Implicit Patches: An Optimized and Powerful Ray Intersection Algorithm for Implicit Surfaces

Jean-Dominique Gascuel

CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

iMAGIS */ IMAG
BP 53, F-38041 Grenoble CEDEX 09, France
email: Jean-Dominique.Gascuel@imag.fr

Abstract

This paper describes an new and optimized direct ray tracing algorithm over complex implicit surfaces generated by skeletons. Its main originality is its ability to avoid unwanted blending between parts of the same object, thanks to the partitioning of the surface into several pieces, so called *Implicit Patches*. Moreover, these patches enable to exploit the properties of local field functions, and to speed-up considerably the rendering. Extensive statistics of the various optimizations proposed are given and discussed. The implementation into the public domain software *RayShade* is sketched.

Introduction

Implicit Surfaces are more and more used throughout the research community. They offer an easy representation of smooth free form objects, even with branching and arbitrary topology (e.g. teepots with "handles"). Moreover, they seem particularly suitable for animation of deformable objects, that may change their topology. One can encounter them in domain as different as descriptive animation [1, 2], physically based simulation [3, 4], or metamorphosis modeling [5].

At iMAGIS, implicit surfaces generated by skeletons are used for computing collision detection and response between soft deformable objects [6, 7], which involves adding extra local terms to the field functions in order to model exact contact surfaces. Fast simplified visualizations [8] similar to the method presented in [9] are used for displaying simulations. But even if these techniques are sufficient for tuning an interactive animation, they cannot be compared with common rendering over tessellated objects or spline surfaces. However, demonstrating the usability of implicit surfaces for standard production must be done by showing comparable, high quality images. This is particularly difficult since current modeling and rendering tools either do not support implicit surfaces, or render them through a tessellation pre-processing. None of them supports local deformations due to contacts, nor unwanted blending avoidance, a problem known for a long time [10, 2] that limits the benefits of implicit surfaces for animation (for instance, the arms and legs of a character should not blend together during an animation, although they both blend with the body).

*iMAGIS is a Joint Project between CNRS, INRIA, INPG, and UJF.

This paper presents a new direct ray tracing algorithm for implicit surfaces, that supports both unwanted blending and collision deformations. As rendering an animation is a compute intensive process, we particularly address the problem of performance: several optimizations to ray-intersection algorithms are proposed and discussed.

The remainder of the paper develops has follows : Section 1 briefly describes implicit surfaces and the associated ray tracing algorithms. The next section, 2, explains how splitting the surfaces into several patches both improves performance, and enables unwanted blending avoidance. Rendering precise contact between two implicit surfaces is addressed in Section 2.2, and optimizations of ray intersection algorithms are discussed in Section 3.

1 Direct Ray Tracing over Implicit Surfaces

An implicit surface is mathematically defined as the isosurface of a field function, i.e. the set of points that satisfy an equation $f(P) = 1$, where f is greater than one inside the object. Thus, the surface surrounds an implicit volume with a well defined inside/outside function: a point is inside the volume if and only if $F(P) > 1$.

Ray tracing algorithms compute images by simulating a ray that follows, backwards, the path the light from the eyes up to the different light sources of the scene (see Figure 1). A ray is shot into each pixel, its intersection with the first surface encountered is computed. Then, secondary rays are sent to each light source to know whether they are occulted or not, as well as in reflection and refraction directions if the surface is either shiny or transparent. All these results contribute to the final color of the pixel.

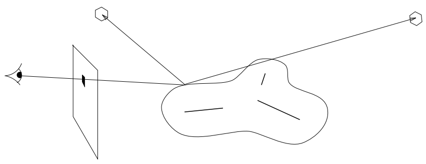


Figure 1: Shooting a ray from the observer eye, through the image, on an object and then to two light sources.

The code that computes ray/surface intersections is the central part of any raytracing program. Contrary to parametric surfaces, implicit surfaces are particularly well suited to direct ray-intersection processing, since the inside/outside function defining the implicit volume enables to compute ray/surface intersections by standard numerical search.

