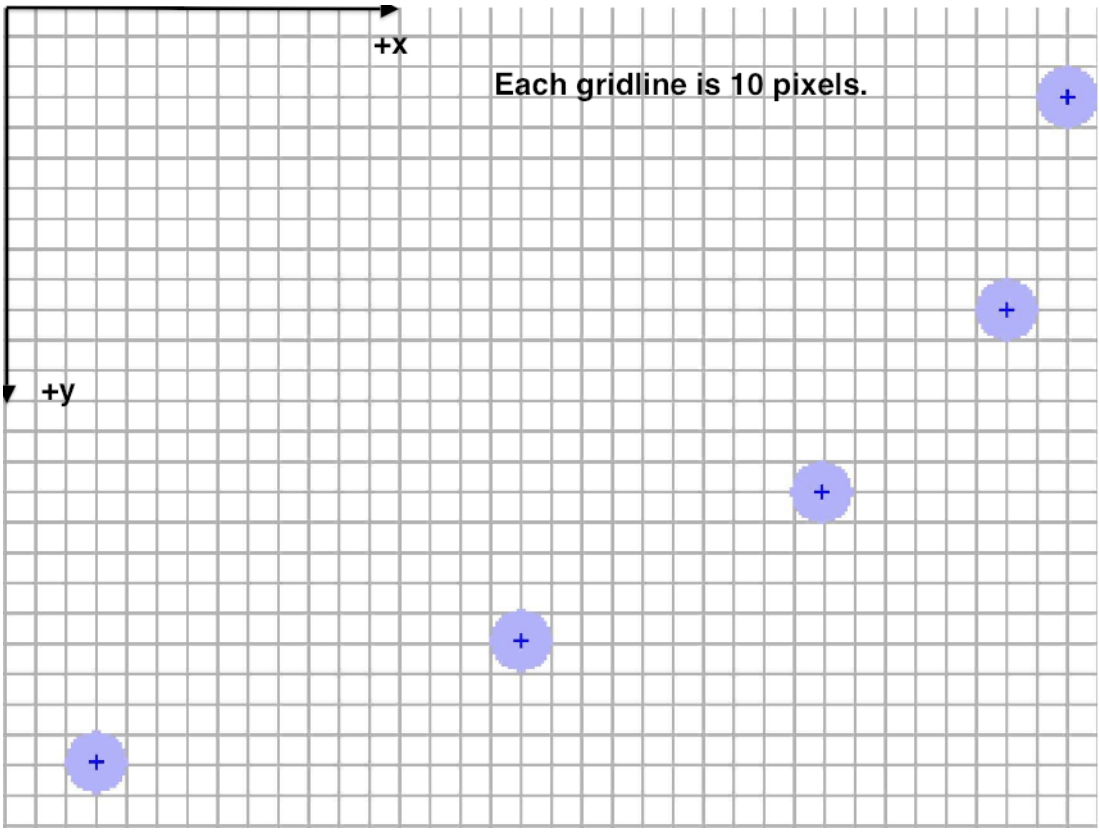

Figure 1.6: A puck on a computer monitor.

2 LAB: Coordinates

Apparatus

meterstick or tape measure
graph paper

Goal

To measure the coordinates of objects in the classroom.

Procedure

1. Define the origin of our coordinate system to be at the left corner of the front of the room (if you are facing toward the front), at the floor.
2. Define the $+x$ direction to be to the right, if facing the front of the room.
3. Define the $+y$ direction to be upward toward the ceiling.
4. Define the $+z$ direction to be along the side wall toward the back of the room.
5. Using a meterstick, determine the coordinates of the center of the seat of your chair.

Further Investigation

C What are the coordinates of the center of your seat?

B (Do C first.) Using graph paper, draw an x-z coordinate system with the origin in the top-left corner of the paper. (Note: it's ok to use your paper in landscape or portrait orientation depending on what is most convenient. You are sketching a top view of the room. Assume that the front wall with the whiteboard is at the top edge of the paper.) Draw the seat of your chair (as a circle) at the correct location on the coordinate system. Assume that the chair is underneath the table. Be sure to indicate the scale of your coordinate system by labeling the axes with both numbers and units. Sketch the boundaries of the room. Select the values of your gridlines so that the coordinate system takes up as much of the paper as possible.

