Using Fields of Directions defined on a Triangulation to Obtain a Topology Preserving Continuous Transformation of a Polygon into Another

Antonio Oliveira
Sara do Nascimento
Sonja A. L. Meerbaum
Programa de Engenharia de Sistemas
COPPE-Universidade Federal do Rio de Janeiro
CP:68511 - 21945-970 - Rio de Janeiro
e-mail: Oliveira@cos.ufrj.br

Abstract

In this article we present an algorithm for the following problem: Obtain a continuous transformation of any simple polygon($\mathbf{L(0)}$) into another ($\mathbf{L(1)}$) with the same number of vertices, generating only simple polygons in between, that is: without introducing contour loops or whiskers during the transformation.The transformation should also take every vertex of one polygon into a corresponding one on the other. None of the best known strategies for Contour Shape Interpolation can solve the general version of this problem, although simple multi-stage transformation methods can do it. Multi-stages transformations however, generate intermediate polygons whose shape is not correlated to those of the extreme ones. The approach which will be presented here although elaborate, offers much better possibilities of getting a real blend of the extreme polygons shape at any intermediate instance. A Continuous Transformation obtained by that method is derived from another one between two Fields of Directions($\mathbf{D(i),i=0,1}$) defined on the same Triangulation $\mathbf{T}$ of an Annular Region($\mathbf{U}$) containing the given polygons. Every trajectory of $\mathbf{D(i)}$ cross $\mathbf{L(i)}$ exactly once what allows us to define an homeomorphism between $\mathbf{L(i)}$ and the graph of a continuous function defined on the external border of $\mathbf{U}$. Besides finding the $\mathbf{D(i)}$s and transforming one into the other the method makes use of three more interpolation steps. The overall complexity of the non-optimized version of the algorithm that will be described here, is $O(|T|^2)$.

1.INTRODUCTION.

The objective of this work is to present a method for solving the following problem(**PROBLEM P1**)
Given two simple polygons $\mathbf{L(i)} = [\mathbf{v}_{i1}, \mathbf{v}_{i2}, \ldots, \mathbf{v}_{im}, \mathbf{v}_{i1}]$, $\mathbf{i = 0,1}$ with the same number of vertices and whose interiors have a non-empty intersection $\mathbf{R}$,we want obtain a continuous transformation ($\mathbf{w} \in [0,1] \rightarrow \mathbf{L(w)}$) of $\mathbf{L(0)}$ into ($\mathbf{L(1)}$) such that:
i)For $\mathbf{k=1,\ldots,m}$, the vertex $\mathbf{v}_{0k}$ of $\mathbf{L(0)}$ is taken into the vertex $\mathbf{v}_{1k}$ of $\mathbf{L(1)}$. Suppose that $(\mathbf{v}_{i1},\mathbf{v}_{i2},\ldots, \mathbf{v}_{im}, \mathbf{v}_{i1})$, $\mathbf{i = 0,1}$ are ordered counterclockwisely.
ii)$\forall$ $\mathbf{w} \in [0,1]$ $\mathbf{L(w)}$ is a simple polygon. This in particular means that topological artifacts such as loops and whiskers do not appear in an intermediate contour generated by the transformation.
None of the most used strategies for Contour Shape Interpolation can satisfy (ii) for every possible case. In those strategies we are including, the simple Linear Interpolation of the extreme polygons corresponding vertices, Turtle-Geometry Interpolation of these corresponding vertices[SED93], Minkowsky Sum Interpolation Method and related ones using Set Operators[ROS91] , Contour Morphing methods inspired in Active Contour or based on other Physical Models [SED92] and Skeleton transforming methods. [SED92] presents a method for Shape Interpolation which does not introduce whiskers but which is not loop-proof. Although none of the methodologies mentioned above are able to solve all instances of **P1**, it can always be solved by means of relatively simple multiple stages approaches. An example: We can continuously transform $\mathbf{L(0)}$ into its convex hull($\mathbf{co(L(0))}$)through the method presented in [OLIV93], then $\mathbf{co(L(0))}$

into **co(L(1))** using the Minkowsky Sun Interpolation Method and finally **co(L(1))** into **L(1)** by using the first method again. All in between polygons generated will be simple ones and the overall time complexity of the three stages is **O(m)** which is of course, optimal. Multiple stages methods, however, generate intermediate polygons whose shape is completely uncorrelated with those of both extreme ones instead of being a blend of those polygons shape as a convincing morphing requires.

In this work we will introduce a methodology for solving problem **P1** which can get reasonable results concerning to the shape blending quality, when applied to get a continous transformation between polygons with many concavities.On the other hand, as it is elaborate, it is certainly, not a good option for simple instances of the problem. Our main objective is to indicate how **Fields of Directions defined on a Triangulation(Tfds)** which are simply, functions associating a direction in R^2 to any triangle of a triangulation, can be used for solving **P1.**

2. BACKGROUND

To describe how the method works to solve problem **P1** we will need some definitions which are given below. A first group of them is relative to general **Tfds**: Let **D** be a **Tfd** defined on a triangulation **T** of a set **U**, **t** a triangle of **T** and **v** a vertex of **t**. We will say that **D(t) points to t at v**(Alternatively: **v is pointed by D(t)**) if the trajectory through **v** deternined by **D** contains other points of **t** reached **after(before**, respect.)**v**. See **figure 1**. If **D(t)** is parallel to an edge of **t**, at one vertex of that edge **D(t)** points to **t** and the other is pointed by **D(t)**. Otherwise, either **t** has a single vertex at which **D(t)** points to **t** or **t** has a single vertex which is pointed by **D(t)**.

We will call a **D-Assignment** to a function associating

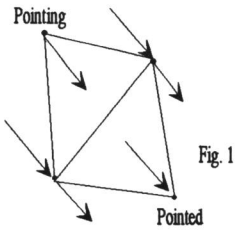

Fig. 1

to every triangle of **T**, one of its vertices, at which **D(t)** points to **t** or is pointed by **D(t)**. A **D-Assignment** is completely determined, except for the triangles where **D(.)** is parallel to an edge. Given two subsets $\mathbf{V_1}$ and $\mathbf{V_2}$ of **U**, we will say that $\mathbf{V_2} >_D \mathbf{V_1}$ if all trajectories reaching $\mathbf{V_1}$, meet $\mathbf{V_2}$ after. A sequence of **U** subsets $(\mathbf{V_1},\mathbf{V_2},\ldots,\mathbf{V_m})$ will be called **D-increasing** if $\mathbf{V_{k+1}} >_D \mathbf{V_k}$, k=1,...,m-1.If **e** is an edge of a triangle **t**,define **i-o type(t,e) as input** if **t** $>_D$ **e**, and **output**, otherwise. In the text below, when there will be no doubts about the **Tfd** being refered we will omit the reference to the **Tfd** in the denominations above and will simply say: **v points to t**, an **Assignment**,etc . If **t** has a vertex **v** which **points to t**(Alternatively: **is pointed by t**) we will say that its **vertex-type** will be **pointing(pointed,** respect.)

