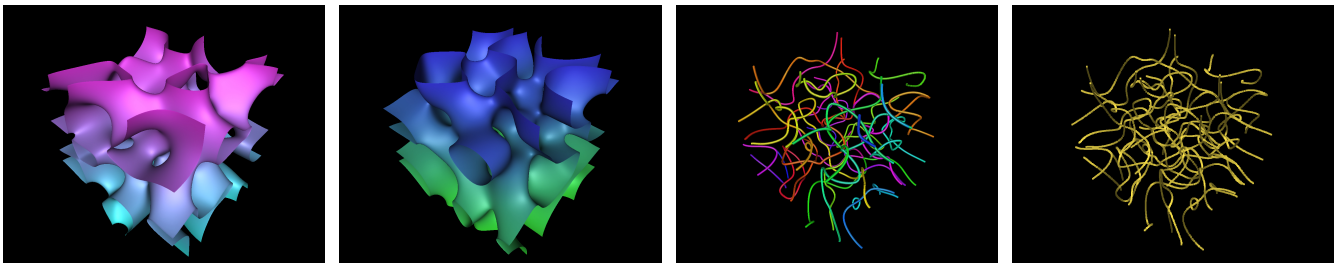
Extraction of Intersection curves from Iso-surfaces on co-located 3D grids

Patric Ljung <plg at itn.liu.se> Anders Ynnerman <andyn at itn.liu.se>

Scientific Visualization Group, Department of Science and Technology, Linköping University



(a) Surface of real field, 310627 triangles

(b) Surface of imaginary field, 308859 triangles

(c) Coloured intersection curves, 17104 line segments, rendered using 4-sided tubes

(d) Intersection points, 6615 points, rendered using tangent shaded points

Figure 1: The intersection of two iso-surfaces resulting in intersection curves and intersection points using the Marching Faces method. The curves represent nodal lines in quantum chaos. In general, bi-isolines are extracted from two scalar fields. The volume size is 128^3 for this chaos illustration, see section 8.

Abstract

This paper presents new methods for efficient extraction of intersection curves between iso-surfaces of any pair of co-located 3D scalar fields. The first method is based on the Marching Cubes algorithm which has been enhanced to produce an additional data structure that makes it possible to reduce the complexity of the general surface intersection extraction from $\mathcal{O}(N^2)$ to $\mathcal{O}(\sqrt{N})$, where N denotes the number of triangles in the arbitrary surfaces. The second method directly extracts the intersection lines based on finding intersection points on the faces of the voxels for two iso-surfaces extracted from a regular grid. A simple classification scheme is used for early termination of testing of voxels that are not intersected by both surfaces.

Also presented is an efficient method for fast curve generation through combination of line segments resulting from the explicit surface intersection method. An indexing structure is used to accelerate access and matching of intersection line segments to be combined into closed or open curves.

The presented methods have been used to identify and visualize nodal lines in 3D quantum and wave chaos data. These data are represented by a volume of complex values and a nodal line is a connected curve where the complex iso-value $z_{iso} = 0 + i0$. This type of chaos is believed to represent physical phenomena present in, for example, quantum mechanics, microwaves, fibre optics, and acoustics.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Boundary representations, Geometric algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures

Keywords: Intersection curves, Isosurfaces, Surface intersection, Feature Detection, Nodal lines visualization, Complex 3D fields

1 Introduction

Finding the intersection of surfaces is of importance in many application areas within mathematics and science. The meaning of the intersections depends on the application field. In most fields a topological sorting of the identified intersection segments is essential so that a set of distinguishable closed and open curves can be generated. The starting point for this investigation has been taken in the intersection of surfaces in chaotic quantum mechanical systems and results are presented in the context of the problems posed by this application. The work is focused on the intersection of iso-surfaces generated from co-located grids. Two alternative methods have been developed to identify the intersection of these surfaces. The first method uses explicitly created surfaces by means of iso-surface extraction algorithms, e.g. the Marching Cubes algorithm [Lorensen and Cline 1987; Montani et al. 1994]. This method is used when the surfaces as well as the intersection curves are of interest for visualization. With a minor enhancement the Marching Cubes algorithm enables efficient computation of the surface intersection with a complexity of $\mathcal{O}(\sqrt{N})$ where N is the average number of polygons in the surfaces. The general surface intersection problem is of complexity $\mathcal{O}(N^2)$.