1.1 Previous Works

Blinn was the first one to suggest the use of ray-tracing for visualizing implicit surfaces. His application was developed for iso-surfaces generated by a set of points with exponential field functions [11]. Algorithms were then developed to render implicit surfaces defined by polynomials of the space coordinates x, y, z [12, 13]. Distance-based field functions usually cannot be described in terms of such polynomials. Other approaches [14, 15, 16, 17, 18] were developed for computing a polygonization of the surface, that enables to use classical rendering softwares to render the resulting triangular meshes.

In 1989, Kalra and Barr [19] proposed a robust direct ray tracing over implicit surfaces. Their algorithm relies on bounds of the field first and second derivative to build a guaranteed intersection algorithm. It also uses hierarchical spatial subdivisions to optimize the whole process. Latter on, [20] proposed a simpler method that compute at once all the intersection between a ray and a soft object, suitable for CSG. This approach is limited to simple field functions (in this case a sum of ellipses) that is not compatible with the general, complex field function of colliding objects used at iMAGIS.

1.2 Background

This paper addresses the rendering problem for a set of surfaces that are particularly well suited for animation: distance-based implicit surfaces, generated by skeletons [21]. Indeed, skeletons are a natural concept for an animator, and moving them is a simple way of performing an animation of the implicit surface that coats them [1]. Moreover, collision detection and response can be processed efficiently and accurately for this kind of surfaces, as explained in [6].

More precisely, the user defines a set of simple primitives (mainly points, but also polylines, circles, cylinders, boxes, ...) Sk_i called the skeletons of the surface, and scalar functions f_i that define the field generated by the Sk_i. The field function f of an object is the sum of the field contributions spawned by each of its skeletons :

$$f(P) = \sum_i f^{(i)}(P)$$

The field contributions $f^{(i)}$ are based on the distance $r_i = d(P, Sk_i)$ between the point P and the i^{th} skeleton Sk_i (e.g. the minimal distance between point P and any point of the primitive Sk_i). For instance, the field functions used in [6], depicted in Figure 2, are :

$$f^{(i)}(r) = \begin{cases} 1 - k(r-e) & ifr \leq e & (linear) \\ 1 - \frac{1}{2}(r-e)(\frac{kr}{e} + k) & ifr \leq e & (quadratic) \\ (r-R)^2(-\frac{ekR+2}{(e-R)^3}r + \frac{e^2kR+3e-R}{(e-R)^3}) & ife < r < R & \\ 0 & ifr \geq R & \end{cases}$$

Which is correct (always positive) only if $\frac{3+ek}{Rk} \geq 1$ holds.

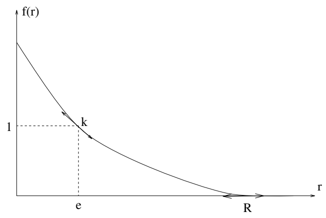


Figure 2: A distance based field function, with its thickness (e), its stiffness (k), and its influence radius (R) parameters.

Throughout this paper, we use a simple set of positive or negative functions that look like the one in figure 2, but the extention to any other kind of distance-based implicit surfaces is straightforwards.

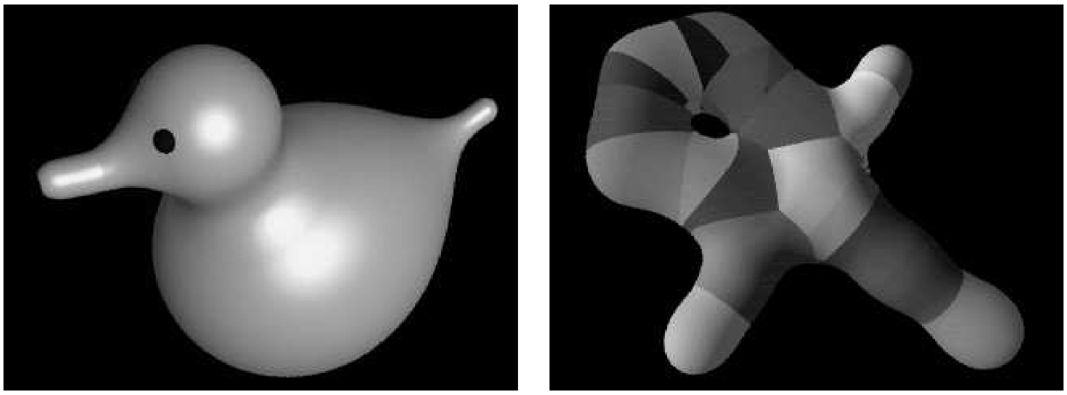
The ray-tracing algorithm for implicit surface presented throughout this paper has been realized by only adding[1] a specific module that computes ray/implicit surface intersections to a well known public domain ray tracing software : RayShade [22].

