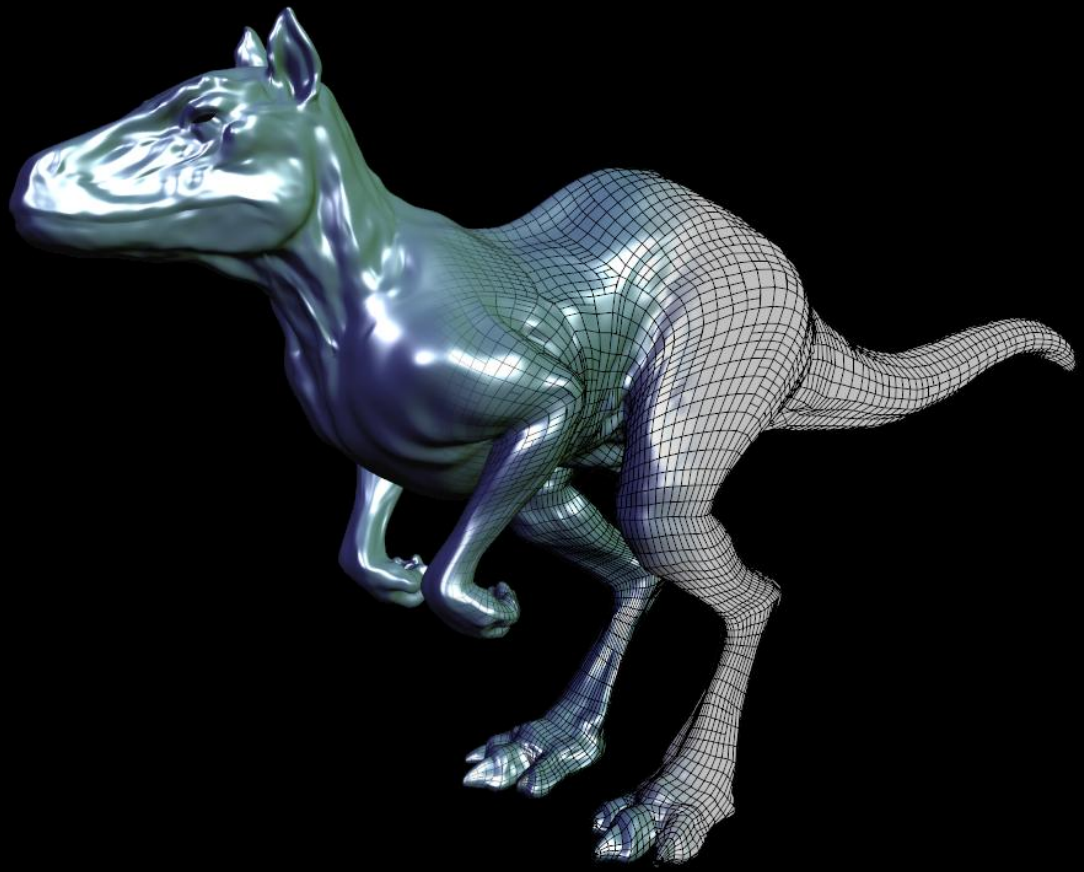
Real-Time Reyes: Programmable Pipelines and Research Challenges

Anjul Patney
University of California, Davis

Real-Time Reyes-Style Adaptive Surface Subdivision



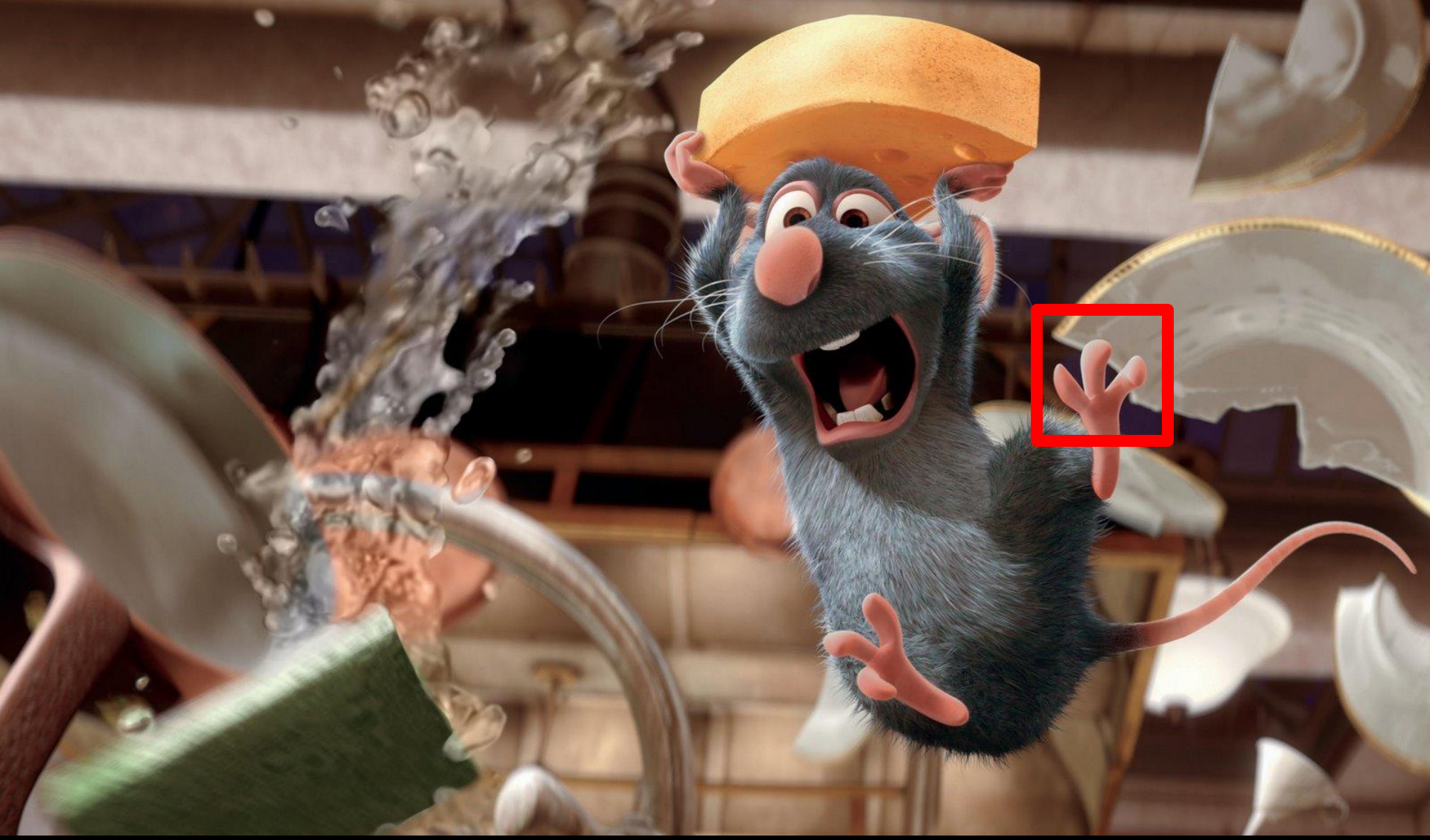
Anjul Patney and John D. Owens

SIGGRAPH Asia 2008 (to appear)