As a second stage to the first method, line segments from the explicit surface intersection are combined into curves. For the type of data under consideration and for the purpose of this visualization, there is a need to identify the different curves. By colouring the distinct curves differently the interpretation of the generated images is improved.

The second method, named Marching Faces (MF), seeks to directly extract the intersections by implicitly considering the surfaces to intersect on the voxel faces in the sampled fields. The latter method avoids the memory consuming generation of polygons to represent the surfaces and only produces a significantly smaller point set for the intersection of the two surfaces and the voxel faces. It uses a simple voxel classification scheme to early detect voxels not intersected by both surfaces in order to avoid further testing. Both methods use efficient data structures for fast lookup of line segments, triangles, and intersection points.

The first method, using the Enhanced Marching Cubes (EMC)

algorithm, is preferred when the visualization of the surfaces is of interest as well as the intersection curves. The intersection algorithm then works with little additional cost. The MF algorithm is faster than EMC and thus preferred when surfaces need not be visualized. Both EMC and MF must visit all the voxels to ensure that all surface patches and intersections are detected, which constitute a fundamental problem since it grows with the volume size. Many methods have been proposed to speed-up iso-surface generation by using additional data structures, for example octrees [Wilhelms and Gelder 1992] or span-space [Livnat et al. 1996]. However, to the best of the authors' knowledge, in order to produce these structures, all voxels must be visited. For interactive pipelined processing of time-varying data sets, such pre-processing is not beneficial unless the accelerating structures can be used repeatedly. Or the following process otherwise would be of a complexity order larger than that of the volume traversal. Volume traversal is, in most cases, the most resource consuming operation but cannot be avoided in the proposed context of interactive pipelined processing of time-varying data sets, i.e. computational steering or visualization of raw, large scale, time-varying data sets.

2 Related work

Several methods for finding the intersection between surfaces have previously been proposed, in particular, the intersection of parametric surfaces have attracted extensive work [Patrikalakis 1993; Sabharwal 1994; Krishnan and Manocha 1997].

The general surface intersection problem is, in the naïve approach, an $\mathcal{O}(N_1 N_2)$ problem, or $\mathcal{O}(N^2)$ if $N \simeq N_1 \simeq N_2$, that is, all elements (polygons) must be tested against each other. By using subdivision methods or hierarchical methods it is possible to avoid intersection testing for patches of surfaces whose bounding volumes do not intersect. Extensive work has been dedicated to efficient extraction of iso-surfaces [Wilhelms and Gelder 1992; Livnat et al. 1996].

Previous work related to the Marching Faces algorithm is the 3D Marching Lines algorithm [Thirion and Gourdon 1996]. In this algorithm random seeds can be automatically placed in voxels of the volume. When a seed detects an intersection curve passing through a voxel, this curve is traced. To ensure detection of all curves, all voxels can also be searched. Even though the The Marching Lines algorithm is similar to the approach in the presented Marching Faces method it differs in some key aspects. The 3D Marching Lines method traces a curve segment when it is encountered. The Marching Lines algorithm determines the intersection points on the edges of the polygons from the first surface by interpolation whereas the Marching Faces method solves the intersection of the voxel-face/iso-surface intersection lines. The MF method also uses a classification scheme to entirely eliminate setting up any surface/voxel-face intersection lines unless both surfaces intersect a voxel cube.

3 Definitions

A scalar field ψ is defined as $\psi : \mathbb{R}^3 \rightarrow \mathbb{R}$. A subscript notation is used to refer to other data sets or variables that are unique to a specific scalar field ψ_i. For instance T_1 is the triangle set for a surface generated from ψ_1.

The angle notation $\langle\ x_i \mid i = 0, 1, \ldots, 7\ \rangle$ is used to denote a vector. A simplified notation $\langle\ x_i\ \rangle_N$ having $i = 0, 1, \ldots, N-1$ is also used. The cardinality $|S|$ of a set S is the number of elements in the set.

Set	0	1	2	3	4	5	6	7
Real	88.13	3.12	6.09	2.17	0.49	0.00	-	-
Imag	88.07	3.06	6.24	2.15	0.48	0.00	-	-
Lines	99.20	0.15	0.24	0.26	0.12	0.03	0.00	0.00

Table 1: Histograms over percentage of voxels with a specific number of primitives, triangles for the surfaces and line segments for lines. Results are based on a 128^3 cube, e.g. 2048383 voxels.

