Applications of Computational Geometry to Computer Aided Design, Computer Graphics and Computer Vision

John Edgar Congote Calle

March 2009

Applications of Computational Geometry to Computer Aided Design, Computer Graphics and Computer Vision

Student: John Edgar Congote Calle
Advisor: Prof. Oscar E. Ruiz

School of Engineering
EAFIT University
Medellín, Colombia

Submitted in partial fulfillment of the requirements for a Masters of Science degree in Engineering from the School of Engineering, EAFIT University

March, 2009

Acknowledgements

This work has been partially supported by the Colombian Council for Science and Technology -Colciencias-. The Spanish Administration agency CDTI, under project CENIT-VISION 2007-1007. VICOMTech Institute and CAD/CAM/CAE Laboratory - EAFIT University. The bunny model is courtesy of the Stanford Computer Graphics Laboratory and Middlebury College for the stereo vision data set.

Contents

Chapter 1

Introduction

Modern industrial applications deal with geometry models that in some stage are impossible to be managed by the human because are beyond the limits of human senses, also some problems need to change the representation of abstract geometry models, for that kind of problems computational solutions should be addressed with the generation of new algorithms and data structures with an optimal utilization of the computational resources. Computational geometry is the discipline which present solutions for that problems, one of the basic structures used in computational geometry is the surface.

The surface can be represented in different ways, like point sets, triangular tessellations, parametric, implicit and explicit. This work is a compilation of applications developed in the CAD CAM CAE Laboratory at EAFIT (Medellin, Colombia) and VICOMTech research center (Donostia - San Sebastian, Spain). Such applications are solutions proposed to different industrial problems, all of them originated in real industrial applications: surface reconstruction and change of model representations.

Particular problems are addressed in this work. Reconstruction of explicit 3D surfaces, cylinders, cones, planes. Triangular tessellation of parametric surfaces. Point set reconstruction from parametric surfaces with curvature sensitivity. Point set reconstruction from projections. Triangular surface tessellation from implicit surfaces. The implementation of the algorithms required some special computational architectures like parallel vectorized machines.

Surface triangulations from parametric representation is a change of representation of the surface, each representation had characteristics that must be kept. Parametric representation of surfaces is very good for modeling and design process but are very difficult in structural analysis however triangular tessellations are very good for structural analysis, so a conversor was generated which keeps the parametric properties of the representation in a triangular tessellated surface which is used for structural analysis.

Implicit surfaces are another form of surface representation very common as an interchange model between surfaces representation, but the correct and fast generation and visualization of surfaces using the current computer hardware

need a change of format of the surface from implicit representation to triangular tessellation. Adaptative Cubical Grid is a modification of the standard algorithm of Marching Cubes. The algorithm generates real-time representation of implicit surfaces and allows the work of implicit surfaces with commodity graphic hardware.

Finally the reconstruction of scenes from projections is a common problem in the computer vision, the problem need to obtain the depth information from stereo cameras, the depth calculation is done by a simply triangulation algorithm, but the matching of the points from the two projections are a very complex problem. For this problem specific graphic hardware was required to process the information in a real time. The implementation of a novel algorithm was created for this problem, which allows the parallel computation of the depth of the projection images.

Chapter 2

Parameter-independent, curvature-sensitive Sprinkle and Star Algorithms for Surface Triangulation

2.1 Context

A project to device a method for the generation of a watertight triangulations sensitives to the curvature from parametric surfaces was developed at CAD CAM CAE Laboratory at EAFIT University. The result of this method generates a watertight triangulated surface and shells ready to be analyzed with FEA. This work has been founded by EAFIT University and the Colombian Council of Research and Technology (COLCIENCIAS) and VICOMTech Research center.

John Congote, research assistant under my direction in the CAD CAM CAE Laboratory, was able to program the application of the devised methods. For such a purpose, theoretical contributions were needed, which appear in:

- A Curvature-Sensitive Parameterization-Independent Triangulation Algorithm. Oscar Ruiz, John Congote, Carlos Cadavid, Juan G. Lalinde. 5th Annual International Symposium on Voronoi Diagrams in Science and Engineering. 4th International Kyiv Conference on Analytic Number Theory and Spatial Tessellations. (Kokichi Sugihara and Deok-Soo Kim, eds.), vol. 2, Drahomanov National Pedagogical University, ISBN 967-966-02-4892-2 (Book), ISBN 978-966-02-4893-9 (CD). September 22-28, 2008 Kiev, Ukraine.

Authors

- Oscar E. Ruiz[1]

- John Congote[1,3]
- Carlos Cadavid[1]
- Juan G. Lalinde[1]
- Guillermo Peris Fajarns[2]
- Beatriz Defez[2]
- Ricardo Serrano[1,2]

1. CAD CAM CAE Laboratory, EAFIT University Colombia
2. Universidad Politecnica de Valencia, Spain
3. VICOMTech Institute, Spain

As co-authors of such publications, we give our permission for this material to appear in this document. We are ready to provide any additional information on the subject, as needed.

Prof. Oscar E. Ruiz
oruiz@eafit.edu.co
Coordinator CAD CAM CAE Laboratory
EAFIT University, Medellin, COLOMBIA

2.2 Abstract

An open area of research in triangulation of parametric surfaces is the difficulty in generating good quality triangles, independently of the underlying particular parameterization of the surface. At the same time, it is important to achieve a triangle density sensitive to the local curvature of the surface. Therefore, a good triangulation must be independent of the parameterization, dependent of the curvature, and it must produce a high quality aspect-ratio triangles. The present article discusses the implementation of two algorithms (*Star* and *Sprinkle*) for the generation of the vertex set for the triangulation of a face F mounted on a parametric surface S. The vertex sets so generated are then processed to produce triangulations in 3D, by using a pseudo - Delaunay algorithm, based on the expansion of edges. Numerous B-Reps have been triangulated with the two methods, with good triangulation quality. The Star algorithm proved to be slower than the Sprinkle one, although its triangles present a slightly better quality.

2.3 Introduction

This article discusses algorithms to triangulate a face F mounted on a parametric surface or 2-manifold $S(u, v)$ in R^3. A face F is a connected subset of S, where $S : R^2 \rightarrow R^3$ is of the form $S(u, v) = [X(u, v), Y(u, v), Z(u, v)]$ with $X, Y, Z : R^2 \rightarrow R$. It is common to assume that $S(u, v)$ is a 1-1 function (no self intersections) and that the $U \times V$ region being the pre-image of $S(u, v)$ is connected.

A triangulation is a planar graph $T = (V_T, E_T)$ in which every vertex participates in (at least) a loop whose size is 3. A natural embedding of a triangulation occurs in R^2, with vertices $p \in V_T$ being points $p = (u, v) \in R^2$ and edges $e = \overline{p_i p_j} \in E$ being straight segments joining vertices p_i and p_j belonging to V_T. A triangulation on a parametric surface $S(u, v) : R^2 \rightarrow R^3$ is the bijective mapping of a triangulation in R^2. In this mapping, each vertex $p_i = (u_i, v_i) \in R^2$ is mapped to the point $S(u_i, v_i)$. Each edge $e = \overline{p_i p_j}$ in R^2 is mapped as a segment in R^3 $e_S(i, j) = \overline{S(u_i, v_i)S(u_j, v_j)}$ (an EDGE e near S).

The literature survey presented next indicates that the parameterization u, v under which a surface is calculated dramatically affects the possibility of generating a topologically and geometrically correct triangulation. In addition to that, even if the triangulation is correct, it may be inconvenient, in that the aspect ratio, quantity, and sensitivity of the generated triangles produce numerically unstable results in the (generally subsequent) process of Finite Element Analysis, or simply may lead to near-infinite runs, useless for practical purposes.

The relation between intervals in the parametric space $|(\Delta u, \Delta v)| = |(u_2 - u_1, v_2 - v_1)|$ and the corresponding distance on S, $|S(u_2, v_2) - S(u_1, v_1)|$ is informally called the *velocity* of the surface. The parameter space suffers a warping in 3D via the function $S(u, v)$ and vice versa. As a consequence, the obvious approach of generating a triangulation in parameter space and to map

it to the surface S produces incorrect or poorly conditioned results.

In this article, two approaches by the authors are compared, which aim to achieve parameter - independent triangulations: (a) star and (b) sprinkle generation of triangulation vertices, followed of a pseudo - delaunay triangulation in 3D space. Given a point $S(u, v)$ in the surface S an additional set of valid neighboring vertices is to be generated, to feed the pseudo - delaunay triangulation in subsequent stages.

2.4 Literature Review

Several classifications of the reviewed literature are possible: in the first place, [30], [6] and [11] treat the re-meshing of an already triangulated B-rep. Level of Detail is tangentially treated in [19], [11] and [37]. [3] and [10] deal with the quasi-equilateral triangulation in F by iterative point search on $U \times V$ 2D parametric space. [38] and [1] pay special attention to the approximation of the face edges as NURBS or Bezier curves in R^2.

In [19] an initial mesh is refined according to the disposition of the observer and the scene lights. An emphasis is set on multi-resolution only on the triangles that actually are seen by the observer. An directed acyclic graph (DAG) is formed, which tracks the modification operations performed on the vertices, edges or faces of a initial model. A Hausdorff distance between the reference and the current surfaces at the modified feature (edge, vertex, face) is evaluated, and the modifications are performed starting at sites with small value of such a measure (i.e. simplifications which only *slightly* modify the current surface when compared with the original one). The algorithms are designed to work in *image space* rather than in *object space*: subdivision is only performed if it does not surpass a threshold in the error introduced in the model, and it has an effect on the image. For example, if a triangle affects only one pixel there is no point in it being further subdivided.

In [38] an emphasis is set in producing watertight tessellations (borderless 2-manifolds in R^3) by using connectivity information. The face-face connectivity between the contiguous faces F_1 and F_2 is represented as a planar trimming curve $C_{1,2}(u)$ that is the common limit between the 2D regions (in parametric space $U \times V$) that bound F_1 and F_2. A curvature-sensitive algorithm places vertices on the $C_{1,2}(u)$ curve. In the current article, the $C_{1,2}(u)$ curve is not required, as the implemented algorithm directly samples the edge curve in R^3 using the *curve* sampling interval specified by the user. In our algorithm, this sample on R^3 is tracked back to the $U \times V$ plane by forming a piecewise linear approximation of the trimming curve $C_{1,2}(u)$.

