

# Ray Casting of Trimmed NURBS Surfaces on the GPU

Hans-Friedrich Pabst   Jan P. Springer   André Schollmeyer  
Robert Lenhardt   Christian Lessig   Bernd Fröhlich

Bauhaus University Weimar · Faculty of Media · Virtual Reality Systems Group

IEEE Symposium on Interactive Ray Tracing 2006

# Motivation

- ▶ Trimmed NURBS surfaces
  - ▶ CAD standard
- ▶ Ray casting
  - ▶ Direct rendering
  - ▶ Pixel-accurate
- ▶ GPU
  - ▶ Lots of gigaflops per value



## Main Goal

Interactive rendering trimmed NURBS surfaces using ray casting on commodity hardware.

# Motivation

- ▶ Trimmed NURBS surfaces
  - ▶ CAD standard
- ▶ Ray casting
  - ▶ Direct rendering
  - ▶ Pixel-accurate
- ▶ GPU
  - ▶ Lots of gigaflops per value

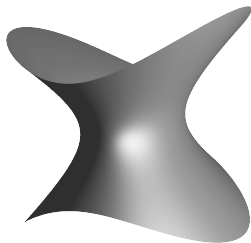


## Main Goal

Interactive rendering trimmed NURBS surfaces using ray casting on commodity hardware.

# NURBS

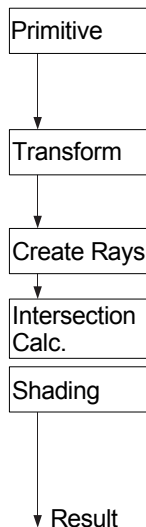
- ▶ NURBS surfaces provide local and explicit control
- ▶ Primitives included, e. g. curve, sphere, cone, cube
- ▶ Compact representation
- ▶ Continuity between curves and patches
- ▶ Trimming allows complex boundaries and topologies



# Outline

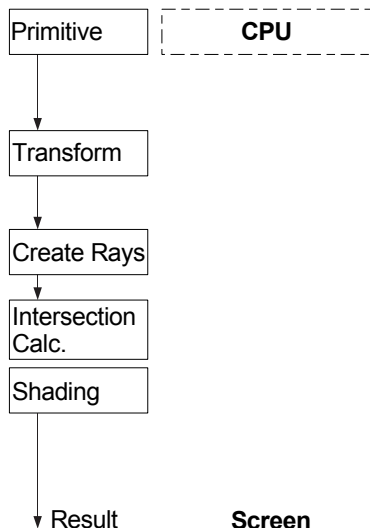
- ▶ Integrate NURBS primitives into hardware graphics pipeline
- ▶ Ray-NURBS intersection and accurate trimming on the GPU
- ▶ Demonstration
- ▶ Results and conclusions

# Surface Rendering: Algorithm Overview



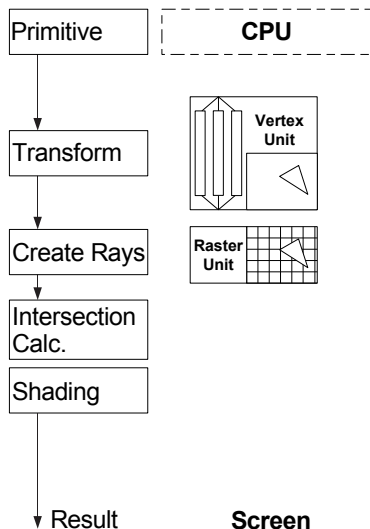
- ▶ Preprocessing
  - ▶ Create bounding volume (convex hull)
  - ▶ Send vertices and parametric data
- ▶ For each surface
  - ▶ Transform convex hull
  - ▶ Rasterize convex hull
  - ▶ Ray-sphere intersection
  - ▶ Ray-triangle intersection
  - ▶ Ray-plane intersection
  - ▶ Ray-ray intersection
  - ▶ Shading

