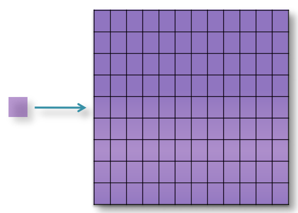
Parallel GPU Algorithms for Interactive CAD

Sara McMains
Adarsh Krishnamurthy

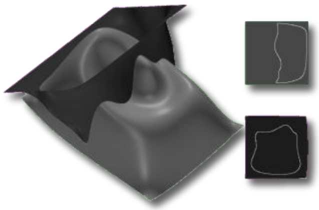
UC Berkeley
Berkeley, CA, USA

Outline



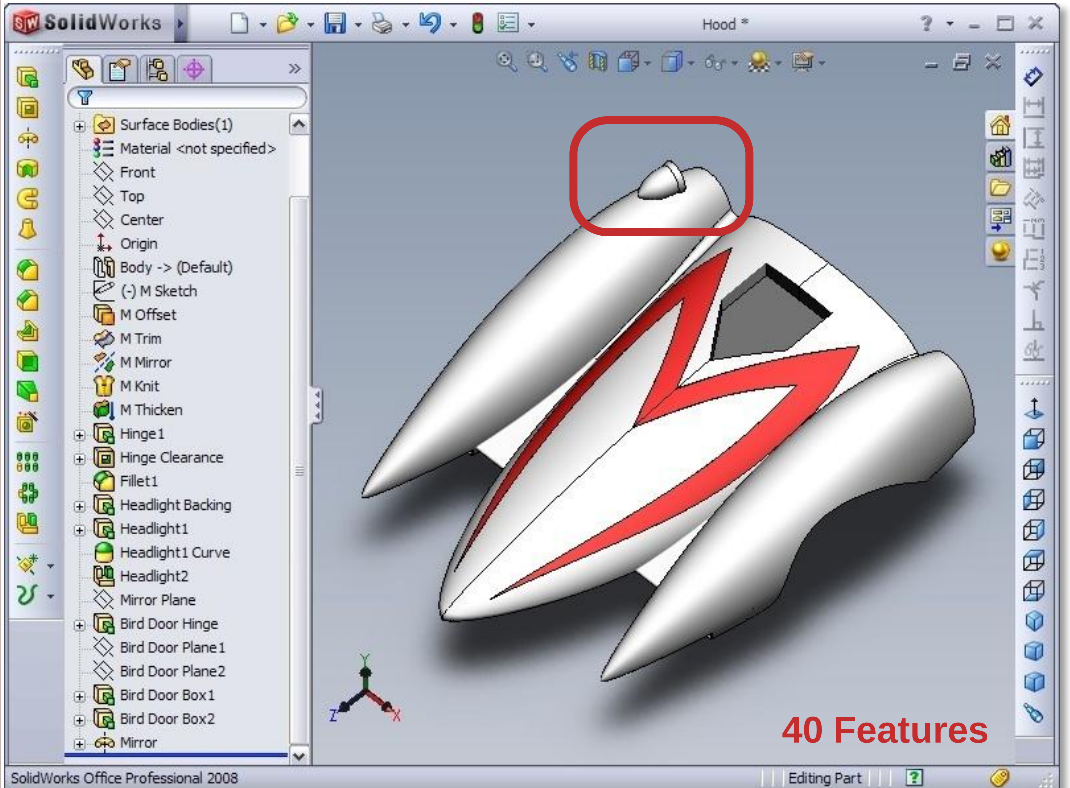
Motivation & Background

NURBS Evaluation



Surface Intersection

Limitations of Existing CAD Systems



Existing CAD systems are slow

Example – model rebuilds
- Surface evaluations
- Boolean CSG operations

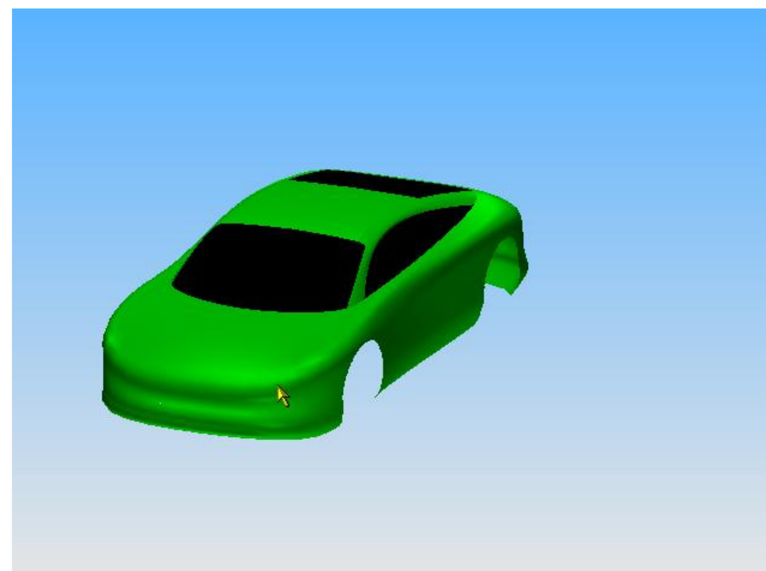
Low interactivity

Require fast algorithms for modeling operations

Limitations of Existing CAD Systems

Trimming on existing commercial CAD software

Lack functionality
Direct modeling operations ▪ Many steps required for trimming, sketching, etc. ▪ Lack immediate visual feedback
Cannot provide interactive feedback



Require algorithms that enhance functionality

GPU-Algorithm Development

Challenges

GPU/CPU operations
- Distribution of work
- Some operations inherently serial

GPU restrictions
- Dynamic loops
- Memory reads and writes
- Single precision

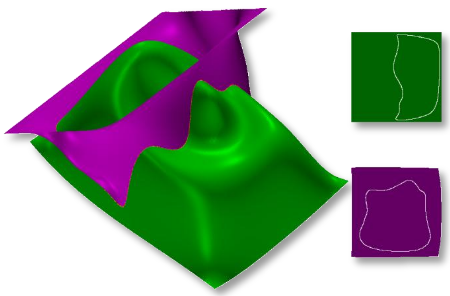
GPU performance guidelines
- Coherent memory reads
- Branchless kernels
- Reduced data read-back from GPU

Multiple GPU vendors
- Implementation: not vendor-specific
- Algorithms: any massively parallel architecture
  - Many-core CPUs