http://graphics.idav.ucdavis.edu/publications/print_pub?pub_id=952

Cinematic rendering looks good

Image courtesy: Pixar



High geometric complexity

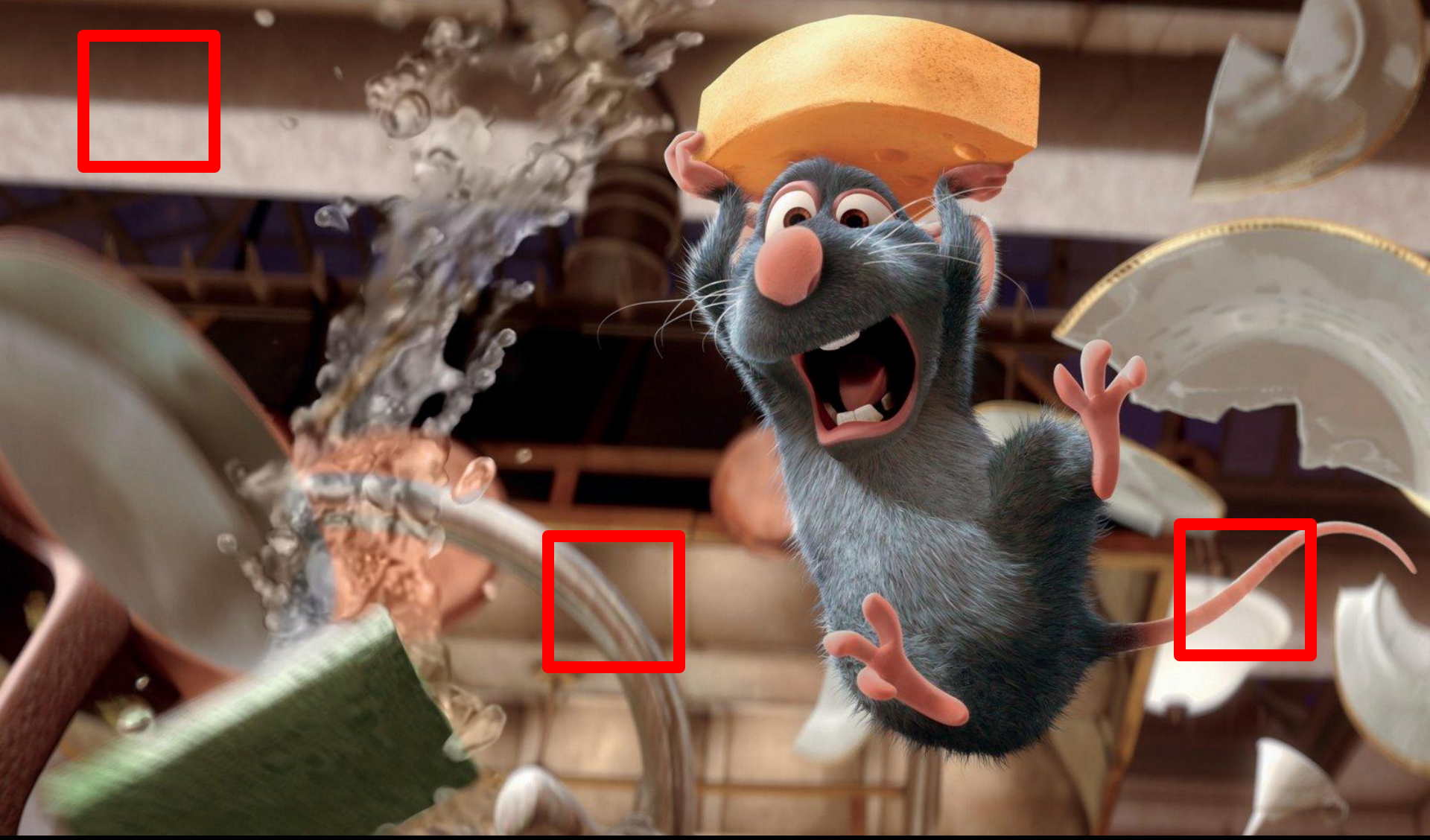
Image courtesy: Pixar

High shading complexity

Image courtesy: Pixar

Motion blur, Depth-of-field

Image courtesy: Pixar



Complex lighting

Image courtesy: Pixar

All this is slow

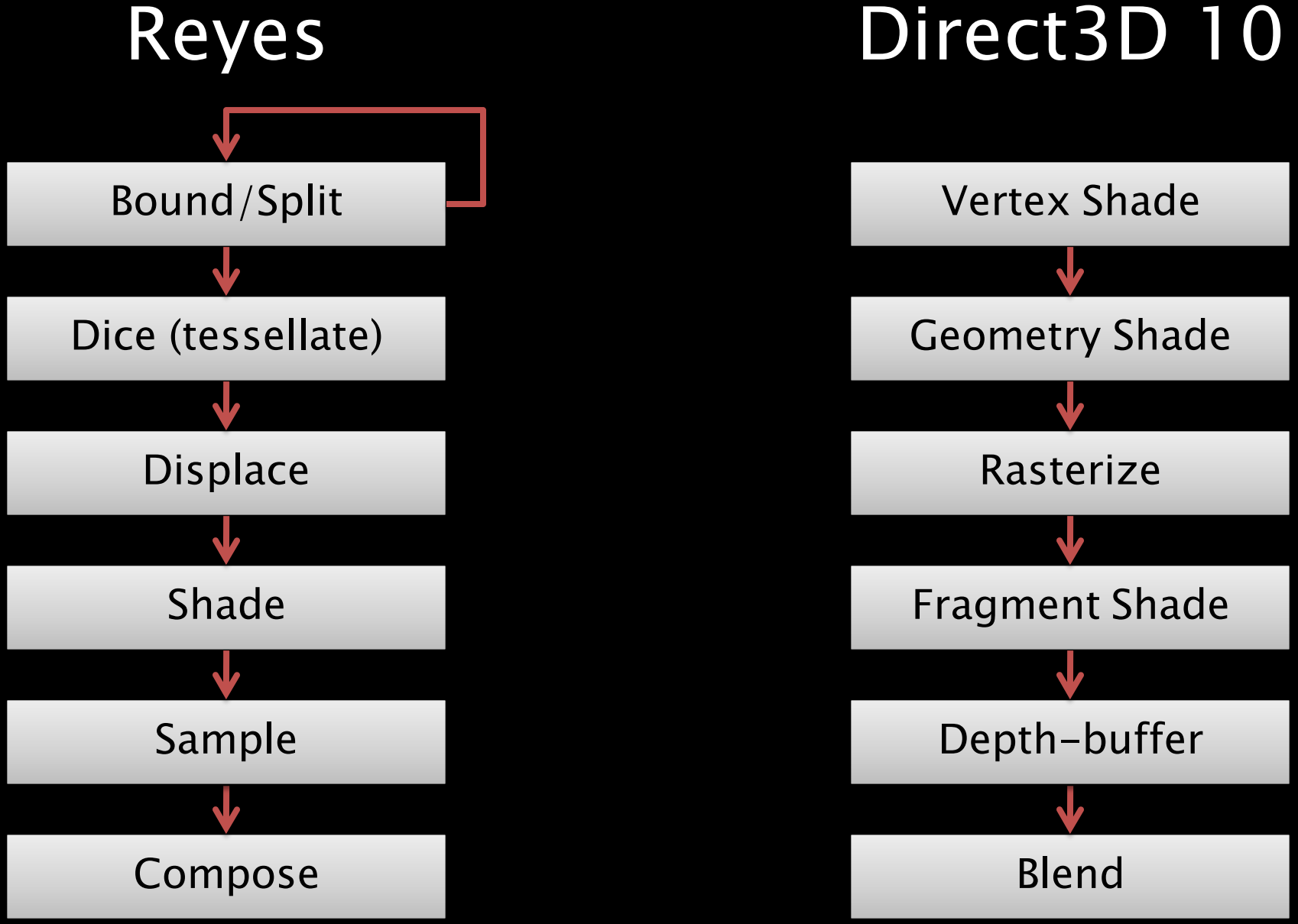
Image courtesy: Pixar

Our Goal

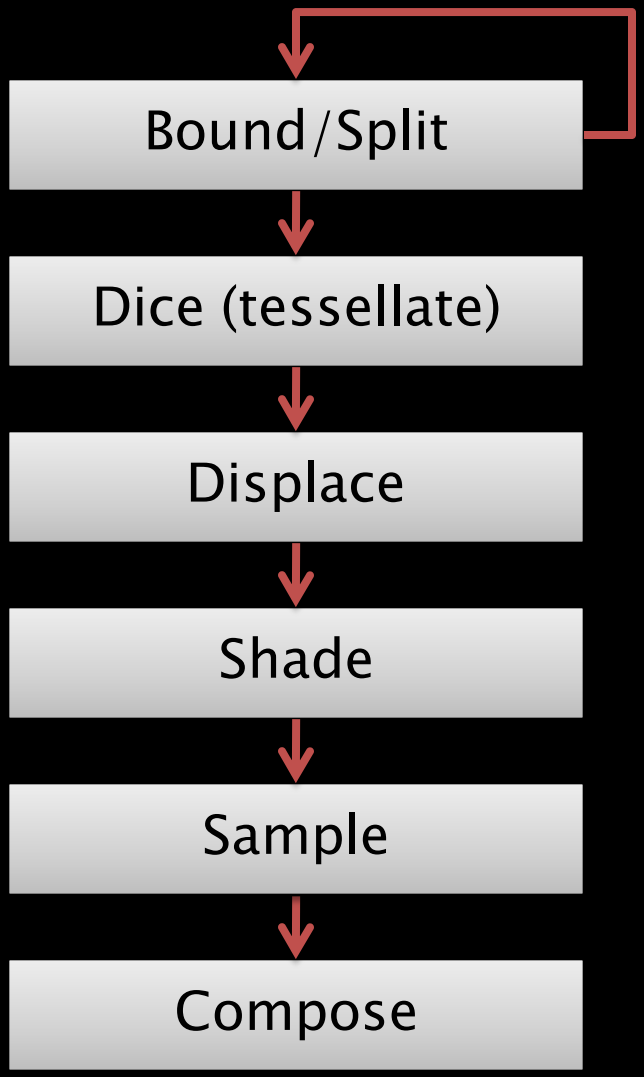
- Make it faster
  - As much as possible in real time
- Use the GPU
  - Massive available parallelism
  - Fixed-function texture/raster units
  - High Programmability

Enter Reyes

- Developed in 1980s, offline
- Pixel-accurate smooth surfaces
- Eye-space shading
- Stochastic sampling
- Order-independent transparency

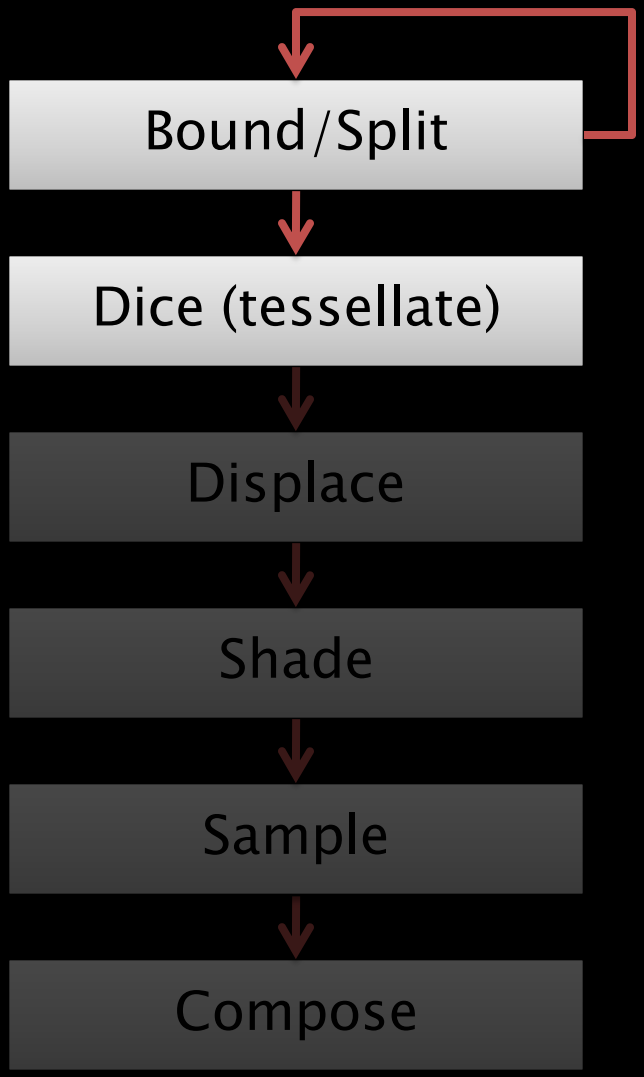

Reyes
Bound/Split
Dice (tessellate)
Displace
Shade
Sample
Compose
Direct3D 10
Vertex Shade
Geometry Shade
Rasterize
Fragment Shade
Depth-buffer
Blend

Reyes



- High-quality geometry
- Convenient surface shading
- Cinematic quality
- Well-behaved (?)

