Current Generation Parallelism In Games

Jon Olick
id Software

SIGGRAPH2008

Beyond Programmable Shading: In Action

Brief History of Parallelism

- **1 Processor**
  - The good old days.
  - Why parallelize? Just wait a little and your programs will get faster.

Brief History of Parallelism

SIGGRAPH2008

- **2 to 3 Processors**
  - Logical splitting of game process into pipelined pieces.
    - Game
    - Rendering
    - Sound
    - Loading/Decompression

Brief History of Parallelism

- **About 6 to 8 Processors**
  - The transition to a job scheduling type architecture
  - 1^{st} order parallelism
    - Game
    - Rendering
    - Sound
    - Physics
    - Collision
    - Loading/Decompression
    - Etc…

Brief History of Parallelism

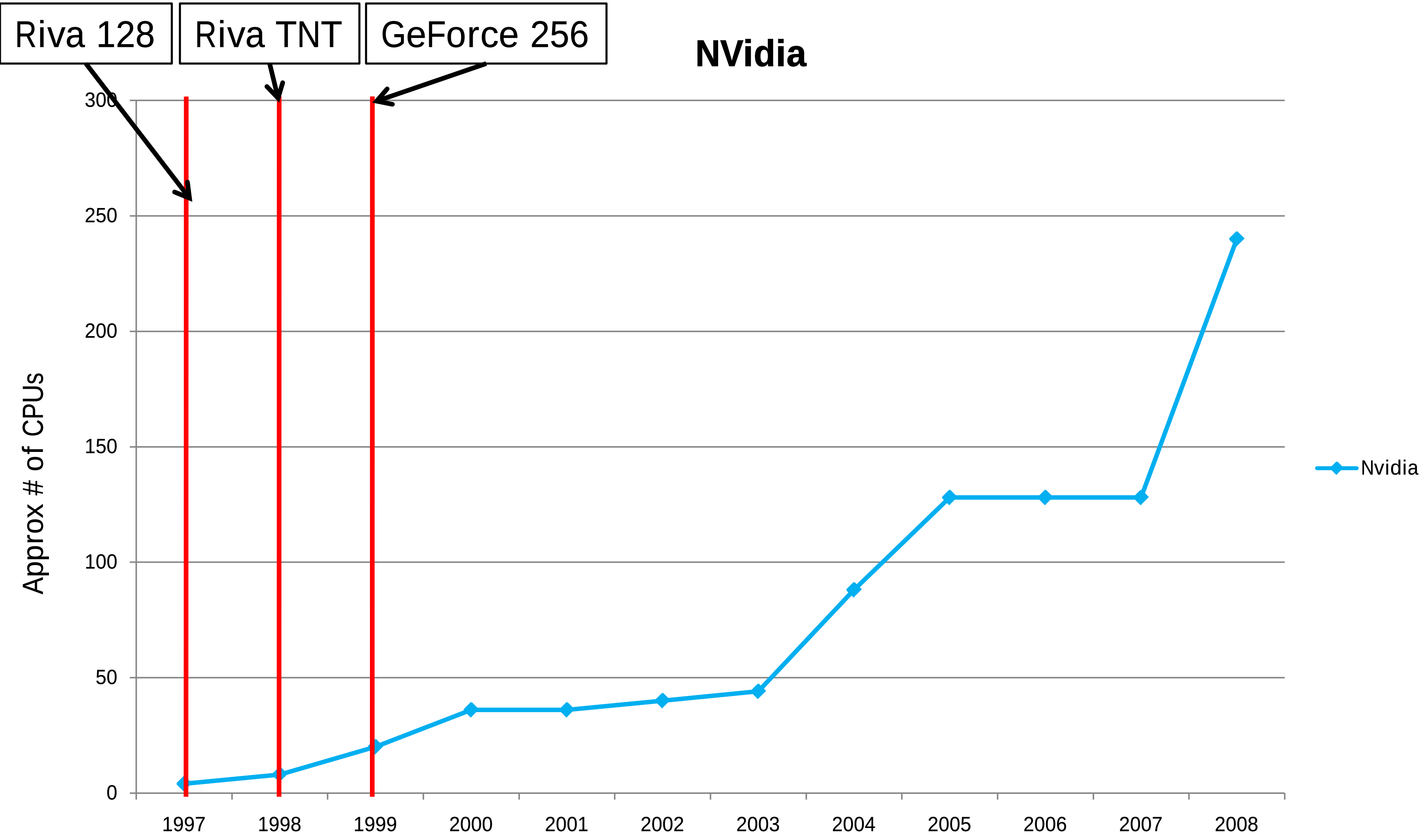
- **About 8 to 16 Processors**
  - End of CPU history.
  - Enter 1998 in GPU history.
    - Approx # of processors as average parallel scalar operations.
  - 2nd order parallelism
  - Jobs which create and manage the resources of other jobs.
    - GPU Command Processor (DMA engine)

Brief History of Parallelism

- **About 16+ processors**
  - 3rd order parallelism
  - GPU Vertex Processors
  - Jobs which create and manage the resources of other jobs which create and manage the resources of other jobs

Brief History of Parallelism

SIGGRAPH2008



Current State of Parallelism

- **Desktop Processors**
  – Intel Core 2 Quad, 4 processors, 3.2 ghz, 102 Gflops
    - Soon to be 8 core?
- **Multimedia Processors**
  – Cell Processor - 8 processors - 3.2 ghz - 192.0 Gflops
    - 1 main, 7 co-processors
- **Graphics Accelerators**
  – GTX 280 - 1.296 ghz – 0.933 Tflops
    - 240 stream processors

SIGGRAPH2008

CELL BROADBAND ENGINE™

PLAYSTATION®3 Processor Overview

SIGGRAPH2008

- Game
- Animation
- Geometry Processing
- Post Processing
- Occlusion Rasterization
- Sorting
- Collision Detection
- Fourier Transform
- (De)Compression
- Not going to cover all of these...

PLAYSTATION®3 Processor Overview

- Parallelize ordinarily sequential CPU processing
- Assist in what is typically considered GPU processing

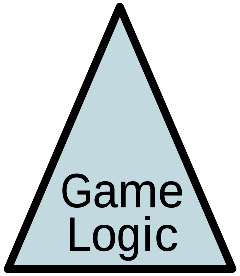
Primary Programming Challenges

SIGGRAPH2008

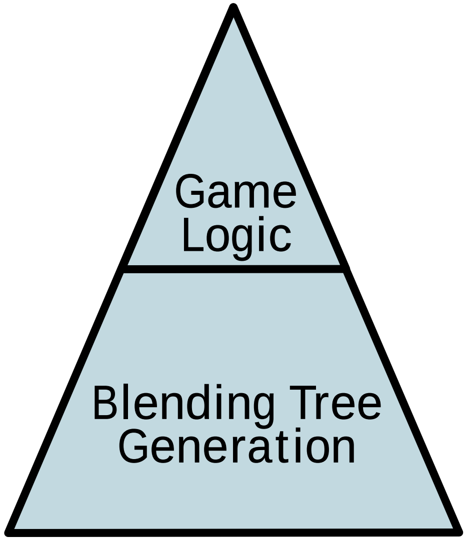
- **Fitting code and data in the 256k local co-processor memory**
  – Best solutions are ones that don't treat the 256k local store as a typical on demand caching architecture
    - Scattered reads/writes bad, sequential reads/writes good
- **Software Pipelining**
- **Only 16 byte aligned reads/writes**
- **Synchronization**

MD6 ANIMATION PROCESSING

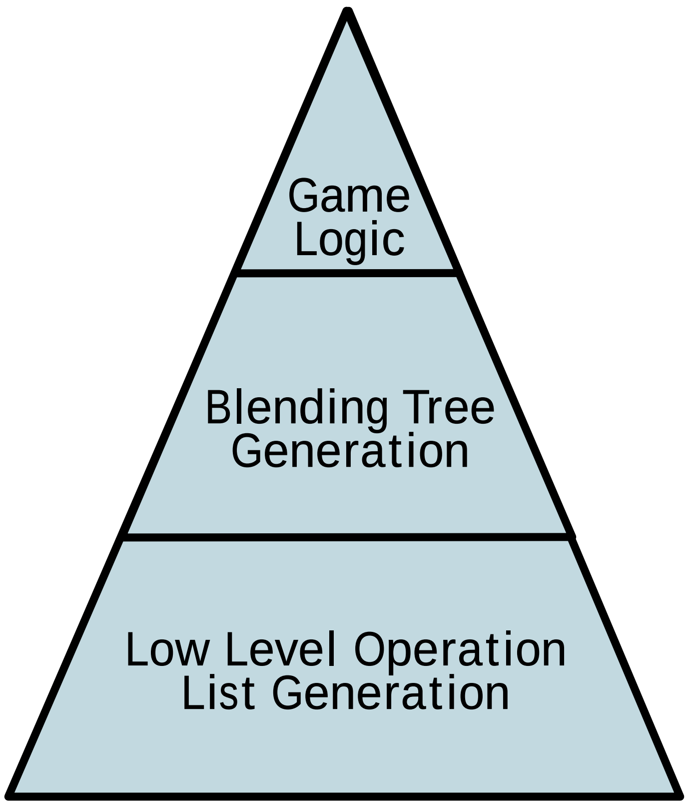
MD6 Animation Processing



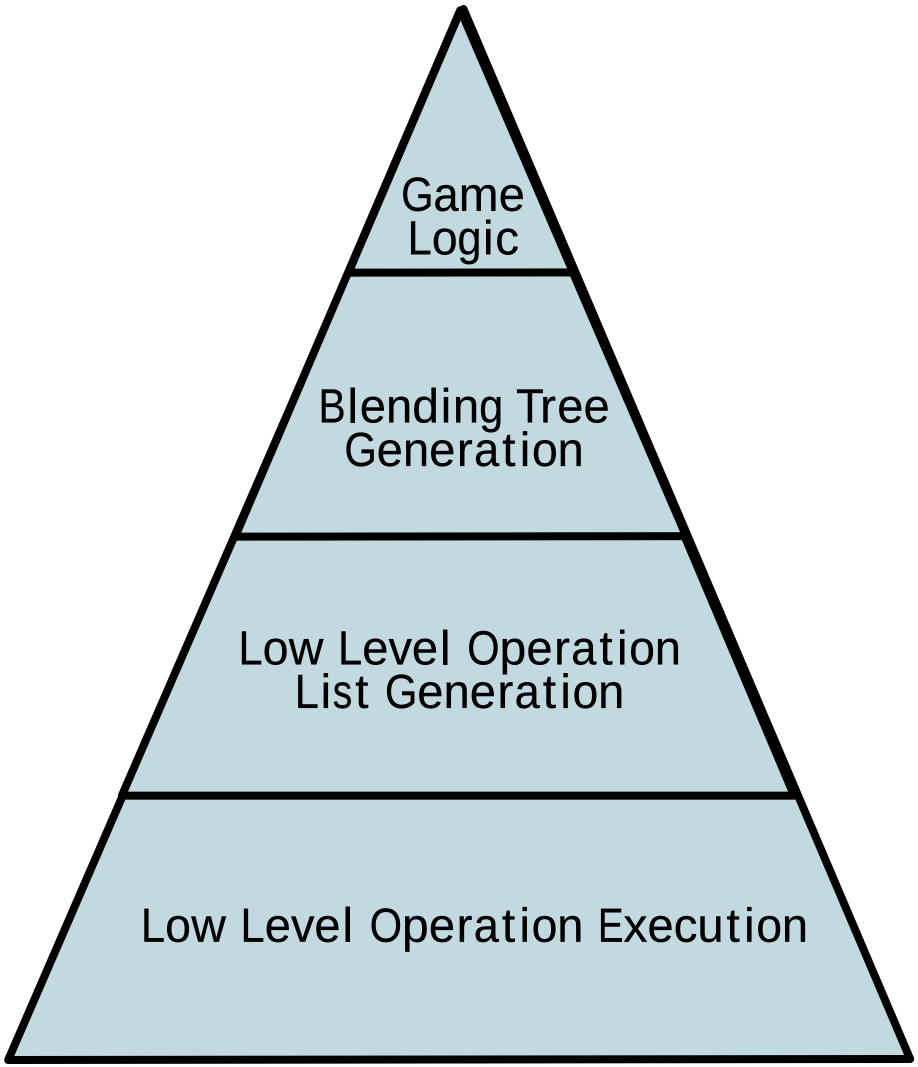
MD6 Animation Processing



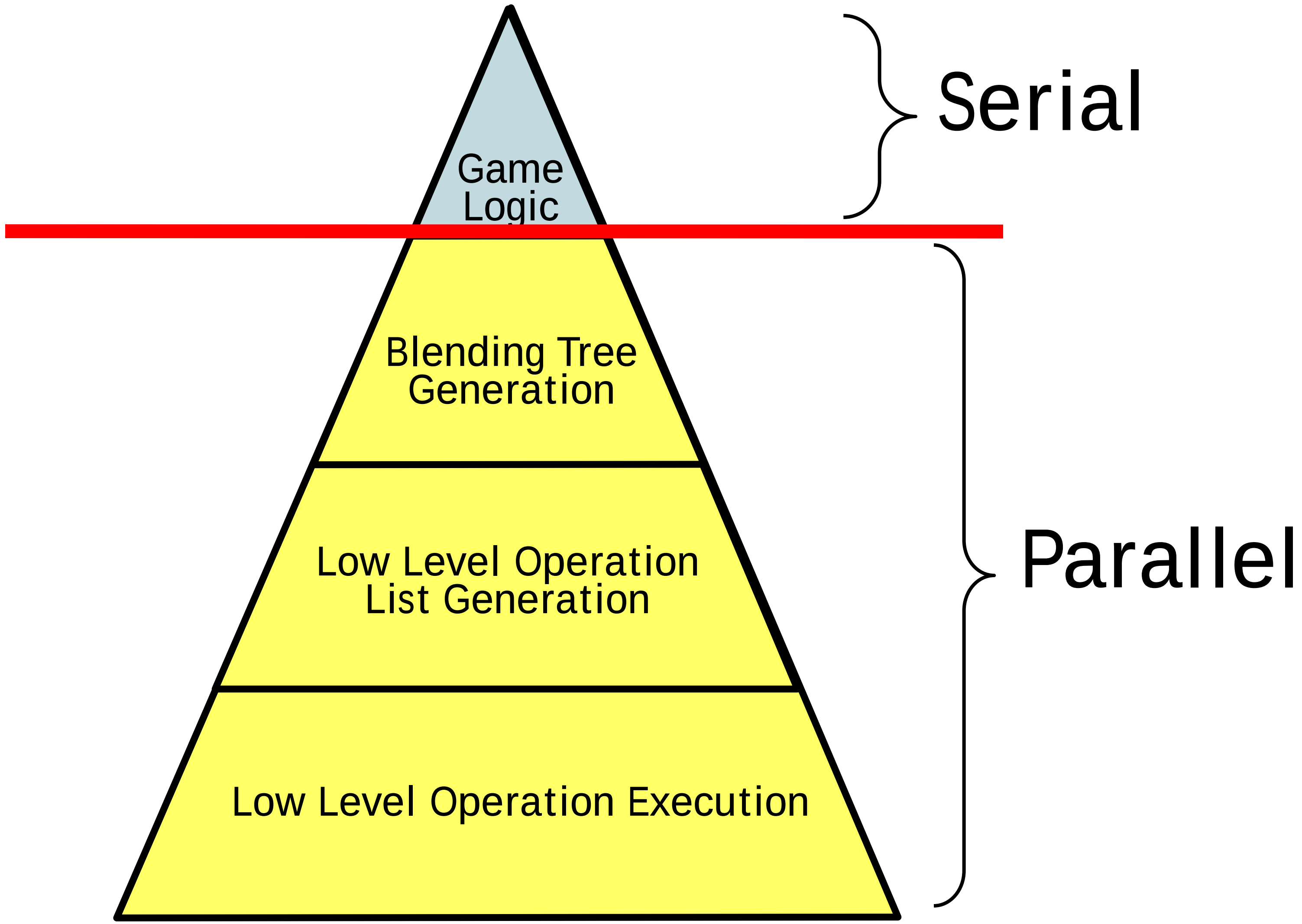
MD6 Animation Processing



MD6 Animation Processing



MD6 Animation Processing



MD6 Animation Webs

SIGGRAPH2008

- **Separates Thinking from Representation**
  - Game Object says what it wants to look like.
  - Animation Webs take care of the rest.
- **Unstructured graph**
  - Each node has a blend tree
- **Designed with simplicity in mind**
  - Animators should animate, not fiddle with nodes.
  - Extract as much information as possible directly from the animation data.

MD6 Animation Processing

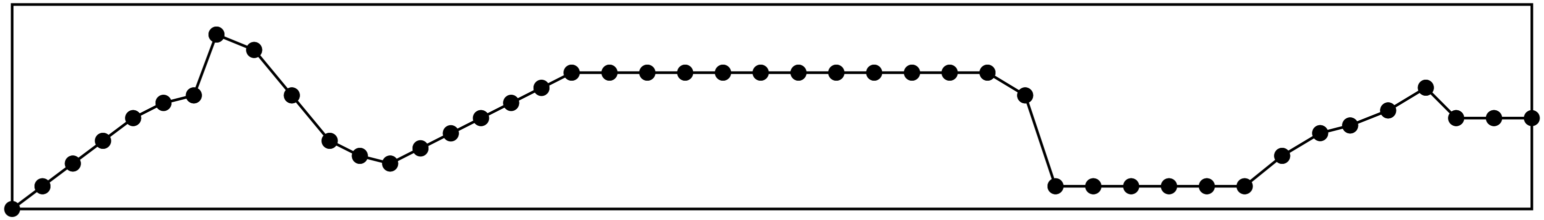
- **Additive Blending**
- **Subtractive Blending**
- **Animation Algebra**
  - Blend Equations
    - Animation blending trees in the form of an equation.
    - Example equation:
      - (animA + animB) – animC

Partial Animation Blending

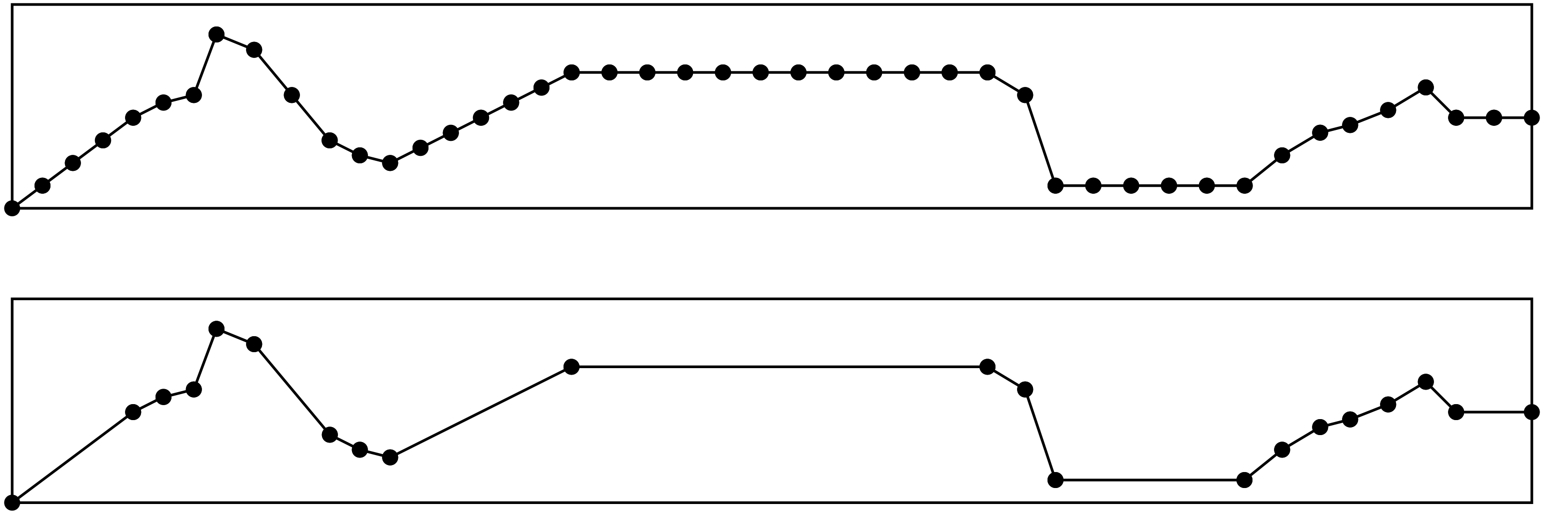
SIGGRAPH2008

- **Generalized play an animation only on the face, torso, etc…**
- **One weight per joint per animation**
- **Compute alpha for slerp via following equation:**
  - For each joint
    - Let w0 = weight of joint in animation A
    - Let w1 = weight of joint in animation B
    - If(w1 > w0)
      - Let alpha = (alpha * w1) / w0
    - Else
      - Let alpha = ((w1 – w0) + alpha * w0) / w1

