Approximate Implicitization and CAD-type intersection algorithms

Tor Dokken and Vibeke Skytt
SINTEF ICT, Department of Applied Mathematics

This work is partly funded by the European Union through the projects:

- GAIA II, IST-2001-35512, 2001-2005
- AIM@SHAPE, FP6 IST NoE 506766, 2004-2007

The starting point is: Geometry representation in CAD-systems

Standardized in ISO 10303 STEP in the early 1990es.

- Degree 1 and 2 algebraic curves and surfaces + torus
- NonUniform Rational B-Spline (NURBS) curves and surfaces
  - Piecewise rational/polynomial curves and surfaces
  - Frequently cubic/bi-cubic, but also higher degrees allowed (and in use)
- Volumes represented by description of the outer shell (and inner shell(s))
- A shell is represented by a patchwork of surface pieces
  - A shell is **not** required to be watertight, small tolerances controlled gaps allowed
  - A surface patch is limited by edges, the edge is limited by two vertices
- Double precision floating point arithmetic used

The representation is not precise!

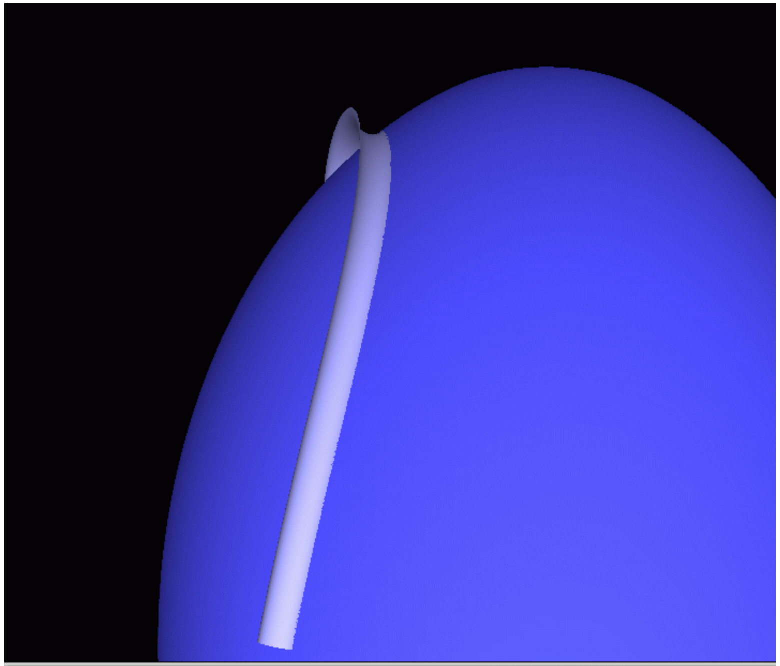
Why is CAD-geometry represented this way?

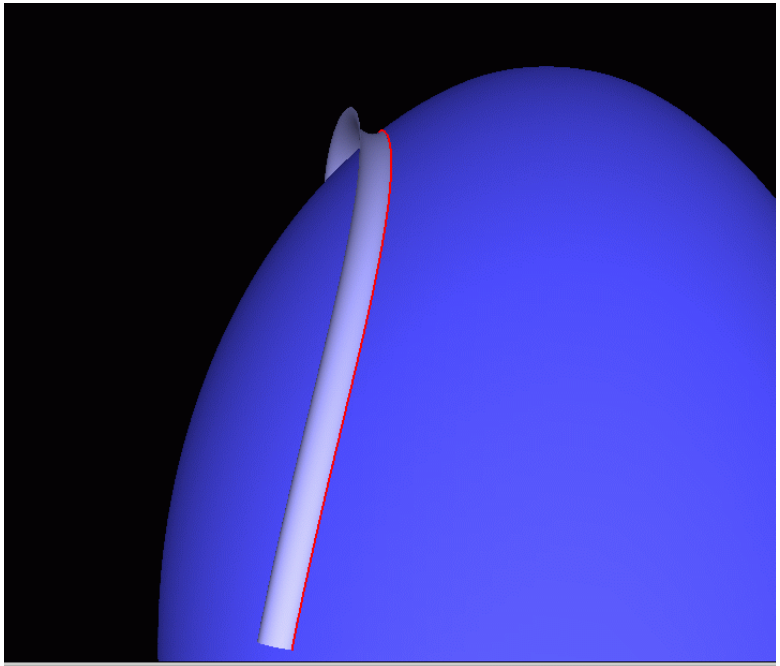
- ISO 10303 standardize the ideas of the late 1980s.
  - Monolithic 3D applications dominated
  - 3D CAD was still immature
  - Computers are now at least 3 orders of magnitude faster
  - Memory sizes are now 2 to 3 orders of magnitude larger

- Consequently 3D CAD is far from optimal but
  - It has penetrated all branches of industry
  - The industry has invested heavily in CAD
  - The CAD-industry has merged into a few dominant vendors
  - Current CAD is good enough for the average user but not for high-end industries such as aerospace, automotive and oil & gas industries

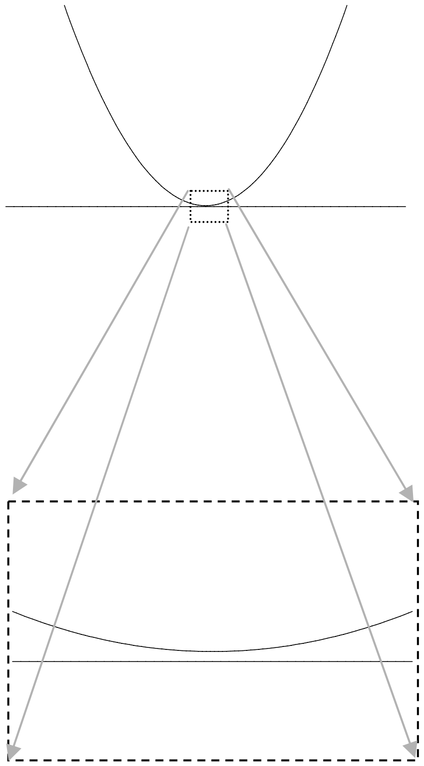
CAD design often produce near singular transitions between surfaces

- Patchwork of surface to build a larger smooth surfaces
- Transition between blending surface and mother surfaces
  - Design intent and what the user believes happens: Tangent continuity
  - Result: Small gaps and near tangent continuous





What does near singular mean?