4 Optimized Surface Intersection

This section describes the Enhanced Marching Cubes (EMC) method, the following intersection algorithm, and the method to combine intersection line segments into topologically sorted curves.

Under the condition that iso-surfaces are requested, the information obtained through the process of iso-surface extraction can be reused to significantly speed-up intersection testing. A few key properties of the triangles generated from the Marching Cubes algorithm is observed.

1. Any triangle can be uniquely located inside only one voxel.
2. The number of triangles in one voxel is a small number between 0 and 5. The vast majority of voxels have 1 or 2 triangles if and only if a surface intersects it.
3. Triangles can be enumerated in monotonically increasing numbers in the voxel traversal order.
4. Most voxels have no triangles at all. For the case shown in table 1, it is found that 88% of the voxels are empty.

These properties are exploited to optimize the surface intersection task. In essence, a subdivided space based on the grid on which the scalar field is sampled is obtained. For each voxel, a maximum of five against five triangles need to be tested for intersection. A voxel without triangles from both surfaces can be directly skipped. Table 1 shows some statistics for the distribution of elements-per-voxel.

4.1 The Enhanced Marching Cubes algorithm

The Marching Cubes algorithm is a well established method for extracting iso-surfaces from volumetric data. Triangles produced are uniquely defined within one voxel. Thus, for any co-located volume, testing of the triangles can simply be done on a per-voxel basis. This can be done immediately by traversing the two volumes simultaneously or by storing an indexing data structure to be used in a second stage. The advantages with the latter method are: The two surfaces can be computed in parallel if multiple CPUs are available. The Marching Cubes code requires only a very simple and restricted enhancement to support this case. Multiple sets of surfaces can be intersected without repeated traversal of the volume data.

The Marching Cubes algorithm is enhanced to produce an additional data structure, a triangle index structure I_T with the same dimensions as the processed volume ψ_i. When the MC algorithm traverses the volume it occasionally produces triangles and stores them in a triangle list $T = \langle\ t_j\ \rangle_N$. For each processed voxel k, any new triangles are appended to the triangle list T. The Enhanced Marching Cubes version then also stores the new size $|T|$ of the triangle list in a triangle index table I_T, $I_T(k) \leftarrow |T|$.

After traversing the full volume, the triangle index structure I_T can be used to query the presence of triangles in a specific voxel and to retrieve the index of the first triangle in that voxel. For two voxels in traversal order sequence, $k-1$ and k the number of triangles n in voxel k is extracted from I_T by

Algorithm: SURFACEISECT
Input: $I_{T_1}, I_{T_2}, T_1, T_2$
Output: L, I_L

```
1   L ← ∅
2   for each voxel k do
3     if I_T1(k−1) = I_T1(k) ∨ I_T2(k−1) = I_T2(k)
4       then continue with next voxel
5     for i = I_T1(k−1) to I_T1(k) − 1 do
6       for j = I_T2(k−1) to I_T2(k) − 1 do
7         APPEND(L, T1(i) ⋂ T2(j))
8       end for
9     end for
10    I_L(k) ← |L|
11  end for
12  return L, I_L
```

Table 2: Pseudo-code for algorithm SURFACEISECT that performs the surface intersection operation.

$$n = I_T(k) - I_T(k-1) \tag{1}$$

with the following boundary conditions for I_T

$$I_T(k) = 0 \qquad k < 0$$
$$I_T(k) = |T| \quad k \geqslant N_{\text{voxels}}$$

After application of the EMC algorithm on two selected scalar fields the output is passed to the surface intersection method.

4.2 Explicit surface intersection

The key element of an efficient surface intersection algorithm is to reduce the number of (triangle) intersection tests. By using the triangle index structure I_{T_i} the proposed method effectively reduces the number of tests. Given the inputs I_{T_1}, I_{T_2}, T_1, and T_2, the algorithm SURFACEISECT is outlined in table 2.

For each voxel k the presence of triangles in both surfaces is determined. If a voxel contains triangles from both surfaces, each pair of triangles are tested for intersection. For each intersection a line segment is generated and stored in the line segment list, L. When all triangle pairs in a voxel have been tested the current size of the line segment list L is assigned to the line segment index table, I_L, similar to the triangle index table. The further use of this structure is described in section 5.