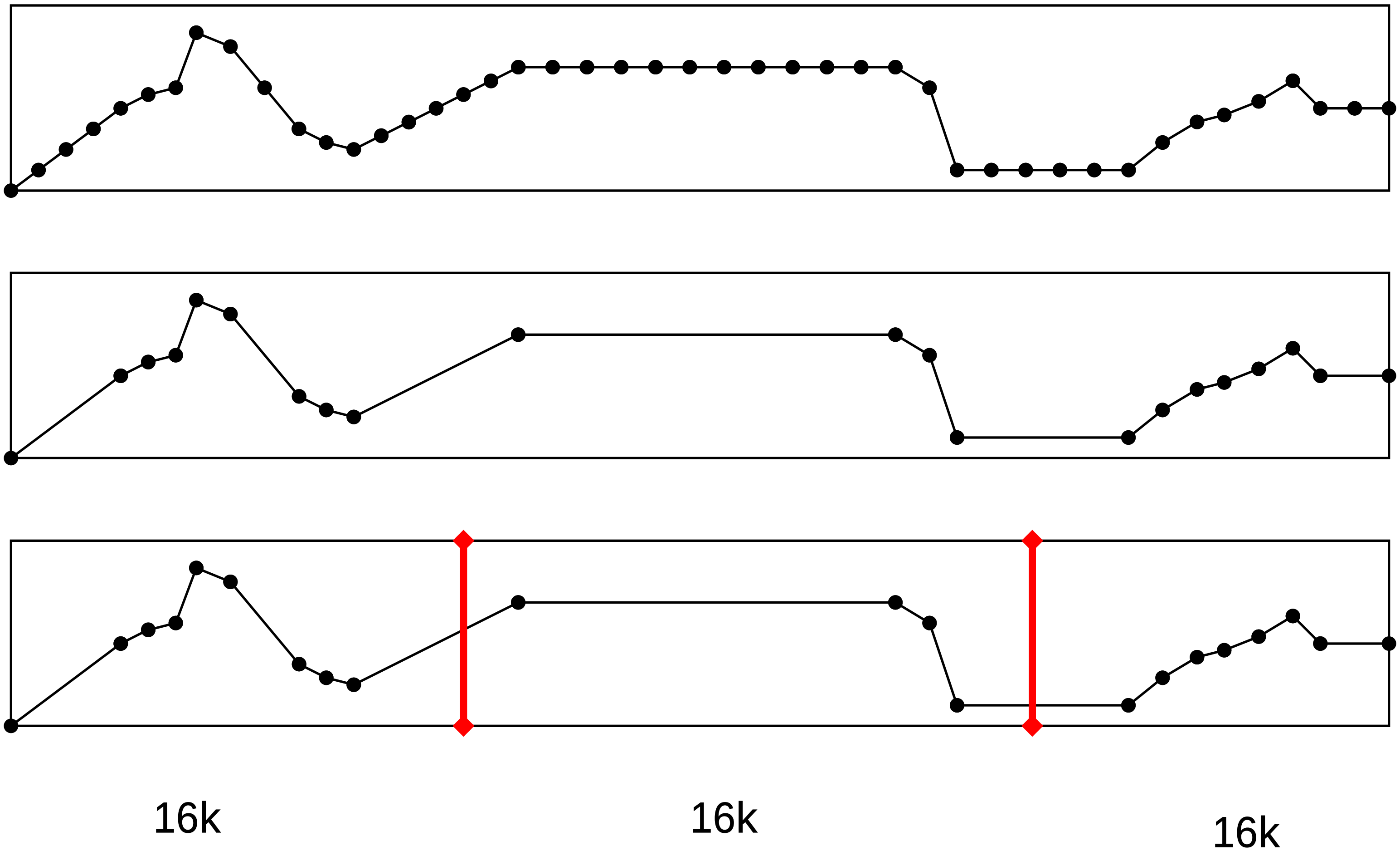
Varying parameter treatment



Varying parameter treatment



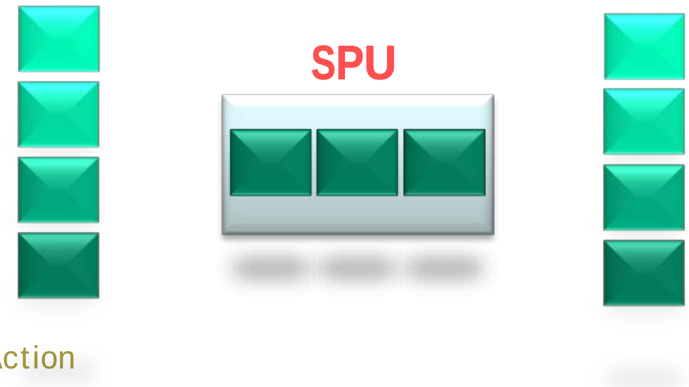
Varying parameter treatment



GEOMETRY PROCESSING

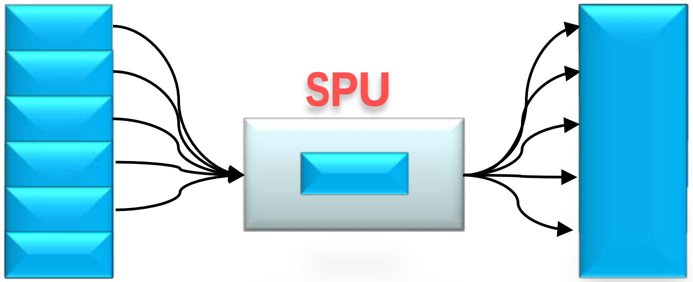
Two modes of usage

- **Primary mode**
  - Use offline tools
  - Partition into vertex sets
  - Use indexed triangles
  - All features of pipeline can be used



Two modes of usage (cont)

- **Secondary mode**
  - Data generated by other tools
  - Formats other than indexed triangles
  - Non-partitioned objects
  - Subset of pipeline features can be used



SPU Geometry Pipeline Stages

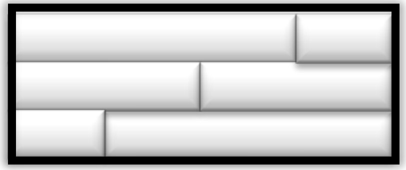
SPU Pipeline

- Vertex Decompress
- Index Decompress
- Blend Shapes
- Skinning
- Progressive Mesh
- Triangle Culling
- Compression
- Output

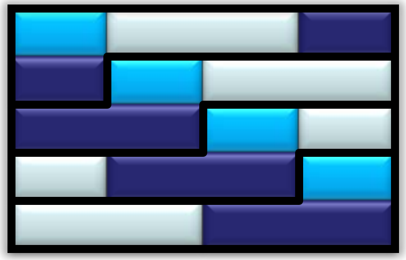
Vertex Decompression

Vertex Attributes

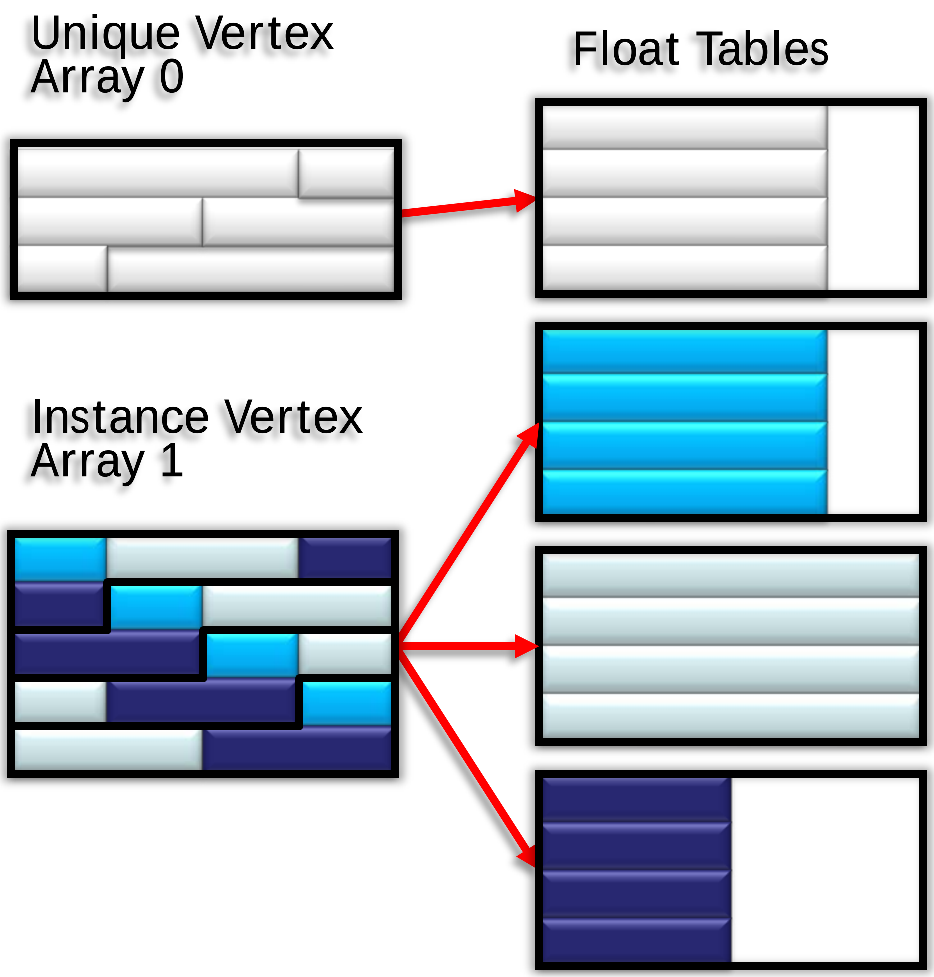
Unique Vertex
Array 0



Instance Vertex
Array 1



Vertex Decompression



24bit Unit Vector

- **Smallest 2 compression**
  - Two smallest components with 10 bits each
    - Encoded from –sqrt(2)/2 to +sqrt(2)/2
  - Largest component reconstructed via
    - Largest = sqrt(1 – smallestA^2 – smallestB^2)
    - One additional bit for sign of largest component.

24bit Unit Vector

SIGGRAPH2008

- **Smallest 2 compression**
  - Two smallest components with 10 bits each
    - Encoded from –sqrt(2)/2 to +sqrt(2)/2
  - Largest component reconstructed via
    - Largest = sqrt(1 – smallestA^2 – smallestB^2)
    - One additional bit for sign of largest component.
- **One more bit to represent W as +1 or -1**
  - For constructing bi-normal from normal and tangent.

N-bit Fixed Point with integer offsets

- **Simple n.x fixed point values**
  - Per-segment integer offset
- **Bit count may vary from attribute to attribute**

Index Decompression

SPU Pipeline

- Vertex Decompress
- **Index Decompress**
- Blend Shapes
- Skinning
- Progressive Mesh
- Triangle Culling
- Compression
- Output

Index Table Construction

- **Index table is created by a vertex cache optimizer**
  - Based on K-cache algorithm
- **Number of vertex program outputs affects Vertex Cache size.**
- **Four vertex mini cache most important optimization factor**

Index Cache Optimizer

SIGGRAPH2008

- **Our vertex cache optimizer produces very regular index data**

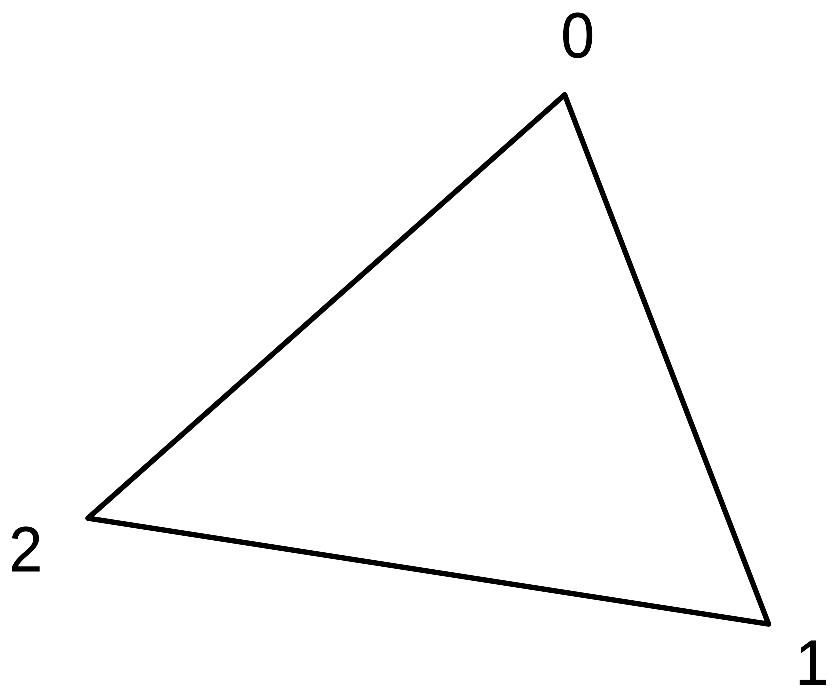
0	**1**	**2**
0	2	3
3	2	4
3	4	5
6	7	8
9	6	8
10	9	11
9	8	11

Index Decompression

- **Our vertex cache optimizer produces very regular index data**

0	**1**	**2**
0	2	**3**
3	2	**4**
3	4	**5**
6	**7**	**8**
9	6	8
10	9	**11**
9	**8**	11

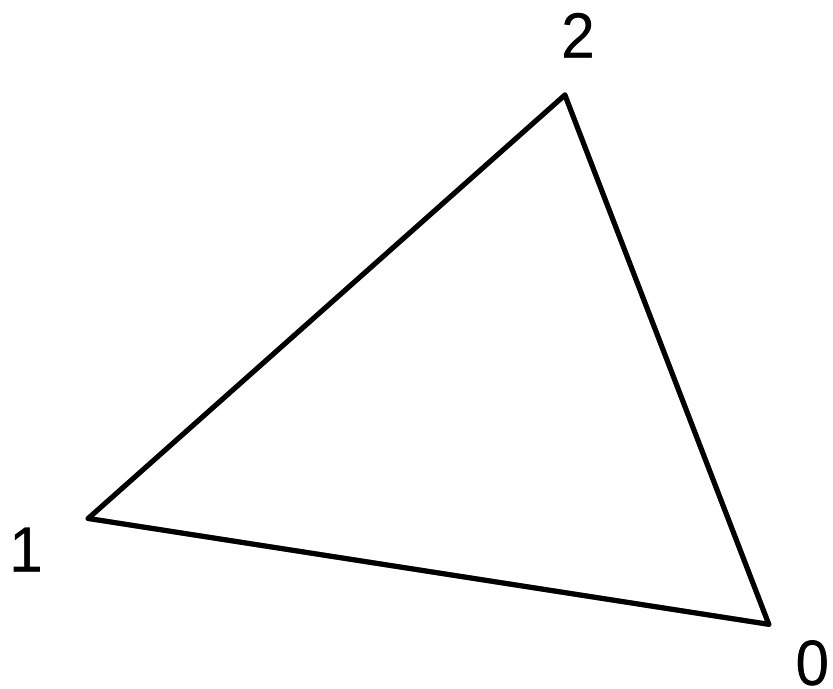
Index Decompression



Triangle Indexes

0	1	2

Index Decompression



<u>Triangle Indexes</u>

2	0	1

Index Decompression

SIGGRAPH2008

- **Before Rotation**

0	**1**	**2**
0	2	**3**
3	2	**4**
3	4	**5**
6	**7**	**8**
9	6	8
10	9	**11**
9	**8**	11

Index Decompression

SIGGRAPH2008

- **After Rotation**

0	**1**	**2**
0	2	**3**
3	2	**4**
3	4	**5**
6	**7**	**8**
6	8	**9**
9	**11**	**10**
11	9	**8**

Index Decompression

00	PREVIOUS INDEX 0	PREVIOUS INDEX 2	NEW INDEX
01	PREVIOUS INDEX 2	PREVIOUS INDEX 1	NEW INDEX
10	PREVIOUS INDEX 1	PREVIOUS INDEX 0	NEW INDEX
11	NEW INDEX	NEW INDEX	NEW INDEX

Index Decompression

85% compression

6.5 : 1

Blend Shapes

SIGGRAPH2008

Blend Shapes in MLB® 08 The Show

Skinning

SPU Pipeline

- Vertex Decompress
- Index Decompress
- Blend Shapes
- **Skinning**
- Progressive Mesh
- Triangle Culling
- Compression
- Output

Skinning on SPUs

```
void SkinVs(float4 inPosition : ATTR0, float4 weights : ATTR3,
  float4 matrixIndex : ATTR4,
  out float4 position : POSITION,
  uniform float4 joints[72], uniform float4x4 modelViewProj)
{
  position = 0;
  for (int i = 0; i < 4; i++)
  {
    float idx = matrixIndex[i];
    float3x4 joint = float3x4(joints[idx+0], joints[idx+1],
                              joints[idx+2]);
    position += weights[i] * mul(joint, inPosition);
  }
  position = mul(modelViewProj, position);
}
```