[1]As well as methods for computing the distance from an arbitrary point to any primitive.

The optimizations we are using are closely related to the one described in [19], although we present several important improvements that should be applicable for various types of implicit surfaces.

Moreover, we adress two specific problems that must be taken into account when surfaces need to be used for animation: the unwanted blending problem, and the precise contact modeling, for which we present a new solution giving better results than the one initially proposed in [6].

2 Rendering Complex Implicit Surfaces



(a) (b)

Figure 3: Implicit objects based on several skeletons, with large radius of influence in (a), and smaller ones in (b). The different colors in (b) correspond to the regions where the different skeletons are predominant

Complex objects modeled with implicit surfaces are generally composed of several skeletons, each of them influencing only a local portion of the shape (indeed, we are using implicit surfaces with finite radius of influence). Since the area where each skeleton contributes may be considerably smaller than the whole object, an interesting pre-processing can be computed: it consists in filtering, for each skeleton, the neighbours that can possibly interact with it.

As a result, we can associate with each skeleton a region of space where his field contribution is the highest, and where the list of neigboring skeletons that possibly contribute to this area is already known. We call *Implicit Patch* the piece of the implicit surface that are is included in this area, and the *neighbours* the skeletons that contribute to a given implicit patch, but that are not the predominant skeleton.

With this formulation, the whole implicit surface can be represented as an union of implicit patches, each of them being associated with a particular skeleton. Implicit patches are displayed with different colors in Figure 3 (b).

Computing the intersection between a ray and a given implicit patch is quite similar to an usual ray/implicit surface intersection, but can be performed much more efficiently thanks to the neighboring skeletons list: We start by computing the intersection between the ray and the implicit surface spawned by the skeleton and by its neighbors only (which is efficient since the whole set of skeletons doesn't need to be considered). Once we have found the first intersection, we check that the field of the primary skeleton is higher than any other else. If not, we reject this root, and try further on.

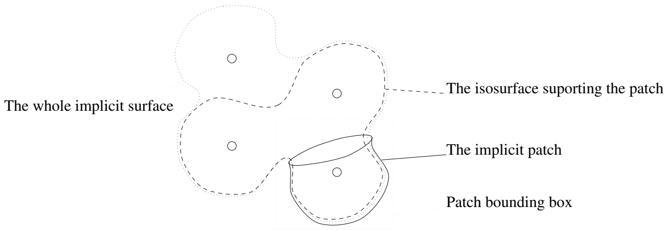


Figure 4: *Implicit Patches* used to model a complex surface. Each implicit patch can be seen as a portion of a traditionnal closed implicit surface that would only be defined by the predominant and neighboring skeletons.

Rendering an implicit surface composed of several implicit patches is simply done by rendering all the patches, that will automatically fit together. Determining which patch is to be intersected by a given ray is facilitated by the use of local bounding boxes associated with each implicit patch. In practice, these local boxes are computed during animations[2], and then stored for final rendering.

As a consequence, the use of implicit patches is more efficient than directly rendering the whole surface:

- The volume of the union of the local bounding boxes is generally much smaller than the volume of a large bounding box including the whole object. In consequence, less extra intersections are computed with rays that do not intersect the surface. In addition, we use intersections with the large bounding box as a preprocessing, which further saves computations.
- Most often, implicit objects are modeled in a way that provides local shape control. In other words, the skeleton's radius of influence are smaller than the object's size. As a consequence, the use of specific neigboring lists save computations at each field evaluation (and a lot of evaluations are needed to find a precise intersection along a ray by binary or linear search).

