Inf-1100, Assignment 5

Submission: 23:59, November 18th, 2022

Description

Your task is to implement a variant of the classic bouncing ball animation.

Each ball should start off at the top of the screen with a random speed in the x direction. Whenever a ball hits a screen boundary it should bounce off at an angle equal to the impact angle and lose some speed. Eventually each ball should come to a rest at the bottom of the screen. Five seconds after coming to a rest a ball should be removed from the screen. To get an idea what a solution might look like, see the short video at https://youtu.be/x04-uMLQHQA

To ensure that the balls come to a rest at the bottom of the screen you need a constant gravity pulling on each ball. The direction of the gravity should be down.

Each ball must be represented as a separate *object* data structure (see "object.h" in the zip file). The object data structure contains all variables pertinent to rendering the object on the screen (translation, scale, model coordinates, etc.). Note that the object data structure uses floats to represent translation coordinates. This is to make it easier to handle very small movements (at points, a ball might be moving at a speed less than a pixel). Cast the translation coordinates into integers before drawing the triangles on the screen.

Your code must keep track of objects (balls) by placing the object data structures in a linked list. You need to create your own linked list implementation. Below is a brief description of the object programming interface:

- **CreateObject** - Create a new object. The function accepts as input parameters a pointer to the SDL screen, a pointer to a model triangle array, and a variable telling the size of the model triangle array. The function returns a pointer to a new object data structure. The model triangle array specified as input parameter should **not** be shared across objects. (Not sharing the model triangle array allows e.g. objects to have different colors.) Perform the necessary memory allocation and copying.
- **DestroyObject** - Free object. The function accepts as input parameters a pointer to an object data structure. The function should free all memory allocated to represent the object (memory allocated for the model triangle array and the object data structure itself).
- **Drawobject** - Draw object on screen. The function accepts as input parameters a pointer to an object data structure. The function must draw the object on the screen by calling DrawTriangle on each of the model triangles. Remember to update scale, translation, etc., in each triangle data structure before invoking DrawTriangle.

Hint: Do not make the bouncing algorithm too complex. Bouncing a ball off a vertical or horizontal surface can be accomplished without resorting to calculating impact angles.

Code

Your starting point is the following set of files:

- drawline.h - Specifies the interface of the drawline function.
- drawline.c - Implements the DrawLine function.
- triangle.h- Specifies the triangle data structure and the interface to the DrawTriangle function.
- triangle.c- Draw triangle and friends.
- teapot_data.h- Coordinates for the classic teapot model.
- sphere_data.h- Coordinates for a sphere model.
- list.h- Specifies the interface of the list functionality. Do not modify this file
- list.c- Empty stubs for each function in the list interface. *Modify this file*.
- object.h - Specifies the interface of the object functionality. Do not modify this file
- object.c - Empty stubs for each function in the object interface. *Modify this file*.
- main.c- Contains the main function and calls to initialize SDL. *Place your bouncing ball code in this file*.
- Makefile - A Makefile for compiling the code.

We've bundled these files in a zip file. Use this zip file as a starting point, as it also includes the necessary SDL files.

Resources

On the Web, there is a wealth of information on how to make a bouncing ball animation. Do a Web search.

To aid you in implementing and debugging your list code, we have bundled a test program that can be used to check your list implementation for errors. Run “make test” to compile the test program, and run it by typing “./testlist”. Note that the test program might give you an OK on functions that are empty.

A tip: don’t try to solve the animation part and the list part at the same time. A possible approach might be to first make sure that you’re able to bounce a single ball that’s not in a list. Then move on to implementing the list, making sure that it passes the tests; then finally populate the list with several balls and see that they all bounce around.

Submission guidelines

For this assignment, we expect two hand-ins:

- A report describing how your C code fulfills the requirements of the assignment text (as printed text, in the PDF format).
- A compressed archive containing all the source-code files.

Do not refer to the C code as self-explanatory. We encourage use of figures to illustrate data structures and pseudo-code to illustrate algorithms.

In particular, your documentation should at least describe the following:

- Your implementation of the object interface.

- Your list implementation.
- How you have implemented animation.
- The algorithm for bouncing balls off screen edges.
- The algorithm for detecting when a ball has come to a rest and how you time removal of the ball from the screen.

Topics that could be discussed in the technical background part of the documentation:

- Dynamic memory allocation.
- Lists vs arrays.

Submission

You should submit the report ('assignment5.pdf') and code (as below) before the deadline above. The submission should be done using Canvas.

You should submit two files, the report and the C source code. We only accept the report as a single file in the PDF format (not scanned). The C source code should be packed in a compressed archive, such as 'zip', 'rar' or 'tar.gz', in the following way:

- The archive contains a folder with your username and the assignment number as its name, eg. eho033-assignment3/. This folder contains your other files.
- The archive itself has the same name, eg. eho033-assignment3.zip

We ask you to do this so that the people looking at your submissions can keep track of which files belong to which student, which otherwise is a lot of extra work for them. Alternate formats will not be accepted.