Mini-symposium: Heterogeneous Computing

1. Tor Dokken: *Geometry Processing and Heterogeneous Computing*
2. Sylvain Lefebvre: *Parallel Example-based Texture Synthesis for Surfaces*
3. André Rigland Brodtkorb: *A Comparison of Three Commodity Level Parallel Architectures: Multi-core CPU, the Cell BE and the GPU*
4. Jon Hjelmervik: *Simplification of FEM-models on multi-core processors and the Cell BE*

CAGD and Heterogeneous Computing

Tor Dokken

SINTEF & CMA

Oslo, Norway

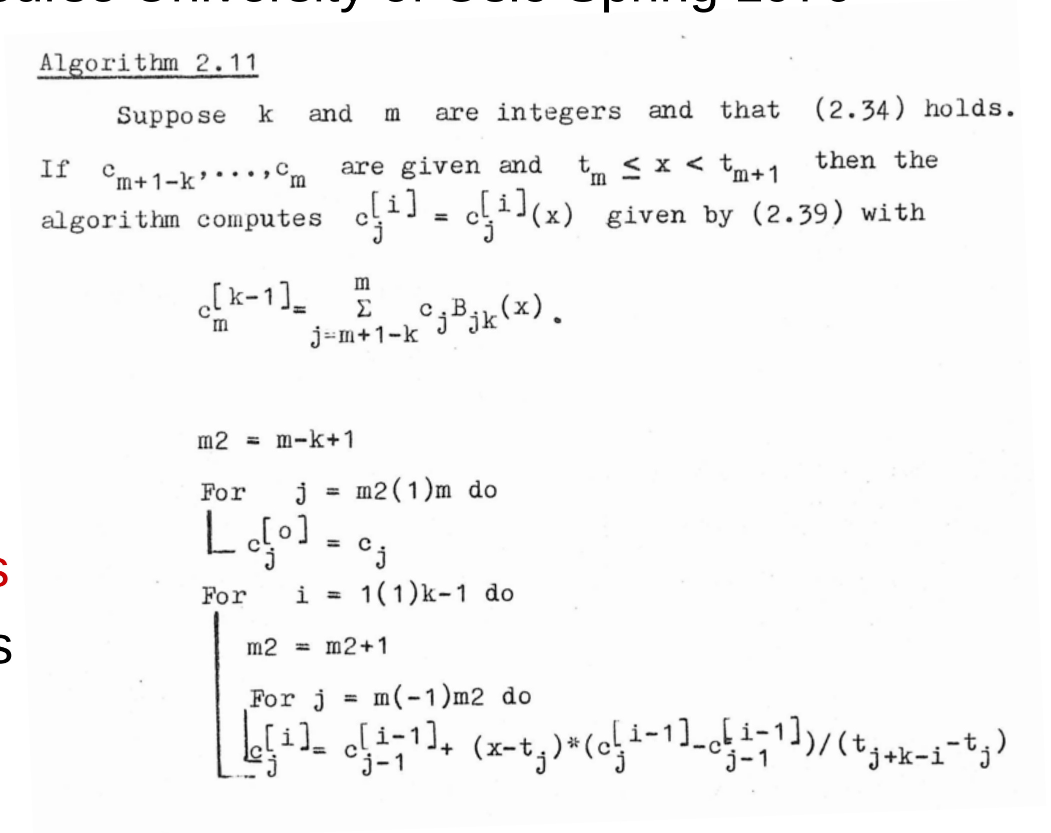
Algorithm design in CAGD influenced by hardware evolution

- Floating point operations were expensive until the start of the 1990s
  - Algorithms structured to reduce number of flops
- Localization of data important to avoid waiting
  - Has been growing in importance with the growth in depth of memory hierarchies. (Registers, L1, L2, L3 cache, primary memory, swapped memory)
  - Will be even more important when only a small part of the chip can be reached in one clock cycle
- Parallel processing has until recently only been of interest within High Performance Computing
  - Most education still aimed at sequential computing

de Boor’s algorithm

- Evaluation of values on B-splines anno 1976
  - Lecture notes spline course University of Oslo Spring 1976