The next section explains how the use of implicit patches for rendering enables to take into accound the "unwanted blending" properties that were specified while designing the implicit model.

2.1 Unwanted Blending

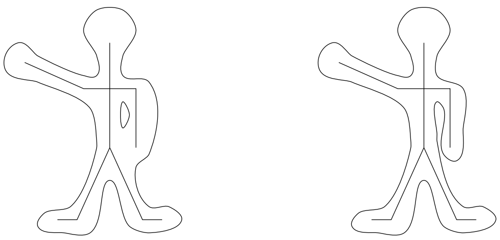


Figure 5: (a) The unwanted blending problem for an articulated character.
(b) The result we want to obtain.

Complex implicit objects that we are experimenting for animation and simulation in our group are of two types. In fluid material [7], the smooth surface surrounds

[2]Local bounding boxes delimiting the area of the surface where a given skeleton is predominant were already used in [7] for accelerating collision detection.

skeletons that are very close from each other. Blending at mid-range distance will be dictacted by the history of chuncks going apart and merging back. On the opposite, in articulated structures, the blending properties between the different parts of a surface are dictated by a structuration that is fixed in advance and does not change during the animation (see Figure 5). In this case, collisions and contacts should be detected between surface areas that correspond to distant links in the articulated structure. If nothing is done, these portions of the surface merely blend in the distance.

The use of Implicit Patches solves this problem gracefully, since the set of skeletons contributing to a patch of the surface is a subset of the full set of skeletons defining the whole surface. If the neigboring list associated with each skeleton correspond to the adequate neigbors in the graph, rendering the implicit patches will automatically avoid unwanted blending.

In our implementation, specifying the lists of neigboring skeletons for each image of an animation is the responsability of the animation system. Figure 6 shows one of our results, where the two extremities of an articulated implicit string are not blending, but colliding together. The next section explains how to model exact contact surfaces during such collisions.

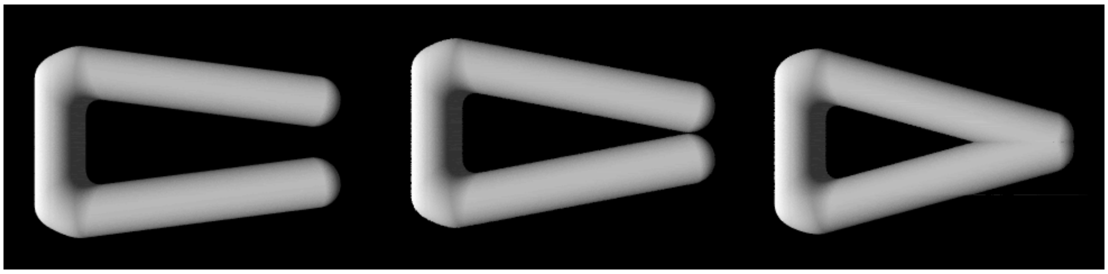


Figure 6: The two extremities of the same surface do not blend together before colliding.

2.2 Contacts Between Surfaces or Patches

Contacts situations between objects are very important in the animation field. When a character is designed, for instance by coating skeletons with an implicit surface, collision and contact situations will generally arise all over the animation. A solution for precise contact modeling with implicit surfaces has already been proposed [6]. However, generating exact contact surfaces is of no benefits if the high-quality rendering software cannot render locally deformed surfaces in contact areas. This section describes our solution to the problem, and gives a new solution for modeling propagation of deformations.

Let us recall shortly the computations involved by colliding implicit surfaces (see [6] for more details). If f_1 and f_2 are the field functions of two colliding objects, then f_1 is modified in the intersection area between the objects by the addition of a compression field term :

$$g_1 = 1 - f_2$$

When f_2 is modified the same way, this insures that the two compressed implicit surfaces will exactly coincide in the contact zone between the objects.