In [30], the authors start with a watertight 2-manifold M with C^0-continuity (a triangulated tessellation), and build a set of parameterizations for M. Each parameterization covers what is called an *internal node* (representing an M_i 2-manifold with border) in the Reeb Graph describing the topological chances in M along the range of a Morse function $f : M \longrightarrow R$. As per the Morse theory, M_i represents a portion of the M manifold, for which f has no singular

points (topological changes of M) and therefore represents the complete log of the topological evolution of M. Four types of M_i are possible: cylinders, cups, caps, and branchings, according to the borders of M_i. For each type, a pre-defined routine is used, which parameterizes M_i. The step of making compatible the parameterizations for $M_i, i = 0, 1, 2, ...$ is avoided by remeshing the parameterizations with higher density at the borders of M_i. In this form, still a series of parameterizations is possible, while guaranteeing a watertight remeshed M_r version of M.

In [3] and [4] a parameterization-independent algorithm is proposed to triangulate a surface. The aim of the authors is to produce a nearly uniform triangulation. That is, a triangulation in which the triangles be quasi-equilateral. A vertex $p = S(u_0, v_0)$ is chosen on $S(u, v)$ and the plane tangent to S at p, $T_p(p)$, is calculated. On $T_P(p)$, a circle with radius R and its regular inscribed polygon with n sides (called Normal Umbrella - NU) are constructed along with the n incident triangles covering the 2Π angle around p. Each angle that contributes to 2Π is projected onto S, with vertex $p = S(u_0, v_0)$ and projection rays perpendicular to $T_p(p)$. The radius R is inversely proportional to the local curvature. Our own implementation of [4] was found that when the region already sampled closes onto itself, in the EDGE neighborhoods or near FACE holes, an illegal overlap of triangles is produced and the algorithm to avoid it is difficult to control.

In [1] the display of a trimmed NURBS face is discussed, in which a *compilation* stage is performed. The compilation stage is equivalent to what other authors call *the triangulation*. The face in parametric $U \times V$ space corresponds to a 2D connected region with holes, bounded by curved Bezier approximations of the NURBS trimming curves. Bezier approximations are used because there exist reasonable algorithms for the finding of a root of a Bezier curve. The region in $U \times V$ space is cut into sub-regions which have monotonically increasing or decreasing values of the U and V parameters. These subregions are triangulated separately. As an improvement, the algorithm implemented in this paper avoids the splitting of the $U \times V$ region into subregions. It also requires only linear intersections (not Bezier ones), leading to a very simple implementation.

[6] presents a mesh-improving method that starts with a t opologically valid although geometrically poor triangular mesh. The geometric degeneracies are classified as *needles* (quasi isosceles triangles that have two vertices very close to each other) and *caps* (triangles with one angle very close to 180°). The elimination of needles is relatively simple. Elimination of each cap requires the slicing of the *whole* mesh along a particular plane, producing an over-population of triangles. The distance between the final and initial triangulations is used to accept or reject the cap and needle elimination. [11] starts from reverse engineering or tessellation triangular meshes to execute quality improvement and property control on them. The article applies the subdivision and simplification functions to augment and diminish the degree of freedom of the mesh, respectively. Several heuristics are applied to refine the mesh: geometric error, face size, faces shape quality, edge size and vertex valence. In neither [6] nor [11] the mesh modifications are evaluated against the original solid, but against an

existing triangulation of it. A comparison with our article is not possible, since our work seeks an initial triangulation for a given solid.

[10] propose a quasi - isometric local mapping from a parametric surface $S(u, v) : U \times V \rightarrow R^3$ by using the control polyhedron (called there the *surface net*) of the parametric surface. The reasoning is that the surface net closely follows the warping of the parametric surface, while at the same time is very similar to a *locally* developable surface (in turn a planar surface). If we assume that a 1-1 function $f : U \times V \rightarrow S_D \subset R^2$ is known (S_D is the developed surface net), then a quasi equilateral triangulation could be calculated on S_D, and taken to the $U \times V$ domain by using f^{-1}. From $U \times V$ the triangulation is taken to R^3 by using the parametric equations $S(u, v)$. The image in $U \times V$ of the quasi equilateral triangles in S_D is not quasi-equilateral, but their image in R^3 would be. The paper presents no examples in which S_D does not exist for the original surface, and a subdivision must be done, but mentions this possibility.

[37] discusses the issue of triangulation a trimmed surface F by sub-dividing a rectangular domain in the $U \times V$ space using Quadtrees. Each quadtree is recursively subdivided if its corner points in R^3 deviate from a plane beyond a prescribed limit. The trimming NURBS curves, which limit the face F to triangulate are represented as piecewise linear in R^3 and in the parametric $U \times V$ space also. The quadtrees which are completely inside the piecewise linear boundary are trivially triangulated. The ones cut by a loop segment are triangulated only in its internal extent. The quadtree portions in $U \times V$ external to the boundary loops are not triangulated. The paper mentions but does not discuss a process of conciliation between the triangulations of adjacent faces in order to have a seamless triangulation at the faces boundaries.

[34] and [36] are quite important references, used in this paper, regarding the triangulation of 2D regions. In the present work, a Constrained Delaunay Triangulation was used, which respects prescribed edges defined on a set of planar points.

Section 2.5 gives a condensed review of continuous differential geometry concepts. Section 2.6 discusses the methodology followed and its mathematical grounds. Section 2.7 presents the results of the proposed and implemented algorithms, while Section 2.8 concludes the article and discusses possible future work directions.

2.5 Curvature Measurement in Parametric Surfaces

A parametric surface is a function $S : R^2 \rightarrow R^3$, which we asume to be twice derivable in every point. The derivatives are named in the following manner

([29], [7], [24], [2]):

$$S_u = \frac{\partial S}{\partial u}; \quad S_v = \frac{\partial S}{\partial v}; \quad S_{uu} = \frac{\partial^2 S}{\partial u^2}; \quad S_{vv} = \frac{\partial^2 S}{\partial v^2};$$
$$S_{uv} = S_{vu} = \frac{\partial^2 S}{\partial u \partial v}; \quad n = \frac{S_u \times S_v}{|S_u \times S_v|} \tag{2.1}$$

with n being the unit vector normal to the surface S at $S(u, v)$.

The Gaussian and Mean curvatures are given by:

$$K = \frac{LN - MM}{EG - FF}; \qquad H = \frac{LG - 2MF + NE}{2(EG - FF)}; \tag{2.2}$$

where the coefficients E, F, G, L, M, N are:

$$E = S_u \bullet S_u; \quad F = S_u \bullet S_v = S_v \bullet S_u; \quad G = S_v \bullet S_v;$$
$$L = S_{uu} \bullet n; \quad M = S_{uv} \bullet n; \quad N = S_{vv} \bullet n; \tag{2.3}$$

Minimal, Maximal, Gaussian, Mean Curvatures from the Weigarten Application

The Weingarten Application ([7], [2]), W is an alternative way to calculate the Gaussian and Mean curvatures.

$$W = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{2.4}$$

with $a_{11}, a_{12}, a_{21}, a_{22}$ being:

$$a_{11} = \frac{MF - LG}{EG - F^2}; \quad a_{12} = \frac{NF - MG}{EG - F^2};$$
$$a_{21} = \frac{LF - ME}{EG - F^2}; \quad a_{22} = \frac{MF - NE}{EG - F^2} \tag{2.5}$$

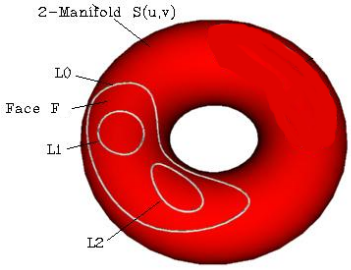
The following facts allow to calculate the curvature measures for S from the Weingarten Application: (i) The eigenvalues k_1 y k_2 of W are called **Principal Curvatures**, with k_1 being the *maximal* curvature and k_2 being the *minimal* curvature (assume that $|k_1| \geq |k_2|$). (ii) $K = det(W)$ is the **Gaussian Curvature**, with $K = k_1 * k_2$. (iii) $2H = trace(W)$ is twice the **Mean Curvature**, with $H = \frac{k_1 + k_2}{2}$. (iv) The maximal and minimal curvatures are: $k_1 = H + \sqrt{H^2 - K}$ and $k_2 = H - \sqrt{H^2 - K}$.

$W * v = k * v$ is the eigenpair equation for the W matrix. The solutions for such an equation are the eigenpairs (k_1, v_1) and (k_2, v_2). Therefore, $W * v_1 = k_1 * v_1$ and $W * v_2 = k_2 * v_2$. The directions of principal curvature *in* $U \times V$ *space* are v_1 and v_2 ($v_1 = (w_{11}, w_{12})$ and $v_2 = (w_{21}, w_{22})$). The directions of maximal and minimal curvatures in R^3 are $u_1 = w_{11} * S_u + w_{12} * S_v$ and $u_2 = w_{21} * S_u + w_{22} * S_v$, respectively.

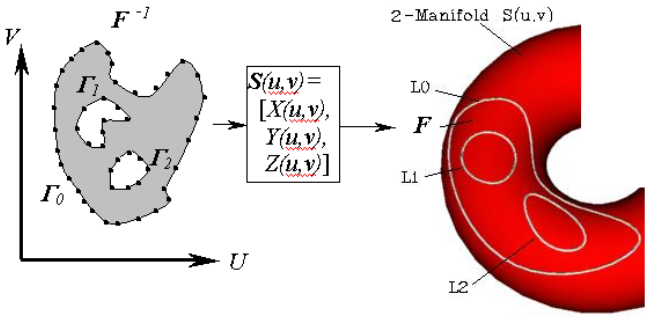
2.6 Methodology

2.6.1 Calculation of F^{-1}

A basic step in calculating a triangulation of a face F (Figure 2.1(a)) is the determination of the pre-image F^{-1} in $U \times V$ space R^2 of such a face, that is, the pre-image of F under the parametric surface $S(u, v)$ (Figure 2.1(b)). F^{-1} is the connected region (possibly with holes) in R^2 bounded by an external curve Γ_0 and internal ones $\Gamma_i, i = 1, ..., n$. In what follows we use the notation $S^{-1}(F)$ to mean F^{-1}. Figure 2.2 shows that the face F is carried by the parametric surface $S(u, v)$. A PL sample of the border ∂F of the face F is carried out to calculate the pre-image $S^{-1}(F)$ on $U \times V$. A calculation of the 2D bounding box $minmax(S^{-1}(F))$ of $S^{-1}(F)$ follows.