- Reduce number of multiplications and divisions
  - k(k-1)   *,/
  - Horner uses k-1 *
- Conversion to local polynomials proposed for repeated calculations
- Floating point operations significantly more costly than other operations



Algorithm 2.11

Suppose k and m are integers and that (2.34) holds. If $c_{m+1-k},\dots,c_m$ are given and $t_m \leq x < t_{m+1}$ then the algorithm computes $c_j^{[i]} = c_j^{[i]}(x)$ given by (2.39) with

$$c_m^{[k-1]} = \sum_{j=m+1-k}^{m} c_j B_{jk}(x).$$

```
m2 = m-k+1
For j = m2(1)m do
    $c_j^{[0]} = c_j$
For i = 1(1)k-1 do
    m2 = m2+1
    For j = m(-1)m2 do
        $c_j^{[i]} = c_{j-1}^{[i-1]} + (x-t_j)*(c_j^{[i-1]} - c_{j-1}^{[i-1]})/(t_{j+k-i}-t_j)$
```

de Boor's algorithm

- Evaluation of values on B-splines anno 1976
  - Lecture notes spline course University of Oslo Spring 1976

- Reduce number of multiplications and divisions
  - k(k-1)   *,/
  - Horner uses k-1 *
- Conversion to local polynomials proposed for repeated calculations
- Floating point operations significantly more costly than other operations

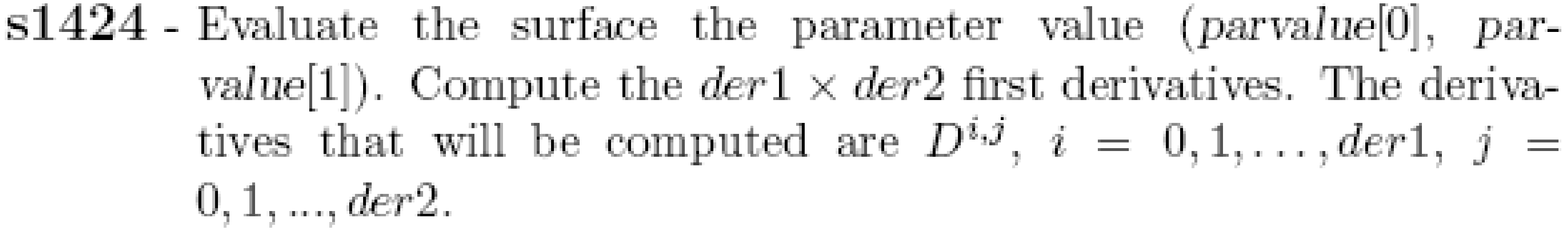
- Experiences from the early 1980s showed
  - Not efficient for parametric curves and surface
  - Conversion to local polynomials gives loss of accuracy
  - Seldom sufficiently repetition of calculation on one polynomial segment

Evaluation of values and derivatives of B-splines represented curves and surfaces

- Around 1985 we asked Tom Lyche and Knut Mørken to help make better methods for evaluating B-spline curves and surfaces

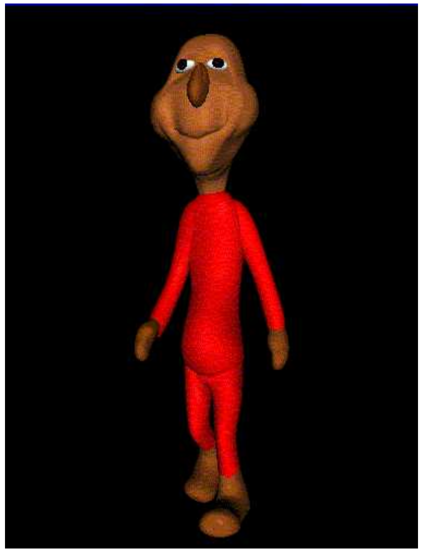
$$\mathbf{p}(s,t)=\sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\mathbf{c}_{i,j}B_{i,k_1}(s)B_{j,k_2}(t)$$