# Surface Rendering: Algorithm Overview



- ▶ Preprocessing
  - ▶ Create bounding volume (convex hull)
  - ▶ Send vertices and parametric data
- ▶ For each surface
  - ▶ Transform convex hull
  - ▶ Rasterize convex hull
  - ▶ Compute ray-surface intersection
  - ▶ Trimming
  - ▶ Shading

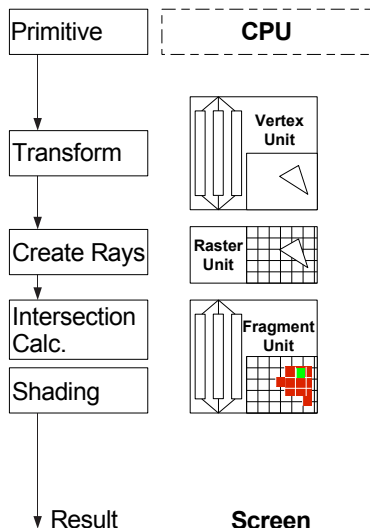
# Surface Rendering: Algorithm Overview



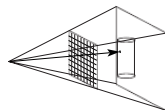
- ▶ Preprocessing
  - ▶ Create bounding volume (convex hull)
  - ▶ Send vertices and parametric data
- ▶ For each surface
  - ▶ Transform convex hull
  - ▶ Rasterize convex hull
  - ▶ Compute ray-surface intersection
  - ▶ Trimming
  - ▶ Shading



# Surface Rendering: Algorithm Overview

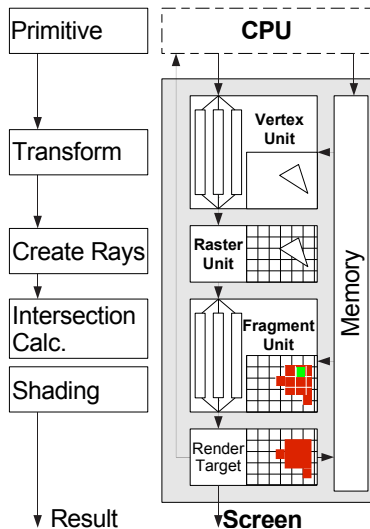


- ▶ Preprocessing
  - ▶ Create bounding volume (convex hull)
  - ▶ Send vertices and parametric data
- ▶ For each surface
  - ▶ Transform convex hull
  - ▶ Rasterize convex hull
  - ▶ Compute ray-surface intersection

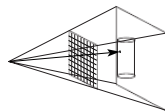


- ▶ Trimming
- ▶ Shading

# Surface Rendering: Algorithm Overview



- ▶ Preprocessing
  - ▶ Create bounding volume (convex hull)
  - ▶ Send vertices and parametric data
- ▶ For each surface
  - ▶ Transform convex hull
  - ▶ Rasterize convex hull
  - ▶ Compute ray-surface intersection



- ▶ Trimming
- ▶ Shading

# Numeric Intersection Computation

- ▶ General root finding vs. geometrical context
- ▶ Methods
  - ▶ Subdivision
  - ▶ Numerical (iterative)
  - ▶ Algebraic
  - ▶ Hybrid
  
- ▶ Newton Iteration
  - ▶ Only parameters of one step necessary
  - ▶ Only function values and partial derivatives needed
  - ▶ Quadratic convergence

# Numeric Intersection Computation

- ▶ General root finding vs. geometrical context
- ▶ Methods
  - ▶ Subdivision
  - ▶ Numerical (iterative)
  - ▶ Algebraic
  - ▶ Hybrid
  
- ▶ Newton Iteration
  - ▶ Only parameters of one step necessary
  - ▶ Only function values and partial derivatives needed
  - ▶ Quadratic convergence

# Numeric Intersection Computation

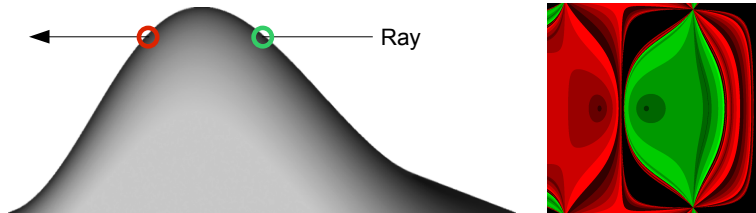
- ▶ General root finding vs. geometrical context
- ▶ Methods
  - ▶ Subdivision
  - ▶ Numerical (iterative)
  - ▶ Algebraic
  - ▶ Hybrid



