Real-Time Reyes-Style Adaptive Surface Subdivision

Anjul Patney, John Owens
University of California, Davis

Offline Rendering



- Looks realistic
- Virtually no visible artifacts
- Renders on clusters of CPUs
  - Slow: hours per frame
  - Flexible rendering pipelines

Images courtesy: Pixar

Real-Time Rendering

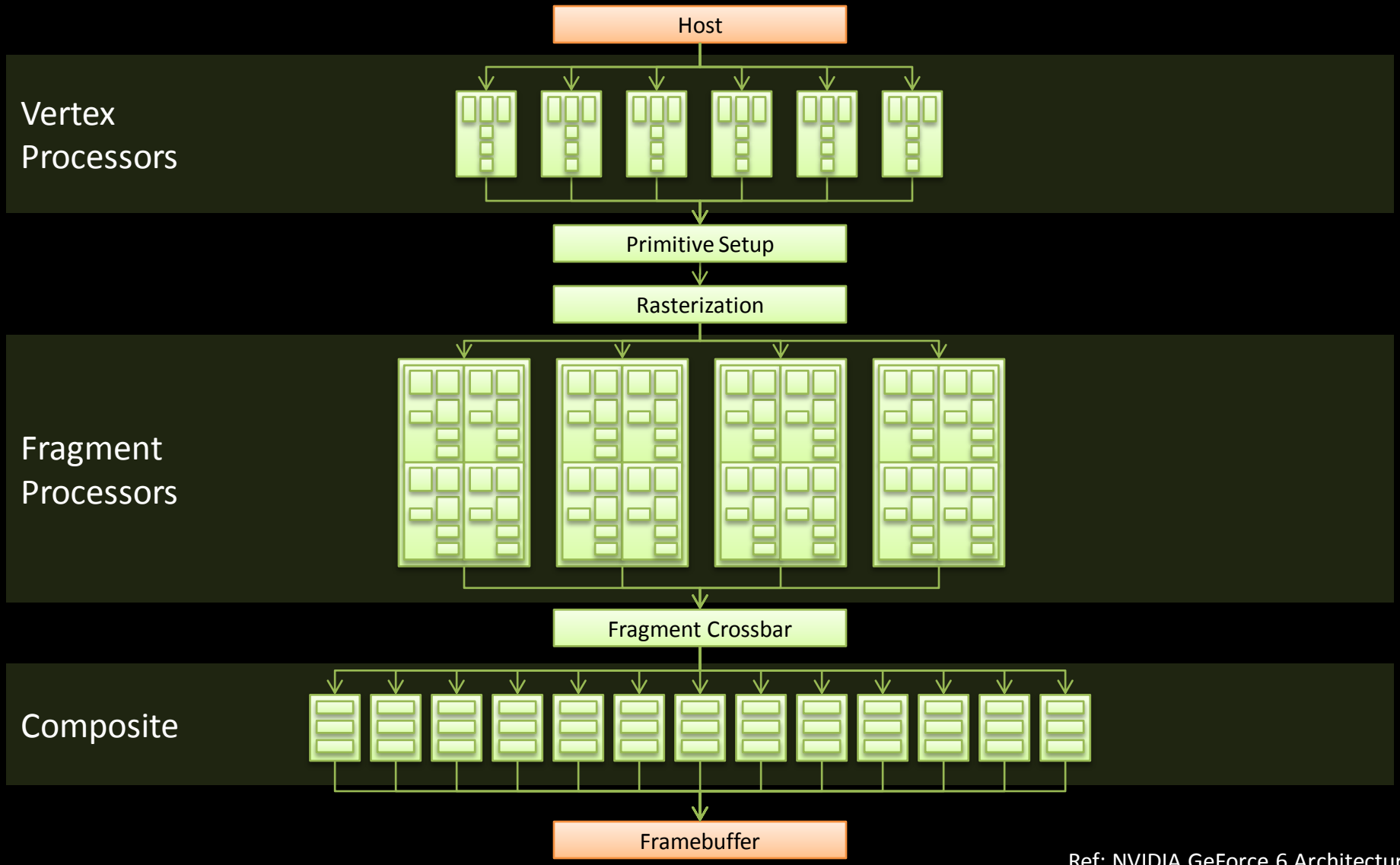
- Looks good enough
- Minor artifacts are OK
- Renders on commodity GPUs
  - Fast: 60+ frames per second
  - Restrictive rendering pipeline

Images courtesy: www.ign.com

There is a gap

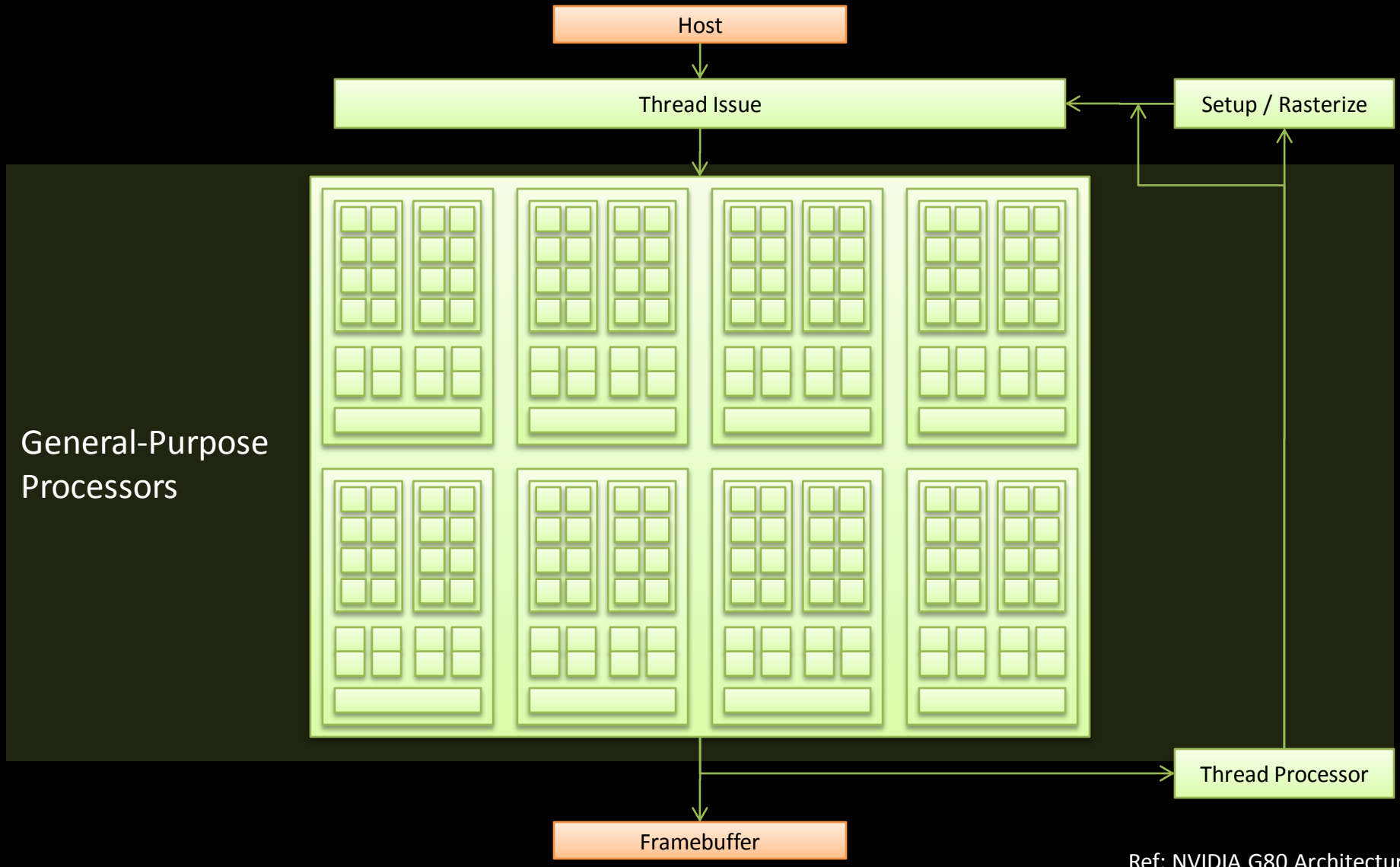
- Geometric complexity
  - From Polygon Meshes to Smooth Surfaces
- Shading complexity
  - From HLSL to RenderMan shaders
- Special Effects
  - Motion Blur, depth-of-field
- Other complex effects
  - Global Illumination, Subsurface scattering, ambient occlusion

But commodity GPUs…



Ref: NVIDIA GeForce 6 Architecture

… have changed a lot



Ref: NVIDIA G80 Architecture

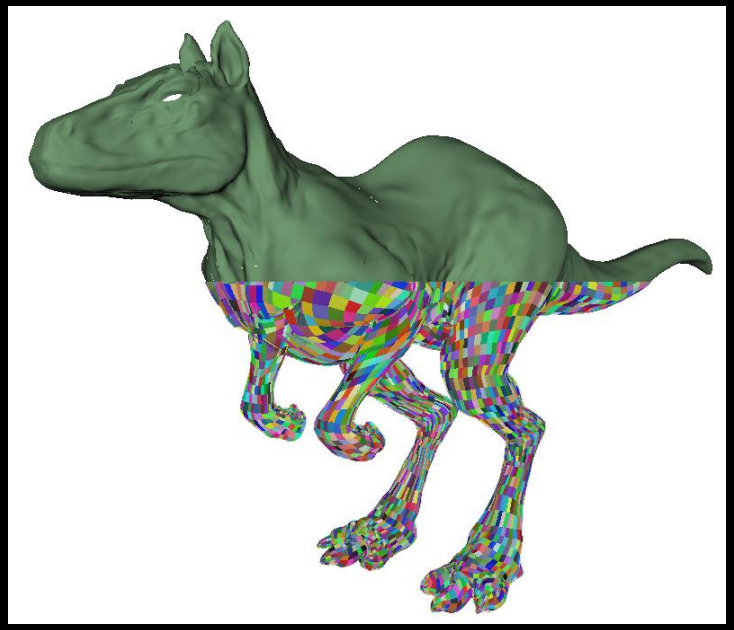
How does this affect things?

- Increased programmability
  - Arbitrary computation
  - Dynamic memory management
  - Irregular data structures
- Flexible Rendering
  - Compute for graphics
  - Offline quality in real-time ?
- But we must be careful

There is a gap

- Geometric complexity
  - From Polygon Meshes to Smooth Surfaces
- Shading complexity
  - From HLSL to RenderMan shaders
- Special Effects
  - Motion Blur, depth-of-field
- Other complex effects
  - Global Illumination, Subsurface scattering, ambient occlusion

Real-Time Reyes-Style Adaptive Surface Subdivision



Anjul Patney and John D. Owens

SIGGRAPH Asia 2008

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

Outline

- Motivation
- Reyes Subdivision – algorithm
  - Challenges
  - Parallel formulation
- Subdivision on GPU – implementation
  - Issues
  - Solutions
- Results

Outline

- Motivation
- Reyes Subdivision – algorithm
  - Challenges
  - Parallel formulation
- Subdivision on GPU – implementation
  - Issues
  - Solutions
- Results

Motivation

- Polygon-based Rendering is insufficient
  - Undesirable artifacts, especially along silhouettes
  - Complicated model representation
  - Model resolution is view-independent
- Can we expect performance from irregular computation on GPUs?
- Can GPUs support completely new pipelines?

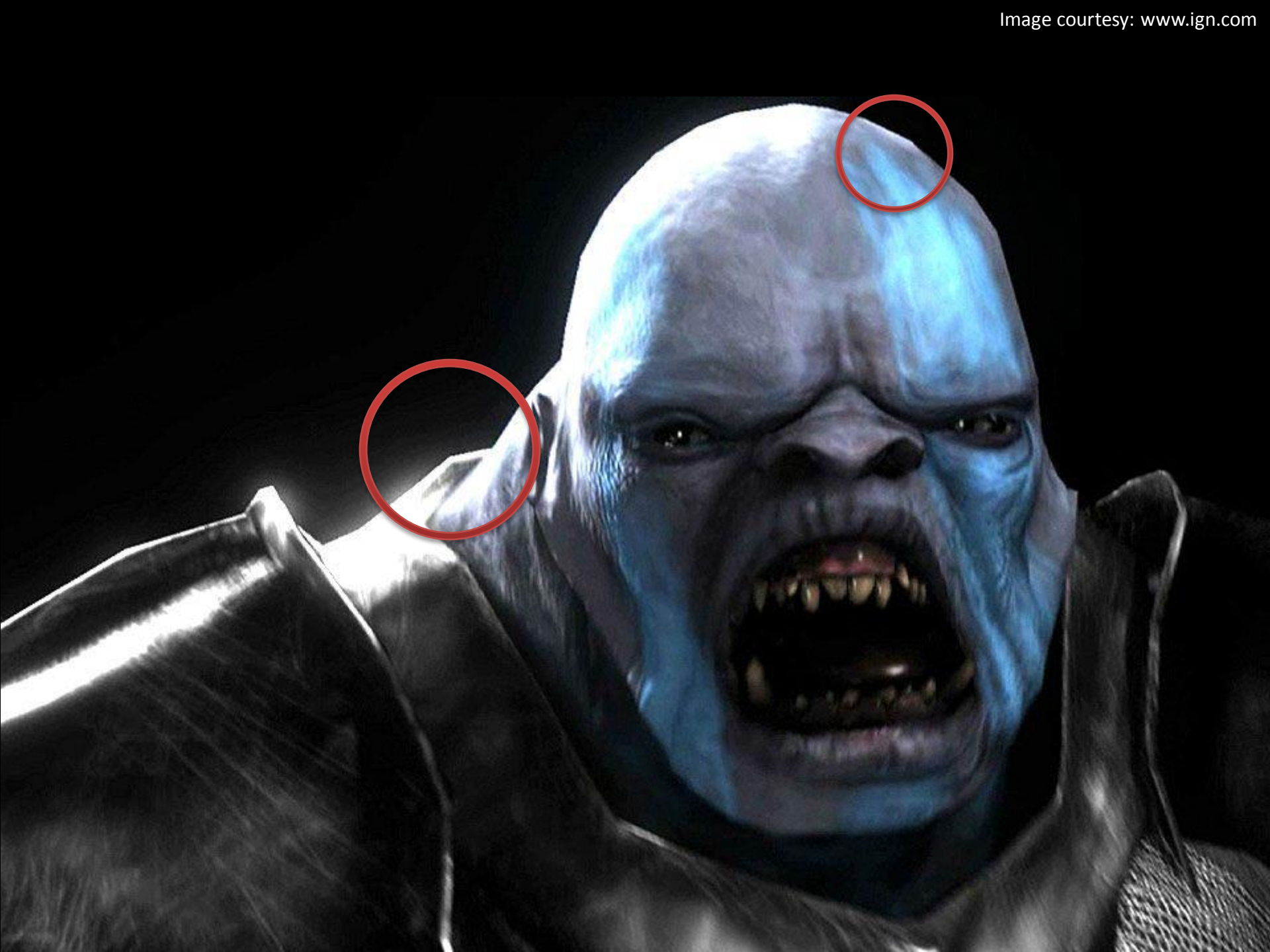
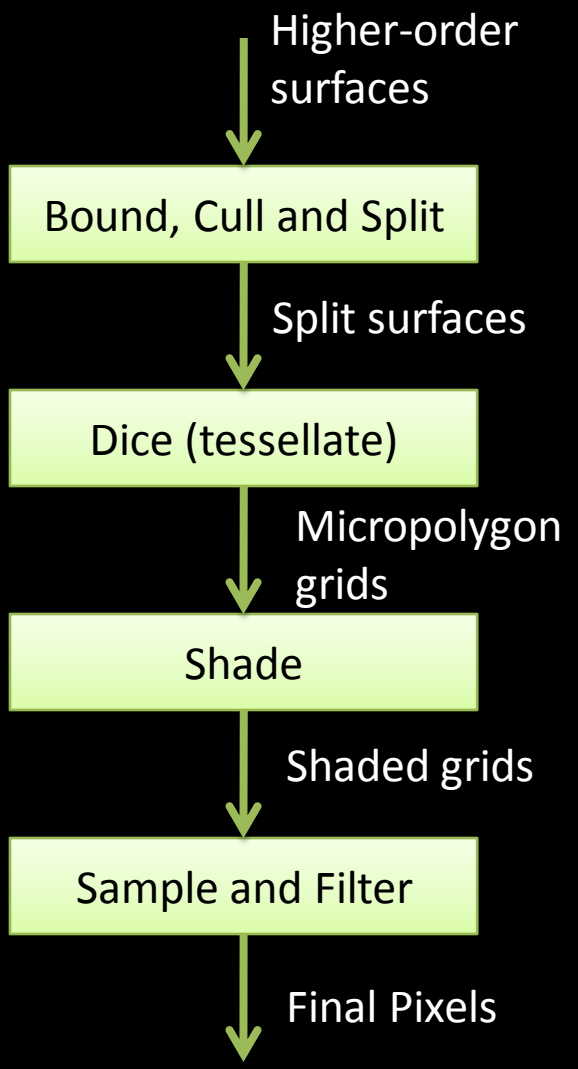


Image courtesy: www.ign.com

Enter Reyes

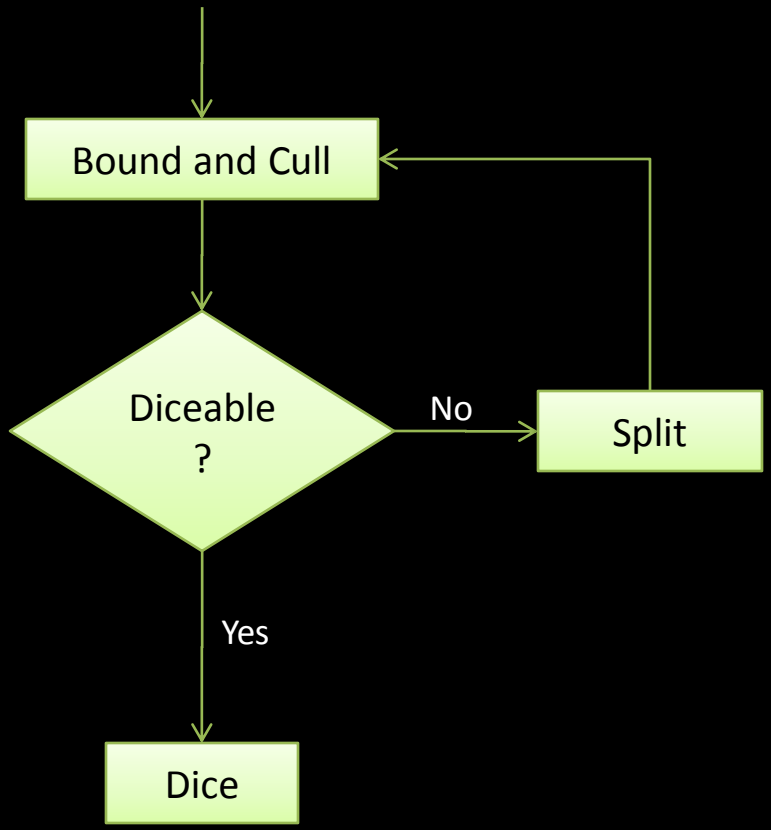
- Industry standard in high-quality rendering
- Forms the architecture beneath RenderMan
- Pipeline features
  - Input: Parametric Surfaces
  - Rendering primitive: 0.5 x 0.5 pixel micropolygons
  - Adaptive tessellation
  - Per-micropolygon programmable shading
  - Stochastic sampling