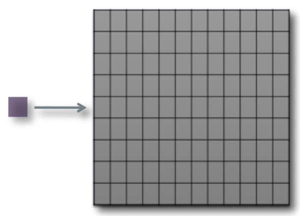
Strategies

Separation of CPU/GPU operations
- NURBS evaluations

Imposing artificial structure to the computations
- Surface-surface intersections

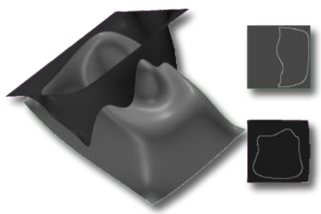


Outline



Motivation & Background

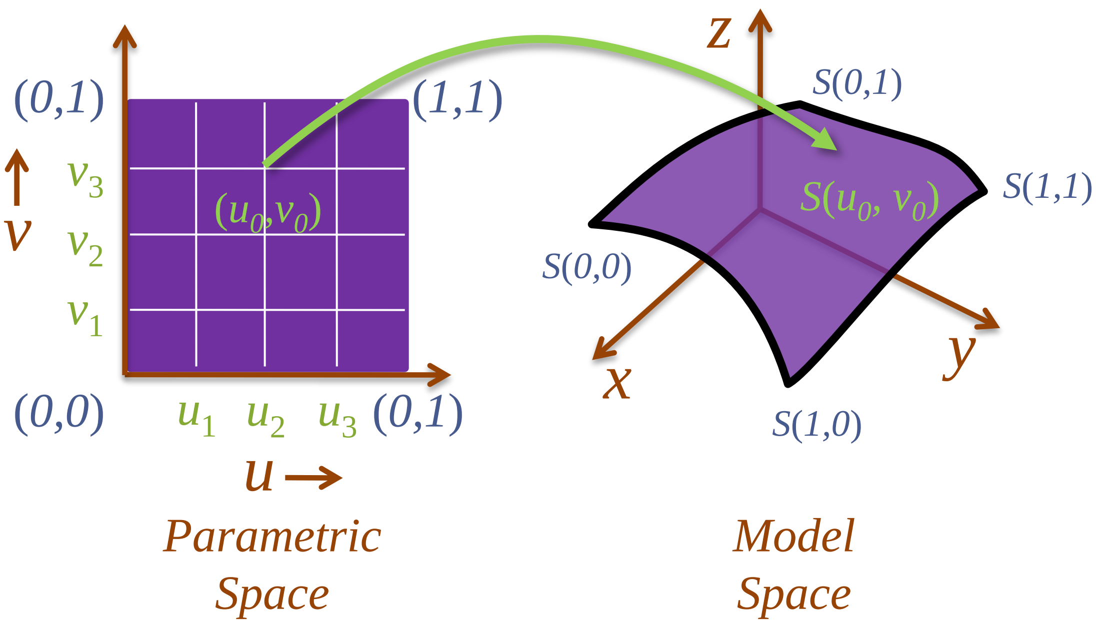
NURBS Evaluation



Surface Intersection

CPU/GPU Task Distribution

NURBS Representation



Non Uniform Rational B-Spline surfaces

De facto surface representation
- Most general spline
- Piecewise-polynomial tensor product surfaces

Compact definition
Defined completely by
- Control mesh
- u and v knot vectors

$$S(u,v) = \frac{\sum_{j=0}^{m}\sum_{i=0}^{n} N_i^p(u) N_j^q(v) w_{ij} P_{ij}}{\sum_{j=0}^{m}\sum_{i=0}^{n} N_i^p(u) N_j^q(v) w_{ij}}$$

$$N_i^p(u) = \frac{u - u_i}{u_{i+p} - u_i} N_i^{p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1}^{p-1}(u)$$

$$N_i^0(u) = \begin{cases} 1 & if \; u_i \le u < u_{i+1} \\ 0 & otherwise \end{cases}$$

NURBS Evaluation

- Several methods for evaluation
  - Power law
    - Numerically unstable for higher degrees
    - Issues with single precision graphics cards
  - Subdivision
    - Requires recursion
    - Not easily parallelizable
  - de Boor evaluation
    - Evaluate higher degree basis functions using lower degree basis functions
- Steps (given parameter ‘u’)
  - Find the knot span in which ‘u’ lies
  - Compute basis function values
  - Multiply basis function values with control points and add

Parallelizing Basis Function Evaluation

$$N_i^p(u) = \frac{u - u_i}{u_{i+p} - u_i} N_i^{p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1}^{p-1}(u)$$

$$N_i^0(u) = \begin{cases} 1 & \text{if } u_i \le u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

- Unroll Recursion
- Start from 0-degree basis function
- Build higher degree basis functions from lower degree
- Evaluate for different parameter values simultaneously

Faster Parallel Implementation

$N_i^p(u)$

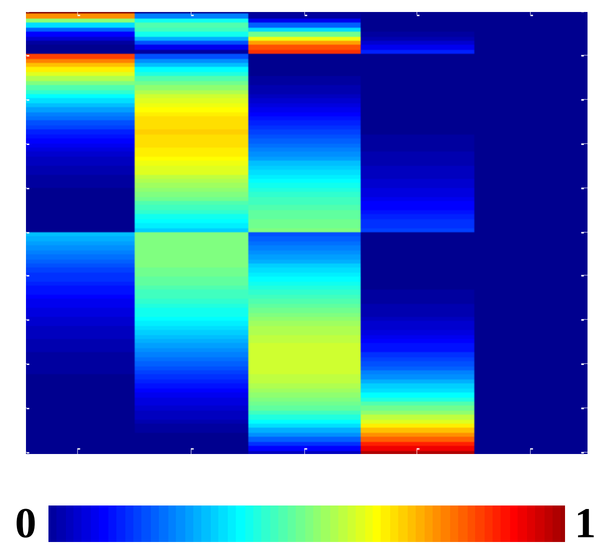
Degree							
0	0	0	0	1	0	0	0
1	0	0	x	x	0	0	
2	0	x	x	x	0		
3	x	x	x	x			

x = non-zero value

Different parameter values u

Degree 3

GPU Implementation



Parallelizing Control Point Multiplication

$$S(u,v) = \sum_{j=0}^{m}\sum_{i=0}^{n} N_i^p(u) N_j^q(v) w_{ij} P_{ij}$$

for j = 0 to 3
    for i = 0 to 3

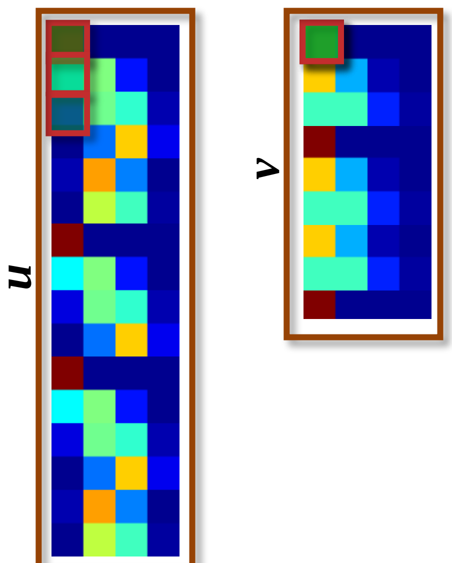
$S(u,v) \mathrel{+}= N_i^3(u) N_j^3(v) w_{ij} P_{ij}$

