Geometric Algorithms and Spatial Data Structures

Final project 2020/2021 – Trapezoidal maps

Introduction

Your task is to implement the incremental construction of the **trapezoidal map** and the associated DAG, in order to perform the point location task. The trapezoidal map must be drawn in the canvas, and it is possible to add interactively (mouse left click) new segments in the canvas (the UI has been already implemented). When you perform a query operation, the trapezoid in which the point is located must be highlighted.

The algorithm must be implemented in C++. We give you a Base Project in which you can find some basic features that are useful to successfully develop your project.

If you find a bug on the code, or if you have questions, please write an email at stefano.nuvoli@gmail.com and g.cherchi@unica.it, or open a discussion on the forum of the course.

Specifications

In order to get the maximum grade, you necessarily have to implement on your own:

- The data structure **DAG**;
- The T**rapezoidalMap data structure** that can perform point location (and the incremental construction of course);
- The **Drawable TrapezoidalMap** (use inheritance!)**.**

Your implementation must follow the **time and memory complexity** of the algorithm. It will be evaluated in the "Efficiency" part of the grade.

Since the segments are already drawn, for the drawable trapezoidal map you can draw only:

- The trapezoidal map trapezoids (the polygons) filled with different colors. Highlight (in some way, for example with a brighter color) the point location output trapezoid after the query.

- The trapezoid vertical lines

If you want to draw the trapezoidal maps points and segments (for debug), it is okay. Just delete the DrawableTrapezoidalMap from the objects of the canvas.

Read comments in the base project files.

Note that the draw() method in every drawable object is called 60 times per second, so execute light code in that method. Let's say it should work in maximum O(n) where n is the number of segments/trapezoids/vertical lines. Pre-compute the data if needed.

When you start the project, I suggest you open the file "trapezoidalmap_manager.cpp". Here we implemented all the user interface functionalities. In a MVC context, we provide VIEW and CONTROLLER. You should implement the MODEL and link the CONTROLLER to it.

All the user interface components are already implemented, you have just to write a few lines of codes in the manager in order to call the data structure methods or algorithm functions **that you WILL implement in separated files, organized in proper folders.** Use the **Object Oriented Paradigm** and try to **keep data structures and algorithms separate and general purpose** (it will be evaluated).

We wrote all the **instructions** (maybe too many!) **in the file, in the form of comments**. You need to fill the following methods with your code:

```
void TrapezoidalMapManager::addSegmentToTrapezoidalMap(const
cg3::Segment2d& segment);
```
You have to perform the incremental step that add a segment to the trapezoidal map.

```
void TrapezoidalMapManager::queryTrapezoidalMap(const cg3::Point2d& point);
```
You have to perform the point location query operation for the given point.

```
void TrapezoidalMapManager::clearTrapezoidalMap();
```
You have to clear all your data structures.

Also you will probably have to initialize your data structures in the **constructor (**`TrapezoidalMapManager::TrapezoidalMapManager()`**)**, and delete them in the **destructor (**`TrapezoidalMapManager::~TrapezoidalMapManager()`**)**. In the destructor, if you have properly written C++ code and followed the given guidelines, you do not actually need to write any code to deallocate elements. READ THE COMMENTS!).

Where you are allowed to write code, you have a separator and a comment like this:

```
//------------------------------------------------------------------
//Comment (instructions, tips, ...)


//##################################################################
```

Do not write code elsewhere (avoid it when it is possible, and it is actually possible), especially in the member functions with the comment **"Do not write code here"**.

We created for you a dataset container (**TrapezoidalMapDataset**) and its drawable version (**DrawableTrapezoidalMapDataset**), that can only contain a set of segments with the properties we need to execute the requested algorithms:

- **non-degenerate** (different endpoints) and **unique** segments;
- each segment is **non-intersecting** with any other in the dataset;
- points are in **general position** (none of them has the same x-coordinate) and **within the bounding box**.

If you try to add (interactively or using the load from file button) invalid points, you will get a message that they will be ignored (no segment is added).