Reyes



- Input:
  Smooth surfaces
- Output:
  0.5 x 0.5 pixel quads
  (micropolygons)

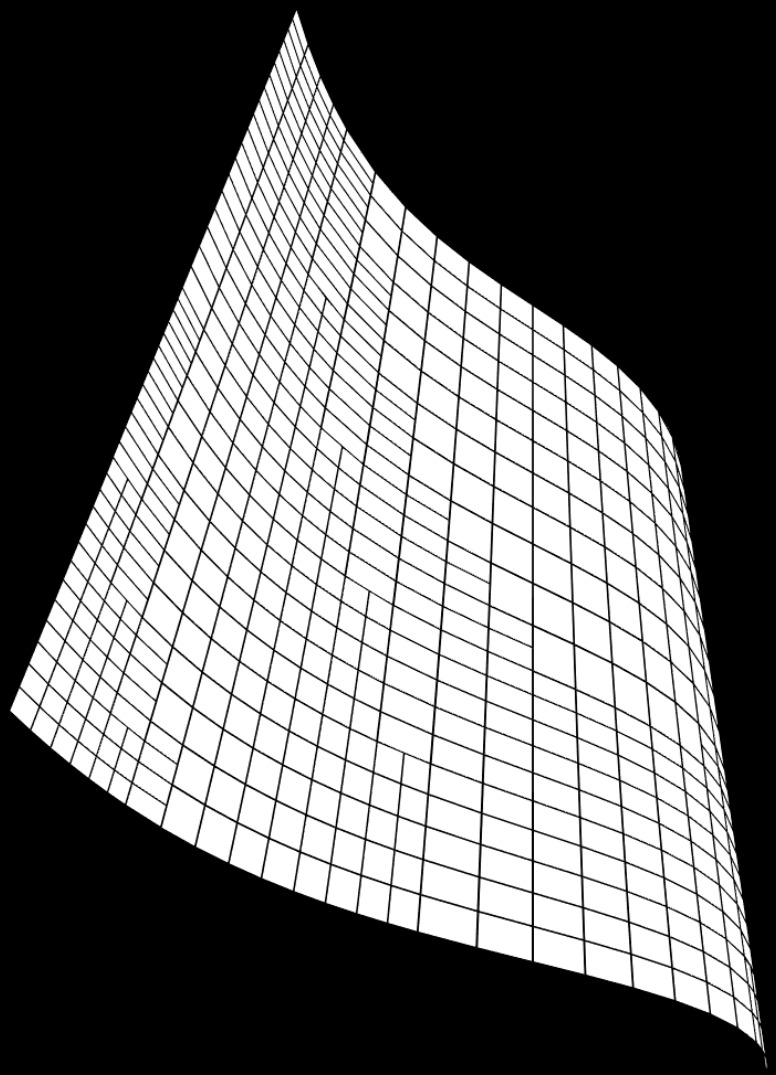
Outline

- Reyes Subdivision – algorithm
- Subdivision on GPU – implementation
- Results
- Limitations

Outline

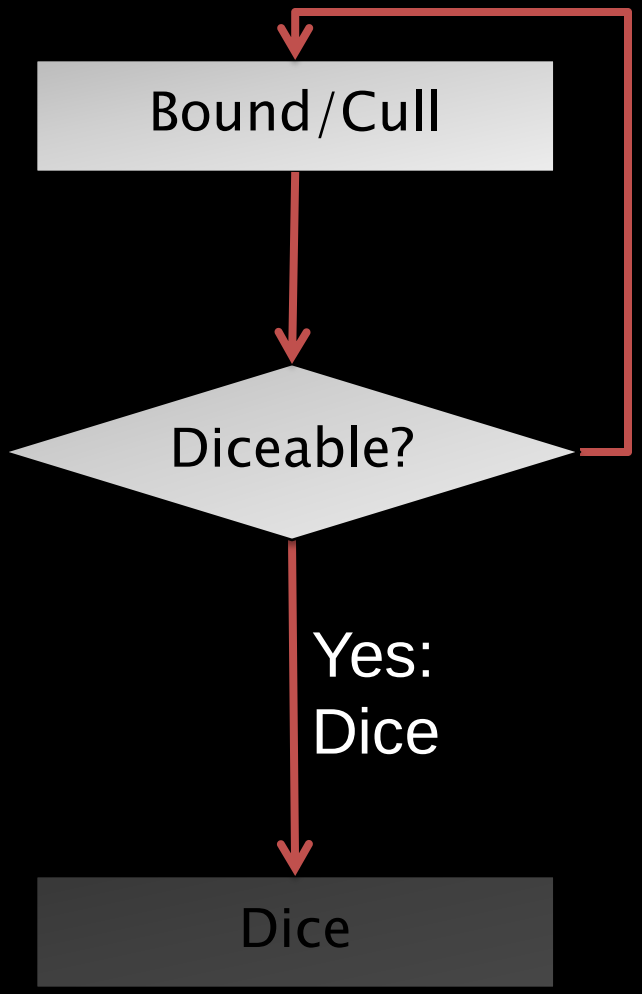
- Reyes Subdivision – algorithm
- Subdivision on GPU – implementation
- Results
- Limitations

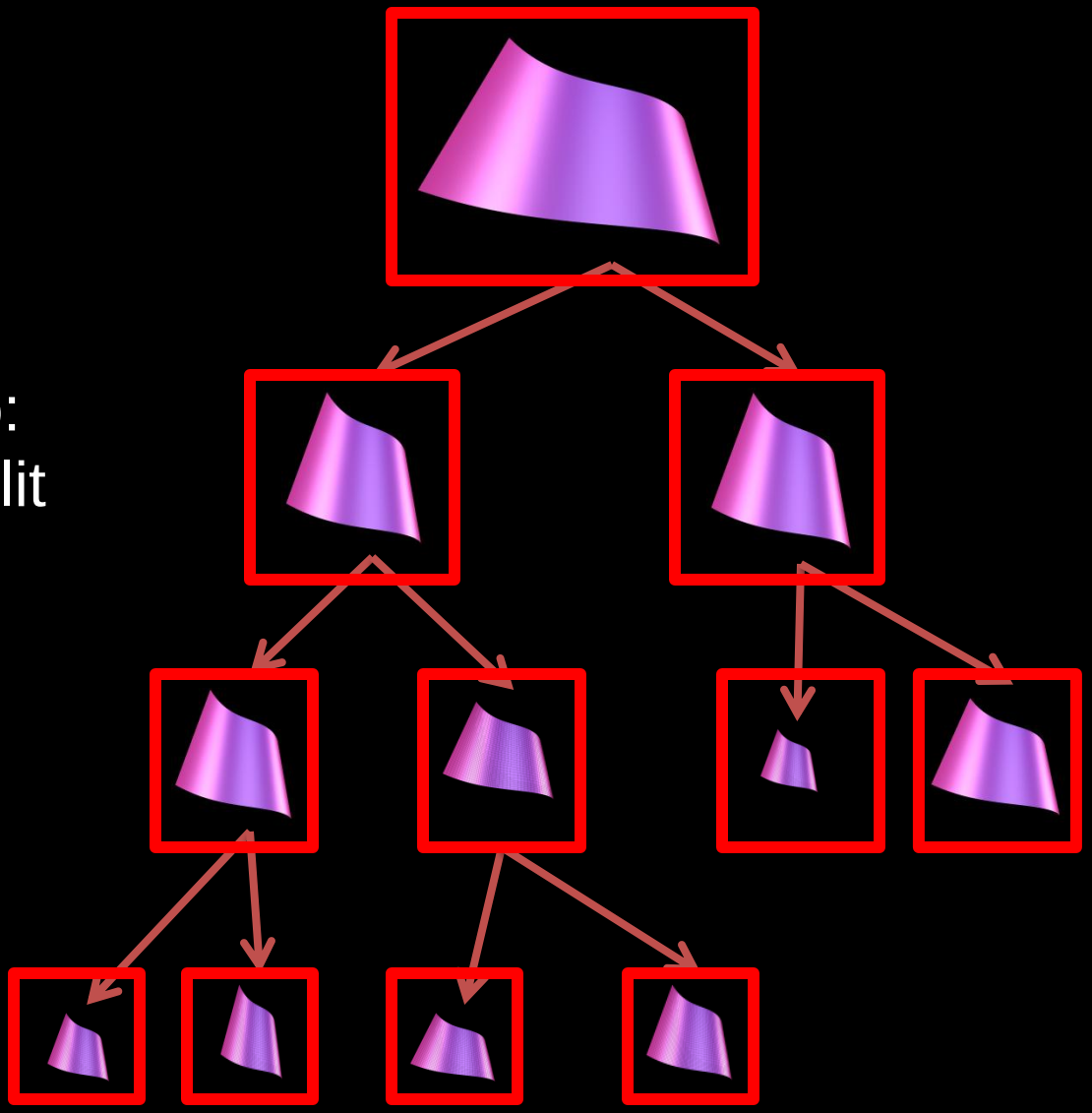
Split-Dice



- Recursively split a surface till dicing makes sense
- Uniformly sample it to form a grid of micropolygons