For each voxel tested, a maximum of 25 triangle pairs need to be tested for intersection, typically 2–3 triangle pairs are tested according to table 1. For two arbitrary surfaces it is shown this intersection method works in $\mathcal{O}(\sqrt{N})$ where N is the number of triangles in a surface, e.g. $N = |T|$, see appendix A for details.

5 Combining line segments

The triangle intersection algorithm SURFACEISECT generates a list L of line segments and a line index data structure I_L. The line index data structure I_L provides fast look-up ($\mathcal{O}(1)$) of the presence of line segments within a local neighborhood.

The algorithm COMBINELINESEGS (table 3) combines line segments into closed or open curves by joining end-points of line-segments and creating lists of points. Each list defining a unique distinguishable curve. This enables an improved visual cue to perceive the different curves by using separate material properties, e.g. colour, over unrelated rendering of all segments with one and the same material property.

By iterating over all the line segments, nearby line segments are selected from the indexing set I_L. For each pair of these nearby line segments the algorithm matches end-points and selects the closest point to join. Since two end-points of two segments will match exactly, within rounding error, an alternative approach could be to stop searching when a match below a small threshold is reached. However, since line segments can be located arbitrary close to voxel vertices, within rounding error, the implemented method finds the point with the smallest error even for such degenerate cases. A threshold is also introduced to establish a maximum distance to allow connection of two end-points. Thus, this constitutes a greedy algorithm that always picks the best/closest end-point.

Algorithm: COMBINELINESEGS
Input: L, I_L
Output: CS

```
1   U(∗) ← ⟨ −1, ∗ ⟩,   CS ← ∅
2   for each line segment i ∉ U do
3     i_F ← i,   i_L ← i,   m ← 1,   m′ ← 1
4     do  [Backward trace]
5       ⟨ i′, m ⟩ ← FINDCLOSESTPOINT(i_F, 1 − m, I_L, U)
6       if i′ ≠ NIL then
          U(i′) ← ⟨ i_F, m ⟩
7         i_F ← i′
8       end if
9     while i′ ≠ NIL ∧ i_F ≠ i_L
10    if i_F ≠ i_L then
11      do  [Forward trace (open curves)]
12        ⟨ i′, m ⟩ ← FINDCLOSESTPOINT(i_F, m′, I_L, U)
13        if i′ ≠ NIL then
14          U(i_L) ← ⟨ i′, m′ ⟩
15          i_L ← i′,   m′ ← 1 − m
16        end if
17      while i′ ≠ NIL ∧ i_F ≠ i_L
18    end if
19    APPEND(CS, LINESEGSTRACE(L, i_F, U))
20  end for
21  return CS
```

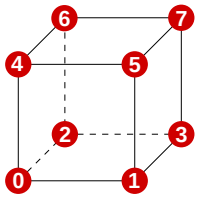
Table 3: Pseudo-code for algorithm COMBINELINESEGS.

The already used line segments are marked to prevent multiple connections and further matching. The used segments are marked in a set U. If segment a is linked to segment b the used set U is assigned $U(a) \leftarrow b$. After matching all line segments, U contains a linkage of points from the combined line segments.

The trace process is continued until the current curve reaches the boundary or connects back to itself, i.e. forms a closed curve. The algorithm first traces the first end-point backwards and identify the first segment index by i_F. If the curve is closed, this tracing ends when the initial segment is reached. Otherwise it continues to trace forward and updates i_L to indicate the last segment index. The traversal of L then continues to trace new curves among the line segments not already used. Note that the voxel is not marked as used, but rather the line segments are, since multiple curves can cross a voxel.

The algorithm finally outputs a list of curves, a curve list CS, each element of this list is a closed or open curve C. A curve consists of a list of curve points. A detailed view of the algorithm COMBINELINESEGS is presented in pseudo-code in table 3. The function FINDCLOSESTPOINT returns the closest matching point in the unused line segments within a local neighborhood. A line segment $L(i)$ has a start and end point, denoted $L(i)_0$ and $L(i)_1$, identified in the pseudo-code by variables m and m'.

Finally, the subroutine LINESEGSTRACE follows the linkage in the used segments set, U, beginning at the first segment index i_F and generates a list of curve points. The curve is defined as 'closed' if $i_F = i_L$, otherwise it is 'open'.



