Accurate thickness computation of a B-Rep model on the GPU

Grégoire Lemasson[a,b], Jean-Claude Iehl[a], F. Zara[a], Vincent Baudet[b], Philippe Arthaud[b], Behzad Shariat[a]

[a]Université de Lyon, CNRS, Université Lyon 1, LIRIS, UMR5205, F-69100, Villeurbanne, France
[b]CT CoreTechnologie, F-69007, Lyon, France

[a] firstname.name@liris.cnrs.fr, [b] f.name@fr.coretechnologie.com

ABSTRACT

In this paper, we present a technique that calculates the thickness of a B-Rep model. This calculation is performed directly on trimmed NURBS and not on a triangular approximation. We determine the thickness in parallel on GPU, by computing the radius of maximal spheres contained within the B-Rep model. The results are presented by a color-coded thickness map. A detailed study of our results demonstrates a very important gain in stability, precision and computation time compared to others approaches.

Keywords

Thickness computation, B-Rep model, Parametric surfaces, Trimming curves, Newton iterations, GPU.

1 INTRODUCTION

In Computer Aided Design (CAD), Boundary Representation models (classically denoted B-Rep) are widely used for industrial design, prototyping and production. B-Rep models contain a set of faces, where each face is described by a parametric surface (*i.e.* NURBS) restricted by oriented trimming curves. Outer trimming curves define the shape of the surface with zero or more inner trimming curves defining the holes. To validate a product design, analysis like thickness check are performed. Indeed, during the design of a molded mechanical part, a particular attention should be given to the thickness of the walls. This parameter is a good indicator of the cooling time, giving information on how the material is deposited during its injection in the mold, and thus where weaknesses could appear. Usually, thickness computation is based on the triangle-based approximation (tessellation) of the model. Consequently, in function of required accuracy, a high-quality tessellation is needed and this process is time and memory consuming and cannot output very high precision results.

Thickness [LDBR05, BBS10] is related to the skeleton or the Medial Axis Transform (*MAT*) of the model, defined by the center of the maximal spheres that can be contained inside the volume of the model. Thickness is the value of the radius of these spheres. To define a skeleton or *MAT*, many methods such as voxel based methods [Pal08, PNK12, RC11] have been proposed. They can be easily parallelized on CPU using threads. Mesh contraction methods [WL12, ATC+08, HWCO+13] using points cloud [JKT13, MBC12] or Voronoi diagram, and Delaunay triangulation [RT95, CKM99] are other commonly used techniques. However, these methods are not precise enough for mechanical design.

A direct computation on parametric surfaces, as proposed by Lambourne *et al.* [LDBR05] seems to be an adapted technique. Here, authors proposed a CPU-based sequential method, but they did not take fully into account some trimming configurations. Indeed, consecutive trimming curves could be only C^0 continuous and the trimming process can output erroneous results. Furthermore their method may produce numerical instabilities in GPU implementation, that uses single precision floats. The instability occurs mainly in high curvature surface zones.

In this paper, we present an efficient alternative parallel solution based on Graphics Processing Units (GPU), to compute the locus of the spheres center of B-Rep models. We seek to satisfy two difficult to reconcile conditions: high accuracy and computational speed. A GPU-efficient implementation must respect two main constraints. Firstly, the data must be organized to allow a coherent memory access and a coherent execution on each thread. Secondly, as the implemented method uses newton iterations, care should be taken to prevent divergence of computations. For theses reasons, the NURBS patches are subdivided into relatively flat patches, more suitable for computations using single precision floats. Moreover, to respect the data coherence, we convert NURBS into patches with the same degree. A similar conversion is applied to the trimming curves.

In summary, the main contributions of this work are:

- proposing a thickness computation method on GPU,

- guaranteeing the stability of numerical computations despite of the use of single precision floats,

- taking fully into account trimmed parametric surfaces on GPU.

This paper is organized as follows. First, in section 2, we present in detail our approach for the computation of the maximal sphere. In section 3, our results are presented and fully discussed. and finally, section 4 presents the conclusion and the perspectives of our work.

2 MAXIMAL SPHERE COMPUTATION

As stated before, the calculation of the thickness consists in finding the radius of the maximal spheres that are at least bi-tangent to the boundary of the model.