- First evaluate values and needed derivatives of basis functions, then multiply with vertices
  - Numerically stable, Improved performance for parametric curves and surfaces
  - Compatible results with Oslo algorithm provided care taken
  - Save operations compared to de Boor algorithm in most cases
- Algorithm later implemented in the SISL-library



s1424 - Evaluate the surface the parameter value (*parvalue*[0], *parvalue*[1]). Compute the $der1 \times der2$ first derivatives. The derivatives that will be computed are $D^{i,j}$, $i = 0, 1, \ldots, der1$, $j = 0, 1, ..., der2$.

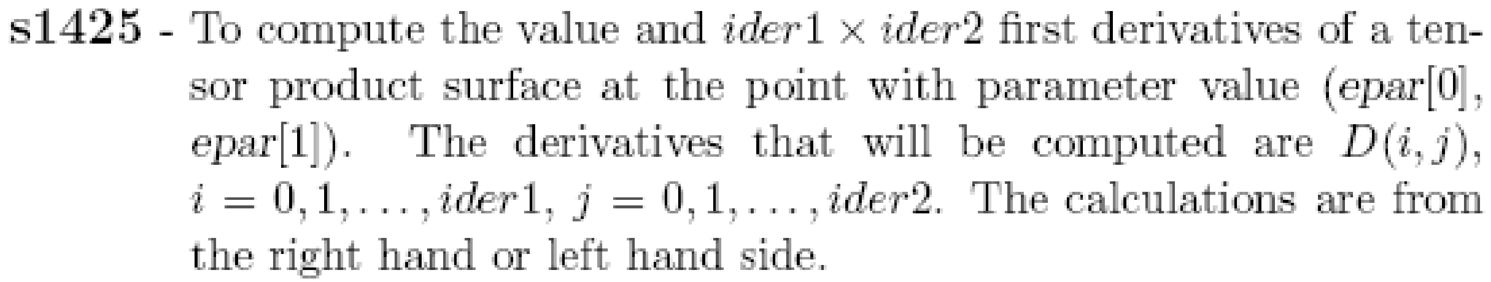
Animation of B-spline surfaces



Example from MobileMedia
Image: SINTEF

In 1997 we started on animation projects using B-splines

- A fairly dense regular grid of points and normals to be evaluated on the surface
- Independent evaluation of points too slow
- Mike Floater implemented an efficient algorithm for the SISL-library (nested loops)



s1425 - To compute the value and $ider1 \times ider2$ first derivatives of a tensor product surface at the point with parameter value ($epar[0]$, $epar[1]$). The derivatives that will be computed are $D(i, j)$, $i = 0, 1, \ldots, ider1$, $j = 0, 1, \ldots, ider2$. The calculations are from the right hand or left hand side.

Data parallel B-spline point and partial derivatives evaluation

- Evaluate $n_1 \times n_2$ points (and possibly partial derivatives) on a B-spline surface of orders (k_1,k_2)
  - Evaluate all non-zero basis functions of B-spline basis in first and second parameter direction and store in $k_1 \times n_1$ matrix $\mathbf{D}_1$ and $k_2 \times n_2$ matrix $\mathbf{D}_2$
  - The surface vertices have a natural matrix structure **C**
  - Remembering the compact storage in $\mathbf{D}_1$ and $\mathbf{D}_2$ the point evaluation can be expressed as a matrix product (remember relative indexing) of $\mathbf{D}_1$,**C** and $\mathbf{D}_2$
- Data parallel processors such as GPUs are very efficient for execution of such matrix products
  - Efficiency dependent on sufficiently large grids
- However, the matrix structures are also well suited for efficient execution on traditional CPUs
- Matrix formulation part of current courses in Spline Methods (e.g. INF-MAT5340 at the University of Oslo)

Matrix formulation of B-spline evaluation

- Talk by Arne Lakså the first day:
  - *A generalized B-Spline matrix form of splines*

$$\mathbf{c}(t) = T_1(t)T_2(t)\ldots T_n(t)\mathbf{c}$$

- Matrix formulations better suited form making parallel algorithms than algorithms formulated by sums.

