3D MORPHISM & IMPLICIT SURFACES

ROMAIN BALP AND CHARLEY PAULUS

Abstract. The purpose of this paper is to present a framework based on implicit surfaces that allows to visualize dynamic shapes, and see how implicit surfaces can be used to perform shape transformation, also called morphism, between shapes of different geometry and even different topology.

1. Introduction

In Computer Vision and Computer Graphics, 3D objects are often modeled as explicit surfaces such as triangulated meshes or parametric surfaces. Such representation are easy to manipulate and intuitive. However, these representations are not necessarily efficient for fitting surfaces or even for shape transformation and automated modeling where the shape can encounter extreme geometrical and topological changes.
Implicit surfaces are well-suited for simulating physically based deformations and for modeling smooth objects as well as for reconstructing surfaces from noisy data sets. They have not gained wide acceptance, in particular because they are quite difficult to render in comparison with explicit surface representations.
This work is divided in two main parts: we will desribe the implicit surface visualisation framework, and the implicit surface schemes used to perform shape transformation.

1.1. Implicit Surface.

Basically, a general implicit surface S_f is defined as the zero-set of a scalar *defining function* $f : \mathbb{R}^3 \mapsto \mathbb{R}$ with

$$f(\mathbf{x}) = 0 \tag{1.1}$$

The implicit surface is defined as $S_f = \{\mathbf{x} \in \mathbb{R}^3 | f(\mathbf{x}) = 0\}$ also called *isosurface* at the value 0. Based on a common convention, we define the interior of the surface S_f, as the domain where f is positive, i.e $f(\mathbf{x}) > 0$. Accordingly, the exterior of the surface is the domain where f is negative, i.e $f(\mathbf{x}) < 0$.

Representing objects with a defining function has a number of advantages, in particular implicit surfaces

Date: January 1, 2007.
Romain Balp *student number* 2005280032.
Charley Paulus *student number* 2006280039.

- have a mesh-independent representation, but the mesh can be generated when needed,
- are guarenteed to be manifold surfaces with no intersections,
- are topologically flexible

These properties are very intersting for morphing when the shapes have different topology or geometry.
There is a lot of representation for implicit surfaces: radial basis function, Moving Least Square... In this paper we have used analytical definition for very simple shapes, and variational representation using radial basis functions.

1.2. **Iso-Surface tracking.** One of the main issues of Implicit Surfaces is how to visualise them. Many approaches have been proposed such as marching cubes, QSplat bounding sphere algorithm, or particle sampling. The method we use here is based on the particle sampling scheme of Witkin and Heckbert [**?**] and the reconstruction algorithm of Chaine [3] based on Delaunay triangulation.
We suppose we already have an implicit surfaces S_f defined by an implicit functions f (this will be discussed later in section 4).We proceed as follow:

- sample the surface S_f using particle sampling,
- compute the Delaunay triangulation of the particles,
- reconstruct a mesh approximating S_f with Chaine's reconstruction algorithm.

2. Particle Sampling

In 1994, Witkin and Heckbert proposed a method to control and visualize shapes represented by parametrized implicit functions [1]. Their idea was to attach some particles to the surface and then make them spread out to reach uniform density using a local repulsion scheme based on an energy minimization.
To attach moving particles to the surface, we derive a basic constraint on particle and surface velocities that establishes, then maintains contact as the system evolves over time.

$$f'(\mathbf{x}, t) = f_{\mathbf{x}}(\mathbf{x}, t) \cdot \dot{\mathbf{x}} \tag{2.1}$$

In practice, we might not have valid initial conditions, and numerical integration errors would cause drift over time. We cure these problems using a feedback term

$$f'(\mathbf{x}, t) = f_{\mathbf{x}}(\mathbf{x}, t) + \Phi f(x, t) \tag{2.2}$$

We then solve for particle velocities

$$\dot{\mathbf{x}} = \mathbf{x} - \frac{f_{\mathbf{x}}(\mathbf{x}, t) \cdot \mathbf{x} + \Phi \cdot f(\mathbf{x}, t)}{f_{\mathbf{x}}(\mathbf{x}, t) \cdot \mathbf{x} \cdot f_{\mathbf{x}}(\mathbf{x}, t) \cdot \mathbf{x}} f(\mathbf{x}, t) \tag{2.3}$$