A CAD model could be defined with more than several tens of thousands of restricted parametric surface with heterogeneous definitions. They may be NURBS from degrees 2 to 5 or more, or constructive surfaces such as blends or offsets, etc. Consequently, for the sampling, as a first step, we have to convert theses surfaces into several homogeneous bicubic Bézier patches using a tolerance of 10^{-4} [PT97]. This high tolerance is used to allow a good approximation of the original surfaces and to have a set of relatively flat patches. The relative flatness ensures a better convergence of the Newton iteration. Similarly to the surfaces, the curves must be converted into monotonic segments [SF09]. Then, we produce a dense sampling of the B-Rep model, generating a points set called s_0. The opposite of the normal direction at each sampled point is computed. Moreover, we perform a coarse sampling of the B-Rep (less dense than s_0), called s_1.

Spheres computation is carried out in two steps. The first step called initialization, finds an approximation of the maximal sphere: For each point O in the set of points s_0, the smallest sphere that tangentially touches O, and is in contact with one other point P belonging to the points set s_1 on the model boundary should be approximated (Fig. 1).

The second step is to refine the approximated sphere using Newton iterations. In this step, we compute spheres that are bi-tangent at both points. Two configurations may exist:

- sphere bi-tangent to two surfaces: surface/surface sphere (*cf.* Fig. 2),

- sphere tangent to one surface and an edge shared by two surface patches: surface/edge sphere (*cf.* Fig. 4).

In our approach, we firstly compute the maximal sphere between two patches (cf. section 2.2). In case of computational failure (when the sphere computation have converged on the outer of the trimming of the surface), a maximal sphere between an edge and a surface is searched (cf. section 2.3). These steps are explained hereafter. The algorithm 1 show our approach.

Algorithm 1 Computation of the maximal sphere

```
for all O ∈ s0 parallelized on GPU do
    compute maximal surface/surface sphere
    if sphere is outer the trimming then
        compute maximal edge/surface sphere
    end if
end for
```

2.1 Initialization

Each point $O \in s_0$ is defined by its coordinates and $\vec{d}$ the opposite direction of the surface normal at O. The second point P belongs to s_1 (full dots in the figure 1).

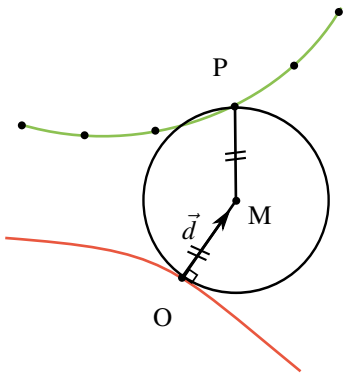


Figure 1: Computation of the approximated sphere.

The center M of the approximated sphere based on the points O and P is defined by $M = O + t\,\vec{d}$, with $t = ||\vec{MP}||$. As in Lambourne *et al.* [LDBR05], whatever P is, t can be defined as following:

$$t = \frac{||P-O||^2}{2\vec{d}.(P-O)} \tag{1}$$

with . being the dot product. This equation can result in spheres lying outside the object, when $\vec{d}.(P-O) < 0$. Hence, for each point O, a good initialization consists in seeking a point P among a points set s_1, such that the resulting $t(P)$ is strictly the smallest positive distance. The identification of P gives another information that is the parametric coordinates of P on the surface S_1 (*i.e.* $P = S_1(u,v)$).

To accelerate the computation of the approximated sphere, the set s_1 is stored into a Bounding Volume Hierarchy (BVH), using Axis Aligned Bounding Box (AABB) of s_1. The aim of the BVH is to find

the minimal approximated sphere among a subset of points, relatively close to the real minimal sphere. For example, considering a point O with an approximated sphere of radius t, the center of the sphere is $M = O + t\vec{d}$. If a smaller sphere exists, it is necessarily computed with a point in s_1 such that the distance between M and P is inferior to $2t$ (the diameter of the current sphere). By consequence, during the traversal of the BVH, we compute the distance of M with the AABB of the BVH node (when the point M is inside the AABB, the distance is null). Only the nodes for which the distance to M is inferior to $2t$ are traversed. At the lowest level of BVH (the leaves), the value of t is updated if a smaller computed radius is found (according to equation 1).

The algorithm 2 summarizes the whole initialization process implemented on GPU.

Algorithm 2 Initialization of the approximated sphere

for all $O \in s_0$ parallelized on GPU **do**
  Find $P \in s_1$ using BVH for which the sphere is minimal
end for

2.2 Maximal sphere between two surfaces

Figure 2 shows a maximal sphere S tangent to the surface S_O at a point O and tangent to the surface S_1 at the point P. Lambourne *et al.* [LDBR05] used a Quasi-Newton method to minimize a system and obtain the maximal sphere, using the equation 1. We propose a more suitable system for a GPU implementation, insensitive to numerical instability. Our solution uses Newton iterations with an adapted initialization. For this, we define the constraints that should be respected by the maximal sphere:

$$\begin{cases} \overrightarrow{MP} \perp S_{1u} \\ \overrightarrow{MP} \perp S_{1v} \\ t = ||\vec{MP}|| \end{cases} \quad (2)$$

with (u,v) a point in the parametric domain of the surface, and S_{1u} and S_{1v} the first derivatives in u and v directions of surface S_1.