- ▶ Newton Iteration
  - ▶ Only parameters of one step necessary
  - ▶ Only function values and partial derivatives needed
  - ▶ Quadratic convergence

# Initial Values for the Newton Iteration

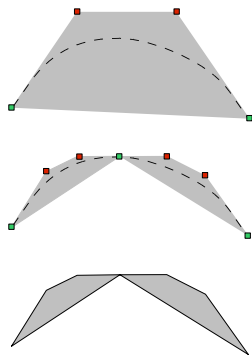
**Problem:** An approximate solution is needed to get a solution  
→ Information about geometrical context can be used



- ▶ Two complementary approaches: subdivision and  $uv$ -texturing
- ▶ Motivation: good initial values will result in fast convergence

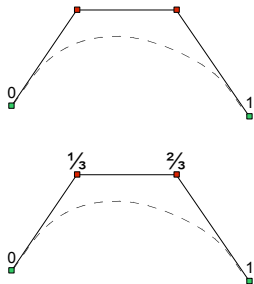
# Subdivision of the Convex Hull

- ▶ Quadratically increasing tightness
- ▶ Trade-off between ray casting and standard graphics pipeline
  - ▶ Minimizes number of fragments/rays
  - ▶ Number of vertices increased
- ▶ Adaptive subdivision
  - ▶ Minimizes number of generated vertices
- ▶ Union of all convex hulls
  - ▶ Approximation of the surface



# Control Point Mapping

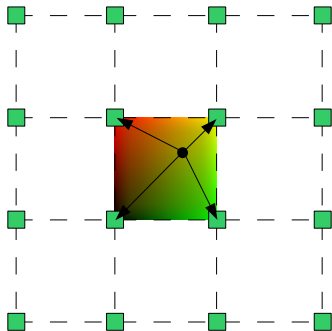
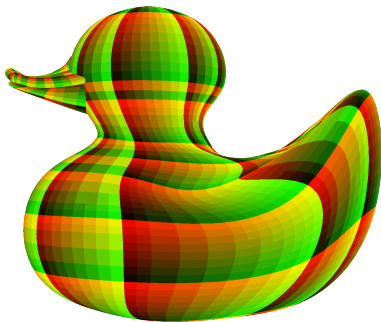
- ▶ Complement outer control points
- ▶ Mapping parameter range between
- ▶ Make use of vertex attributes
- ▶ Control point mapping is subdivision-aware
- ▶ Subdivision increases quality
- ▶ Valid for control meshes
- ▶ Valid for convex hull





# UV-Texturing

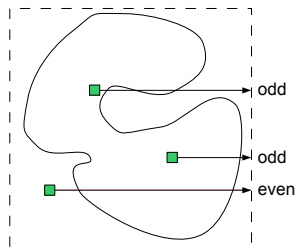
- ▶ Interpolated guess for each ray
- ▶ View-dependent  $uv$ -guess



*"Subdivision together with texturing the convex hull generates appropriate initial values."*

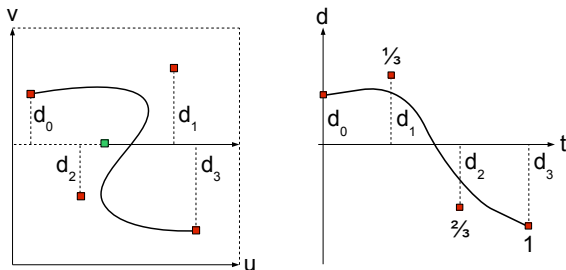
# Trimming: Algorithm Overview

- ▶ Ray casting in parameter domain
- ▶ Similar to point-in-polygon test
- ▶ Bézier form provides exact representation of NURBS curves
- ▶ Accurate intersection computation using Bézier Clipping