A (Do B and C first.) Show the coordinates of every seat in the classroom. If you think carefully about it, you can make a few measurements and assumptions and then calculate and draw the positions of the chairs in the room. You do not have to measure every chair.

3 PROGRAM: Introduction to GlowScript and VPython

Apparatus

Computer
GlowScript – www.glowscript.org

Goal

The purpose of this activity is to write your first program in a language called Python. We will use the web app GlowScript that converts Python to JavaScript so the program can run in a web browser. GlowScript provides the same functions available in the Python module called Visual. Together Python and Visual are named VPython. As a result, GlowScript can be considered the web-based version of VPython. You will probably read or hear the terms GlowScript and VPython used interchangeably; however, there are some important differences. I tend to think of the language as VPython (Python + Visual) and the web app as GlowScript.

We use VPython because it allows you to do vector algebra and to create 3D objects in a 3D scene. The capability of 3D graphics with vector mathematics makes it a great tool for simulating physics phenomena. In this activity, you will learn:

- how to use GlowScript, the web-based integrated development editor (IDE) for writing and running VPython.
- how to structure a simple computer program in VPython.
- how to create 3D objects such as spheres and arrows.

Setup

Go to http://www.glowscript.org/ and create an account. You will need a Google account because GlowScript uses your Google account for authentication. After logging in, you will see a link to "your programs are here." Click this link to enter the IDE.

Procedure

Creating folders and files

1. Once you log in and follow the link to your programs, you are in the GlowScript IDE. Click the Add Folder tab to create a new folder. A pop-up window appears as shown in Figure 3.1. Because I must run your programs, make the folder public.
2. With the folder name highlighted orange (showing you are in the folder), click the link **Create New Program** and name the program `intro` .

Starting a program: Setup statements

3. Notice GlowScript types the first line of the program for you.

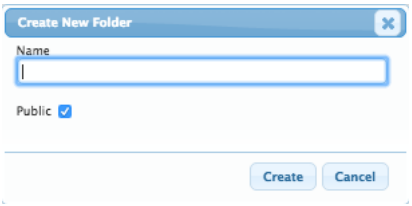

Figure 3.1: Create a new folder in GlowScript.

```
GlowScript 1.1 VPython
```

Every GlowScript program begins with this setup statement. It tells GlowScript you are writing VPython code.

4. Also, notice there is no "save" menu. Like Google Docs, GlowScript automatically saves your program as you are typing it.

Creating an object

5. Now for your first VPython command, let's make a sphere. Skip a line in order to make your code more readable, and on line 3, type:

```
sphere()
```

This statement tells the computer to create a sphere object.

6. Run the program by clicking **Run this program**. GlowScript exits the edit mode and enters the run mode. You should see a white sphere on a black background like Figure 3.2. This is called the `scene` .

