COMP203P: Scenario Week 4
Art Gallery Competition

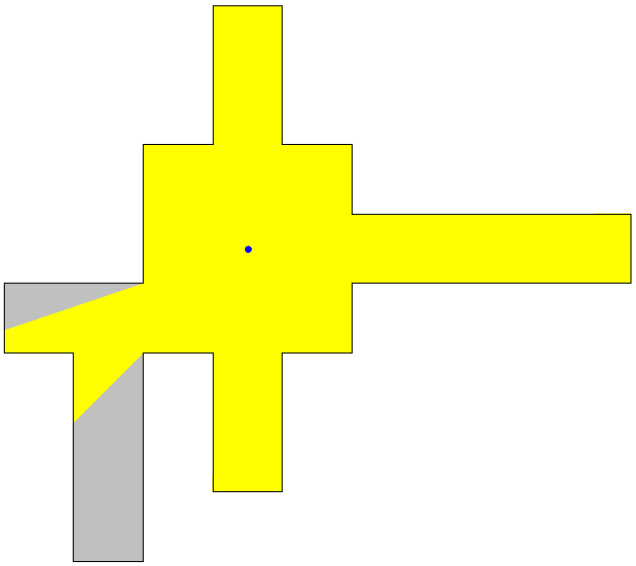
University College London
Department of Computer Science
scenario@cs.ucl.ac.uk

22–26 February 2016

1 Art Gallery Problem

This scenario week will be dedicated to the *Art Gallery Problem*. The problem is concerned with finding the minimal *complete* set of immobile guards (or, if you prefer, surveillance cameras) who together can observe the whole space of a gallery with precious pieces of art. Your team has been hired by a directorate of a big museum with a number of galleries with the purpose of developing an application for finding a good *complete* set of guard positions for each of the galleries.

Formally, a gallery is represented as a simple two-dimensional polygon, without holes or self-intersections, but not necessarily convex. For instance, the following figure illustrates a lonely guard in the middle of a gallery, with his visibility area painted yellow:



The guards cannot move, but they can see as far as possible unless their line of sight is interrupted by an obstacle (e.g., a wall). As indicated by the grey areas in the figure above, our lonely guard cannot see the whole gallery alone, so he needs some friends to help him. Indeed, putting a guard (or a camera) at each corner of a gallery would solve the problem. However, since the budget

of the museum is limited, it is unlikely that the directorate will go with such solution, as it is too costly. Instead, we should try to figure out what is the *minimal* number of guards, and what are their positions within the gallery (or on its boundary).

Unfortunately, the problem of finding the positions for the smallest possible set of guards for an arbitrary gallery in an automatic way turns out to be very difficult and is, in fact, *NP-hard*,[1] which means that at this moment humanity doesn't have an efficient algorithm to solve it.

That said, it doesn't mean that the problem is not worth solving, and as of today many methods exist that work *well enough* in terms of the size of the solution. For instance, in 1975, Václav Chvátal proved a theorem, which states that if the number of vertices in the polygon, representing the gallery, is n, then it can be guarded by at most $\lfloor n/3 \rfloor$ guards.[2] This number is always sufficient, and is sometimes necessary for some classes of polygons. The same work also presented what is nowadays a textbook algorithm for solving the art gallery problem with at most $\lfloor n/3 \rfloor$ guards. Since then, many specialized algorithms have been found to solve the problem in a more optimal way (i.e., with a smaller set of guards) for some specific classes of polygons.

Your main task for this week will be to investigate different ways of solving the Art Gallery Problem and compare them, designing an algorithm, which performs as well as possible on a diverse selection of different polygons, delivering for each a *complete set of guards*, who can together observe the whole interior of a given gallery. Since different algorithms can perform differently on various polygons, the directorate has decided to arrange a *competition*, so solutions submitted by the teams will be tested on the same set of polygons and ranked accordingly.

But this is not all! Yet another rival team of hackers has approached the directorate of the museum in an attempt to be hired instead of you, demonstrating their solutions for *some other galleries*. Fortunately for your team, *all* their solutions are flawed and aren't, in fact, complete guard sets. Unfortunately for your team, you will have to prove it by providing a refutation for each of their polygon/guard set pairs. Therefore, you will have to develop a procedure for *checking* a supposedly incomplete set of guards (who cannot see the whole gallery) and finding a *refutation* for it, i.e., a point within the polygon, which is not visible from any of the guards.

Section 2 explains the details of each of task of the scenario and describes the grading system. Section 3 provides answers to some possible questions. Section 4 lists some useful resources.