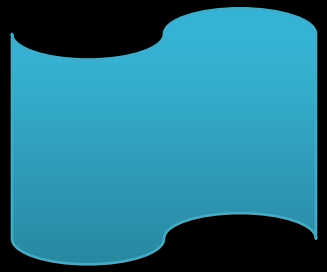


Outline

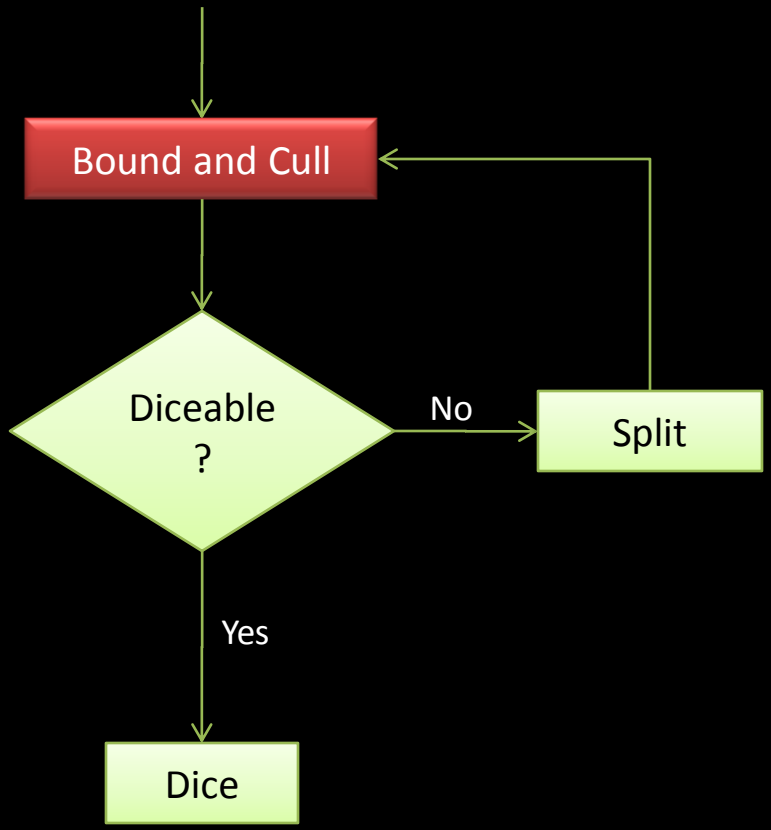
- Motivation
- Reyes Subdivision – algorithm
  - Challenges
  - Parallel formulation
- Subdivision on GPU – implementation
  - Issues
  - Solutions
- Results