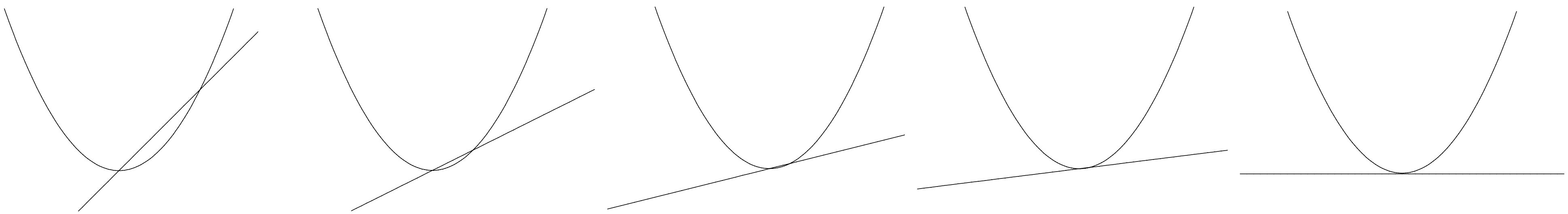
- Seen from far away –
  - Intersection interval
- Zooming by a factor of 10x
  - No intersection
- Many computer displays have less than 1200 x 1600 pixels
  - When visualizing an object of size 1000 mm, the smallest visible details will be approximately 1mm
  - Production tolerance in most cases significantly smaller….
  - The displayed image also distorted by tessellation

Partial coincidence

Traditional CAD-type intersection algorithms focus on non-singular intersections

- An intersection curve between two surfaces is transversal when the normals of the intersecting surfaces are non-parallel along the intersection curve
- Sinha’s theorem (1985):
  - If two smooth surfaces S_1 and S_2 intersect in a common loop then there is a point P_1 inside the loop in S_1, and there is a point P_2 inside the loop in S_2 such that the normal N_1 in P_1 is parallel to the normal N_2 in P_2.
- If the normal fields of two surfaces do not overlap, no closed intersection loop is possible, and the intersection is transversal.
  - Repeated subdivision of surfaces with overlapping normal fields will, provided the intersections curve is transversal, eventually results in subproblems where Sinha’s theorem can be applied. (Loop destruction)
  - However, when the intersection is near singular it will take a very long time….

Singular and near singular intersections



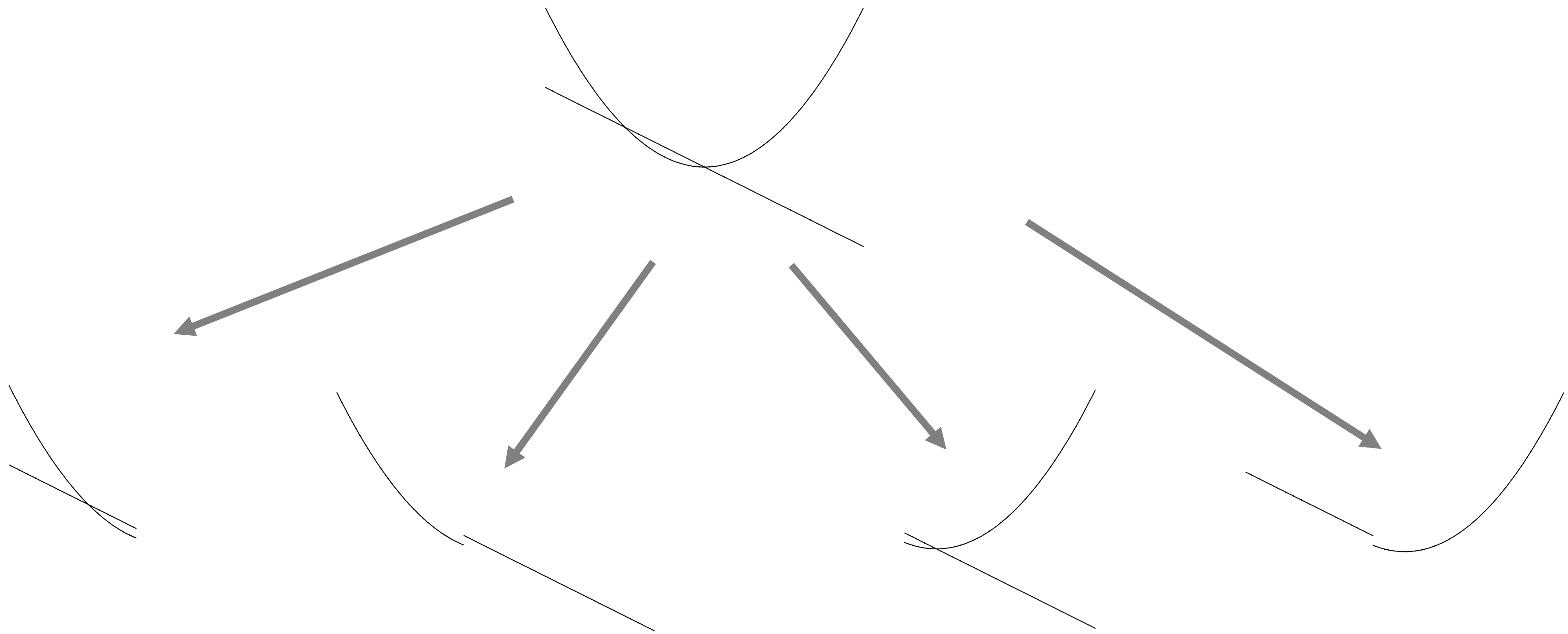
2 points!

2 points?
1 singular point?
An interval?

2 points?
1 singular point?
An interval?
No point?

