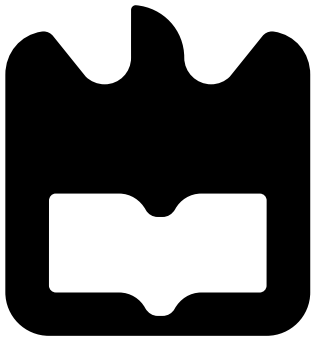
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2012

Luís Carlos Marques Paulo

Aplicação de técnicas de morphing em bases de dados espacio-temporais
Morphing techniques in spatiotemporal databases



o júri / the jury

presidente / president

Prof. Doutor Joaquim João Estrela Ribeiro Silvestre Madeira
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

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 / acknowledgements

Queria agradeçer à minha família e amigos pelo apoio que me deram ao longo deste tempo, sem eles teria sido muito mais dificil chegar até aqui.

Queria também agradeçer ao José Moreira e ao Paulo Dias, meu orientador e co-orientador, pelo esforço e paciência para comigo ao longo deste ano.

Resumo

Com o desenvolvimento das tecnologias de captura remota de dados permitindo a captura e transmissão de dados geo-referenciados ao longo do tempo, apareceram muitas aplicações que necessitam de ferramentas para tratamento de dados espácio-temporais eficientes. Isto levou ao desenvolvimento de uma grande variedade de métodos e serviços para implementação de sistemas de informação que trabalham com objetos que podem ser modelados como objetos moveis. No entanto, a investigação em bases de dados com objetos moveis de geometrias complexas focou-se essencialmente em modelos de dados espácio-temporais e linguagens de interrogação, existem ainda diversas questões sobre a aquisição de dados espácio-temporais.

Esta tese propõe um conjunto de ferramentas para aquisição de dados espácio-temporais em mudança contínua a partir de fontes de dados discretos, tais como imagens de satélite. Para gerar os dados utilizam-se técnicas de *morphing* de polígonos, já existentes. A técnica usada é apresentada em detalhe nesta tese. Para inserção dos dados espácio-temporais na base de dados usada foi criado um *framework* em JAVA que permite converter esses dados em dados compatíveis com a base de dados espácio-temporal. A *framework* desenvolvida também permite a inserção desses dados na base de dados. São também apresentadas uma serie de medidas para avaliar a qualidade dos dados gerados. Com a criação de data sets reais procura-se aumentar a qualidade das bases de dados espácio-temporais existentes e futuras. Neste contexto é também apresentado um método para utilização de técnicas de *morphing* para resolver operações espácio-temporais na base de dados.

Abstract

With the development of remote sensing technologies allowing capturing and transmitting geo-referenced data repeatedly along time, there are many applications demanding for efficient tools to deal with spatio-temporal data. This has led to development of a whole spectrum of methods and services for implementation of information systems dealing with objects that may be modelled as moving points. Although, database research on moving objects with complex shapes has mainly focused on spatio-temporal data models and query languages, there are still issues to be solved regarding for example the acquisition of spatio-temporal data.

This thesis proposes a set of tools to acquisition of continuously changing spatio-temporal data from discrete sources, such as satellite images. To generate the continuous data from the discrete data source we resort to existing polygon morphing techniques, which are presented in detail in this thesis. To insert the generated data into the spatio-temporal database, a JAVA framework was created that can transform the generated data into spatio-temporal data compatible with the representations used in spatio-temporal databases. We also present a set of measures to evaluate the quality of the generated data. With the creation of real data sets we strive to improve the quality of the existing and future spatio-temporal databases. In that context we also present a method to use morphing techniques to solve spatio-temporal operations in the database.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years a lot of work has been done in the area of spatio-temporal databases and handling of spatio-temporal data. Examples of applications are tracking of moving objects like forest fires, floods, oil leaks propagation or icebergs movement. Tracking these events may help in understanding phenomena such as the melting rate of an iceberg or the fire propagation rate in some sort of terrain. To study these kind of events remote devices capable of capturing and send geo-referenced data (GIS) are commonly used. These devices send the captured information over time to its receptors. A problem when using these devices is that the data received is discrete, and the phenomena are continuous events. To solve this problem the discrete data have to be transformed into continuous data to allow a better comprehension of the event. A solution to this problem is to apply a morphing techniques to recreate the event based on the existing observations. In doing so a real data set of spatio-temporal data is created. That data set can be used to test and develop spatio-temporal databases. Due to lack of such data sets the developers of the spatio-temporal databases normally use only simple synthetic data sets.

This thesis demonstrates that morphing techniques are a viable solution to some problems in spatio-temporal databases. Chapter 2 gives an overview of morphing techniques, a detailed presentation of the morphing algorithm used in this work and the results of the tests performed to evaluate the quality of the moving objects data representations. The algorithms were implemented in Java. Chapter 3 focuses on the application of the algorithm to implement the clipping operation in spatio-temporal databases. First a discussion about the existing spatio-temporal databases will be done, followed by a presentation of the spatio-temporal database used in the thesis. Next follow the details of the implementation of a clipping operation in a spatio-temporal extension developed in Oracle 11g, including the methods to integrate the Java classes implementing the morphing into the database and an application to create and insert spatio-temporal data in the database. Finally results for the insertion of spatio-temporal data and the intersection operation are presented and discussed. This work aims to create the tools that can aid in the development of spatio-temporal applications and databases, as well as improve the quality of the existing ones.

Chapter 2

Acquisition of continuously changing spatio-temporal data

To acquire spatio-temporal data from a discrete data source such as images, first the regions of interest must be identified and segmented from the image, then these regions are converted to polygons, where the morphing algorithm is applied to generate the continuous data and recreate the event. In this thesis a set of satellite images of an iceberg where used to create the dataset.

2.1 Image segmentation

The fist step to recreate an event using the discrete data retrieved by the sensing devices is to obtain the data in each of these discrete moments. In this case since satellite images were used as the source of data to generate the spatio-temporal data, an image segmentation application was used. This application uses some morphological image operators to extract the icebergs contours, grey scale conversion and active contours, see Figure 2.1 and 2.2. This application was developed by André Filipe da Silva Oliveira, for the course project of the first cycle degree in Technologies and Information Systems, see [1].

The segmented polygon is generated by applying a vertex reduction algorithm, presented in Section 3.5, to the resulting iceberg contour. By using this algorithm the contour of an iceberg can be reduced to a polygon using less than 10% of its initial vertices, with a minimal definition loss.

2.2 Morphing overview

Polygon or shape morphing consists in a "metamorphosis" of a shape into another through a deformation process. The result of this process is an interpolation where the original shape is morphed into the target shape. Using two shapes there are a lot of different animation sequences that can be considered valid solutions to the morphing problem. A good morphing sequence must avoid vertex interception during its animation or great deformations in the shape form. If these conditions are not fulfilled then the sequence will not seem real to the user. In last years some techniques have been developed to try to solve this problem, however these techniques lack some automatism and require manual work to obtain a solution. The

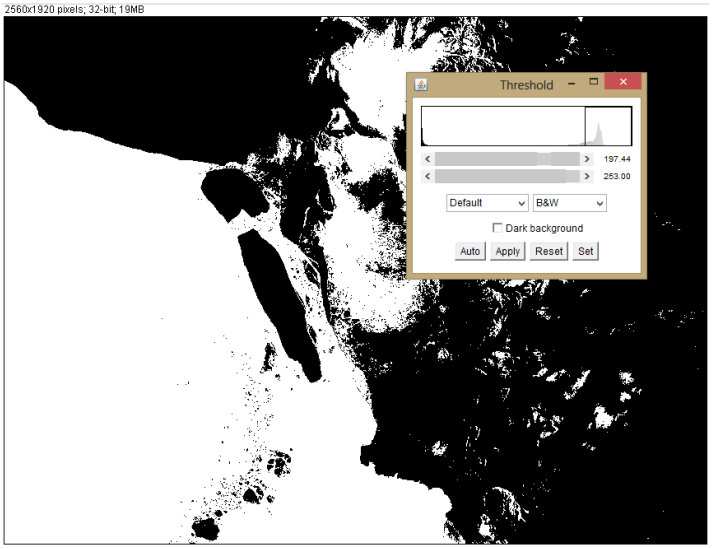


Figure 2.1: Applying threshold operation to image

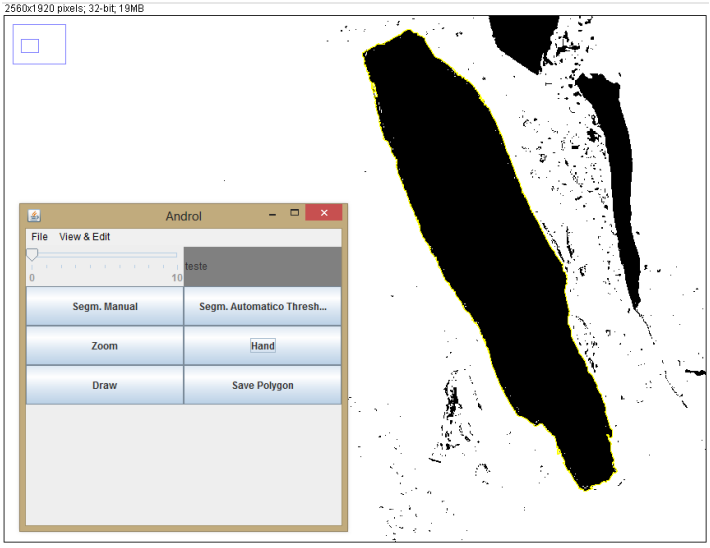


Figure 2.2: Applying active contours to the black and white image

morphing techniques can be applied in areas like computer graphics, modelling, animation or film production.

To create a morphing algorithm it is necessary to overcome two major problems. First the algorithm must find the vertex correspondences between the source and target polygons, also known as vertex correspondence problem (VCP). The second problem consists in calculate the motion vectors for each vertex from its start position, in source polygon, to its end position, in target polygon, also known as vertex path problem (VPP).

2.2.1 Vertex correspondence problem

The first step aims at finding a good vertex correspondence between the source and target polygons, refereed as S and T respectively. The solution for this problem is not as trivial as it seems. While it is a simple task for us, humans, for a computer it is not the case. For example imagine there are two different polygons, S with n vertices and T with m vertices, then exists $n \times m$ distinct correspondences between the vertex of the two polygons.

A solution for this problem can be seen as a mapping function, where for each of S vertices exists one and only one correspondence in the vertices of T. In many cases the number of vertices in S and T are different, hence it is necessary to add new vertices to S or T so that both possess the same number of vertices, then the mapping can be performed. Even in the cases where S and T have the same number of vertices, finding a good solution is not easy. In a good solution interception of vertices should not exist during its animation, meaning that the path travelled by a vertex from its start position to its end position should not cross the path travelled by other vertices. To avoid such scenarios this problem should be taken into consideration not only in the vertex path problem, but also in the vertex correspondence problem. If this is done then scenarios like the one represented in Figure 2.3 can be avoided. In this scenario it is impossible to generate a good morph sequence using the represented vertex correspondences, since the section of the shapes composed by the vertices that intercept each other can not be morphed without creating vertex interceptions.

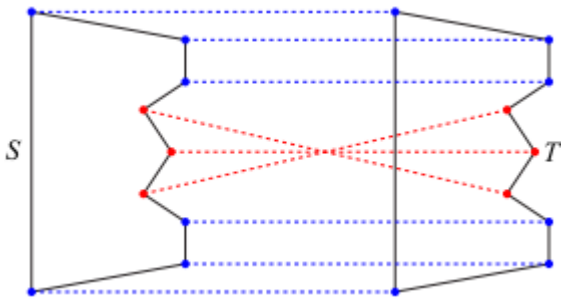


Figure 2.3: Vertex path interception example

Simple methods like corresponding the S vertex to the T closest vertex can create interceptions, Figure 2.4, that can not be solved obtaining a good solution to the vertex path problem.

This step requires manual work, since it normally requires some kind of user input, and it is crucial to avoid vertex interception. To obtain the vertex correspondences usually the algorithm calculates descriptors for each vertex and then creates a graph with the correspondences between all vertex of the two shapes, the value of each graph node is the value of the

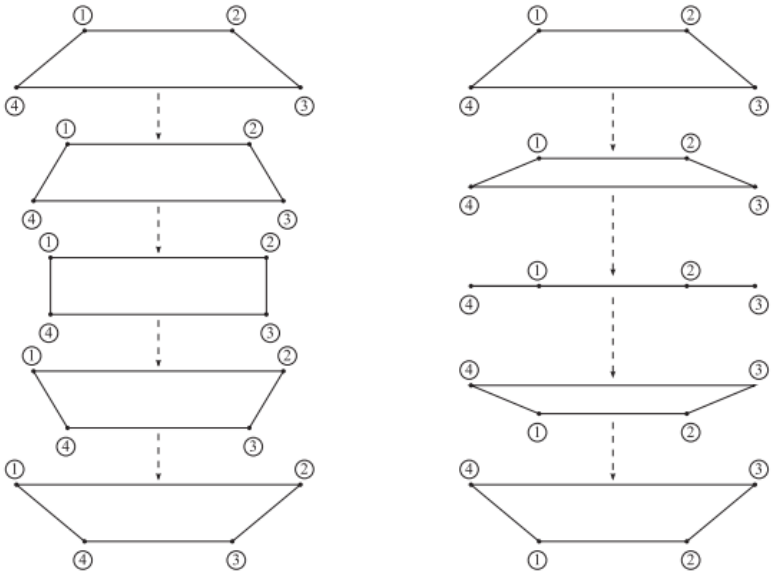


Figure 2.4: Vertex interception example 2

calculated correspondence between two vertex. A solution to the vertex correspondence is found by calculating the minimum cost path of the graph. In the various developed algorithms the most relevant appear to be the ones proposed by Sederberg et al. [2] and Liu et al. [3].

Thomas W. Sederberg et al. [2] propose a physically based approach where a shape is considered a piece of wire that can be deformed, stretched and shrunk until it turns to the other shape. The user must define the physical properties of that wire, the cost to change the wire is calculated based in these properties. During the morphing process this is the cost of the vertex correspondence. To prevent vertex interception a penalty value is used always a wire interception is detected. This approach was suggested in 1992 and some improvements have been proposed by Yuefeng Zhang [4].

Ligang Liu et al.[3] propose an approach based in human perception where the vertex correspondence is calculated using similarity measures between some key vertices, called feature points. These similarities are calculated based in geometric properties of the feature point and its adjacent vertices. Using this method, regions that are similar in the original and target shape are mapped together to create an appealing result to the user as explained in detail in Section 2.4.

On the context of this thesis the adopted method to solve this problem was the proposed by Ligang Liu et al [3], this approach takes into consideration the local information of the feature points (see Section 2.3), according to some similarity criteria to create the vertex correspondences. This way the algorithm can preserve the most relevant features of the shapes and it is efficient, since the feature points are normally a small portion of the vertices. This method is also resilient to geometric transformations such as translation, rescaling, and rotation since it is based in topological information of each vertex.

2.2.2 Vertex path problem

The vertex path is crucial to create an intuitive morphing sequence. Using simple methods like vertex interpolation can be very easy but, in many cases, can create undesired deformations during the morphing process, this happens because this method does not preserve the polygon shape. In many cases using only linear vertex paths create self interceptions, see Figure 2.5. A good solution to the VPP should avoid vertex interceptions, reduce the deformation during the process and create a natural morphing sequence from the source to target polygon.

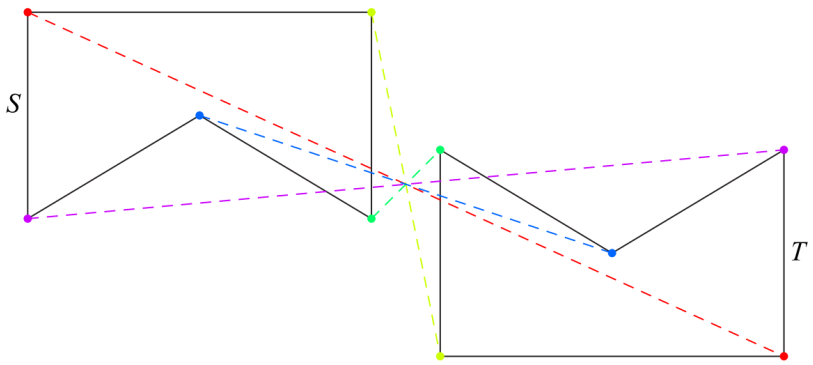


Figure 2.5: Vertex interceptions caused by linear animation paths

The methods proposed to solve the vertex path problem normally require no manual work, unlike the previous ones. The more interesting methods appear to be the ones proposed by Craig Gotsman and Vitaly Surazhsky [5], Michal Shapira and Ari Rappoport [6] and Thomas W. Sederberg et al [7]. All these methods require that a solution to the vertex correspondence problem is provided, before the solution for the vertex path problem is calculated.

The method proposed by Thomas W. Sederberg et al [7] is an improvement to the linear vertex interpolation. In this method the polygons are not considered a set of vertices but a set of edge lengths and angles. These are the values that are interpolated during the morph process. By interpolating these values it generates a smoother morphing sequence than a normal vertex interpolation. Some adjustments are also applied to the angle–edge interpolation to further improve the results. This method is an extension to the previously proposed by the same author [2].

In the method proposed by Craig Gotsman and Vitaly Surazhsky [5] the vertex path solution is found by calculating a morph of two compatible planar triangulations. The triangulations must have a set of corresponding points, the vertex correspondences. In this method, instead of interpolating the vertex coordinates, the vertex barycentric coordinates are interpolated. The barycentric coordinates represent the relative position of a point inside a triangle (2D) or a tetrahedron (3D). First a compatible internal triangulation of the two polygons is calculated, then a morph between the two triangulations is generated. By using this internal triangulations it is guaranteed that no self interception is generated. This method is restricted to simple polygons or polygons with a single hole, otherwise the compatibility of the two triangulations is not guaranteed. In Figure 2.6 an example of morphing of planar triangulations between two polygons is shown.