We need also some definitions to specify two classes of **Tfds** used by the method. In all of them and in the remainder of this text, **U** will be a polygonal manifold of genus one whose interior contains **L(i), i =0,1** and whose hole is contained in $\overset{o}{R}$. The **Outer Border**(Alternatively: the **Inner Border**) of **U** will be refered by $\mathbf{U}_{out}$($\mathbf{U}_{in}$,resp.). Also, as all **Tfds** refered from now on, will be defined on a triangulation of **U** we will not necessarily make reference to that anymore.

A **Tfd D** will be called **Admissible** if the system of its trajectories satisfy the two following properties:

i) All trajectories determined by **D** start at a point on $\mathbf{U}_{out}$ and end at $\mathbf{U}_{in}$.

ii)No two trajectories determined by **D** intersect each other.

An admissible **Tfd** has neither singularities, nor closed trajectories and can be equivalently defined as one satisfying the following properties:

i) Every vertex **v** on $\mathbf{U}_{out}$ **points** to a single triangle and is pointed by none.

ii) Every vertex on $\mathbf{U}_{in}$ is **pointed** by a single triangle and points to none.

iii)Every vertex in $\overset{o}{U}$ **points** to a single triangle and is **pointed** by only one other. See **figure 2**. An admissi-

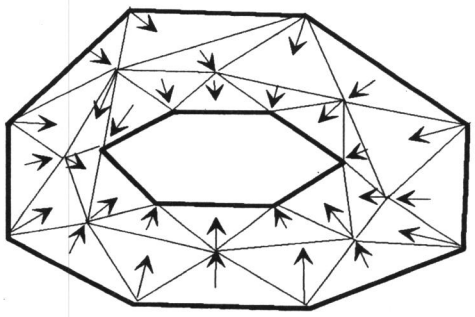
Fig. 2

ble **Tfd D** will be said a **convexer for L(i), i** = 0,1 if any trajectory determined by **D** intersects **L(i)** only once(See **figure 3**). If the triangulation where **D** is defined is constrained by **L(i)** then **D** will be a **convexer for L(i)** iff every vertex of **L(i) points** to a triangle inside **L(i)** and **is pointed** by a triangle outside it.

Finally, let **D** be an Admissible **Tfd** defined on a triangulation **T** of **U** and call $\Upsilon_{\mathbf{T}}$ to the set of these Tfds. The four functions presented below are well defined

i)s_D: **U**$\rightarrow$ $\mathbf{U}_{out}$, such that $s_D(p)$is the starting point

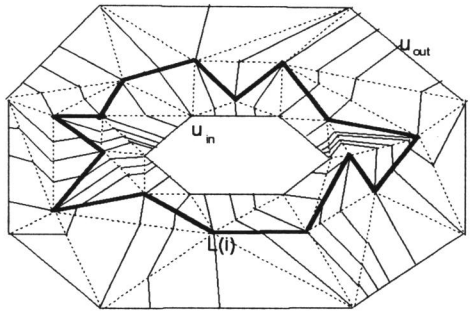

Fig. 3

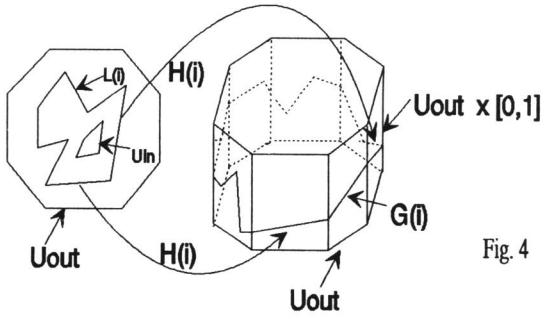

Fig. 4

of the trajectory through **p** determined by **D.**

ii)Δ_D: **U**$\rightarrow [0,1]$ | $\Delta_D(\mathbf{p}) \triangleq$ the ratio between the length of the part from $\mathbf{s}_D(\mathbf{p})$ to **p** of the trajectory through **p** determined by **D**, and the total length of that trajectory.

iii)$\mathbf{H}_D$:**U**$\rightarrow \mathbf{U}_{out}\times[0,1]$ | $\mathbf{H}_D(\mathbf{p}) \triangleq(\mathbf{s}_D(\mathbf{p}),\Delta_D(\mathbf{p}))$. $\mathbf{H}_D$ is a piecewise rational homeomorphism composed of less than | **T** | ^{2} rational functions.

iv)$\Gamma:\Upsilon_T \rightarrow \Gamma(\Upsilon_T)$ | $\Gamma(\mathbf{D}) = \mathbf{H}_D$. Γ is continuous considering any norm for both domain and image since their dimensions are finite.

3. THE METHOD.

The approach for solving problem **P1** mentioned before can now be outlined as follows:

First of all we get a convexer **D(i)** for each **L(i)**, **i = 0,1**. Both **D(i)**s have to be defined on the same triangulation of **U**. Call **P1.1** to the problem of finding these convexers. We must observe that the two **L(i)**s may not have a common convexer but, adequately translating one of them it is always possible to get that property. However we will consider here that the **L(i)**s have already been properly positioned to favour their shapes blending during the transformation and for that reason we do not think about moving them. Having the convexers we determine, for **i=0,1**, the functions $\mathbf{s}_i = \mathbf{s}_{D(i)}$ and $\Delta_i = \Delta_{D(i)}$ which are the coordinate functions of the homeomorphism $\mathbf{H(i)}\equiv\mathbf{H}_{D(i)}$. Call **P1.2** to the problem of having an Admissible **Tfd D**, determine finite representations of the functions $\mathbf{s}_D$ **and** Δ_D, which we have to solve for each **D(i)**. Also observe that **H(i)** takes **L(i)** onto the graph **G(i)** of the function $\mathbf{f(i)}:\mathbf{U}_{out}\rightarrow$ **[0,1]** such that $f(\mathbf{i})(s_i(q)) = \Delta_i(q)\ \forall\ \mathbf{q} \in \mathbf{L(i)}$. See **fig.4**. Now, let $(\rho_k^i = (s_i(v_k^i), \Delta_i(v_k^i)))$; $k = 1, ..., m$; **i= 0,1** and consider the following two problems:

P1.3) Obtain a continuous transformation of **G(0)** into **G(1)**, which takes ρ_k^0 into ρ_k^1,**k =1,...,m** and such that any **G(w)** generated in between is the a graph of a function $\mathbf{f(w)}:\mathbf{U}_{out} \rightarrow$ **[0,1]**.