Newton iterations.

In order to solve the above geometrical constraints, we use Newton's method:

$$X_{n+1} = X_n - J^{-1}(X_n).R(X_n), \quad (3)$$

where R is the root to be found ($R = 0$), J is the Jacobian matrix of R, and X_n is the vector of parameters to be determined.

With $P = S_1(u,v)$, we propose to solve the system presented in equation 2 with the following equation:

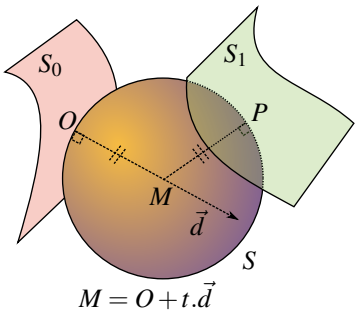


Figure 2: Computation of thickness using a sphere. The thickness between S_0 and S_1 is the length of the radius of the sphere, *i.e.* $||\vec{MP}||$ or $||\vec{MO}||$.

$$X_n = \begin{bmatrix} u_n \\ v_n \\ t_n \end{bmatrix}, R(X) = \begin{bmatrix} \vec{S_{1u}}(u,v).(S_1(u,v) - (O + t\vec{d})) \\ \vec{S_{1v}}(u,v).(S_1(u,v) - (O + t\vec{d})) \\ t - ||S_1(u,v) - (O + t\vec{d})|| \end{bmatrix} \quad (4)$$

Newton iterations are performed while $||R|| > \varepsilon$, with ε user-defined value, generally set to 10^{-4}. Statistically in our experiments, at most 15 iterations are sufficient to obtain the required precision.

2.3 Maximal sphere between a surface and an edge

In some configurations a maximal sphere could be tangent to the surface S_0 and the edge defined by two surfaces, S_1 and S_2 as illustrated in Fig. 4. This configuration appears when the computed point P is outer the surface S_1 (*cf.* Fig. 3.). The intersection between the surface S_1 and the surface S_2 is represented by an edge (C_{3D}).

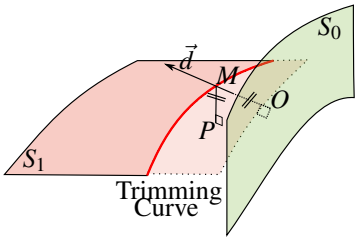


Figure 3: A surface/surface sphere computed outer the trimming.

For example in the Fig. 4, $C_{3D} = C_1 \in S_1 = C_2 \in S_2$. Then, we can obtain the parametric coordinate u of the point P on C_{3D} with $P = C_{3D}(u)$.

Obviously, the maximal sphere between the surface S_0 at the point O, and the curve C_{3D} should respect the following constraints:

$$\begin{cases} \overrightarrow{MP} \perp C'_{3D} \\ t = ||\vec{MP}|| \end{cases} \quad (5)$$

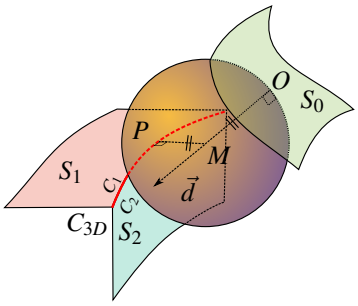


Figure 4: The spherical thickness between a surface and a 3D curve.

with C'_{3D} is the first derivative of C_{3D} at the point P. Newton iterations are also used to determine the maximal sphere.

2.3.1 *Detection of edge/surface sphere computation using trimming curves.*

To determine if the previous surface/surface computation has failed, the location of the projection of the center of the sphere on S_1 must be checked according to the trimming curves. For example, in the Figure 5, M is the center of the surface/surface sphere tangent to the surface S_1 at the point P. But this latter is located on the trimmed portion and therefore the surface/surface sphere is inadequate.

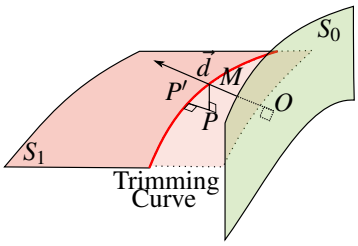


Figure 5: A point P lies on the trimmed portion. Point P is projected on the trimming curve.

