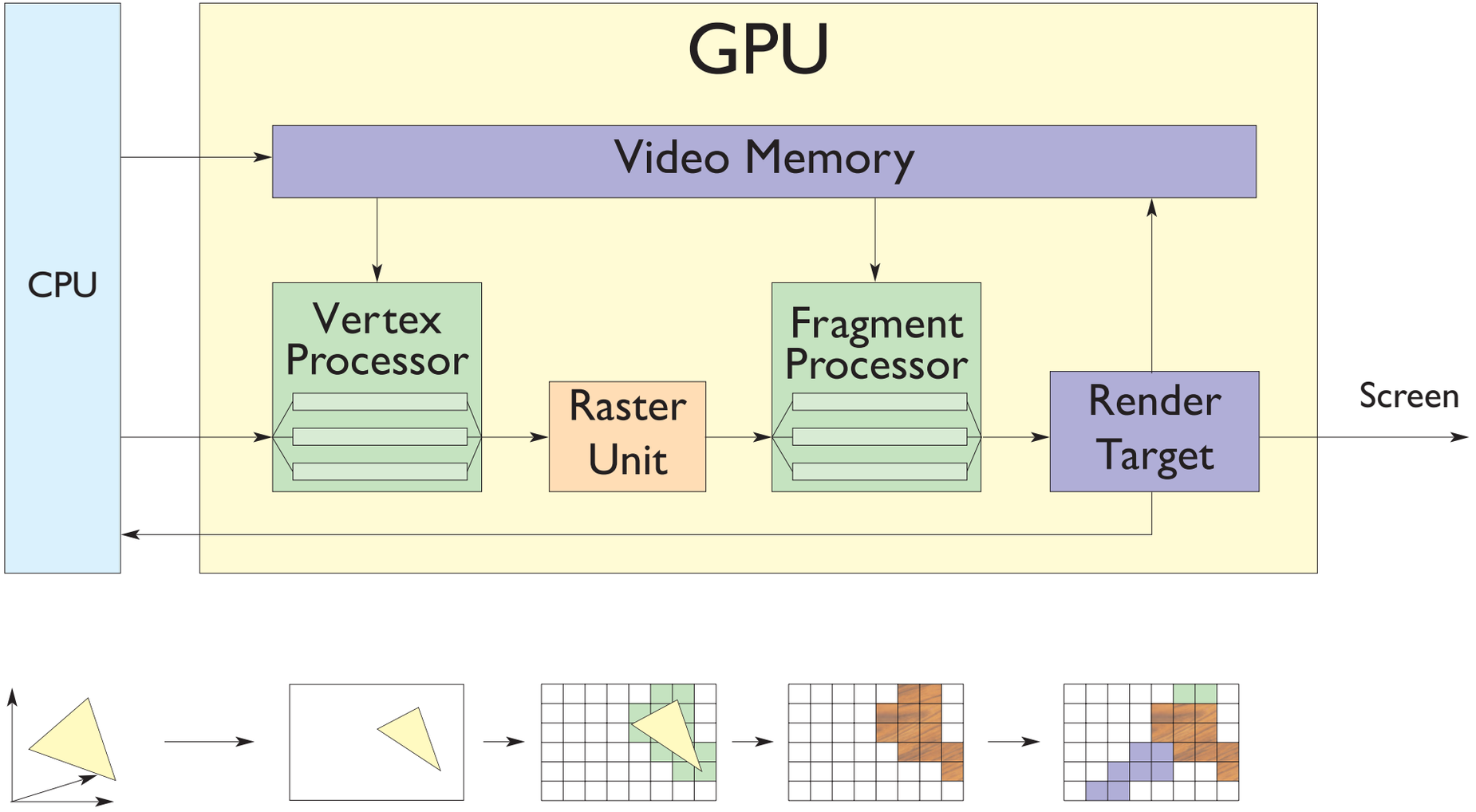
Direct Rendering of Trimmed NURBS Surfaces

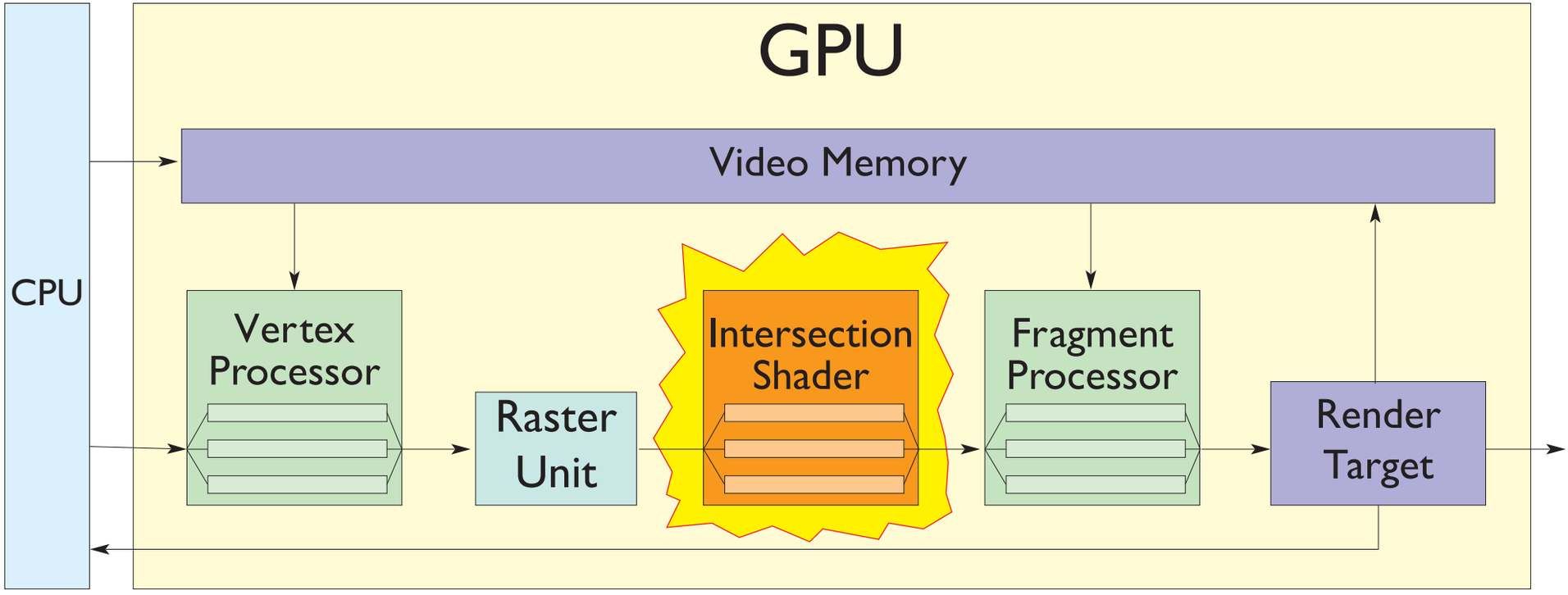
Fakultät Medien
Bauhaus-Universität Weimar

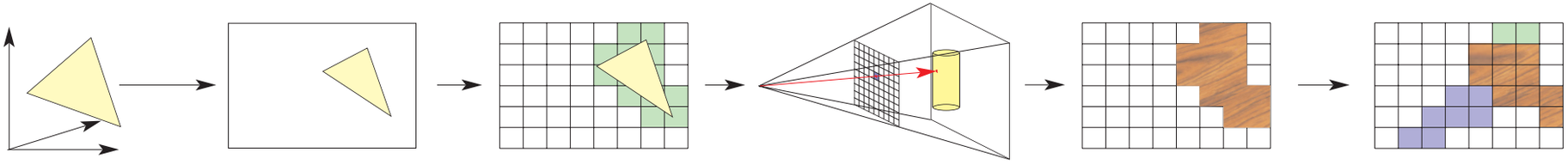
dgp
dynamic graphics project
UNIVERSITY of TORONTO

Hardware Graphics Pipeline



Extended Hardware Graphics Pipeline

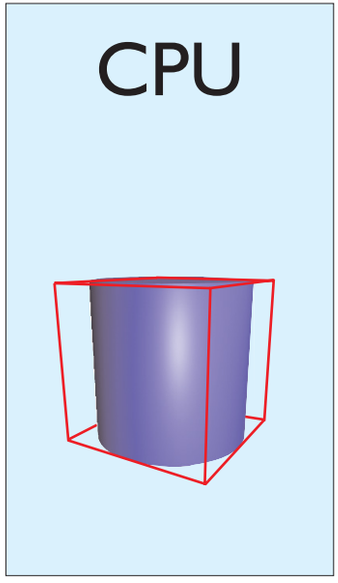




Intersection Shader

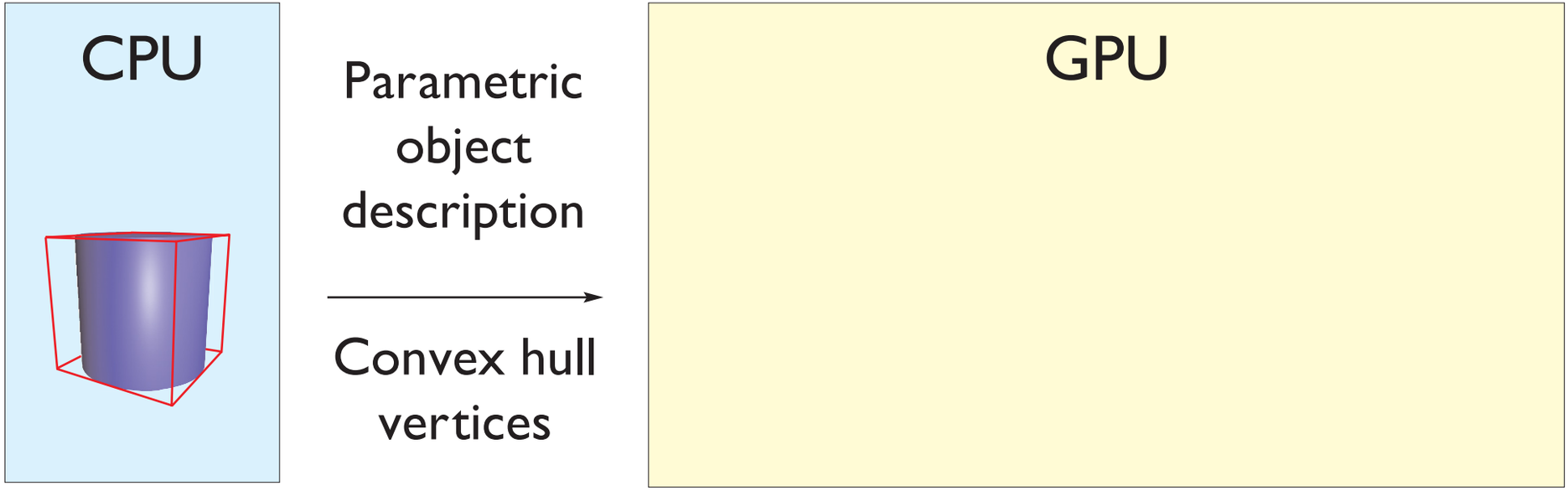
- Conceptual extension of the hardware rendering pipeline
- Optional stage after the rasterization unit for non-triangular geometry
- Specifically for ray object intersection tests
  - Analytically, e.g. for quadrics
  - Numerically, e.g. for NURBS surfaces
- Can perform early ray termination test

