Polygon Offsetting. Contraure i expandir meshes 2D i 3D mantenint les seves propietats

Ricard Sampedro Ochoa

Resum— En aquest projecte es desenvoluparan diferents implementacions d'algorismes de polygon offsetting, contraent i expandint polígons i d'algorismes per a propagar propietats del polígon original als resultants del procés d'offseting. Aquest projecte ve motivat per una investigació prèvia duta a terme durant les practiques d'empresa realitzades a HP i que es pretén continuar amb aquest projecte. Havent dut a terme la implementació dels algorismes plantejats s'extrauran conclusions sobre les característiques dels diferents algorismes de polygon offsetting i com afecten a l'hora de propagar les seves propietats assignades i sobre com aquests mateixos algorismes es podrien aplicar sobre el mateix problema amb meshes tridimensionals i com afectaria això a la propagació de propietats.

Paraules clau—polygon offsetting, fabricació additiva, propagació de propietats, python, c++, VTK, CGAL, Clipper

Resumen—En este proyecto se desarrollaran distintas implementaciones de algorismos de polygon offsetting, contrayendo o expandiendo polígonos y de algorismos para la propagación de propiedades assignadas al polígono original a los resultantes del proceso de offsetting. Este proyecto esta motivado por una investigación precia llevada a cabo durante las prácticas de empresa realizadas en HP i que se pretende continuar con este proyecto. Habiendo llevado a cabo la implementación de los algorismos planteados se extraerán conclusiones sobre las características de los diferentes algorismos de polygon offsetting y como afectan estos al proceso de propacacion de propiedades y también, sobre como estos mismos algorismos podrían ser implementados sobre el mismo problema con meshes tridimensionales y, como esto afectaría también a la propagación de propiedades.

Palabras clave—polygon offsetting, fabricacion aditiva, propagación de propietades, python, c++, VTK, CGAL, Clipper

Abstract— In this project will be developed diferent implementations of algorithms of polygon offsetting, contracting or expanding polygons and algorithms for the propagation of assigned properties from the original polygon to the resulting ones of the offsetting process. This project is motivated by a previous investigation conducted during an internship in HP and is being expanded with this project. Having implemented the specified algorithms, conclusions will be extracted about the nature of the different polygon offsetting algorithms implemente and how they affect the process of property propagation and also about how these same algorithms could be implemented for the same problem in a 3 dimensional space with 3 dimensional meshes and how that would affect the property propagation process.

Index Terms—polygon offsetting, additive manufacturing, property propagation, python, c++, VTK, CGAL, Clipper

1 Introduction

In this document will be explained the development of the project "Polygon Offsetting. Contraure i expandir meshes 2D i 3D mantenint les seves propietats". For that end, both the motivation for this project and the objectives defined will be explained, followed by a review of the state of the art, making a point of reviewing the different types of solution for this problem that have been studied and the concepts necessary for the easy comprehension of this document. This will be followed by the methodology used for this project's development, the results obtained and the conclusions that those results allow to reach.

- *E-mail de contacte: soysampi@gmail.com*
- *Menció realitzada: Enginyeria de Computació*
- *Treball tutoritzat per: Debora Gil Resina (Departament de ciències de la computació)*
- *Curs 2015/16*

1.1 Motivation

This project's motivation comes from a previous project realized during my internship in HP this last year, in the 3D Software team and that now this project allows me to expand on.

That project consisted of the investigation and development of an algorithm that allowed, given a closed 3D mesh based on vertices and faces where properties (such as color or material composition) had been assigned to the vertices, to offset the mesh (contract and expand the mesh inwards or outwards), obtaining a new one where its faces are at a constant distance of the old's and also allowed for the transfer of the properties to the new mesh through all the intermediate steps taken to reach the final result.

To expand or contract a tridimensional mesh is a very complex and costly process, so as a first approach to this problem and to allow to reach some conclusions about the nature of the problem and the best way to solve it the scope was limited to an investigation to this same problem in a 2 dimensional space, working with 2d meshes consisting of polygons composed of vertices and edges from which conclusions will be reached about the nature of the problem and how it translates to a 3 dimensional space.
To investigate this problem, we'll be focused on techniques of polygon offsetting because their results coincide with the one's we're looking for and from what we gathered from the literature we've found, even though there's no implementation of property propagation for cases similar to ours, given the strategies used in polygon offsetting algorithms, we think we can implement a propagation algorithm which produces the expected results.

1.2 Objectives

1. To investigate different methodologies and families of algorithms that allow to solve the problem of offsetting a polygon.

2. Implement algorithms representative of the families of solutions investigated obtaining a polygon offset.

3. Implement an algorithm to propagate the properties assigned from a starting polygon to the result throughout all the intermediate steps.

4. Reach conclusions about the positives and the negatives of each solution and be able to extrapolate those conclusions to the same problem in a 3 dimensional space.

2 STATE OF THE ART

In this section will be explained the theoric concepts required to understand the diferent strategies for polygon offsetting and property propagation that have been studied explaining the general structure of their algorithms and how they relate to the objectives of this project.

2.1 Offsetting strategies

Polygon offsetting (contracting and expanding a polygon) is an important problem in computer assisted manufacturing, as offsetting a model to manufacture helps for instance to take into account milling errors and gouging in the building process in subtractive manufacturing, to find the accessible area for a tool of a given radius or for mold manufacturing, among other uses.

Given this, several different strategies have arisen to tackle this problem, which can be grouped in these groups according to the strategy used:

2.1.1 Vertex Propagation

One strategy regarding offsetting polygons is by propagating the vertices of the original polygon along the bisection of the angle formed by the 2 edges that coincide on each vertex until the distance to the original polygon is reached, or until all the edges, when propagated collapse on a single point, similar to the implementation of the crashing cycles algorithm, as specified on David Eppstein's paper "Raising roofs, crashing cycles and playing pool" [5]

Whenever when propagating 2 vertices, the lines they follow intersect, or when a vertex when propagated, would go out of the bounds delimited by the temporary vertices offset at that distance, we record an **event** which consists on a change on the result polygon's topology respect the original.

