Jon Hjelmervik, Erik Bjønnes, Christopher Dyken, Franz Fuchs

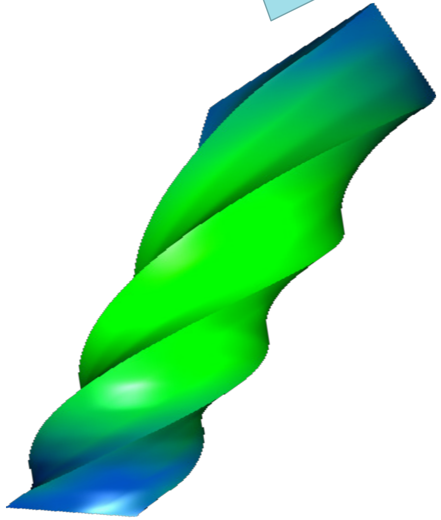
Fast Volumetric Visualization of Isogeometric Models

March 20 2013, GPU Technology Conference, San Jose

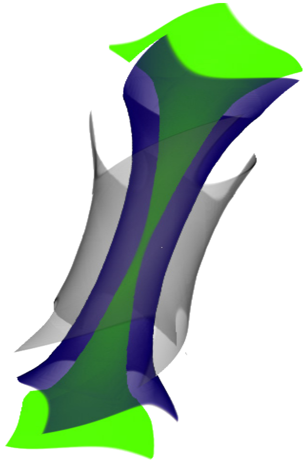
What will be covered?

CAD Models

Simulation Data



Volume Rendering



CAD Data Representation

The engineers Pierre Bézier and Paul de Casteljau from Renault and Citroen started using freeform surfaces in the 1950s. This was the starting point for Non-Uniform Rational B-Splines (NURBS) as the standard tool for curve and surface description in Computer Aided Design (CAD).

- Piecewise polynomials
- Numerically stable
- Surfaces are described using a control polygon
    - Approximates the surface
    - Surface lies within the "convex hull" of the polygon

$$S(u,v) = \sum_i \sum_j c_{i,j} B_i(u) B_j(v)$$

Analysis Data Representation

Finite Element Methods (FEM) has formed the basis for analysis software for more than half a century. Typical uses are temeprature or structural analysis of physical objects before they are produced.

The basis is to split the domain (object) into a finite set of elements, typically triangles or tetrahedra. The solution (temperature) is expressed as a linear combination of basis functions defined over the elements.

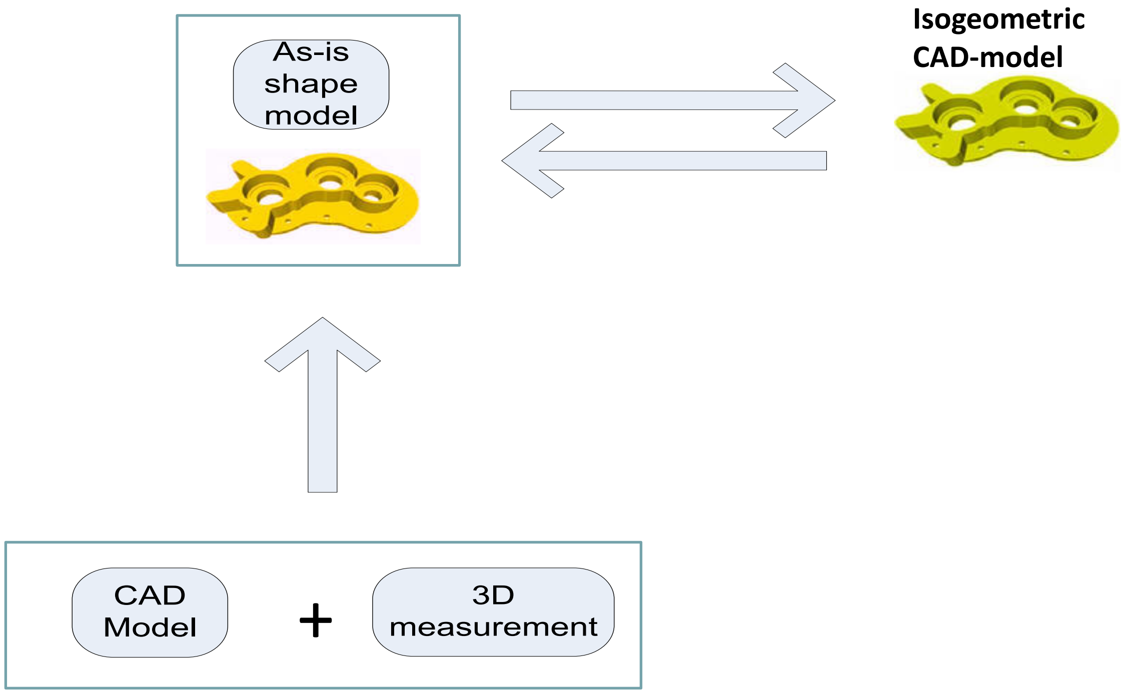
The basis functions are often higher order polinomials, but not the same used to describe the geometry. Why?

Isogoemetric analysis

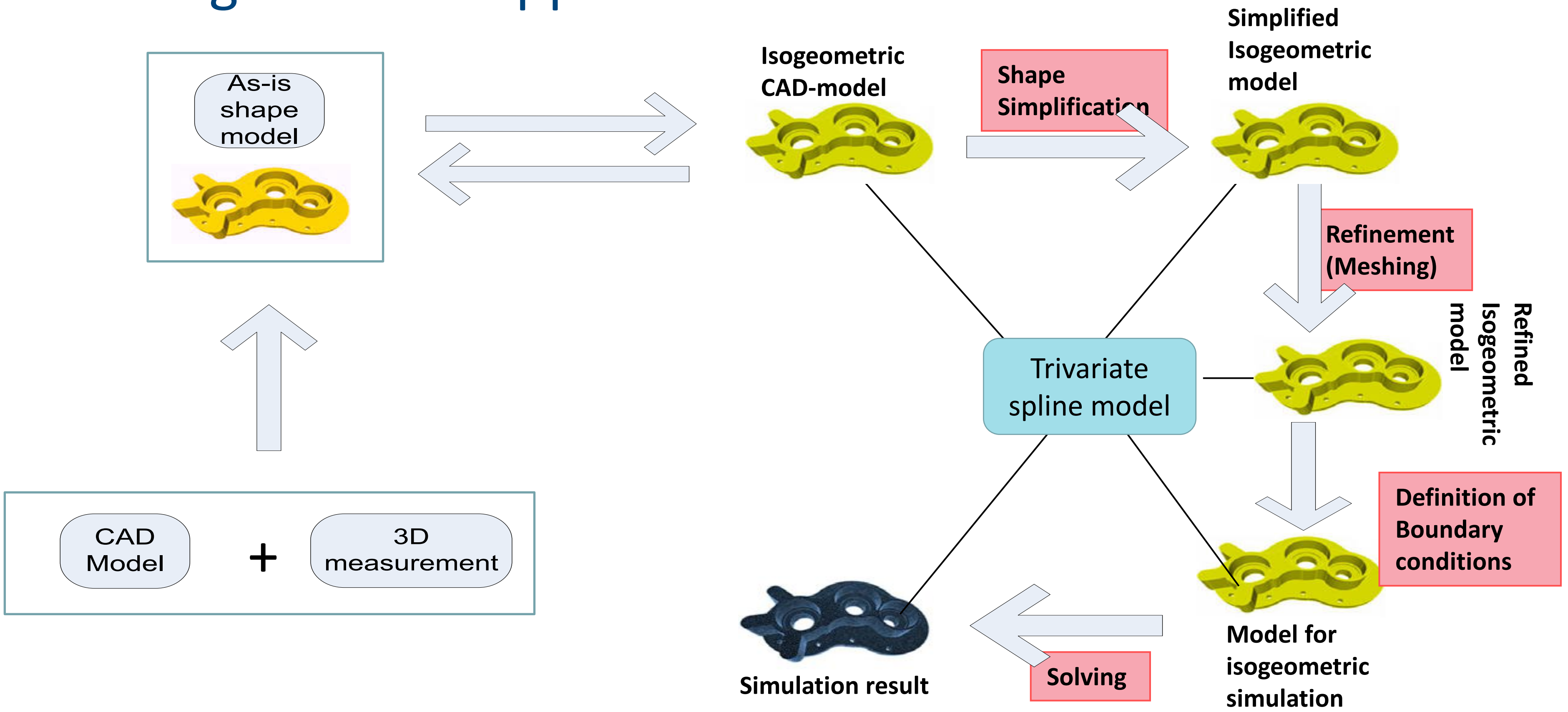
Vision: *Use the same model throughout the objects lifetime.*