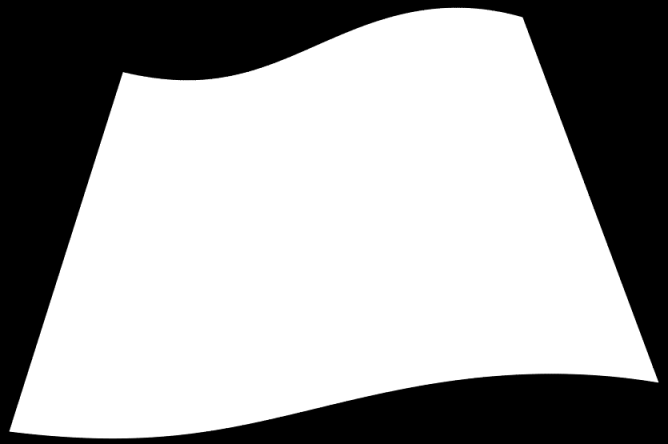
Split



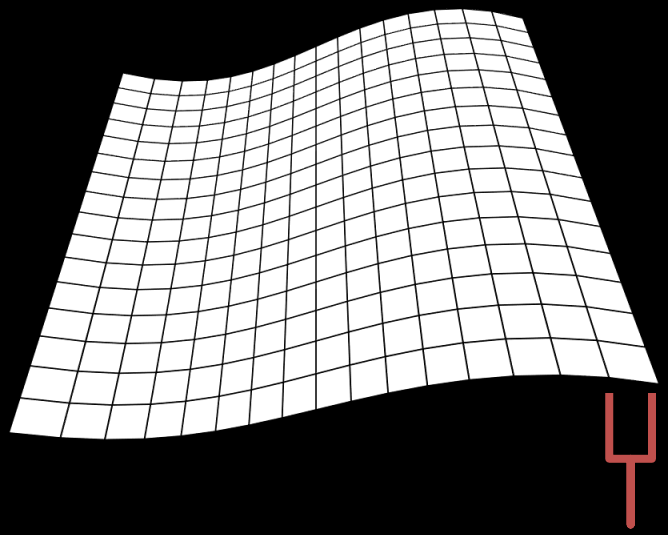


Dice

Post-split surface



Grid

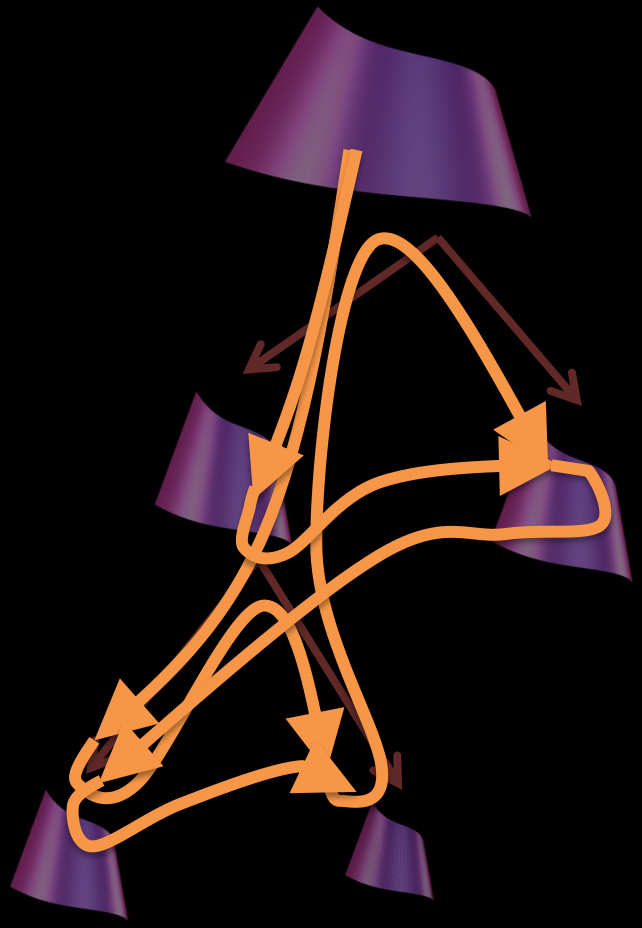


Micropolygon

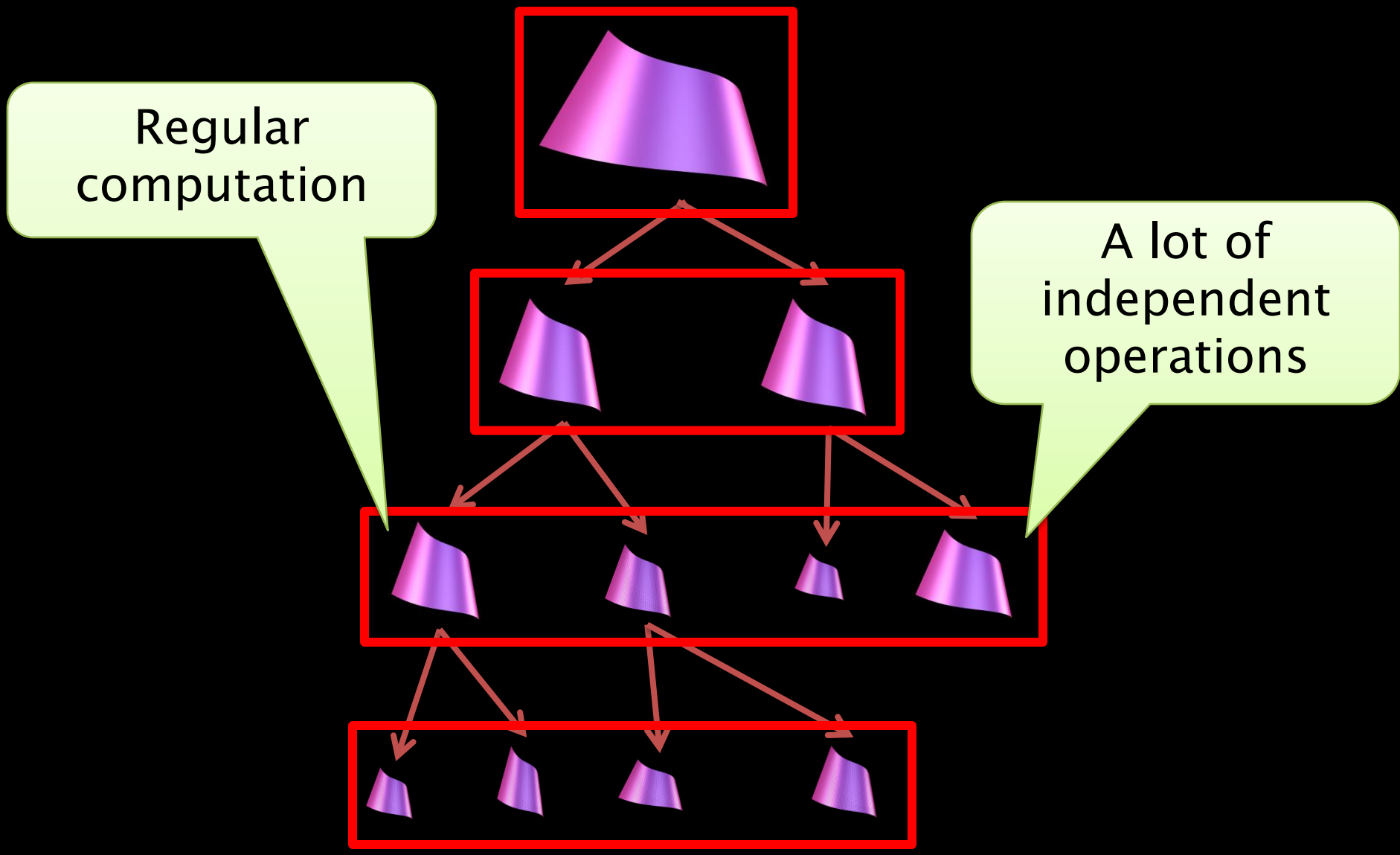
Split-Dice is hard

- Split
  - Recursive, serial
  - Rapid primitive generation/destruction
- Dice
  - Huge memory demand

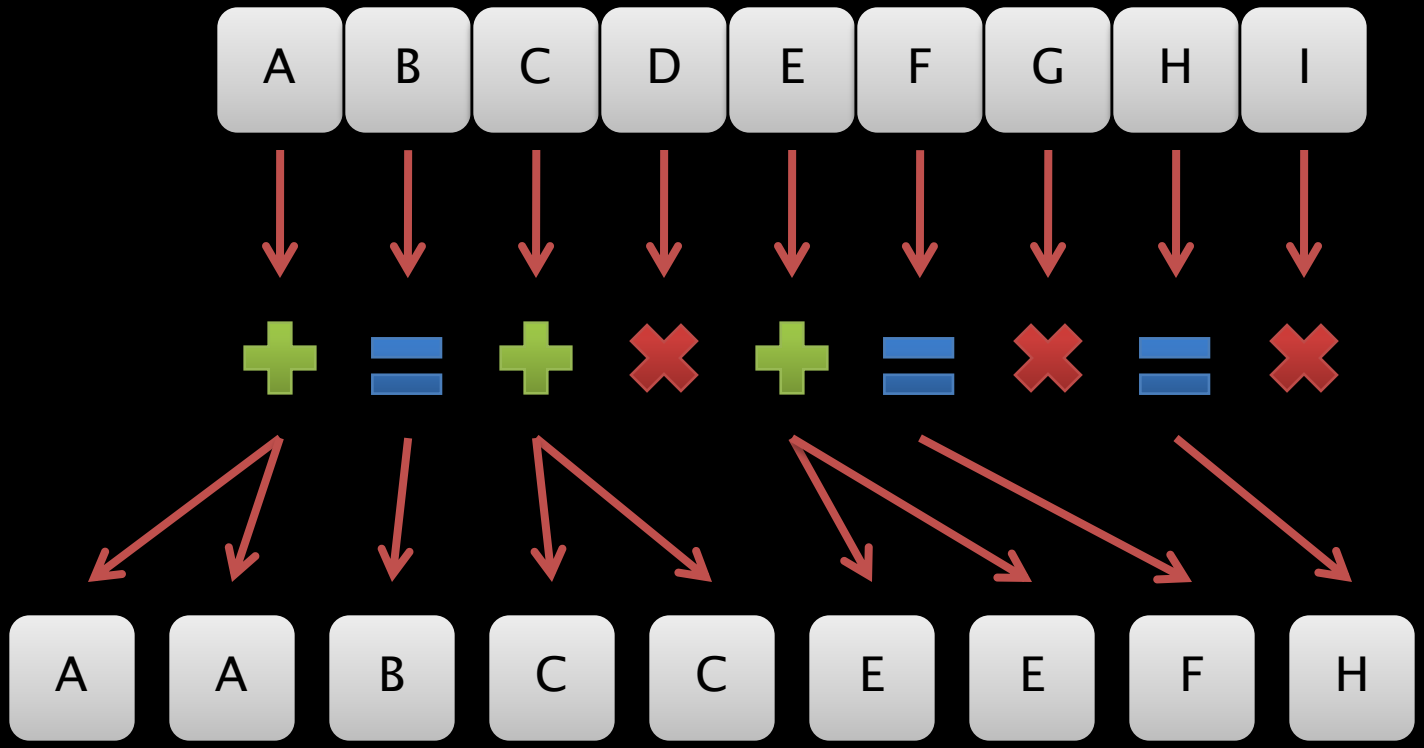
Can we do this in parallel?