P1.4) Obtain a continuous transformation of homeomorphism **H(0)** into **H(1)**, such that any **H(w)** generated in between is also an homeomorphism between **U** and $\mathbf{U}_{out} \times [0,1]$. Since the function Γ given above is continuous we can solve this problem by:

i)Find a continuous transformation of **D(0)** into **D(1)**, such that any **D(w)** generated in between is also an admissible **Tfd**.

ii) Obtain **H(w)** as Γ**(D(w))**, what means solving problem **P1.2** for **D=D(w).**

$\mathbf{L'(w)} = \mathbf{H(w)}^{-1}\mathbf{(G(w))}$ is certainly a simple closed curve but the fact of $\Delta_w(.)$ not being linear implies that it is not in general a polygon. For that reason to get a solution of **P1** we obtain an appropriate polygonal approximation **(L(w))** of $\mathbf{H(w)}^{-1}\mathbf{(G(w))}$whose contour has no self intersections and which varies continuously with **w**. Call **P1.5** to the problem of obtaining that approximation from **H(w)** and **G(w).**

To help the comprehension of the method description given above we will rewrite it in a slightly different way.

- Through the convexer **D(i)** we obtain a system of reference **S(i)** for **U** where the coordinates of the points on **L(i)** form a set **G(i)** which is a Graph of a piecewise rational function defined on $\mathbf{U}_{out}$; **i = 0,1.** Geometrically speaking **G(i), i = 0,1** can be interpreted as the image of **L(i)** obtained by an homeomorphism of **U** onto the bounded cylindrical surface $\mathbf{U}_{out}$ **[0,1]** which takes the **D(i)**-trajectories into linear segments parallel to $\mathbf{e}_3$ **= (0,0,1).** The simple fact of the lines parallel to $\mathbf{e}_3$ crossing the **G(i)**s at a single point makes solving problem **P1** for the **G(i)**s much simpler than for the **L(i)**s, **i = 0,1.** In fact for the **G(i)**s the problem can be solved simply employing adequate linear interpolations with the only possible difficulty coming from the fact of $\mathbf{U}_{out}$ being a closed curve as it will be seen in section 6.

If **S(0) = S(1)= S**, once **G(w), w** $\in$ **[0,1]** is found, **L'(w)** can be obtained as the set of points in **U** whose coordinates in **S** are in **G(w).**. Otherwise, we need to transform **S**(0) into **S**(1)(Problem **P1.4**) simultaneously with transforming **G(0)**into **G**(1) and **L'(w)** will be obtained as above but in relation to a variable system **S(w)**. As **L'(w)**is not in general a polygon it is replaced by one, **L(w)**, approximating it. This approximation has to be obtained in such a way that the

function ($\mathbf{w}\in[0,1]\rightarrow\mathbf{L(w)}$) is continuous.
The remainder of this article is dedicated to show how to solve all the five subproblems defined above.

4. P1.1 - GETING CONVEXERS.

First of all we find for **i=0,1** a triangulation($\mathbf{T}_i$)of **U** whose **n** vertices are those of $\mathbf{L}_i$, $\mathbf{U}_{in}$ and $\mathbf{U}_{out}$ and which is constrained by the edges of the three polygonal lines. This can be done in **O(n)** time by using Chazelle's algorithm. For sake of simplicity , we will suppose that no triangle of $\mathbf{T}_i$ has all three vertices on $\mathbf{U}_{in}$ or $\mathbf{U}_{out}$, what always can be made. The next step is to find a convexer($\mathbf{D}'_i$) for $\mathbf{L}_i$ defined on $\mathbf{T}_i$ using the procedure **Convexer** given below. Finally we get convexers for each $\mathbf{L}_i$ defined on the same triangulation through the following steps:

i) Get **S**, the subdivision of **U** whose faces are the intersections of $\mathbf{T}_0$ and $\mathbf{T}_1$ triangles.

ii) Obtain a triangulation **T** of **U**, by interpolating every face of **S**.

iii)For **i=0,1** and any triangle **t** of **T** define $\mathbf{D}_i(\mathbf{t})$ = $\mathbf{D}'_i(\mathbf{t}_i)$, where $\mathbf{t}_i$ is the triangle of $\mathbf{T}_i$ containing **t.**

The process above can generate very ill conditioned triangles what can require a further treatment. Alternative, more elaborate methods for generating convexers can be used to reduce the ocurrence of this problem. Procedure **Convexer** given at the end of this section, obtains the definition of a convexer for a polygon **L,** supposed to be one of the **L(i)**s, restricted to one of the regions delimited by **L** and one of the **U** border components. This region is identified by the parameter **border.** If **border** is **in**(Alternatively: **out**), **Convexer** generates a Tfd **D** where all vertices of **L** points to a single triangle(are pointed by a single triangle, respect.) in the region inside(outside,respect.) **L** and all vertices of $\mathbf{U}_{in}$ ($\mathbf{U}_{out}$, respect.) are pointed by a single triangle(points to a single triangle, respect.).

Procedure Convexer(border $\in$ {in, out}, **L simple polygon** like the **L(i), i = 0,1** defined above, **T** triangulation; like the **T(i), i = 0,1** also defined above)
/* Consider that the vertices of **L**(Alternatively: $\mathbf{U}_{border}$) are ordered counterclockwisely, in a list. **POL(i), i=0,**..., **|L|-1** (**BORDER(i), i=0,**..., $|\mathbf{U}_{border}|$**-1**,respect.) */

Function Direction($\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$: **2D vectors)** $\triangleq$ unit vector of ($2*\mathbf{v}_1$ - ($\mathbf{v}_2$ + $\mathbf{v}_3$))

{If(border = **in**) **signborder** = 1
else **signborder** = -1;
Find $\mathbf{T}_{border}$ be the set of triangles of **T** which are in the region delimited by **L** and $\mathbf{U}_{border}$;
Find an edge [$\mathbf{v}_0$,$\mathbf{v}'_0$] such that $\mathbf{v}_0 \in \mathbf{L}$ and $\mathbf{v}'_0 \in \mathbf{U}_{border}$;
Reorder (**POL(i)**) and (**BORDER(i)**) to make **POL(0)** = $\mathbf{v}_0$ and **BORDER(0)** = $\mathbf{v}'_0$;
Find $\mathbf{i}_{change}$ = the least **i** such that [$\mathbf{v}_0$,**POL[i]**] is an edge met after **e** when we rotate around $\mathbf{v}_0$ starting at [$\mathbf{v}_0$,**POL[1]**] in the cclock. direction;
For every tringle **t** in $\mathbf{T}_{border}$ do.
If (All three vertices of **t** are in **L**) do:
{Let **POL(i)** ,**POL(j)** and **POL(k)**,
i < **j** <**k** be the vertices of **t**;
If (**i** = **0** and **k** > $\mathbf{i}_{change}$) make **D(t)** be **Direction(POL(k), POL(j),** $\mathbf{v}_0$)
* **signborder** ;
Else make **D(t)** be **signborder*** **Direction(POL(j), POL(i), POL(k))**;}
Else
{Put the vertices of **t** in a list (**v(i),** **i=1,2,3**) in such a way that **[v(1),v(2)]** and **[v(1),v(3)]** do not belong to **L** or $\mathbf{U}_{border}$ and the angle **<v[2],v[1],v[3]>** is positive. Let $\mathbf{t}_i$, be the other triangle adjacent to **[v(1),v(i)] ,i=2,3** and make j=1;
While(**v(j)** $\notin$ $\mathbf{t}_2$ or **v(j)**ε $\mathbf{t}_3$) j=j+1;
Let **v'** and **v''**) be the vertices of $\mathbf{t}_2$ different to **v(j)**;
Make $\mathbf{D}(\mathbf{t}_2)$ =**signborder** * Direction(**v(j)**,**v'**,**v''**);
If **v(j)** $\in$ $\mathbf{U}_{border}$ make $\mathbf{D}(\mathbf{t}_2) = -\mathbf{D}(\mathbf{t}_2)$. }}