What about other surfaces?

- Box spline surface
  - A regular grid should be straight forward
- Triangular Bezier surfaces
  - A regular triangular grid should be straight forward
- T-splines will be more challenging
  - “Extensive” T-structure will require independent evaluation of each sample point
  - Challenging for current GPUs
  - Probably better suited for Cell BE due to fast data transfer between Fat and Thin cores

Hardware evolution

- Current hardware
  - Multi-core CPUs (Fat cores)
  - Graphics cards (Data parallel, Thin cores)
    - Slow exchange of data between Fat (CPU) and Thin cores (GPU)
  - Cell BE (1 Fat, 8 Thin cores)
  - Double precision recently introduced on GPUs
- Expected evolution
  - Multi core chips combining some fat cores and many thin cores with access to same memory
    - AMD Fusion (Planned release 2009)
  - Only a small part of the chip can be reached in on clock cycle

Regular grid evaluation of tri-variate representation

- Iso-geometric analysis replacing Finite Elements by tri-variate NURBS
  - Straight forward extensions of NURBS bi-variate matrix representation to tri-variate representation
- Tetrahedral Bezier volumes
  - Should be straight forward
- Box-splines
  - Should be straight forward
- Trivariate T-splines
  - Even more complicated than the bi-variate case

Challenges for grid evaluation: Watertight network of T-spline objects

Basis function for T-splines are more complex than the tensor product basis functions for NURBS.