- NURBS are used also for the solution
- Pioneered by Tom Houghes from University of Texas
- Has proven to yield high quality analysis results
- The most promising approach for unified data representation

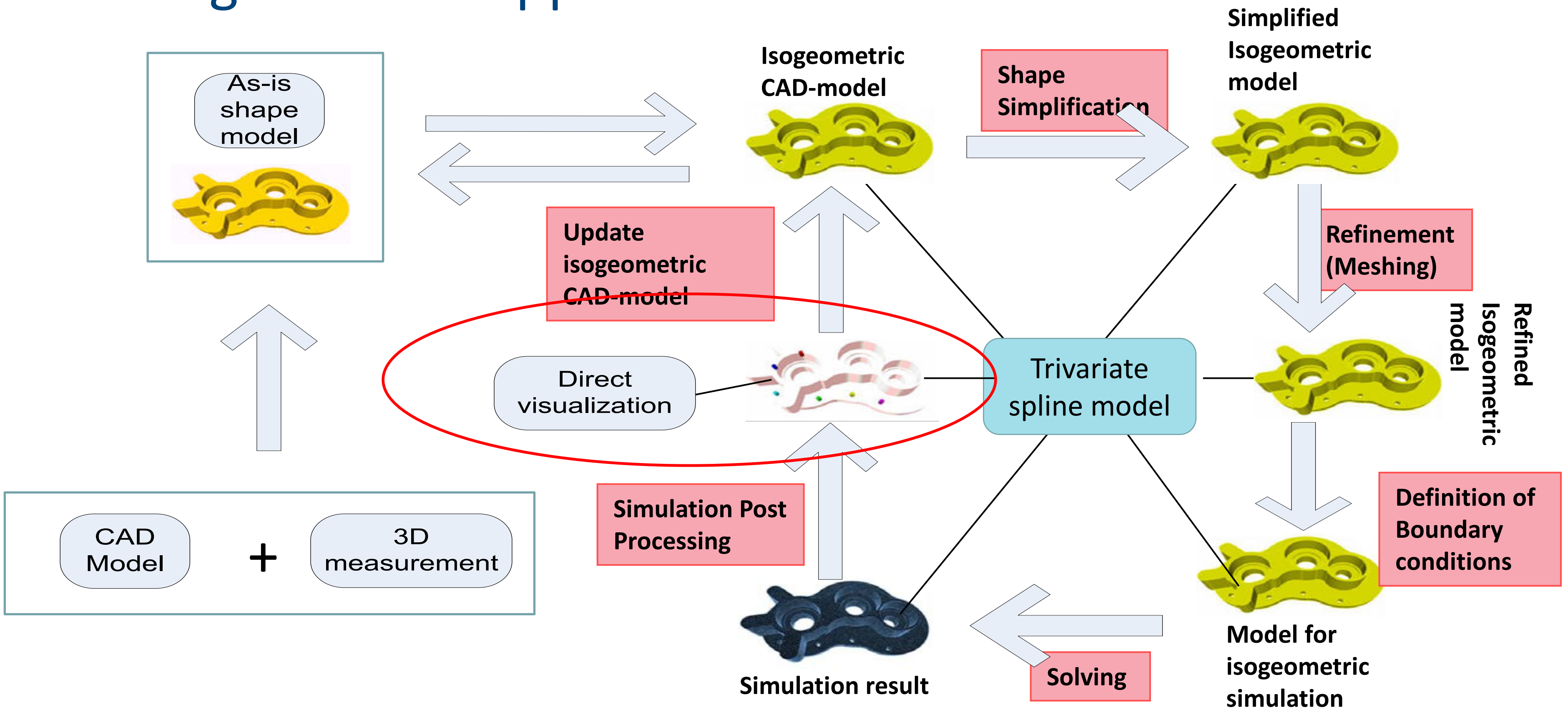
The Isogeometric Approach



The Isogeometric Approach



The Isogeometric Approach

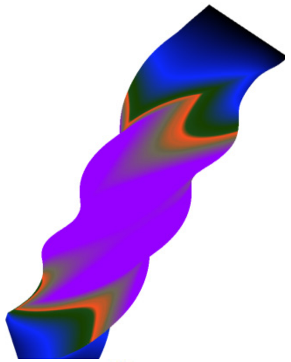

As-is shape model
Isogeometric CAD-model
Shape Simplification
Simplified Isogeometric model
Update isogeometric CAD-model
Refinement (Meshing)
Refined Isogeometric model
Direct visualization
Trivariate spline model
Simulation Post Processing
Definition of Boundary conditions
CAD Model
+
3D measurement
Simulation result
Solving
Model for isogeometric simulation

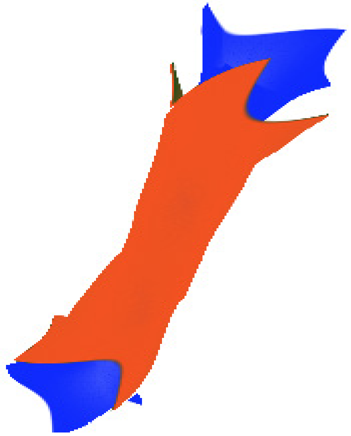
Visualization of Simulation Results

Engineers need reliable visualization of simulation data on the CAD model

- Render the bounding surface of the model
  - Pixel perfect rendering demonstrated
  - Gives no information of the inside

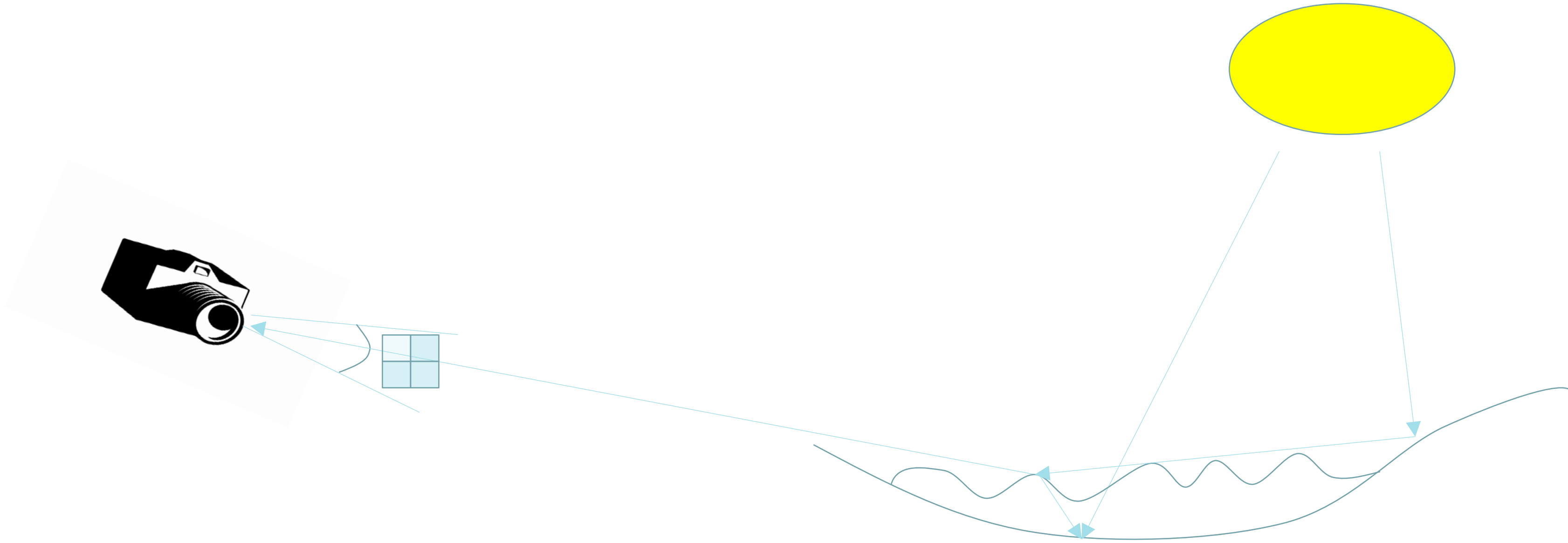
- Iso-surface extraction
  - Render the surface where the simulation has a given result
    - Temperature equals 10°C





