Realtime Visualization of Implicit Objects with Contact Control

Frederic Triquet Laurent Grisoni Philippe Meseure Christophe Chaillou
LIFL (University Lille 1), UPRESA CNRS 8022, 59655 Villeneuve d'Ascq, France
[triquet|grisoni|meseure|chaillou]@lifl.fr

Abstract

This article presents an extension of classical marching cubes for implicit objects associated with blending control. In the framework of classical blend graph[Guy and Wyvill 1995], a technique for handling blending is presented, based on mostly boolean operations, and hence allowing efficient implementation. What is more, the resulting tesselation is topologically closer to the theoretical topology of the mathematical object considered than classical marching cubes tesselations. We also propose an optimization, that we call **partial time stamping**, that significantly diminishes the amount of redundant field values evaluation during marching cubes tesselation, and has the advantage of being compatible with blending control. This technique can be seen as the extension of an optimization technique described in [Triquet et al. 2001], that was not designed for blending control. Our technique is finally compared to classical marching cubes implementations, as a reference. In this final section it is shown that our technique permits high performance visualization of implicits objects combined with blending control.

1 Introduction

Implicit objects appeared to be very useful in several computer graphics fields, including for example soft object modeling[Wyvill et al. 1986; Bloomental et al. 1997; Bloomenthal 1995], and skinning animated models, especially for physically-based animation [Gascuel 1993; Cani 1998]. Such a modeling tool provides nice properties, including automatic blending, compact representation, and no topological constraints. Yet, some special cases suffer from those properties. Among them, blending is a property that has sometimes to be controlled. Figure 1 illustrates this problem. This object is an example of an implicit skinning: body skeleton, using implicit skin, provides a soft approximation of the resulting object. However, the blending between the arm and the hip is not desirable.

As a result, it appears useful to be able to control such a blending property. For that purpose, several techniques exist. Most of them rely on the notion of group between primitives: primitives from a given group can blend together, whereas no blending is allowed between primitives from different groups. Some techniques can also be combined with blending control: in order to simulate the natural behavior of soft and deformable objects touching each other, local deformations due to collisions may be added.

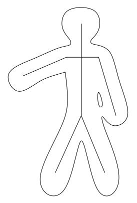


Figure 1: Typical blending problem

For real-time tesselation of implicit objects, two main classes of algorithms can be found: seed-based algorithms[Witkin and Heckbert 1994; Heckbert 1998; de Figueiredo et al. 1992; Desbrun et al. 1995] and space partitioning techniques[Wyvill et al. 1986; Lorensen and Cline 1987; Bloomenthal 1988; Bloomenthal 1994]. Those two classes of methods directly provide triangles used by graphics hardware. Seed-based techniques often handle topology using differential analysis techniques, and hence involve quite heavy computations, which make them difficult to use for high-performance tesselation. Beside, when handling dynamic implicit objects (i.e. objects composed of animated primitives), topological changes within the theoretical isosurfaces make the tesselation problem even more complicated. On the other hand, marching cubes based techniques are simpler to implement, and thanks to systematic treatments of space (at least in the basic definition of the algorithm [Lorensen and Cline 1987]), do not need any particular structuration of the object.

This paper proposes a tesselation technique combined with blending control that does not add any topological inconsistencies in regard to the theoretical result. The algorithm is presented, as well as the proposition of an adaptation of marching cubes optimizations previously published in [Triquet et al. 2001] to the problem of blending control, introducing what we called *partial time-stamping*.

The paper is organized as follows: the next section presents a rapid overview of previous techniques, both for blending control and deformation around contact areas. We also discuss them in the framework of real-time visualization. This section also points out the problems that occur for those techniques. Section 3 presents important points of a previously published paper dealing with high speed marching cubes. Starting from the problems raised in 2, we then propose our method in section 4, and show some experimental results in section 5.

2 Previous works

This section describes some previously published techniques that were proposed for controlling the blending between soft objects, as well as deforming the contact area, in order to simulate contact between deformable objects. We first enumerate techniques for those two problems, and then, discuss some technical issues about them in regard to marching cubes.