Skinning on SPUs

30% Performance Improvement

Skinning on SPUs

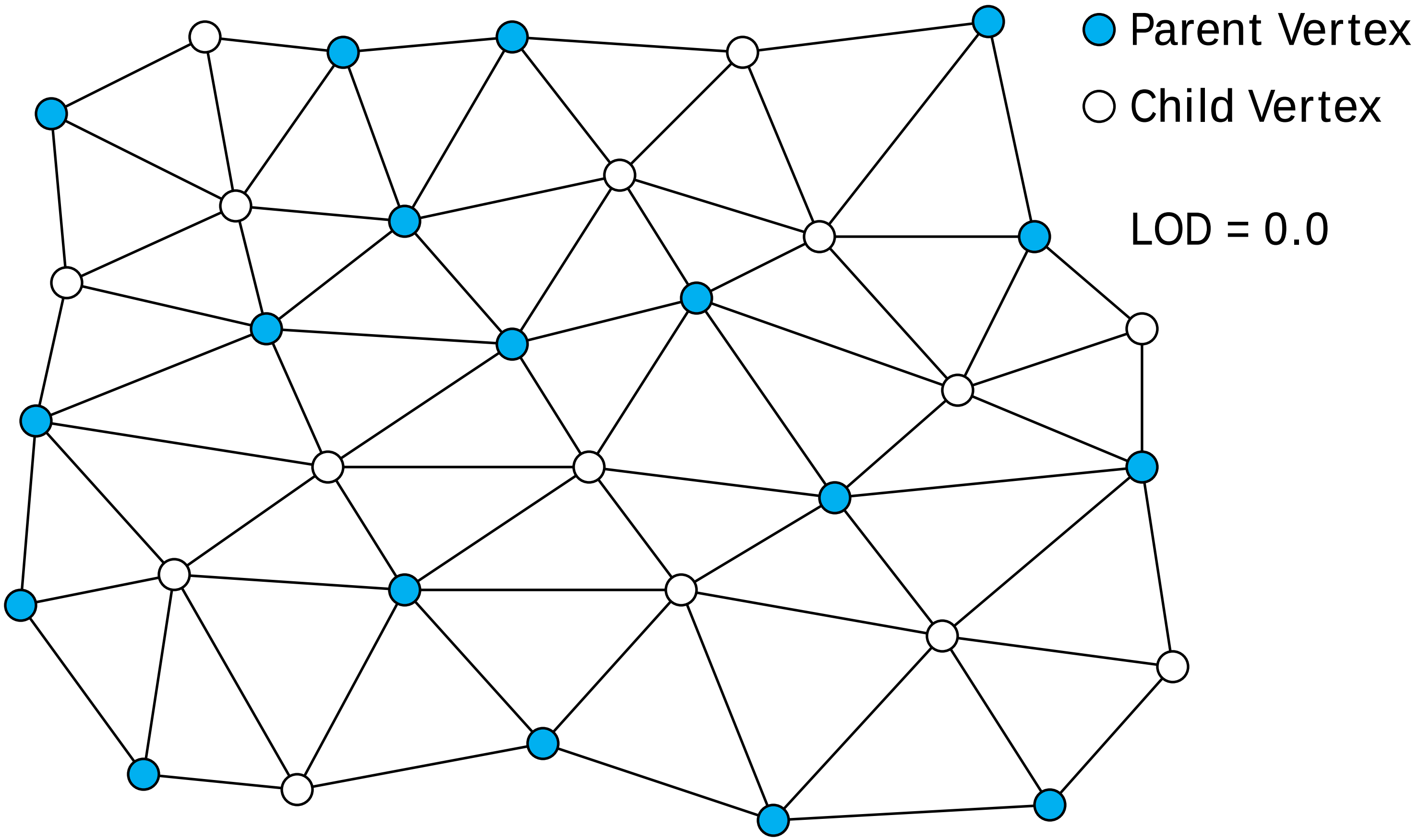
30% Performance Improvement

*Shadow map generation.... **70%!***

Discrete Progressive Mesh

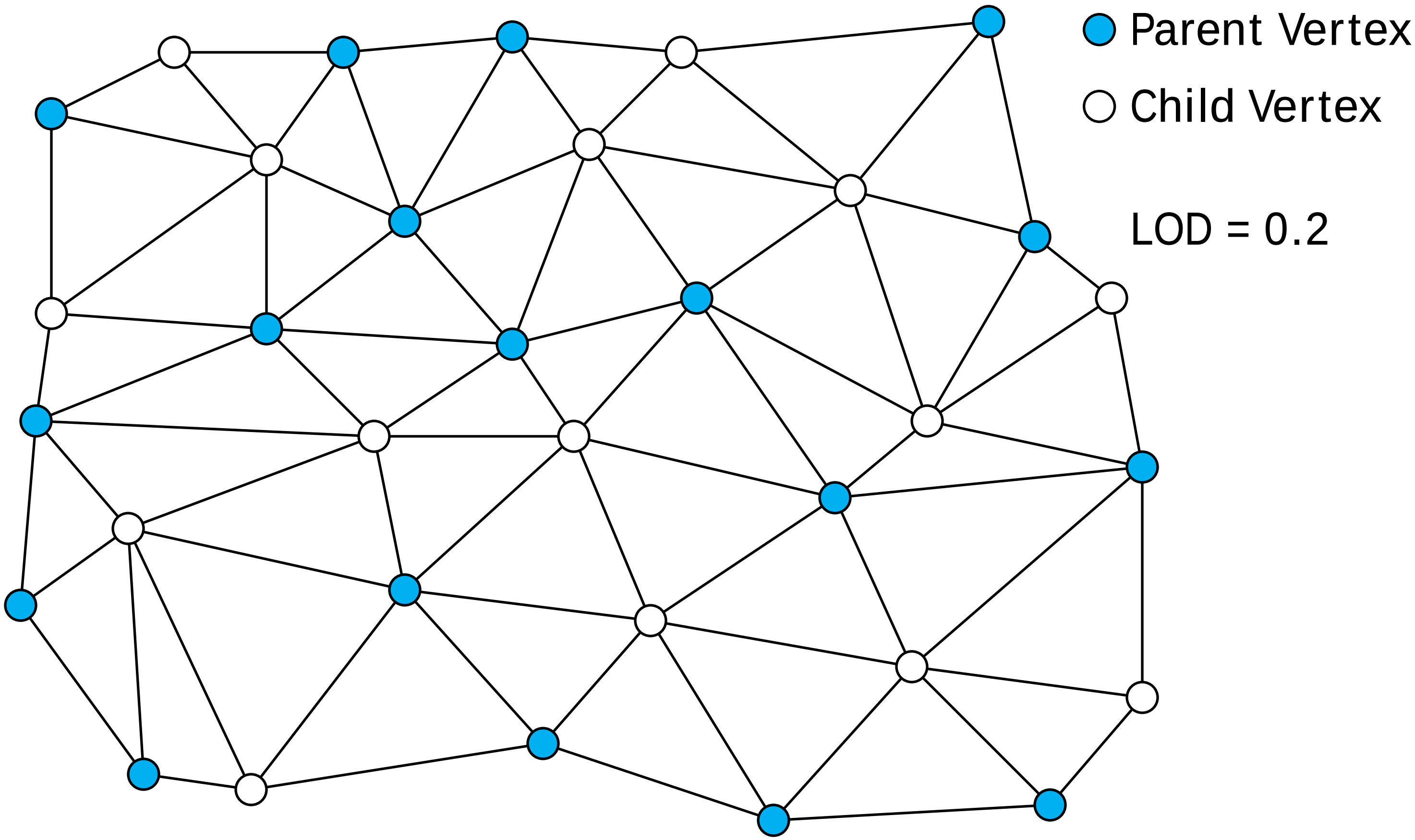
- **Smoothly reduces the triangle count as a model moves into the distance**
- **With discrete progressive mesh, the LOD calculation is done once for an entire object**

At an LOD there are two types of vertexes



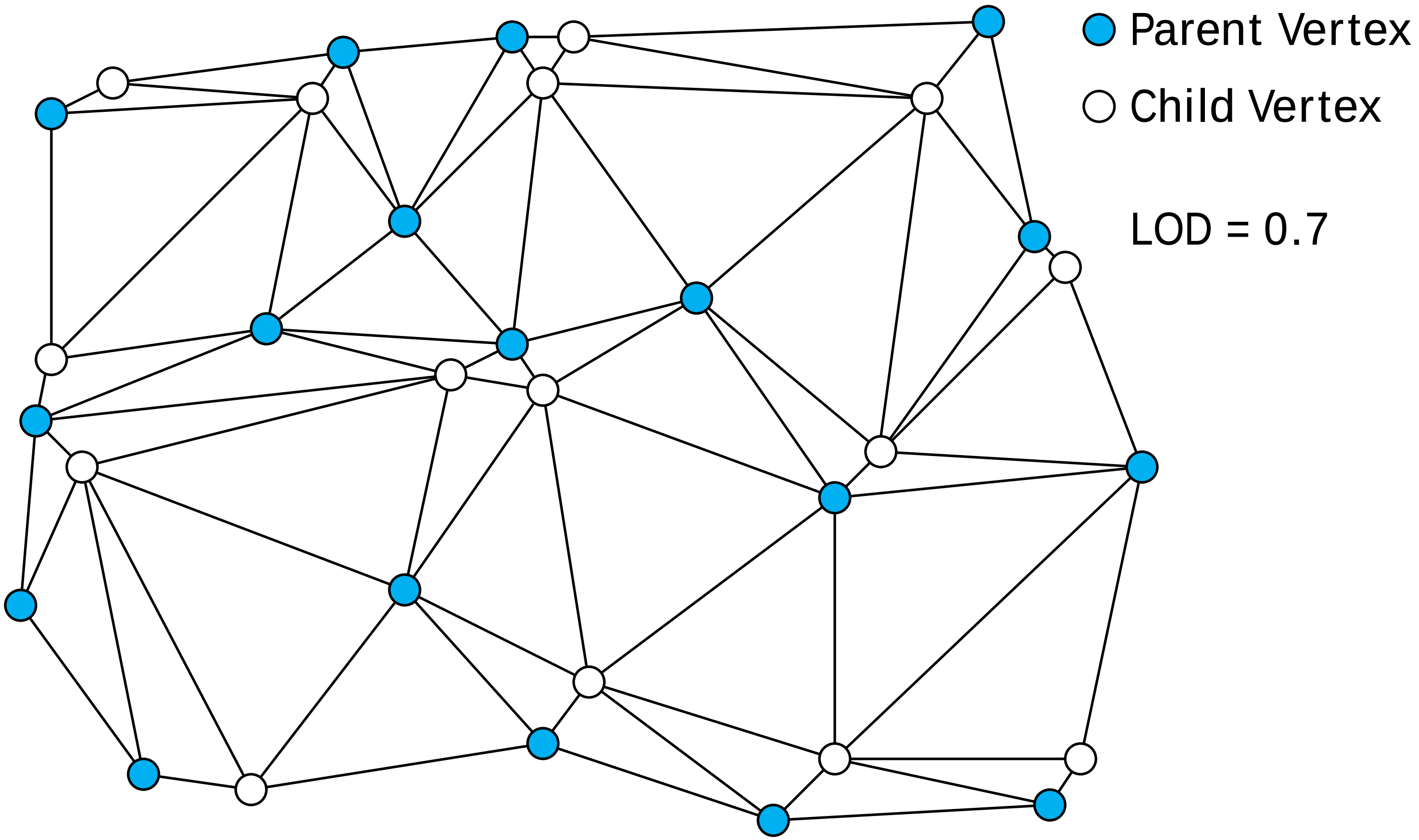
As the LOD level decreases, the children "slide" towards their parents

SIGGRAPH2008

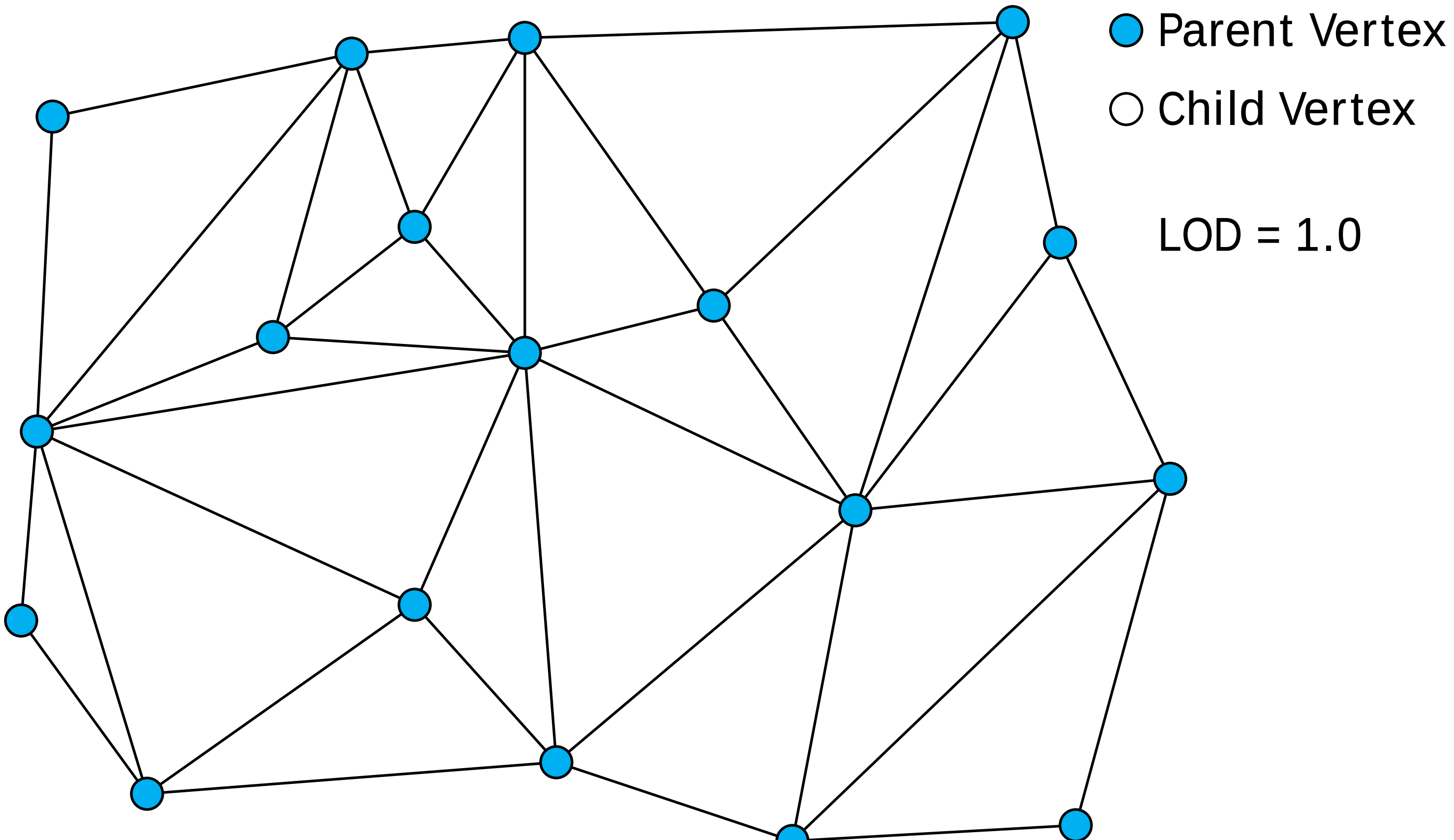


The children continue to move towards their parents

SIGGRAPH2008



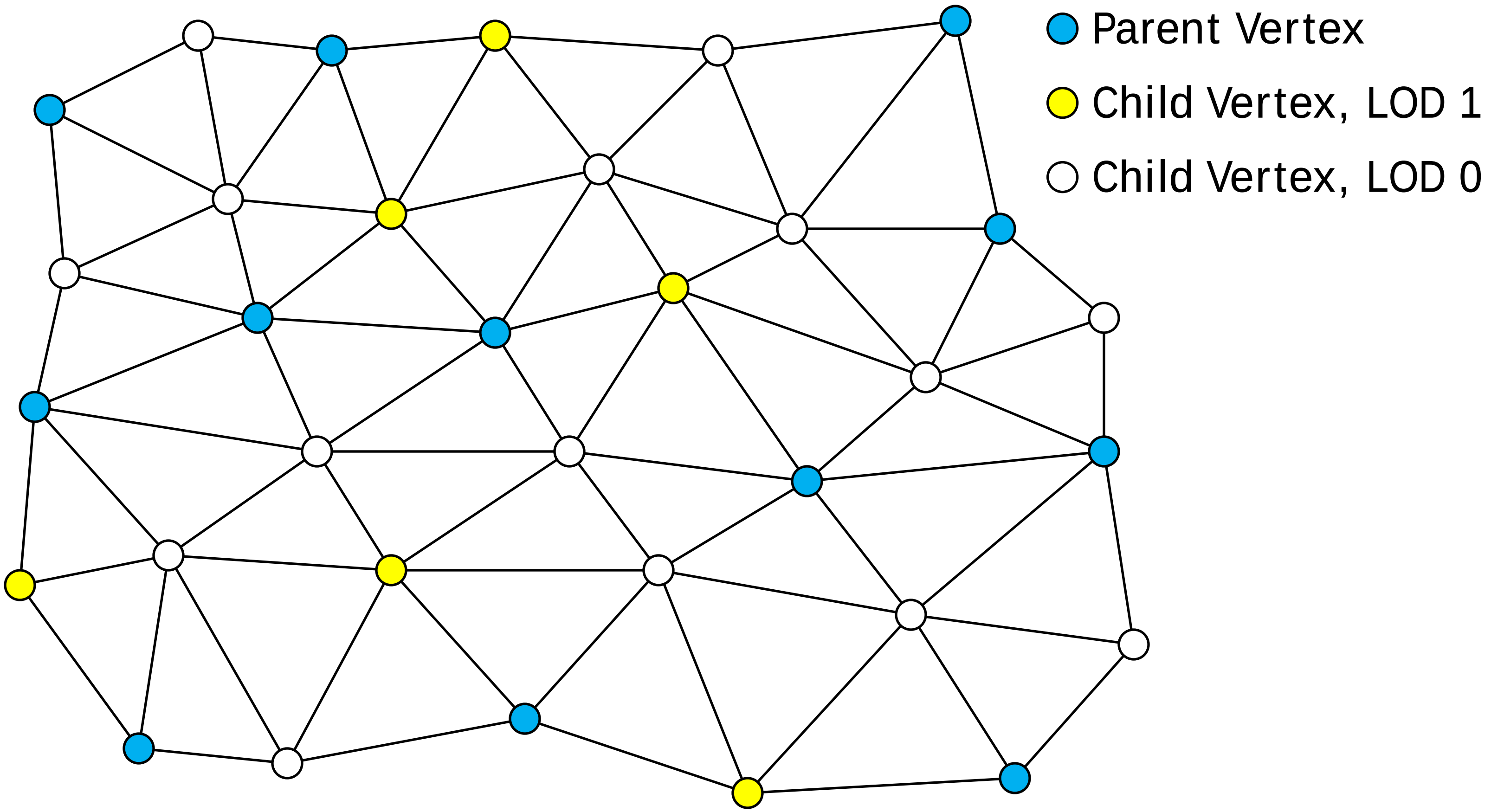
At the next integral LOD, all child vertexes disappear as do the triangles



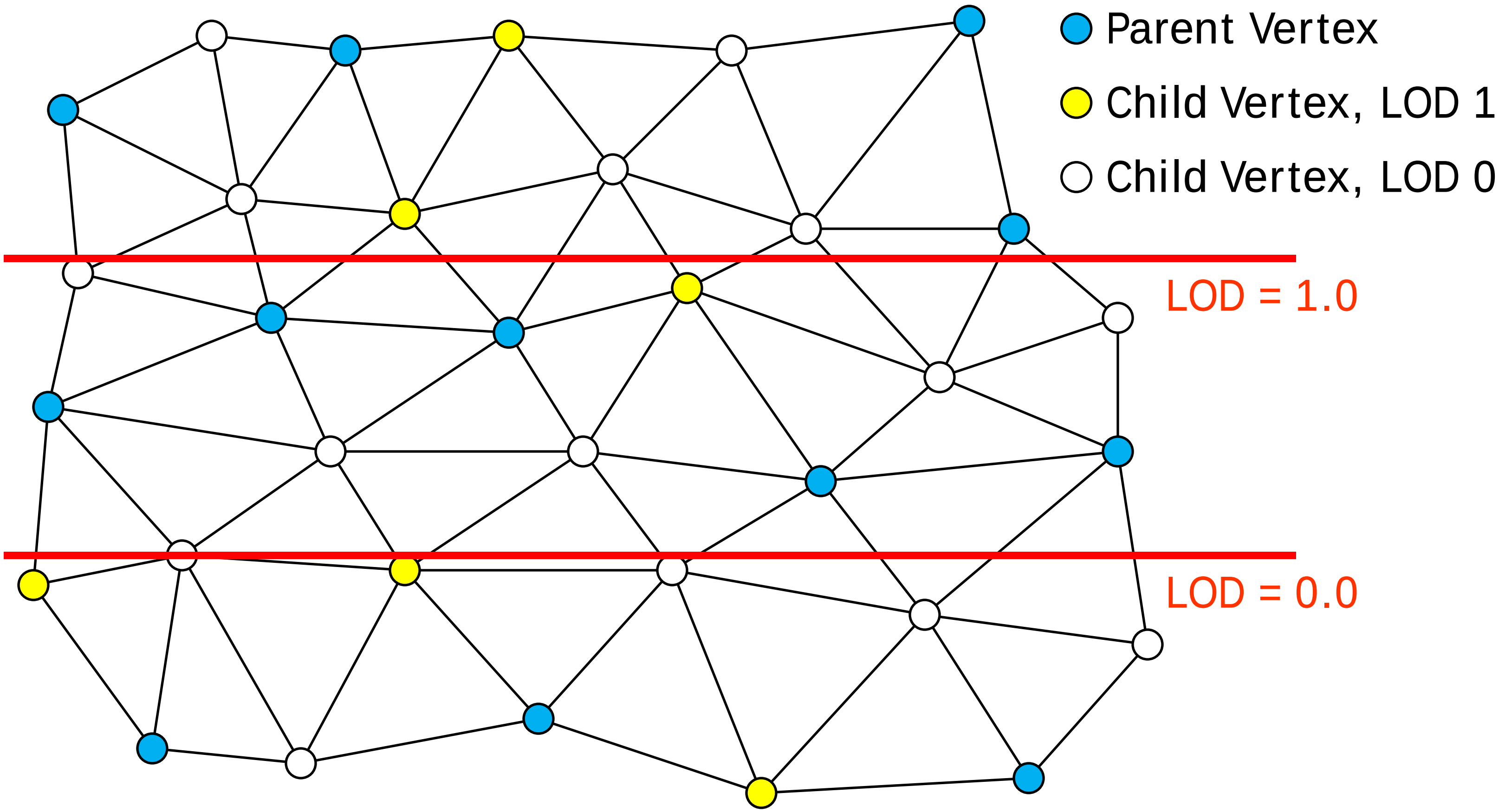
Continuous Progressive Mesh

- **Like discrete progressive mesh, child vertexes move smoothly toward their parents**
- **However, the LOD is calculated for each vertex instead of just once for the object**