(a) Face defined on a 2-Manifold (donnut)



(b) Forward Map $S(u, v)$ from $F^{-1} \subset R^2$ onto F

Figure 2.1: Pre-image of a 3D Face F.

2.6.2 Star Algorithm

The calculation and sampling of the pre-image of F, $S^{-1}(F)$ (Figure 2.2) is a pre-condition for starting the Star algortihm. This pre-processing initializes a queue Q of the 3D vertices that are accepted for the triangulation of face F.

These initially determined vertices are the isometric sampling of the loops L_i forming the boundary of F.

As the Star algorithm proceeds, a vertex $p \in Q$ is extracted from Q. A star is calculated around p (Figure 2.3(a)), lying on the plane $\Pi[p, n]$ tangent to S at p with normal n (Equation 2.5). The star has radius $r(H)$, and six (6) vertices. The radius $r(H)$ of the star is inversely proportional to the mean curvature H of S at p (Figure 2.3(a)). A number of vertices of the star are rejected if they (their projection on S) fall outside F. Additional vertices are rejected if they are too close to the already accepted vertices inside F. The survivor vertices are projected on F and included in the queue Q. The algorithm for vertex calculation stops when Q is empty.

After the vertex set V_T on the face F is determined, a modified Delaunay Triangulation with the points in V_T proceeds, which is explained later.

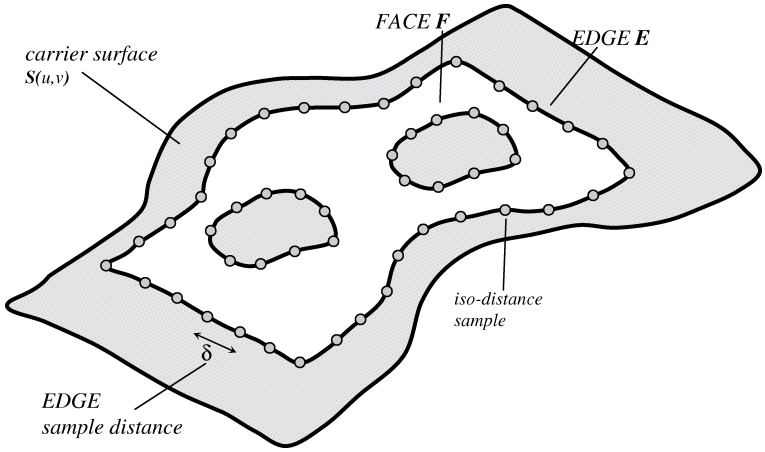
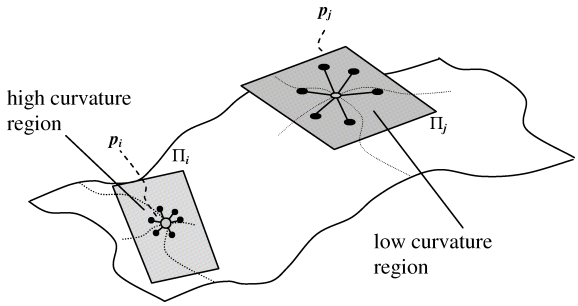


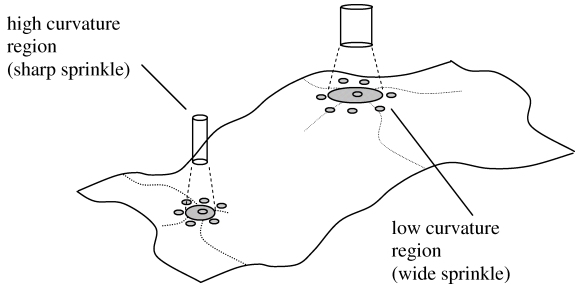
Figure 2.2: The iso-distance sample of edges on F generate the triangulation initial vertex set.

2.6.3 Sprinkle Algorithm

Figure 2.3(b) displays the fact that the creation of random points (u, v) inside F^{-1} is encouraged in neighborhoods in which the curvature H is high. This is done in this manner: (i) Create a point (u, v) with (u, v) being random numbers and (u, v) inside $minmax(S^{-1}(F))$. (ii) Return to step (i) if (u, v) is outside F^{-1}, (iii) Assess (u, v) against curvature as follows: define a small radius $r(H)$ if a local high curvature $S(u, v)$ exists, and vice versa. (iv) Reject (u, v) and go back to (i) if the 3D ball $B(S(u, v), r(H))$ contains another vertex already marked on the face F. (v) Accept (u, v) if the ball $B(S(u, v), r(H))$ on F contains no other vertex already accepted and mark $S(u, v)$ on F as an acceptable vertice for the future triangulation. Include $S(u, v)$ in the vertex set



(a) Star algorithm. Sampling distance as function of the curvature



(b) Sprinkle algorithm. Sprinkle focus as a function of the curvature

Figure 2.3: Comparison between Star and Sprinkle algorithms for generation of triangulation vertices.

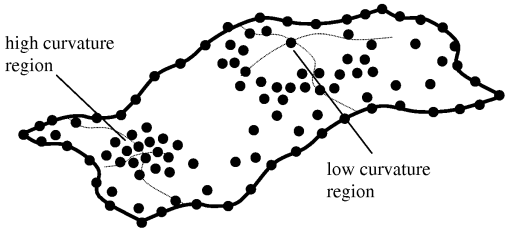
V_T.

One knows that the face F is already sufficienty sprinkled by vertices if a given number N_t of trials to mark vertices $S(u, v)$ on F fails. The effect of such heuristic is hinted in Figure 2.4(a). The result of the algorithm acting on a test B-rep is displayed in Figure 2.4(b). This B-Rep (called *Yello*) is specifically generated to have faces with velocity in the u direction being dramatically diferent from the velocity in v direction, with which a grid generated in the $U \times V$ space will produce a triangulation with folds and plies. In contrast, Figure 2.4(b) shows excellent aspect ratio (nearly equilateral) triangles.

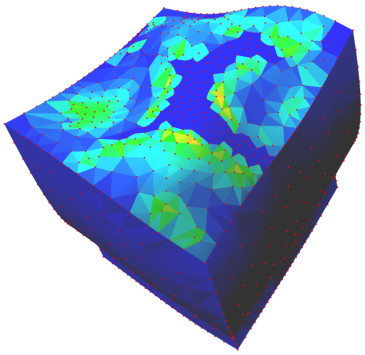
A modified Delaunay Triangulation is calculated with the points in V_T. Results are shown in Figure 2.4(b).

2.6.4 A Pseudo - Delaunay Triangulation

To generate the connectivity information of the set of discretized points two approaches were used. The first one is to calculate a triangulation in parametric space (using [35]). This approach does not produce satisfactory results whenever the warping of the $U \times V$ space with respect to the R^3 euclidean distance is



(a) Sprinkle span as function of local curvature



(b) Random Sprinkle of points in R^3 space. Yello B-Rep.

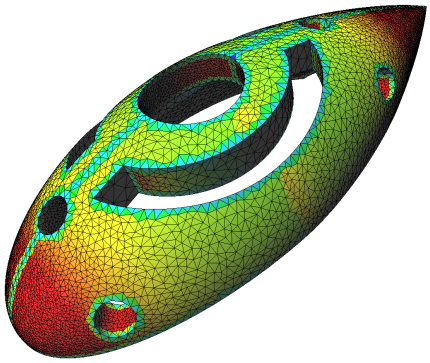
Figure 2.4: Regions with higher curvature are sprinkled with smaller sampling interval.

non-homogeneous (see Introduction section). The second one (called *Pseudo-Delaunay*) is described next.

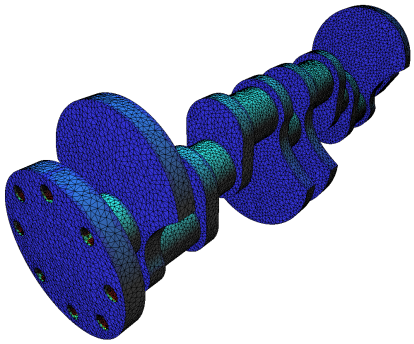
The pseudo-delaunay triangulation is built with vertices of the V_T set. The invariant of the algorithm establishes that there exists a sequence of edges Q_E that are the (advancing) frontier of the triangulation, and a set of availabe vertices V_T which are to be used, along edges of Q_E, to make triangles and therefore to advance the frontier of already triangulated region. The algorithm stops when the triangulated region equals F (i,e. when the set of available edges for expansion (forming a triangle with a vertex) is exhausted.

An initial PL approximation of ∂F, the loops bounding F, is available (see previous sections) in a sequence Q_E. An edge $e = (p_i, p_j)$ is chosed from Q_E. A vertex $p \in V$ is identified such that the triangle $t = (v_i, v_j, v)$ is a pseudo-Delaunay one (see below). The edge $e = (p_i, p_j)$ is replaced in Q_E with the edges (p_i, p) and (p_j, p) in the queue Q_E.

A pseudo-Delaunay triangle $t = (p_i, p_j, p)$ with $p_i \in V_T$ is tested in this



(a) Helmet.



(b) Crank Shaft.

Figure 2.5: Helmet and Crank Shaft.

manner: (a) Find the circumcenter c_t of the triangle t and the radius r_t of the planar circle containing t. (b) Consider the ball $B_t(c_t, r_t)$ centered in c_t with radius r_t. (c) Test every vertex q of V for inclusion on B_t. If no $q \in V$ is inside B_t, t is a pseudo-Delaunay triangle (Figure 2.8).

2.7 Results

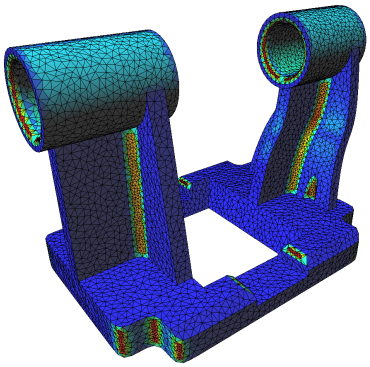
Several Boundary Representations were used as data sets for testing the two triangulation vertex generation algorithms (Star vs. Sprinkle). The quality of the triangulations by the two methods is roughly similar. The times for generating the vertex sets are significantly different (see Conclusions section).

The algorithms have been tested with the following data sets: Helmet, Crankshaft, Bearing, Pre-columbian Fish, Coupling, Gruyere. The results appear in Figures 2.5(a), 2.5(b), 2.6(a), 2.6(b), 2.6.3, 2.9(a) and 2.9(b).

