Origasmi: Questions & Answers

Shape Files

Q1. Is there a maximum area that would fit all shapes?

This is not given. However, you may assume that all coordinates in the shape files may be parsed and represented using the double-precision binary floating-point format (see Q11).

Q2. Is it allowed for polygons of the same shape to overlap?

Yes. A shape is defined as the *union* of its polygons. You can visually imagine the polygons as being superimposed on top of each other to produce the shape. Note that a grid point that is contained within multiple (overlapping) polygons still counts as a single point for the scoring function.

Q3. Can a shape file be empty?

No. Every original shape must contain at least one polygon, and hit at least one grid point. This means that, for any given original shape, $|O| \geq 1$.

Q4. Are the vertices for each polygon in the shape file enumerated in a clockwise or anticlockwise order?

This is not given. Each polygon may define its vertices in either order. Both orders should be identically interpreted as defining a closed, filled polygon.

Q5. Can shape files interleave vertices from different polygons?

No. All vertices for a given polygon will be specified contiguously.

Q6. Can shape files define polygons that have just one vertex?

No. You may assume that each polygon in the shape files will consist of at least three vertices.

Q7. What is the allowed range of polygon identifiers?

You may assume that the range will not exceed $[0, 32768)$.

Operations Files

Q8. Is it allowed to output a blank operations file?

Yes. A blank operations file will be interpreted as having no operations, leaving the 2×1 paper in its initial position. The same applies for operations files that consist of just empty lines.

Q9. Do nullipotent operations (such as `T 0 0`) contribute towards the operations count?

Yes.

Q10. Do empty lines contribute towards the operations count?

No.

Real Numbers

Q11. What data type should be used for representing real numbers ($r \in \mathbb{R}$)?

The scoring tool uses the double-precision binary floating-point format, defined as binary64 in the $\text{IEEE } 754$ standard. In C-based programming languages, this would correspond to the `double` data type. You are encouraged to use this data type or its closest representation in your language.

Q12. How should real numbers be formatted for output?

Real numbers should be formatted using decimal notation. Representations in other bases, such as octal or hexadecimal, are not supported. Scientific notation is permitted, but should not be necessary; if used, it must conform to the E notation of C-based languages (for example, `1.2E-6`). The period, `.`, must be used as the decimal separator (where required), and the hyphen-minus, `-`, must be used as the negative sign (where required). Thousands separators are not allowed. Infinities and NaN are obviously not allowed.

Q13. What precision should be used for outputting real numbers?

The binary64 representation can maintain around $15–17$ significant decimal digits of precision, so you should aim to output up to this many decimal places. Note that there is a hardcoded limit of 128 characters per line in the scoring tool; do not exceed this length.

Q14. What about floating-point inaccuracies?

Due to the finite precision with which real numbers can be represented in a digital format, floating-point computations may suffer from a loss of accuracy.

For example, one would expect that evaluating $1.0 - 0.9 - 0.1$ should give exactly 0.0. However, under the binary64 representation, the actual result would be around $-2.77\text{E}{-17}$. Suppose, for the sake of the argument, that you decide to translate the initial paper upward by the negation of this amount: `T 0 2.77E-17`. If you do not implement error tolerance, then it would appear that all the grid points lying on the x-axis (such as $(0, 0)$) no longer belong to this shape, which is an incorrect result when reasoning about real numbers.

To compensate for this inaccuracy, we introduce the notion of error tolerance into our scoring algorithm. Specifically, grid points whose shortest distance to the nearest boundary of a shape is less than a certain threshold (called “epsilon”) would still be considered as belonging to the shape, even if they appear to lie outside it. The value for this epsilon is fixed at 0.00000095367431640625 (which is 2^{-20}).

Notwithstanding, it is still possible for minor discrepancies to arise in some boundary cases. The final values depend on the implementation-specific sequence of floating-point operations,

which would expectedly vary from one program to another. In cases of unresolved discrepancies, the scores generated by the scoring tool are taken to be authoritative.

Scoring Function

Q15. Should grid points that lie exactly on a shape's boundary be considered a part of the shape when calculating the score?

Yes. For example, both points $(0, 0)$ and $(2, 1)$ would be considered a part of the shape formed by the initial paper. Per Q14, even points that lie a very small distance outside the boundary would be considered a part of the shape.

Q16. Is it possible for $|S|$ to change after a translate or a rotate operation?

Yes. One would be correct in assuming that translate and rotate operations do not change the real area of the submission shape. However, the value of $|S|$, which represents the number of grid points in the submission shape, only has an approximate relationship to the shape's area, due to the discrete (non-infinitesimal) resolution that is used for defining the grid points. Thus, translate or rotate operations may cause the shape to hit a different number of grid points, even though its area has not changed.