The point P is calculated during the computation of surface/surface sphere. Then, we have $P = S_1(u,v)$, where u and v are the results of the first step (*cf.* equation (4)). Thus, to determine if P is outside the trimmed surface, we must check if the parameters u and v are outside the trim boundary of S_1. We use the cross product $\times$ to determine if the point is onside or outside of the trim boundary. Let $p_c = c(t_p)$ be the projection of P on the trimming curve. If $(\overrightarrow{Pp_c} \times c'(t_p))_{.w} > 0$ (with $c'(t)$ the first derivative of $c(t)$, p_c the nearest point on $c(t)$ and w the third coordinate of the vector), the intersection is inside the shape (see Figure 6).

Clearly if the point p_c is the nearest point on $c(t)$ to P then $\overrightarrow{Pp_c} \cdot c'(t_p) = 0$. Consequently, we use Newton iterations with (as in [Sch90]):

$$t_{n+1} = t_n - \frac{(c(t_n) - P) \cdot c'(t_n)}{c'(t_n)^2 + c''(t_n) \cdot (c(t_n) - P)} \qquad (6)$$

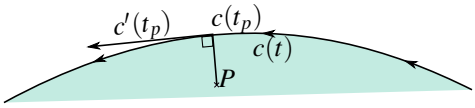


Figure 6: Intersection point P inside the trim boundary with $(\vec{Pp_c} \times c'(t_p))_{.w} < 0$.

where t_n is an approximation of the parametric coordinate t_p at the n^{th} iteration, and $c''(t)$ is the second derivative of $c(t)$. In this case, the system converges after three iterations. The first approximation of t_n (*i.e.* t_0) is obtained by the projection of the point P on a linear approximation of the trimming curve, $c(t)$. The linear approximation is a polyline passing through the points $c(0), c(1/3), c(2/3)$ and $c(1)$.

Corner

The trimming loop is a set of consecutive trimming curves, and in some cases the point P can be located at the intersection of two consecutive curves. If these consecutive trimming curves are not at least C^1 continuous at their junctions, a continuity break corner can appear (see Figure 7). In this case the location of the point P is uncertain.

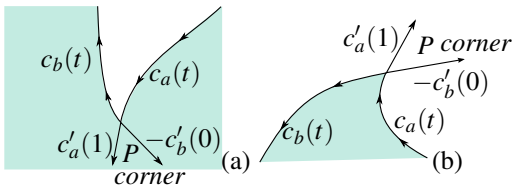


Figure 7: Illustration of *corner*: a) concave or b) convex. The colored zone is located within the face (non trimmed portion).

If the intersection P is in a corner, P is nearest to the curve $c_a(t)$ at the point $c_a(1)$, and nearest to the curve $c_b(t)$ at the point $c_b(0)$, but according to the curve, P can be detected inside or outside the face.

To solve this problem, we defined two categories of corners: concave and convex corners. When the third coordinate of the cross product $(c'_a(1) \times c'_b(0))_{.w} < 0$, P is inside a concave corner and consequently within the face (see Figure 7-a). Reciprocally, if $(c'_a(1) \times c'_b(0))_{.w} > 0$, P is inside a convex corner and consequently outside the face (see Figure 7-b).

2.3.2 *Edge/surface computation*

During the trimming computation, the projection of the point P on the trimming curve C_{3D} (the point P' in the Figure 5) is computed with $P' = C_{3D}(u)$. The u parameter is used in the next step to initialize the system of equations (5).

Newton iterations. The system presented in equation (5) is then solved using Newton iterations (*cf.* equa-

tion 7) to obtain the maximal sphere defined by parameters u and t.

$$X_n = \begin{bmatrix} u_n \\ t_n \end{bmatrix}, R(X) = \begin{bmatrix} \vec{C'_{3D}}(u).(C_{3D}(u) - (O + t\vec{d})) \\ t - ||C_{3D}(u) - (O + t\vec{d})|| \end{bmatrix} \quad (7)$$

Generally, a maximum of 20 iterations are performed to obtain an accurate value.

3 RESULTS

We compared the time performance of our method executed on a graphic card Nvidia GeForce GTX 580 M with the OpenCL language (`http://www.khronos.org/opencl/`), with the thickness checker of 3D Evolution software package, an industrial software developped by CT-Coretechnologie (`http://www.coretechnologie.de`). Three geometrical models called *Lens*, *Mask* and *Mask1* of the rear light of a car were used (see Figure 8 and Table 1). The model *Mask1* is a part of the model *Mask*. It is a good compromise between complexity and data size.