2 Tasks and Grading

For this week, you will be working in groups of four members. Each group is assigned a **unique identifier** (a name of an animal specie) and a **password**, which will be mailed to you at the beginning of the week. As customary, we advise to keep your password secret, as it will be used for submission of the results by your team to the testing server.

There will be five tasks of different difficulty, and we encourage you to make the best of splitting the workload to solve them in parallel. The maximal grade for this scenario week is **100 points**. The table below outlines the distribution of the points between several tasks:

[1] `https://en.wikipedia.org/wiki/NP-hardness`

[2] `http://www.sciencedirect.com/science/article/pii/0095895675900611`

Task Name	Maximal Score (points)	Details
Computing the Set of Guards (Part 1)	30	Section 2.1
Checking the Set of Guards (Part 2)	20	Section 2.2
Visualisation of the Solutions	15	Section 2.3
Implementation Report	15	Section 2.4
The Competition	20	Section 2.5

The next subsections outline the details of the tasks.

2.1 Computing the Set of Guards (Part I)

In this task, you will be given a text file `guards.pol` (available from the Moodle page of the course), containing definitions of **30 simple polygons** (with no holes or self-intersections).

Each line contains a polygon number, followed by its definition after a colon (ignoring possible spaces between other lexical tokens). The definition of a polygon is a list of coordinates of its vertices (x, y), where x, y can be integers or *double-precision* floating-point numbers. The sequence of the vertices is arranged in a way that the interior of the polygon will stay *on the left*, when one "walks" from one vertex to the next one. The successor of the last vertex in the list is the first vertex. For instance, the following file describes two polygons in the defined format, numbered 1 and 2, correspondingly and depicted in Figure 1:

`guards.pol`

```
1:  (0, 0), (2, 0), (2, 1), (1, 1), (1, 3), (0, 3)
2:  (0, 0), (5, 0), (5, 2), (4.2312351, 1.234), (1, 1), (0, 2)
```

Your goal for this task is to compute, for each polygon, a set of guards, which together can see its whole interior, *and the size of the set is not larger than Chvátal's boundary* $\lfloor n/3 \rfloor$.

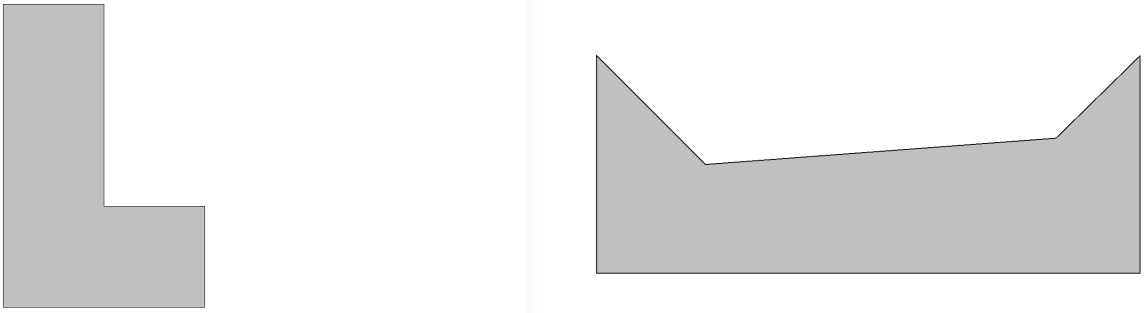
The solution for this task is a text file. You can implement your algorithm in any programming language of your preference and use any libraries you consider necessary. You don't have to (and should not) submit the code.

The file with the results should start with the first line containing the name of the team and the second line being its password. If those don't match, the file will not be accepted by the system. The remaining lines should contain the solutions in the following format:

polygon_number : comma_separated_list_of_guards

where guards are represented as pairs of their coordinates (x, y). A solution for each problem, along with its number, should be placed on a separate line. There is no specific order imposed on the sequence of the guards or solutions. For instance, a solution for the above file `guards.pol`, submitted by the team `tiger` with a password `lt671vecrskq` might look as follows:

```
tiger
lt671vecrskq
2:  (0, 2), (4.3, 1)
1:  (0.2, 2.5), (2, 0.5)
```



(a) Polygon `1`

(b) Polygon `2`

Figure 1: Two simple non-convex polygons.

The text file with the solution should be submitted in the form of **Part 1** of the following page:

`http://artgallery.cs.ucl.ac.uk`

WARNING! Parts of the input are specified via *double-precision floating points*, which assumes working with ε-equality instead of equality.[3] Your solutions may contain double-precision floating-point numbers, as well. The server uses $\varepsilon = 0.0000000001$, therefore all values with difference smaller than 0.0000000001 will be considered *equal*. This value of ε is unsound for arbitrary floating-point computations, but should suffice for the solutions of the problems in this scenario.