This ensures the particles are attached to the surface during its evolution. Now we want to be able to control their positions over the surface, i.e. have a good sampling.
Witkin and Heckbert made particles to spread out to uniform density by local repulsion, relying on the finiteness of the surface to limit growth, by using a simple repulsion scheme. The energy of particle i due to particle j is defined as

$$E_{ij} = \alpha e^{-\frac{\|r_{ij}\|^2}{\sigma_i^2}} \tag{2.4}$$

where α and σ_i are tuning parameters. For a set of n particles, the energy of the particle i is therefore

$$E_i = \sum_{i=1}^{n} E_{ij} \tag{2.5}$$

Ultimately, we would like to reach the global minimum of each Ei by varying the particle positions on the surface. Finding the global minimum is impractical, but we can find a local minimum by gradient descent : each particle moves in the direction that reduces its energy fastest. We therefore choose each particles desired velocity to be negatively proportional to the gradient of energy with respect to its position

$$\dot{\mathbf{x}}_i = -\sigma_i^2 E_{\mathbf{x}_i}^i = \sum_{j=1}^{n} \mathbf{r}_{ij} E_{ij} \tag{2.6}$$

This simple repulsion ensures to have an uniform distribution of the particles on the surface, but it can be very slow to compute. If the energy is well designed, only the closest neighbors of a point $\mathbf{x}_i$ will contribute to its repulsion factor. To determine the closest neighbors, we use the Delaunay triangulation $\mathscr{T}_D$: we assume that the closest neighbors of $\mathbf{x}_i$, geometricaly speaking, are also neighbors of $\mathbf{x}_i$ in the triangulation. This way we only process the neighbors in the triangulation. Still we may have some strange results, as all the neighbors in the triangulation are not geometricaly close (we can think of the awkward example of a sphere), to overcome this, we compute the repulsion of the point $\mathbf{x}_i$ with its 6 closest neighbors in the triangulation $N_{\mathbf{x}_i}^{\mathscr{T}_D}$

$$\dot{\mathbf{x}} = \sum_{\mathbf{x}_j \in N_{\mathbf{x}_i}^{\mathscr{T}_D}}^{n} \mathbf{r}_{ij} E_{ij} \tag{2.7}$$

Using these 6 neighbors, we can also compute automaticaly the value of σ_i in the Gaussian repulsion. We set it to the distance to the nearest neighbor $\mathbf{x}_k$ over $\sqrt{2}$

$$\sigma_i = \frac{r_{ik}}{\sqrt{2}} \tag{2.8}$$

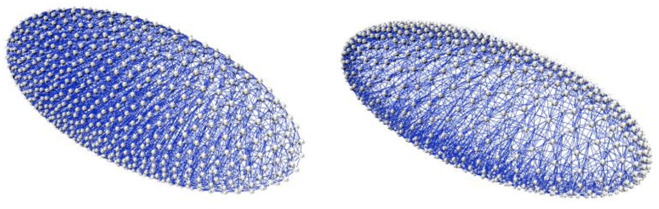

FIGURE 1. Uniform and Curvature adaptive sampling

The local Gaussian described here works very well for uniform sampling, but most of the time we would like to have a curvature dependant sampling. The best way to achieve this is to compute a Riemann metric based on local curvature, but this also results in more computational efforts.
We have been working on an other repulsion scheme, which seems to give some good results. The main idea is to compute the repulsion in the tangent plane of the point $\mathbf{x}_i$: we project the 6 nearest neighbors in the tangent plane, then, we minimize a concave energy using a Levenberg-Marquardt algorithm in 2 dimension. This ensure fast convergence of the particules and a pretty good sampling which can be easily adapted to curvature.

3. Delaunay Triangulation

One of the main caracteristic of our method is that it's based on Delaunay triangulation. We use it for the particle sampling, and we use it for the surface reconstruction. Hence we have a perfect triangulation, the problem is that computing Delaunay is very expensive in terms of computational operations. Each time a particule moves the triangulation may change, and when it should be updated, how do we update it ?
The simple and naive way, is to compute Delaunay from scratch each time one of the Delaunay certificate becomes not valid in the triangulation. A more reasonable approach is to try to recompute the triangulation only where it is needed.
Basically there are two steps in the Delaunay update: first we check that the triangulation is embedded, that is we check no cells have changed orientation; second we verify the in sphere property. To update the triangulation, there are three possibilities:

- remove all the points that violate either the embedded or the in sphere property, we retriangulate and re-insert them,
- remove and re-insert the points one by one,
- flip the cells concerned.