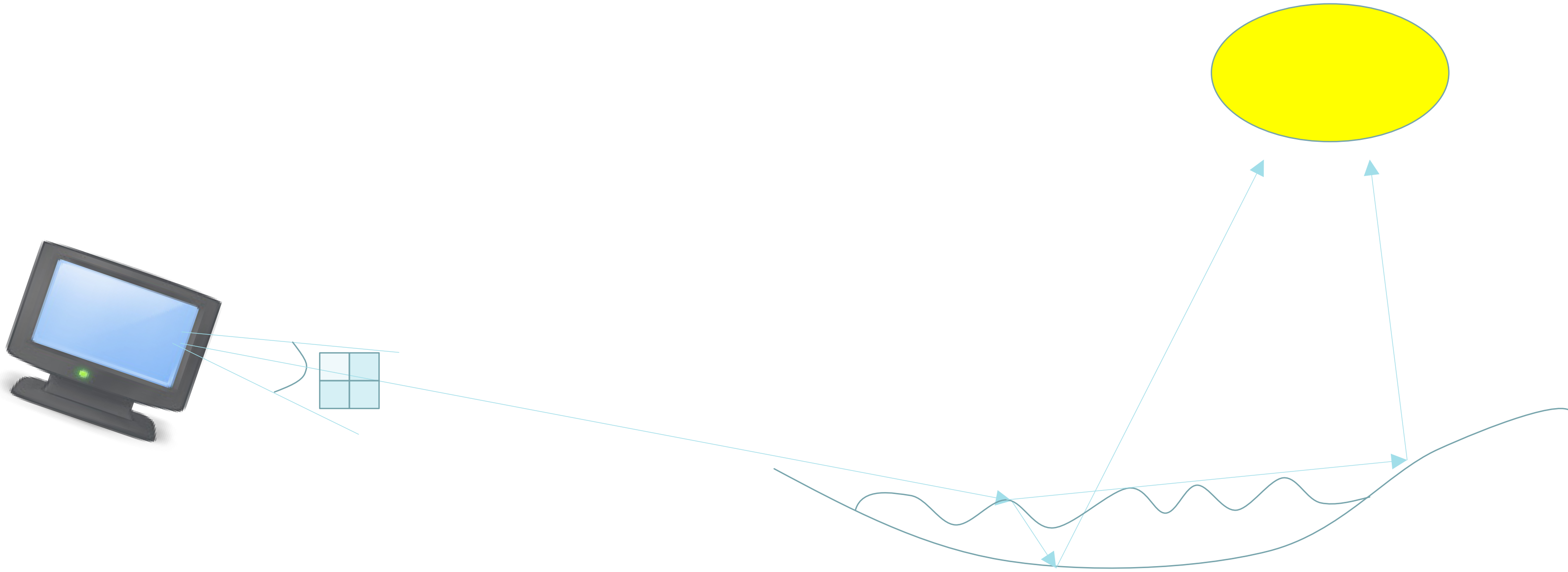
Ray Tracing, Background

A digital camera has a set of pixels, detecting incoming light.



Ray Tracing

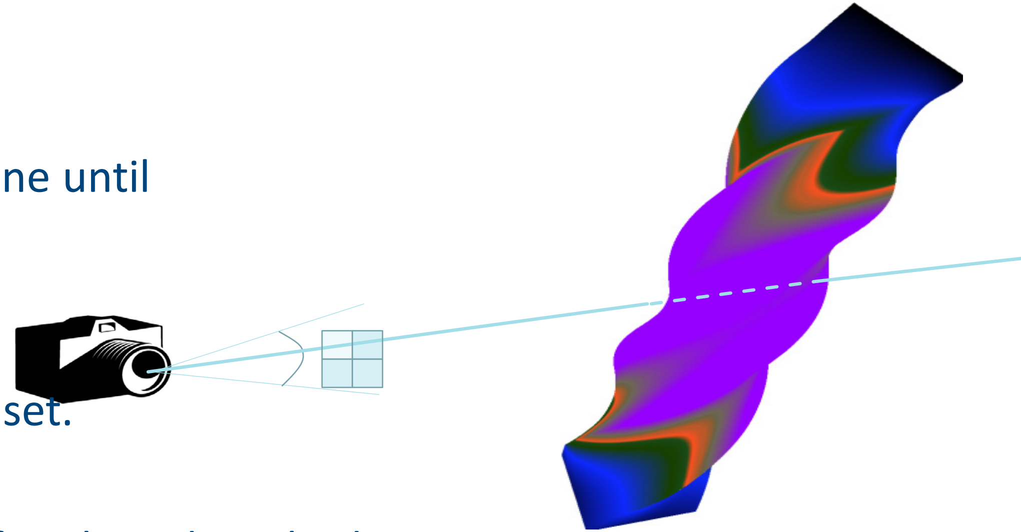
Ray tracing follows rays back through a virtual scene.



Volumetric Ray Casting

Ray-casting is ray-tracing's cheap sibling

- Follows one ray per pixel in a straight line until
  - It leaves the scene
  - It reaches opaque surface



- Provides insight from inside of the data set.
- Flexible
- Allows more diffuse rendering than surface based methods
- Computationally intensive
- Well known for box-based scalar fields

Setting Color to the Volume

In volume visualization it is common to use a transfer function describing the light emission and absorption throughout the volume.

$$I(s_n) = I(s_0)\underbrace{e^{-\int_{s_0}^{s_n} \kappa(t)\,dt}}_{\text{absorption}} + \int_{s_0}^{s_n} \overbrace{g(s)}^{\text{emission}} \underbrace{e^{-\int_{s}^{s_n} \kappa(t)\,dt}}_{\text{absorption}} ds$$

- The emission and absorption terms depend on the material the light pass
  - In this case: simulation result
  - A transfer function is used to map scalar values into material properties

Ray-Casting of Isogeometric Models

Find all ray-surface intersections

Sort the ray-surface intersections

Perform the ray-casting

Pixel-Perfect Spline rendering

- The rasterizer is designed as a blistering fast ray-triangle intersection implementation
- Each bounding surface can be sampled to form a triangulation.
- The questions are:
  - How many triangles do we need?
  - Where should the triangles be located?
  - Can we ensure correctness?
  - Will it be fast?
    - All computations performed at the GPU
    - Triangles never leave the GPU