Just adding those compression terms would not preserve the smoothness of an implicitely-defined object. [6] proposes to use a purely geometric model to preserve

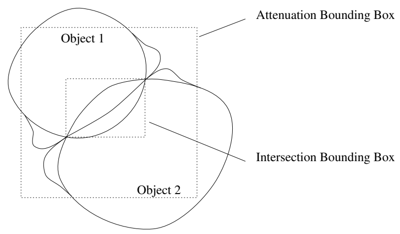


Figure 7: Two colliding implicit surfaces, with deformation bounding boxes, and the smoothing bump added.

the surface G_1 continuity in contact area, thus modeling the local transversal propagation of deformations. If a_1 is the propagation field term to be added to f_1 in the attenuation zone where deformations propagate, smoothness in insured by:

$$\frac{d}{dP}a_1(P) = \frac{d}{dP}f_2(P)$$

for all points P that lie on the border of the intersection between the undeformed surfaces.

To take collisions and contacts into account while processing ray-tracing, we have added a few key-works to the ray-shade format: the animation software stores informations such as "contact" or "propagation" for each implicit patch, together with the name(s) of the implicit patch(es) with which a contact has been detected. Using these informations for adding the adequate extra terms to field functions is straightforwards. Examples of collision and contact situations are depicted in Figure 8.

3 Ray–Patches Intersection Optimizations

As stressed before, the most important part of a ray-tracing process is the module that computes the intersection between a surface and a ray. This section explains how intersections between implicit patches and rays can be optimized.

A ray is defined by it's starting point, it's direction, and the minimum and maximum distance between which the intersections should be searched.

3.1 LG-Implicit Surfaces [KB89]

In 1989, Karla and Barr introduce a new method to compute high quality ray tracing of implicit surfaces [19]. The main lines of the algorithm are presented here, as well as improvements that are pertinent for our model.

The basic idea of the algorithm is to look at the bounds of the first (L) and the second (G) derivatives of the field function f, over region of space. The following three paragraphs explain the three main parts of the algorithm.

3.1.1 L and G Constants

For a given axis-oriented box region of space, the maximum and minimum distances to each skeleton are approximated by the (max and min) distances to the skeleton's bounding box. If the min distance is less than the influence radius of the skeleton, then the bounds of the first and second derivative of f are evaluated, and summed up. This evaluation is done by bounding $|f'|$ and $|f''|$ over the interval $[0, R] \cap [\min, \max]$.

When the region of interest gets smaller, the L and G constants become smaller as well, and the ray-surface intersection algorithm speeds up.

3.1.2 Creation of the Spatial Subdivision

An octree is created during a pre-processing **for each implicit object**. It permits to prune the intersection algorithm on cells where no intersection can occurs, and to store the L and G values over leaf cells, in order to avoid recomputing them during intersections. Since the intersection algorithm is optimized and guaranteed locally inside each leaf cells, several cells should be checked for the same ray/implicit intersection test.

Several heuristics are used to prune the tree. Let us present them, in the order in which they are successively applied during the recursive creation of the octree. Here b is the radius (half diagonal) of the current cell, and C its center :

1. If $|f(C)-1| > bL$ no intersection can occur, so the $f(C)$ value is used to mark the cell either fully inside or fully outside the object.
2. If the cell is at the maximum depth, a leaf cell is stored in the octree. This guarantees that the recursive creation of the octree always finishes.
3. If the cell is deep enough, and a probing at the eight cell corners (and eventually at the center), it demonstrates that parts of the cell are inside, and parts are outside, then a leaf cell is stored.

If none of this criteria applies, the current cell is divided into eight sub-cells that are recursively explored.

The marked cells (in or out) are implemented as *magic* pointers, and do not use memory space. The leaf cells store L and G value, the internal cells store eight pointers their children.

3.1.3 Searching for the First Intersection

When searching for an intersection with an implicit surface, the octree is first traversed, and all intersected leaf cell are checked in the ray order.

For each tested cell, a series of heuristics are applied in order to guarantee that the first intersection is found. We call A and B the entry and exit points of the ray on the cell's box.