In their survey on Delaunay triangulations update methods Guibas and Russel [2] concluded that flipping was the fastest method, even if in

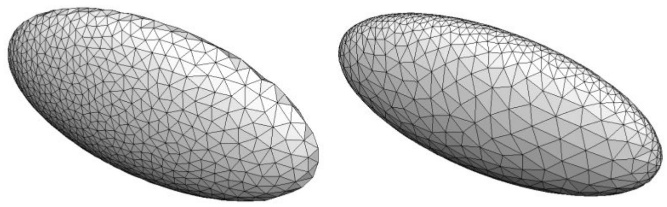

FIGURE 2. Elipsoid reconstructed from uniform and curvature adaptive particel sampling

3D we may find some case where a cell is not flippable. Unfortunately, the flip procedure is not easy to implement. For now we use the second method, which proves to be faster than the first.
We are currently able to work interractively on surfaces with 300 points. The Delaunay triangulation is the bottle-neck of our method, but we believe we can optimize it, first with the flip, and second, by updating only when the points have moved a lot in their neighborhood.

4. Surface Reconstruction

Given a set of points that lie on or near an object surface, we consider the problem of computing a piecewise linear approximation of this unknown surface. We present here the algorithm of Chaine [3] that we use for our framework.

4.1. **Convection Model.** To measure the distance between a surface Γ and a set of points Σ, Zhao, Osher and Fedkiw defined an energy function which is a weighted area of the surface (the farthest an element of surface is to its closest point, the bigger the weight is).

$$E(\Gamma) = \left(\int_{x \in \Gamma} d^p(x) ds \right)^{1/p} \quad 1 \leq p \leq \infty$$

where $d(x)$ is the distance from point x to its closest point in Σ. The goal is to find a surface that minimizes this energy (global distance). They propose a variational equation that runs a gradient descent from an acceptable initial enclosing of the surface towards the interior.

4.2. **Convection and Computational Geometry.**

4.2.1. *Geometric Properties.*
The result of the convection model proposed by Zhao, Osher and Fedkiw [5] is a triangulated oriented surface that is included into the 3D Delaunay triangulation of the set of points. In 2D, we define a half-edge $\widehat{P_1 P_2}$ as the oriented edge from P_1 to P_2. This half-edge supports the diametral half-disk located on its left side. These definitions can be easily extended to 3D. *Given a set Σ of points, the result of the convection of a bounding curve towards Σ is a closed oriented pseudo-curve composed of a set of half-edges. These halfedges are oriented towards*

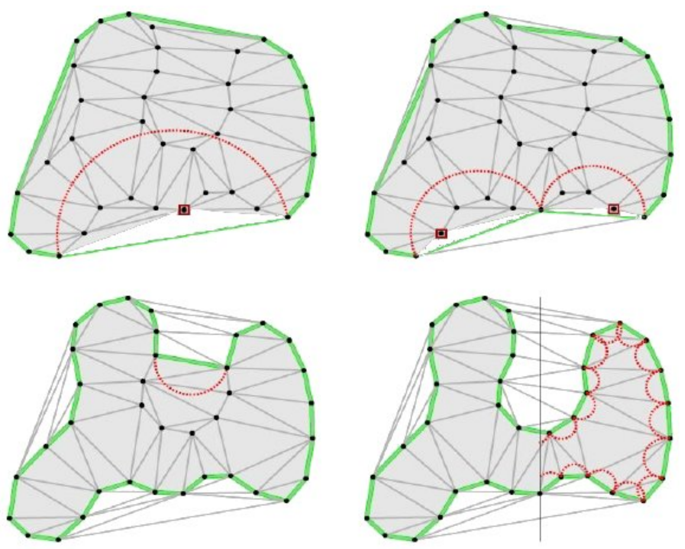
FIGURE 3. 2D Convection towards a 2D set of points

the interior of the curve, they are supported by edges of the 2D Delaunay triangulation of Σ *and they all meet the Gabriel property. The term pseudo-curve is used to mean that different parts of the evolving curve can locally meet common geometric information.* C_{ij} denotes the