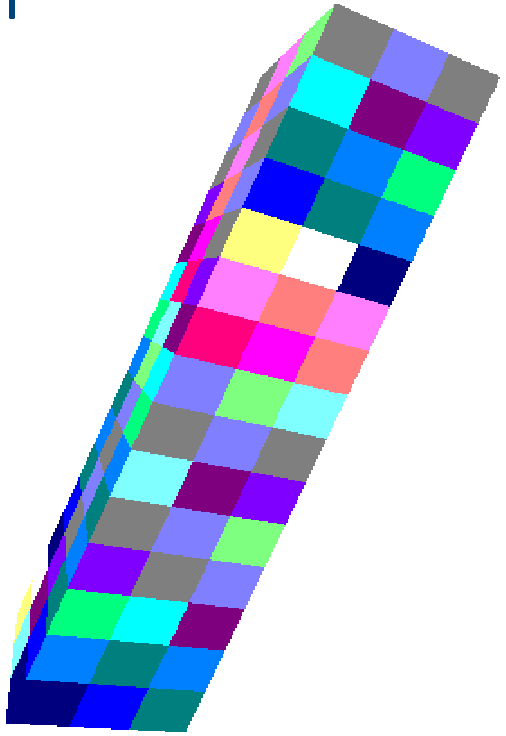
Hardware Tessellator

Split the surface into patches

Compute tessellation level(s) for each patch

Let the GPU generate the connectivity

Evaluate the surface position for each sample



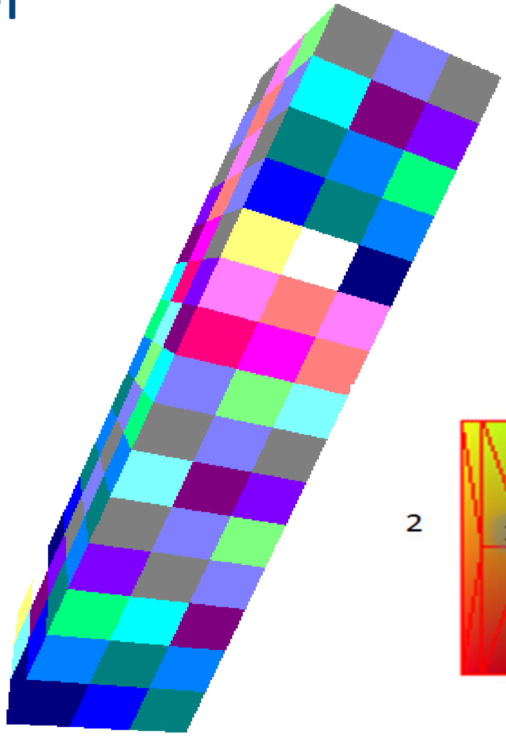
Hardware Tessellator

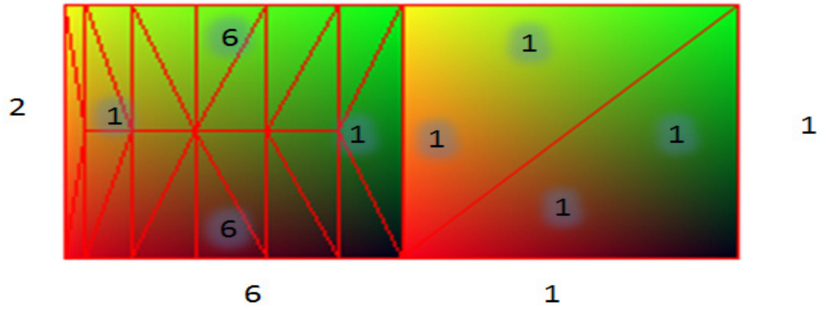
Split the surface into patches

Compute tessellation level(s) for each patch

Let the GPU generate the connectivity

Evaluate the surface position for each sample





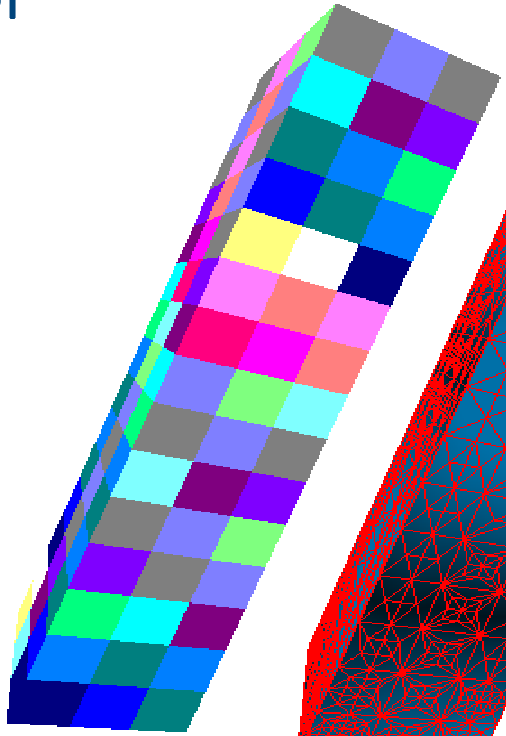
Hardware Tessellator

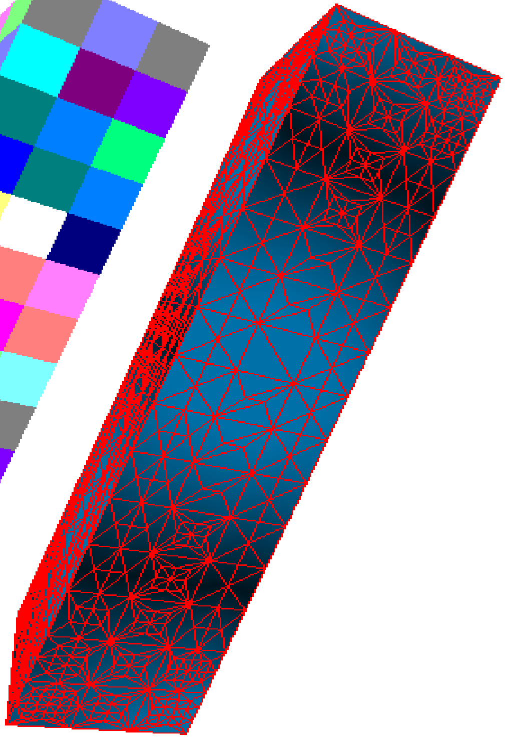
Split the surface into patches

Compute tessellation level(s) for each patch

Let the GPU generate the connectivity

Evaluate the surface position for each sample





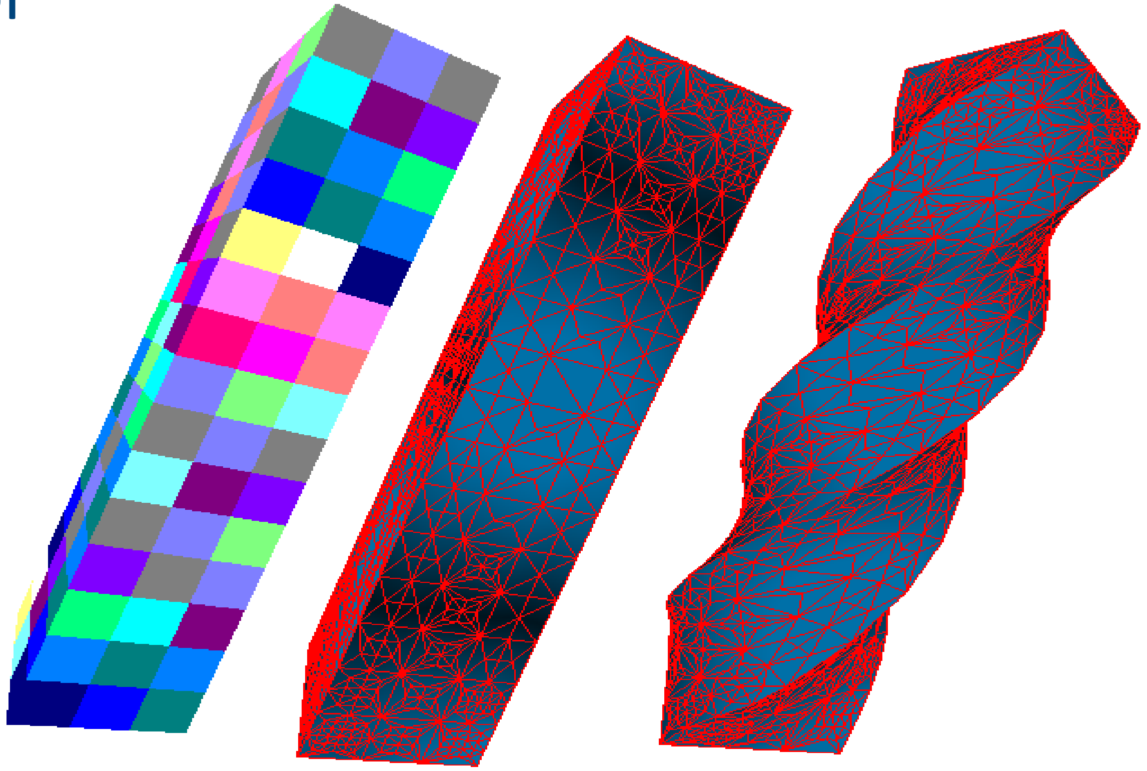
Hardware Tessellator

Split the surface into patches

Compute tessellation level(s) for each patch

Let the GPU generate the connectivity

Evaluate the surface position for each sample



Tessellation Requirements

- Each bounding surface is rendered independently
- Algorithm will be generalized to locally refined splines
- Watertightess
    - Abutting patch edges share tessellation level
    - Achieved by only using information from the edge
- Correctness
    - Can be achieved using upper bound on second order derivatives

