Universidade de Aveiro
2013
Departamento de Eletrónica, Telecomunicações e Informática

Pedro Alexandre Vieira Mesquita

Técnicas de Morphing Para Representação de Objetos Móveis Geográficos

Morphing Techniques For Representation of Geographical Moving Objects

Universidade de Aveiro
2013
Departamento de Eletrónica, Telecomunicações e Informática

Pedro Alexandre Vieira Mesquita

Técnicas de Morphing Para Representação de Objetos Móveis Geográficos

Morphing Techniques For Representation of Geographical Moving Objects

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor José Manuel Matos Moreira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Professor Paulo Miguel de Jesus Dias, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / presidente

Prof. Doutor Joaquim Estrela Ribeiro Silvestre Madeira

Professor Auxiliar da Universidade de Aveiro

Vogais

Prof. Doutor Alexandre Miguel Barbosa Valle de Carvalho

Professor Auxiliar da Universidade do Porto (Arguente Principal)

Prof. Doutor José Manuel Matos Moreira

Professor Auxiliar da Universidade de Aveiro (Orientador)

Prof. Doutor Paulo Miguel de Jesus Dias

Professor Auxiliar da Universidade de Aveiro (Co-orientador)

Agradecimentos /
Acknowledgments

During these six years attending this course, there were many people who marked me and with whom would be impossible to reach this moment.

Starting with my family who supported and sustained me during these long years.

To all my colleagues who were always there when it was necessary and with whom I shared unforgettable moments from the sleepless nights spent studying to the parties.

To my supervisor José Moreira who always trusted that I would complete the work on time and to my co-supervisor Paulo Dias who was always there sending work when things seemed stuck.

A special thanks to my English teacher and friend Marina Valle who helped correcting part of the English of this work.

palavras-chave

Objetos moveis, técnicas de morphing, aquisição de informação espaciotemporal, segmentação

resumo

Um objeto móvel é uma entidade cuja posição e forma se alteram continuamente ao longo do tempo. Tais objetos existem no mundo real e é possível capturar amostras discretas destes utilizando por exemplo imagens de satélite. Essas capturas representam estados do objeto em diferentes instantes de tempo. Cada amostra discreta é representada por um conjunto de *pixel*s. Para representar o movimento dos objetos é necessário extrair uma representação vetorial dessas capturas e aplicar técnicas de *morphing* para modelar a transformação dos objetos.

Nesta dissertação são apresentados dois métodos de *morphing* para representar o movimento de objetos em bases de dados espácio-temporais. Foram ainda desenvolvidas ferramentas para automatizar o processo de segmentação a partir de sequências de imagens reais (fotos de satélite). Estas ferramentas são um primeiro passo para a criação de conjuntos de dados reais com uma dimensão significativa que possam ser utilizados para testar e validar os algoritmos de representação de objetos móveis em bases de dados. Os trabalhos nesta área têm-se focado na criação de representações de objetos móveis válidas e não consideram aspectos qualitativos como a deformação dos objetos durante as transformações. As experiências têm sido realizadas utilizando apenas dados sintéticos.

keywords Moving objects, Morphing techninques, spatiotemporal data acquisition, segmentation

Abstract

A moving object is an entity whose position and shape are continuously changing over time. Such objects exist in the real world and it is possible to capture discrete samples of them using for example satellite images. Those captures represent the characteristics of the object at different time instants. Since each snapshot is in raster mode, it is necessary to extract a vectorial representation of those captures and to apply morphing techniques to model the transformation of the objects between snapshots.

In this dissertation two morphing methods are used to represent the movement of an object. The development of tools to improve and automate the process of segmentation from sequences of real images (satellite images) was also one of the focuses of this work. These tools are a first step for the generation of real world datasets with significant size that can be used to test and validate the algorithms to represent moving objects in databases. This is an important issue because previous works have focused on creating valid movement data representations at all times and do not consider qualitative features such as the objects' deformation during the transformations. The experiments were limited to synthetic datasets.

I. Content

II. List of Pictures

III. List of Tables

IV. List of Graphics

V. Acronyms

Item/Term	Description
MOST	*Moving Objects Spatio-Temporal*
ADT	*Abstract Data Types*
MLPQ	*Management of Linear Programming Queries*
GIS	*Geographic Information System*
PReSTO	*Parametric Rectangle Spatio-Temporal Object*
DBMS	*Database Management System*
MBR	*Minimum Bounding Rectangle*
VCP	*Vertex Correspondence Problem*
VPP	*Vertex Path Problem*
ICP	*Iterative Closest Point*
SVD	*Singular Value Decomposition*
PCA	*Principal Component Analysis*
JMAT	*Java MATrix*
VPI	*Vertex Path Intersection*
OBBTree	*Oriented Bounding Box Tree*

1 Introduction

There are many scenarios that originate entities with both temporal and spatial components. Those phenomena can be instigated by natural causes (volcanic eruptions, tsunamis, melting glaciers, movement of hare of animals, etc.) or by man-made actions (oil spills, deforestation, fires, growth of the urban areas, etc.).

Those scenarios create objects which position and shape are continuously changing over time, named moving objects. A more detailed definition of moving objects is present in Chapter 2.

In the past years some research studies have been made to deal with this kind of objects. Those studies led to the implementation of databases and information systems dealing with sequences of snapshots of moving object captured for example by satellites.

However the acquisition of the geometries of the object from a sequence of real images and the simulation of the movement are still understudied subjects, since it is necessary to obtain the movement of the objects using static snapshots. (Tøssebro and Güting 2001) presents a solution to retrieve moving objects from a set of polygons, however it does not guarantees that the geometry of the objects is valid at all times and in (McKenney and Webb 2010) another solution is proposed to solve that problem. However this work focuses in the avoidance of self-intersections on the intermediate polygons and creates deformations, mostly when rotations and concavities are present in the objects. Both these solutions were tested using only synthetic datasets but no experiments were performed to test the quality of the movement data representation.

More recently, (Paulo 2012) proposes the application of morphing techniques to represent moving objects in spatiotemporal databases. This is an exploratory work that implemented all the process of retrieving moving objects from a sequence of images and storing them in a database. This work divided the morphing into two phases: the correspondences between two consecutive geometric representations of the moving objects, and the linear interpolation between the correspondent vertices. This work was one of the few that used real data source to test the

algorithms and the results proved the feasibility of morphing techniques to solve some complex problems of spatiotemporal objects.

However there were some issues to be solved to guarantee more accurate results. After the acquisition of the geometric information of the object from the images, the segmented polygons suffer from noise or self-intersection. In Chapter 3 of this dissertation we propose a solution for this problem and a method to automatize the segmentation process of a sequence of images, which tries to reuse parameters used in a successful segmentation of an image for the acquisition of the rest of the polygons in that sequence.

In (Paulo 2012) a morphing technique is presented that can be used automatically, without any user intervention, however one conclusion of that work was that this algorithm could be improved if a first correspondence between two vertices in a source and a target polygons was entered manually. Chapter 4 presents solutions to automatically determine this first correspondence and some refinements to improve the morphing algorithm. An alternative algorithm to create the correspondences between polygons is also described in this chapter.

The third problem addressed in this work is related to the path of the vertices resulting of the morphing techniques. In (Paulo 2012) this problem was not solved, since only linear interpolation was used which may originate invalid topologies during the transformation of the objects. These invalid topologies occur because the path of the vertices may intersect originating deformations or self-intersections on the intermediate polygons. Solutions to this problem are addressed in Chapter 5.

The chapters presented in this dissertation start by a general description of the problem referring the solutions in the literature, followed by a description of the solutions implemented in (Paulo 2012). These solutions are discussed in detail and their limitations are highlighted. Then the solutions implemented in this work are presented. Each chapter ends with a summary referring their main topics.

Chapter 6 presents the results obtained using real world datasets for all the implemented algorithms. This chapter is divided in three subsections, referring each of the main problems treated in this dissertation. The first problem is the acquisition of the vectorial representation of the objects from sequences of satellite images. This subsection presents the tests and results of the algorithms implemented to reduce the number of vertices of the polygons and the automatized segmentation method. The second subsection contains the results of the correspondence method implemented comparing them with manual correspondences. The last subsection presents the results of the morphing techniques implemented using the best correspondence algorithm and the alternative methods implemented. In this subsection the similarity between the calculated polygon and the real polygons is computed. Each of these subsections ends with general conclusions of the algorithms.

Finally Chapter 7 presents the contributions of this work as well as some aspects to improve in the future.

2 Representation of Moving objects in databases

The term moving object is an abstraction generally used to identify entities whose position and shape change continuously over the time. There are two different approaches to represent them. When the goal is to analyze the trajectories, the object may be represented just by a point in the space, at each time instant (Shim and Chang 2000). When it is necessary to analyze the full extent of the object, like the shape, the object may have a geometric representation like a polygon defining the boundaries of the object. Considering the detail which the data will be analyzed, those polygons may be the full shape of the object or approximations, such as *Minimum Bounding Rectangles* (MBR) or circles (P. Revesz 2010).

Formally a moving object can be represented as a triple $(\mathcal{I}, \mathcal{S}, \mathcal{V})$ where $\mathcal{I} \in \mathbb{R}_0^+$ represents the time, $\mathcal{S} \in \mathbb{R}^2$ denotes the geometry or location of the object and $\mathcal{V} : \mathbb{R}^2 \times \mathbb{R} \longrightarrow \mathbb{R}^2$ is the transformation function of the object (translation, rotation and deformation) which is continuous both in time and space (Chomicki et al. 2003).

2.1 Data Models

Some strategies have been studied to represent moving objects in databases. In the *Moving Objects Spatio-Temporal (*MOST*)* data model described in (Sistla et al. 1997), a moving object has dynamic and static attributes. The former are values that change in the database without any user update. These values are *motion vectors* that represent the transformation of the object over time. Using these vectors it is possible to retrieve data not only from the present position of an object but also from its near future. When the trajectory of the object changes, it is necessary to update the *motion vector*. The updates are triggered by external events. This model can only be applied to objects that can be represented by a single point that moves in the space. To represent objects with

their geometric shape two alternatives have been studied. The constraint data model and the abstract data type (ADT) model, both using discrete representations of the object.

The constraint data model presented in (Grumbach, Rigaux, and Segoufin 2001) represents a moving object as an infinite set of points resulting from unions and disjunctions of inequations or equations. One example of databases using the constraint model to deal with moving objects is the MLPQ/PReSTO developed at the University of Nebraska-Lincoln and available in ("MLPQ System (Version 5)" 2009). This database merges the MLPQ/GIS (short for Management of Linear Programming Queries / Geographic Information System) and the PReSTO (short for Parametric Rectangle Spatio-Temporal Objects) system.

The MLPQ/GIS is described in (Kanjamala, Peter Z. Revesz, and Y. Wang 1998) and it is an extension of the MLPQ constraint database system defined in (P. Z. Revesz and Li 1997) that uses SQL extended language with linear arithmetic constraints and allows to aggregate operators. The MLPQ/GIS has a user graphical interface to facilitate the creation of database queries, and the translation of the queries is done in an independent module.

The PReSTO system is described on (Cai, Keshwani, and Peter Z. Revesz 2000) and it defines parametric rectangles as a tuple $(x^{[}, x^{]}, y^{[}, y^{]}, from, to)$, where $x^{[}$ and $x^{]}$ refer to the lower and upper bound of x and $y^{[}$ and $y^{]}$ to the lower and upper bound of y during time $t \in [from, to]$. To retrieve the parametric rectangles from a moving object it is used the representation of the object in two different times, the object is decomposed into rectangles, and each rectangle from one representation is paired with the rectangles in the other.

The ADT model described in (Güting et al. 2000) uses *moving points* and *moving regions* to represent a moving object. A single *moving point* is used when only the position of the object is important and represents a point in a plane and its *motion vector*. A *moving region* represents the shape of the object as a collection of *moving points*, where each *moving point* represents a different coordinate of the object.

One important characteristic of the ADT representation is the sliced approach introduced on (Forlizzi et al. 2000), which describes a moving object as a set of *slices*. Each *slice* is a different geometric representation of the object in a different time instant and the changes between consecutive *slices* is translated by simple temporal functions.

The Figure 2.1 represents a moving point. The vertical lines in the left graphic and the horizontal planes in the right represent different slices applied to an object moving on an 1D and a 2D planes, respectively. With these slices it is possible to capture the position of the point at that time instant.

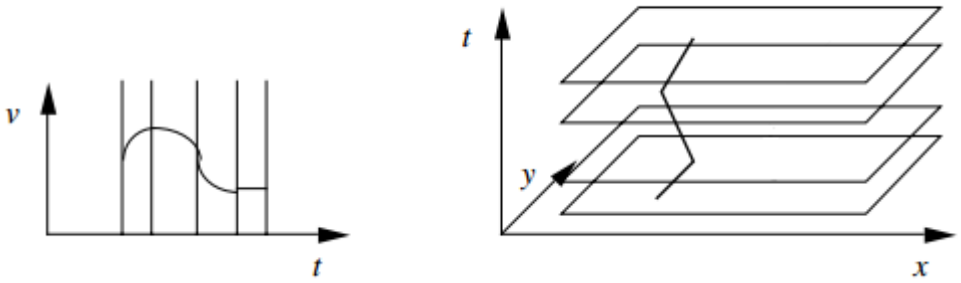


Figure 2.1 - Example of the sliced representation of a moving point, source (Forlizzi et al. 2000)

The Figure 2.2 represents the movement of a polygon between two *slices* of the *sliced representation*, a *motion unit*. In this case the slices represent the real position and shape of an object captured from snapshots and the movement between slices was estimated using linear interpolation.

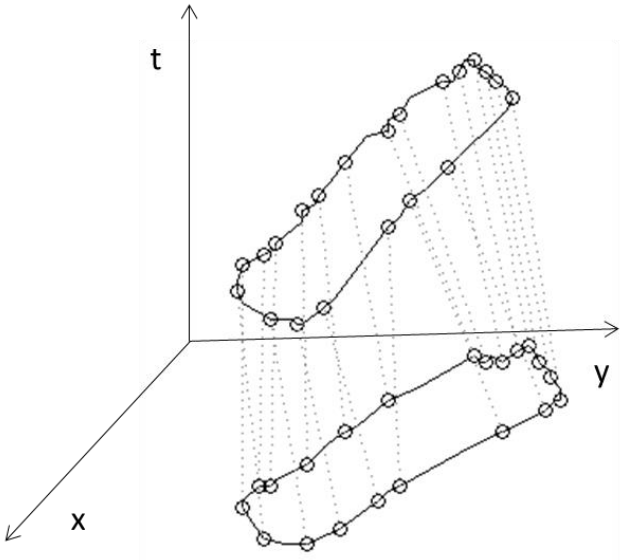


Figure 2.2 – Example of a motion unit of a moving object

This representation allows the creation of a continuous representation of an object even when only discrete samples are available, since the intermediate results can be calculated using the moving function between snapshots.

The ADT's are a combination of data types and operations. Then it is possible to incorporate ADT's into systems that allow the integration of new data types and new operations. The Extensible Database Management System (DBM) is an example of this kind of systems.

The possibility to easily integrate the ADT's into Extensible DBMS turns this model the most used in the research studies and in the implemented databases dealing with moving objects.

The Secondo database (Güting, Behr, and Düntgen 2010) is an extensible DBMS platform created with the intent of teaching and researching and it was developed at University of Hagen in 1995. This is a prototype that enables researchers to test and compare results of different data types, since it is possible to change and create new types or operations. The model and the types mentioned in (Güting et al. 2000) are already implemented in this platform; and tools to create and display animations of the moving objects stored are also available.

The Hermes framework, a research prototype for efficient location-based data management is presented in (Pelekis et al. 2006). This framework aims to help in the modeling, construction and querying of moving objects databases. This framework exploits the spatial data types of the *Oracle 11g* database and the temporal components introduced in (Pelekis 2002). This framework is the only one that allows the representation of arcs and circles. The circles can be represented using *Moving_Polygons* or *Moving_Circle* object type. A *Moving_Circle* only needs three *Moving Points* to define the object (Pelekis et al. 2010).

In (Matos, Moreira, and Carvalho 2012) a discrete database is implemented using *Oracle 11g*. The main types of this database are: the *mPoint*, composed by an unique identifier, a time interval, the position at the beginning of the time interval and a variability function describing the movement of the point during its time interval. The *mPoint* has a unique identifier and the *mRegion* is constituted by a time interval and a set of *mPoint*'s. The operations available in this database are: projections which can return a time interval, a spatial object or a scalar value, e.g., the velocity or the spatial footprint of the object; predicates which return a true or false values such as, results of intersections or unions; and clipping which is the filtering of some part of the object according to some rule.

2.2 Movement data representations

The representation of the objects' movement has been subject for three research studies. In (Tøssebro and Güting 2001) is proposed an algorithm to create correspondences over two consecutive discrete representations of the moving object, that the authors call *rotating plane*. The simplest case is the interpolation between two convex polygons. The *rotating plane* algorithm fixes a segment of a polygon and rotates a plane around this segment until a correspondent vertex or segment is reached in the other polygon. But this algorithm only works using convex polygons. The concave polygons are divided into a tree composed by convex hull of their concavities. The convex hulls of the concavities of two different polygons are matched calculating the overlapped area between them. To the matched concavities is applied the *rotated plane* algorithm.

In (McKenney and Webb 2010) is referred that the *rotating plane* algorithm has a problem since it does not avoid the occurrence of self-intersection in the estimated polygons, so it is proposed another algorithm to do the correspondences of two consecutive discrete representations. This algorithm forces that the vertices are processed in counter-clockwise order starting in the leftmost and lower vertex, which is called by the authors the *cyclic order*. Thus the name *cyclic order algorithm* will be used in future references of this algorithm in this dissertation.

Finally in (Paulo 2012) is proposed the use of morphing techniques to calculate the movement of the objects. In this master thesis the perceptually based approach (Liu et al. 2004) is used to determinate the correspondences between two polygons and the linear interpolation is used to compute the movement of the polygons. This work also proposes a segmentation tool to retrieve the polygons for a sequence of real world images.

2.3 Case studies

Two kinds of polygons are used in (Paulo 2012). The synthetic polygons are geometric shapes like triangles and pentagons created manually and were used to test the implemented methods. The Figure 2.3 shows the synthetics polygons of (Paulo 2012).

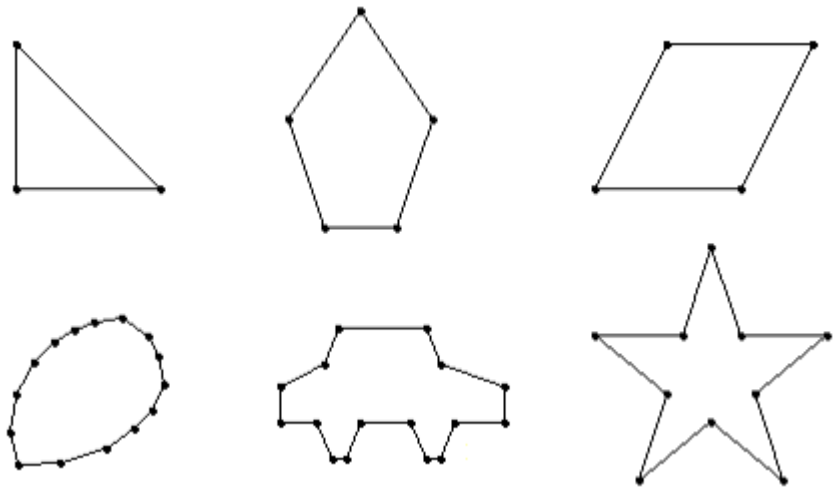


Figure 2.3 - Example of some synthetic polygons used in (Paulo 2012)

The other kind of data used were real world moving objects retrieved from a sequence of 10 satellite images captured by the satellite Terra over the Ross Sea at Antarctica during the last two months of 2004.

Those images are actually available in ("RossSea Subsets" n.d.) and it is possible to observe the movement of the icebergs B15a and B15j during those months. The B15a iceberg is around 120 kilometres long, with an area exceeding 2500 square kilometres and it was the largest free-floating

object for five years until it broke into smaller icebergs. The B15j iceberg is around 45 kilometres long, with an area of almost 650 square kilometres. The movement of the B15a is mainly translational while the B15j is principally rotational. The icebergs belonging to this dataset will be referred using their names with the prefix *ice*. The Figure 2.4 is the satellite picture capture on December 4th of this dataset.

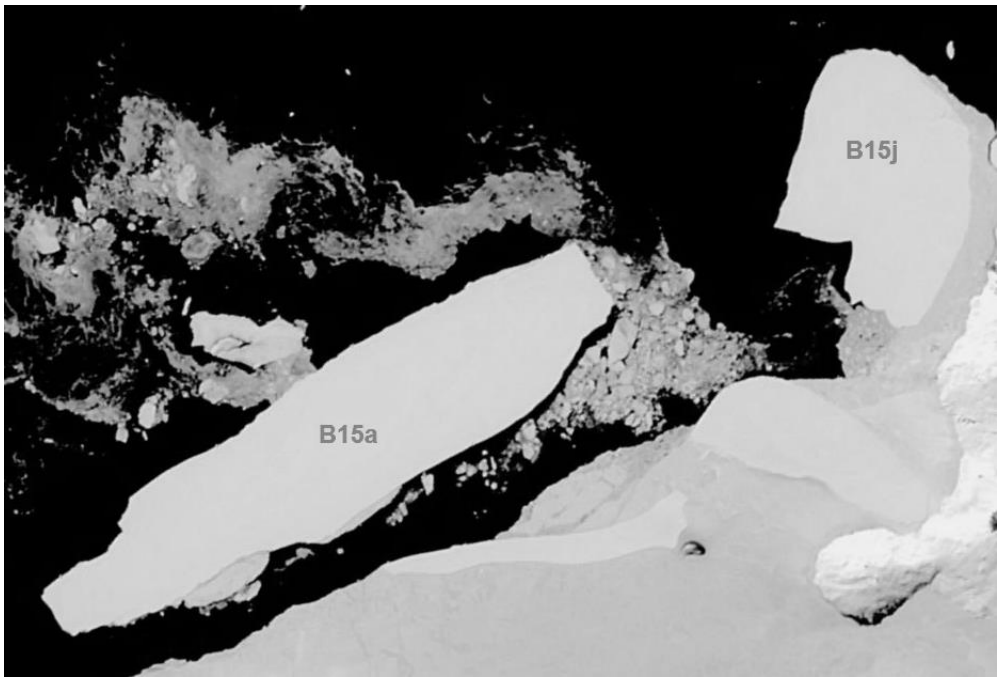


Figure 2.4 - Image of the icebergs B15a and B15j in 04-12-2004

In this thesis two more datasets were used. A video composed by 7 satellite images captured by the *Envisat* satellite over the Atka Bay during October of 2009. In those pictures it is possible to observe the movement of the C19c iceberg. This iceberg is 37 by 24 kilometres and it was a part of the iceberg C19 which calved from the Ross Ice Shelf on May 2002. The movement of this iceberg is mainly rotational with some translation. The movie with this iceberg is actually available in (Münster and Wesche 2009). Figure 2.5 is one frame of the movie capturing the scene on November 11th 2009.

Figure 2.5 - Image of the C19c iceberg at 24-11-2009

The last iceberg dataset was retrieved from the a low resolution video available in ("Ross movie" 2007). This movie is 30 seconds long and depicts the evolution of the iceberg B15 from its formation in 14^{th} April of 2000 until 10^{th} October of the same year. In this video it is possible to observe the split of this iceberg into smaller pieces like the B15a, B15b and B15d. Originally the B15 iceberg was the nearly 300 kilometres length and 40 kilometres width, and it was one of the largest ever seen at the time of its formation in March 2000.

The Figure 2.6 shows one frame of the movie with the B15 iceberg captured on April 13th 2000.

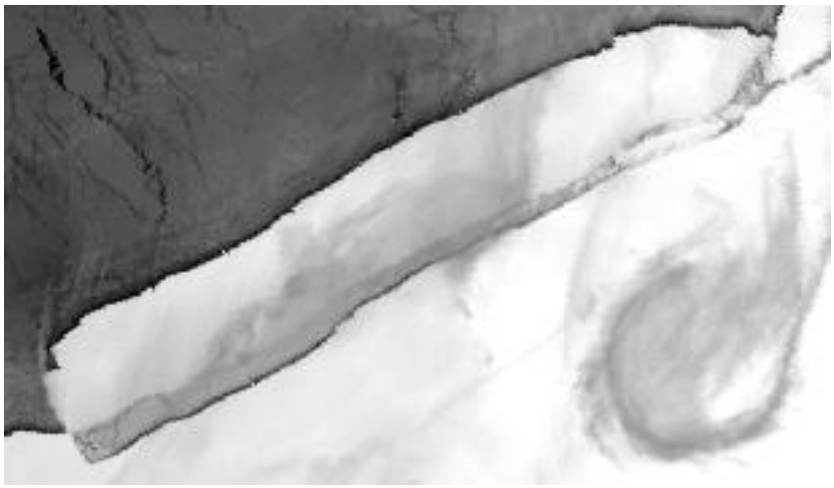


Figure 2.6 - Original B15 iceberg at April 13, 2000

After forming the B15d, the B15a fragment is the origin of the icebergs used in Luis Paulo's dissertation (Paulo 2012), described in this section.

Since this dataset has an iceberg with the same name the dataset used in (Paulo 2012), to refer icebergs from this dataset the prefix *ross* is added to the icebergs name, while the prefix *ice* is used in the other dataset.

The Figure 2.7 shows the fragmentation of the iceberg B15a forming the B15d.

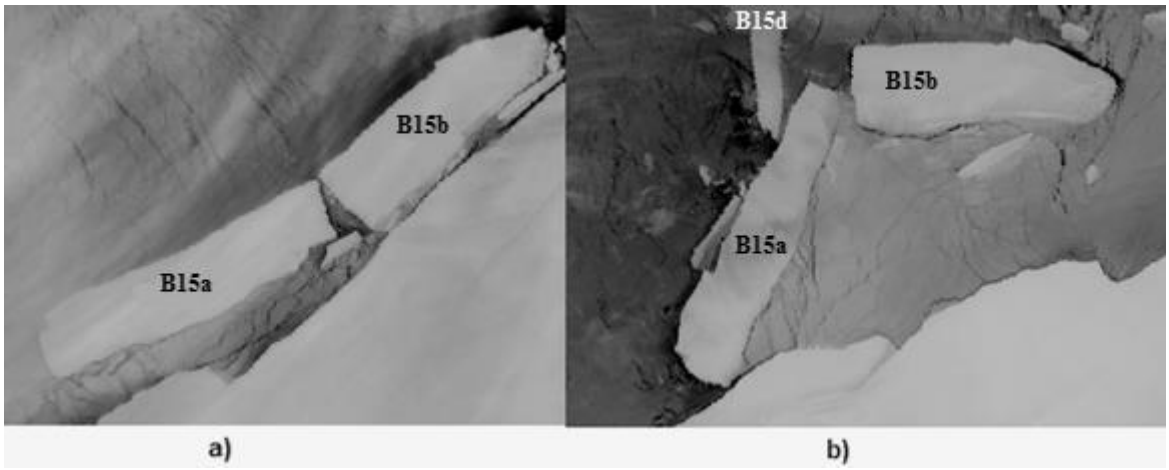


Figure 2.7 – Split of icebergs, a) B15 broke into B15a and B15b at 23-05-2000 and b) B15a formed the B15d at 12-08-2000

The Table 2.1 shows the main properties of the datasets and the icebergs used.

	ross_b15a and ross_b15b	***ice_B15a***	***ice_B15j***	***C19c***
Number of images	30	10	10	8
Time interval	irregular: 1 to 29 days	2-10 days	2-10 days	1-2 days
Movement type	Translation and Rotation (occasional but significant)	Translation	Rotation	dynamic movement: Translation and Rotation are significant

Table 2.1 – Datasets properties

In Table 2.1, the first row indicates the name of icebergs used; the second row gives the number of images in the sequence; the third row indicates the average time interval between observations and the last row indicates the predominant type of movement of the icebergs.

Some additional tests were done using datasets containing fire spread sequences. The datasets description and some preliminary results are presented in Annex A.

2.4 Summary

This chapter gives the context of this work and puts in evidence some of the current challenges to create data representations for moving object databases. It presents the description of moving objects as well as the main data models used to represent the moving objects: the ADT and the constraints databases approaches.

Some examples of systems using those data models are presented like the MLPQ/PReSTO, a constraint database and the *Secondo* a database that uses Extensible DBMS integrated with an ADT of moving objects.

The problem covered in this work is the creation of data representations for moving object databases. This chapter presents three works addressing this problem. The main challenges are: the implementation of semi-automatic procedures to extract the polygons representing the shape of the moving objects in a sequence of snapshots; to define a correspondence between the vertices of the polygons extracted from consecutive snapshots; and to define the vertex paths to model the transformation of the moving objects. It also presents the datasets used in the remainder of this work.