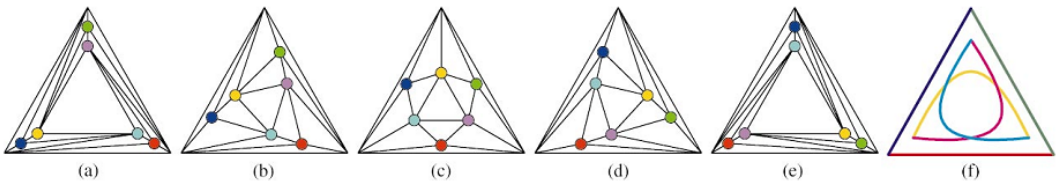


Figure 2.6: Polygon triangulations example, source [5]

In [6], Michal Shapira and Ari Rappoport proposed a similar method, but this method, instead of calculating the morph of the internal triangulations, it calculates the morph of what they call star–skeletons. A star–skeleton is composed of two parts: a decomposition in star–shaped pieces and a skeleton that connects all the star–shaped pieces. Each star–shaped piece have a star origin. The star origin is a point inside the polygon where all the boundary points are visible. The skeleton is a planar graph that connects all star origins, see Figure 2.7. The morph is generated by blending the skeletons and then the vertex associated to each star–shaped piece.

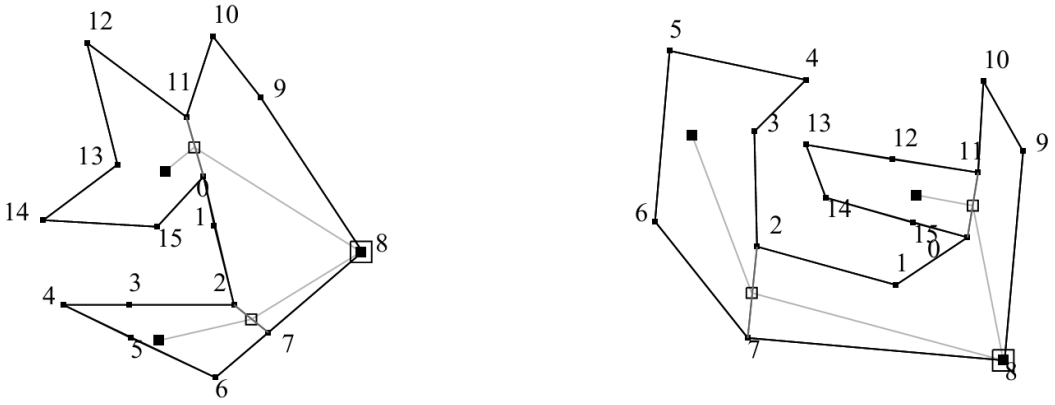


Figure 2.7: Star–skeleton decomposition example, source [6]

2.3 Feature Points

To implement the algorithm proposed by Ligang Liu et al [3], first it is necessary to detect the feature points of the geometric shapes to use. There are several methods to detect feature points in shapes, the one used was proposed by Dmitry Chetverikov et al [8]. The reasons to use this algorithm were its efficiency and simplicity.

For geometric shapes a feature point represent a point in a 2D plan. These points have other properties besides its coordinates. Using only the feature points and its properties it is possible to represent the polygon or shape they belong, like the vertices of a polygon or the points of a shape. The feature points can represent the shapes where they belong, normally using a reduced number of points, however in some cases some details can be discarded. Each feature point is normally associated to an area of high curvature in the shape or polygon and its properties are calculated based on that curve.

The detection of feature points in geometric shapes is done in two steps. In the first step are calculated which of the vertex are candidates to became feature points, the second step determinates which of the candidates are feature points.

2.3.1 Candidate detection

To verify that a point in a geometric shape is a feature point candidate, that point must match the following set of rules,

$$d_{min} \leqslant ||P_i - P_i^+|| \leqslant d_{max}$$

$$d_{min} \leqslant ||P_i - P_i^-|| \leqslant d_{max}$$

$$\alpha \leqslant \alpha_{max},$$

where d_{min}, d_{max} and α_{max} are parameters defined by the user and represent the minimum distance between the points, the maximum distance between the points and the maximum angle between the points P_i, P_i^+ and P_i^- respectively. The distance between points is the euclidean distance, $\sqrt{(P1_x - P2_x)^2 + (P1_y - P2_y)^2}$, represented by $||P1 - P2||$. P_i, P_i^+ and P_i^- represent respectively the reference point, the next point and the previous point. The value of the angle α is calculated using the following formula:

$$\alpha = \arccos \frac{a^2 + b^2 - c^2}{2ab},$$

where $a = ||P_i - P_i^+||$, $b = ||P_i - P_i^-||$ and $c = ||P_i^- - P_i^+||$, see Figure 2.8.

If a point P_i fulfil these rules then that point is eligible to became a feature point and is considered a candidate. The value of the angle α is calculated for all P_i^- and P_i^+ pairs in range $[d_{min}, d_{max}]$ of P_i and the lowest angle is considered the *sharpness* of the feature point. Another propriety that can be extracted is *convexity* of the point: if the condition $b_x c_y - b_y c_x \geqslant 0$ is verified then the point is convex, if not then it is concave.

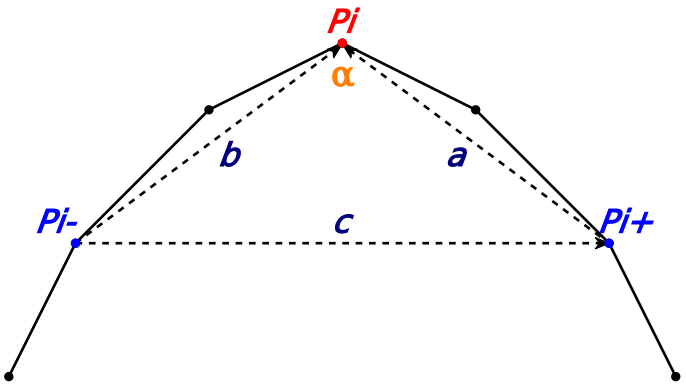


Figure 2.8: Calculating α angle

2.3.2 Candidate selection

After the candidates have been detected the next step is the selection of which of these candidates are feature points of the geometric shape and which will be discarded. The first step detects which points fulfil the requirements to became a feature point, however in one curve of the shape can exist multiple candidates. This step consists in selecting the candidate that best defines this curve. First a neighbourhood for each of the candidates will be defined using the value of d_{max}. Then if in a neighbourhood of a candidate, P_i, exists another candidate, P_j, that have a *sharpness* value greater than P_i, P_i will no longer be considered a candidate, in the end is left the candidate that best defines the curve. That candidate will be considered the feature point of the curve.

2.3.3 Feature Points in polygons

The methods of feature point detection have been developed for geometric shapes with zones of high curvature, but that principles can also be applied to polygons. In polygons the difficulty of finding feature points is reduced, since each vertex can be considered a feature point. However it is preferred that the feature points are detected using the previous method, to ensure that vertices in almost linear edges can be ignored. Ignoring these vertices carry few impact in the general form of the polygon and in most cases these changes are not even noticed by the users. Another detail to have into account are the values of d_{min} and d_{max}, used for candidate detection. To make the algorithm more robust these values should be calculated using the average edge length, so that the algorithm will not be affected by the size of the polygon neither by operations such as scaling.

2.4 Vertex correspondence

The vertex correspondence problem discussed in Section 2.2.1 can be solved using feature points and its properties as similarity measures to calculate the correspondences between the vertices of S and T. This method is similar to the way humans solve this problem. By using sections of the shapes that are associated to a feature point, called *regions of support*, and calculating some descriptors of the feature point and the associated region (see Section 2.4.2) it is possible to calculate a similarity value between two regions and create the vertex correspondences.

2.4.1 Regions of support

A region of support (ROS) consists in a set of points that are associated to a feature point. Usually these points belong to the curve described by the feature point. The region of support of a feature point Fp_i contains the points in the interval $[Fp_{i-1}, Fp_{i+1}]$, where Fp_{i-1} and Fp_{i+1} are the previous and next feature points to Fp_i. The points $[Fp_{i-1}, Fp_i]$ are called the *region of left*(ROL) and the points $[Fp_i, Fp_{i+1}]$ are called *region of right*(ROR). Both ROL and ROR have a maximum number of points, this number can be defined as $\frac{totalPoints}{nFeaturePoints}$, where $totalPoints$ is the number of points in the shape and $nFeaturePoints$ is the number of feature points in the shape.

Each region of support has some descriptors that define its topological properties. To calculate these descriptors some algebraic principles will be used. First the center of the

ROS, P_c, is calculated using the formula

$$P_c = \frac{1}{nPoints + 1} \sum_{i=0}^{nPoints} P_i,$$

where $nPoints$ is the number of points contained by the ROS, then is calculated the covariance matrix of the ROS,

$$COV = \frac{1}{nPoints + 1} \sum_{i=0}^{nPoints} (P_i - P_c)^T (P_i - P_c)$$

Using the covariance matrix's eigenvectors and eigenvalues it is possible to calculate an approximation of the normal and tangent vectors of the curve made by the ROS points. To define which of the eigenvectors is the tangent and which is the normal vector, the bisector of the angle of the feature point is calculated. Then is calculated the dot product of the eigenvectors and the bisector, the eigenvector with the smallest dot product is considered the tangent vector and the one with the larger value is considered the normal vector.

2.4.2 Geometric measures

The descriptors calculated in Section 2.4.1 and the feature point describe the portion of the shape represented by the ROS using numeric values. These values are used to compare the similarity of the curve to other curves of different geometric shapes or polygons. There are three geometric measures to distinguish the ROS from each other. These measures are:

- Feature variation

The feature variation, $\sigma(P_i)$, of the ROS of a feature point P_i is defined as

$$\sigma(P_i) = \xi \frac{\lambda_N}{\lambda_N + \lambda_T},$$

where $\xi = -1$ if P_i is convex or $\xi = 1$ otherwise, λ_N and λ_T are the normal and tangent eigenvalues calculated in Section 2.4.1. This variation measures the deviation of neighbour points of ROS from the tangent direction at P_i, meaning that if the ROS points distribution resemble a line segment then its feature variation is close to 0 otherwise it tends to $|-1, 1|$.

- Feature side variation

Feature side variation, $\tau(P_i)$, is

$$\tau(P_i) = \frac{\sigma(ROR) + \sigma(ROL)}{2},$$

where $\sigma(ROR)$ and $\sigma(ROL)$ are the feature variation of the ROR and ROL respectively, which are calculated using the previous formula but without ξ factor, having $\sigma(ROR) = \frac{\lambda_N^R}{\lambda_N^R + \lambda_T^R}$ and $\sigma(ROL) = \frac{\lambda_N^L}{\lambda_N^L + \lambda_T^L}$. Like feature variation it measures the linearity of the region. However unlike feature variation, the value of each subregion influences the result. If the two regions differ then this value differ from the value of feature side variation. If the two regions are similar then the value approaches the value of feature side.

- Feature size

Feature size measures the size of ROS relatively to the size of the original polygon. It is the a percentage of the perimeter that ROS represents, and is calculated as

$$\rho(P_i) = \frac{\rho^R(P_i) + \rho^L(P_i)}{2},$$

where $\rho^R(P_i)$ $\rho^L(P_i)$ are the percentage of the perimeter that ROR and ROL represent respectively. If this measure have a high value then it represents a large part of the polygon and have a great importance, small values means that the ROS is a very small fraction of the polygon and is less importance.

2.4.3 Similarity costs

Using the similarity measures feature variation, feature side variation and feature size, shown in Section 2.4.2, its is possible to create a function that calculates the similarity between two feature points ROS. That similarity is called the similarity cost and is used to calculate the vertex correspondence. The idea behind this process is to match portions of source polygon S with similar portions of target polygon T as much as possible. The formula to calculate the similarity cost of a pair of feature points S_i, T_j, where S_i is a feature point of S and T_j is a feature point of T, is given by

$$SimCost(S_i, T_j) = \sigma(S_i, T_j) \sum_{q=\sigma,\tau,\rho} \omega_q \Delta_q(S_i, T_j),$$

where $\sigma(S_i, T_j)$ is a weight value that defines the importance of the correspondence given by its visual relevance for the user, and its given by $\sigma(S_i, T_j) = max(\rho(S_i), \rho(T_j))$. The relevance factor is percentage of the size of the polygon that is represented by the region, the bigger the size the greater the relevance for the human user. ω is a weighting function defined by the user to variate the weight of each geometric measure in the cost. These weights must be defined as $\omega_q \geq 0$ and $\sum_{q=\sigma,\tau,\rho} \omega_q = 1$. Finally Δ_q is the function that compares the three different geometric measures of the two feature points and gives the similarity value. That value is calculated for each geometric measure using the following method:

$$\Delta_\sigma(S_i, T_j) = |\sigma(S_i) - \sigma(T_j)|,$$

$$\Delta_\tau(S_i, T_j) = \frac{|\sigma(ROL(S_i)) - \sigma(ROL(T_j))| + |\sigma(ROR(S_i)) - \sigma(ROR(T_j))|}{2},$$

$$\Delta_\rho(S_i, T_j) = \frac{|\rho^L(S_i) - \rho^L(T_j)| + |\rho^R(S_i) - \rho^R(T_j)|}{2},$$

where σ, τ, ρ are as defined in Section 2.4.2. $ROL(S_i)$, $ROR(S_i)$, $ROL(T_j)$ and $ROR(T_j)$ refer to the region of left(ROL) and region of right(ROR) of feature points S_i and T_j.

The values of similarity belong to [0, 2]. The closer to 0 the more similar the feature points are and the closer to 2 the more dissimilar the points and ROS are. Using these similarity costs it is possible to match the feature points of S and T, but in most cases the number of feature points of the two polygons is not the same and so, it makes it necessary to discard some feature points when creating the correspondences. It makes more sense to discard the points that are less relevant to the human eye. For that purpose, it is used a function that calculates the discard cost for a feature point.

2.4.4 Discard costs

To estimate the cost of discarding a feature point for the morphing process it is used a function that uses the geometric measures calculated in Section 2.4.2. These measures allow estimating the impact of the feature point and his ROS for the user. This value can be calculated as

$$DisCost(S_i) = \Phi(P_i) \sum_{q=\sigma,\tau,\rho} \omega_q |q(P_i)|,$$

where σ, τ, ρ are the same as used to calculate the similarity cost as well as the values of ω. The value $\Phi(P_i)$ represents the relevance of the ROS associated to the feature point to the whole polygon and is equal to $\rho(P_i)$. Using Φ as a coefficient it increases the discard cost as the size of the region grows relatively to the total polygon size.

2.4.5 Calculating the correspondences

The feature point correspondences between two polygons or shapes S and T can be represented as a mapping function. Where $J(r) : S_i \rightarrow T_j$, S_i is the feature point i of S and T_j is the feature point j of T. To reduce the deformation during the morphing process the mappings should be restricted to adjacent feature points of S and T, creating a path of consecutive feature points $J(r|r+1) : S(i|i+1) \rightarrow T(j|j+1)$. However if S have n feature points and T have m feature points and $n \neq m$, some feature points must be skipped. A new mapping sequence can be considered for these cases: $J(r|r+1) : S(i|i+k_1) \rightarrow T(j|j+k_2)$, where $k_1, k_2 < k$ and k limits the number of skips. The value for a mapping $J(r)$ should consider the values of the feature points S_i and T_j similarity, see Section 2.4.3, and the value of the discarded feature points since $J(r-1)$, see Section 2.4.4. The value of the mapping is given by

$$\delta(S(i|i+k_1), T(j|j+k_2) = \sum_{l=i}^{i+k_1-1} DisCost(S(l)) +$$

$$\sum_{l=j}^{j+k_2-1} DisCost(T(l)) + SimCost(S(i+k_1), T(j+k_2))$$

This way the mappings for all feature points can be calculated. These set of mappings can be seen as a path, Γ, from S feature points to T feature points. The solution for the vertex correspondence problem is the path with the lowest value. The value of a path is calculated as the sum of all mappings,

$$Cost(S, T, \Gamma) = \sum_{r=1}^{R} \delta((S(i_{r-1})|i_r), (T(j_{r-1}|j_r))$$

The solution is $Cost(S, T) = min(Cost(S, T, \Gamma))$, and Γ contains all the mappings $J(r)$.

Chapter 3

Morphing implementation

The morphing algorithm presented in [3] was implemented in Java language, given its good integration with the ORACLE database that will be needed further for the algorithm implementation, its cross platform and high level object oriented capabilities. The Java swing framework was also used for visual proposes and application interfaces.

The first step towards the solution was the creation of the new data structures needed for the algorithm, the shapes, points and feature points. This data structures are an extension, of the already existing geometric objects from the *java.awt* package. This extension allows an easy graphic representation and access to the implemented methods. The class MyPolygon2D represents a polygon composed by 2 dimension points and a list with the detected feature points. A feature point is represented by the class FeaturePoint2D, this class stores the feature point properties and descriptors explained in Section 2.3.

The implementation of the morphing solution can be found in the package *morphing* of the source code, this package is divided in two packages *morphing.vcp* and *morphing.vpp*.

The package *morphing.vcp* contains the solution described in Section 2.4 for the vertex correspondence problem discussed in Section 2.2.1. In this package the code for detection of feature points and calculation of vertex correspondences can be found, respectively in the classes, *FeaturePointDetector* and *PolygonCorrespondences*, some details of these classes will be explained in the next subsections.

3.1 FeaturePointDetector class

As defined in Section 2.3 there are two steps to find the feature points of a polygon: first the detection of the candidates and then a selection from that candidates. For both these steps a set of values to detect the feature points are given, the values are, d_{min}, d_{max} and α_{max}, defined in Section 2.3.1. These values are used by the constructor *FeaturePoint-Detector(double max_angle, double min_angle, double d_min, double d_max)* to create a new *FeaturePointDetector*, where the parameters *d_min*, *d_max* and *max_angle* correspond to the values of d_{min}, d_{max} and α_{max}, the parameter *d_min* is used to guarantee a minimum distance between the points when choosing a feature point. By setting this minimum distance in the candidate detection step, the detection of feature points that have a very small relevance can be avoided. Also if there are points too close the angle will be very sharp and it can lead to the detection of some false candidates, see Figure 3.1. The *d_min* parameter ensures that these points are not considered when detecting candidates. Another relevant aspect is how

distance of two points is calculated. This distance is the sum of the length of the edges from the source point to the target, instead of the linear distance. This way cases where all points are within the *d_max* distance from each other, if this value is to great, this avoids a loop of verifications or invalid distances as can be seen in Figure 3.2.