Knowing the assignment of a triangle **t** and its vertex-type, we use **signborder** and the function **Direction** to generate **D(t)**. The vertex type of **t** is determined in function of **signborder** and the polygonal line where the vertex assigned to **t** is on. A less straightforward task is assigning vertices to triangles. For doing that, **Convexer** divides the set of triangles into two classes according to they have all vertices on **L** or not. Each class is treated differently, as follows.

i) If the three vertices of a triangle **t** are on **L,** then, considering only the side of **L** where **t** is, it has an edge which separates it from $\mathbf{U}_{border}$. Thus, **Convexer** assigns the triangle to the vertex opposite to that edge. Considering that the **L** vertices are ordered counterclockwisely in a list **POL** the identification of the vertex above can be made in function of the **t** vertices position in that list.

ii)The triangles having vertices on $\mathbf{U}_{border}$, form a circular sequence. Suppose we traverse this sequence in the counterclockwise direction. A triangle **t** of that sequence is assigned by **Convexer** to the unique of its vertices, which is still a vertex of the next triangle but does not belong to the one following the next. See **fig.5** below.

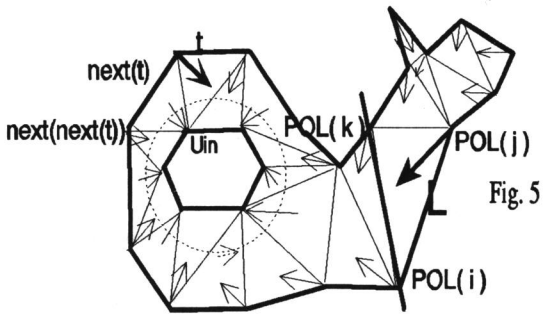

Fig. 5

5. P1.2 - OBTAINING G(i);i=0,1

To describe the process used for solving this problem and **P1.3**, we need at first to introduce the following notation

Given two points **p** and **q** on $\mathbf{U}_{out}$ we will represent the segment traversed when going from **p** to **q** along $\mathbf{U}_{out}$, moving in the **counterclockwise**(Alternatively: **clockwise**) direction by $[\mathbf{p,q,cclock}]_{U_{out}}$($[\mathbf{p,q,clock}]_{U_{out}}$,respect.)

We will also need the four functions below:

i) Dist._U$_{out}$: $(\mathbf{U}_{out})^2$ **x** {**counterclockwise, clockwise**} | **Dist._U**$_{out}$ **(p,q,s)** = the length of $[\mathbf{p,q,s}]_{U_{out}}$

ii) Interp._U$_{out}$: $(\mathbf{U}_{out})^2$ **x [0,1] x** { **counterclockwise, clockwise**} | **Interp._U**$_{out}$**(p,q,w,s)** is the point **r** such that **Dist._U**$_{out}$**(p,r,s)** = **w** × **Dist._U**$_{out}$**(p,q,s)**.

iii)For any $\mathbf{w} \in [0,1]$,δ_w^{in}(Alternatively: δ_w^{out}):$\mathbf{U} \rightarrow R^+$ | $\delta_w^{in}(\mathbf{p})$(Alternatively: $\delta_w^{out}(\mathbf{p})$) is the length of the **D(w)**-trajectory through **p** from **p** to $\mathbf{U}_{in}$($\mathbf{U}_{out}$, respect.)

Back to problem **P1.2** we observe, first of all, that to generate the graphs $\mathbf{G}_i$, **i = 0,1** we only need to describe the functions $\mathbf{s}_i$, $\delta_i^{in}$and δ_i^{out}on $\mathbf{L}_i$, since by definition **G(i)** is the graph of the function **f(i):** $\mathbf{U}_{out} \rightarrow [0,1]$ | $\mathbf{f(i)}(\mathbf{s}_i(\mathbf{p})) = \Delta_i(p) = \delta_i^{in}(p) \;/\; (\delta_i^{in}(\mathbf{p}) + \delta_i^{out}(\mathbf{p}))$.Due to the piecewise linearity of those functions, for having that description we only need to know these functions at the vertices of $\mathbf{L}_i$ and at the points where the trajectories through vertices of $\mathbf{U}_{in}$ and $\mathbf{U}_{out}$ cross $\mathbf{L}_i$. It is reasonable suppose that $\mathbf{U}_{in}$ and $\mathbf{U}_{out}$ are convex polygonal approximations of the circle with a bounded number of edges. Thus, finding these crossing points is sublinear in time. Having found one of these crossing points(**v**) on the open segment $(\mathbf{v}_k^i,\mathbf{v}_{k+1}^i) \subset \mathbf{L}_i$, we make it correspond to a point $\mathbf{v}'$ on the open segment $(\mathbf{v}_k^{1-i},\mathbf{v}_{k+1}^{1-i})$ on $\mathbf{L}_{1-i}$. The restriction of $\mathbf{v}'$ to $(\mathbf{v}_k^{1-i},\mathbf{v}_{k+1}^{1-i})$ is sufficient for the topology of the extreme scenes continue being the same after the inclusion of $(\mathbf{v}, \mathbf{v}')$ in the set of pairs of points to be matched. That inclusion will make simpler finding $\mathbf{G(w)}$, $\mathbf{w} \in (0,1)$. Points like **v** or $\mathbf{v}'$ which are on $\mathbf{L}_i$, together with the original vertices of $\mathbf{L}_i$, are all ordered counterclockwisely and put into a list $(\mathbf{y}_m^i)$, **i = 0,1.**. As for the $\mathbf{L}_i$,$\mathbf{U}_{in}$ and $\mathbf{U}_{out}$, the vertices are already given in counterclockwise order, every $(\mathbf{y}_m^i)$ can be obtained through a merging process. For every $\mathbf{y}_m^i$we find then $\mathbf{s}_i(\mathbf{y}_m^i)$,$\delta_i^{in}(\mathbf{y}_m^i)$ and $\delta_i^{out}(\mathbf{y}_m^i)$. Having these three last lists **G(i)**can be fully determined. To make explicit how this can be done, we observe that **f(i)** is a piecewise rational fuction whose breaking points are included in $(\mathbf{s}_i(y_m^i))$ and such that for any point **q** on the segment $[\mathbf{s}_i(\mathbf{y}_m^i),\mathbf{s}_i(\mathbf{y}_{m+1}^i)$, **cclock**$]_{U_{out}}$ the value of **f(i)(q)** can be expressed as follows: Let λ be **(1 / Dist._U**$_{out}$**(**$\mathbf{s}_i(\mathbf{y}_m^i)$**,** $\mathbf{s}_i(\mathbf{y}_{m+1}^i)$**,cclock.))** × **Dist._U**$_{out}$**(**$\mathbf{s}_i(\mathbf{y}_m^i)$**,q,cclock.)**. Then:
$\mathbf{f}_i(\mathbf{q}) = (\lambda.\delta_i^{in}(\mathbf{y}_{m+1}^i) + (1\text{-}\lambda).\delta_i^{in}(\mathbf{y}_m^i))/(\lambda.(\delta_i^{in}(\mathbf{y}_{m+1}^i) + \delta_i^{out}(\mathbf{y}_{m+1}^i)) + (1\text{-}\lambda).(\delta_i^{in}(\mathbf{y}_m^i) + \delta_i^{out}(\mathbf{y}_m^i))$.