- The relative position and orientation of curves and surfaces determines if an intersection is:
  - Transversal
  - Near singular (tolerance dependent!)
  - Singular

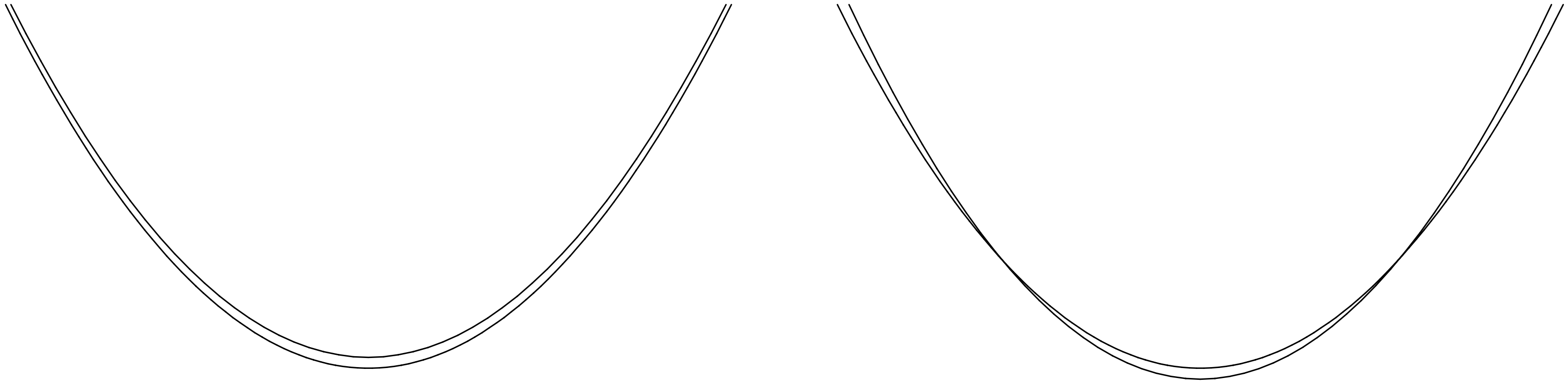
Recursive subdivision to try to make simpler subproblems



- Each intersection singled out in a simple subproblem

Recursive subdivision do not efficiently sort out all singular or near singular situations

- Difficult to decide if sculptured near parallel curves and surfaces intersect or not



- Deep levels of recursion necessary
- Where to subdivide has to be considered with care

Most CAD-intersection algorithms have no quality guarantee

- Simplistic algorithms are fast & often produce the correct result
  - Intersect triangulations of the surfaces
  - Lattice evaluation – intersect mesh of curves in each surface with the other surface to possibly generate points on all intersection branches
  - Marching/refinement of identified intersection tracks
- Recursive algorithms slower, sometimes extremely slow
  - More calculations, guarantee for clearly transversal intersections
  - Deep levels of recursion in near singular cases
  - For singular intersections traditional recursive intersection algorithms will not work (well)
    - Cut off strategies necessary to avoid infinite recursion
    - Improved approaches needed

Improvement of intersection algorithms by combining parametric and algebraic representations

- Improved approaches for separating surfaces
- Simplification of intersection problems to the parameter domain of one of the surfaces
- Determine that two surfaces only touch along a boundary curve

Most often an algebraic surface approximating the part of the surface addressed suffice.

Approximate implicitization (Dokken 97)

- In stead of the global correct (high degree) algebraic representation we want to find an algebraic approximation to the curve or surface that is closer than a given tolerance in a defined region of interest.
  - Well behaved numeric method “Approximate Implicitization” have been developed
    - Proven numeric well behaved rounding error
    - High convergence rates
    - Use modified LU-decomposition or Singular Value Decomposition
  - Algebraic degree can be considerably lower than the theoretical exact degree. (For bicubic total degree 4 or 6, opposed to the exact degree 18)
  - Sufficiently efficient to be an efficient tool for determining intersection, near intersection or separation of surfaces intersected
- The method is an exact implicitization method if proper algebraic degree chosen (and exact arithmetic used)

The approximate implicitization factorization

- Assume that the surface $\mathbf{p}(s,t)$ has bi-degree (n_1,n_2)
- Assume that q has total degree m and that $\mathbf{b}$ is a vector containing the unknown coefficients of q
- The combination $q(\mathbf{p}(s,t))$ is a polynomial of bi-degree (mn_1, mn_2)
- Collect basis functions of bi-degree (mn_1,mn_2) in $\alpha(s,t)$
- Then $q(\mathbf{p}(s,t))$ can be factorized

$$q(\mathbf{p}(s,t)) = (\mathbf{D}\mathbf{b})^T \boldsymbol{\alpha}(s,t).$$

The factorization

$$q(\mathbf{p}(s,t)) = \left(\mathbf{D}\mathbf{b}\right)^T \boldsymbol{\alpha}(s,t).$$

- An element in **D** is the product of a maximum of m coefficients of $\mathbf{p}(s,t)$ and a constant, where m is the total degree of q.
- If $\mathbf{p}(s,t)$ is a Bezier surface of bi-degree (n_1, n_2) then $\alpha(s,t)$ is a Bernstein basis of bi-degree (mn_1, mn_2).

Properties of the factorization

$$q(\mathbf{p}(s,t)) = (\mathbf{Db})^T \boldsymbol{\alpha}(s,t).$$

- If **Db=0** and **b≠0** then **b** contains the coefficients of an exact algebraic representation of total degree *m* of **p**(*s,t*).
- If $\alpha(s,t)$ is a Bernstein basis then $\|\boldsymbol{\alpha}(s,t)\|_2 \le 1,$ and
$$|q(\mathbf{p}(s,t))| = \left|(\mathbf{Db})^{\mathrm{T}} \boldsymbol{\alpha}(s,t)\right| \le \|\mathbf{Db}\|_2.$$
- Let $\sigma_{\min}$ be the smallest singular value of **D**, then
$$\min_{\|\mathbf{b}\|_2 = 1} \max_{(s,t)\in\Omega} |q(\mathbf{p}(s,t))| \le \sigma_{\min}.$$
- Singular value decomposition of **D** can be used to find approximate solutions

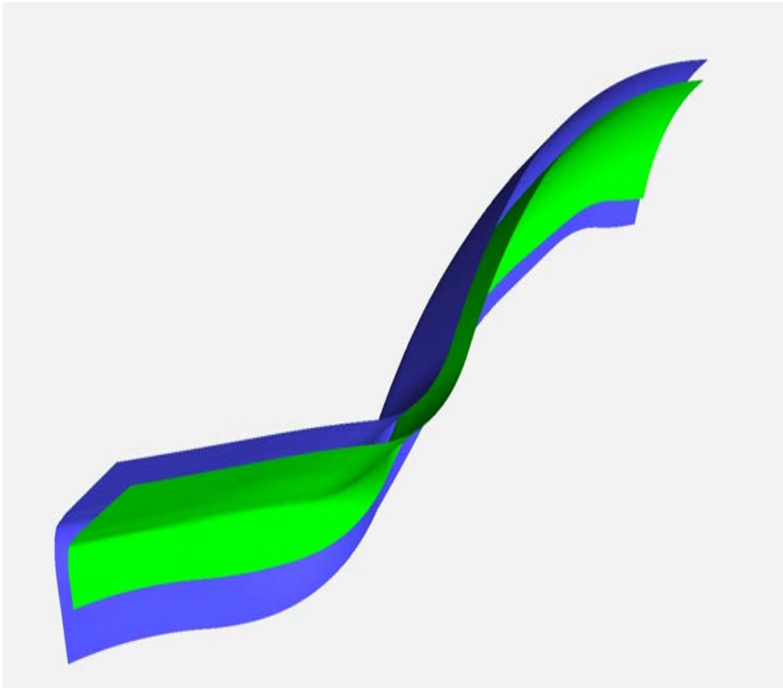
The algebraic/parametric combination used for separation of surfaces