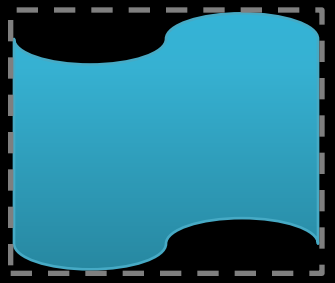
Reyes Subdivision



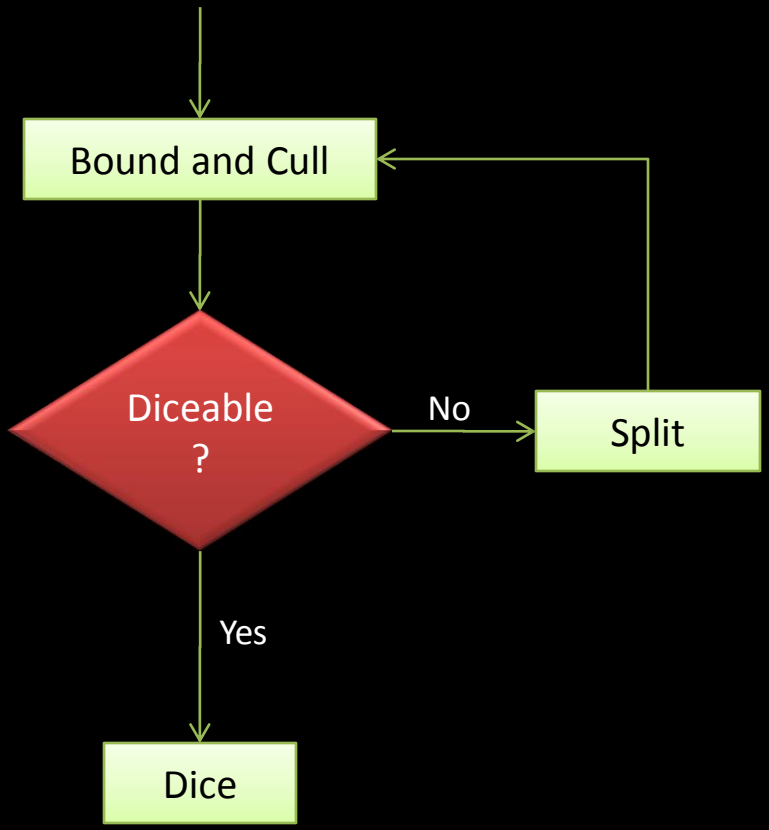


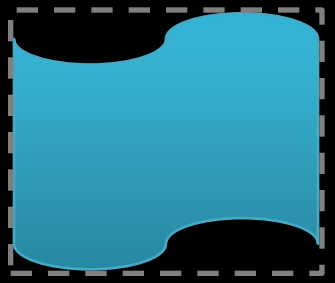
Reyes Subdivision



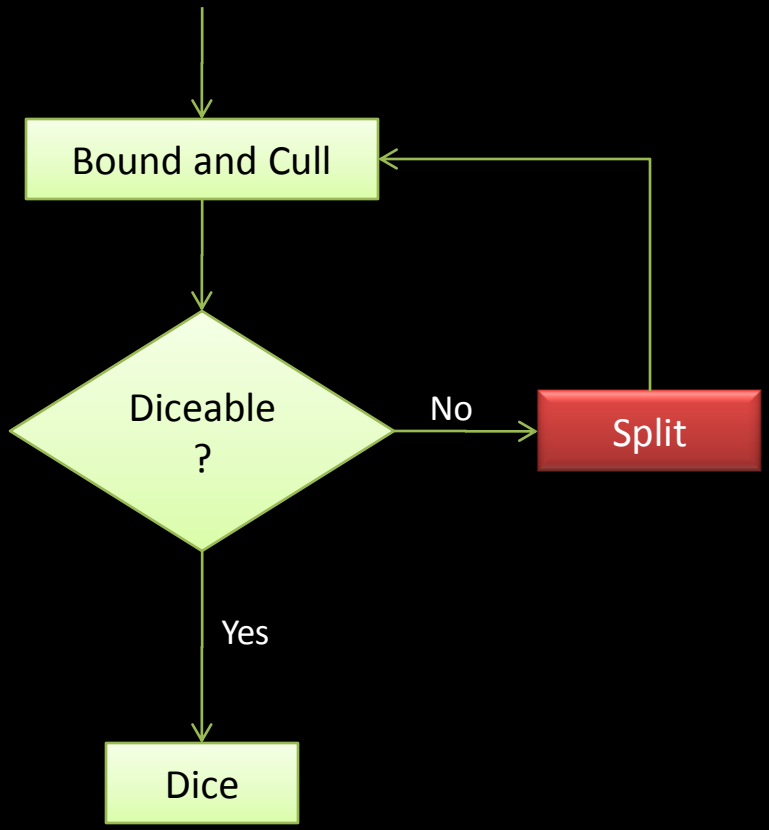


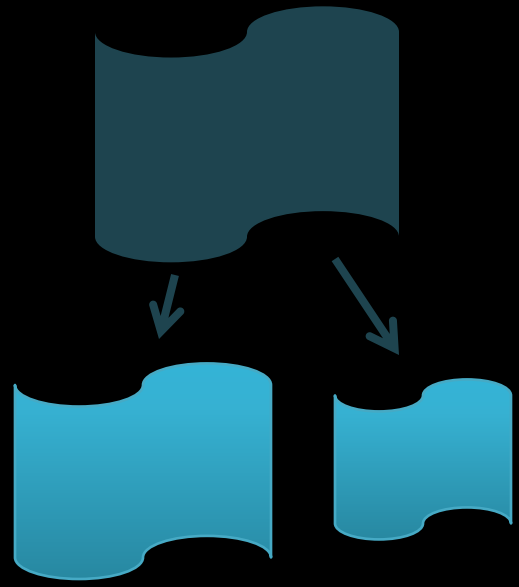
Reyes Subdivision



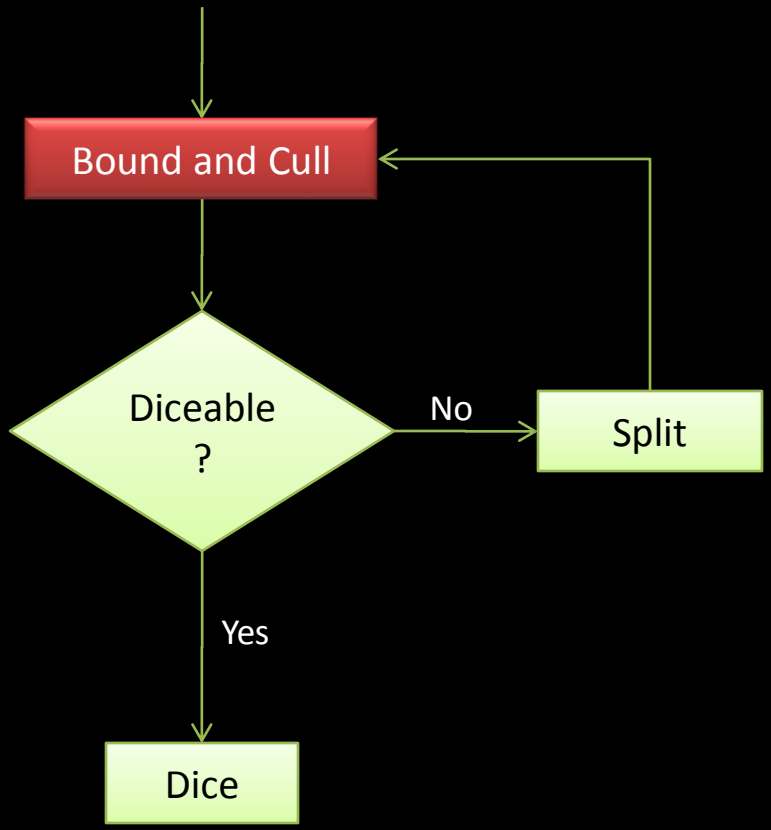


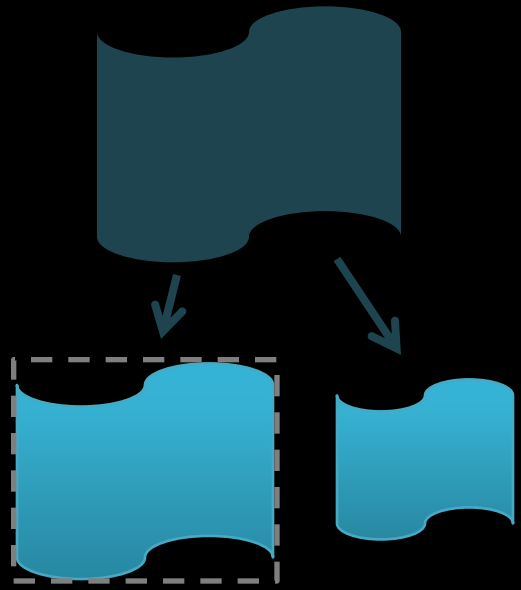
Reyes Subdivision



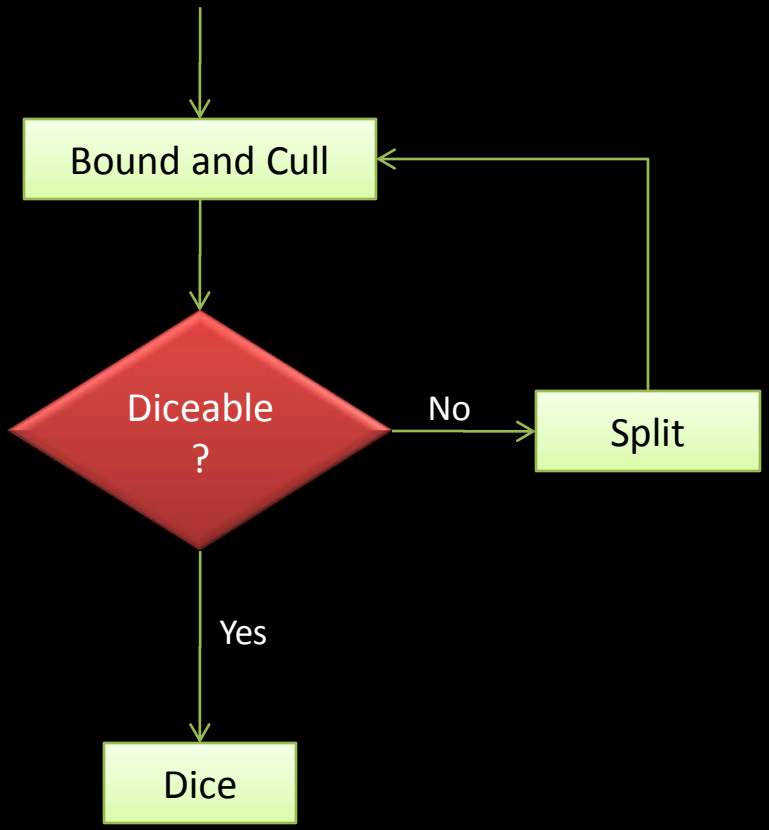


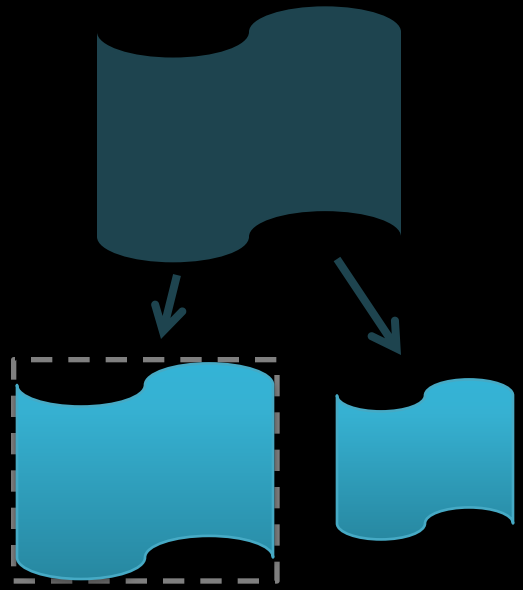
Reyes Subdivision



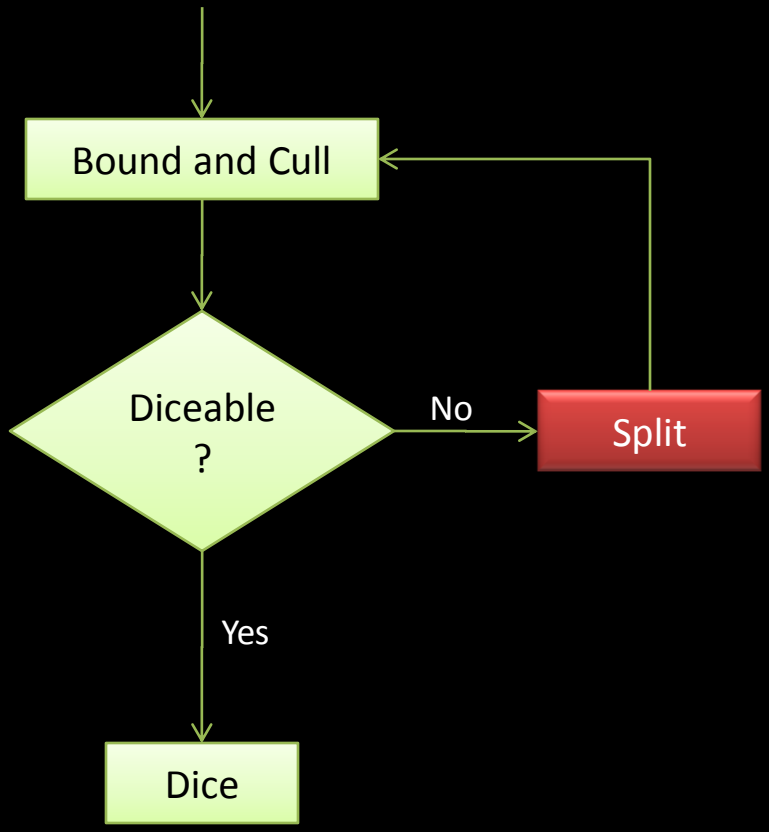


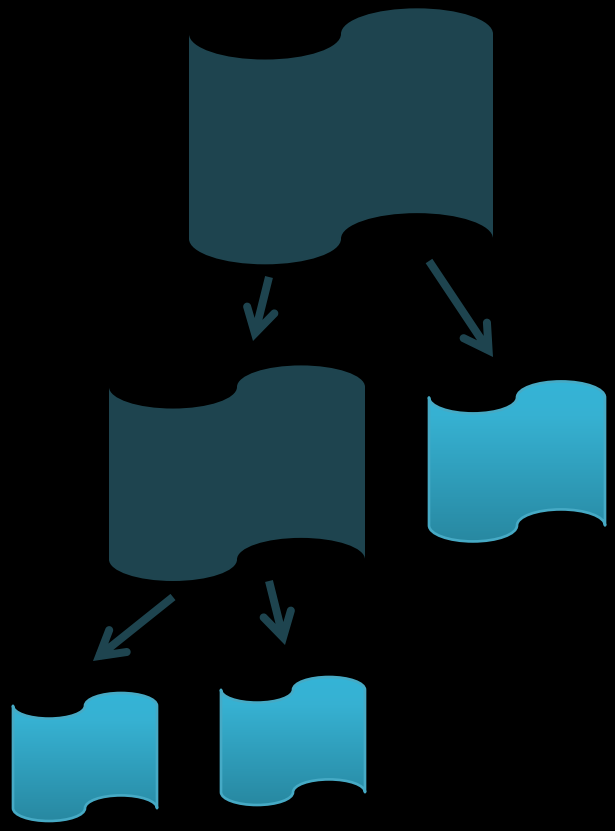
Reyes Subdivision



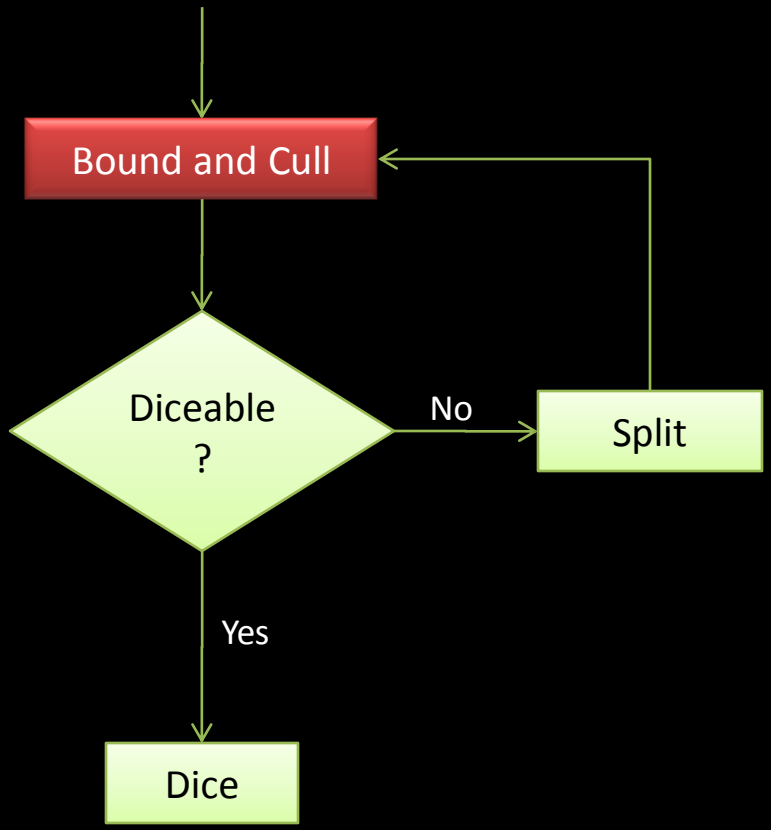


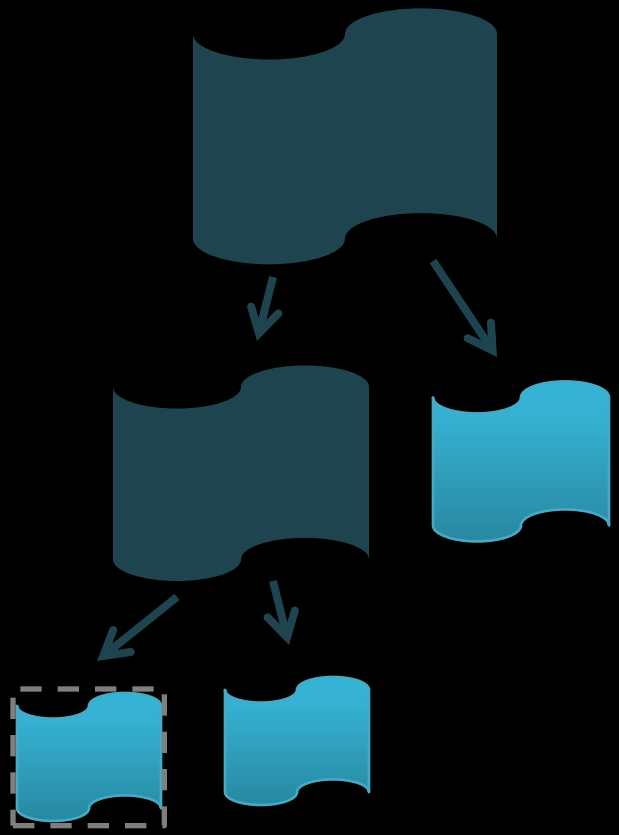
Reyes Subdivision



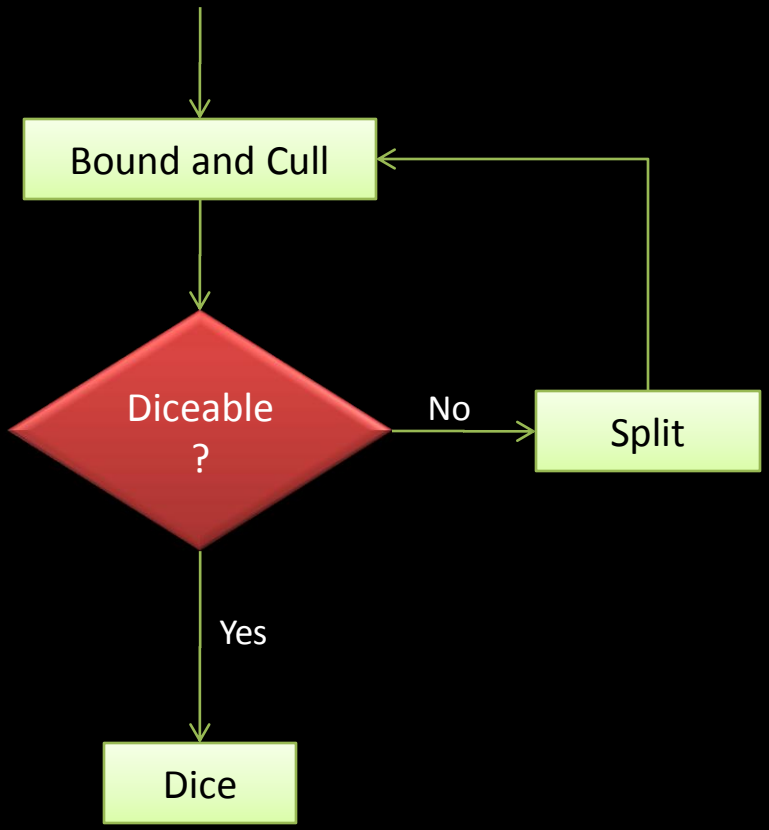


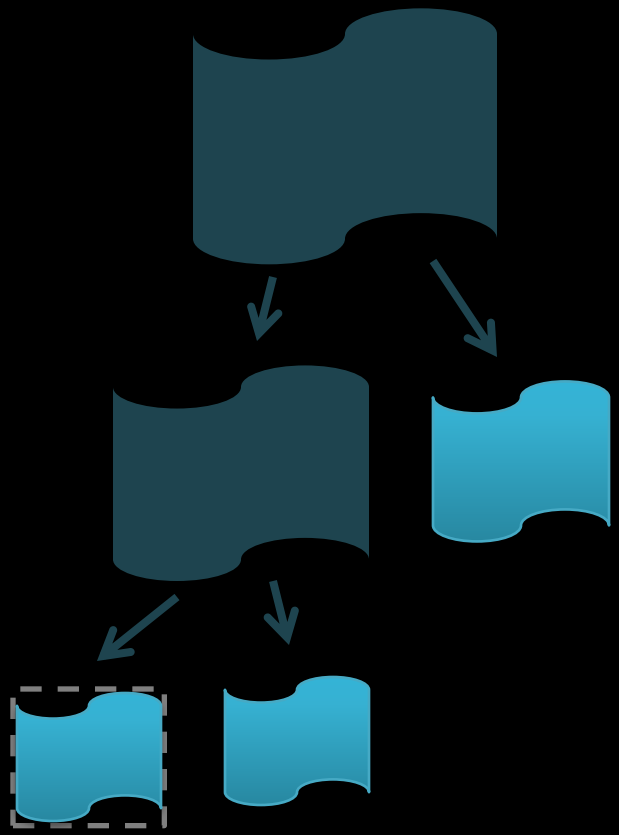
Reyes Subdivision



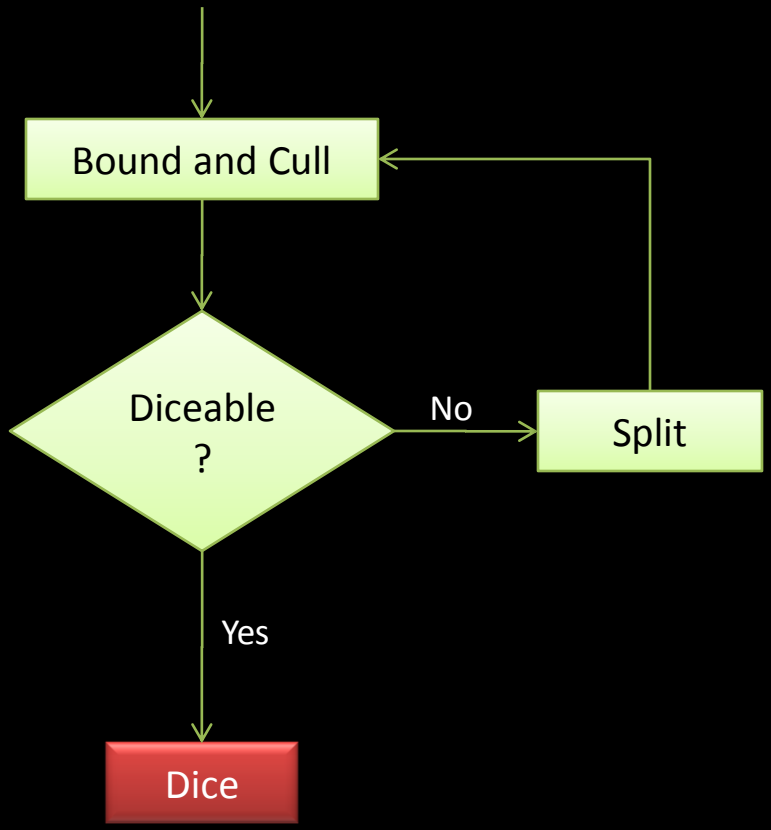


Reyes Subdivision

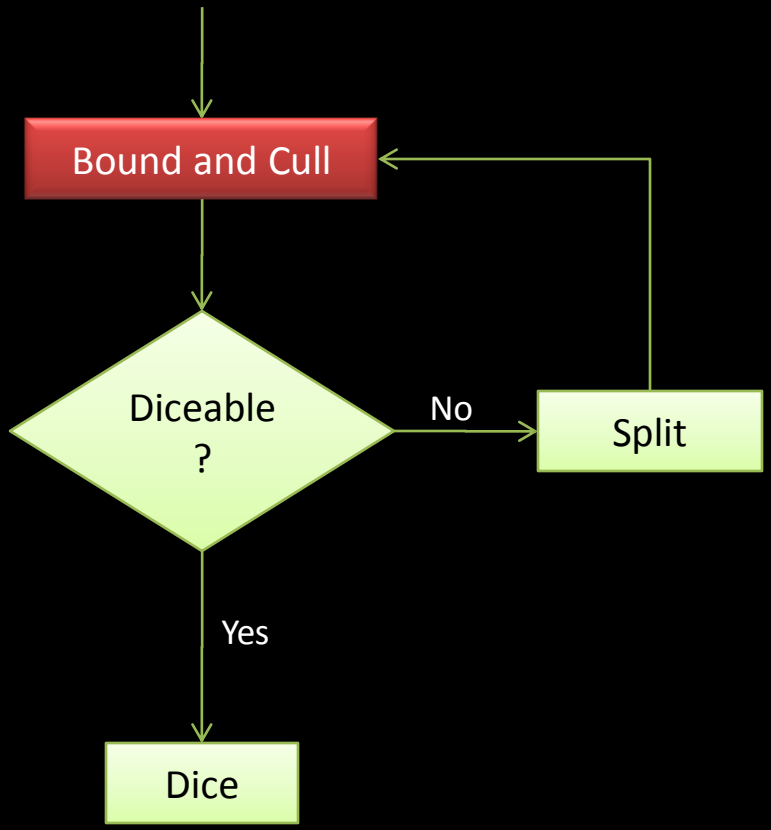




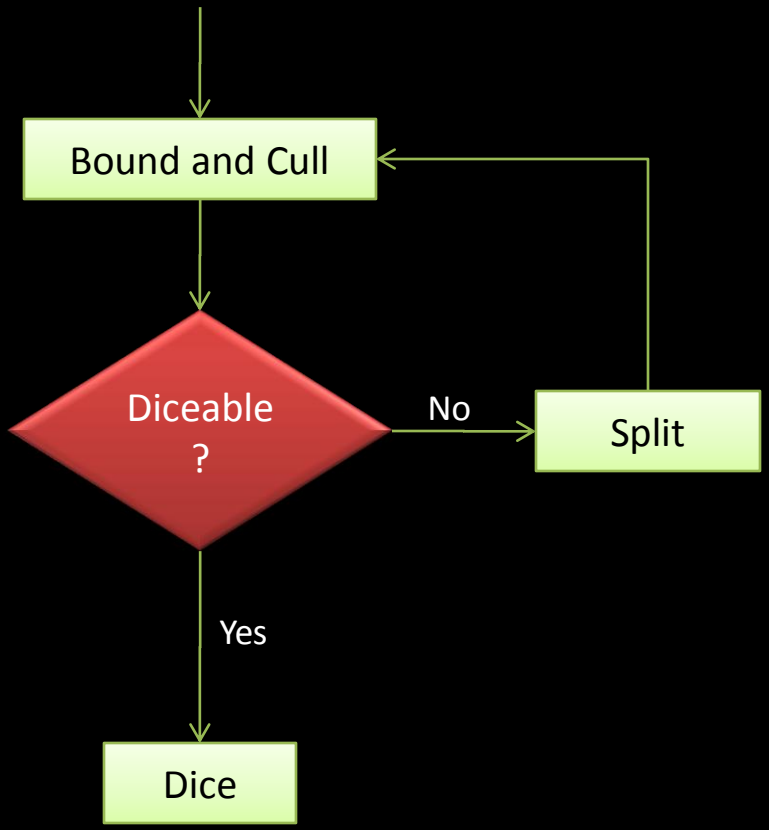
Reyes Subdivision



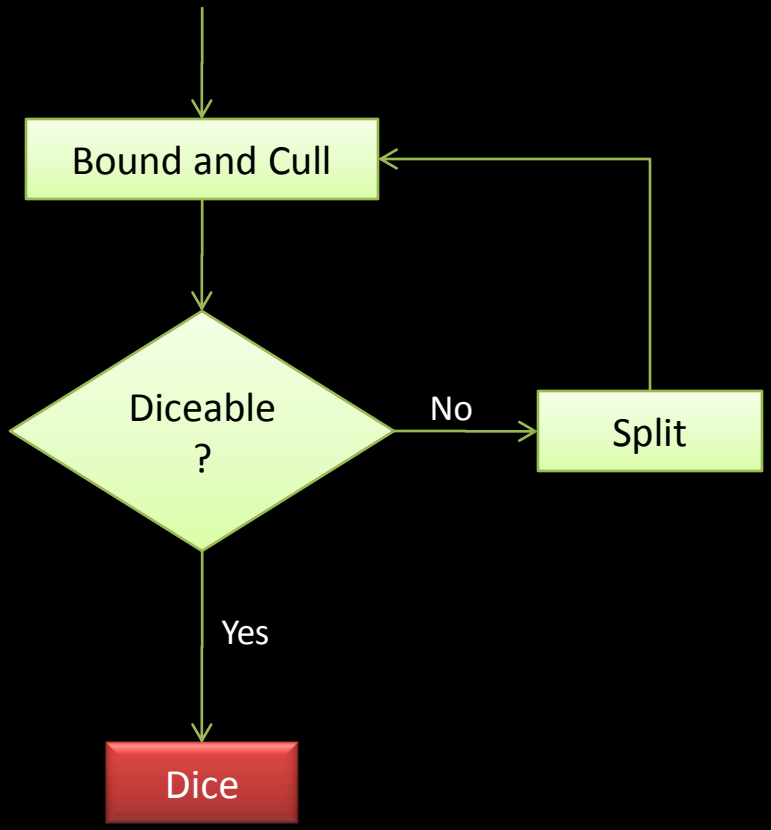
Reyes Subdivision



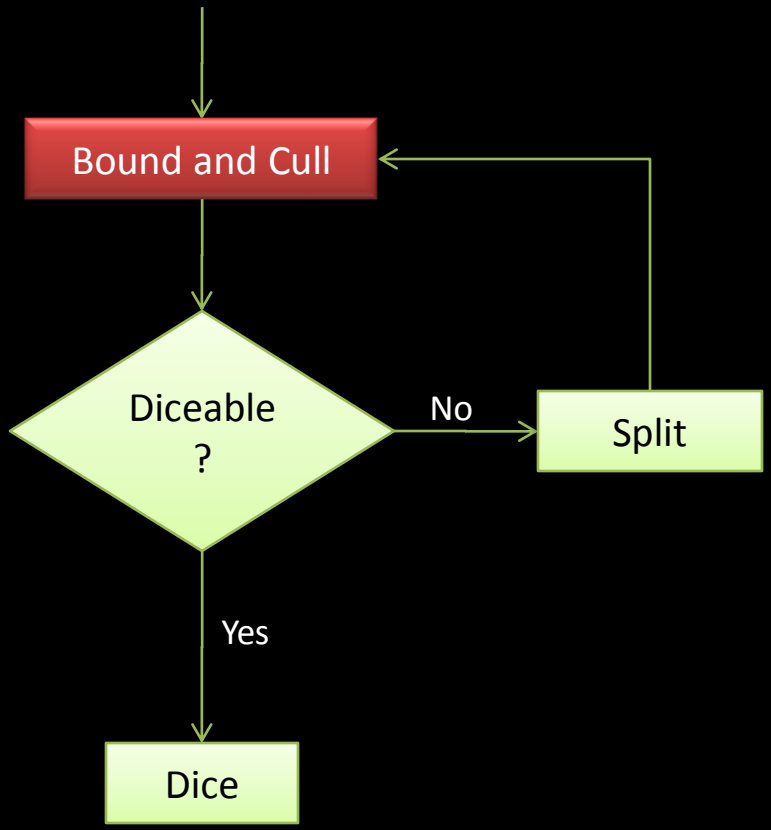