3 Extracting objects from images

In the datasets described in Section 2.3, each snapshot contains the objects represented in raster mode, i.e. sets of pixels. However the data models from Section 2.1 use the vector representation of those objects. To retrieve the vector representation of an object from its raster representation it is necessary to apply a segmentation algorithm.

3.1 Segmentation

The simplest approach to obtain the shape of an object from a scene is manually defining the contours of each object for each scene. The result is directly related with the user's ability to define the boundaries of the objects. Other option is to apply algorithms of segmentation on scenes so that the user could more easily get the shapes of the objects. An example of such an algorithm is the *AB-Snake* described in (Andrey and Boudier 2006).

The AB-*Snake* algorithm uses a starting point to define the point from where the snake will start the search for the edges of an object. In the Figure 3.1 the start point is the centre of the ellipse in the top left image. The rest of the images demonstrate the growing of the *snake* in the search for the edges.

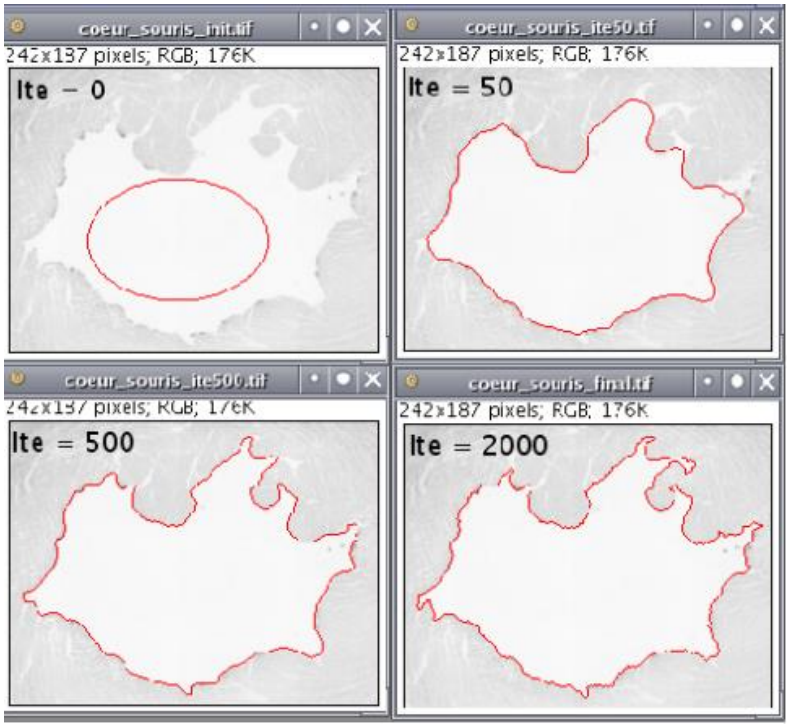


Figure 3.1 – Example of the AB-Snake algorithm segmentation, source (Andrey and Boudier 2006)

The different colours and noise on the scenes difficult the definition of the edges. To improve the results of those algorithms, image simplification can be used as a processing step previous to the segmentation. Thresholding algorithms, e.g., *Otsu Thresholding* algorithm, can simplify the pixel values of the images using a threshold and a histogram. The *Otsu Thresholding* converts a grey scaled image into a binary image, dividing the grey scaled pixels of an image into background and foreground and determining the variance of each one.

Figure 3.2 depicts the image (in centre) resulting from the application of the *Otsu Thresholding* algorithm on the image on the left. The image in the right show the maximum and minimum values used in the simplification.

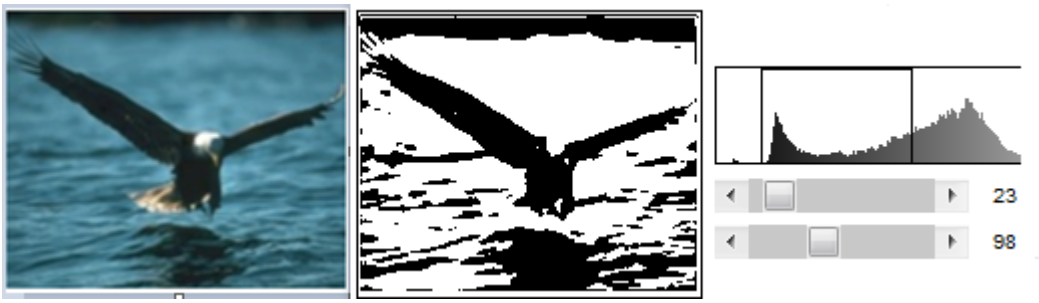


Figure 3.2 – Example of the of Otsu Thresholding algorithm

A more complex solution to simplify the image is presented in (Adamek, O'Connor, and Murphy 2005), where the image is divided into various sections of different colors. The sections are defined mapping the image into a weighted graph, where the links represent the cost of merging different regions. The regions linked with the minimum cost are merged and the algorithm continues until a minimum cost is reached and the image is divided into simple sections. Figure 3.3 shows the results of this algorithm (image on right) applied to the image on the left.

Figure 3.3 - Example of the algorithm (Adamek, O'Connor, and Murphy 2005), source (Adamek, O'Connor, and Murphy 2005)

(Paulo 2012) used the application described in (Oliveira 2011) to retrieve polygons from images. This application was implemented in JAVA using the *ImageJ* library. With this application it is possible to open a sequence of images and obtain a polygon per image of the sequence. There are two options to segment an image, manually, and automatically. With the manual method the user has to select each vertex of the polygons. The automatic option uses the *AB-Snake* algorithm of the *ImageJ* library combined with a thresholding algorithm that filters the pixels values by a threshold selected by the user. The *ImageJ* library has available various thresholding algorithms enumerated in (Landini 2013), and a user can opt which one to use. Two new features were added in (Paulo 2012): the possibility to simplify the number of vertices of the polygons using the Ramer-Douglas-Peucker algorithm (Douglas and Peucker 1973), where the user inputs the threshold value and the option to save the polygons to text files.

In this dissertation a new application was created to perform the segmentation. This application was based in (Paulo 2012) and (Oliveira 2011) with extra tools to improve the polygons and to automatize the process. To improve the polygons a tool to manually adjust the polygons obtained by the segmentation was created. To automatize the process a simplification algorithm based in the Douglas-Peucker algorithm already implemented in (Paulo 2012) and a technique to automatically segment a set of images without the interaction of the user were used.

The Douglas-Peucker algorithm was implemented in (Paulo 2012), and it was created to reduce the number of vertices of a curve, but the same idea can be applicable to polygons. This

algorithm starts with a line connecting two vertices of a polygon and iteratively adds the other vertices in descending order of the perpendicular distance with the segment.

Figure 3.4 shows the perpendicular distance of the point 3 to the segment formed by the point 1 and 2. This distance is presented by the dashed segment.

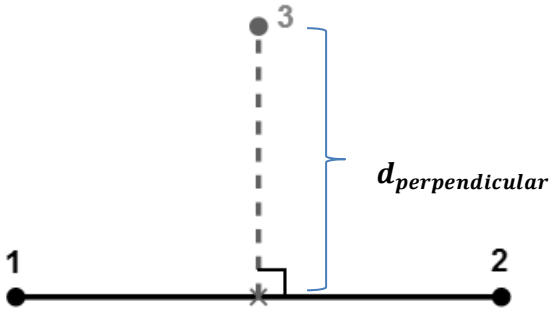


Figure 3.4 - Perpendicular distance of the point 3 to the segment 12, source (Paulo 2012)

The algorithm stops when all the vertices have been added or until the perpendicular distances of all the vertices to be added is greater than a threshold given by the user. The Figure 3.5 shows the results of applying different thresholds on the Douglas-Peucker algorithm to a polyline. In this image it is possible to verify the number of vertices is smaller with higher thresholds.

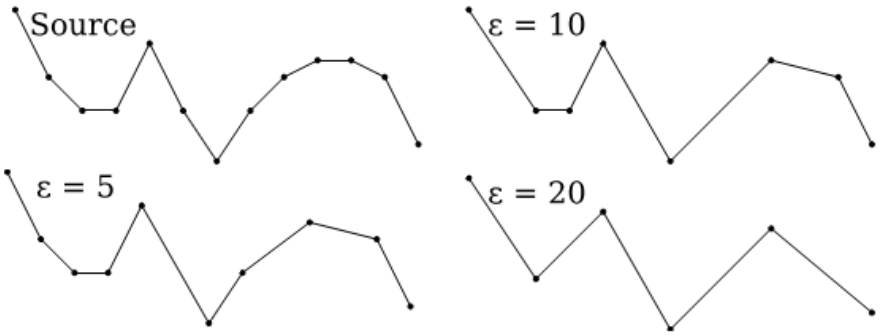


Figure 3.5 - Example of the Douglas-Peucker algorithm with different thresholds, source (Paulo 2012)

3.2 Post Processing

The Douglas-Peucker algorithm implemented in (Paulo 2012) was a tool that allowed the reduction of the vertices of a polygon, turning it simpler. However the user had to choose the threshold to be applied. The method to determinate this value automatically was implemented in this dissertation and it is presented in this section as well as some other tools to adjust the resultant polygons.

3.2.1 Iteratively detect the threshold to the Douglas-Peucker algorithm

The results of the Douglas-Peucker algorithm are limited by the threshold used to simplify the polygons and it may be any real number.

(Zhao, Sheng, and H. Guo 2009) introduces a way to determine the threshold value so the simplified polygon is not too different from the original one. To measure the similarity between the original polygon and the simplified one it is used the formula $Sim = \frac{S'}{S}\prod_{i=1}^{n}(1-\frac{R_i}{S})$, where S is the Minimum Bounding Rectangle (MBR) of the original polygon, S' represents the MBR of the simplified polygon and R_i the MBR of the part surrounded by the edge *i* of the simplified polygon and the correspondent set of edges on the original polygon.

The authors of the original work have used MBR's since they are faster to compute, however in this dissertation the real polygons were used.

Figure 3.6 represents an original polyline in black and in red the same polyline simplified. The yellow areas are the R_i used in this dissertation to calculate the similarity. The threshold *t* was determined iteratively, firstly finding the *t* more similar to the similarity coefficient such that $t \in \mathbb{Z}^{+} and\ t \leq similarity\ coefficient$. The next iteration will test a decimal threshold value. To do that it is added to the previous *t* a new decimal case in each iteration. This is repeated until the similarity coefficient is found or the *t* has 10 decimal cases.

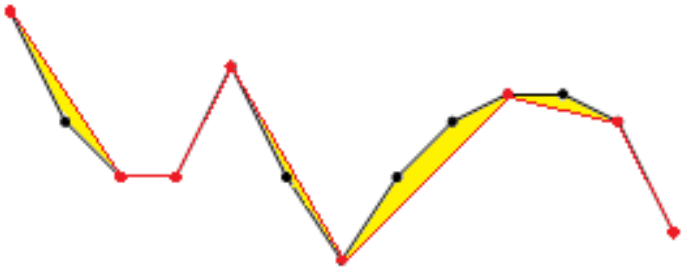


Figure 3.6 – A original and simplified polylines overlapped

3.2.2 Edition

Even after filtering an image it is possible that some noise still remains in the polygon originating deformation. The noise may be smaller fragments of the icebergs or clouds that cover some part of the object as in left image from Figure 3.7.

To solve this problem a tool was implemented to remove sets of vertices from a polygon. This tool removed all the vertices that belonged to a rectangle manually defined by the user. The right image from Figure 3.7 depicts the segmentation result containing noise, in this case a cloud.

The image on the right shows the resulting polygon after the deletion of the vertices limiting the cloud.

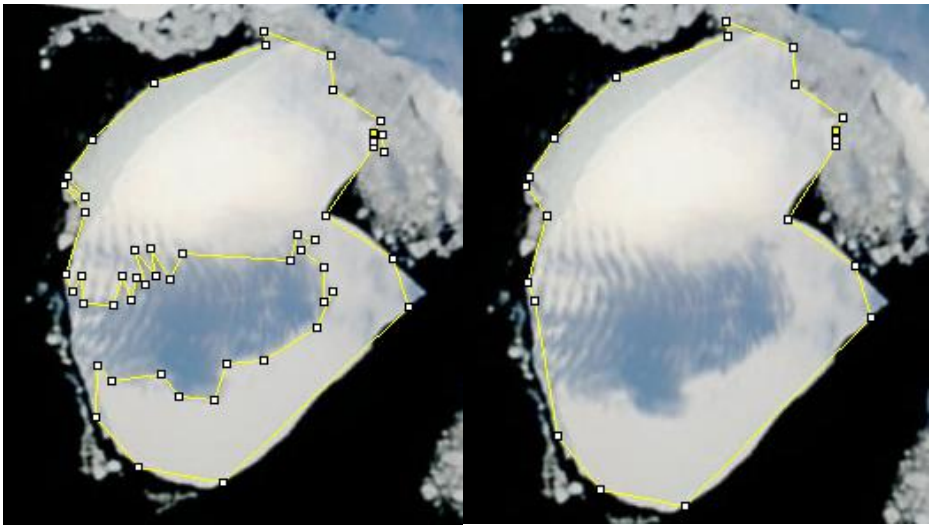


Figure 3.7 - Deformation originated by a cloud and the result obtained only removing vertices

Using the simplification algorithm it is possible that this algorithm removes too many vertices, provoking a loss of detail in the images. To solve this problem two other options were added to the application: add and move vertices. Figure 3.8 shows in the left, a polygon resulting from automatic simplification and where some vertices were misplaced. With the tools to move and add vertices it was possible to adjust the polygon. The resultant polygon is presented in the right image.

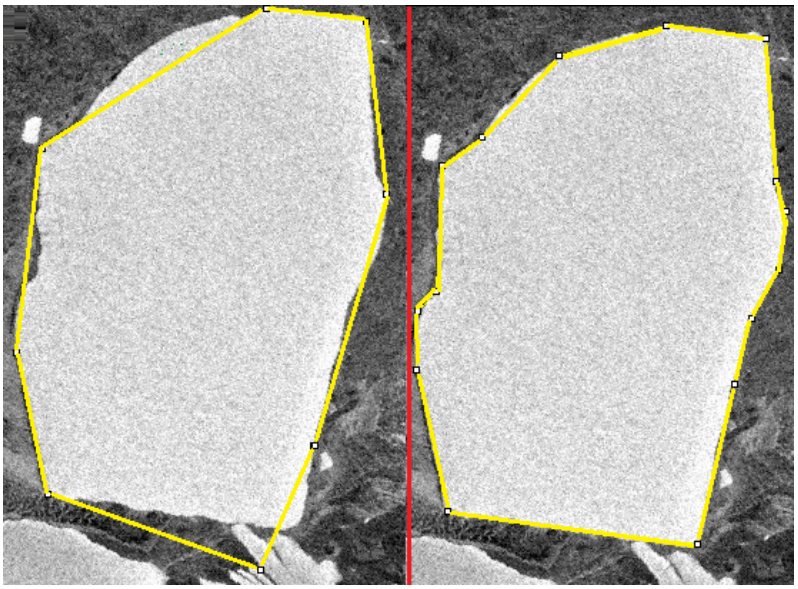


Figure 3.8 Polygon after simplification (left) and after addition and shift of vertices (right)

Besides the edition options implemented, the new segmentation application allows opening a sequence of images that had already been segmented and reedit the polygons if required. So a sequence of images with their polygons may be reopened, the saving of the data must meet some rules. The polygons have a specific file type and the name of this file must contain the identification of the image that originated the polygon and the identification of the polygon in the image. These identifications are usually the order of the images and polygons.

3.3 Semi-Automatic Segmentation

The methods described until now aim to improve the polygons obtained using the AB-*Snake* algorithm. This section describes a method to automate the segmentation process of sequence of images.

The idea of this method is to reuse the values used on the segmentation of one image to all the remaining sequence. In this process the user starts by segmenting one image using the *AB-Snake* algorithm. Then the algorithm will reapply the same values used in the segmentation of this first image to the subsequent images of the sequence. For each subsequent image the resulting polygons will be tested using a similarity algorithm.

The starting point of the *AB-Snake* algorithm is one of the key points shared between different images. Other key points are the maximum and minimum thresholds values used as well as the thresholding algorithm. The last key variable is a flag informing if the polygon was simplified by the user. If one polygon A has this flag active, this polygon was simplified and all the polygons that use the key variables of it will also be simplified. All these values will be the same to all posterior images.

This method will try to find the new polygons in all remaining images until the last image. If some error occurs, the application shows the image and the polygon where the error was detected accompanied by an error message. The user can ignore the error and proceed with the segmentation process, or remove the image from the sequence where the error occurred and continue or stop the semi-automatic segmentation.

An error occurs when the polygon on the previous image is too different from the polygon on the actual image. This difference is calculated by determining the matching distance between the two polygons using the algorithm of (Arkin et al. 1991). In (Arkin et al. 1991) is mentioned that two polygons are similar if their matching distance is smaller than 0.5. This method is detailed in Section 4.1.2 along with its implementation.

There are other options to determinate if two polygons are similar in (Veltkamp and Hagedoorn 2000) are resumed some of those options. The *area of symmetric difference* and the L_p

distances are two examples of metric that can be used to match polygons mentioned in (Veltkamp and Hagedoorn 2000). The *area of symmetric difference* of two shapes A and B is $area((A - B) \cup (B - A))$, that forms the areas of the two shapes not overlapping and the L_p *distances* is defined by the function: $L_p = (\sum_{i=0}^{k} |x_i - y_i|^p)^{1/p}, p \geq 1$, if $p=2$ this formula returns the *Euclidean Distance*.

The algorithm implemented was the method mentioned in (Arkin et al. 1991) since this same method with some adjustments may be used to determine the correspondences between the vertices of the polygons (Zhao, Sheng, and H. Guo 2009). This algorithm does not take into account the scale of the polygons, so it is necessary to compare the areas of the polygons. If the difference between the area of the previous polygon and the new one is greater than a percentage of the area of the previous polygon, an error is also generated.

To find the polygons on the next image the *AB-Snake* algorithm is used with the same threshold applied in the previous image. The starting point of the edge detector algorithm used first the starting point used in the prior image or, in case of error the centre of the polygon on the previous image.

3.4 Summary

This chapter describes the main steps to obtain the vector representation of objects from images in raster mode. It describes the segmentation algorithm used in this work, the *AB-Snake* algorithm together with a simplification algorithm the *Otsu thresholding* algorithm. These algorithms were already used in (Paulo 2012), that added the Douglas-Peucker algorithm to reduce the number of vertices of the resulting polygons. However this algorithm needs a threshold value to define how much the polygons will be simplified. In this dissertation a method to determine this threshold automatically is implemented and presented in Section 3.2.1. In addition to this method, tools to edit the retrieved polygons were implemented (Section 3.2.2). In the end of this chapter a method to automatize all the process of segmenting sets of images is presented (Section 3.3).

The segmentation of images is only a small but important step to represent moving objects. With the conclusion of this step the moving objects have their geometric representation in fixed time instants. After this it is necessary to determinate the morphing between the retrieved instants.

The morphing has two main problems to be solved, the vertex correspondence problem (VCP) and the vertex path problem (VPP). The next two chapters address these two points.

4 Polygons Matching

After obtaining the vector representation of the objects it is necessary to apply some techniques to represent their movements. In (Paulo 2012) morphing techniques were applied to the objects. A morphing technique consists in the solving of two main problems, the vertex correspondence problem (VCP) and the vertex path problem (VPP).

This section will focus in solving the first problem, the VCP.

4.1 Vertex Correspondence Problem

To define the movement of an object and represent it with linear functions it is necessary to find a matching between the vertices from one polygon with the vertices in the other.

For humans, choosing correspondences between vertices of two polygons is a simple and intuitive task, however to find an algorithm to perform this task is not so trivial. This problem is usually known as the VCP.

The Figure 4.1 shows the correspondences between two polygons, S and T. In this example some vertex correspondences are not correct, since the path of the vertices will intersect (red).

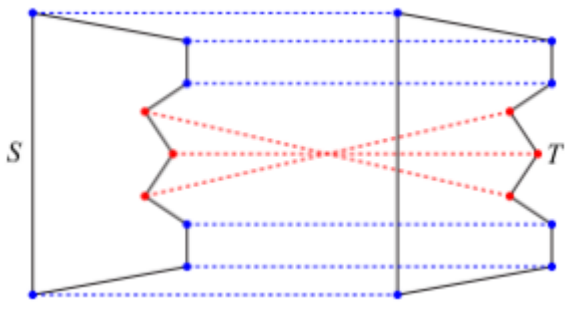


Figure 4.1 – Self-intersection duo to bad correspondences, source (Paulo 2012)

To solve this problem, some solutions have already been proposed. The *Perceptually-based* approach, described in (Liu et al. 2004), finds the vertices that best define the shape of the polygons, determines a descriptor for each of those vertices and computes the correspondences using those descriptors.

Other approach is presented in (Sederberg and Greenwood 1992). In this method the shapes of the objects are considered as set of wires that stretch and bend. The stretch and bend of the wires have costs. The minimum total cost to transform one shape to the other is calculated and it defines the correspondences.

The use of turning functions of the polygons, that are graphical representations of the polygons using their edges length and angles, is proposed in (Zhao, Sheng, and H. Guo 2009). In this case the correspondences are formed finding the minimum difference between the turning functions of two different objects, choosing different referential vertices. A referential vertex is the vertex that defines the beginning of the turning functions.

Figure 4.2 shows a polygonal curve and its respective turning function.

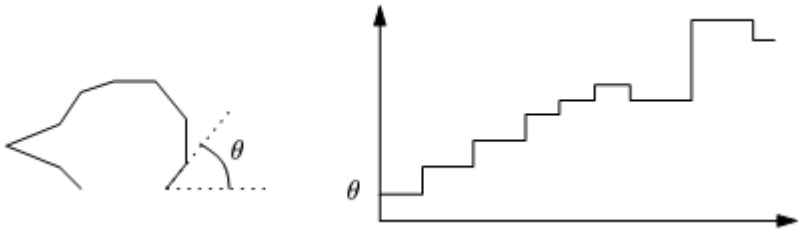


Figure 4.2 - Polygonal curve and respective turning function, source (Veltkamp and Hagedoorn 2000)

The first solution presented in this dissertation is based on (Paulo 2012) where the *Perceptually-based approach* from (Liu et al. 2004) was implemented.

4.1.1 Perceptually based approach

This method uses the option referred on (Chetverikov 2003) to select the points that best define the shape of the polygons, called *feature points.* The first step to determinate the *feature points,* is to search for the candidate points. A vertex p is a candidate if there is a vertex before and other after p, p^- and p^+, respectively, such that the distances between p and p^- and the distance between p and p^+ are inside a pre-defined interval, greater than d_{min} and lower than d_{max}, and the angle formed by these three vertices, α is smaller than a maximum angle. Figure 4.3(a) shows a candidate point p.

The second step is to choose the *feature points* from all the candidates. All candidates belonging to different curves are considered *feature points.* When one curve has more than one candidate point, the candidate with greater sharpness is selected as *feature point* and the rest of the candidates in this curve are discarded. The Figure 4.3(b) shows a curve with two candidate points, and their sharpness is represented as α.

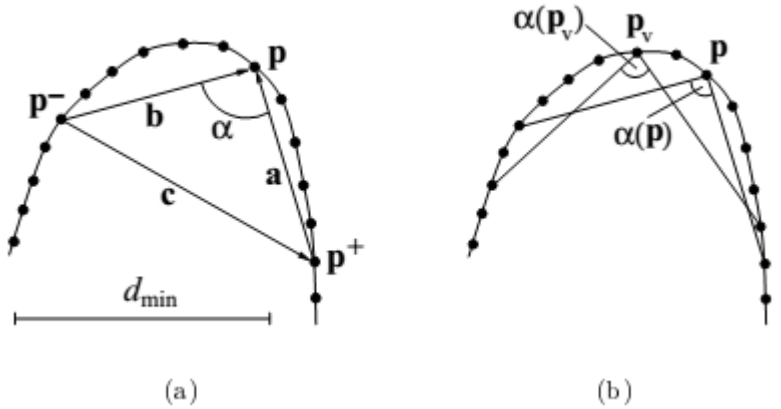


Figure 4.3 – Detecting *feature points*, source (Chetverikov 2003)

After determining the *feature points* the correspondences were calculated following the algorithm described in (Liu et al. 2004). This algorithm uses regions of support to calculate the similarity between *feature points.* A *region of support* is formed by the *feature point* before, p^- and the *feature point* after it, p^+, or if the next and previous *feature points* distances are greater than a maximum, by the vertices at those distances. Each *region of support* contains several descriptors. A descriptor describes the statistical and topological properties of a region using algebraic principles. In the figure above the descriptors of p_v and p are represented by $\propto (p_v)$ and $\propto (p)$, repectively, and $\propto$ represents the angles of the points.

Figure 4.4 depicts a polygon with its *feature points* numbered and with different colours. For each feature point it is possible to observe its *region of support* marked by two dashed arrows with the same colour as its source feature point.

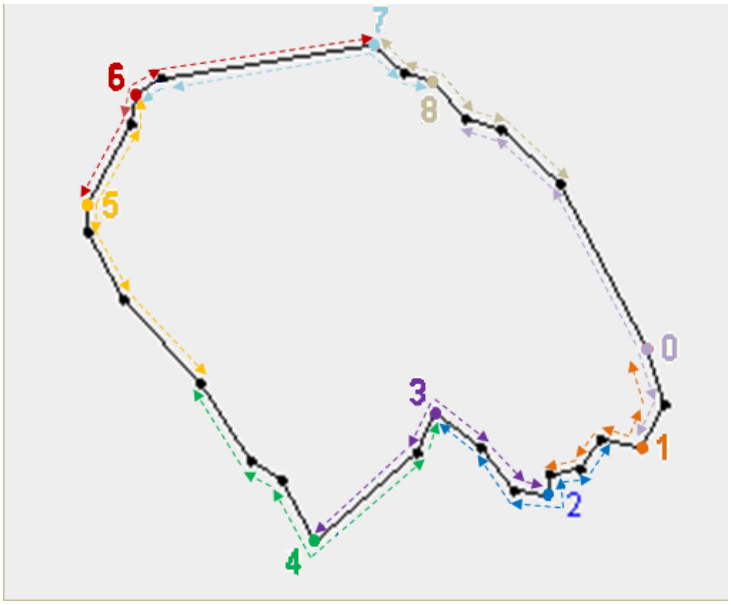


Figure 4.4 – Polygon with its feature points and respective regions of support

The correspondences between *feature points* are calculated using a dynamic programing algorithm based on a similarity function.

The vertices that are not *feature points* are matched using sections of the polygons limited by two corresponding *feature points*. Figure 4.5 shows two sections of two polygons, where one section is limited by the *feature points S(i)* and S*(i+1)* and the other by the *feature points T(i)* and T*(i+1)*. In this same image is possible to observe that the *feature point S(i)* corresponds to the *feature point T(i)* and *S(i+1)* corresponds to the *feature point T(i+1)*. The intermediate vertices are matched using the index of the vertices in each section.

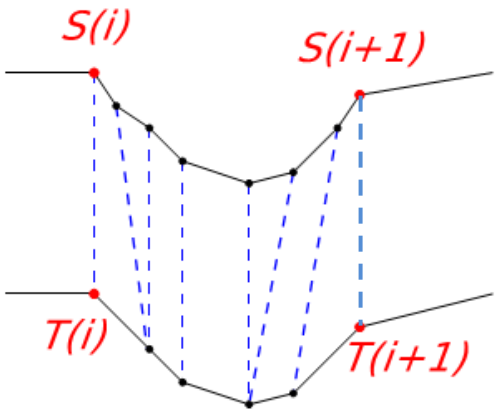


Figure 4.5 - Example of correspondences on a section, source (Paulo 2012)

For example the number of vertices between *S(i)* and *S(i+1)* is 6, while between *T(i)* and T(i+1*)* there are only 4 vertices. If both sections had the same number of vertices then the vertices would be matched according to their indexes one to one. However since this is not the case the section formed by *T(i)* to *T(i+1)* has vertices with more than one corresponding vertex from the section *S(i)* to *S(i+1)* what may cause intersections or deformations. In Section 4.3.2 is presented a solution to solve this problem.

4.1.2 Calculating correspondences using turning functions

The algorithm described in (Zhao, Sheng, and H. Guo 2009) is an alternative to the *perceptually based algorithm*. This algorithm was implemented in this work to compare the results with those obtained using the solution implemented in (Paulo 2012) and, if possible, to merge them to improve the results.

The approach mentioned in (Zhao, Sheng, and H. Guo 2009) uses turning functions to represent the polygons. A turning function of a polygon is defined by the angle of a counter-clockwise tangent and the sum up of the length of the edges starting on some referential vertex.

Figure 4.6 shows a polygon and its respective turning function. The X axis of the turning functions represents the sum of the edges length starting in a referential vertex.