Exploring the parallelism of the **D(i)**-trajectories inside a same triangle finding all $\mathbf{s}_i(\mathbf{y}_m^i)$, $\delta_i^{in}(\mathbf{y}_m^i)$ and $\delta_i^{out}(\mathbf{y}_m^i)$ can be made in linear time without the necessity of entirely traversing the trajectories through every $\mathbf{y}_m^i$.

6. P1.3 - OBTAINING G(w), w ε [0,1]

Obtained the lists, $(\mathbf{s}_i(\mathbf{y}_m^i))$, $(\delta_i^{in}(\mathbf{y}_m^i))$ and $(\delta_i^{out}(\mathbf{y}_m^i))$ introduced in the last section we are ready to determine **G(w)** by means of the following steps:

i)For every pair $(\mathbf{s}_0(\mathbf{y}_m^0),\mathbf{s}_1(\mathbf{y}_m^1))$ choose a direction $\mathbf{dir}_m$(counterclockwise or clockwise) and make $\mathbf{z}_m(\mathbf{w})$ **= Interp_U**$_{out}$**(**$\mathbf{s}_0(\mathbf{y}_m^0)$**,**$\mathbf{s}_1(\mathbf{y}_m^1)$**,w,** $\mathbf{dir}_m$**)**.

ii)Also determine $\forall$ **m** $\widetilde{\delta}_w^{out}(\mathbf{z}_m(\mathbf{w})) = \mathbf{w}.\ \delta_1^{out}(\mathbf{y}_m^1) + (\mathbf{1\text{-}w})\ \delta_0^{out}(\mathbf{y}_m^0)$ and obtain $\widetilde{\delta}_w^{in}(\mathbf{z}_m(\mathbf{w}))$ in an analogous way.

iii) Then obtain **G(w)** from $(\mathbf{z}_m(\mathbf{w}))$, $(\widetilde{\delta}_w^{in}(\mathbf{z}_m(\mathbf{w})))$ and $(\widetilde{\delta}_w^{out}(\mathbf{z}_m(\mathbf{w})))$ in the way described for obtaining **G(i)** from $(\mathbf{s}_i(\mathbf{y}_m^i))$, $(\delta_i^{in}(\mathbf{y}_m^i))$ and $(\delta_i^{out}(\mathbf{y}_m^i))$, respectively, at the end of the last section. Replace also **f(i)** by a function **f(w)** with the same characteristics. The

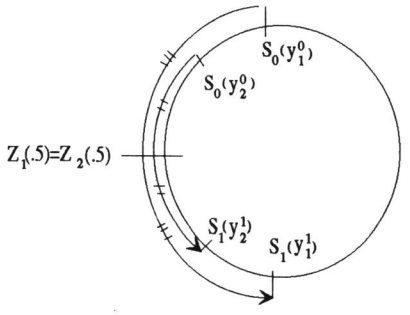

Fig. 6

only possible difficulty concerns to the choice of the directions $\mathbf{dir}_m$ to avoid the situation pictured in figure 6, where due to a wrong choice of $\mathbf{dir}_1$ we will have that $\mathbf{z}_1(.5) = \mathbf{z}_2(.5)$, In situations like that the method used for computing **f(w)** on $\mathbf{U}_{out}$ can issue more than one value for the same point. Due to the

fact that both $(\mathbf{s}_0(\mathbf{y}_m^0))$ and $\mathbf{s}_1(\mathbf{y}_m^1)$ are counterclockwisely ordered, situations like that of the example will not happen if we satisfy the condition below for any pair $(\mathbf{s}_0(y_m^0),\mathbf{s}_1(y_m^1))$:

"If for any **dir**, $[\mathbf{s}_0(\mathbf{y}_m^0),\mathbf{s}_1(\mathbf{y}_m^1),\mathbf{dir}]_{U_{out}} \subseteq [\mathbf{s}_0(\mathbf{y}_0^0),\mathbf{s}_1(\mathbf{y}_0^1),\mathbf{dir}]_{U_{out}}$ then $\mathbf{dir}_m = \mathbf{dir}$ and.$\mathbf{dir}_0 \neq \mathbf{dir}$. Otherwise $\mathbf{dir}_m = \mathbf{dir}_0$."

If the condition above is true for one **m**, all $\mathbf{dir}_m$ will be perfectly determined. If it is always false, all $\mathbf{dir}_m$ should be equal and its determination can be made through a geometrical criterium like:

"If $\sum(\mathbf{Dist.\_U}_{out}(\mathbf{s}_0(\mathbf{y}_m^0),\mathbf{s}_1(\mathbf{y}_m^1),$ **cclock.**)) $\geq$ $\sum(\mathbf{Dist.\_U}_{out}(\mathbf{s}_0(\mathbf{y}_m^0),\mathbf{s}_1(\mathbf{y}_m^1),$ **clock.**)) then for all **m**, $\mathbf{dir}_m$ = clockwise else $\mathbf{dir}_m$ = counterclockwise. "

7. P1.4 - TRANSFORMING TFDs.

A continuous transformation of one Admissible Tfd **D(0)** into another **D(1)** which mantains that property for the intermediate Tfds generated, will be called in this section an **Admissible Transformation**. During an Admissible Transformation, the only possible way of the Assignment of a triangle **t** changing is the following(See fig. 7): 1)At a time $\mathbf{w} \in [0,1]$ of the transfor-