- Evaluation of T-spline basis functions more complex than NURBS basis functions

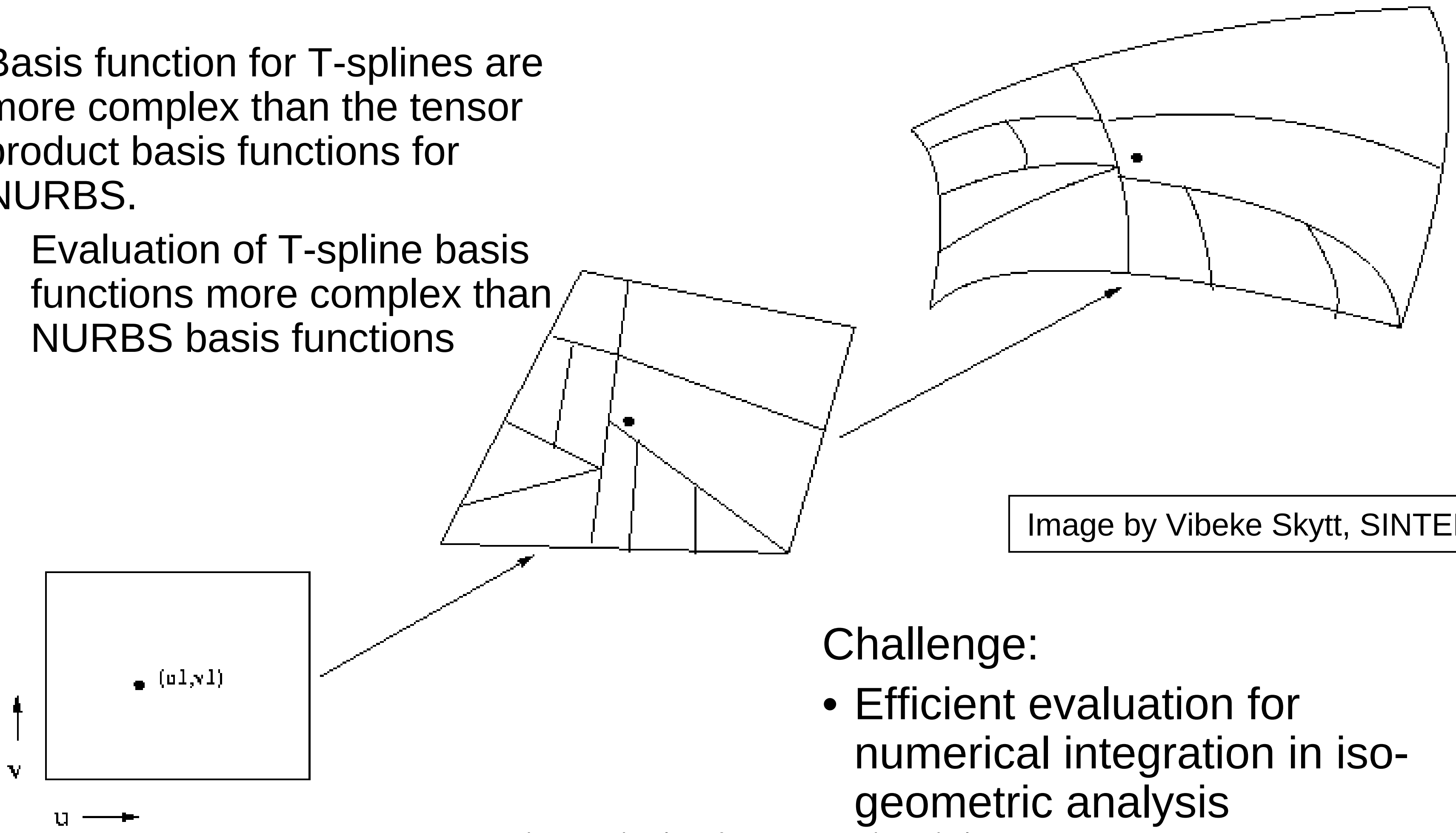


Image by Vibeke Skytt, SINTEF

Challenge:

- Efficient evaluation for numerical integration in iso-geometric analysis

Making water tight models

- The T-spline surface or volume can to be structured into regions:
  - T-spline structure: Regions to be refined to match adjacent objects exactly
  - NURBS structure: Regions that keep their tensor product nature

