SCAN-CONVERSION OF IMPLICIT SURFACES WITH LIPSCHITZ CONDITION

Huub van de Wetering, Martijn de Kort, Kees van Overveld

Eindhoven University of Technology, Department of Mathematics and Computing Science
wstahw@win.tue.nl

ABSTRACT

A scan-line rendering technique for implicit functions with the Lipschitz condition is described; it handles orthogonal projections and is based on computing the cycles in the intersection of the surface and a scan-plane. Infinite implicit surfaces and surfaces consisting of multiple parts are supported.
The technique is also viable for fast prototyping of implicit surfaces; for this application the objects are rendered as series of (partial) cycles which can be viewed interactively in 3-dimensions.

1. INTRODUCTION

Implicit surface modelling (See [2] for an overview) has become a powerful method for modelling geometric objects. Rendering techniques for implicit surfaces mostly consist of a slow polygonization phase followed by a standard scan-conversion of polygons. In this paper we propose a technique for directly scan-converting implicit surfaces with the Lipschitz condition. Earlier results on scan-conversion of implicit surfaces are available in [4] for quadratic surfaces and more recent in [9] for algebraic surfaces. These algebraic surfaces may be combined with CSG operators and the rendering is based on quickly finding for each scan-line the x-coordinates of the silhouette points and the x-coordinates of the intersection points of the basic surfaces in the CSG expression. This method differs completely from the approach taken here and it is applicable to a different domain of surfaces.

Both in [10] and [12] time coherence is used to facilitate interactive modelling of implicit surfaces. In [12] a particle system is used for visualisation, while in [10] the visualisation is done via a dynamic polygonization scheme. In section 6 we show that the scan-conversion technique can be adapted to supply a fast prototyping tool. The technique does not use time coherence and line segments are used for visualisation.

In section 2 the implicit functions we consider are introduced. In the section 3 we introduce the scan-conversion algorithm and elaborate on its parts. We give some improvements on the basic algorithm in 4. We present some examples in section 5 and give some conclusions and directions for future work in the sections 7 and 8, respectively.

2. LIPSCHITZ FUNCTIONS

Definition 1 An implicit surface is the collection of the solutions of an equation of the form: $f(p) = 0$.
An implicit volume is the collection of the solutions of an inequality of the form: $f(p) \geq 0$.

Definition 2 Lipschitz Function
A function f is a Lipschitz function if and only if $|f(q) - f(p)| \leq ||q - p||$, for all p and q.

Note that generally a more liberal definition of Lipschitz functions is used namely a λ exists such that $|f(q) - f(p)| \leq \lambda||q - p||$, for all p and q. Replacing $f(p)$ by $\lambda^{-1} f(p)$ results in f being replaced by a Lipschitz function with guaranteed factor 1 while the associated implicit surface remains the same.

An important class of Lipschitz functions is based upon distance-induced functions.

Definition 3 distance-induced function
A distance-induced function is a function based on the geometric distance to a skeleton S and it has the following form (where $r > 0$): $f(p) = r -$"minimal distance of p to any point on S"

Property 1 A distance-induced function is a Lipschitz function.

Proof 1 Let S be a skeleton object and let the distance-induced function f be given by $f(p) = r - d(p, S)$, for some $r > 0$ where $d(p, S) = min\{||p - q|| : q \in S\}$. Because of the triangular inequality, $||p - s|| \leq ||p - q|| + ||q - s||$, for all s in S. Hence, $d(p, S) \leq ||q - p|| + d(q, S)$. And since the last formula is symmetric in p and q, $|d(q, S) - d(p, S)| \leq ||q - p||$ and finally $|f(p) - f(q)| \leq ||q - p||$.

Simple examples of distance-induced functions have skeletons like a point or a line segment. In [5] it is shown that Lipschitz functions can be combined with CSG modeling and soft blending. These functions are used here. The unblended versions of the union, intersection, and difference operators are defined below. For the blended versions we refer to the original article.