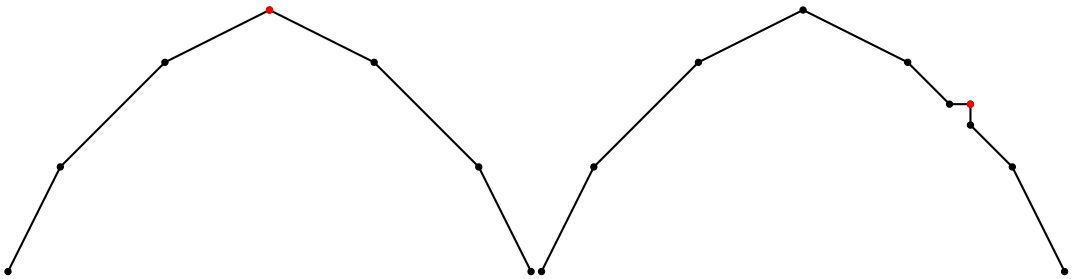


Figure 3.1: Feature point detection with(left) and without(right) d_min parameter usage

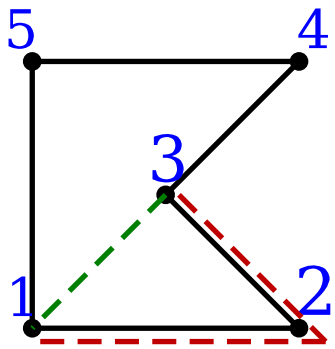


Figure 3.2: edge distance(red) against linear distance(green)

The two steps to find the feature points of a polygon are implemented in the method *getFeaturePoints(MyPolygon2D polygon)*. First the list of candidates is created, this list stores all the detected candidates. All the vertices of the polygon will be tested and the ones that match all the conditions defined in Section 2.3.1 will be added. Once all the candidates are found the candidate selection is done. In this step the point that best define a curve of the polygon is selected using the parameter *d_max* to define the range of the validation.

3.2 RegionOfSupport class

The class RegionOfSupport represents a Region of support as defined in Section 2.4.1. A new RegionOfSupport is created using a list of points containing all the points of that region and its feature point as parameters to the constructor *RegionOfSupport(List<Point2D> points, FeaturePoint2D fp)*. This class will calculate the center, the covariance matrix and the normal and tangent eigenvectors/eigenvalues, as defined in Section 2.4.1. The list of points that compose a RegionOfSupport are given in the method *getRegionOfSupportPoints(FeaturePoint2D fp)* provided by the class MyPolygon2D. This method gets the points in the interval $[Fp_{i-1}, Fp_{i+1}]$ with the maximum $\frac{totalPoints}{nFeaturePoints}$ points, as defined in Section 2.4.1. This allows restricting the size of the ROS and avoiding that many points of other curves appear in the wrong regions, see Figure 3.3.

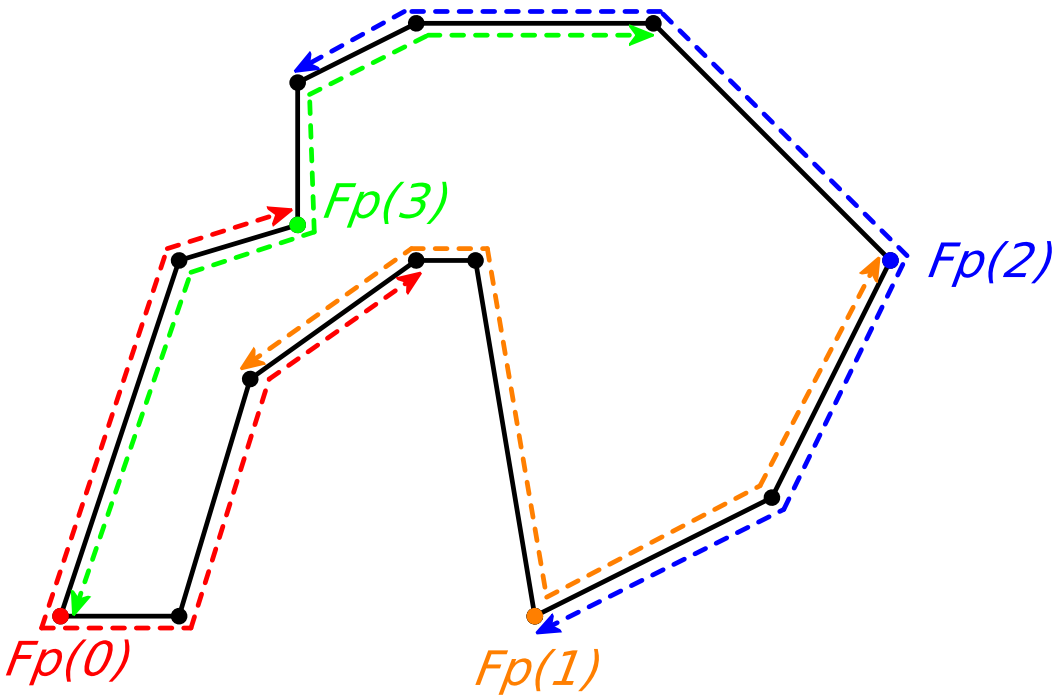


Figure 3.3: Example of feature points and its ROS in a polygon

To obtain the normal and tangent eigenvectors/eigenvalues, first the bisector of the angle must be calculated. The bisector is calculated using the three points that form the least opening angle of the feature point in that region, these points were stored in the FeaturePoint class during the feature point detection process. This way calculating the angle bisector is very simple, once the bisector is calculated the dot product between the, already calculated, eigenvectors and the bisector are calculated and the eigenvector with the smallest dot product is considered the tangent vector. Once the normal and tangent eigenvectors/eigenvalues are defined the geometric measures defined in Section 2.4.2 are calculated and stored in the corresponding feature point.

3.3 PolygonCorrespondences class

This class address the problem described in Section 2.4. To create a new instance of a PolygonCorrespondences the constructor *PolygonCorrespondences(MyPolygon2D S, MyPolygon2D T, double wFvariation, double wFside, double wFsize, double weightSim)* can used. The parameters *wFvariation*, *wFside* and *wFsize* correspond to Feature variation, Feature side variation and Feature size defined in Section 2.4.2. The value of *weightSim* defines the weight of the similarity value of two feature points in the process of finding the correspondences (see Section 2.4.3). If this value is $]0, 1[$ then the similarity have a higher importance, if the value is greater than 1 then it have a lower importance and the possibility of discarding feature points is increased. The parameters S and T used in the constructor are the source and target polygons where the correspondences will be calculated and must have the feature points already detected.

The correspondences between the feature points of S and T are calculated using the methods *getFeaturePointCorrespondences(int skips)* or *getFeaturePointCorrespondences(int s_init, int t_init, int skips)*, where the parameter *skips* defines the number of skips to be used, the k value defined in Section 2.4.5. The parameters *s_init* and *t_init* define an initial correspondence in the second method. Using an initial correspondence in the process of finding the correspondences reduces the number of combinations that will be tested. If an initial correspondence is not set the algorithm will test all the possible initial correspondences and choose the best one, the number of initial correspondences is $n \times m$ where n is the number of feature points of S and m the number of feature points in T. The initial correspondence is set by the user to try to define a good correspondence to improve the results of the correspondences.

The best correspondence between S and T is considered the minimum value of all the possible correspondences of S and T feature points. The total of possible correspondences is $k^2 \times n \times m$ if an initial correspondence is set or $(k \times n \times m)^2$ if one is not defined. Without any optimization the time required to find the solution will increase too fast to be considered usable for polygons with many feature points. To reduce this time some optimizations must be implemented. The method suggested by the literature was the dynamic programming, this method reduces the cost of calculating a path by first filling a matrix with all the possible vertex correspondences and its costs. By doing so this method can reduce the complexity of the algorithm to $k^2 \times n \times m$, however the cost to initialize the matrix is $k^2 \times n \times m$. Other algorithm was implemented to further reduce the complexity, this new algorithm (fast algorithm) does not guarantee the optimal solution, however a reasonable solution is always achieved. This algorithm was a complexity of $max(n, m) \times k^2$ and does not require an additional matrix. To reduce the execution time in any of the previous algorithms a matrix with the discard and similarity costs between feature point is created to accelerate the calculation the correspondence function.

Once the correspondences between feature points are established, the correspondence between the polygons vertices can be set. The correspondences between the vertices are calculated between sections of the two polygons, these sections are defined by the points between two feature points, see Figure 3.4. The points of the section defined by the feature points S_i and S_{i+1} correspond to the points of the section defined by T_i and T_{i+1}. Normally one section contains a different number of points than the other, to solve this problem some points of one section can correspond to more than one point in the other section, see Figure 3.5.

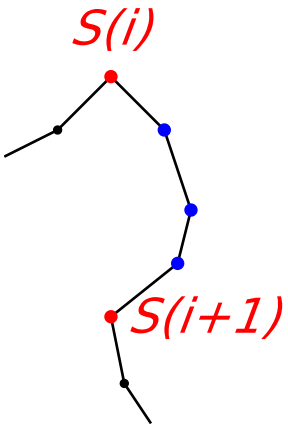


Figure 3.4: Polygon section example, the points of the section are the points between S_i and S_{i+1}

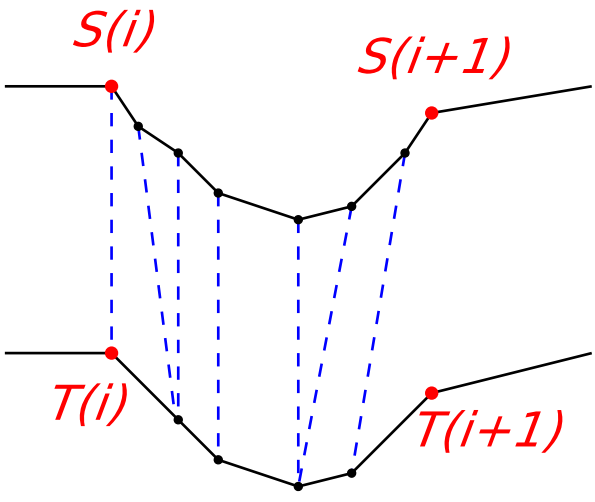


Figure 3.5: Vertex correspondence example

3.4 VPCalculator class

The solution to the vertex path problem is implemented in this class. This solution is a simple vertex interpolation, as previously explained this method is very simple and in many cases creates not intuitive morph sequences. This method was first implemented to see the results of the VCP, since the results at that time were satisfactory the method was left to improve later as there was yet many important issues to attend, and the development of a better solution would involve a large time investment while our priority was towards other issues.

The class receives a list of vertex correspondences between two polygons, these correspondences can be calculated by the class in Section 3.3. A correspondence between two vertices contains the vertex coordinates and other optional information, as the vertex correspondence cost.

Before the vertex paths are calculated two new polygons are created, S' and T', this new polygons topologies are identical to S and T respectively. S' has the same number of vertices than T' however that number of vertices can be different from S or T. That happens because in most cases the sections of S and its correspondent section of T can have a different number of vertex. In that cases a $n \rightarrow m$ correspondence is created and new vertex are added to the sections of S' and T' so that these sections have $max(n, m)$ vertex. The new vertex will start or end in the coordinates of an already existing vertex. For example in a $1 \rightarrow 2$ vertex correspondence, the new vertex will be added to S' in the same position of the vertex of S, then the two vertex, $S'(i)$ and $S'(i+1)$, will move to the coordinates of $T(i)$ and $T(i+1)$. In a $2 \rightarrow 1$ correspondence the new vertex will be added to T', in the same coordinates of $T(i)$, and the two vertex of S will move to the correspondent vertex of T'. Following the previous method the morphing sequence of S to T can then be considered a morphing sequence between S' and T', where S' and T' have the same number of vertex.

Once S' and T' are defined calculating the linear vertex paths are very simple. These paths are defined as a 2D vector, $v = v_x, v_y$, and are associated to each vertex of S'.

$$v_x = T'(i)_x - S'(i)_x$$

$$v_y = T'(i)_y - S'(i)_y$$

To create an animation of the morphing sequence a time interval should be associated to the morphing sequence, that time interval will define the amount of time that the animation will take. To calculate the velocity of a vertex the components of the movement can be calculated from the vector v. A method that calculates a representation of a moving point is implemented in getMPointUnit(int id, Timestamp start, Timestamp end), the representation of moving points and polygons will be described in the Section 4.3.

3.5 ReducePolygon class

This class provides methods to reduce the number of points needed to represent a polygon. This method can be used when importing a polygon from the segmentation application refereed in Section 2.1. The method used to reduce the number of points was the algorithm known as Douglas–Peucker algorithm proposed by Urs Ramer, in 1972, and by David Douglas and Thomas Peucker, in 1973. The algorithm was conceived to create an approximation of a curve using line segments using a threshold value, ε, see Figures 3.6 and 3.7. However its

principles can also be applied to create an approximation of a polygon. This approximation will became very similar to the initial polygon if a small threshold is used.

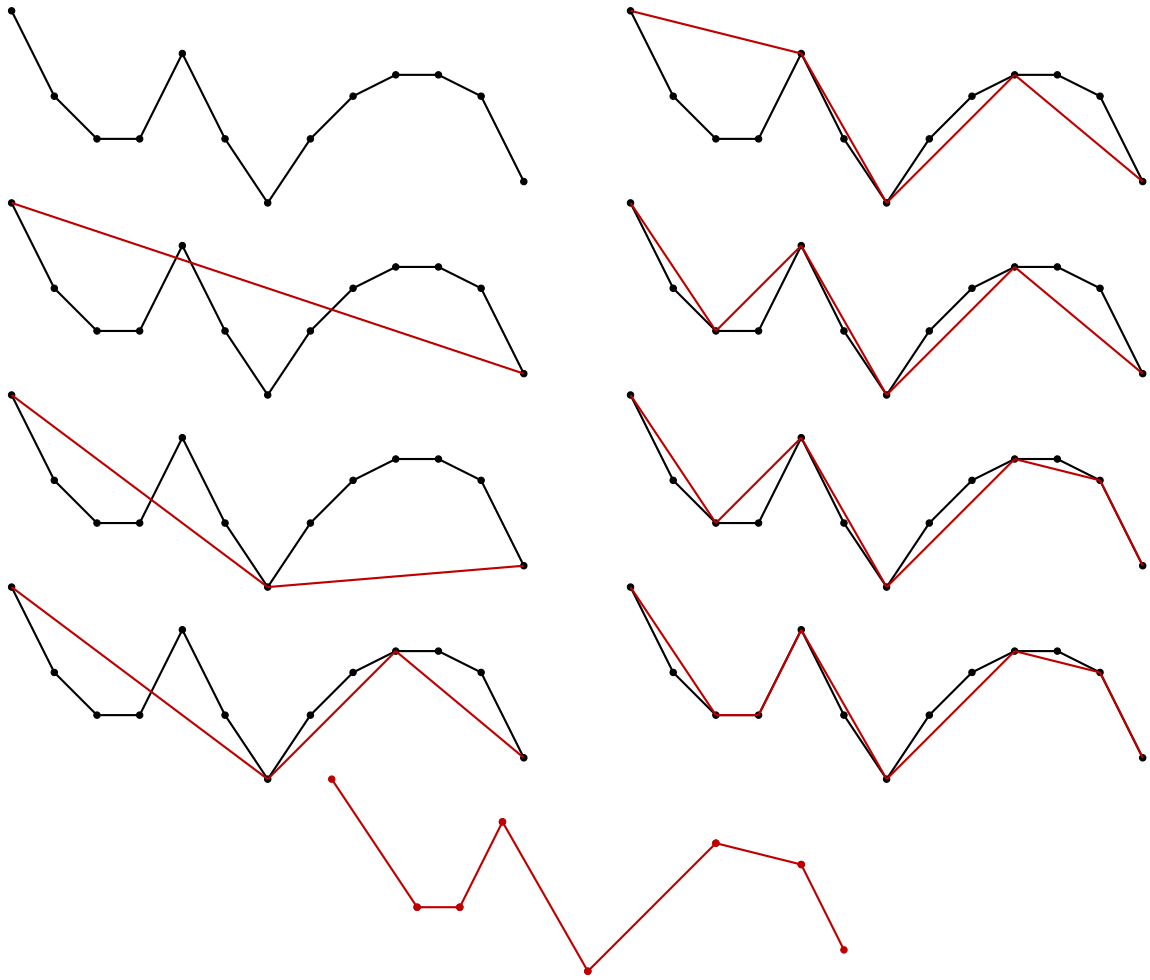


Figure 3.6: Douglas–Peucker algorithm step by step

This algorithm simplifies a polygon by recursively adding polygon vertex to a line segment, the line is created by two points of the initial polygon. Then the point of the initial polygon that have the greatest perpendicular distance to that line is added, creating a new line segment. The new line segments will became the approximation of the polygon. New points will be added until no point is at a greater perpendicular distance of a line segment than the specified value defined by the user. That value is the threshold used by the Douglas–Peucker algorithm, ε, that will define the precision of the approximated polygon, the lower the value the closer the approximation will be from the initial polygon.

The perpendicular distance of a point to a line is the length of the line segment perpendicular to the initial line that ends in the point, this length can be considered the smallest distance of the point to the line see Figure 3.8.