In this regard we can find 2 kinds of events:

- **Closing events**: 2 vertices when propagated, having the lines they followed intersect, causing the edge between them to collapse. This eliminates the 2 vertices, which are substituted for a new one, situated at the position the intersection would occur and which is propagated in the bisection of the angle formed by the neighbor edges of the one that has collapsed.

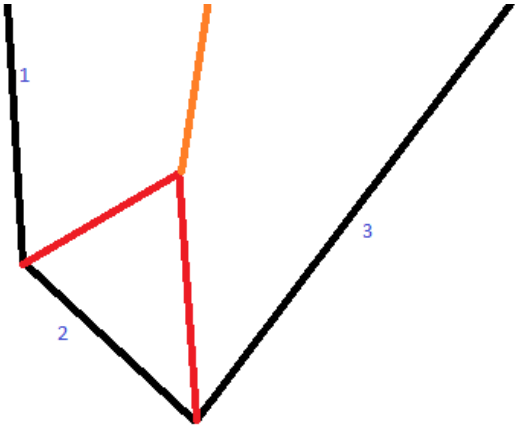

Fig. 1. Vertex from edges 1 and 2 intersects with vertex between 2 and 3

- **Split events**: when a vertex, when propagated would intersect with an edge of the polygon offset at the same distance this vertex is propagated to, which only can happen with vertices with angles associated bigger than 180 degrees. This eliminates the vertex and substitutes it with 2 new ones, connected to the edge the old vertex intersected with and each of the edges the old vertex was formed of.

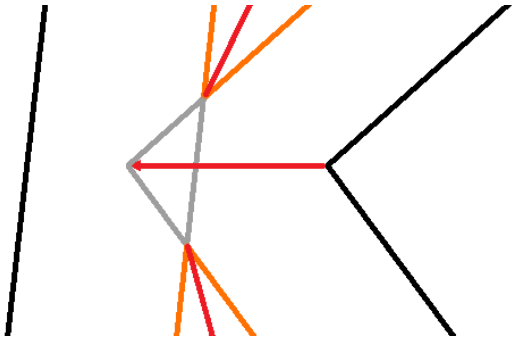

Fig. 2. Split event, resulting in a vertex turning into 2, splitting an edge. Gray: edges resulting due to vertex propagation.

The propagation process is repeated with the new polygon until every vertex collapses on a single point or until each vertex reaches the target distance.

If the algorithm ends by virtue of every vertex collapsing on a single point, the resulting geometric structure is known as the **straight skeleton** of the polygon.

2.1.2 Edge Propagation

Another strategy regarding offsetting polygons is by translating the edges of the original polygon along its normal vector the specified distance.

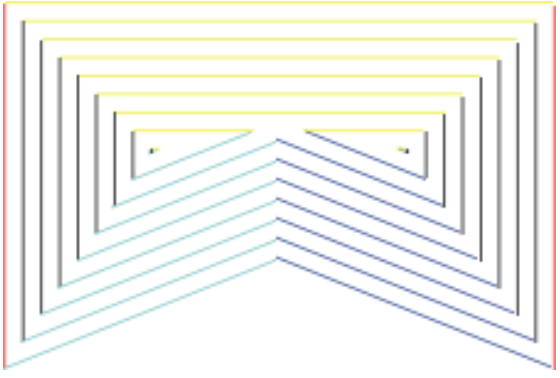

Fig. 3. Edge Propagation at several distances, with corresponding edges colored accordingly

This may lead to edges not connecting anymore with their neighbor or by edges intersecting with other ones if not taking into account.

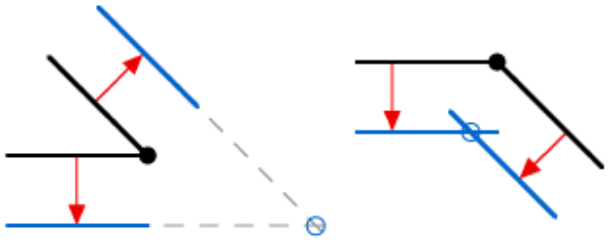

Fig. 4. Left: Edges failing to connect due to the translation.Right: Edges intersecting due to an inwards translation

This is solved in different ways according to the implementation, be it by correcting these cases as they come, extending the edges, shortening them or simply eliminating them if they collapse or by using different strategies by clipping the edges when intersecting.

2.1.3 Boolean Operation Techniques

Another family of strategies are those based around using Boolean operations, such as the one used in the Clipper library, based around using Minkowski sums [10] to calculate the offset, by calculating the Minkowski sum between the polygon and a radius defined circle or other geometric brush, depending of the parameters used, returning the inner or outer polygon according to the sign of the offset calculated.

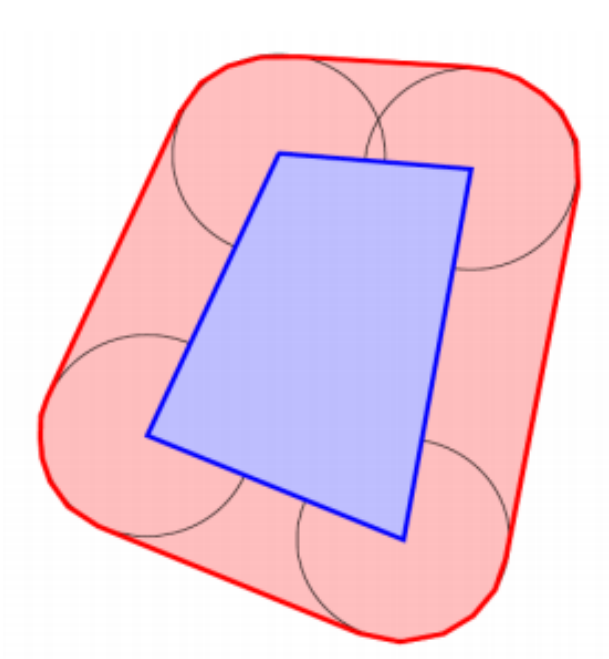

Fig. 5. Quadrilateral (blue), offset by the Minkowski sum of a sphere (red).