Reyes Subdivision

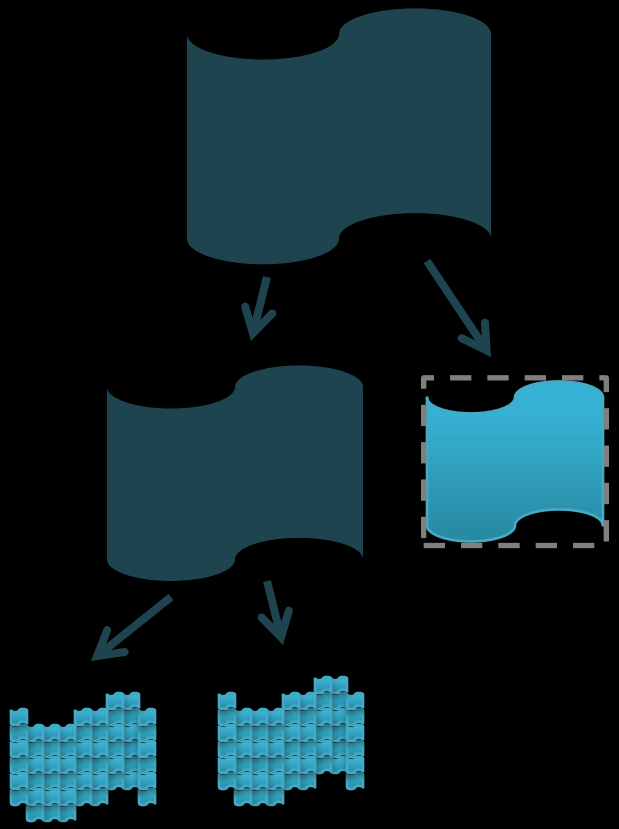


Reyes Subdivision

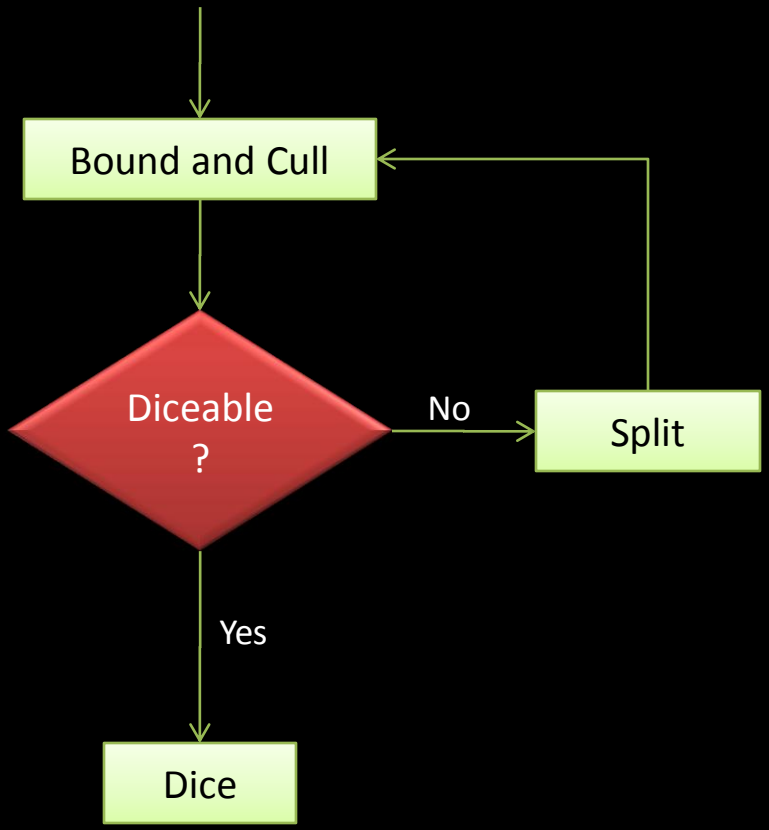


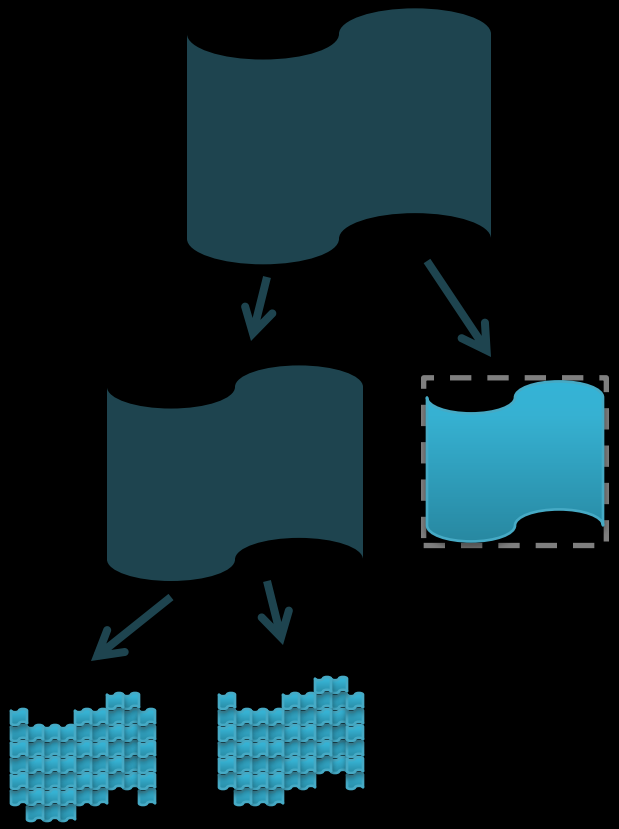
Reyes Subdivision



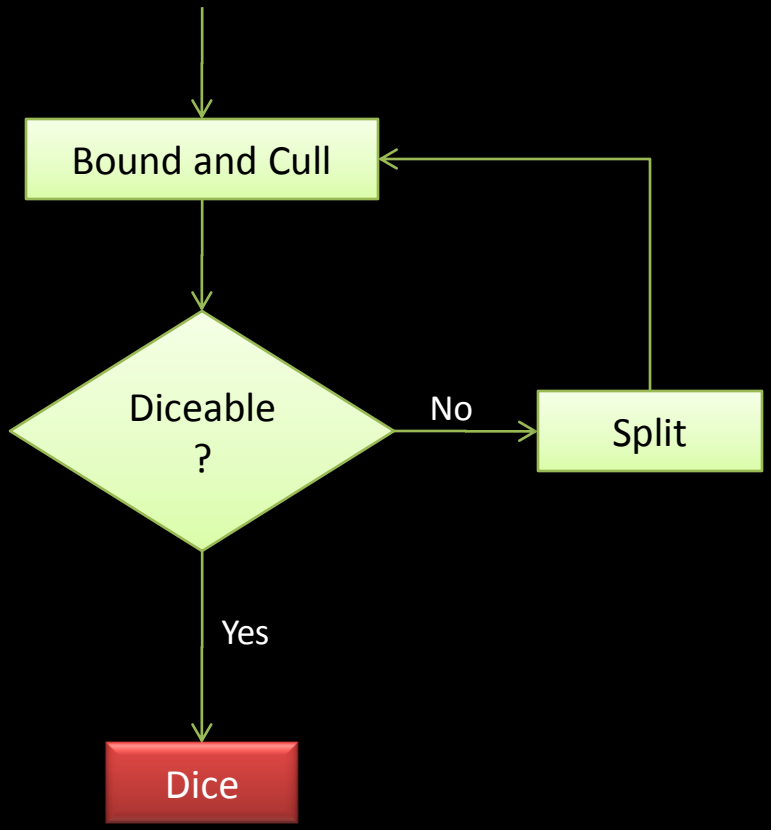


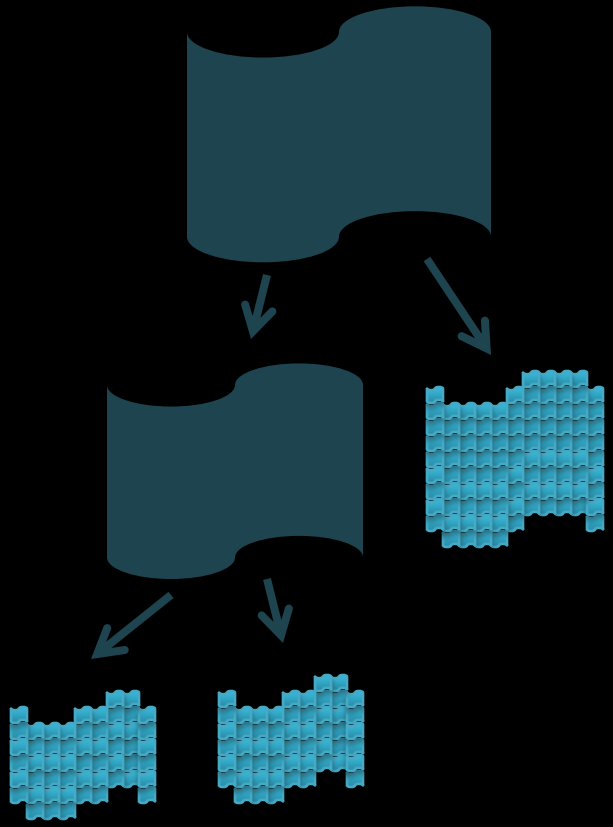
Reyes Subdivision





Reyes Subdivision

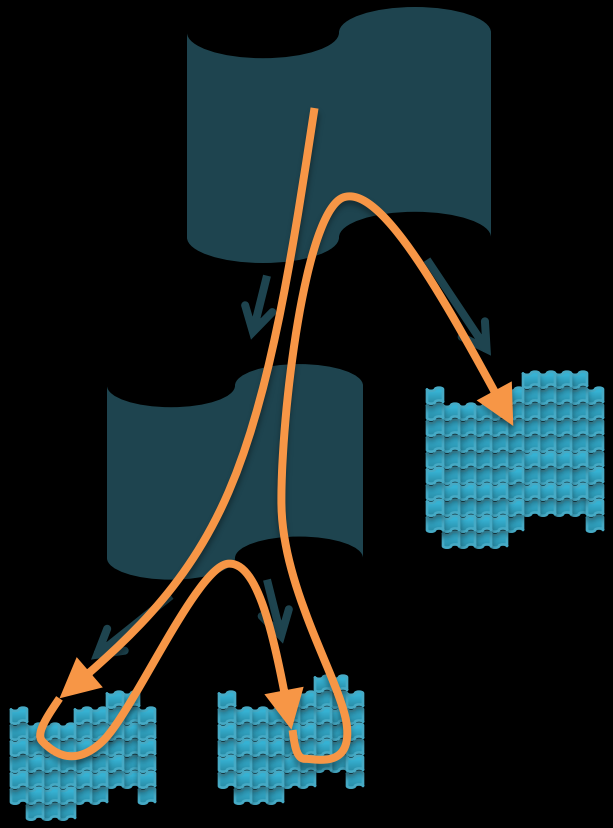




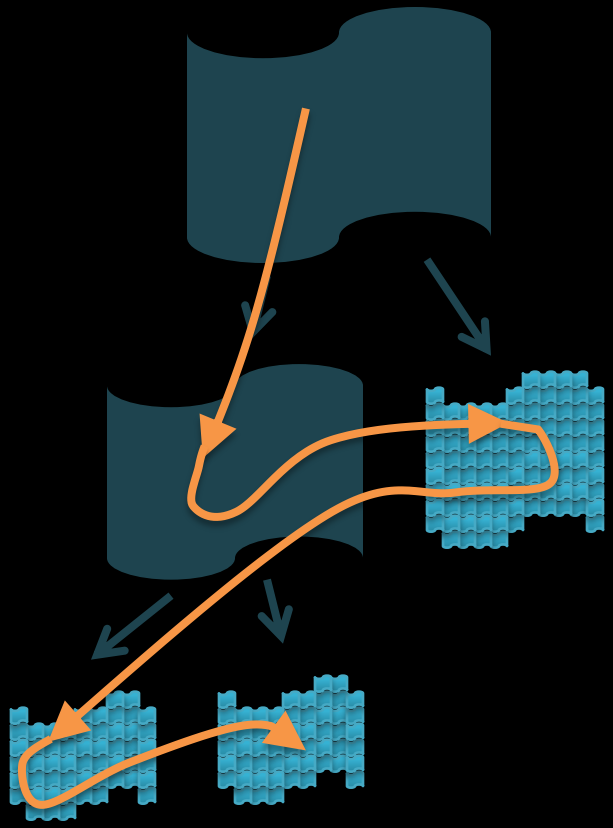
What is bad?

- Depth first subdivision is recursive!
- List of primitives is not static
  - Cull, split may destroy or generate primitives
- Unbounded memory
  - Dicing produces a huge number of micropolygons

Can we do this in parallel?



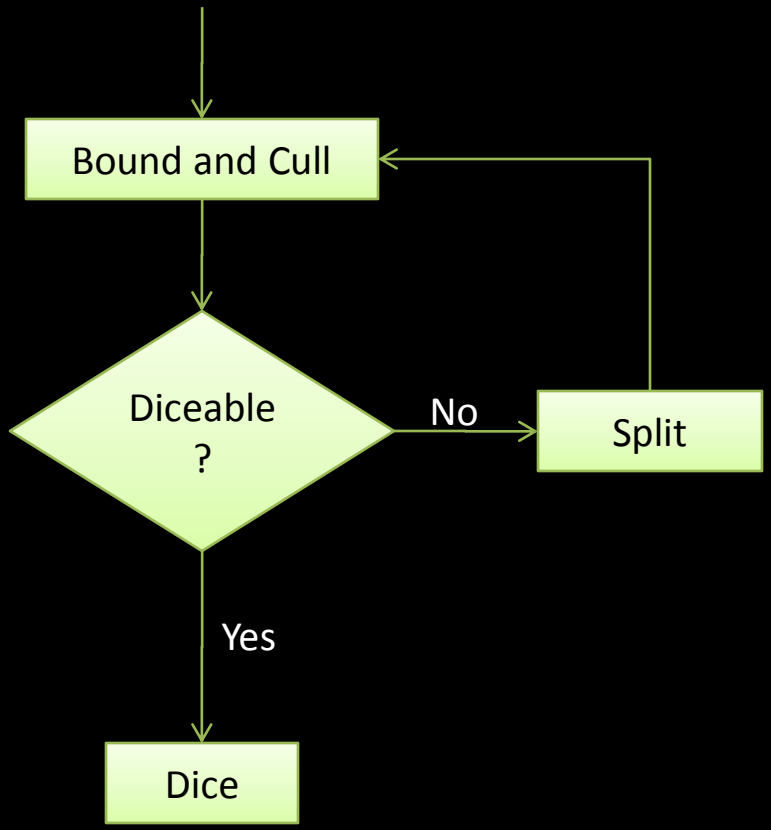
Can we do this in parallel?

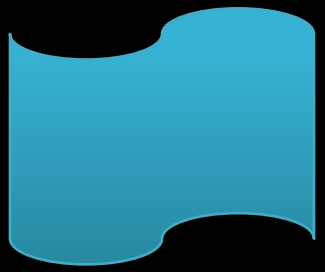


Can we do this in parallel?

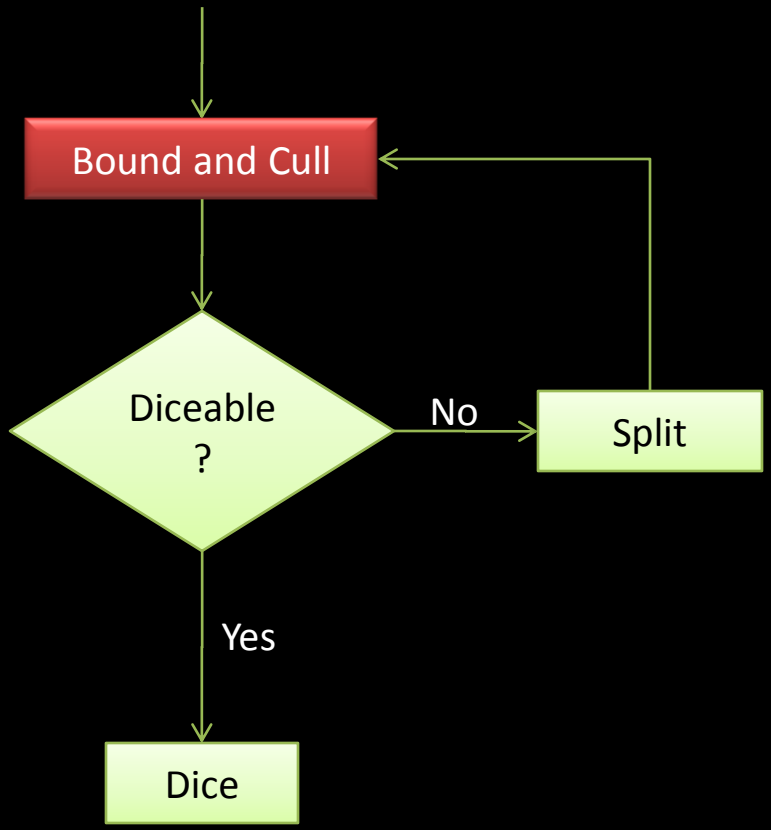
- A lot of independent operations
  - Our simplest model:
    - 5k primitives, 1.2M micropolygons
  - Massively Parallel workload
- Regular Computation
  - Bound/Split/Dice all primitives together
  - SPMD friendly