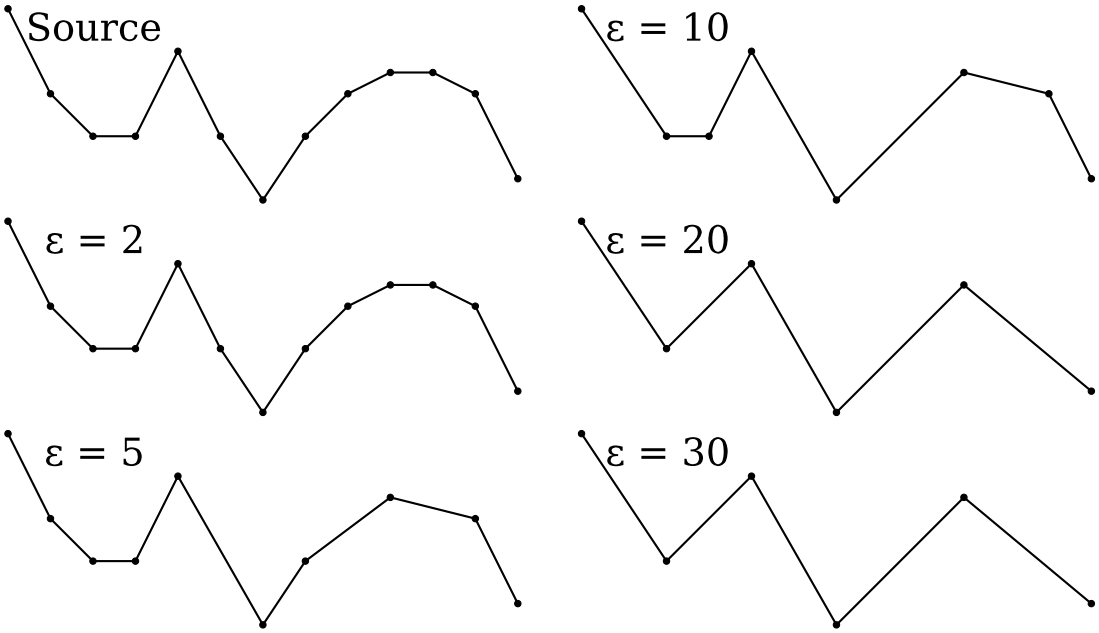


Figure 3.7: Douglas–Peucker ε value variation

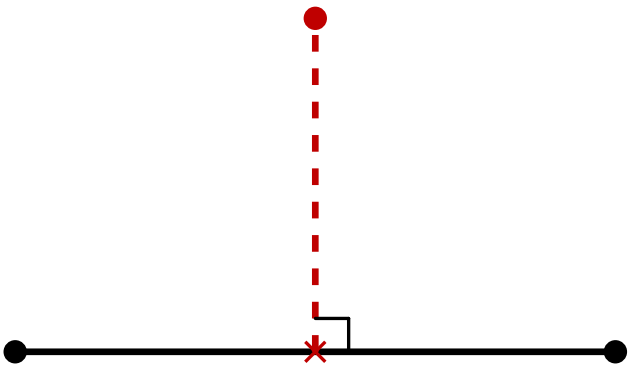


Figure 3.8: Perpendicular distance example

3.6 Results

In this section some tests done with the implemented method will be presented and the results obtained will be explained. The tests will use a synthetic data set composed by some simple geometric shapes. These tests intend to show the weight of different parameters used in the various steps during the morph and some limitations in the solution. The synthetic data set consists of twelve distinct polygons, these polygons will be numbered from 1 to 12 and that number will be reference to each polygon, the polygons are shown in Figure 3.9.

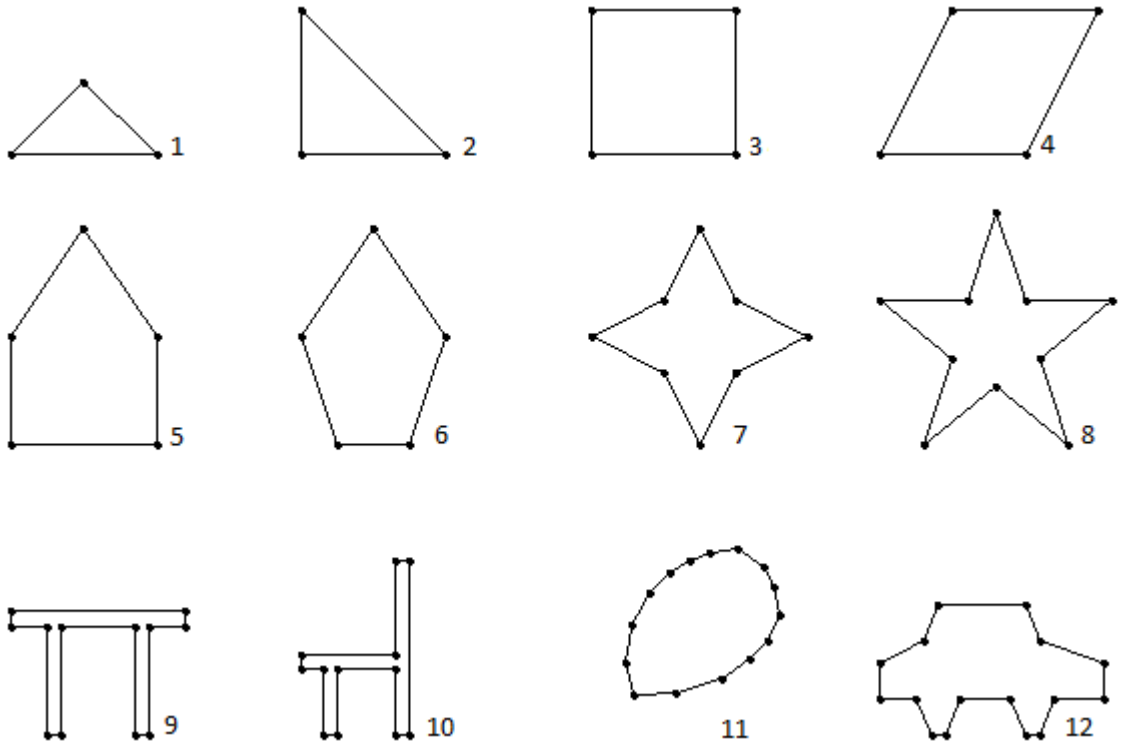


Figure 3.9: Synthetic polygons used

3.6.1 Feature point detection test

The first implemented test focus in the detection of feature points in polygons. By changing the values of *max_angle*, *min_angle*, *d_min* and *d_max* different feature points will be detected. The detection of feature points is crucial for obtaining a good vertex correspondence, for it will define the regions to be compared and matched during the vertex correspondence process. If the regions are dissimilar it is hard to create a good morphing sequence.

To reduce the user interaction required, this test will try to find the more generic parameters to use to obtain reasonably good results. The optimal parameters change for each case so to find more generic results a set of parameters that yield adequate results in most cases must be defined. To find values for *d_min* and *d_max* that are not affected by the polygon size or scale operations the average edge length is calculated and used as reference. The value to test will be a factor applied to this average length, for *d_min* this value shall be in

the interval $[0, 1[$ and for *d_max* this value should be $[1, 3[$. To test the parameters a simple application was created, in this application the user can load polygons and change the values of *max_angle*, *min_angle*, *d_min* and *d_max* to see the different detected feature points and its ROS, Figures 3.10 and 3.11 are examples of the application main frame and a Region of support frame.

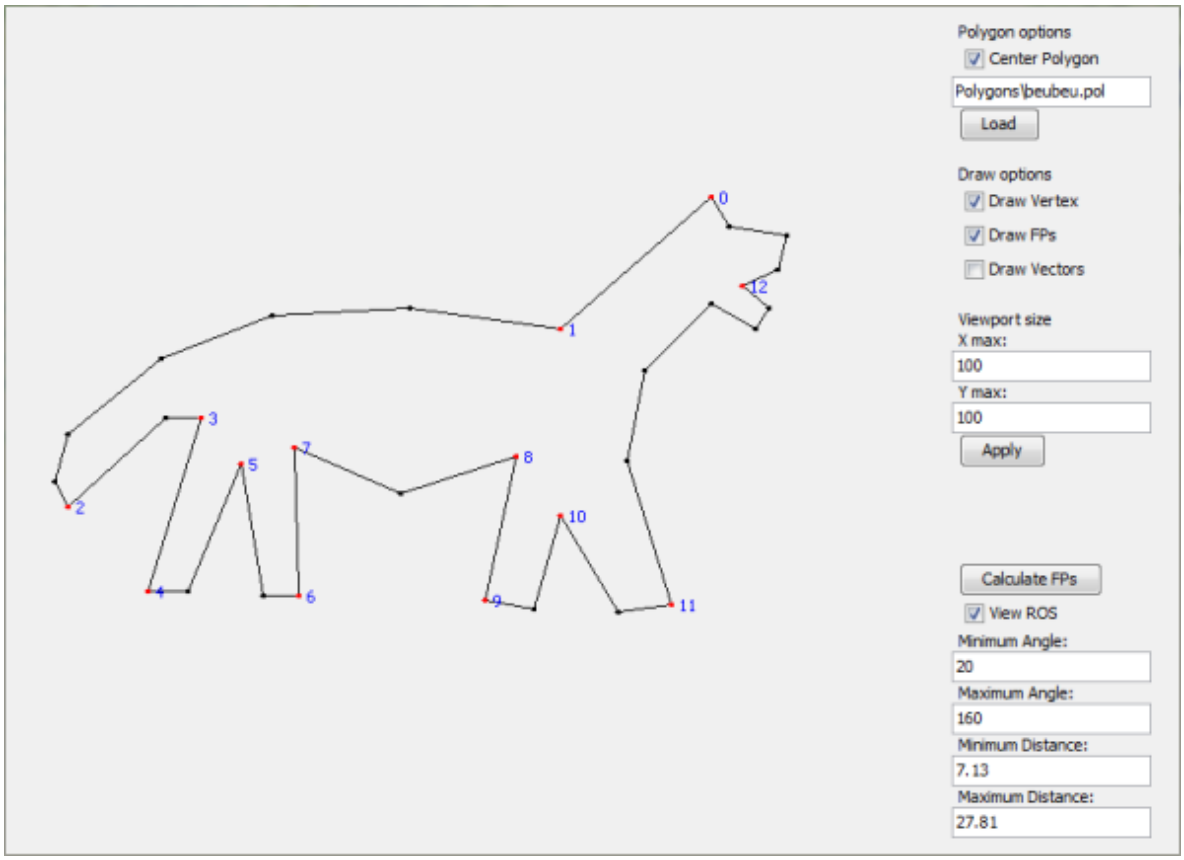


Figure 3.10: Feature point test application

When a polygon is loaded the average edge length is calculated. The *d_min* is $\frac{average\ edge\ length}{3}$ and the value for *d_max* is $3 \times average\ edge\ length$. These values where chosen because they work relatively well for many polygons, the values for *max_angle* and *min_angle* are 160 and 20 respectively, with this values the angles with few relevance are discarded. All these values can be changed to see the different results. Using the previous values to detect the feature points, the following feature points where detected in the synthetic data set. In Figure 3.12 the polygons are shown and the red dots represent a feature point. As we can see the feature points detected in each polygon can define good ROS the define relevant areas of the polygons.

3.6.2 Vertex correspondence test

To test the algorithm in the vertex correspondence problem, the obtained correspondences will be compared with a manual correspondence between some polygons from the hypothetical data set, using the feature points detected in the last test. The manual correspondence will be set using the polygon regions similarity as base. The number of matches will be calculated

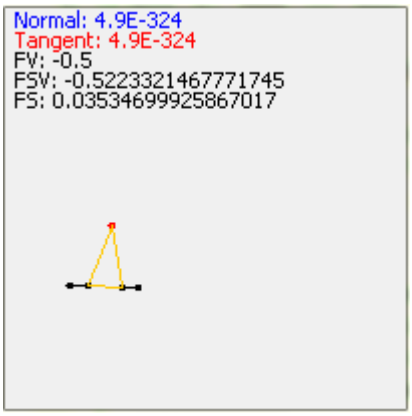


Figure 3.11: Region of Support 5, its maximum opening triangle(yellow), feature point(red), vertices and geometric measures

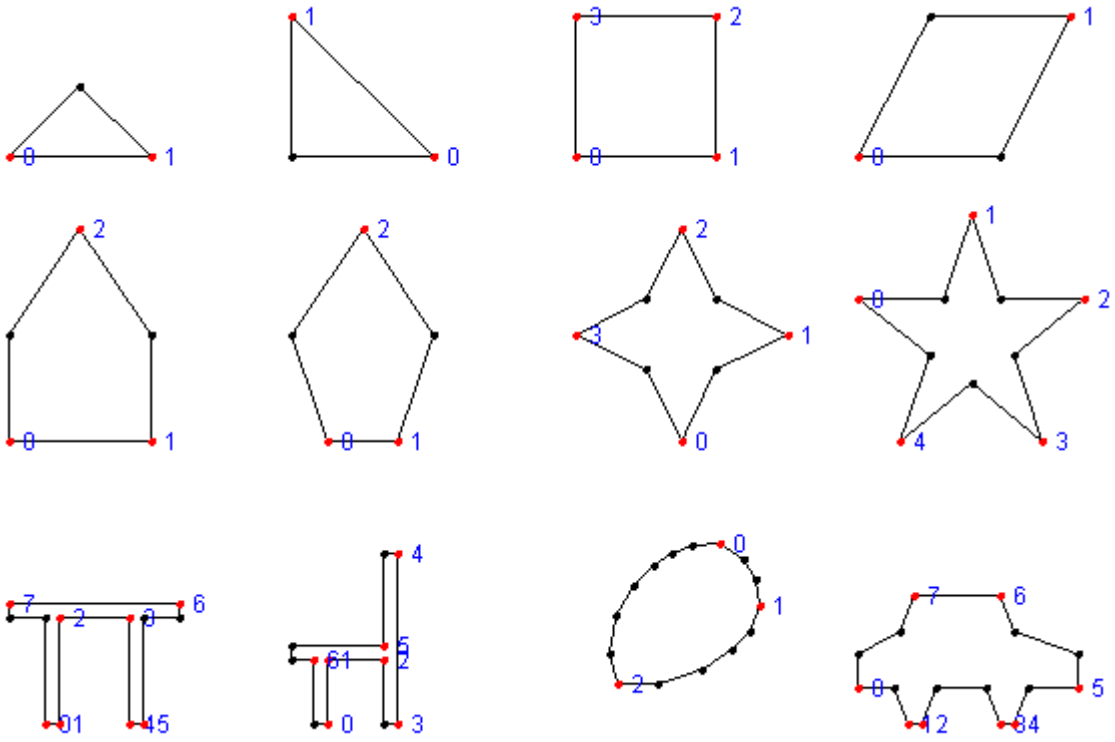


Figure 3.12: Detected feature points

and be used to evaluate the precision of the algorithm, however it should be taken into account that several solutions can be found for the correspondence problem, so other results can also be considered. In this test the results of the different correspondence algorithms will also be compared, the fast correspondence algorithms, described in Section 3.3 will be compared with minimal cost. In this test the similarity and discard costs (see Sections 2.4.3 and 2.4.4), used where the proposed in the article [3], these values where $\omega_q = \frac{1}{3}$ and the maximum skips set to 2.

The results, correspondences and comments are listed bellow:

- Correspondence between polygons 1 and 2

The manual correspondence was $\{0 \to 0;\ 1 \to 1\}$, where $0 \to 0$ denotes a correspondence between feature point 0 of polygon 1 and feature point 0 of polygon 2. The feature points are shown in Figure 3.12. In this case both algorithms matched the manual correspondence with 100% precision.

- Correspondence between polygons 3 and 4

The manual correspondence was $\{0 \to 0;\ 2 \to 1\}$. In this case the fast correspondence algorithm result was $\{0 \to 0;\ 3 \to 1\}$ and the minimum cost result was $\{0 \to 0;\ 1 \to 1\}$. These two results are different but represent a similar solution to the problem. Both solutions differ from the manual correspondence but represents a possible solution to the problem.

- Correspondence between polygons 3 and 5

The manual correspondence was $\{0 \to 0;\ 1 \to 1;\ 2 \to 2\}$. The minimum cost algorithm get the same result as the manual solution. The fast correspondence algorithm gives $\{1 \to 0;\ 2 \to 1;\ 3 \to 2\}$ as the solution, this solution may differ from the other but represent an adequate solution to the problem.

- Correspondence between polygons 3 and 7

The manual correspondence was $\{0 \to 0;\ 1 \to 1;\ 2 \to 2;\ 3 \to 3\}$. The minimum cost algorithm gets the same result as the manual solution. The fast correspondence algorithm considered $\{0 \to 2;\ 1 \to 3;\ 2 \to 0;\ 3 \to 1\}$ as the solution, this solution may differ from the other but represent the same solution only considering a different initial correspondence.

- Correspondence between polygons 5 and 6

The manual correspondence was $\{0 \to 0;\ 1 \to 1;\ 2 \to 2\}$. In this case both the algorithms give the same results as the manual solution.

- Correspondence between polygons 5 and 11

To this two polygons the manual correspondence was $\{0 \to 0;\ 1 \to 1;\ 2 \to 2\}$. In this case the fast correspondence algorithm matched the manual solution, while the minimum cost algorithm yield the solution $\{0 \to 0;\ 2 \to 1\}$.

After analysing the previous results, a very good result was obtained in almost every case. Even in cases where the result was not optimal, it was acceptable for the two different algorithms. These results set a good base to the morphing algorithm since the feature point detection is one of the most important steps.

3.6.3 Translation test

In this test the morphing algorithm will be tested in a polygon translation, a polygon will be translated and the morphing result will be evaluated. In this case the topology of the two polygons will be identical and the correspondences calculated will be a direct correspondence between the feature points of S and T. The vertex paths should be equal to the translation applied to the T polygon.

Since there are no deformation and the vertex paths are linear the implemented solution can easily solve this kind of morphing, as can be seen in Figure 3.13. In this example the polygon 3 is translated $v_x = 30$ and $v_y = 50$, that as also the vertex path of each vertex.

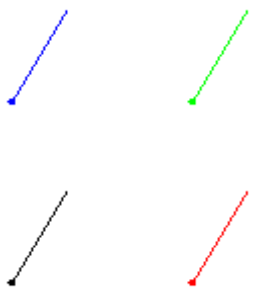


Figure 3.13: Vertex paths from polygon 3 translation, the points represent the vertex start position and the lines the vertex path, each color represent a distinct vertex

3.6.4 Rotation test

In this test a polygon will be rotated and the resulting morphing sequence is evaluated. Once again the S ans T polygons topologies are identical, only the vertex coordinates differ. The vertex paths should avoid vertex interceptions and the polygon shape should be maintained. The experiments show that even if the correspondences are correct, the vertex path solution implemented gives no guarantee that no interceptions or unnecessary deformations do not appear. To evaluate the algorithm result two criteria where considered, the number of interceptions and the area variation during the morph process.