Curve sampling

- How dense do we need to sample a curve?
  - Given the projection matrix
  - And a screen space tolerance (e.g. in pixels)

- If the curve is sufficiently smooth, we know that

- The second order de $|I_2 g - g| \le \left(\frac{\Delta}{2}\right) \max g''$ ted in eye space
- How does the projection fit?

Curve sampling formula

- Based on the previous formula, we find that a C^2 curve g, should be sampled by:

$$\Delta_u \quad \leq \quad \frac{2\epsilon^y}{C} \Big/ \left(\frac{\lceil g_z'' \rceil \lceil y \rceil}{\lfloor z^2 \rfloor} + \frac{\lceil g_y'' \rceil}{\lfloor z \rfloor} \right)$$

Where ε^y is screenspace tolerance in y direction, g''_y is the y component of the second derivative of the curve.

By symmetry we find the similar requirement from ε^x

Note: Only information belonging to the edge is used, guaranteeing watertightness

Interior Sampling Distance

Based on interpolation error for Lagrange Interpolation:

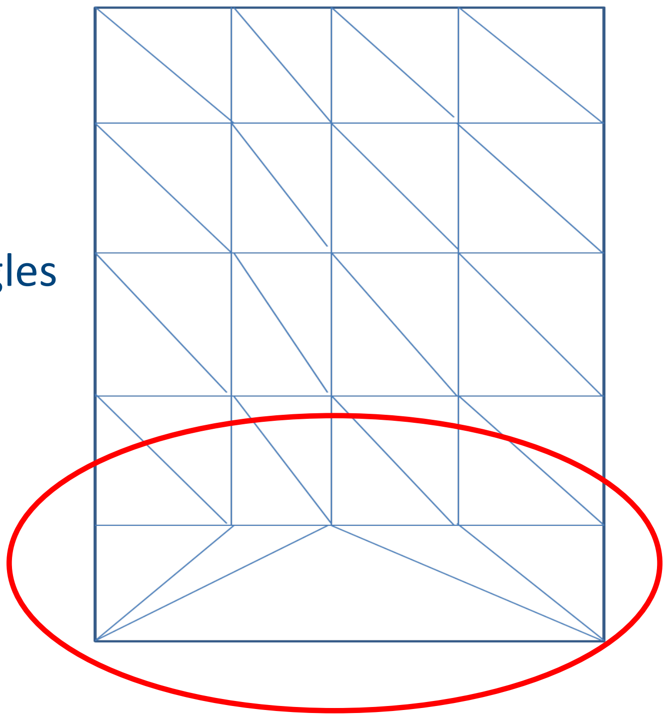
$$|R(u,v)| \leq \frac{\Delta_u^2}{8} \max |fuu| + \frac{\Delta_v^2}{8} \max |fvv| + \Delta u \Delta v \max |fuv|$$

Which leads to:

$$\begin{aligned} \frac{8\epsilon^x}{A} \quad \geq \quad & \Delta_u^2 \left(\frac{\lceil z_{uu} \rceil \lceil x \rceil}{\lfloor z^2 \rfloor} + \frac{\lceil x_{uu} \rceil}{\lfloor z \rfloor} \right) \\ & + \Delta_v^2 \left(\frac{\lceil z_{vv} \rceil \lceil x \rceil}{\lfloor z^2 \rfloor} + \frac{\lceil x_{vv} \rceil}{\lfloor z \rfloor} \right) \\ & + 8\Delta_u \Delta_v \left(\frac{\lceil z_{uv} \rceil \lceil x \rceil}{\lfloor z^2 \rfloor} + \frac{\lceil x_{uv} \rceil}{\lfloor z \rfloor} \right) \end{aligned}$$

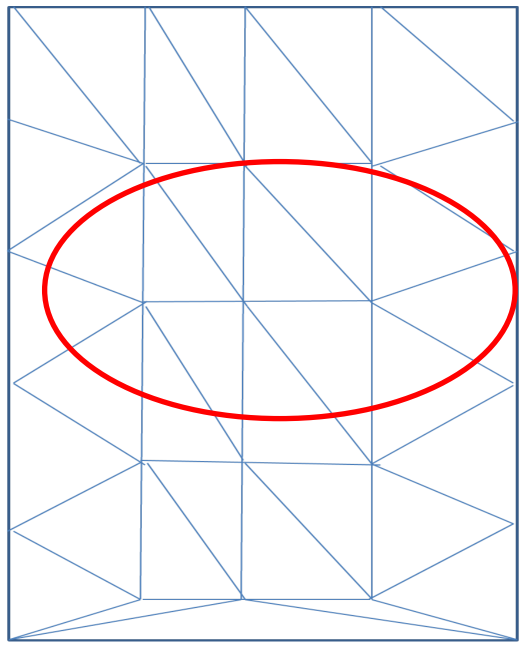
Boundary adjustment

- Tolerance may be exceeded near the boundary:
- Control shader can detect where this may happen
  - Compute a new height for the first row of triangles
  - Other areas may now need higher tessellation

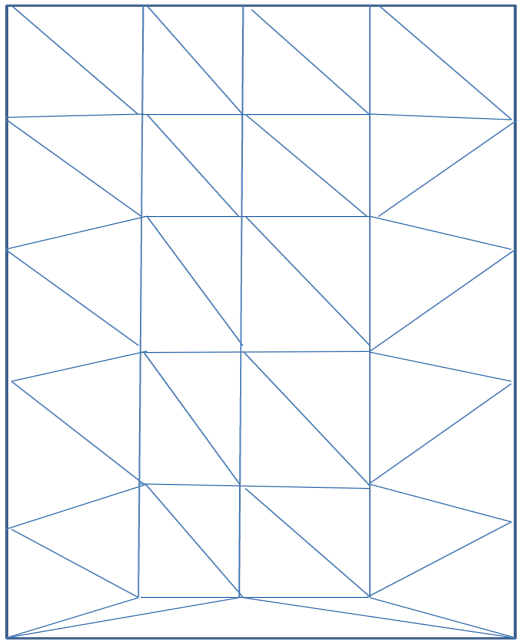


Boundary adjustment

- Tolerance may be exceeded near the boundary:
- Control shader can detect where this may happen
  - Compute a new height for the first row of triangles
  - Other areas may now need higher tessellation
- Control shader can detect if extra row of triangles is required