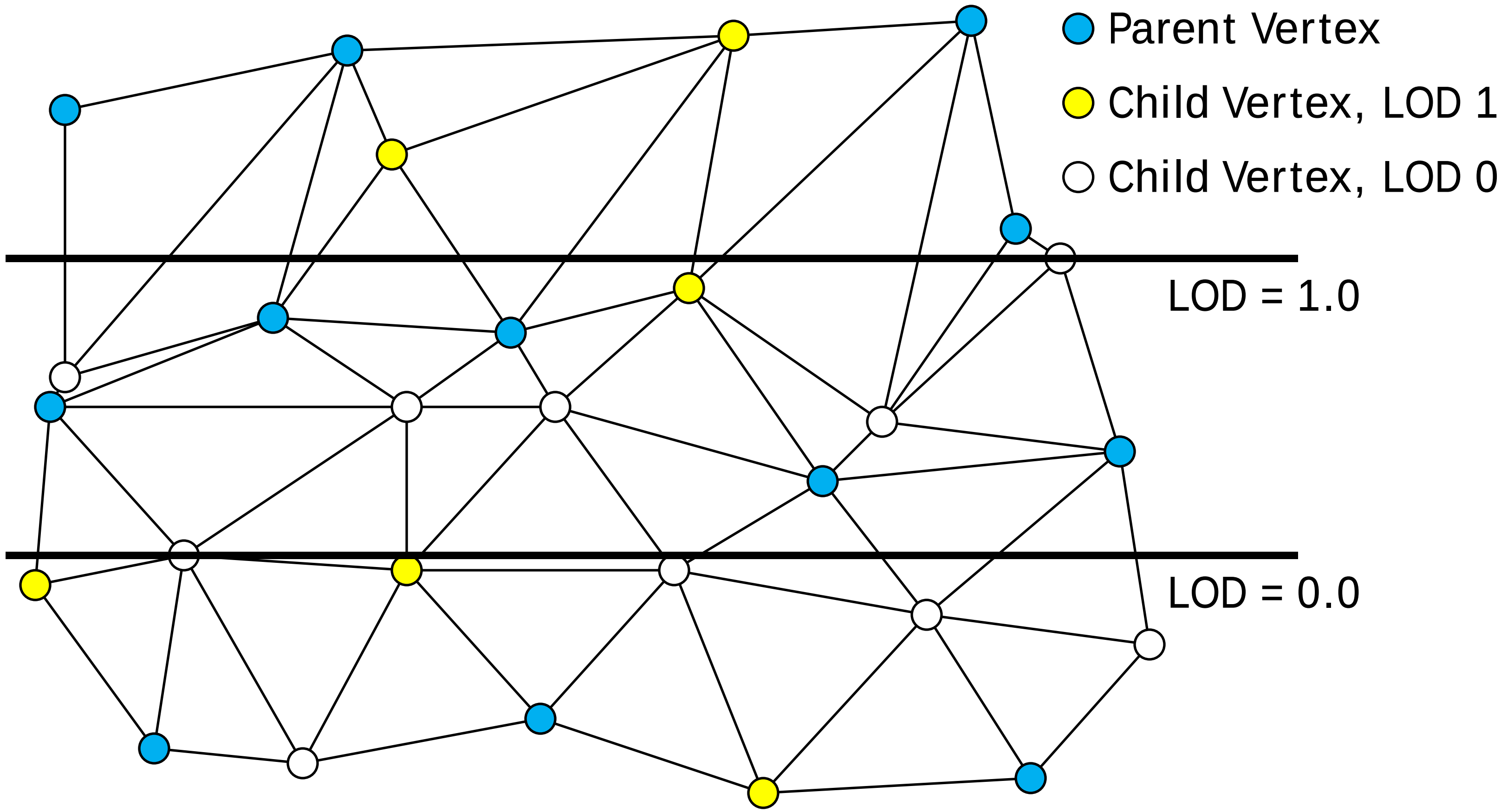
Vertex set about to undergo continuous progressive mesh



A single vertex set can straddle several LOD ranges



Vertexes move depending on their distance

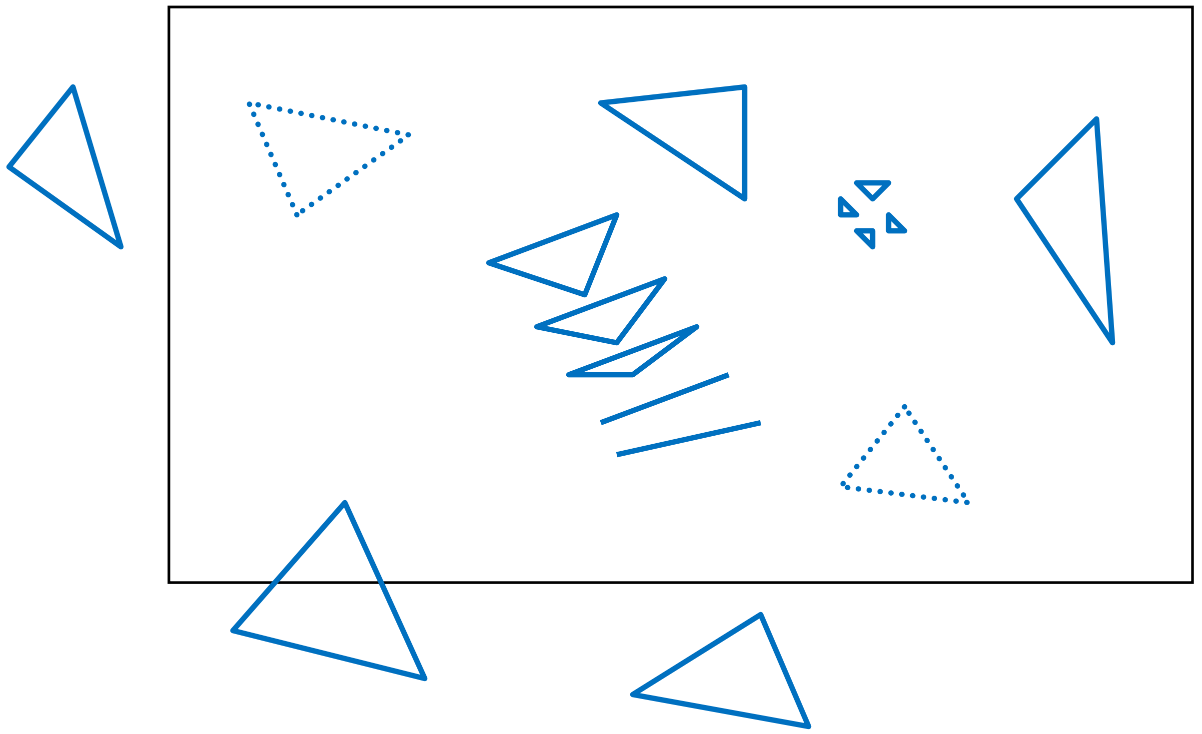


Triangle Culling

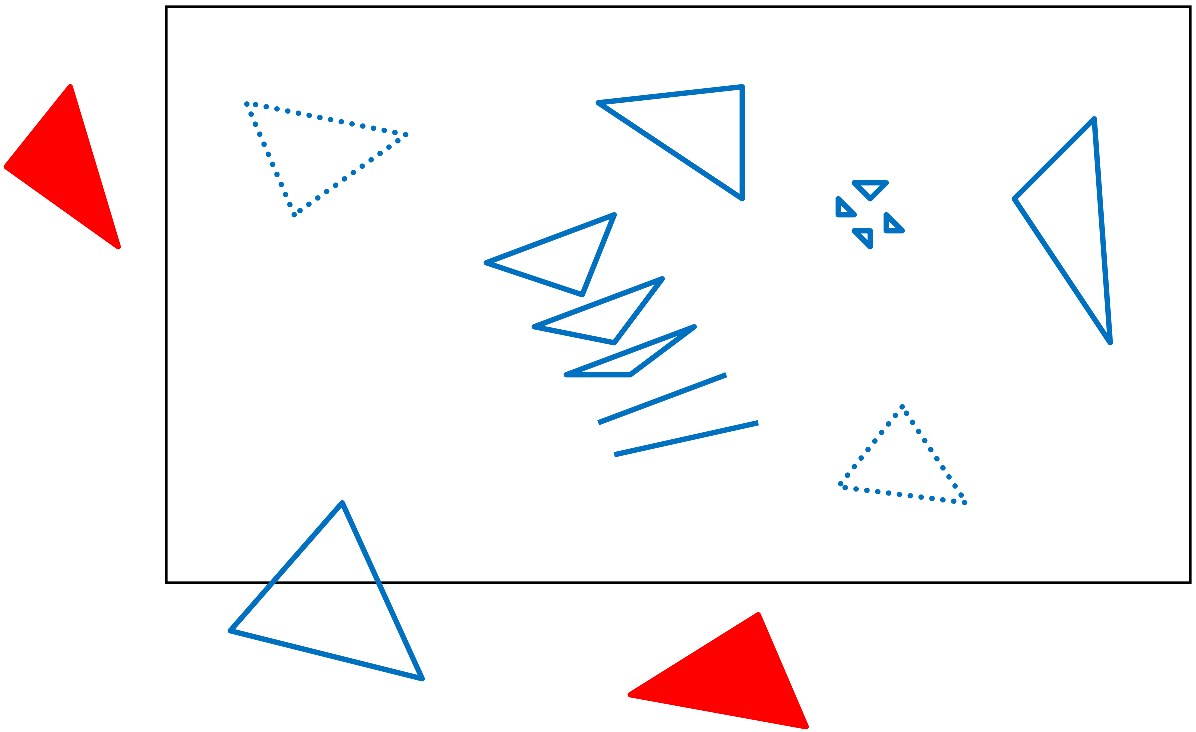
SPU Pipeline

- Vertex Decompress
- Index Decompress
- Blend Shapes
- Skinning
- Progressive Mesh
- Triangle Culling
- Compression
- Output

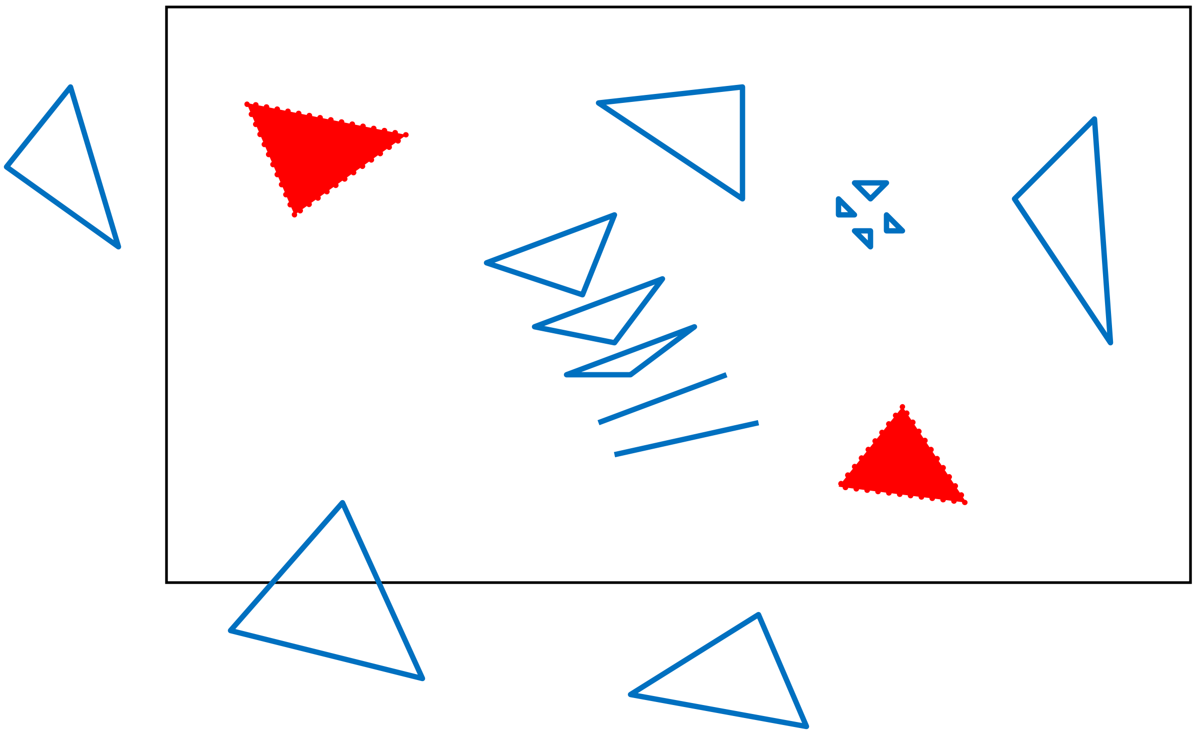
Up to *70%* of triangles do not contribute to final image.