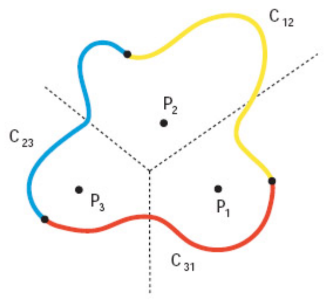

FIGURE 4. Initial bouding

piece of curve intersecting with Voronoi cells of P_i and P_j. We consider the result of the convection of C_{ij} to the surface Σ. The Voronoi cells of P_i and P_j are adjacent, so P_iP_j is included into the Delaunay triangulation of the points. In the gradient descent, the points inside of the Voronoi cell of P_i (resp. P_j) are attracted towards P_i (resp. P_j). The equidistant points are attracted towards the middle of P_iP_j. • If the half-edge $\widehat{P_1P_2}$ does not meet Gabriel property : The half-disk supported by $\widehat{P_1P_2}$ contains at least another point of the set. Let's note P_k such a point. P_k is hidden by $\widehat{P_1P_2}$. P_k is connected to P_i and P_j in the Delaunay triangulation and lies in the half-plane delimited by $\widehat{P_1P_2}$. The points of C_{ij} meet the Voronoi cell of P_k on their way to the attractor. At that time, C_{ij} can be split into C_{ik} and C_{kj}. • If the

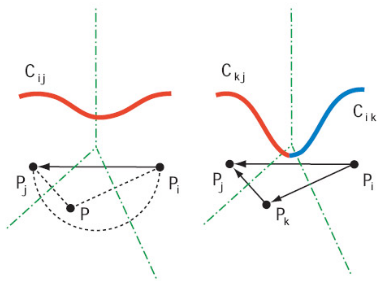

FIGURE 5. Split the piece of curve when intersecting a third Voronoi cell

half-edge $\widehat{P_1P_2}$ does not meet Gabriel property : The piece of curve C_{ij} does not meet a third Voronoi cell. In this case the result of convection of C_{ij} is $\widehat{P_1P_2}$.

4.2.2. *Convection result through a geometric algorithm.*
A geometric algorithm can be derived from the previous section. It will make the surface shrink into a 3D Delaunay triangulation of the points until it fits Σ and every half-facet meets Gabriel property.

```
For every facet f,
   if it does not meet Gabriel property
      delete f
   if it does not cause local intersection for f
   and its coupled half-facet
      replace f by the 3 half-facets hidden by it
   else
      delete the coupled half-facet of f
```

4.3. **Oriented nature of the aperture condition.**

A half-edge (or a half facet in general) is shrunk if its interior half-disk (half-ball) is empty. But the algorithm does not look at the entire ball. It only takes into account the internal skeleton of the surface. In 2D, it can be shown that this limitated aperture condition does not change the result of the convection.

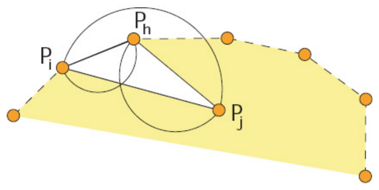

FIGURE 6. 2D Convection Property

A discovered half-edge $\widehat{P_1P_2}$ must have its coupled half-facet meeting the Gabriel property, unless it could not have been included into the evolving surface.

4.4. **Extension of the convection process.** In presence of cavities or pockets exceeding the size of a certain cylinder, the convection process can be stuck. If P is a point blocking a cavity, the distance from P and its neighbors (inside and outside the cavity) should be small compared to the size of the blocking facet. But it is not the case in presence of a cavity. So it is necessary to assume that the density in the points set is characteristic of the density of the points on the surface. The aperture condition is improved : a half-facet is now opened if its size is not coherent with the local density of the sampling.

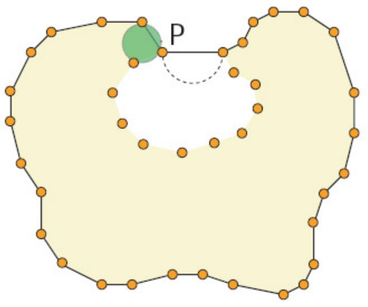

FIGURE 7. Cavity

5. Shape Transformation

5.1. **Simple Morphing.** It is very easy to perform morphing between two shapes S_f and S_g using implicit surfaces. We can simply blend their defining functions f and g with a C^0 continuous blending function $w : \mathbb{R} \mapsto [0,1]$, the morphing shape at the time t is simply defined by the function

$$(1 - w(t))f(\mathbf{x}) + w(t)g(\mathbf{x}) \tag{5.1}$$

We can this way perform a morphing between a sphere and a torus in a realistic way (see figure 8). But the problem with this formulation, is that the user has no control on the transformation, and the value of the scalar field f and g can lead to wierd results if the shapes are really to different.