Intersection Shader



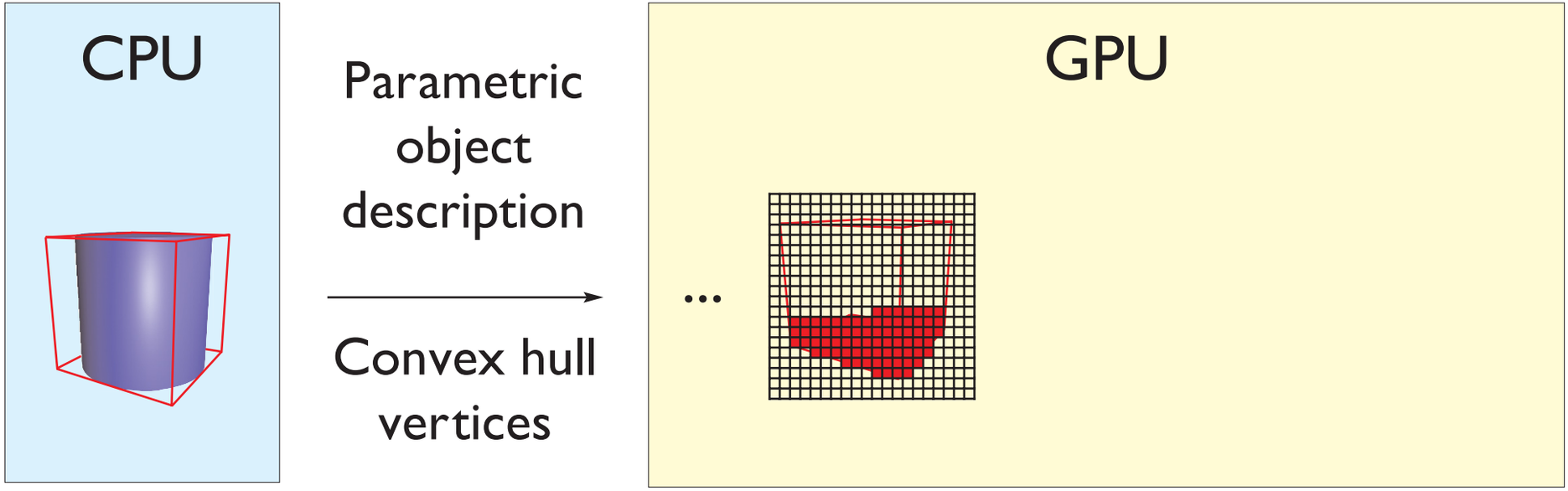
Convex hull

Intersection Shader



Convex hull

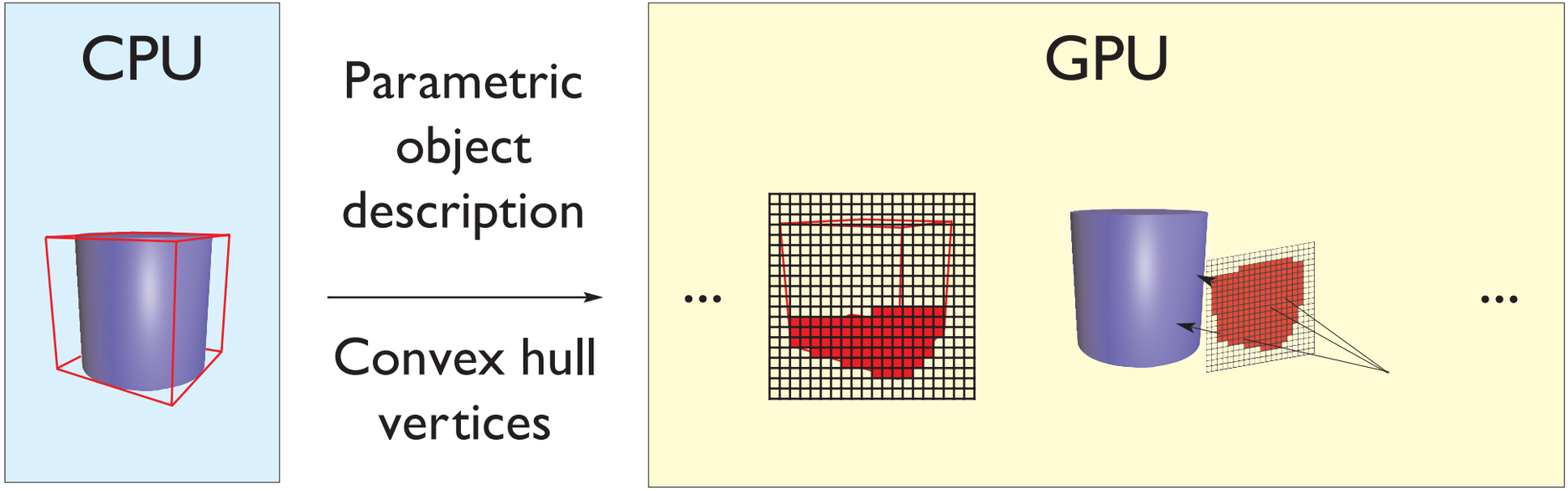
Intersection Shader



Convex hull

Candidate rays, and ray directions are generated

Intersection Shader



Intersection Shader

- Each eye ray corresponds to exactly one fragment
- Current hardware
  - Ray generation is performed in the vertex shader and rasterization unit
  - Intersection shader is part of the fragment shader program

Convex Hulls

- Used to reduce the number of ray object intersection tests
- Rays which are generated by the projection of the convex hull form a candidate set of rays
- “Inverse” acceleration structure
- Tightness of convex hulls can be used to trade CPU vs. GPU computation time

Intersection Shader

Pros	Cons

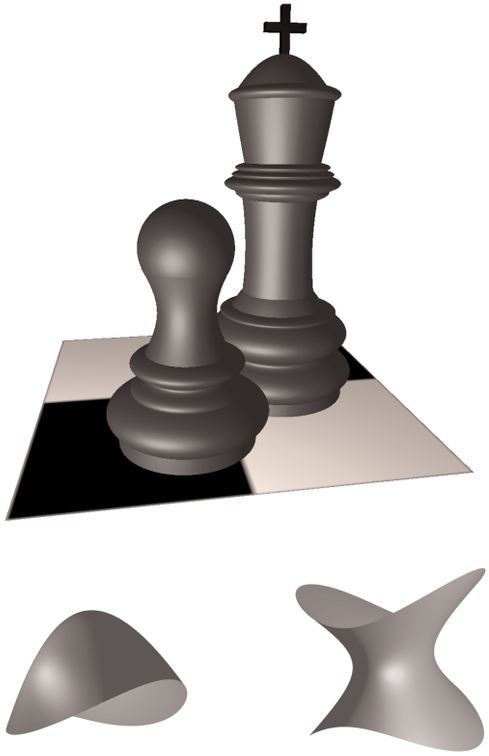
Intersection Shader

Pros	Cons
• No tessellation and minimal storage • Pixel-accurate - silhouettes, - intersection curves, - normals • Seamless integration into standard graphics pipeline	

Intersection Shader

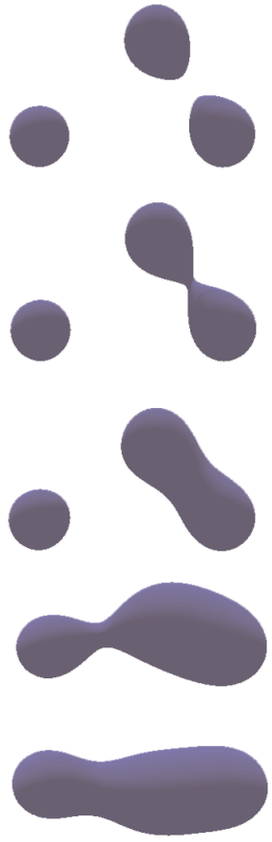
Pros	Cons
• No tessellation and minimal storage • Pixel-accurate - silhouettes, - intersection curves, - normals • Seamless integration into standard graphics pipeline	• Ray casting is expensive • Artifact free intersection test can be difficult to perform

Intersection Shader



Quadrics

NURBS



Node-based
Implicit Surface

Ray Casting of NURBS Surfaces on the GPU

Overview

1. Patch is subdivided in a set of sub-patches
2. A Convex hull is computed for each sub-patch and its parameter range in (u,v) space is determined
3. Candidate rays are generated by rendering the convex hulls
4. Intersection test is performed using Newton Iteration
5. Trimming is performed using Bézier Clipping and Point-In-Polygon test

Preprocessing

1. Generation of convex hulls
   - Conversion of NURBS surfaces into Bézier patches
   - Subdivision of Bézier patches
     - User-provided level, a flatness criteria or a minimum size of a convex hull primitive determines the number of subdivision steps
   - Convex hull consists of a set of polygonal primitives

Preprocessing

2. Conversion of trimming curves into Bézier curves
   - Necessary for Bézier clipping algorithm

Preprocessing

2. Conversion of trimming curves into Bézier curves
   - Necessary for Bézier clipping algorithm
3. Generation of fragment programs
   - Customised for surface properties

Preprocessing

2. Conversion of trimming curves into Bézier curves
   - Necessary for Bézier clipping algorithm
3. Generation of fragment programs
   - Customised for surface properties
4. Parameters are stored in textures
   - Control points and knot vectors of surfaces
   - Control points of trimming curves

Curve and Surface Evaluation

- Bézier curves / surfaces
  - *de Casteljau* algorithm is used for evaluation
  - Efficient because linear interpolation is supported directly in hardware

Curve and Surface Evaluation

- Bézier curves / surfaces
  - *de Casteljau* algorithm is used for evaluation
  - Efficient because linear interpolation is directly supported in hardware
- NURBS curves / surfaces
  - *de Boor* algorithm is used for evaluation
  - Only knot spans which are relevant to evaluate the curve at a certain point are copied into registers

Ray Patch Intersection

- Ray is represented as intersection of two planes

$$P_1: \ \vec{n}_1 \cdot \bar{x} - d_1 = 0$$
$$P_2: \ \vec{n}_2 \cdot \bar{x} - d_2 = 0$$

Ray Patch Intersection

- Ray is represented as intersection of two planes

$$P_1: \ \vec{n}_1 \cdot \bar{x} - d_1 = 0$$
$$P_2: \ \vec{n}_2 \cdot \bar{x} - d_2 = 0$$

- NURBS surface

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_i^p(u) \cdot N_j^q(u) \cdot \bar{p}_{i,j}$$

Ray Patch Intersection

- Ray is represented as intersection of two planes

$$P_1: \ \vec{n}_1 \cdot \bar{x} - d_1 = 0$$
$$P_2: \ \vec{n}_2 \cdot \bar{x} - d_2 = 0$$

- NURBS surface

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_i^p(u) \cdot N_j^q(u) \cdot \bar{p}_{i,j}$$

- Objective function system

$$F(u,v) = \begin{pmatrix} n_1 \cdot S(u,v) - d_1 \\ n_2 \cdot S(u,v) - d_2 \end{pmatrix}$$