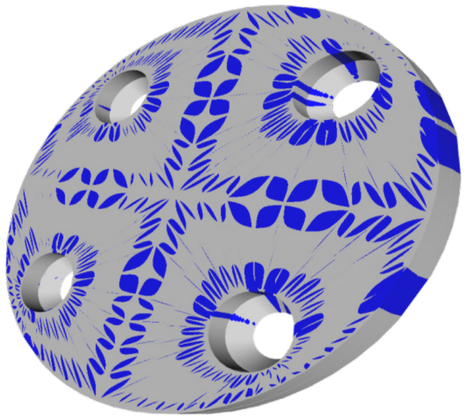
Boundary adjustment

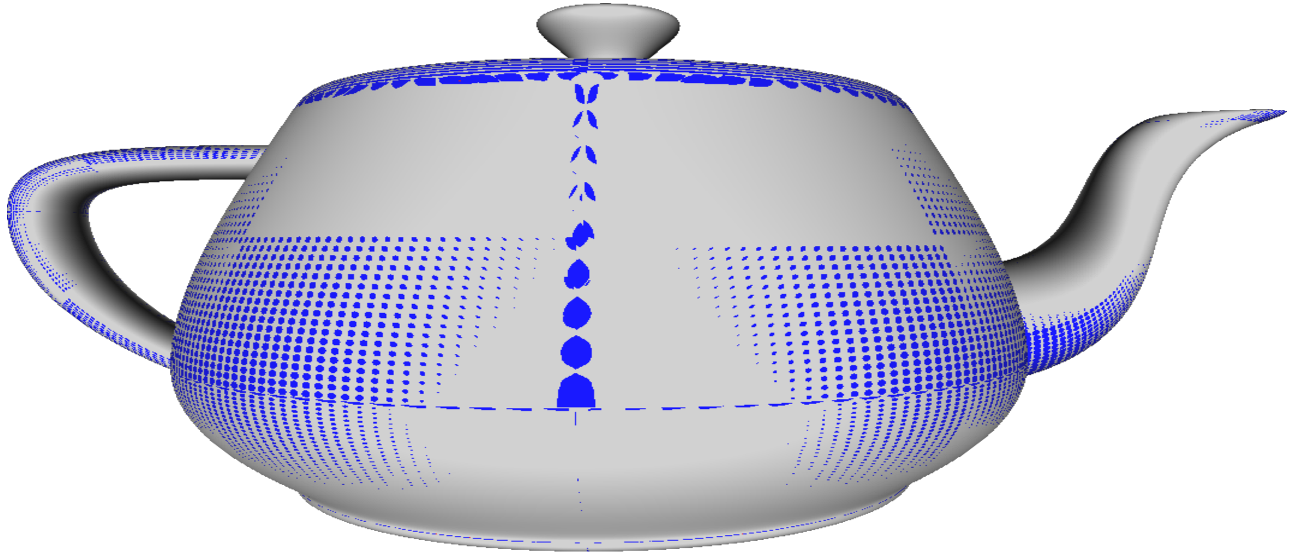


- Tolerance may be exceeded near the boundary:
- Control shader can detect where this may happen
  - Compute a new height for the first row of triangles
  - Other areas may now need higher tessellation
- Control shader can detect if extra row of triangles is required

- Note: All decisions is performed in the control shader
- The evaluation shader performs a affine transformation to the parameter values to stretch the domain

Accuracy





Verification of approximation error: Areas with tessellation error between 0.1 and 1 pixel are marked with blue, the remaining areas have smaller error.

Tessellation Demo

http://www.youtube.com/watch?v=KOsDBx8yEt0

Properties

- Accurate

- Possible to extend to large scale models
  - Less than two triangles per Beziér patch

- Extendable beyond Spine surfaces

- Single pass rendering
  - Easy to integrate without loss of performance

Ray-Casting of Isogeometric Models

Render bounding surface

Sort the ray-surface intersections

March along each ray

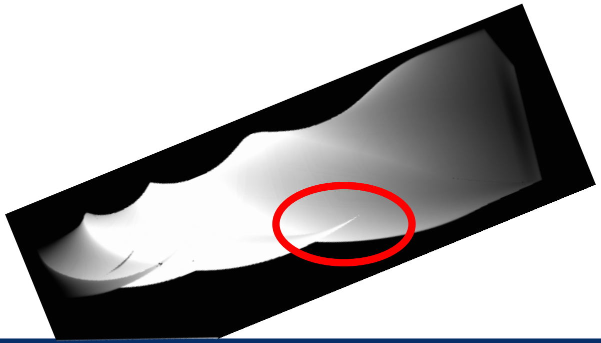
LR-Splines Teaser

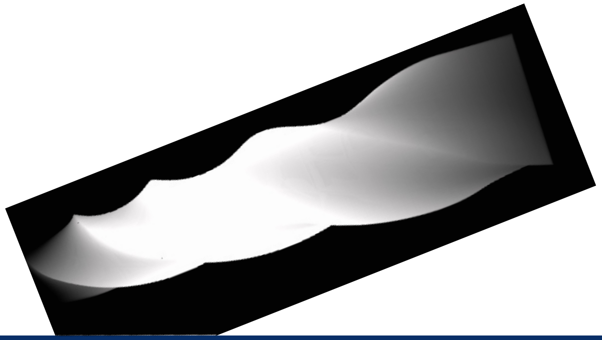
http://www.youtube.com/watch?v=49789yhNz6M

Rendering semi-transparent objects

Computer Graphics use depth test to find the first ray-triangle intersection

- Semi-transparent objects may be sorted on the CPU and rendered in back-to-front order (or front-to-back)
- Incorrect ordering gives visual artifacts
- Sorting is complicated for non-convex objects





Depth-Peeling

Popular multi pass rendering method

- First pass renders the shell closest to the camera (standard)
- Subsequent pass use depth buffer as input texture
- Each subsequent pass renders the next layer away from the camera

Performance considerations

- Sending the geometry through several passes may be expensive
  - Can be improved by using transform buffers
- Number of passes may be difficult to determine
- Deferred rendering will require extra memory

Linked List Approach

Two-pass rendering method where the first pass records all hits and stores them in a linked list.

- The fragment shader use a global atomic counter for memory management
- Each new hit is stored at the head of the list
- First implementation used Direct3D
  - Due to requirement of atomic operations available in DirectX11 and OpenGL4.2

- The memory usage may be reduced by allocating a memory for a set of hits (pages).

Linked List OIT Improved

Based on the work of Crassin on order-independent transparency, we implemented an algorithm for sorting ray-model intersections.

- To reduce the load on the counter each allocation gives memory page for four hits
- A per-pixel atomic counter is used to index into the page
- A per-pixel atomic counter is used as a mutex

- The linked list is traversed and the hits are sorted as a part of the ray-casting shader stage

Last Minute Update

Ethan Kerzner has a OIT poster presentation here at GTC, requiring a last minute update.

- The memory can be allocated in the control shader
- We need to estimate the number of pixels created from a patch
- One atomic counter per patch
- No need for mutex

Ray-Casting of Isogeometric Models

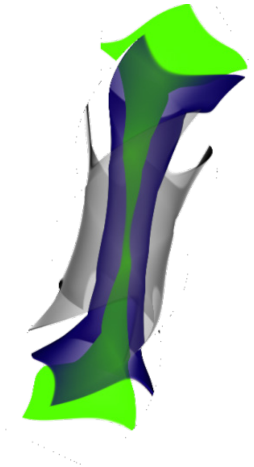
Render bounding surface

Sort the ray-surface intersections

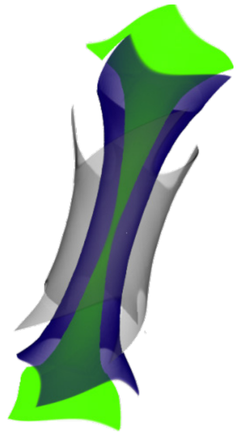
March along each ray

Linear Ray-Casting

The rays are supposed to be linear in the geometry domain. In our case, this is not linear in the parameter domain.