Let $\mathbf{p}(s,t)$, $(s,t)\in\Omega_1$ and $\mathbf{r}(u,v)$, $(u,v)\in\Omega_2$, be two rational surfaces

- Decide that two surfaces do not intersect by finding an algebraic surface $q(x,y,z)=0$ separating the surfaces

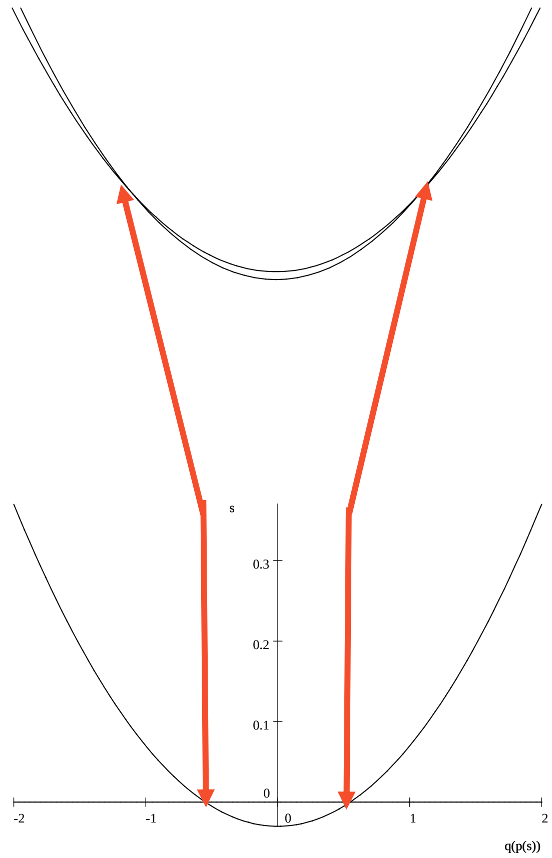
$$q(\mathbf{p}(s,t))>c \text{ and } q(\mathbf{r}(u,v))<c.$$

- Find the approximate algebraic surface by approximate implicitization



The algebraic/parametric combination for determining the topology of an intersection

- The intersection of two parametric curves $\mathbf{p}_1(s)$ and $\mathbf{p}_2(t)$, can be simplified if implicit representations of at least one of curves exist: $q_1(x,y)=0$ and $q_2(x,y)=0$.
- The combination $q_1(\mathbf{p}_2(t))=0$ transforms the intersection of two parametric curves to finding the zeroes of an univariate polynomial
- Easily extended to surfaces – use approximate implicitization