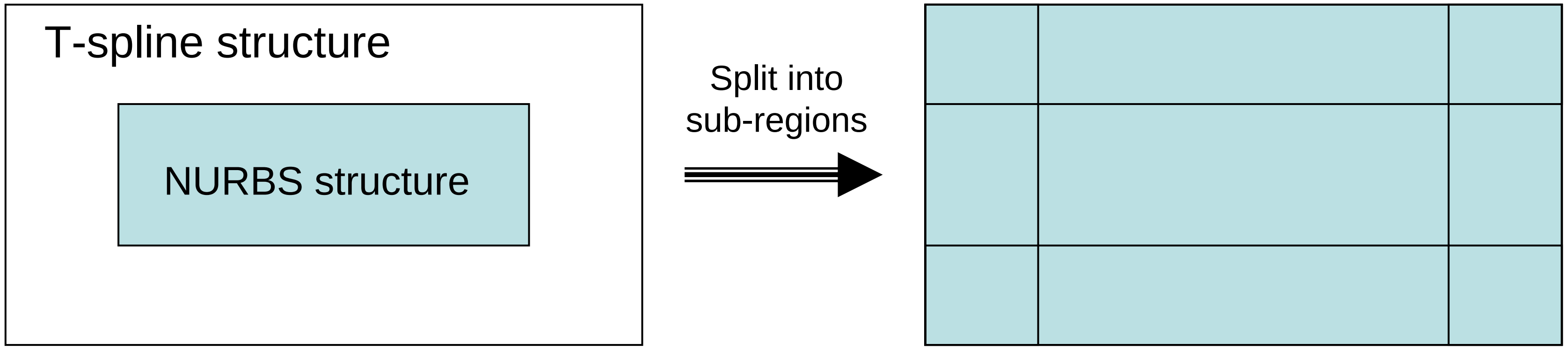


Image by Vibeke Skytt, SINTEF

Evaluation on future combined Fat and Thin multi-core chips with shared memory

- Alternative 1:
  - Evaluate T-spline structures on Fat-cores
  - Evaluate NURBS-structures on Thin cores
- Alternative 2:
  - Convert T-structures to NURBS-structures on Fat-cores
  - Evaluate on Thin-cores
- ….

Communication between GPU and CPU on the PCI-Express bus can be a bottleneck for this approach. Future chips combining Thick and Thin cores better adapted.

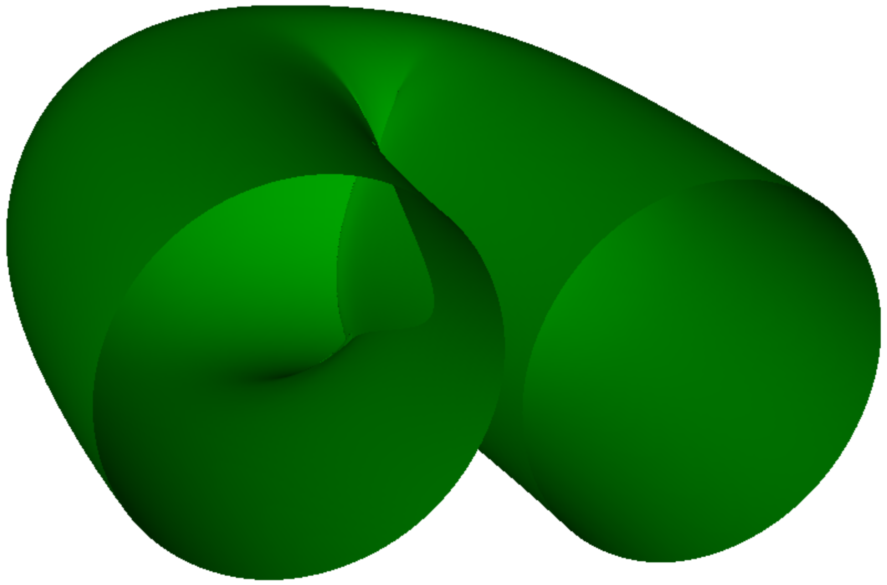
Subdivison algorithms

- Using the Oslo algorithm for knot insertion:
  - Calculate discrete B-splines and store in a matrix
- Use matrix products as in grid evaluation
  - Take care to avoid trivial multiplications