Newton Iteration

- Each step takes the form

$$(u,v)_{k+1} = (u,v)_k - J^{-1}F((u,v)_k)$$

where the Jacobian J is given by

$$J = \begin{pmatrix} n_1 \cdot \frac{\partial S(u,v)}{\partial u} & n_1 \cdot \frac{\partial S(u,v)}{\partial v} \\ n_2 \cdot \frac{\partial S(u,v)}{\partial u} & n_2 \cdot \frac{\partial S(u,v)}{\partial v} \end{pmatrix}$$

Newton Iteration

- Newton Iteration terminates if
  - an approximation for a root has been found

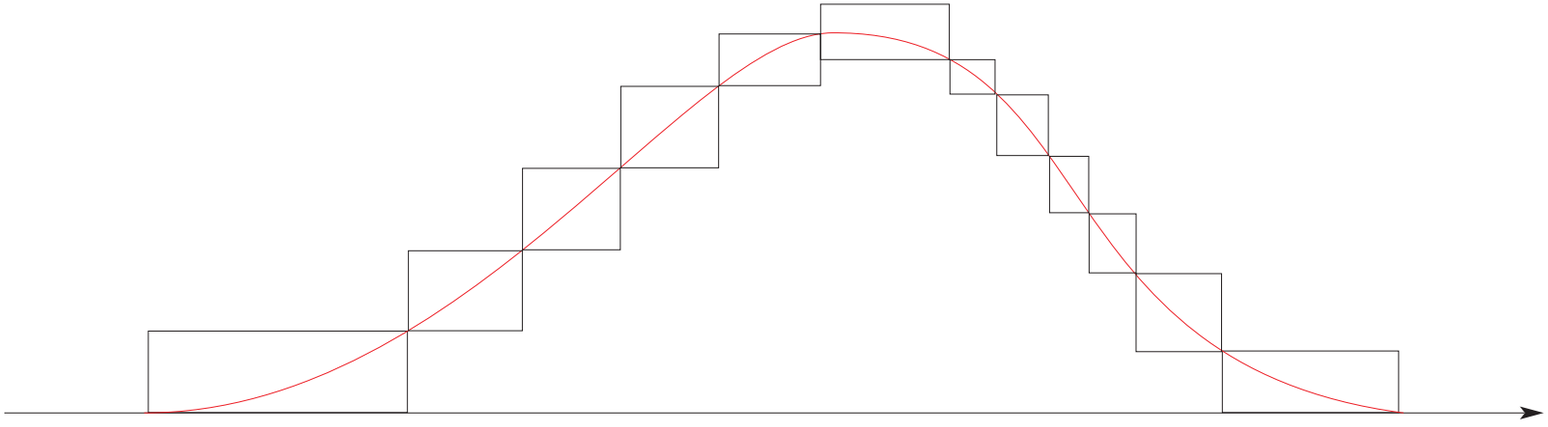
$$\left| F\left((u,v)_k\right)\right| < \varepsilon ,$$

  - the maximum number of iterations has been exceeded,
  - the algorithm diverges, i.e.

$$\left| F\left((u,v)_{k+1}\right)\right| > \left| F\left((u,v)_k\right)\right|$$

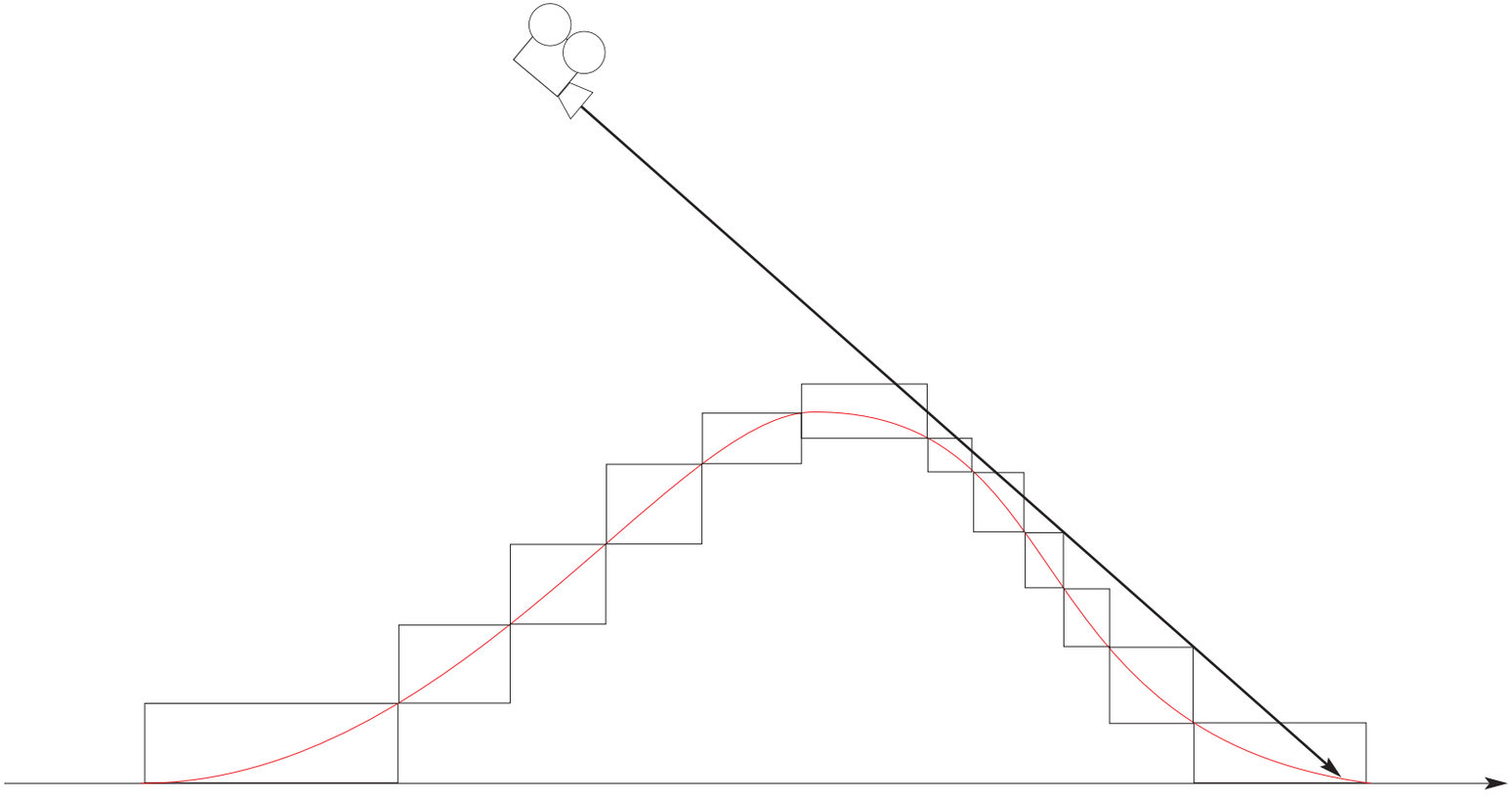
Newton Iteration

- Newton Iteration has to be implemented in one rendering pass



Newton Iteration

- Newton Iteration has to be implemented in one rendering pass



Initial Values

1. Fixed initial guess for each primitive of the convex hull
   - Center of (u,v) parameter space enclosed by primitive of the convex hull

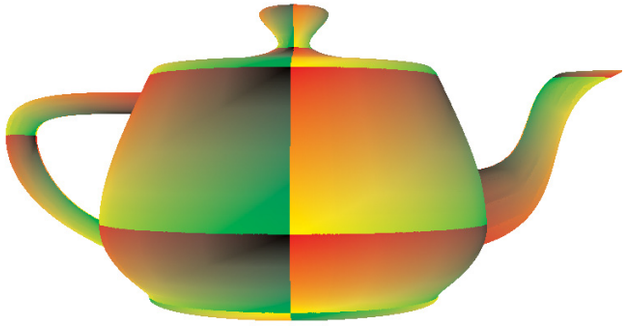
Initial Values

1. Fixed initial guess for each primitive of the convex hull
   - Center of (u,v) parameter space enclosed by primitive of the convex hull
2. (u,v) Texturing
   - (u,v) parameter stored for each vertex during preprocessing
   - Parameters are linearly interpolated along the surfaces of the convex hull

Initial Values

2.1. View-independent (u,v) Texturing

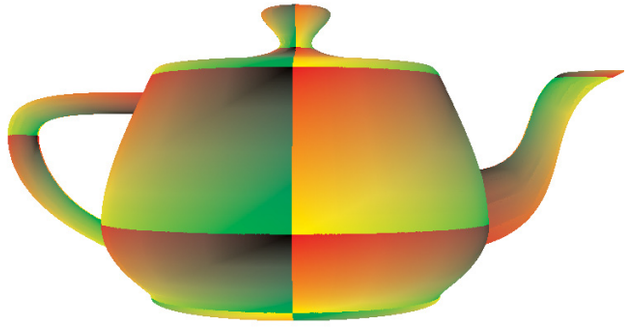
- The (u,v) parameters as computed during the preprocessing are interpolated



Initial Values

2.1. View-independent (u,v) Texturing

- The (u,v) parameters as computed during the preprocessing are interpolated



2.2. View-dependent (u,v) Texturing

- Vertex shader: Intersection test between the ray from the camera to the vertex and the surface patch
- (u,v) parameters of the intersection points are interpolated

Trimming

- Based on point-in-polygon test
- A ray is shot along the parametric surface and the number of intersections with the trimming curves is counted
  - The parity of the number of intersections determines if the fragment is inside or outside the trimming region
- Bézier clipping is used to determine the intersection of the ray with the trimming curves

Bézier Clipping

- Mixture between subdivision-based and numerical intersection algorithm
- Can be used for ray Bézier curve and ray Bézier patch intersection test
- Guaranteed to find all intersection points
  - Some special cases for ray surface patch intersection

Bézier Clipping

- Ray is given in implicit form

$$ax + by - c = 0$$

Bézier Clipping

- Ray is given in implicit form

$$ax + by - c = 0$$

- Bézier curve is given by

$$C(u) = \sum_{i=0}^{n} \bar{p}_i B_i^{(n)}(u), \qquad \bar{p}_i = (x_i, y_i)$$

Bézier Clipping

- Ray is given in implicit form

$$ax + by - c = 0$$

- Bézier curve is given by

$$C(u) = \sum_{i=0}^{n} \overline{p}_i B_i^{(n)}(u), \qquad \overline{p}_i = (x_i, y_i)$$

- Objective function (distance field)

$$d(u) = \sum_{i=0}^{n} d_i B_i^{(n)}(u), \qquad d_i = ax_i + by_i - c$$

Bézier Clipping

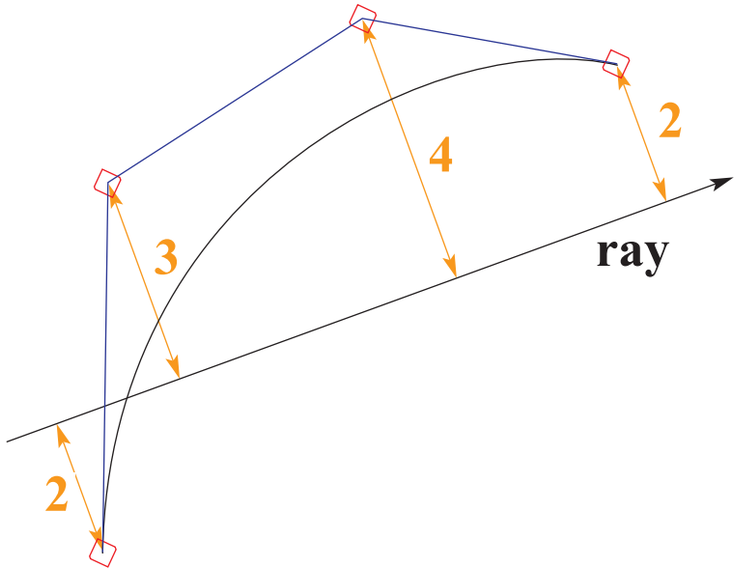
- Objective function as explicit, non-parametric Bézier curve

$$D(u) = (u, d(u)) = \sum_{i=0}^{n} D_i B_i^{(n)}(u), \qquad D_i = (u_i, d_i)$$

- Because $D(u)$ is a Bézier curve, the control points are evenly spaced, i.e.