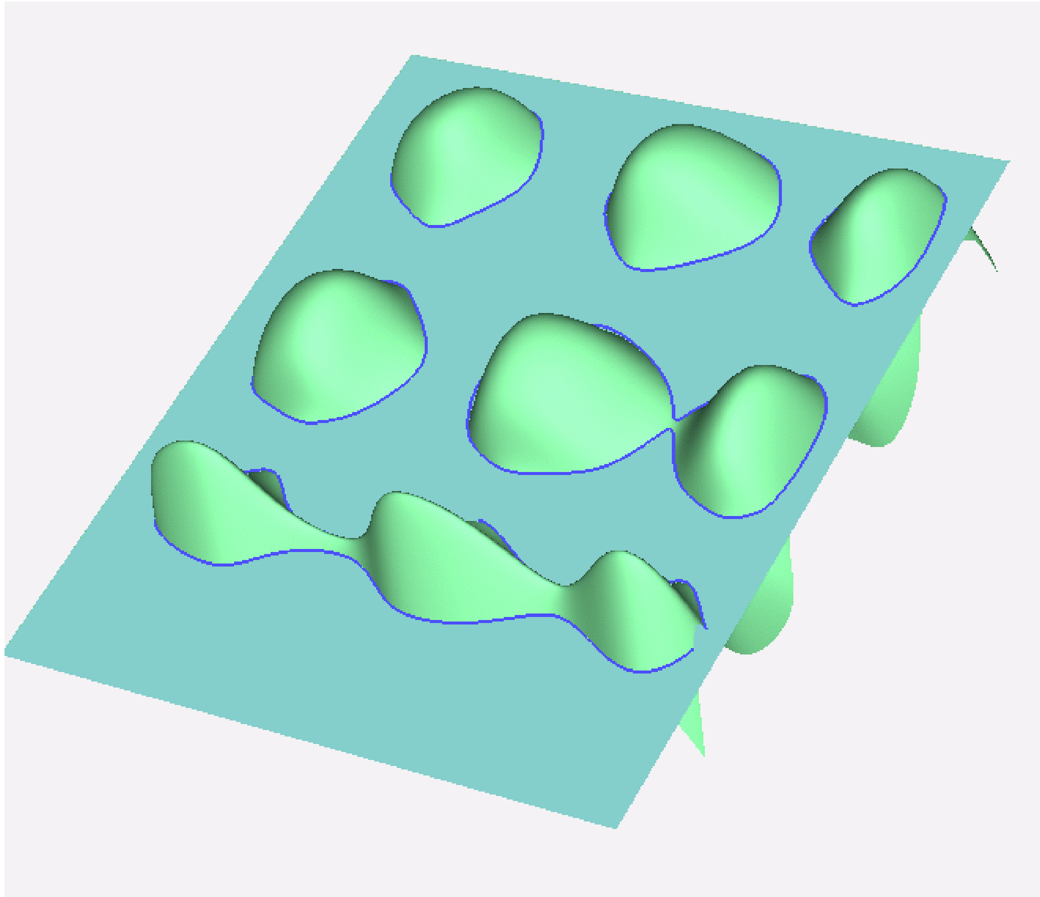


Special use of approximate implicitization in self-intersection

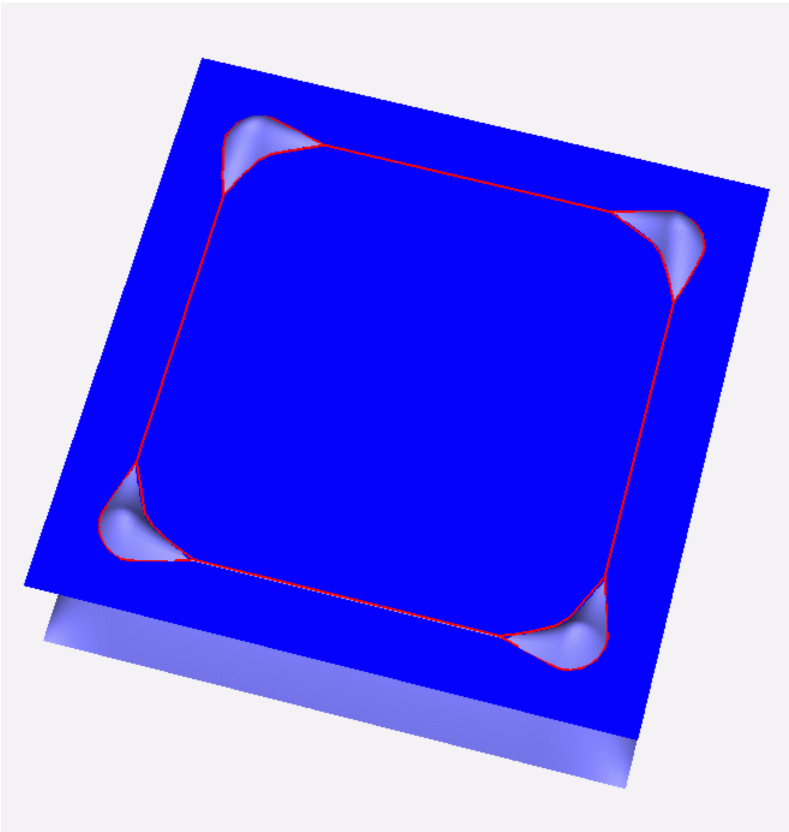
- As part of a recursive surface self-intersection algorithm, adjacent surface subpatches have to be intersected
  - There will always be an intersection along the common edge
  - An approximate implicit surface following the normal of the surface (or a fixed direction) along the edge between the subpatches is made, and used for deciding if the edge intersection is the only intersection between the subpatches

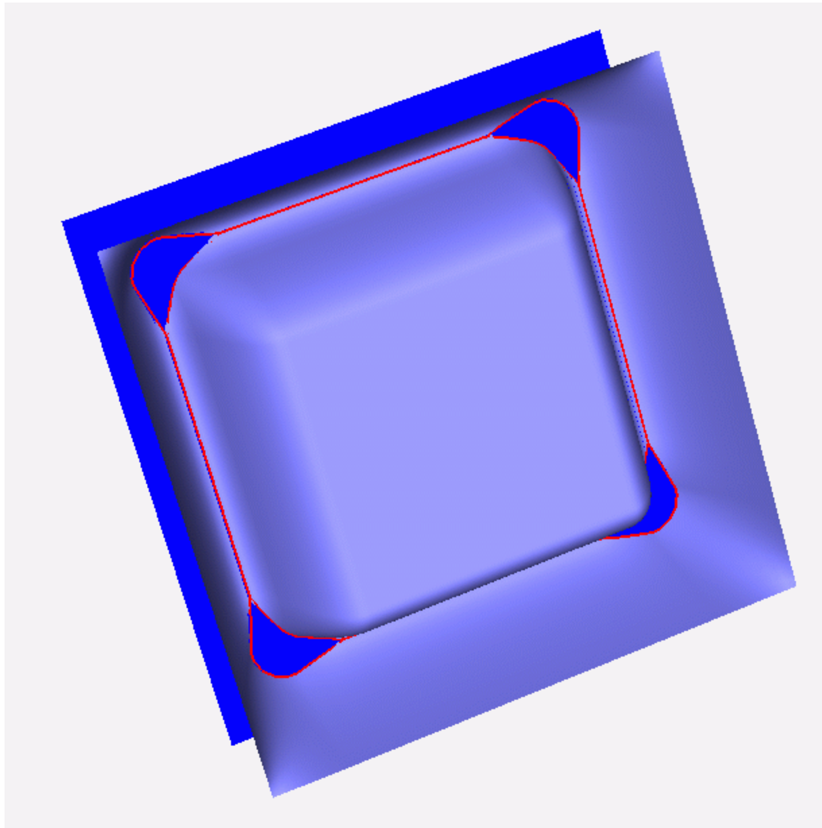
An example where traditional recursive intersection algorithms work well

- Intersection of a plane parametric surface and a varying parametric surface producing many intersection loops