Face	No.
Left	0
Right	1
Front	2
Back	3
Bottom	4
Top	5

Figure 2: Voxel vertices and face numbering.

Algorithm: MARCHINGFACES
Input: V_1, V_2, iso_1, iso_2
Output: P, I_P

```
1   P ← ∅
2   for each voxel k do
3     b1 ←VOXELCONFIG(V1(k), iso1)
4     b2 ←VOXELCONFIG(V2(k), iso2)
5     f1 ← FaceTable[b1]
6     f2 ← FaceTable[b2]
7     f ← f1 & f2 & FACECHECK(k)
8     for each face j in f(j) ≠ 0 do
9       c1 ←FACECONFIG(V1(k), j, iso1)
10      c2 ←FACECONFIG(V2(k), j, iso2)
11      l1 ←SETUPLINES(V1(k), c1)
12      l2 ←SETUPLINES(V2(k), c2)
13      P' ←INTERSECTLINES(l1, l2)
14      APPEND(P, P')
15    end for
16    IP(i) ← |P|
17  end for
18  return P, IP
```

Table 4: Pseudo-code for algorithm MARCHINGFACES.

6 Implicit intersection of two iso-surfaces

The second method for finding the intersection of two iso-surfaces is based on directly extracting points of intersection from two co-located scalar fields. The creation of surface elements is avoided and computation is minimized by quickly discarding voxels and voxel faces where the two surfaces do not intersect. The points of intersection are defined by the intersection of two iso-surfaces and voxel faces. The authors have called this method Marching Faces due to its similarity to the Marching Cubes algorithm and its focus on intersection points on voxel faces.

The Marching Faces algorithm starts with two scalar fields ψ_1 and ψ_2 and traverses the voxels of these fields concurrently. The pseudo-code for MARCHINGFACES is found in table 4. Each voxel is classified by constructing a voxel configuration vector b (an unsigned byte). Equation 2 defines this vector, having the values d_i at the vertices and d_{iso} is the queried iso-value. See figure 2 for the numbering of vertices and faces.

$$b = \langle\, d_i < d_{\text{iso}} \,\rangle_8 \tag{2}$$

In the pseudo-code this operation is performed by the subroutine VOXELCONFIG.

To simplify the loop construction for handling of the boundary voxels where two opposing faces need checking, a function FACECHECK that returns a binary vector for ruling out testing of faces 1 (right), 3 (back), and 5 (top) for interior voxels.

Two voxel configuration vectors, b_1 and b_2, are evaluated from the two fields. Depending on the configuration, certain faces can potentially hold an intersection between the surfaces. A face with a mixed combination of vertices below and above the iso-value has the surface intersecting it. The possible voxel faces are encoded in a 256 byte array, **FaceTable**, for quick lookup of potential faces. By using the face-numbering in figure 2 it requires six bit codes. The bit-wise binary-and operator (&) on the face vectors from each voxel yields non-zero bits identifying faces that might hold a surface intersection point since the two surfaces intersect the same voxel face. For each of these faces the face configuration, c, is determined and encoded in four bits as illustrated by equation 3. This operation is carried out by the subroutine FACECONFIG.

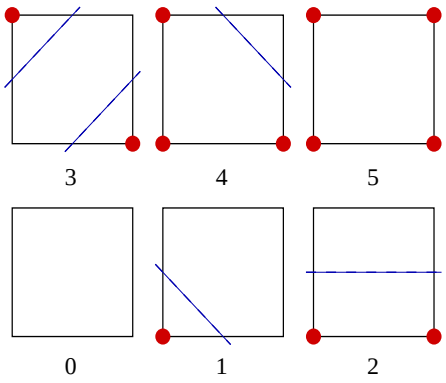


Figure 3: Face configurations. Red dots indicate a value below the iso-value. The blue dashed lines is the topological alignment of the iso-surface intersection.