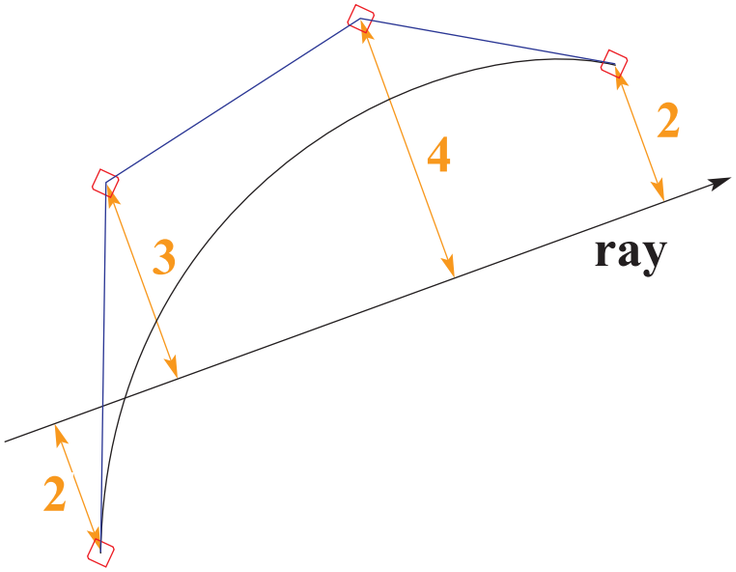
$$u_i = \frac{i}{n}$$

Bézier Clipping

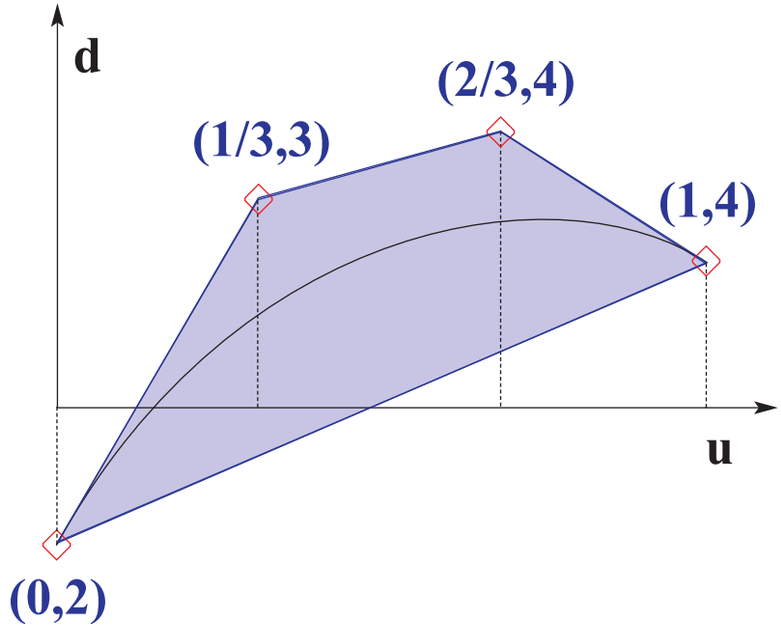


Bézier curve

Bézier Clipping

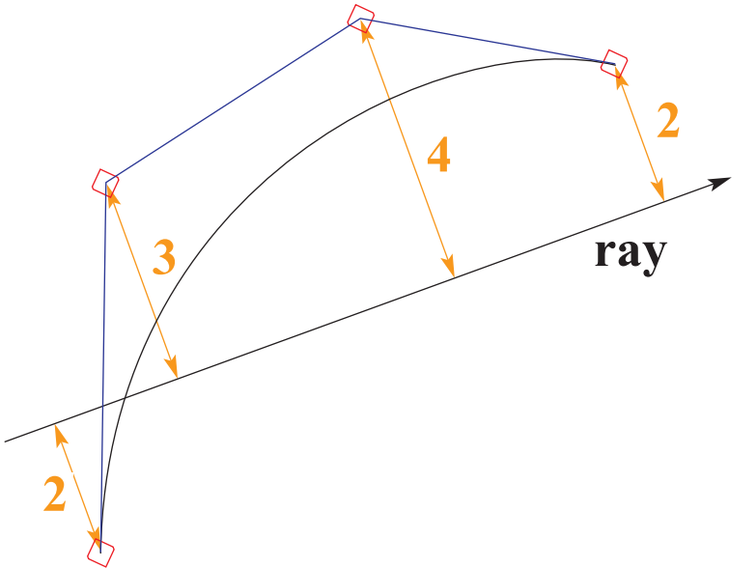


Bézier curve

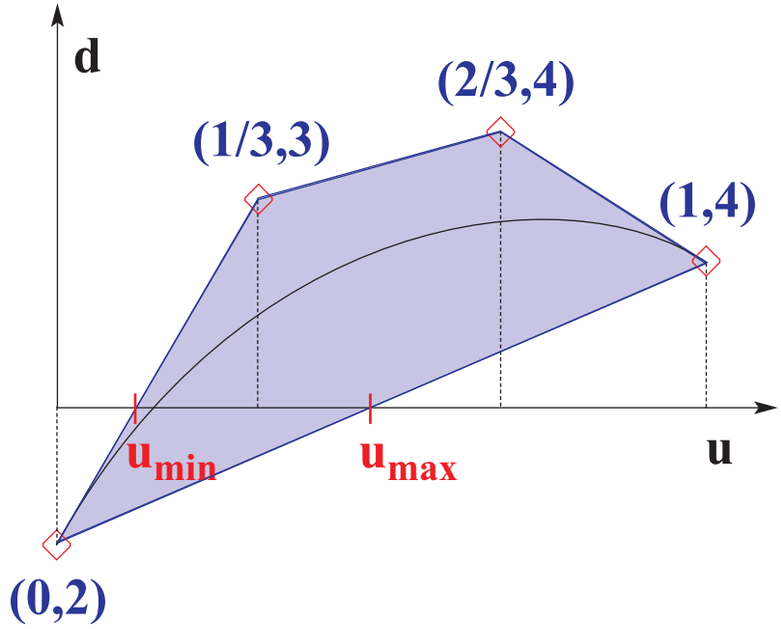


Explicit Bézier curve

Bézier Clipping



Bézier curve



Explicit Bézier curve

Bézier Clipping

- If the convex hull DOES NOT intersect the $d=0$ axis then no intersection point exists
- If the convex hull DOES intersect the $d=0$ axis then the convex hull property for Bézier curve guarantees that an intersection lies between u_{min} and u_{max}
- The de Casteljau algorithm can be used to subdivide the curve into three segments
  - Two of the segments, $[0, u_{min})$ and $(u_{max}, 1]$, do not contain a ray curve intersection

Bézier Clipping

- If the $[u_{min}, u_{max}]$ is bigger than some predefined threshold then the interval contains two intersections and is itself subdivided
  - The search is continued in each subinterval separately
- The interval contraction is applied until the interval $[u_{min}, u_{max}]_k$ is smaller than a predefined threshold
  - The intersection point is the center of the final interval $[u_{min}, u_{max}]_k$

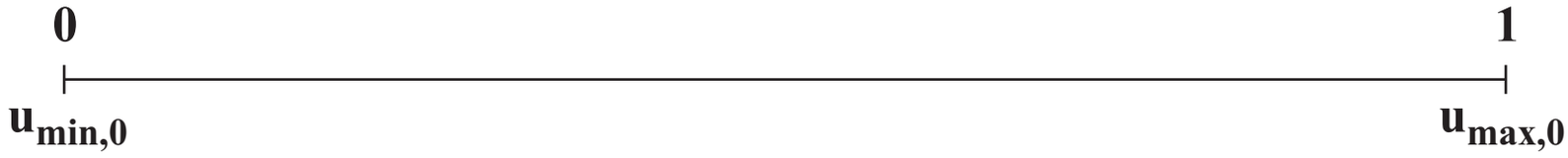
Bézier Clipping on the GPU

- Algorithm has been reformulated to be iterative
  - For efficiency reasons implemented as a state machine on the GPU
- The convex hull is reconstructed only locally
  - Only the edges which contain u_{min} and u_{max} are found and the corresponding control points are stored
  - Reduces the number of registers necessary

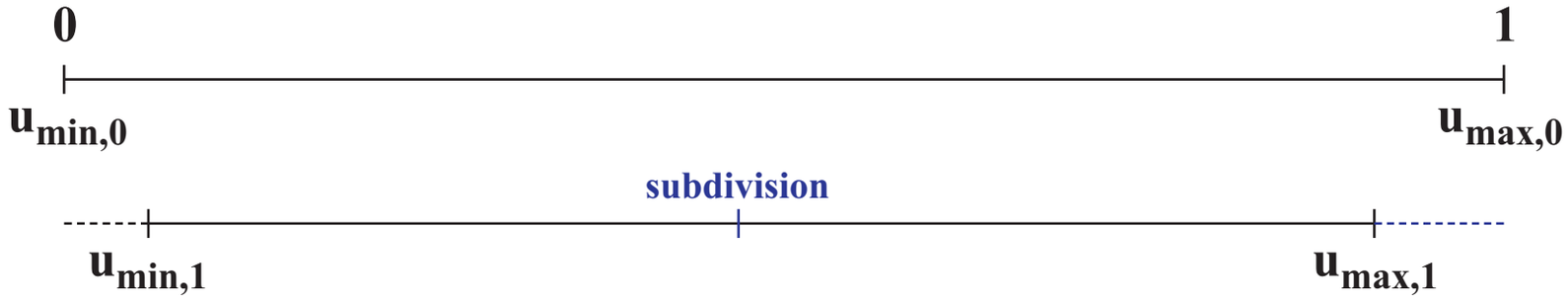
Bézier Clipping on the GPU

- If a subdivision of $[u_{min}, u_{max}]_k$ is necessary, the search is continued only in the left subinterval
  - The limited number of register on the GPU does not allow to store different right subintervals
  - A "global" right subinterval is used which requires only one scalar for its representation
- After the search terminated in the left subinterval it is continued in the "global" right subinterval