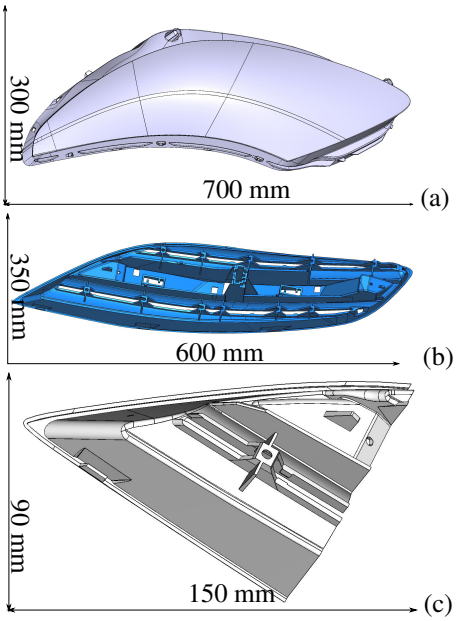


Figure 8: (a) The Lens model. (b) The Mask model. (c) The Mask1 model.

	Lens	Mask	Mask1
# bicubic patches	88,581	27,188	12,793
# cubic trimming curves	27,793	12,793	4,586

Table 1: Number of patches and trimming curves in the three used B-Rep models

Speed-up. Figure 9 shows the speed-up obtained between the thickness checker of 3D Evolution and our GPU-based thickness technique for the three above objects. The more the sampling is dense, the more the acceleration with respect to Evolution is important. For *Mask1* model, (more than 3.5 millions sampling points) the acceleration is more than 100 times faster than 3D Evolution industrial software. This speedup can be explained by two main reasons. When the number of points is large: (1) all the threads on the GPU are used; (2) consecutive points are stored in the neighbouring threads.

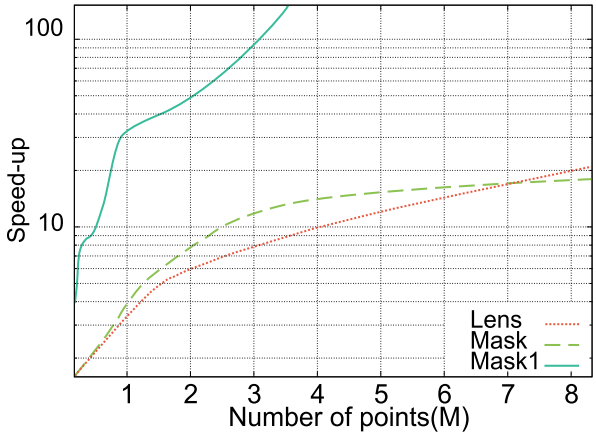


Figure 9: Performance comparison between 3D Evolution and our method.

Computation time. Figure 10 shows the computation time in seconds for the three above B-Rep models using several millions of threads. Our thickness computation exhibits a linear complexity behavior. Indeed, only the number of spheres increases while the size of the sampling used for the initialization remains constant.

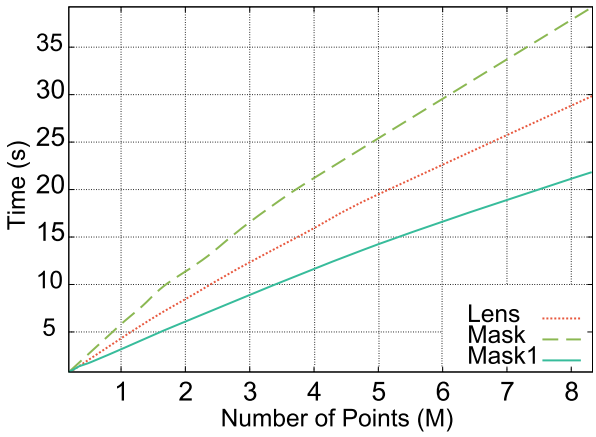


Figure 10: Computation time for the three models: *Lens*, *Mask* and *Mask1*.