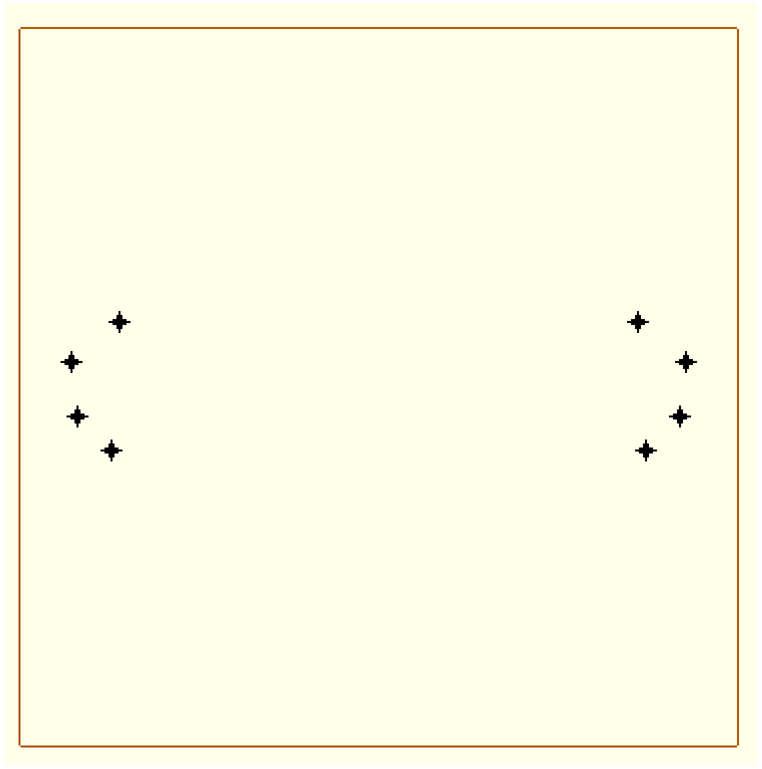
Intersection algorithms

- Intersections of NURBS with guarantee is always resource consuming.
- Use the parallel resources to understand the complexity of the problem
  - Decide if a simple approach is sufficient for finding all intersections
  - Find intersection regions where more advanced approaches necessary