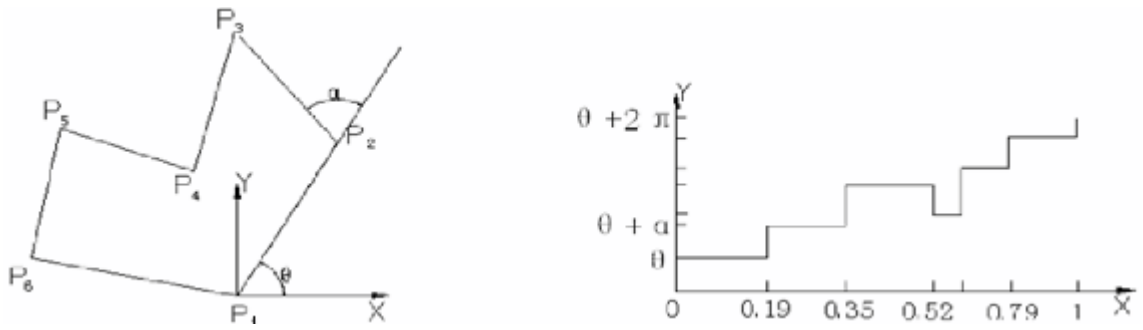


Figure 4.6 – A polygon and its respective turning function, source (Zhao, Sheng, and H. Guo 2009)

In this implementation the length of the edges is scaled so the total perimeter of the polygon is 1. This allows the comparison of polygons when the scale of the images is different. The Y axis is the summed up values of the turning angle at each vertex. The turning angle of the referential vertex is the counter-clockwise angle formed by X axis and the edge formed by the referential vertex and the next vertex in a counter-clockwise order. The rest of the turning angles are calculated summing the previous turning angle to the counter-clockwise external angle of each vertex. This representation is invariant under translation, and the rotation of the polygons is represented by a shift in the Y axis, as it is showed in Figure 4.7. In this figure it is possible to observe a source polygon A and a target polygon B. The polygon B is the polygon A rotated by

180º and below the polygons it is possible to observe the turning functions of both polygons using the points marked as 0 as referential points. The only difference between the A and B turning functions is that B is shifted 180 units in the Y axis.

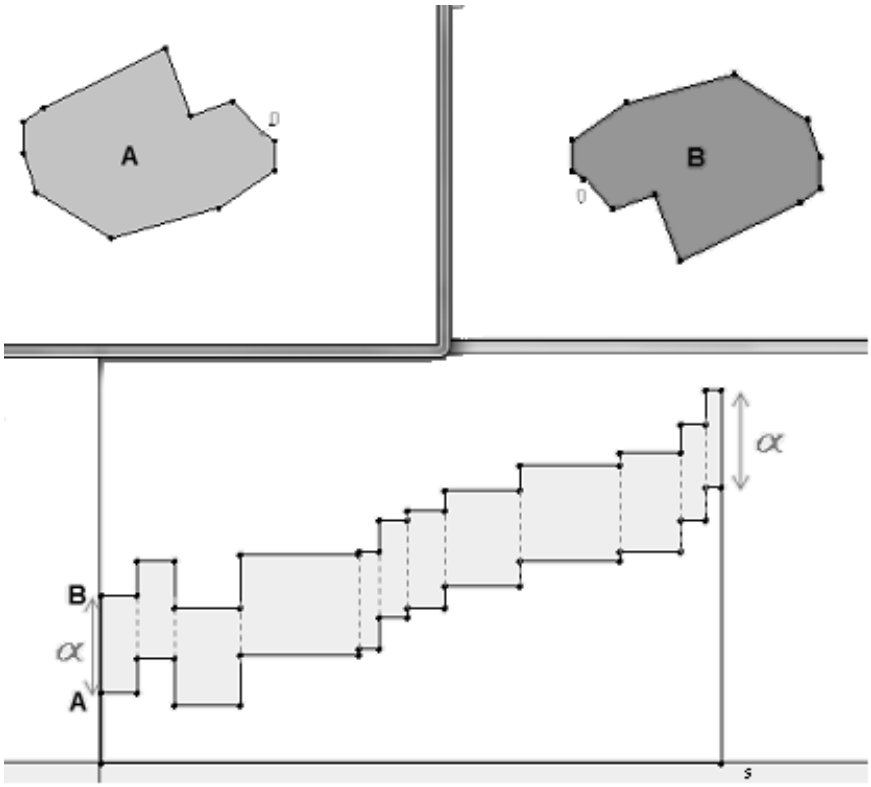


Figure 4.7 - Turning functions of two polygons; A the original polygon; B the polygon A rotated α°

The matching distance of two polygons represented by their turning function $f(s)$ and $g(s)$ respectively, is $\left\{\min_{t\in[0,1]}[\int_0^1[f(s+t)-g(s)]^2\ ds-[\theta^*(t)]^2]\right\}^{\frac{1}{2}}$, where *s* represents the distance of a point belonging to the edges of the polygon to the referential vertex of that polygon divided by the total perimeter of the polygon; *t* is the amount by which the initial referential vertex is shifted along the edges of the polygon, $\theta^*(t)$ is the best value by which the *f(s)* polygon rotates and $\theta^*(t)=\int_0^1 g(s)ds-\int_0^1 f(s)ds-2\pi t$.

The Figure 4.8 shows the turning function of *f(s)* and *g(s)*, and according to (Zhao, Sheng, and H. Guo 2009) the calculation of the matching distance is just the sum of the width values of each strip multiplied by the square of its height for all strips.

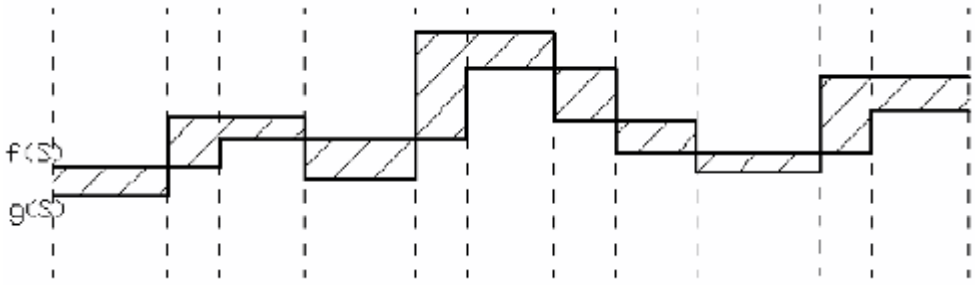


Figure 4.8 - Strips formed by the turning functions f(s) and g(s), source (Zhao, Sheng, and H. Guo 2009)

For each referential vertex there is a different turning function, so a polygon with *m* vertices has m turning functions, $f_i(s), i \in [1,2,\dots,m]$ and for a polygon with n vertices $g_j(s), j \in [1,2,\dots,n]$, n turning functions. Fixing the vertex $p \in [1,2,\dots,m]$ as referential point of $f_i(s)$ and fixing the point $q \in [1,2,\dots,n]$ as referential point of $g_j(s)$, two vertices match only if the following two rules are respected:

Rule 1: only if $j = q$ then

$$\int_0^1 \left[f_p(s) - g_j(s) + \left[\int_0^1 g_j(s)\, ds - \int_0^1 f_p(s) ds \right] \right]^2 ds = Min \ where \ j \in [1,2,\dots,n]$$

Rule 2: only if $i = p$ then

$$\int_0^1 \left[f_i(s) - g_q(s) + \left[\int_0^1 g_q(s)\, ds - \int_0^1 f_i(s) ds \right] \right]^2 ds = Min \ where \ i \in [1,2,\dots,m]$$

Following the Rule 1 it is necessary to determine the j values of $g_j(s)$, that return the minimum values to the formula on Rule 1. Each $p \in [1,2,\dots,m]$ of $f_i(s)$ has a minimum value.

In the Rule 2 it is necessary to determine the i values of $f_i(s)$, that return the minimum values to the formula (*Min*) on Rule 2. In this rule each $q \in [1,2,\dots,n]$ of $g_j(s)$ has a minimum value.

Two vertices match when the minimums in each rule are returned by the same pair of referential vertices, $j = q$ and $i = p$.

To verify the resultant correspondences, (Zhao, Sheng, and H. Guo 2009) proposes to evaluate the average distance between the points from one polygon to the other and only if $|d_i - \bar{d}| < 3s$, the points are corresponding points. d_i represents the distance of one coordinate of the points (*x* or *y*), in f to the corresponding point in g, $\bar{d}$ the average difference between coordinates and *s* the standard error of the coordinates. With this formula it is possible to remove

the correspondences with greater distance than the average distance of all the other correspondences that usually are bad correspondences. However the authors of (Zhao, Sheng, and H. Guo 2009) refer that this method is not fault prove and the only way to really verify the correspondences is using visual judgment.

4.1.3 Implementation

The algorithm described above was implemented in Java and it is divided into three classes.

The class *Strip* has the scaled dimension of each edge and the turning angle of each vertex as attributes.

The class *TurningPolygon* has a constructor that receives a *MyPolygon2D* and an integer representing the index of the referential vertex. This class has as attribute an array of *Strips*, and it is responsible to calculate the turning angles and scaled length.

Finally the class *TurningCorrespondence* is where the correspondences between the polygons are calculated. To do that, this class stores two arrays of *TurningPolygon*'s representing the $f_i(s)$ and $g_j(s)$. The method named *rule* receives the indexes of the referential vertices of both polygons and it calculates the matching distance between the turning functions of the polygons using the referential vertices selected. The next step is to travel for all the strips in both functions and to find the discontinuity values, zones on the graphics where the Y values changes. The difference between the heights at these points of discontinuity (*h*) and their locations (*s*) are stored in an array. The final step is to multiply each *s* by the square of the *h* stored at the same index.

To implement the rule 1, a referential vertex from the polygon A is fixed, which generates a unique turning function. Then the minimum matching distance between the turning function of the polygon A and all the turning functions of the polygon B generated by changing the referential vertices of B is calculated. The vertex identity of B that generates the minimum matching distance is stored in an array, with the index equals to the index of the referential vertex of A. This is repeated until all vertices of the polygon A were referential vertices.

The calculation of the rule 2 is similar to the rule 1, but polygons A and B invert their roles, and a new array is used to store the identities of the referential vertices.

At the end, the values of the arrays are compared and only if the value *j* stored in the index *i* of one array is equals to the index of the value *i* stored in the index *j* of the other array those vertices are possible matches.

Figure 4.9 shows two polygons with the correspondences resulting of this algorithm. In these pictures the vertex with number 0 on the left polygon matches the vertex with the same number in the right polygon. The vertices marked with green circles are vertices where the calculated correspondences are visually wrong. To verify if the vertices are good correspondences we used the test described previously comparing the distance of each matched vertices and the average distance

between the matched vertices. The results using the verification are showed in the Figure 4.10 in this case the three problematic correspondences were removed.

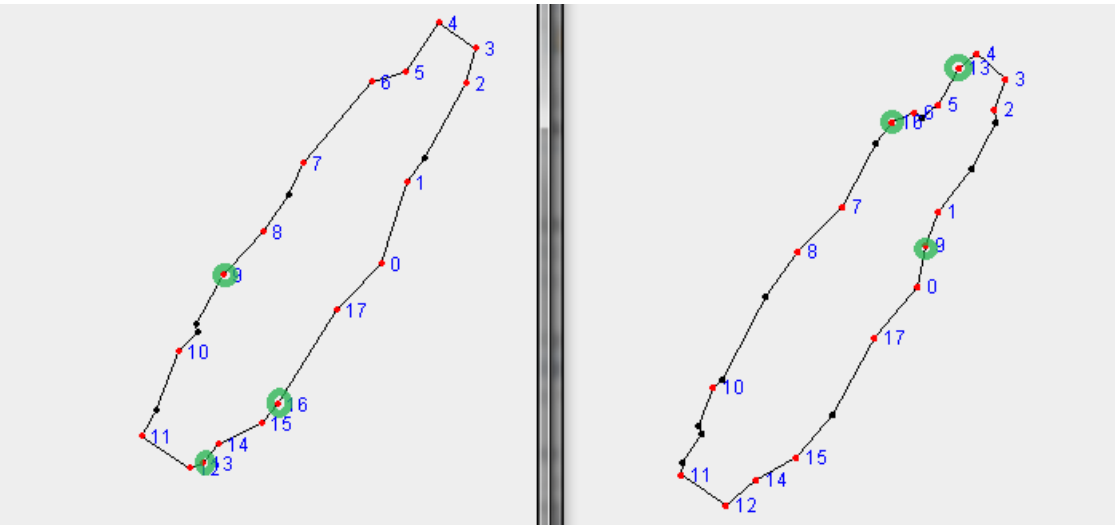


Figure 4.9 – Example of correspondences using this algorithm without verifying the vertices

However in some cases, where there is some rotation involved, this method proved not to be precise and removed more correspondences than it should.

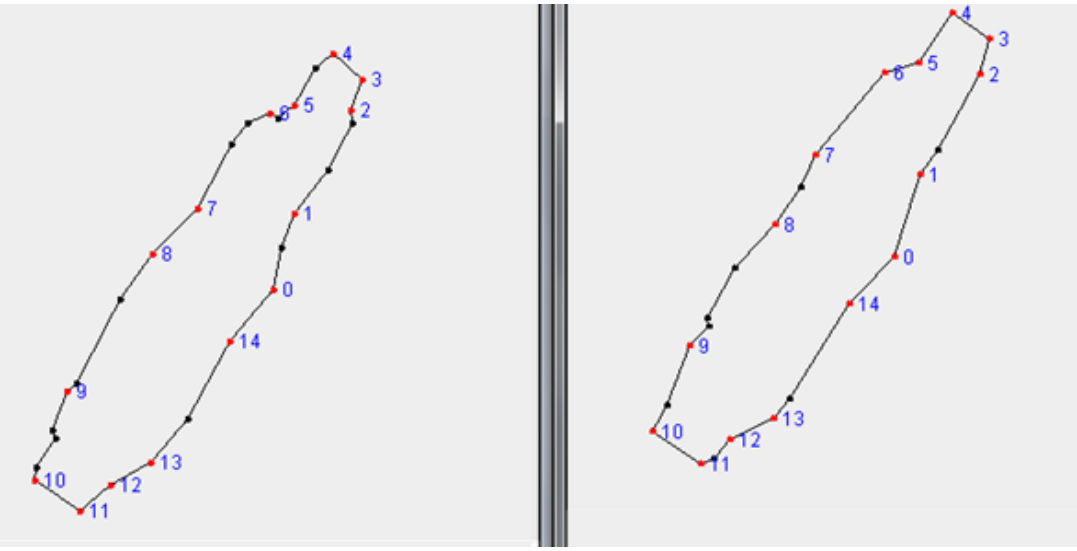


Figure 4.10 – Correspondences after hypothesis test

Figure 4.11 shows the potential correspondences between the polygons A1 and B1. In this case all the matches were acceptable however after the hypothesis test all the correspondences

except one were removed. The polygons A2 and B2 show the remaining correspondence after the test.

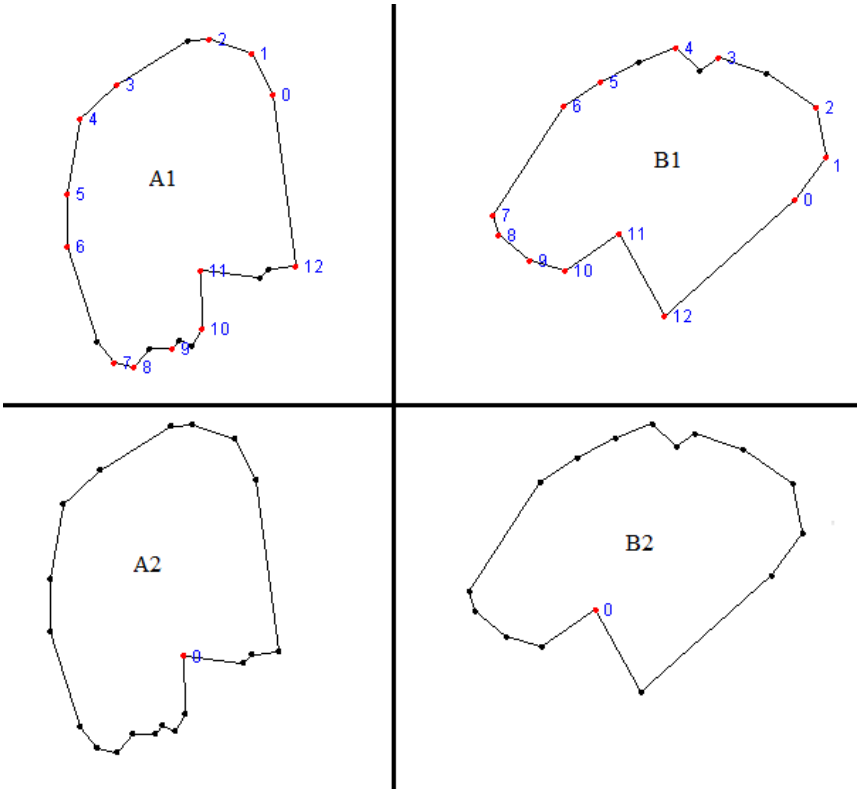


Figure 4.11 – Corresponding points after and before the hypothesis test

To minimize this problem instead of using the coordinates x and y separately, the distance between the points is used. In the case that all correspondences are removed, the option used is the one described in Section 4.3.1.2.

Figure 4.12 shows the results of the improved test.

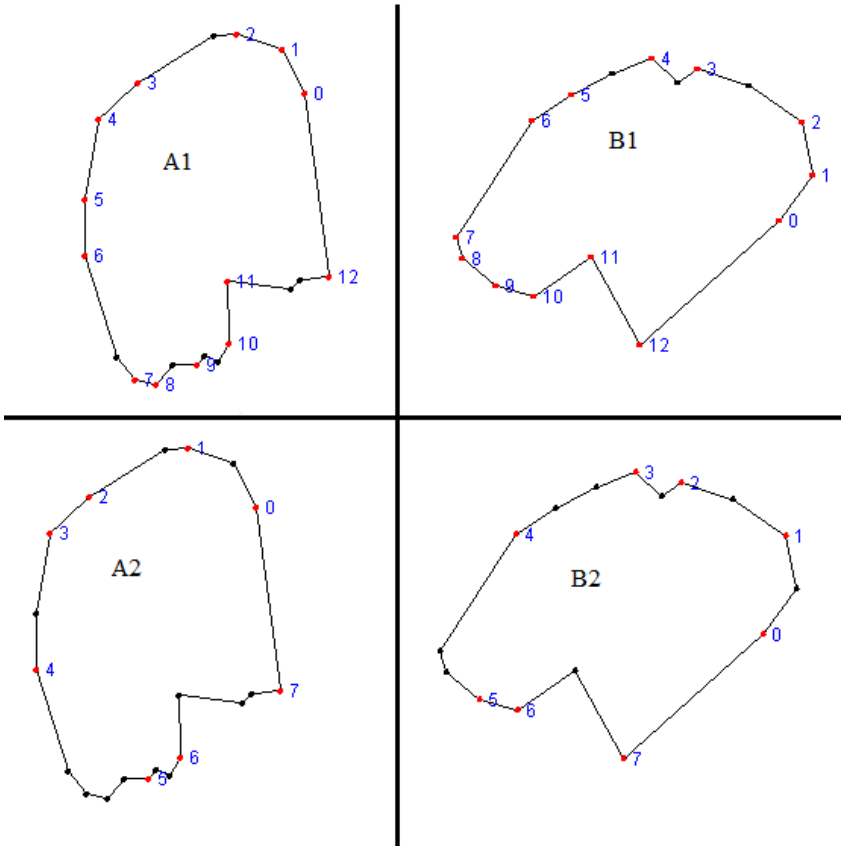


Figure 4.12 - Correspondent points after and before the new hypothesis test

With this test there were some good matches removed, however more than half of the matches remained comparing to the original hypothesis test. In this example not remove vertices would be the best solution but as it was mentioned before this is a necessary step.

4.2 Polygon Alignment

The alignment of two polygons can be used in many different tasks. In this dissertation it is used to edit the polygons, find a first vertex correspondence between two polygons and to calculate the rotation angle of the polygons.

4.2.1 Overview

To align two polygons it is necessary to determinate the transformation that changes a polygon into the other. The most used methods to calculate this transformation are the Iterative Closest Point (ICP) (Besl and McKay 1992) and the Procrustes Analysis (Ross 2004).

The ICP algorithm is an algorithm that does not need any correspondences. As it is explained in (Ju 2012), the ICP algorithm firstly calculates an initial orientation of the two different shapes using the Principal Component Analysis (PCA) (Shlens 2005) and then iteratively improves the alignment using an SVD algorithm.

The PCA gives the axes that represent the variation of the points which are the eigenvectors of a covariance matrix M. So to obtain the axes of a polygon it is necessary to determinate the matrix M for each polygon, where $M = PP^T$ and P has in its columns the distance between each point of the polygon and its centroid.

The eigenvectors that interest to do the alignment are the eigenvector representing the greatest variation between points and the eigenvector that represents the smallest, this variation is called eigenvalues. Figure 4.13 shows the eigenvectors of interest for the mesh of points.

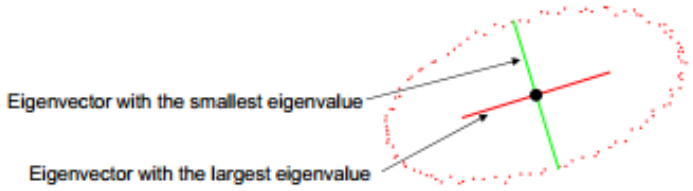


Figure 4.13 – Polygon with the respective eigenvectors, source (Ju 2012)

Having the axes calculated for each polygon, using the PCA, the next step is to calculate a rotation matrix R that will align the eigenvectors of both polygons, $R = BA^T$, where A and B are two matrices containing the orthogonal and normalized axes as columns for each polygon. Figure 4.14 shows two meshes of points overlapped and their respective eigenvectors of interest. In this image is easily observed that the alignment of the eigenvectors of both meshes of point will align the meshes. This alignment is obtained using the R matrix.

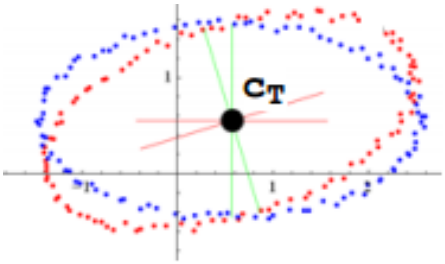


Figure 4.14 – Two polygons with their PCA axes unaligned, source (Ju 2012)

Each iteration of the SVD needs the correspondences between points of both meshes. These correspondences are in the simplest implementation defined by the nearest point on the other polygon. The iterations continue until a termination criterion is reached. This criterion may be defined by the user limiting the number of iterations or limiting the minimum improvement per iteration. The improvement is calculated using the difference between the Root Mean Square Distances in consecutive iterations.

The Root Mean Square distance is calculated using the formula $\mathrm{R}MSD = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2 + (y_i - \bar{y})^2}{2N}}$..

There are many variants of ICP algorithms, mainly in the choice of the correspondent points. The article (Rusinkiewicz and Levoy 2001) refers some of those variants.

4.2.2 Methods Implemented

The methods to align two polygons implemented are described in this section.

4.2.2.1 Maximize Overlap Area

This is a simple method that aligns the centroids of both polygons and measures the overlap area resulting from the continuous rotation of one of the polygons until the rotation angle reached 360 degrees. The angle returned is the angle from which the overlap area has the greatest value.

The Figure 4.13 depicts the relationship between the overlap area and the rotation angle. In this figure the blue polygon is fixed and the yellow polygon is rotated. The green area is the overlap area of the polygons. It is important to note that the polygons were translated so that their centroids can have the same coordinates.

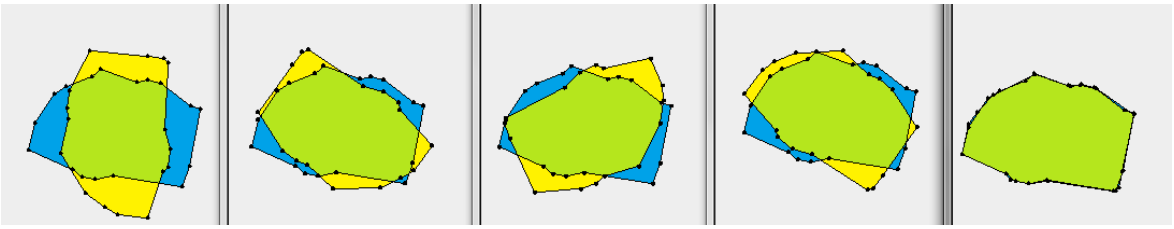


Figure 4.15 – Example of the maximize overlap area

This method was implemented in the class *MaximizeOverlapArea* that has a method that receives a source and a target *MyPolygon2D* objects and returns an angle in radians. The target polygon is translated so its centroid is aligned with the centroid of the source polygon. The rotation angle starts in 1 and ends with 360 and it is incremented in each iteration. The rotation is applied to the target polygon to all its vertices using the rotation method implement in (Paulo 2012) and the method *getOverlapArea* implemented in the *MyPolygon2D* class calculates the overlap area from the two polygons. This method converts the polygons to objects of the type *Area* from the *java.awt.geom* package. This conversion happens because this is the only object in the default java libraries that allows operations between shapes like intersections and unions.

After having the intersection resultant from both *Area*s, using the *intersect* method of the *Area* class that returns another *Area* object, it is necessary to associate a value to that object. Since the *Area* object does not have a method that returns the numeric value for the area, this is

accomplished converting the *Area* object into a *ShapeRoi* from the *ImageJ* library and then converting it into an array of *MyPolygon2D*'s, that is the object with a method to calculate the *double* value of the area, implemented in (Paulo 2012). An intersected *Area* may return various *MyPolygons2D* the overlapped area is the sum of the areas of all *MyPolygon*'s returned.

Each overlap area is compared to the greatest overlap area stored in previous iterations or to -1 if it is the first one. If the newest area is larger than the previous one it replaces the previously stored, nothing changes otherwise.

The angle that originates the greatest overlap area is returned from this method.

It is important to note that this method only works when the two polygons are in the same scale.

4.2.2.2 Iterative Closest Point

In this dissertation the ICP algorithm used is the one described in (Ju 2012).

To represent the matrices the library *JMat* available in ("JMAT | Free Science & Engineering software downloads at SourceForge.net" n.d.) was used. This library has methods to calculate the SVD, and an implementation of the PCA is available in ("JMAT to compute the Principal Component Analysis" n.d.).

The ICP algorithm is implemented in the ICP class and has a method to calculate the iterative SVD receiving two *MyPolygon2D*'s, the maximum number of iterations and the minimum difference in the improvement of each iteration. The method *icp* is responsible to calculate the ICP alignment and it receives a source and a target *MyPolygon2D's,* the number of maximum iterations and the minimum difference between consecutive iterations. This method calls the *pcaAlignemt* method in the PCA class that receives the source and target *MyPolygon2D*'s and returns the target polygon rotated using the PCA algorithm.

However since the PCA algorithm sometimes returns matrices with determinant equals -1, that represent reflections, instead of matrices with determinant equals 1, rotations, when the determinant of the returned matrix is different than 1, the angle is calculated using the method described in Section 4.2.2.1.

The correspondences between vertices are chosen using the *findNearestPoint* method present in the class *MyPolygon2D*. This method receives a *Point2D* and returns the vertex in the polygon nearest to this point. This search is done measuring the distance between this point and all the vertices of the polygon returning the vertex with smaller distance.

Figure 4.16 depicts the alignment using the ICP algorithm. In this case the starting polygons on the left do not need to be translated, since this is integrated in the PCA algorithm. The centre image is the resulting polygons after the PCA alignment. And the last image is the result of the iterative SVD algorithm.

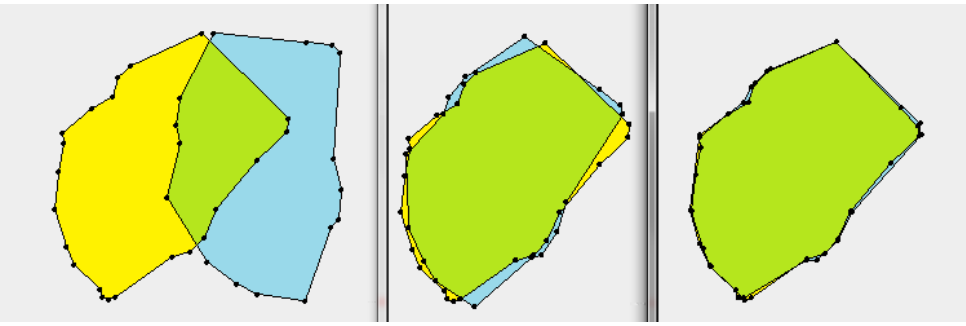


Figure 4.16 – Example of alignment using the ICP algorithm

4.3 Optimizations of the Perceptually-based algorithm

The algorithm implemented in (Paulo 2012) can be optimized not only improving the vertices correspondences between two polygons, but also in the computation time to obtain the results.

The computation time and results of the algorithm can be reduced by sending the first correspondence to the algorithm, methods to determine this first correspondence are described in the following section Having the starting correspondence the algorithm does not need to test all possible combinations between *feature points* to choose the one that costs less, and if this correspondence is correct, the errors that would arise from a wrong first correspondence are avoided. Other way to improve the correspondence results is to match the vertices number and their distribution along the polygons.