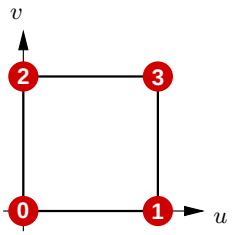


Figure 4: Numbering of the face's vertices and alignment on the parameter axis u, v.

$$c = \langle\, d_i < d_{\text{iso}} \,\rangle_4 \tag{3}$$

where the values d_i at the vertices i of the face are numbered as in figure 4. Figure 3 shows the unique six possible configurations of a face. By rotation all 16 combinations can be obtained. Each dashed line represents the topological location of the iso-surface. The points, p_j, of intersection on the edges with the corresponding iso-surface is determined by means of linear interpolation. Every pair of points, p_j and p_{j+1}, then identify an iso-line segment on the face. These lines are generated in the subroutine SETUPLINES. In face configuration 3, four intersection points are determined. This case represents an ambiguous configuration. The interpolated gradient orientation can be used to deterministically determine the best choice in this case.

By testing each pair of iso-line segments for the two fields on a face for intersection, one or two points on a face can be found to identify intersection points between iso-surfaces and the voxel face. Intersection curves thus pass through these points. The points are first defined in the local u, v coordinate system. The subroutine

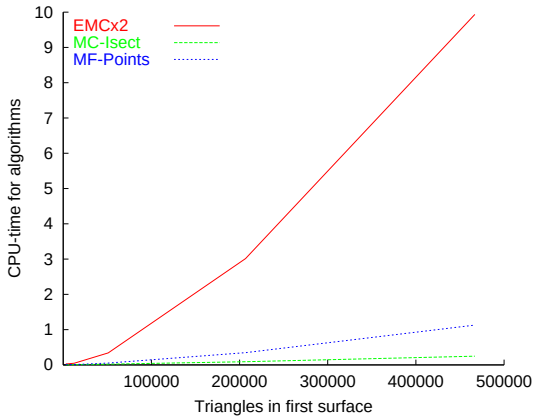


Figure 5: Execution times for the Enhanced Marching Cubes algorithm for two surfaces (EMCx2), Triangle intersection algorithm (MC-Isect), and the Marching Faces algorithm (MF-Points) as a function of the number of polygons in the first iso-surface. The variations of surface sizes have been generated by varying the sampling cube size. Tests were performed on a Laptop PC with Intel Pentium III CPU at 1 GHz and 512 MB of internal memory.

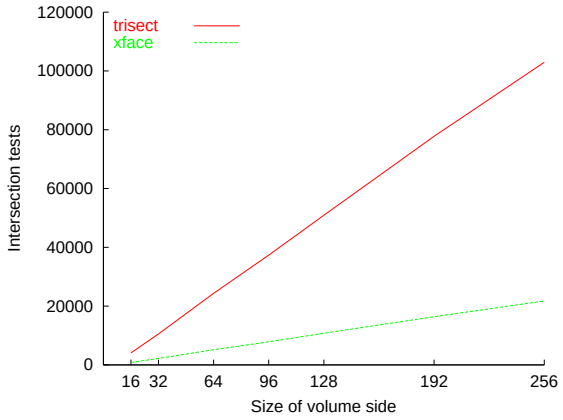


Figure 6: Performed intersection tests as a function of the side of the volume. The label 'trisect' refers to the triangle intersection algorithm using the triangles from two EMC passes, the graph specify number of triangle intersection tests. The label 'xface' refers to the Marching Faces algorithm, the graph specify the number of intersection tested voxel faces having a sign change at the vertices. Both methods are linear to the side of the volume, however, the MF method has a smaller constant in the ordo-notation making it faster.

INTERSECTLINES finds all line segments in l_1 intersecting with a line segment in l_2. The intersection points identified are returned in P', along with the computed tangential directions, as described below.

The gradient $\nabla\psi$ at u, v is estimated by bi-linear interpolation of approximated gradients at the vertices of the face. The gradients at the vertices are approximated by central difference, see equation 4, or a partially single sided difference at the volume boundary.

$$(\nabla\psi)_i(\mathbf{r}) = \frac{1}{2\delta}\Big(\psi(\mathbf{r}+\delta\hat{e}_i) - \psi(\mathbf{r}-\delta\hat{e}_i)\Big), \quad i = 1,2,3 \quad (4)$$

By taking the cross-product of the normalized gradients from the two iso-surfaces an estimate of the curves tangential direction is obtained at the point of intersection. For two almost co-planar surfaces the magnitude of the cross-product becomes very small, this measure can be used to place a weighting or confidence on the tangential direction. The estimated tangent can also be used as the direction of the curves' first order derivative for higher order curves. However, in the current implementation the tangent is only used for tangent shading [Zockler et al. 1996] for the rendered intersection points, see figure 1d.