All the segments lie inside the bounding box declared in the TrapezoidalMapManager. Note that the test datasets and the random generated datasets contain only segments inside the given bounding box. **Do not change the values of the bounding box in the define of the manager header file.**

Please if you find any case in which the random generation or the dataset produce segments not satisfying the above properties, write an e-mail to stefano.nuvoli@gmail.com or open a discussion in the forum.

THINK ABOUT YOUR STRUCTURE BEFORE WRITING CODE! PLANNING IS ESSENTIAL!

The trapezoidal map **is not** a DAG (it **uses** a DAG as an **associated** search data structure), the DAG **is not** a trapezoidal map (and **does not contain** a trapezoidal map). TrapezoidalMap and DAG are two separate data structures.

As example, we show two valid modeling options:

1. TrapezoidalMap could handle the construction and query in its inner methods, so it would contain the DAG and use its methods to perform the operations. It is easy to implement, but a bit less modular and general purpose. However, it is fine: it is not an error to model the TrapezoidalMap to have these responsibilities.
2. You could see construction and query as algorithms working on a TrapezoidalMap (which just handles the geometry of trapezoids and its points/segments) and a DAG (that allows for searching in the structure). This is a bit more complicated, but a better structure, because, in this case TrapezoidalMap and DAG are two separate general purpose data structures used by algorithms.

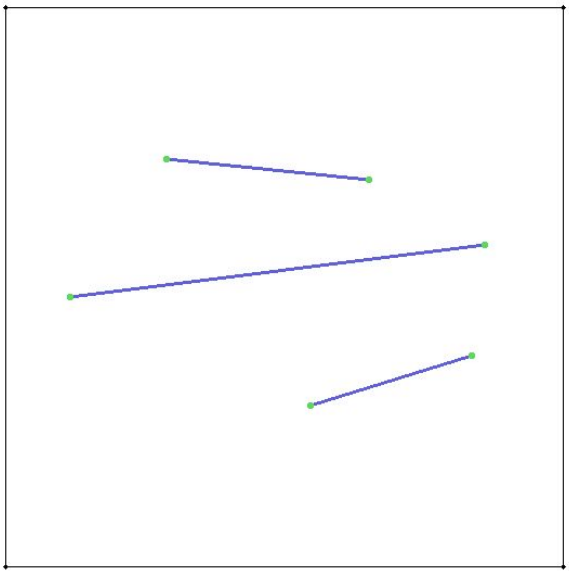
IMPORTANT: READ CAREFULLY! (THERE WOULD BE NO EXCEPTIONS).

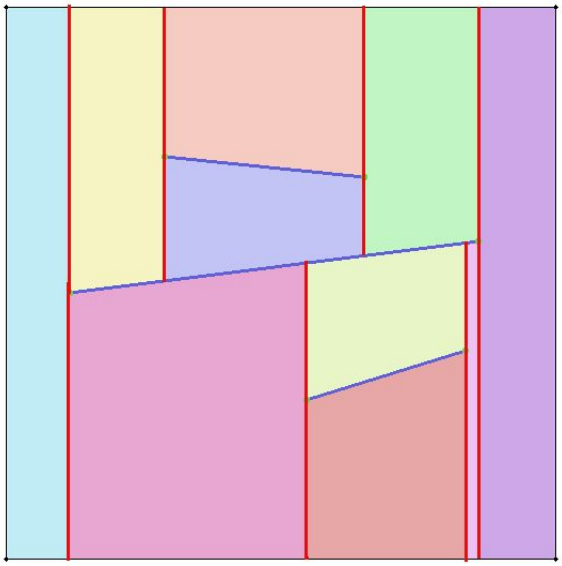
This is an ***individual* project:** you can collaborate in order to solve **high level problems,** but projects **with similar pieces of code or similar patterns** will be not tolerated**: *both* projects will be rejected and further actions will be taken.** We assure you that we are able to catch similar patterns in codes, you can refactor as many variables you want and change your code. Probably, your project will be rejected if you copy even a function. **Do not develop your code side by side with another person. Do not plan your project structure with another person. Do not look at already submitted projects, not even to take inspiration.**

You cannot use external libraries. Whoever **uses** (or, even worse, **copies**) some **complex code** from an external library or an external website (StackOverflow included) will be penalized. Of course, you can copy a few lines of code, as long as you understand them. If you need to use some simple code taken by an external link or library, **please quote the link in a comment**. You can obviously use all the data structures/algorithms in the standard library.