Definition 4 CSG function
Given two distance-induced functions f and g, we define three CSG functions by:

i. $(f \cup g)(p) := max(f(p), g(p))$
ii. $(f \cap g)(p) := min(f(p), g(p))$
iii. $(f \setminus g)(p) := min(f(p), -g(p))$

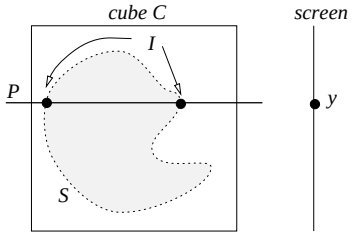

Figure 1: cross section of cube C showing scan-line y, scan-plane P, surface S, and intersection $I = S \cap C \cap P$

The observations in the following property were already made in [7]; both there and here the property is used to prune away empty regions of space. In [6] the property is used for fast ray-implicit surface intersection.

Property 2 Let f be a Lipschitz function. And let p be a point and Bp be the open ball with center p and radius $|f(p)|$.

i. An implicit surface S defined with f has NO overlap with Bp if $p \notin S$

ii. An implicit volume V defined with f contains Bp if $p \in V$

iii. An implicit volume V defined with f is disjoint with Bp if $p \notin V$

Proof 2 For all r in S ($f(r) = 0$), it holds according to the Lipschitz condition that $|f(p)| \leq ||p - r||$. From this observation (i) follows immediately.

If $f(p) < 0$ then $f(Bp)$ contains no elements with a different sign than $f(p)$ this is due to (i) and the continuity of f. Hence, (ii) and (iii) hold.

3. SCAN-CONVERSION

We consider an implicit surface S based on the Lipschitz function f. We generate an image of $S \cap C$ with C a given cube. This image is formed by orthogonal projection of S on the screen $z = 0$.

The image of S will be rendered scan-line by scan-line (see algorithm 3). In order to do so the intersection I of S, C and P is considered where P is a so-called scan-plane, a plane orthogonal to the screen and containing a scan-line (see figure 1). The intersection I is a set of cycles or partial cycles. In order to render I it is approximated by line segments that are ultimately rendered in a one-dimensional z-buffer.

3.1. approximating a 2D implicit curve

Several techniques for approximating an implicit curve or surface exist. One of the most used algorithms is Marching Cubes [13, 8]. Here we search for an efficient algorithm that can approximate all (partial) cycles with line segments; the number of line segments should be preferably low and hence the algorithm should be adaptive in some way. Marching Cubes, however, is non-adaptive and moreover assumes

```
for each scanline y
    P := "scanplane of scanline y"
    I := C ∩ S ∩ P;
    "approximate I with line segments";
    "render line segments in a z-buffer"
```

Algorithm 1: scanline-rendering of implicit surface S in cube C

that the object is connected. Other techniques for approximating implicit curves or surfaces are based on hierarchical subdivisions [1, 7, 11]. As in [11] we will use a quadtree. The adaptiveness is obtained by conditionally dividing the nodes of the tree based upon local geometry of the implicit function. The algorithm for one scan-line is shown in al-

```
"generate quadtree Q";
for each leaf l of Q
  if l is a straddling node then
    "approximate l with line segment(s)"
```

Algorithm 2: approximation of implicit curve with line segments

gorithm 3.1. First a quadtree is generated and next for each straddling leaf of the quadtree line segments are generated for approximating the implicit curve. A straddling node is a node with at least one side with alternating signs in its endpoints.

In the next two sections the quadtree generation and the approximation with line segments are explained further.

3.1.1. quadtree generation

We need subdivision criteria with which we can decide for a (square) node N of the tree if subdivision is needed. The length of the sides of N is r. The function values from N in its corner points are: f_{ll}, f_{lr}, f_{ur}, and f_{ul} and in its center point f_c.

maximal tree depth Do not divide when the maximal recursion depth $maxdepth$ has been reached.