In this test some polygons where rotated and the following results were obtained to each case:

- Polygon 3, 45 degrees rotation

  This rotation does not yield vertex interception but the polygon area reduces and then expands again. Even through the topology is maintained, the polygon suffers an unnecessary deformation, see Figures 3.14 and 3.15.

- Polygon 5, 90 degrees rotation

  Similar to the previous rotation the polygon area is reduced and then expanded instead of being constant, see Figures 3.16 and 3.17.

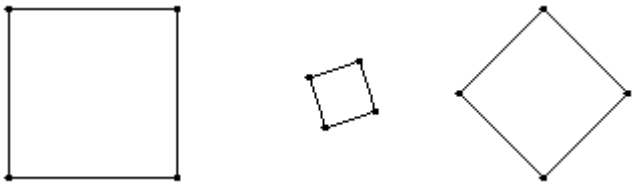


Figure 3.14: Polygon 3, 45 degrees rotation

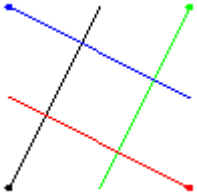


Figure 3.15: Vertex paths from polygon 3, 45 degrees rotation

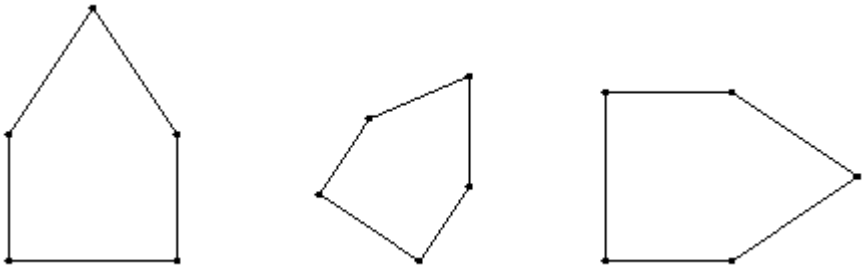


Figure 3.16: Polygon 5, 90 degrees rotation

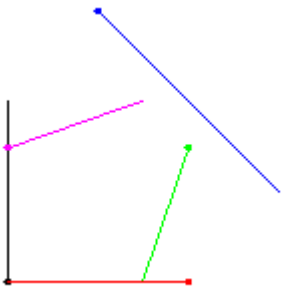


Figure 3.17: Vertex paths from polygon 5, 90 degrees rotation

- Polygon 5, 180 degrees rotation

  In this rotation the area variation is even more severe, in this case the polygon area reaches 0 and the polygon is reduced to a line, see Figures 3.18 and 3.19.

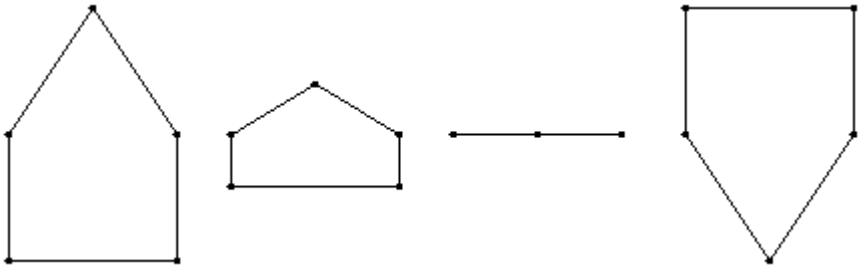


Figure 3.18: Polygon 5, 180 degrees rotation

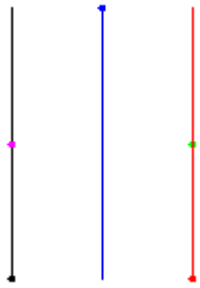


Figure 3.19: Vertex paths from polygon 5, 180 degrees rotation

3.6.5 Deformation test

For this case the morphing between two similar polygons are tested, to simplify the vertex correspondence the results from the correspondences between the pairs of polygons $1 \rightarrow 2$, $3 \rightarrow 4$ and $5 \rightarrow 6$ will be used, the correspondences were presented in Section 3.6.2. Once again the vertex interceptions and area variation will be used as measures to evaluate the morph result.

- Morph $1 \rightarrow 2$

  In this morphing no interception occurred but as this morph is similar to a rotation, the polygon area reduces and expands during the morph. The final result is acceptable but a more "rigid" polygon deformation should have been the result. The area variation can be seen in Figure 3.20.

- Morph $3 \rightarrow 4$

  In this case there are also no vertex interception and once more the area variation is not linear. The morph sequence should be more natural, but with the used vertex

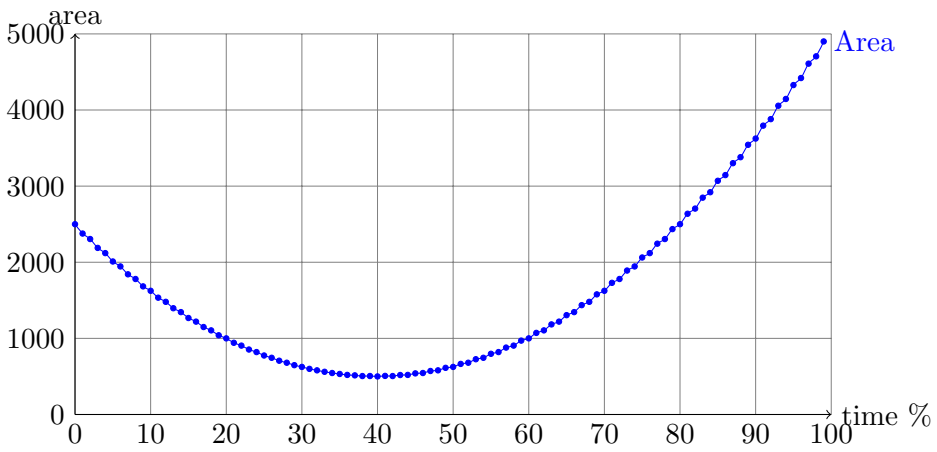


Figure 3.20: Area variation in morph 1 → 2

---

correspondence the deformation is more than the necessary. The area variation can be seen in Figure 3.21.

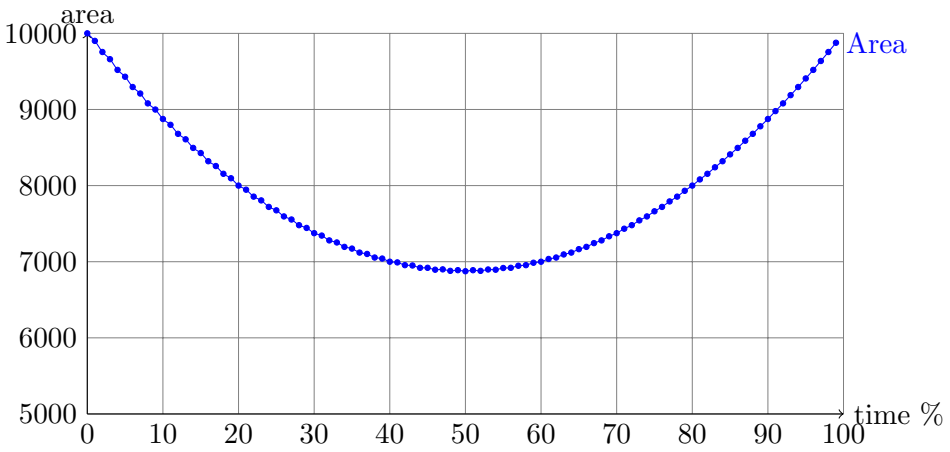


Figure 3.21: Area variation in morph 3 → 4

- Morph 5 → 6

  This morph can be considered the ideal result, there are no vertex interceptions, the area variation is linear and the resulting morph sequence seams a natural morph from polygon 5 to polygon 6. The area variation can be seen in Figure 3.22.

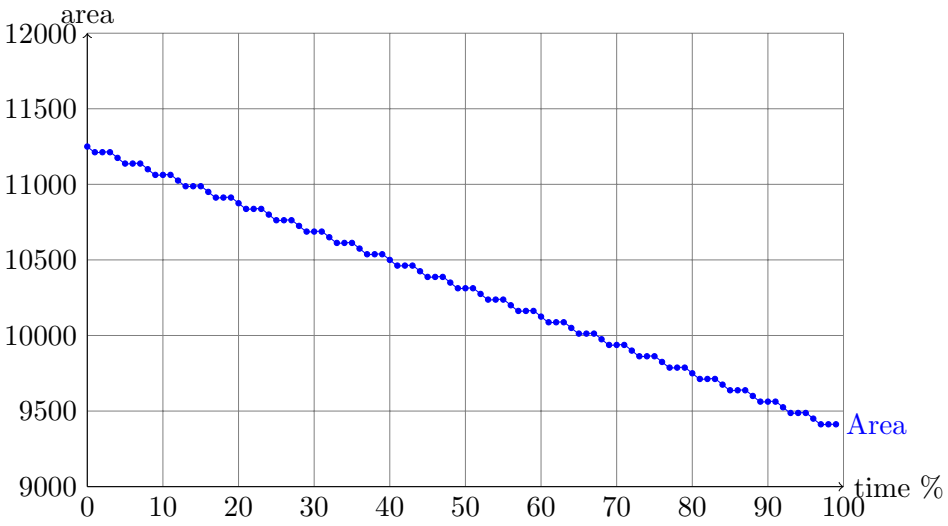


Figure 3.22: Area variation in morph 5 → 6

Chapter 4

Moving objects representation in spatio-temporal databases

4.1 Overview of moving objects databases

A spatio-temporal database is a database that is able to represent, manage, process and retrieve spatio-temporal data. Since spatio-temporal data changes both in time and space, a moving object is an abstraction commonly used in spatio-temporal databases research to denote geographical entities that can move and change shape over time. The database provides a set of operations over this objects like intersections, distances, area calculation or projections.

Although there were many research works on the development of spatio-temporal methods for dealing with moving points, that is, moving objects whose size and shape do not need to be represented in the information system, research on spatio-temporal methods for dealing with moving objects with complex shapes has received minor attention. In this context, the abstract definition of a moving object can be given by a triple(τ, ς, ν) where $\tau \subset \mathbb{R}$ is a time interval, $\varsigma \subset \mathbb{R}^2$ denotes the geometry of the moving object at a certain time instant and $\nu : \mathbb{R}^2 \times \mathbb{R} \rightarrow \mathbb{R}^2$ is a continuous function defining the transformation of the geometry ς during τ [9]. The semantics of this abstract representation is $m = (x, y, z) \in \mathbb{R}^2 \times \mathbb{R} | (\exists x')(\exists y')(x', y') \in \varsigma \bigwedge t \in \tau \bigwedge (x, y) = \nu(x', y', t)$.

This abstract representation of moving objects has been implemented on different discrete data models suitable for implementation in databases. The most important ones are the constraints databases ([10]) and abstract data types ([11] and [12]) approaches. In recent years, it was given particular attention to the ADTs approach because this data model can be smoothly built into extensible DBMS, such as object-relational DBMS.

The solutions based on abstract data types represent the objects movement as an ordered sequence of motion units [13]. The motion within a unit is given by simple functions,linear functions in most proposals, describing the movement of each vertex during a fixed time interval, and must be consistent with several restrictions for ensuring that the geometry and the topology of the region is valid for every time instant in that interval Figure 4.1.

This approach was firstly implemented on the top of Secondo, a prototype DBMS for research and teaching [14]. It is to the author knowledge the only implementation where the spatio-temporal data model and query language are completely integrated into a DBMS environment. It allows representing different types of moving objects, namely, moving points,

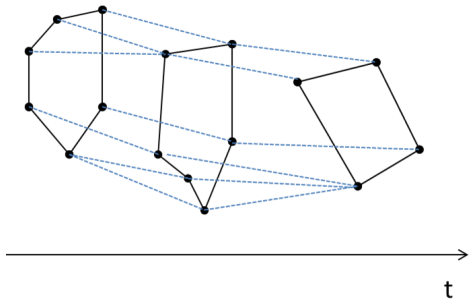


Figure 4.1: A moving object motion unit

moving lines and moving regions with complex shapes, including moving regions with holes. This prototype also implements a wide variety of spatio-temporal algorithms for the evaluation of projections, set operators, predicates and clipping operations, which are described with detail in [13].

Object-relational DBMS currently in commercial use, as the Oracle DBMS, allow developing extensions to deal with complex data types that can be attached to the database and manipulated using SQL, this data types can be build to represent moving objects. This is the case of Hermes, which is an Oracle extension for the representation and management of spatio-temporal data [15]. The spatio-temporal data types derive from the Oracle Spatial data types and the temporal data types released by TAU Temporal Literal library [16]. This is the only extension where the edges connecting the vertices defining the boundary of moving regions can be straight lines or arcs. Although, due to the complexity of the underlying data model, this extension has only a limited number of algorithms to perform spatio-temporal operations dealing with moving objects with complex shapes. For details on this work it is recommended to read the technical report [17].

The Spatio-temporal Object Cartridge(STOC) is also an extension for the representation of continuously and discretely changing spatial data [18]. However, this extension has only two data types for the representation of moving objects, moving point and moving rectangle. This means that it is only possible to represent approximations, minimum bounding boxes, of moving objects with complex shapes.

The extension proposed in [19] also introduces spatio-temporal data types and operations for dealing with moving objects in Oracle 11g. The data model is inspired in [11] but this paper also introduces a technique for the evaluation of spatio-temporal operations based on a discretion of the temporal dimension using a temporal quantum. This means that a spatio-temporal operation is decomposed into a sequence of spatial operations over snapshots(projections) of the moving objects. The technique is simple to implement and may be adapted for several types of movement functions, linear, quadratic, etc. However, the implementation of spatio-temporal clipping operations requires an additional post-processing step to assemble the results of the spatial operations over each snapshot back into a moving data type. The paper presents a simple interpolation method to accomplish this task but the authors argue that the results are not satisfactory and need improvement.

The framework used in this thesis was the one developed by Luís Matos et all. [19]. This framework is implemented as an extension for the ORACLE 11g and provides support for moving points and moving regions. These data types and the operations over these are resumed in Sections 4.3 and 4.4.

4.2 Architectures for spatiotemporal databases

According to M. Breunig et all [20], there are three main architecture that can be considered when implementing a database extension, these three architectures are monolithic, layered and extensible. In a monolithic based solution all the features are plugged into the DBMS kernel. This improves performance as all operations are supported by the database core, however the implementation is very demanding and each modification or expansion must be implemented in the DBMS kernel code. In a layered architecture all the changes are implemented in a layer on top of the DBMS core. This layer uses a predefined interface to communicate with the DBMS. This architecture has the advantage of reducing the implementation time, as the implementation is outside of the core then its performance may decrease. The extensible architecture combines the best of the two previous models, since the support for new features in DBMS is done outside its core. The architecture implemented in the ORACLE DBMS can be considered extensible and so it makes it simpler to develop extensions for the DBMS. Considering that the ORACLE DBMS already have a spatial support, it is a good candidate for the development of a spatio-temporal expansion.

4.3 Data model

The spatio-temporal expansion for the ORACLE database, used in this work provides data types and operations for the representation and querying of moving objects. The base of this implementation is the moving point, **MPoint**, that consists of a collection of moving point units, **Movement**, and an **id** associated to that **MPoint**. The **Movement** contains a dynamic number of **MPointUnit** where each unit represents the movement of the point during a given time interval. This data type consists of a structure that contains 3 data types:

- **Point**, this data type is a representation of a simple 3D point and is defined by its coordinates x, y, z, represented as **Number**s. The **Point** defines the initial location of in a **MPointUnit**, as the extension only considers objects moving in a 2D space, the value of z is zero.

- **TimeInterval**, this data type defines the time interval that the **MPointUnit** represents. The time interval consists of a start and an end date, both ORACLE Date types.

- **VariabilityFunction**, this data type represents the translation of the **Point** during the **TimeInterval** and it is defined by the 3 components of the movement v_x, v_y, v_z, represented as **Number**'s in the ORACLE database. In resemblance to the **Point**, the value of v_z is also zero.

Using the previously defined **MPoint** it is possible the create more complex shapes named **MRegion** that consists of a VARRAY or a Table of **MRegionUnit**. A **MRegionUnit**, like a **MPointUnit**, is a representation of a moving region during a defined time interval, and consists of a **TimeInterval** and a list of **MPoint** that define the region in that time window. The list of **MPoint** is a VARRAY or a Table that contains the ids of the **MPoint**. By using this model it is possible to create a moving region that can change the number of points as well as its positions over time, as represented in Figure 4.1.