7 Results

In figure 5 the timings for iso-surface and intersection lines extraction are shown for varying volume sizes with a fixed surface geometry. As can be seen, iso-surface extraction requires the most work. However, if iso-surface rendering is required, there is a small additional amount of work required to extract the intersection curves. However, the advantage of the Marching Faces algorithm is clear whenever surfaces are not required. Nonetheless, both the Enhanced Marching Cubes and the Marching Faces algorithms traverse all the voxels in the volumes and so these operations expand with the number of voxels. Profiling of the Marching Faces algorithm has shown that almost all execution time stems from traversing the volume and classifying each voxel. The computation has been reduced by using a simple caching scheme to save half of the bit-set of the voxel configuration when the algorithm advances to the following voxel. This scheme roughly halves the execution time of the Marching Faces algorithm.

In figure 6 the number of intersection tests versus the size of the cube side are shown. The graph clearly shows the linear relationship to the volumes' side for both intersection methods and that the more computational expensive part of the algorithm is bounded by linear complexity. Traversal of volumetric data is however a memory intensive task. By using blocking or bricking of the volume the data locality can be improved and thus yield higher cache hit-rates. See [Parker et al. 1998] for a discussion and implementation of an optimized bricking technique.

It can be concluded that both methods have an $\mathcal{O}(\sqrt{N})$ complexity for intersection testing ($N = |T|$). In terms of iso-surfaces, whose size depends on the side of the volume, the surface intersection methods are both linear with respect to the side of the volume. For the algorithm described in SURFACEISECT this reasoning holds if the iso-surface extraction is excluded. However, for the Marching Faces algorithm, the volume must be traversed. By expressing the algorithm complexity using s, the size of the side of the volume, we get $\mathcal{O}(s^3 + \sqrt{N}) = \mathcal{O}(s^3 + s) = \mathcal{O}(s^3)$.

8 Applications

Chaotic behavior is a phenomenon of importance in many fields of physics and has applications on multiple scales, including quantum mechanics, fibre optics, acoustics, and microwaves. The chosen application for this work is quantum chaos. Random superposition of monochromatic plane waves using the Berry wave function [Berry 1977] has been suggested as an approximation [Stöckmann 1999]. This wave function is then defined as a three-dimensional complex function $\psi : \mathbb{R}^3 \to \mathbb{C}$.

$$\psi(\mathbf{r}) = \sum_{j=1}^{N} a_j e^{i(\mathbf{k}_j \cdot \mathbf{r} + \phi_j)} \quad (5)$$

where a_j and ϕ_j are random numbers and $\mathbf{k}_j$ are random directions,

N defines the number of mixing states. In a bounded space like a cube, the function ψ must vanish on the boundary and equation 5 is rewritten to the form of equation 6

$$\psi(x,y,z) = \sum_j c_j \sin(k_{xj}x)\sin(k_{yj}y)\sin(k_{zj}z) \qquad (6)$$

where c_j are complex mixing coefficients, i.e. transformed versions of a_j and ϕ_j in equation 5. The sampling cube is slightly smaller than the domain of ψ in order to reveal the interior structure. See images in figure 1 for an example of a chaotic parameter region.

The properties of this function, e.g. where this function is zero $(0 + i0)$ defines the nodal lines of the wave-function. These nodal lines are of interest in applications such as electron transport in quantum mechanics. Nonetheless, the intersection of iso-surfaces has many other applications in the general fields of mathematics and physics.

The goal of this work is to provide fast extraction of these nodal lines so that interactive exploration of the properties of the nodal lines and corresponding electron transport can be conducted. For interactive exploration the authors take to mean that the entire process of computation of the complex field, extraction of nodal lines and visualization should be fast enough to provide acceptable update rates and user interaction.

9 Conclusions and future work

Two methods have been developed for efficient surface intersection of iso-surfaces, as embedded in volumetric data-fields. The algorithms have been tested on a complex dataset representing 3D chaos. By adding efficient data structures constructed on the fly, the performance has been improved for the intersection algorithm for triangular surfaces as generated by the Enhanced Marching Cubes algorithm. By using early classification of voxels in the Marching Faces algorithm the work is reduced for implicit intersection testing (roughly proportional to the side of the volume) to be almost negligible in comparison to volume traversal.

