High Performance generalized cylinders visualization

Laurent Grisoni Damien Marchal

LIFL, University of Lille 1, UPRESA CNRS 8022, Batiment M3
59655 Villeneuve d'Ascq Cedex, France
[Laurent.Grisoni|Damien.Marchal]@lifl.fr

Abstract

This paper studies the problem of highly deformable generalized cylinders real-time rendering. Some efficient schemes for high axis curvature detection are presented, as well as an incremental non-uniform sampling process. We also show how the recent 3D card "skinning" feature, classical in character animation, can be derived in order to allow for very high frame-rate when rendering such generalized cylinders. Finally, an algorithm is presented, that permits the object to dynamically adapt its display process, for guaranteed frame-rate purposes. This algorithm dynamically modifies the different sampling parameters in order to achieve optimal quality visualization for a given pre-imposed frame-rate.

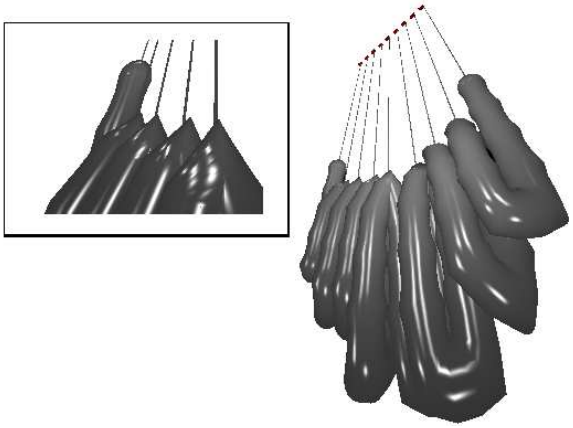

Figure 1. classical uniform tessellation of a simple generalized cylinder. Top-left: zoom on axis high-curvature area.

1. Introduction

Widely used in computer graphics modeling, generalized cylinders are now quite commonly used. Since their original definition [3], several declinations of the original model have been presented, and extensively studied [8, 9, 14, 19]. One major advantage of such a model is that a 3D shape is conceptually simplified into several 1d-curves, and the process of such a shape design, simplified to several 1d-function definition [8]. This allows for efficient design of topologically simple objects. One of the major problems often encountered with such models is that of conversion to polygonal approximation (this process is called *tessellation*), e.g. for further real-time visualization. It is well known that simple poly-line extrusion raises tessellation problems for high-curvature points (see Figure 1 for an example of it).

This comes from two main problems when handling high-curvature points on the cylinder axis: first, it may occur that the sharp angle is simply missed in the sampling. The second problem is the fact that tessellated sections can overlap, hence providing the user with self-intersecting mesh (see figure 2 for illustration of those two problems).

This paper presents three contributions to the problem of high-performance visualization of such objects: first, we present an incremental technique for non-uniform sampling of the cylinder axis. Second, we introduce another way of constructing the polygonal approximation used for fast visualization. Last, we present an algorithm that allows for the rendering process to adapt itself to a specific frame rate, which can be quite useful when designing graphically complex applications that might possibly be executed on a computer with limited rendering possibilities.

The paper is organized as follows: section 2 defines the notation used in this paper, as well as classical techniques for high-curvature detection along the axis, and generalized cylinder rendering. Section 3 presents our incremental solution for adaptive sampling of the axis. Section 4 shows how hardware-based skinning can be extended to a subclass of generalized cylinders, and significantly reduce