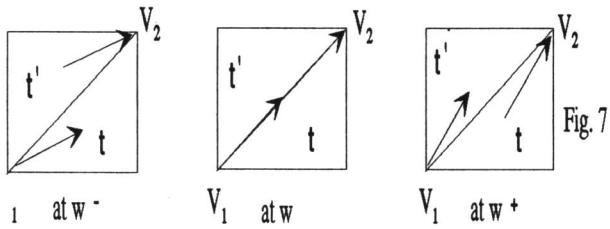

Fig. 7

mation, for an edge $(\mathbf{e}=[\mathbf{v}_1,\mathbf{v}_2])$ of **t** which separates it from an adjacent triangle $\mathbf{t}'$, we have that: $\mathbf{D(w)(t)}$ are $\mathbf{D(w)(t')}$ equal and parallel to **e.**

2) A little before at $\mathbf{w}^-$, **t** is assigned to one of the $\mathbf{v}_i s$,(say: $\mathbf{v}_1$)and $\mathbf{t}'$ is assigned to the other($\mathbf{v}_2$). **t** and $\mathbf{t}'$ have different vertex-types and $\mathbf{D(w^-(t))}$ and $\mathbf{D(w^-)(t')}$ are almost equal to an unit vector($\mathbf{u}_e$) parallel to **e.** Both are on the same side in relation to that vector.

3) A little after at $\mathbf{w}^+$,**t** is assigned to $\mathbf{v}_2$ and $\mathbf{t}'$ to $\mathbf{v}_1$. With respect to the situation at $\mathbf{w}^-$, we can notice that **t** and $\mathbf{t}'$ have exchanged both their assignments and vertex-types. Also, $\mathbf{D(w^+(t))}$ and $\mathbf{D(w^+)(t')}$ are now, both on the other side of $\mathbf{u}_e$.

We will call an exchange of assignments occuring in the conditions above an **event** and the edge where it happens, the **event edge.**

Now, let Π be an Admissible Transformation and $(\mathbf{e}_n)$ be the sequence of edges where the events determined by Π occur, ordered according to the time of the event to which they are related. Knowing $(\mathbf{e}_n)$, although it is not necessarily possible to recover Π, it is always possible to determine another Admissible Tranformation . Hence we can concentrate on finding at first an appropriate sequence of event edges and then obtain from it an Admissible Transformation. This last step is a simple interpolation problem which will not be seen here. The **Algorithm A** given below, constructs that appropriate sequence of edges, if, for both $\mathbf{i} = \mathbf{0,1}$, there is a **D(i)**-increasing sequence of triangles from one adjacent to $\mathbf{U}_{out}$ to one adjacent to $\mathbf{U}_{in}$.This can always be made true if the **D(i)** have no cicles, by refining the triangulation along a selected trajectory of each **D(i).** If this condition is true then every set of triangles has an element which is **minimal** for $>_{D(i)}$ in the set.

Algorithm A labels triangles **Solved or Unsolved.** The label **Solved,** means that the current Assignment and vertex-type of the triangle are already those of **D(1)**. The Main Loop of Algorithm A is very simple: The algorithm chooses a triangle which is minimal_for $>_{D(1)}$ among the **Unsolved** ones and by means of a sequence of events determined by the recursive routine **Events_finding,** it makes the assignment and vertex-type of that triangle be those of **D(1).**Then the triangle is labeled **Solved** and the cycle is repeated until all triangles are **Solved.** Choosing to solve a minimal for $>_{D(1)}$ **Unsolved** triangle we manage to restrict the action of the routine **Events_finding** to the set of **Unsolved** triangles. Thus, if a triangle is made **Solved,** its Assignment and vertex-type will not be changed anymore.

ALGORITHM A:

Input: Two Admissible **Tfds D(i), i = 0,1**, such that there is a **D(i)**-increasing sequence of triangles from $\mathbf{U}_{out}$ to $\mathbf{U}_{in}$.

D(i)-Assignments **A(i), i = 0,1.**

Main Procedure

{Label all triangles of **T Unsolved** ;

For all $\mathbf{t} \in \mathbf{T}$ Make $\alpha(\mathbf{t}) = \mathbf{A(0)(t)}$ and $\mathbf{v\_t(t)} =$ **D(0)**-vertex-type**(t);**

While there are **Unsolved** triangles do:

{Pick a **Unsolved D(1)**-minimal triangle τ;

If $\alpha(\tau) \neq \mathbf{A(1)}(\tau)$ do

{Make e = $[\mathbf{A}(\tau),\mathbf{A(1)}(\tau)]$ and execute **Events_Finding(e,τ)**}

Label τ **Solved.**}}

The routine **Events_finding** giving below makes use of the following operator: Consider two adjacent triangles **t** and $\mathbf{t}'$ and let **e** be their common edge. The **Reflection by e** of another edge $\mathbf{e}'$ of $\mathbf{t}'$(Alternatively: **t**) is the edge of $\mathbf{t}(\mathbf{t}'$, respect.) having a vertex in common with $\mathbf{e}'$ which is not **e** .

Procedure Events_Finding(τ: triangle, e: edge of τ)}
{Make τ' be the triangle separated from τ by **e** and **e'** be the edge of τ' opposite to $\alpha(\tau')$.
Make $\mathbf{L}(\tau',\mathbf{i})$, **i=0,1** be an ordered pair containing the edges of τ' different from **e** and | if $\mathbf{e}' \neq \mathbf{e}$ then $\mathbf{L}(\tau',2) = \mathbf{e}'$.
For **i = 1 to 2** do
If (**io-type**(τ',**L**(τ', i)) $\neq$ **io-type**(τ,**Reflection by e** of **L**(τ', i))
Events_Finding($\mathbf{e}', \tau'$);
Exchange the values of $\alpha(\tau)$ and $\alpha(\tau')$ and of $\mathbf{v\_t}(\tau)$ and $\mathbf{v\_t}(\tau')$;
Append **e** to the **List_of_Events_Edges** and Return}.

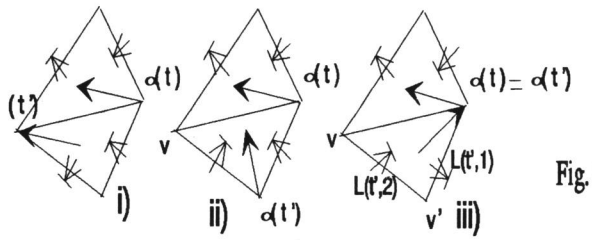

Fig. 8

Refer simply by **Condition** to the only condition checked by **Events_Finding.**. The text below is related to the fig. 8