4.3.1 Getting the first correspondence

This dissertation explores two methods to determine the first correspondence to be used in the correspondence algorithm of (Paulo 2012), using the alignment of the polygons and choosing the nearest *feature points* of two different polygons or using the turning correspondence algorithm.

4.3.1.1 Using polygon alignment

Having two polygons aligned it is possible to obtain a correspondence between the vertices of the polygons using the distances between the vertices in one polygon to the vertices in the other polygon.

The *feature points* of the polygons are calculated using the method implemented in (Paulo 2012). The first correspondence is formed by the *feature points* with the smallest distance in the aligned polygons. The rest of the correspondences is determined using the algorithm of (Paulo 2012).

The method *getNearestFeaturePoint* was implemented in the *MyPolygon2D* class to determinate which *feature points* were the nearest. Its behavior is similar to the *findNearestVertex* used in the ICP algorithm and described in Section 4.2.2.2, which unlike the *findNearestVertex* only verifies the distance of vertices that are *feature points* in both polygons.

4.3.1.2 Using the Turning Correspondences

Another way to obtain the first correspondence between two polygons is to use the best correspondence obtained using the algorithm described in Section 4.1.2. Using the turning correspondences, a correspondence is the best one if it has the minimum matching distance.

This method reduces the time needed by the perceptually based algorithm to determinate the correspondences, however it is necessary to account with the time needed by the turning algorithm to calculate the first correspondences.

4.3.2 Match Vertices Number

In the method used in (Paulo 2012), two polygons with different number of *feature points*, or with different distribution of vertices may originate incorrect correspondences, that will generate deformations during the morphing.

Figure 4.17 depicts two polygons and the *feature points* calculated using the method implemented by Luis Paulo. The resulting *feature point* correspondences using this algorithm were from the polygon on the left to the polygon on the right:

$$4 \rightarrow 7;\ 5 \rightarrow 8;\ 0 \rightarrow 0;\ 1 \rightarrow 2;\ 2 \rightarrow 3;\ 3 \rightarrow 4.$$

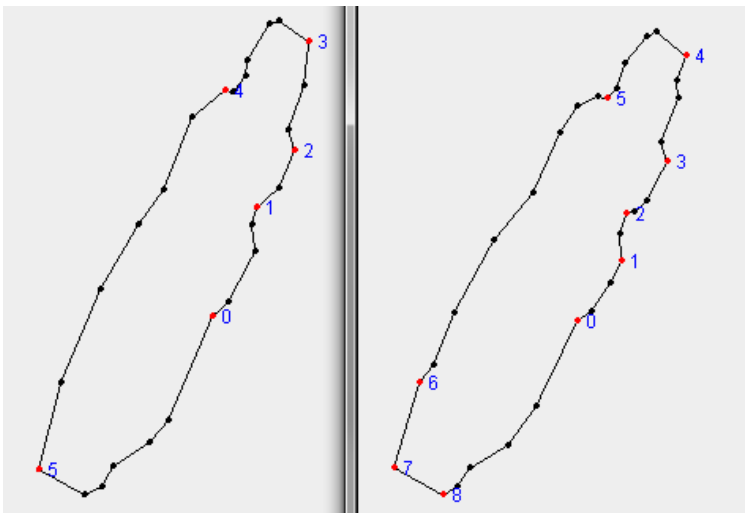


Figure 4.17 – Two polygons with different feature points distribution

This set of correspondences will originate a deformation during the morphing from one polygon to the other. Figure 4.18 shows the intermediate polygons using linear interpolation of

those correspondences. In this figure it is possible to see the deformations that occur using the correspondences presented previously.

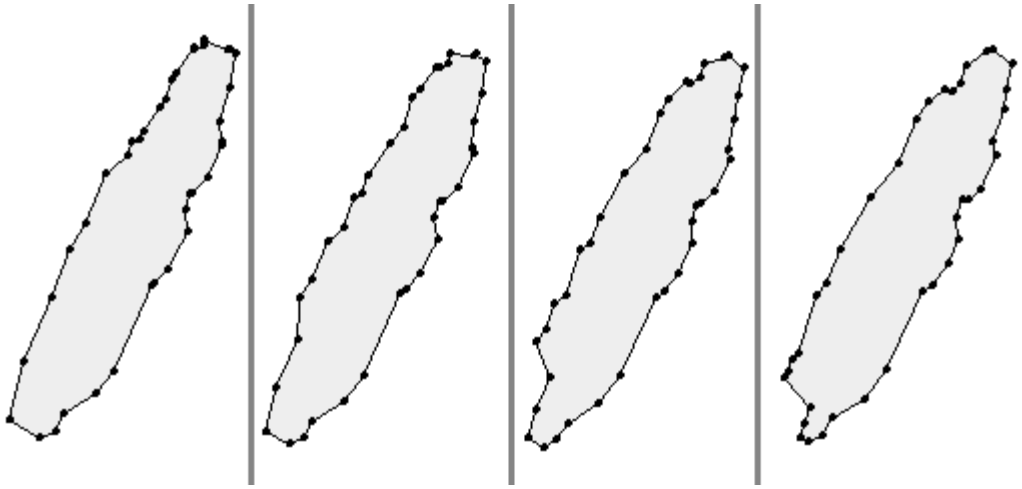


Figure 4.18 – Intermediate polygon resultant from the linear interpolation

To minimize this problem, the number of *feature points* of two different polygons is matched in a way that their distribution along the boundaries of the polygons is similar in both.

4.3.2.1 Match feature points number

To match the *feature points* number between a polygon A and a polygon B, the method implemented has four steps.

Firstly both polygons are aligned. This was achieved using the algorithm described in the section 4.2.2.2.

Secondly the *feature points* of each polygon are calculated, using the algorithm implemented in (Paulo 2012).

The third step is to process all *feature points* of the polygon A and associate them with the corresponding vertices in the polygon B. If no association is possible, a *feature point* is added in the polygon B or it is removed from A if no addition is possible. A *feature point* is marked for removal if there is no vertex available in B at the scaled distance of the previously associated *feature points* or if it is not possible to add a new vertex at that distance on B because a vertex is already there. A vertex is unavailable if it is already associated with other *feature point.*

The fourth step is to process the *feature points* of the polygon B following the same rules of the step 3.

The first *feature point* of the polygon A analyzed in step 3 is matched with the nearest point belonging to the edges of the polygon B, if such point is not a vertex of the polygon, a vertex is

added to the polygon in those coordinates and it is considered as a new feature point of the polygon B.

The processed *feature points* and their matches are stored into two *arrays donePolA* and *donePolB* to the polygon A and B respectively. These arrays are used by the remaining *feature points* to find their nearest neighbours. These neighbours are used to limit the distances where new *feature points* may be added.

Figure 4.19 shows two polygons with different number of *feature points*, in this case the *feature points* 1 and 0 from the polygon A had already been processed and their matches are the *feature point 2* and 0 on the polygon B. If the next *feature point* to be analysed is the *feature point 1* of the polygon B, that it is between two other vertices (2 and 0), the matching vertex in polygon A must be between the matching boundary *feature points* (1 and 0). This is assured using the *donePolA* and *donePolB* arrays and will result in the addition of a new *feature point* in polygon A.

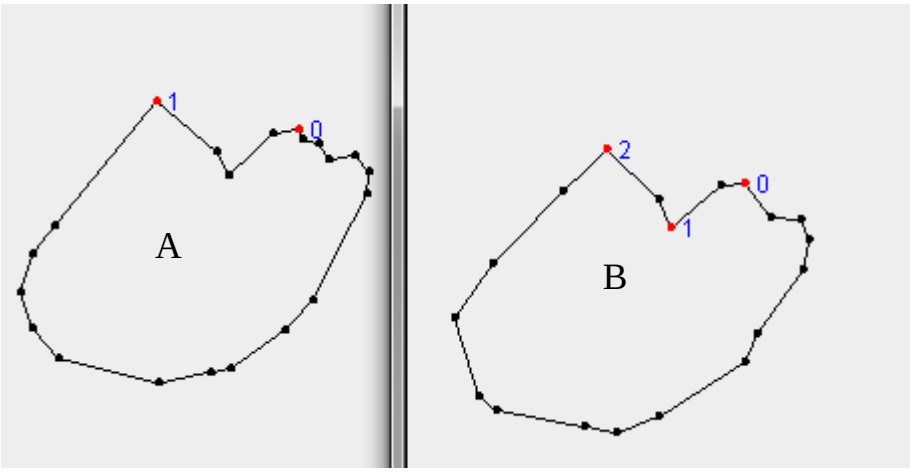


Figure 4.19 – Example of neighbours' boundaries

The Figure 4.20 shows a possible configuration when adding new *feature points*. The *feature points* 0 and 1 of both polygons had already been processed and the *feature point 2* on polygon *B* is missing a matching point in the other polygon. In this case the distance between the *feature point 1* and 2 is calculated and used to maintain the distance between the *feature point* 1 and the *feature point* to be added on the polygon A. Since the perimeters of the polygons are different the distance is scaled, multiplying it by the perimeter of the polygon A and dividing it by its own perimeter. The first vertex on the polygon A to be tested is the nearest vertex of the *feature point* 2 of the polygon B after the alignment.

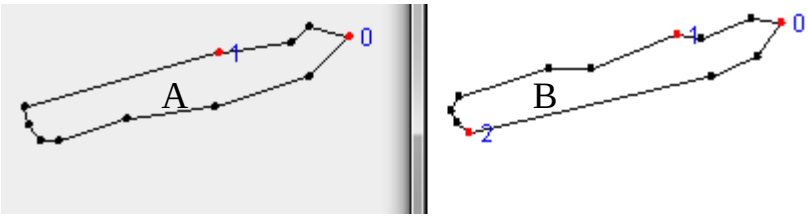


Figure 4.20 – Example of initial configuration

In the case depicted in the Figure 4.21, where the *feature point 0* and 1 from polygon A match the *feature points* 0 and 2 in the other polygon and the next vertex to be analysed is the *feature point 1* of the polygon B, the search for the nearest vertex will not work since there is no vertex between the bounding matching *feature points* in the other polygon. The solution implemented will search for the nearest point from the *feature point 1*of the polygon B in the edges of the polygon A, after alignment. In this example a point in the edge formed by the *feature point 0* and 1 on the polygon A. However it is necessary to verify the bounds and the scaled distance to assure that the vertex is inside the bounds.

Figure 4.21 – Example of no vertex between feature points

The last option implemented was to insert vertices directly at the scaled distance from the previous *feature point*, and since the scaled distance is used it is assured that the vertex will be inside the boundaries. If this fails then the *feature point* is removed, because there is no position available in the other polygon.

The Figure 4.22 shows the result of the application of this algorithm to the polygons A1 and B1. Initially the polygons A1 and B1 had 2 *feature points* and 3 *feature points* respectively. After the application of the algorithm the resulting *feature points* are 4 *feature points* on each one, two were added to the polygon A1 and one was added to the polygon B1.

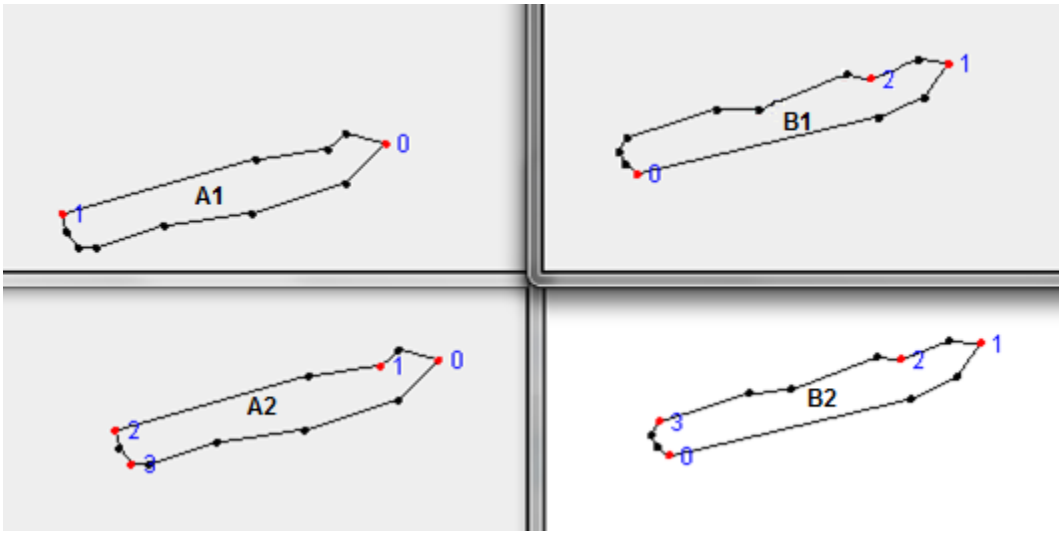


Figure 4.22 – Two polygons before (up) and after (down) feature points matching

4.3.2.2 Match non feature points number

After having the *feature point* numbers matched, and their correspondences, the matches of the number of the rest of the vertices can be divided in the match of each section limited by each consecutive *feature point.*

The two polygons in the last row of the Figure 4.22 have 4 different sections. And in each section is necessary to add vertex to match the vertices number but also to distribute them in a similar way in both polygon. This is done using scaled distances between the vertices on each matching section.

The Figure 4.23 shows the polygons of Figure 4.22 with their vertices number matched. The added vertices are presented in green in Figure 4.23.

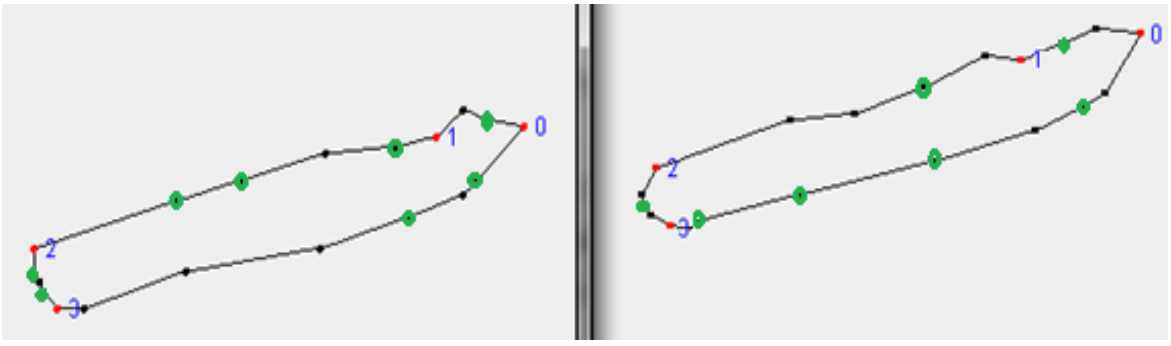


Figure 4.23 – Result of matching vertices

4.4 Summary

This chapter is dedicated to the vertex correspondence problem of the morphing algorithms. It described the p*erceptually based algorithm* implemented in (Paulo 2012) and presented an alternative algorithm to calculate the correspondence between two polygons. This alternative uses turning functions of the polygons. A turning function is a type of representation of a polygon that uses distances between their vertices and their angles.

Some methods to align polygons are also discussed in this chapter. These methods are used to improve the results of the *perceptually based algorithm*, matching the number of vertices between polygons and getting a first correspondence.

Having the VCP solved the next step to represent the movement of a moving object is to solve the VPP. The next chapter is dedicated to solve this problem.

5 Vertex Path Problem

In this chapter two solutions for the VPP problem will be discussed. The *Cyclic Order* algorithm created to avoid intersections on polygons (McKenney and Webb 2010), and an alternative solution decomposing the movement of the objects into two different steps: an alignment followed by an interpolation.

5.1 Overview

In the literature there are many algorithms to apply the morphing between two polygons. The linear interpolation is the simplest of those algorithms. This method connects two points using a straight line. Applying this concept into matched vertices of two polygons it is possible to obtain a linear function to each pair of matched vertices to represent their movement. However this algorithm may originate invalid topologies, when the movement of the polygon is not exclusively linear, because some vertex paths may intersect and will originate deformations or self-intersections on the representation of the moving object. This problem is commonly known as Vertex Path Intersection (VPI) problem, and it is the main problem from the (VPP). Figure 5.1 shows the vertex paths between two polygons (dashed lines) using linear interpolation. It is possible to observe that the path will intersect and the polygon would degenerate to a point during the morphing.

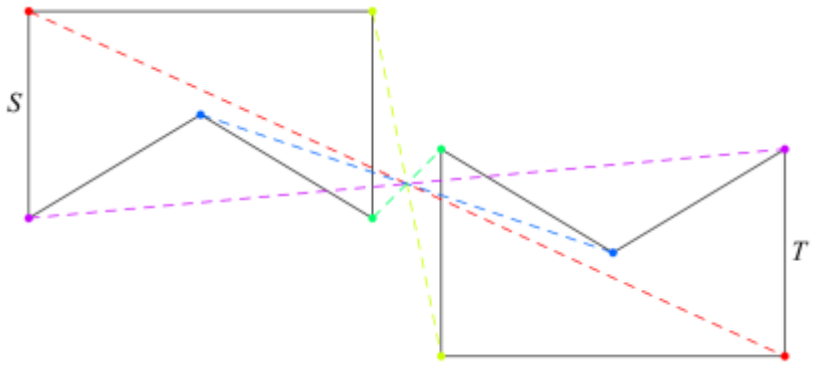


Figure 5.1 - Vertex Path Intersection using linear paths

To solve the problem of VPI, there are many different approaches referred in the literature. The solution proposed in (Sederberg et al. 1993) uses a solution similar to a turning function since it relates the angle and length of each corresponding edge to determinate the intermediate polygon. According to (Málková et al. 2009), this method returns good results when both the source and the target polygons are similar, but it fails to polygons highly dissimilar. Another disadvantage of this method is that it does not address the self-intersections problem.

In (Iben, O'Brien, and Demaine 2006) it is presented an algorithm that uses energies to determinate which polygon should move. In this algorithm both polygons must have the same number of vertices, otherwise, vertices are added to one of them. The energies of each polygon are calculated using Euclidean distances and the polygon with greater energy is moved towards the one with a lower one. The necessity to have polygons with the same number of vertices and to calculate the energies of the polygons, in each time instant create extra complexity to this method.

Other option is to use triangulations as in (Gotsman and Surazhsky 2001). This is according to the authors the only technique that guarantees a simple intermediate polygon from a simple source and target polygons. The problem is basically to determinate the planar triangulation and do the morphing between corresponding points, that is achieved interpolating the barycentric coordinates of the vertices. The Figure 5.2 depicts the morphing of a triangle inside a quadrilateral. The image in the right shows the movement of the triangle during the morph. A problem from this method is that it needs to triangulate the polygons and to assure that those triangulations are compatible with each other which are not always simple tasks.

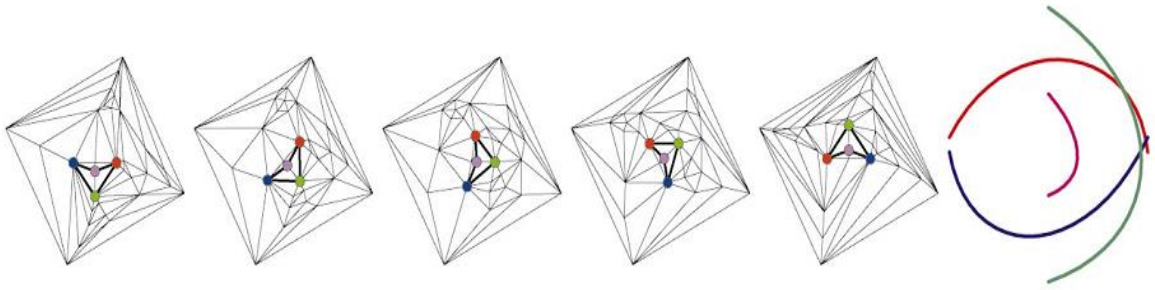


Figure 5.2 – Example of morphing simple quadrilaterals using triangulations, source (Gotsman and Surazhsky 2001)

The start skeleton method introduced in (Shapira and Rappoport 1995) decomposes the polygon into a star set and a skeleton. The star set is composed by a set of star shaped polygons, each one possessing a different star origin. A star origin is the point where all the other vertices of the star polygon are visible. The visible vertices are represented as polar coordinates relative to star origin. The skeleton is a tree composed of the star origin points and the midpoint sharing edges between two different star polygons. Each vertex of the skeleton has a direction. To the root this direction is the x axis, to the others it is the vector from the vertex to his parent. The morphing is accomplished doing the linear interpolation of the root skeleton on cartesian coordinates and all the other points on polar coordinates which implies the interpolation of distances and angles. (Málková et al. 2009) detects a problem for this method. When the polygons are dissimilar, it may be difficult to calculate compatible skeletons for the polygons.

The Figure 5.3 shows two polygons with their star skeletons.

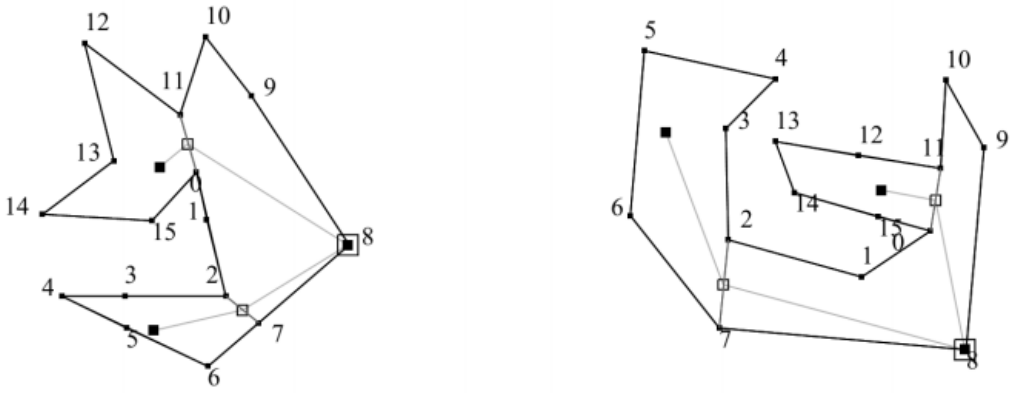


Figure 5.3 – Example of two polygons with compatible star-skeletons, source (Shapira and Rappoport 1995)

(Málková et al. 2009) describes a method where a source and a target polygon are overlapped. The morphing occurs by absorption and growing of the areas that did not overlap. The overlapping area of a source polygon A and target polygon B are named the core, the section C in

the Figure 5.4 and the parts of the source polygon not belonging to the core (P1, P2 and P3 from the Figure 5.4), are absorbed by the core and the parts of the target polygon not belonging to the core (Q1, Q2, Q3 from the Figure 5.4) grow from there.

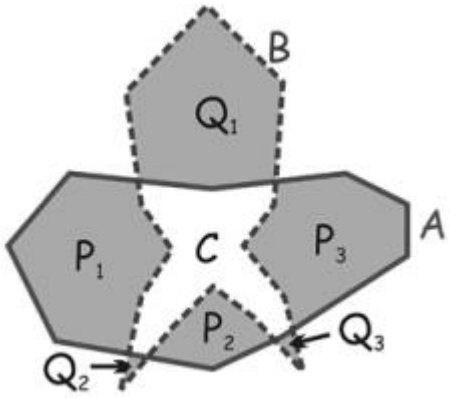


Figure 5.4 – Example of morphing of polygon A and polygon B, source (Málková et al. 2009)

The morphing of the growth or abortion can be determined by a perimeter growing method, where the vertices will travel by the boundaries of the polygon until they reach their matches. The matches are calculated using the distances that are symmetric in both polygons.

The Figure 5.5 shows the distances of the vertices for the section P3 of Figure 5.4 during the absorption process, in this case the vertices will be moving by the edges of the polygon until the vertex with symmetric distance is reached.

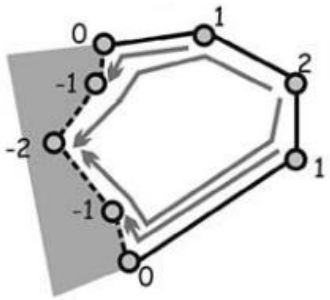


Figure 5.5 – Example of the vertex correspondence on P3 absorption, source (Málková et al. 2009)

One problem to this method is that it does not take into account the orientation of the polygons. If one target polygon is rotated in relation to a source one, the non-overlapped areas of the polygons will shrink and expand even if they are similar but in different positions causing deformations.

The *Cyclic order* algorithm presented in (McKenney and Webb 2010) uses a strategy where one edge in one polygon is reduced to a vertex on the other and a vertex in one polygon is extended to an edge. This algorithm only uses convex hull to determinate the transformation and so one concavity is always converted into a vertex. This algorithm requires that the vertices are processed in counter-clockwise order, starting in the left most vertex of each polygon. However this process does not consider the orientation of the polygons and it will not maintain the shape of the polygon when some rotation exists between the polygons.

Figure 5.6 shows the path of the vertices of the light grey polygon to the dark grey one using this algorithm.

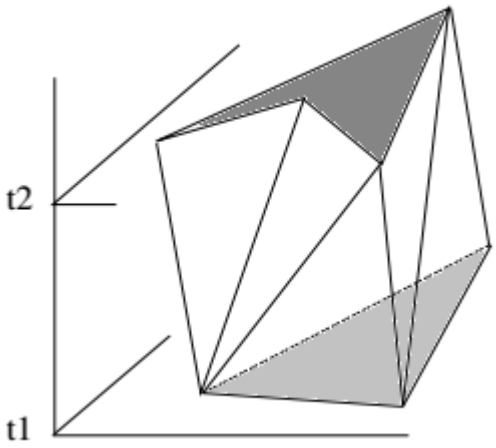


Figure 5.6 – Example of the morphing correspondence using the Cyclic Order algorithm, source (McKenney and Webb 2010)

In the dissertation (Paulo 2012), the only method to do the morph presented is the linear interpolation.

5.2 VPI Solutions Implemented

5.2.1 Cyclic Order Algorithm

To avoid vertex path intersections the algorithm (McKenney and Webb 2010) was implemented. This article proposes a morphing algorithm that given a valid input region a valid region will always be generated during the transformations. A valid region is a spatial region without self-intersections.

This algorithm has a very different data model than the one implemented in (Paulo 2012), since it needs not only the coordinates of the vertices but also the angles. The *moving segment* is composed by a segment and a point forming what they call a *delta triangle*. Since a concavity will be morphed into a single point, it is necessary to calculate the convex hulls of the polygons.

The correspondences between two polygons are calculated using the *cyclic order*, and *progress angles*.

The *cyclic order* defines the order from which each vertex must be visited. It is counter-clockwise starting in the *least most point* in the *least most segment*. The *least most segment* is the edge that contains the smaller *least point* or the smallest angle from two segments containing the same *least point*. The angle is measured counter-clockwise between the edge and a segment emanating up from its *least point*. The *least point* of a segment is the point *(x, y)*, that has the smallest *x* or if there are two vertices with the same coordinate *x* the one with smallest *y* coordinate. To choose from which polygon the next segment will be processed the *progress angles* of the two selected edges are compared. The edge with the smaller *progress angle* is morphed to the *primary point* of the selected edge in the other polygon.

A *progress angle* measures how much a polygon has been processed. This angle is the counter-clockwise angle formed by a vertical line extending down from the *primary point* of each edge and the edge itself. The *primary point* is the first point from the segment when traversing it in the *cyclic order*.

However if concavities exist, the comparison between *progress angles* does not translates the progress on the polygon, since the values of the *progress angle* would not be exclusively in ascending order. The Figure 5.7 depicts a concave polygon and the *progress angles* of each edge. In this picture is possible to verify that inside the concavity the angle decreases, the angle in E is smaller than the angle of F. To solve this problem only convex polygons or convex hulls of concave polygons are used.

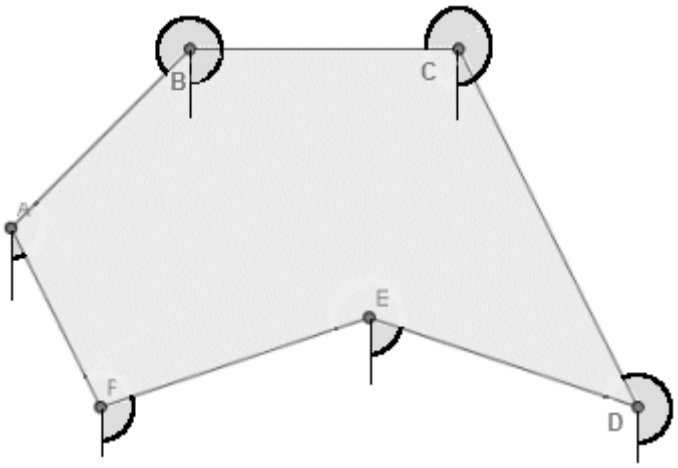


Figure 5.7 – A concave polygon with respective progress angles

A concavity is handled as a simple segment and it is totally morphed into a vertex on the other polygon. In the Figure 5.7, the edges FE and ED would morph into the same vertex in the other polygon.