2.1 Blending control

Brian and Geoff Wyvill first pointed out the importance of blending control [Wyvill and Wyvill 1989]. They suggested to create primitive groups, within which blending is allowed, and with no interaction between primitives from different groups.

The idea consisting in distributing primitives among disjoined groups, works well for objects that have well-distinguished parts. Yet, in some special cases, this technique becomes quite involving: refer to the character of figure 1. It cannot be correctly represented in this way: his arms, legs and trunk have to belong to the same group in order to obtain correct blending of the surfaces at the shoulders and the hips, and as a consequence, arms and legs still blend together. Nevertheless, even if some group subdivision tricks could be used in this special case, the problem in the general case remains.

Wyvill's idea has been further improved in [Opalach and Maddock 1993; Guy and Wyvill 1995], where an adjacency graph describes the object structure. Each vertex of the graph stands for a primitive (that, by definition here, cannot be self-intersecting) and the edges represent a blend relation between primitives. In [Desbrun et al. 1995], the calculation of the potential at a given point P takes benefit of this structure by keeping only the maximal contribution of the non-blending involved skeletons. This contribution is calculated using classical summing. The four following steps sum the technique up [Desbrun et al. 1995]:

1. compute all the field contributions at point P,
2. select the field f_i that is preponderant,
3. add the maximal contribution from groups of skeleton that blend together and are all neighbors to S_i in the graph,
4. return the resulting value.

2.2 Local deformations

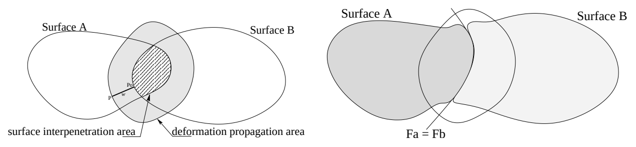
Cani-Gascuel[Gascuel 1993] describes a technique that controls blending between implicit primitives, and also simulates the contact between soft objects, by creating local deformations near the contact area.

The deformation of a surface a by another surface b is obtained by modifying the field function as follows:

$$G_{a,b}(x,y,z) = F_a(x,y,z) - F_b(x,y,z) + isovalue \qquad (1)$$

where F_a (respectively F_b) is the field function associated with surface a (resp. b) and G_a the one associated with the new deformed surface a (deformed by b). The idea to use a negative field function in order to compress such surfaces was formerly proposed by Blinn in [Blinn 1982].

When an inter-penetration is detected, each surface is deformed according to all the concerned objects. Within the inter-penetration zone, the contact area is determined by taking equation 1 and the symmetric expression (swapping a and b). As a result, the points lying on the contact area satisfy $F_a = F_b$ (see figure 2b). Around this inter-penetration



(a) Initial surfaces (b) Deformed surfaces

Figure 2: Contact and deformations between two implicit surfaces

zone, a given quantity is added to the field values. This quantity depends on the thickness w of the zone where deformations are propagated, a given parameter α characterizing the size of the created bulge and the distance between the point we consider and the inter-penetration zone. As shown on figure 3, the typical shape of this function (defined on $[0, w]$) keeps C^1 continuity at the exterior limit of the deformation area (see figure 2b).

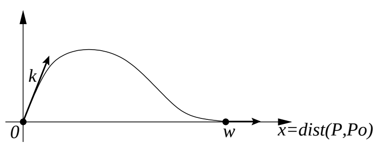


Figure 3: Shape of function used for field value deformation in [Gascuel 1993].

Opalach and Cani[Opalach and Cani 1997] simplify the problem: they build their deformation term using the field values associated with the surface causing the deformation (for example, surface B in figure 2). The shape of this function is shown in figure 4. In the deformation zone, the field

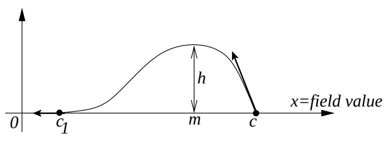


Figure 4: Shape of function D used in Equation 2 [Opalach and Cani 1997]

function for surface A becomes:

$$F_{A_2}(p) = F_A(p) + D(F_B(p)) \qquad (2)$$

where function $deform$, defined on $[c_1, c]$ satisfies the following properties:

$deform(c) = 0$	surface C^0 continuity
$deform'(c) = -1$	deformation around contact area
$deform(c_1) = 0$	C^0 continuity of the surface
$deform'(c_1) = 0$	C^1 continuity of the surface at the border of deformed zone

2.3 Discussion

Some special tesselation technique has to be derived when handling blending control and contact deformation. In other words, controlling the blending of implicit surfaces closely depends on the display scheme. The previously described methods for blend control give good results when used in a ray-tracing context or with a seed-based tesselation. However, they do not work well with a marching cubes algorithm. As a result, a specific visualization method is required.