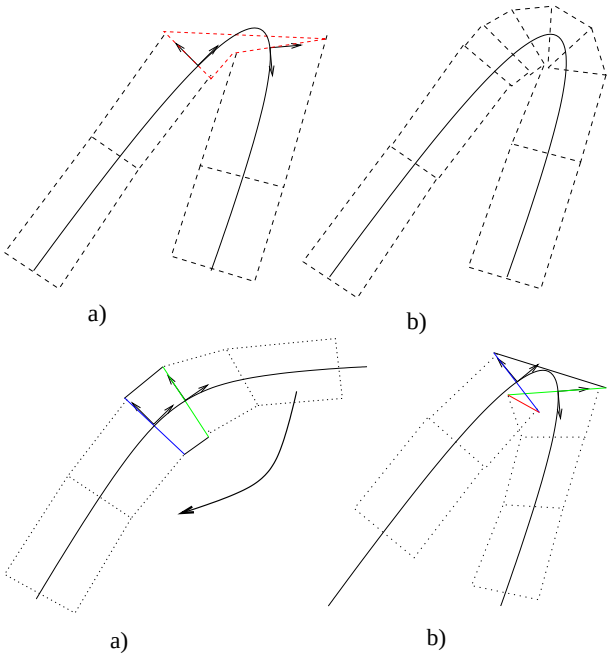

Figure 2. classical tessellation problems on generalized cylinders. Top line: inadequate sweep tessellation (left) and correct one (right). Bottom line: mesh self-intersection when deforming initial sweep.

the software-side computation, to the benefit of hardware-side, in the global rendering process. Section 5 discusses the guaranteed frame rate constraint, and presents a technique to automatically adapt the different parameters of the display process so that rendering respects a given frame rate. Finally, in section 6, some results, along with measures, are presented.

2. Generalized Cylinders

We do not plan to completely define sweeps, as they are quite a classical graphics modeling tool. We refer the reader to [8] for a classification, and extensive terms definition. In this section, we first simply set the notations we will use in this article, and give the limitation of the techniques described in this article. Second, we present classical techniques of sweep tessellation, including non-uniform axis sampling, and multiresolution techniques.

2.1. Definition and limitations

A generalized cylinder G is primarily defined by two simple analytic objects: an axis $C(u)$ (an arbitrary parameterized curve, e.g. a spline, or subdivision curve) and a cross section function $S(u)$, a planar shape the definition of which depends on the u value. For each value of u, the intersection of G with the plane orthogonal to curve C is exactly the cross section $S(u)$. We will refer to rotational primitives as the special objects where $S(u)$ is isotropic, i.e. equivalent to a circle of radius $r(u)$ (which is called profile function). Twists define G using a rotation of angle $\theta(u)$ (this function is the twisting function) around the first derivative vector $C'(u)$ direction, that is applied to an initial cross section curve S to give $S(u)$ (see e.g. [10] for simple examples). Interpolational cross-section objects are objects for which cross-sections are defined using key sections defined for key parameter values. Intermediate sections are generated as interpolation between the key cross sections [10].

Our work is primarily devoted to the (somewhat classical) case where $S(u)$ can be seen as the image of an original section S, combined with a profile curve $r(u)$, and a twisting function $\theta(u)$. Sampling technique can be generalized to interpolational objects, but hardware based rendering technique as described in this article is not correct anymore for such generalized cylinders (see section 4 for precise explanation of such a limitation). For the remainder of this article, we hence consider that our sweep is constructed using an axis $C(u)$, a cross-section S, a profile curve $r(u)$, and a twisting function $\theta(u)$.

2.2. Classical tessellation process and axis sampling techniques

Classical sweep tessellation process involves an iterative treatment. Some polygonal approximations of cross-sections $S(u)$ are created for several key u values, and two consecutive sections generate a stripset mesh information, that locally approximates the sweep. The core point of such a process is the question of the positioning of the key frames along the axis. Bloomenthal [6] presents efficient numerical techniques for calculating the reference points, and get rid of hypothetical twists of Frenet basis around the axis, that could introduce undesired and uncontrolled twists on the resulting object. The other point, about key frames positioning, is the selection of adequate parameter u values, in order to position the sampling points along the axis. Typically, it involves studying the analytic structure of axis $C(u)$, in order to isolate maximal curvature points, and give proper tessellation of the sweep. Spline curves are quite a common way to achieve axis definition [19]. For such analytic structures, many different tools exist in regard of high-curvature point isolation: interval analysis [22, 17, 4], symbolic root-isolation [11], wavelet decomposition using semi-orthogonal or biorthogonal B-spline multiresolution analysis [18, 16]. Apart from simple oversampling, inter-