1. If $|f(A)-1| > L|| \overrightarrow{AB} ||$ there cannot be an intersection.
2. If $A \approx B$, then there is no intersection in the AB interval. This test insure that the recursive subdivision of the interval finishes.
3. When $\left(\Delta f(\frac{A+B}{2}) \cdot \overrightarrow{AB} < G|| \overrightarrow{AB} ||^2\right)$, f' (in the ray direction), may go to zero between A and B. This means that there could be more than one intersection between the ray and the surface. Hence, it is necessary to subdivide the $\overrightarrow{AB}$ interval and to recurse.

If none of the previous actions apply, then one and only one intersection can exists between A and B. An optimized binary search (Ridder [23]) is started if $(f(A)-1)(f(B)-1) < 0$.

max depth	0	2	4	5	6	7	10
KB89 preproc. time (s)	0.62	1.73	50	244	710	?	?
KB89 inter. time (s)	368	302	213	167	139	?	?
KB89 memory (Kb)	529	1 220	18 770	34 820	101 770	?	?
New preproc. time (s)	0.7	2.4	37.5	96	365	543	8933
New inter. time (s)	369	318	204	194	177	175	140
New memory (Kb)	529	1 210	5 890	13 506	9 485	12 915	268 445

3.2 Improving the Octree Pruning

This section introduce two extra heuristics that improve the spatial subdivision quality of the octrees (better space/creation time and space/traversal time ratios). The number we give indicate where they should be used during the successive tests of section 3.1.2.

1 bis. The aim is to use both G and the actual gradient of f at the center of the cell, to eliminate further cells. We use the condition $|f(C)-1| > b(||\Delta f(C)||+bG)$. Using this criteria in place of the original one gives a net gain of 5% of the total rendering time (and 6% in total memory used). Using it after the original criteria gives gains of 10% (and 12%).

3 bis. An oracle is applied to decide if the subdivision of the current cell will improve the L/G constants : If $\frac{b}{\min} < Dratio$ holds, a leaf cell is created. If not, the original algorithm continues, and subdivides the cell.

In order to further prune the tree, the results are examined :

4. If the children are either all marked in or all marked out, then the current cell is also marked (in or out).
5. If all the children cell are leaf cells, and if the maximum G among the children is not too different from the current G value, then the children are deleted, and a leaf cell is stored for the current cell.

If none of these rules applies, then the current cell is actuallyt stored, with pointers to its eight possible children.

The following table compare statistics computed by summing the costs for the test set (see figure 8 for various maximum depth (with min depth=3), with $Dratio = -1$ (which is quite equivalent to the original Kalra & Barr's algorithm), and with $Dratio = 2$. The first line is the preprocessing time (for the whole set of images), mainly used to build the octrees. The second one is the time spend to compute the images. And finally, the last one is the total memory malloc'ed by the program. The question mark results have not been computed, because of memory exhaustion.

Together, these new criteria enable to increase the maximum octree depth, for the same preprocessing costs (time and/or memory). On large images, (which is not the case for these preliminary statistics), the speedup of the intersection phase could become quite significant.

3.3 Improving the Octree Traversal

We again added some heuristics to improve the Kalra and Barr's algorithm, described in section 3.1.3.

0 bis. First of all, the field value f is computed at the box entry A. If $f(A) = 1$, the intersection A is returned. In our test set, this conditions sometimes appears (0.01%). As $f(A)$ needs to be computed anyway, this test is quite cheap, so we keep it (at least as a sanity check).

2 bis. If $(f(A)-1)(f(B)-1)>0$ then we compute f's gradient, and use it to compute f's derivative in the ray direction.

If $|f(A)+f(B)| > \overrightarrow{\Delta f(\frac{A+B}{2})} \cdot \overrightarrow{AB} + G|| \overrightarrow{AB} ||^2$, then there cannot be a root between A and B. This test is more efficient than the one on L, because the ray direction is used to have a better bound condition.

With these additions, the total rendering is improved by another 14%.