For the tesselation computation, a problem occurs during the combination of marching cubes with classical deformation techniques. In order to illustrate this problem, let us focus on the results we get when applying, for example, Desbrun's field value computation[Desbrun et al. 1995] to the marching cubes algorithm.

Figure 5 shows the real implicit surface that should be approximated with triangles and the field values at the vertices of the grid. These field values result in the tesselation shown in figure 6a, which is slightly different to the one that should be built (figure 6b).

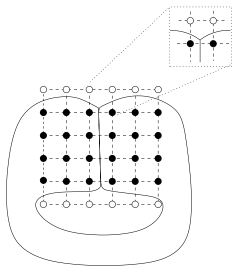


Figure 5: Initial configuration. The exact isosurface is drawn, as well as the marching cube grid. Black points are marked as being inside the surface whereas white points are considered as being outside the surface)

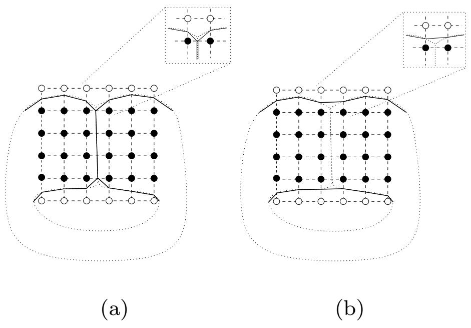


Figure 6: Comparison between ideal tesselation (a) and constructed one (b) using classical marching cube

The tesselation we build is incorrect within the cubes containing both surfaces. For the considered configuration of those cubes, it is not possible to obtain a correct tesselation. Indeed, since we remember the field value of the preponderant primitive, the nodes lying on the left (respectively right) half of the grid correspond to the left-hand (resp. right-hand) primitive (figure 5). Thus, the two lower nodes of the figure 6 are considered to be *inside* the surface, while the two upper nodes are considered to be *outside*. Therefore, there is no reason for the marching cubes algorithm to build an intersection point along the lower edge (see figure 6b). It appears natural, at this step, to allow the marching cubes to do several partial tesselations whithin such a cube and to superimpose them in the final rendering..

When using a graph for controlling primitives blending, one generally tesselates separately each group of blending primitives. Such a method ensures topological consistency in regard to contact. Yet, for reasons described in [Guy and Wyvill 1995], it is not possible, in the general case, to keep the idea of grouping primitives in order to handle the case of self-colliding objects in such a manner. As a result, one has either to subdivide groups, or (and this is our choice here) to consider that the notion of group does not hold. Each primitive is considered separately. Such an assumption does not solve the classical problem of self-colliding object, since we consider that no primitive can be self-colliding. This is always true in the case of discrete skeletons. The generic problem is encountered when controlling the blending of a self-colliding, one piece object (and which topology has to be preserved, as for the character on figure 1). See for example the snake presented in figure 8, made using blobs. A "by group" decomposition, as previously presented, is not possible. On the other hand, seeing this snake as the union of small, simple, not self-intersecting nor self-colliding (i.e. blobs), primitives, makes the use of blending graph possible, and handling of auto-colliding snake, conceivable.

The notion of primitive group does not hold under the assumptions defined above. The question hence remains to know how to use the partial tesselation idea presented by Wyvill. In the framework of marching cubes tesselation, the notion of "group" within which all the primitives blend, can actually be found, locally to a tesselation cube. Indeed, for a cube several primitives have a non-null contribution to the global potential field. Among this list, it is theoretically possible, using the blend graph, to retrieve some sub-lists of primitives such that all the primitives of a sub-list blend together, and that no blending relation between primitives of different sub-lists does exist. For the considered cube, those local sub-lists are the groups that can be used for partial tesselation. Tesselating a cube does not involve a single classical tesselation, but the amount of sub-lists for a given cube defines the number of tesselations that are necessary for this cube.