val analysis provides quite an efficient way for fast partitioning of curve into parts of different interests in regard of curvature [4]. Symbolic root finding is also efficient in spline context: in most cases, spline basis functions are low degree piece-wise polynomials, or rational polynomials, and are good candidates for such numerical techniques. Wavelets provide a good theoretical framework for curve analysis, they are very flexible, and are a good way to detect high-curvature points. Yet, they are quite computation expensive, and not affordable in a high-performance rendering framework. Moreover, only B-splines and NURBS are provided with such tools, what makes it not available for all spline models. Some heuristic techniques have to be found for determining which point can be ignored when willing to simplify a curve, and the determination of such heuristic is more or less simple, depending on the B-spline structure one uses (we refer the reader to [16, 15] for a more detailed explanation of this technical point).

It is to note that curves defined using subdivision processes can somewhat take advantage of the same tools as splines, as it is possible, using theoretical impulse response of the subdivision filters, to consider subdivision curves in a formalism quite similar to that of splines.

Next section discusses the algorithm we used for efficient spline high-curvature detection, and the associated axis sampling process.

3. High curvature detection and adaptive sampling

The underlying idea behind the sampling process we discuss here is somewhat fairly simple. Most interesting spline models have the regularity property [21]. This analytic property has among its consequences that no undesired oscillation appears in a given spline segment. In other words, a curve overall shape can somewhat be pre-determined by studying the spline control points configuration. In this section, we consider a spline axis C. In our tests, we used uniform cubic B-spline axis, but most of the technique can be easily generalized to many other spline cases, as most of the spline properties used remain available for most of classical models.

3.1. Algorithm description

the sampling process uses the control points sequence $(P_0, P_1, \ldots, P_n)$ of C, the corresponding key parameter values $(u_0, u_1, \ldots, u_n)$ (e.g. knot vector for B-splines model), and some threshold value ε, as entry variables. The sampling algorithm is split into two steps:

- A sequence of scalar values $\{c_i\}_{i=0\ldots n}$ is defined using (in the following equation the notation . stands for the standard euclidian scalar vector product):

$$c_i = \overrightarrow{P_{i-1}P_i}.\overrightarrow{P_iP_{i+1}}$$

This sequence is then normalized:

$$\forall i, c_i \longleftarrow \frac{c_i}{max_{k=0\ldots n}(|c_k|)}$$

.

Extremal index i values are treated using ghost points defined using Bessel technique [12]. Each spline segment connection is hence associated to a numerical value c_i that measures the local control point "perturbation". It is to note that, depending on the spline model one uses, extremal indices for the $\{c_i\}_{i=0\ldots n}$ sequence might have to be slightly modified (e.g. extremal B-splines): such possible modifications can be easily done without loss of generality. Figure 3 shows a visual evaluation of this step, associating different color to each segment, depending on the numerical values obtained for scalar product.

Figure 3. An example of spline segment coloration using our curvature isolation process, during interactive sweep deformation. Darker segments correspond to those that the algorithm selects as high curvature ones.

- For all the parameter segments $[u_i, u_{i+1}]$, if $|c_i| < \varepsilon$ and $|c_{i+1}| < \varepsilon$, then the spline segment $[u_i, u_{i+1}]$ is treated as a whole axis sample by display (see section 4.2 for details about this treatment). In any other case, a symbolic root extraction is performed. In the case of uniform cubic B-splines, the maximal curvature parameter point on a segment satisfies to a linear equation. For all piecewise polynomial splines, such an

equation can be written; in most practical cases, the polynomial spline degree is low enough so that such equation can be solved symbolically. Once this maximal curvature m_i parameter point is calculated, then the parameter segments $[u_i, m_i]$ and $[m_i, u_{i+1}]$ are displayed, according to the technique described in section 4.2.