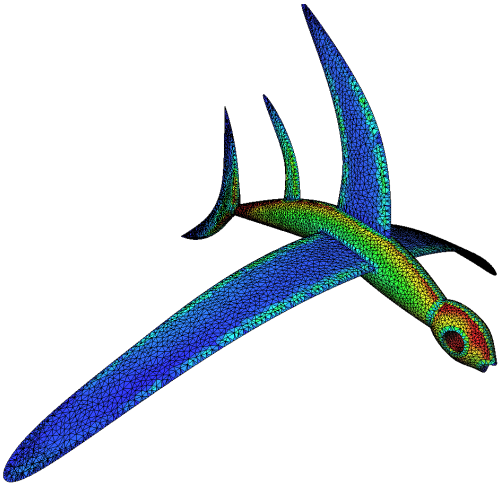
2.8 Conclusions and Future Work

The Star algorithm produces slightly better triangles than the Sprinkle algorithm. This advantage is to be expected, since the Star algorithm, by definition, creates on the tangent plane $\Pi[n, p]$ a regular hexagon (i.e. 6 equilateral triangles) incident to the vertex p to expand, and projects them onto the F surface. In contrast, the Sprinkle algorithm is considerably more efficient for generating the vertex point set than the Star algorithm. The execution times are displayed in the following figures: Helmet, 2.10(a), Aphrodite data Set. 2.10(b) and Hand 2.10(c). The statistics presented are concentrated in the generation of the vertex set, because the algorithm (pseudo-Delaunay) for the connection of the vertex set is common.

Future work is needed in the aspects of the numerical assessment of the goodness of the triangle set, specifically in the application of Finite Element Analysis.



(a) Bearing.



(b) Pre-columbian Fish.

Figure 2.6: Bearing and Pre-columbian Fish.

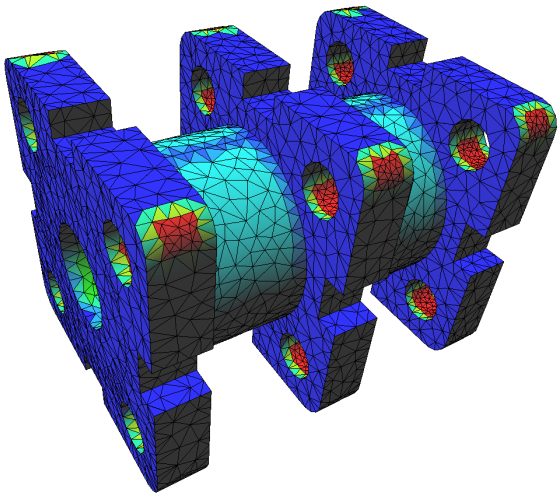


Figure 2.7: Coupling Triangulation

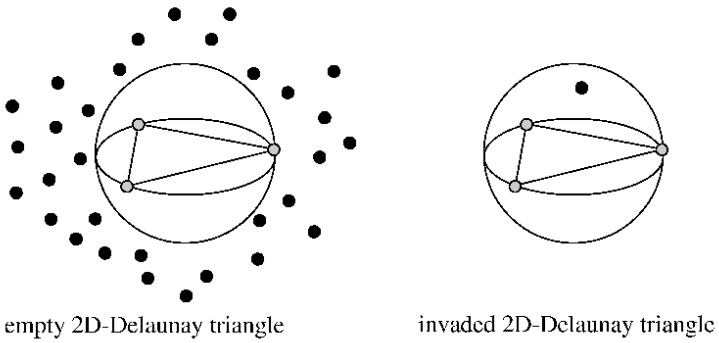
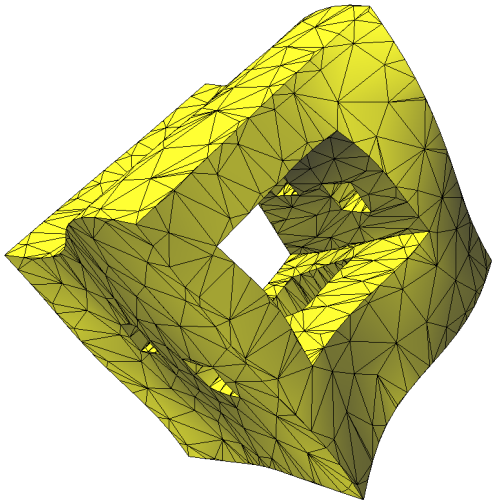
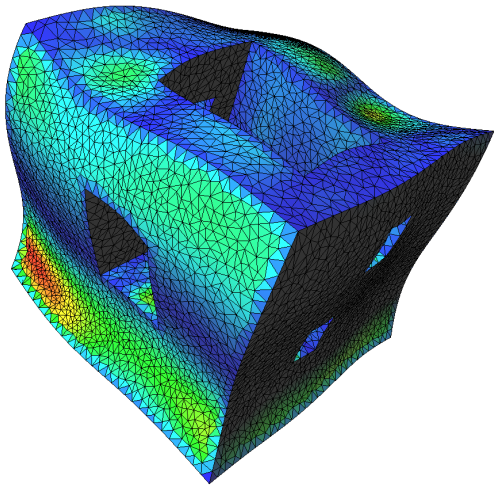


Figure 2.8: Pseudo-Delaunay validation of triangles lying on the face F.

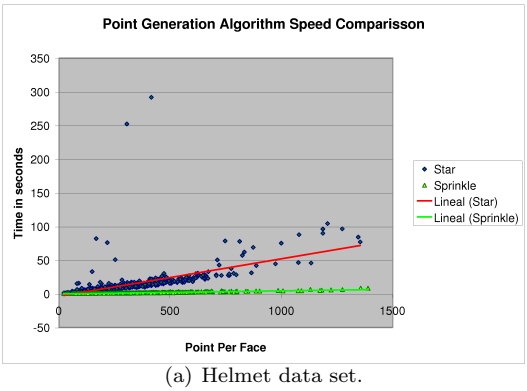


(a) Gruyere B-Rep triangulated with the Star Algorithm

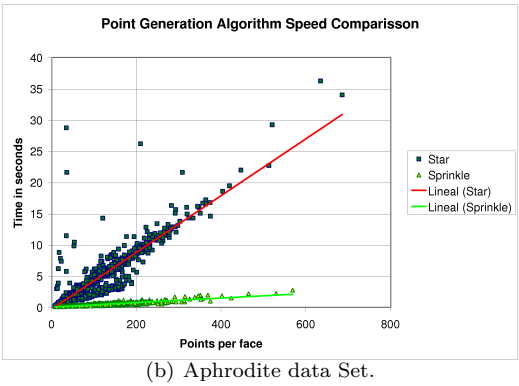


(b) Gruyere B-Rep triangulated with the Sprinke Algorithm

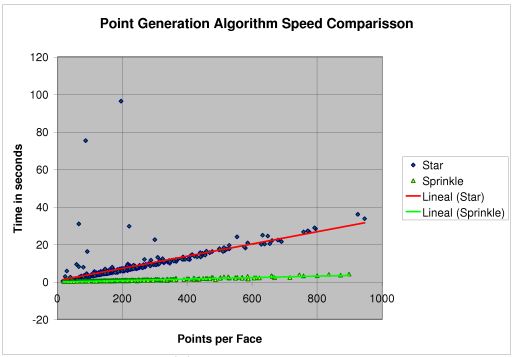
Figure 2.9: Triangulations of Gruyere Boundary Representation



(a) Helmet data set.



(b) Aphrodite data Set.



(c) Hand data set.

Figure 2.10: Comparisson between Star and Sprinkle algorithms for generation times for triangulation vertices.

Chapter 3

Adaptative cubical grid for isosurface extraction

3.1 Context

A project to device a method for the generation of real-time triangular tessellation of implicit surfaces was developed at VICOMTech Institute. The result of this method generates a triangular tessellation of implicit surfaces which can be displayed in commodity computers in real time. This work has been founded by EAFIT University, the Colombian Council of Research and Technology (COLCIENCIAS) and The Spanish Administration agency CDTI, under project CENIT-VISION 2007-1007, VICOMTech Institute.

John Congote, research assistant under my direction in the CAD CAM CAE Laboratory, was able to program the application of the devised methods. For such a purpose, theoretical contributions were needed, which appear in:

- Adaptative Cubical Grid For Isosurface Extraction. John Congote, Aitor Moreno, Inigo Barandiaran, Javier Barandiaran, Oscar E. Ruiz. 4th International Conference on Computer Graphics Theory and Applications GRAPP-2009. ISBN 978-989-8111-67-8, pp 21-26. Feb 5-8, 2009. Lisbon, Portugal.

Authors

- John Congote[1,2]
- Aitor Moreno[2]
- Iñigo Barandiaran[2]
- Javier Barandiaran[2]
- Oscar E. Ruiz.[1]

1. CAD CAM CAE Laboratory, EAFIT University Colombia

2. VICOMTech Institute, Spain

As co-authors of such publications, we give our permission for this material to appear in this document. We are ready to provide any additional information on the subject, as needed.

Prof. Oscar E. Ruiz
oruiz@eafit.edu.co
Coordinator CAD CAM CAE Laboratory
EAFIT University, Medellin, COLOMBIA

3.2 Abstract

This work proposes a variation on the Marching Cubes algorithm, where the goal is to represent implicit functions with higher resolution and better graphical quality using the same grid size. The proposed algorithm displaces the vertices of the cubes iteratively until the stop condition is achieved. After each iteration, the difference between the implicit and the explicit representations are reduced, and when the algorithm finishes, the implicit surface representation using the modified cubical grid is more detailed, as the results shall confirm. The proposed algorithm corrects some topological problems that may appear in the discretisation process using the original grid.

3.3 Introduction

Surface representation from scalar functions is an active research topic in different fields of computer graphics such as medical visualisation of Magnetic Resonance Imaging (MRI) and Computer Tomography (CT) [20]. This representation is also widely used as an intermediate step for several graphical processes [27], such as mesh reconstruction from point clouds or track planning. The representation of a scalar function in 3D is known as implicit representation and is generated using continuous algebraic iso-surfaces, radial basis functions [8] [25], signed distance transform [12] or discrete voxelisations.

The implicit functions are frequently represented as a discrete cubical grid where each vertex has the value of the function. The Marching Cubes algorithm (MC) [22] takes the cubical grid to create an explicit representation of the implicit surface. The MC algorithm has been widely studied as has been demonstrated by Newman [26]. The output of the MC algorithm is an explicit surface represented as a set of connected triangles known as a polygonal representation. The original results of the MC algorithm presented several topological problems as demonstrated by Chernyaev [9] and have already been solved by Lewiner [21].