5.2. **Variational Interpolation.** To transform a shape into an other, Turk and O'Brian [4] proposed to use variational interpolation. Assuming those two shapes are designed in N dimension, we first put them an $N + 1$ dimension space. If we call t the suplementary dimension, we put the first shape in the hyperplane $t = 0$ and the second shape in the hyperplane $t = T$. Then we compute a smooth function that

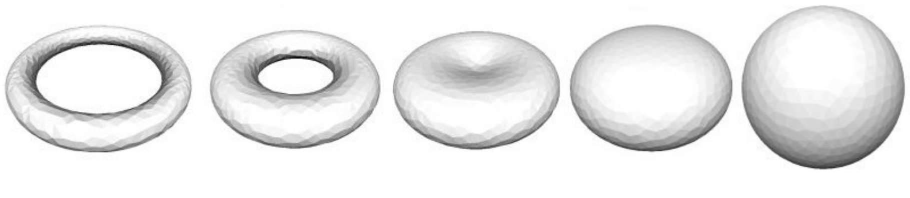
FIGURE 8. *Simple morphing between a torus and a sphere*

passes through them. Once this function is computed, we only need to evaluate it at a certain t in order to get an intermediate shape of the morphing process. To morph a 2D shape into an other for instance, we put the first shape in a 3D space at height $t = 0$, and the second shape at height $t = T$. Then we find a smooth 3D function that passes through the two shapes. To get an intermediate shape, we evaluate the function at a certain height (that represents time) between 0 and 1.

Computing this function is a particular case of variational interpolation. Indeed, this general method does not necessarily need a first shape and a second shape, but only a set of constraint points associated with values. Given a set of constraint points $\{\vec{c_1}, \vec{c_2}, ..., \vec{c_k}\}$ and a set of associated values $\{h_1, h_2, ..., h_k\}$, we want to find a function $f(\vec{x})$ that satisfies $\forall i, f(\vec{c_i}) = h_i$. This is where variational techniques are used. We compute the energy E of the function f, which is a measurment of its quality (the smoother, the better). Then we minimize this energy. In 2D space, this energy is

$$E = \int_\Omega \left(\frac{\partial^2 f}{\partial x^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial y}\right)^2 + \left(\frac{\partial^2 f}{\partial y^2}\right)^2 \tag{5.2}$$

The term *variational* comes from the fact that we compute partial derivatives. Variational interpolation can be extended to higher dimensions. In our case the goal is to create a 4D interpolation function that will link two 3D shapes.

In order to find an interpolation function that minimizes equation 5.2, we can use weighted sums of the radial basis function $\phi(\vec{x}) = \|\vec{x}\|^2 \cdot \log(\|\vec{x}\|)$. Then we have

$$f(x) = \sum_{j=1}^{n} d_j \cdot \phi(\vec{x} - \vec{c_j}) + P(\vec{x}) \tag{5.3}$$

where

- d_j are the weighted coefficients we want to compute
- ϕ is the radial basis function
- $\vec{c_i}$ are the constraint points
- $P(\vec{x})$ is a degree 1 polynom that accounts for the linear and constant portions of f

Then we apply the interpolation constraints $f(\vec{c_i}) = h_i$ to this equation.

$$h_i = \sum_{j=1}^{n} d_j \cdot \phi(\vec{c_i} - \vec{c_j}) + P(\vec{c_i}) \tag{5.4}$$

We can rewrite that equation as a linear system, as it is linear with respect to the d_j and the coefficients of $P(x)$. In 3D for instance, this system is

$$\begin{pmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \cdots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \cdots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_2^x & \cdots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \cdots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \cdots & c_k^z & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

where $\phi_{ij} = \phi(\vec{c_i} - \vec{c_j})$.

Using the triharmonic function ϕ we ensure this matrix to be non singular, we use LU decomposition to solve this system.

In order to build the implicit function correctly, we need to specify some boundaries constraints, so that the function *knows* where inside and outside is. The values h_i will be used for that. In our example, the shape corresponds to the points satisfiying $f(\vec{c_{i_1}}) = 0 = h_{i_1}$. Each point $\vec{c_{i_1}}$ has a normal $\vec{n_{i1}}$. To point $\vec{c_{i_2}} = \vec{c_{i_1}} - k \cdot \vec{n_{i_1}}$ (with k a constant) we associate $h_{i_2} = 1$. It means this point is *inside* the shape.

