Computer Science II — Homework 2 — Rectangle Graphics.

This assignment is due Thursday, January 31 at 11:59:59pm and is worth 65 points toward your homework grade.

In computer graphics, when generating an image of a virtual world containing a series of objects (plants, walls, doors, trees, furniture, actors, etc.), a fundamental problem is to determine which objects are visible from any particular camera location (and orientation) and which object is "seen by" (generates) any particular pixel in the camera. Computer scientists and engineers have created a series of sophisticated algorithms, data structures, and hardware implementations to address this problem. This homework assignment will investigate a toy version of the problem.

Our objects will just be rectangles parallel to the xy-plane and one depth (z) unit thick. The camera will always be pointed along the positive `z` axis. The visual effect is one of a set of rectangular cardboard cut-outs parallel to the image.

There will be four different inputs to the program —- one to specify an object, one to move an existing object, one to find the object along a particular line of sight (a simple form of the "ray tracing" that is the core operation of many high-end graphics engines), and one is to check if the objects at their current positions are inconsistent — in other words if they intersect each other. These are some of the core operations needed for image creation; we will not be creating actual images.

Here is the format of the input, with details to follow.

- add id x y z w h
- move id x y zx
- ray x y z
- inconsistent

Using a Euclidean coordinate system, `x`, `y` and `z` specify the center of an object (or a camera location), and `w` and `h` specify an object's width (in the x direction) and height (in the y direction). There will be no errors in the input format.

Here is a detailed description of the commands:

- The `add` command says to add an object with the given id, placing it at the given position, and specifying its width and height. The id is just a string. If an object with the same id has already been added, output an error message and do not add the new object.
- The `move` command indicates that the object with the given id should be moved to the given location. If an object does not exist with the given id your program must output an error message. You are not required to check if another object is already present at the new location or if the movement will result in two objects overlapping.
- The `ray` command should find and output the id of the first object intersected by a ray that starts at the given `x`, `y` and `z` location and proceeds along the positive `z` axis. Stated another way, the id output should be from the object with the smallest depth (the z value of the object position, ignoring the 1 unit thickness) greater than `z` such that `x` and `y` are contained in its rectangle. If no objects are reached along the ray,

the program should output `<none>`. If two (or more) objects are reached at the same time (same z value), then your program should just output the first one it finds. If the ray does not pass *through* an object, it does not intersect it.

- The `inconsistent` command should check to see if any pair of objects overlaps. When checking two objects you need to check if they overlap in z (remember, objects are one unit thick) and overlap in their x-y rectangles. If two objects do overlap they are inconsistent. Output each pair of inconsistent objects on a separate line of output. The output order should be in increasing order of depth for the object with the smaller z. If this object intersects multiple objects, the order of these objects should be the order of their depths. Break ties in any way you choose (or don't even bother to worry about it). If no objects are inconsistent, simply output `<none>`. Two objects that simply touch each other are not inconsistent.

Input / Output

The command input will come from a single file and the output will be to a second file, both specified on the command-line. The program will end when the input ends. The output should include an echoing of the input commands so that the results are easy for the graders (and for you) to read. Output a single blank line after the output in response to a command. The format of the output should follow the example on the web page as closely as possible.

Discussion

The heart of the program is the definition of an `Object` class. Start by specifying the necessary operations for the class and turn these into specifications for member functions. Be sure you are clear about what information each `Object` must store — this defines the member variables. Then you will be ready to implement and test the class. Once you have done this, you will be able to write the implementations of the member functions. One of your logic challenges is reliably determining when two rectangles intersect.

Once you have the `Object` class completed, you should be able implement the main function and any additional functions to handle the input and process the objects.

Your code should be split across multiple files, with a header (.h) file and an implementation (.cpp) file for each class. These files should be zipped together with a `readme.txt` file prior to submission. See the on-line instructions on the course web page for futher details.

readme.txt

Please address the questions in the `readme.txt` file posted on the course web site and submit it with your source code.