2.2 Property propagation

Regarding the propagation of properties to an offset polygon from its original, there's no implementations given the conditions specified in this project.

To propagate the properties of a polygon to a new one means to assign values to the vertices from an offset polygon according to those of the original polygon, following a structure defined by the algorithm.

We require that our implementation has a way to remember all the changes in the polygon's topology along the offsetting process, so that we might propagate the properties along the process correctly.

A structure that accomplishes that is the **straight skeleton**, as explained in **2.1.1 Vertex Propagation,** as we can propagate the properties of the original polygon to the new one using the structure of the skeleton for the propagations.

Given an implementation of a vertex propagation offsetting strategy, obtaining the straight skeleton to propagate the properties is trivial, as it's already calculated during the offsetting process, but in an edge propagation strategy the skeleton is not calculated.

3 Methodology

The methodology followed for the development of this project has been the "Waterfall", defining objectives and milestones to meet in an ordered fashion, according to the dependencies between them. Every objective required an investigative process and a degree of research that has been constant during all the development of the project and has lead to some changes in the objectives planned in the beginning

The milestones planned align directly with the objectives of this project and only exist dependencies requiring the input processing be completed before anything else and that the display of the results be completed for the project to be considered complete. Every other milestone could be reached simultaneously, since they do not share any dependencies and could be developed at the same time

4 Development

In this section we'll review the development of the different algorithms implemented along the duration of this project, the tools used and the functionality of each of the algorithms implemented.

4.1 Development environment used

This project has been developed on a Windows environment, using Python with VTK bindings for its display, Matlab 2013b and C++ in a Visual Studio 2013 environment, using the libraries VTK 7.0, CGAL 4.3 for testing purposes and Clipper for C++.

4.2 Input processing

As input for the vertex propagation algorithm, we use slices of a 3d mesh, sliced in runtime in Python, which then are read by our algorithm and shortly processed.

For the edge propagation algorithm we use binary gif images, generously offered by our tutor Debora Gil.

These images are fed to a matlab script which, using "contour following" function returns a list of the vertices that form the contour of the polygon in the image, which are saved in a txt file prepared for being read afterwards. This form of input, of course, brings the discretization errors that come with using 2d images as input, resulting on what could be seen as straight edges, being interpreted as jagged.

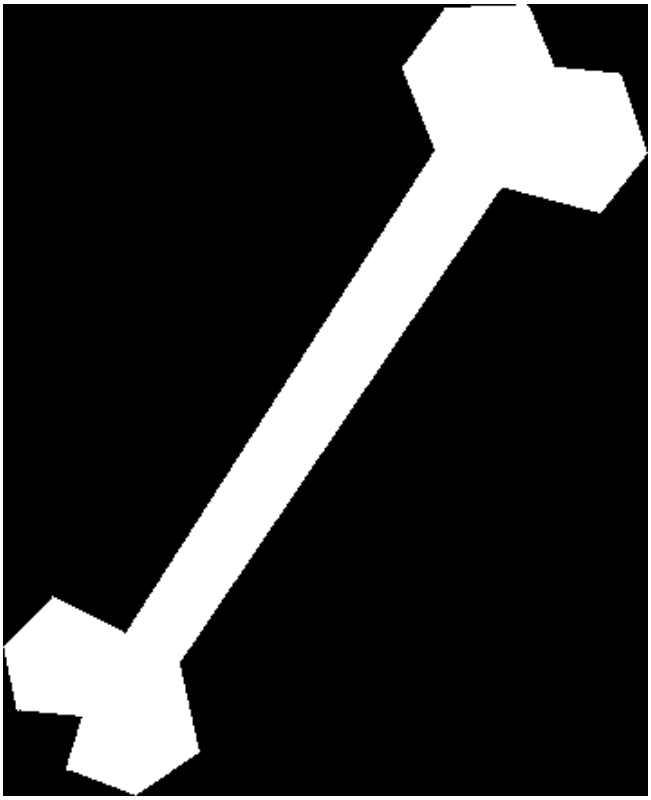

Fig. 6. Test image Bone-1.gif, binary image, containing 1 or 0 according wether that point is solid or not, respectively

The images, when processed are scaled at 100 times the original size, to allow for better precision results when used as input in the propagation algorithms.

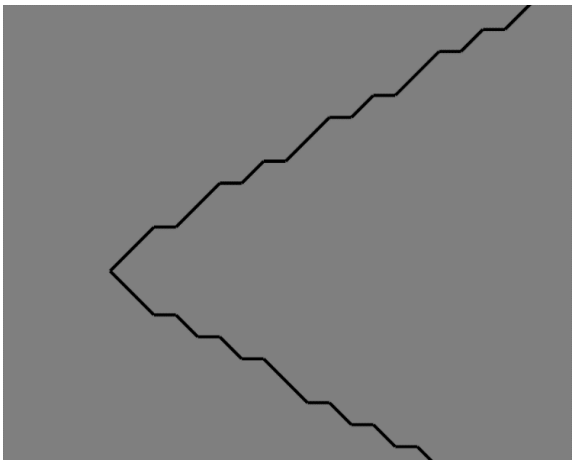

Fig. 7 Interior of the "mouth" of a Pacman used as input image. The edge is not a straight line, but jagged edges, coming from the step of reading from an image.

4.3 Vertex Propagation Algorithm

Our vertex propagation algorithm is programmed on a collection of python scripts, implementing the Edge Propagation algorithm as explained on section 2.1.1 using the VTK library to visualize the results.

The algorithm is divided in 2 phases:

1. In the first phase we initialize the structures we are going to use for our vertex propagation, calculating the angle for every vertex to propagate in, the destination point given the offsetting distance and we calculate the starting intersections between all the starting vertices, which we put in a queue sorted by the distance to the origin at which they happen.

2. We process the event on top of the queue, which is the closest event to the wall for each of the vertices involved, otherwise it could be an event superseded for another one closer to the origin. We then add the new vertices that result from this event to our list of vertices and for each of these new vertices calculate all the intersections with the old vertices. We repeat this phase until either all vertices can reach their destination without intersections or all have collapsed due to events.

