SIGGRAPH2009 NEW ORLEANS

# A Real-time Micropolygon Rendering Pipeline

**Kayvon Fatahalian**
**Stanford University**

Detailed surfaces
Credit: DreamWorks Pictures, Shrek 2 (2004)

Credit: Pixar Animation Studios, Toy Story 2 (1999)

Credit: Crytek

(one pixel)
Zoom in (pixels)

(one pixel)
Micropolygons

Camera defocus

Motion blur

# Rendering goals

**Highly detailed surfaces: micropolygons**
**(But interoperate with large triangle rendering in same frame)**

**Accurate camera defocus and motion blur**

**[Near future] real-time system**

**It is inefficient to render micropolygons using the current graphics pipeline.**

# How does the real-time graphics pipeline evolve to enable efficient micropolygon rendering?

How should surfaces be tessellated into micropolygons?

How can micropolygons be rasterized efficiently?

How expensive is motion blur and defocus?

Should the pipeline shade like GPUs or like REYES?

(changes to algorithms, abstractions, and HW are fair game)

# TESSELLATION:

## How do we add adaptive tessellation to the pipeline?

# Generate micropolygons on demand

# D3D tessellates parametric surfaces

Coarse Vertex Processing

Primitive Processing — Compute tessellation factor

Tessellate — Generate new triangles

New in D3D11

Fine Vertex Processing — Compute position of new triangle verts

Rasterize+Cull

Fragment Processing

Frame buffer ops

Fixed function =

# Need adaptive tessellation

- Uniform tessellation per patch over/under tessellates

# Performing good adaptive tessellation is very important.

# Complexity pays for itself.

Less surface evaluation
Less shading
Less (and more efficient) rasterization

# Split-dice adaptive tessellation

## Split: recursively divide primitive (adaptive)
## Dice: simple uniform tessellation (fast)

Base Patch

Micropolygon Mesh

Split

Split (again)

Set Rates

Dice

# D3D11 tessellation

- Don't split, just dice
  - Tessellation factors for primitive edges
  - "Fancy" dicer stitches edges to uniform interior

REYES dicing

D3D11 dicing

# Parallel, adaptive tessellation must avoid cracks

# Offline techniques suboptimal for real-time

- Retain the entire micropolygon mesh and "stitch" adjacent subpatches
  - Very robust
  - Hard to parallelize (can't stream)
- Use only power-of-two dicing factors
  - Easier to parallelizable
  - T-junctions
  - Over tessellates

# Split-dice for a real-time pipeline

- DiagSplit: simple solution
  - Adaptive tessellation
  - Independent subpatch processing
  - Crack-free
  - No power-of-two constraints

Fisher et al. [Siggraph Asia 2009]

# DiagSplit: idea 1

- Leverage D3D11 tessellator as advanced dicer
  - Edge-based tessellation factor function: h(edge)
  - Let h evaluate displacement

Dicable: send to Dicer

Evaluate h on subpatches

# DiagSplit: idea 2

- Scale area of internal polygons to hit target area precisely

# DiagSplit: idea 3

- Permit "diagonal" (non-isoparametric) splits

Non-isoparametric patch split

Direct3D 11
Power-of -Two
Split-Dice + Stitching
DiagSplit

# What is the cost of splitting?

- Most time is spent evaluating final surface positions
- Then you’ve got to rasterize and shade
- Avoiding over tessellation pays for itself

- 10 million MP/sec on one core of Intel I7

# Parallel tessellation is an active area

- Expressibility of parametric surfaces is increasing
  - Approximate Catmull-Clark surfaces [Loop (TOG 2008)]
  - Extension to creases and corners [Kovacs (I3D 2009)]

- Variety of implementations
  - Depth-first (cache efficient, tiny footprint)
  - Breath-first subdivision (fits current GPU compute model well)
    - Patney (SIGGRAPH Asia 2008), Patney (HPG 2009)
    - Eisenacher (I3D 2009)
    - RenderAnts
  - CUDATess [Schwarz (Eurographics 2009)]

D3D11 Pipeline
Coarse Vertex Processing
Primitive Processing
Tessellate (Dice)
Fine Vertex Processing
Micropolygon Pipeline
Coarse Vertex Processing
Primitive Processing
Split
Tessellate (Dice)
Fine Vertex Processing
Tessellation Rate Func
Bounding Box Func
Programmable =
Fixed function =

# RASTERIZATION:

## How efficiently can micropolygons be rasterized? (and what's the cost?)

# Rasterization

# Step 1: per-polygon preprocessing (setup)

- Clip, back face cull, compute edge equations
- Make point-in-polygon tests cheap

