**Course Real-Time Rendering**
**Winter term 2019/20**

**Lab Class** **November 11 and 12, 2019**

# Collection of Exercises 2

**Solutions due to: December 04, 2019**

In this second lab class you will implement an algorithm that computes the convex hull of a given set of 2D points. Though we recommend to use C++, it is also possible to use JAVA as programming language. We suggest Visual Studio Community C++ 2019 (for Windows, it is free), Eclipse (Windows/-Linux/Mac), Sublime (Windows/Linux/Mac) or build tools like *make*, *cmake*, *javac* as programming environment.

## Submission and Grading

The exercise can be solved as a group of one or two students. We encourage you to thoroughly solve the exercise on your own. Plagiarism can lead to an exclusion from the course. If you have any problems or questions don't hesitate to ask your professor Rhadamés Carmona (recs34@gmail.com, Bauhausstr. 9a, third floor, guest room).

The solution for this exercise have to be submitted until **December 04, 2019, 11:59 pm** to our **moodle** website. The solutions will be evaluated by your professor using own test cases. The total number of points granted will be proportional to the number of correct answers to problems instances. The grading will be announced on moodle.

For grading, your implementations will be tested on the following online compilers:

- C++ will be tested on: http://cpp.sh/
- JAVA will be tested on: https://www.jdoodle.com/online-java-compiler/

For submission, you have to implement your solutions as single .cpp or .java files (i.e. `exercise21.cpp` and `exercise22.cpp` or `exercise21.java` and `exercise22.java`). The first line(s) must include your name(s) as comment in order to document authorship(s).

As an example, if you are a group of two students and submit in C++, a submission file should be structured as follows:

```
// author: <forename> <family name>
// author: <forename> <family name>

#include <iostream>

// function definitions can be placed here

int main(){
/*
your implementation starts here
*/
return 0;
}
```

As another example, if you are a single student and submit in JAVA, a submission file should be structured as follows:

```
// author: <forename> <family name>

public class MyClass {
    public static void main(String args[]) {
/*
your implementation starts here
*/
    }
}
```

**We strongly recommend to test your code on these online compilers before submission.** In particular, the output format of your implementations should be exactly as it is specified in the exercise. For grading, the output will be compared line by line with an online software (e.g. https://text-compare.com/). In order to obtain full grade, the output generated by your implementation has to be identical with the expected output specified in the exercise.

## Exercise 2.1

**[10 points]**

**Convex Hull - Quick-Hull**

Implement a program that computes the convex hull for a set of $n$ 2D points using the Quick-hull algorithm.

**Input** The input consists of one problem instance. It starts with a text line with an integer number $n$ indicating the size of input (number of points), with $3 <= n <= 1000$. It is followed by $n$ lines containing the integer coordinates ($x$ and $y$) of each point, with $-350 <= x, y <= 350$.

**Output** The output contains a single line indicating the number of points ($m$) of the convex hull, followed by $m$ lines with the points of the convex hull in CW order (one line per point). Each point has to be printed in a single line containing the integer coordinates $x$ and $y$ separated by one blank space. The first printed point should be the bottom-left most point of the convex hull. The convex hull printed must have minimal size. Thus, three or more collinear points are not allowed in the output.

**Sample input**

```
5 # set of 5 points
3 3
1 1
0 1
-1 -1
0 2
```

**Sample output**

```
3 # convex hull contains 3 points
-1 -1
0 2
3 3
```

## Exercise 2.2

**[10 points]**

**Onion layers**

Dr. Kabal, a well recognized biologist, has recently discovered a liquid that is capable of curing the most advanced diseases. The liquid is extracted from a very rare onion that can be found in a country called Onionland. But not all onions of Onionland are worth to take to the lab for processing. Only those onions with an odd number of layers contain the miraculous liquid. Quite an odd discovery!

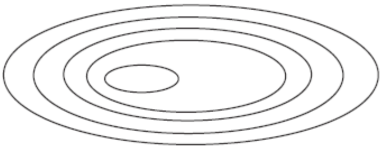

Figure 1: Onion from Onionland.

Dr. Kabal has hired some research assistants to collect and analyse onions for him. Since he does not want to share his discovery with the world yet, he did not tell the assistants to look for onions with special properties. Instead, each assistant has the task of collecting onions, and selecting points from each of the layer's outer borders, so that an approximation of the layer structure of the onion can be reconstructed later. Dr. Kabal told the assistants that the next step will be a ”complicated analysis” of these points. In fact, all he will do is simply to use the points to count the number of layers in each of the onions, and select the ones with an odd number of layers.