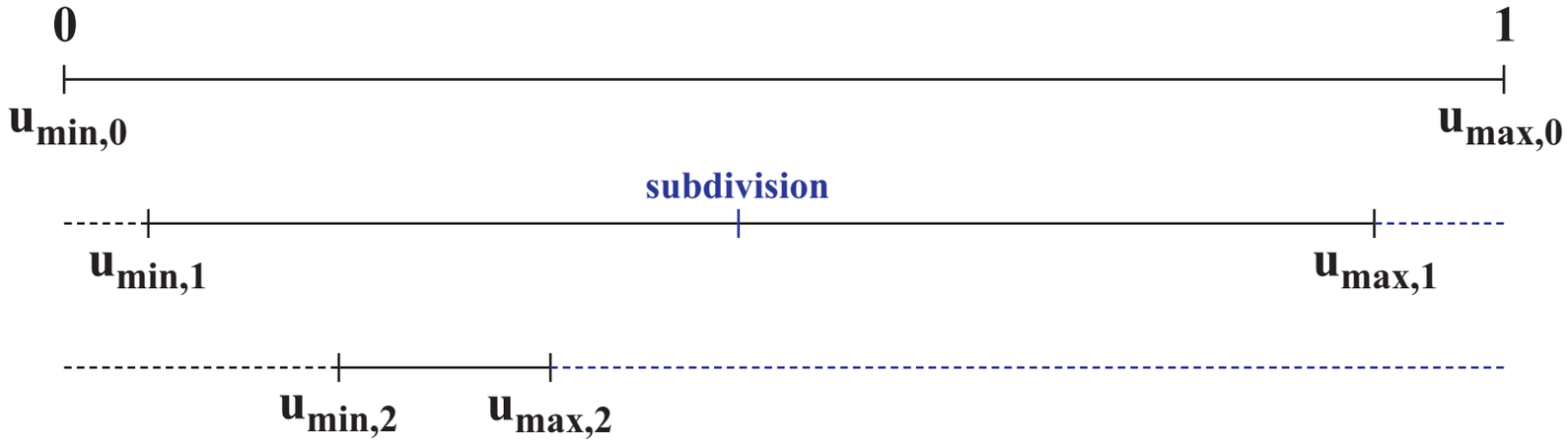
Bézier Clipping on the GPU



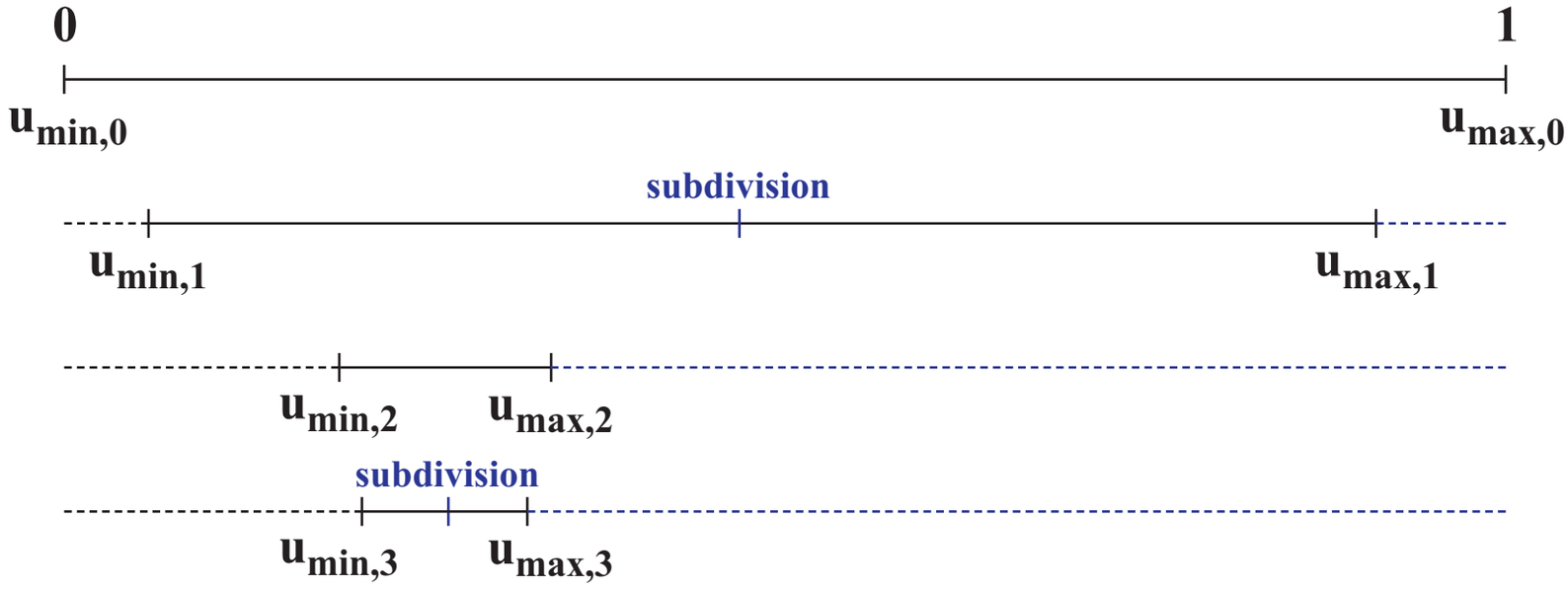
Bézier Clipping on the GPU



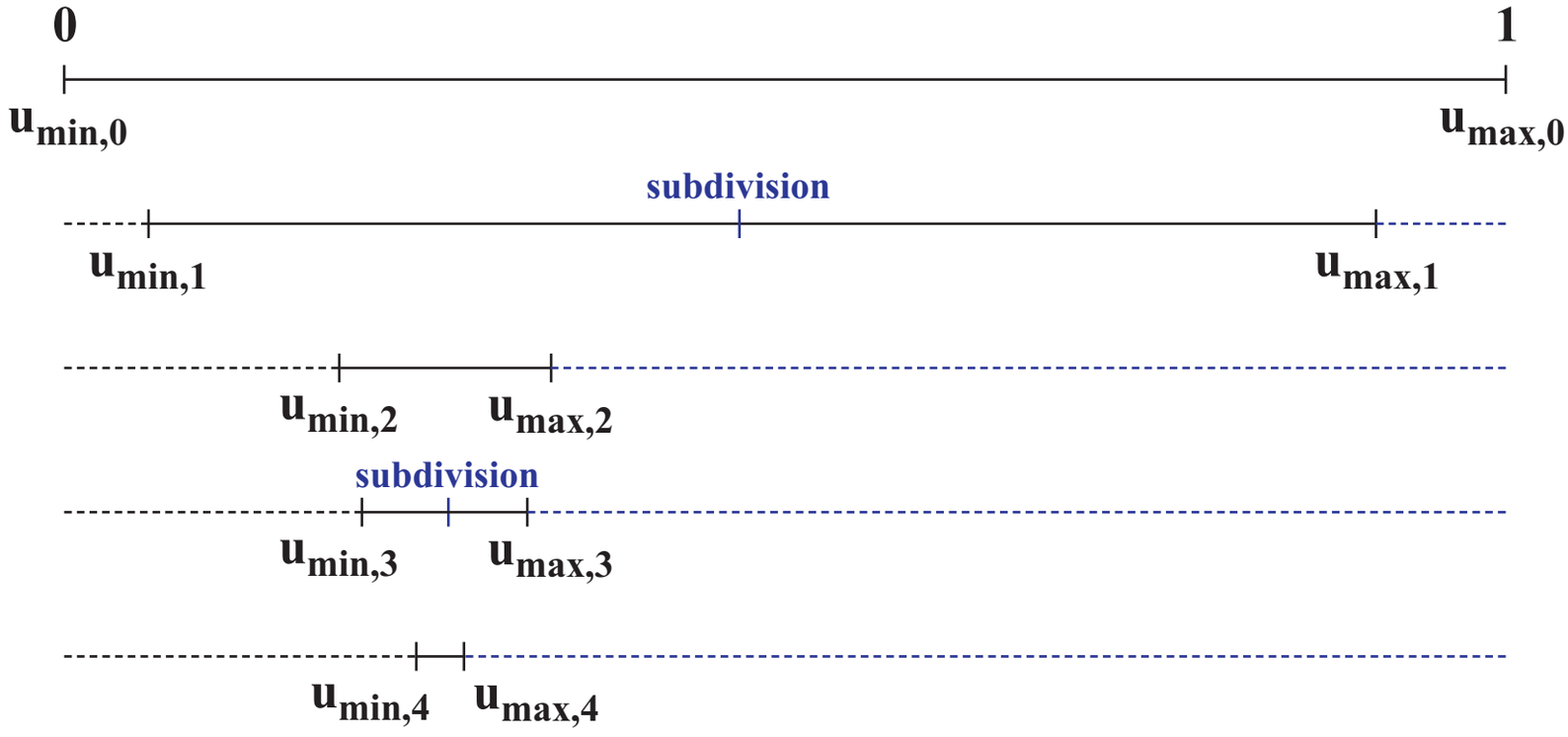
Bézier Clipping on the GPU



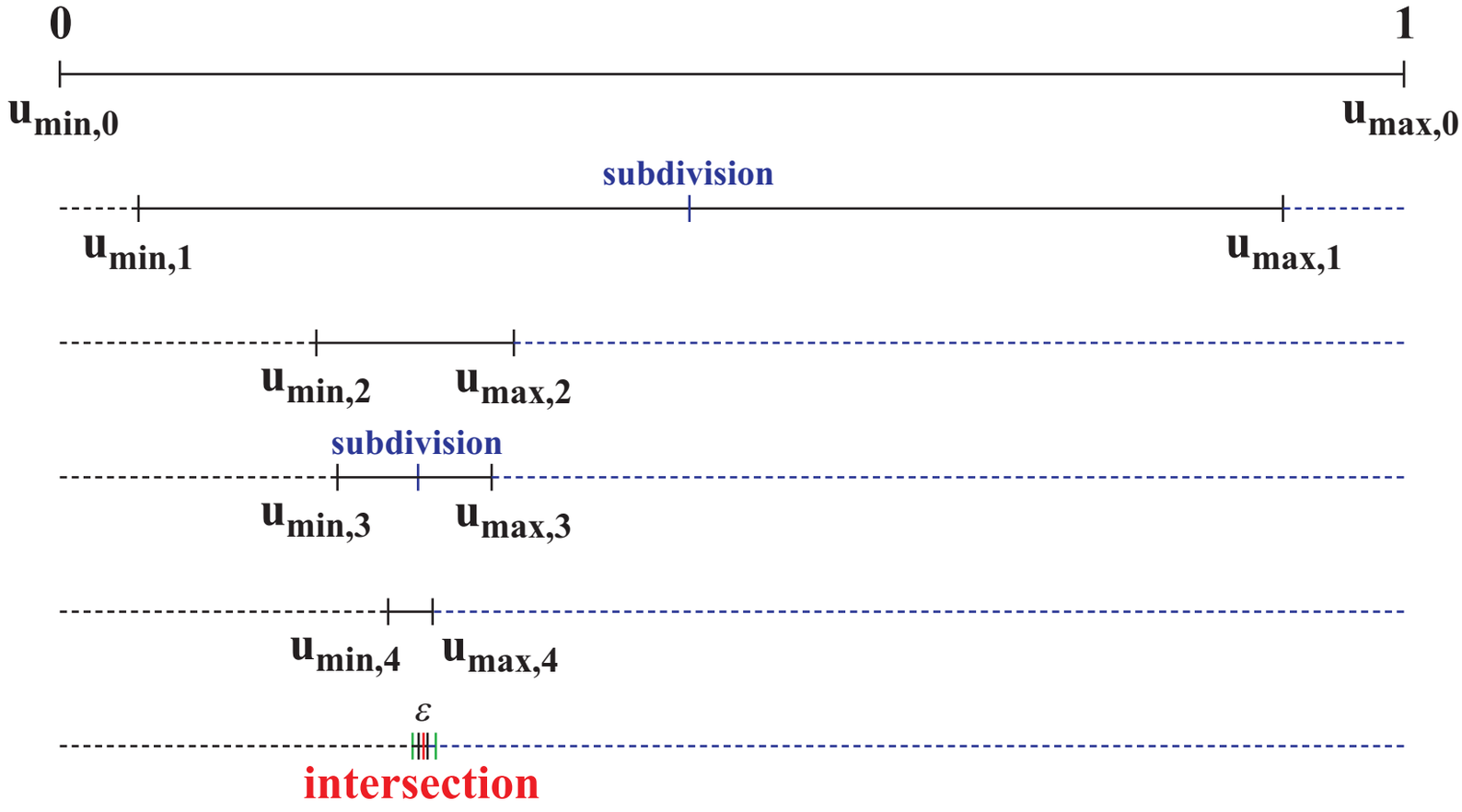
Bézier Clipping on the GPU



Bézier Clipping on the GPU



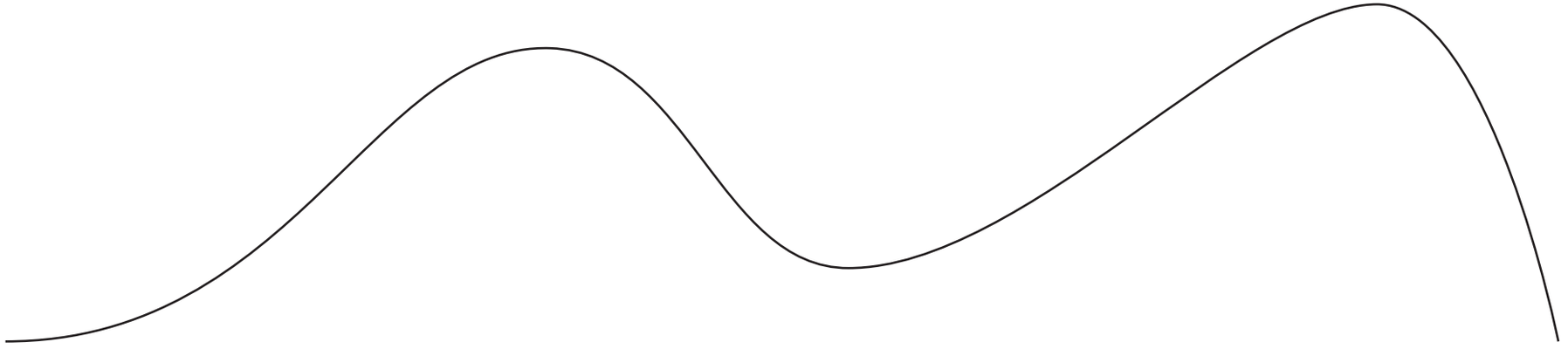
Bézier Clipping on the GPU



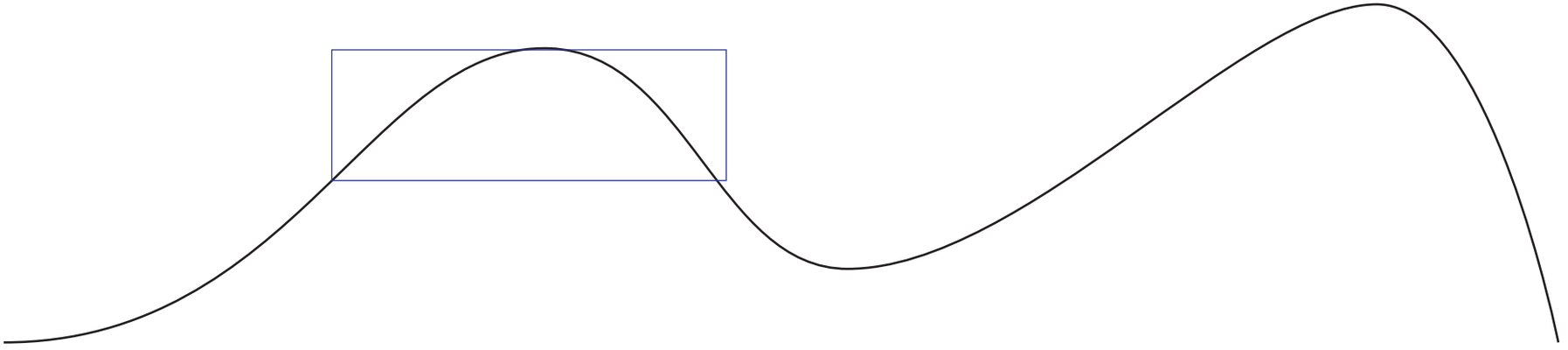
Early Ray Termination

- Reduces the number of intersection tests
- Intersection shader program is not executed for a ray if it can be guaranteed that any intersection which would be found is behind the one that has already been encountered for the fragment