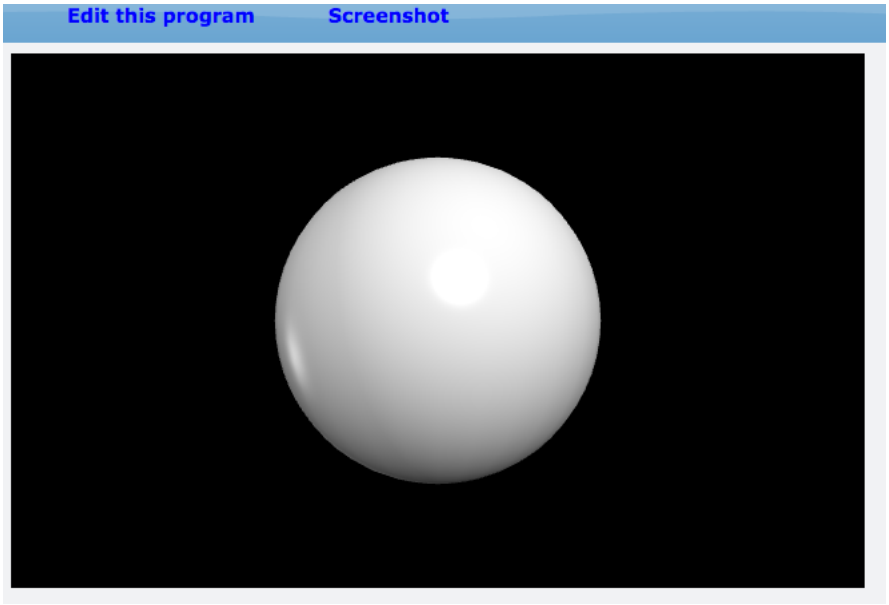

Figure 3.2: Your first VPython program–a sphere.

The 3-D graphics scene

By default the sphere is at the center of the scene, and the "camera" (your point of view) is looking directly at the center.

7. If you are on a PC, hold down both mouse buttons and move the mouse forward and backward to make the camera move closer or farther away from the center of the scene. On a Mac, hold down the option key while moving the mouse forward and backward. This is how you *zoom* in VPython.

8. Hold down the right mouse button alone and move the mouse to make the camera "revolve" around the scene, while always looking at the center. On a Mac, in order to rotate the view, hold down the Control key while you click and drag the mouse. This is how you *rotate* the scene in VPython.

   By default, when you first run the program, the coordinate system is defined with the positive x direction to the right, the positive y direction pointing up toward the top edge of the monitor, and the positive z direction coming out of the screen toward you. You can then rotate the camera view to make these axes point in other directions relative to the camera.

Error messages: Making and fixing an error

GlowScript tells you when there is a syntax error in your program. (Logic errors are much more difficult to fix!) To see an example of an error message, let's try making a spelling mistake.

9. Change line 3 of the program to the following:

```
phere()
```

10. Run the program.

    There is no function or object in VPython called phere(). As a result, an error message pops up. The message gives the *approximate* line number where the error occurred and a description of the error, as shown in Figure 3.3.

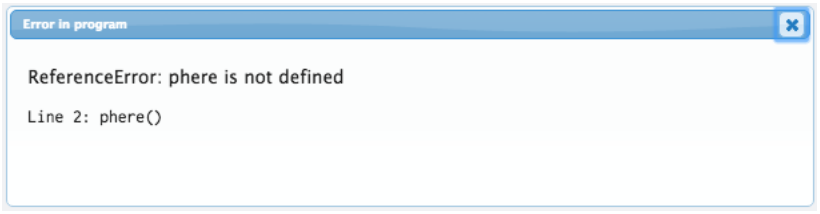

Figure 3.3: An error message in GlowScript.

The line number may be off, as it is in this case but is usually close.

11. Correct the error in the program by clicking **Edit this program** and returning to the editor. Once in editing mode, you can click the ☒ to close the error message.

    There are two types of errors: (1) syntax errors which might be a typing or coding mistake and (2) a programmatic errors so the program runs correctly but does something other than what you intended. The error message helps you find the first of these. Finding errors that cause a program to act differently than you intended is much more difficult and is a skill you will develop in this course.

Changing attributes (position, size, color, shape, etc.) of an object

Now let's give the sphere a different position in space and a radius.

12. Change line 3 of the program to the following:

```
sphere(pos=vector(-5,2,3), radius=0.40, color=color.red)
```

13. Run the program. Experiment with other changes to pos, radius, and color, running the program each time you change an attribute.

14. Answer the following questions:

What does changing the `pos` attribute of a sphere do? What does changing the `radius` attribute of a sphere do? What does changing the `color` attribute of a sphere do? What colors can you use? You can try `color=vector(1,0.5,0)` for example. The numbers stand for RGB (Red, Green, Blue) and can have values between 0 and 1. Can you make a purple sphere? Note that colors such as cyan, yellow, and magenta are defined, but not all possible colors are defined. Choose random numbers between 0 and 1 for the (Red, Green, Blue) and see what you get.