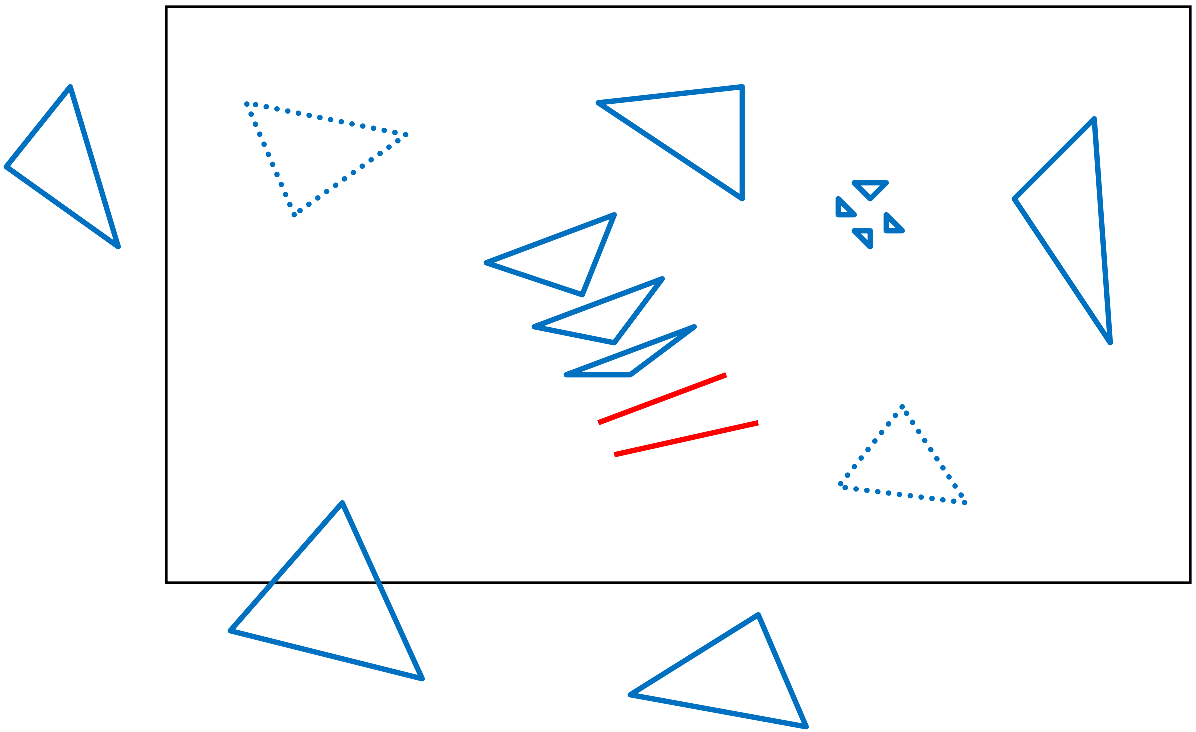
Off Screen Triangles



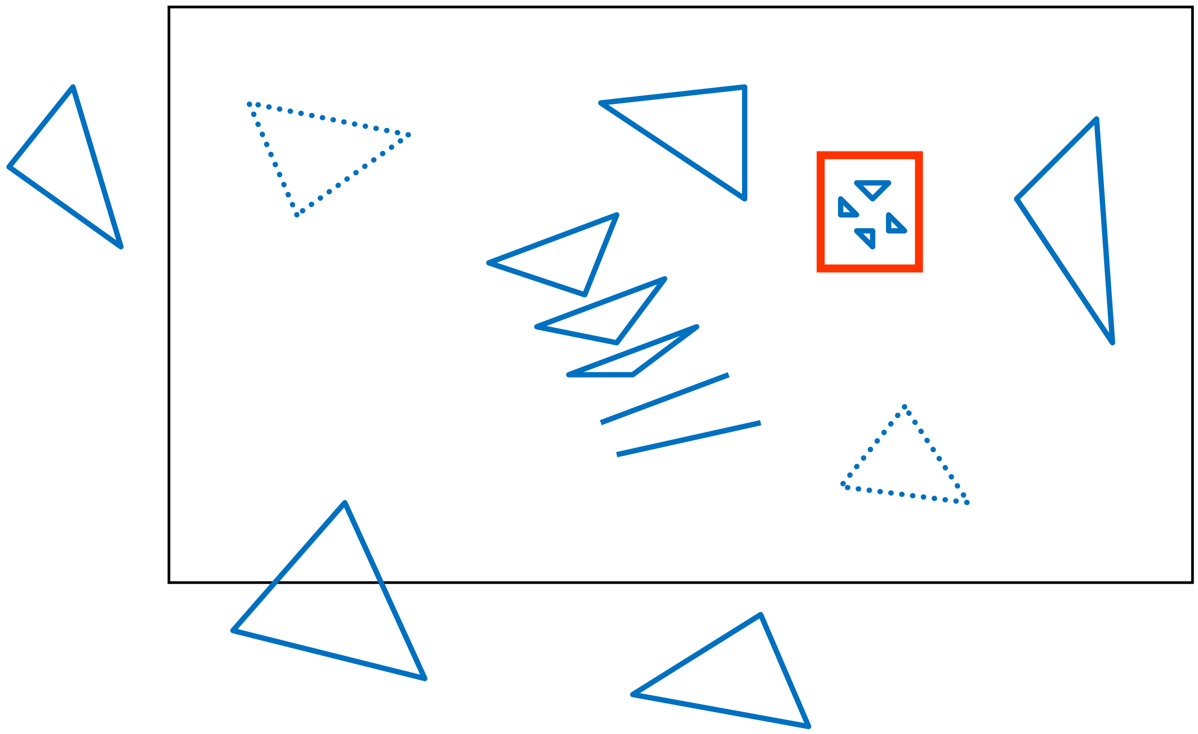
Back Facing Triangles



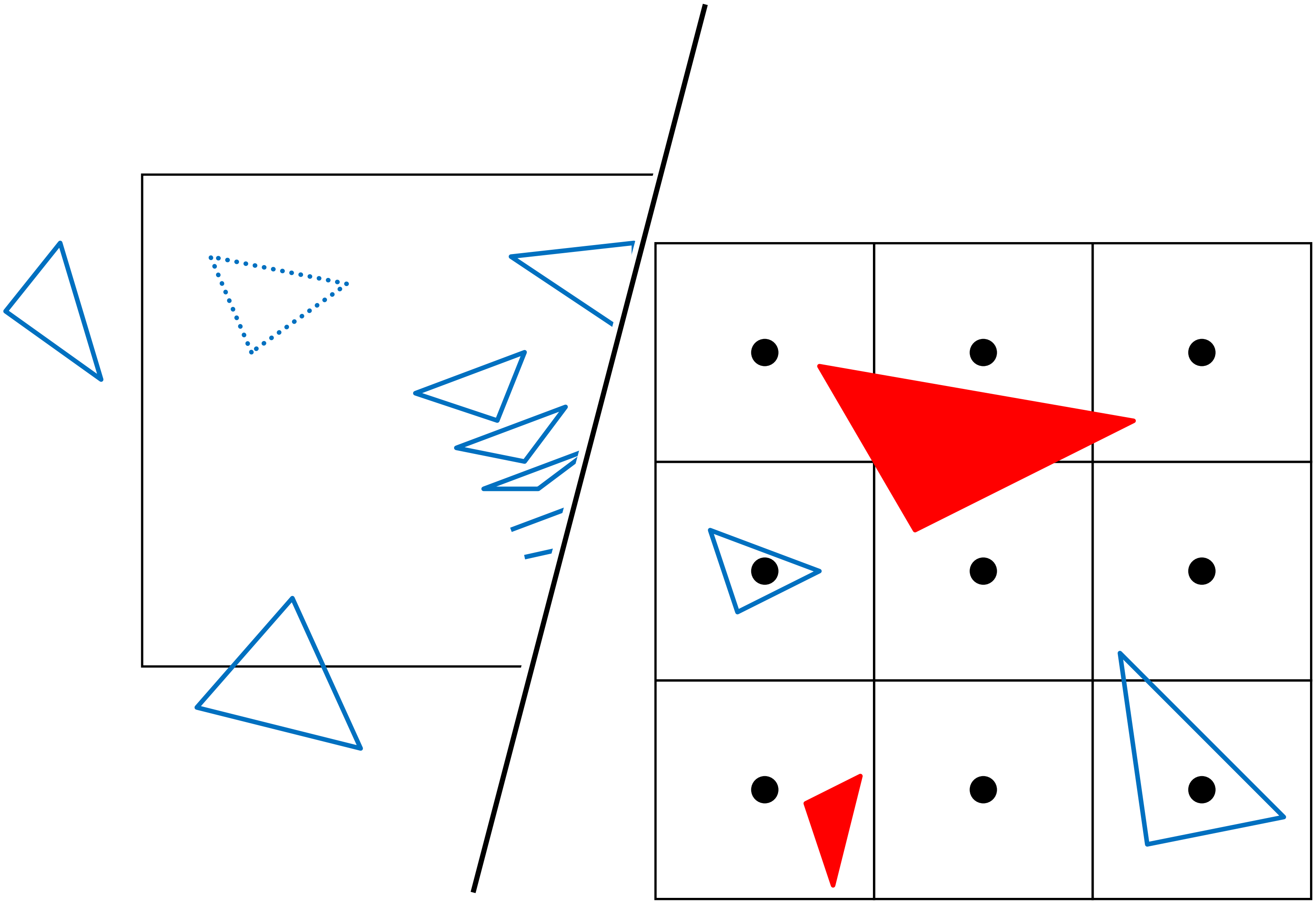
Zero Area Triangles



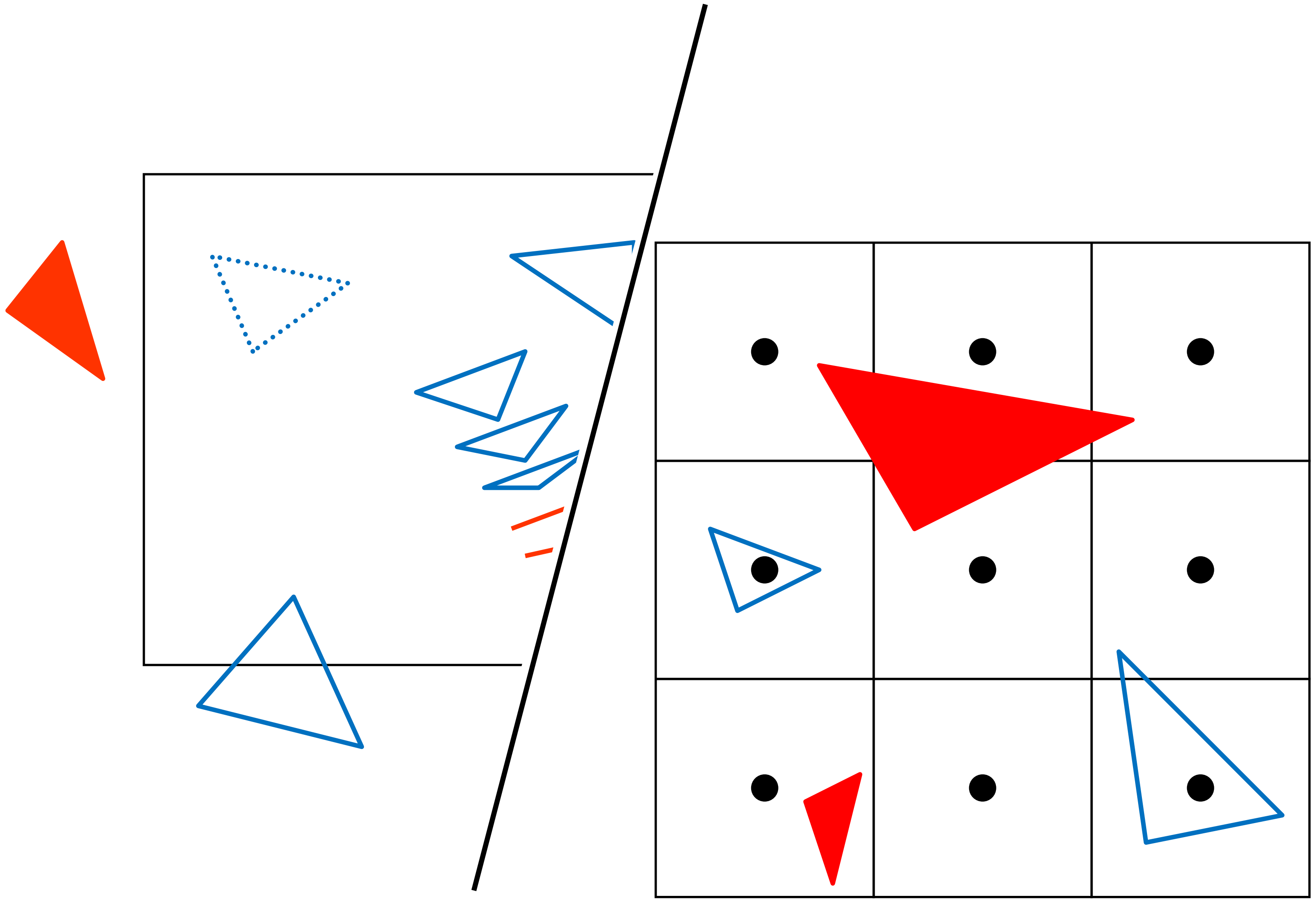
Zero Area Triangles



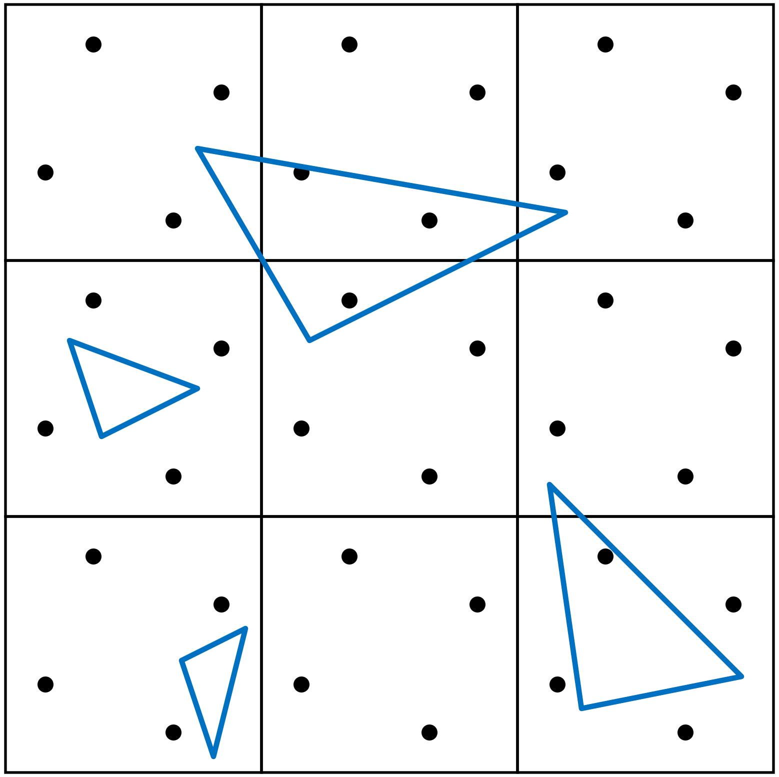
No Pixel Triangles



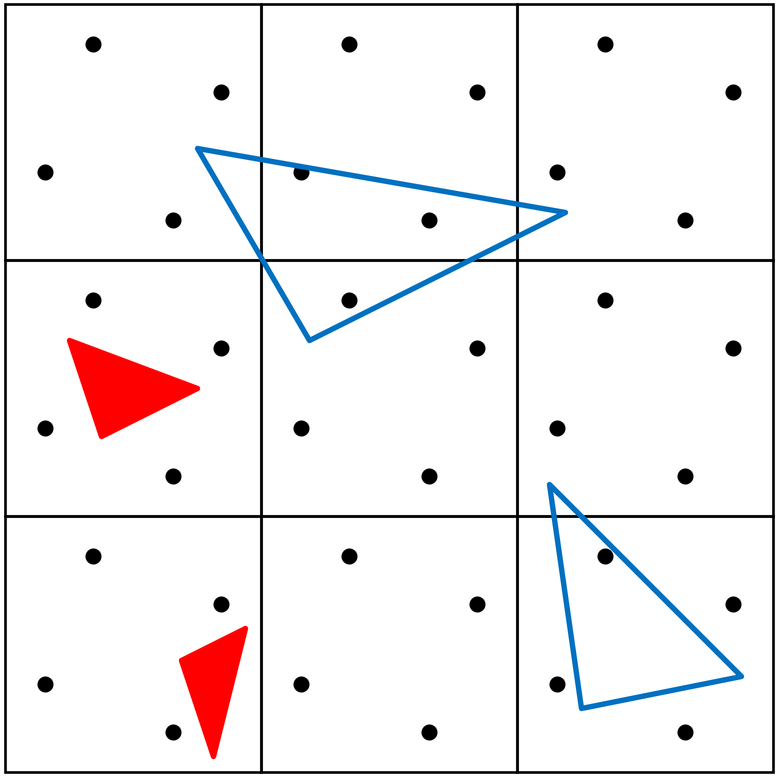
Triangle Culling



Multisampling adds some complications…



Culled

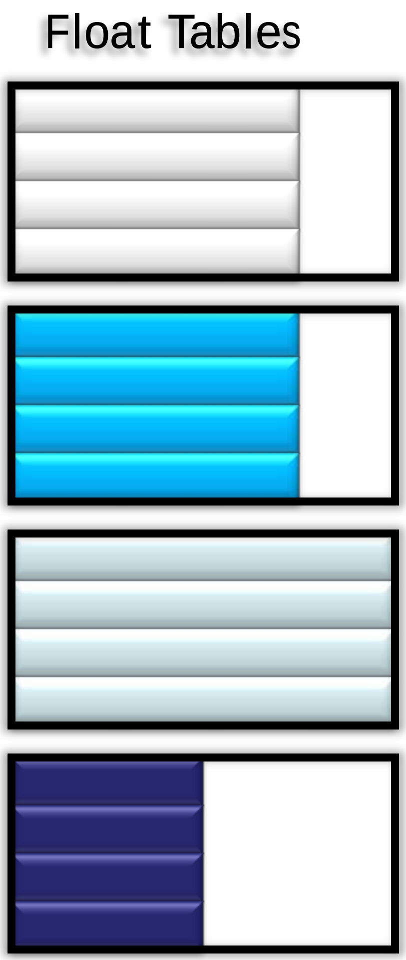


Triangle Culling

10% to 20%

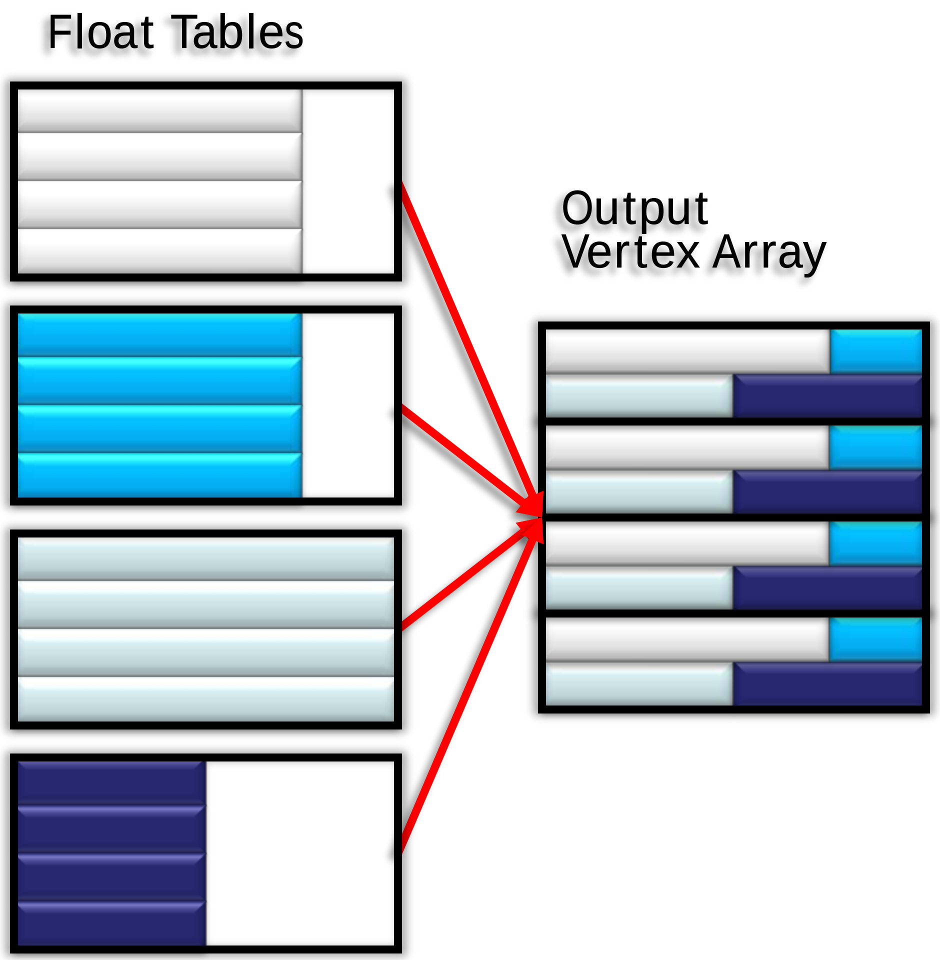
Performance Improvement

Compression for Output


Float Tables

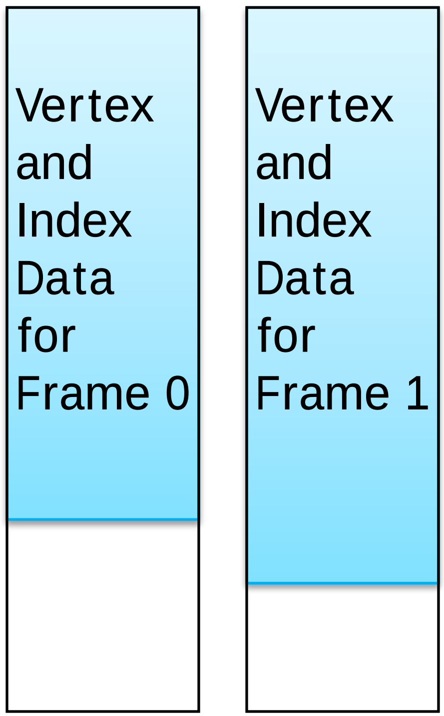
When done, the vertex attributes are compressed into one output stream

SIGGRAPH2008



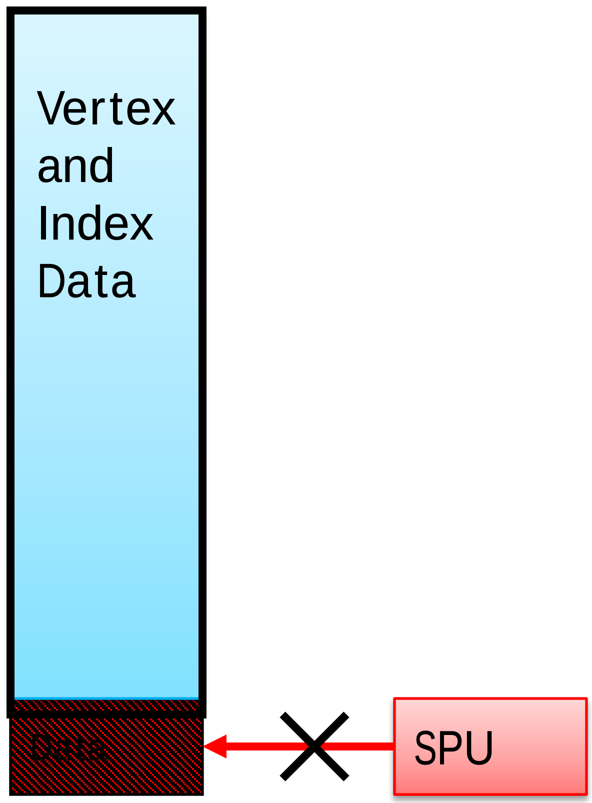
Output Buffering Schemes

Double Buffer

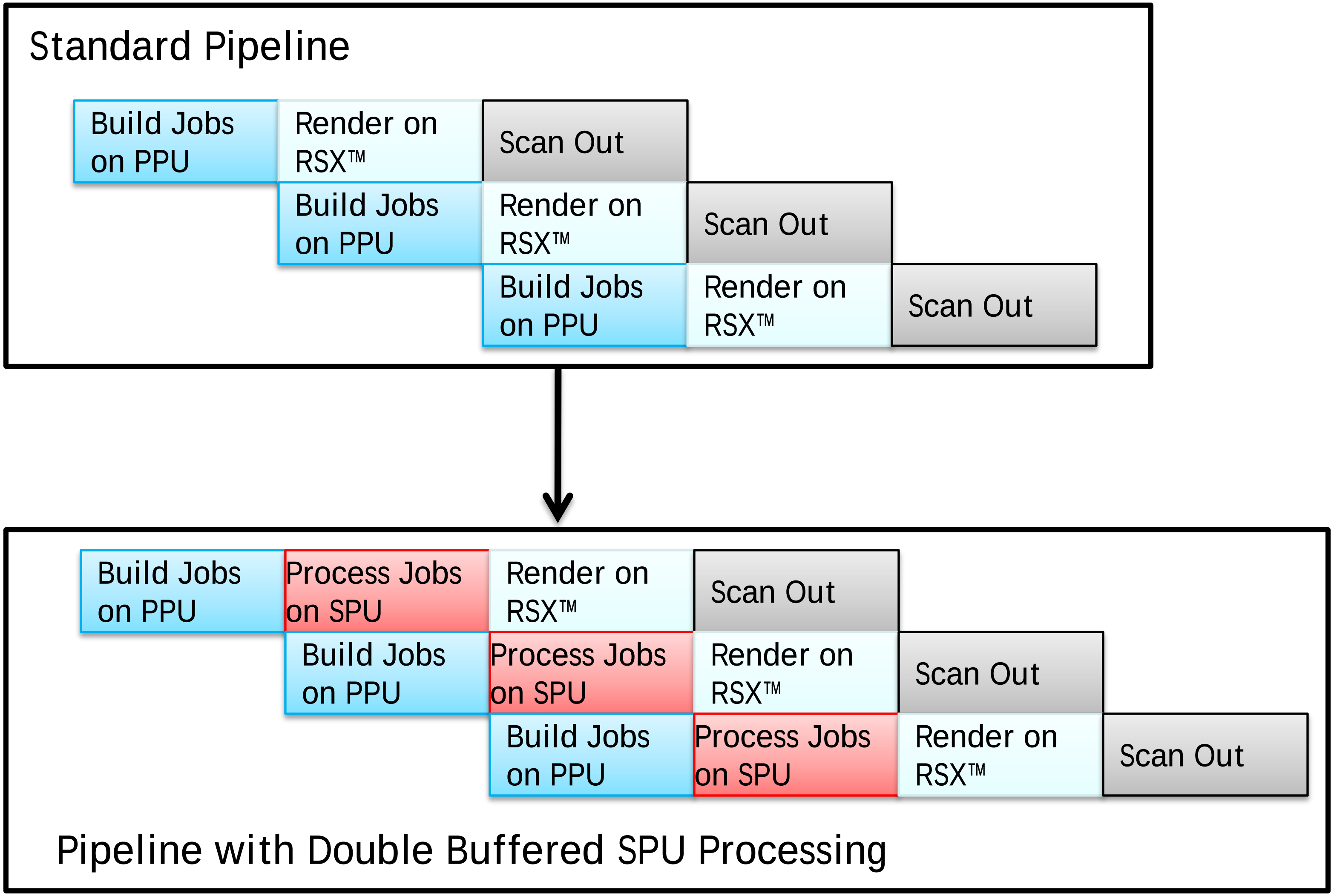


- **Each buffer stores vertex and index data for an entire frame**
- **SPUs atomically access a mutex which is used to allocate memory from a buffer**
- **Uses lots of memory**