The implementation of this algorithm was abandoned before completion due to:

- The complexity of calculating Split events, as they require that for every propagation of a vertex you update the position of the rest in case that a Split event may occur. These calculations proved to be very complex to implement with the due time and as so were never implemented, leading to discarding this type of algorithm altogether.

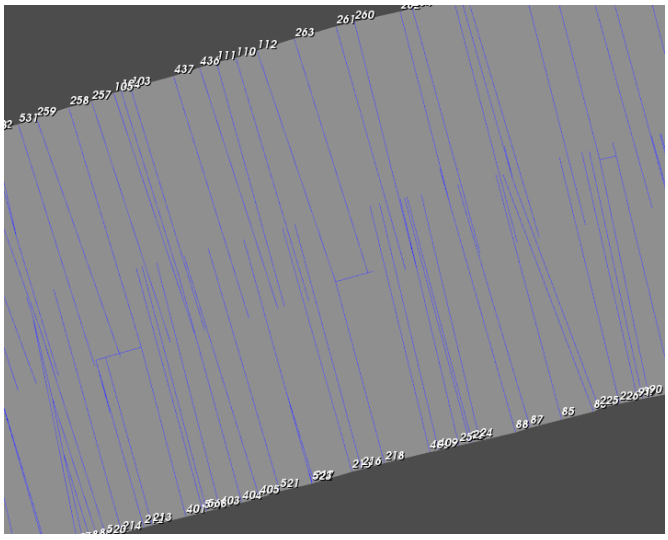

Fig. 8 Vertices when translated should create split events, defining a part of the skeleton through the center of the polygon

- Errors when detecting the intersections due to precision errors due to floating point math, where closing events that should be detected, are not, leading to artifacts like the ones shown in the figure below.

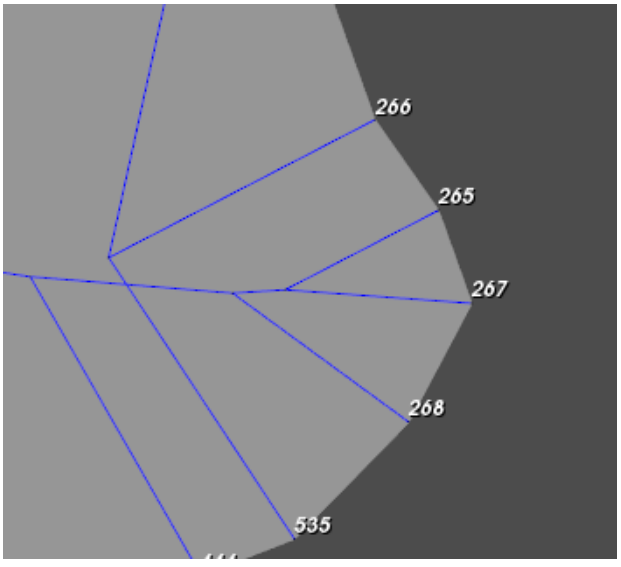

Fig. 9 The intersection between vertices 535 and 266 should not happen, since 535 intersects first with the propagation of the vertex coming from the intersection of 268

- Execution time is proportional to the amount of vertices, edges and collisions between them, growing at rates, in worst case collisions = vertices^2, not including Split events, that as mentioned before, add the added cost of requiring updating the state of all vertices when processing every event.

Given this, implementation of this algorithm was halted, with partial results, in favor of a simpler edge propagation algorithm.

4.4 Final Offsetting Algorithm

Our chosen offsetting algorithm is programmed on C++ and compiled with VC12 (Visual Studio 2013), implementing Clipper's polygon offsetting function [9] It also counts with an alternative mode using Vertex Propagation as implemented in the CGAL library, but the runtimes can be up to 10 minutes in this mode due to not having been optimized, so it's only been used for testing purposes and it's not recommended for use.

This algortihm offsets a given polygon, returning a contracted or expanded polygon, using Clipper's offsetting technique based on Minkowski sums, allowing for offsetting of simple polygons quickly with accurate results without having to calculate the polygon's straight skeleton or propagating the edges.

The algorithm begins by reading the input data already preprocessed by the input processing matlab script. This implementation reads the input polygon in integer precision, since Clipper uses it for its calculations to avoid errors of floating point math so all the values are upscaled to maintain precision and to allow lower step ranges.

It then initializes a set of objects with the input data so they can be treated in a common way by different implementations and the visualization algorithm.

Then, the data is translated to the structures needed for the implementation being used and the implementation returns a new polygon offset according to the parameters given, which goes through the same translation process in reverse, to be then visualised.

It receives a polygon, a step distance and a goal distance and returns a list of offset polygons at every step distance until the goal distance is reached or until the next step returns an empty polygon, meaning that it could not be completed, due to the polygons having collapsed onto a single point.

4.5 Property Propagation Algorithm

This algorithm received the original polygon and several levels of offsetting and propagates properties assigned to the vertices of every polygon to the next offset level.

Since the input images being used have not properties assigned to the vertices, a first step done is assigning properties to every vertex, in our case, we use color as property since it's easy to reflect on the image results, but other properties could be used in it's place, such as height of the vertex, material composition, etc.

Then, having received all the offsetting levels and assigned random properties to the original, it begins transferring the properties from one level to the next.

To transfer the properties, the algorithm establishes relations between the vertices of one level to the next, assigning every vertex on a level a "heir" vertex on the next level, being a vertex's heir, the closest vertex on the next level, until all vertices are assigned a heir.

Then every vertex on the next level calculates the value of its properties given the properties of the vertices that have it as "heir", calculating the median between them and assigning it as its property.

Given this, there's a special case that must be treated, that occurs when a set of vertices on a level wouldn't have "heirs" on the next due to them having collapsed on a single point during the offsetting process. This is controlled by not assigning a vertex an "heir" if the closest vertex on the next level is outside of a distance range, since then it would assign as "heir" a vertex unrelated to it.