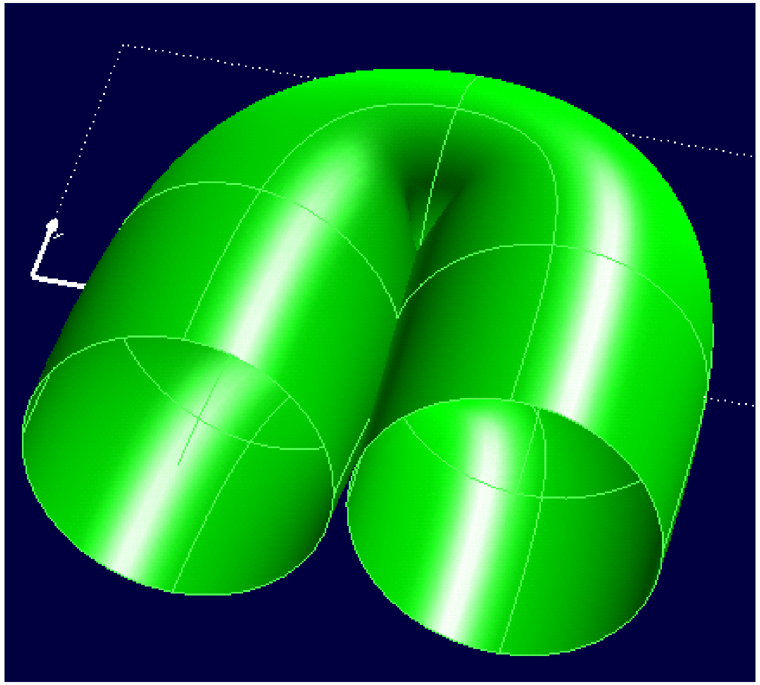


Singular intersection curves and loops





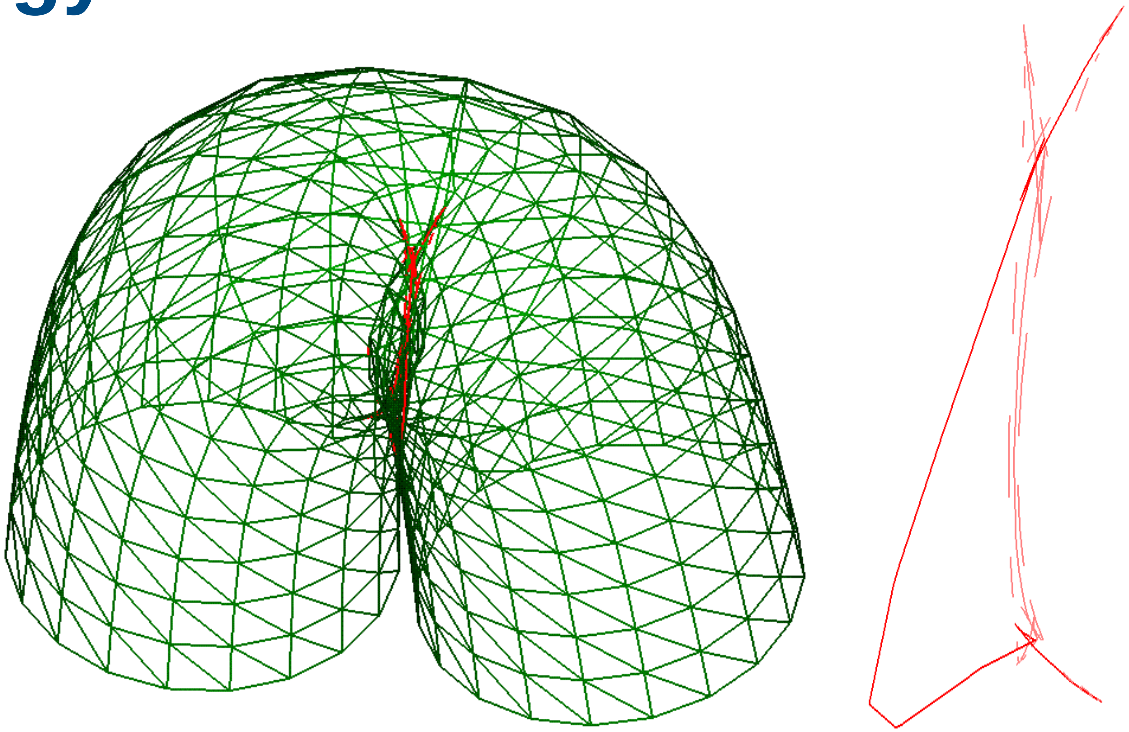
Surface self-intersection can give complex intersection topology



Self-intersecting bi-cubic B-spline surfaces.

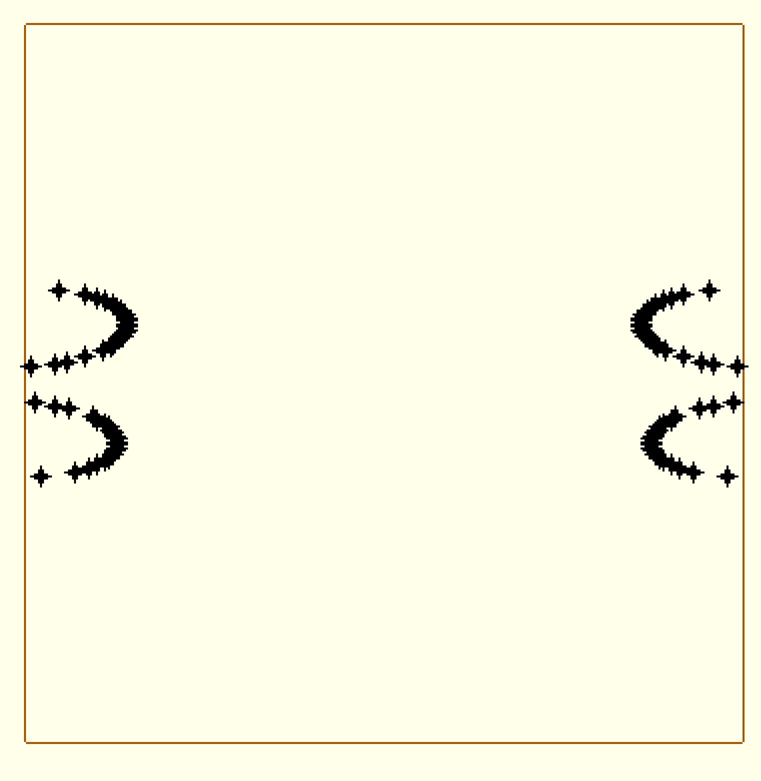
- 39 × 70 polynomial pieces
- Single knots