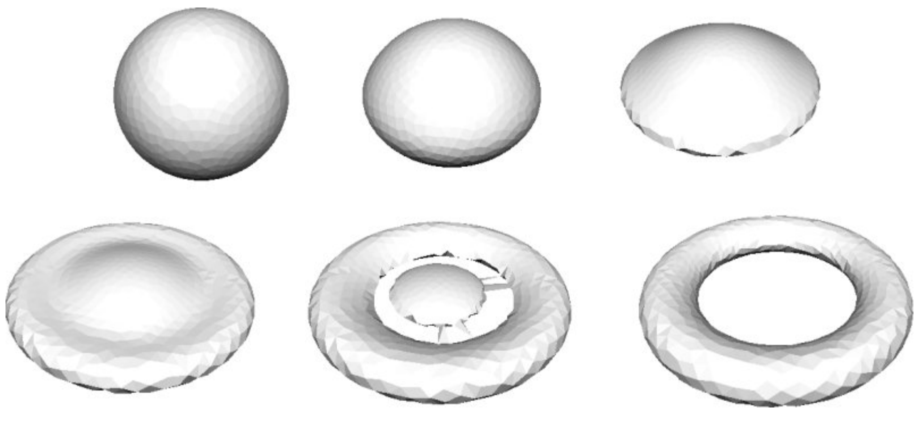

FIGURE 9. *Morphing between a sphere and a torus using a 4D implicit surface*

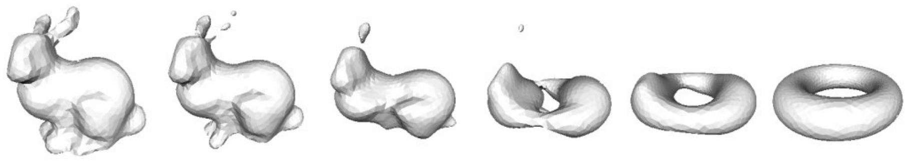

FIGURE 10. *Morphing between the Stanford Bunny and a torus using a 4D implicit surface*

6. Implementation

The framework has been coded in C++ with the CGAL library on a Linux system. The surface tracking framework and the implicit surface computation are completely independant. The program does not run in real time with more than 300 points because of the triangulation, and we don't expect it to be real time. This is still a developement version and there are many things to fix. The goal of this project is to be able to work on a dynamic shape interactively and to generate a sequence of mesh in an appropriate data structure. Until now, we have just been storing an uncompressed sequence of mesh. An other issue is the choice of the representation of the implicit surface. There are no efficient representation to describe a dynamic model, a lot of work has been made for high detailed static models, and for simple deformable shapes which are quite limited in comparison to explicit mesh capabilities.

7. Conclusion

Implicit surfaces offer a nice alternative to perform shape transformation and in a more general way to capture moving objects, even if they have proved to be quite complex to control compared to an explicit model like a mesh. It is easy to handle extreme topological and geometrical changes but not to control them. We can observe that the choice of the representation influences greatly the result, and even on a very simple example like the sphere and the torus.

We would like to thank J.B. Debard (Master student in the lab of Computer Graphics) for is great contribution on the particle sampling and the Delaunay triangulation, and R. Chaine for providing us the source code of her reconstruction algorithm.

References

[1] Andrew P. Witkin and Paul S.Heckbert, *Using Particles to Sample and Control Implicit Surfaces*, In Proceedings of the 21st annual conference on Computer graphics and interactive techniques, pages 269277. ACM Press, 1994.

[2] Leonidas Guibas and Daniel Russel. *An empirical comparison of techniques for updating delaunay triangulations.* In *SCG 04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 170179, New York, NY, USA, 2004. ACM Press.

[3] R. Chaine. *A geometric-based convection approach of 3-d reconstruction.* In Symposium on Geometry Processing, pages 218229, 2003.

[4] Greg Turk and James F. O'Brien. *Shape transformation using variational implicit functions*, In Proceedings of *ACM SIGGRAPH '99* (1999), pp. 335-342.

[5] H.K.Zhao, S. Osher, and R. Fedkiw. *Fast surface reconstruction using the level set method.* In Proceedings of *IEEE Workshop on Variational and Level Set Methods in Computer Vision* (VLSM), 2001.