The relations between vertices calculated during this process will be more similar to the structure of the straight skeleton depending on the size of the step in the offsetting process, given an infinitely small step, or a step that made every offsetting level coincide with every event of the straight skeleton would return the straight skeleton exactle.

But for our purposes, this propagation of properties is accurate for every level, allowing for the interpolation of the values in every point of the edges of every offsettinglevel.

4.6 Visualization Algorithm

Our visualization algorithm is based on the VTK library [7] and is shared between all the algorithms implemented, with different implementation adapting to the bindings of the programming language used and using a common object structure according to the implementation of the VTK display pipeline.

Every offsetting level is treated as a vtkPolydata, which are then loaded onto the vtkRenderer object as separate vtkActor objects, allowing for better performance when displayed and being able to select which levels to be displayed separately.

This visualization algorithm has 2 modes of usage:

- A screentaking mode, where the program takes a high resolution screen capture of the results.
- An interactive render mode, where the program creates a window where you can navigate the end results and visualize them natively using VTK's interactive render methods.

5 RESULTS

This project results consist of 2 programs and the scripts used for the treatment of the input images, implementing different techniques of polygon offsetting and implementing an algorithm of property propagation for given polygons.

The first one, vertex_propagation, consists on a series of python 2 scripts, requiring importing the VTK bindings:

- Classes_polyreduction: script containing the code related to the different classes created for the project, such as vertices, cycles (given name to vertex due to propagate), etc…
- Simplegeometry: small library containing a collection of geometric functions, used in the algorithm and collected for ease of use.
- Simplevtk: small library containing functions implementing the vtk library, collected on a single file for ease of use and to keep most vtk related scripts separated from the rest of the logic.
- Testangle: small testing script, used for testing angle operations, such as calculating the angle between 2 line segments, sharing a common vertex or not, containing also its own visualization code.
- Testreduction: main script, implementing the vertex propagation algorithm logic. Point of entry of all the code and can be executed directly from console without parameters of entry.
- Testreductionaux: auxiliary script, containing individual functions used for displaying test results in different manners and grouping some extra visualization scripts, mainly for testing purposes.

This program's development was stopped during the duration of this project and as such, it only results partial results, not having implemented the more complex usage cases and is presented for completion purposes.

The second one, Polygon_offsetting, consists of a Visual Studio 2013 C++ solution and a compiled executable, having as dependencies VTK minimum 7.0 (not included in the code) for C++, CGAL 4.3 (not included in the code) and the Clipper library. It includes a small file explaining the dependencies installation process, for ease of use.

The code is split between a headers and source code, each containing the corresponding part of each file:

- CGAL_Offsetting: C++ code, containing the CGAL implementation of the offsetting functions, added for completion purposes, since runtimes are extremely long and it's not preferred for use.
- Clipper: Clipper mathematic library, contained in a single file to eliminate a dependency and package the project's code with the library already prepared.
- ContourExtract: C++ code, containing a first implementation contour extraction tool, based on VTK packaged with the code for completion purposes. Later substituted by a Matlab script.
- Display: File containing the rendering functions, isolating all vtk dependencies to this file.
- FileReader: C++ code for the treatment of the input files and translation to the common object structures used in the main code.
- Geometry_functions: C++ code, similar to its counterpart in the python program, containing geometric functions, recopilated in a single file for ease of use and for standardization of their implementation independently to library used.
- Main: Entry point for the code, calling the other code files and starting the execution. At the start there's fields to configure the program before compilation.
- Property_Management: Code related to the management of the properties assigned to the polygons and to the propagation algorithms.
- Test_imports: test code for testing that the dependencies are installed correctly and the solution is configured properly.
- TFG_Classes: Code containing the classes used in this code, common regardless of library being used.

This code contains the main implementation used and returns image files with the results, a VTK window with an interactive render with the results and implementations of polygon offsetting for both the CGAL library and Clipper, though usage of Clipper is preferred.

Library used and type of output is defined on compilation, using the flags defined at the start of the Main file.

Included, is also, the matlab scripts used for the treatment of the input images:

- Image_Processer_v2: small matlab script, implementing the contour detection script and prepared for the treatment of batches of numbered images at a time.
- Contour_following: matlab script, implementing a contour detection script, returning a list of vertices, ordered in clockwise fashion.

In the appendix are included small snippets of result images, since imagesize floats around 10.000 pixels * 10.000 pixels, to be able to represent small step distances and to allow for more detailed results when saved to an image format.

With the code developed, we have reached the planned objectives and allows us to reach conclusions, regarding the questions posed in this project's premise.

6 Conclusions

In this project, different polygon offsetting algorithms have been implemented, with the aim of propagating properties across an original polygon to an objective polygon according to the established objectives, for what could be a theoretical use in additive manufacturing as established.

As result, I've obtained a fully completed implementation of a polygon offsetting algorithm while also implementing, albeit partially different solutions for the problem posed, we've also implemented a property propagation algorithm, based on the use of the Clipper library and the use of Boolean images as input and also gained a better understanding of the polygon offsetting and property propagation problems, their current implementations in a 2 dimensional space and which techniques excel in which conditions, allowing us to reach conclusions about the nature of these problems and how a future implementation for 3d meshes could work:.

6.1 Vertex Propagation Algorithms

- Prone to arithmetic errors, such as those caused by floating point arithmetic and can be unreliable if this issues are not dealt with.

- Are derivatives of the calculation of the straight skeleton, returning it as a result if implemented correctly and allow for easier results at distances where polygons would collapse, not needing treatment for these cases.

  In a 3 dimensional environment:

- Less feasible to implement since the number of vertices in a given 3d mesh can be orders of magnitude greater than a 2d polygon and this could lead to much greater running times.

- The treatment of the equivalent of the "events" proposed in a 3d environment is more complex and requires for more complicated treatment.

- Other strategies scale much better both in complexity and in ease of implementation, number of vertices scaling in a greater way between 2d polygons and 3d meshes than number of faces.

6.2 Edge Propagation Algorithms

- More robust against arithmetic errors, since implementations deal with vector operations in a greater mesure than angle calculations compared to Vertex Propagation algorithms.