3 Fast marching cubes

An implementation, in the "always blending" case, has been presented in detail in [Triquet et al. 2001]. In this article, a collection of algorithmic techniques is used in order to reach small tesselation times even for quite complex iso-surfaces (about 200 primitives in 25 ms on a high-end consumer PC). We sum up here important parts of this article in regard to the problem addressed (i.e. adaptation to implicit objects combined with blending control), so that the present paper is somewhat self-contained, and that the arguments that makes the proposed algorithm derivation important, quite natural.

In this algorithm[Triquet et al. 2001], the sources of optimizations to take into account are:

- First, space partitioning, well-known for improving ray-tracing[Cazals et al. 1995], collision detection, radiosity[Cohen-Or et al. 1998], is used for primitives sorting: the tesselation space is partitionned using simple cubes. For non-ambiguous description, we will, in the remainder of this paper, call *cube* the space partitioning unit used during marching cubes tesselation, and *voxel* the space partitioning unit used for primitive grouping. For each voxel, a list of primitives is built, made of primitives which influence area intersects the considered voxel. This ensures that, for any point of a voxel, **including** the points of the frontier, the primitives that have non-null influence on the point are all among the voxel primitive list (although some of this list might have null influence). See figure 7 for an example of list construction. It can be seen that for a given voxel, it might happen that the overall potential evaluation does not involve all the primitives belonging to the voxel list.

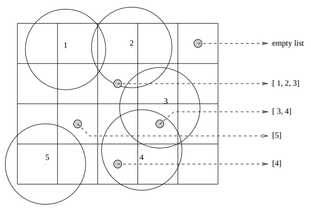


Figure 7: Example of voxel partitioning, combined with 5 primitives. Each primitive belongs to all the voxel lists its influence area intersects.

- For each cube, an unsigned integer value is stored, representing the frame number at which the cube has been treated. This number, called *time-stamp* identifies the current tesselation. At the end of a given tesselation, all the processed cubes have the same time-stamp value. Checking if a cube has already been treated during the current tesselation is equivalent to comparing the current time-stamp with the cube time-stamp. Such marking permits fast tesselation. It is likely that temporal coherency might be used for further optimizations. Yet, since our main framework is that of highly deformable objects (see section 5), such coherency had only but few chances to achieve real improvement (indeed, even small deformations induce new tesselation), and hence, was not the priority of our work.

- In order to combine both multipart objects handling and the minimization of treated cubes amount, a preprocessing of the global primitive list stacks a cube laying on the isosurface for each primitive. Time-stamping ensures that no cube can be stacked twice.

- A cube is then popped from the stack. This cube is used as a seed, and cubes are propagated on the isosurface, in quite a classical manner[Bloomenthal 1995]. Time-stamping value checking prevents multiple treatments.

- We store the vertices field values of each treated cube. Using time-stamping, it is quite easy to avoid redundant field evaluation, using previously calculated values: for a given cube, considering the neighbors the cube shares vertices with, and their time-stamp, one can easily decide whether the neighbor field values can be reused for the shared vertices: if the neighbor time-stamp value equals the current time-stamp value, then the field values are consistent for the current tesselation.

The next section describes in a more detailled manner our technique of tesselation, based on marching cubes. We describe a technique for extraction of primitive sub-lists for a cube, using only bitwise logical operations, and therefore allowing optimal implementation. We also describe some specific optimizations.

4 Implementation of the Blending Control

The framework we are interested in here is the tesselation of an object composed of simple, non-self-colliding primitives (see section 2 for discussion). Apart from the primitives, a blend graph is built, defining the way the primitives blend all together.