The MC algorithm divides the space in a regular cubical grid. For each cube, a triangular representation is calculated, which are then joined to obtain the explicit representation of the surface. This procedure is highly parallel because each cube can be processed separately without significant interdependencies. The resolution of the generated polygonal surface depends directly on the input grid size. In order to increase the resolution of the polygonal surface it is necessary to increase the number of cubes in the grid, increasing the amount of memory required to store the values of the grid.

Alternative methods to the MC algorithm introduce the concept of generating multi-resolution grids, creating nested sub-grids inside the original grid. The spatial subdivision using octrees or recursive tetrahedral subdivision techniques are also used in the optimisation of iso-surface representations. The common characteristic of these types of methods is that they are based on adding more cells efficiently, to ensure a higher resolution in the final representation.

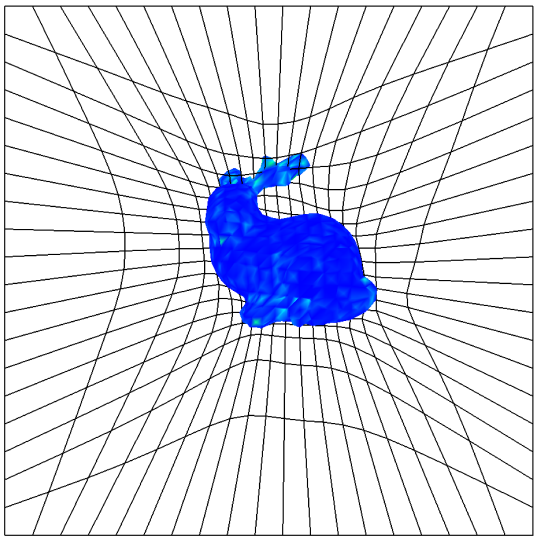


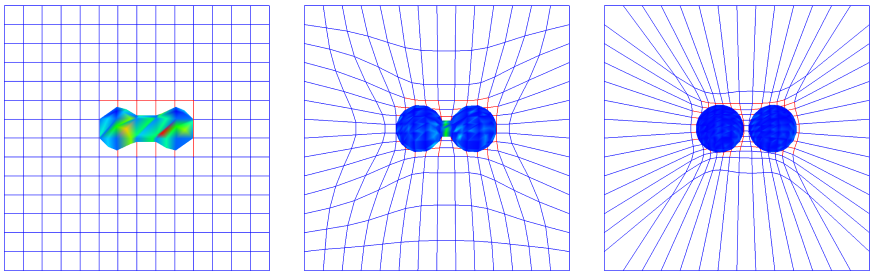
Figure 3.1: Optimised Grid with 20^3 cubes representing the bunny.

This work is structured as follows: In Section 3.4, a review of some of the best known MC algorithm variations is given. Section 3.5 describes the methodological aspects behind the proposed algorithm. In Section 3.6 details the results of testing the algorithm with a set of implicit functions. Finally, conclusions and future work are discussed in Section 3.7.

3.4 Related Work

Marching Cubes (MC) [22] has been the *de facto* standard algorithm for the process generating of explicit representations of iso-surfaces from scalar functions or its implicit definition The MC algorithm takes as an input a regular scalar volumetric data set, having a scalar value residing at each lattice point of a rectilinear lattice in 3D space. The enclosed volume in the region of interest is subdivided into a regular grid of cubes. Each vertex of all cubes in the grid is set the value of the implicit function evaluated at the vertex coordinates. Depending on the sign of each vertex, a cube has 256 (2^8) possible combinations, but using geometrical properties, such as rotations and reflections, the final number of combinations is reduced to 15 possibilities. These 15 surface triangulations are stored in Look-Up Tables (LUT) for speed reasons. The final vertices of the triangular mesh are calculated using linear interpolation between the values assigned to the vertices of the cube. This polygonal mesh representation is ideally suited to the current generation of graphic hardware because it has been optimised to this type of input.

MC variations were developed to enhance the resolution of the generated explicit surfaces, allowing the representation of geometrical details lost during



(a) Original Grid. The two spheres are displayed as a singular object due to the poor resolution in the region

(b) Intermediate Grid. Both spheres are displayed well, but are still joined

(c) Final Grid. The new resolution displays two well shaped and separated spheres with the same number of cubes in the grid

Figure 3.2: 2D slides representing three different states in the evolution of the algorithm of two nearby spheres

MC discretisation process. Weber [42] proposes a multi-grid method. Inside an initial grid, a nested grid is created to add more resolution in that region. This methodology is suitable to be used recursively, adding more detail to conflictive regions. In the final stage, the explicit surface is created by joining all the reconstructed polygonal surfaces.

It is necessary to generate a special polygonisation in the joints between the grid and the sub-grids to avoid the apparition of cracks or artifacts. This method has a higher memory demand to store the new values of the nested-grid.

An alternative method to refine selected region of interest is the octree subdivision [33]. This method generates an octree in the region of existence of the function, creating a polygonisation of each octree cell. One of the flaws of this method is the generation of cracks in the regions with different resolutions. This problem is solve with the Dual Marching Cubes method [31] and implemented for algebraic functions by Pavia [28]

The octree subdivision method produces edges with more than two vertices, which can be overcome by changing the methodology of the subdivision. Instead of using cubes, tetrahedrons were used to subdivide the grid, without creating nodes in the middle of the edges [17]. This method recursively subdivides the space into tetrahedrons.

The previous methodologies increment the number of cells of the grid in order to achieve more resolution in the regions of interest. Balmelli [5] presented an algorithm based on the movement of the grid to a defined region of interest using a warping function. The result is a new grid with the same number of

cells, but with higher resolution in the desired region.

Our method is also based on the displacement of the vertices of the grid, obtaining dense distribution of vertices near to the iso-surface. (see Figure 3.2)

3.5 Methodology

The proposed algorithm is presented as a modification of the MC algorithm. The principal goal is the generation of more detailed approximations of the given implicit surfaces with the same grid resolution.

Applying a selective displacement to the vertices of the grid, the algorithm increases the number of cells containing the iso-surface. In order to avoid self-intersections and to preserve the topological structure of the grid, the vertices are translated in the direction of the surface. The displacement to be applied to each vertex is calculated iteratively until a stop condition is satisfied.

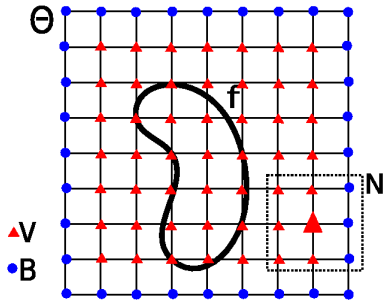


Figure 3.3: Grid nomenclature, Θ cubical grid, $f(x, y, z) = 0$ implicit function, N vertex neightboor, V vertices inside the grid, B vertices at the boundary of the grid

Let be Θ a rectangular prism tessellated as a cubical honeycomb, W the vertices of Θ [Eq. 3.1], B the boundary vertices of Θ [Eq. 3.2], and V the inner vertices of Θ [Eq. 3.3]. For each vertex $v_i \in V$, a N_i set is defined as the *26 adjacent* vertices to v_i, denoting each adjacent vertex as $n_{i,j}$ [Eq. 3.4]. (see Figure 3.3)

$$W = \{w_i / w_i \in \Theta\} \tag{3.1}$$
$$B = \{b_i / b_i \in \delta\Theta\} \tag{3.2}$$
$$V = W - B \tag{3.3}$$
$$N_i = \{n_{i,j} / n_{i,j} \text{ is } j\text{th neighbourgh of } v_i\} \tag{3.4}$$

The proposed algorithm is an iterative process. In each iteration, each vertex v_i of the grid Θ is translated by a d_i distance vector, obtaining a new configuration of Θ, where *i)* the topological connections of the grid are preserved, *ii)* the number of cells containing patches of f are greater than, or equal to, the

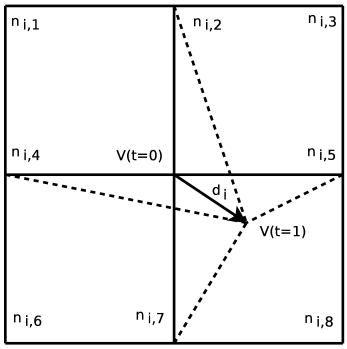


Figure 3.4: two consecutives iterations are show where the vertex v is moved between the iterations $t = 0$ and $t = 1$. The new configuration of the grid is shown as dotted lines.

previous value, and *iii)* the total displacement [Eq. 3.7] of the grid is lower and is used as the stop condition of the algorithm when it reach a value Δ(see Figure 3.4).

The distance vector d_i is calculated as shown in [Eq. 3.6] and it can be seen as the resultant force of each neighbouring vertex scaled by the value of f at the position of each vertex. In order to limit the maximum displacement of the vertices and to guarantee the topological order of Θ, the distance vector d_i is clamped in the interval expressed in [Eq. 3.5]

$$0 \leq |d_i| \leq \text{MIN}\left(\frac{|n_{i,j} - v_i|}{2}\right) \tag{3.5}$$

$$d_i = \frac{1}{26}\sum_{n_{i,j}} \frac{n_{i,j} - v_i}{1 + |f(n_{i,j}) + f(v_i)|} \tag{3.6}$$

$$\sum_{v_i} |d_i| \geq \Delta \tag{3.7}$$

The algorithm stops when the sum of the distances added to all the vertices in the previous iteration is less that a given threshold Δ [Eq. 3.7] (see Algorithm 1).

repeat
  s := 0;
  foreach *Vertex* v_i **do**
    $d_i := \frac{1}{26} \sum_{n_{i,j}} \frac{n_{i,j}-v_i}{1+|f(n_{i,j})+f(v_i)|}$;
    mindist := MIN $\left(\frac{|n_{i,j}-v_i|}{2}\right)$;
    $d_i := \bar{d_i}$CLAMP $(|d_i|, 0.0, \text{mindist})$;
    $v_i := v_i + d_i$;
    s := s + $|d_i|$;
  end
until s $\geq \Delta$;

Algorithm 1: Vertex Displacement Pseudo-algorithm. $|x|$ represents the magnitude of x, $\bar{v}$ represents the normalised vector of v

3.6 Results

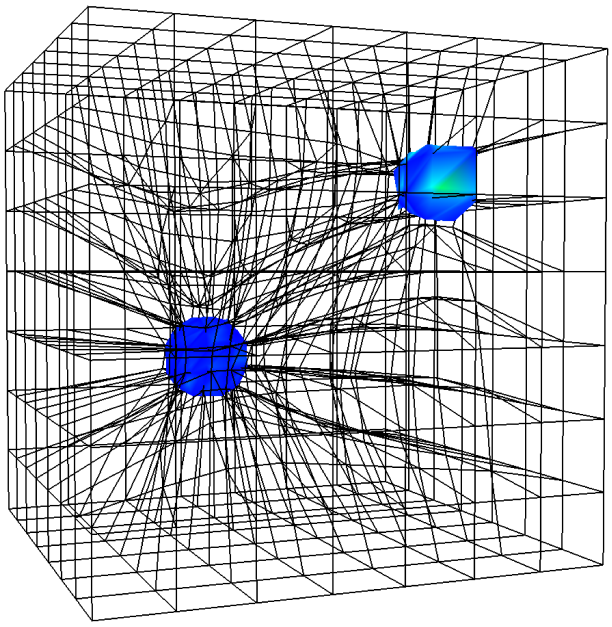


Figure 3.5: Two balls in different positions with a scalar function as the distance transform, representing the behaviour of the algorithm with different objects in the space.