Lipschitz condition From property 2 an easy criterion follows immediately. If $|f_c| > \frac{1}{2}\sqrt{2}r$ the node is either completely inside or completely outside of the implicit curve. In either case no subdivision is needed.

minimal tree depth Divide when the minimum recursion depth $mindepth$ has not been reached.

uncertain situation Divide when four sign alterations occur on the sides of the node: $|sign(f_{ul}) - sign(f_{ll}) + sign(f_{lr}) - sign(f_{ur})| = 4$ where $sign(x)$ is -1 if $x < 0$ and 1 otherwise.

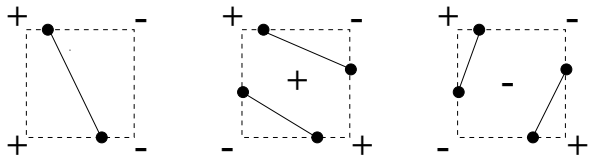

Figure 2: line segments in straddling nodes

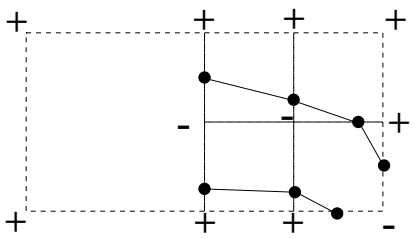

Figure 3: crack problem

tolerance Divide if information of function values in corners is insufficient for accurate approximation of the curve. We use the following heuristics: if the function value of the implicit function on more than, say α, of the length of a side is not known to be different from zero, the side will be subdivided. For side *ul-ll* this can be formulated as: $r - |f_{ul}| - |f_{ll}| > \alpha$. Hence, details smaller than size α may be missed.

maximal normal vector deviation At places of high curvature the curve needs to be sampled more accurately. This is obtained by dividing further when the angle between a gradient in an end point of a proposed line segment and the normal of the line segment is larger than a given angle ϕ.

3.1.2. line segments in straddling nodes

Let N be a straddling node. That means that on either two or four sides of the node the implicit function changes its sign. For each of the sides on which a sign change occurs the zero point of the function is approximated. If two sign changes occur the corresponding zero points are connected with a line segment. If four sign changes occur the sign of the function value in the center of the node is used to resolve the ambiguity as is shown in algorithm 3.1.2 and in figure 2. The function `zero(p,q)` returns a point on the line segment pq where the implicit function vanishes. The array of points `p[]` contains afterwards either 2 or 4 points; the line segments in the node are `p[0]p[1]` and if applicable also the line segment `p[2]p[3]`.

3.1.3. crack problem

Strictly using the subdivision criteria as stated above would result in a quadtree where neighboring nodes that contain part of the curve may have different levels of subdivision.

```
point p[4]; i=0;
sul=sign( f_ul ); sur=sign( f_ur );
sc =sign( f_c );
sul=sign( f_ll ); sll=sign( f_lr );

if sul!=sur then
   p[i] := add(ul,ur); i:= i+1;
if sc!=sul then
   if (sul!=sll)) then
      p[i] := zero(ul,ll); i := i+1;
   if (sur!=slr) then
      p[i] := zero(ur,lr); i := i+1;
else
   if (sur!=slr)) then
      p[i] := zero(ur,lr); i := i+1;
   if (sul!=sll)) then
      p[i] := zero(ul,ll); i := i+1;
if (sll!=slr)) then
   p[i] := zero(ll,lr); i := i+1;
```

Algorithm 3: generation of line segments in a leaf

This might result in so-called cracks. Cracks reveal themselves by missing line segments. (see figure 3). This problem can be solved by, if necessary repeatedly, traversing the tree and subdividing nodes that form a crack with one of their neighboring nodes.

4. IMPROVEMENTS

In this section we give some directions for improving the scan-line algorithm and we indicate which improvements are currently implemented and which will be implemented later on.