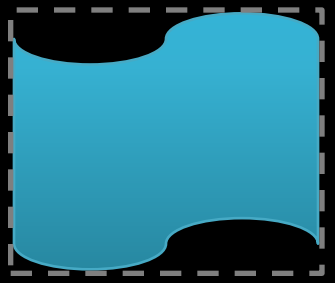
Parallel Reyes Subdivision



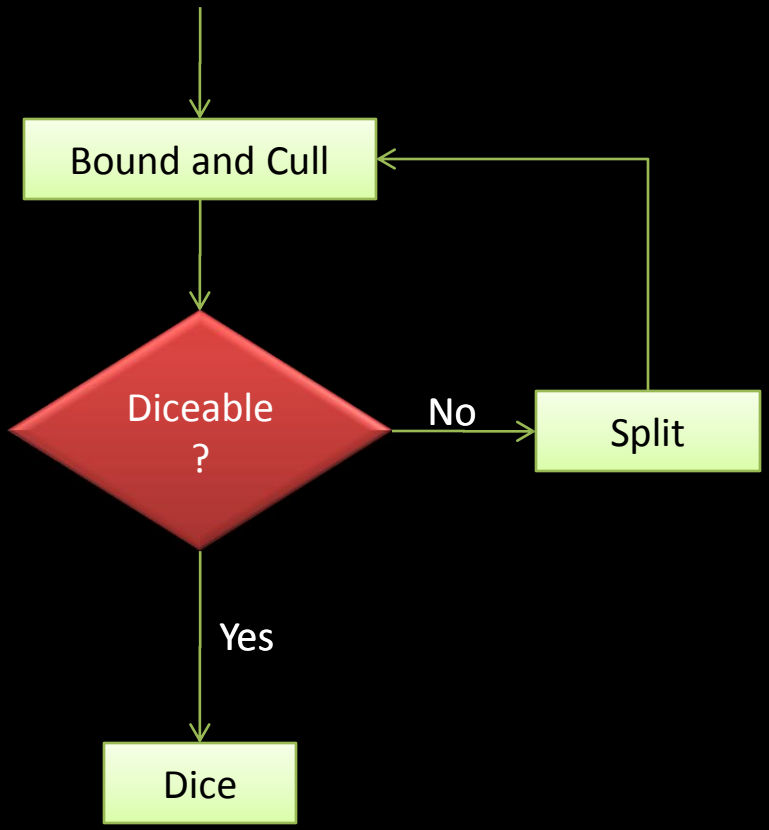


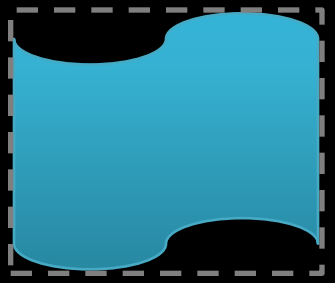
Parallel Reyes Subdivision



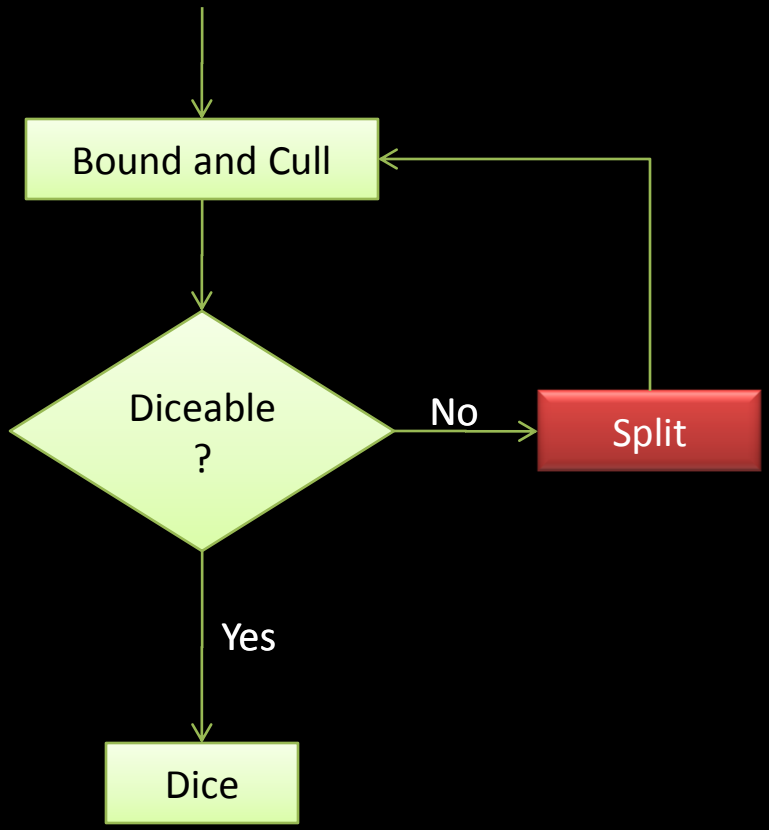


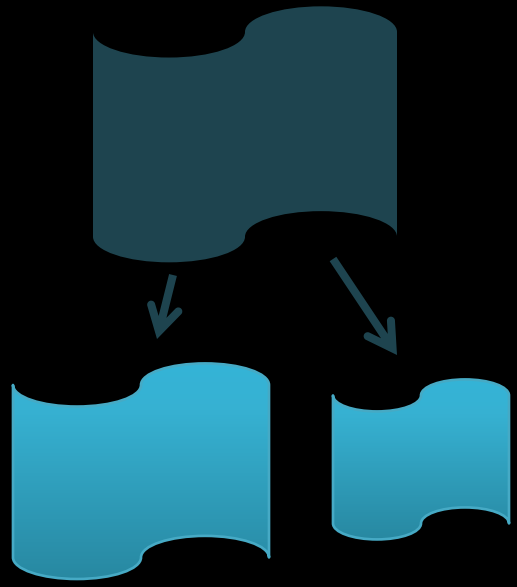
Parallel Reyes Subdivision



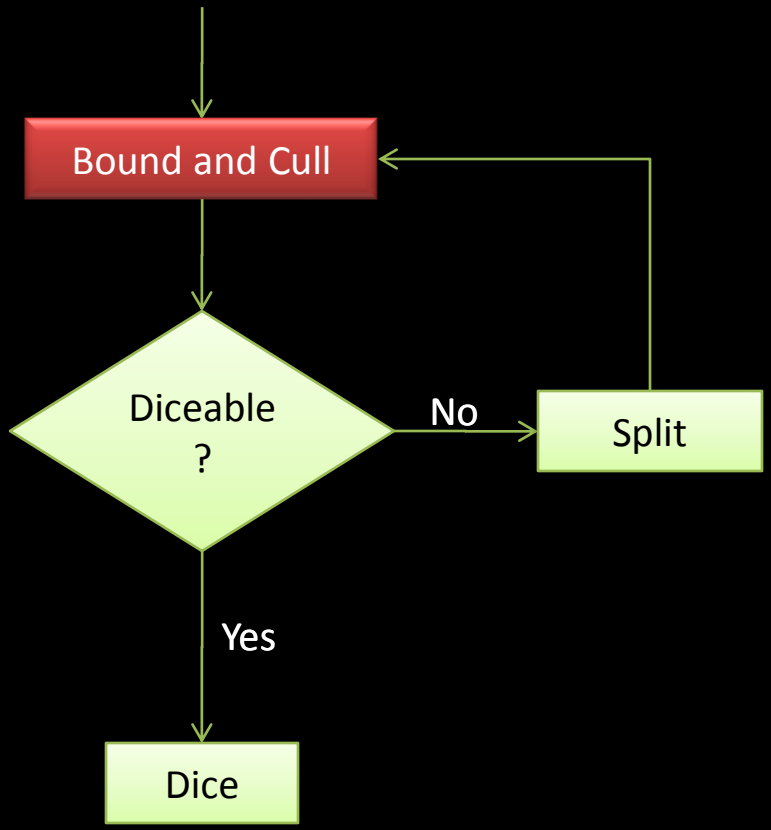


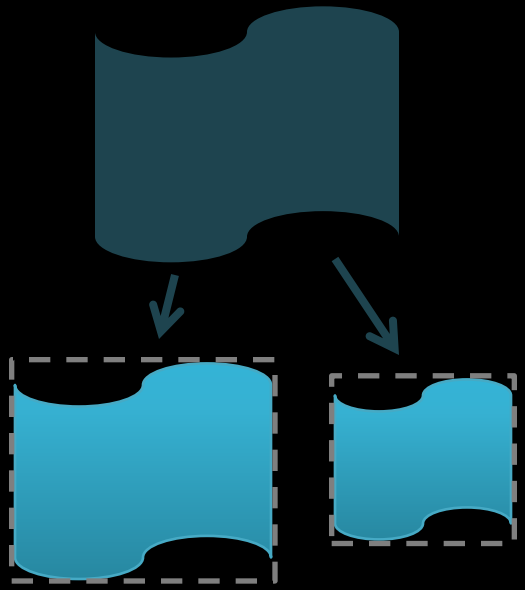
Parallel Reyes Subdivision



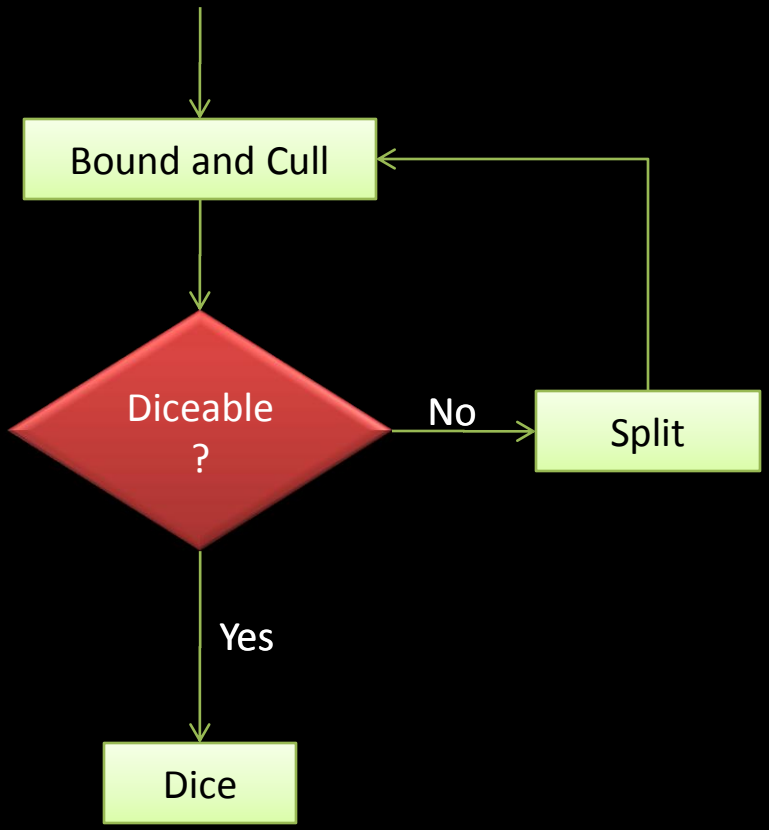


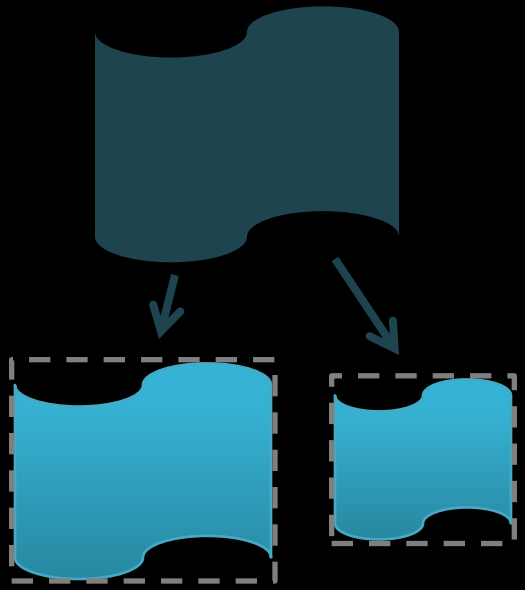
Parallel Reyes Subdivision



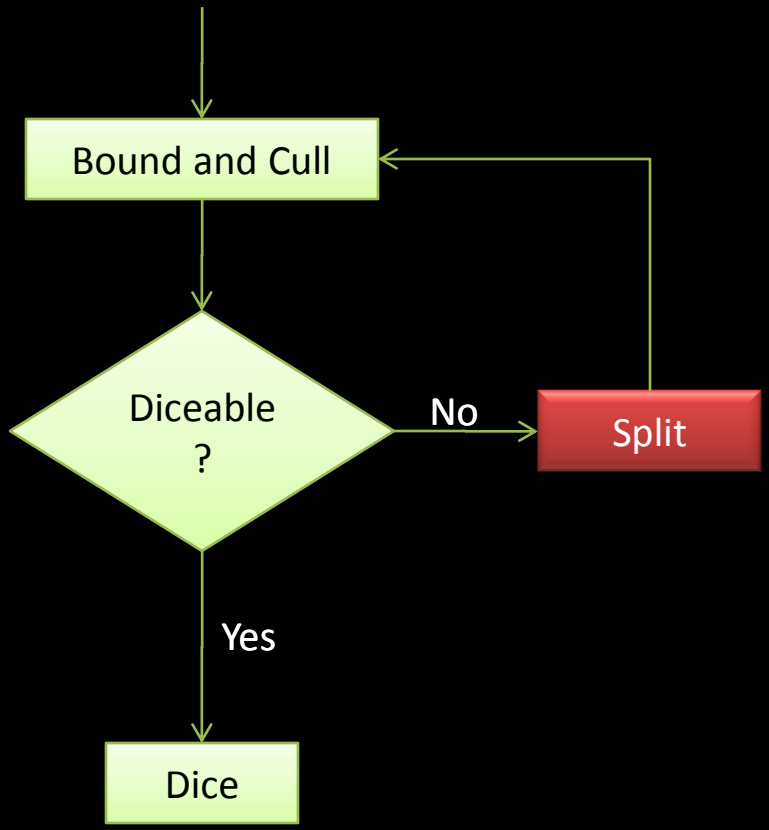


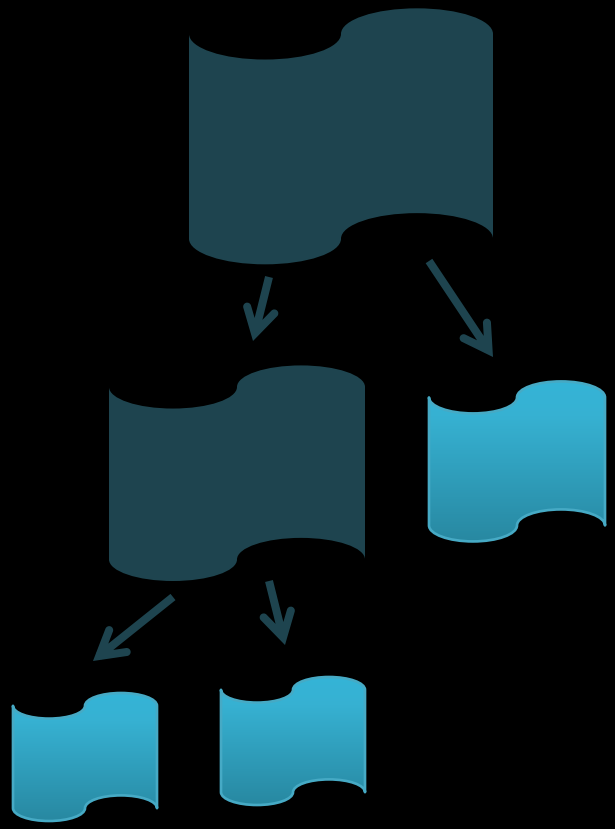