Accuracy. The accuracy depends on the number of iterations of the Newton's method. In our work, we used a fixed number of iterations to have a coherent execution. We can compute the average accuracy of the method using $||R||$ (*cf.* section 2.2) where for $||R|| = 0$ the absolute precision is obtained. In average, the precision of surface/surface computation is around 10^{-6}(see Figure 11. The average precision for the curve/surface calculation

is around 10^{-6} mm. (see Figure 12). With this pre-

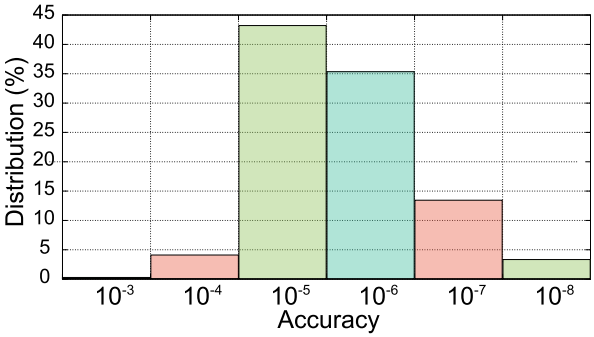


Figure 11: The distribution of the surface/surface Newton iterations precision.

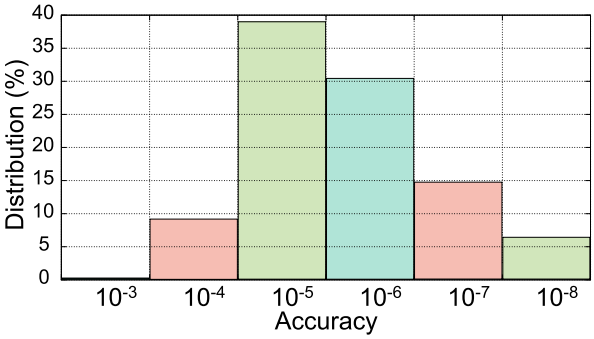


Figure 12: The distribution of the 3Dcurve/surface Newton iterations precision.

cision, the Newton iterations ensure that the sphere is maximal and it is tangent to two surfaces, or to a surface and a curve. In Figure 13, we can see different maximal spheres computed with our thickness method. The accuracy on 3D evolution depend of the model tessellation, so to achieve the same accuracy as our method would require a very fine tessellation.

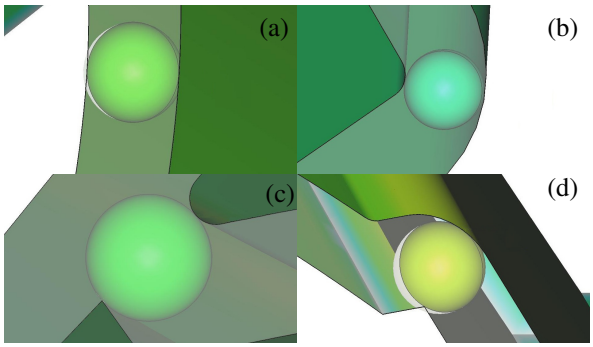


Figure 13: Different results of our method. a) and b) show a maximal sphere between two surfaces. c) and d) show the maximal sphere between one curve and one surface.

The thickness results of our approach are represented by color coded thickness map (Figure 14) illustrates a very good behaviour of our algorithms.

Memory consumption. Figure 15 illustrates the memory consumption of our approach. Whatever the num-

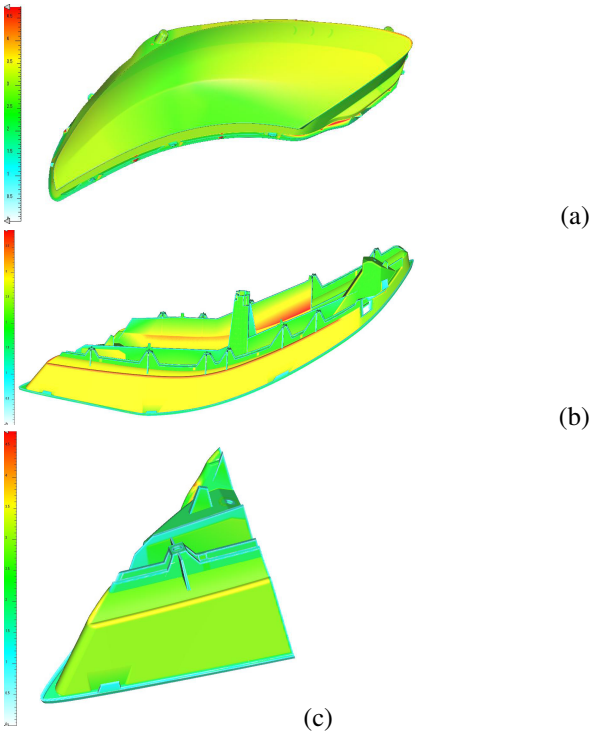


Figure 14: Colour coded thickness map for *Lens* (a), *Mask* (b) and *Mask1* (c) model. White color represents a radius equal to zero. Red color represents the maximum radius of the spheres. The colors in ascending order are: white, blue, green yellow and red.

