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

40 Features
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

<table>
<thead>
<tr>
<th>GPU/CPU operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution of work</td>
</tr>
<tr>
<td>Some operations inherently serial</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic loops</td>
</tr>
<tr>
<td>Memory reads and writes</td>
</tr>
<tr>
<td>Single precision</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU performance guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coherent memory reads</td>
</tr>
<tr>
<td>Branchless kernels</td>
</tr>
<tr>
<td>Reduced data read-back from GPU</td>
</tr>
</tbody>
</table>

## Strategies

<table>
<thead>
<tr>
<th>Separation of CPU/GPU operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>NURBS evaluations</td>
</tr>
</tbody>
</table>

| Imposing artificial structure to the computations |
| Surface-surface intersections      |

## Multiple GPU vendors

<table>
<thead>
<tr>
<th>Implementation: not vendor-specific</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms: any massively parallel architecture</td>
</tr>
<tr>
<td>Many-core CPUs</td>
</tr>
</tbody>
</table>
Outline

Motivation & Background

NURBS Evaluation

Surface Intersection

CPU/GPU Task Distribution
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

NURBS Representation

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Degree</th>
<th>( N_i^p(u) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 x x x 0 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 x x x x x 0 0</td>
</tr>
<tr>
<td>3</td>
<td>x x x x x x 0 0</td>
</tr>
</tbody>
</table>

\( x = \text{non-zero value} \)

GPU Implementation

Different parameter values \( u \)
Degree 3

15 September 2010
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
\[ S(u,v) += N_i^3(u)N_j^3(v)w_{ij}P_{ij} \]

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)

Parallel Implementation

Basis Functions
Control Mesh
Evaluation Mesh
Results

![Graph showing evaluation times for Serial CPU and Parallel GPU evaluations. The graph includes lines for CPU, GPU1, GPU2, GPU3, and GPU4. The x-axis represents the number of evaluation points, and the y-axis represents evaluation time in seconds. The serial CPU evaluation time is 5.84 seconds, while parallel GPU evaluations are significantly faster, with GPU4 showing the fastest time at 0.13 seconds. There is a 40x improvement in performance compared to the serial CPU evaluation.]
Results – CUDA vs. GPGPU

Higher Texture Initialization Time

Evaluation Time (s)

Number of Points

CUDA CUDA NoPP GPGPU
CUDA Textures GPGPU Packed GPGPU NoPP

Better scaling for large evaluation
Outline

Motivation & Background

NURBS Evaluation

Surface Intersection

Structured Computations
Surface-Surface Intersection

Model Space

Parametric Spaces

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

15 September 2010
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

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

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

$K$ – 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

- 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

Parametric Space

Model Space

Surface 1

Surface 2
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
Results

Surface-Surface Intersection Timing

- 50x faster
- More accurate

Graphs showing the comparison between GPU-Accelerated and ACIS for Surface-Surface Intersection Timing. The graphs illustrate the time taken and the number of points evaluated at different tolerances, indicating significant improvements in both speed and accuracy.
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