The file being submitted might not contain *all* solutions, so only presented ones will be graded by the system. The files with typesetting errors will not be accepted for grading. A solution for a specific polygon **will not** be accepted if at least one of the following conditions holds:

1. there is a guard in a solution, whose coordinates lie *outside* of the corresponding polygon;
2. there is a point in a polygon, which is *not* visible from any of the guards in the solution;
3. the size of the solution is larger than $\lfloor n/3 \rfloor$, where n is the size of the polygon.

Grading of a solution typically takes at least 3-4 minutes, and might take significantly longer, depending on the current server load. Once grading is complete, the statistics for the solution will appear in the joint score table for the **Part 1**. Notice that only the **last** solution is taken into account, so all previous results for a team are superseded by the next submitted solution.

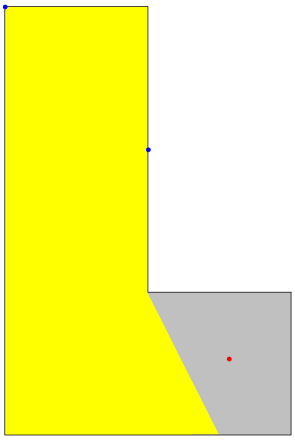
The server will stop accepting solutions at 14:00 GMT, 26 February 2016. Make sure to submit your best results by then.

The maximal grade you can get for this part is **30 points**, which corresponds to the number of the polygons in the assignment.

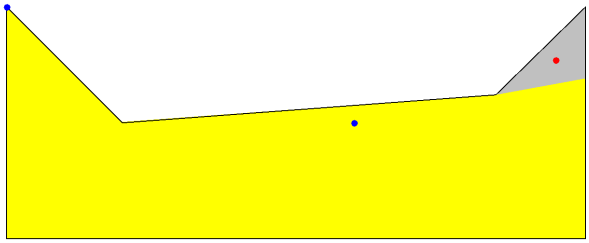
2.2 Checking the Set of Guards (Part 2)

In this task, you will have to develop a procedure for checking a set of guards for *completeness* and finding a *refutation* for a given polygon P and an *incomplete* set of guards S. A refutation is a single point within P, such that it is not visible from any of the guards in S.

[3] `https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/`



(a) Polygon with guards `1`



(b) Polygon with guards `2`

Figure 2: Galleries with incomplete guard sets.

As an input, you get the text file `check.pol` (available on the Moodle page of the week), containing **20** lines, each of which corresponds to a numbered polygon with an *incomplete* set of guards in the following format:

polygon_number: polygon; guards

where *guards* is a comma-separated list of points on the plane. For instance, the following toy example represents an input for this problem:

```
1:  (0, 0), (2, 0), (2, 1), (1, 1), (1, 3), (0, 3); (0, 3), (1, 2)
2:  (0, 0), (5, 0), (5, 2), (4.2312351, 1.234), (1, 1), (0, 2); (0, 2), (3, 1)
```

The visualisation of these two inputs is shown in Figure 2, with red dots marking possible refutations in the "grey area" of a corresponding gallery.

The main part of the solution file, whose **first two lines are your team name and password** (just like in **Part 1**), should contain numbered refutations, **one for each polygon/guard set**. For instance, a possible solution file by the team `tiger` for the problems in `check.pol` above might look as follows:

```
tiger
lt671vecrskq
1:  (1.56, 0.53)
2:  (4.74, 1.53)
```

These are the refutations, corresponding to the red dots in the grey ares in Figure 2.

You should submit your text files with the solutions to the **Part 2** submission web form at

`http://artgallery.cs.ucl.ac.uk`

The solution for this task is also a text file. You can implement your checking algorithm in any programming language of your preference and use any libraries you consider necessary. You don't have to (and should not) submit the code.

Similarly to part one, a solution might be incomplete, i.e., omit solutions for some part of the input. The last submitted solution supersedes the previous result (with both accepted and failed parts).

WARNING! When working with *double-precision floating points* in this part, don't forget about using ε-equality instead of plain equality (see page 4 for details).

The server will stop accepting solutions at 14:00 GMT, 26 February 2016.
Make sure to submit your best results by then.

The maximal grade you can get for this part is **20 points**, which corresponds to the number of the problems in the assignment.