Linear in parameter domain



Linear in geometry domain

Curve marching

- Computing the inverse of spline volumes is ineficient
  - $\longrightarrow$ Step direction and size must be in parameter space

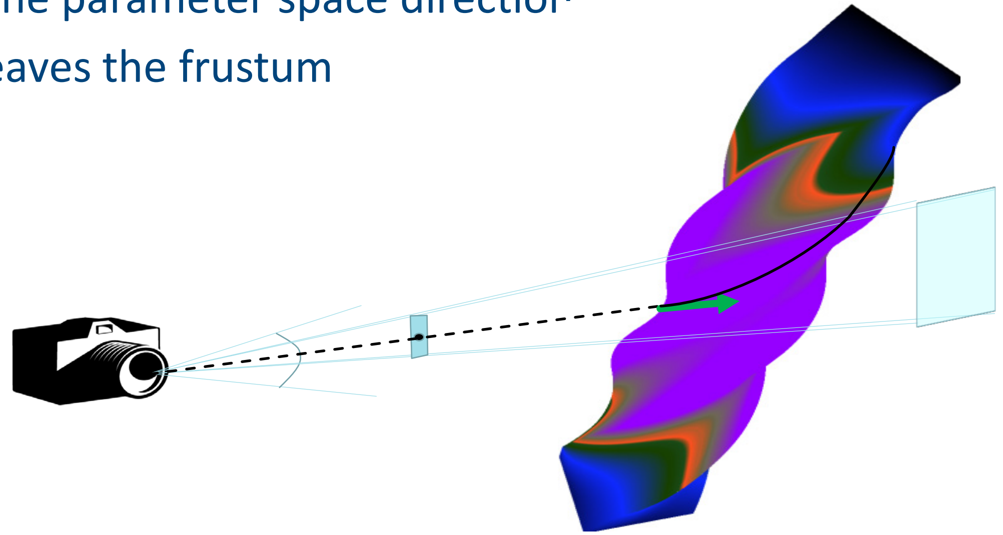
- Remember: a both the geometry and solution fields are represented as trivariate splines, e.g.

$$V(u, v, w) = \sum_i \sum_j \sum_k c_{i,j,k} B_i(u) B_j(v) B_k(w)$$

- To achieve pixel accurate rendering
  - Sample curve must be within the pixels' frustum
  - Scalar field must be sampled sufficiently dense to capture details

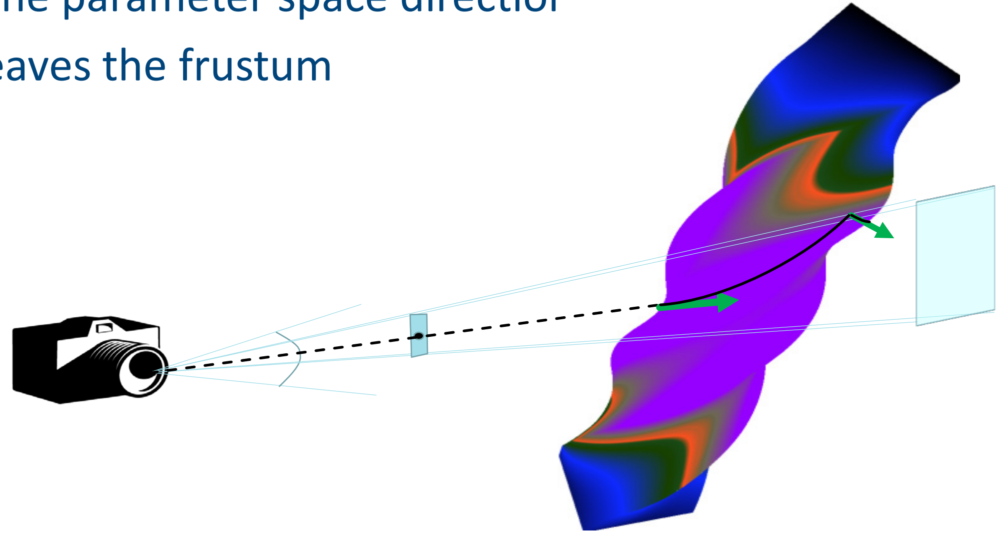
Ray-Casting Curve Example

- Shoot a ray through the pixel center until it hits the bounding surface
- Find a model space direction to follow into the volume
- Use Jacobian of the volume to find the parameter space direction
- Determine where the space curve leaves the frustum
- Find a new direction
  - And curve



Ray-Casting Curve Example

- Shoot a ray through the pixel center until it hits the bounding surface
- Find a model space direction to follow into the volume
- Use Jacobian of the volume to find the parameter space direction
- Determine where the space curve leaves the frustum
- Find a new direction
    - And curve



Cutting Curves

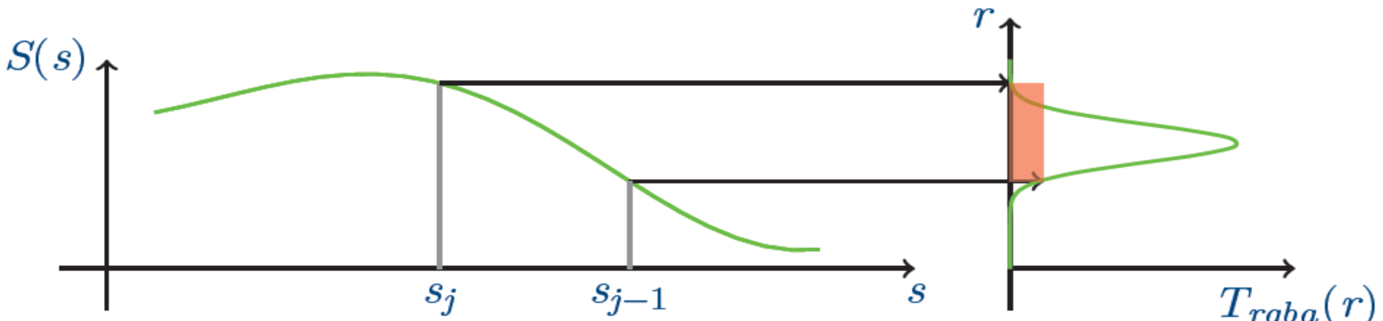
How do we detect when a curve leaves the frustum?

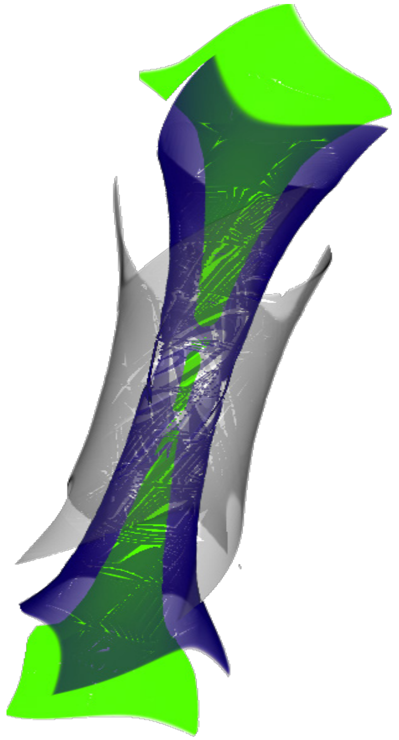
- First we split the curve in polygonal regions
  - Determining where it leaves a knot interval
- Each region can be expressed as Beziér curves
  - Using blossoming
- It is possible to use the control polygon to find where the curve leaves the frustum
  - May be too expensive in practice

Scalar field sampling density

The required sampling density is dependent on

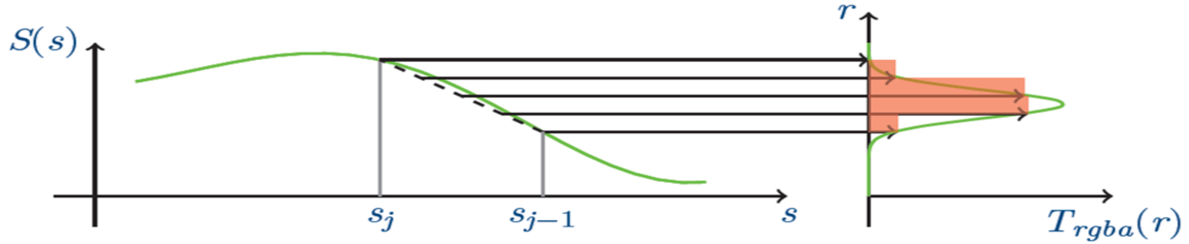
- The range of the scalar values
- The derivatives of the transfer function
  - Can be remedied by pre-integrated transfer function
- The depth complexity of the curve