- Convex hull property for parametric patches enables early ray termination based on minimal depth value of bounding primitive for sub-patch

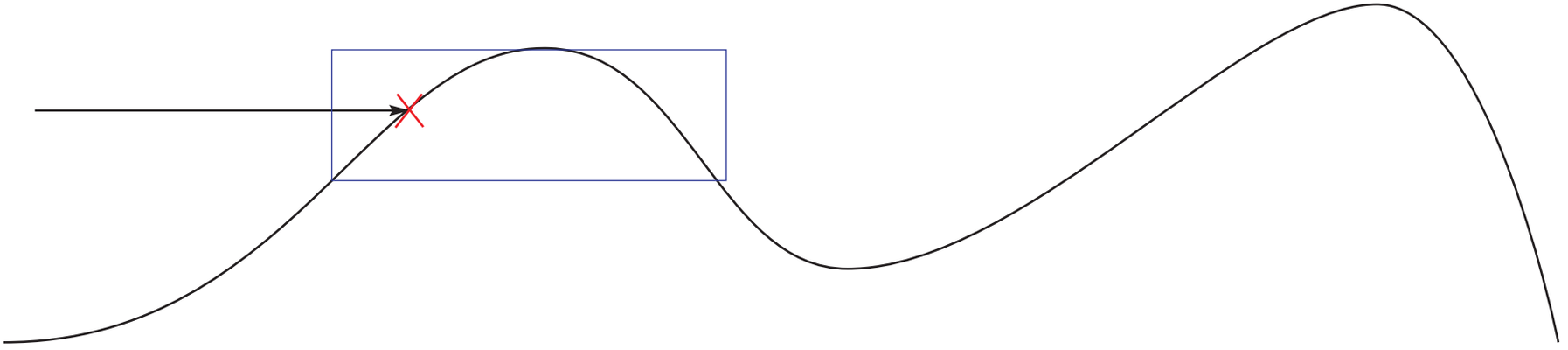
Early Ray Termination



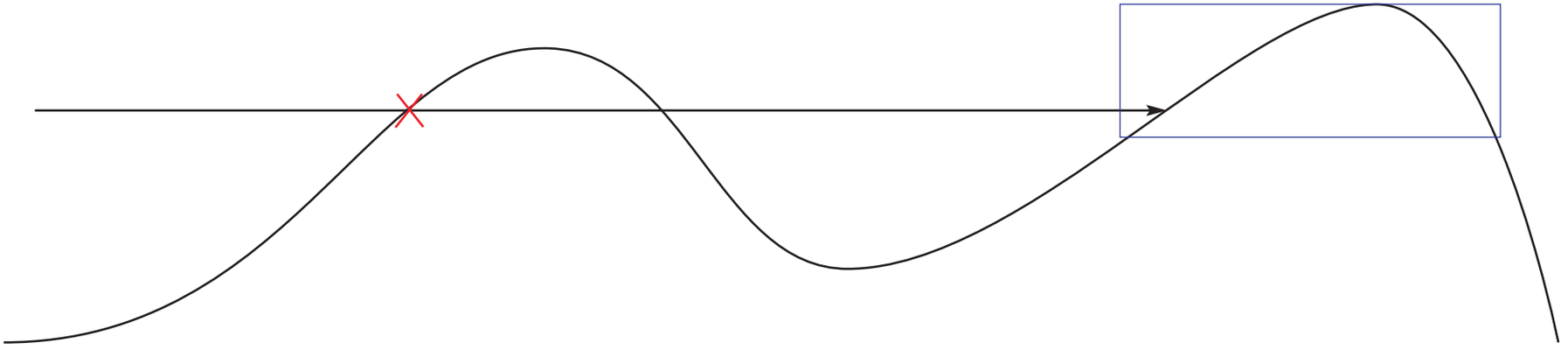
Early Ray Termination



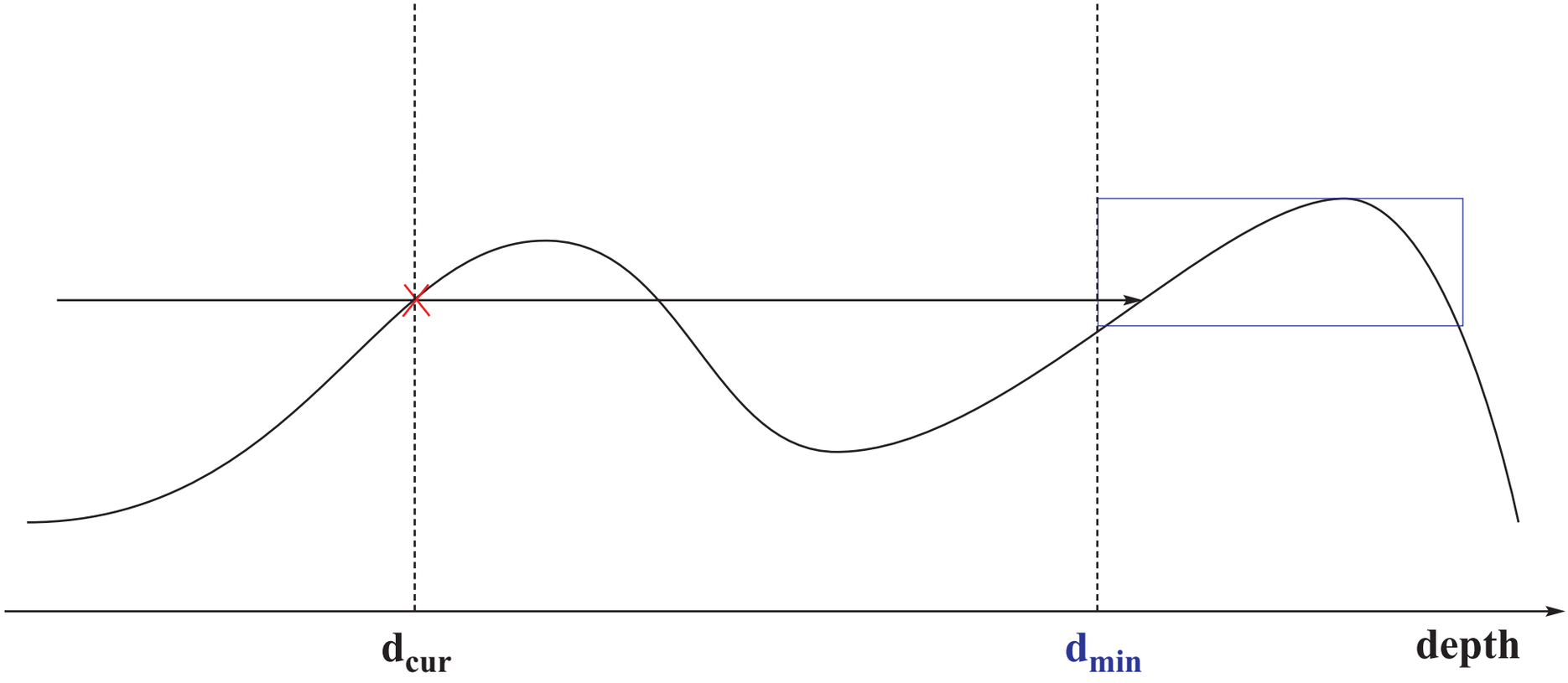
Early Ray Termination



Early Ray Termination



Early Ray Termination



Early Ray Termination

- GPU provides Early-Z
  - BUT cannot be used if depth value is changed in the fragment shader program
  - Depth value has to be written for consistent depth buffering