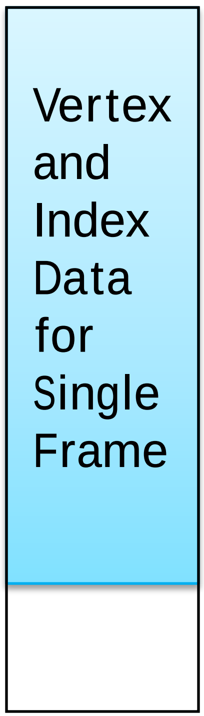
It is possible to completely fill a buffer



Double buffering adds a frame of lag

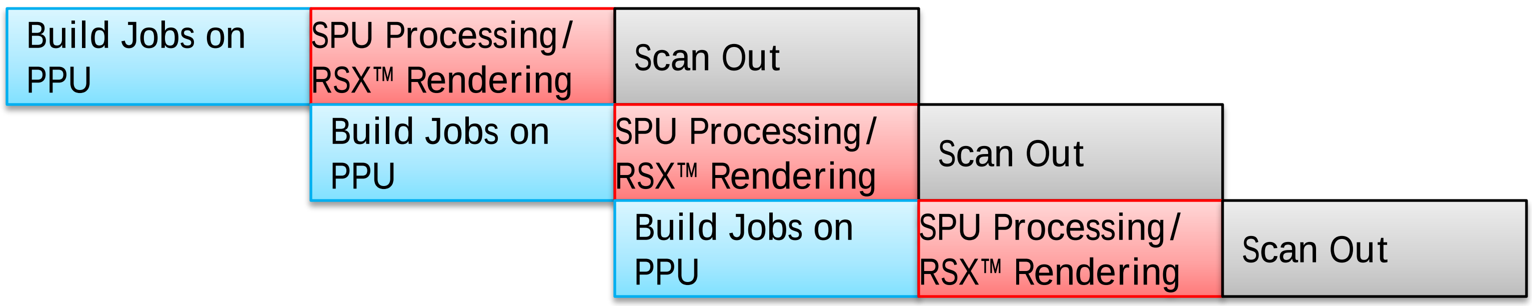


Single Buffering



- Uses only half the memory!
- Still possible to completely fill the buffer

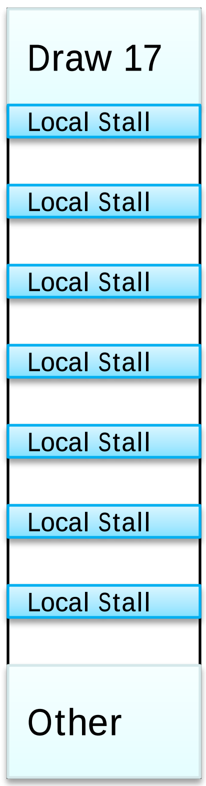
Single Buffering has a shorter pipeline



- **Vertex and index data is created just-in-time for the RSX™**
- **Draw commands are inserted into the command buffer while the RSX™ is rendering**
- **Requires tight SPU↔RSX™ synchronization**

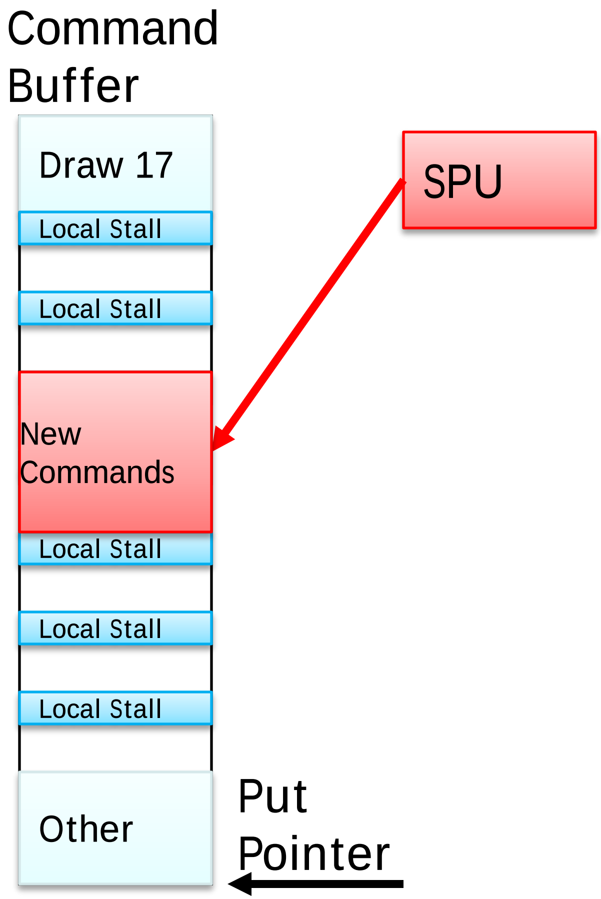
SPU↔RSX™ Synchronization Using Local Stalls

Command Buffer

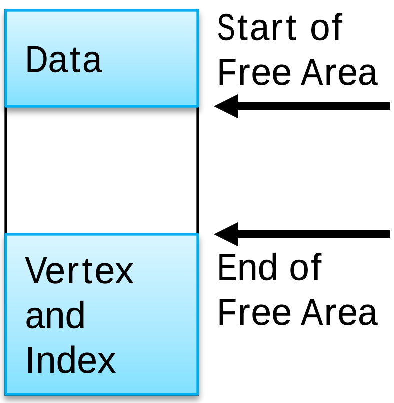


- Place local stalls in the command buffer where necessary
- RSX™ will stop processing at a local stall until it is overwritten by new commands
- SPUs will generally stay ahead of the RSX™, so stalls rarely occur

SPU will overwrite local stalls when it outputs a set of new commands



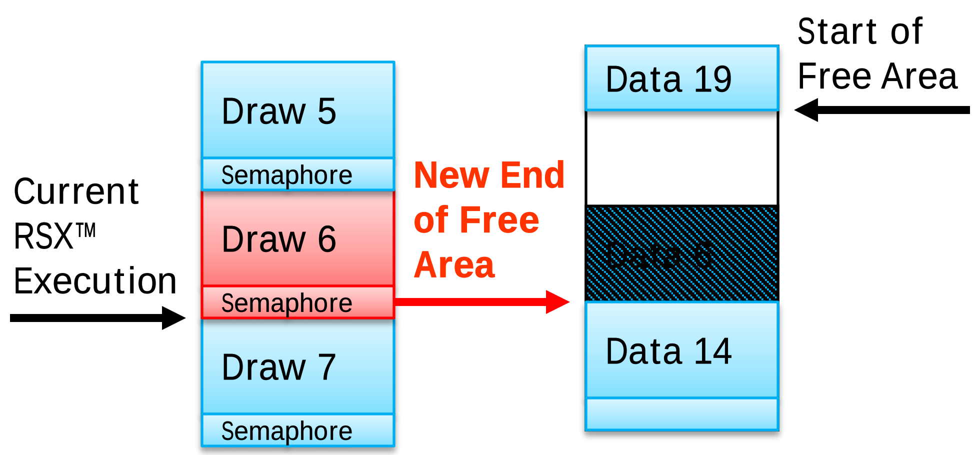
Ring Buffers



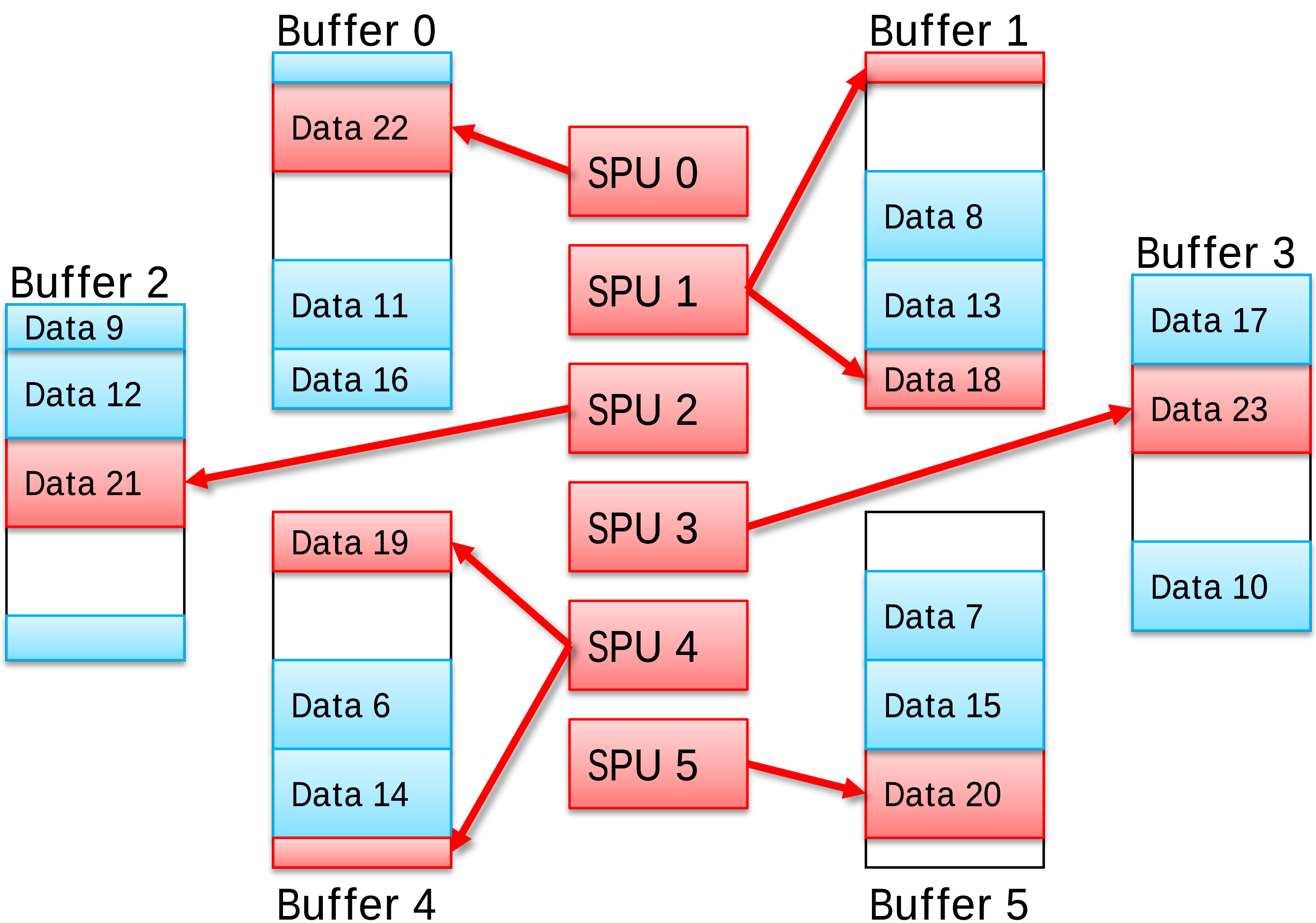
- **Small memory footprint**
- **Will not run out of memory**

RSX™ writes a semaphore once a chunk of data has been consumed

- **A command to write a semaphore needs to be added to the command buffer after all commands that use the data**
  - The value of the semaphore to be written is the new end of free area pointer



Each SPU has its own buffer



Geometry Performance

	CYCLES / TRIANGLE
VERTEX DECOMPRESSION	10.5
INDEX DECOMPRESSION	12.3
BLEND SHAPES (PER SHAPE)	11.0
VERTEX TRANSFORM + TRIANGLE CULLING	30.4
MATRIX PALETTE SKINNING	34.4

Software Pipelined C with SPU Intrinsics

```
do
{
    m1  = in1;
    in1 = si_lqx(pIn1, offset);
    m2  = in2;
    in2 = si_lqx(pIn2, offset);
    m3  = in3;
    in3 = si_lqx(pIn3, offset);
    temp2 = si_selb(m3, m1, mask_0X00);
    si_stqx(out1, pOut1, offset);
    temp3 = si_selb(m2, m1, mask_00X0);
    si_stqx(out2, pOut2, offset);
    temp1 = si_selb(m1, m2, mask_0X00);
    si_stqx(out3, pOut3, offset);
    offset = si_ai(offset, 0x30);
    out2 = si_shufb(m2, temp2, qs_bCaD);
    out1 = si_selb(temp1, m3, mask_00X0);
    out3 = si_shufb(m3, temp3, qs_caBD);
} while(si_to_int(offset) != 0);
```

Software Pipelined C with SPU Intrinsics

```
do
{
     m1  = in1;
     in1 = si_lqx(pIn1, offset);
     m2  = in2;
     in2 = si_lqx(pIn2, offset);
     m3  = in3;
     in3 = si_lqx(pIn3, offset);
     temp2 = si_selb(m3, m1, mask_0X00);
     si_stqx(out1, pOut1, offset);
     temp3 = si_selb(m2, m1, mask_00X0);
     si_stqx(out2, pOut2, offset);
     temp1 = si_selb(m1, m2, mask_0X00);
     si_stqx(out3, pOut3, offset);
     offset = si_ai(offset, 0x30);
     out2 = si_shufb(m2, temp2, qs_bCaD);
     out1 = si_selb(temp1, m3, mask_00X0);
     out3 = si_shufb(m3, temp3, qs_caBD);
} while(si_to_int(offset) != 0);
```

Up to 20x faster than naive C/C++

1 SPU

1 SPU

800,000+

Triangles Per Frame
at 60 Frames per Second

1 SPU

800,000+

Triangles Per Frame
at 60 Frames per Second

<u>60% of which are culled!</u>

Next Generation Parallelism In Games

Jon Olick
id Software

SIGGRAPH2008

Beyond Programmable Shading: In Action

GAME ENTITY PROCESSING

Game Entity Processing

- **Current Generation**
  - Serial Processing of entities in a giant for loop.

```
for(int i = 0; i < numEntities; ++i) {
        entity[i]->Think();
}
```

Game Entity Processing

- **Current Generation**
  - Serial Processing of entities in a giant for loop.
- **Next Generation**
  - Parallelism via Double Buffering
  - Every entity runs in parallel with each other with no dependency stalls.
  - Each entity can only read from previous frame’s results
  - Each entity can only write to itself

Game Entity Processing

- Record the progress of the game and replay to debug.
- Single thread and randomize processing of entities to help find bugs.
- Can protect memory so that bad accesses cause exceptions to enforce double buffering rules.

Game Entity Processing

- What about entities which have dependant entities?
- Bucketing and Synchronization Points

Ray Casting
The Next Generation

Why Ray Casting?

Why Ray Casting?

- **A good question...**

?

- **Back in Quake 1**
  - If you had to make a decision between an additional CPU and a Graphics Card which would you choose?

- **Back in Quake 1**
  - If you had to make a decision between an additional CPU and a Graphics Card which would you choose?
  - Why is this any different today?

- **Back in Quake 1**
  - If you had to make a decision between an additional CPU and a Graphics Card which would you choose?
  - Why is this any different today?
  - Its not any different.

Why Ray Casting?

- **What value does it provide to developers?**

Why Ray Casting?

- **What value does it provide to developers?**
  - Shorter & Cheaper Development
  - Higher Quality Games

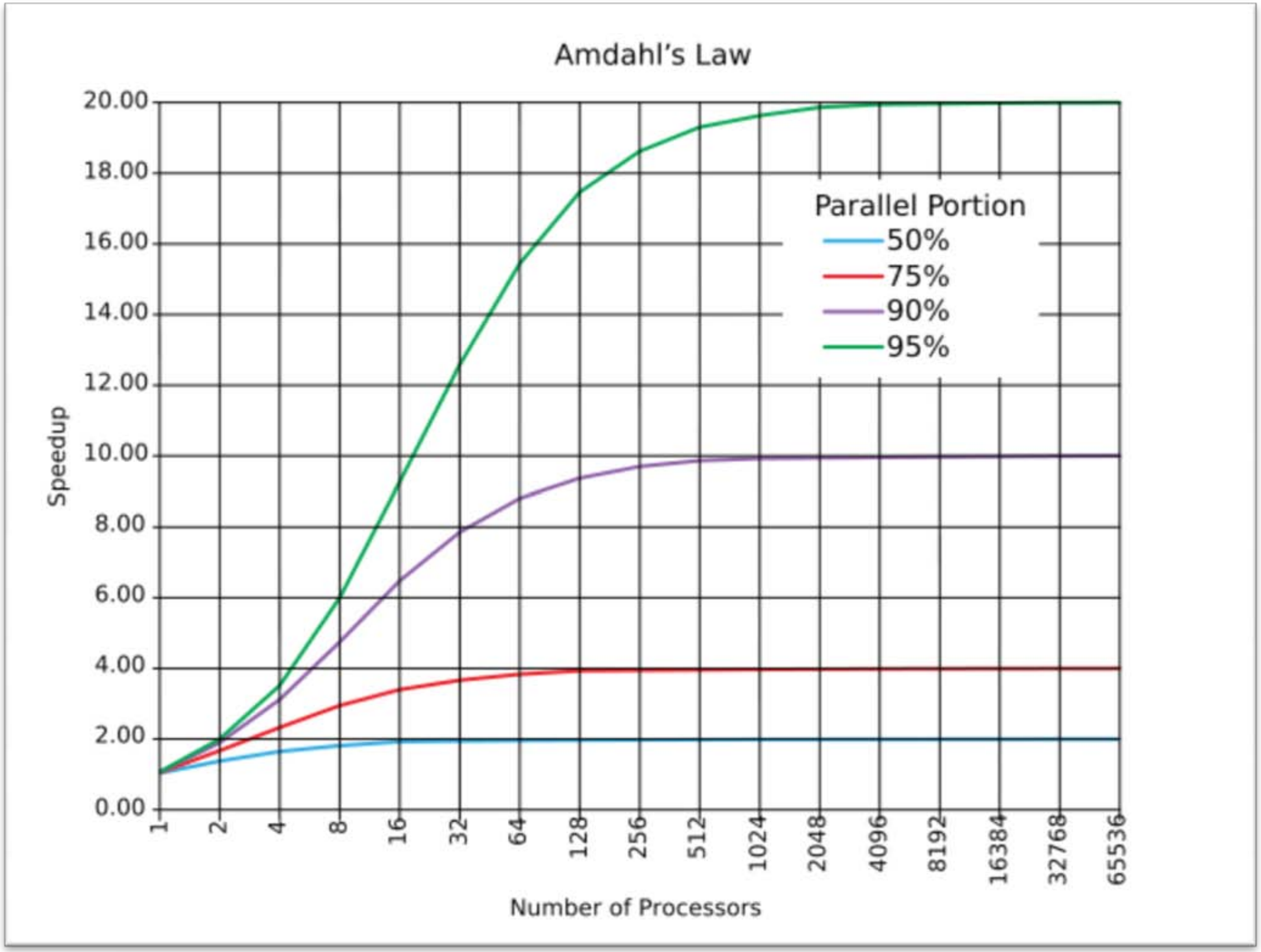
Why Ray Casting?