Autoscaling and units

VPython automatically zooms the camera in or out so all objects appear in the window. Because of this autoscaling, the numbers for the `pos` and `radius` can be in any consistent set of units, like meters, centimeters, inches, etc. For example, this could represent a sphere with a radius 0.20 m at the position $(2, 4, 0)$ m. In this course we will often use SI units in our programs ("Systeme International", the system of units based on meters, kilograms, and seconds).

Creating a box object

Another object we will often create is a box. A box is defined by its position, axis, length, width, and height as shown in Figure 3.4[1].

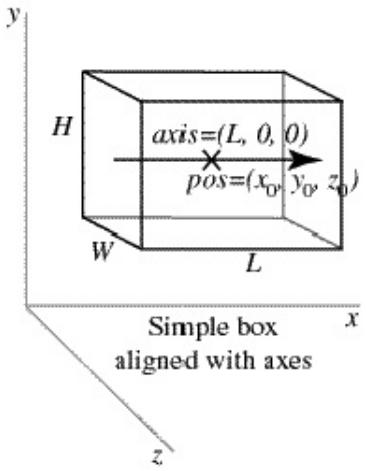

Figure 3.4: Attributes of a box.

15. Type the following on a new line, then run the program:

```
box(pos=vector(0,0,0), size=vector(2,1,0.5), color=color.orange)
```

The length, width, and height of the box are expressed as a vector with the attribute: `size=vector(L,H,W)` .

[1]Image from `http://www.glowscript.org/docs/VPythonDocs/box.html`

16. Change the length to 4 and rerun the program.

17. Now change its height and rerun the program.

18. Similarly change its width and position.

> Which dimension (length, width, or height) should be changed to make a box longer along the y-axis? Change your code now to check your answer.
>
> What point does the position of the box refer to?
>
> (a) the center of the box
>
> (b) one of its corners
>
> (c) the center of one of its faces
>
> (d) some other point

Comment lines (lines ignored by the computer)

Comment lines start with a # (pound sign). A comment line can be a note to yourself, such as:

```
# units are meters
```

Or a comment can be used to remove a line of code temporarily, without erasing it.

19. Put a # at the beginning of the line creating the box, as shown below.

```
#box(pos=vector(0,0,0), size=vector(2,1,0.5), color=color.orange)
```

20. Run the program. What did you observe?

21. Uncomment this line by deleting the # and run the program again. The box now appears.

Naming objects; Using object names and attributes

We will draw a tennis court and will change the position of a tennis ball.

22. Clean up your program so it contains only the following objects:

    A green box that represents a tennis court. Make it 78 ft long, 36 ft wide, and 4 ft tall. Place its center at the origin.

    An orange sphere (representing a tennis ball) at location $\langle -28, 5, 8 \rangle$ ft, with radius 1 ft. Of course a tennis ball is much smaller than this in real life, but we have to make it big enough to see it clearly in the scene. Sometimes we use unphysical sizes just to make the scene pretty.

    (Remember, you don't type the units into your program. But rather, you should use a consistent set of units and know what they are.)

23. Run your program and verify that it looks as expected. Use your mouse to rotate the scene so you can see the ball relative to the court. Your program should look like the one below.

```
1 GlowScript 1.1 VPython
2
3 box(pos=vector(0,0,0), size=vector(78,4,36), color=color.green)
4
5 sphere(pos=vector(-28, 5, 8), radius=1, color=color.orange)
```

24. Change the position of the tennis ball to $\langle 0, 6, 0 \rangle$ ft.

25. Run the program.

26. Sometimes we want to change the position of the ball after we defined it. Thus, give a name to the sphere by changing the `sphere` statement in the program to the following:

```
tennisball=sphere(pos=vector(0, 6, 0), radius=1, color=color.orange)
```

We've now given a name to the sphere. We can use this name later in the program to refer to the sphere. Furthermore, we can specifically refer to the attributes of the sphere by writing, for example, `tennisball.pos` to refer to the tennis ball's position attribute, or `tennisball.color` to refer to the tennis ball's color attribute. To see how this works, do the following exercise.