The proposed algorithm was tested with a set of implicit functions as distance transforms (see Figure 3.5) and algebraic functions (see Figure 3.6(a)). For demonstration purposes, the number of cells has been chosen to be very low

GS_1^3	MC. QLTY.	AMC. QLTY.	GS_2^3
10	0.958555	-	-
20	0.369976	0.32257	10
30	0.188298	0.186994	20
40	0.129414	0.127588	30
50	0.094878	0.092761	40

Table 3.1: Quality is measured as the average distance between the mesh vertices and the real surfaces. The columns 1 and 2 represent the quality of the bunny model (MC. QLTY.) using the given cubical grid size (GS_1) with the standard MC algorithm. The columns 3 and 4 shows the quality (AMC. QLTY.) using the given cubical grid size (GS_2) with the proposed algorithm after 30 iterations. The quality values (columns 2 and 3) have been aligned to be as equal as possible, showing that to achieve a target quality, the grid size using the proposed algorithm is less than the required using the standard MC algorithm.

to aid in the visual detection of the improvements produced by the algorithm. For the visualisation process we use Marching Tetrahedra because it produces correct topological representation of the iso-surface, and allows the identification of the topological correctness of the algorithm.

The obtained results of the algorithm are visually noticeable, as is shown in Figure 3.6. Without using the algorithm, the two spheres model is perceived as a single object (see Figure 3.2). In an intermediate state the spheres are still joined, but their shapes are more rounded. In the final state, when the algorithm converges, both spheres are separated correctly, each one being rendered as a near perfect sphere. Thus, using the same grid resolution and the proposed algorithm, the resolution of the results has been increased.

The proposed algorithm iteratively increases the number of cells containing the surface, adding more detail to the new representation. Figure 3.6(b) shows the incremental evolution of such a number of cubes containing the surface, tending toward a doubling of the number. The average distance triangulation vertices to the original surface was calculated as presented in table 3.6. The proposed algorithm can represent the surface with a good quality with a fraction of the amount of cells required with the original MC algorithm.

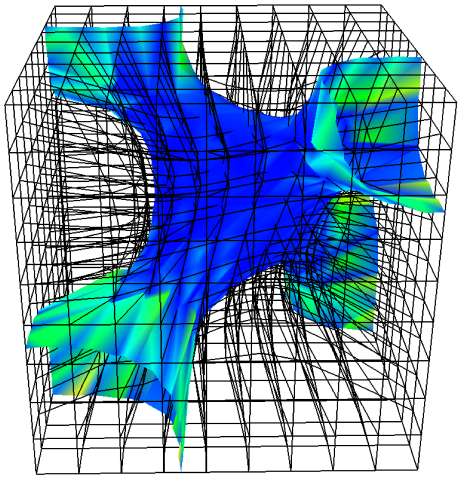
The total displacement of the vertices in each iteration is decreasing rapidly toward zero after a number of iterations as show in the Figure 3.6(c). Despite the seemingly high number of iterations, the algorithm is executed only once for static functions, and can be processed in the background. Even when the implicit function is a time variant, the cubical grid can be reused as the input cubical grid for the next algorithm execution. When the implicit function changes smoothly, the algorithm quickly re-converges after just a few iterations significantly reducing the computational effort.

3.7 Conclusions and Future Work

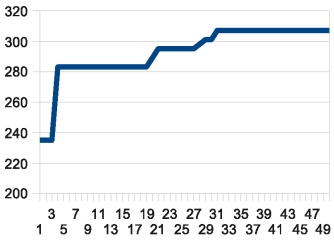
Our proposed iterative algorithm has shown significant advantages in the representation of distance transform functions. With the same grid size, it allows a better resolution by displacing the vertices of the cube grids towards the surface, increasing the number of cells containing the surface.

The algorithm was tested with algebraic functions, representing distance transform of the models. The generated scalar field has been selected to avoid the creation of regions of false interest, which are for static images in which these regions are not used.

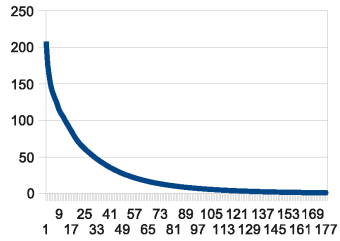
The number of iterations is directly related to the chosen value Δ as it is the stop condition. The algorithm will continuously displace the cube vertices until the accumulated displacement in a single iteration is less than Δ. In the results, it can be seen that this accumulated distance converges quickly to the desired value. This behaviour is very convenient to represent time varying scalar functions like 3D videos, where the function itself is continuously changing. In this context, the algorithm will iterate until a good representation of the surface is obtained. If the surface varies smoothly, the cube grid will be continuously and quickly readapted by running a few iterations of the presented algorithm. As the surface changes may be assumed to be small, the number of iterations until a new final condition is achieved will be low. The obtained results will be a better real-time surface representation using a coarser cube grid.



(a) Algebraic function rendered with an optimised grid



(b) Cube increment (Y axis) vs. Iteration evolution (X axis)



(c) Total displacement of the grid (Y axis) vs. Iteration evolution (X axis)

Figure 3.6: Grid evolution for an algebraic function, comparing the number of cubes which contains the iso-surface and the total displacement of the vertices plotted against the stage of execution of the algorithm

Chapter 4

Realtime dense stereo matching with dynamic programming in CUDA

4.1 Context

A project to device a method for the calculation of depth from to projected images in real-time was developed at VICOMTech Institute. The result of this method generates a dense depth map which indicates the distance from the camera of the captured images in real-time using the graphic card for processing . This work has been founded by EAFIT University, the Colombian Council of Research and Technology (COLCIENCIAS) and The Spanish Administration agency CDTI, under project CENIT-VISION 2007-1007, VICOMTech Institute.

John Congote, research assistant under my direction in the CAD CAM CAE Laboratory, was able to program the application of the devised methods. For such a purpose, theoretical contributions were needed, which appear in:

- Publication pending

Authors

- John Congote[1,2]
- Iñigo Barandiaran[2]
- Javier Barandiaran[2]
- Oscar E. Ruiz.[1]

1. CAD CAM CAE Laboratory, EAFIT University Colombia
2. VICOMTech Institute, Spain

As co-authors of such publications, we give our permission for this material to appear in this document. We are ready to provide any additional information on the subject, as needed.

Prof. Oscar E. Ruiz
oruiz@eafit.edu.co
Coordinator CAD CAM CAE Laboratory
EAFIT University, Medellin, COLOMBIA

4.2 Abstract

Real-time depth extraction from stereo images is an important process in computer vision. This paper proposes a new implementation of the dynamic programming algorithm to calculate dense depth maps using the CUDA architecture achieving real-time performance with consumer graphics cards. We compare the running time of the algorithm against CPU implementations and demonstrate the scalability property of the algorithm by testing it on different graphics cards.

4.3 Introduction

Dense depth map calculation is a common problem in computer vision. Where in the image a depth distance is calculated for every pixel, generating a 3D scene from the given image. The problem has been widely studied and diverse approaches to solve the problem have been proposed [41] [18]. Furthermore, the problem had been tracked since 2001 by Scharstein and Szeliski [32] who had also defined a taxonomy for stereo vision algorithms dividing them into local, global and scan-line, or semi-global methods. Depth map calculation is very useful in the area of robotics, 3D scene reconstruction, and in the emerging field of 3D television with technologies such as Philips WOWvx.

Real-time (**RT**) and near real-time algorithms to generate depth maps have improved substantially in recent years due to the following two factors. First, research in the field has developed new algorithms which reduce the complexity of algorithms and the increase in computational power required by the **CPU**, however these advances have been more dramatic with Graphic Processing Unit (**GPU**). New technologies such as Compute Unified Device Architecture (**CUDA**) opens a new promising field of work for these kinds of problems. Recently, much of the research has been carried out on the implementation of different depth estimation algorithms in **GPU**.

Local methods are more straightforward to implement using **GPU**. These methods were measured and compared by Gong [15] and Tombari[39]. Also global methods were implemented in **GPU**, as shown by Gibson[14] and Yang[43]. The highest quality results are achieved by using methods based on Belief Propagation (**BP**)[18]. However, these methods are computationally intensive. Methodologies that strike a good balance between velocity and quality are algorithms that use a combination of a local method, dynamic programming and a post processing step, as demonstrated by Wang [40]. Our proposal is based mainly on this work.

Dynamic Programming (**DP**) has been a difficult problem to solve in parallel architectures, as shown by Galil [13]. Some implementations of similar (**DP**) algorithms have been implemented in **GPU**, Manavski [23]. A useful practical implementation of the **DP** algorithm for depth map calculation has been attempted in the past by Gong [16] without achieving significant improvements over comparative **CPU** implementations.

