Efficient GPU Rendering of Subdivision Surfaces

Tim Foley, 2017-03-02
Collaborators

• Activision
  • Wade Brainerd

• Stanford
  • Matthias Nießner

• NVIDIA
  • Manuel Kraemer
  • Henry Moreton
Subdivision surfaces are a powerful modelling primitive

Smooth surface (+creases)
Arbitrary input topology
Animation
Level of detail
Subdivision surfaces are a powerful modelling primitive

- Smooth surface (+creases)
- Arbitrary input topology
- Animation
- Level of detail
Subdivision surface rendering is not common in games

Performance

Ease of Integration
Performance

Our method is up to 3x faster than previous non-approximate schemes

Ease of Integration

Our method can work in a single draw pass (no compute)
Can use with existing vertex shaders for animation
Outline

• Background
  • Subdivision surfaces
    • GPU tessellation hardware
• Prior work
• Overview of our approach
• Performance evaluation
• Conclusion
Outline

• Background
  • Subdivision surfaces
    • GPU tessellation hardware
• Prior work
• Overview of our approach
• Performance evaluation
• Conclusion
Catmull-Clark Subdivision

base mesh
Defined by repeated application of subdivision rules

base mesh
Defined by repeated application of subdivision rules

base mesh

subdivided once
Defined by repeated application of subdivision rules

- base mesh
- subdivided once
- limit surface
Limit surface for face depends on local neighborhood
Limit surface for face depends on local neighborhood

Connectivity of 1-ring vertices
Regular faces are easy to evaluate

Limit surface equivalent to bicubic B-spline patch
Irregular Face
Extraordinary Vertex

valence \neq 4
Outline

• Background
  • Subdivision surfaces
  • GPU tessellation hardware
• Prior work
• Overview of our approach
• Performance evaluation
• Conclusion
GPU Rasterization Pipeline
Tessellation Stages

Vertex Fetch -> Vertex Shader -> Hull Shader -> Tessellator -> Domain Shader -> Geometry Shader -> Rasterizer -> Fragment Shader -> Blend
Hull Shader \rightarrow \text{Tessellator} \rightarrow \text{Domain Shader}
primitive base vertices

control points

Hull Shader → Tessellator → Domain Shader

www.gameworks.nvidia.com
primitive
base vertices

control points

domain
locations

Hull Shader

Tessellator

Domain Shader

www.gameworks.nvidia.com
primitive base vertices

control points

tessellated primitive post-tessellation vertices

domain locations

Hull Shader → Tessellator → Domain Shader

www.gameworks.nvidia.com
Crux of the Challenge

Limit surface of irregular face defined by recursive subdivision

Expands to many faces with many control points

Variable: depends on subdivision depth

Tessellation hardware wants fixed # of control points per face
Outline

• Background
  • Subdivision surfaces
  • GPU tessellation hardware

• Prior work
  • Overview of our approach
  • Performance evaluation
  • Conclusion
Submit one primitive for each irregular face

Use a fixed # of control points

Submit many primitives for an irregular face

Each of which is simple to evaluate
Submit one primitive for each irregular face

Use a fixed # of control points

Submit many primitives for an irregular face

Each of which is simple to evaluate
Exact Evaluation

• Perform Eigen analysis on subdivision matrix
  • Offline process for each topological configuration

• Project base vertices into Eigen space
  • Yields a fixed # of control points

• Matrix exponentiation in domain shader
  • Many floating-point operations

[Stam 1998]
Approximate irregular faces with simpler patch

• Bicubic Bezier
• Bicubic Gregory (20 control points)  [Loop and Schaefer 2008]
• Fast evaluation  [Loop et al. 2009]
• No support for semi-sharp features (creases)
• Approximation affects tangents, parameterization
Submit one primitive for each irregular face

Use a fixed # of control points

Submit many primitives for an irregular face

Each of which is simple to evaluate
Subdivide and submit many primitives per face

• Feature adaptive subdivision (FAS)
  • Generate sub-face control points using compute kernels
  • Many submit many primitives, depending on subdivision level
  • Need to address T-junctions between sub-faces

• Dynamic feature adaptive subdivision (DFAS)
  • Enables non-uniform subdivision levels

[Nießner et al. 2012]
[Schäfer et al. 2012]
Issues with Feature-Adaptive Subdivision

- many primitives
- T-junctions
Outline

• Background
  • Subdivision surfaces
  • GPU tessellation hardware
• Prior work
• Overview of our approach
• Performance evaluation
• Conclusion
Take recursive subdivision hierarchy...

many primitives

T-junctions
Summarize using a single primitive

- one primitive
- no T-junctions
Two key ideas

Use a quadtree to map domain locations to sub-faces

Output a variable # of control points from a Hull Shader
Two key ideas

Use a quadtree to map domain locations to sub-faces

Output a variable # of control points from a Hull Shader
Submit one primitive per base face to tessellator

base subdivision face
Tessellator produces domain locations for evaluation

base subdivision face

domain locations
Map domain location to correct sub-face

base subdivision face

domain locations
Map domain location to correct sub-face

base subdivision face
domain locations
Map domain location to correct sub-face

using a quadtree data structure

base subdivision face
quadtree
domain locations
Map domain location to correct sub-face

using a quadtree data structure
Quadtrees can be built ahead of time, and shared
Quadtrees can be built ahead of time, and shared depend only on 1-ring topology
Quadtree leaf node tells us which control points to use
Quadtree leaf node tells us which control points to use.
Quadtree leaf node tells us which control points to use

base subdivision face

B-spline control points
Evaluate sub-face using its control points

- base subdivision face
- B-spline control points
- tessellated primitive
Two key ideas

Use a quadtree to map domain locations to sub-faces

Output a variable # of control points from a Hull Shader
Hull Shader
Hull Shader

Domain Shader

cut-up point buffer
Hull Shader

Domain Shader

count points needed for tessellation factor 3.0
More details in paper

- Collapsing repeated structure in quadtrees
  - Most faces need only one tree traversal step!

- Sorting control point stencils for efficient evaluation
  - Minimize number of control points needed for given tessellation factors
  - Arrange control points for efficient SIMD computation
Outline

• Background
  • Subdivision surfaces
  • GPU tessellation hardware
• Prior work
• Overview of our approach
• Performance evaluation
• Conclusion
Up to 3x faster than Adaptive Subdivision

*non-exact
Up to 3x faster than Adaptive Subdivision

*non-exact
Big Guy  
Monster Frog  
Armor Guy  
Sterling  

low complexity  
no crease tags  
higher complexity  
semi-sharp crease tags  

(© Disney/Pixar)  
(© 2014 DigitalFish, Inc. All rights reserved)  

www.gameworks.nvidia.com
Benefit decreases as fraction of regular faces increases
Only some methods can handle semi-sharp creases
Benefit decreases as fraction of regular faces increases
Big Guy  |  Monster Frog  |  Armor Guy  |  Sterling

low complexity  |  no crease tags  |  higher complexity  |  semi-sharp crease tags

(© 2014 DigitalFish, Inc. All rights reserved)

(© Disney/Pixar)
Armor Guy has a greater fraction of irregular faces
Outline

• Background
  • Subdivision surfaces
  • GPU tessellation hardware
• Prior work
• Overview of our approach
• Performance evaluation
• Conclusion
Conclusion

• A simpler and faster way to render subdivision surfaces
  • Up to 3x faster than state-of-the-art methods
  • Single draw pass
  • Can use existing shaders for animation

• Integration in open-source OpenSubdiv library is in progress
• Interested engine developers should contact NVIDIA
Thank You