Parallel Split



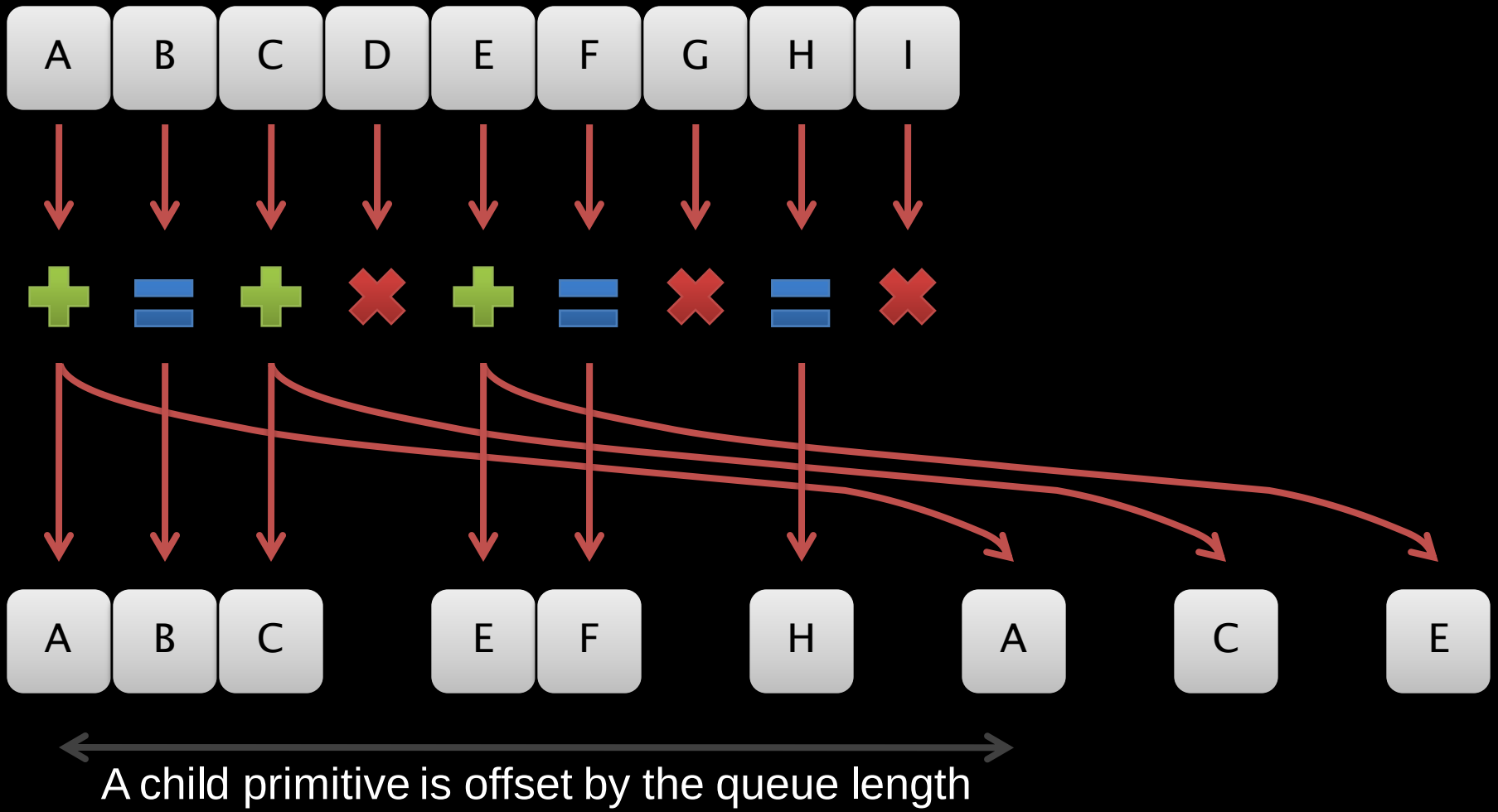
Analogy: A Dynamic work queue



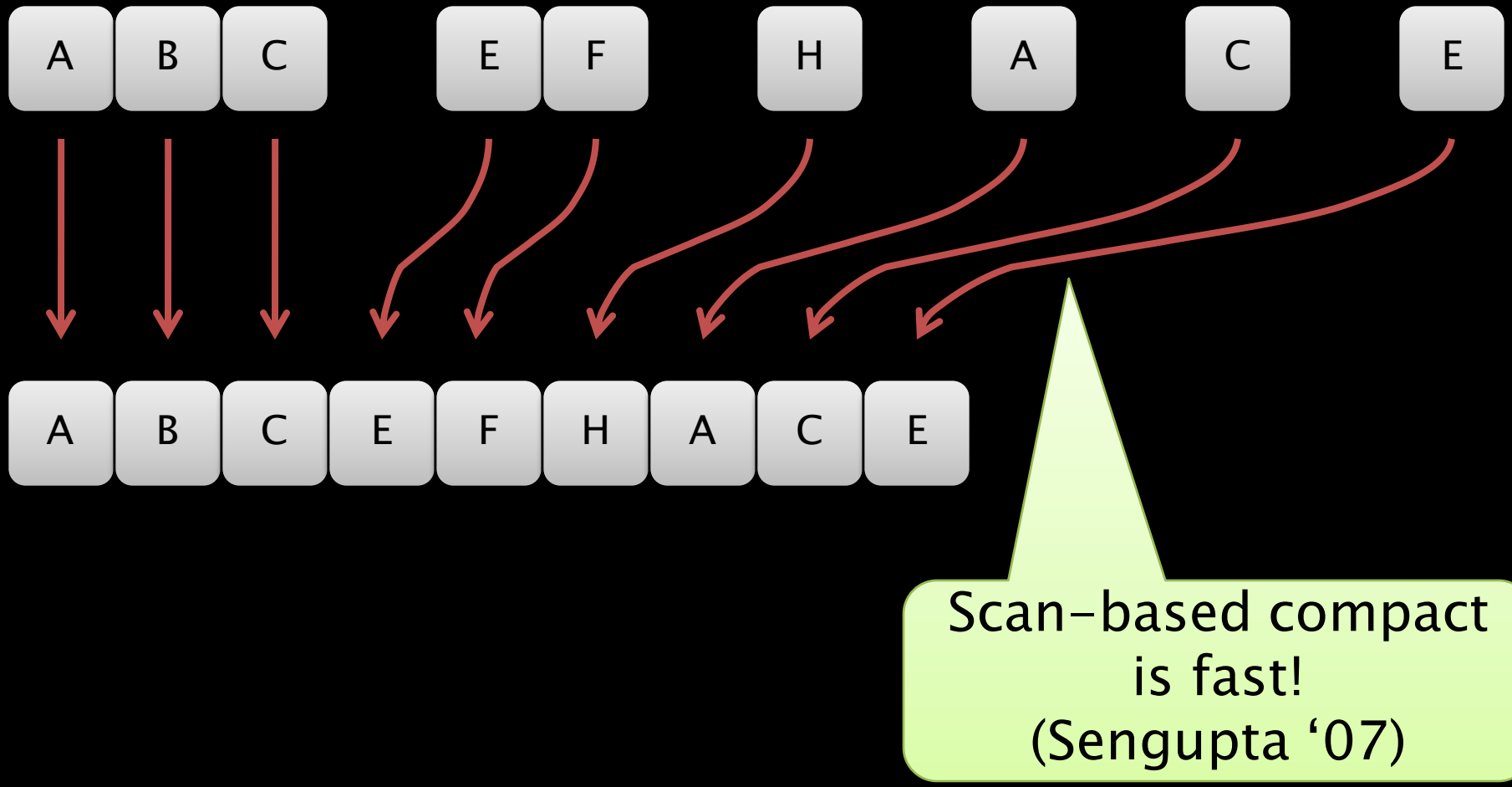
How can we do these efficiently?

- Creating new primitives
  - How to dynamically allocate space?
- Culling unneeded primitives
  - How to avoid fragmentation?

Our Choice – keep it simple…



…and get rid of the holes later



Storage Issues for Dice

- Too many micropolygons
  - Cannot reject early
- Screen-space buckets (tiles)
  - In parallel ?
  - Ideal bucket size ?

Outline

- Reyes Subdivision – algorithm
- Subdivision on GPU – implementation
- Results
- Limitations

Platform

- NVIDIA GeForce 8800 GTX
  - 16 SIMD Multiprocessors
  - 16KB shared memory
  - 768 MB memory (no cache)
- NVIDIA CUDA 1.1
  - Grids/Blocks/Threads
  - OpenGL interface

Implementation Details

- Input choice: Bicubic Bézier Surfaces
  - Only affects implementation
- View Dependent Subdivision every frame
  - Single CPU-GPU transfer
- Final micropolygons written to a VBO
  - Flat-shaded and displayed

Kernels Implemented

- Dice
  - Regular, symmetric, parallel
  - 256 threads per patch
  - Primitive information in shared memory
- Bound/Split
  - Hard to ensure efficiency

Bound/Split: Efficiency Goals

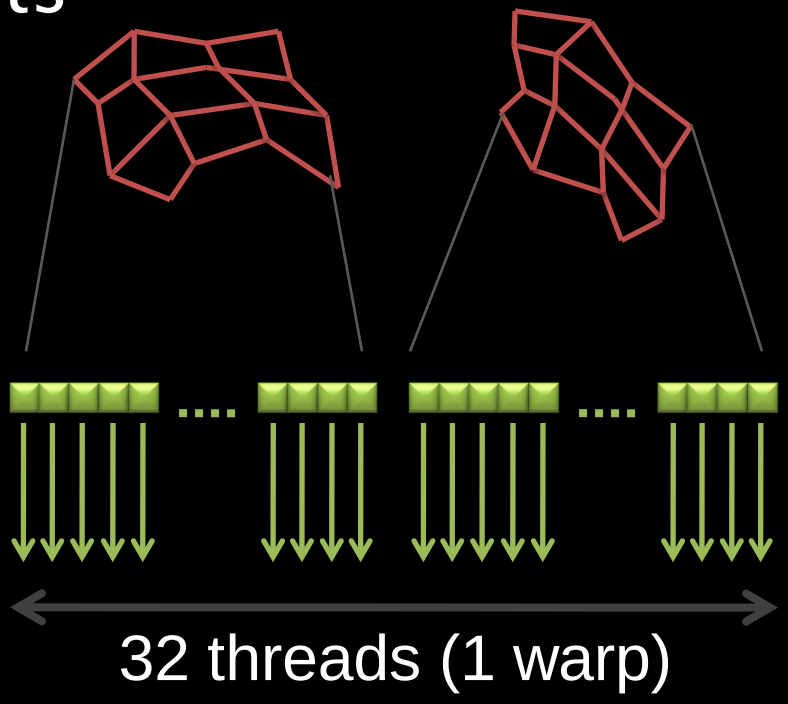
- Memory Coherence
  - Off-chip memory accesses must be efficient
- Computational Efficiency
  - Hardware SIMD must be maximally utilized

Memory Coherence during Split

- Compact work-queue after each iteration
  - Primitives always contiguous in memory
- Structure-Of-Arrays representation
  - Primitive attributes adjacent in memory
- 99.5% of all accesses fully coalesced

SIMD utilization during split

- Intra-Primitive parallelism
  - Independent control points
  - Negligible divergence