Parallel Reyes Subdivision



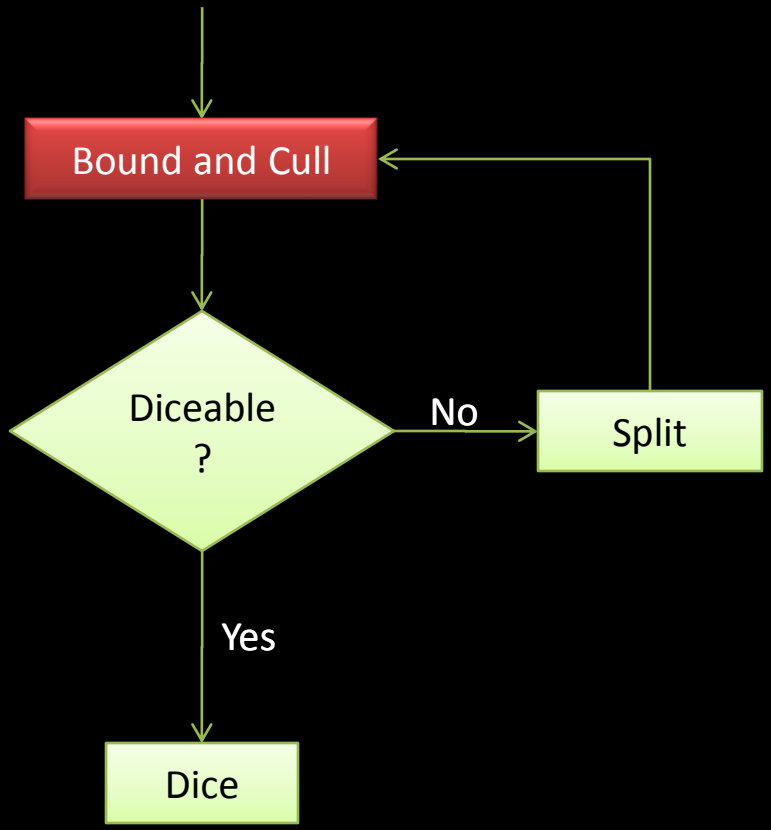


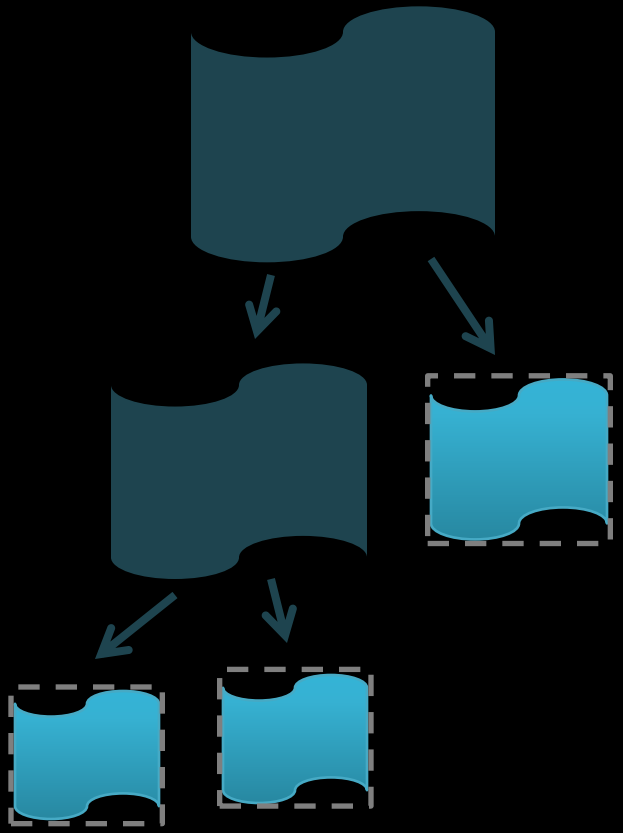
Parallel Reyes Subdivision



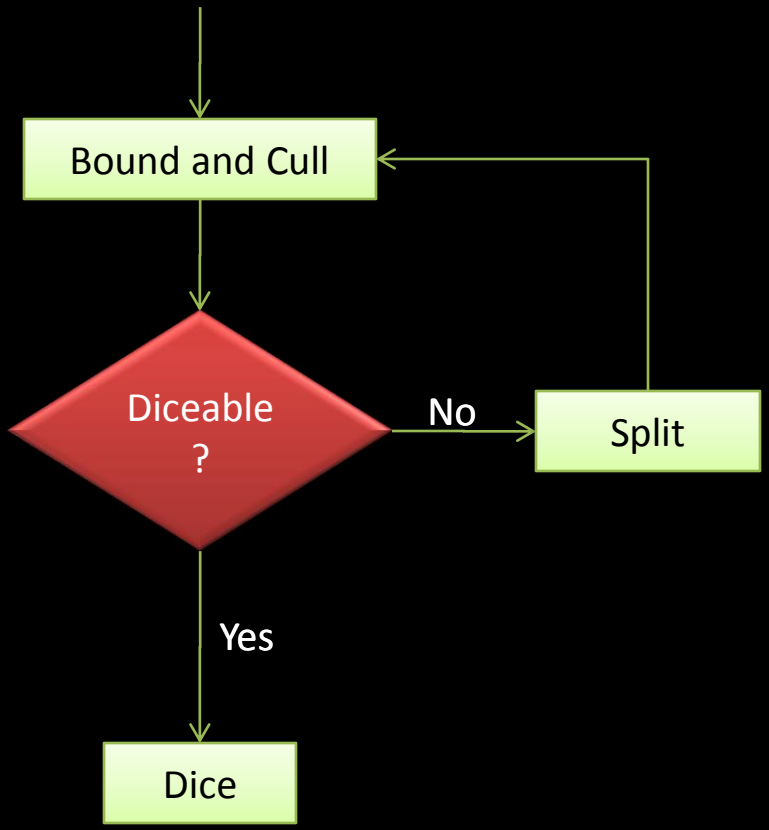


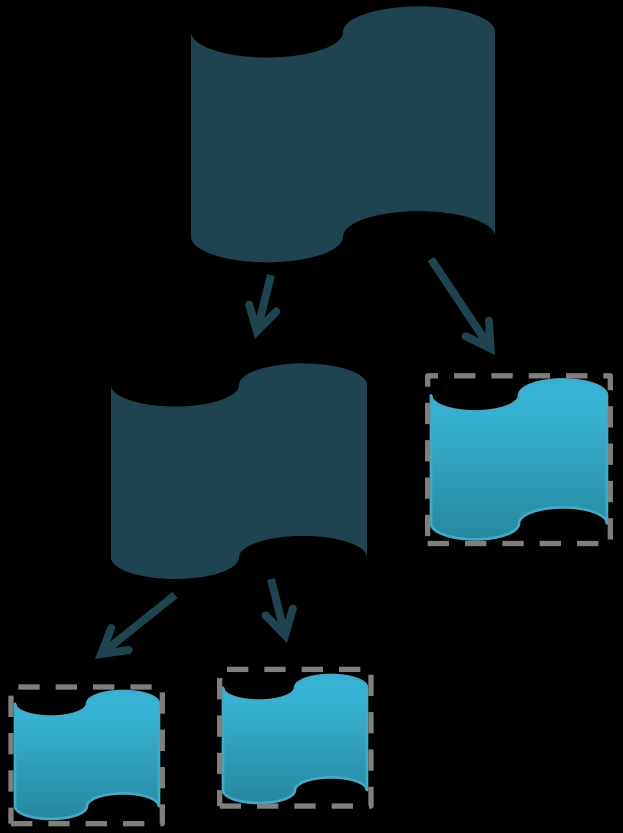
Parallel Reyes Subdivision



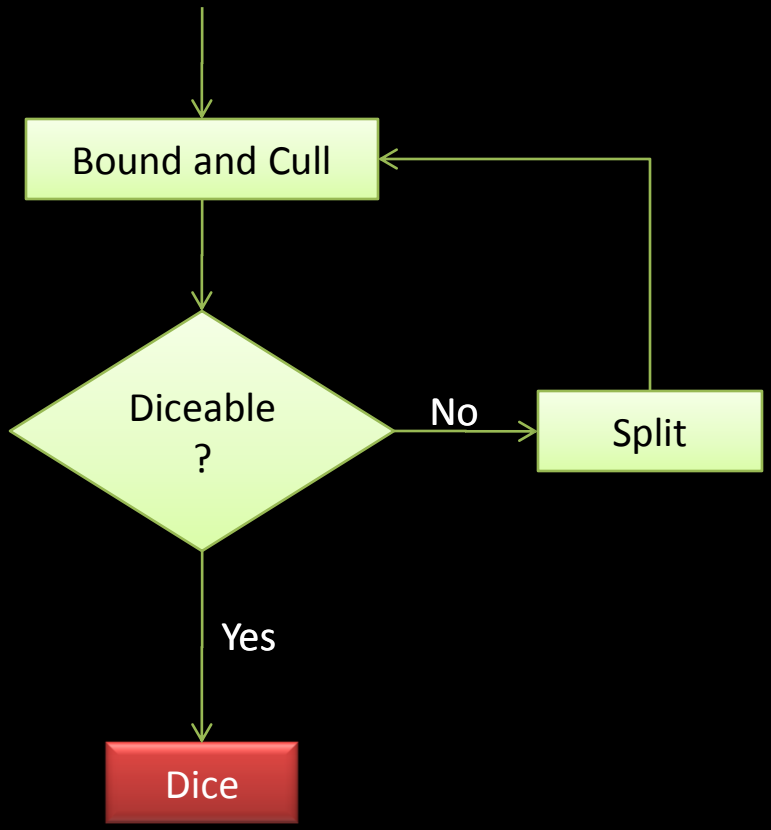


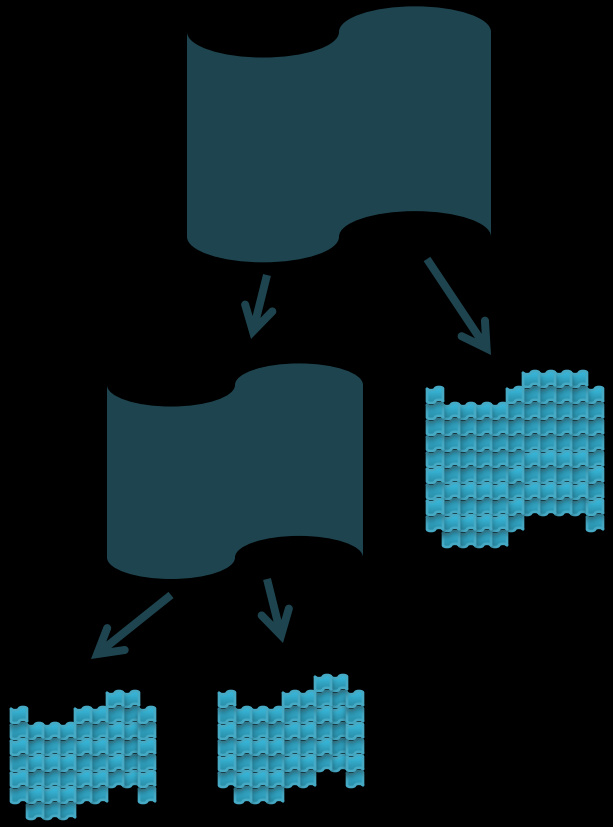
Parallel Reyes Subdivision



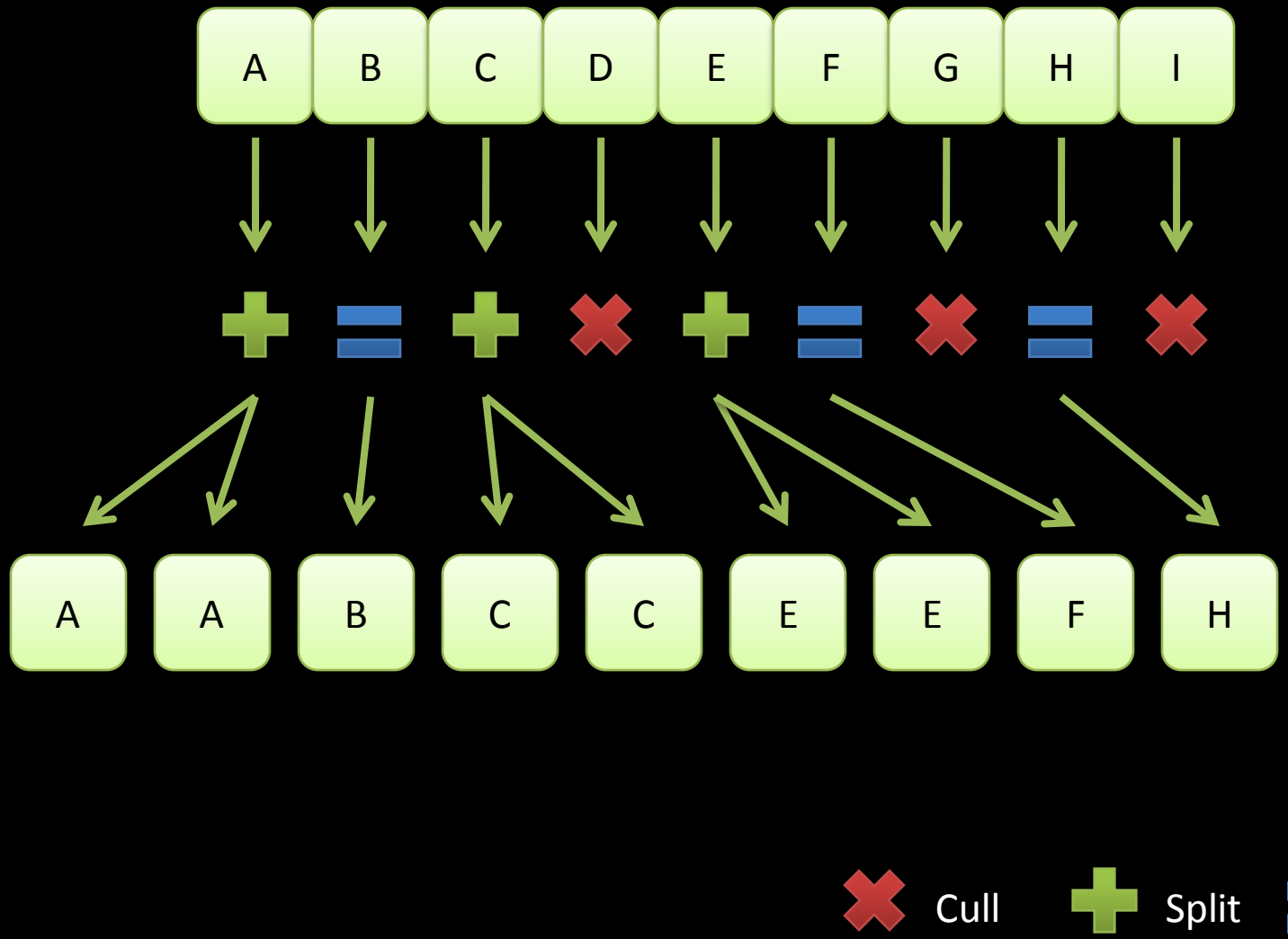


Parallel Reyes Subdivision





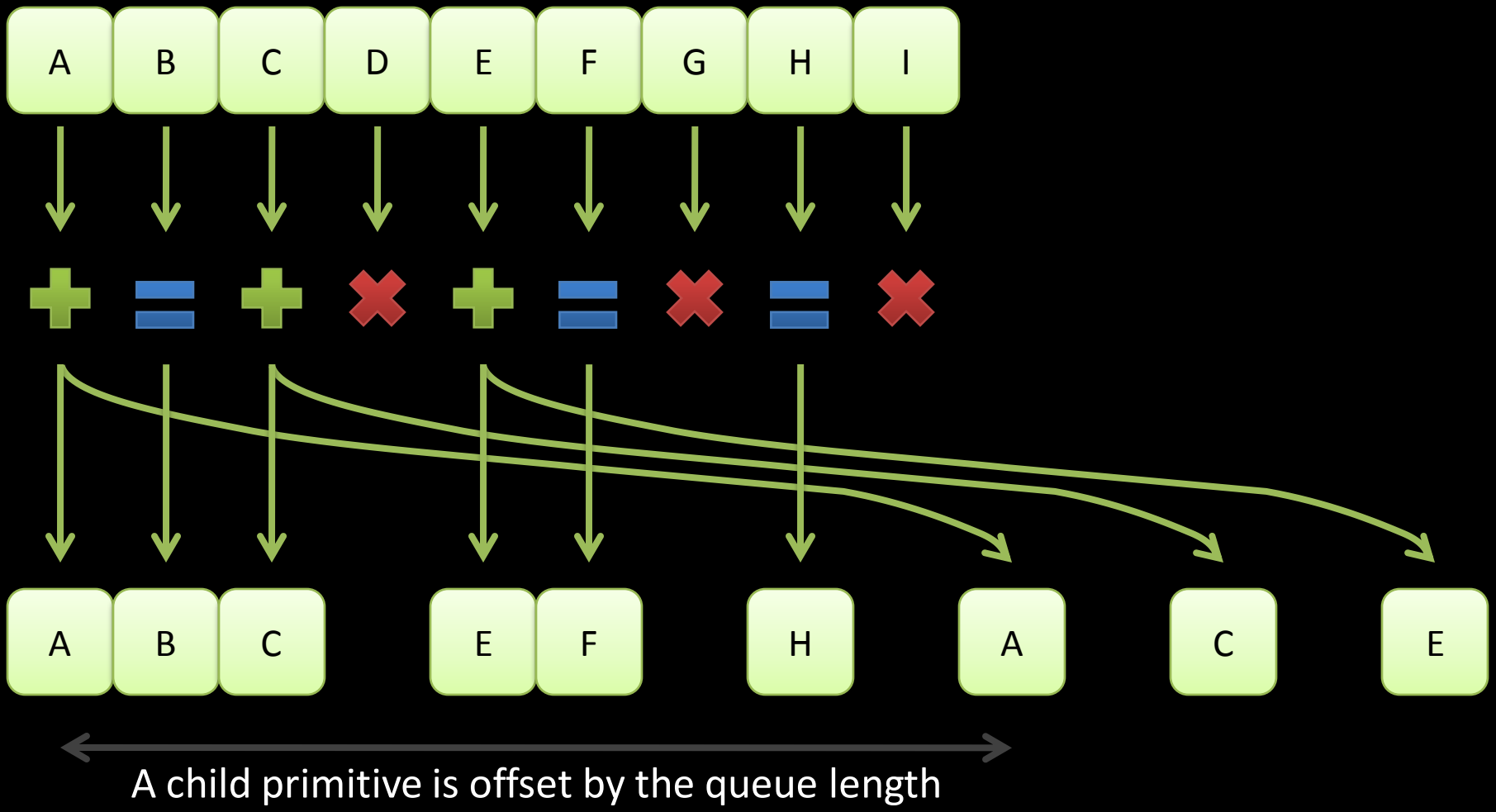
Analogy: A Dynamic Work Queue



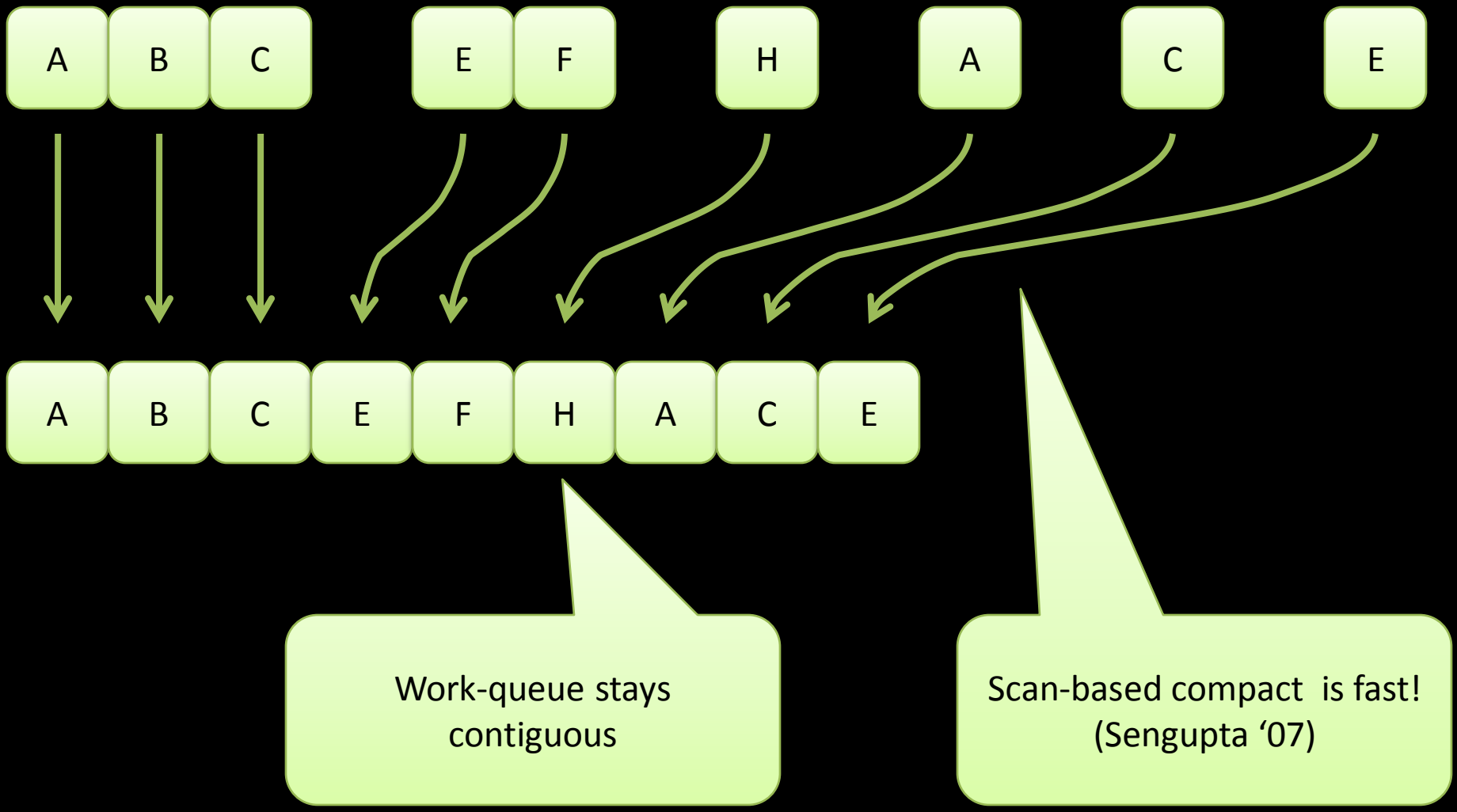
How can we do these efficiently?