Courtesy think3

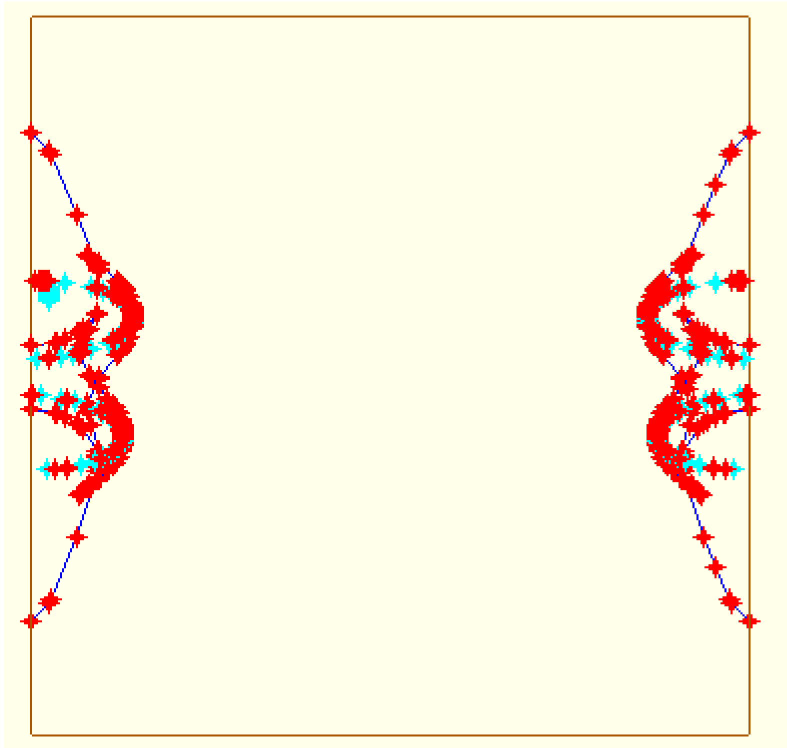


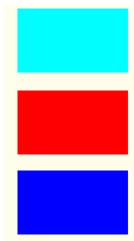
Wire frame of surface with self-intersection curves. The self-intersection curves displayed alone to the right

Parameter domain self-intersection trace



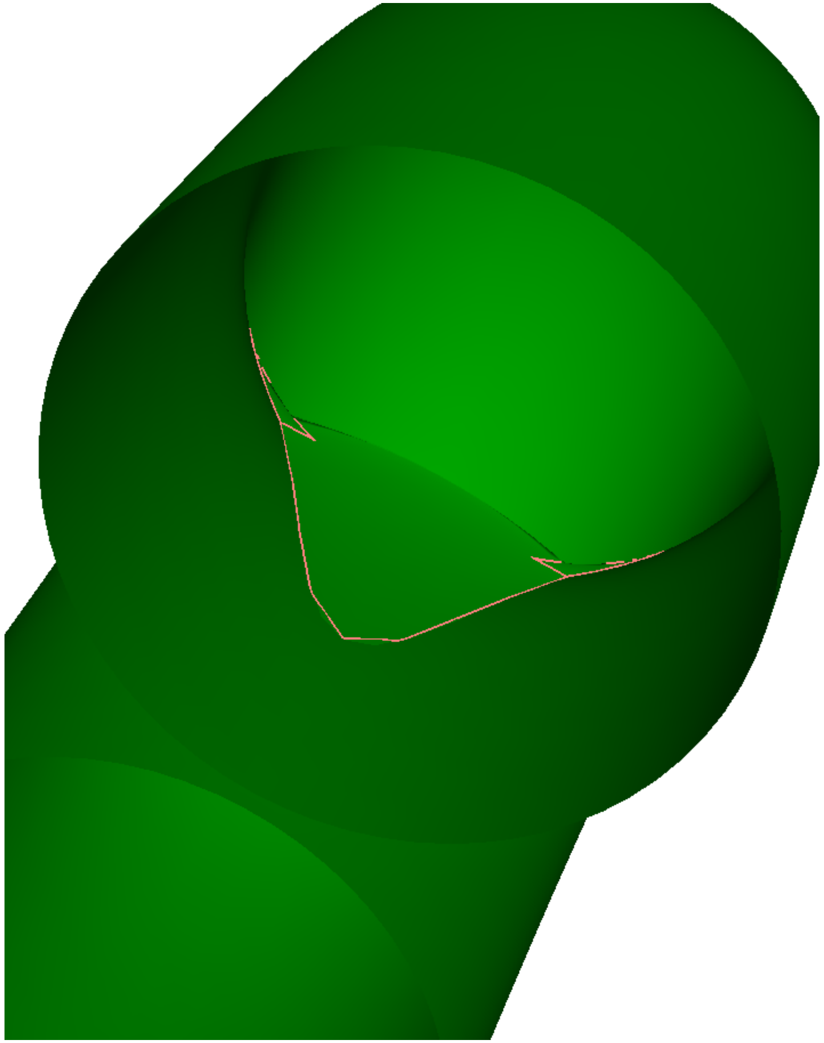
Singular points (vanishing normal)