3.2. Comments

The regularity property ensures that high curvature segments are detected within the first pass of the process. It only involves simple and fast computation, and fits requirement for real-time computation. The second pass positions sampling points where necessary, in order to solve the problems mentioned on Figure 2. This sampling process is somewhat quite similar to that described in [11]. The main difference lies in the simplification done in order to have high-performance, relaxing theoretical validations of the sampling scheme: in particular, our scheme does not guarantee the minimality of the sample number in regard of any error criteria. It simply ensures that no high curvature point will be missed. This process makes that a spline is turned into a set of particularized points along the curve: all the parameter values corresponding to segment extremities, and additional parameter values m_i where high-curvature has been isolated. the resulting parameter segments are displayed using hardware-based technique, described in the next section.

4. Hardware based rendering using skinning

The tessellation technique we use is based on the skinning extension of most recent 3D cards [2].

4.1. Skinning principle

This OpenGL extension (sometimes called *vertex blending*) is classically devoted to character deformation, and is presented by classical documentation as a tool that allows for continuous deformation of a mesh defined as a skin over an animated skeleton. The principle of this extension is the following: it is classical to define a simple geometric transformation, using some 4x4 matrix, that defines a composition of rotation, translation, and scale, and allows for an initial mesh to be positioned anywhere within a given scene, along with simple deformations. Skinning uses, for an object, two matrices (i.e. two different positions of the object), and for all the vertices of the object, a scalar value (called *weight*) that is used to interpolate the vertex position. Precisely speaking, given an initial vertex ν^0, of weight $\omega(\nu^0)$, and two transformations M_1 and M_2, the resulting position ν of the vertex that is used for display is given by :

$$\nu = \omega(\nu^0)M_1\nu^0 + [1 - \omega(\nu^0)]M_2\nu^0$$

.

Such transformation is done by hardware, given the two matrices for a given mesh, and the weights for the mesh vertices. No CPU computation is involved in this process. Setting optimal weights for a given deformation is, in general, a difficult question. Bloomenthal presents a technique for automatically setting weights when skinning is used to character animation [7]. Such a transformation process allows for simple deformation of initial mesh. Figure 4 shows a simple 2D example of what is obtained using initial rectangle deformation, using weights that only depend on the position z along the rectangle axis. On this figure, the weight function has the same structure as blending functions described for other purposes in [5]. Controlling the weight distribution curve gives control on the way the 3D card will interpolate from the first transformation to the other.

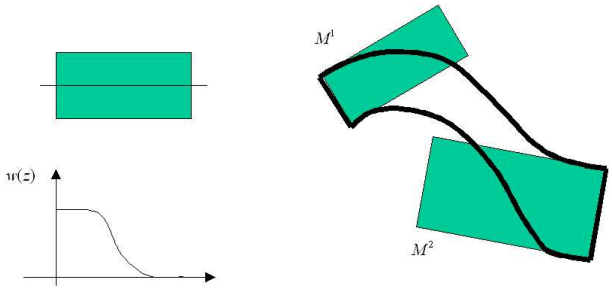

Figure 4. Simple skinning application to a simple polygonal primitive. Combination of M_1 and M_2 geometric transformations on the object on top-left, using the weights of bottom-left side, gives the result drawn in bold (right).