Early Ray Termination

- GPU provides Early-Z
  - BUT cannot be used if depth value is changed in the fragment shader program
  - Depth value has to be written for consistent depth buffering
- Manual Early-Z can be implemented by branching in the fragment shader
  - Current depth value from depth buffer is compared to minimal depth value of convex hull

Early Ray Termination

- OpenGL: Simultaneous read and write of buffer results in undefined behaviour
  - No memory-cache synchronisation
  - Parallel processing of rays / fragments
- In practice reading from the currently bound depth buffer is possible (Nvidia boards)
  - Depth buffer is accessed as texture, similar to shadow mapping

Early Ray Termination

- Reading from depth buffer is *conservative*
  - Depth buffer values are monotonically decreasing within one render pass
  - Values obtained in the intersection shader are always equal or greater than the correct value
  - Depth buffering is performed after fragment shader
- Some "unnecessary" intersection tests are performed but images are artifact free

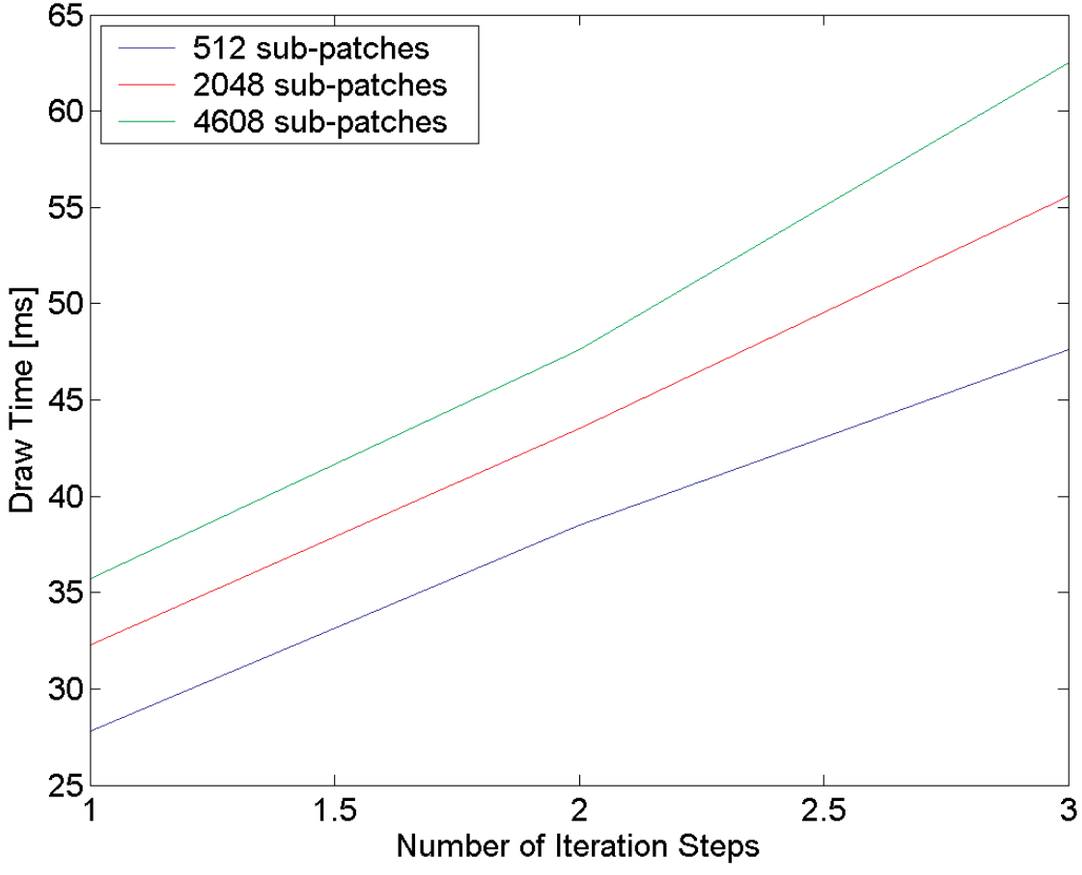
Early Ray Termination

- Reading from depth buffer is *conservative*
  - Depth buffer values are monotonically decreasing within one render pass
  - Values obtained in the intersection shader are always equal or greater than the correct value
  - Depth buffering is performed after fragment shader
- Some "unnecessary" intersection tests are performed but images are artifact free
- Requires front-to-back sorting of convex hull primitives

Results

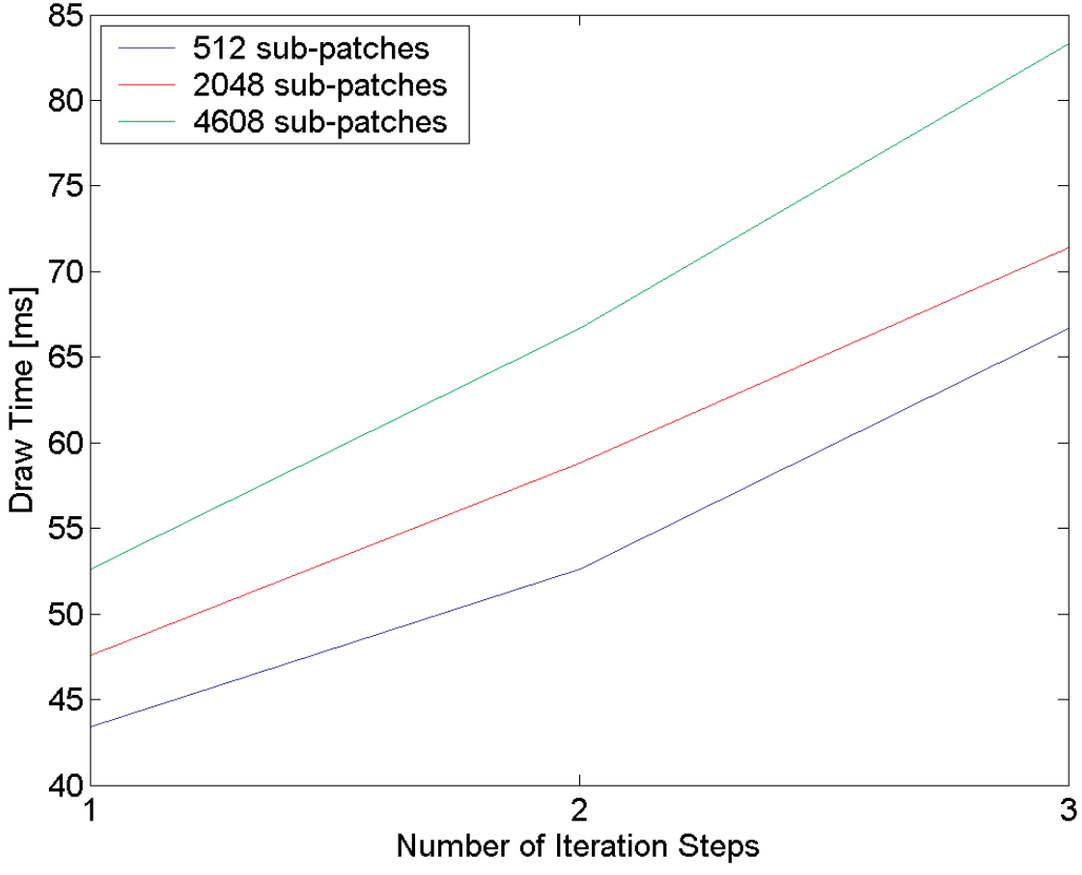
- All tests performed at 1280x1024
- 80% of screen covered by objects
- AMD Athlon 64 3000+
- Single Nvidia Quadro FX 4500
  - SLI usually nearly doubles the performance
- Results are draw times in ms

- No trimming curve
- Different number of sub-patches



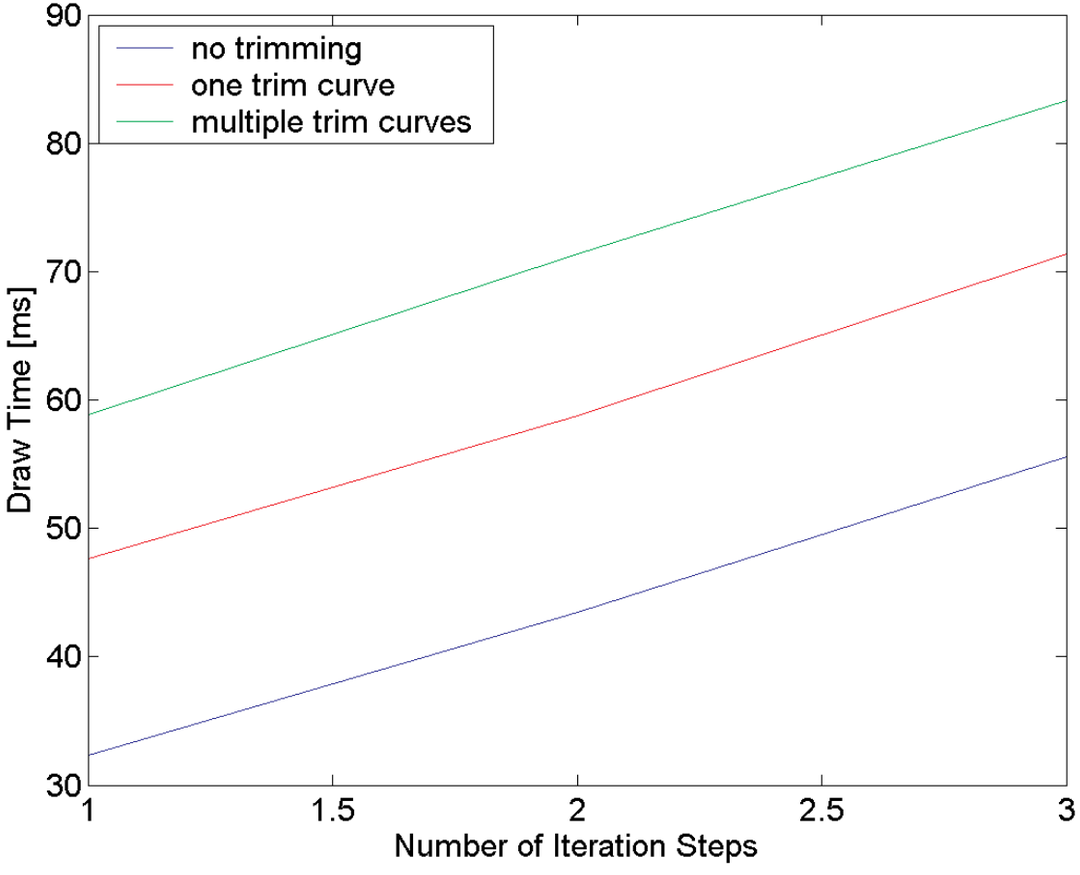
Patches	Sub-Patches	Number of Iterations		
		1	2	3
32	512	27.8	38.5	47.6
32	1024	32.3	43.5	55.6
32	4608	35.7	47.6	62.5