CPU

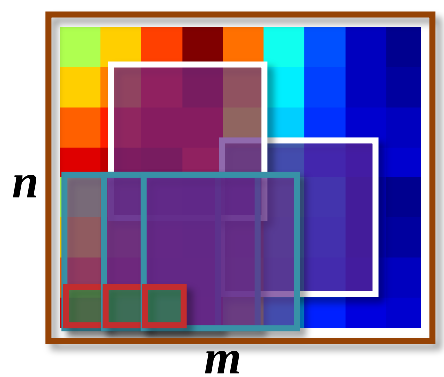
GPU Kernel

Faster Parallel Implementation

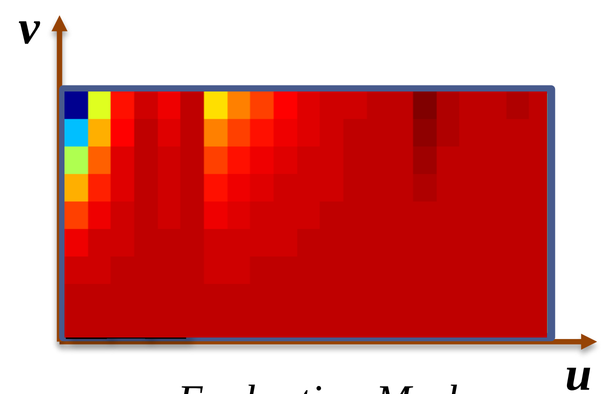
- Example for bi-cubic case
- Only a sub-mesh of control points need to be multiplied
- Perform multiplication operation in a GPU Kernel
- Add each multiplicative term in parallel (16 terms for bi-cubic)