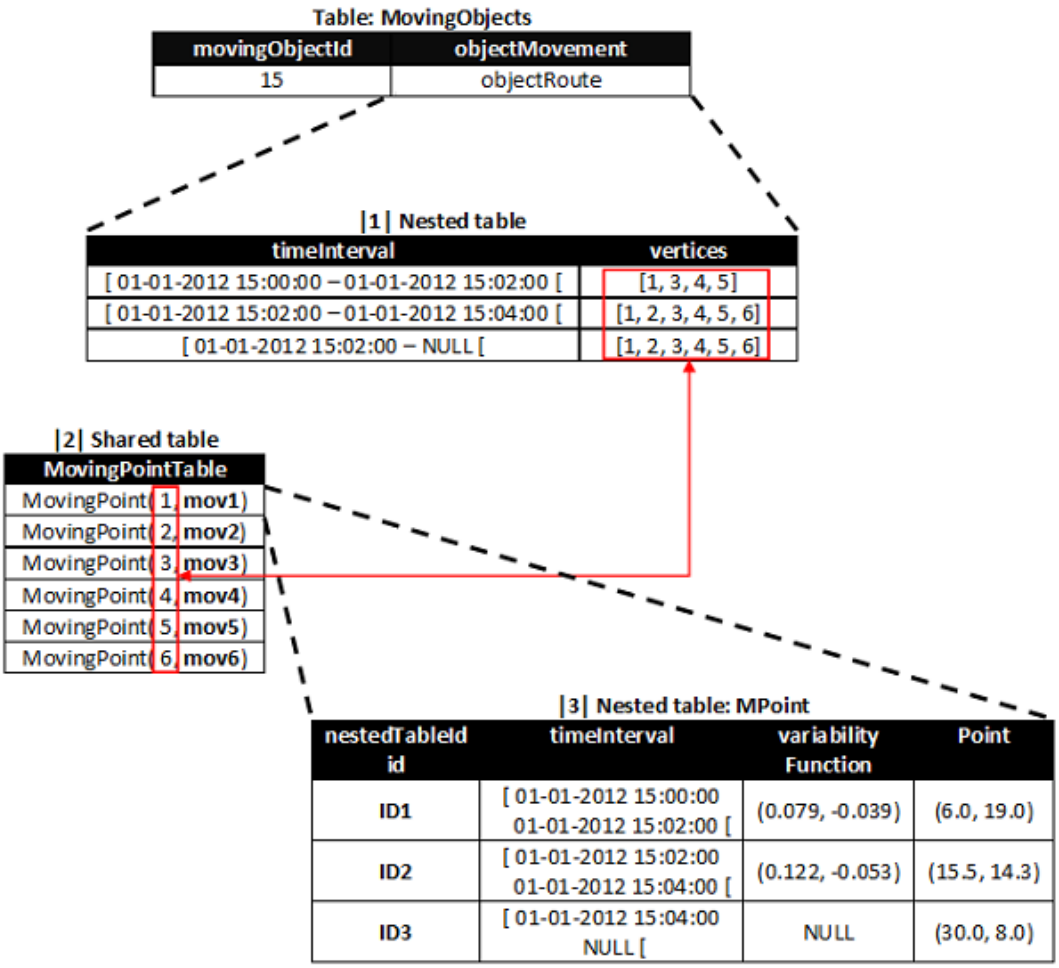


Figure 4.2: Moving region storage in database, source [19]

To use this data model in the ORACLE database a table named MPointTable must be created to store all the **MPoint**s. A table to store the **MRegion**s can also be created to store the moving regions. It is important to notice that the **MPoint**s can be shared between regions to reduce the redundancy and the storage space, see Figure 4.2. Since the points that define a region can change over time it is easy to link several morphing sequences into one to create a moving region. All that is needed is to create a **MRegionUnit** for each morphing using the results of the algorithm and then join the **MRegionUnit**s in a single **MRegion**.

For details and formal definitions on the data model consult the article [19].

4.4 Operations

The algorithms proposed in [19] to implement more complex operations dealing with moving regions have two or three main steps:

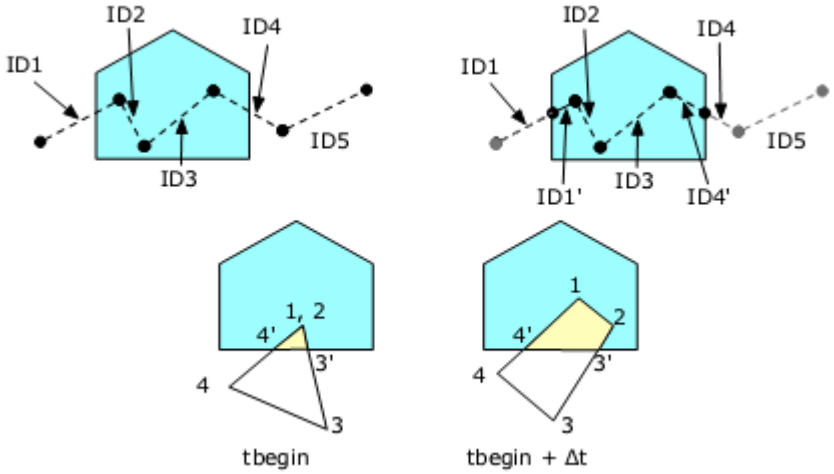


Figure 4.3: Clipping operation for a moving point and moving region, source [19]

---

1. Transform a moving object into spatial (**sdo_geometry**) objects;
2. Perform spatial operations using ORACLE Spatial functions;
3. Assemble the results into a moving object;

The operations can be separated in 3 categories:

- **Projections** are operations to obtain numerical, temporal ar spatial features from a moving object. The footprint of a moving object is the area travelled by this object over an associated time. For instance the spatial projection of a **MPoint** is a line. If the time of a projection is an instant, then the result will be the representation of that object at that time instant.
- **Predicates** results are *true* or *false*. This kind of operations normally test if a certain condition happens, for example if a moving object is within a certain distance from another object or if it intersects some other geometry.
- **Clipping** are the most complex operations and normally must follow the three steps enumerated early, unlike the other two categories that can be solved in just two steps. The Clipping operations enable filtering spatio-temporal values according to a given criteria. The result is a subset of the initial spatio-temporal value for which the criteria holds. Examples of these operations can be a moving object during a time interval or the intersection of a moving object with a certain geometry, in Figure 4.3 is presented an example of the intersection of a moving point and moving region with a static geometry.

For more details about these operations and its implementations see the article [19].

4.4.1 Intersection of moving objects

This operation can be considered a clipping operation and the result of the intersection of two moving regions will be either empty, another moving region or several moving regions. In

Figure 4.4 are shown 3 examples of moving object interceptions: in the first the two regions never intercept (empty case), in the second the two moving regions intercept in the cyan filled rectangular area (simple moving region case), in the third case the interception of the two regions in some instants is composed on the distinct regions (multiple moving region case). Due to some limitations on the used spatio-temporal extension only the first two cases will be considered. "As the current version of the data model does not include a data type for the representation of multiple moving regions, the spatial clipping algorithms can only operate with convex shapes. This ensures that the intersection of two moving regions returns always a single moving region", citation from [19].

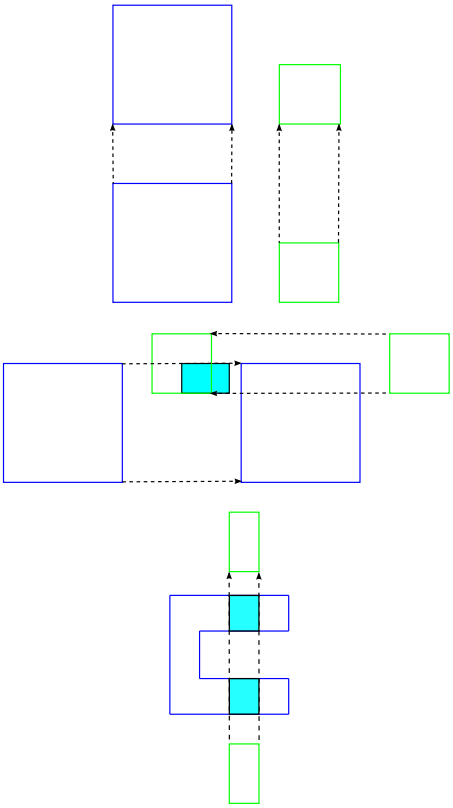


Figure 4.4: Object intersection results

4.5 Implementation of clipping operation

4.5.1 Java integration

A solution for the moving objects intersection operation can be found by using polygon morphing algorithm described in Chapter 2, using the JServer introduced in the *Oracle8 i*. The JServer integrates a Java Virtual Machine(JVM), called Aurora, enabling runtime environment and Java class libraries to be executed in the database. The Aurora JVM can execute Java methods as Java Stored Procedure(JSP). The JServer also provides integration of the Java methods and classes with the PL/SQL language used to create and define the spatio-temporal extension. This way an easy integration of the previous implemented algorithm is

possible with the ORACLE database, see Figure 4.5.

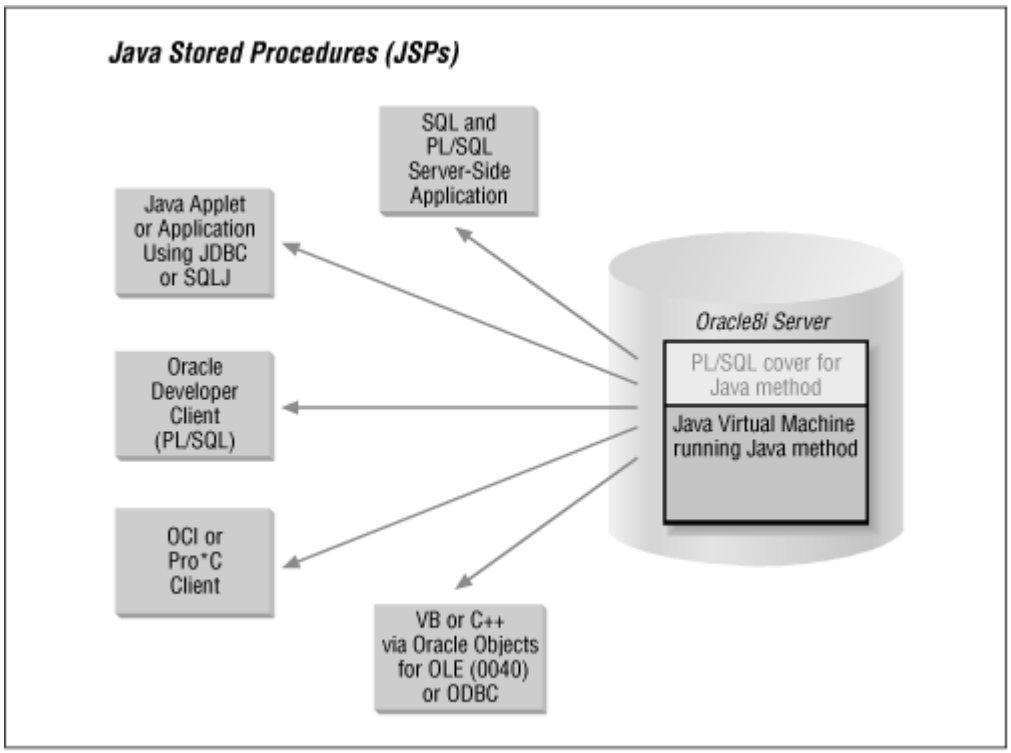


Figure 4.5: Accessing JSPs from within the Oracle database, source [21]

To use a Java method there are five steps that must be followed:

- Compile the Java source code;
- Load all classes or jar package needed into the database using the *loadjava* tool via a command line utility;
- Create a wrapper using PL/SQL to publish the Java methods as PL/SQL functions;
- Grant the required privileges to the PL/SQL wrapper functions;
- Call the PL/SQL wrapper functions.

For details see Chapter 9 of [21].

A PL/SQL wrapper can be created via the CREATE OR REPLACE FUNCTION statement by using the AS LANGUAGE JAVA clause as defined in the PL/SQL syntax. An example of how to create the wrapper for a JAVA function can be seen below,

```
CREATE OR REPLACE FUNCTION jMorphPolygon(sdo1 SDO_GEOMETRY, sdo2
SDO_GEOMETRY)
RETURN numberArray
AS LANGUAGE JAVA
NAME 'morph.PolygonMorph.calcMorph(oracle.sql.STRUCT, oracle.sql.STRUCT) return
oracle.sql.ARRAY';
```

this example creates a wrapper for a function that receives two SDO_GEOMETRY objects as *oracle.sql.STRUCT* and returns an array of *double* as an *oracle.sql.ARRAY*. The array of doubles contains the coordinates for each point and its movement as,

$$\{V0_{cx},\ V0_{cy},\ V0_{cz},\ V0_{vx},\ V0_{vy},\ V0_{vz},$$

$$V1_{cx},\ V1_{cy},\ V1_{cz},\ V1_{vx},\ V1_{vy},\ V1_{vz},\ ...$$

$$Vn_{cx},\ Vn_{cy},\ Vn_{cz},\ Vn_{vx},\ Vn_{vy},\ Vn_{vz}\}$$

where $Vx_{cx,cy,cz}$ correspond to the (x, y, z) coordinates of the point x and the $V0_{vx,vy,vz}$ to the movement of the x point in the (x, y, z) dimensions. The *oracle.sql.STRUCT* and *oracle.sql.ARRAY* classes are imported from the ODBC package, that is a JDBC API extension for Oracle Databases.

There are several advantages to use a Java method to implement the morphing algorithm instead of a PL/SQL implementation. The main advantage is that the Java methods can execute sequential programming task with greater efficiency while PL/SQL focuses in database tasks. In the moving objects intersection method the Java method handles the morphing related tasks and the insertion of information in the database is handled by the PL/SQL method, taking advantage of the best of each language to solve the problem. Using Java to implement the sequential programming operations also reduces the amount of time needed to implement the solution and do not require more data types and information on the database.

There are other technologies that allow the interaction between Java and the Database like JDBC. This solution could be used to insert the data into the DB instead of using PL/SQL. The JDBC is widely known and uses a JAVA API that provides a connection to most DBMS and enables JAVA programs to execute SQL statements and manage the information on those DBs. In fact, the application shown in Section 4.7 uses JDBC to interact with the database.

4.5.2 Operation structure

The challenge that was proposed in this work was to implement a clipping operation for moving regions with complex shapes for the spatio-temporal extension proposed in [19]. The actions performed were the following:

- In the first step the method starts by calculating the intersection of the time interval of the two moving regions. Then the continuous time interval is discretized in instants, using a configurable argument, as proposed in [19].

- The second step consists in calculating the static intersection of the two moving regions for each time instant. To calculate these intersections it is created a spatial projection of each region at each time instant and finally using ORACLE Spatial functions the interception is computed.

- The third step assembles all the static interceptions in a new moving object of type **MRegion**. By using the morphing algorithm to create a morph sequence between two static interceptions, it is possible to create the **MRegionUnit** to represent the moving region from the clipping operation. The result of the clipping method will be the **MRegion** that contains all the **MRegionUnit**s generated. However to create a **MRegion** its **MPoint**s must be saved in the MPointTable in the database. This step is explained in Section 4.5.3.

It is also important to refer that the first and third steps are implemented in PL/SQL while the second step uses the JAVA wrapper.

4.5.3 Insert database data

As said before a moving region contains a collection of **MPoint** ids. The result of the intersection operation is a moving region and so, it is required to first store the **MPoint**s in the MPointTable, but it creates a problem because the result of any operation should be temporary unless the user decides to save it. However since the **MPoint**s are stored in a table they became persistent. The solution to this problem was to create a table that keeps the record of these **MPoint**s so that the user can remove then when they are no longer needed. This table is called TMP_MPoints. The table records the **MPoint**s created by the method using its MPId, its creation date and the SESSION_USER.

4.6 Accessing the data base using JDBC

The JDBC is a framework that enables JAVA applications to use a wide variety of DBMS. Using JDBC it is possible to connect to the DBMS and to run queries in the DB without need to perform any configuration besides the connection string. To use JDBC to access the DB with the spatio-temporal extension some additional classes must be created to map the new data structures of the extension to JAVA objects, see Figure 4.6. These new objects will be called entities and each entity correspond to a data structure discussed in the Section 4.3. By analyzing the class diagram shown in Figure 4.6 it is possible to see that the entities have a relation to each other similar to the relations between the data structures on the database. An entity can have two kinds of builders, one using a STRUCTURE or ARRAY and another using JAVA data types. The first type constructs the entity from data received from the database (via JDBC), where a STRUCTURE from the JDBC contains a generic data structure and an ARRAY contains a VARRAY from the database. The second type constructs the entity using JAVA data types as double, Timestamp, BigDecimal, etc. An entity can also contain some attributes corresponding to attributes in the database. Each data type in the database is mapped into a type in JAVA, like a number to a BigDecimal or a DATE to a Timestamp. Another important method shared by all the entities is the *toSQL* that creates a string containing the SQL statement that can be used in a query to create, delete or alter a record.

To run a query in the database the JDBC provides a series of classes namely the *Statement*, *CallableStatement* and *PreparedStatement*. By using these classes an user can run a SQL query in the database using a *Connection* and a String containing the SQL statement to execute. The *Connection* class provided by the JDBC enables the user to create a connection to the database using a connection string, it also provides methods to create save points and transactions. In Figure 4.7 represents how a JAVA application can access the database.