Given a triangle **t** and an edge **e** adjacent to $\alpha(\mathbf{t})$, **Events_Finding** generates a sequence of events such that performing the Triangles Assignment Exchanges associated to them we make $\alpha(\mathbf{t})$ be the other vertex adjacent to **e**. Hence, after applying **Events_Finding** to (τ, [$\alpha(\tau)$,$\mathbf{A(1)}(\tau)$]) we will have: $\alpha(\tau) = \mathbf{A(1)}(\tau)$. In the last Assignment Exchange performed by **Events_Finding**(τ,**e** = [$\alpha(\tau)$,**v**]) the triangles taking part should be τ and τ', the other triangle adjacent to **e.** For that be possible we should have $\alpha(\tau')$=**v.** This will be the case iff **Condition** will be false for both $\mathbf{L}(\tau',\mathbf{i})$. In this case **Events_Finding** outputs **e** and returns(See fig.8.i). If $\alpha(\tau') \neq \mathbf{v}$ consider at first, the case where **Condition** is true for only one $\mathbf{L}(\tau',\mathbf{i}) = \mathbf{e}'$. This will only happen iff $\alpha(\tau')$ is the vertex of τ' opposite to **e** and $\mathbf{e}' = [\mathbf{v},\alpha(\tau')]$. In this case we can make $\alpha(\tau')$ become **v** by calling **Events_Finding**(τ', e')(See fig.8-ii). If **Condition** is true for both $\mathbf{L}(\tau',\mathbf{i})$ then $\alpha(\tau') = \alpha(\tau)$ and $\mathbf{L}(\tau',\mathbf{1})$(Alternatively $\mathbf{L}(\tau',\mathbf{2})$) is the edge of τ' adjacent to $\alpha(\tau)$(**v**,respect.) and opposite to $\mathbf{v}(\alpha(\tau)$, respect) In this case we cannot obtain $\alpha(\tau') = \mathbf{v}$, by applying **Events_Finding** to τ' and a single $\mathbf{L}(\tau',\mathbf{i})$. We should instead apply it to (τ',$\mathbf{L}(\tau',\mathbf{1})$) to obtain $\alpha(\tau') = \mathbf{v}'$, the vertex of τ' opposite to **e.** As $\alpha(\tau')$ has become adjacent to $\mathbf{L}(\tau',\mathbf{2})$ and we can now apply **Events_Finding**(τ', $\mathbf{L}(\tau',\mathbf{2})$) to finally make $\alpha(\tau') = \mathbf{v}$.(See fig. 8-iii).

Among several results relative to the convergence and complexity of **Algorithm A** and to the topology of its output, the most important are the following:

i)Call **Stack** to the stack of parameters of calls to **Events_Finding** not yet completed at a moment of Algorithm A execution. Then **Stack** has no repeated edges or triangles and never contains a **Solved** triangle nor an edge of one of them. From that we can derive the finite termination of Algorithm **A.**

ii) Let **D** be any admissible **Tfd** having $\alpha(.)$ as a **D**-Assignment and the vertex-type of the triangles given by **v_t(.)** Then **D** has no cycles..

iii) If $\mathbf{n} = |\mathbf{T}|$ and $\lambda =$
$\max_{t\varepsilon T}$ { minimum number of **T** triangles that a path from **t** to the **U**-border has to cross)}

then **Algorithm A** is $\mathbf{O(n\lambda)}$ in time and $\mathbf{O(n)}$ in space.

8. P1.5 - CONSTRUCTING L(w).

Recall that $\mathbf{H(w)}$: $\mathbf{U} \rightarrow \mathbf{U}_{out} \times [0,1]$ is the function which takes a point $\mathbf{p} \in \mathbf{U}$ into $(\mathbf{s}_{D(w)}(\mathbf{p}), \Delta_{D(w)}(\mathbf{p}) = \delta_w^{out}(\mathbf{p}) / \delta_w^{out}(\mathbf{p}) + \delta_w^{in}(\mathbf{p})))$. $\mathbf{s}_{D(w)}$, δ_w^{out} and δ_w^{in} are all linear functions on each componnent of a partition **P** of **U**. Those components can be:

i) trapezia delimited by the intersections of two **D(w)**-trajectories through **T**-vertices with a triangle **t** of **T** and by two edges of that triangle .

ii) triangles, when one of the two **D(w)**-trajectories mentioned above is the one through a vertex of **t**.

We can find a simple polygonal line **L(w)** which is an approximation of $\mathbf{H(w)}^{-1}(\mathbf{G(w)})$ and satisfies the properties that ($\mathbf{w} \in [0,1] \rightarrow \mathbf{L(w)}$) is a continuous function and its limit when $\mathbf{w} \rightarrow \mathbf{i}$ is $\mathbf{L(i)}$, $\mathbf{i} = \mathbf{0,1}$.

At first we determine precisely the intersection of $\mathbf{H(w)}^{-1}(\mathbf{G(w)}$ with every **D(w)**-trajectory through a vertex of **T**, what can be made through the following steps.

•For every vertex **v** of the triangulation **T** starting at **v** and moving always in the direction of **-D(w)** (Alternatively: in the direction of **D(w)**) until $\mathbf{U}_{out}$($\mathbf{U}_{in}$, respect.) is reached, determine $\mathbf{s}_w(\mathbf{v})$ and accumulating the length of the linear segments traversed obtain $\delta_w^{out}(\mathbf{v})$($\delta_w^{in}$ (**v**)respect). Make δ_w^{total} $(\mathbf{s}_w(\mathbf{v})) = \delta_w^{out}(\mathbf{v}) + \delta_w^{in}(\mathbf{v})$.

•Order the set { $\mathbf{s}_w(\mathbf{v})$ | $\mathbf{v} \in \mathbf{T}$} to obtain a list ($\mathbf{z}'_n$), where these points on $\mathbf{U}_{out}$ appear in counterclockwise order.

•For every $\mathbf{z}'_n$ let **j** be such that $[\mathbf{z}_j(\mathbf{w}),\mathbf{z}_{j+1}(\mathbf{w}),\mathbf{cclock}]_{U_{out}}$ contains $\mathbf{z}'_n(\mathbf{w})$ and $\lambda \in [0,1]$ be (**Dist._U**$_{out}(\mathbf{z}_j(\mathbf{w}),\mathbf{s}_w(\mathbf{v}),$ **cclock**) / **Dist._U**$_{out}(\mathbf{z}_j(\mathbf{w}),\mathbf{z}_{j+1}(\mathbf{w}),\mathbf{cclock})$. The sequence ($\mathbf{z}_m(\mathbf{w})$) has been obtained when we solved problem

P1.3.

Obtain $\widetilde{\delta}_w^{out}(\mathbf{z}'_n(\mathbf{w})) = \lambda\ \widetilde{\delta}_w^{out}(\mathbf{z}_{j+1}(\mathbf{w})) + (1-\lambda)\ \widetilde{\delta}_w^{out}(\mathbf{z}_j(\mathbf{w}))$ and $\widetilde{\delta}_w^{in}(\mathbf{z}'_n(\mathbf{w}))$ in an anlogous way. Then make $\Delta_w(\mathbf{v}_{w,n}) = \widetilde{\delta}_w^{out}(\mathbf{z}'_n(\mathbf{w}))\ /\ (\widetilde{\delta}_w^{in}\mathbf{z}'_n(\mathbf{w})) + \widetilde{\delta}_w^{out}\mathbf{z}'_n(\mathbf{w})))$ and starting at $\mathbf{z}'_n(\mathbf{w})$ and moving along the $\mathbf{D(w)}$-trajectory through $\mathbf{z}'_n(\mathbf{w})$ a distance of $\Delta_w(v_{w,n}).\delta_w^{total}\mathbf{z}'_n(\mathbf{w}))$ determine $\mathbf{v}_{w,n}$. See fig. 9.

