ECS40 Spring 2018 2018-05-12

Homework 3 due Thu 2018-05-24 at 23:59

Use the handin directory hw3 to submit your work

Packing circles and rectangles

Description

In this assignment, you are asked to implement a program (named **`checkpack`**) that, given a rectangular domain and a list of circles and rectangles, can check if the packing configuration is valid, in the sense that all shapes fit within the domain and there are no overlaps between shapes. This problem occurs e.g. in electronic device design when packing electronic components on a printed circuit board.

The program should create a visual representation of the domain and shapes as a scalable vector graphics (svg) file. An svg file can be visualized using a browser such as Firefox, Safari, Chrome or Internet Explorer. See https://en.wikipedia.org/wiki/Scalable_Vector_Graphics .

The positions and sizes of the shapes are specified using integers. The domain is a rectangle of fixed size (600x500). Circles of arbitrary position and radius are read from standard input. Rectangles of arbitrary position and sizes are read from input. Circles and Rectangles can appear in any order in the input.

The program should write on standard output the svg representation of all the shapes, and include in addition the following diagnostic message:
- **`ok`** if the configuration is valid, i.e. all shapes fit in the domain and there are no overlaps between shapes.
- **`overlap`** if the shapes fit in the domain but some of the shapes overlap
- **`does not fit`** if any of the shapes does not fit in the domain.

The diagnostic message should appear when rendering the svg file as in Figure 1.

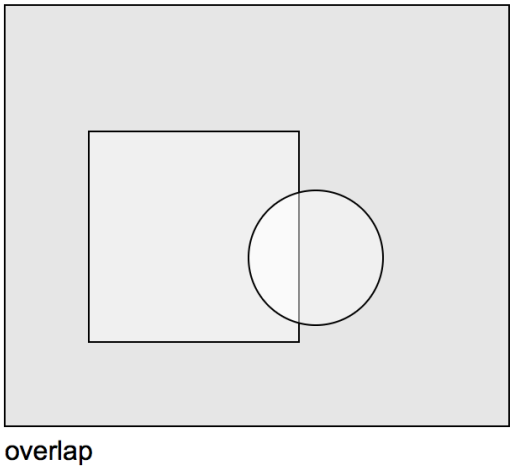

Figure 1: Example of rendering of an svg file.

HW3 Implementation

The implementation relies on the classes **`Domain`**, **`Point`** and **`Shape`**. The header files **`Domain.h Point.h Shape.h`** are provided and must not be modified. The programs **`testPoint.cpp`** , **`testShape.cpp`** and **`checkpack.cpp`** are provided and must not be modified. All source files should compile without warning. The executables should build without warning. You must implement the classes **`Domain`** (file **`Domain.cpp`**) **`Point`** (file **`Point.cpp`**) and the classes **`Circle`** and **`Rectangle`** (file **`Shape.cpp`**).
You must provide a **`Makefile`** that builds all executables by using the command

```
$ make
```

Input

Input for the **`checkpack`** program is read from standard input. Each line of input defines a shape to be added to the domain. The first character on each line determines whether the shape is a circle ('C') or a rectangle ('R'). The position and size of the shape is described by integers in the rest of the input line.
A rectangle is specified by the position of its lower left corner, its width and its height.
Example: Rectangle with corners at (10,20), (40,20), (40,60), (10,60)

```
R 10 20 30 40
```

A circle is specified by the position of its center and its radius
Example: Circle centered at position (30,40) with radius 20

```
C 30 40 20
```

It can be assumed that input will consist of valid characters (only 'R' or 'C') and the appropriate number of positive integers.

The **`checkpack`** program should be used as follows:

```
$ ./checkpack < test.in > test.svg
```

It must be possible to visualize the file **`test.svg`** using a browser (Firefox, Chrome, etc.)

The program **`checkpack`** should reproduce exactly the example svg files provided when using the provided input files. Use the Unix **`diff`** function to check for differences.