Q17. What is the total range spanned by the grid points?

There is no fixed range. The two-dimensional grid of points starts at $(0, 0)$, and extends to the furthest vertex (belonging to either the original shape or the submission shape) in all four directions, with a gap of 0.015625 (the resolution) between each grid point and its closest neighbours.

In other words, the grid may be said to consist of the following set of points:
$\{ \ (0.015625\,\mathrm{x},\ 0.015625\,\mathrm{y}) \ | \ \mathrm{x} \in \mathbb{Z} \ \text{ and } \ \mathrm{y} \in \mathbb{Z} \ \}$.

Q18. Do submissions have to match the shape exactly to receive a score?

Obviously, no. You should aim to come up with the best approximation that maximizes your score, per the given definition of the scoring function.

File Formats

Q19. What newline convention should be assumed?

The Unix convention uses a line feed (represented using the `"\n"` escape sequence in C-based languages), whilst the Windows convention uses a carriage return immediately followed by a line feed (represented using the `"\r\n"` escape sequence).

For shape files, you may assume that we will be using the Unix convention. For origami operations files, you may use either the Unix or the Windows convention, irrespective of which operating system you request for your program. Note that a carriage return by itself (`"\r"`) will not be treated as a newline.

Q20. Do files need to be terminated with a newline?

They can, but they don't have to. For shape files, you may assume that the last vertex will be followed by a final newline. For origami operations files, you are free to choose whether to append a newline after the last operation.

Q21. What character encodings are supported?

The character encoding must be strictly US-ASCII. You may instruct your program to use an ASCII-derived encoding, such as ISO-8859-1 or UTF-8, provided that you only use its ASCII-conforming subset of characters.

Submission Programs

Q22. Are programs allowed to spawn threads?

Yes, provided that the parent process terminates before the one-minute time limit.

Q23. Are programs allowed to spawn sub-processes?

Yes, provided that the entire process hierarchy terminates before the one-minute time limit. Per Q38, the main process (that was launched to run your program) should be the last to terminate. If your child processes cannot be programmed to terminate gracefully before the time limit has elapsed, you would need to abort them from the parent process.

Q24. Are programs allowed to access the Internet?

No. Internet access will not be supported.

Q25. Are programs allowed to run with administrator privileges?

No. You should assume that programs will be run under an account with restricted privileges (for security reasons).

Q26. In which directory will programs be placed to be run for evaluation?

This is not given, and may be any arbitrary directory on the evaluation machine. However, you may assume that the current working directory of the process running your program will be set to the directory containing your submission.

Q27. Can absolute file paths be hardcoded into the programs?

No, you should not assume any absolute file paths. For the original shape file and origami operations file, you must use the file paths specified as the command-line arguments to your program (see Q35).

Q28. Can programs create temporary files?

Yes, temporary files are allowed. You may create such temporary files in the current working directory of the process running the program. You should ensure that their filenames do not collide with other files in the working directory (such as your program).

Q29. Is it allowed to use another version of the programming language or environment?

The list of supported versions given in the specifications document represents the actual versions of the runtime environments that will be installed on the evaluation machine. You are allowed to target other versions, but it remains your responsibility to ensure that your program is fully compatible with the stated version in the specifications document.

Q30. What if the program requires a particular extension, patch, or service pack to run?

You should preferably aim for your programs to run on the standard version of the operating system and runtime environment. If you have any custom requirements, you should acquire the explicit confirmation of the judges through the designated Google group before the competition ends.

Q31. Should programs aim to take one minute to complete?

Not really. Submissions must ensure that they complete within a *maximum* of one minute. It is possible that you might come up with a winning entry that requires substantially less time to complete.

Submission Contents

Q32. Are submissions allowed to contain additional files, such as DLLs, that are required for the program to run?

Yes. Your submission may contain executables, dynamic-link libraries (DLL), resources, and any other file formats. You should package all your files into a .zip or .tar.gz archive that is uploaded as your submission to the programming competition website.

Q33. Are submissions allowed to make use of third-party libraries (such as open-source projects)?

Yes, provided that the libraries were published before the programming competition commenced. You must include all libraries that do not come as standard with the runtime environment as part of your submission. In the case of proprietary libraries, you must ensure that your licence permits them to be bundled with your submission. Pirated software is strictly prohibited.

Libraries that will require any installation or custom configuration to run on the evaluation machine are not supported. (See Q30 for extensions of the software framework of your programming language.)

Q34. How will the submission be executed by the evaluation tool?