2.3 Visualisation of the Solutions

You can provide an application for visualising the polygons and the results of your algorithms (it will also make testing of your solutions easier). The app shouldn't feature sophisticated input components and can just take text files, similar to the ones describing problems in **Part 1** and **Part 2** as console inputs. Your implementation of the visualizer should feature the following elements:

1. drawing the polygon shapes;
2. drawing visibility areas from specific guards;
3. drawing refutations for incomplete guard sets.

The applications should be demonstrated to TAs in a dedicate time slot between 14:00 and 17:00, 26 February 2016.
Book yourself a slot by mailing your preferred time and team name to
`scenario@cs.ucl.ac.uk`

The maximal grade you can get for this part is **15 points**.

2.4 Implementation Report

None of the previous assignments required you to submit any working code. However, to get additional points you can submit a short report, describing your implementations of the procedures for **Part 1** and **Part 2** of the scenario. In particular, your report should outline the following components of your development:

1. Which language has been used to implement the algorithms and what external libraries were used? Which libraries were used to implement the visualisation?
2. Which geometric algorithms (including auxiliary ones) were implemented/invented for constructing the guard set in **Part 1**?

3. Which algorithms (including auxiliary ones) were implemented/invented for checking the guard set in **Part 2** and finding refutations?
4. What are the complexities/experiences dun times for the implemented solutions for **Part 1** and **Part 2**.
5. How were the algorithms for computing/checking guards sets tested?
6. How were the input files processed, and output files produced?
7. How the design/implementation workload split between the members of the team?
8. A link to the repository with the development (can be made public after the scenario week).

The reports should be submitted to the Moodle page or mailed to
`scenario@cs.ucl.ac.uk`
by 17:00, 26 February 2016.

The maximal grade you can get for this part is **15 points**.

2.5 The Competition

Some extra points can be earned from the competition, which takes place among the results from **Part 1**. The solutions will be ranked per polygon according to the size of submitted complete guards sets (smaller is better). Per-polygon ranks will be aggregated to compute the total ranks, which are displayed by the corresponding score table. Teams that submitted acceptable solutions for all the problems of **Part 1** will be ranked amongst each other first, followed by other teams. In the case of equal results, the same ranking might be assigned to multiple teams.

You can earn up to **20** points for this part. The following table describes mapping of rankings to the points earned (with comments):

Rank	Score
1	20 (and a chance to openly brag about this)
2–3	15 (you are still awesome)
4–5	10 (good job, too)
6–7	5 (not bad)
≥8	0 (sorry)

Good Luck!

3 Frequently Asked Questions

Q: We have a problem with a task and/or do not understand the problem and the requirements for the solutions. How can we clarify this?

A: Ask a TA during the dedicated sessions or send an e-mail to `scenario@cs.ucl.ac.uk`. The first option is preferable, as your e-mail might be answered only in a few hours.

Q: Where should we submit the solutions for **Part 1** and **Part 2**?

A: The URL for solutions is `http://artgallery.cs.ucl.ac.uk`.

Q: What are the deadlines for the automatically-checked solutions?

A: The deadline for **Part 1**, **Part 2** and **Competition** is **14:00 GMT** ($\pm\varepsilon$), **26 February 2016**. After that moment the server `http://artgallery.cs.ucl.ac.uk` will stop accepting solutions.

Q: We didn't get an email with the confirmation that our solution has been processed. What should we do?

A: Grading a solution on an "empty" server takes about 3-4 minutes. Under high loads, processing your solution might take up to half an hour. If the results don't appear in a score table by then and you don't get a notification email, please, contact the Scenario Week organizers (`scenario@cs.ucl.ac.uk`).

Q: We got an email reporting some errors in our solution. However, the scoreboard also shows some parts as "N/A", even though no issuees were reported for them. How so?

A: You solution file might be missing some parts (i.e., solutions for specific polygons etc.). In the case of errors in the solution, only those are reported in an automatic email, without mentioning missing parts of the submitted file. Missing parts are only reported if the rest of the solution file is okay.

Q: In Part 1, can we submit empty solutions for some problems (e.g., having lines like `20:`)?

A: No, the whole submission file will be then rejected by the server input validator. To submit a partial solution, just omit these parts from the file.

4 Useful Resources

- Art Gallery Problem page on Wikipedia:
  `https://en.wikipedia.org/wiki/Art_gallery_problem`.
- Joseph O'Rourke. *Art Gallery Theorems*:
  `http://cs.smith.edu/~orourke/books/ArtGalleryTheorems`
- Visibility Polygons page on Wikipedia:
  `https://en.wikipedia.org/wiki/Visibility_polygon`