- Vectorized Split
  - 16 Threads per patch

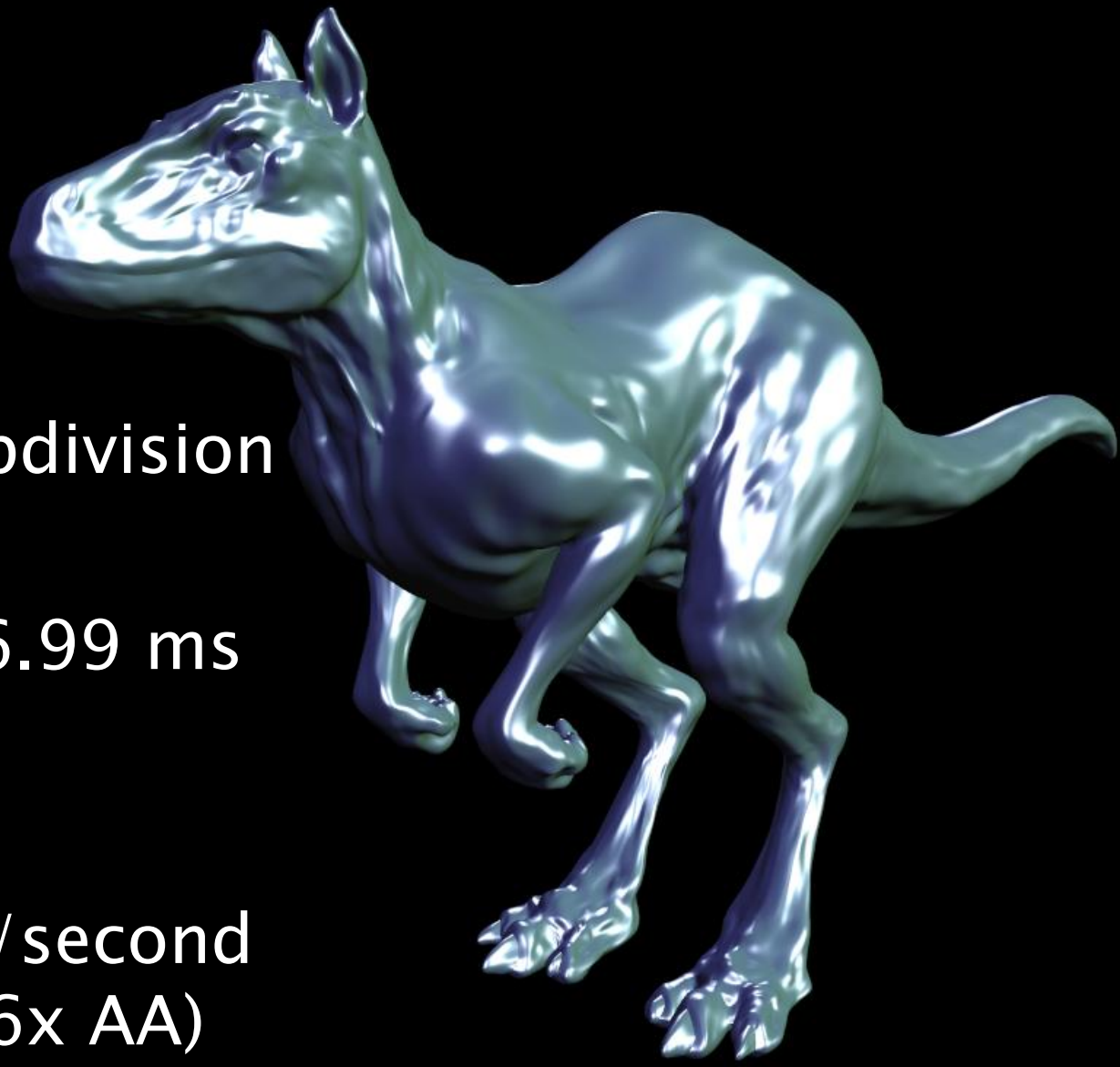
- 90.16% of all branches SIMD coherent

Outline

- Reyes Subdivision – algorithm
- Subdivision on GPU – implementation
- Results
- Limitations

Results – Killeroo

- 14426 grids
- 5 levels of subdivision
- Bound/Split: 6.99 ms
- Dice: 7.21ms
- 29.69 frames/second (19.92 with 16x AA)



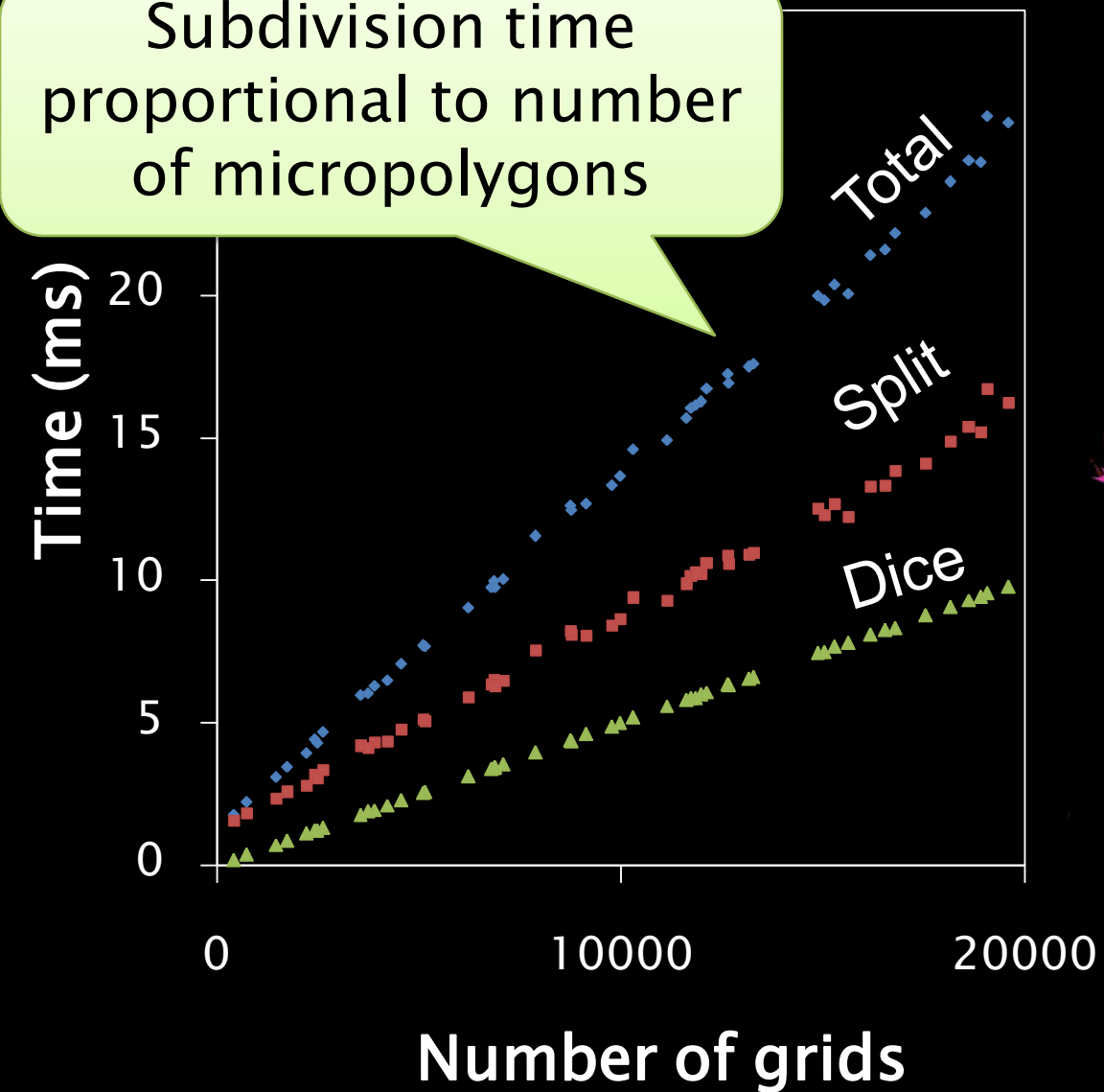
Killeroo model courtesy: Headus Inc.

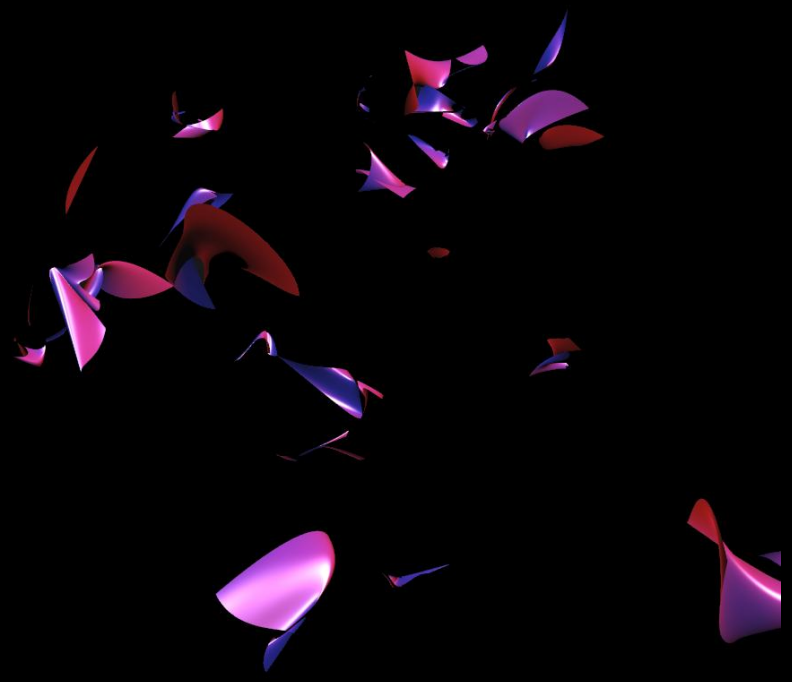
Results – Teapot

- 4823 grids
- 11 levels of subdivision
- Bound/Split: 3.46 ms
- Dice: 2.42 ms
- 60.07 frames/second (30.02 with 16x AA)