However there are some concavities that must be dealt with before performing the correspondences because their shape may cause self-intersections. In (McKenney and Webb 2010) they are called *intersecting concavities. Intersecting concavities* are sets of two or more concavities where one point in one concavity is completely surrounded by a second concavity. An example of such polygon is showed in Figure 5.8.

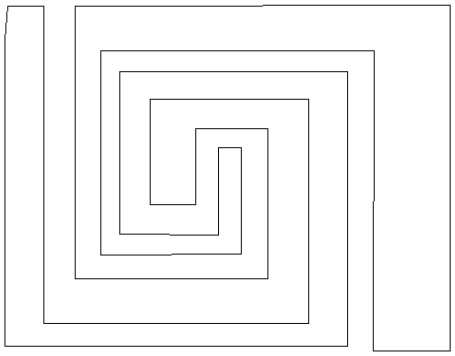


Figure 5.8 – An Example of a polygon with intersecting concavities

To treat these kinds of polygons it is proposed to use *OBBTree interference detection* algorithm to detect when the delta triangles of a concavity intersect triangles of other concavity without sharing the same end point. The end point is the matched vertex on the other polygon for an edge.

When *intersecting concavities* are detected two additional transformations must occur, an *evaporation* before the morphing algorithm and a *condensation* after. E*vaporation* is the collapse of the triangles of a triangulated concavity towards an interior point on the source polygon and *evaporation* is the inverse to the target region.

However such shapes are unusual in nature, so this case was not implemented in this thesis.

5.2.1.1 Convex Hull

To use this solution it was necessary to implement an algorithm to calculate the convex hull of a polygon. The algorithm chosen to do this task was the *Melkman Convex Hull algorithm* (Melkman 1987) that has a linear time complexity and it is an online algorithm that uses a *Dequeue*. A *Dequeue* is a *queue* that can be accessed both from the top and the bottom.

This algorithm starts with any three consecutive vertices of the polygon that form the triangle showed in the Figure 5.9. These vertices are stored in the *Dequeue* from bottom to top as

{3, 1, 2, 3}. When they are read from the *dequeue* in top to bottom order, a clockwise hull is returned. If the *dequeue* is read from bottom to top then the convex hull will be in counter-clockwise order.

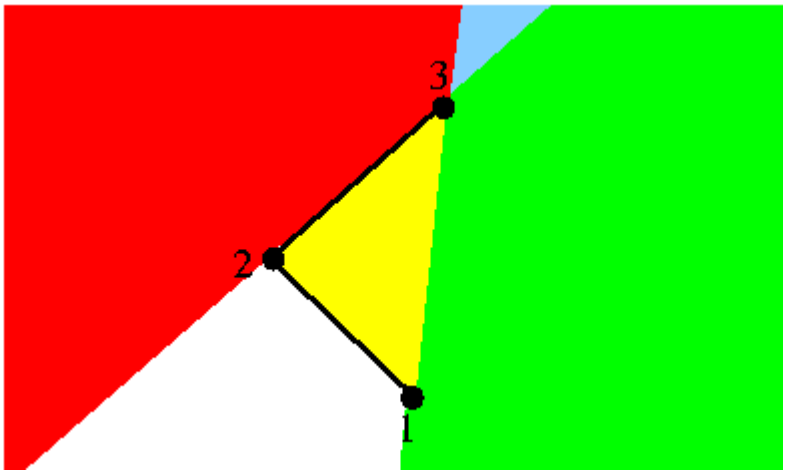


Figure 5.9 – Melkman Convex Hull algorithm, source (Aloupis n.d.)

This algorithm is applied to simple polygons. The next vertex will belong to one of the coloured regions on the Figure 5.9. If a vertex falls into the yellow region, it is ignored, since it belongs to a concavity, otherwise it is added to both the top and the bottom of the *dequeue*. If a vertex belongs to the red region, the vertices in the top of the *dequeue* are deleted until a convex turn is found, else if it falls into the green region the vertices from the bottom of the *dequeue* are deleted until a convex turn is found. Finally if the vertex belongs to the blue region both steps from the red and green regions are done.

The implementation of the algorithm was based in the code presented in (Shahriyar n.d.), using an *ArrayList* to simulate the *Dequeue* and the class of *Point2D* from the Java library.

Figure 5.10 presents the result of the convex hull algorithm implemented.

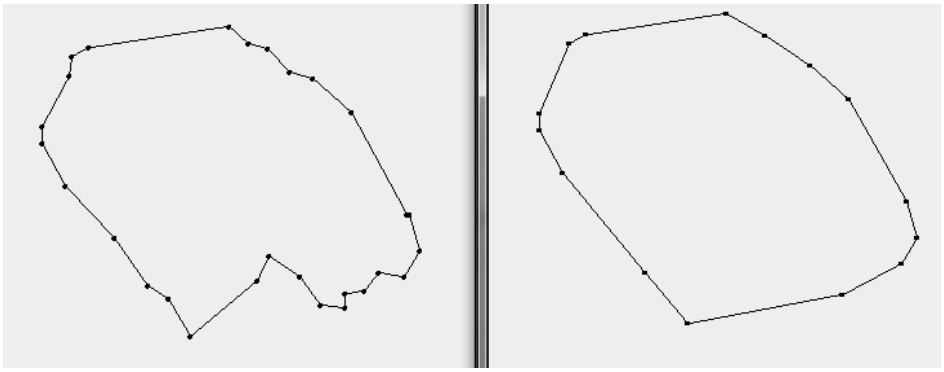


Figure 5.10 – A result from the implemented algorithm

5.2.1.2 Class Edge

The class *Edge* attributes are: two *Point2D*, representing the endpoints, its length, the progress angle and the angle.

The angle and progress angle are calculated using trigonometric functions. Another method implemented in this class is the *compareEdges* that is responsible to compare two edges following the algorithm rules, firstly verifying which one has the smaller *least point* and if the *least points* are equal comparing the angles.

5.2.1.3 Class MovingSegment

The attributes of the *MovingSegment* class are: an *Edge;* a *Point2D;* and a Boolean flag to distinguish if a point expands to a segment or an edge converges into a point.

5.2.1.4 Class MorphNoVPI

This class contains the method responsible to determinate the morphing between two polygons.

This method receives two *MyPolygons* and returns an *ArrayList* of *Correspondences.* The class *Correspondence* was implemented in (Paulo 2012).

The first step of the algorithm is to calculate the convex hulls of the polygons and to store them in arrays of *Point2D.* Then the edges of the polygons are ordered using the cyclic order.

Figure 5.11 represents a source and a target polygon that have in yellow and orange the angles of the edges. The left and right polygons are the source and target polygons respectively. The vertex A is the leftmost vertex, whose X coordinate is the smallest, so the *least segment* is one of the edges that have this vertex as *least point*, edges [AB] or [AE]. From those two the edge with smaller angle is the *least segment.* The *least segment* for the source polygon is [AE] and so it is the last edge to be processed from this polygon. In the other polygon the edge formed by the vertices 1 and 4 is the *least segment* of that polygon.

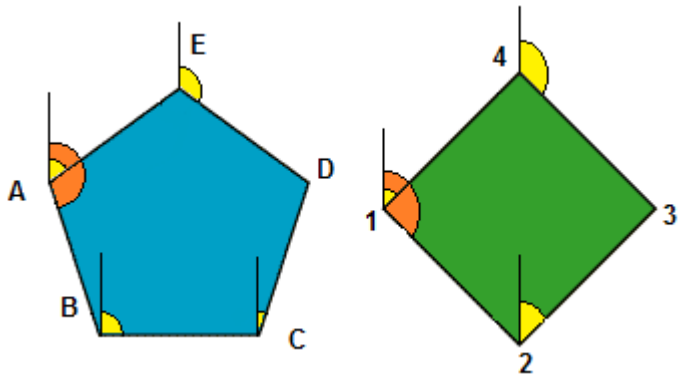


Figure 5.11 – A source and target polygon with the edges angles

In the first iteration the edges containing the *least point* as their *primary point* are the edges to be processed, the *processing edges*.

The moving segment between A, the *primary point* of the *processing edge* of the source polygon and the *processing edge* of the other polygon, [12] is created.

If the *processing edge* of the square was part of a concavity all the edges until a vertex belonging to the convex hull of this polygon would be added to moving segments mapping the same vertex, A.

Then a *moving segment* is created using the *processing edge* of the pentagon, [AB] and the *secondary point* of the *processing edge*, vertex 2. If the *processing edge* of the pentagon is part of a concavity, all edges belonging to that concavity are mapped to the *secondary point* of the selected edge (vertex 2).

The next step is to calculate the *progress angles* of the next edges in cyclic order of the convex hulls of both polygons. The polygon with the smaller *progress angle* and with non-processed edges remaining, repeats the process described using the next segment in cyclic order not yet mapped into a vertex.as its *processing edge.* This is repeated until all edges of both polygons had been mapped.

5.2.2 Separate rotation from the rest of the morphing

The rotation of the polygons is one of the main reasons of the occurrence of VPI and deformations during the morphing of two polygons. This section presents a solution to minimize those problems. The proposed solution is two divide the transformations of a polygon into three components: translation, rotation and deformation. As it is showed in Figure 5.12, this method receives a source polygon S and a target polygon T and using the methods described in Section 4.2.2.2 translates and rotates polygon T until both polygons are aligned, creating the rotated polygon O over the polygon S. This polygon is then translated to its original position, creating the polygon T1 from Figure 5.12.

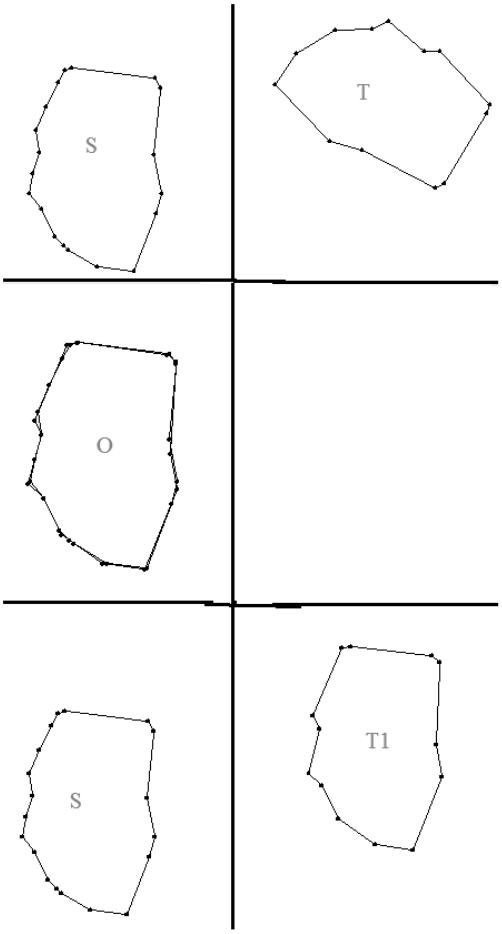


Figure 5.12 – Source and target polygon in aligned and returned to the original positions

To measure the angle it is chosen a vertex from T and the vertex with the same index from T1 those vertices are connected to the centroid of both polygons, forming two vectors.

The Figure 5.13 shows a polygon T in red and a polygon T1 in green after its translation to the original position. The point 0 is the centroid of both polygons and the dashed lines represent the vectors used to measure the angles. And at yellow is the rotation angle retrieved. This angle is the counter-clockwise angle between the vector u and v.

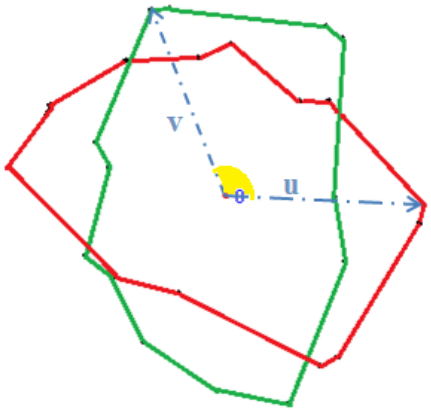


Figure 5.13 Two polygons translated after alignment

The book (Anton and Rorres 2010) explains that angle between two vectors *u* and *v* can be calculated using the formula $\theta = arcos\frac{u.v}{\|u\|\|v\|}$, where *u.v* is the dot product between the two vectors and is calculated using the formula $u.v = x_u \times x_v + y_u \times y_v$ where $\| \|$ determines the magnitude of the vectors and is defined by $\|a\| = \sqrt{x_a^2 + y_a^2}$. However these formulas do not return negative angles since $\cos(\theta) = \cos(-\theta)$. To overcome this problem the cross product of the vectors is computed since it is negative when the angle is negative and positive otherwise. The formula to calculate the cross product between the vectors *u* and *v* is $u \times v = x_u\, y_v - x_v\, y_u$.

After having the S and the T1 polygons and the rotation angle, the morphing technique is applied between these two polygons that do not contain rotations. The rotation is applied to the polygon retrieved by the morphing technique at each time instant using a parcel of the angle proportional to the time instants simulated.

This method can be used to different types of morphing techniques.

5.3 Tools implemented

Some tools were created during the development of this project, to evaluate and tests the algorithms.

One of those tools is the *FeaturePointsEditor* that enables a user to add and remove feature points manually from two polygons. This tool uses the scroll of the mouse to do the zoom in and zoom out of the window, and moving the mouse while pressing the right mouse button will drag the origin of the plane in the same direction than the mouse.

When clicking with the right mouse over a vertex of the polygon this vertex may become a *feature point* or if it was already a *feature point* it ceases to be a *feature point*.

Since the *feature points* are stored in an *ArrayList* the removal and addiction of *feature points* coincides with the removal and addiction in the array, however it is necessary that the order of the feature points respect the vertices order. In the removal there is no problem since the array itself shifts the *feature points* with an index superior to the one removed. To add *feature points* it is necessary to find the index where to add it.

Figure 5.14 represents a polygon with the vertices order in a counter-clockwise order represented by the blue arrow. The point marked in green is the vertex to be added as *feature point.* The index to attribute to this new *feature point* is found following the orange arrows until a *feature point* is reached, in this case the *feature point 0.* These arrows must start in the vertex to be added as *feature point* and follow the clockwise order. When this point is found the index of the new *feature point* must be the index of the *feature point* found incremented by one, or 0 if no *feature point* were found.

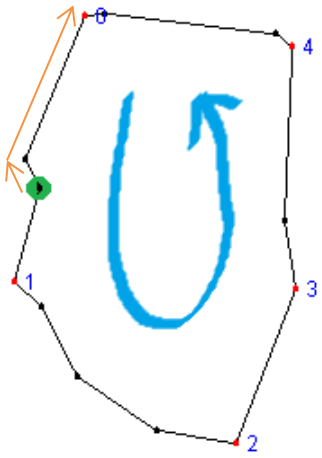


Figure 5.14 – Polygon to add feature points

This same tool allows the user to define the first correspondence between two polygons and if it is necessary it is possible to save the feature points to a file so they can be reopened by this same application, A file containing the correspondences is also created.

The *morphingDemo* is a tool to visualize the result of any of the morphing algorithms implemented with the image from where the polygon was retrieved in background. With this tool it is possible to evaluate and compare the results, and at the same time verify if the problem is of the morphing or of the segmentation.

The Figure 5.15 shows a sequence of frames captured using this tool. In this figure is possible to observe the movement of the polygon during the sequence of the images.

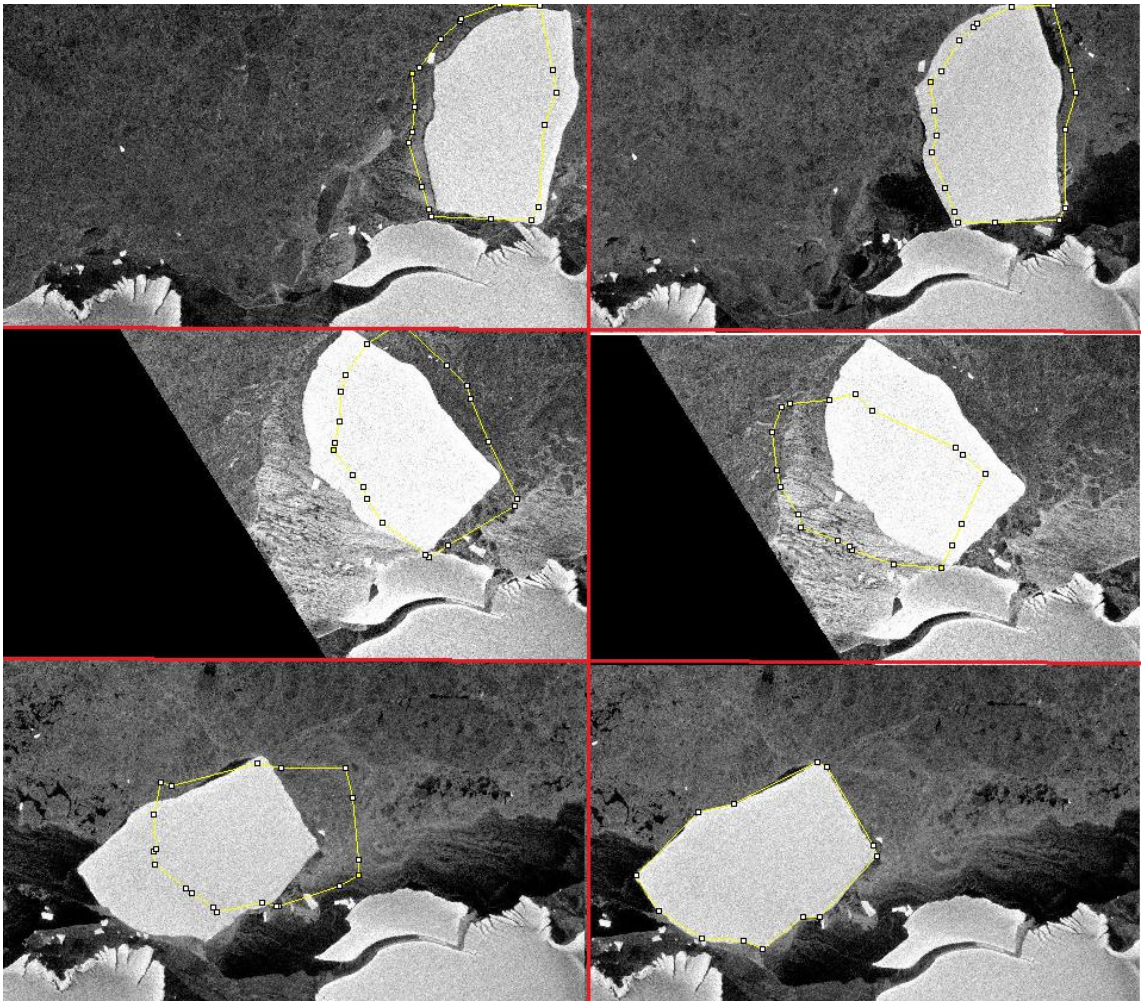


Figure 5.15 – Sequence of morphing

5.4 Summary

This chapter presents solutions for the Vertex Path Problem. This is the second problem of the morphing. Two main approaches are explored to solve this problem. The solution from (McKenney and Webb 2010) and the application of rotation angles.

The first is an algorithm that defines the order from which vertex is processed.

The second solution divides the movement of a source object to a target one into translation, rotation and deformation. A rotation angle is calculated using an alignment method of Section 4.2. This angle is used to rotate the target polygon and a morphing technique is applied to the rotated polygon and the source one. The resulting polygon is rotated using fractions of the rotation angle. Since this is the last step to calculate the morphing the missing stage is to test the implemented algorithms.

6 Results

This chapter presents the results obtained using the algorithms and the tools described in the previous chapters. The aim is to evaluate and compare those algorithms. Section 6.1 presents the results on the extraction of polygons from image sequences. The main focuses of this section are the simplification algorithm presented in Section 3.2.1 and the semi-automatic segmentation procedure described in Section 3.3. Section 6.2 shows the results obtained using the correspondence algorithms described in the Chapter 4. The results of the different morphing algorithms are presented in Section 6.3.

6.1 Extracting objects from images

The extraction aims the conversion from raster to vector representations of the objects. The vector representation must be the most approximate possible from the original object. The simplification reduces the number of points needed to represent the shape of the object and it can be important in some aspects like the performance of the algorithms.

6.1.1 Polygon Simplification

As mentioned in Section 3.2.1, the implemented algorithm uses a similarity coefficient to calculate the threshold for the Douglas-Peucker algorithm. This coefficient defines how similar are the simplified polygon with the original one, comparing the polygon areas. The coefficient recommended in the paper (Zhao, Sheng, and H. Guo 2009) was 0.9 but as it is showed by the icebergs C on Figure 6.1, Figure 6.2 and Figure 6.3, this coefficient returns polygons that do not have enough vertices to define the entire shape of the icebergs. Tests using a greater similarity coefficient, 0.95, returned polygons with more vertices and better definition of their shapes. In the case of Figure 6.4, a similarity coefficient of 0.97 was used (B polygon) since the value 0.95 represented by the polygon C in the figure do not have enough vertices to represent the full extent of the polygon. This is due to the lack of resolution of the images of this dataset.

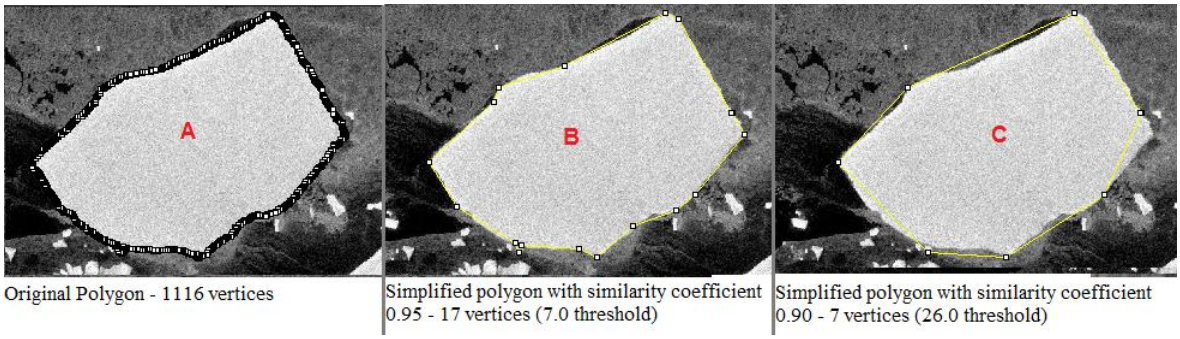


Figure 6.1 – Simplification of polygon A in the c19c dataset

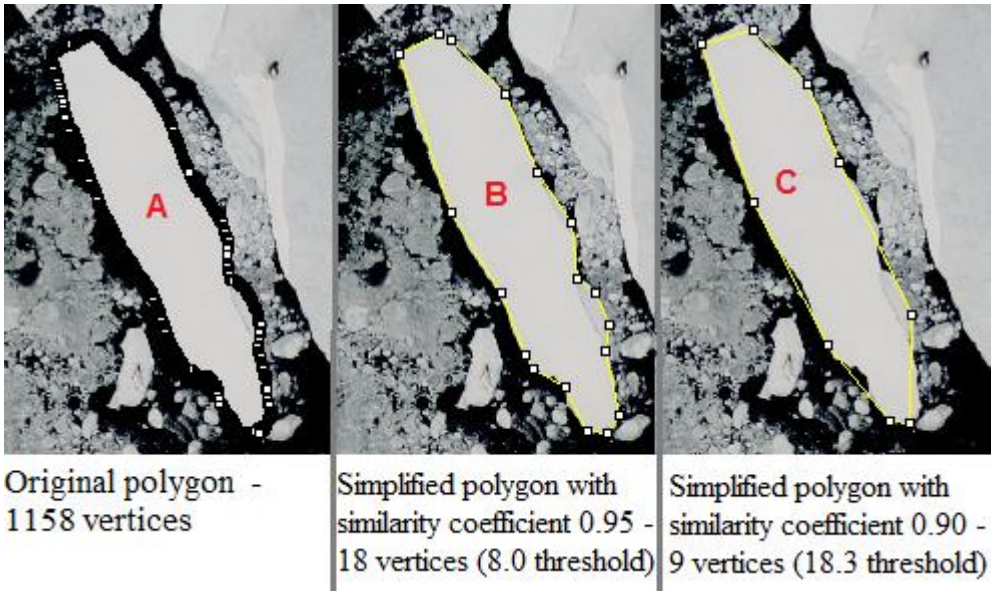


Figure 6.2 - Simplification polygon A in the *ice_b15a* dataset

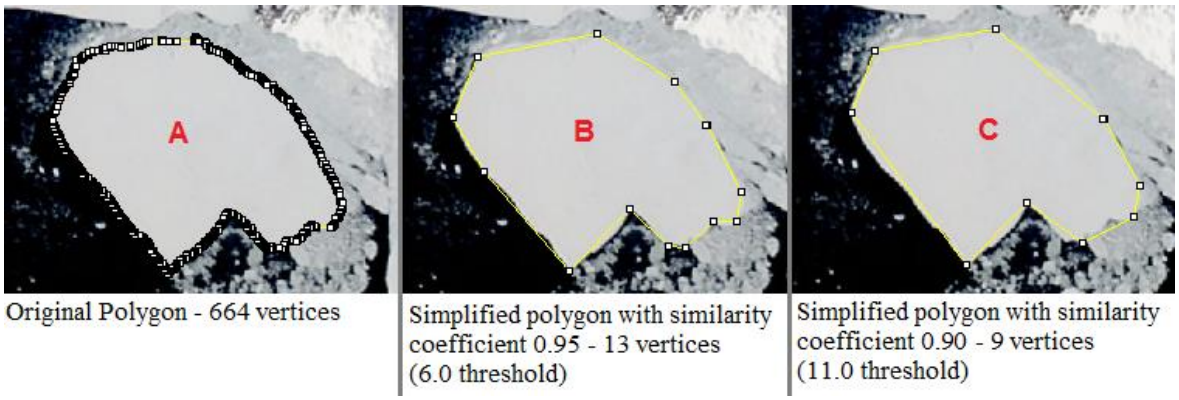


Figure 6.3 – Simplification polygon A in the *ice_b15j* dataset

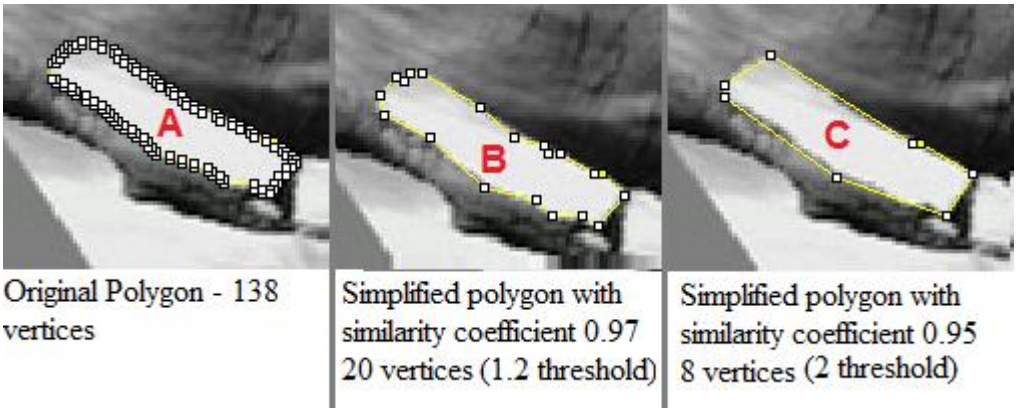


Figure 6.4 - Simplification polygon A in the *ross_b15a* dataset

Table 6.1 shows the final results with the simplification algorithm used. For each iceberg the table indicates the average number of the vertices before applying the simplification algorithm (column 2), the average number of vertices after the algorithm is applied (column 4) the average of the *Douglas-Peucker* algorithm threshold determined (Section 3.2.1) and used in the simplification method (column 3) and the average perimeters of the polygons (column 5). As it was mentioned in Section 3.2.1 the threshold is calculated iteratively, testing different values until a similarity coefficient is reached. The coefficient 0.97 could be used in all the cases, returning polygons with more vertices that the ones represented by the B polygons on Figure 6.1, Figure 6.2 and Figure 6.3. However in the following tests reported in this work, the values that returned the smaller number of vertices without deforming the polygons, the coefficient 0.97 in the *ross* dataset and 0.95 to the rest of the datasets were used.

Iceberg	Vertice Number before Simplification	Douglas-Peucker Threshold Calculated	Vertice Number after Simplification	Average perimeter (pixels)
ice_b15a	1319	7.67	21	1347,82
ice_b15j	665	6.0	17	727,32
ross_b15a	166	1.25	25	143,97
ross_b15b	91	1.50	17	120,76
c19c	1424	7.54	21	1053,91

Table 6.1 – Average number of vertices after and before the simplification

6.1.2 Semi-Automatic Segmentation

The goal of the semi-automatic segmentation is to reduce the interaction of the users during the segmentation process. This process tries to reuse the values used in the segmentation of one

image in the rest the images of this sequence. The first polygon must be determined by the user and the others are automatically determined. However some problems may occur during this process. To detect those problems the determined polygon is compared with the previous one and when they differ significantly an error is prompted to the user which may ignore it and the process continues, or the user may correct the result of the segmentation manually.