4.2. Adaptation to generalized cylinders visualization

Such a definition gives quite powerful tessellation elementary cell: as we are interested in generalized cylinders defined using a constant profile (see section 2.1), we can use a more complex tessellation primitive than traditional polygon. Figure 5 shows a 3D example of tessellation elementary cell: those deformed cylinders are generated using a constant, linear, cylinder, and weight function similar to that of figure 4; only M_2 transformation matrix varies throughout the deformations. It is important to note that

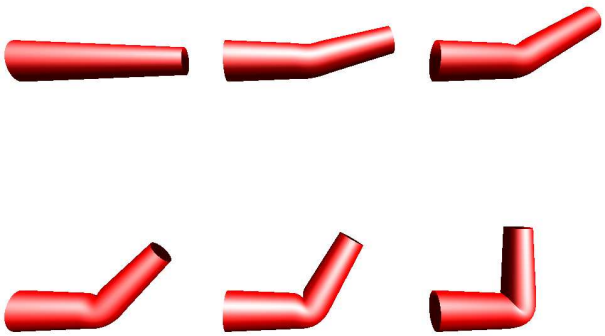

Figure 5. Example of tessellation primitive (simple rectilinear cylinder) combined with different M_2 transformations using vertex blending. Picture extracted from [2].

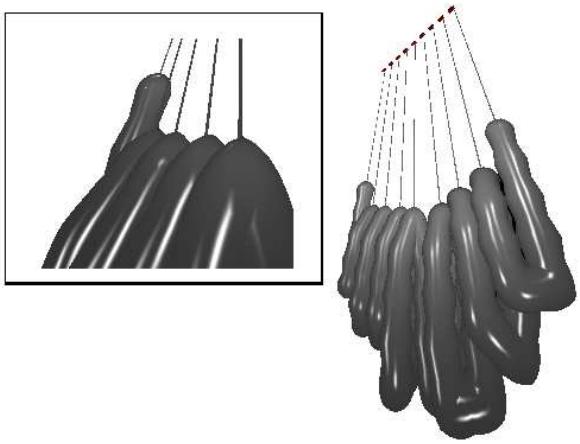

Figure 6. High quality rendering using vertex weight. Top-left: close-up on high curvature points. To be compared with tessellation created in Figure 1.

all those deformations are generated using exactly the same initial cylinder.

In the general case, when the profile is not a circle, we use as display primitive a linear extrusion of the initial profile. Combining this simple, precalculated, shape with the deformation process described above allows us to significantly reduce the software side of the tessellation process, and reduce it to transformation matrices computation. This is the key point in our tessellation process, described in details below. The overall tessellation algorithm can be seen as follows:

- the sampling process (see section 3) gives a set of parameter intervals.
- for each of the intervals generated by algorithm described in section 3, two frames (one for each segment extremity) are calculated along the axis C, using tools given in [6]. Each frame gives local orientation of the sweep cross-section. Combined with profile function value $r(u_i)$, and twist function $\theta(u_i)$, we have local rotation, translation, and scale necessary to position our tessellation primitive, and generate M_1 and M_2 for the considered parameter interval.

Figure 6 shows an example of tessellation achieved using this technique, using circular section (hence, the display primitive is a deformed cylinder). One can see that high-curvature points are handled in a much better way, and that the overall tessellation is somewhat better than that of figure 1. The software part of this tessellation only involves calculus of geometric transformations, and no vertex position is explicitly evaluated and transmitted to the 3D card during the display process. Since the tessellation primitive is constant throughout the time (only matrix changes from a parameter interval to another), it can be stored in a *display-list*, using *vertexArray* technology [2], in order to take full advantage of graphical data transmission optimizations.

4.3. Comments

A few points are necessary to comment. First, the use of skinning explains the limitation mentioned in section 2.1, about the type of generalized cylinders this visualization technique can handle. It is mentioned above that the tessellation primitive is combined with axial deformations. In order to achieve this, it is necessary that the cross section did not change continuously throughout the generalized cylinder.

Second, it is also to note that there is no mathematical proof available that such a deformation tool can properly handle the classical tessellation problems shown on figure 2. Yet, as one can see on figure 6, the visual results are significantly improved by this technique.