4.1 Global tesselation algorithm overview

The algorithm we propose here is derived from the one described in section 3. The main steps of this algorithm are:

- the same notion of *cube* and *voxel* is present here, respectively for tesselation cube, and space partitioning;

- the tesselation of a given frame F implies (same as section 3, but more precise, and with specific notations):

  - calculation of the new current time-stamp t_F, which only involves an integer addition.
  - Let us call N the total number of primitives P, all stored in a list $L = \{P_0, \cdots, P_{N-1}\}$. For each primitive P_i, a cube of the isosurface is calculated, using simple propagation in a particularized direction, and is stacked for further *treatment*.
  - Each cube of the stack is used as an original seed for tesselation: until the stack is empty, a cube c is popped off the stack, *treated*, then marked as treated using the time-stamp value t_F, and the cubes among c neighbors that have older time-stamp value are stacked for treatment. See section 4.3 for details about the time-stamp value, and its precise use. Due to previous stack creation from primitives list, it could happen that a cube might be treated twice: A simple time-stamp value check prevents this.

- The *treatment* of a cube c for a frame F implies:

  - Just like for technique described in section 3, a list L_c of primitives is built. L_c can be rewritten as the set $\{P_{\sigma(0,c)}, \cdots, P_{\sigma(n_c-1,c)}\}$, where σ stands for a function that maps the n_c primitives that have non-null influence on c to their original indices in L. See section 4.2 for further details.
  - L_c is partitioned into sub-lists $L_{c,0}, \cdots, L_{c,\alpha[c]-1}$. Those sub-lists are such that they represent, locally, the classical blend graph: Let define $P_a \in L_{c,i}$ and $P_b \in L_{c,j}$. Then P_a and P_b blend together if and only if $i = j$ (i.e. do not blend if $i \neq j$). See section 4.2 for details about the technique used for partitioning L_c. Note that when the blending relation is not transitive within a cube, then blend is not an equivalence relation, and no partitioning is possible. The solution we adopt in this case is to duplicate "pivot" primitives between non blending groups: for example, if P_i blends with P_j, P_j with P_k, but P_k does

not blend with Pi, then there will be two groups, $\{P_i, P_j\}$ and $\{P_j, P_k\}$.

– c is tesselated $\alpha[c]$ times in a marching cube like manner, the i-th tesselation corresponding to the primitive sub-list $L_{c,i}$. Each tesselation involves field evaluation using deformation techniques described in section 2.2.

Let us examine the critical case of the snake (Figure 8): here, the blending relation are composed of the groups $\{P_i, P_{i+1}\}$. In other words, the matrix representing the blending graph is a band matrix, with band width equal to 3, all elements of it being equal to 1. The duplication process we use entails that all primitives appear in two groups. Several groups will hence be computed, causing as many tesselations of the cube. However, such a case tends to be really rare since it only occurs when several (non blending) primitives lie within the same cube.

4.2 Efficient primitives sorting

We describe here a technique for manipulating the blend graph and creating the sub-lists needed by the global tesselation algorithm. The blend graph is represented by a two dimensional binary matrix M such that $M(i,j) = 1$ if P_i and P_j blend together, and 0 otherwise. In our C++ implementation, M is stored as a matrix of *long long unsigned int*, which has the drawback to constrain the number of primitives manipulated to be a multiple of the size of this datatype, but presents the important advantage to be very compact in regard to further manipulation, as we will see in this section. As a result, and without loss of generality, we can consider that the line $v(i)$ of matrix M is composed of an unsigned binary value, which j-th bit expresses if P_i blends with P_j or not.

Our framework here is that of a cube c for which we have built the list L_c. The very first step of the manipulation is the correspondance to be done between the list and a binary word (that we will abusively note using the same notation) where the i-th bit value is 1 if $P_i \in L_c$, and 0 otherwise. It is very practical to see all the involved lists as binary words of that kind, as, for example, testing if $P_i \in L_c$ can be written in a C-like syntax using bitwise operations, $L_c\&(1 << i)$. Merging two primitive lists A and B (e.g. for building the list L_C from the voxel lists) is simply evaluating $A|B$. In the result, P_i is present if the i-th bit equals 1. On most common nowadays architecture, we thus take benefit of merging lists of 64 primitives using one single bitwise logical operation. Using this idea, the operations we manipulate for extracting the sub-lists $L_{c,0}, \cdots, L_{c,\alpha[c]-1}$ from L_c are all part of a simple loop:

1. $\alpha[c] = 0$;
2. while L_c is not null:
    - calculate i as the first non-null bit in L_c /*this can be done using iterative shifting combined with AND masking*/;
    - retrieve $v(i)$ from M;
    - $L_{c,\alpha[c]} = L_c\&v(i)$;
    - $L_c = L_c \wedge L_{c,\alpha[c]}$ /*the sub-list $L_{c,\alpha[c]}$ is withdrawn from current list L_c. This, because in our special case we have $L_{c,\alpha[c]} \subset L_c$, that corresponds to a bitwise logical XOR operation */;
    - $\alpha[c] = \alpha[c] + 1$;

For the sake of simplicity, we did not include in this algorithm the special case of non-transitive blending relation within a cube list. Yet, it is quite simple to use bitwise logical operations for detecting the primitives that have to be shared by groups, and use this information for creating correct sub-lists. Such an algorithm description presents the immediate advantage to be both of a very high description level, and easily implemented in a very simple and efficient code, fully optimized by most common compilers.

4.3 Partial time stamping

Classical time stamping[Jevans et al. 1988], within the framework of blending controlled implicit object tesselation, is not fully usable: Its main goal was to prevent redundant potential evaluation by assigning a tesselation time to each cube. When treating a cube, the neighbor concerned by a given shared face is first checked, and, if its time-stamp value corresponds to the current value, then this means that the potential value stored can be used. When controlling blending, several tesselations per cube are possible, and as a result it is useless to know if the cube has the correct time-stamp, since several potentials would be present. Therefore, the question would remain, to know which value is correct for the primitives we consider.

Yet, such a philosophy can be reused under some slightly stronger assumptions: Within an implicit object having non-blending primitives, a high percentage of cubes does not involve any number of sub-lists greater than 1, i.e. do not have non-blending local groups. As a result, those cubes, under the condition that the neighbor does not need several treatments either, could take benefit of time-stamping anyway.

This is why we defined what we called *partial time-stamping*, which is a binary word made of the concatenation of classical time-stamp value, and a boolean information, expressing if the $\alpha[c]$ value, calculated by the algorithm presented in section 4.2, equals 1 or not. This simply allows for determining if we can re-use the previously calculated field values or not. The value stored as time-stamp per cube c, instead of value t_F for classical time-stamping, is now (in C-like syntax) $(t_F << 1)|(\alpha[c] == 1)$.

5 Experimental results and measures

This work takes place in a more complex software environment, being devoted to physical-based simulation. In figure 8 is presented a surface made of 100 primitives where the group technique cannot be applied, as the object is "continuously" defined, and self colliding.

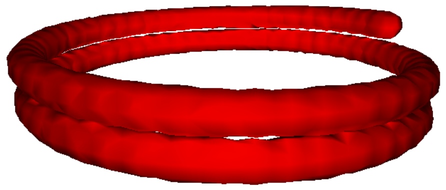


Figure 8: One hundred primitives with blending control

Figure 9 displays the wire version of the tesselation of 3 interacting blobs. One can see that the deformations applied to the surfaces are realistic, and that the topology of the result matches the intuitive topology of the theoretical isosurface.

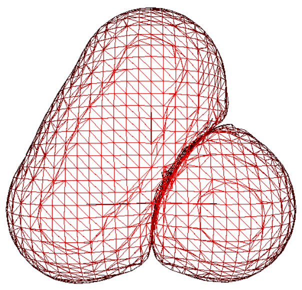


Figure 9: Wireframe display of 3 primitives (blobs)

Figure 10 shows the time required to tesselate a surface, depending on the number of primitives. Those measures have been performed on various versions of the object shown on figure 8, being deformed periodically like a spring along its axis.

A comparison is made in figure 10 between the tesselation times required by different implementation techniques: Our initial tesselator (without blending control), the technique presented in this paper (with blending control), and Bloomenthal's implementation[Bloomenthal 1995] (which, in its initial formulation, cannot consider blending control). Bloomenthal's code, since it has been written to be easily understood, was not very optimized. However, the space partitioning technique can speed up a lot the tesselation process, as shown in figure 10.