- 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



Outline

- Motivation
- Reyes Subdivision – algorithm
  - Challenges
  - Parallel formulation
- Subdivision on GPU – implementation
  - Issues
  - Solutions
- Results

Platform

- NVIDIA GeForce 8800 GTX
  - 16 SMs, each with 32-wide effective SIMD
  - 16KB shared memory per SM
  - 768 MB total GPU memory, no cache
- NVIDIA CUDA 1.1
  - Grid/Block/Thread programming model
  - OpenGL interface through shared buffers

Implementation Details

- Input primitives – Bicubic Bézier Surfaces
  - Choice of primitive only affects implementation
- View Dependent Subdivision every frame
  - CPU-GPU input transfer only once
  - Suitable for animating control points
- Final micropolygons sent to OpenGL as a VBO
  - Flat-shaded and displayed for preview

Kernels Implemented

- Dice
  - Regular, symmetric computation on a highly parallel workload
  - 256 threads per primitive
  - Primitive information in shared memory
- Bound/Split
  - Non-trivial to ensure efficiency in implementation

Bound/Split: Efficiency Goals

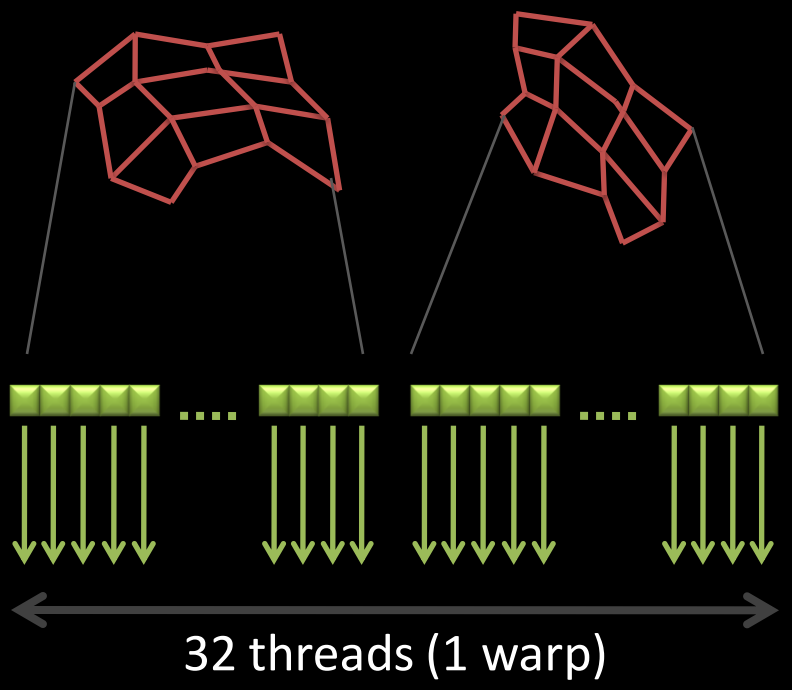
- Memory Coherence
  - Off-chip memory accesses must be efficient
- Computational Efficiency
  - Hardware SIMD must be maximally utilized

Memory Coherence

- After each iteration, work queue is compacted
  - Primitives always contiguous in memory
- Structure-Of-Arrays representation
  - Attributes across primitives adjacent in memory
- 99.5% of all accesses were fully coalesced

SIMD Utilization

- Intra-Primitive parallelism
  - A primitive's control points are mostly independent
  - Execution path divergence is negligible
- 16 Threads per primitive
  - Vectorized Bound/Split
  - Use shared memory for communication
- 90.16% of all branches were SIMD coherent



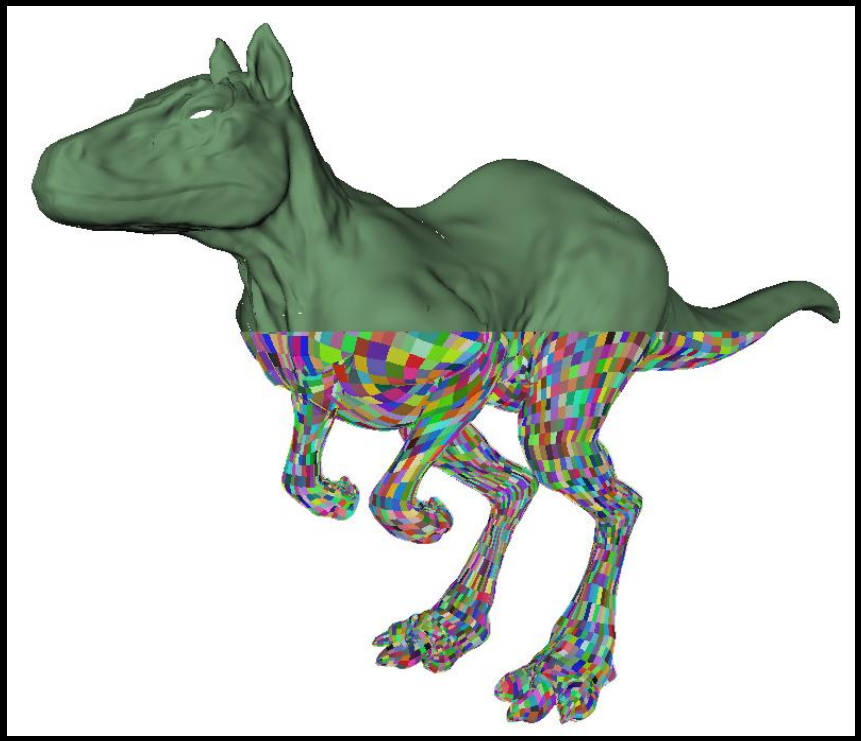
Outline

- Motivation
- Reyes Subdivision – algorithm
  - Challenges
  - Parallel formulation
- Subdivision on GPU – implementation
  - Issues
  - Solutions
- Results

Results - Killeroo

- 11532 patches → 14426 grids
- 5 levels of subdivision
- Bound/Split: 6.99 ms
- Dice: 7.21 ms

- 4.2 frames per second (subdivision-only: 70)

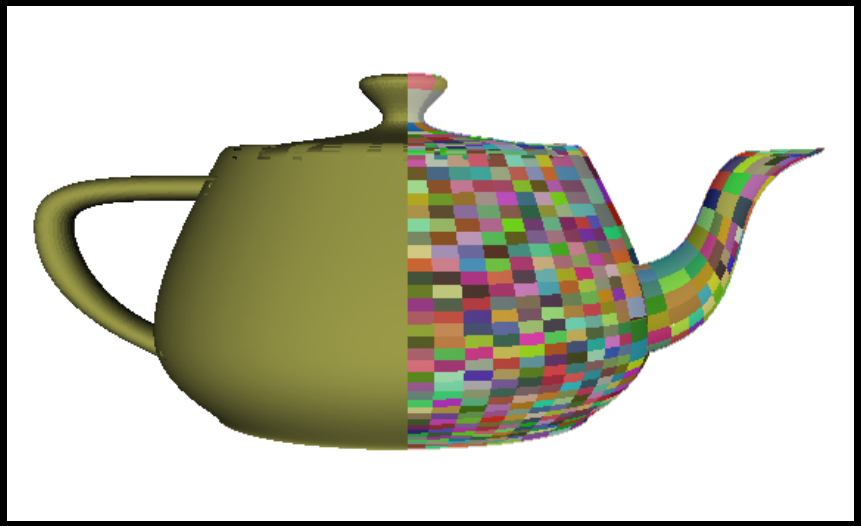


Killeroo NURBS model courtesy headus 3D tools: http://headus.com.au

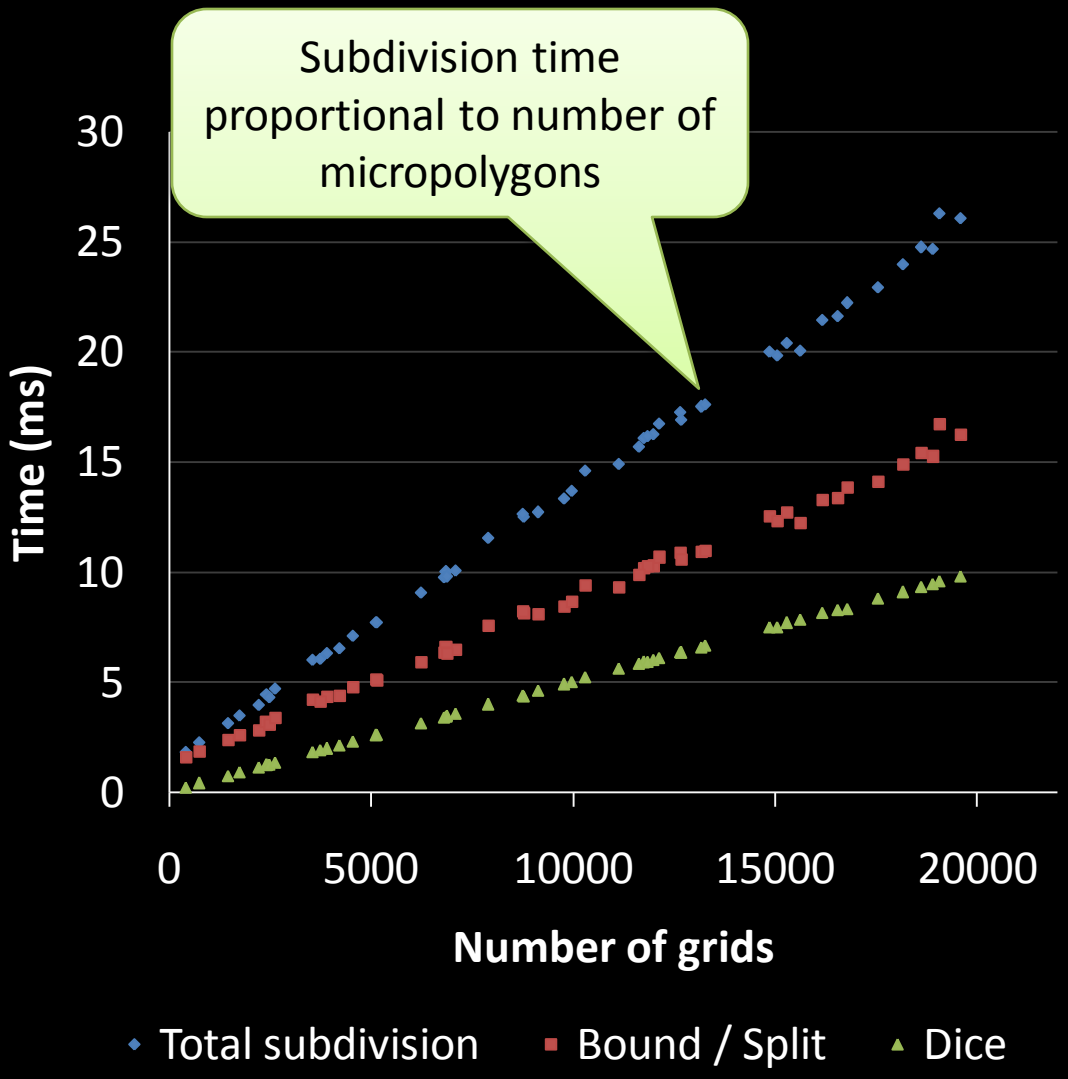
Results - Teapot

- 32 patches → 4823 grids
- 11 levels of subdivision
- Bound/Split: 3.46 ms
- Dice: 2.42 ms

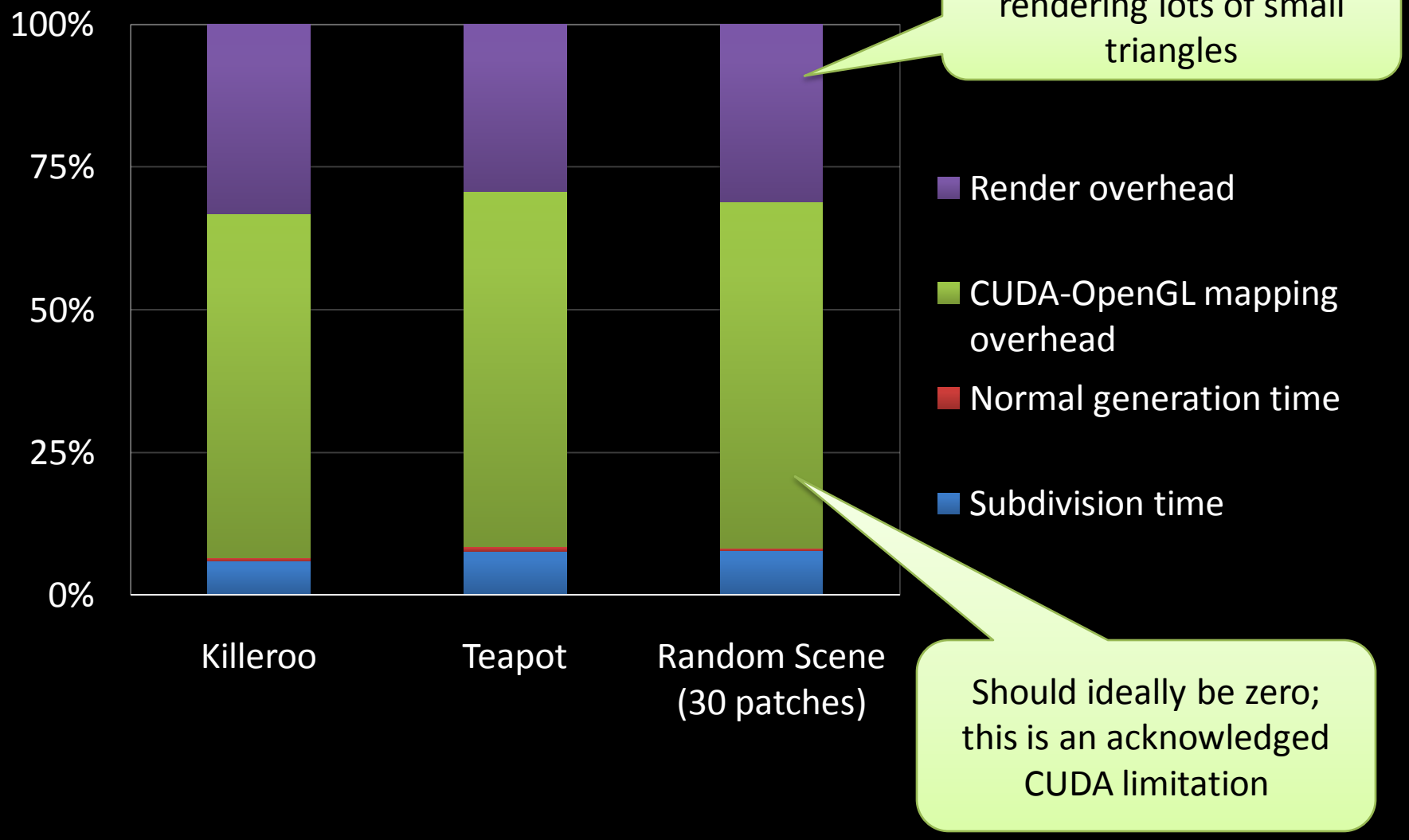
- 12.4 frames per second (subdivision-only: 170)



Results – Random Models



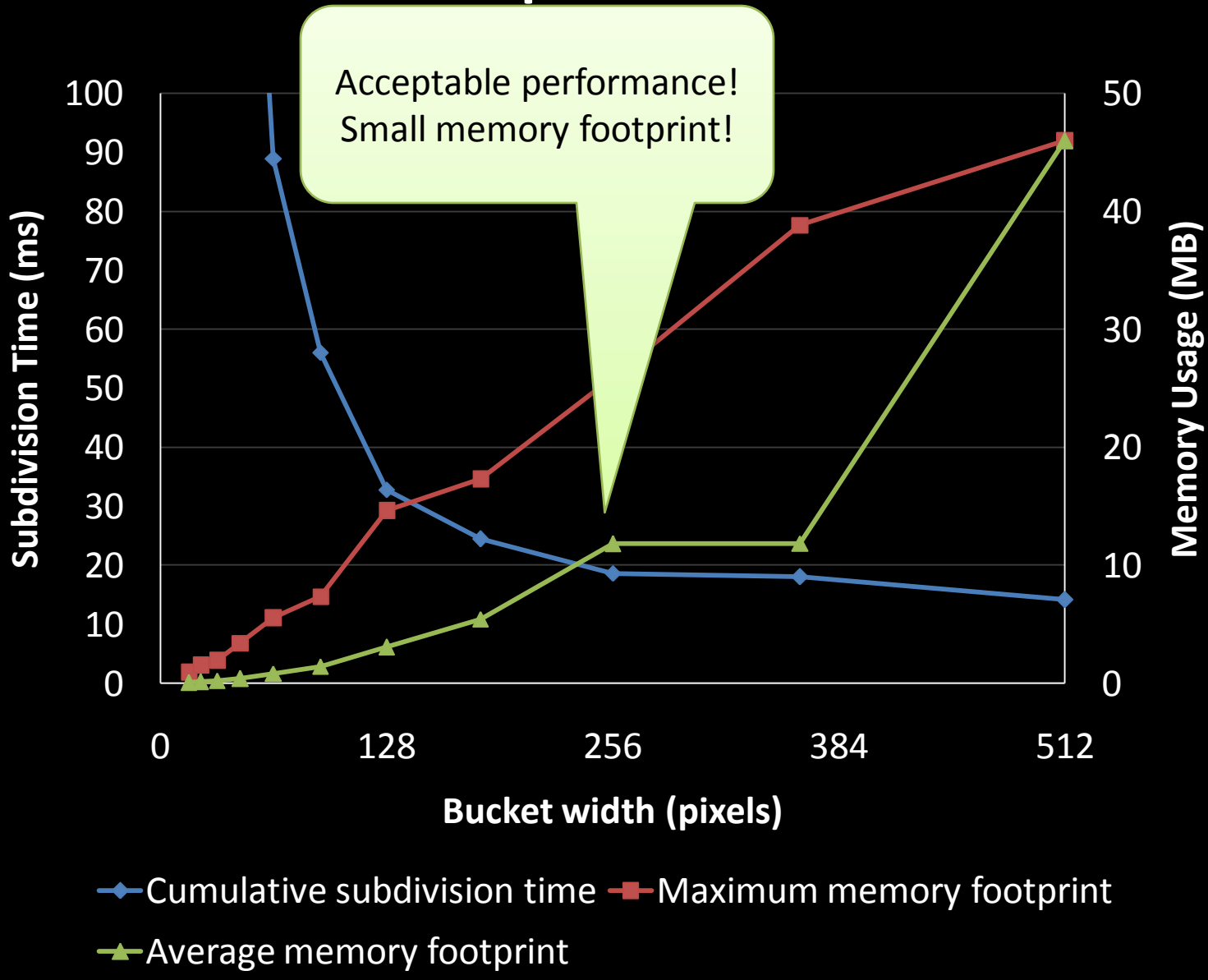
Overheads


100%
75%
50%
25%
0%
Killeroo
Teapot
Random Scene (30 patches)
Render overhead
CUDA-OpenGL mapping overhead
Normal generation time
Subdivision time
GPUs are inefficient at rendering lots of small triangles
Should ideally be zero; this is an acknowledged CUDA limitation