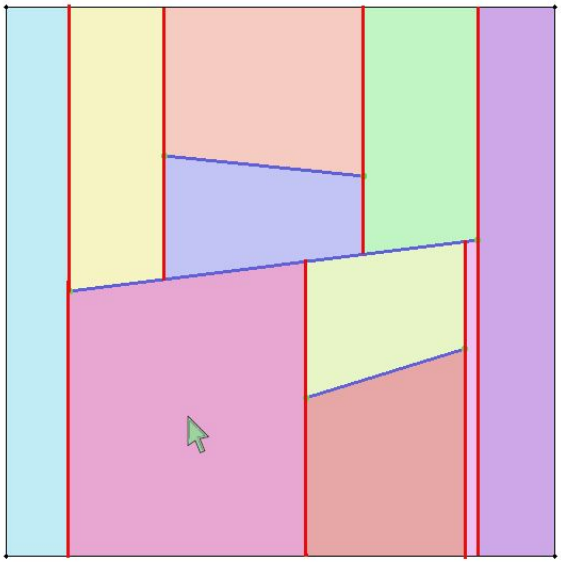
Examples

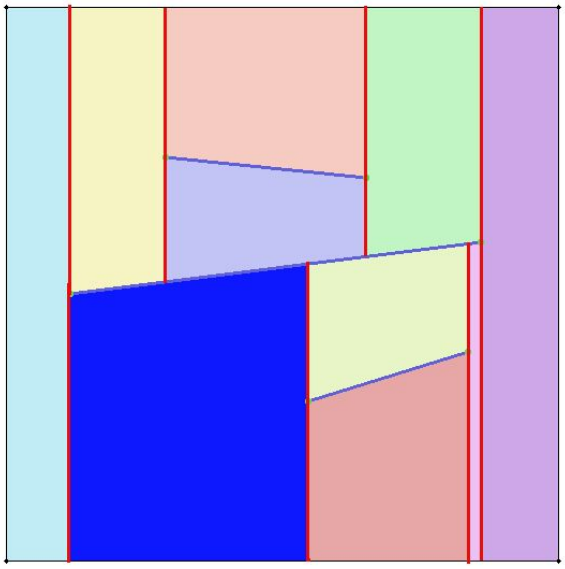
Construction of trapezoidal map (file **slide_example.txt,** it is similar to the slide example).





Query example (trapezoid highlighted in blue, but you can choose any color):





Tips (read all of them, at least twice a day!)