Results – Random scenes

Subdivision time proportional to number of micropolygons





Render time Breakdown

Rendering lots of small triangles

Rendering

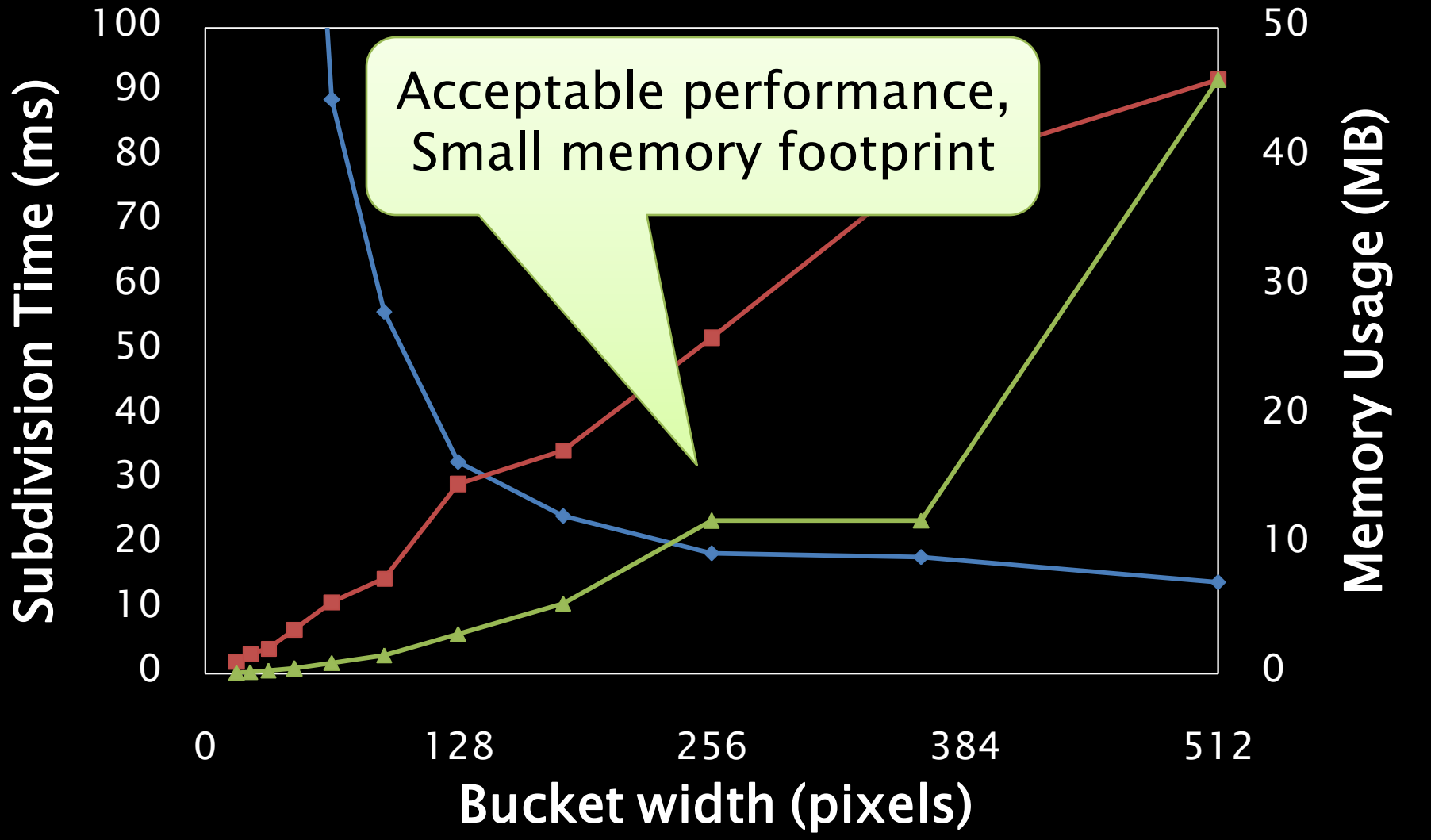
CUDA-OpenGL transfer

Normal generation

Subdivision

Should ideally be zero: CUDA limitation

Results – Screen-space buckets



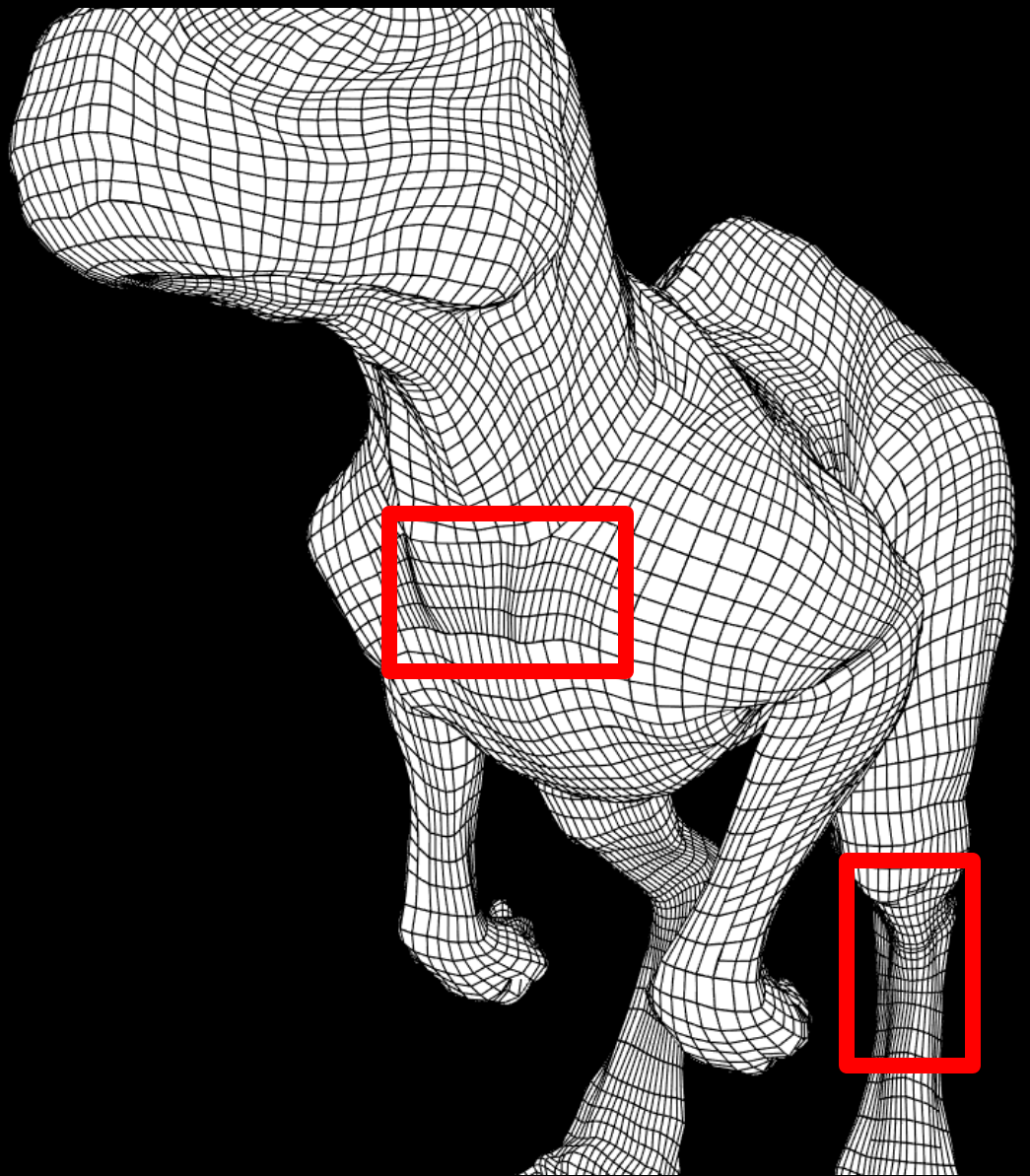
Outline

- Reyes Subdivision – algorithm
- Subdivision on GPU – implementation
- Results
- Limitations