Note that in those measures, only the tesselation time is given, not the time needed for visualizing the resulting object (which is graphics hardware dependant, and independant from the algorithm described in this paper).

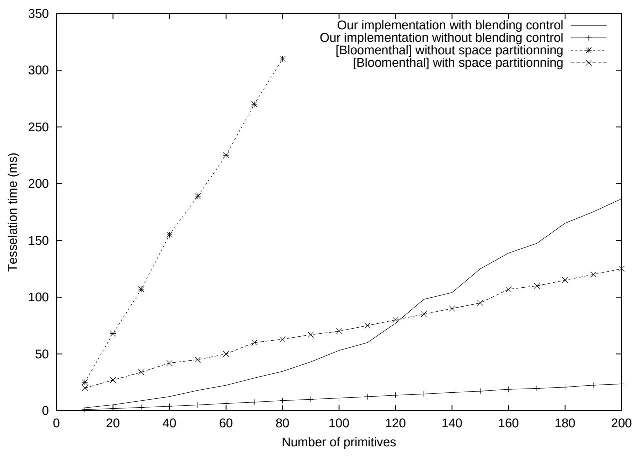


Figure 10: Tesselation time using several techniques

The tesselation method presented in this paper is used in a simulator of laparoscopic surgery which, for accurate simulation reasons, has to be an application with high framerate performances. In this simulator, intestines are modeled using 165 primitives. Its tesselation, made of about 40000 triangles, is entirely constructed at each frame. Due to the high deformability of the model, and the context of physical simulation that introduces small, yet existing, oscillations, no time coherency is used. The overall application, using our tesselation technique combined with real-time physical manipulation of the intestines, based on physical simulation, runs approximatively at 7 images per second on a Pentium IV at 1.7GHz using a *Quadro II* graphic card (although we can reach 23 images per second in the same context when desactivating the blending control). Figure 11 shows a laparoscopic tool manipulating these intestines. As it can be seen, our technique allows us to control the blending and the deformations of the surfaces built around complex skeletons.

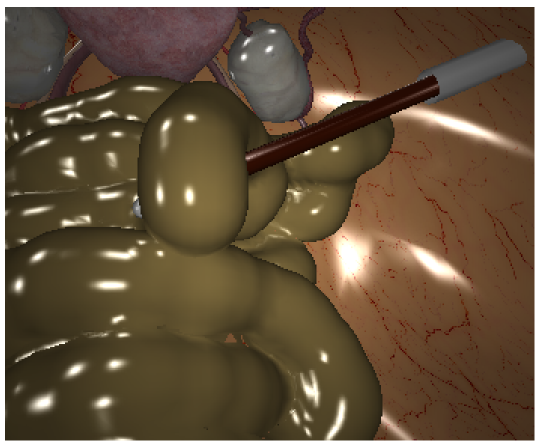


Figure 11: Manipulating implicit intestines in laparoscopic surgery simulation

6 Conclusion and future work

This paper presented a framework for high-performance tesselation of implicit objects, that provides blending control. The proposed tesselation technique does not introduce further topological inconsistencies, in regard to classical ambiguities introduced by marching cubes. The blend graph proposed implementation allows for treatment using only bitwise logical operations, hence simply enables optimal results. We also presented an optimization technique called partial time stamping that significantly diminishes redundant field value evaluation.

An interesting question, above all in regard to the optimizations proposed for for marching cubes, would be to study the applicability of this technique to other tesselation algorithms, such as marching tetrahedrons [Bloomental et al. 1997]. Another direction of research would also be to see in which measure the work presented here can be adapted to the case of continuous skeletons, such as convolution surfaces based on subdivision skeletons [Cani and Hornus 2001]. In this new framework, the assumption stating that the primitives used are not self-colliding no longer holds, and we think that multiresolution-based tesselation techniques [Grisoni et al. 1999] would be a way to take this problem into account.

References

Blinn, J. F. 1982. A generalization of algebraic surface drawing. *ACM Transactions on Graphics 1*, 3 (July), 235–256.