It is also to note that raising the number of vertices on the tessellation primitive provides smoother deformation of it (which is quite natural). Yet, this smoothing involves only small computation overhead, as such deformation process is totally performed by hardware. The only overhead is the one needed by transmission of the tessellation primitive to the graphic card. This is one of the key points of the next section.

5. Guaranteed frame-rate

The whole visualization process described above involves two parameters, that makes the visualization more or less accurate. Naturally, the faster the process, the less accurate it is. Threshold value ε (see section 3) determines how many parameter intervals will be used (this number is at least the number of spline segments on the axis C). The complexity of the primitive mesh used for tessellation also determines the quality of the overall shape: the coarser the primitive is, the coarser the deformation performed by the 3D card is. This complexity can be defined using an integer n. In our implementation, we actually use several versions of the same tessellation primitive (in our example, a cylinder). The number of polygons on each version is a $O(2^i)$ for $i = 0, \ldots, n$. For a given parameter interval, raising the number of vertices on the tessellation primitive produces smoother interpolation, with only a little measured computation overhead.

The tessellation system is combined with an algorithm that compares the current framerate to a reference framerate, and adapts automatically the ε value, and complexity n of the primitive used. The global process uses several representations of the tessellation primitives, from coarser one to finest. The principle of the algorithm is fairly simple. Actually it can be seen as a geometric version of iterative energy minimization processes [1]: considering the distance between measured framerate and needed one, tuning functions are applied to ε and n. Figure 7 illustrates this.

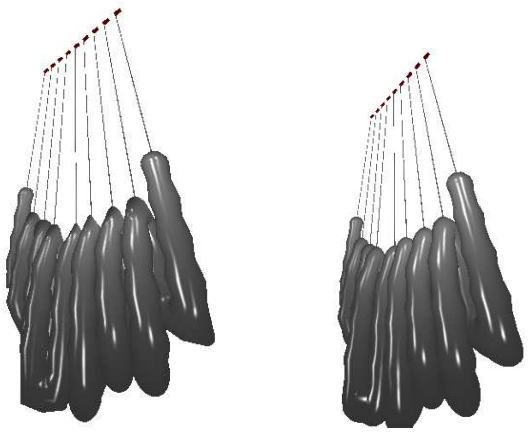

Figure 7. low quality tessellation (left) and high quality tessellation (right) obtained using our automatic frame-rate adaptation system. Respectively 900 and 460 frames/sec, on a GeForce4 based system.

Precisely speaking, the iterations from one display configuration (ε_k, n_k) to the next one $(\varepsilon_{k+1}, n_{k+1})$ uses the following relations:

$$\begin{aligned} \varepsilon_{k+1} &= \varepsilon_k + \alpha.(f_0 - f_k) \\ m_{k+1} &= m_k + \beta(\varepsilon_{k+1} - \varepsilon_k) \\ n_{k+1} &= \lfloor m_{k+1} \rfloor \end{aligned}$$

Where m_k is a floating point version of the integer variable n_k, α and β are two arbitrary constants, f_k is the current measured framerate, and f_0 the desired one.

In our implementation, we used $\alpha = 10^{-4}$ and $\beta = 5$. The tessellation process stabilizes itself at the desired framerate in less than a second.

6. Tests and results

Our test code belongs to a surgical simulator [20]. The generalized cylinder rendering technique described in this paper is used for visualizing a virtual intestine, within laparoscopic surgery simulation system [13]. For our tests, we used the simulation scene presented on figure 3, where the axis shape of the cylinder is determined by physical simulation, influenced by a virtual small sphere, controlled by the user. The weight function we used is a piecewise-polynomial function defined in [5]. Two different configurations have been tested: computer 1 is bi-athlon 2000 using GeForce4, computer 2 is bi-athlon 1600 using GeForce2mx. Hardware based technique appears to run significantly better on machine 1 (actually 5 times faster according to our measures), where high quality rendering provides 460 frames per second, and adaptive sampling combined with automatic frame-rate adaptation allows for a rendering pass that takes less than a millisecond, providing the user with a quality similar to that shown on left side of Fig. 6.