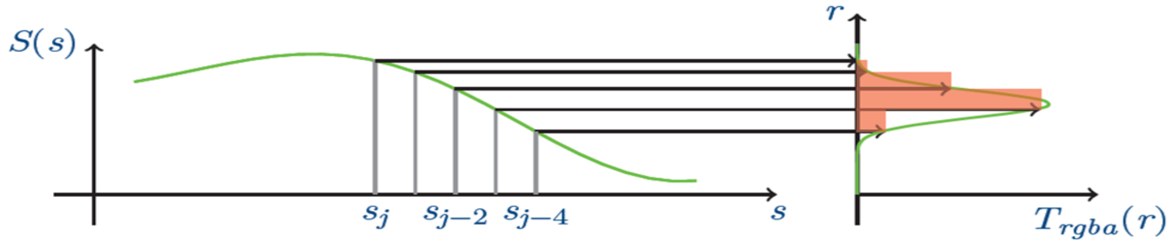


Subsampling Transfer Function

The artefacts can be removed either by only subsampling the transfer function

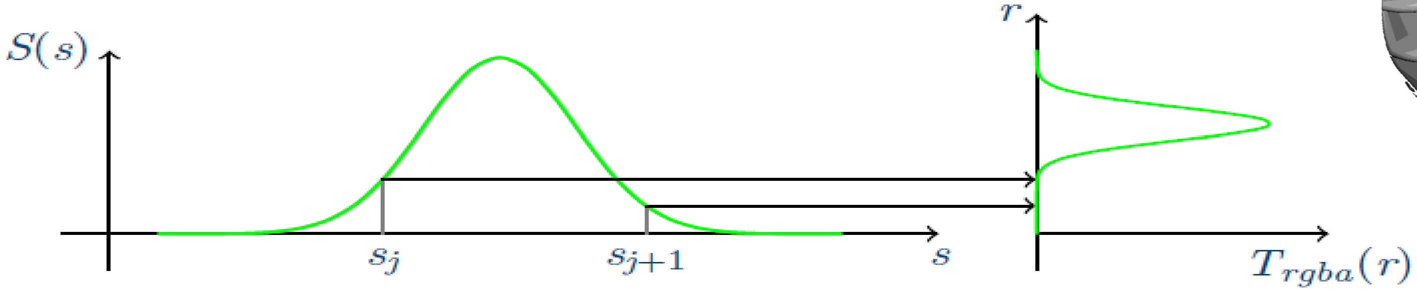


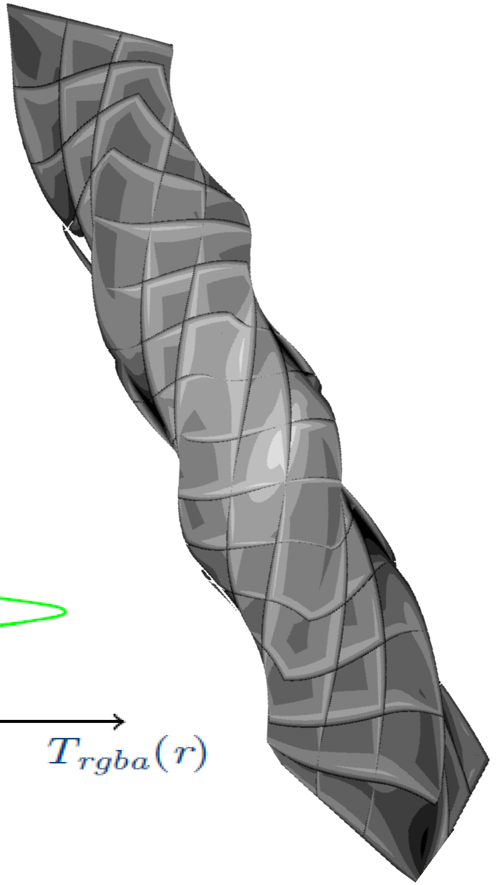
or also the scalar field



Subsampling Transfer Function

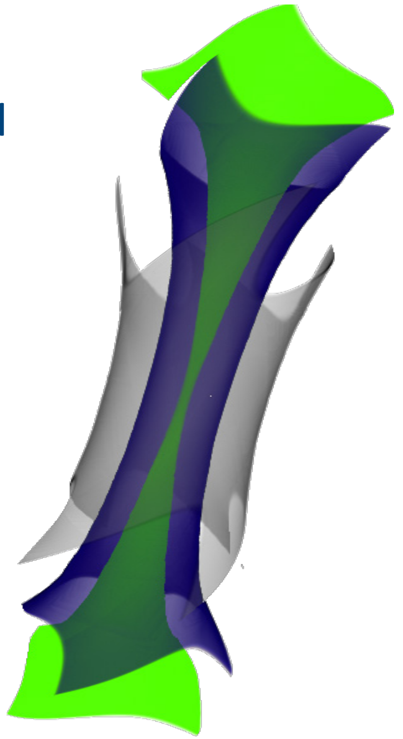
If both the scalar function and the transfer function deviates from linear to subsampling the transfer function alone is not enough





Adaptive sampling

- Compute the derivative of the transfer function
- Create mipmaped texture containing the maximal value for each texel
  - Mipmapped
- Find the max and min scalar values for the curve
  - Use the control polygon
- Lookup the derivative function to find an appropriate step size
  - Use the mipmap layer where the range only hits two texels



Compute Shader

During the progress of this work we have investigated several implementations

- GLSL using deferred fragment shader
    - The most standard implementation
- GLSL using compute shader
    - Very easy to integrate into existing OpenGL application
- CUDA
    - More optimalization opportunities
    - Best profiling tools

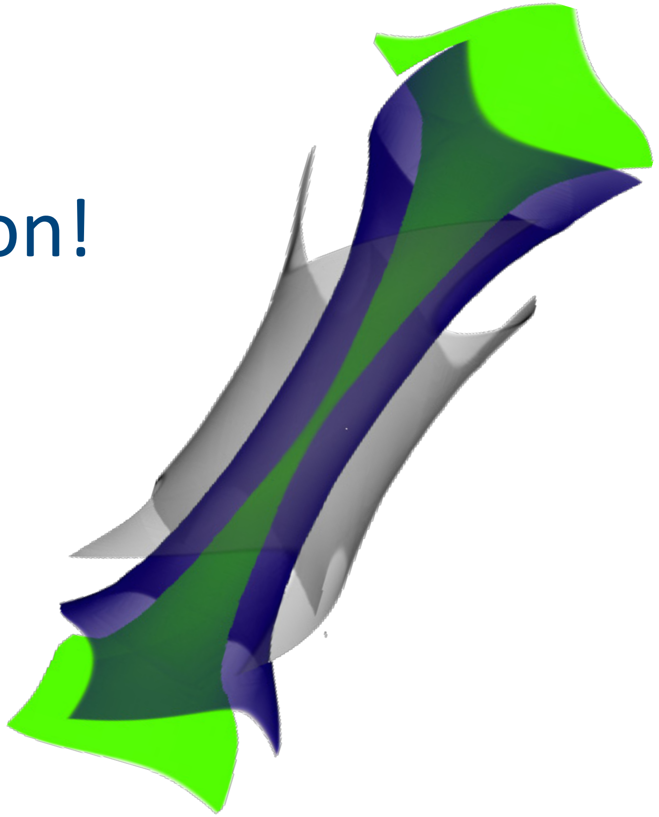
In our experience, the fragment shader gave the best performance, since the ray caster is fully compute bound

Demo

http://www.youtube.com/watch?v=TbiawHZitR8

Thank you for your attention!

Questions?



Contact info:

Email: jon.hjelmervik@sintef.no

Youtube: www.youtube.com/user/JonHjelmervik