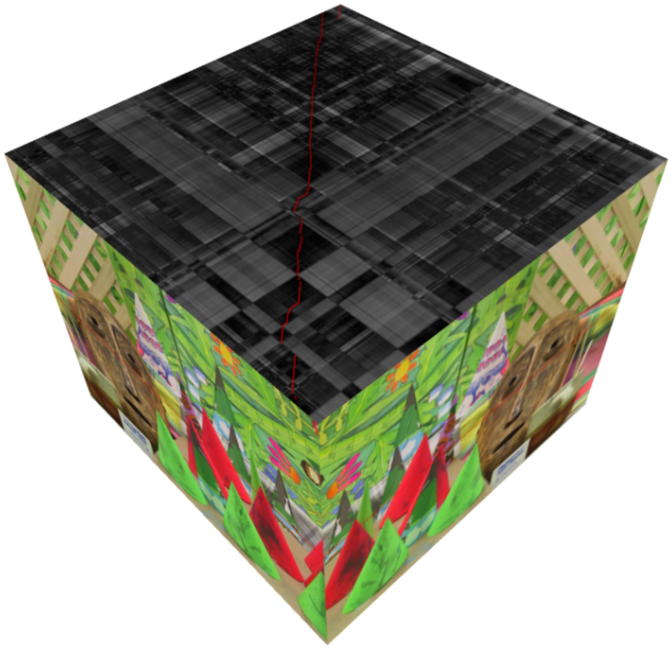
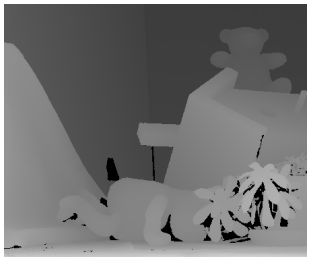


Figure 4.1: Calculated Matrix M_{190} of the cones model, and the solution obtained with the **DP** algorithm

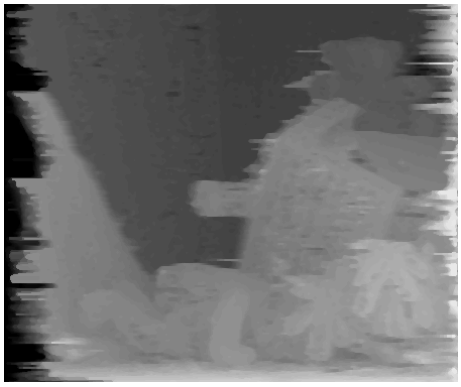
The contribution of this paper is the implementation of the **DP** algorithm in **GPU** using the Nvidia **CUDA** architecture. The algorithm calculates dense depth maps from stereo images. For benchmarking we use a comparable **CPU** implementation as a baseline and measure scalability of the GPU implementation across multiple **GPU** configurations. The results prove that the algorithm, with **RT** performance, can be viably implemented on inexpensive consumer graphics cards.

The paper is structured as follows: The original algorithm for the calculation of the dense depth map is explained in section (4.6), the basics of the Nvidia **CUDA** architecture shall be described in section (4.5). The parallelisation methodology applied in this paper is detailled in section (4.7). The results of the quality of the algorithm and a comparison of running times for each configuration are presented in section (4.8), finally conclusions are presented in section (4.8).

(a) Left image



(b) Ground Truth



(c) Calculated Disparity

Figure 4.2: Depthmap calculation for teddy image of Middlebury stereo vision data set. the calculated disparity was done using our algorithm and a median filter as a posprocessing step.

4.4 Glosary

DP: Dynamic Programming

RT: Real Time

CPU: Central Process Unit

GPU: Graphic Process Unit

CUDA: Computer Unified Device Architecture

BP: Belief propagation

GPGPU: General Purpose computation on GPU

SAD: Sum of Absolute Differences

RGB: Red-Green-Blue color space

4.5 CUDA

The graphics processor is new processing unit type common in the personal computer that are optimised for parallel processing, the evolution of this kind of hardware in recent years has presented a new possibility of complex computations that were restricted in the past because of the excessive time required for the computation or execution of the algorithms making them impractical. The General Purpose computation on **GPU**s (**GPGPU**) is a new programming paradigm which generates new kinds of algorithms that can be processed in the graphics units via new frameworks such as **CUDA** from Nvidia, which provides a new subset of the C language with parallel primitives.

CUDA provides the execution of parallel threads each joined in blocks, each thread has access to a global memory and also some other faster memory spaces such as texture, shared memory cache, which can be used to improve the running time of the algorithm. These features give the possibility of an effective implementation of the **DP** algorithm in the **GPU**. This work was a challenging undertaking as explained by Gong [16].

4.6 Dynamic Programming

Dense depth map calculation is a process (See figure 4.2), in which, given two images I_l and I_r each pixel of I_l is linked to a pixel in I_r. In our case the match can be expressed as a displacement for each pixel $p(x, y) \in I_l$ to a pixel $q(x', y') \in I_r$. The images are previously rectified so that the epipolar lines of the images are coincident with the scan-lines, this indicates that each pixel from the I_l could have their pair in the same line in the I_r and viceversa. We can prove that $y = y'$ so the displacement is $d = x - x'$. The output of the algorithm is a disparity map where for each pixel in I_l we have a disparity value d.

The problem of dense depth map estimation is *NP-hard*. Approximations used to solve it is by a generating cost function for the estimation. This cost function is defined per pixel and the differences between the neighbouring pixels could also be used. The cost per pixel is normally calculated by an aggregation cost function such as **SAD** defined as the sum of the absolute differences of the linked pixels colour in **RGB**. The problem of selecting a good match between the pixels is solved by **DP**. The **DP** approach returns a good match for each scan line of the images, but the differences between scan lines is a typical problem with this method.

The **DP** method is: For each line y in both images I_l and I_r a matrix M_h is created with dimensions $W \times D_{max}$ where W and H is the width and height of the images, and D_{max} is the expected *maximum disparity*. This matrix M_h is filled with the cost corresponding to calculated disparity d for the pixels $p(x, y)$ and $q(x + d, y)$. The cost matrix M_h is calculated with the equation (4.1) where the values of λ are assigned emipiricaly and represent the penalty of the change in the disparity value between neighboring pixels. The aggregation cost **SAD** is calculated between two pixels with the equation (4.2) where p_y and q_y are the

8	10	12	16	18	20	24	26	28	32
7	9	11	15	17	19	23	25	27	31
6	8	10	14	16	18	22	24	26	30
5	7	9	13	15	17	21	23	25	29
4	6	8	12	14	16	20	22	24	28
3	5	7	11	13	15	19	21	23	27
2	4	6	10	12	14	18	20	22	26
1	3	5	9	11	13	17	19	21	25

Table 4.1: Parallel steps for dynamic programming for a image with 10 pixels of width and a calculation of maximum disparity of 8, the number of the maximum number of threads performing calculation are 3. The pattern shows the order of cost calculation in the matrix M_h

values of Red-Green-Blue (**RGB**) colour of the pixels in line y of the currently processed scan lines. A graphical representation of the cost matrix is in the figure 4.1.

$$M_h(x,d) = SAD(x,d) + \mathbf{MIN}(\lambda + M_h(x-1,d-1), \\ M_h(x-1,d), \lambda + M_h(x,d+1)) \quad (4.1)$$

$$SAD(x,d) = \mathbf{ABS}(p_y(x)_r - q_y(x+d)_r) + \\ \mathbf{ABS}(p_y(x)_g - q_y(x+d)_g) + \mathbf{ABS}(p_y(x)_b - q_y(x+d)_b) \quad (4.2)$$

The extraction of the best path, is achieved by a back tracking process, starting in the position $M(W, D_{max})$ and following the minimum path, assigning the disparity value d corresponding to the last position of the path in each dimension of W.

4.7 Parallel DP

Our proposed parallelisation method of **DP** algorithm was based in the **CUDA** framework, which found a parallelisation pattern presented in the table 4.1 of the Matrix M_h. The pattern defines the parallel steps which can be executed simultaneously by the block of threads. This number changes dynamically in the execution of the algorithm, However this is managed by the architecture, along with the global, shared memory and syncronization functions.

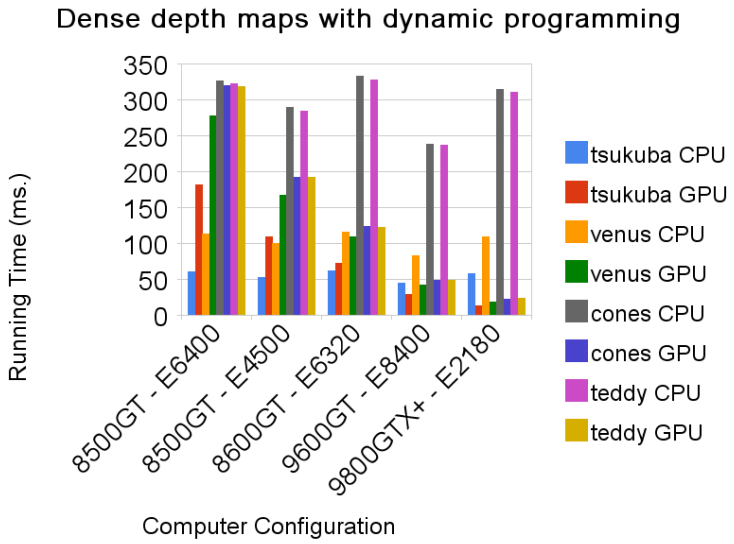
The presented algorithm 2 uses a block of threads to calculated the cost matrix M_h. The size of the block of threads is $(1 \times BX)$, being BX a number between 1 and $\frac{D_{max}}{2}$, in each iteration the threads select an available cell to calculate the cost function, an available cell is a cell which has had all of their required values already computed. The position of the cell is calculated in the

variables c and d. For each cell (c, d), that is different for each thread, the minimum value of the neighbours $(c-1, d+1)$, $(c-1, d)$, $(c, d-1)$ is calculated in t. Then the current cell (c, d) is update with the value of the aggregation cost function between the pixels $p(c, h)$ and $q(c+d, h)$ of the images I_l and I_r and the minimum value of the neighbours t.

The back tracking step is processed after filling the matrix M_h, we calculate the path for each scanline in parallel.

Input: I_r,I_l,W,H,D_{max}
Output: M_h
$h \leftarrow$ **blockIdx.x**
$j \leftarrow$ **threadIdx.y**
for $i \leftarrow 0$ **to** $\frac{W \cdot D_{max}}{\textbf{blockDim.y}} + D_{max} - \frac{\textbf{blockDim.y}}{2}$ **do**
  $c \leftarrow j + \left(\textbf{blockDim.y} \cdot \frac{i-2j}{D_{max}}\right)$
  $d \leftarrow (i - 2j) \mod D_{max}$
  $t \leftarrow +\infty$
  if $(c-1 \geq 0) \wedge (c-1 < W)$ **then**
    if $(d+1 \geq 0) \wedge (d+1 < D_{max})$ **then**
      $t \leftarrow \lambda + \min(t, S[d+1])$
    end
    if $(d \geq 0) \wedge (d < D_{max})$ **then**
      $t \leftarrow \min(t, S[d])$
    end
  end
  if $(c \geq 0) \wedge (c < W)$ **then**
    if $(d-1 \geq 0) \wedge (d-1 < D_{max})$ **then**
      $t \leftarrow \lambda + \min(t, S[d-1])$
    end
  end
  if $t = +\infty$ **then**
    $t \leftarrow 0$
  end
  if $(c \geq 0) \wedge (c < W) \wedge (d \geq 0) \wedge (d < D_{max})$ **then**
    $S[d] \leftarrow$ `AggregationFunc(`c, h, d, I_l, I_r, W, H`)`$+t$
    $M_h[c, d] = S[d]$
  end
  `syncthreads()`
end
return M_h

Algorithm 2: **CUDA** implementation of the **DP** algorithm, the number of running threads in the algorithm must be lower than the half of the D_{max} value.