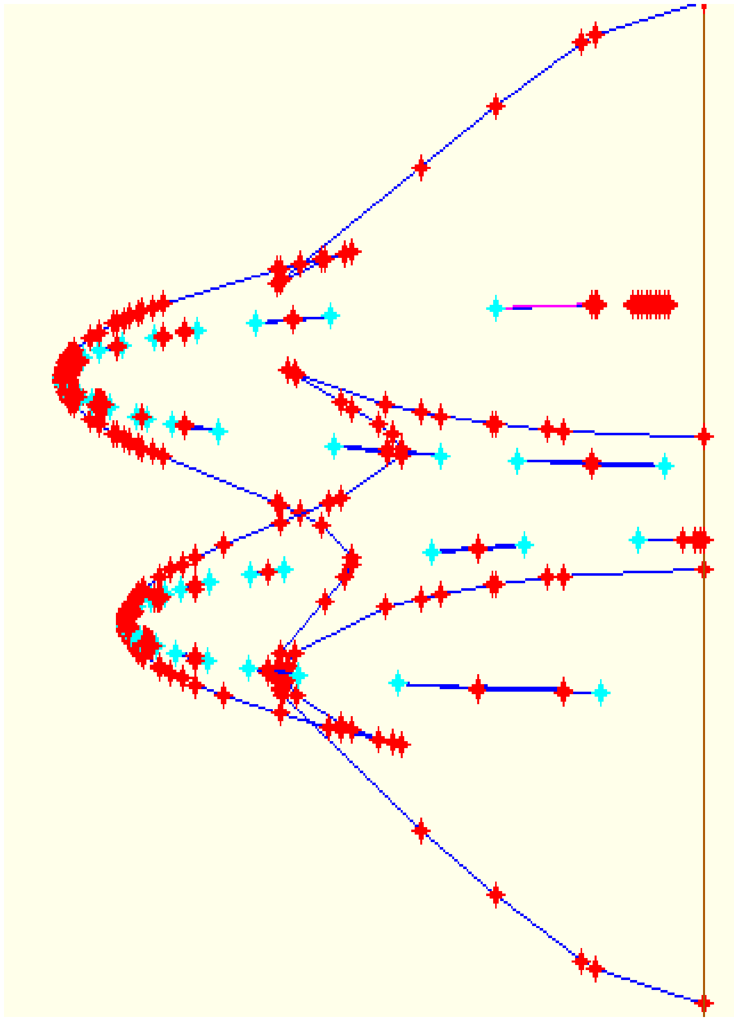




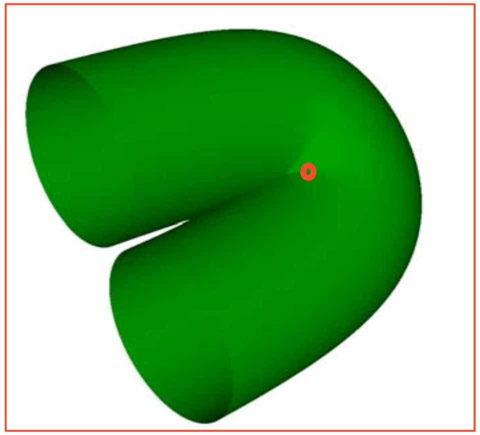
←Singular point
←Ordinary point
←Curve

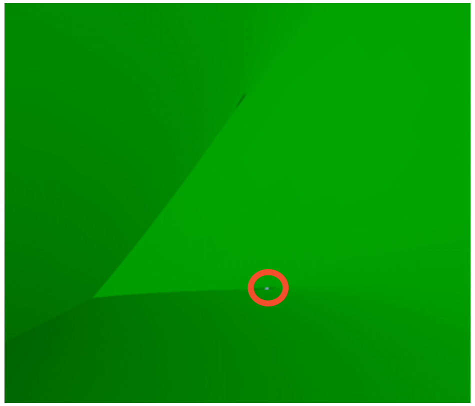
More details

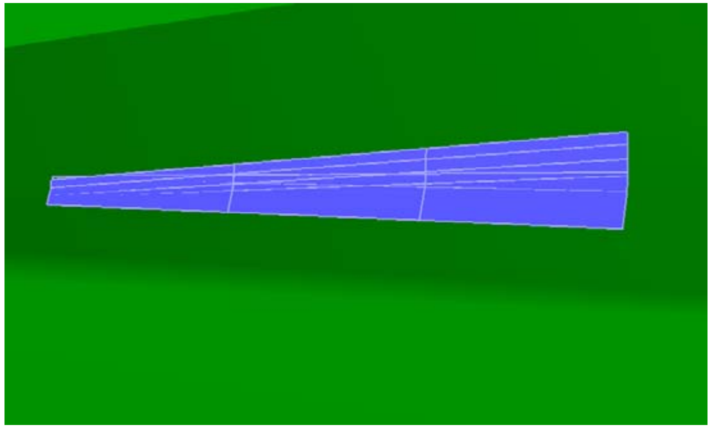


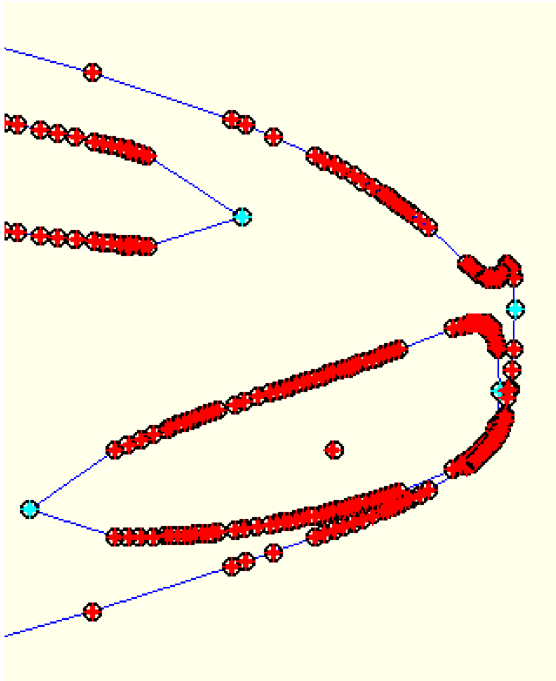


Intersection within intersection



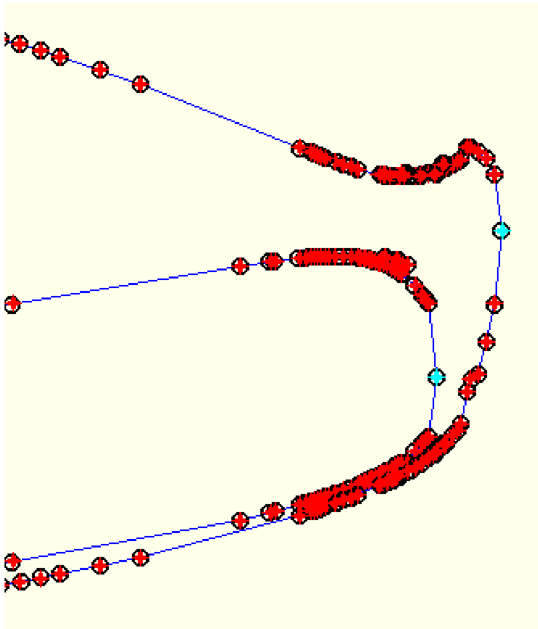






A small self-intersection loop very close to the global self-intersection

The distance between the loops is 1/10000 of the width of the parameter domain.



The GAIA Surface Self-intersection code

- Originally we aimed at two types of self-intersections
  - Global**. Two completely different pieces of the surface intersect. Provided that the surface is split into relevant sub surfaces, global self-intersections can be computed as surface-surface intersections..**
  - Local**. A local self-intersection will appear as a small loop or a cusp. The surface normal will become very small in the vicinity of a local self-intersection**
- During testing we realized that cusp ridges (curves where the surface normal vanish) are more frequent than expected in self-intersection
  - **The ridges do not in general follow constant parameter lines**
  - **Offset surfaces, duct type surfaces**
- **The self-intersection code uses the GAIA surface surface intersection code and has posted new challenges to this code**

Future work

- Improve the intersection code by:
  - Further testing and debugging
  - Improve speed of approximate implicitization
  - Implement new strategies for the combination of recursive subdivision and approximate implicitization
  - GPU-acceleation
- To better understand what is going on we will in our Paralle3D project (www.sintef.no/parallel3d)
  - Improve visualization tools to be able to zoom into more detail. The current viewers do not allow fine enough tessellation
  - Combine viewers for the parameter domain and 3D
  - Combine viewers for algebraic and parametric surfaces