Figure 6.5 shows the errors returned when using the semi-automatic segmentation to retrieve the ice_*b15a* iceberg.

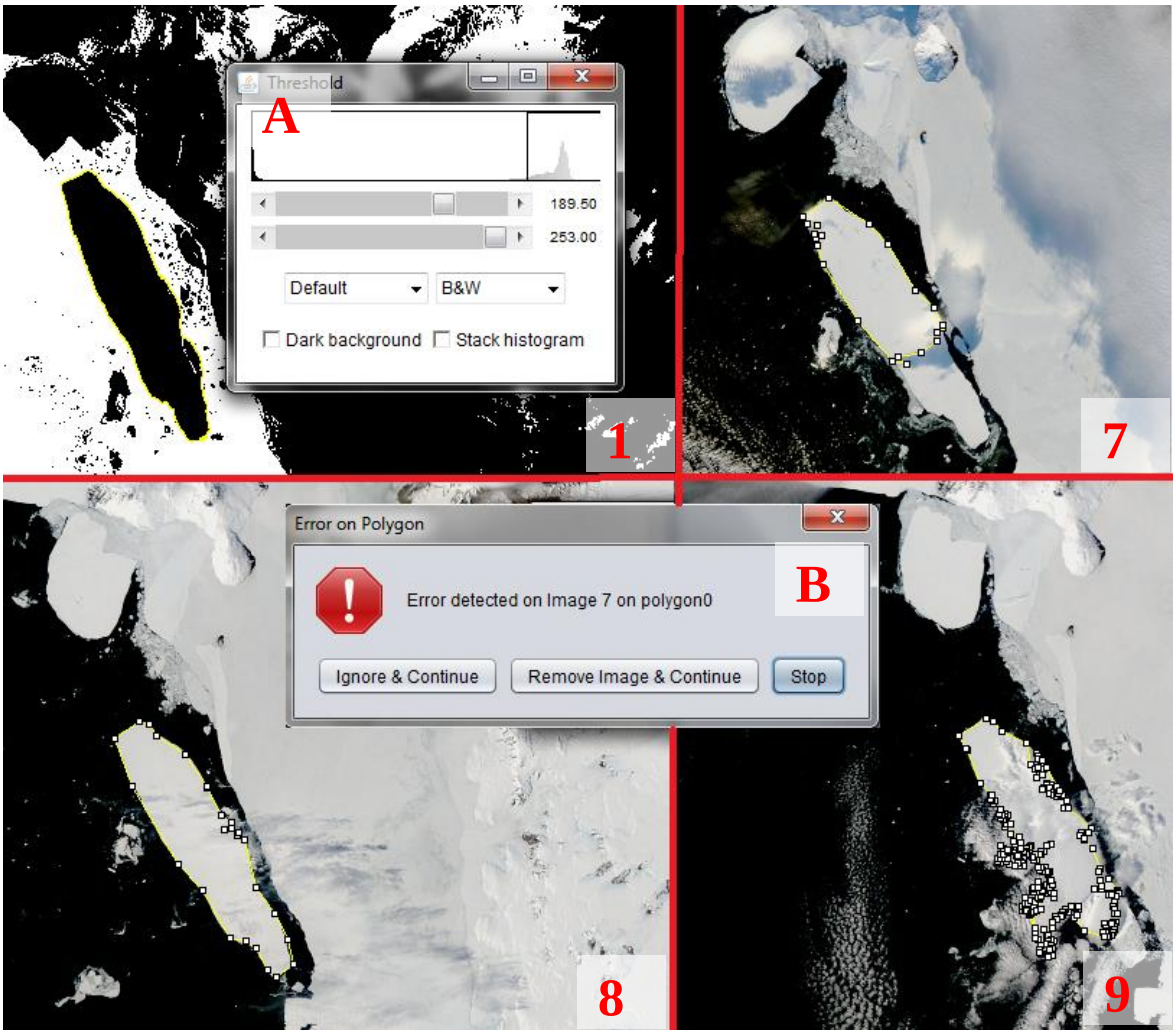


Figure 6.5 – Semi-Automatic Segmentation false negatives

The image 1 represents the first segmentation of the images, when the AB_*Snake* algorithm is applied to the point chosen by the user. The images from 1 to 7 are automatically segmented without problems and the user does not need to interect with the system. In the image 7 the semi-automatic-segmentation located the iceberg, however since there is a cloud in the image the iceberg is cutted almost in half. In this case the user can choose if he wants to continue the segmentation

ignoring this error (false negative), stop the process and edit the polygon, or removing the image from the sequence if no segmentation is possible (options presented by the B menu). In this case the ignore option was chosen and the algorithm returned the error in the image 8. This is clearly a false negative that occur because the polygon from 7 has deformations that had been ignored, so it is ignored. The polygon returned in image 9 has many deformations. These deformations occur because there is a cloud overlapping part of the polygon. In this case only the manual segmentation would return the exact shape of the iceberg, so this image is removed. The last frame of the sequence was segmented successfully. The polygon from 7 was posteriorly adjusted.

The results obtained are described in Table 6.2.

Column 2 represents the number of false negatives detected by the algorithm. These are the errors that the user may opt to ignore since the segmentation is correct but the similarity algorithm returned error. In these cases it is possible to continue the process without segment again the image, keeping the returned polygon and continuing the semi-automatic segmentation. The column 3 shows the total errors detected, including the false negatives. The last column indicates the number of images of each sequence.

Icebergs	False Negatives	Total errors	Total Images
ice_b15a	1	3	10
ice_b15j	2	4	10
c19c	3	6	7

Table 6.2 - Errors detected for each iceberg

For the icebergs *ice_b15a* and *ice_b15j* this method proved to be good, since the user only has to segment 2 polygons and the others were automatically detected. The false negatives are errors were some deformation happens on the polygons and the similarity algorithm returned error that may be ignored. The rest of the errors happened because the images contain noise, like clouds that overlap part of the icebergs, and even after adjusting the threshold or the start positions from the *AB-Snake* algorithm would return errors that would need future corrections of the user.

On the *c19c* dataset, the semi-automatic segmentation returned errors for almost all the images of the sequence. The ignored errors happened when the image detected the polygon with some noise, like waves or near fragments of other icebergs. In those cases the user can easily edit the polygon doing the necessary adjustments, typically removal of vertices. The rest of the errors on this dataset happened because the position of the iceberg is completely different, between consecutive images leading to the need of specifying a different starting point for the Snake

detector algorithm. A possible improvement to this algorithm is to calculate a movement vector for the objects, using its position in different images and apply this vector to the next images.

In the case of the *ross* dataset the results were bad because of the low resolution of the images, noise and differences in the luminosity. The low resolution and the noise turn difficult to detect the boundaries of the polygons even in a manual segmentation. The differences on luminosity need different threshold values to return the boundaries of the icebergs, so its results were removed from the table.

In Figure 6.6 are present two different frames from the *ross* dataset. In these frames is possible to verify the difference on the luminosity occurring in the frames.

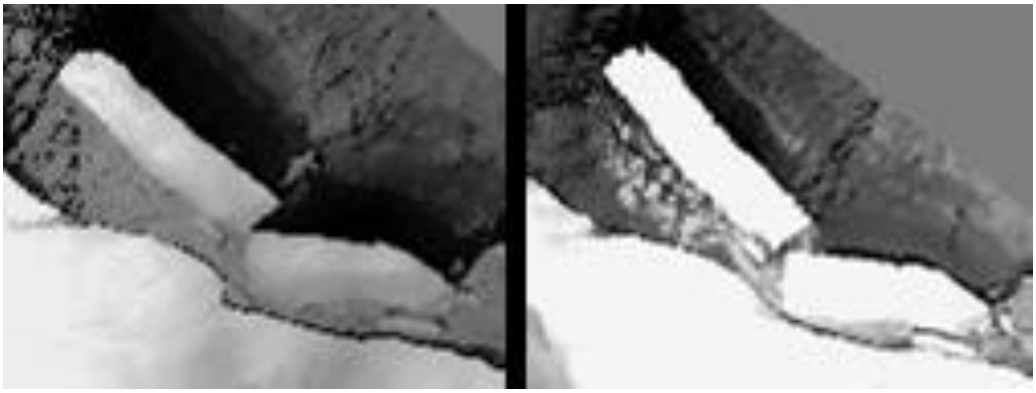


Figure 6.6 – Ross dataset frames

After this step had been concluded, some polygons were manually adjusted, removing and moving some vertices using the methods described in Section 3.2.2. In order to obtain sets of good quality objects to test the morphing algorithms

6.2 VCP

This section will deal with the evaluation of the various methods to determine correspondences. In (Zhao, Sheng, and H. Guo 2009) it is mentioned that the most reliable way to verify correspondences is visually. In this work we use correspondences selected manually between all consecutive polygons. Then the resultant correspondences of the methods described in Chapter 5 were compared with the manual correspondences.

The results of the comparisons between the manual correspondences and the correspondences obtained by the algorithms are presented in this section. The tables containing the results (Table 6.3, Table 6.4, Table 6.5, Table 6.6, Table 6.7, Table 6.8 and Table 6.9) have 4 columns. Column 1 identifies the icebergs and datasets; Column 2 represents the average percentage of correspondences that matched exactly the manual selection; Column 3 depicts the average distances between the correspondent point using this algorithm and the correspondent point

of the manual correspondences; Column 4 shows the relation between the non-matching distances and the perimeters.

In Figure 6.7 it is possible to observe, on left, the manual correspondences determined to two polygons of the *ice_b15a* iceberg and on right, the correspondences obtained using the turning algorithm described in Section 4.1.2.

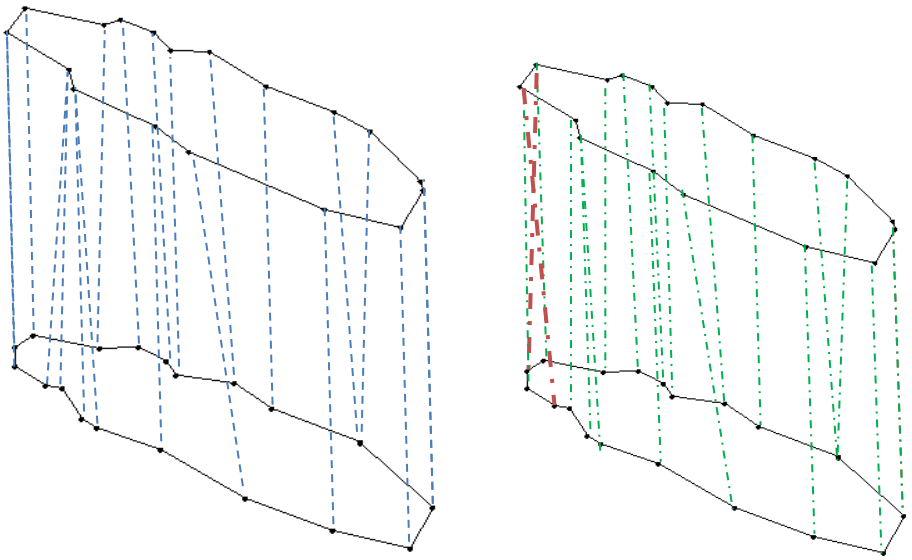


Figure 6.7 - Example of correspondences

In red are the correspondences that do not match with the manual ones. For those correspondences it is calculated the matched distance *d*, as it is depicted in Figure 6.8.

Figure 6.8 – Distance between non-matching correspondences

6.2.1 Turning algorithm

Table 6.3 and Table 6.4 show the results of the correspondences using the turning algorithm described in Section 4.1.2. As it is mentioned in that section this algorithm represents the polygons by their turning function, which creates a relation between the angle and the length of the edges using a referential vertex in each polygons. The correspondence is calculated by finding the minimum areas formed by the functions of two polygons using different referential points.

Iceberg without simplification	Matched correspondences (%)	Average distance (pixels)	Distance / Perimeter
ice_b15a	2	213,62	0,159
ice_b15j	14	93,90	0,129
ross_b15a	11	50,14	0,348
ross_b15b	18	17,04	0,141
c19c	1	76.36	0,072

Table 6.3 - Results of correspondences using the turning algorithm with original polygons

Iceberg simplified	Matched correspondences (%)	Average distance (pixels)	Distance / Perimeter
ice_b15a	89	88,93	0,066
ice_b15j	74	50,91	0,070
ross_b15a	79	11,28	0,078
ross_b15b	85	8,44	0,070
c19c	81	71,27	0,068

Table 6.4 - Results of correspondences using the turning algorithm with simplified polygons

Comparing the values of the two tables, it is possible to conclude that the results with the simplified polygons are better than the results of the original polygons.

The results of Table 6.4 show that this method retrieves correspondences very similar to the manual approach. The percentages of matched correspondences are all above 70% and the relationship between the distances of non-matched correspondences and its equivalent match are all below 0.08 (4^{th} column). With these results it seems that this method is a good method to obtain correspondences between polygons.

6.2.2 Perceptually based algorithm

The algorithm tested in this section was the *perceptually based* algorithm implemented in (Paulo 2012) , described in Section 4.1.1. This algorithm uses vertices that better define the shape of the polygons, *feature points* and using sections formed by those vertices compares the *feature points* from one polygon to the *feature points* in the other.

Table 6.5 and Table 6.6 show the results correspondence results obtained using original and simplified polygons respectively.

Iceberg without simplification	Matched correspondences (%)	Average distance (pixel)	Distance / Perimeter
ice_b15a	4	186,74	0,139
ice_b15j	2	107,62	0,148
ross_b15a	10	23,10	0,160
ross_b15b	14	19,06	0,158
c19c	0	281,36	0,267

Table 6.5 - Results of correspondences using perceptually based algorithm in the original polygons

Iceberg simplified	Matched correspondences (%)	Average distance (pixel)	Distance / Perimeter
ice_b15a	37	196,49	0,146
ice_b15j	27	137,75	0,189
ross_b15a	17	32,19	0,224
ross_b15b	36	21,56	0,179
c19c	8	240,36	0,228

Table 6.6 – Results of correspondences using perceptually based algorithm in the simplified polygons

Comparing the values of the Table 6.5 and Table 6.6 it is possible to conclude that the percentages of the matched correspondences were bigger in the simplified polygons and the averages distances are similar in both cases, the difference between the matched correspondences happens because there are more vertices in the original polygons, creating more different correspondences. However since the average distance remained similar, the correspondences obtained were not too far from the ones using the simplified polygons.

The percentages of correspondences that matched the manual method are all below 40% (column 2) and the relation between their perimeter and the distance between correspondences are all above 0.14 (column 4). Comparing these results with the results obtained using the turning algorithm (Table 6.3) it is possible to conclude that this algorithm alone does not assure good correspondences between the polygons.

6.2.3 Modified *perceptually based* approach with first correspondence

The Table 6.7 shows the results of the improved *perceptually based* algorithm. This improvement was to feed the algorithm with the first correspondence. This first correspondence was calculated using the turning functions, as it is described in Section 4.3.1.2. In this case the polygons without simplification were not tested since they did not return better results to neither one of the algorithm individually.

Iceberg simplified	Matched correspondences (%)	Average distance (pixel)	Distance / Perimeter
ice_b15a	39	200,13	0,149
ice_b15j	49	67,30	0,093
ross_b15a	23	26,51	0,184
ross_b15b	42	16,77	0,139
c19c	57	76,46	0,073

Table 6.7 - Results of perceptually based method with first correspondence

The results show that this change on the algorithm improved all the results of the *perceptually based* algorithm. In the case of the *c19c* iceberg, there is an improvement from 8% to 57% for the matched correspondences and a reduction on the average distances of non-matching correspondences (~22% to ~7%). These results however are still worse than the results of the turning algorithm correspondences (Section 6.2.1).

6.2.4 *Modified perceptually based* algorithm with vertex number matching

To try to improve the results of the perceptually based algorithm another modification was implemented. In this case the method described in Section 4.3.2 was used to match the number of vertices between two polygons and their distribution along the edges. The first correspondence was determined using the polygons alignment. Table 6.8 and Table 6.9 display the results of this method.

Iceberg without simplification	Matched correspondences (%)	Average distance (pixel)	Distance / Perimeter
ice_b15a	0	52,04	0,039
ice_b15j	2	35,70	0,049
ross_b15a	4	5,24	0,036
ross_b15b	3	4,82	0,040
c19c	0	106,52	0,101

Table 6.8 - Results using perceptually based algorithm with matched vertices number on original polygons

Iceberg simplified	Matched correspondences (%)	Average distance (pixel)	Distance / Perimeter
ice_b15a	2	13,28	0,010
ice_b15j	4	9,89	0,014
ross_b15a	5	3,03	0,021
ross_b15b	11	3,04	0,025
c19c	3	12,53	0,012

Table 6.9 - Results using perceptually based algorithm with matched vertices number on simplified polygons

Comparing the data on the Table 6.8 and Table 6.9 is possible to conclude that this method returns better results both in the matched vertices as in distances to the simplified polygons, since the matched correspondences are greater and the distances between non-matching vertices smaller.

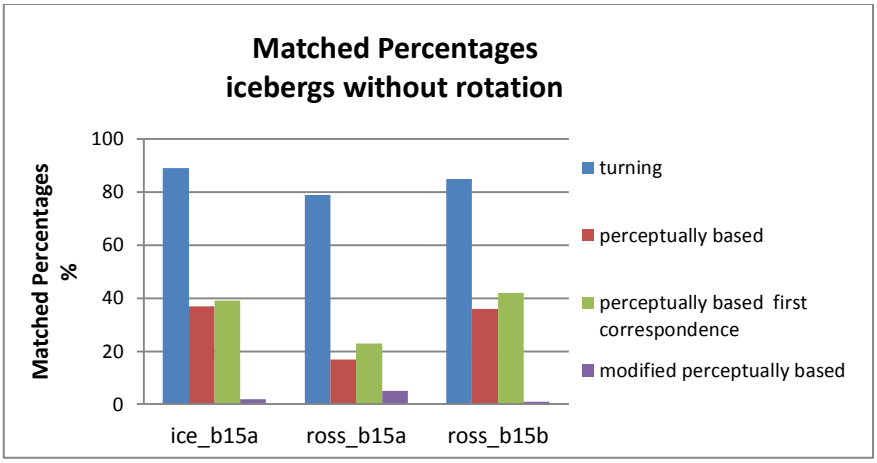
The percentages of matched correspondences are poor because with the addition and removal of vertices it is possible that the manual correspondences used in the other methods may not still be the best ones. This factor makes impossible to compare the turning algorithm and this algorithm using only those values. An analysis of the average distance, however, shows that this algorithm returned matches very similar to the manual correspondences, since there was a great increase on the average distances, proving that this algorithm returned good correspondences even with a decrease of the matched correspondences.

6.2.5 Discussion

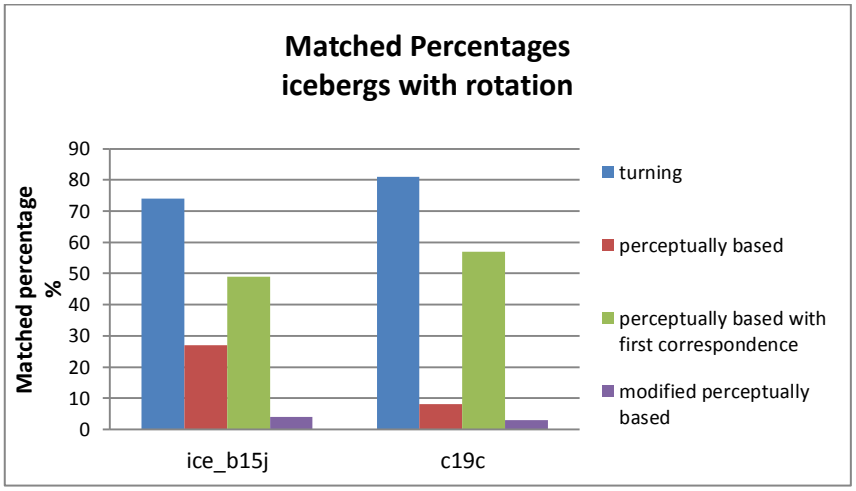
Analyzing the correspondence algorithms it is possible to conclude that the *perceptually based* algorithm alone does not return good results, see Section 6.2.2. The results on Sections 6.2.3 and 6.2.4 proved that it is possible to improve the results of this algorithm by feeding it a first

correspondence or by matching the vertices number and distribution of the polygons. From these improvements the matching of number and distribution of the vertices proved to be the better one.

Graphic 6.1 and Graphic 6.2 show the results of the correspondence algorithms to the icebergs without and with rotation in their movement.



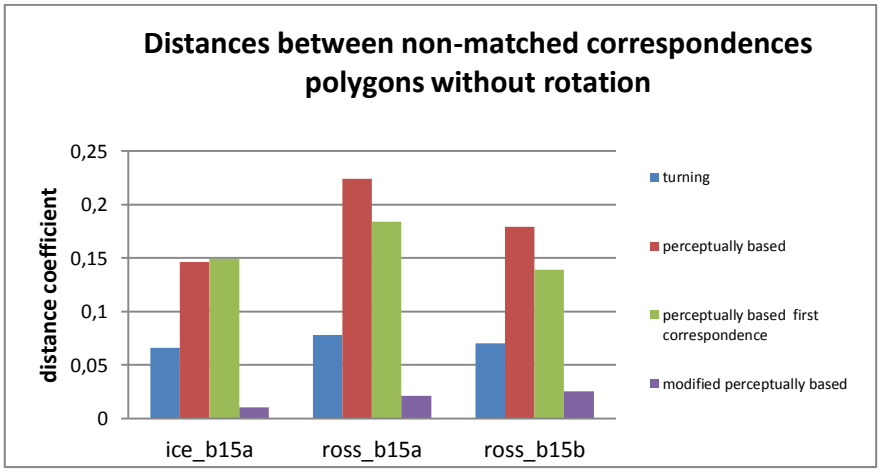
Graphic 6.1 – Matched Percentages to icebergs with movement mainly translational



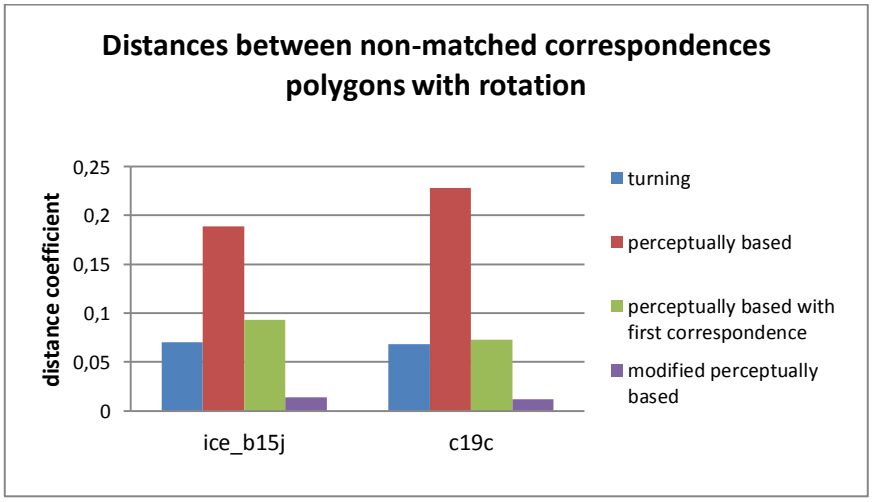
Graphic 6.2 – Matched Percentages to icebergs with movement mainly rotational

Another method tested in this section was the turning algorithm. As it is showed in the Graphic 6.1and Graphic 6.2, this algorithm proved to be better than the simple perceptually based algorithm (Section 6.2.2) and the perceptually based algorithm fed with a first correspondence (Section 6.2.3).

Graphic 6.3 and Graphic 6.4 show the averages distance of non-matching correspondences to polygons without and with rotation on their transformations.



Graphic 6.3 – Distances between non matched correspondences on icebergs with movement mainly rotational



Graphic 6.4 - Distances between non matched correspondences on icebergs with movement mainly rotational

Comparing the distance between non-matched correspondences, Graphic 6.3 and Graphic 6.4, it is possible to conclude that the perceptually based algorithm with vertices number matched and the turning algorithm are the ones with better results.

However with the data acquired it is impossible to determine which one is the best, so both algorithms were tested in the next section.

6.3 Morphing

The morphing algorithms implemented were tested using an intersperse approach. This approach uses three geometric representations of the same moving object extracted in three different time instants, $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$. The movement between the snapshot $\mathcal{P}_1$ and $\mathcal{P}_3$ is calculated

using a selected morphing algorithm. Then the area of $\mathcal{P}_2$ is compared to the area of the morphing representation at $\mathcal{P}_2$ instant. The Figure 6.9 depicts this process.

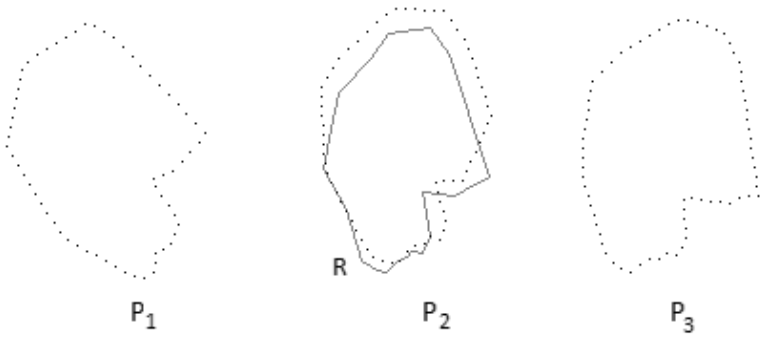


Figure 6.9 – Similarity between captured and estimated polygons

The dotted polygons represent, from the left to the right, the shape of *B-15j* captured on November, 24, December 1 and 3, 2004 ($\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$, respectively). The polygon delimited by a solid line at the middle denotes the shape of the *B15j* on December 1 calculated using any morphing algorithm.

The similarity measure is given by the formula:

$$Similarity = 1 - \frac{Area(\mathcal{R} \cup \mathcal{P}_2) - Area(\mathcal{R} \cap \mathcal{P}_2)}{Area(\mathcal{R} \cup \mathcal{P}_2)}.$$

The methods tested in this section were the linear interpolation and the method described in Section 5.2.1. For those methods the similarity values were calculated using the similarity formula above. In the Table 6.10, Table 6.11, Table 6.12 and Table 6.13 the *Similarity* columns represent the average similarity calculated using the positions of $\mathcal{R}$ and $\mathcal{P}_2$ and the *Similarity (centred)* columns the similarity values calculated using $\mathcal{P}_2$'s centroid and $\mathcal{R}$'s centroid aligned. This option is added since the movement of the polygons may not be exclusively a geometric translation between snapshots. With the *Similarity (centred)* values is possible to evaluate the transformations that occur in the polygons.

Figure 6.10 shows an estimated polygon T in its original position. If the similarity is calculated in this position the result would be 0, although both polygons are similar.

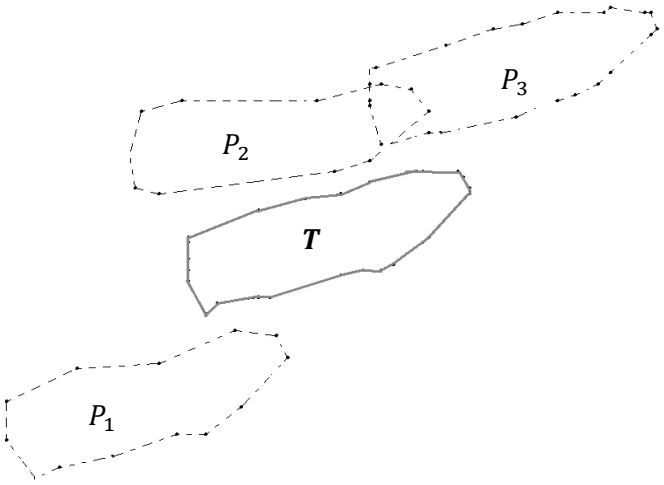


Figure 6.10 – Estimated polygon T in its original position

Figure 6.11 shows the same polygons, but in this case the polygon T is centred with the polygon $P_2$and then their similarity will be not 0 but 74%

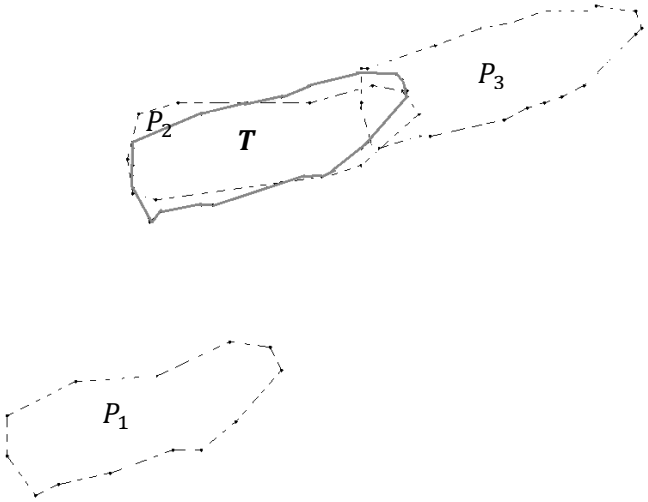


Figure 6.11 – Estimated polygon T centred with the real polygon

6.3.1 Linear Interpolation

As it was mentioned before, the results of the turning algorithm and the *modified perceptually* based algorithm with vertex number matching were inconclusive. So the linear interpolation was tested using both of those algorithms

The Table 6.10 shows the results obtained using both methods.