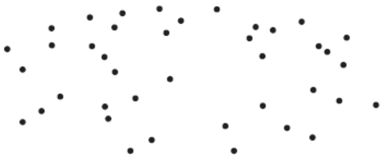

Figure 2: Extracted points.

It is clear that the approximation obtained by Dr. Kabal, from the points collected, might have a different shape than the original onion. For instance, only some of the points of the onion shown in Figure 1 would be extracted in the process, giving rise to a set of points as shown in Figure 2.

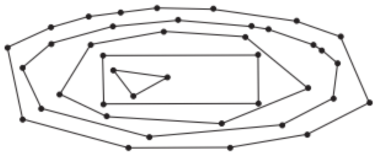

Figure 3: Approximation of the original onion layers.

With these points Dr. Kabal will try to approximate the original layers of the onion, obtaining something like what is shown in Figure 3. The approximation procedure followed by Dr. Kabal (whose result is shown in Figure 3) is simply to recursively find nested convex polygons such that at the end every point belongs to precisely one of the polygons. The assistants have been obliged to select points in such a way that the number of layers in the approximation, if done in this recursive manner, will be the same as in the original onion, so that is fine with Dr. Kabal. The assistants are also aware that they need at least three points to approximate a layer, even the innermost one. Thus, a single point or a single line of 2 or more collinear points is not considered a layer.

**Input** The input consist of one problem instance. It starts with a text line containing an integer number $n$ indicating the number of input points, with $3 <= n <= 1000$. It is followed by $n$ lines containing the integer coordinates ($x$ and $y$) of each point, with $-350 <= x, y <= 350$.

**Output** The output contains all onion layers found, starting with the outermost layer. Each layer (convex hull) starts with an integer number indicating the number of points ($m_i$) of layer$_i$. The points of each layer have to be printed in CW order (one line per point), starting with the bottom-left most point. Print each layer with minimal size. Thus, three or more collinear points are not allowed in the output.

## Sample input

```
11 # set of 11 points
2 2
3 0
0 1
1 3
2 1
0 4
4 4
```

```
3 3
4 2
2 3
2 4
```

**Sample output (2 onion layers found)**

```
5 # 1st onion layer (convex hull) contains 5 points
3 0
0 1
0 4
4 4
4 2
3 # 2nd onion layer (convex hull) contains 3 points
2 1
1 3
3 3
```

## Useful hints

- Consider degenerated cases. Several (or all in the worst case) input points may be identical or collinear.
- Since the input points have integer coordinates with a fixed range, it is possible to determine if three points are collinear without arithmetic errors.
- We provide a small OpenGL framework (for Windows and Linux) that allows you to visually verifiy the output (a single convex hull or multiple onion layers) generated by your implementation, for any input- and output-file that fullfills the specified format (cmp. Figure 4 and 5). Please download a zip file from moodle.
- If you implement Graham's algorithm, we recommend sorting the input points only one time.
- Keep track of the remaining points after removing each layer.

## Additional Information

A good online reference for C++ is:

- http://www.cplusplus.com/

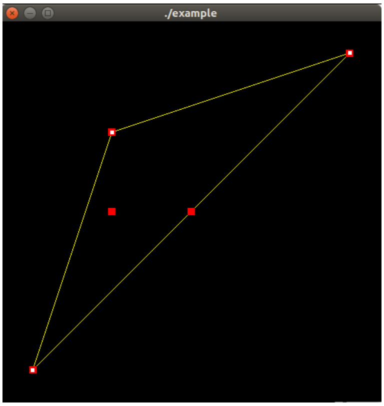

Figure 4: OpenGL-output for exercise 2.1: Input points are rendered in red color, the convex hull is rendered in yellow color, points belonging to the convex hull are marked in white color. Note that collinear points are not marked in white color.

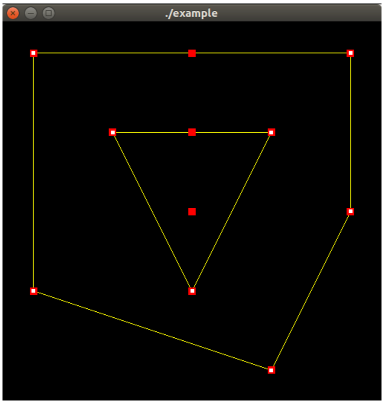

Figure 5: OpenGL-output for exercise 2.2: Input points are rendered in red color, the onion rings are rendered in yellow color, points belonging to an onion ring are marked in white color. Note that collinear points are not marked in white color.