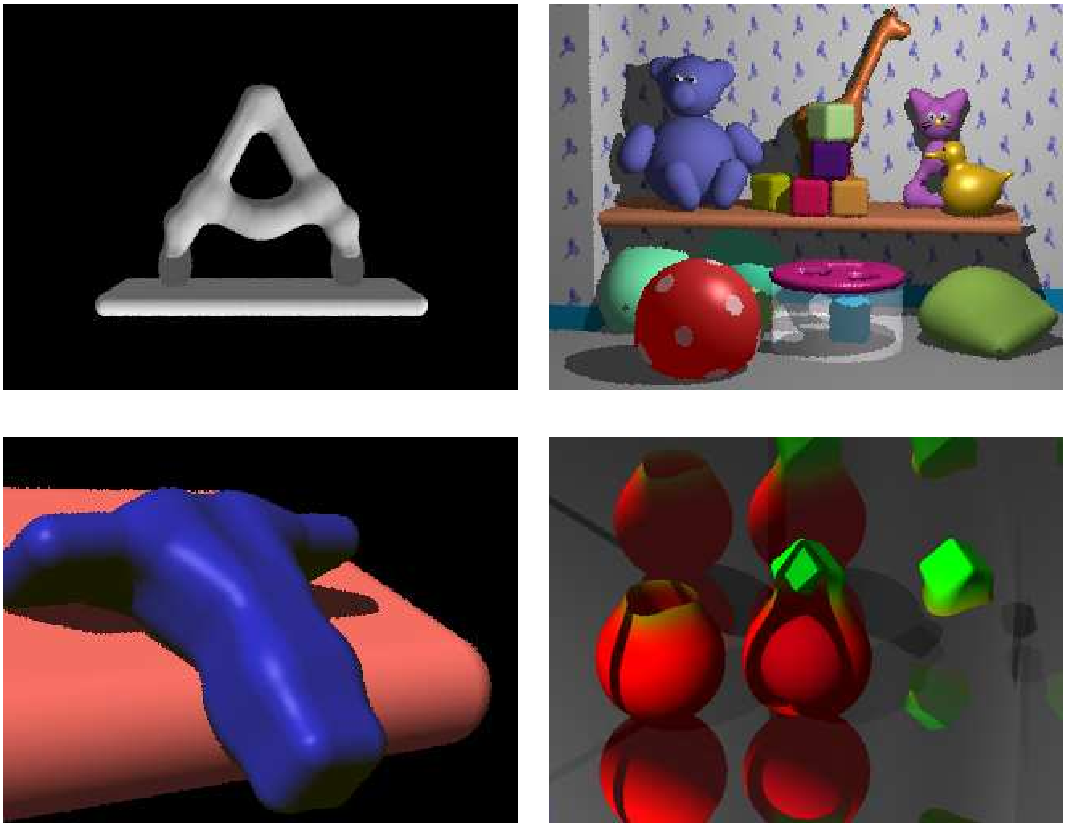


Figure 8: The images (as well as the three other ones) used to gather the statistics presented in the paper.

3.4 Optimizing Patches

Implicit patches can be treated the same way as implicit surfaces. The only difference is that the root box of the octree should only contain the portion of the surface where the primary skeleton's field is the highest. This automatically introduces a partition of space that optimizes the skeleton field computed for each patch, hence accelerating the rendering for implicit surface composed of several patches with small influence radius.

3.5 Optimizing Deformed Surfaces

For each deformed surface (either patch or not), we build a second octree based on the deformed field. As the deformed field computation needs to fire rays to the opponent un-deformed object (to get the distance to the intersection area), the optimizations will be used twice, which considerably accelerate deformed surface intersections.

The L and G bounds for the deformed field function are computed by summing the bounds for the un-deformed field, the bounds of the opponent objects, and bounds from the attenuation function.

4 Conclusion

This paper present a method for high quality direct ray tracing on implicit surfaces that collide, deform and avoid unwanted blending. The classical rendering technics have been adapted and improved significantly.

Current focus is on finding even better and cheap oracles for the octree pruning, and better bounds for the attenuation function.

Acknowledgements

A first optimization of our implicit surfaces rendering – also using the algorithm introduced by Karla and Barr – has been started by Richard Dumond and Kevin Novins, at iMAGIS, during summer 1994. It provided some of the ideas on which the optimization section is based.