Basis Functions

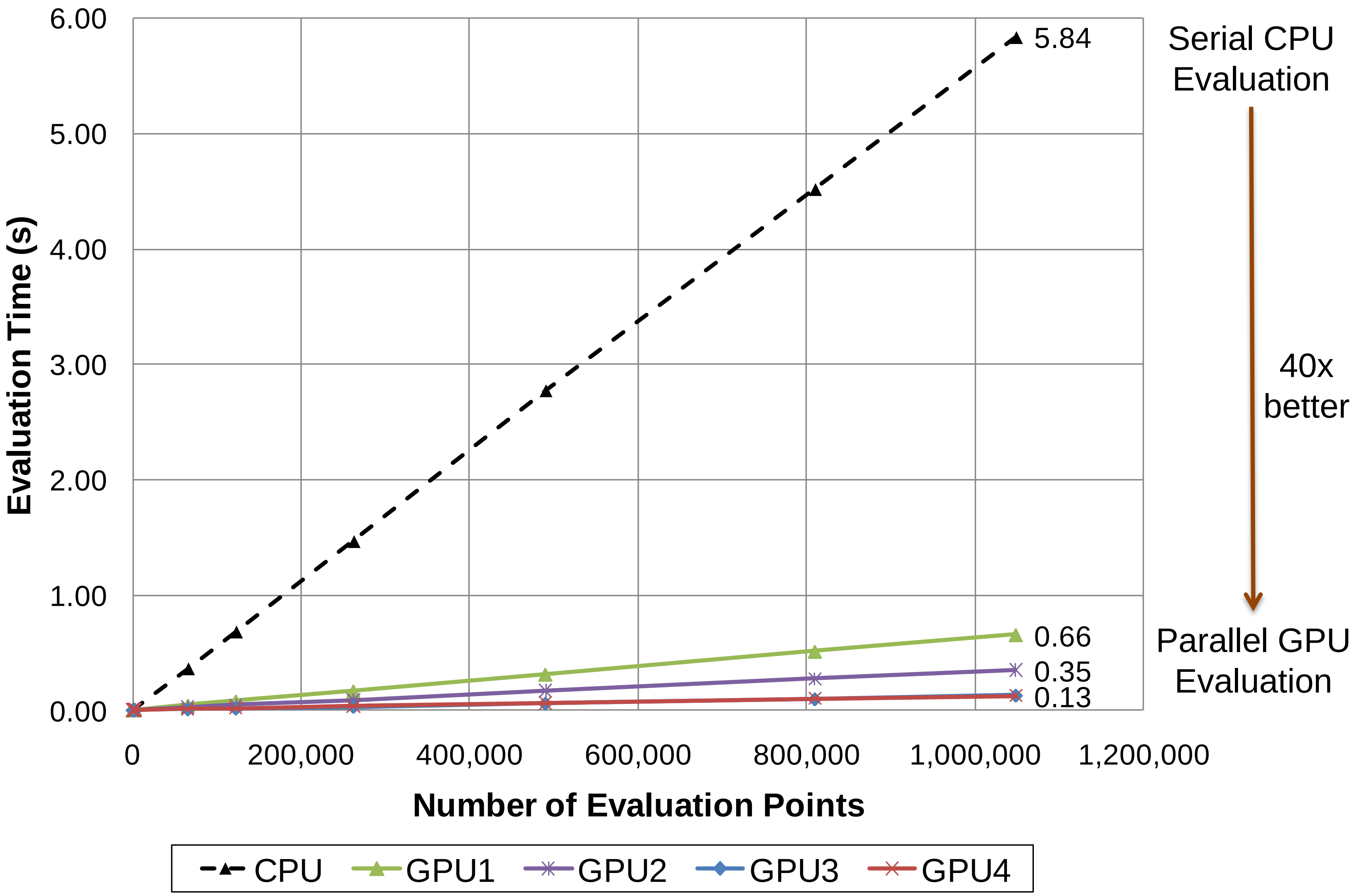


Control Mesh

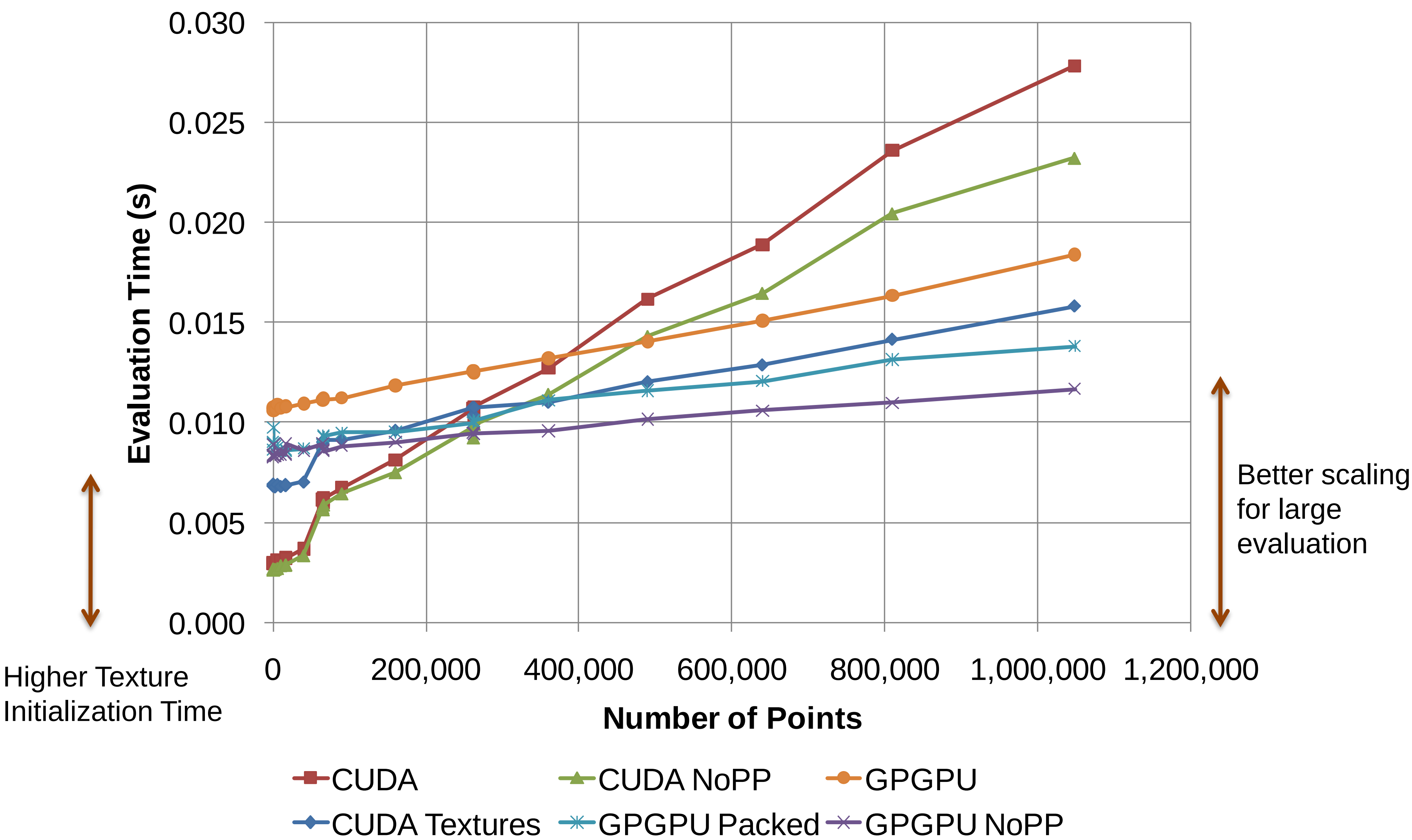


Evaluation Mesh

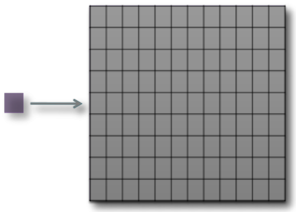
Results

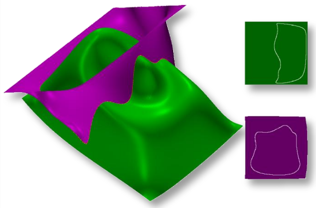

6.00
5.00
4.00
3.00
2.00
1.00
0.00
Evaluation Time (s)
0
200,000
400,000
600,000
800,000
1,000,000
1,200,000
Number of Evaluation Points
5.84
0.66
0.35
0.13
Serial CPU Evaluation
40x better
Parallel GPU Evaluation
CPU
GPU1
GPU2
GPU3
GPU4