- **What value does it provide to end users?**

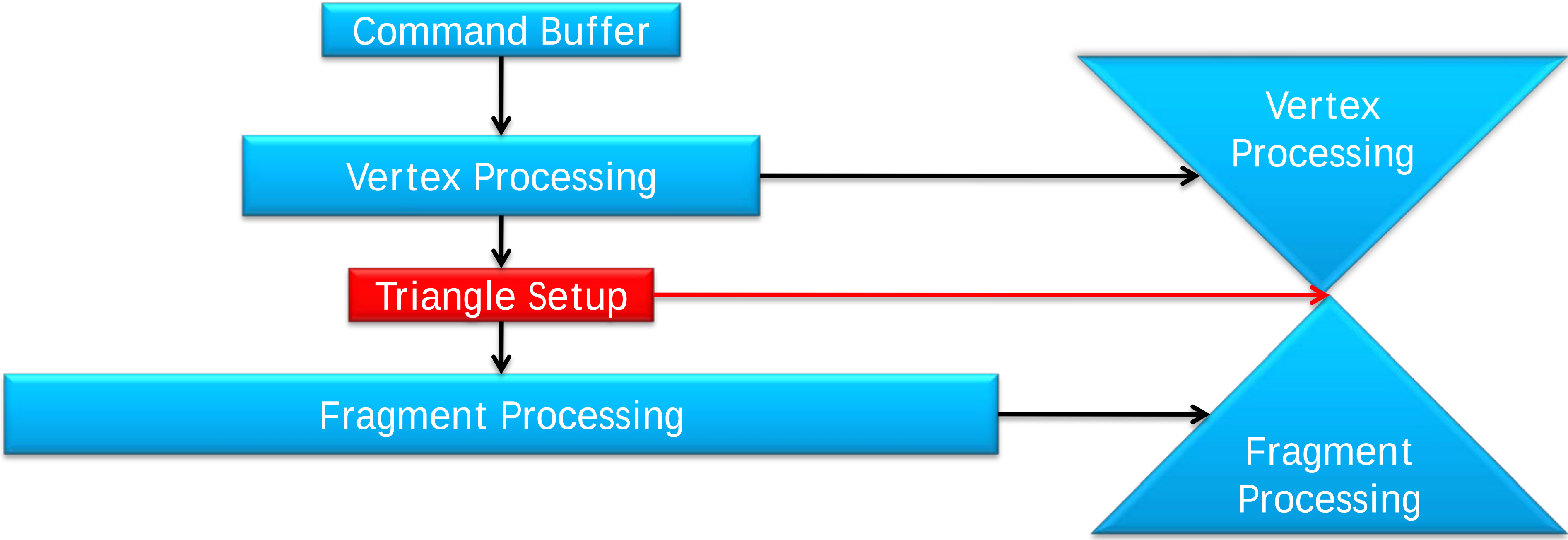
Screenshot From E3 Rage Video

Screenshot From E3 Rage Video

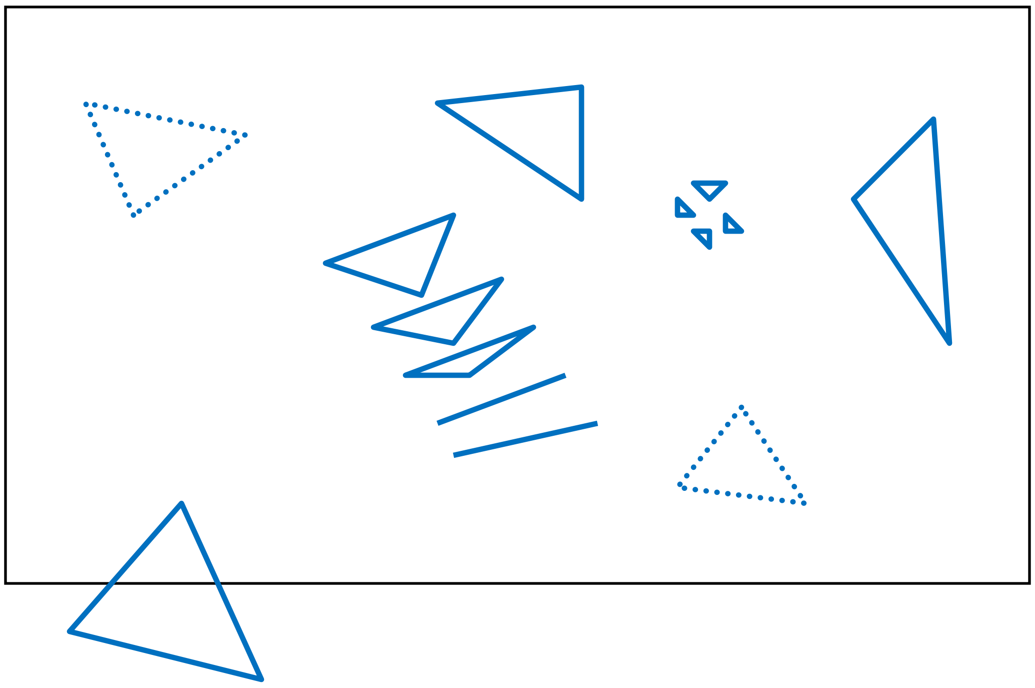
Why Ray Casting?