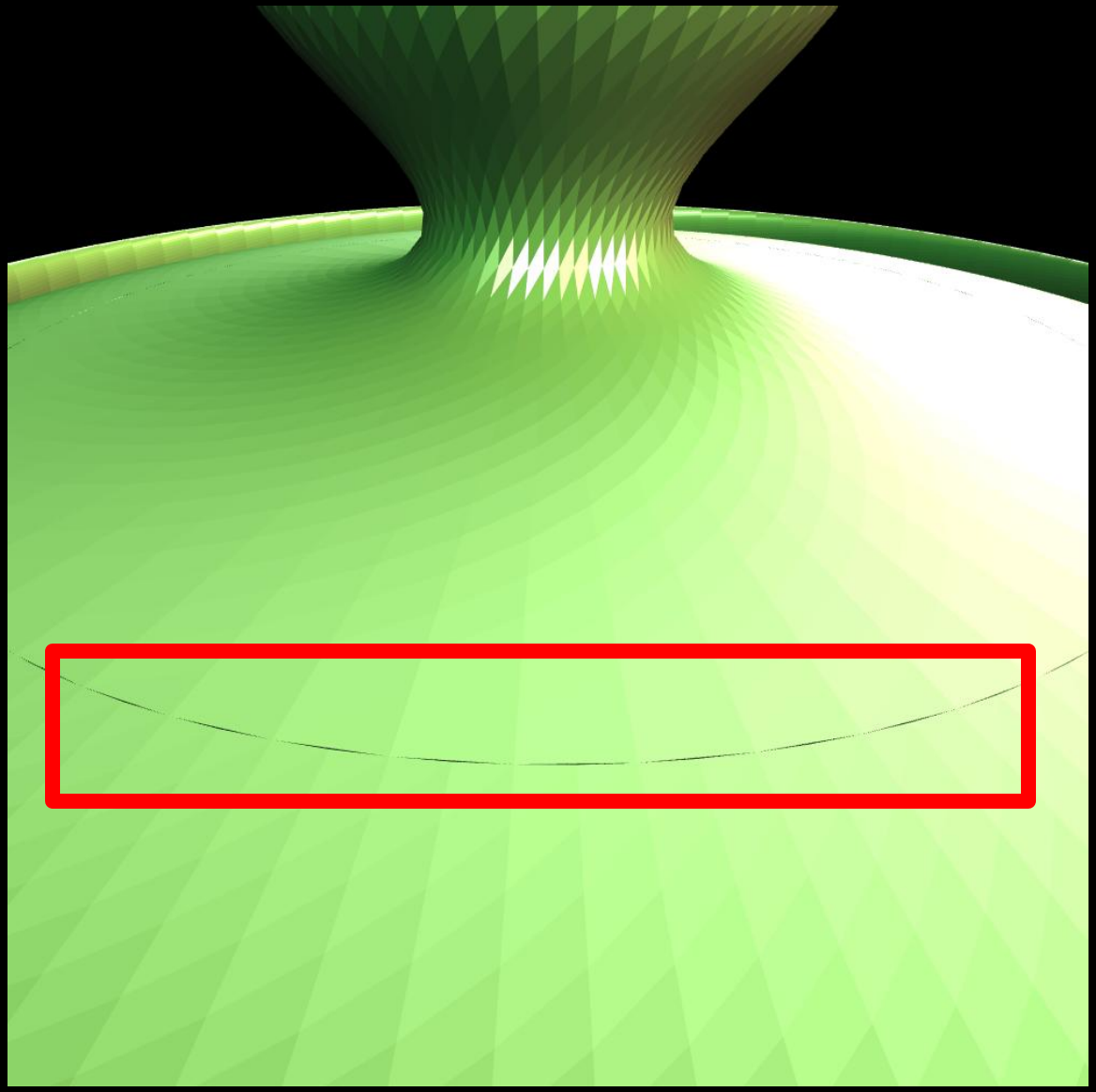
Limitations

- Can’t split and dice in parallel
- Uniform dicing
- Cracks

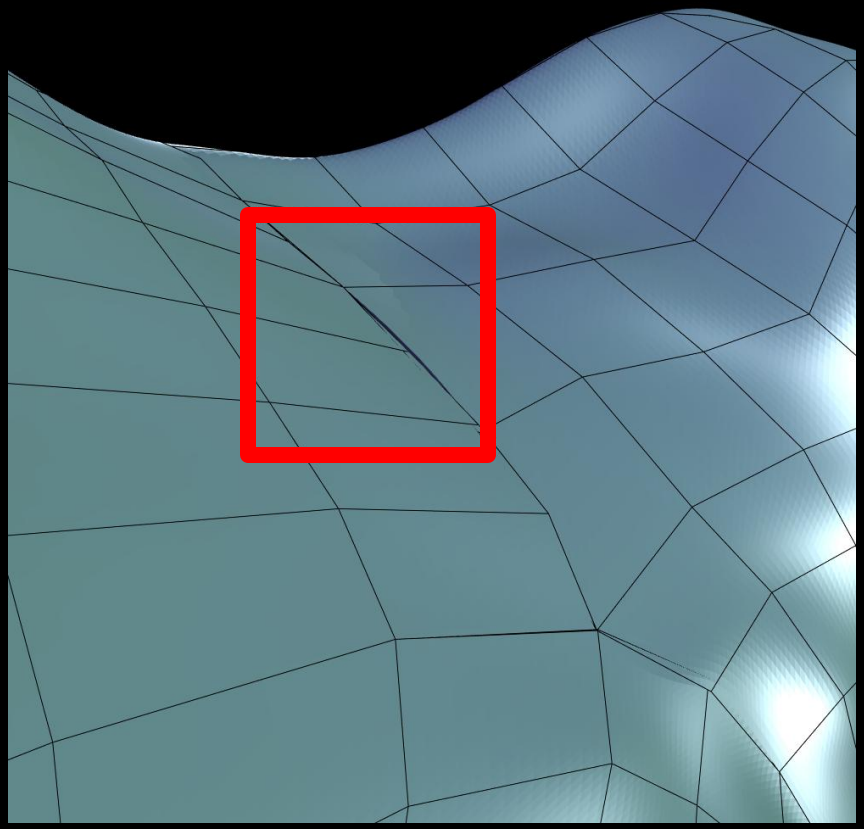
Limitations – Uniform dicing

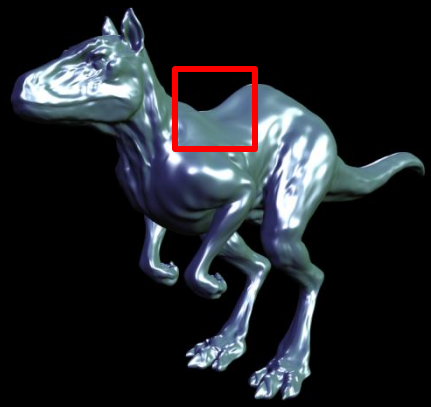


Limitations – Cracks



Limitations – Cracks





Conclusions

- Breadth-first recursive subdivision
  - Suits GPUs
  - Works fast
- ~~Dicing~~ Programmable tessellation is fast
  - 500M micropolygons/sec
- It is time to experiment with alternate graphics pipelines

Reyes in real-time rendering

- Visually superior to polygon pipeline
- Regular, highly parallel workload
- Extremely well-studied
- Good candidate for a real-time system?

Future Work

- Cracks, Displacement mapping
- Rest of Reyes
  - Offline quality shading
  - Interactive lighting
  - Parallel Stochastic Sampling (Wei 2008)
  - A-buffer (Myers 2007)

Thanks to

- Feedback and suggestions
  - Per Christensen, Charles Loop, Dave Luebke, Matt Pharr and Daniel Wexler

- Financial support
  - DOE Early Career PI Award
  - National Science Foundation
  - SciDAC Institute for Ultrascale Visualization

- Equipment support from NVIDIA

Teapot Video

Real-time footage

Killeroo Video

Real-time footage

Real-Time Reyes-Style Adaptive Surface Subdivision

BACKUP SLIDES

CUDA Thread Structure

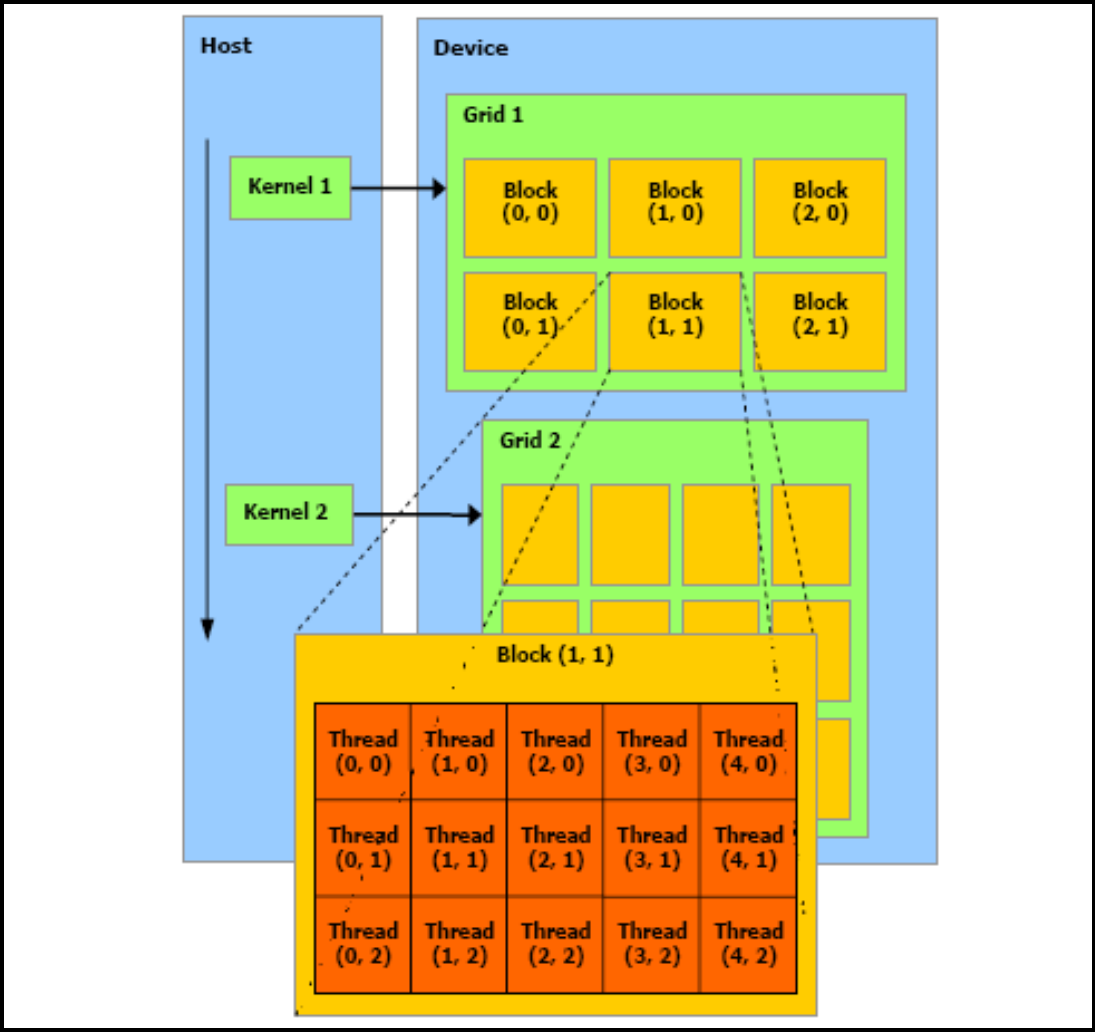


Image courtesy: NVIDIA CUDA Programming Guide, 1.1

CUDA Memory Architecture

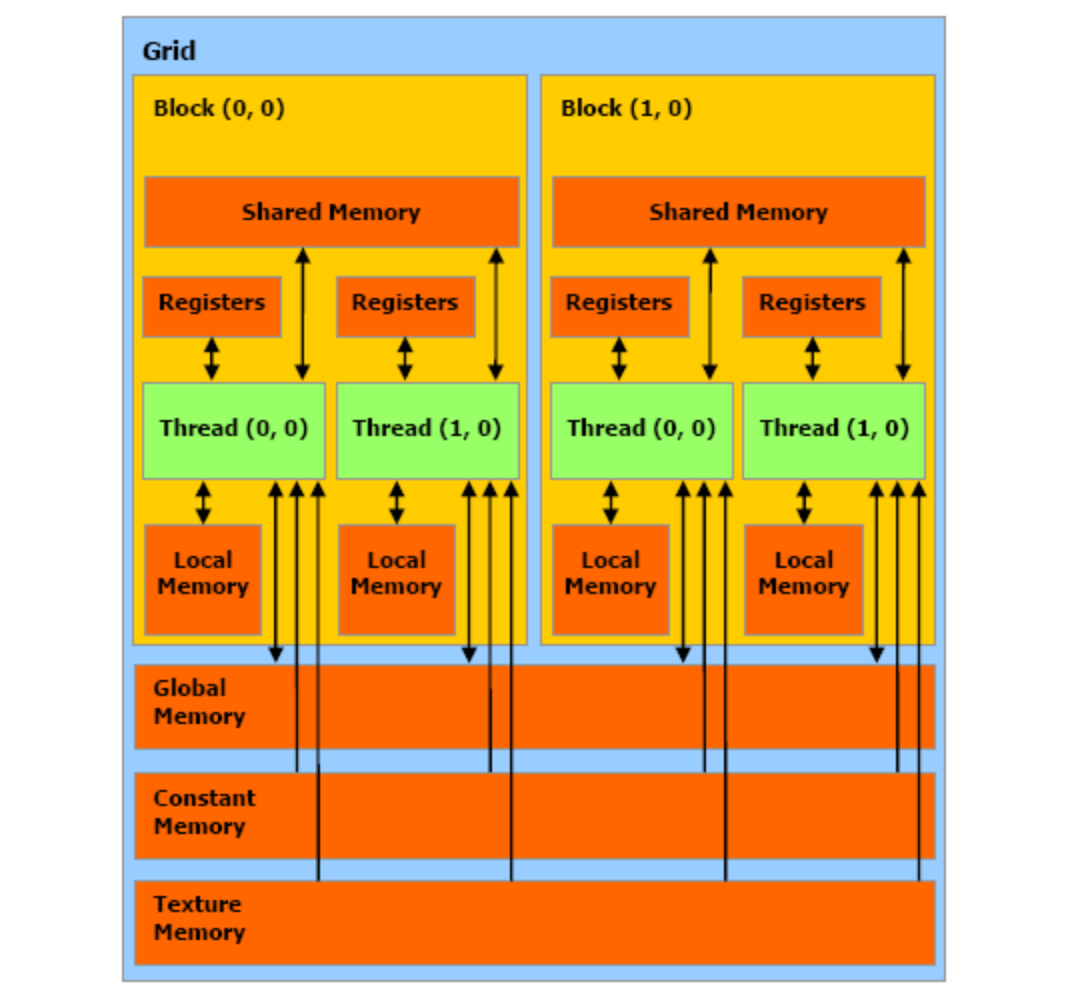


Image courtesy: NVIDIA CUDA Programming Guide, 1.1

Displacement Mapping

- Fairly simple if cracks can be avoided
  - Displace adjacent grids together

Shading & Lighting

- Interactive preview
  - Lpics / Lightspeed
- Shaders
  - Intelligent textures
  - File access
- Shadows

Composite/Filter Stuff

- Stochastic sampling
  - 20x speedup on a GPU (Wei 2008)
- A-buffer

Special Effects

- Motion Blur and DOF
- Global Illumination
- Ambient Occlusion