Results – CUDA vs. GPGPU



Outline





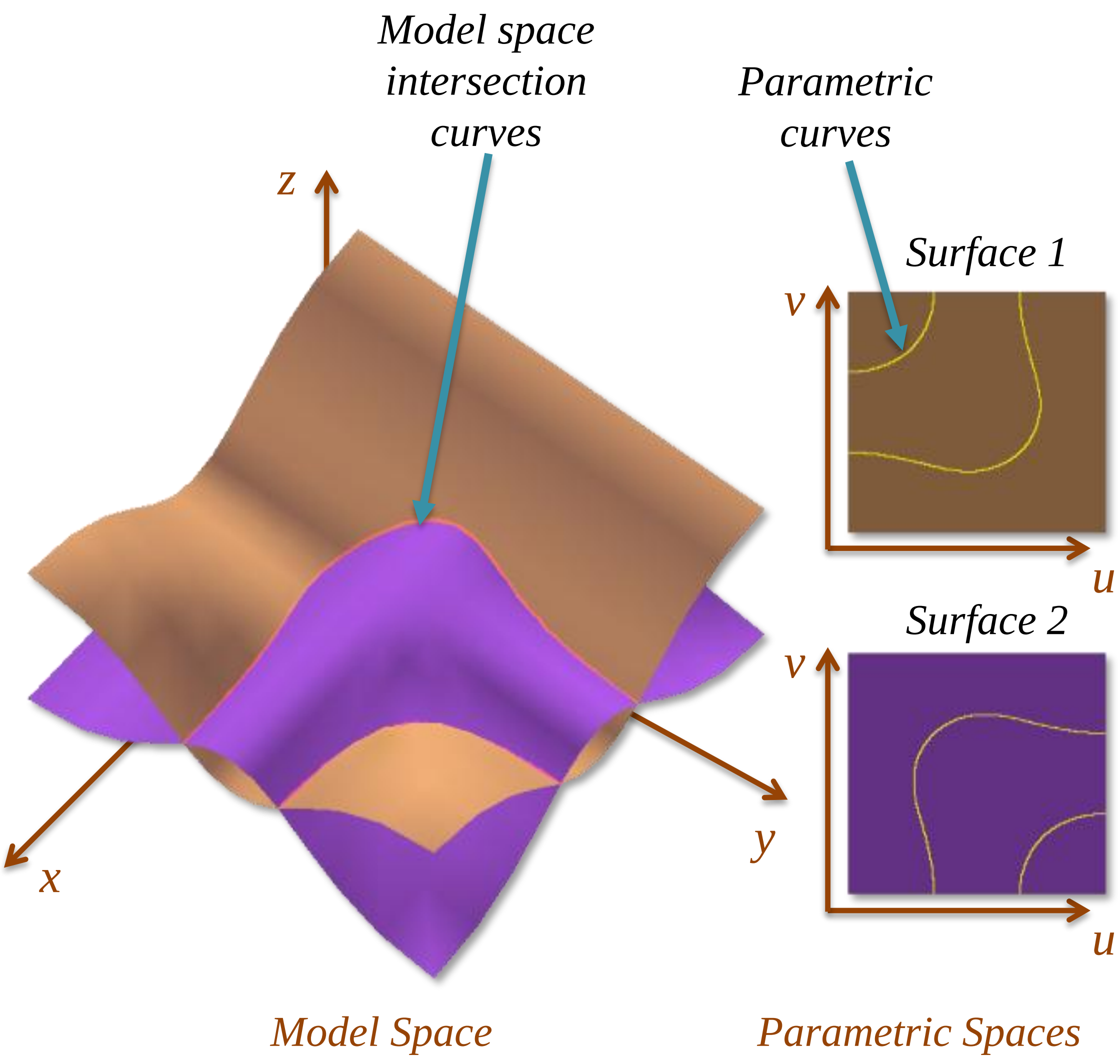
Motivation & Background

NURBS Evaluation

Surface Intersection

Structured Computations

Surface-Surface Intersection



Conventional Methods

- Newton-Raphson iteration
- Find single intersection point
- Curve marching techniques

[Barnhill et al. 1990]

Disadvantages

- Inherently serial operations
- Difficult to parallelize
- Slow

Parallelizable Solution
Use surface bounding-boxes

Surface Bounding-Boxes

Fit Axis-Aligned Bounding-Boxes (AABBs)

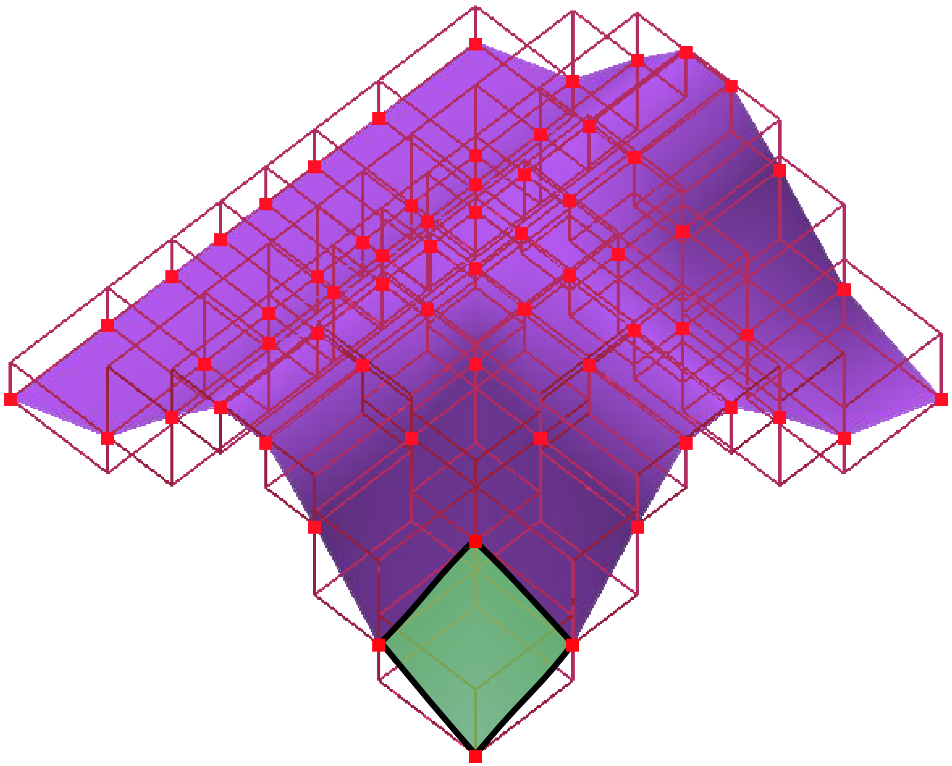
- Use grid of points already evaluated
- Find min, max x, y, & z coordinates of four adjacent evaluated points

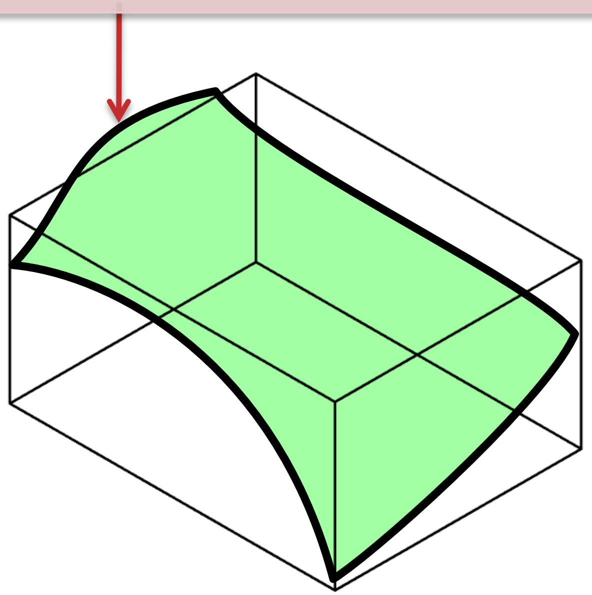
Advantage over OBBs

- Easier intersection tests
- OBB intersection fragment program significantly longer and complex