# Step 2: compute candidate sample set

- Coarse reject/accept of samples

Coarse uniform grid

# Step 2: compute candidate sample set

- Coarse reject/accept of samples

Coarse uniform grid

Hierarchical descent

# Step 2: compute candidate sample set

- Coarse reject/accept of samples

Coarse uniform grid

Hierarchical descent

# Step 2: compute candidate sample set

- Coarse reject/accept of samples

Coarse uniform grid

Hierarchical descent

# Step 3: point-in-polygon tests

- Test "stamp" of samples against polygon simultaneously (data-parallel)

[Pineda 88]
[Fuchs 89]
[Greene 96]
[Seiler 08]

# Micropolygons: more polygons = more setup

# Micropolygons: coarse reject not useful

# Micropolygons: large stamps yield low efficiency

47% of tested samples inside triangle

6% of tested samples inside triangle

# Re-implement rasterizer

(Fatahalian et al. HPG 2009)

- Rasterize many micropolygons simultaneously

# Reimplement rast to increase efficiency

## (2.5 to 6x more efficient)

# Micropolygon rasterization is expensive

## Primary visibility computation:

- 1080p resolution, 30 Hz
- 4x multi-sampling
- Simple scene (10 M micropolygons)

## Estimated cost of GPU SW implementation:

- Approximately 1/3 of high-end GPU

**Fixed-function micropolygon rasterization is appealing**

# MOTION BLUR AND DEFOCUS:

## How expensive are motion blur/defocus?

# Motion blur and defocus

- Many 2D-techniques for approximating blur
  - [Sung 02]
  - [Demers 04]
- Stochastic point sampling
  - [Cook 84, Cook 86]
  - [Akenine-Moller 07]

# Moving micropolygon

# Soccer jump

16x multi-sampling

64 unique time samples

# Enabling motion/defocus blur costs 3 to 7x more

- Point-in-polygon tests are more expensive
- Algorithms that support blur perform more point-in-polygon tests

Stationary geometry,
perfect focus:

Tests 2.5x more samples
Cost 3x more

# Performance varies with motion

# Most efficient algorithm depends on scene conditions

# [Pixar method] costs increase sharply with defocus

D3D11 Pipeline
Coarse Vertex Processing
Primitive Processing
Tessellate (Dice)
Fine Vertex Processing
Rasterize+Cull
Micropolygon Pipeline
Coarse Vertex Processing
Primitive Processing
Split
Tessellation Rate Func
Bounding Box Func
Tessellate (Dice)
Fine Vertex Processing
open and close shutter
vertex positions
Rasterize (5D)
shutter duration,
lens parameters
Unchanged =
Programmable =
Fixed function =

# SHADING:

## Should a micropolygon pipeline shade like today's GPUs or like REYES?

D3D11 Pipeline
Coarse Vertex Processing
Primitive Processing
Tessellate (Dice)
Fine Vertex Processing
Rasterize+Cull
Fragment Processing
Frame buffer ops
Micropolygon Pipeline
Coarse Vertex Processing
Primitive Processing
Split
Tessellation Rate Func
Bounding Box Func
Tessellate (Dice)
Fine Vertex Processing
Cull+Shade?
Rasterize (5D)
Cull+Shade?
Frame buffer ops
Unchanged =
Programmable =
Fixed function =

# GPUs shade micropolygons inefficiently

micropolygon input

Four shading samples output

**At least 4x overshade!**

(must share shading results across micropolygons)

# MSAA implementations break down with blur

(inefficient to represent as hit mask)

Motion-blurred polygon input

Covered samples are spread widely across screen

# Future fragment shading?

**Decoupled sampling: Ragan-Kelley et al.**

# REYES-style shading

- Perform shading computations at vertices
- Requires connectivity for derivatives
  - Not a problem: system has the diced mesh
- Decoupled shading for motion blur and defocus "just works"

# Overshade in a “REYES” system

- Overtessellation
- Conservative, coarse occlusion culling

# Shading system design space

Current GPU fragment shading

Sample in screen space

Shade only what's visible

Fine early Z

Deferred shading

Shade before exact visibility known

Sweet spot?

REYES: shade at diced grid vertices

Sample in object space

# SUMMARY

# Micropolygon Pipeline

**Good adaptive tessellation Is very important**

**DiagSplit: Build split on top of D3D11 Tess**

**Abstract SPLIT as non-programmable stage**

# Micropolygon Pipeline

**Shading on surface: Need derivatives & good tessellation**

**We've got the diced grid for connectivity**

# Micropolygon Pipeline

**Rasterization: re-implement for micropolygons**

**Fixed-function HW likely**

**Point sampling for motion blur and defocus are 3-7x more costly**

# Micropolygon Pipeline

**Must share shading across micropolygons to avoid overcompute**

**(quad per MP is inefficient)**

**Motion blur/defocus change fragment shading (even without micropolygons)**

# It's coming!

- We'll have the flops from future HW
- Lot's of active research on algorithms

- Real-time micropolygon rendering will be possible in the future

# Thank you