Before evaluation, all the files in the submitted .zip or .tar.gz archive will be extracted into a dedicated directory. The evaluation tool would set its working directory to the directory containing the extracted files, and execute the following command through the operating system's command-line interface:

```
origasmi shape_file operations_file
```

…where *shape_file* and *operations_file* are replaced by the file paths of the original shape file and origami operations file, respectively (see Q35). On our Linux system, the `PATH` environment variable would include the current directory (for example, by having set `PATH=$PATH:.`). Thus, calling `origasmi` would be equivalent to calling `./origasmi`.

For the above command to launch your program, your submission must include a suitably-named file that can be run directly by the operating system. For Windows, this file should be an executable named origasmi.exe or a batch file named origasmi.bat (case-insensitive). For Linux, this file should be an executable or shell script named origasmi (case-sensitive).

By using a batch file (for Windows) or shell script (for Linux), you may execute any custom command for launching your program, as demonstrated in the examples below. It is important that you pass on the command-line arguments to your program. Linux shell scripts should start with the appropriate shebang, such as `#!/bin/sh` for the Bourne shell (or compatible).

For example, suppose that your main program is a Windows executable named MyProgram.exe (which, for some reason, cannot be renamed), and requires a DLL named MyLibrary.dll. For your submission to be run by the evaluation tool, you should create a batch file named Origasmi.bat (case-insensitive) that launches the executable. All three files should be packaged into an Origasmi.zip archive, which you upload as your submission. The batch file should consist of the line:

```
MyProgram.exe %1 %2
```

…where the `%1` and `%2` are required for passing the two command-line arguments to the program.

You may assume that the `PATH` environment variable includes the directories of all runtimes for the supported software frameworks. (For example, the Windows machine might contain "C:\Program Files\Java\jre7\bin" in its `PATH` for the Java installation.) Thus, your batch file or shell script may call the relevant runtime executable to launch your program just by specifying its filename. Per Q27, you must not assume any absolute paths for the runtimes.

For example, if your submission consists of a Java program packaged as a JAR archive named origasmi.jar, then your batch file or shell script should contain the line:

```
java -jar origasmi.jar %1 %2
```

On the other hand, if your submission consists of a Python script named origasmi.py, then your batch file or shell script should contain the line:

```
python origasmi.py %1 %2
```

And similarly for all other runtimes.

Evaluation

Q35. What kind of file paths may be provided in the command-line arguments?

The command-line arguments may specify either absolute or relative paths. Relative paths should be interpreted as relative to the current working directory of the process running the program. You can assume that each path will not exceed 80 characters in length, and will not contain any spaces.

Q36. What happens if the program does not create an operations file?
What happens if the program does not terminate within one minute?
What happens if the operations file exceeds 1000 operations?
What happens if the operations file contains an invalid line?

The submission will not be awarded any points for that particular problem. No consideration will be paid to the valid part of the operations file that may have been written.

Q37. Will the program be run again if it crashes or generates an invalid operations file?

No. Each program will be run exactly once on each problem instance.

Q38. When will the origami operations file be read by the evaluation tool?

The evaluation tool will wait until the process that was launched for running your program terminates. Once this process terminates, the evaluation tool may immediately read your origami operations file and compute its score (even if the one-minute time limit has not yet elapsed). Thus, if your program spawns sub-processes, you should ensure that the main process is the last to terminate. If your main process, or any of its descendant processes, takes longer than one minute to terminate, it would be killed, and the operations file ignored.

Scoring and Visualization Tools

Q39. What environment is needed to run the scoring tool?

The scoring tool is provided in two versions: as a Portable Executable (PE) for Windows, and as an Executable and Linkable Format (ELF) file for Linux. In both versions, the scoring tool is a 32-bit executable that runs on x86-compatible architectures (including x86-64).

In most cases, you should be able to run the correct version of the scoring tool directly on your system without installing any prerequisites. On 64-bit versions of Linux, you might need to install some 32-bit compatibility libraries. One solution would be to install ia32-libs through the following command:

```
sudo apt-get install ia32-libs
```

Q40. What environment is needed to run the visualization tool?

The visualization tool is a .NET executable that runs on .NET Framework 3.5 Client Profile (or later), or Mono 2.10.8 (or later), on either Windows or Linux. To install Mono on Linux, you may issue the following command:

```
sudo apt-get install mono-complete
```

The visualization tool requires the iTextSharp library, which is provided with the tool as itextsharp.dll. This file should be present in the same directory as visualize.exe. The same applies for the visualize.exe.config file.

Q41. How can the scoring tool be run?

The scoring tool is a console application, named score.exe, that must be run with exactly two command-line arguments:

1. the filename of the original shape file (that you were expected to approximate)
2. the filename of the origami operations file (that your program has generated)

These are the same arguments that would be supplied to your submission program.