The indexing structures used for performance enhancement are currently implemented as volumes, which consumes internal memory. The authors consider these structures could be implemented primarily as two-dimensional structures, or by using hash-tables in order to reduce the memory requirement. It is also of interest to investigate the possibilities of caching obtained information for reuse in order to speed-up the intersection of consecutive data sets based on space-time coherence.

10 Acknowledgement

The authors wish to thank Karl-Fredrik Berggren at the Department of Physics and Measurement Technology, Biology and Chemistry at Linköping University for an interesting and encouraging problem to solve, which also happens to yield beautiful geometries. Financial support from the National Graduate School in Scientific Computing, the Swedish Research Council, and the Swedish Foundation for Strategic Research is acknowledged.

A Triangle Intersection Complexity

In this section a computational complexity of $\mathcal{O}(\sqrt{N})$ is deduced for two arbitrary surfaces. Assuming two arbitrary[1] surfaces with N_1 and N_2 triangles. The probability p_i for a voxel to contain any triangle in the triangle surface T_i is

$$p_i = \frac{N_i}{s^3} \qquad (7)$$

where s is the side of the volume. The probability p for two independent surfaces to intersect in a voxel is then $p = p_1 p_2$. For surfaces generated on a per voxel basis, the number of triangles in a surface is typically proportional to s^2, i.e. $N_i = c_i s^2$ for an unknown constant c_i.

$$p = p_1 p_2 = \frac{N_1 N_2}{s^6} = \frac{c_1 c_2 s^4}{s^6} = \frac{c_1 c_2}{s^2}$$

The number of triangle intersection tests t can then be expressed as

$$t = s^3 p = \frac{s^3 c_1 c_2}{s^2} = c_1 c_2 s$$

With $N_i = c_i s^2 \Rightarrow s = \sqrt{N_i/c_i}$ we rewrite t as

$$t = c_1 c_2 s = c_1 c_2 \sqrt{\frac{N_i}{c_i}} = c\sqrt{N_i}, \qquad c = c_1 c_2/\sqrt{c_i} \qquad (8)$$

The number of triangle intersection tests for two surfaces is thus proportional to $\sqrt{N}$, $(N \simeq N_1 \simeq N_2)$, given arbitrary surfaces. Two more or less identical surfaces leads to N intersection tests while two surfaces never intersecting yield 0.

[1] Arbitrary means a random distribution of the surface over the voxels in the volume and that the surfaces are not correlated.

References

BERRY, M. V. 1977. Regular and irregular semiclassical wave functions. *Journal of Physics A 10*, 2083–2091.

KRISHNAN, S., AND MANOCHA, D. 1997. An efficient surface intersection algorithm based on lower-dimensional formulation. *ACM Transactions on Graphics 16*, 1 (January), 74–106.

LIVNAT, Y., SHEN, H.-W., AND JOHNSON, C. R. 1996. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics 2*, 73–84.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of SIGGRAPH '87*, ACM Press, 163–169.

MONTANI, C., SCATENI, R., AND SCOPIGNO, R. 1994. A modified look-up table for implicit disambiguation of marching cubes. *Visual Computer 10*, 6, 353–355.

PARKER, S., SHIRLEY, P., LIVNAT, Y., HANSEN, C., AND SLOAN, P.-P. 1998. Interactive ray tracing for isosurface rendering. In *Proceedings of IEEE Visualization '98*.

PATRIKALAKIS, N. M. 1993. Surface-to-surface intersections. *IEEE Computer Graphics and Applications 13*, 1 (January), 89–95.

SABHARWAL, C. L. 1994. A fast implementation of surface/surface intersection algorithm. In *Proceedings of the 1994 ACM symposium on Applied computing*, ACM Press, 333–337.

STÖCKMANN, H.-J. 1999. *Quantum Chaos: An Introduction*. Cambridge University Press, Cambridge, UK.

THIRION, J.-P., AND GOURDON, A. 1996. The 3D Marching Lines algorithm. *Graphical Models and Image Processing 58*, 6 (November), 503–509.

WILHELMS, J., AND GELDER, A. V. 1992. Octrees for faster isosurface generation. *ACM Transactions on Graphics 11*, 201–227.

ZOCKLER, M., STALLING, D., AND HEGE, H.-C. 1996. Interactive visualization of 3d-vector fields using illuminated stream lines. In *Proceedings of IEEE Visualization '96*, IEEE Computer Society, 107–113,474.