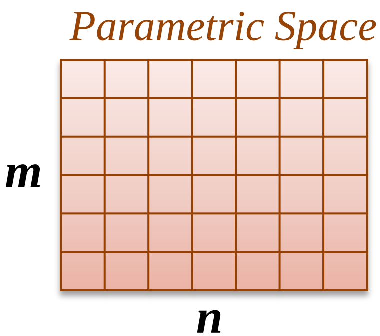
Problem using evaluated coordinates

- Surface patch may penetrate out of the bounding-box





Curvature-based Surface Bounds

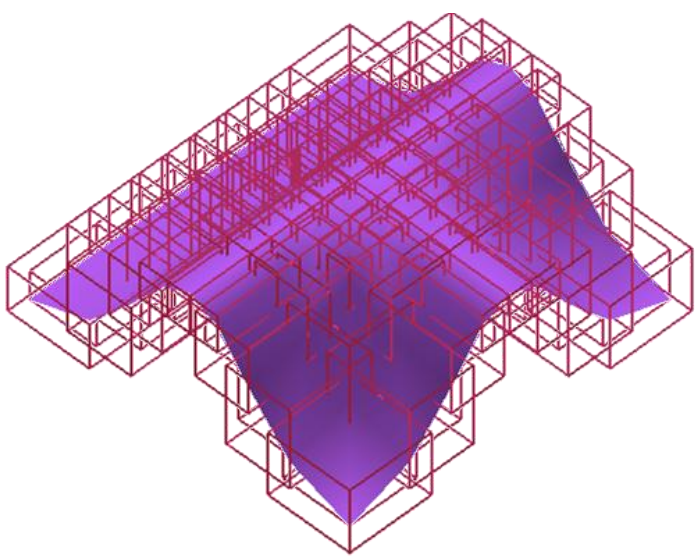


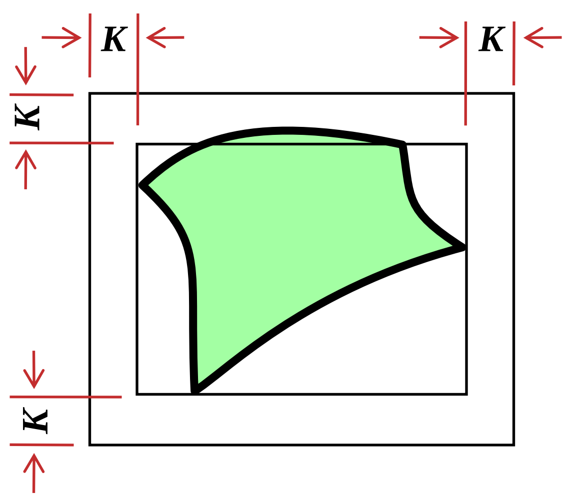
$$M_1 = \mathrm{Max}(\partial^2 S/\partial u^2)$$
$$M_2 = \mathrm{Max}(\partial^2 S/\partial u\, \partial v)$$
$$M_3 = \mathrm{Max}(\partial^2 S/\partial v^2)$$

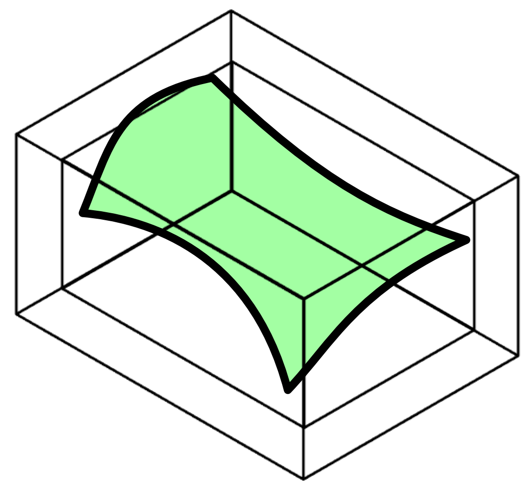
$$K = \frac{1}{8}\left(\frac{1}{n^2} M_1 + \frac{2}{nm} M_2 + \frac{1}{m^2} M_3 \right)$$

- K – Maximum deviation of the surface from piecewise-linear approximation
- Calculate bounding-box based on coordinates
- Increase size of bounding box by K

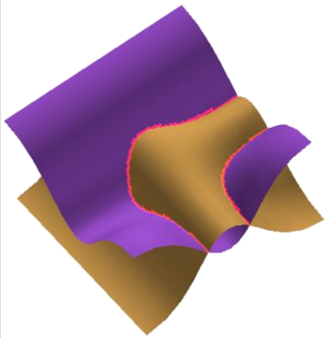
[Filip et al. 1986]

