Motivation

Geometry generation / modification is more and more shifting to GPU:

- **Real-time deformation**
  - Cloth and skeletal animation
  - Non-linear deformations
  - ...

- **Real-time tessellation**
  - Trimmed NURBS models
  - Subdivision surfaces
  - ...
Motivation

How to perform collision detection for GPU-generated deformable geometry?

Recent approaches:

- Classic collision detection on CPU
  - e.g. hierarchical collision detection with refitting of hierarchy
  - Drawback:
    - Replication of deformed geometry on CPU required (readback or recalculation)

Recent approaches:

- Image-space approaches on GPU
  - e.g. using stencil buffer
  - Drawbacks:
    - limited accuracy
    - speed dependent on accuracy
    - sometimes restricted topology
- Combined CPU-GPU approaches
  - determine only potential collisions on GPU
  - Drawback:
    - again replication of deformed geometry required
Motivation

How to perform collision detection for GPU-generated deformable geometry?

- Our approach:
  - Hierarchical (object space) collision detection on GPU
    - Fast
      - efficient hierarchy generation and traversal on the GPU
      - no replication of geometry on CPU
    - Maximum accuracy
      - (i.e. exact collisions for triangular surfaces, user defined accuracy for high-order surface approx.)
    - Well-suited for deforming geometry
      - because the hierarchy is reconstructed per-frame
    - Provides all contact pairs (as needed for collision response)

Basic Idea

Given: Parameterized deformable models

- Efficient data structure on GPU is essential
  - Represent input models as geometry images
    - Can be generated on the GPU from the given parameterization

- Per-frame:
  - Build AABB quadtree from geometry image and store it in mipmaps
  - Perform hierarchical collision test on the AABB quadtrees
AABB Hierarchy Generation

- Leaf level:
  - Bounding box for each quad
  - Stored in two textures (min point & max point)
  - Simple shader with two render targets

- Bottom up hierarchy generation
  - Custom (min/max) mipmap generation
Hierarchical collision detection

- Breadth-first traversal

Hierarchical collision detection

- **Overlap test:** Two AABBs overlap, if
  \[
  \min(bMax_0, bMax_1) - \max(bMin_0, bMin_1) \]
  non-negative in all components

→ Easy to compute in fragment shader
Hierarchical collision detection

Collision test of level $n$:
- **Input**: Stream of overlapping AABB pairs of level $n-1$
- **Output**: Stream of all level $n$ children of these AABB pairs with those marked (X) that do not overlap

- Each AABB in quadtree has 4 children
  $\Rightarrow$ 16 child AABB pairs for each AABB pair

How to continue with level $n+1$?

Given collision test of level $n$:
- **Input**: Stream of overlapping AABB pairs of level $n-1$
- **Output**: Stream of all level $n$ children of these AABB pairs with those marked (X) that do not overlap

**Task:**
- Transform output of level $n$ to input of level $n+1$
  $\Rightarrow$ known as "non-uniform stream reduction" [Horn 2005]
Non-uniform stream reduction

- Idea similar to recent approaches
  [Greß 2004, Horn 2005]
  but
  - extended to 2D
  - more efficient (runtime $8/3n$ instead of $n \log n$)

- Basic idea in 1D
  - Task:
    
    ![Diagram](image)

Non-uniform stream reduction

- Basic idea in 1D
  1. Count number of non-$\times$ cells for these regions:
     (calculated bottom-up)

    ![Diagram](image)
Non-uniform stream reduction

- Basic idea in 1D
  2. Count number of non-\(\times\) cells to the left of each entry (calculated top-down)

\[
\begin{align*}
0 + 3 &= 3 \\
1 &+ 2 = 0 \\
\end{align*}
\]

\[
\begin{array}{cccc}
A \times & C & D \times & G \times \\
1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 \\
\end{array}
\]
Non-uniform stream reduction