Bloomental, J., Bajaj, C., Blinn, J., Cani-Gascuel, M., Rockwood, A., Wyvill, B., and Wyvill, G. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc.

Bloomenthal, J. 1988. Polygonisation of implicit surfaces. *Computer Aided Geometric Design 5*, 341–355.

Bloomenthal, J. 1994. An implicit surface polygonizer. *Graphics Gems IV*.

Bloomenthal, J. 1995. Bulge elimination in implicit surface blends. In *Implicit Surfaces'95*, 7–20. Proceedings of the first international workshop on Implicit Surfaces.

Cani, M.-P., and Hornus, S. 2001. Subdivision curve primitives: a new solution for interactive implicit modeling. In *Shape Modelling International*.

Cani, M.-P. 1998. Layered models with implicit surfaces. In *Graphics Interface (GI'98) Proceedings*. Invited paper, published under the name Marie-Paule Cani-Gascuel.

Cazals, F., Puech, C., and Drettakis, G. 1995. Filtering, clustering, and hierarchy construction: a new solution for ray tracing complex scenes. In *EUROGRAPHICS'95 Proc.*, 371–382.

Cohen-Or, D., Fibich, G., Halperin, D., and Zadicario, E. 1998. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. *Computer Graphics Forum 17*, 3. Eurographics'98 Proc.

de Figueiredo, L., Gomes, J., Terzopoulos, D., and Velho, L. 1992. Physically-based methods for polygonization of implicit surfaces. In *Graphics Interface'92*, 250–257.

Desbrun, M., Tsingos, N., and Gascuel, M.-P. 1995. Adaptive sampling of implicit surfaces for interactive modelling and animation. In *Implicit Surfaces'95*, 171–186. Proceedings of the first international workshop on Implicit Surfaces.

Gascuel, M.-P. 1993. An implicit formulation for precise contact modelling between flexible solids. *Computer Graphics 27*, 313–320.

Grisoni, L., Crespin, B., and Schlick, C. 1999. Multiresolution Implicit Representation of 3D Objects. Technical report, University of Bordeaux I, http://www.lifl.fr/~grisoni.

Guy, A., and Wyvill, B. 1995. Controlled blending for implicit surfaces. In *Implicit Surfaces'95*, 107–112. Proceedings of the first international workshop on Implicit Surfaces.

Heckbert, P. 1998. Fast surface particle repulsion. *Report CMU-CS-97-130, April 1997. Appeared in SIGGRAPH 98 course notes*.

Jevans, D., Wyvill, B., and Wyvill, G. 1988. Speeding Up 3D Animation For Simulation. *Proc. SCS Conference*, 94–100. Proc. MAPCON IV (Multi and Array Processors).

Lorensen, W., and Cline, H. 1987. Marching cubes: a high resolution 3d surface construction algorithm. *Computer Graphics 21*, 4 (July), 163–169. Proceedings of SIGGRAPH'87 (Anaheim, California, July 1987).

Opalach, A., and Cani, M.-P. 1997. Local deformations for animation of implicit surfaces. *SCCG'97 (Bratislava, Slovakia)* (jun). http://w3imagis.imag.fr/Membres/Marie-Paule.Cani/mouImplicite.html.

Opalach, A., and Maddock, S. 1993. Implicit surfaces: Appearance, Blending and Consistency. In *Fourth Eurographics Workshop on Animation and Simulation*, 233–245.

Triquet, F., Meseure, P., and Chaillou, C. 2001. Fast polygonization of implicit surfaces. *WSCG'2001 (Plzen, Czech Republic) 2* (feb), 283–290. http://wscg.zcu.cz.

Witkin, A., and Heckbert, P. 1994. Using particles to sample and control implicit surfaces. *Computer Graphics* (July), 269–278. Proceedings of SIGGRAPH'94.

Wyvill, B., and Wyvill, G. 1989. Field functions for implicit surfaces. *The Visual Computer 5* (Dec.), 75–82.

Wyvill, G., McPheeters, C., and Wyvill, B. 1986. Data structure for soft objects. *The Visual Computer* (Aug.), 227–234.