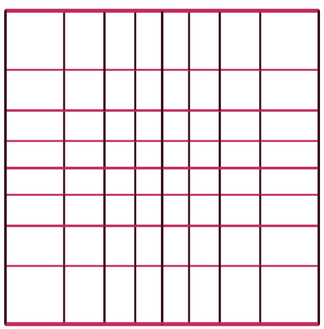




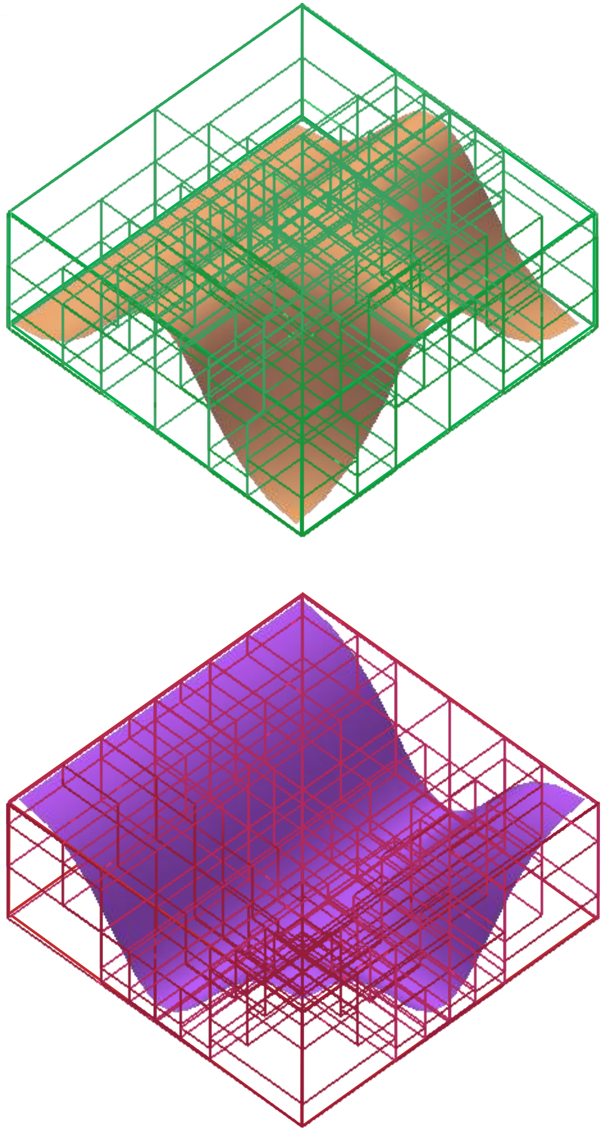


Surface-Surface Intersection Algorithm



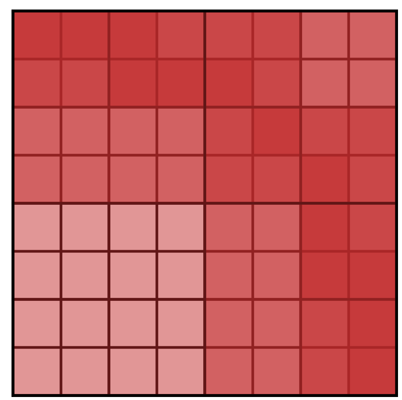


Parametric Space

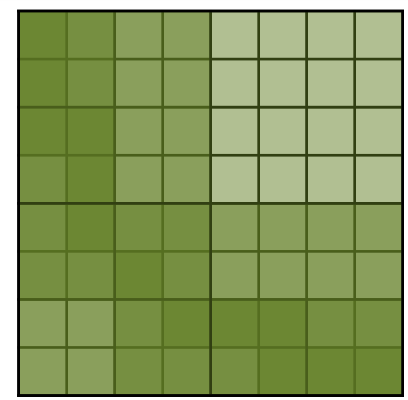


Model Space

- Construct finest level bounding-boxes based on user-specified tolerances
- Construct bounding box hierarchies
- Traverse the hierarchy from coarsest to finest level while keeping track of intersecting bounding-boxes
- Get bounding-boxes at finest level
- Intersect linearized patches on CPU to get points on the intersection curve
- Fit a polyline through the points



Surface 1



Surface 2

GPU Implementation

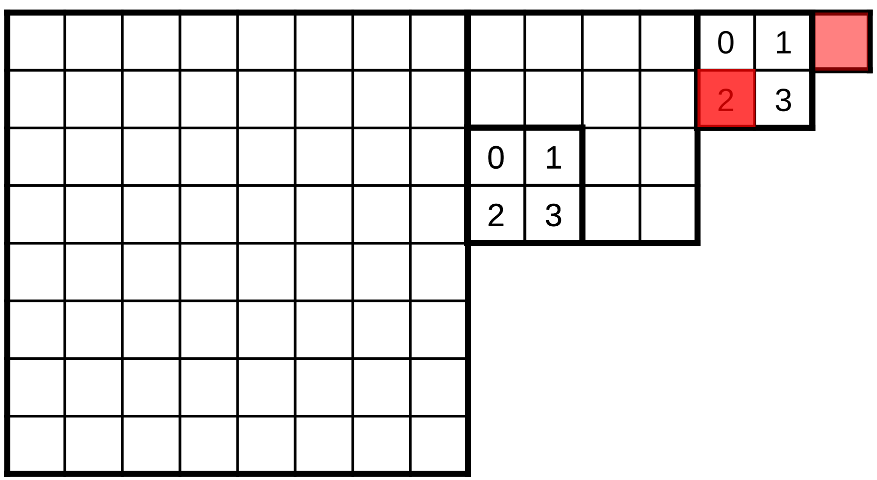
Surface 2

Surface 1

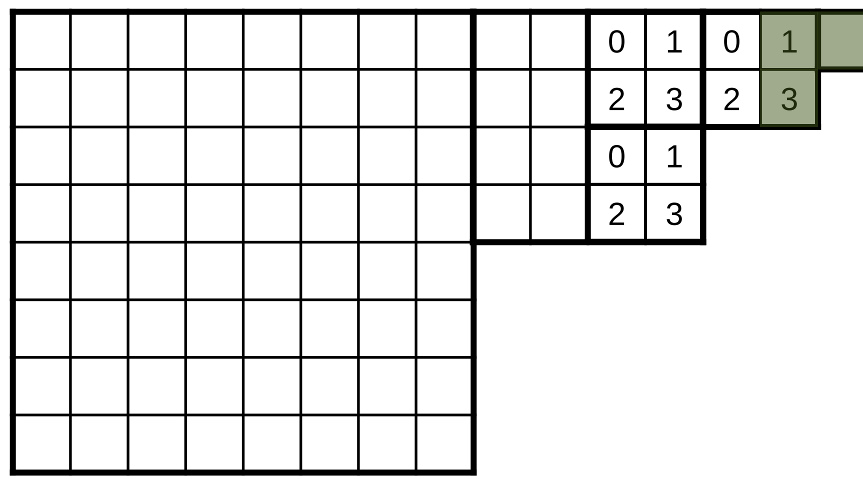
00	01	02	03	00	01	02	03
10	11	12	13	10	11	12	13
20	21	22	23	20	21	22	23
30	31	32	33	30	31	32	33

Intersection Texture

- Construct bounding box hierarchies
- Check largest box for intersection
- Check and track subsequent levels using the GPU
- Test for intersection in sets of 4 boxes from each surface
- GPU acceleration effective when more boxes intersect at finer levels