- Several strategies exist for closing the polygonafter translating the edges, with varying complexities, but with lower complexities than a vertex propagation algorithm.

- Make the propagation of properties more difficult, since they include precision errors, according to the propagation step used.

In a 3 dimensional environment:

- If implemented using the meshes faces as one would the polygon's edges (translating the faces along the normal and then dealing with the intersections or extensions of faces), the strategy is exactly as specified for a 2d polygon.

- Can still be more arithmetically robust than a Vertex Propagation based solution, due to the strategy not dealing with as many angle calculations, since it consists mostly of vector/point operations.

- Share the same problem with the 2d polygon algorithm when dealing with property propagation, their accuracy depending on the step size and amount.

6.3 Boolean Operation Algorithms

- Relatively fast computationally, in spite of their apparent complexity.

- Provide robust results in a great array of cases, adapting better to fringe cases, such as having several polygons in the same structure at the same time when offsetting

In a 3 dimensional environment:

- Implementation would be absolutely equal as in 2D in concept, instead of using the Minkowski Sum between 2 polygons, it would use the Minkowski Sum of 2 meshes.

- Complexity would scale with the number of vertices as with 2d polygons, increasing runtime significantly in comparison

6.4 Property Propagation

- Given the straight skeleton of a polygon, the propagation of the properties is trivial, since vertices of the original are related to vertices on any offset polygon according to the skeleton's bones.

- Given an offset level, if the offset step used is precise enough or it coincides with the "events" as defined in **3.1 Vertex Propagation Algorithm,** the propagation is exactly the same as done through the straight skeleton.

In a 3 dimensional environment:

- Obtaining the straight skeleton of a 3D mesh is more costly than in a 2D space, so unless a good algorithm to calculate it is used, the assignation of relations between vertices has to be done using strategies like the one used in this project, trying to approximate the results of the use of the straight skeleton.

- Regarding the calculation of the propagated value, the algorithm is the same and will depend on the kind of value to propagate.

Given this, we consider that the objectives posed have been met in the allotted time.

7 FUTURE WORK

Future work regarding this project could include a full implementation of a Vertex propagation-based solution given the knowledge gained during this project's development, or the implementation of a 3d offsetting algorithm, based on any of the techniques studied, which could be used for additive manufacturing for the propagation of materials defined on the surface of a given object to its interior.

Also future work, could include the study of strategies for polygon offsetting exclusive for 3d meshes and see how they compare against the ones already studied.

ACKNOWLEDGEMENTS

I'd like to thank my tutor during this project, Debora Gil, for the input images and all the help she provided me, Jordi Roca, Jordi Sanroma, Elias Maidanik and Luis Condes for their comments and for lending me their ears during the initial investigation and the following development.

BIBLIOGRAPHY

[1] Fernando Cacciola, A CGAL implementation of the straight skeleton of a simple polygon with holes. Internet: http://fcacciola.50webs.com/CGAL_straight_skeleton.pdf [last accessed: 22/June/2016]

[2] CGAL Open Source Project, CGAL Manual, Chapter 24: 2D Minkowski Sums. Internet : http://i.cs.hku.hk/~wchu/cgal_manual/doc_html/cgal_manual/Minkowski_sum_2/Chapter_main.html [last accessed: 22/June/2016]

[3] [3] Su-Jin Kim, Dong-Yoon Lee and Min-Yang Yang, Offset Triangular Mesh Using the Multiple Normal Vectors of a Vertex. Internet: http://brweb.haltonrc.edu.on.ca/202204/ICE3/V1Nos1to4_33.pdf [last accessed: 22/June/2016]

[4] Franz Aurenhammer, Straight Skeleton of a simple polygon. Internet: http://jeffe.cs.illinois.edu/open/skeleton.html [last accessed: 22/June/2016]

[5] David Eppstein, Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. Internet: http://jeffe.cs.illinois.edu/pubs/cycles.html html [last accessed: 22/June/2016]

[6] Fernando Cacciola, A Survey of Polygon Offsetting Strategies [2003]. Internet: http://fcacciola.50webs.com/Offseting%20Methods.htm [last accessed: 22/June/2016]

[7] VTK Project, VTK 7.0 documentation. Internet: http://www.vtk.org/doc/release/7.0/html/ htm [last accessed: 22/June/2016]

[8] Peter Palfrader and Martin Held, Computing Mitered Offset Curves Based on Straight Skeletons. Internet: https://www.palfrader.org/research/2015/Palfrader,%20Held%20-%20Computing%20Mitered%20Offset%20Curves%20Based%20on%20Straight%20Skeletons,%20CADA%202015%20[PaHe15]%20-%20draft.pdf [last accessed: 22/June/2016]

[9] Angus Johnson, Clipper Library Webpage. Internet: http://www.angusj.com/delphi/clipper.php [last accessed: 22/June/2016]

[10] Angus Johnson, Clipper Library Documentation. Internet: http://www.angusj.com/delphi/clipper/documentation/Docs/Overview/_Body.htm [last accessed: 22/June/2016]

[11] Xiaorui Chen, Sara McMains, Polygon Offsetting by computing winding numbers [2005]. Internet: http://www.me.berkeley.edu/~mcmains/pubs/DAD05OffsetPolygon.pdf [last accessed: 22/June/2016]

[12] Yong Chen, Hongqing Wang, David W Rosen, Jarek Rossignac, A Point-Based Offsetting Method of Polygonal Meshes. Internet: http://www.cc.gatech.edu/~jarek/papers/OffsetYong.pdf [last accessed: 22/June/2016]

[13] Francois Mourougaya, Contour Following. Internet: http://www.mathworks.com/matlabcentral/fileexchange/14947-contour-following [last accessed: 23/June/2016] Etc.