27. Start a new line at the end of your program (perhaps line 7) and type:

```
print(tennisball.pos)
```

28. Run the program.

29. Look at the text below the 3D scene. The printed vector should be the same as the tennis ball's position.

30. Add a new line to the end of your program (perhaps line 9) and type:

```
tennisball.pos=vector(32,7,-12)
```

When running the program, the ball is first drawn at the original position but is then drawn at the last position. (Note: whenever you set the position of the tennis ball to a new value in your program, the tennis ball will be drawn at that position.) This may happen so quickly that you do not notice the tennis ball drawn at the two locations.

31. Add a new line to the end of your program (perhaps line 11) and type:

```
print(tennisball.pos)
```

(Or just copy and paste your previous print statement.)

32. Run your program. It now draws the ball, prints its position, redraws the ball at a new position, and prints its position again. As a result, you should see the following two lines printed:

```
<0, 6, 0>
<32, 7, -12>
```

Of course, this happens faster than your eye can see it which is why printing the values is so useful.

Analysis

All games with graphics include objects on the screen. The game programmer must specify the positions and dimensions (sizes) of the objects using 2D or 3D vectors.

C Do all of the following. You are going to create objects for the game *Frogger*. We will only use spheres and boxes for this part.

1. Create a new blank file and name it *frogger-C*.
2. Create a green box for the frog that is at the location $< 0, -100, 0 >$, has a length=10, height=10, and width=10 units. Name the box `frog` .
3. Create a yellow sphere for a lily pad at $< -60, 100, 0 >$ with a radius of 10. Name the sphere `lilypad` .
4. Create a blue box for the water that is at the location $< 0, 0, -10 >$, has a length=150, height=220, and width=10 units. Name the box `water` .

5. Rotate the scene. Is the lily pad inside the water or on top of the water? Is the frog inside the water or on top of the water?
6. Is physics used in this program? Why does the frog not move in this program?

B Do everything for **C** along with the following modifications and additions.

1. Create a new blank file and name it *frogger-B*.
2. Copy from your previous program (labeled C) and paste it into this program. Often, this is the fastest way to start a new program.
3. Create another yellow sphere for a lily pad at $< 60, 100, 0 >$ with a radius of 10. Name the sphere `lilypad4` .
4. In between these two lily pads, create two more named `lilypad2` and `lilypad3` so the lily pads are equally spaced.
5. Create a gray road that is exactly half the height of the water. It should extend from the middle of the blue box to the bottom end of the blue box.
6. Create a long cyan box on the left side of the road and a short magenta box on the right side of the road, between the frog and the water. Name them `car1` and `car2` .
7. Print the positions of the cars and the frog.

Figure 3.5 is an example program that fits the criteria for a B.

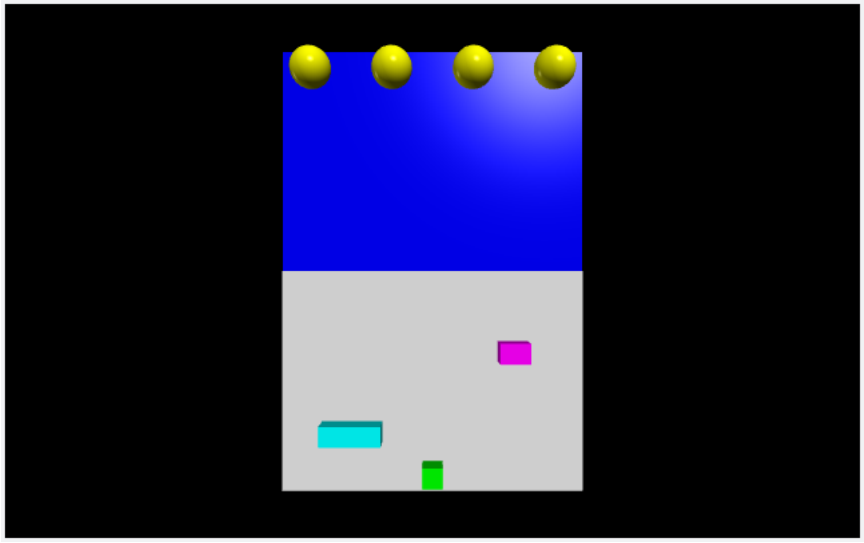

Figure 3.5: The scene required for a B.