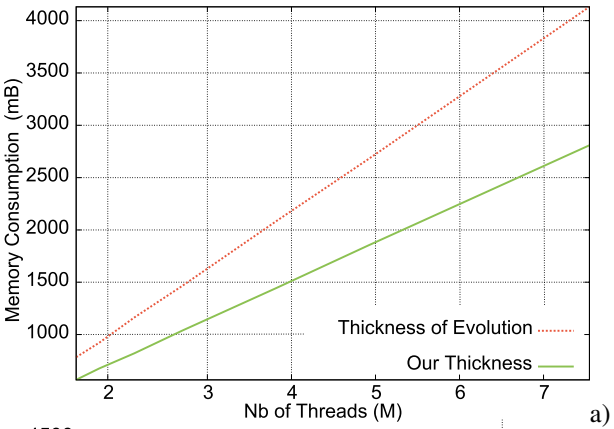
ber of threads is, this memory consumption does not change. The Figure 16 shows the total memory consumption according to the number of spheres for *Lens*, *Mask* and *Mask1* models. This figure shows that the GPU version is less memory consuming than the CPU version (3D Evolution).

Model	BVH	Patches	Trimming curves	Total
Lens	12.94	23.50	1.45	37.90
Mask	5.38	9.69	0.69	15.76
Mask1	0.43	0.82	0.25	1.5

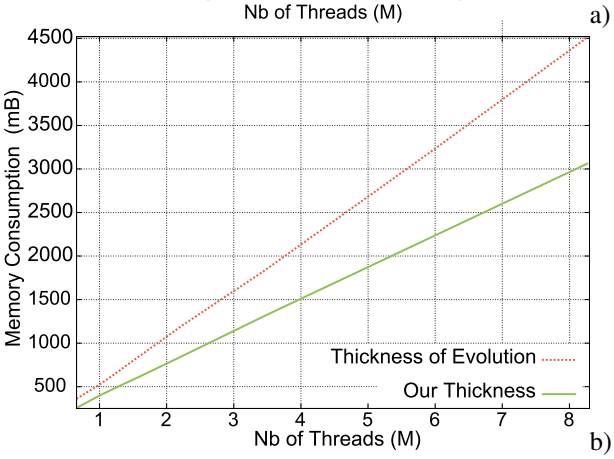
Figure 15: Memory consumption in Mb

Comparison with Lambourne's method. We also implemented the method presented in [LDBR05] on a core i7-280Qm 2.30 GHz processor to compare it with our method. Performance differences between the two methods for the three *Lens*, *Mask* and *Mask1* models were relatively constant and our method is on average 18 times faster. This quasi-constant acceleration factor can be explained by the similarity between the two methods.

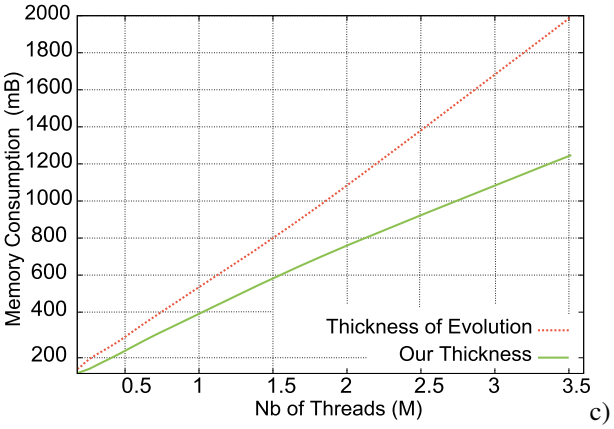
To compare both methods on the same basis, we also implemented Lambourne's method on the GPU. However, this method is not directly implementable on the GPU. Indeed, the use of NURBS of variable degree



a)



b)



c)

Figure 16: The memory consumption in mB according to the number of threads for the thickness of Evolution and our thickness computation on the model *Lens* (a), *Mask* (b) and Mask1 (c).

does not respect data consistency or instructions coherences of GPU programming. Moreover, the interrogation of certain high degree surfaces can induce numerical instability, when using 32-bit floats. Consequently, we converted all surfaces into bicubic Bezier patches.

Lambourne *et al.*used a Quasi-Newton method to minimize their system. As stated before, this system implemented on GPU can produce instability. The Figure 17 shows color coded thickness map output by both methods. We can observe that in the left image a set of red dots that does not appear on the right image, produced by our approach. These points represent the non convergence of the computations. This instability is due to the second derivative terms of the Hessian matrix. Indeed the second derivatives are subject to instability. On the contrary, our method use first derivatives for the Jacobian matrix that are simpler to compute and more stable.