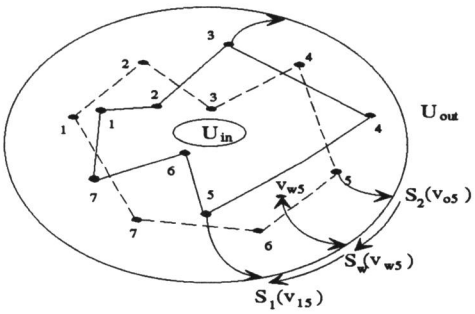

Fig.9

The list $(\mathbf{v}_{w,n})$ will contain the intersections of $\mathbf{H(w)}^{-1}(\mathbf{G(w)})$ with the $\mathbf{D(w)}$-trajectories through vertices of $\mathbf{T}$ ordered counterclockwisely. Now, we will determine a polygonal line $\mathbf{L}_n(\mathbf{w})$ from $\mathbf{v}_{w(n-1)}$ to $\mathbf{v}_{w,n}$, completely contained in the region delimited by the $\mathbf{D(w)}$-trajectories through these points which does not contain any other $\mathbf{v}_{w,j}$, $\mathbf{j \neq n\text{-}1,n}$. Hence $\mathbf{L}_n$ (w) does not cross $\mathbf{L}_j$ (w), $\mathbf{j \neq n}$. The determination of $\mathbf{L}_n(\mathbf{w})$ can be made by the following procedure:

Procedure Finding_Ln(w)

Let $\mathbf{R}_{w,n-1}$ be the $\mathbf{P}$ set containing $\mathbf{v}_{w(n-1)}$ an, d having vertices on the $\mathbf{D(w)}$-trajectory through $\mathbf{v}_{w,n}$. Make $\mathbf{R} = \mathbf{R}_{w,n-1}$, $\mathbf{L}_n(w)(\mathbf{0}) = \mathbf{v} = \mathbf{v}_{w(n-1)}$, $\mathbf{k = 0}$ and execute:

While$(\mathbf{L}_n(\mathbf{w})(\mathbf{k}) \neq \mathbf{v}_{w,n})$
{$\mathbf{k = k + 1}$;
If $\mathbf{R}$ has an edge $\mathbf{e} = [\mathbf{q}_{n-1}, \mathbf{q}_n]$ such that $\mathbf{q}_j \in \mathbf{D(w)}$-trajectory through $\mathbf{v}_{w,j}$, $\mathbf{j = n\text{-}1,\ n}$ and $(\Delta_w(\mathbf{v}) - \Delta_w(\mathbf{q}_{n-1}).\ (\Delta_w(\mathbf{v}_{w,n}) - \Delta_w(\mathbf{q}_n)) < \mathbf{0}$ then make:
{$\lambda = (\Delta_w(\mathbf{v}_{w,n}) - \Delta_w(\mathbf{q}_n))\ /\ (\Delta_w(\mathbf{v}) - \Delta_w(\mathbf{q}_{n-1}))\ -\ (\Delta_w(\mathbf{v}_{w,n}) - \Delta_w(\mathbf{q}_n))$;
$\mathbf{L}_n(\mathbf{w})(\mathbf{k}) = \lambda\mathbf{q}_{n-1} + ((1-\lambda))\ \mathbf{q}_n$,
$\mathbf{v} = \mathbf{q}_{n-1}$ and $\mathbf{R}$ be the other $\mathbf{P}$ set adjacent to $\mathbf{e}$.}}

Making $\mathbf{L(w)}$ be the union of all $\mathbf{L}_n$ (w) we will have all the properties required. The overall complexity of the whole process above can, of course, be $\mathbf{O(|T|^2)}$. Figures 10, 11 and 12 below shows a simple application of the methodology described in this article.

9.REFERENCES

[OLIV93] A.A.F. Oliveira and A.J.A da Cruz; Transformação de Curvas Poligonais: Um algoritmo Linear a partir de uma Triangulação;Anais do Sibgrapi VI,267-276(1993).

[ROS91] J.Rossignac and A.Kaul; Solid Interpolating Deformations: Construction and Animation of PIPs, Proc.Eurographics 91,493-505(1991).

[SED92] T.Sederberg and E.Greewood; A Physically Based Approach to Shape Blending; Computer Graphics 26(2), 25-34(1992).

[SED93] T.Sederberg, P. Gao, G. Wang, and H. Mu; 2-D Shape Blending: An Intrinsic Solution to the Vertex Path Problem; Computer Graphics Proceedings, 1993.

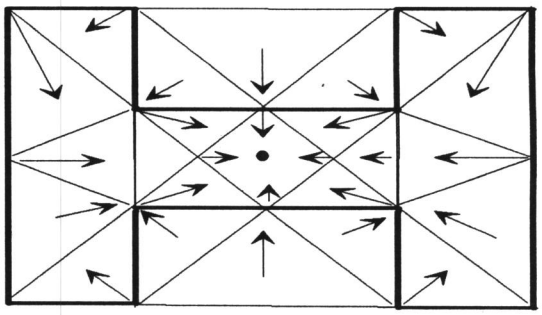

fig.10

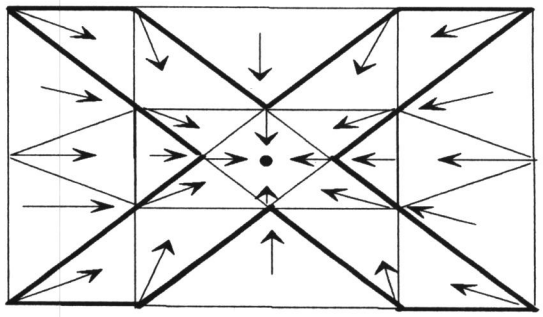

fig.11

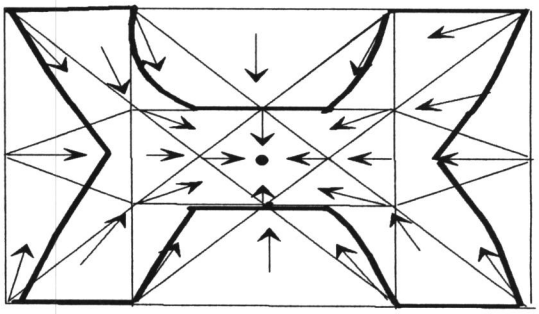

fig.12