Iceberg	Turning Correspondences		Improved Feature Based Approach	
	Similarity (%)	Similarity (centred) (%)	Similarity (%)	Similarity (centred) (%)
ice_b15a	89	93	89	93
ice_b15j	72	78	73	79
ross_b15a	73	78	74	79
ross_b15b	74	84	73	84
c19c	39	57	40	60

Table 6.10 – Morphing results of the linear interpolation

Analyzing the table above it is possible to verify that this morphing algorithm returned similar results to all datasets. The results were better for the *ice_b15a* dataset, since the movement of this iceberg is mainly translational and the images of the dataset have good quality. The Figure 6.12 depicts a linear interpolation of a polygon with rotation, in this case the *ice_b15j*. As it is showed in the figure by the polygon T, the polygon tends to shrink and then expand to the final form. This happens because the rotation of the object is modeled using linear paths.

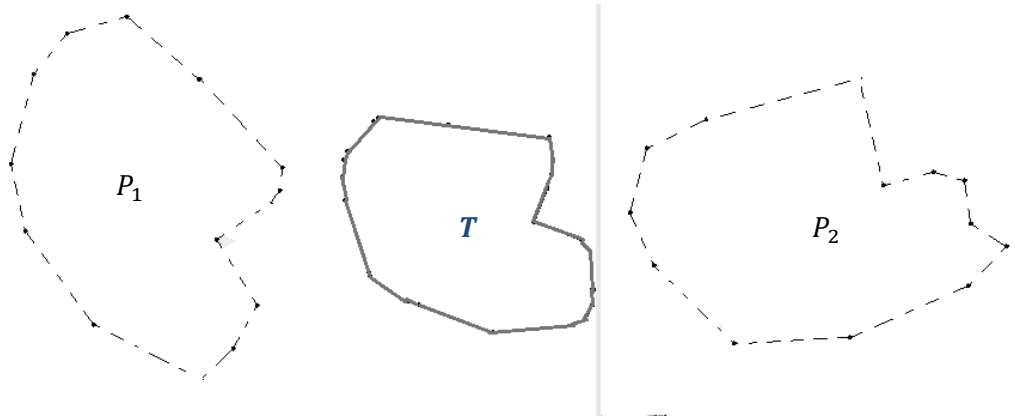


Figure 6.12 - Result of the linear interpolation with rotating objects

As it was expected the similarity values after and before the alignment of the centroids of the polygons are different, increasing when their centroids are aligned, an example of this case is presented in Figure 6.11. The values of both algorithms returned similar similarity values, proving that both are good options to do the correspondences.

6.3.2 Cyclic Order algorithm

The Table 6.11 shows the results obtained using the algorithm described in Section 5.2.1. This table only has 3 columns because the correspondences are calculated by the algorithm itself, while in the previous tests, the algorithm only used linear interpolation to represent the movement of the polygons. This situation may cause VPI's, since the correspondence is done comparing the characteristics of the vertices and there is no regard by the order of those correspondences. Applying the linear interpolation in those cases may originate invalid topologies on the polygons. The *Cyclic Order* algorithm avoids this kind of problems since all the vertices are visited in order.

The results using this algorithm are presented in Table 6.11.

Iceberg	Similarity (%)	Similarity (centred) (%)
ice_b15a	88	91
ice_b15j	76	83
ross_b15a	75	80
ross_b15b	73	82
c19c	48	72

Table 6.11 – Morphing results of the Cyclic Order algorithm

The similarity values obtained using this algorithm proved to be better than the values retrieved using the linear interpolation. These results show that it may be a good idea to investigate more morphing algorithm to simulate the movement and deformation on the moving objects.

However looking at Figure 6.13 and Figure 6.14, it is possible to observe the deformation that this algorithm creates when there is some rotation on the polygons (*ice_b15j*). The similarity values are greater than the values of the linear interpolation because the area of the polygon estimated is always proportional.

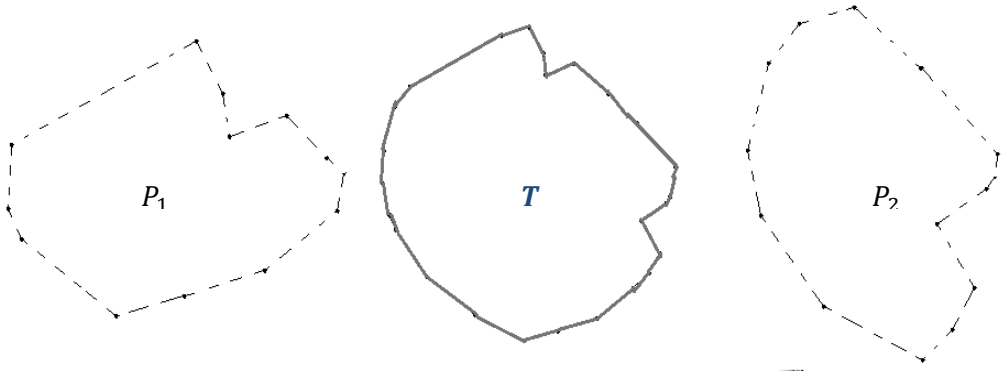


Figure 6.13 – Example of the Cyclic Order algorithm with objects with rotation ice_b15j

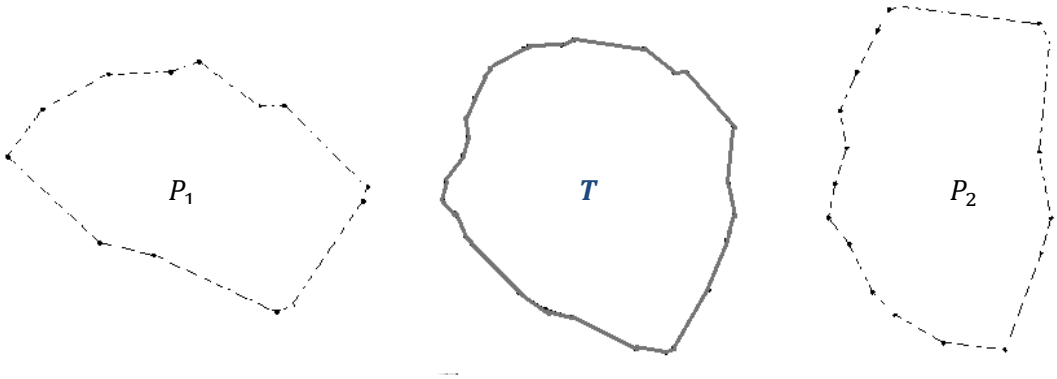


Figure 6.14 - Example of Cyclic Order algorithm with objects with rotation c19c

The Figure 6.15 shows that the behaviour of this algorithm with objects with only translation movements (*ross_b15b*) returns the expected results without deformations.

Figure 6.15 - Example of the Cyclic Order algorithm with objects with only translation

6.3.3 Linear Interpolation and Rotation

Dividing the movement of an object into translation and rotation is the other method tested.

It is important to note that the results of this method are directly related with a good detection of the rotation angle between two polygons. This angle is calculated using the ICP algorithm described in Section 4.2.2.2.

The Table 6.12 displays the results using this algorithm.

Iceberg	Turning Correspondences		Improved Feature Based Approach	
	Similarity (%)	Similarity (centred) (%)	Similarity (%)	Similarity (centred) (%)
ice_b15a	87	90	87	90
ice_b15j	78	85	78	86
ross_b15a	70	75	69	76
ross_b15b	71	83	70	83
c19c	45	75	78	74

Table 6.12 – Morphing results linear interpolation plus rotation

The Figure 6.16 shows the intermediate polygon T (*ice_b15 j*), calculated using the improved feature based approach with rotation angles. This was applied to the same polygons as in the Figure 6.12 and this image shows that the deformation was avoided using this method. Similar results occur using the turning algorithm.

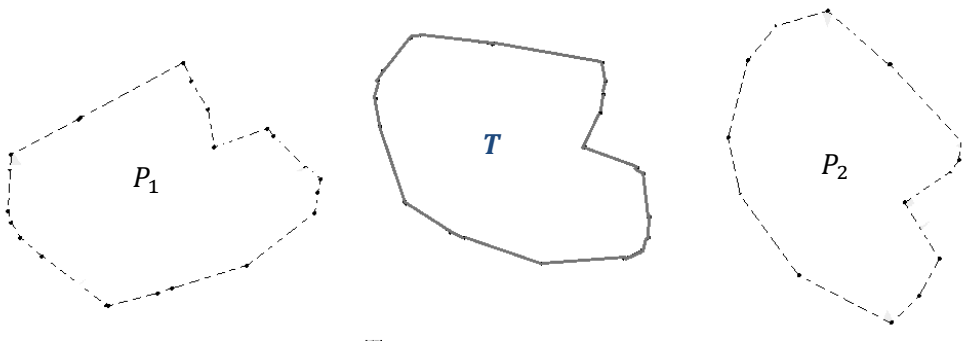


Figure 6.16 – Linear interpolation with rotation angle in polygons with rotation

The Figure 6.17 shows that when no rotation exists in the polygons the algorithm does not generate large deformations.

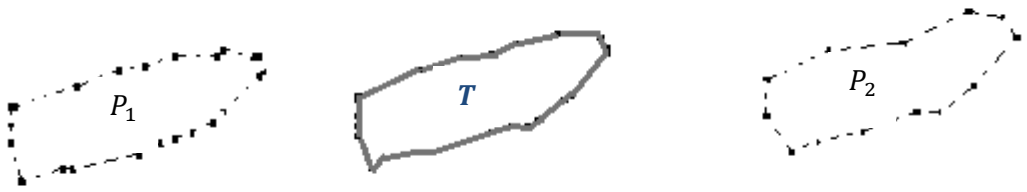


Figure 6.17 - Linear interpolation with rotation angle in polygons without rotation

As showed in the Table 6.12 and in Figure 6.16, this method improved the results to all the polygons that have significant rotations on their movement (*c19c* and *ice_b15 j)*, for the other polygons there was a small decreases that may be justified by errors on calculating the rotation angles.

6.3.4 Cyclic Order Algorithm with rotation

The Table 6.13 shows the results of the *Cyclic Order* algorithm applied to two aligned polygons.

Iceberg	Similarity (%)	Similarity (centred) (%)
ice_b15a	87	89
ice_b15j	78	85
ross_b15a	73	78
ross_b15b	71	82
c19c	45	75

Table 6.13 - Morphing results of the Cyclic Order with rotation

Analyzing these results applying the rotation to this algorithm does not have a great effect in the results. There were only small improvements in some of the similarity values.

Visually there were notable improvements in the polygons with rotation, Figure 6.18. But some deformation still happens with concavities. But when they do not exist the deformations are very small (Figure 6.19).

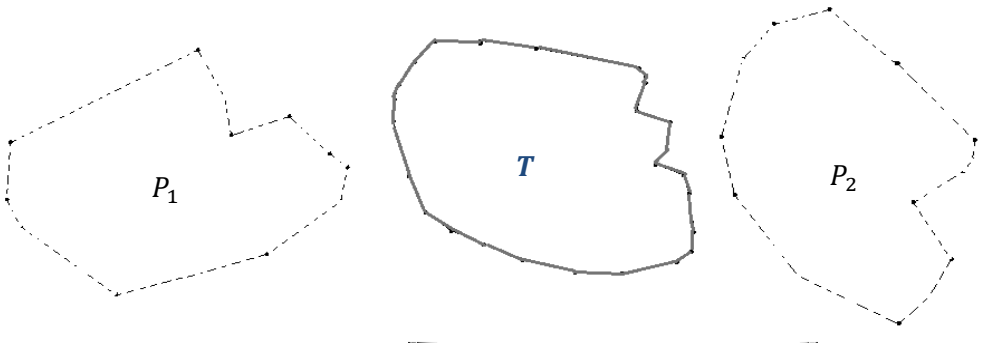


Figure 6.18 – Application of the Cyclic Order algorithm with rotation angles on polygons with rotation

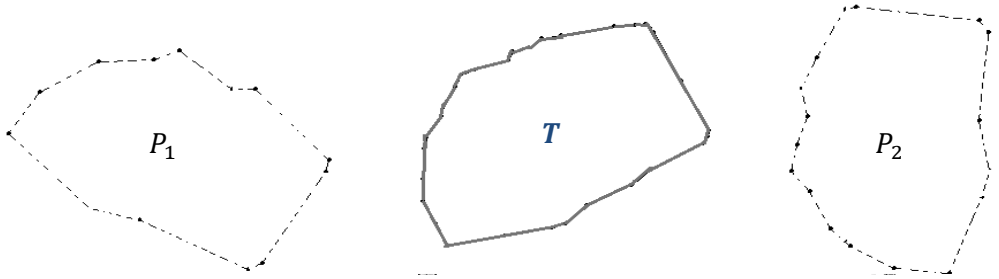
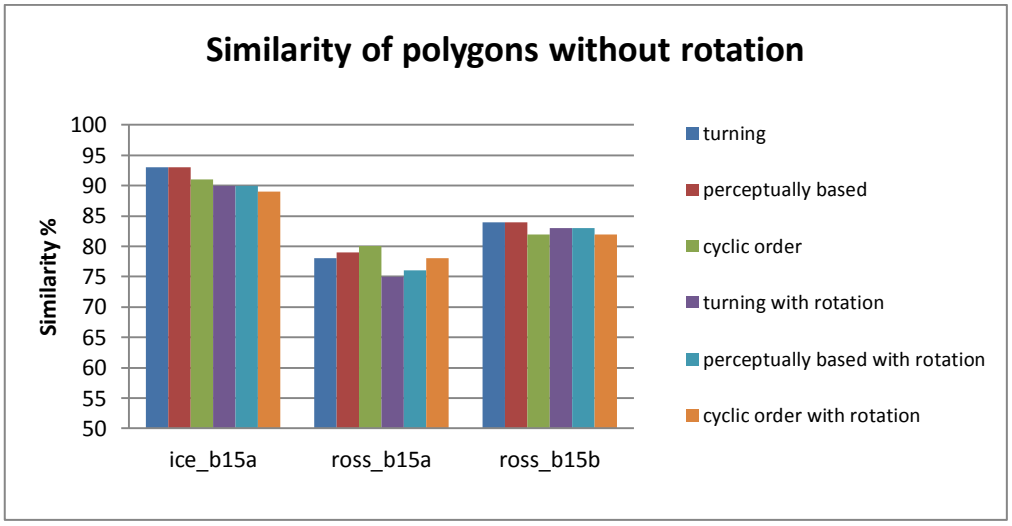


Figure 6.19 - Application of the Cyclic Order algorithm with rotation angles on polygons with rotation with smaller concavities

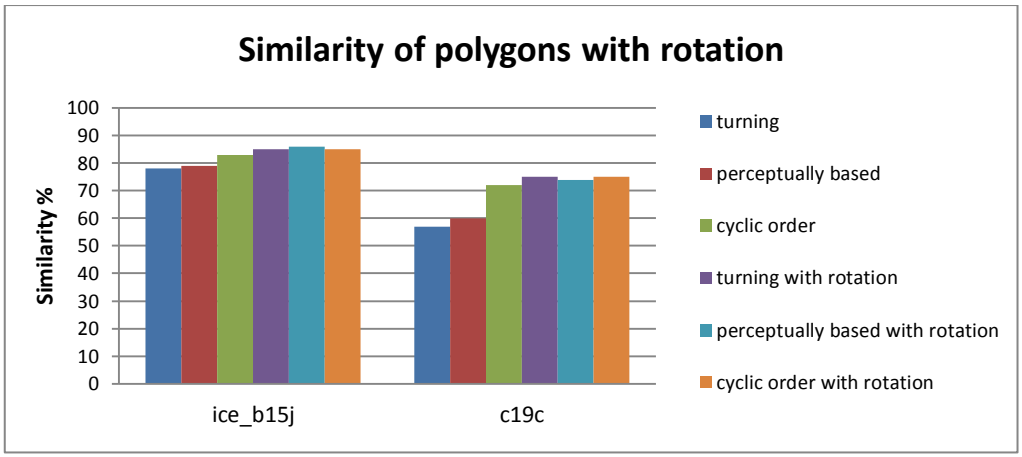
6.3.5 Discussion

Analysing the results of the morphing algorithm it is possible to conclude that linear paths are a good choice when the movement of the objects is approximately translational, this is showed in the Graphic 6.5, where the results are very similar with all the algorithms.



Graphic 6.5 - Similarity of polygons without rotation

However when some rotation exists such approach may be improved with the use of a rotation angle, as it is showed in the Graphic 6.6, where with rotation angles the algorithms return greater similarity results.



Graphic 6.6 - Similarity of polygons with rotation

The turning algorithm and the modified perceptually based approach returned similar results between them for all cases, what turns any of those algorithms a good choice when the linear interpolation is to be used. The results of the *Cyclic Order* algorithm are similar to the results of the linear interpolation with rotation angle, however the intermediate polygons of this algorithm will be more complex having at least the number of vertices equal to the sum of the origin and the target.

7 Conclusions

7.1 Contributions

The linear interpolation is the method implemented by current solutions to model the changes of an object between consecutive snapshots. In this work we present some improvements and some morphing alternatives to this method, comparing all of them using real-world datasets.

The semiautomatic segmentation introduced in this dissertation aimed to the automation of the segmentation process of real world datasets with a great amount of images. The preliminary results show that the main difficulties occur when the images in the sequences do not share the same resolution and luminosity and the dataset does not have enough images so that the position of the object does not change completely in two consecutive snapshots. The automation of this process is important so it is possible to acquire great amounts of real world data with the minor user interaction, what would allow the validation of methods using real world data with great dimension instead of just using synthetic datasets.

Regarding the correspondences (finding matching points between consecutive polygons), four different algorithms were implemented and tested with the datasets introduced in Section 2.3 from real image. Analyzing correspondences results the turning algorithm and the *improved perceptually based* algorithm with vertex distribution matched proved to be the best solutions and with similar results. Using their correspondences in the morph also returned similar results, showing that any of these algorithms is a good choice. However the *improved perceptually based* algorithm needs preprocessing of the polygons to match the vertices number and their distribution along the boundaries of the polygon, an extra step that may introduce some errors for example in the initial alignment. Due to this additional complexity, and with the results obtained, the algorithm that seems more suitable is the turning algorithm presented in Section 4.1.2.

Regarding the morphing techniques, the tests with linear interpolation alone present some limitations, in particular when the polygons suffer rotation between snapshots. In this situation, it is

recommended to divide the movement of the polygons in translation plus rotation and deformation. This approach returned the best results for the morphing algorithm for the datasets containing rotation on the movements of their objects. When no rotation is present this method returned similar results to the simple linear approach. All the morphing algorithms tested returned similar results. In that case any of them may be used to compute the movement of the moving objects.

During the correspondences evaluation, however, it was concluded that the *improved perceptually based* algorithm was more complex than the turning algorithm. So it is only necessary to compare the linear interpolation of the turning correspondences and the *Cyclic Order* algorithm. In that case it is possible to conclude that the turning algorithm is still the best one to create the morphing. This conclusion happens by the fact that the *Cyclic Order* algorithm introduced in (McKenney and Webb 2010) needs to calculate the convex hulls of the polygons and any vertex not belonging to a concavity will have at least two correspondent vertices in the other polygon, forming a triangle, which is more complex to process than only a line.

7.2 Future Work

This dissertation focuses in the acquisition of the geometric representations of the moving objects from real world datasets and compares different algorithms to represent the movement of those geometries. However the term moving object usually implies the integration of this kind of entities into databases.

The work described in this dissertation is a preliminary step to allow the future integration of these objects into databases. But the actual moving objects databases and information systems only deal with linear interpolations of the objects. The results presented in this work show that this approach has limitations and it would be important to study the possibility to introduce other algorithms besides linear interpolations in rotating moving objects databases.

References

Adamek, Tomasz, Noel E. O'Connor, and Noel Murphy. 2005. "Region-based segmentation of images using syntactic visual features." *WIAMIS 2005 - 6th International Workshop on Image Analysis for Multimedia Interactive Services,*: 13–15.

Aloupis, Greg (Computational Geometry Lab). "Melkman 1987." http://cgm.cs.mcgill.ca/~athens/cs601/Melkman.html (May 9, 2013).

Andrey, Philippe, and Thomas Boudier. 2006. "Adaptive active contours (snakes) for the segmentation of complex structures in biological images." *ImageJ Conference*.

Anton, Howard, and Chris Rorres. 2010. 773 *Elementary Linear Algebra: Applications Version*. John Wiley & Sons.

Arkin, E M, L P Chew, D P Huttenlocher, Klara Kedem, and J S B Mitchell. 1991. "An efficiently computable metric for comparing polygonal shapes." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 13(3): 209–216.

Besl, P.J., and H.D. McKay. 1992. "A method for registration of 3-D shapes." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14(2): 239–256.

Cai, Mengchu, Dinesh Keshwani, and Peter Z. Revesz. 2000. "Parametric Rectangles: A Model for Querying and Animation of Spatiotemporal Databases." *Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology*: 430–444.

Chetverikov, Dmitry. 2003. "A Simple and Efficient Algorithm for Detection of High Curvature Points in Planar Curves." In *Computer Analysis of Images and Patterns SE - 91*, eds. Nicolai Petkov and MichelA. Westenberg. Springer Berlin Heidelberg, p. 746–753.

Chomicki, Jan, Sofie Haesevoets, Bart Kuijpers, and Peter Revesz. 2003. "Classes of Spatio-Temporal Objects and their Closure Properties." *Annals of Mathematics and Artificial Intelligence* 39(4): 431–461.

Douglas, David H, and Thomas K Peucker. 1973. "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature." *Cartographica: The International Journal for Geographic Information and Geovisualization* 10(2): 112–122.

Forlizzi, Luca, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. 2000. "A data model and data structures for moving objects databases." In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data - SIGMOD '00*, New York, New York, USA: ACM Press, p. 319–330.

Gotsman, Craig, and Vitaly Surazhsky. 2001. "Guaranteed intersection-free polygon morphing." *Computers & Graphics* 25(1): 67–75.

Grumbach, Stéphane, Philippe Rigaux, and Luc Segoufin. 2001. "Spatio-Temporal Data Handling with Constraints." *Geoinformatica* 5(1): 95–115.

Güting, Ralf Hartmut, Thomas Behr, and Christian Düntgen. 2010. "SECONDO: A Platform for Moving Objects Database Research and for Publishing and Integrating Research Implementations." *IEEE Data Eng. Bull.* 33(2): 56–63.

Güting, Ralf Hartmut, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. 2000. "A foundation for representing and querying moving objects." *ACM Transactions on Database Systems* 25(1): 1–42.

Iben, Hayley N, James F O'Brien, and Erik D Demaine. 2006. "Refolding planar polygons." In *Proceedings of the twenty-second annual symposium on Computational geometry - SCG '06*, Sedona, Arizona, USA: ACM Press, p. 71–79.

"JMAT | Free Science & Engineering software downloads at SourceForge.net." http://sourceforge.net/projects/jmat/ (May 10, 2013).

"JMAT to compute the Principal Component Analysis." http://jmat.sourceforge.net/_index.html (May 10, 2013).

Ju, Tau. 2012. "Lecture 7: Alignment." 18. http://www.cse.wustl.edu/~taoju/cse554/lectures/lect07_Alignment.pdf (May 10, 2013).

Kanjamala, Pradip, Peter Z. Revesz, and Yonghui Wang. 1998. “MLPQ/GIS: A GIS using Linear Constraint Databases.” *C. S. R. Prabhu, Proc. 9th COMAD international conference on managemnet of data*: 389–393.

Landini, Gabriel. 2013. “Auto Threshold.” http://fiji.sc/Auto_Threshold (May 15, 2013).

Liu, Ligang, Guopu Wang, Bo Zhang, Baining Guo, and Heung-yeung Shum. 2004. “Perceptually Based Approach For Planar Shape Morphing, Computer Graphics and Applications.” *Proceeding PG ’04 Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*: 111–120.

Málková, Martina, Jindřich Parus, Ivana Kolingerová, and Bedřich Beneš. 2009. “An intuitive polygon morphing.” *The Visual Computer: International Journal of Computer Graphics* 26(3): 205–215.

Matos, Luís, José Moreira, and Alexandre Carvalho. 2012. “A spatiotemporal extension for dealing with moving objects with extent in Oracle 11g.” *SIGAPP Appl. Comput. Rev.* 12(2): 7–17.

McKenney, Mark, and James Webb. 2010. “Extracting moving regions from spatial data.” In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, New York, NY, USA: ACM, p. 438–441.

Melkman, Avraham A. 1987. “On-line construction of the convex hull of a simple polyline.” *Information Processing Letters* 25(1): 11–12.

“MLPQ System (Version 5).” 2009. http://www.cse.unl.edu/~revesz/MLPQ/mlpq.htm (May 6, 2013).

Münster, Angelika Humbert Uni, and Christine Wesche. 2009. “ekstroemisen_atka_oct2009.” http://www.awi.de/fileadmin/user_upload/Infrastructure/Stations/Neumayer_Station/Neumayer/ekstroemisen_atka_oct2009.gif (May 15, 2013).

Oliveira, André. 2011. “Edição e visualização de obejctos geográficos dinâmicos”. Technical Report, Departamento de Eletrónica, Telecomunicações e Informática, University of Aveiro, Aveiro.

Paulo, Luis. 2012. 70 “Aplicação de técnicas de morphing em bases de dados espacio-temporais.” Master Thesis, Departamento de Eletrónica, Telecomunicações e Informática, University of Aveiro, Aveiro.

Pelekis, Nikos. 2002. “STAU: A spatio-temporal extension toORACLE DBMS.” PhD thesis, Department of Computation, UMIST, England

Pelekis, Nikos, Elias Frentzos, Nikos Giatrakos, and Yannis Theodoridis. 2010. 136 “Supporting Movement in ORDBMS – the “HERMES” MOD Engine”. Technical Report, Department of Informatics,

University of Piraeus, Hellas.

Pelekis, Nikos, Yannis Theodoridis, Spyros Vosinakis, and Themis Panayiotopoulos. 2006. “Hermes – a framework for location-based data management” eds. Yannis Ioannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthes, Mike Hatzopoulos, Klemens Boehm, Alfons Kemper, Torsten Grust, and Christian Boehm. *Proceeding EDBT’06 Proceedings of the 10th international conference on Advances in Database Technology* 3896: 1130–1134.

Revesz, P. Z., and Yiming Li. 1997. “MLPQ: a linear constraint database system with aggregate operators.” *Proceeding IDEAS ’97 Proceedings of the 1997 International Symposium on Database Engineering & Applications*: 132.

Revesz, Peter. 2010. “Moving Objects Databases.” In *Introduction to Databases*, Springer London, p. 111–135.

Ross, Amy. 2004. 8 *Procrustes Analysis*. http://www.cse.sc.edu/~songwang/CourseProj/proj2004/ross/ross.pdf.

“Ross movie.” 2007. http://www.atsr.rl.ac.uk/images/sample/ross/index.shtml (May 15, 2013).

“RossSea Subsets.” http://rapidfire.sci.gsfc.nasa.gov/imagery/subsets/?project=antarctica&subset=RossSea&date=11/15/2004 (May 15, 2013).

Rusinkiewicz, S, and M Levoy. 2001. “Efficient variants of the ICP algorithm.” In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on,* , p. 145–152.

Sederberg, Thomas W., Peisheng Gao, Guojin Wang, and Hong Mu. 1993. “2-D shape blending: an intrinsic solution to the vertex path problem.” In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’93*, New York, New York, USA: ACM Press, p. 15–18.

Sederberg, Thomas W., and Eugene Greenwood. 1992. “A physically based approach to 2--D shape blending.” *ACM SIGGRAPH Computer Graphics* 26(2): 25–34.

Shahriyar, Rifat. 11 “Space Efficient Convex Hull Algorithms”. Technical Report, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Bangladesh. http://www.buet.ac.bd/cse/users/faculty/rifat/CGP.pdf.

Shapira, M., and A. Rappoport. 1995. “Shape blending using the star-skeleton representation.” *IEEE Computer Graphics and Applications* 15(2): 44–50.

Shim, Choon-Bo, and Jae-Woo Chang. 2000. “Spatio-temporal representation and retrieval using moving object’s trajectories.” In *Proceedings of the 2000 ACM workshops on Multimedia - MULTIMEDIA ’00*, New York, New York, USA: ACM Press, p. 209–212.

Shlens, Jonathon. 2005. “A Tutorial on Principal Component Analysis”. Technical Report, Systems Neurobiology Laboratory, Salk Insitute for Biological Studies, San Diego, California.