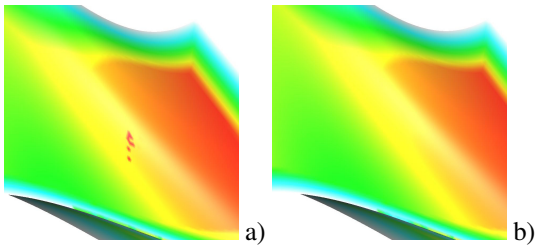

a) b)

Figure 17: Comparison of the results. a) Lambourne *et al.*, b) our Method.

4 CONCLUSIONS

In this paper, we have presented a thickness computation using maximal spheres on a B-Rep model. This computation uses the massively parallel architecture of the GPUs. This permitted to improve existing methods, by obtaining faster, more stable and more precise results with single precision floats.

We also presented a method for regions containing holes, and applied this technique for direct trimming of parametric surfaces. This method provides very good results even if the consecutive trimming curves are only C^0 continuous. In summary, this method can be applied to all parametric curves, and does not need complex pre-computations.

The maximal spheres centres represent a sampling of the skeleton of the object. This can help the computation of the mid-face or neutral fibre of the object, commonly used in finite element analysis. For this, in a near future, we plan to refine the medial axis transform in the branching zones, where a tri-tangent calculation is necessary, and to suppress unwanted skeleton branches.

5 REFERENCES

[ATC+08] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, 27(3), 2008.

[BBS10] Terence M. Bahlen, Willem F. Bransvoort, and Allan D. Spence. Extraction and visualization of dimensions from a geometric model. *Computer-Aided Design & Applications*, pages 579–589, 2010.

[CKM99] Tim Culver, John Keyser, and Dinesh Manocha. Accurate computation of the medial axis of a polyhedron. In *Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications*, pages 179–190, New York, NY, USA, 1999. ACM.

[HWCO+13] H. Huang, S. Wu, D. Cohen-Or, M. Gong, H. Zhang, G. Li, and B.Chen. L1-medial skeleton of point cloud. *ACM Transactions on Graphics*, 32:65:1–65:8, 2013.

[JKT13] Andrei C. Jalba, Jacek Kustra, and Alexandru Telea. Surface and curve skeletonization of large 3d models on the gpu. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(6):1495–1508, 2013.

[LDBR05] J. G. Lambourne, Z. Djuric, D. Brujic, and M. Ristic. Calculation and visualisation of the thickness of 3d cad models. In *Shape Modeling and Applications, 2005 International Conference*, pages 338–342, 2005.

[MBC12] Jaehwan Ma, Sang Won Bae, and Sunghee Choi. 3d medial axis point approximation using nearest neighbors and the normal field. *The Visual Computer*, 28(1):7–19, 2012.

[Pal08] Kálmán Palágyi. A 3d fully parallel surface-thinning algorithm. *Theor. Comput. Sci.*, 406:119–135, 2008.

[PNK12] Kálmán Palágyi, Gábor Németh, and Péter Kardos. Topology preserving parallel 3d thinning algorithms. In Valentin E. Brimkov and Reneta P. Barneva, editors, *Digital Geometry Algorithms*, volume 2 of *Lecture Notes in Computational Vision and Biomechanics*, pages 165–188. Springer Netherlands, 2012.

[PT97] Les Piegl and Wayne Tiller. *The NURBS Book (2Nd Ed.)*. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[RC11] Benjamin Raynal and Michel Couprie. Isthmus-based 6-directional parallel thinning algorithms. In *Proc. of the 16th IAPR international conference on Discrete Geometry for Computer Imagery*, pages 175–186, Berlin, Heidelberg, 2011. Springer-Verlag.

[RT95] Jayachandra M Reddy and George M Turkiyyah. Computation of 3d skeletons using a generalized delaunay triangulation technique. *Computer-Aided Design*, 27(9):677 – 694, 1995.

[Sch90] Philip J. Schneider. Graphics gems. In Andrew S. Glassner, editor, *Graphics gems*, chapter An algorithm for automatically fitting digitized curves, pages 612–626. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[SF09] Andre Schollmeyer and Bernd Fröhlich. Direct trimming of nurbs surfaces on the gpu. In *ACM SIGGRAPH'09*, pages 47:1–47:9, New York, NY, USA, 2009. ACM.

[WL12] Chris G. Willcocks and Frederick W. B. Li. Feature-varying skeletonization: Intuitive control over the target feature size and output skeleton topology. *Vis. Comput.*, 28(6-8):775–785, 2012.