I would like to thank Mathieu Desbrun, Nicolas Tsingos and Marie-Paule Gascuel for implementing the animation of the new implicit patches; Agata Opalach, and Jean-Christophe Lombardo for providing test images. And finally, a special thank goes to Marie-Paule Gascuel for multiple discussions and carefully re-reading this paper.

References

[1] B. Wyvill, C. McPheeters, and G. Wyvill. Animating soft objects. *The Visual Computer*, pages 235–242, August 1986.

[2] Agata Opalach and Steve Maddock. Implicit surfaces: Appearance, blending and consistency. In *Fourth Eurographics Workshop on Animation and Simulation*, Barcelona, Spain, 1993.

[3] Alex Pentland and John Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics*, 23(3):215–222, July 1989. Proceedings of SIGGRAPH'89 (Boston, MA, July 1989).

[4] Demetri Terzopoulos and Dimitri Metaxas. Dynamic 3-D models with local and global deformations : deformable super quadrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-13(7):703–714, July 1991.

[5] Brian Wyvill. Metamorphosis of implicit surfaces. *Modeling, Visualizing, and Animating with Implicit Surfaces (SIGGRAPH'93 course notes Number 25, Anaheim, CA, USA)*, August 1993.

[6] Marie-Paule Gascuel. An implicit formulation for precise contact modeling between flexible solids. *Computer Graphics*, pages 313–320, August 1993. Proceedings of SIGGRAPH'93.

[7] Mathieu Desbrun and Marie-Paule Gascuel. Highly deformable material for animation and collision processing. In *5th Eurographics Workshop on Animation and Simulation*, Oslo, Norway, September 1994.

[8] Mathieu Desbrun, Nicolas Tsingos, and Marie-Paule Gascuel. Adaptive sampling of implicit surfaces for interactive modeling and animation. In *Implicit Surfaces'95*, Grenoble, France, April 1995.

[9] Andrew Witkin and Paul Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics*, pages 269–278, July 1994. Proceedings of SIGGRAPH'94.

[10] Brian Wyvill and Geoff Wyvill. Field functions for implicit surfaces. *The Visual Computer*, 5:75–82, December 1989.

[11] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, pages 235–256, July 1982.

[12] Pat Hanrahan. Ray tracing algebraic surfaces. *Computer Graphics*, July 1983. Proc. of SIGGRAPH'83.

[13] Alan E. Middleditch and Kenneth H. Sears. Blend surfaces for set theoric volume modeling systems. *Computer Graphics*, 19(3), July 1985. Proceedings of SIGGRAPH'85.

[14] W. E. Lorensen and E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, July 1987. Proceedings of SIGGRAPH'87.

[15] Jules Bloomenthal. Polygonisation of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[16] Cornelia Zahlten and Hartmut Jurgens. Continuation methods for approximating isovalued complex surfaces. In *Eurographics'91*, pages 5–19, Vienne, Autriche, September 1991.

[17] J. Wilhelms and A. Van Gelder. Octree for faster isosurface generation. *Transactions on Graphics*, 11(3):210–227, July 1992.

[18] Paul Ning and Jules Bloomenthal. An evaluation of implicit surface tilers. *IEEE Computer Graphics and Applications*, 13(6), November 1993.

[19] D. Kalra and A. H Barr. Guaranteed ray intersections with implicit surfaces. *Computer Graphics*, 23:297–306, November 1989.

[20] Geoff Wyvill and Andrew Trotman. Ray tracing soft objects. In *Modeling, Visualizing and Animating with Implicit Surfaces*. SIGGRAPH Courses notes 90, 1993.

[21] Jules Bloomenthal and Brian Wyvill. Interactive techniques for implicit modeling. *Computer Graphics*, 24(2):109–116, March 1990.

[22] Craig Kolb and Rob Bogart. Rayshade 4.0. Available from `http://www-graphics-.stanford.edu/cek/rayshade/`, 1991.

[23] William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes in C, second edition*. Cambridge University Press, New York, USA, 1992.