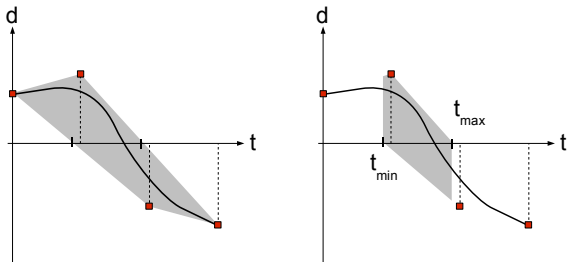
# Bézier Clipping

- ▶ Numerical root finding algorithm (subdivision)
- ▶ Makes use of the convex hull property



- ▶ Transformation into local equidistant coordinate system, invariant with respect to the intersection points

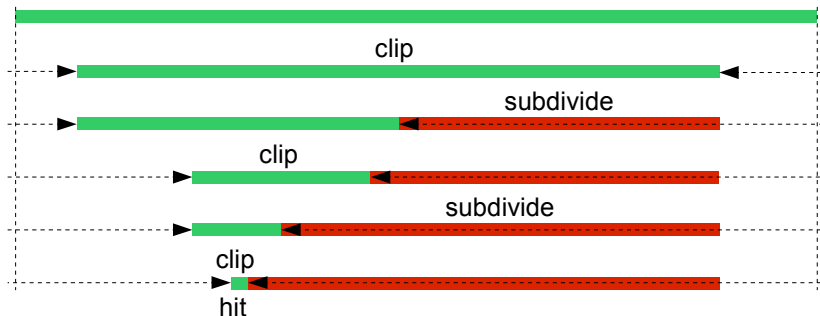
# Bézier Clipping (cont.)



- ▶ Compute convex hull intersections with  $t$ -axis
- ▶ Split curve at  $t_{min}$  and  $t_{max}$  ("clipping")
- ▶ Interval contraction driven by clipping and subdivision

# Iterative Bézier Clipping

**Problem:** Subdivision implies recursive processing of sub-intervals



- ▶ Only two intervals at a time are needed
- ▶ Only one scalar value needed to represent remaining interval

# Iterative Bézier Clipping (cont.)

- ▶ Favors re-computation over storing values
- ▶ Consists of a state machine inside a loop to simulate function calls
- ▶ Intersection test takes advantage of Bernstein-Bézier form
  
- ▶ Properties
  - ▶ Iterative depth-first algorithm
  - ▶ Enumerates roots in ascending order

*"This is the first implementation of a subdivision-like **single pass** algorithm on current graphics hardware."*

# Direct Trimming

- ▶ Interactive manipulation of existing control points possible
- ▶ Complements the direct rendering of surfaces
- ▶ Can also be used for trimming triangulated patches



# Limitations

- ▶ Hardware and tool chain
  - ▶ Registers, writeable memory
  - ▶ Compiler, graphics driver, debugging
- ▶ Algorithm
  - ▶ Artifacts (ray-surface intersection)
  - ▶ Trimming without acceleration data structure
- ▶ Large models
  - ▶ Usually one program per surface
  - ▶ Limited degree ( $\approx 6 \times 6$ ):  
 $M + 2N \leq 19$  with  $N \leq M$

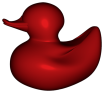







# The Trimmed Utah Teapot

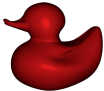



Iteration + Manipulation

# Results

Figure	Triangles	Subdivision	FPS <sub>4</sub>	FPS <sub>8</sub>	
Duck	3732	1 × 1	18	15	
	15648	2 × 2	20	16	
	63602	4 × 4	20	16	
Teapot	3092	2 × 2	33	24	
	12698	4 × 4	36	28	
	51160	8 × 8	30	25	
Teapot <sup>Orient</sup>	3092	2 × 2	22	18	
	12698	4 × 4	24	19	
	51160	8 × 8	22	18	
Teapot <sup>NV</sup>	3092	2 × 2	18	15	
	12698	4 × 4	19	16	
	51160	8 × 8	17	15	

