Course Real-Time Rendering
Winter term 2019/20

Lab Class **October 28 and 29, 2019**

Collection of Exercises 1

Solutions due to: November 13, 2019

In this first lab class you will implement two algorithms that operate on geometric data. 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 exercises can be solved as a group of one or two students. We encourage you to thoroughly solve the exercises 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 solutions for this exercise have to be submitted until **November 13, 2019, 11:59 pm** to our **moodle** website, which will be announced in the next lab class. The solutions will be evaluated by your professor using own test cases. The total number of points granted per exercise 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. `exercise11.cpp` and `exercise12.cpp` or `exercise11.java` and `exercise12.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 1.1

[11 points]

Intersection of two line segments

Input The input consists of one ore more problem instances. Each problem instance is a single text line containing eight integer numbers: x_0 y_0 x_1 y_1 x_2 y_2 x_3 y_3. The input values are separated by one blank space and have to be in the range $[0, 10]$. The points (x_0, y_0) and (x_1, y_1) correspond to the first segment, (x_2, y_2) and (x_3, y_3) correspond to the second segment. The end of the input is defined by $x_0 = y_0 = x_1 = y_1 = x_2 = y_2 = x_3 = y_3 = 0$.

Output For each problem instance, print one text line with the following specification:

1. If the line segments do not intersect, print `no intersection`.
2. If the line segments intersect in just 1 point, print a single line with the x and y coordinates of the intersection point, rounded to 2 decimal digits and separated by one blank space. Example: 5.10 -6.32
3. If the line segments intersect in one segment, print `segment intersection`

Sample input (four problem instances)

```
0 0 0 1 1 0 1 1
0 0 2 2 2 0 0 2
5 5 5 5 0 0 5 5
5 6 8 9 8 6 5 9
0 0 0 0 0 0 0 0
```

Sample output (result of four problem instances)

```
no intersection
1.00 1.00
5.00 5.00
6.50 7.50
```

Useful hints

- ▶ Consider degenerated cases.
- ▶ Avoid division by zero.
- ▶ We recommend to implement and use the orientation functions (see lecture slides), before calculating the exact intersection point.
- ▶ Although the input coordinates are integer, the coordinates have to be stored as floating point variables with double precision.

- ▶ After printing every line, flush the output buffer (e.g. `fflush(stdout)` if you are using C, `cout.flush()` if you are using C++, and `System.out.flush()` if you are using Java). Thus, if the program crashes after several test cases, we may still grade the incomplete output.

Exercise 1.2

[9 points]

Point in Triangle

Given three 2D points (triangle vertices), rasterize the triangle in a virtual screen of 21×21 pixels. Rasterization is the process of converting graphics primitives (triangles in this case) into pixels.

Input The input consists in several problem instances. Each problem instance is a single triangle, given by a text line containing three 2D points (x_0, y_0), (x_1, y_1), (x_2, y_2), with integer coordinates $0 <= x_i <= 20$ and $0 <= y_i <= 20$. Thus, the virtual screen is a 2D array of 21×21 pixels. The end of the input is defined by $x_0 = y_0 = x_1 = y_1 = x_2 = y_2 = 0$.

Output For each problem instance, you have to print 21 lines containing 21 ascii characters each. To avoid ambiguity, the first printed line represents the pixels of the virtual screen with $y = 20$ (upper screen pixels), and the last printed line represents the lower pixels of the virtual screen with $y = 0$ (lower screen pixels). The character associated to pixel (x, y) is **’*’** if (x, y) is inside (or on the edge of) the triangle; otherwise, the pixel is ’.’. Print an empty line after these 21 lines to separate the output between consecutive test cases.

Sample input (two problem instances)

```
1 1 1 10 10 1
5 5 7 5 6 6
0 0 0 0 0 0
```

Sample output (two rasterized triangles)

```
.....................
.....................
.....................
.....................
.....................
.....................
```

```
.....................
.....................
.....................
.....................
.*...................
.**..................
.***.................
.****................
.*****...............
.******..............
.*******.............
.********............
.*********...........
.**********..........
.....................

.....................
.....................
.....................
.....................
.....................
.....................
.....................
.....................
.....................
.....................
.....................
.....................
.....................
.....................
......*..............
.....***.............
.....................
.....................
.....................
.....................
.....................
```

Useful hints

- For simplicity, check for every pixel (x, y) of the virtual screen, if that pixel is inside (or on the edge of) the triangle.

- In this exercise, use the implicit line equation (instead of barycentric coordinates) to determine if one point is inside (or on the edge of) the triangle. Since the triangle points have integer coordinates, use the version of the implicit equation which has no divisions. Thus, there will not be arithmetic errors inside the discrete screen space of 21×21 pixels.
- The coordinates of the bottom left pixel are $(0, 0)$, and $(20, 20)$ for the top right pixel.

Additional Information

Please refer also to the literature presented in the course.

- Computational Geometry, Algorithms and Applications (3rd edition), Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars, Springer 2008

A good online reference for C++ is:

- http://www.cplusplus.com/