The efficiency of the scan-line algorithm is determined for a great deal by the number of function evaluation that are needed. In order to reduce the costs involved with function evaluation two directions can be followed: faster function evaluations and less function evaluations.

faster function evaluations The scan-line algorithm given above was tested with the implicit functions defined in [5]. These functions are distance-induced functions that are combined in CSG expressions using blending operators. The resulting functions are Lipschitz functions with factor 1. The evaluation speed of these functions can be greatly improved by lazy evaluation techniques: a function value in a subexpression is only evaluated when it is suspected to contribute to the function value of the whole expression. For each subexpressions f is given a bounding box Bf. Let f_{Bf} be an implicit function with Bf as implicit volume and with $f_{Bf}(p)$ the signed distance from p to Bf. Bounding boxes Bf are constructed such that $f(p) \leq f_{Bf}(p)$. Figure 4 shows an example where lazy evaluation is applicable: the function value in point p is to be computed; according to defi-

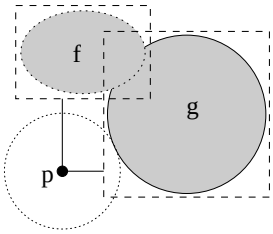

Figure 4: lazy evaluation of CSG function $f \cup g$

nition 4 the value is given by $(f \cup g)(p) = min(f(p), g(p))$. First $g(p)$ is evaluated since the bounding box of g is closer to p than the bounding box of f. And finally, there turns out to be no need to evaluate $f(p)$ since $g(p) \geq f_{Bf}(p) \geq f(p)$. The lazy evaluation technique uses axis parallel bounding boxes for each subtree of the CSG-function. These bounding boxes can be computed following some simple rules. In this form lazy evaluation has been implemented.

A further, yet unimplemented, improvement uses the S-bounds method of [3] for finding tighter bounding boxes.

less function evaluations There are at least two ways to reduce the number of function evaluations:

i. The current algorithm does not exploit the coherence between successive scan-lines. Hence, incrementally computing the quadtrees is an option: two consecutive quadtrees are expected to have more or less the same structure and the function values in the computed nodes are more or less the same. This last remark can be clarified by noting that since f is a Lipschitz function $|f(x, y, z) - f(x, y+1, z)| < 1$. Furthermore it is only at the lowest level of the quadtree that we require exact function values.
   An other way of exploiting spatial coherence is by building an octree of the surface to a certain level and use this octree in order to quickly find empty nodes in the quadtrees.
   Neither of these two coherence options has been implemented yet.

ii. Currently the complete intersection of the cube with the surface and a scan plane is computed and later projected on the screen. Instead of using a z-buffer we can use the position of the node from which a line segment originated to sort line segments. It is clear that for scan conversion then only the front part of a quadtree needs to be generated. See figure 5 for an illustration of this optimization. This optimization is currently not implemented.

In [14] blended and unblended CSG primitives are combined. Especially unblended CSG primitives are problematic since inaccurate rendering of the intersection curves of two primitives is distracting. The solution for this followed in [14] is also valid here: whenever a line segment combines two points of from different primitives, care must be taken to find a good approximation of the point of intersection of the two primitives within the node. This technique can be used here to reduce the need for a large maximum depth of the quadtree. In the current implementation this is not implemented yet and hence for an accurate rendering of unblended CSG primitives a high maximum depth has to be used.

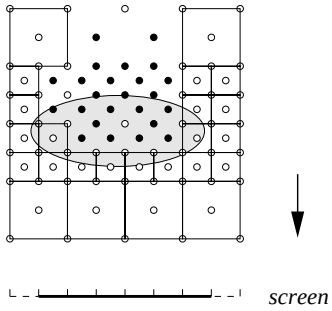

Figure 5: partially generated quadtree

A final improvement replaces orthogonal projection by perspective projection. This can be done in two ways. Firstly, the computed line segments can be projected using a perspective projection. This method is used in section 6. Secondly, the entire implicit surface can be transformed such that an orthogonal projection is the perspective projection of the original surface. In the last case care must be taken that the resulting implicit function still has the Lipschitz condition. The second method is not implemented.