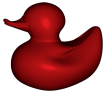



Resolution 1280 × 1024, screen covering 80% of width, GPU Geforce 7900 GT

# Results

Figure	Triangles	Subdivision	FPS <sub>4</sub>	FPS <sub>8</sub>	
Duck	3732	1 × 1	18	15	
	15648	2 × 2	20	16	
	63602	4 × 4	20	16	
Teapot	3092	2 × 2	33	24	
	12698	4 × 4	36	28	
	51160	8 × 8	30	25	
Teapot <sup>Orient</sup>	3092	2 × 2	22	18	
	12698	4 × 4	24	19	
	51160	8 × 8	22	18	
Teapot <sup>NV</sup>	3092	2 × 2	18	15	
	12698	4 × 4	19	16	
	51160	8 × 8	17	15	





Resolution 1280 × 1024, screen covering 80% of width, GPU Geforce 7900 GT

# Results

Figure	Triangles	Subdivision	FPS <sub>4</sub>	FPS <sub>8</sub>	
Duck	3732	1 × 1	18	15	
	15648	2 × 2	20	16	
	<b>63602</b>	4 × 4	20	16	
Teapot	3092	2 × 2	33	24	
	12698	4 × 4	36	28	
	<b>51160</b>	8 × 8	30	25	
Teapot <sup>Orient</sup>	3092	2 × 2	22	18	
	12698	4 × 4	24	19	
	<b>51160</b>	8 × 8	22	18	
Teapot <sup>NV</sup>	3092	2 × 2	18	15	
	12698	4 × 4	19	16	
	<b>51160</b>	8 × 8	17	15	

Resolution 1280 × 1024, screen covering 80 % of width, GPU Geforce 7900 GT

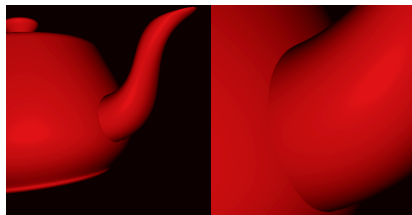
# Results

Figure	Triangles	Subdivision	FPS <sub>4</sub>	FPS <sub>8</sub>	
Duck	3732	1 × 1	18	15	
	15648	2 × 2	20	16	
	63602	4 × 4	20	16	
Teapot	3092	2 × 2	33	24	
	12698	4 × 4	36	28	
	51160	8 × 8	30	25	
Teapot <sup>Orient</sup>	3092	2 × 2	22	18	
	12698	4 × 4	24	19	
	51160	8 × 8	22	18	
Teapot <sup>NV</sup>	3092	2 × 2	18	15	
	12698	4 × 4	19	16	
	51160	8 × 8	17	15	

Resolution 1280 × 1024, screen covering 80% of width, GPU Geforce 7900 GT

# Conclusions

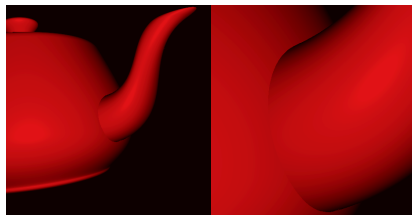
- ▶ Direct rendering
  - ▶ Ideal solution for CAD
  - ▶ Low CPU overhead
  - ▶ Minimal storage
- ▶ Pixel-accurate
  - ▶ Silhouettes
  - ▶ Interpenetrations
  - ▶ Normals
- ▶ Future Work
  - ▶ Reliable ray-surface intersection test
  - ▶ Trimming acceleration data structure



*"Higher order primitives will complement triangles as the primary rendering primitive."*

# Conclusions

- ▶ Direct rendering
  - ▶ Ideal solution for CAD
  - ▶ Low CPU overhead
  - ▶ Minimal storage
- ▶ Pixel-accurate
  - ▶ Silhouettes
  - ▶ Interpenetrations
  - ▶ Normals
- ▶ Future Work
  - ▶ Reliable ray-surface intersection test
  - ▶ Trimming acceleration data structure



*"Higher order primitives will complement triangles as the primary rendering primitive."*

# The End.

Thank you for your attention.