- One trimming curve
- Different number of sub-patches

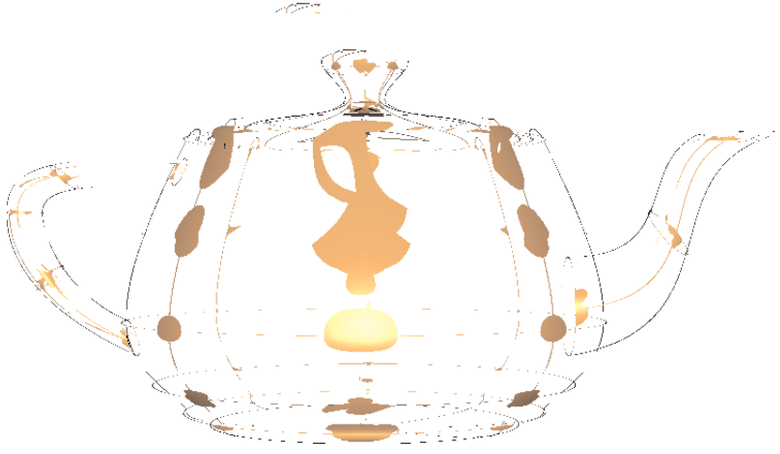


Patches	Sub-Patches	Number of Iterations		
		1	2	3
32	512	43.4	52.6	66.7
32	1024	47.6	58.8	71.4
32	4608	52.6	66.7	83.3

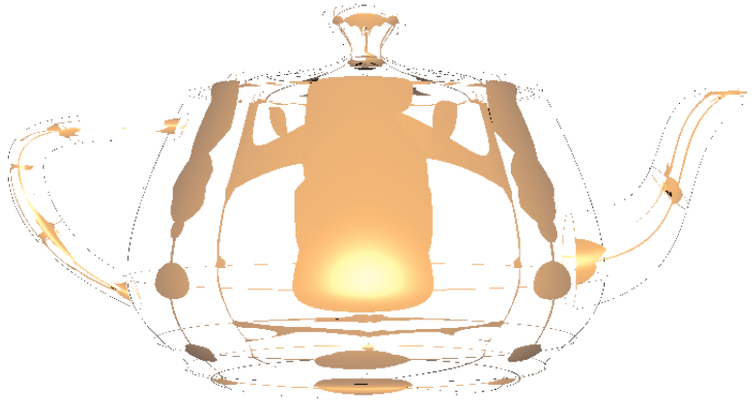
- Different number of trimming curves
- 2048 sub-patches



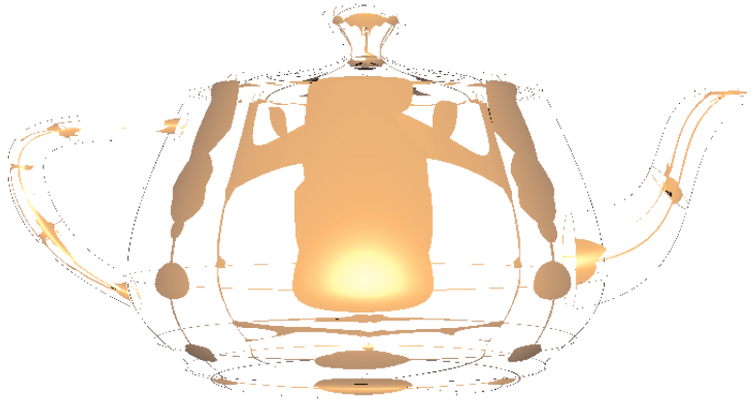
- Newton Iteration
  - 1, 2, 3 and 4 Newton Iteration steps (ltr, ttb)

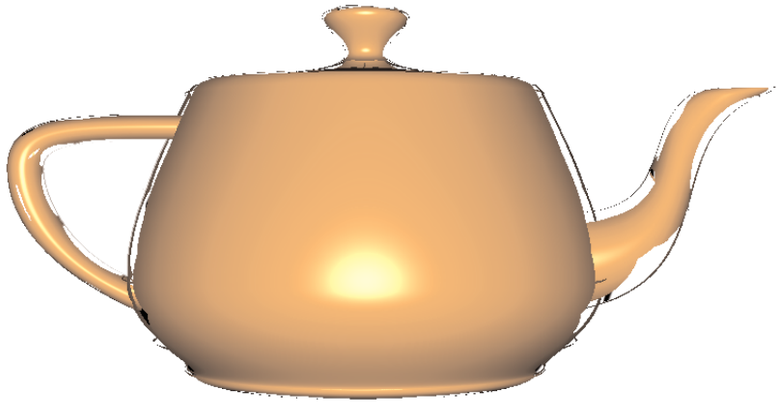


- Newton Iteration
  - 1, 2, 3 and 4 Newton Iteration steps (ltr, ttb)



- Newton Iteration
  - 1, 2, 3 and 4 Newton Iteration steps (ltr, ttb)

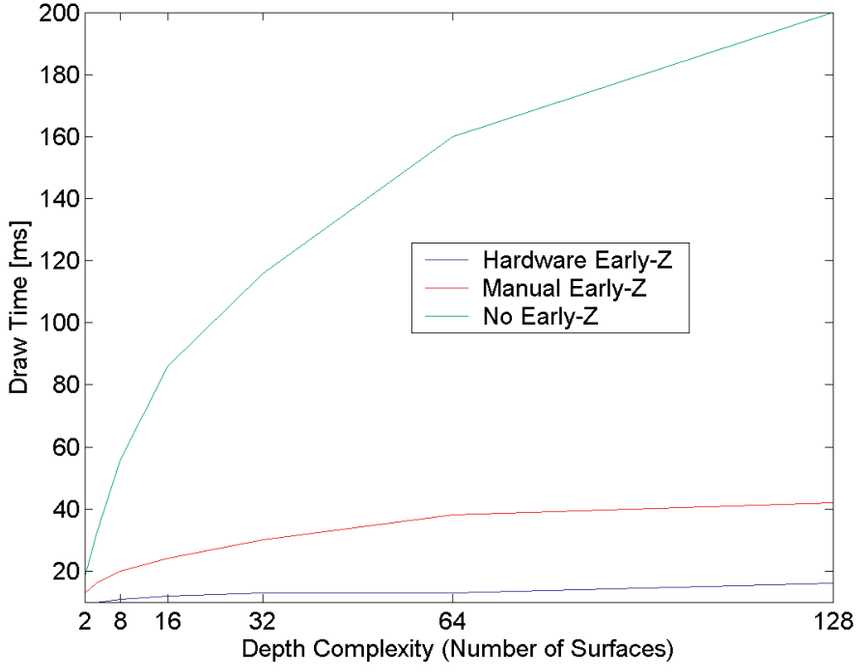




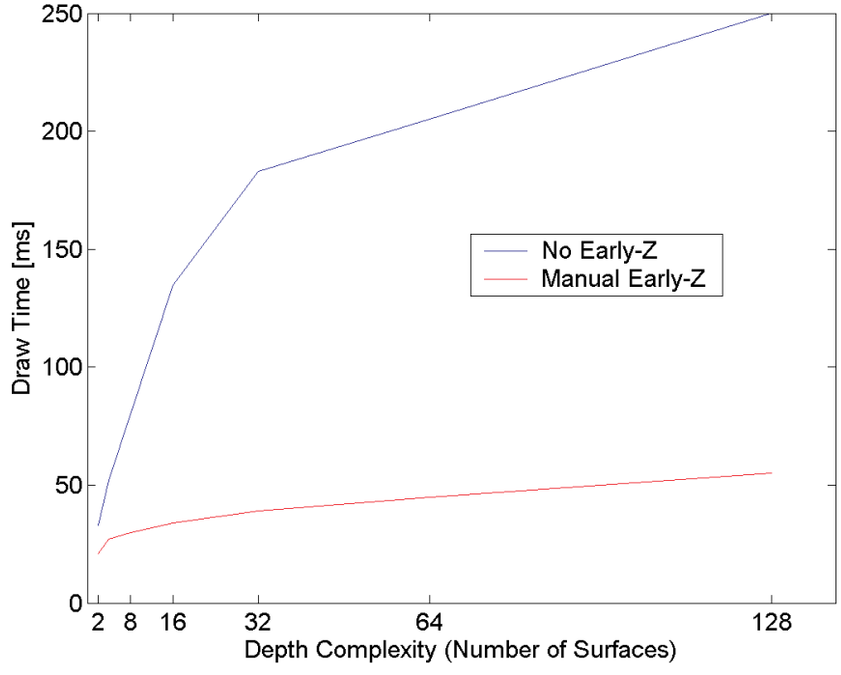
- Newton Iteration
  - 1, 2, 3 and 4 Newton Iteration steps (ltr, ttb)

- Early ray termination

Quads



NURBS



Comparison with Guthe et al. [2005]

	Guthe et al.	GPUCast
Integration into hardware graphics pipeline	Seamless	Seamless
Pixel error	Estimator for screen space error	Pixel-accurate (some artifacts through non-convergence)
Trimming	Texture-based, often high resolutions necessary	Pixel-accurate

Comparison with Guthe et al. [2005]

	Guthe et al.	GPUCast
Number of trims	Unlimited	Currently max. 10
Multi-GPU setups	Does not scale	Scales well
Quality at surface intersections	Appropriate tessellation factor hard to determine	Pixel accurate without additional computations
Possible model complexity	Complex models possible but real-time not guaranteed in these cases	Limited to medium complex models on current hardware

Limitations

- At the very limit of GPU resources
  - Especially registers
  - Adding functionality or improving the current one is (very) hard
  - Many GLSL compiler bugs
    - Got better for NV4X over the years
    - Alternative SH was used

Limitations

- Usually one fragment program per surface
- Limited degree of NURBS surfaces (max. degree 7)
- Images are not artifact free
  - Newton Iteration does not always converge
- No acceleration structure for trimming
  - Currently for each fragment all trimming curves have to be tested
  - About 10 trimming curves are supported efficiently
- Manual Early-Z is not officially supported

Future Work

- Requires DirectX10 graphics hardware
- Reliable intersection tests
  - Bezier Clipping
  - Subdivision-based methods
  - Resultant-based methods
- Acceleration structure for trimming
  - First experiments with a quad tree are promising
- NURBS with arbitrary degree

Conclusions

- Pixel-accurate rendering of NURBS surfaces without tessellation
- Real-time performance for medium size scenes
- Number of trimming curves limited
- Some image artifacts because of non-converging intersection tests
- Next generation of GPUs provides processing power and feature to overcome current limitations

DEMO

Slides available at

www.dgp.toronto.edu/~lessig/nurbs