7. Conclusion

In this article we presented a global technique for high-quality, high-performance rendering of highly deformable generalized cylinders, using both efficient axis sampling incremental algorithm, and hardware based visualization technique. We also introduced a self-parameterizing algorithm, that allows a static code to guarantee a given rendering speed on any machine, degrading the rendering quality when needed.

References

[1] *Numerical Recipes*. Cambridge University Press. Available at http://www.nr.com/.

[2] Nvidia programmer online reference. http://developers.nvidia.com/.

[3] G. Agin and T. Binford. Representation and description of curved objects. *IEEE Trans. On Computers*, 25(4):439–449, 1976.

[4] J. Berchtold, I. Voiculescu, and A. Bowyer. Interval arithmetic applied to multivariate bernstein-form polynomials. Tech. Report 31/98, School of Mechanical Engineering, Univ. of Bath, October 2000.

[5] C. Blanc and C. Schlick. Extended field functions for soft objects. *Implicit Surface'95 Proc.*, pages 21–32, 1995.

[6] J. Bloomenthal. Calculation of reference frames along a space curve. In *Graphics Gems (A. Glassner ed.)*, pages 567–571, New York, 1990. Academic press.

[7] J. Bloomenthal. Medial-based vertex deformation. In *Symposium on Computer Animation Proc.*, pages 147–151, San Antonio (USA), July 2002.

[8] W. Bronsvoort, P. V. Nieuwenhuizen, and F. Post. Display of profiled sweep objects. *The Visual Computer*, 5:147–157, 1989.

[9] S. Coquillart. A control point based sweeping technique. *IEEE Computer Graphics & Applications*, 7(11):36–44, 1987.

[10] B. Crespin, C. Blanc, and C. Schlick. Implicit sweep objects. *Computer Graphics Forum (Eurographics'96 Proc.)*, 14(3):165–174, August 1996.

[11] G. Elber and E. Cohen. Second order surface analysis using hybrid symbolic and numeric operators. *ACM Transaction on Graphics*, 12(2):160–178, April 1993.

[12] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, inc. second edition, 1990.

[13] L. France, A. Angelidis, P. Meseure, M.-P. Cani, J. Lenoir, F. Faure, and C. Chaillou. Implicit representations of the human intestines for surgery simulations. *MS4CMS'02 Proc.*, November 2002.

[14] C. Grimm. Implicit generalized cylinders using profile curves. In *Implicit Surface'99 Proc.*, 1999.

[15] L. Grisoni. *Elements de multiresolution en modelisation geometrique (in french)*. PhD thesis, University of Bordeaux I, 1999.

[16] L. Grisoni, C. Schlick, and C. Blanc. Hermitian b-splines. *Computer Graphics Forum*, 18(4):237–248, December 1999.

[17] J. C. Hart, A. Durr, and D. Harsh. Critical points of polynomial metaballs. *Implicit Surface'98 Proc.*, pages 69–76, 1998.

[18] R. Kazinnik and G. Elber. Orthogonal decomposition of non-uniform bspline spaces using wavelets. *Eurographics'97 Proc.*, pages 27–38, August 1997.

[19] M. Kim, T. Chang, J. Lee, and S. Hong. Direct manipulation of generalized cylinders based on b-spline motion. *The Visual Computer*, 14(5):228–239, 1998.

[20] P. Meseure and C. Chaillou. A deformable body model for surgical simulation. *Journal of visualization and Computer Animation*, 11(4):197–208, September 2000.

[21] L. Piegl and W. Tiller. *The NURBS book*. Springer Verlag, 1995.

[22] J. Snyder. Interval analysis for computer graphics. *Computer Graphics (SIGGRAPH Proc.)*, 26(2):121–130, July 1992.