5. EXAMPLES

Figures 6 to 11 show some examples; all of them are 256 x 256 pixels. The figures are rendered using environment mapping. Below a table is given with the execution times in CPU seconds for a Sun ultra enterprise 3000.

	figure	seconds	#primitives
intersection of 2 tori	9	3.8	2
torus	6	6.6	1
difference of 2 tori	7	8.2	2
union of two tori	8	12.3	2
cube with holes	11	13.5	4
walker	10	19.7	14

All the figures use $maxdepth = 7, mindepth = 3, \alpha = 3$ and $\phi = 17$. Reducing the $maxdepth$ to 6 reduces the execution times typically with a factor two, e.g. for the torus example it takes 3.5 seconds with $maxdepth = 6$ without any visual effect on the end result. However examples containing details with high curvature, like the difference of two tori, are no longer rendered accurately at these details if $maxdepth = 6$.

6. FAST PROTOTYPING

The scan-conversion algorithm as discussed above computes a set of line segments foreach scan-plane; the segments are rendered in a z-buffer. The segments form a 3-dimensional approximation of the implicit surface and can easily be rendered and viewed from different directions in a standard viewing tool. For an interactive modelling tool of implicit surfaces, however, the generation of the approximation is in general too slow. But a practical and interactive modelling scheme can be imagined as follows: instead of generating the line segments in each scan-plane, generate only line segments once in every n planes. This will speed up the approximation proces by a factor n. Figure 12 shows a (rotated) approximation of a table where $n = 4$. This 3-dimensional model was generated in approximately 2 seconds. This way of modelling turns out to be practical; when designing global effects like positioning primitives the fast prototyping method is fast and mostly accurate enough; when details or final results must be displayed the aforementioned scan-conversion algorithm is used.

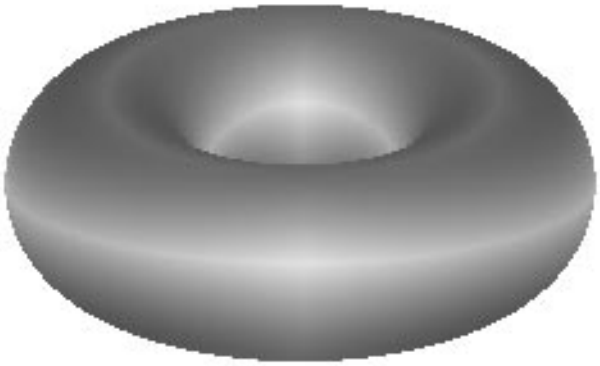

Figure 6: torus

7. CONCLUSIONS

We have given an algorithm for directly scan-converting implicit surfaces with Lipschitz condition using an orthogonal projection. The algorithm can handle infinite surfaces and surfaces consisting of multiple parts. Several ways for improving the algorithm are given. Furthermore, it can be used for fast prototyping by producing a set of line segments approximating the implicit surface.

8. FUTURE WORK

Apart from the suggestions for improvements in section 4, we aim to work at a polygonization algorithm based on connecting cycles, on a collision detection algorithm using the lazy evaluation techniques combined with careful selections of bounding volumes, and on enhancing the scan-conversion algorithm and the fast prototyping using time coherence.

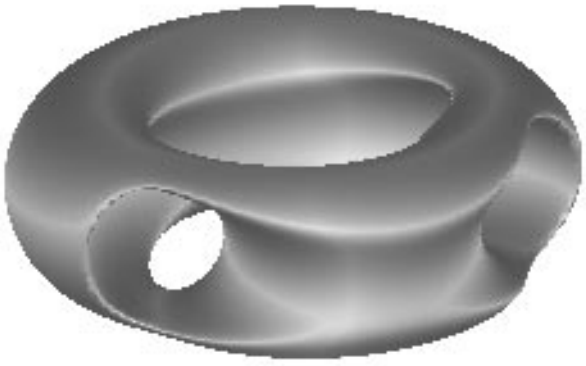

Figure 7: blended difference of two tori

9. REFERENCES

[1] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5(4):341–356, 1988.

[2] Jules Bloomenthal, Chandrajit Bajaj, Jim Blinn, Marie-Paule Cani-Gascuel, Alyn Rockwood, Brian Wyvill, and Geoff Wyvill, editors. *Introduction to Implicit Surfaces*. Series in Computer Graphics and Geometric Modeling. The Morgan Kaufmann, 1997. ISBN 1-55860-233-X.