Example surface self-intersection



Rational cubic B-spline surface
11 x 16 coefficients

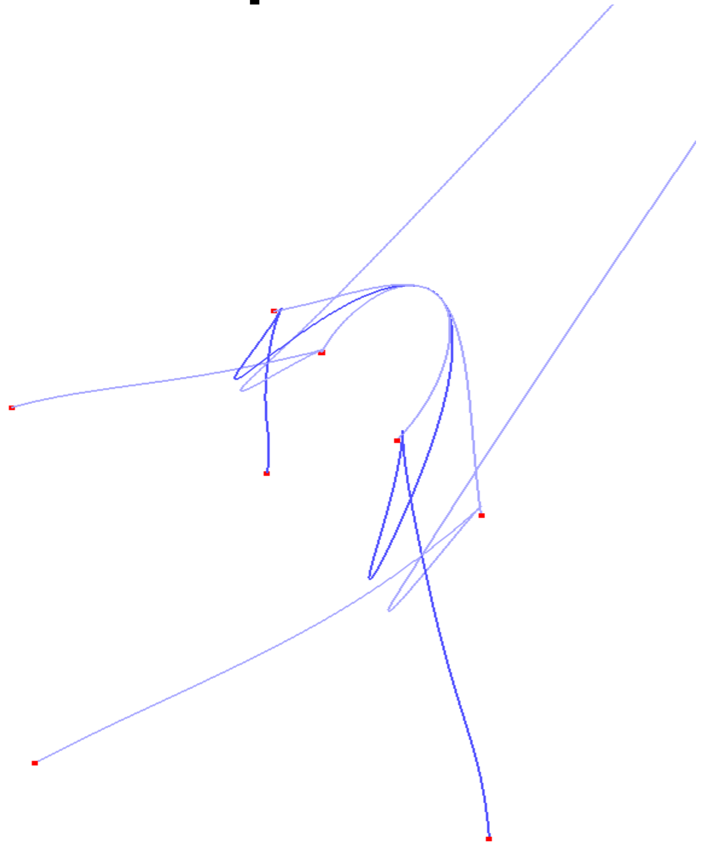


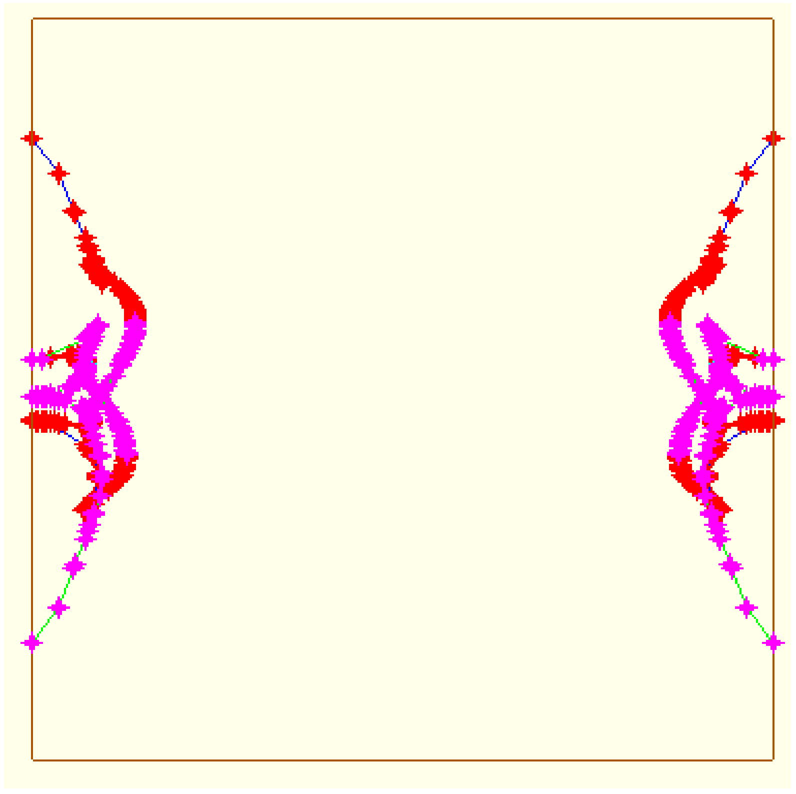
Eight singular points

Example courtesy of think3.
Images by Vibeke Skytt, SINTEF

Fairly simple parallel algorithms can detect if and where self-intersections may be located, and their complexity

Example surface self-intersection





Self-intersection curve in parameter domain

Details of 3D self-intersection curves

Fairly simple parallel algorithms can detect if and where self-intersections may be located, and their complexity

Example surface ray-casting

- Was addressed in detail in the talk by Martin Reimers on Thursday:

  Ray Casting Algebraic Surfaces using the Frustum Form

- Autumn 2006 GPU-generation from NVIDIA. Image size 512×512
  - Real time ray-casting of algebraic degree up to 10 to 12
  - For degree 20, two to four frames a second
- Expected significantly improved performance on most recent NVIDIA generation of GPUs

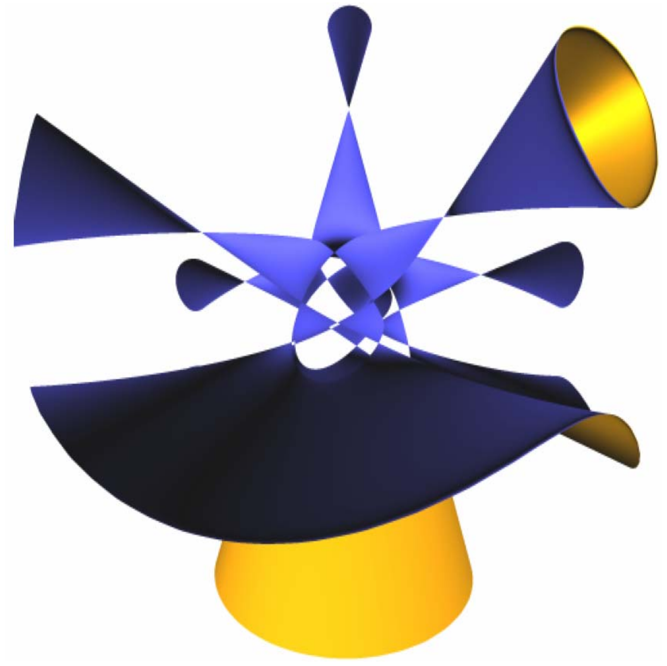


Image by Johan Seland, SINTEF

Conclusion

- The emerging heterogeneous computational resources (multi-core, GPU, Cell BE) challenge current algorithmic approaches within CAGD,
- and have the potential of drastically improving performance of algorithms within CAGD