- Basic idea in 1D
  2. Count number of non-\(\times\) cells to the left of each entry (calculated top-down)

\[
\begin{array}{cccccc}
+ 1 &=& 2 & + 0 &=& 1 \\
0 + 1 &=& 1 & 0 + 1 &=& 2 & 1 + 0 &=& 3 & 0 + 1 &=& 4 & 0
\end{array}
\]

\[
\begin{array}{cccccc}
A & X & C & D & X & G & X
\end{array}
\]
Non-uniform stream reduction

- Basic idea in 1D
  3. Write non-\(\times\) cells to the compacted stream according to calculated indices

\[
\begin{array}{cccc}
A & C & D & G \\
0 & 1 & 2 & 3 & 3 & 3 & 4 \\
A & C & D & \times & \times & G & \times \\
\end{array}
\]

- Analogous technique in 2D
  \(\bigcirc\) cells (non-overlapping AABB pairs) should be removed

1. Count number of \(\bigcirc\) cells (overlapping AABB pairs) bottom-up:
   - Simple (additive) mipmap generation
Non-uniform stream reduction

- Analogous technique in 2D

2. Count number of cells to the left of each entry according to shown cell ordering ("Z order"):

- Top-down
  (as in 1D but using quad-tree instead of binary tree)
Non-uniform stream reduction

- **Analogeous technique in 2D**

  2. Count number of cells to the left of each entry according to shown cell ordering ("Z order"):

  - Top-down (as in 1D but using quad-tree instead of binary tree)

  ![Diagram showing cell ordering]

Non-uniform stream reduction

- **Extended technique in 2D**:

  3. Direct cells to the compacted stream according to calculated indices

  - performed in vertex shader

  - while fragment shader calculates collision test of level $n+1$: for each overlapping AABB pair, 16 child pairs have to be tested

  $\Rightarrow$ Efficient combined vertex & fragment shader usage:

  - 16 fragm. per non-culled vertex
Extensions

- Stenciled geometry images
  - "Stencil value" in alpha channel: set to 1 if texel corresponds to existing surface element
  - Additional condition for AABB overlap:
    \[ \min(bMax_0 \cdot \alpha, bMax_1 \cdot \alpha) > \frac{1}{2} \]

- Efficient self collision test based on [Volino et al. '94]
  - Basic idea:
    - Self-intersection only if surface forms a loop
    - Adjacent AABB pairs which contain a more or less flat surface can be culled
    - Flat surface parts are detected by evaluating all associated surface normals
    - This can be done hierarchically
  - Integrated easily into AABB hierarchy generation and traversal
    - See paper
Application to parametric surfaces

- Collision detection for trimmed NURBS
  - [Guthe et al. 2005]: GPU-based trimming and tessellation of NURBS and T-Spline surfaces
  - NURBS surfaces are rendered into RGB channels of geometry image
  - Trimming is rendered into alpha channel of geometry image (stenciled geometry image)

Results

deforming geometry  
trimmed NURBS surface
Results & Comparison

Potential restrictions:

- requires creation of geometry image, 
  however:
  - easy to generate and use for many GPU-based tessellation and deformation approaches
- simple hierarchy (balanced quadtree) with simple bounding volumes (AABBs),
  however:
  - very fast to generate \( \rightarrow \) well-suited for deforming geometry
  - fast to traverse because of cache-efficient memory layout
    - GPU-cache-friendly: 2D textures with mipmaps
    - breadth-first layout in general more cache-friendly than depth-first layout (see next talk)

Comparison of hierarchy traversal speed with classic CPU-based approaches (RAPID 2.01, PQP 1.3)

- with optimized hierarchies / more complex bounding volumes (but depth-first memory layout)

for non-GPU-generated, non-deforming geometry:
Results & Comparison

- Comparison for deforming geometry
  with fast CPU collision detection library (OPCODE 1.03):
  - with memory and cache optimizations
  - with refitting of AABB hierarchy for deforming geometry
    (but which requires readback of GPU-generated geometry)

![Graph showing comparison times](image)

Conclusions:
- ideal for *deforming* geometry
- advantageous for complex *GPU-generated* geometry:
  - hierarchies need not be stored permanently
    → largely reduced memory requirements compared to CPU approaches
  - tesselation accuracy for collision can be adjusted on the fly
Thank you