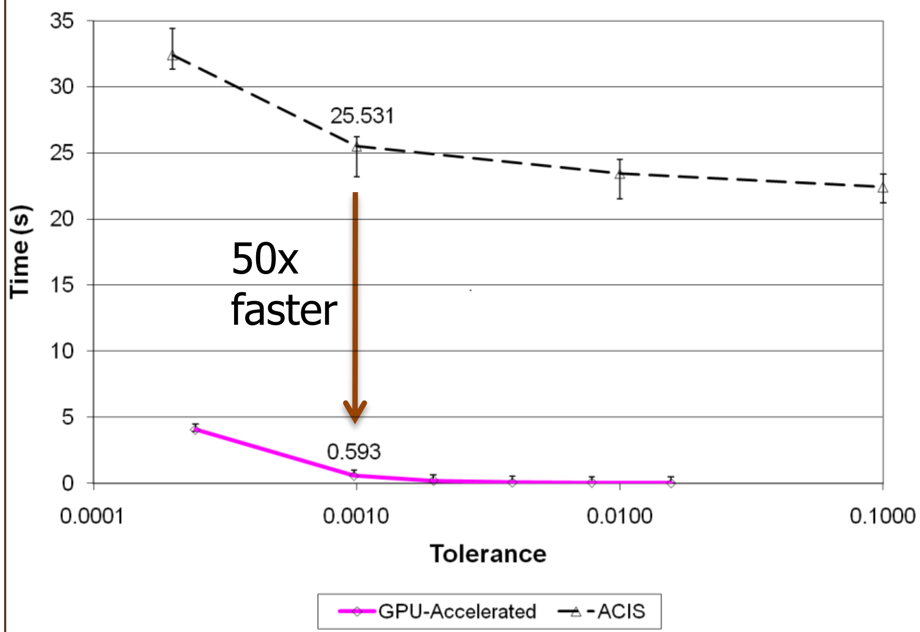
Surface 1

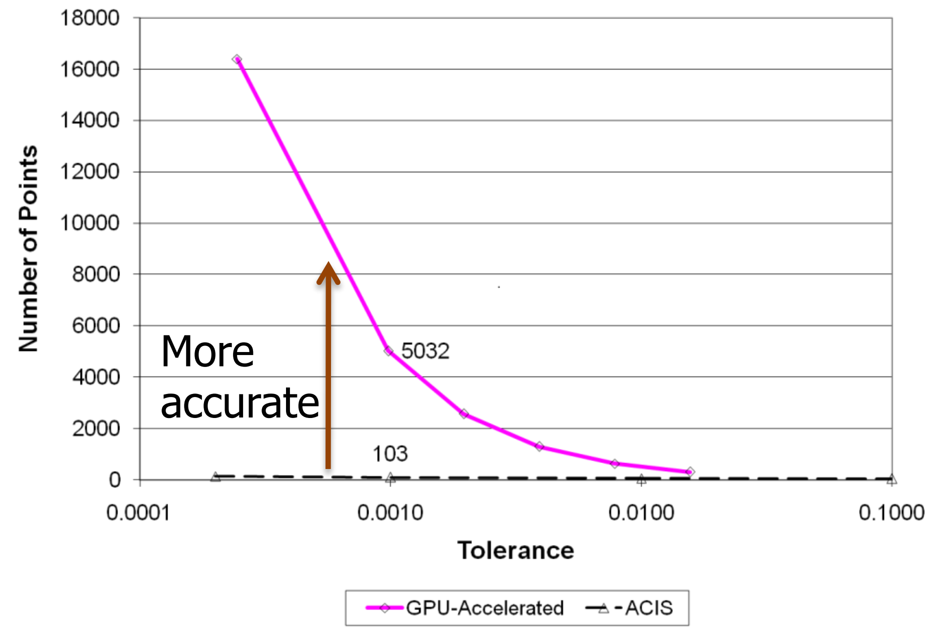


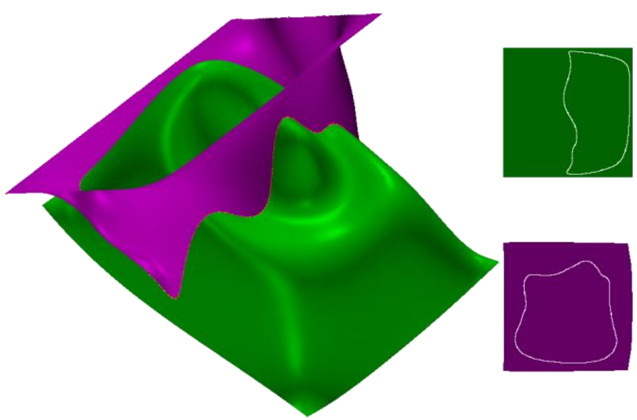
Surface 2

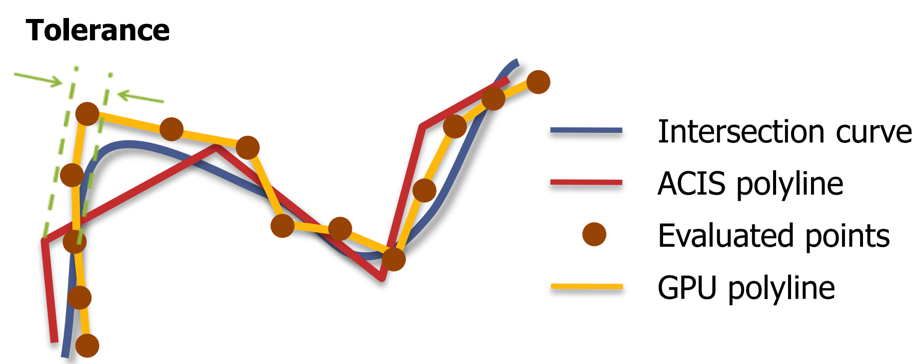
Results

Surface-Surface Intersection Timing









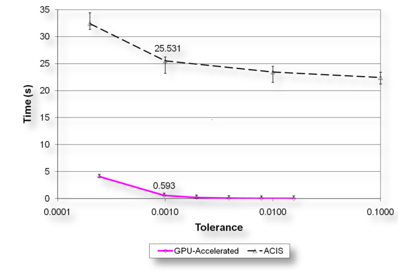
GPU Programming Insights

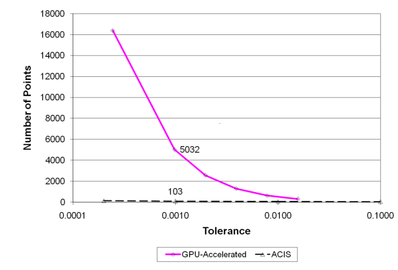
Dramatic performance gains
- Frequently orders of magnitude improvement
- However, requires GPU-optimized algorithms

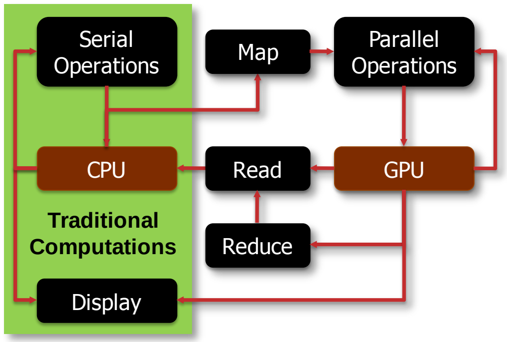
Compare both speed and accuracy
- CPU and GPU algorithms compared may be fundamentally different
- GPU algorithm needs to be faster and be at least as accurate as the CPU algorithm

Guaranteed user-specified tolerances
- Enables direct adoption of GPU algorithms in CAD

GPU framework
- Reduce development time for new algorithms
- Helps in performance tuning and optimization







Acknowledgments

- Kirk Haller

- Funding Sources
  - SolidWorks Corporation
  - UC Discovery
  - NSF

- Equipment
  - NVIDIA
  - AMD