4.7 Loading spatiotemporal data into the database

The use of entities and JDBC makes development of applications using the spatio-temporal database easier. A good example is the application to insert spatio-temporal data into the

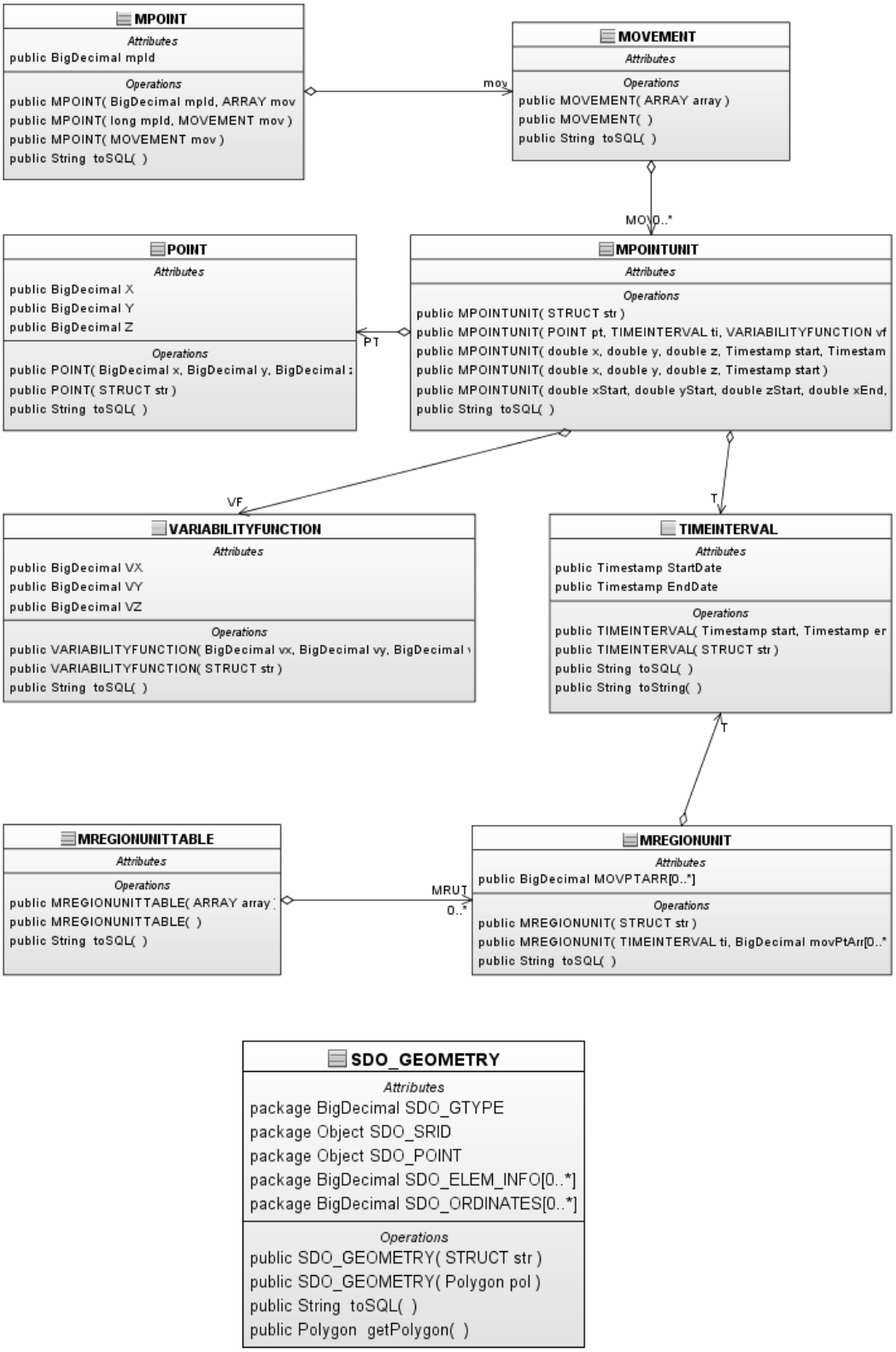


Figure 4.6: JAVA entities class diagram

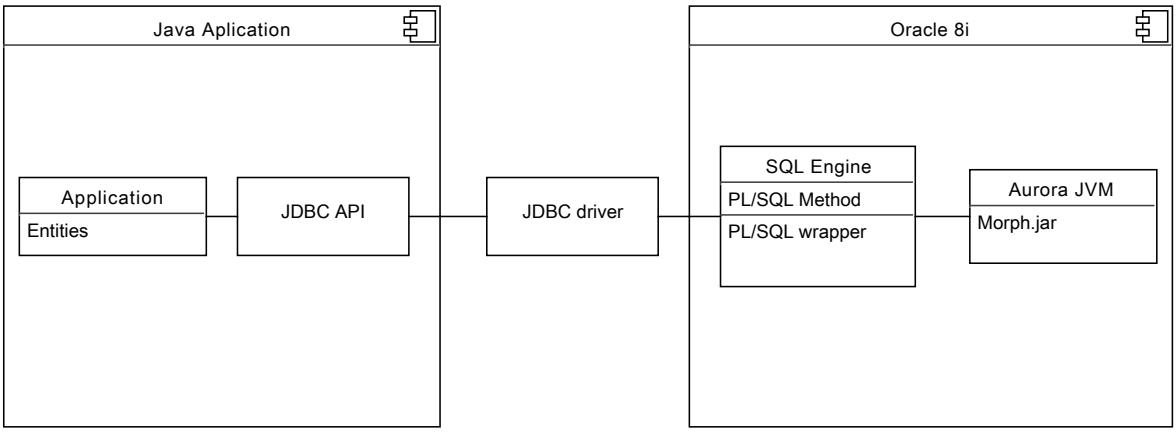


Figure 4.7: Application access to database diagram

database. This application creates a morphing sequence from the segmented polygons obtained from the segmentation application described in Section 2.1. This morphing sequence is converted into spatio-temporal data and inserted in to the database using JDBC. The application was used to construct the movement of the objects and to store it in the spatio-temporal database.

Figure 4.8 depicts the layout of the application where the source and target polygons are displayed at, in the lower half a series of controls are available in three tabs. The first two tabs provide controls to load the polygons and to detect the Feature Points, the third tab controls the vertex correspondence and the insertion of the data into the database.

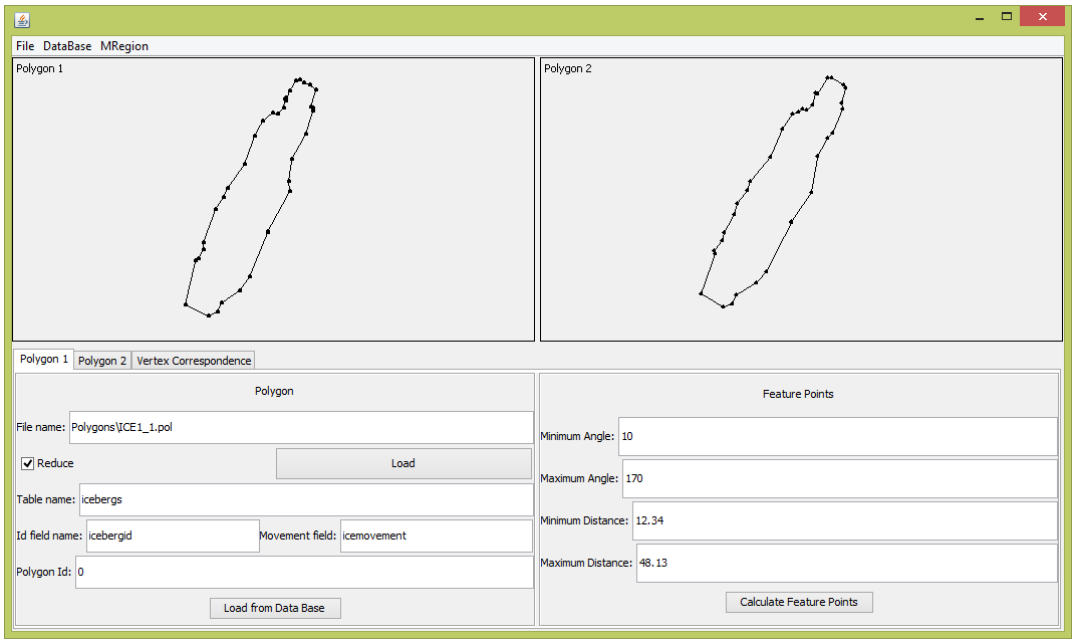


Figure 4.8: Application - load polygon and detect FPs

To create a morphing sequence first a polygon must be loaded, either from a file or from a table existing in the DB. When loading a polygon from the DB, the last instance of the

loaded **MRegion** is selected, which means that new data can be inserted to the moving object. Once the polygon is loaded the feature points must be detected from the two polygons, see Figure 4.8. In the third tab the correspondences are created manually or automatically. Finally the user can insert the resulting morphing sequence in the database or see a preview of the sequence to evaluate its quality before inserting the data, see Figure 4.9.

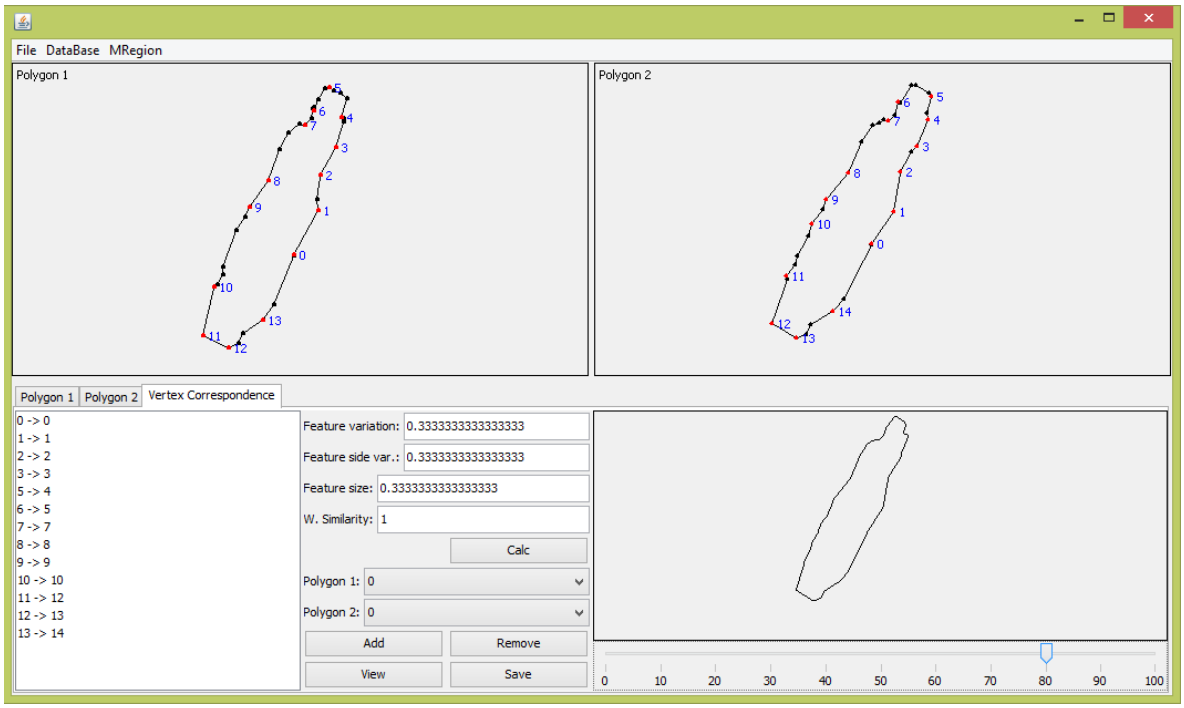


Figure 4.9: Application - create morph sequence and insert data

The application also enables the user to view the geometry of a **MRegion** at any instant of its time interval as shown Figure 4.10, where the scroll bar controls the time instant that is represented.

4.8 Case study

The case study used in this work is a set of satellite images of icebergs taken in the same region during several weeks. An iceberg can move and change shape during time and provides a good real data set where the algorithm can be applied. To create this data set the shape of the icebergs must be obtained from the satellite images, using the image segmentation application presented in Section 2.1, then the resulting shapes are used to create a sequence of polygons. Each sequence of polygons can be used to create a morphing sequence that will represent the iceberg during a time interval defined by the dates of the images.

The case study consists of ten satellite images of the iceberg known as *B-15*, this iceberg is considered the world largest iceberg, measuring about $400km$ long and $40km$ wide. The satellite images focus some of its fragments, more notably the $B-15a$ and the $B-15j$ which will be refereed as iceberg 1 and 2. An example of a images of the two icebergs can be seen in Figure 4.11, where to bigger iceberg is the iceberg 1 and the smaller iceberg the iceberg 2. For iceberg 1 the most predominant movement is a translation along the cost line where the

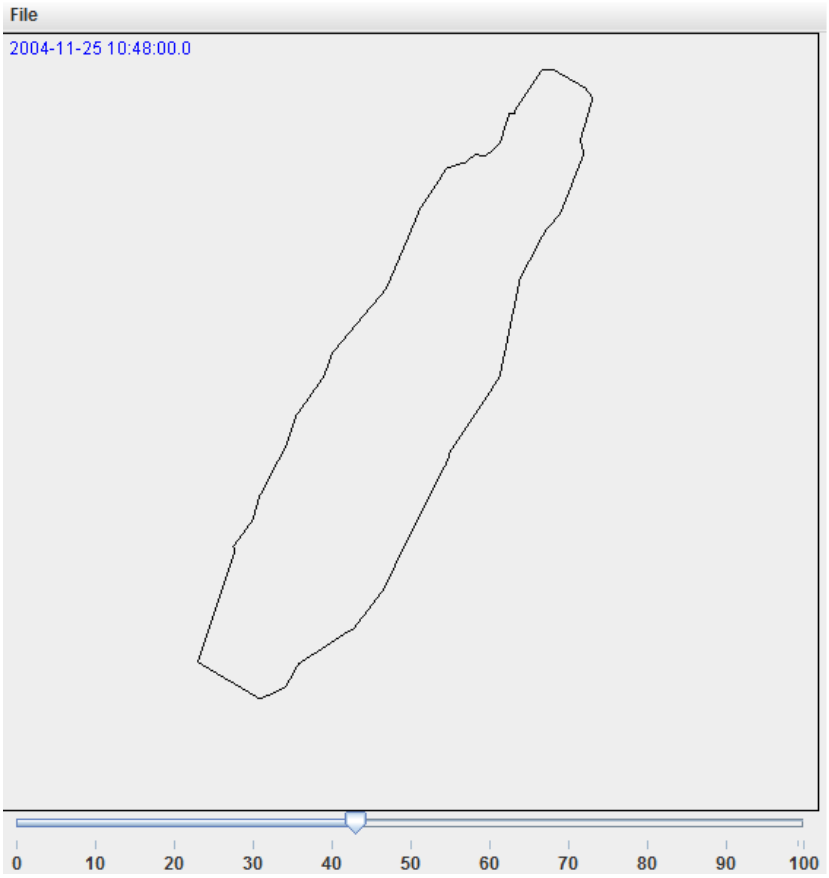


Figure 4.10: Application - view of a **MRegion**

iceberg 2 suffers mostly a rotation around its center. Both the icebergs suffer deformations during the sample time due to melting or fragmentation. The images used in this case study are taken at the following dates: 19-11-2004, 21-11-2004, 26-11-2004, 02-12-2004, 04-12-2004, 07-12-2004, 13-12-2004, 20-12-2004, 23-12-2004 and 02-01-2005.

The satellite images were download from the NASA site, *http://www.nasaimages.org/*, however the data set is no longer available.

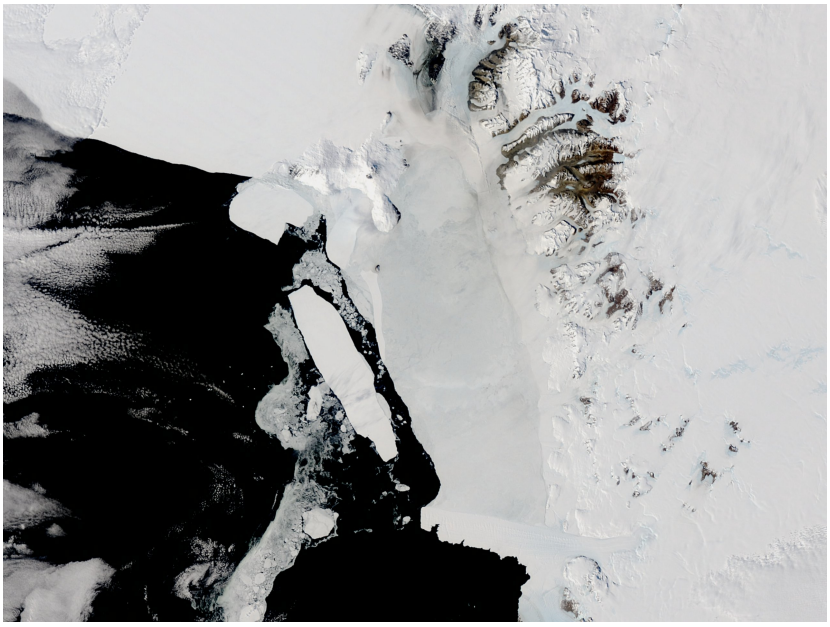


Figure 4.11: Satellite image taken in 02-12-2004

To create a morphing from the polygons sequence the application presented in Section 4.7 was used. The application can read two polygons contours from files and create a polygon morph between this two polygons. It is possible to see and change the feature point detection or the vertex correspondence parameters from the application interface. The vertex correspondences can be changed or created manually. An animated morphing can also be viewed in this application.

To test the morph algorithm using this real data set some metrics where defined to evaluate the quality of the morph obtained. The metrics are:

- Intersections

  This measure tests the occurrence of vertex intersections during the morphing process. To test if there are vertex intersections 100 frames are generated for a morphing sequence and for each frame it is tested if any of its edges intercepts another, the result is the number of frames where occurs one or more edge interceptions. The value of this measure variate from 0 to 100 as the number of frames with edge interceptions variate.

- Snapshot similarity

  In this measure the similarity between two shapes is estimated. The similarity can be calculated as $1 - \frac{Area(E \bigcup S) - Area(E \bigcap S)}{Area(E \bigcup S)}$, where E is the estimated shape calculated by the morphing algorithm and S is the shape obtained from an image, see in

Figure 4.12 the yellow and red polygons correspond to S and E, respectively. To use this measure three snapshots of an iceberg are sorted by date, for example by using the images on the dates 19-11-2004, 4-12-2004 and 7-12-2004. The three polygons are segmented, then is applied the algorithm to generate a morphing sequence from the polygons corresponding to dates 19-11-2004 and 7-12-2004. In Figure 4.12 the green polygon correspond to the iceberg of image taken in 19-11-2004 and the blue polygon the iceberg from image taken in 7-12-2004. The resulting area from $Area(E \bigcup S) - Area(E \bigcap S)$ corresponds to the area denoted in black in Figure 4.13 and represents the difference between the estimated polygon E and the segmented polygon S. The greater this is area in relation to the total area $Area(E \bigcup S)$ the greater the difference between this two polygons. This measure tends to 1 when the 2 polygons are similar.