[14] Stefan Huber, Computing Straight Skeleton and Motorcycle Graphs: Theory and Practice. Internet: https://www.sthu.org/research/publications/files/phdthesis.pdf [last accessed: 22/June/2016]

APPENDIX

A1. Result images

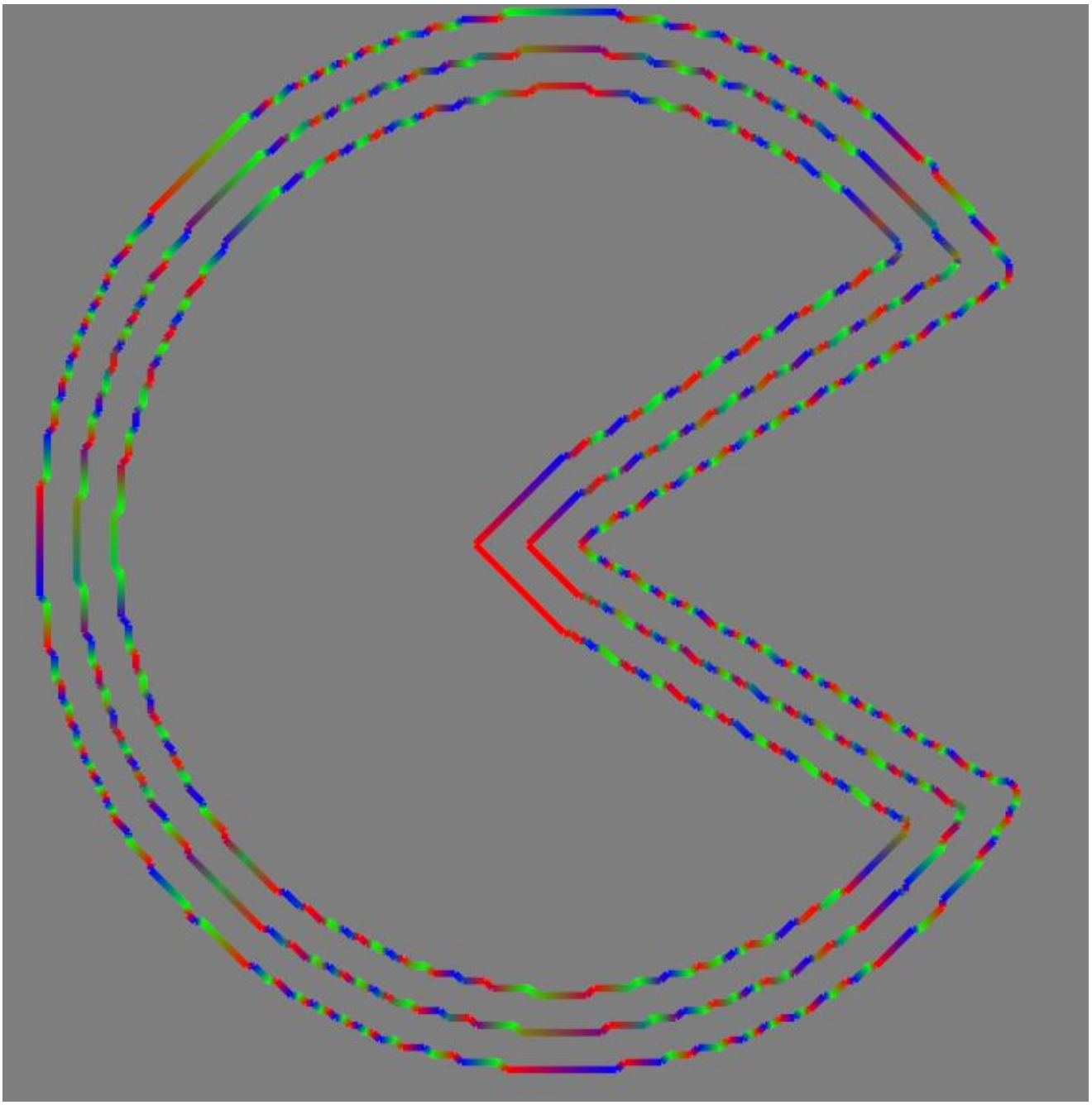

Fig. 10. Pacman image, offset 2 levels propagating the properties along the offset polygons' vertices