A Do everything for **B** along with the following modifications and additions.

1. Create a new blank file and name it *frogger-A*.
2. Copy from your previous program (labeled B) and paste it into this program.
3. In the top right corner of the GlowScript window, click the link to **Help**. This opens the documentation window. Click the menu to **Choose a 3D object** and view the list of objects shown in Figure 3.5.
4. Select the cylinder and read how to create a cylinder.
5. Change the lily pads so they are thin cylinders that appear to float on top of the water.
6. Use the cylinder object to create 3 logs of different lengths in the water.

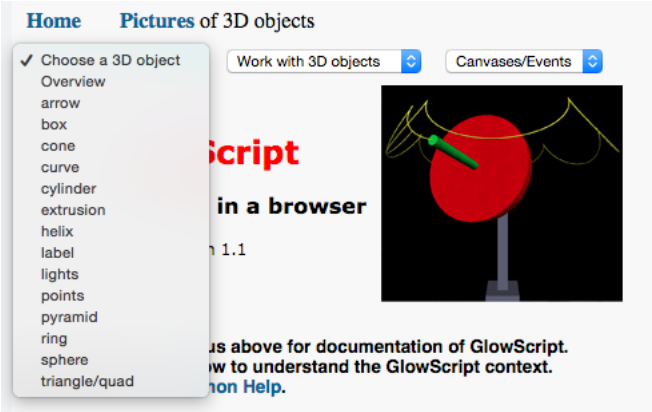

Figure 3.6: The scene required for a B.

7. Now, click the menu to **Work with 3D objects** in the documentation and select **Textures**. Read how to specify a texture. You will probably want to click the link to the example program that demonstrates the pre-defined textures.
8. Change the three wooden logs so they use the wood texture.

Figure 3.7 is an example program that fits the criteria for a B.

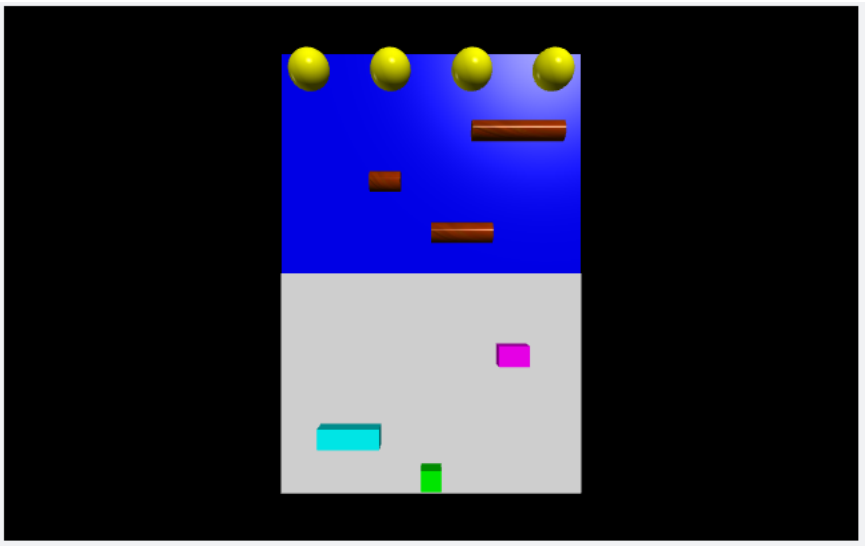

Figure 3.7: The scene required for an A.