72dpi, 28.22x21.17 cm, bb=0 0 800 600

Figure 4.3: Running time of the **DP** algorithm in different **CPU** and **GPU** configurations. The difference between the two 8500GT graphic card is because the different clock rates of the cards.

4.8 Results

The results of the algorithm were compared against a **CPU** implementation, as can be seen in figure 4.3. The **GPU** version of the algorithm improves the running time of the algorithm with modern graphics cards. Different processors were used in the test, as can be seen, the evolution of the running time of the algorithm between various types of **CPU** did not generate a substantial difference, but the evolution of the **GPU** implementation shows a significant improvement in the running time of the algorithm.

The images used in the benchmark are the middlebury images of tsukuba, cones, venus and teddy. The time presented is the average running time of the same algorithm in 10 test cycles eliminating the extreme data, and the **DP** algorithm was ran 10 times every test cycle, this was done with the objective of discarding measurement error due to race conditions in the computer and the precision problems with the **PC** clock.

Conclusion

We have presented a parallel implementation for a **DP** algorithm that takes benefits from the **GPGPU** computing model. Our implementation shows a high scalability running on **CUDA** maximising the performance of modern

GPUs. These allows us to implement real-time stereo methods with increasing resolution and precision.

Chapter 5

Conclusion

Solutions for the four geometric problems have been presented in this work. Such solutions combine methodologies presented in different fields of computational geometry, computer graphics and computer vision. The successful application of the implemented solutions to real problems in the industries for which they were built is one important contribution of this work.

The methodologies presented in this work represent and advance in the academic field of the problems with the novel algorithms presented and also the practical implementation in real environments with good results represent a complete cycle of the I+D+I process.

Skills in literature reviewing, scientific rhetoric, paper writing and oral presentation have been developed and strengthened throughout this work.

The valuable interaction with advisors, professors, and researchers at EAFIT University and Institutions abroad was essential in the successful development of this work. It was of particular importance the contact with other cultures, values and working environments.

Bibliography

[1] Salim S. Abi-Ezzi and Srikanth Subramaniam. Fast dynamic tessellation of trimmed nurbs surfaced. *Computer Graphics Forum*, 13("Issue 3"):107–126, 1994.

[2] Eduardo Aguirre. Geometria diferencial de curvas y superficies. Technical report, Departamento de Geometria y Topologia, Universidad Complutense Madrid, 2006. Apuntes de clase.

[3] Marco Attene, Bianca Falcidieno, Michela Spagnuolo, and Geoff Wyvill. A mapping-independent primitive for the triangulation of parametric surfaces. *Graph. Models*, 65(5):260–273, 2003.

[4] Marco Attene, Bianca Falcidieno, Michela Spagnuolo, and Geoff Wyvill. A mapping-independent primitive for the triangulation of parametric surfaces, 2003.

[5] Laurent Balmelli, Christopher J. Morris, Gabriel Taubin, and Fausto Bernardini. Volume warping for adaptive isosurface extraction. In *Proceedings of the conference on Visualization '02*, pages 467–474. IEEE Computer Society, 2002.

[6] M. Botsch and L. Kobbelt. A robust procedure to eliminate degenerate faces from triangle meshes. In *Vision, Modeling and Visualization (VMV01), Stuttgart, Germany, November 21 - 23, 2001*, 2001.

[7] Manfredo Do Carmo. *Differential geometry of curves and surfaces*, pages 1–168. Prentice Hall, 1976. ISBN: 0-13-212589-7.

[8] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, New York, NY, USA, 2001. ACM.

[9] E. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. Technical report, Technical Report CERN CN 95-17, 1995.

[10] Wonjoon Cho, Nicholas M. Patrikalakis, and Jaime Peraire. Approximate development of trimmed patches for surface tessellation. *Computer Aided Design*, 30(14):1077–1087, December 1998.

[11] H. Date, S. Kanai, T. Kishinami, I. Nishigaki, and T. Dohi. High quality and property controlled finite element mesh generation from triangular meshes using multi-resolution technique. *Journal of Computing and Information Science in Engineering*, 5(4):266–276, 2005.

[12] Sarah F. Frisken, Sarah F. Frisken, Ronald N. Perry, Ronald N. Perry, Alyn P. Rockwood, Alyn P. Rockwood, Thouis R. Jones, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. pages 249–254, 2000.

[13] Zvi Galil and Correspondence Kunsoo Park. Parallel dynamic programming. Technical report, Department of Computer Science Columbia University, 1991.

[14] J. Gibson and O. Marques. Stereo depth with a unified architecture gpu. *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pages 1–6, June 2008.

[15] Minglun Gong, Ruigang Yang, Liang Wang, and Mingwei Gong. A performance study on different cost aggregation approaches used in real-time stereo matching. *Int. J. Comput. Vision*, 75(2):283–296, 2007.

[16] Minglun Gong and Yee-Hong Yang. Near real-time reliable stereo matching using programmable graphics hardware. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 924–931, Washington, DC, USA, 2005. IEEE Computer Society.

[17] Akinori Kimura, Yasufumi Takama, Yu Yamazoe, Satoshi Tanaka, and Hiromi T. Tanaka. Parallel volume segmentation with tetrahedral adaptive grid. *ICPR*, 02:281–286, 2004.

[18] A. Klaus, M. Sormann, and K. Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, 3:15–18, 0-0 2006.

[19] R. Klein, A. Schilling, and W. Strasser. Illumination dependent refinement of multiresolution meshes. In *Proceedings of Computer Graphics International (CGI '98)*, pages 680–687, Los Alamitos, CA, 1998. IEEE Computer Society Press.

[20] Přemysl Kršek. Flow reduction marching cubes algorithm. In *Proceedings of ICCVG 2004*, pages 100–106. Springer Verlag, 2005.

[21] T. Lewiner, H. Lopes, A. Vieira, and G. Tavares. Efficient implementation of marching cubes' cases with topological guarantees. *Journal of Graphics Tools*, 8(2):1–15, 2003.

[22] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):169–169, 1987.

[23] Svetlin Manavski and Giorgio Valle. Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment. *BMC Bioinformatics*, 9(Suppl 2), 2008.

[24] Ángel Montesdeoca. Apuntes de geometría diferencial de curvas y superficies. Santa Cruz de Tenerife, 1996. ISBN: 84-8309-026-0.

[25] Bryan S. Morse, Terry S. Yoo, Penny Rheingans, David T. Chen, and K. R. Subramanian. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 78, New York, NY, USA, 2005. ACM.

[26] Timothy S. Newman and Hong Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5):854–879, October 2006.

[27] Carlos Cadavid Oscar E. Ruiz, Miguel Granados. Fea-driven geometric modelling for meshless methods. In *Virtual Concept 2005*, pages 1–8, 2005.

[28] Afonso Paiva, Helio Lopes, Thomas Lewiner, and Luiz Henrique de Figueiredo. Robust adaptive meshes for implicit surfaces. *SIBGRAPI*, 0:205–212, 2006.

[29] S. Palacios and O. Ruiz. Calculo de curvatura en superficies parametricas. A review of Literature on Curvature Measurement, December 2007.

[30] Giuseppe Patane, Michela Spagnuolo, and Bianca Falcidieno. Para-graph: Graph-based parameterization of triangle meshes with arbitrary genus. *Computer Graphics Forum*, 23(4):783–797, 2004.

[31] Scott Schaefer and Joe Warren. Dual marching cubes: Primal contouring of dual grids. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, pages 70–76, Washington, DC, USA, 2004. IEEE Computer Society.

[32] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7–42, 2002.

[33] Raj Shekhar, Elias Fayyad, Roni Yagel, and J. Fredrick Cornhill. Octree-based decimation of marching cubes surfaces. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 335–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[34] J. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1–3):86–95, 2002. citeseer.ist.psu.edu / shewchuk01delaunay.html.

[35] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[36] Jonathan Richard Shewchuk. General-dimensional constrained delaunay and constrained regular triangulations i: Combinatorial properties. Technical report, Department of Electrical Engineering and Computer Sciences University of California at Berkeley, Berkeley, CA 94720, December 2005. To appear in Discrete & Computational Geometry.

[37] C. Shu and P. Boulanger. Triangulating trimmed nurbs surfaces. In *International Conference on Curves and Surfaces, Saint-Malo, France*, pages 381 – 388, May 2000.

[38] Wolfgang A. G. Stöger and Gerhard Kurka. Watertight tessellation of b-rep nurbs cad-models using connectivity information. In *Proceedings of the International Conference on Imaging Science, Systems and Technology, CISST '03, June 23 - 26, 2003, Las Vegas, Nevada, USA, Volume 2*. eds. Hamid R. Arabnia and Youngsong Mun. CSREA Press, 2003.

[39] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda. Classification and evaluation of cost aggregation methods for stereo correspondence. *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[40] Liang Wang, Miao Liao, Minglun Gong, Ruigang Yang, and David Nister. High-quality real-time stereo using adaptive cost aggregation and dynamic programming. In *3DPVT '06: Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, pages 798–805, Washington, DC, USA, 2006. IEEE Computer Society.

[41] Zeng-Fu Wang and Zhi-Gang Zheng. A region based stereo matching algorithm using cooperative optimization. *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[42] Gunther H. Weber, Oliver Kreylos, Terry J. Ligocki, John M. Shalf, Bernd Hamann, and Kenneth I. Joy. Extraction of crack-free isosurfaces from adaptive mesh refinement data. In *Data Visualization 2001 (Proceedings of VisSym '01)*, pages 25–34. Springer Verlag, 2001.

[43] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nistér. Real-time global stereo matching using hierarchical belief propagation. In *BMVC*, pages 989–998. British Machine Vision Association, 2006.