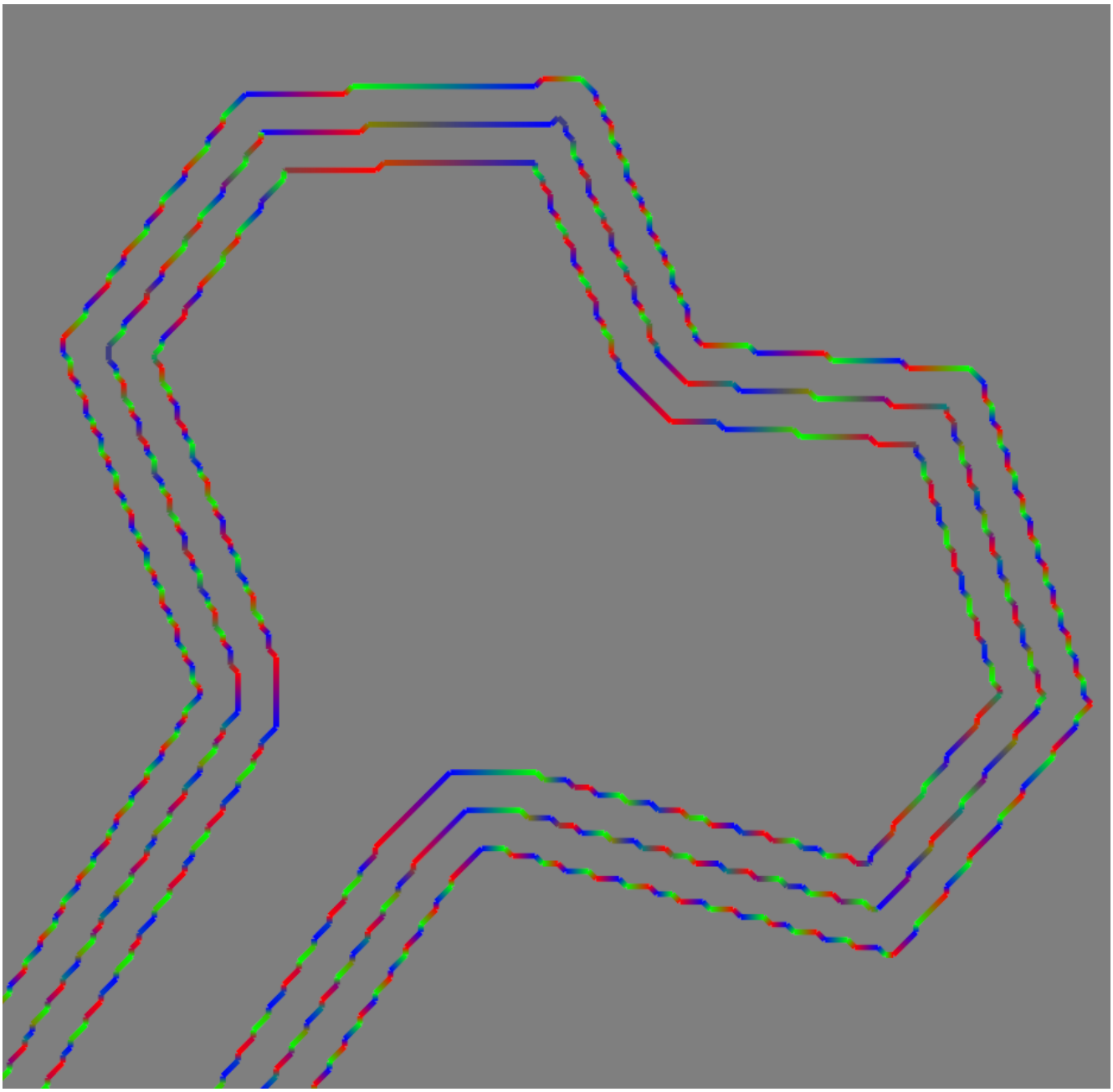

Fig. 11. Detail of Bone-1.gif, after 2 levels of propagation of properties. To note, the change of tones of the vertices of the interior polygons after the elimination of old vertices.

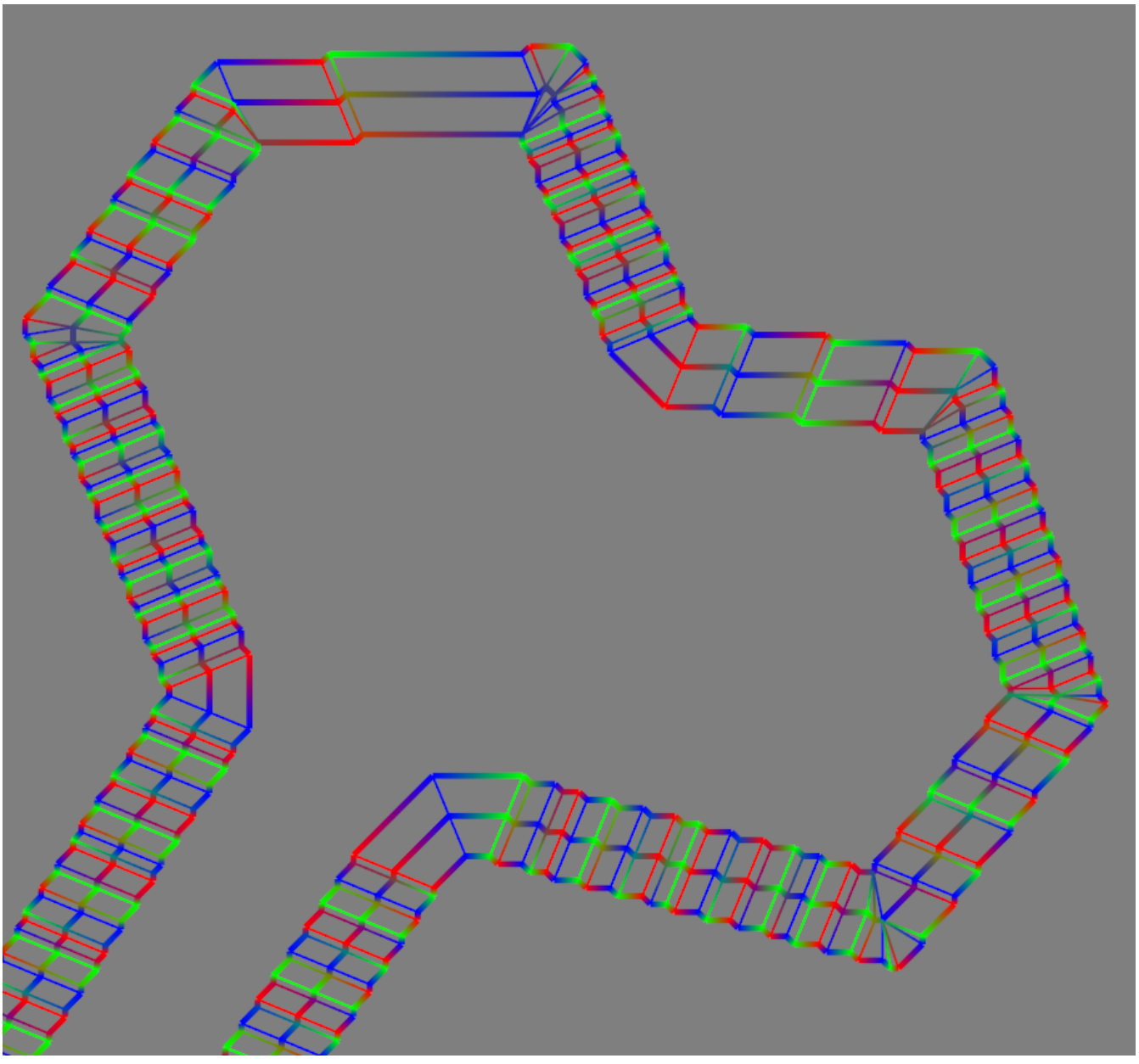

Fig. 12. Figure 11, after having enabled the rendering of segments denoting the relation between vertices established for the propagation of properties