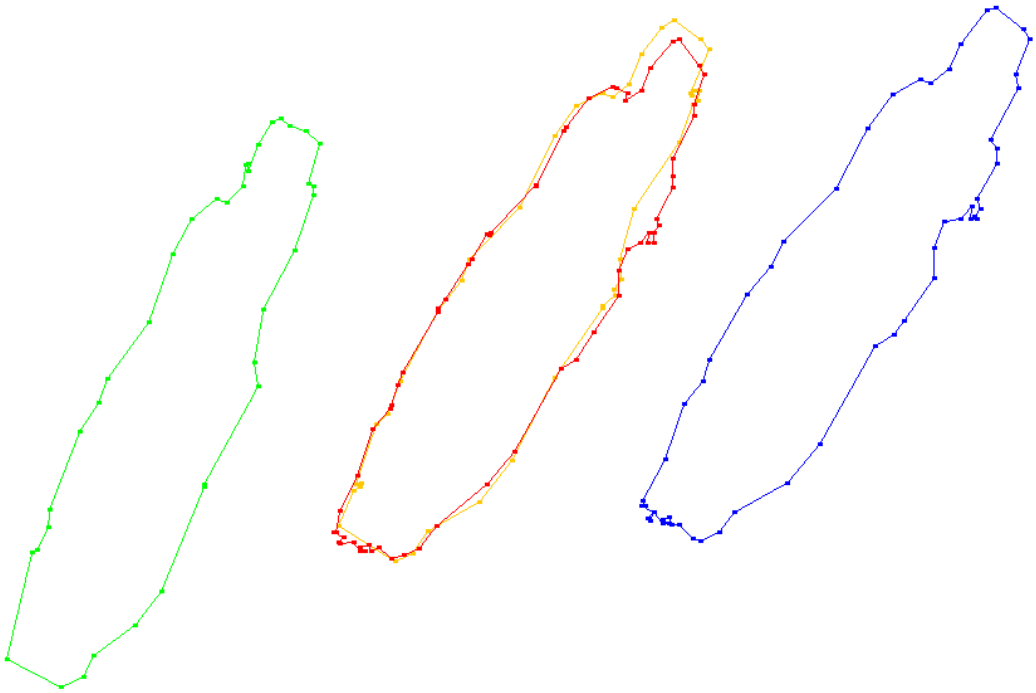


Figure 4.12: Snapshot similarity

With the measures defined above a set of tests for iceberg 1 and 2 where done. These tests consisted in creating several morph sequences using the polygons obtained by segmentation of the images. The morphing were either fully automatic or manual, were the manual uses an initial correspondence set by the user. Each morph was calculated and loaded into the database, then the value for each measure was calculated and the obtained results are shown in Table 4.1 and Table 4.2. Some of the values in the tables are missing, because edge interceptions can cause invalid polygon shapes at some instants. In these instants no shape is returned by the database and these situations were discarded from the final results.

By analysing the values it is possible to see that the average value for the snapshot similarity is about 80% and in more than half cases no intersection is generated. The average value for the snapshot similarity is a very good value considering the lack of precision of the segmentation application used, which reduces the precision of the results. In some cases the estimated polygon E is more similar to the actual iceberg than the segmented polygon S. It

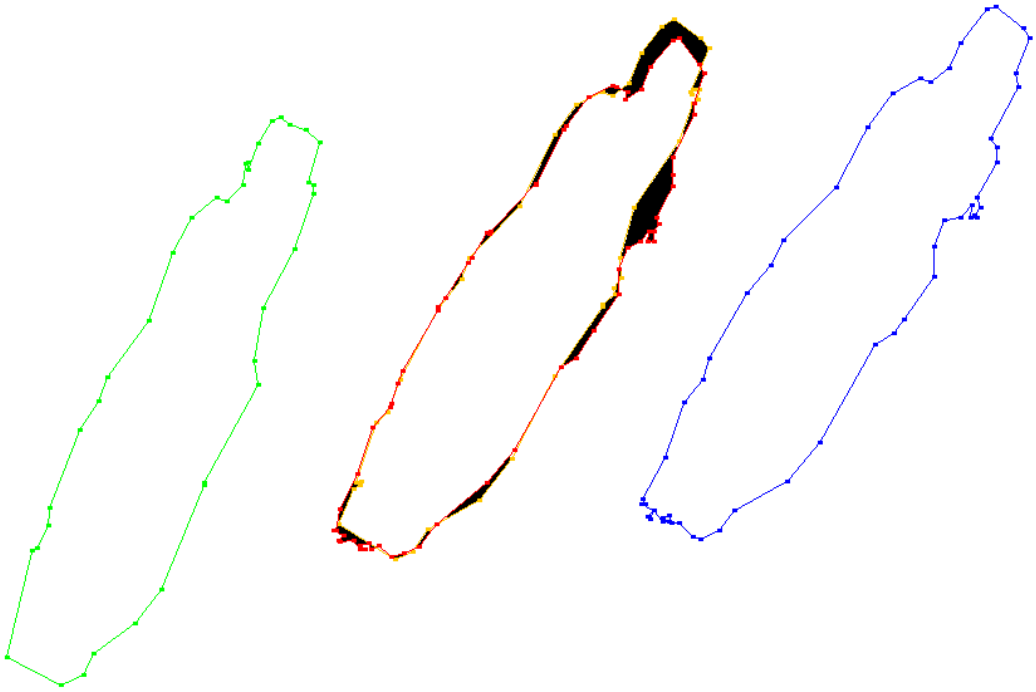


Figure 4.13: Snapshot similarity: difference area(*black*)

is also important to consider that the movement of the iceberg is not always linear nor its melting or fragmenting rates, which also introduces some error between the estimated iceberg shape and the real one.

4.9 Results

In this section some results of the moving object clipping method will be presented, the results consists of an intersection of two moving objects, a triangle and a square. The first region consists of a rectangular triangle that transforms into another rectangular triangle, see Table 4.3 to view points coordinates. The second region is a large square with 5000 pixels length and centred in $(2500, 2500)$ that moves to $(2500, 2400)$, the Table 4.4 show the points coordinates.

The Figure 4.14 shows the result of the clipping of the two **MRegion**s. Each sub figure represents the result at a given instant. The polygons with black edges are the two **MRegion**s, and the polygon with red edges and dark grey filling is the result of the clipping of the two **MRegion**s. As can be seen the clipping polygon corresponds to the interception of the moving regions.

The intersection clipping algorithm should only be used to calculate the clipping between two convex shapes, to avoid cases where the resulting intersection have more than one region, see Figure 4.4.

Morph				Results	
Begin	E/S	End	Type	Intersections	Similarity
19-11-2004	26-11-2004	4-12-2004	Automatic	61	84%
			Manual	2	—
26-11-2004	4-12-2004	7-12-2004	Automatic	0	82%
			Manual	34	—
4-12-2004	7-12-2004	20-12-2004	Automatic	0	—
			Manual	0	—
19-11-2004	26-11-2004	7-12-2004	Automatic	28	71%
			Manual	0	87%
19-11-2004	4-12-2004	7-12-2004	Automatic	28	78%
			Manual	0	90%
26-11-2004	4-12-2004	20-12-2004	Automatic	0	81%
			Manual	0	82%
26-11-2004	7-12-2004	20-12-2004	Automatic	0	77%
			Manual	0	79%

Table 4.1: Results for iceberg 1

Morph				Results	
Begin	E/S	End	Type	Intersections	Similarity
19-11-2004	21-11-2004	26-11-2004	Automatic	30	—
			Manual	12	—
21-11-2004	26-11-2004	2-12-2004	Automatic	50	—
			Manual	0	80%
26-11-2004	2-12-2004	4-12-2004	Automatic	0	92%
			Manual	0	90%
19-11-2004	21-11-2004	2-12-2004	Automatic	0	87%
			Manual	0	80%
19-11-2004	26-11-2004	2-12-2004	Automatic	0	76%
			Manual	0	76%
26-11-2004	4-12-2004	20-12-2004	Automatic	0	74%
			Manual	35	—
26-11-2004	7-12-2004	20-12-2004	Automatic	0	89%
			Manual	35	—

Table 4.2: Results for iceberg 2

MPoint	Start	End
MPoint 1	$(-200, 100)$	$(100, 100)$
MPoint 2	$(-100, 100)$	$(200, 100)$
MPoint 3	$(-200, 200)$	$(200, 200)$

Table 4.3: Mpoints for MRegion 1 (Triangle)

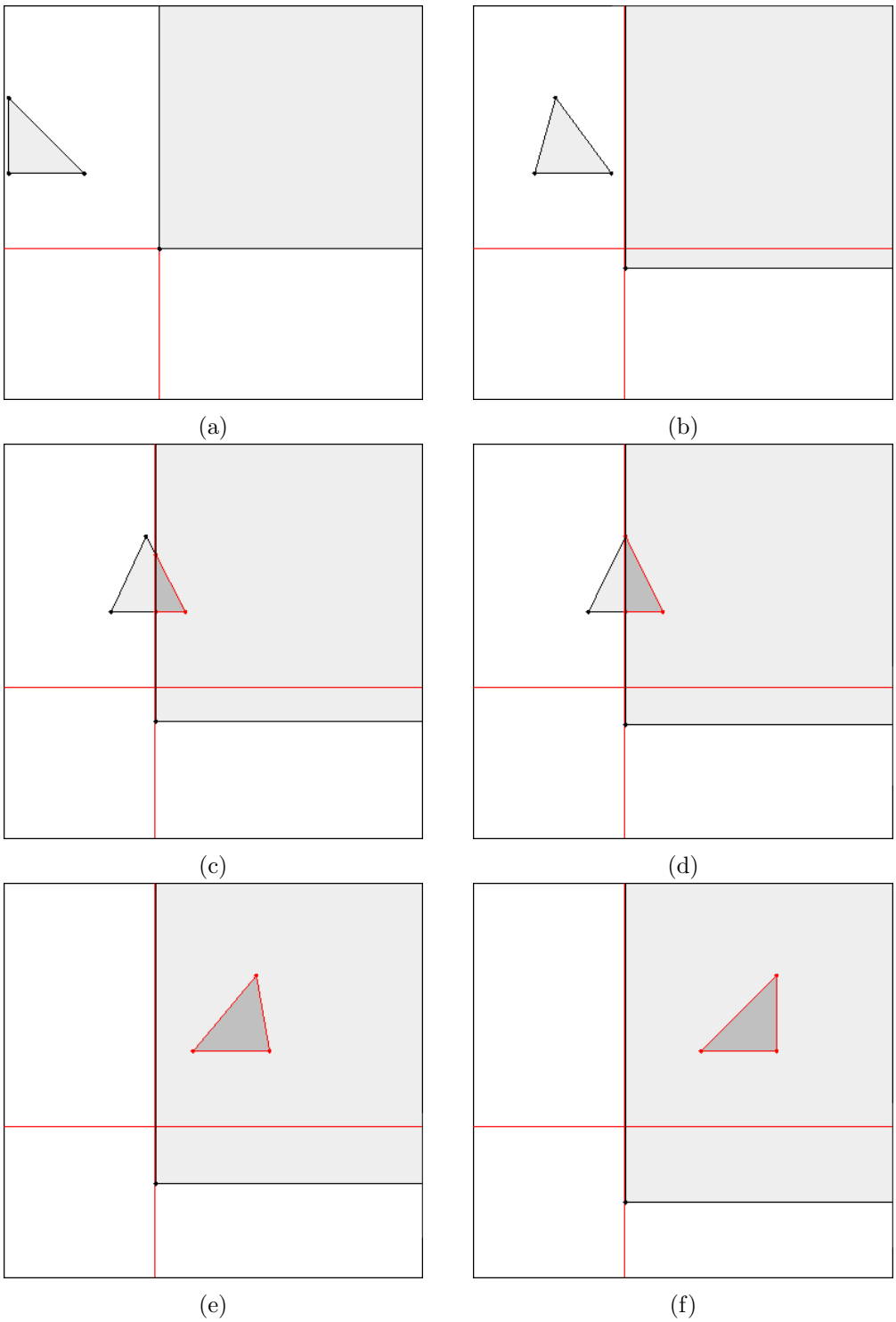


Figure 4.14: Intersection of MRegion 1 with MRegion 2

MPoint	**Start**	**End**
MPoint 1	$(0, 0)$	$(0, -100)$
MPoint 2	$(5000, 0)$	$(5000, -100)$
MPoint 3	$(5000, 5000)$	$(5000, 4900)$
MPoint 4	$(0, 5000)$	$(0, 4900)$

Table 4.4: Mpoints for MRegion 2 (Square)

Chapter 5

Conclusions

This work presents several techniques proposed in different domains of research that may be used to implement adequate tools to insert data of moving objects into spatio-temporal databases. It also presents a framework for the acquisition of moving objects data based on morphing techniques. This is the first work presenting a set of methods for creation and evaluation of moving objects representations from real-world data sources. The experiments show that it is possible to achieve good quality representations for moving objects with extent and it also reveals that there are issues like the intersection of vertex paths and the representation of rotation transformations that require future investigation.

With the developed work we were able to create a spatio-temporal data set from a collection of satellite images and load it to a spatio-temporal database. The application to load the data is simple to use and can easily be used to create more complex data sets in different scenarios. The clipping operation developed also fulfils its requisites, however it still have room for improvements, such as use the existing **MPoint** to set a initial correspondence and reduce the number of **MPoint** created.

5.1 Contributions

With the methods proposed in this work it is possible to create real data sets and load them to spatio-temporal databases. To our knowledge, this work is a first step towards the development of efficient tools for the acquisition of spatio-temporal data in moving objects databases. Using this data sets it is possible to better test and to improve the quality of the existing and future spatio-temporal extensions. We also present some tools to evaluate the quality of the generated moving objects.

Since the technologies used during the development of this project are widely known it makes the work simpler to reuse and improve in the future, proving a basis to the development of tools for spatio-temporal data acquisition in real-world applications.

This work also shows that is possible to implement morphing algorithms to solve some of the more complex operations in a spatio-temporal extension.

Part of the work presented in this dissertation has been accepted for presentation and publication at GEOProcessing 2013: *Luís Paulo, José Moreira, Paulo Dias. Morphing Techniques in Spatio-temporal Databases. In proceeding of the Fifth International Conference on Advanced Geographic Information Systems, Applications, and Services, February 2013.* (To appear). With this publication we hope that some improvements and applications of this

work appear in the future.

5.2 Future Work

In the development process of this work we can across a series of new issues and challenges. Most of these issues were solved and the solution to them is presented in this thesis. We consider two issues that are subject to improvements in the future, a better solution to the vertex path problem and an algorithm to "connect" the **MPoint**s of the **MRegionUnit**s generated by the morphing algorithm. To the first issue some solutions were presented in Section 2.2, namely in [5], [6] and [7]. We suggest the implementation of the method proposed in [6], since in creates a skeleton of the polygon, it retains the general shape of the polygon better than the other methods. The method can also calculate the vertex path for non convex shapes, making it a more robust algorithm.

Bibliography

[1] André Filipe da Silva Oliveira. Edição e visualização de obejctos geográficos dinâmicos. Technical report, Departamento de Electrónica Telecomunicações e Informatica, Universidade de Aveiro, June 2011.

[2] Thomas W. Sederberg and Eugene Greenwood. A physically based approach to 2d shape blending. *SIGGRAPH Comput. Graph.*, 26:25–34, July 1992.

[3] Ligang Liu, Guopu Wang, Bo Zhang, Baining Guo, and Heung-Yeung Shum. Perceptually based approach for planar shape morphing. In *Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, PG '04, pages 111–120, Washington, DC, USA, 2004. IEEE Computer Society.

[4] Yuefeng Zhang. A fuzzy approach to digital image warping. *IEEE Comput. Graph. Appl.*, 16:34–41, July 1996.

[5] Craig Gotsman and Vitaly Surazhsky. Guaranteed intersection-free polygon morphing. *Computers and Graphics*, 25(1):67 – 75, 2001. Shape Blending.

[6] Michal Shapira and Ari Rappoport. Shape blending using the star-skeleton representation. *IEEE Comput. Graph. Appl.*, 15(2):44–50, March 1995.

[7] Thomas W. Sederberg, Peisheng Gao, Guojin Wang, and Hong Mu. 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, pages 15–18, New York, NY, USA, 1993. ACM.

[8] Dmitry Chetverikov and Zsolt Szabo. A simple and efficient algorithm for detection of high curvature points in planar curves, 1999.

[9] Jan Chomicki, Sofie Haesevoets, Bart Kuijpers, and Peter Revesz. Classes of spatio-temporal objects and their closure properties. *Annals of Mathematics and Artificial Intelligence*, 39(4):431–461, December 2003.

[10] Stéphane Grumbach, Philippe Rigaux, and Luc Segoufin. Spatio-temporal data handling with constraints. *Geoinformatica*, 5(1):95–115, March 2001.

[11] Ralf Hartmut Güting, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, March 2000.

[12] José Antonio Cotelo Lema, Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. Algorithms for moving objects databases. *Comput. J.*, 46(6):680–712, 2003.

[13] Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli, and Markus Schneider. A data model and data structures for moving objects databases. *SIGMOD Rec.*, 29(2):319–330, May 2000.

[14] Ralf Hartmut Gting, Thomas Behr, Victor Almeida, Zhiming Ding, Frank Hoffmann, and Markus Spiekermann. Secondo: An extensible dbms architecture and prototype. Technical report, Department of computer science, University of Hagen, 2004.

[15] Nikos Pelekis, Yannis Theodoridis, Spyros Vosinakis, and Themis Panayiotopoulos. Hermes - a framework for location-based data management. In *In Proceedings of the Proceedings of the 10th international conference on Advances in Database Technology*. Springer-Verlag, 2006.

[16] N. Pelekis, B. Theodoulidis, University of Manchester Institute of Science, and Technology. Department of Computation. *STAU: A Spatio-Temporal Extension for the Oracle DBMS*. UMIST, 2002.

[17] Nikos Pelekis, Nikos Pelekis, Yannis Theodoridis, and Yannis Theodoridis. An oracle data cartridge for moving objects. Technical report, Department of Informatics, University of Piraeus, December 2007.

[18] Lei Zhao, Peiquan Jin, Lanlan Zhang, Huaishuai Wang, and Sheng Lin. Developing an oracle-based spatio-temporal information management system. In *Proceedings of the 16th international conference on Database systems for advanced applications*, DASFAA'11, pages 168–176, Berlin, Heidelberg, 2011. Springer-Verlag.

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

[20] M. Breunig, C. Turker, Michael Bhlen, S. Dieker, R.H. Guting, Christian Sndergaard Jensen, L. Relly, P. Rigaux, H.J. Schek, and M. Scholl. *Architectures and implementations of spatio-temporal database management systems*, volume 2520, pages 263–318. Springer, 2003.

[21] Steven Feuerstein. *Oracle PL/SQL Programming Guide to Oracle8i Features*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1st edition, 1999.