- **C++ is not Java!** Study C++ from the slides we provided, and if needed, on the internet. If you need any help with C++, contact us by mail (stefano.nuvoli@gmail.com, g.cherchi@unica.it). We are completely aware it is not an easy language.
- Algorithms must be defined **AS A FUNCTION** (in a namespace, if you want to hierarchically organize your functions) and **NOT AS METHOD (OR STATIC METHODS) IN THE CLASSES** (it is not Java).
- Data structures have to be defined in classes. Try to define a class for each data structure: each one must implement the methods of their responsibilities. Separate data structures from algorithms!
- Try to **AVOID using dynamic allocation** whenever it is possible (it will be evaluated!). We remind C++ is not Java, objects can be allocated in the stack without dynamic allocation (new), so we suggest you study the slides and see some examples on the internet. Avoid pointers, unless it is necessary!
- You will need to i**nitialize some objects in the constructor**. Use C++ style attribute initialization (it is different from writing an assignment operator in the constructor). It is different from Java!
- The base project compiles with zero warnings with GCC. With clang (macOS) you have just some warnings about the use of deprecated OpenGL functions (ignore them). **Try to submit a project with zero warnings, at least the important ones (for example unused variables warnings are not a problem)**. Warnings are indicative of bad and error-prone coding. A zero-warnings code does not mean that is good, but good code always produces zero warnings.
- Please organize your code following the OOP (**Object Oriented Paradigm**). **Keep separated algorithms from data structures**. Write short methods which solve standalone problems when it is possible.
- Separate the **definition** of a class or a function (files .h) from its **implementation** (files .cpp). You can do the same for templated or inline methods: you have to include the source file at the end of the header file and make sure that each implemented method in the source file is "inline" or templated. Check some examples in the cg3lib folder or on the internet.
- Do not use **"using namespace ..."** on headers files.
  http://stackoverflow.com/questions/5849457/using-namespace-in-c-headers
- **STRICTLY AVOID** the usage of **global variables** and other shortcuts.
  http://c2.com/cgi/wiki?GlobalVariablesAreBad
- Use **const** keyword (evaluated). If you don't know how to use const keyword, take a look at the following tutorial.
  http://www.cprogramming.com/tutorial/const_correctness.html

- Asserting is a powerful debugging tool to test correctness of your algorithms at runtime. https://stackoverflow.com/questions/1571340/what-is-the-assert-function
- **Debugger** is your friend and it is a very powerful tool. In the 90% of the situations, using a breakpoint is faster (and a smarter choice) than using std::cout (or std::cerr). The debugger allows you to find the exact point where your application is crashing (and why) and to see the state of all the variables in every scope during the execution of your application. If you have never used a debugger, **this should be the best time to start**.
- Comment your code, documentation is important. It is not asked to comment each operation and line of your code, but it is required to write **high-level comments which explain what a function or a block of code does**.
- Follow the guidelines given during the short C++ course!

Submission

You must submit your final project through github or bitbucket, and you are asked to **commit and push often during the development**.

When you are ready to submit the project for evaluation, make sure that you have the final version of your project in github, and upload on moodle a pdf file which show how you organized the project. You have to explain **(high-level)** how you organized the structure of your project (folder, data-structures, algorithms) and what are your results (maybe some images, time efficiency for the test files in the dataset). Write also about the problems: cases in which your algorithm does not work. **Write in the pdf file the link of your github repository.**

IMPORTANT: READ CAREFULLY! (THERE WOULD BE NO EXCEPTIONS).

Projects with a **too small number of commits** (let's say less than 20) **will be rejected**. Obviously we do **NOT** want to see **2 large commits and 18 small commits** (in which you modify 2 lines in a file). We suggest committing and pushing every time you complete a functionality and, in any case, at the end of every day you have worked on your project.

You can use your own **PRIVATE** repository of github and bitbucket.

Access here to this link to have an assigned private repository from our account.

https://classroom.github.com/assignment-invitations/ade7bae671e014052787740fd1e141a1

1. Access to your github account;
2. Accept the assignment in order to have a private repository;

3. Push the base project (just the content of GAS_2021_TrapezoidalMaps folder) on your repository (**it must be your first commit**);
4. Commit your changes often and do not forget to push at the end;

Grade and Deadlines

The grade (total of 34/30) is composed of:

8/30: **Correctness** of the project;

6/30: **Documentation**, code readability, code style;

10/30: **Structure** of the project, code modularity;

10/30: **Efficiency** in time and memory, complexity of the algorithms.

The deadline for the submission of the project is May 31st. If you submit your project after May 31st 23:59, your grade will have a **malus** (-1) every month. This is a summary table:

Submitted before	Maximum Grade
23:59 May 31st	34/30
23:59 June 30th	33/30 (-1)
23:59 July 31st	32/30 (-2)
23:59 August 31st	31/30 (-3)
23:59 September 30th	30/30 (-4)

After September 30th, you will not be able to submit the project for evaluation. You will have to wait for the end of the winter semester 2021/2022 to get your new project (and you will get a significant penalty in the grade).