[3] Stephen Cameron. Efficient bounds in constructive solid geometry. *IEEE Computer Graphics and Applications*, 11(3):68–74, May 1991.

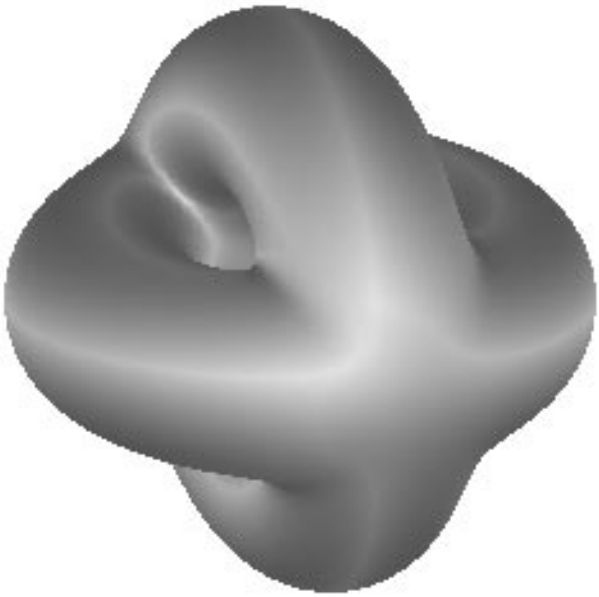

Figure 8: blended union of two tori

Figure 10: walker

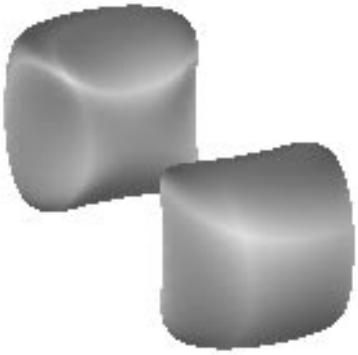

Figure 9: blended intersection of two tori

Figure 11: cube with three cylinders subtracted

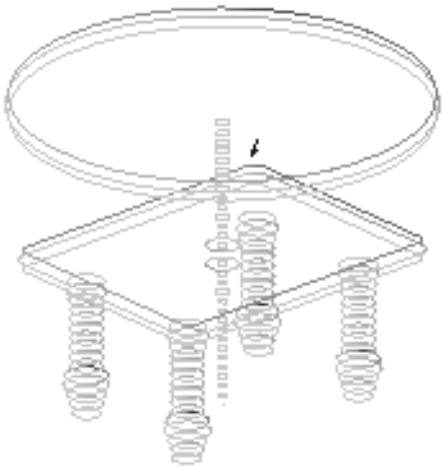

Figure 12: a (rotated) 3-dimensional approximation with $n = 4$

[4] J. Roy Davis, Roger Nagel, and Walter Guber. A model making and display technique for 3-D pictures. pages 47–72, October 1968.

[5] Daniel Dekkers, Kees van Overveld, and Rob Golsteijn. Combining CSG modeling with soft blending using Lipschitz-based implicit surfaces. *to be published*.

[6] John C. Hart. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(9):527–545, 1996. ISSN 0178-2789.

[7] Devendra Kalra and Alan H. Barr. Guaranteed ray intersections with implicit surfaces. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 297–306, July 1989.

[8] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[9] Thomas W. Sederberg and Alan K. Zundel. Scan line display of algebraic surfaces. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 147–156, July 1989.

[10] Barton T. Stander and John C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 279–286. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[11] Gabriel Taubin. Distance approximation for rasterizing implicit curves. *ACM Transactions on Graphics*, 13(1):3–42, January 1994. ISSN 0730-0301.

[12] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 269–278. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[13] Brian Wyvill, Craig McPheeters, and Geoff Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

[14] Brian Wyvill and Kees van Overveld. Polygonization of implicit surfaces with constructive solid geometry. *to be published*.