On Windows, the scoring tool may be run by navigating to its directory and calling:

```
score.exe shape_file operations_file
```

On Linux, you might need to reference the current directory:

```
./score.exe shape_file operations_file
```

The scoring tool will write its results to the standard output (stdout). In case of error, an error message will be written to the standard output.

Q42. How can the visualization tool be run?

The visualization tool is a console application, named visualize.exe, that must be run with at least three command-line arguments:

1. the filename of the original shape file (that you were expected to approximate)
2. the filename of the origami operations file (that your program has generated)
3. the filename of the target report (that the tool will create)

… an optional sequence of switches (to alter the tool's behaviour)

On Windows with the .NET Framework installed (see Q40), the visualization tool may be run by navigating to its directory and calling:

```
visualize.exe shape_file operations_file report_file
```

On Linux with Mono installed (see Q40), the call would be:

```
mono visualize.exe shape_file operations_file report_file
```

If successful, the visualization tool will create a report in Portable Document Format (PDF) that demonstrates the effects of your sequence of operations on the submission shape. In case of error, an error message will be written to the standard output.

The optional switches (if used) must be specified *after* the three filenames. The '–' prefix is a required part of each switch. Supported switches are:

`-images`	In addition to the PDF report, the tool will save the image generated for each operation into a distinct Portable Network Graphics (PNG) file. These images will be stored under a *report*_images subdirectory.
`-final`	The PDF report will consist of just one page, corresponding to the final result after the entire sequence of operations is processed. This switch reduces the time needed to generate the PDF report, as well as its file size.
`-silent`	Prevents the tool from writing progress updates to the standard output.

Q43. Can the visualization tool be run without an operations file?

No. The second command-line argument must be a valid origami operations file. If you are only interested in visualizing the original shape, then you may supply an empty operations file.

Q44. Can the visualization tool be made to run faster?

If you are only interested in the final result, you should include the `-final` switch, which can improve the tool's speed significantly for large operations files (see Q42). If you are only interested in the score, you should use the scoring tool instead, which is even faster.

Q45. Are submissions allowed to call the scoring tool and/or the visualization tool provided by the judging panel?

Yes. If you do this, make sure to include the scoring tool and/or the visualization tool within your submitted archive (see Q32). Notwithstanding, you are discouraged from calling the provided tools as part of your programs, for performance reasons. The tools may take anywhere from milliseconds to minutes to complete (depending on the operations file). This might degrade the performance of your program if done too frequently. More importantly, your program retains responsibility for ensuring that these sub-processes do not result in the one-minute time limit being exceeded (see Q23).

Q46. What if there is a bug in the scoring tool and/or visualization tool?

If you are getting an unexpected result, first make sure that you are handling floating-point inaccuracies properly per Q14. If you are convinced that there is a bug in the tools, you should immediately bring this to the attention of the judging panel through the designated Google group. It would be helpful if you could write a simple sample shape file and operations file that demonstrate the problem, and attach these to your message. (However, make sure that you don't inadvertently divulge your algorithm to your competitors through the contents of these files.)

The judging panel will evaluate such reports as soon as possible. If the bug is confirmed, we will fix the tools and make updated versions available to all teams, announced through the Google group. If the bug is rejected, we will inform you through a reply to the Google group.

Q47. What is the difference between the scores reported by the scoring tool and the visualization tool?

The scores reported by the scoring tool and in (the final page of) the visual report should always be identical for a given shape file and operations file. However, the tools are compiled from distinct codebases, with the scoring tool being optimized for performance; thus, discrepancies might be possible. In such cases, please contact the judging panel immediately (using the instructions given in Q46). In case of unresolved discrepancies (for example, due to floating-point inaccuracies, as discussed in Q14), the scoring tool takes precedence.

Q48. What is the range rendered by the visualization tool?

The visualization tool renders a 4×4 area that spans along the $[-2, 2]$ interval on both axes. Shapes are allowed to have vertices that lie outside this area, which will still be counted for scoring, but will be clipped from the visual report.

Q49. Why is the visualization tool outputting truncated numbers in the generated PDF?

This is done due to visual space constraints. The truncation is only performed on the numbers when being written to the PDF; internally, their values would always be represented using the precision of the binary64 format.

Q50. What is the meaning of the various terms used in the scoring and visualization tools?

O	Original points	the set of grid points in the original shape (i.e. the shape you are trying to approximate)
S	Submission points	the set of grid points in the submitted shape (i.e. the shape that your program generates)
S ∩ O	Common points	the set of grid points in common between the submitted shape and the original shape
S \ O	Extraneous points	the set of grid points in the submitted shape that are not in the original shape
O \ S	Missing points	the set of grid points in the original shape that are not in the submitted shape