Silvani, Xavier, Frédéric Morandini, and Jean-Luc Dupuy. 2012. “Effects of slope on fire spread observed through video images and multiple-point thermal measurements.” *Experimental Thermal and Fluid Science* 41(0): 99–111.

Sistla, A. Prasad, Ouri Wolfson, Sam Chamberlain, and Son Dao. 1997. “Modeling and Querying Moving Objects.” *Data Engineering, 1997. Proceedings. 13th International Conference on*: 422–432.

Tøssebro, Erlend, and Ralf Hartmut Güting. 2001. “Creating Representations for Continuously Moving Regions from Observations.” *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*: 321–344.

Veltkamp, Remco C., and Michiel Hagedoorn. 2000. “Shape Similarity Measures, Properties and Constructions.” 467–476.

Zhao, Dongbao, Yehua Sheng, and Hengliang Guo. 2009. “An algorithm for automatically matching corresponding points on homonymous map features.” In *Proc. SPIE 7146, Geoinformatics 2008 and Joint Conference on GIS and Built Environment: Advanced Spatial Data Models and Analyses, 71461J*, eds. Lin Liu, Xia Li, Kai Liu, and Xinchang Zhang.

Annex A. Preliminary results using fire datasets

A.1 Datasets

This annex presents some preliminary results that were obtained using new series of datasets. Those datasets are three movies showing the fires spread documented in several experiments performed to study the effects of slope on fire spread (Silvani, Morandini, and Dupuy 2012). These experiments were performed in a laboratory using a 3m X 9m platform. The snapshots were captured at regular time intervals and the predominant movement of the fire front is translation. In this experience three different videos were used, each one with different slopes.

These datasets are different from the iceberg since the shapes and movement of the objects is different. These are an interesting sort of objects to study, since this is one of the main environmental problems affecting Portugal in the summer season.

The Figure A.1 is a sequence of the fire images using the slope of 30 degrees. These three images are the first, the middle and the last one of a sequence of 26 frames.

Figure A.1 – Fire sequence using 30 degree on slope

The Figure A.2 depicts the first the middle and the last images on a sequence of 29 frames using the slope of 20 degrees.

Figure A.2 – Fire sequence with 20 degrees of slope

Figure A.3 has the first the middle and last frames of a fire experiment using no slope. In this case the sequence has 37 frames.

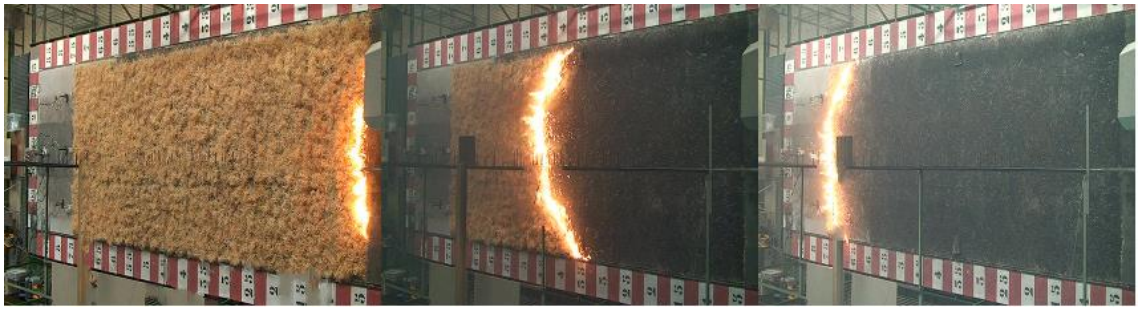


Figure A.3 - Fire sequence without any slope

A.2 Extracting objects from images

Using these videos it was possible to obtain two types of moving objects: the flames and the burned area of the fires.

The flames were segmented using the segmentation tool described in Section 3.1. Since the flames of the fires are very irregular polygons, the simplification technique described in Section 3.2 was used with similarity coefficient of 0.85.

The Table A.1 shows some properties of the flames segmented.

Slope	Frames number	Average Vertices number	Perimeter (pixels)
0	37	37	1024
20	29	49	1643
30	26	59	2019

Table A.1 - Flames properties

Figure A.4 shows two fire images segmented from the images with 30 degree slope in left and from the sequence without any slope on right.

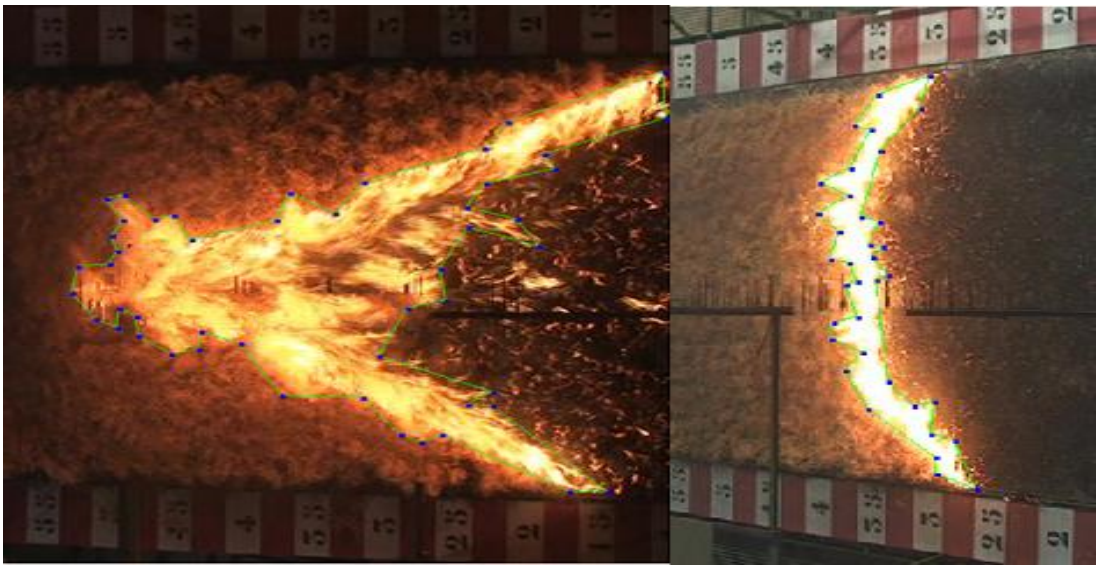


Figure A.4 – Segmented flames

A.2.1 Burned area

Another type of objects that can be retrieved from these image sequences is the burned area. The burned area was obtained using the previous segmentation connecting the upper most and the down most vertices to the margins of the images.

The Table A.2 shows the principal properties of this kind of objects.

Slope	Average Vertices	Perimeter (pixels)
0	29	1623
20	16	1689
30	21	1680

Table A.2 – burned area properties

Figure A.5 shows two different images with the burned area segmented, represented by the shadowed areas.

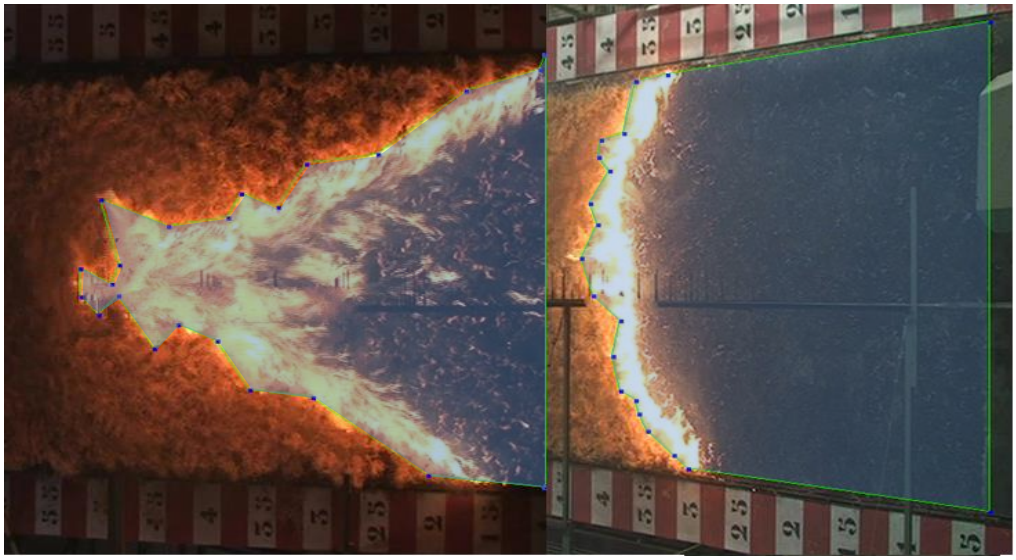


Figure A.5 – Burned area segmented

A.2.2 Discussion

The tool used to extract the polygons from the flames and the burned area of the videos returned the shape of the objects. However the main studies of these objects only need to use abstractions of them, then a MBR of the shapes or a tool to determine the skeleton of the flames would be more desirable methods, since they return simpler polygons easier to analyse in the remaining processes.

A.3 VCP

This section shows the results obtained using the correspondence algorithms described in Chapter 4. The tests are the same applied to the iceberg datasets described in Section 6.2.

A.3.1 Turning algorithm

Table A.3 shows the results obtained using the turning algorithm described in Section 4.1.2 in the flames of the fires.

Slope	Matched correspondences (%)	Distance / Perimeter
0	38	0,037
20	30	0,032
30	28	0,036

Table A.3 - Results of the turning algorithm correspondence on the flames

With the values presented in the tables above is possible to conclude that the results of this algorithm are not good to the flames since the matched correspondences are all below 40%. This happens because the shape of the flames has a lot of changes between consecutive frames.

Table A.4 present the turning algorithm results in the burned areas.

Slope	Matched correspondences (%)	Distance / Perimeter
0	63	0,048
20	71	0,061
30	52	0,052

Table A.4 - Results of the turning algorithm correspondence on the burned areas

To the burned areas the results are similar to the ones obtained with the icebergs. This happens because only part of the polygon changes between two consecutive frames.

A.3.2 Perceptually based algorithm

The correspondences, obtained using the *perceptually based* algorithm, (Section 4.1.1) are presented in the next two tables.

Table A.5 shows the results to the flames of the fires.

Slope	Matched correspondences (%)	Distance / Perimeter
0	10	0,227
20	4	0,278
30	9	0,199

Table A.5 – Results using the perceptually based algorithm in the flames

Table A.6 presents the results to the burned areas.

Slope	Matched correspondences (%)	Distance / Perimeter
0	38	0,103
20	34	0,147
30	17	0,215

Table A.6 – Results to the perceptually based algorithm on the burned areas

The matched correspondences to the flames are bad (all below 10%), to the burned areas these correspondences increased.

A.3.3 Modified perceptually based approach with first correspondence

The algorithm tested in this section is the perceptually *based* algorithm fed with a first correspondence from the turning algorithm. This algorithm is described in Section 4.3.1.2

Table A.7 shows the results from this algorithm to the flames of the fires.

Slope	Matched correspondences (%)	Distance / Perimeter
0	11	0,156
20	14	0,130
30	16	0,117

Table A.7 – Results using the perceptually based algorithm fed with a first correspondence to the flames

Table A.8 presents the results to the burned areas of the fires.

Slope	Matched correspondences (%)	Distance / Perimeter
0	31	0,110
20	36	0,133
30	30	0,145

Table A.8 - Results using the perceptually based algorithm fed with a first correspondence to the burned areas

The results presented in the two tables above show that in some cases the matched correspondences are very similar with the values of the *perceptually based* algorithm, (slope 0 from both cases and slope 20 from the burned areas). To the other cases the results improved.

A.3.4 Modified perceptually based algorithm with vertex numbers matching

This section presents the values, obtained using the *perceptually based* algorithm, in polygons with the same number of vertices. The algorithm is described in Section 4.3.2.

Table A.9 shows the results using this algorithm in the flames of the fires to each slope.

Slope	Matched correspondences (%)	Distance / Perimeter
0	1	0,021
20	1	0,048
30	1	0,027

Table A.9 – Results using the perceptually based algorithm in polygons with the same number of vertices in the flames

Table A.10 presents the results to the burned areas of the fires using this algorithm.

Slope	Matched correspondences (%)	Distance / Perimeter
0	3	0,013
20	1	0,032
30	2	0,030

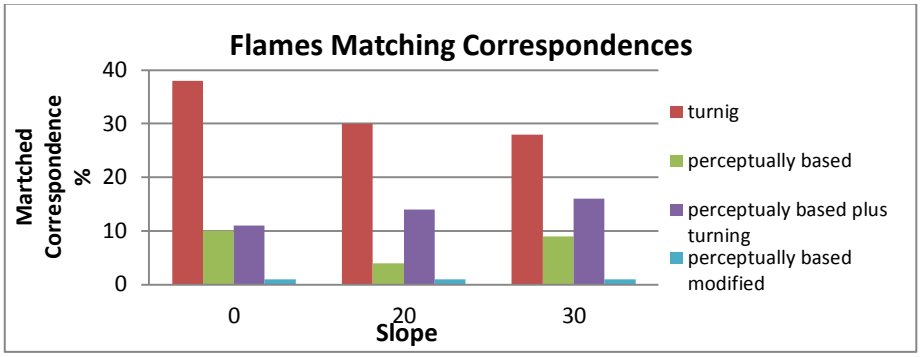
Table A.10 - Results using the perceptually based algorithm in polygons with the same number of vertices in the burned areas

The results of the tables above show the smaller matched correspondences, however as it happened in the icebergs the distances between non matched correspondences are the smaller.

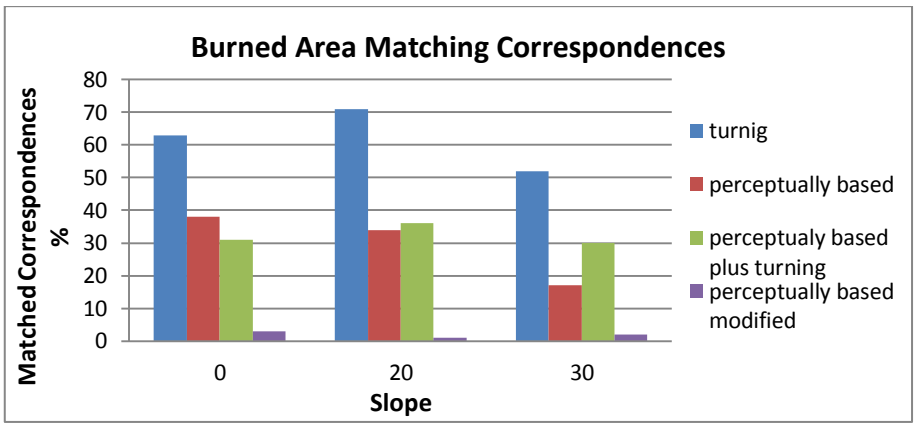
A.3.5 Discussion

Graphic A.1 and Graphic A.2 show the matched correspondences percentages of the flames and the burned areas respectively. In both graphics is possible to verify the results to the VCP algorithm tested. With these graphics is possible to conclude that using matching correspondences the turning algorithm returned the best results. The graphics show that given the first

correspondence using the turning algorithm to the *perceptually based* approach improved almost all the results.

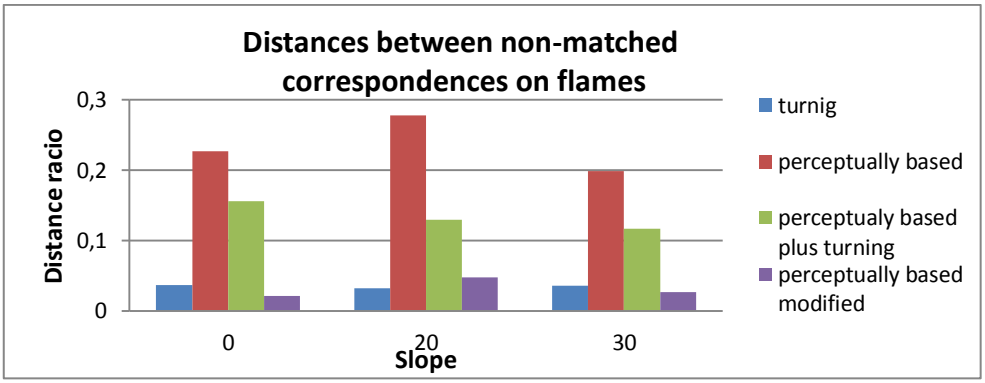


Graphic A.1 - Flames Matching Correspondences

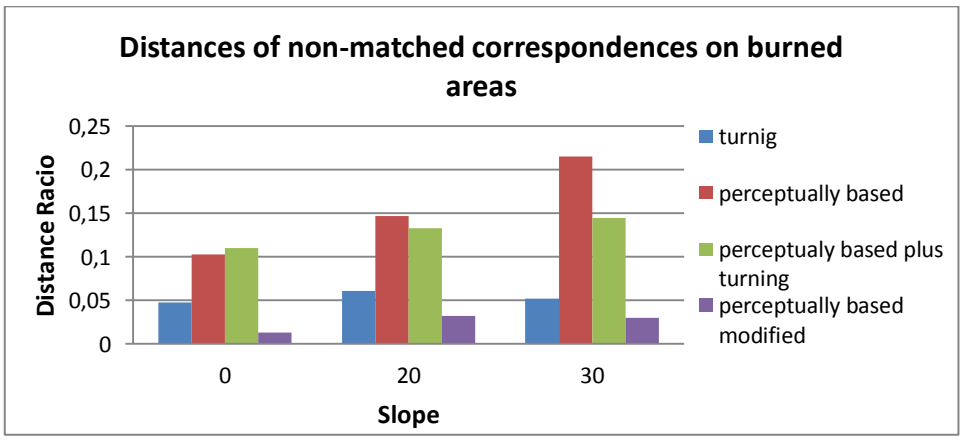


Graphic A.2 - Burned Area Matching Correspondences

Graphic A.3 and Graphic A.4 show the average distance ratio between non matching correspondences. It is possible to conclude that the turning algorithm and the modified *perceptually based* algorithm have similar results and are the best ones.



Graphic A.3 - Distances between non-matched correspondences on flames



Graphic A.4 - Distances of non-matched correspondences on burned areas

A.4 Morphing

This section presents the results using the method described in Chapter 5. This method was used to both the flames and the burned areas of the fire sequences. The morphing algorithms tested were the linear interpolation using the correspondences of the turning algorithm and the improved *perceptually based* approach and the *cyclic order* algorithm. Since the datasets do not contain rotations the algorithms using rotation angles were not tested.

A.4.1 Linear interpolation

The results, obtained using the linear interpolation are presented in Table A.11 (flames) and Table A.12 (burned areas).

slope	Turning Correspondences		Improved Feature Based Approach	
	Similarity (%)	Similarity (centred) (%)	Similarity (%)	Similarity (centred) (%)
0	57	54	60	58
20	60	54	48	45
30	63	61	66	64

Table A.11 – Morphing results obtained using the linear interpolation in the flames

slope	Turning Correspondences		Improved Feature Based Approach	
	Similarity (%)	Similarity (centred) (%)	Similarity (%)	Similarity (centred) (%)
0	85	85	96	97
20	91	91	90	90
30	75	74	86	85

Table A.12 - Morphing results obtained using the linear interpolation in the burned areas

The results on the tables above show that the linear interpolation returned better results to the burned areas. This happens because in those cases the shapes of the polygons are simpler.

Figure A.6 shows a linear interpolation between two flames. In the middle is possible to see the polygon interpolated in blue and the shape and position of the polygon at that instant in yellow.

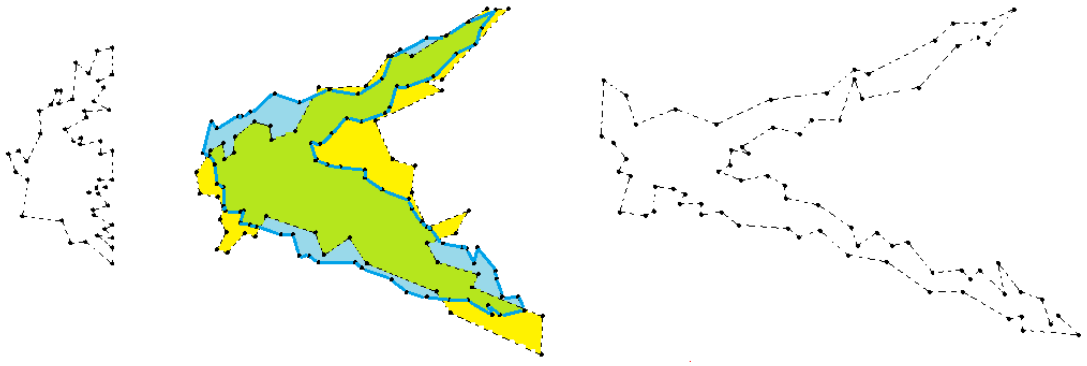


Figure A.6 – Linear interpolation of the flames

Figure A.7 shows the morphing between two polygons of the burned areas. The green area is the overlapped area between the real shape of the polygon at instant *t* and the estimated polygon at that instant determined by the linear interpolation.

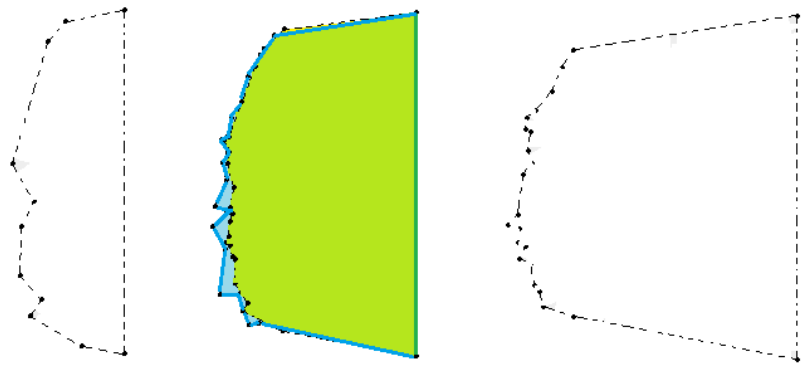


Figure A.7 – Linear interpolation of the burned areas

The results to the different correspondence algorithms returned similar values. The values of the centred and not centred similarities are similar because the time interval and the movement of the objects between consecutive frames are constant.

A.4.2 Cyclic Order algorithm

The Table A.13 and Table A.14 show the results using the *Cyclic Order* algorithm.

scope	Similarity (%)	Similarity (centred) (%)
0	45	44
20	43	40
30	54	51

Table A.13 – Results of the cyclic Order algorithm for the flames

scope	Similarity (%)	Similarity (centred) (%)
0	97	97
20	95	95
30	88	87

Table A.14 - Results of the cyclic Order algorithm for the burned areas

The results using the *cyclic order* algorithm for the burned areas were all above 85%, proving to be a good option to create the morph between those objects. Using the flames the results were worst (between 40 and 55%), this happened because the flames have a lot of concavities.

Figure A.8 depicts the morphing using the *cyclic order* algorithm. In this picture is possible to observe the deformation happening because the polygon has a big concavity.

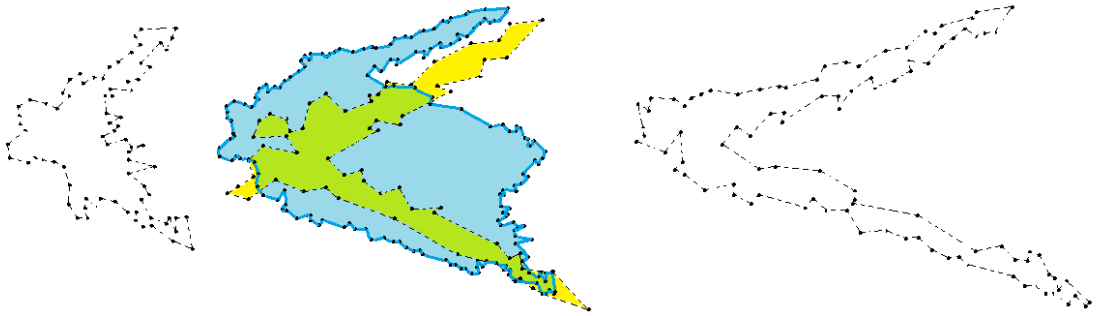


Figure A.8 – Morphing using the cyclic order algorithm on the flames

Figure A.9 shows the estimated polygon using this algorithm (blue polygon). In this case since the concavity causing the deformation on the flame case does not exist, the resulted polygon is more similar to the real one.

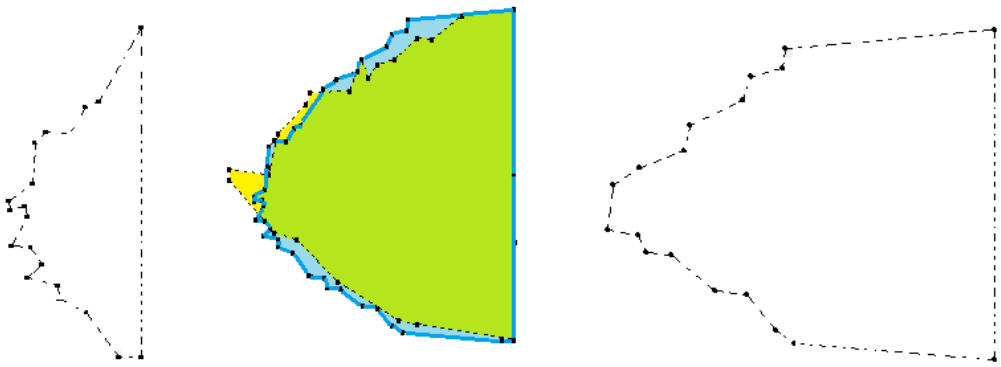
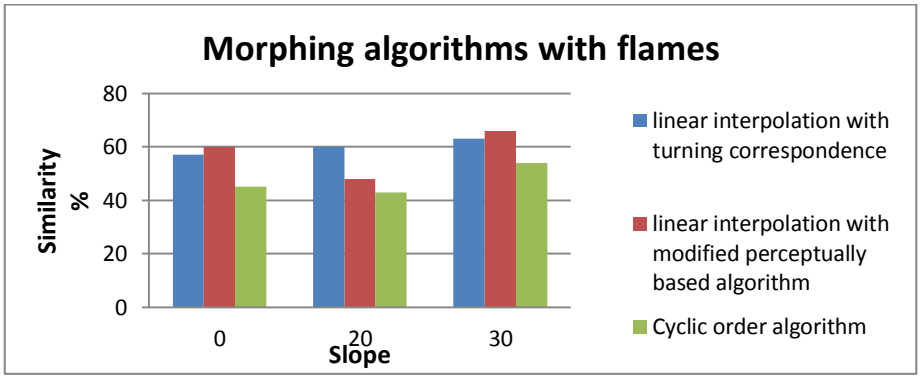


Figure A.9 - Morphing using the cyclic order algorithm on the burned area

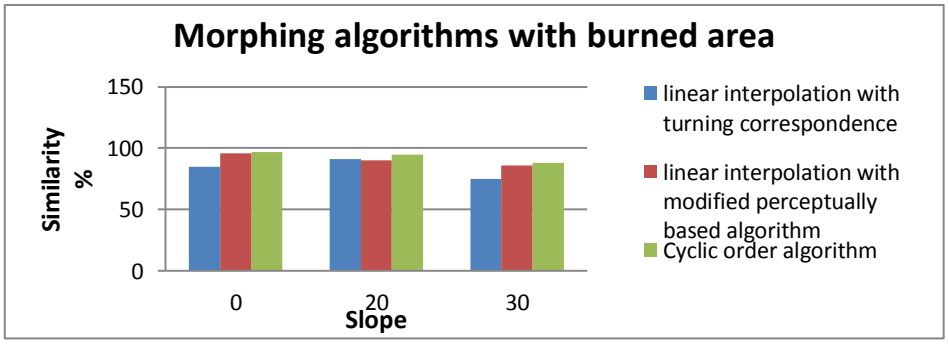
A.4.3 Discussion

Graphic A.5 and Graphic A.6 shows the results of the morphing algorithm for the flames and the burned areas respectively. These graphics show that both algorithm returned similar results in case of the burned areas, but to the flames the *cyclic order* algorithm returned the worst results.

The burned area results are better than the flames results to any of the algorithms tested.



Graphic A.5 - Morphing algorithms with flames



Graphic A.6 - Morphing algorithms with burned area

A.5 Conclusions

This annex presents some results to the fire datasets using the same tools and methods applied in icebergs datasets. The results show that it is necessary some work, mainly in the segmentation to improve the results. It is important to define exactly what is important to extract from the images, since to some studies may be only necessary to use an abstraction of the objects.

The turning correspondences algorithm and the improved *perceptually based* algorithm with polygons with the same number of vertices were the VCP algorithms with better results, but due to the complexity of the modified *perceptually based*, the algorithm that seems more suitable is the turning algorithm presented in Section 4.1.2. Regarding the morphing techniques used both algorithm of the linear interpolation returned similar results and any of them may be used to create the movement of the objects. However duo the concavities present in the flame sets the *cyclic order* algorithm returned the worst results of all the algorithms tested, proving that this algorithm is a good choice if the polygons are simple with few concavities. In the case of the burned areas any if these algorithms would be a good choice since they returned similar results.