Storage Issues

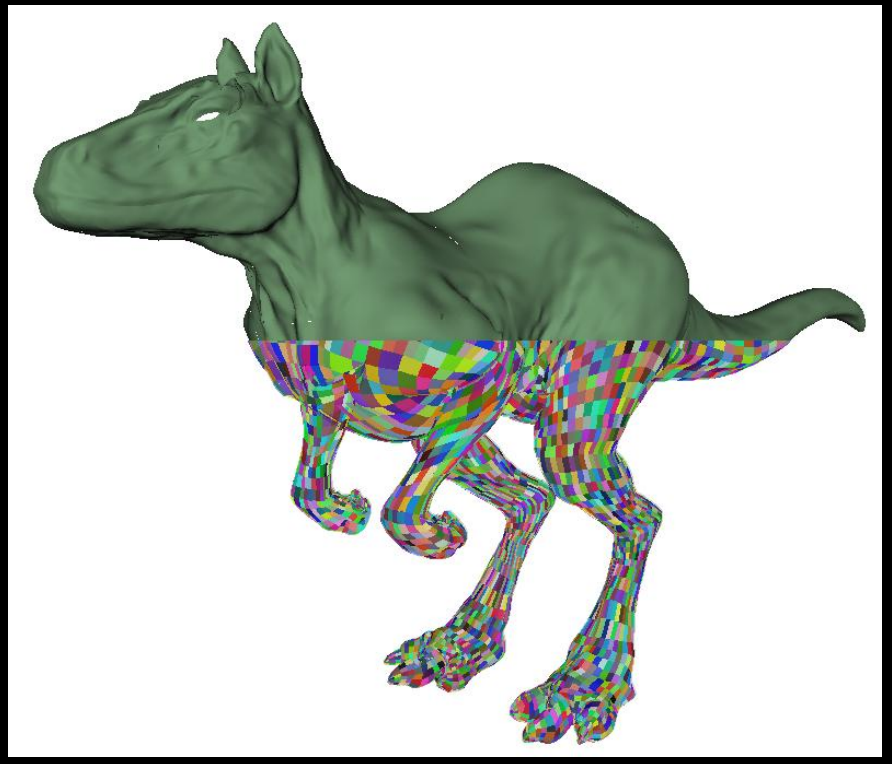
- Reyes pipeline suffers from unbounded memory demand
  - A huge number of micropolygons are generated
  - Transparency and Blending preclude early rejection
- Most implementations use screen-space buckets
  - But how does this work in parallel?
  - Large buckets present a more parallel workload
  - Small buckets have a smaller memory footprint

Screen-Space Buckets



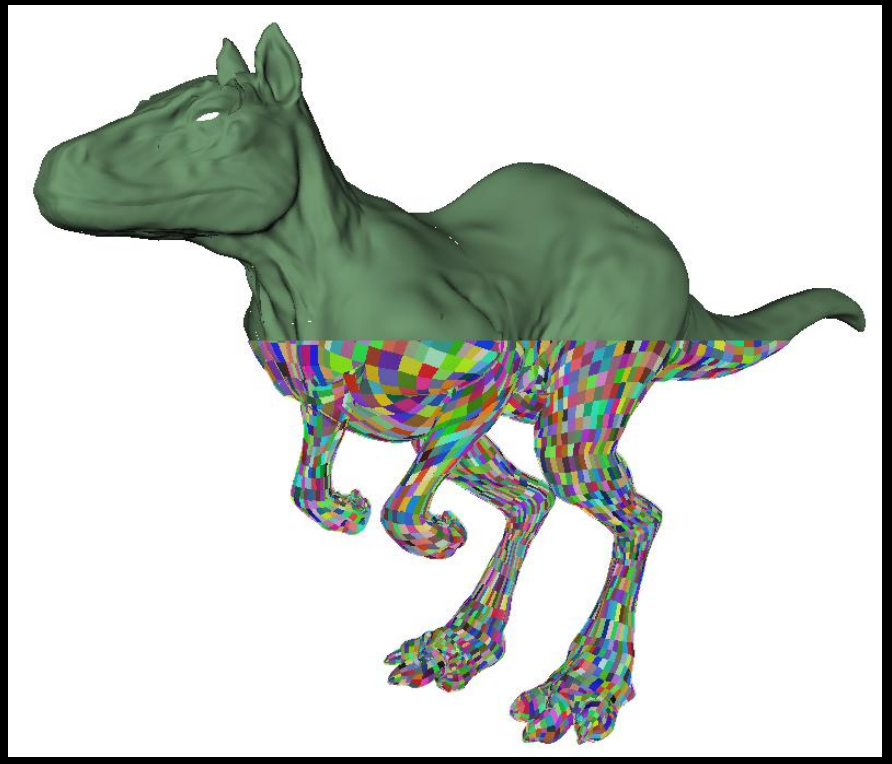
Limitations

- All primitives must be split before dicing
- Cracks / Pinholes
- Uniform Dicing is wasteful



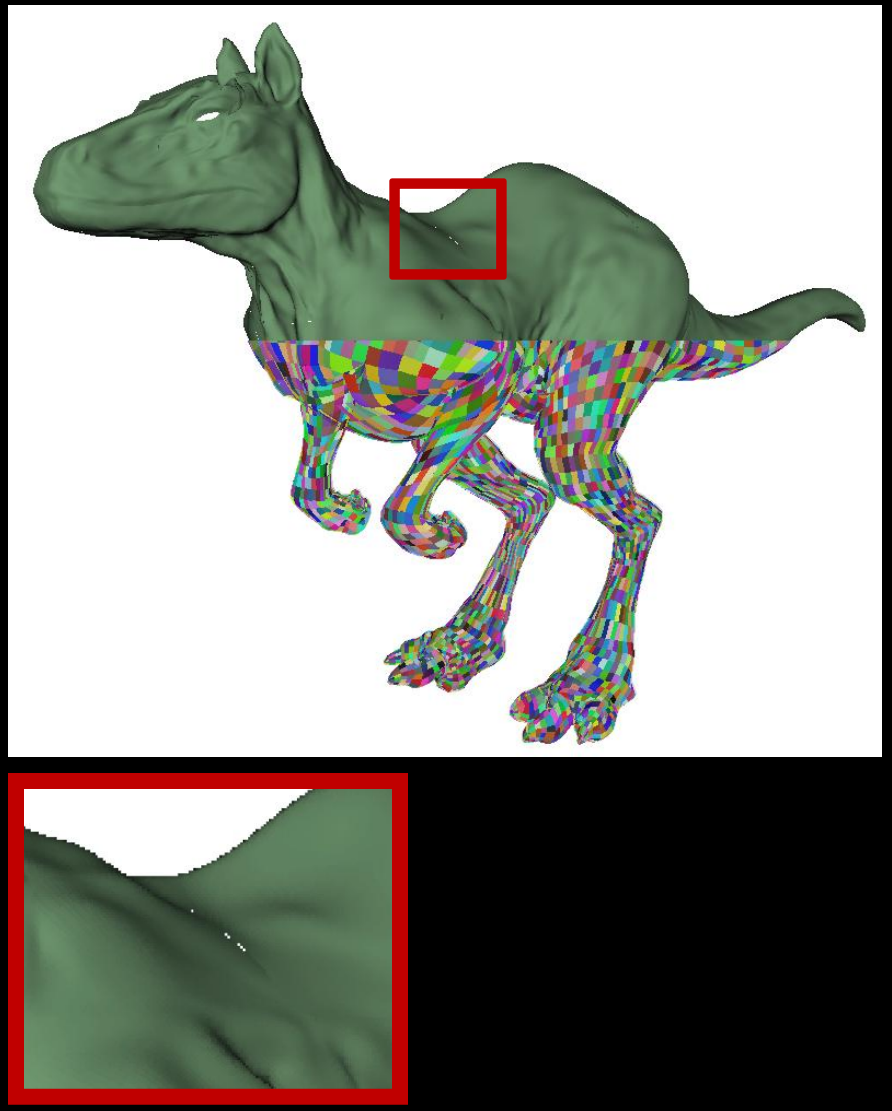
Limitations

- All primitives must be split before dicing
- Cracks / Pinholes
- Uniform Dicing is wasteful



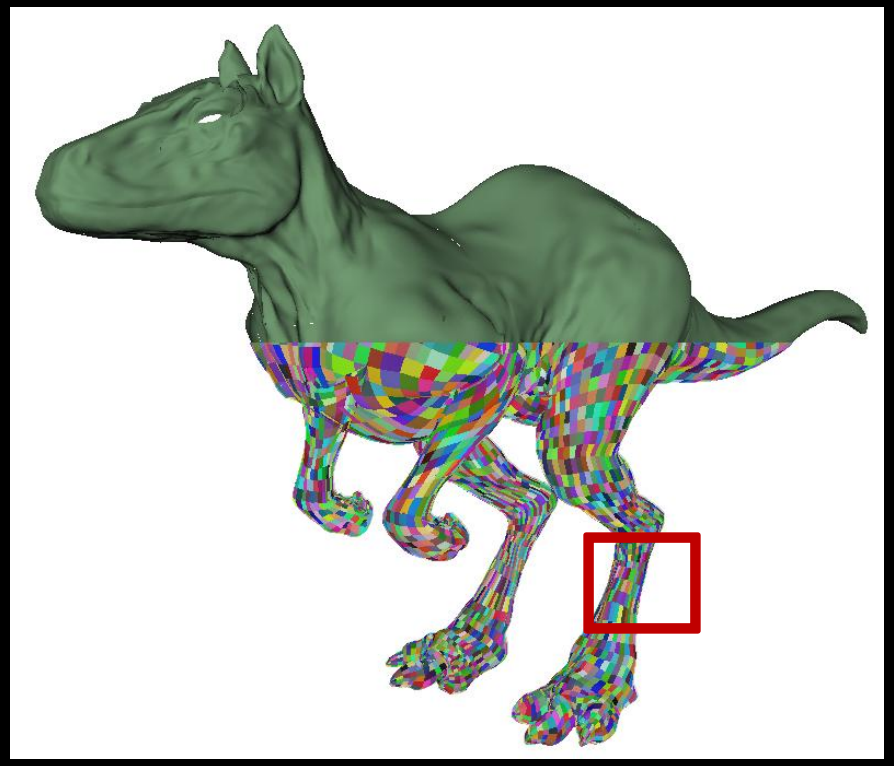
Limitations

- All primitives must be split before dicing
- Cracks / Pinholes
- Uniform Dicing is wasteful



Limitations

- All primitives must be split before dicing
- Cracks / Pinholes
- Uniform Dicing is wasteful



Conclusions

- Recursive subdivision maps well to current GPUs
  - And works fast!
  - It is advantageous to use smooth primitives in interactive rendering
- Fixed-function tessellation can be emulated
  - Dicing is already very fast (2 Mgrids/sec)
- It’s time to experiment with alternate pipelines

Future Work

- Crack Filling
  - Add dummy polygons during post-processing
- More of Reyes
  - Displacement Mapping
  - Offline quality Shading on GPUs
  - Parallel Stochastic Sampling (Wei ‘08)
  - A-buffer

Thanks to

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

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

- Equipment support from NVIDIA

Real-Time Reyes-Style Adaptive Surface Subdivision

BACKUP SLIDES

CUDA Thread Structure

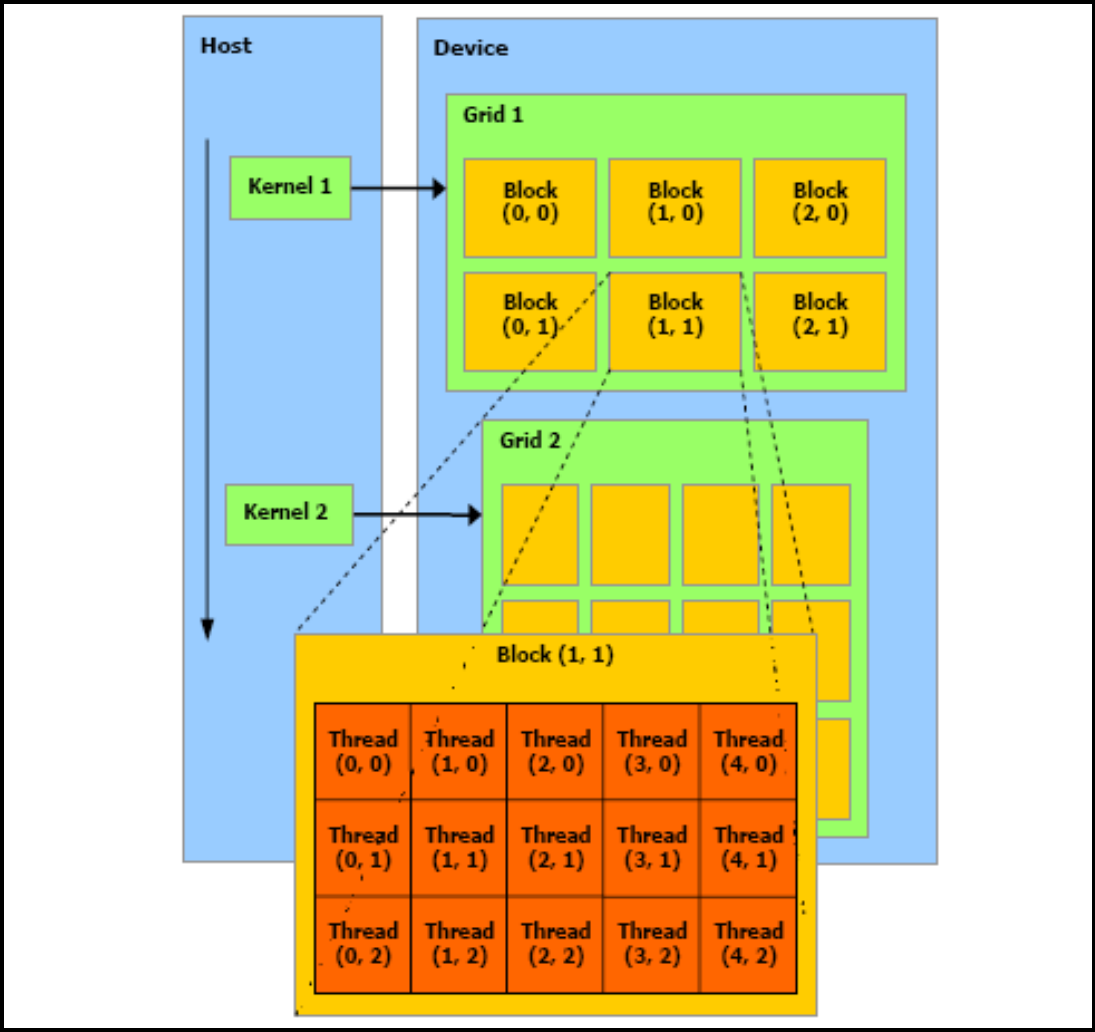


Image courtesy: NVIDIA CUDA Programming Guide, 1.1

CUDA Memory Architecture

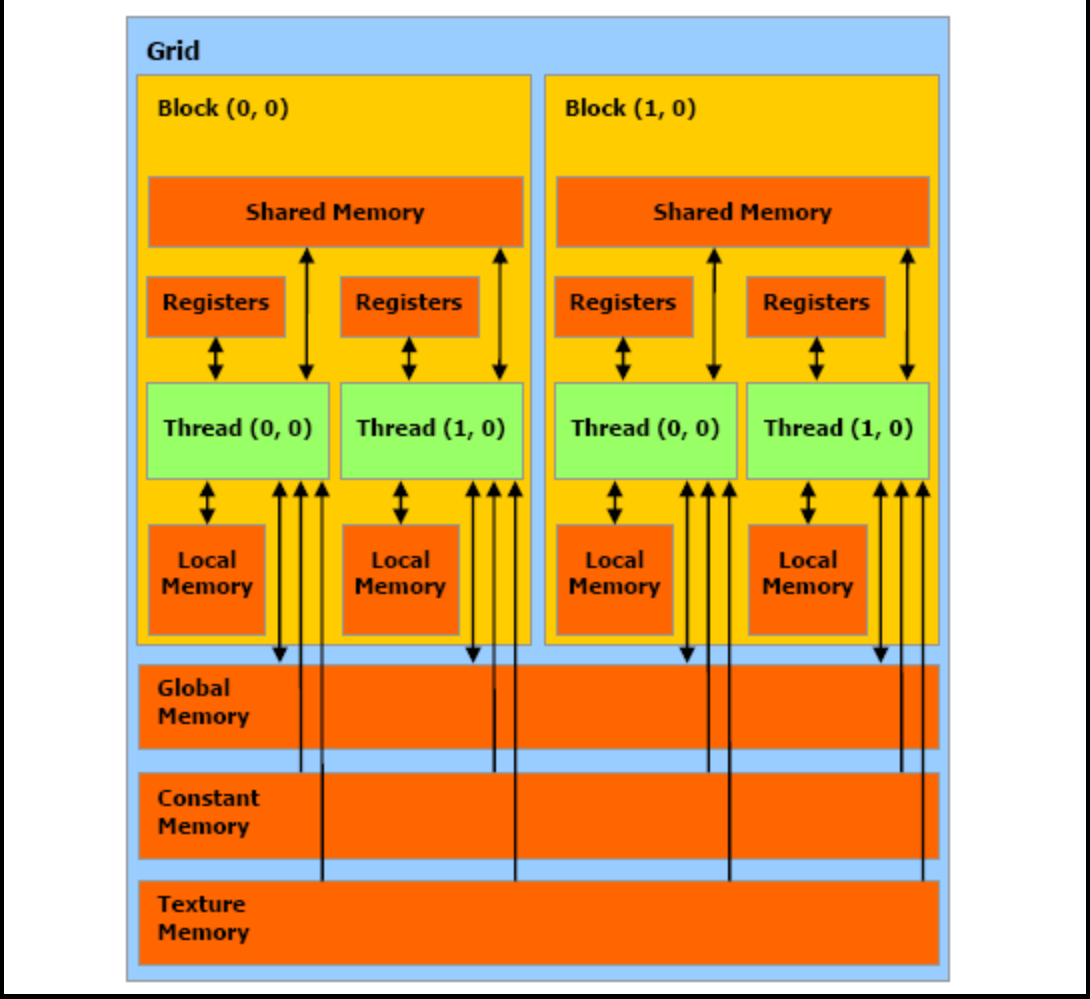


Image courtesy: NVIDIA CUDA Programming Guide, 1.1