Point class

The **`Point`** class represents a point in the x-y plane, using integer coordinates x and y. **`Point`** objects can be added and subtracted, and the square of the norm of a point can be calculated (the square of the norm is computed rather than the norm, in order to perform all operations with integer arithmetic, and avoid the use of a square root function).
`operator>>` should read two **`int`**s from an input stream to set the values of members **`x`** and **`y`**.
`operator<<` should print the point to an output stream in the following format (for example if x=3 and y=4): **`(3,4)`**

Domain class

The **`Domain`** class defines the rectangular region of fixed size (600x500) in which the shapes are packed.
`void addShape(const Shape* p)` is a function used to add a **`Shape`** pointer to the vector of **`Shape`** pointers **`s`**.

`void draw(void)` is a function that generates the entire svg output. It first checks whether all shapes fit into the **`Domain`** (hint: this can be done by defining a **`Rectangle`** object representing the domain and using the Shapes' **`fits_in`** functions). It checks whether there are overlaps between shapes (using the Shapes' **`overlaps`** functions). On the basis of these tests, it determines the diagnostic message ("**`ok`**", "**`overlap`**" or "**`does not fit`**"). It then prints the svg header, the svg representations of all shapes (using the Shapes' **`draw()`** functions), the diagnostic message, and the svg trailer on standard output. See the example test output svg files for details about the svg output format and the contents of the svg header and trailer.

Shape class

The **`Shape`** class is an abstract base class from which **`Rectangle`** and **`Circle`** are derived.
`bool fits_in(const Rectangle& r)` is a pure virtual function that should return **`true`** if the **`Shape`** fits in the **`Rectangle r`**.
`void draw(void)` is a pure virtual function that writes the svg description of the **`Shape`** on standard output.

Rectangle class

The **`Rectangle`** class is derived from **`Shape`** and represents a rectangle. The position of the Rectangle is its lower left corner.
`void draw(void)` is a virtual function that writes the svg description of the **`Rectangle`** on standard output, e.g.

```
<rect x="50" y="70" width="25" height="80"/>
```

Circle class

The **`Circle`** class is derived from **`Shape`** and describes a circle. The position of the Circle is its center.
`void draw(void)` is a virtual function that writes the svg description of the **`Circle`** on standard output, e.g.

```
<circle cx="10" cy="30" r="50"/>
```

Create a tar file **`hw3.tar`** containing all source files and the **`Makefile`**. Submit your tar file using **`$ handin cs40 hw3 hw3.tar`**

Notes

1) This problem is an example of use of the double dispatch technique described in Lecture 11.

2) When implementing the various **`overlaps`** virtual functions, use the fact that the **`Circle::overlaps(const Rectangle& r)`** function does the same thing as the **`Rectangle::overlaps(const Circle& c)`** function. You only need to implement one of the two functions, and you can have the second function call the first one.

3) To determine whether a Circle of radius r centered at (x_c, y_c) overlaps with a Rectangle, use the following algorithm:
a) Compute the coordinates (x_n, y_n) of the point of the Rectangle that is closest to the Circle center (x_c, y_c) (note: this point is not necessarily a corner of the Rectangle). Use the formulas

$$x_n = \min\left(\max(x_c, x_r), x_r + w\right)$$
$$y_n = \min\left(\max(y_c, y_r), y_r + h\right)$$

where (x_r, y_r) is the position of the lower left corner of the Rectangle, and w and h are the Rectangle's width and height.

b) Compute the distance d between the point (x_n, y_n) and the Circle center (x_c, y_c). If $d < r$, there is overlap between the Circle and the Rectangle. When testing if $d < r$, use the equivalent condition $d^2 < r^2$ which involves only integer arithmetic, and avoids the use of the square root function.

4) Rectangles that share an edge do not overlap. For example, the rectangle defined by the points (10,20), (40,20), (40,60), (10,60) does not overlap with the rectangle defined by the points (40,30), (60,30), (60,70), (40,70). Likewise, rectangles can share edges with the domain boundary.

5) This assignment requires you to write a program that verifies that a given packing of circles and rectangles is valid. The problem of finding an optimal packing of a given set of circles and rectangles is much more complex, and falls into the category of "NP complete" problems. See e.g. https://en.wikipedia.org/wiki/Packing_problems