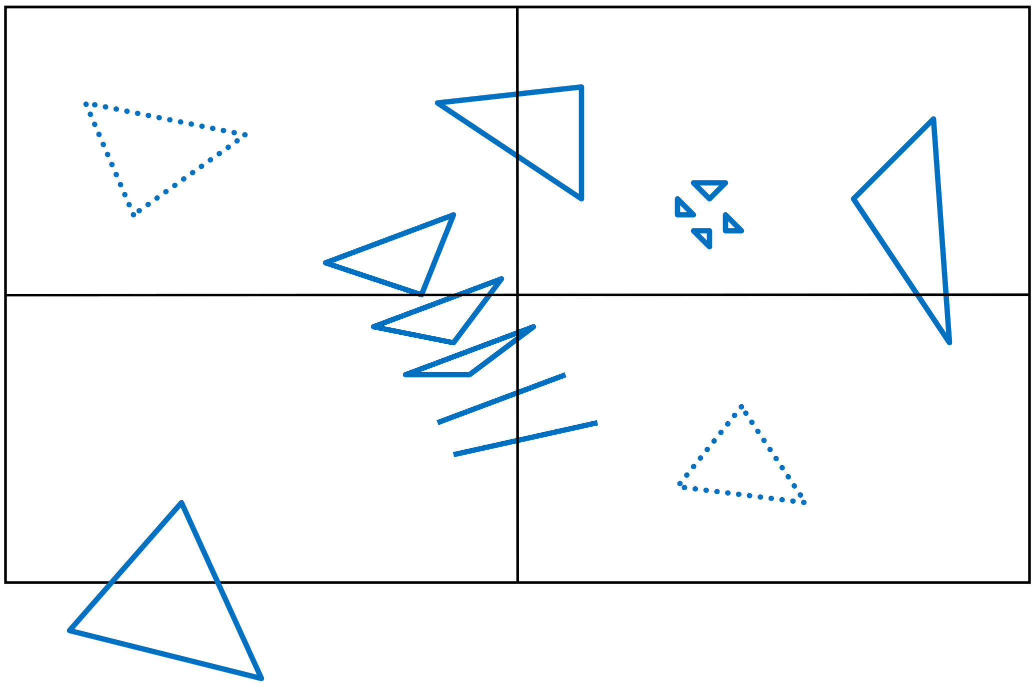
Current State of Rasterization



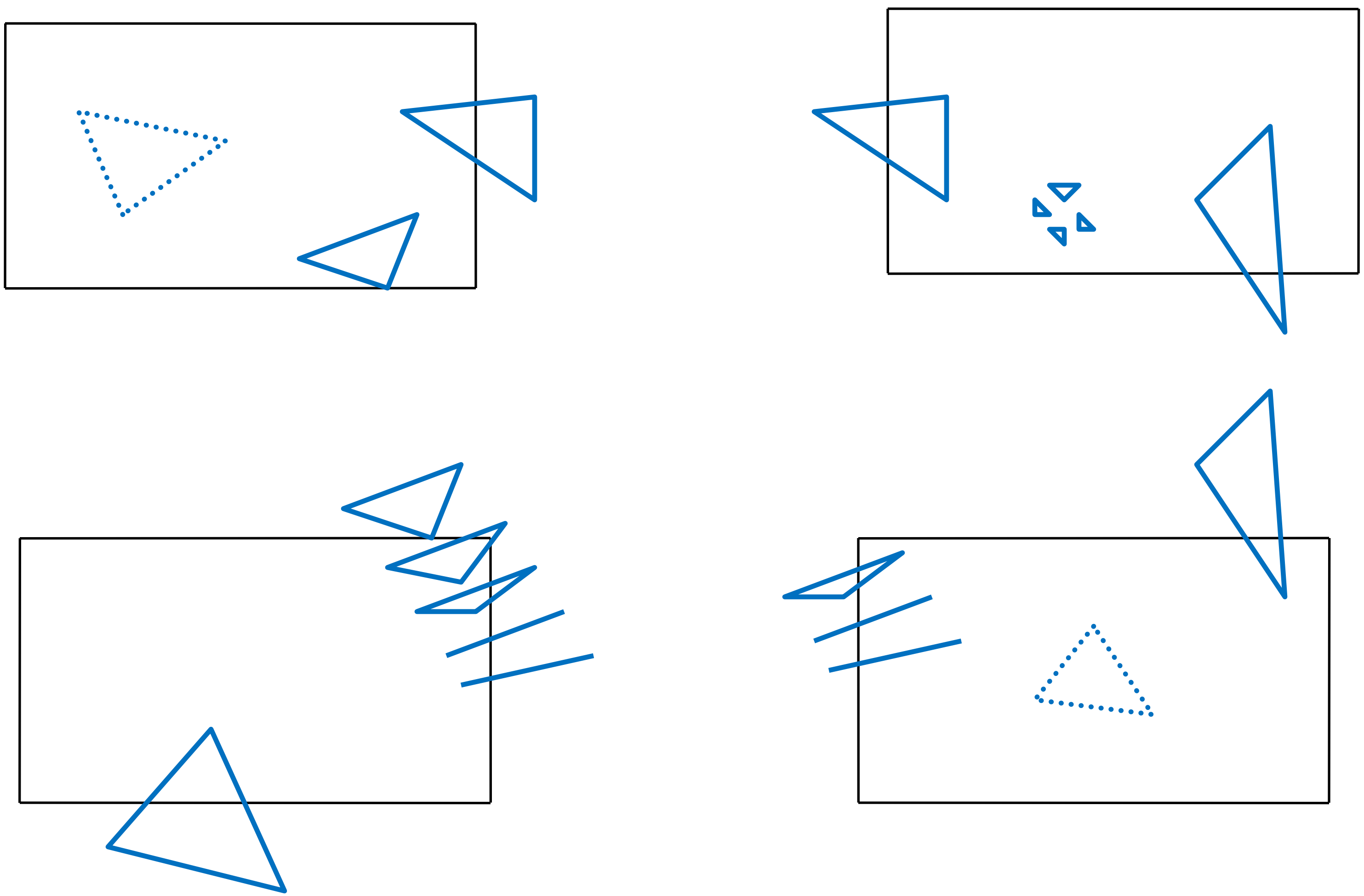
Future of Rasterization



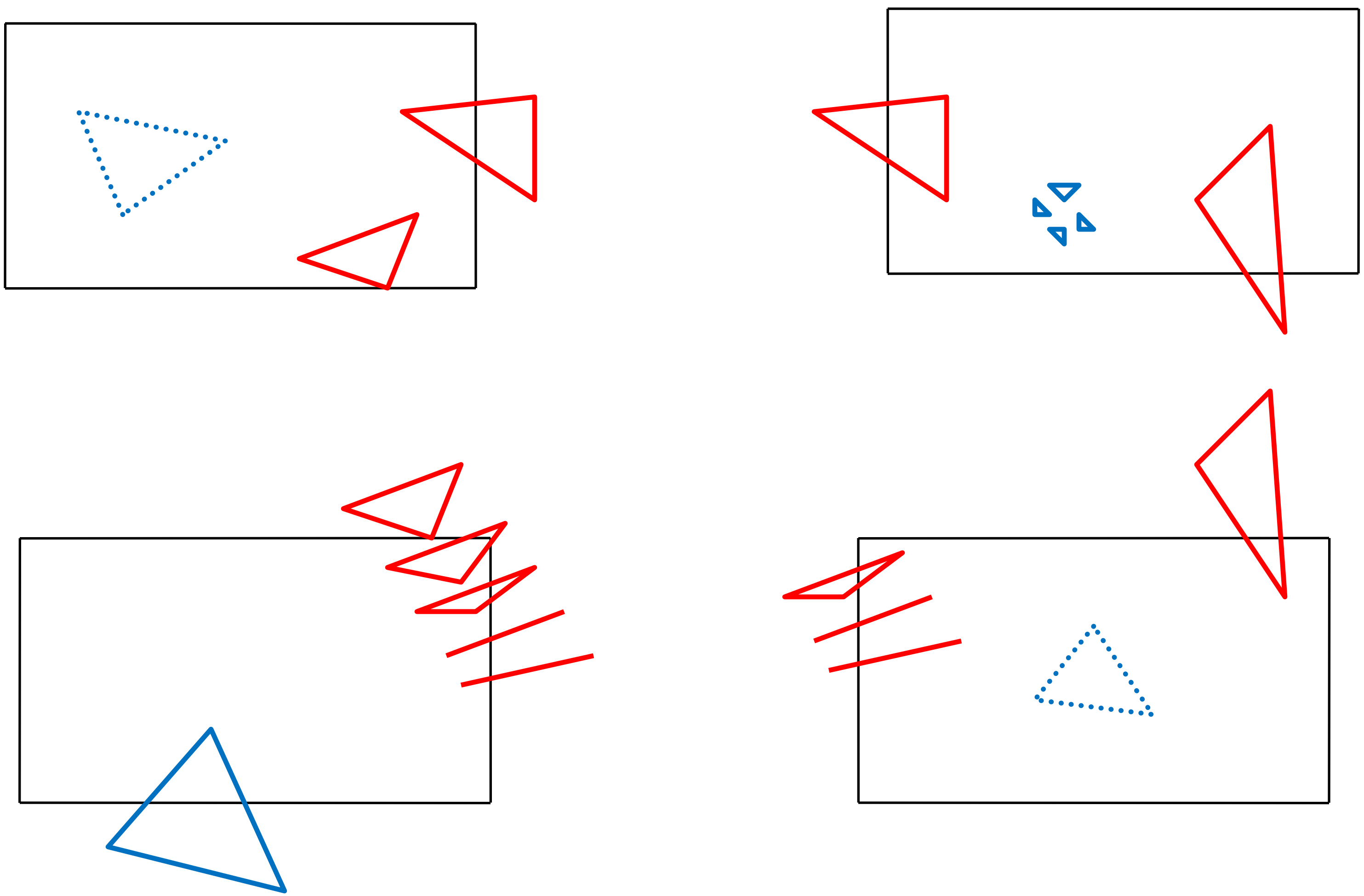
Future of Rasterization



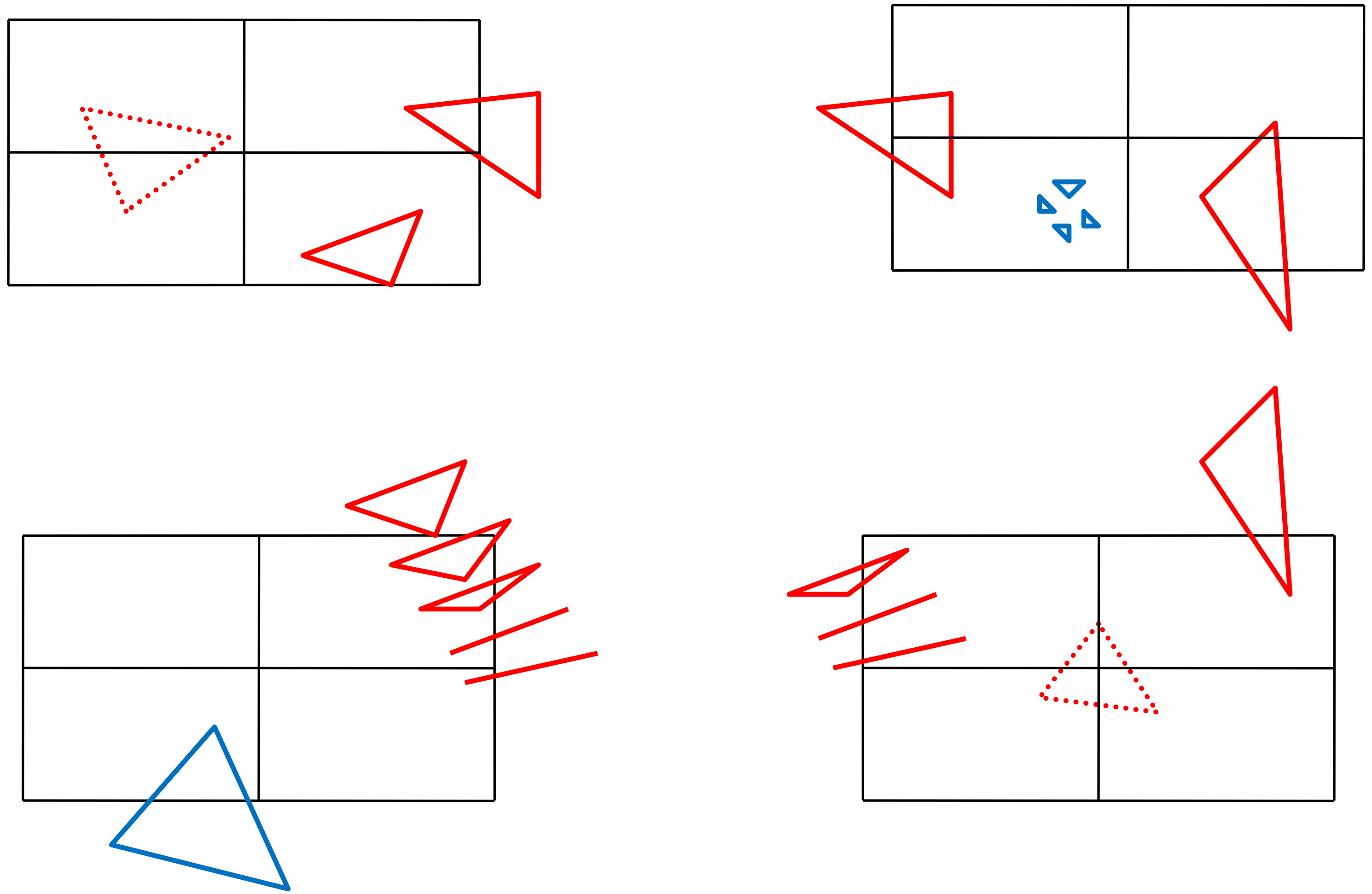
Future of Rasterization



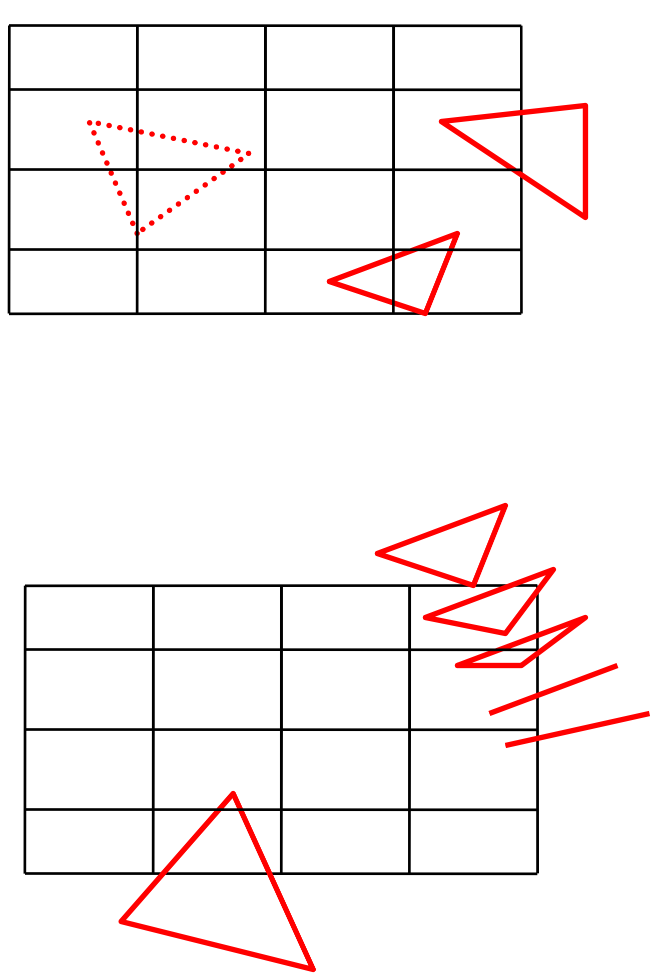
Future of Rasterization



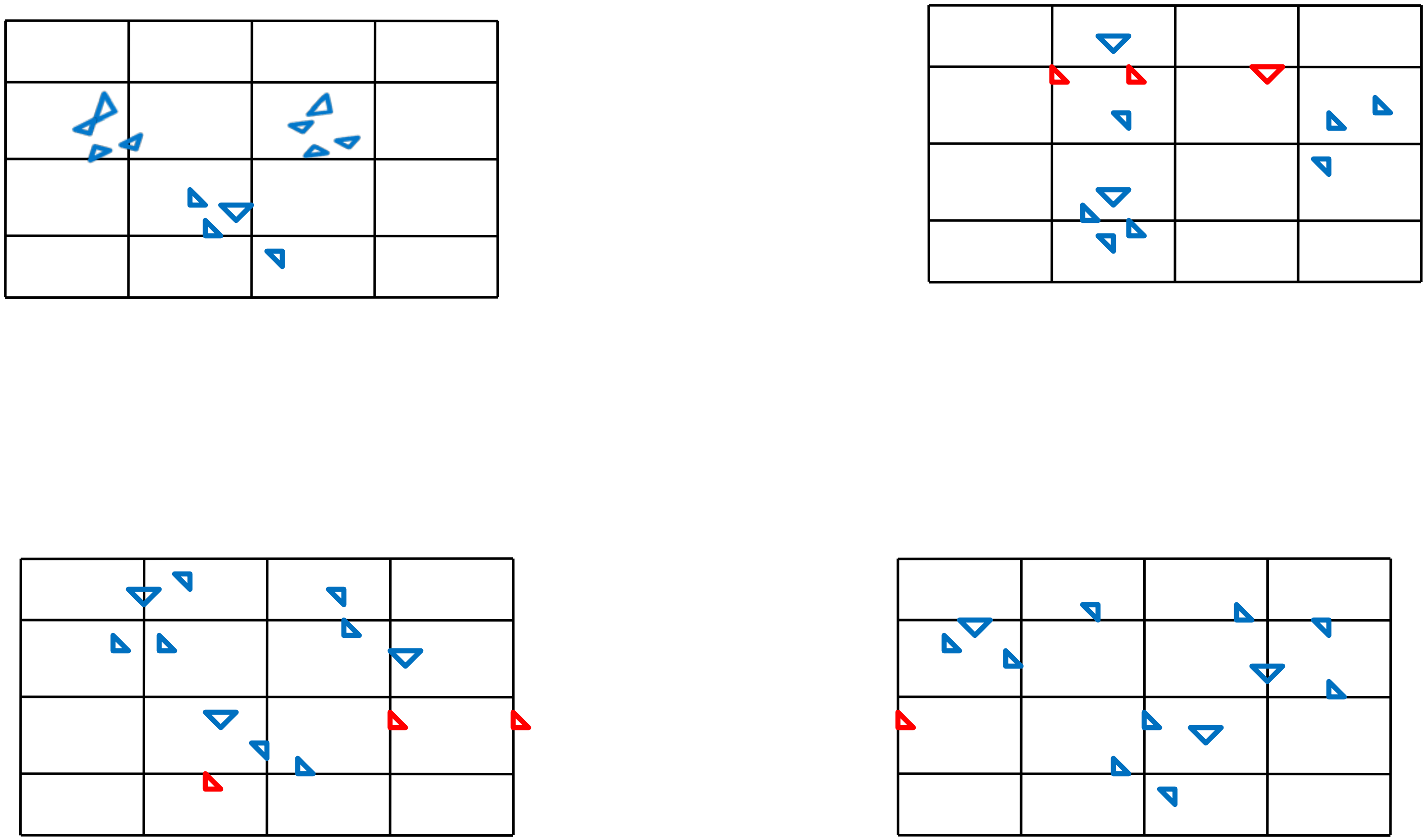
Future of Rasterization



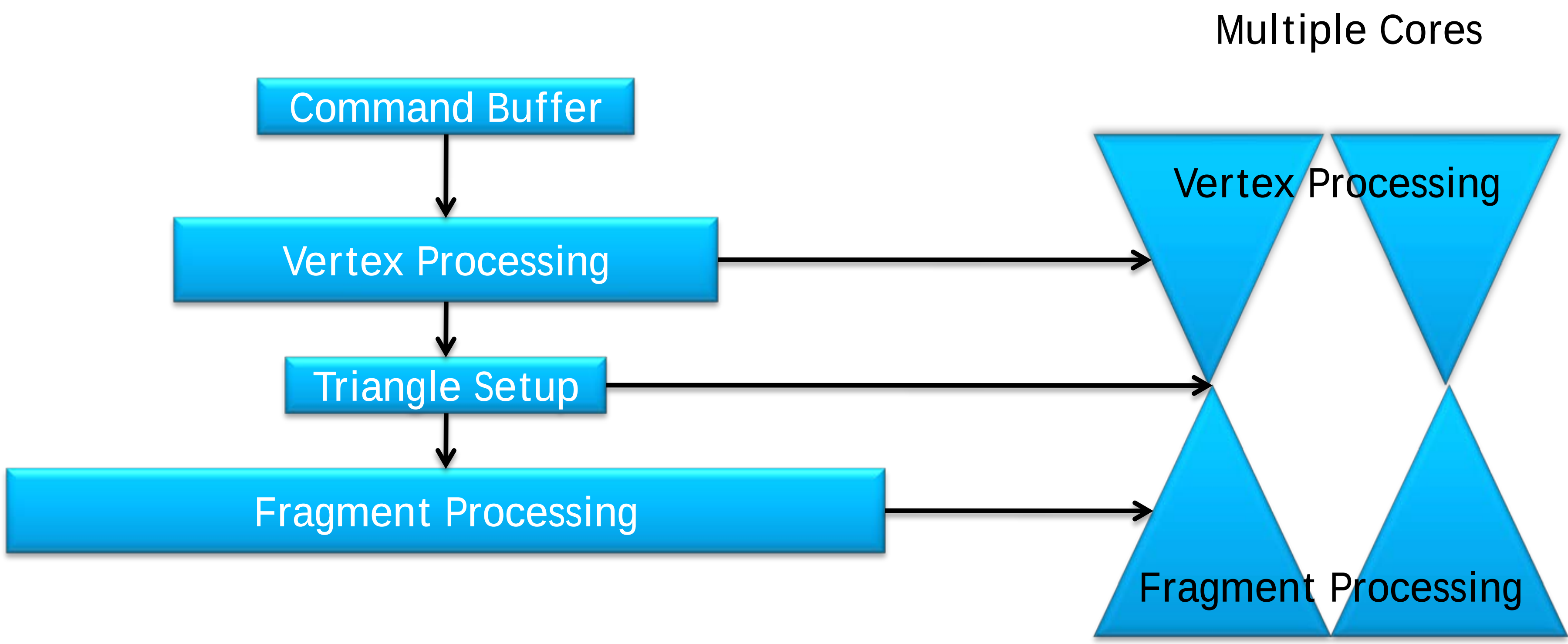
Future of Rasterization



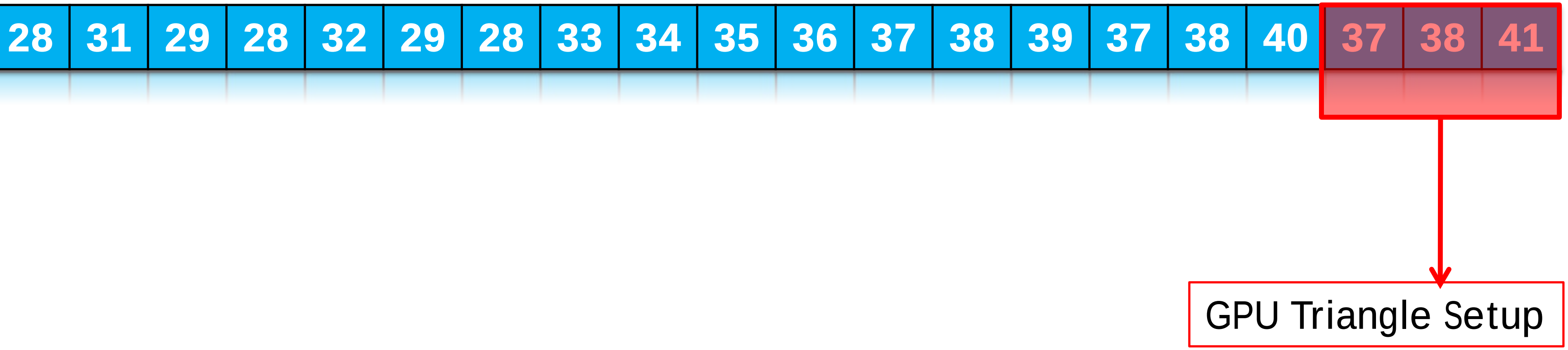
Future of Rasterization



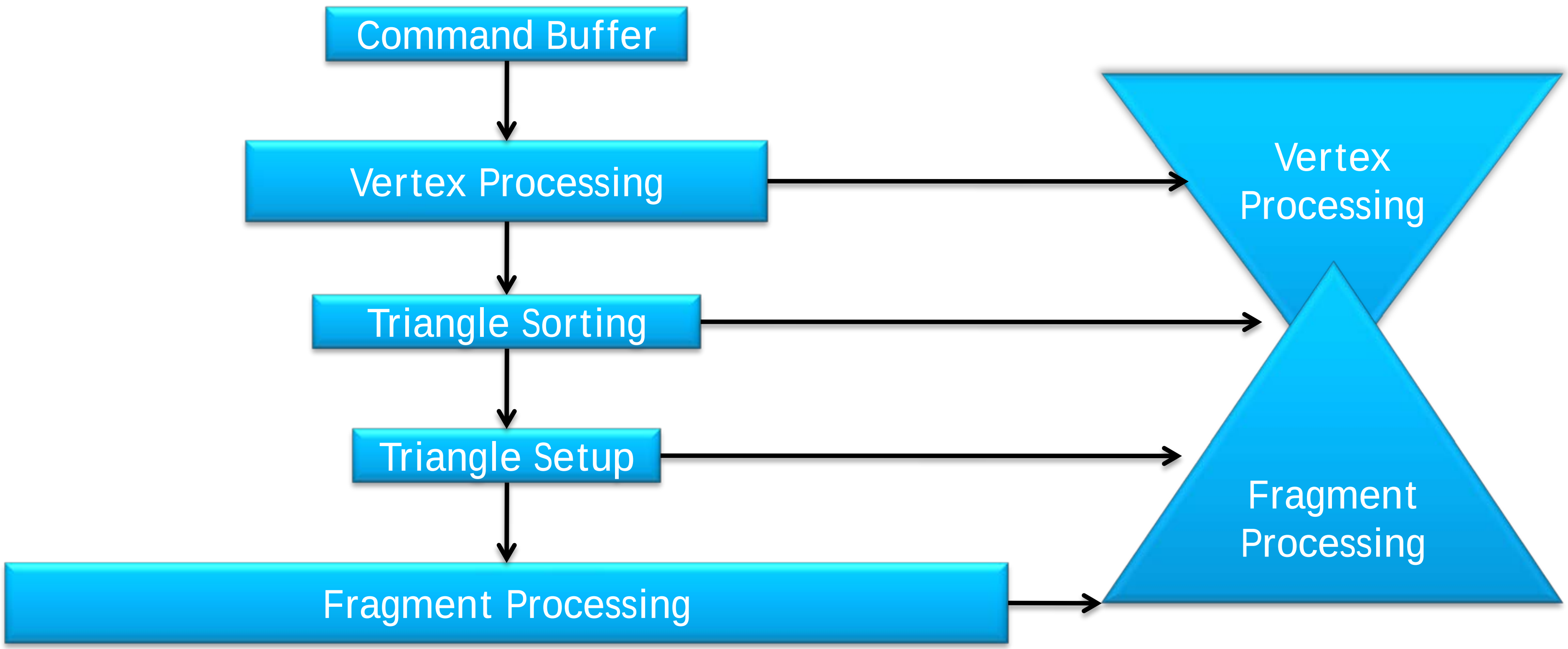
Future of Rasterization



Future of Rasterization



Future of Rasterization



Future of Rasterization x2

0	1	2
0	2	3
3	2	4
3	4	5
6	7	8
9	6	8
10	9	11
9	8	11
9	8	12
13	14	15
15	14	16
16	14	17
16	14	18
16	14	19

GPU Triangle Setup

Future of Rasterization x2

SIGGRAPH2008

0	1	2
0	2	3
3	2	4
3	4	5
6	7	8
9	6	8
10	9	11
9	8	11
9	8	12
13	14	15
15	14	16
16	14	17
16	14	18
16	14	19

GPU Triangle Setup

Future of Rasterization x2

0	1	2
0	2	3
3	2	4
3	4	5
6	7	8
9	6	8
10	9	11
9	8	11
9	8	12
13	14	15
15	14	16
16	14	17
16	14	18
16	14	19

GPU Triangle Setup

Future of Rasterization x2

SIGGRAPH2008

0	1	2
0	2	3
3	2	4
3	4	5
6	7	8
9	6	8
10	9	11
9	8	11
9	8	12
13	14	15
15	14	16
16	14	17
16	14	18
16	14	19

GPU Triangle Setup

Future of Rasterization x2

0	1	2
0	2	3
3	2	4
3	4	5
6	7	8
9	6	8
10	9	11
9	8	11
9	8	12
13	14	15
15	14	16
16	14	17
16	14	18
16	14	19

GPU Triangle Setup

Why voxels, and not triangles?

Why voxels, and not triangles?

- **Unique Texturing is possible with rasterization**
  - Rage – idTech 5

Why voxels, and not triangles?

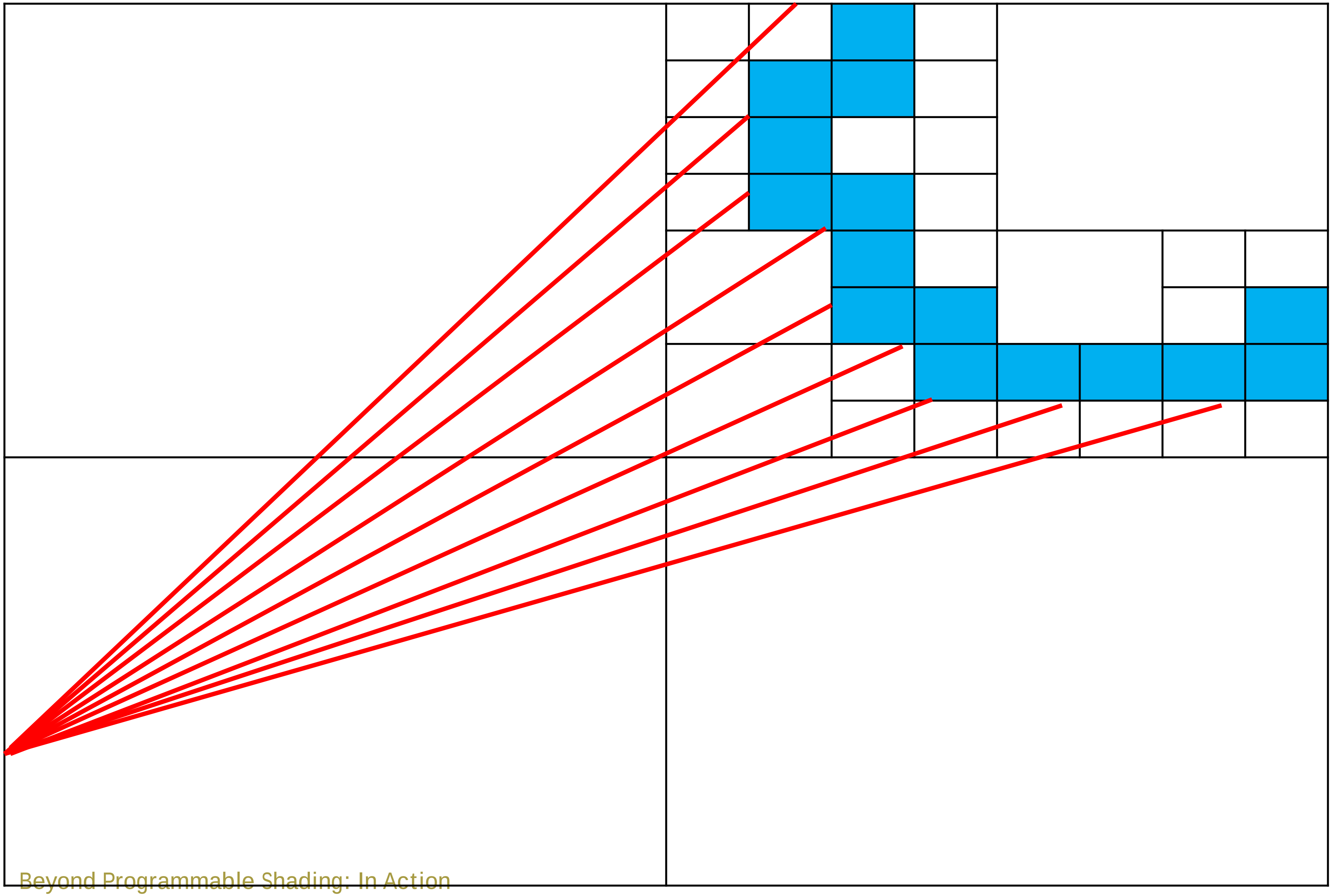
- **Unique Texturing is possible with rasterization**
  - Rage – idTech 5
- **Unique Geometry is possible with rasterization**
  - Progressive Mesh

Why voxels, and not triangles?

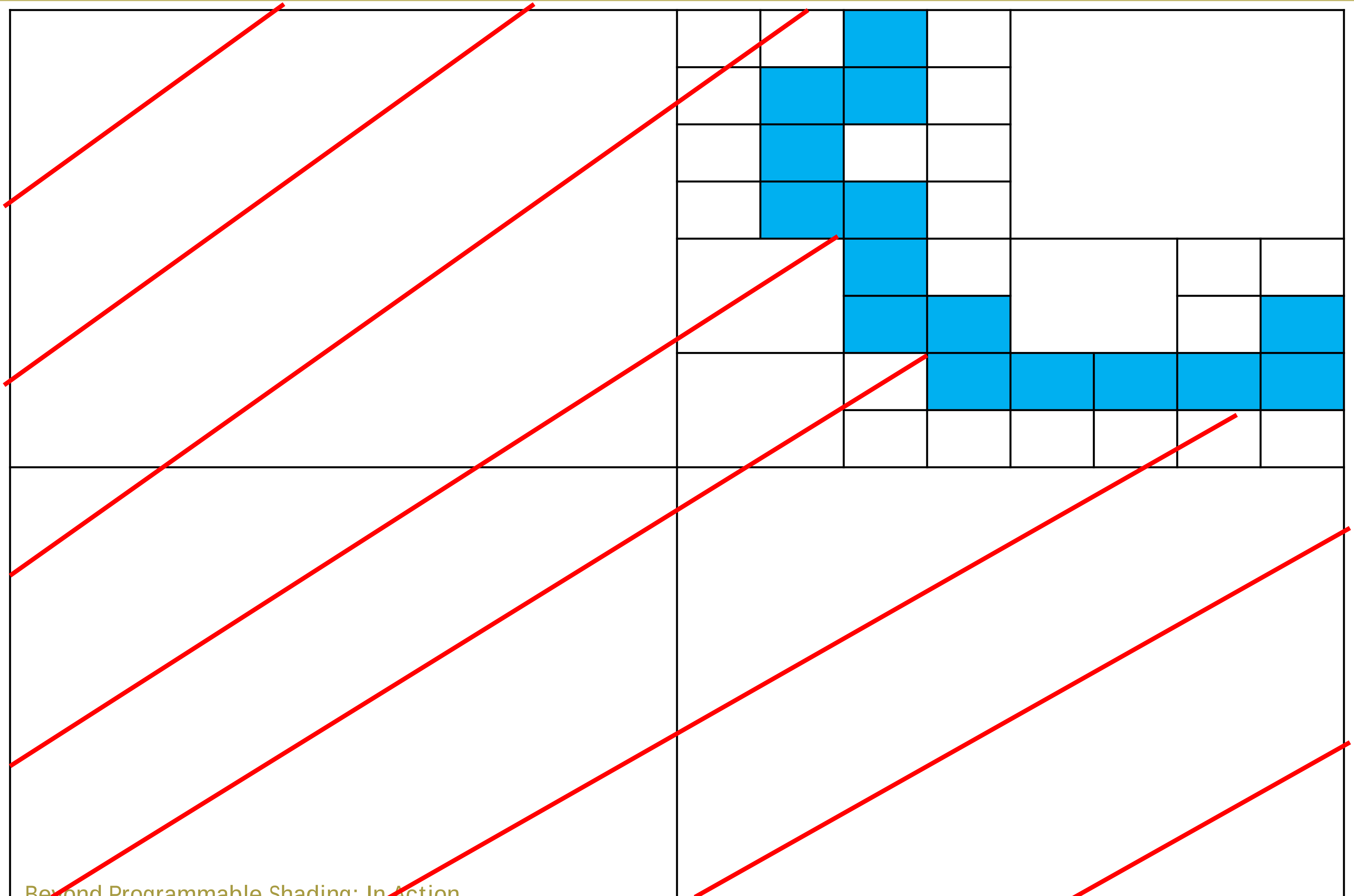
- **Unique Texturing is possible with rasterization**
  - Rage – idTech 5
- **Unique Geometry is possible with rasterization**
  - Progressive Mesh
- **SVO Solves Two Problems in One**
  - Unique Texturing & Unique Geometry

Why is the control flow efficient?

SIGGRAPH2008

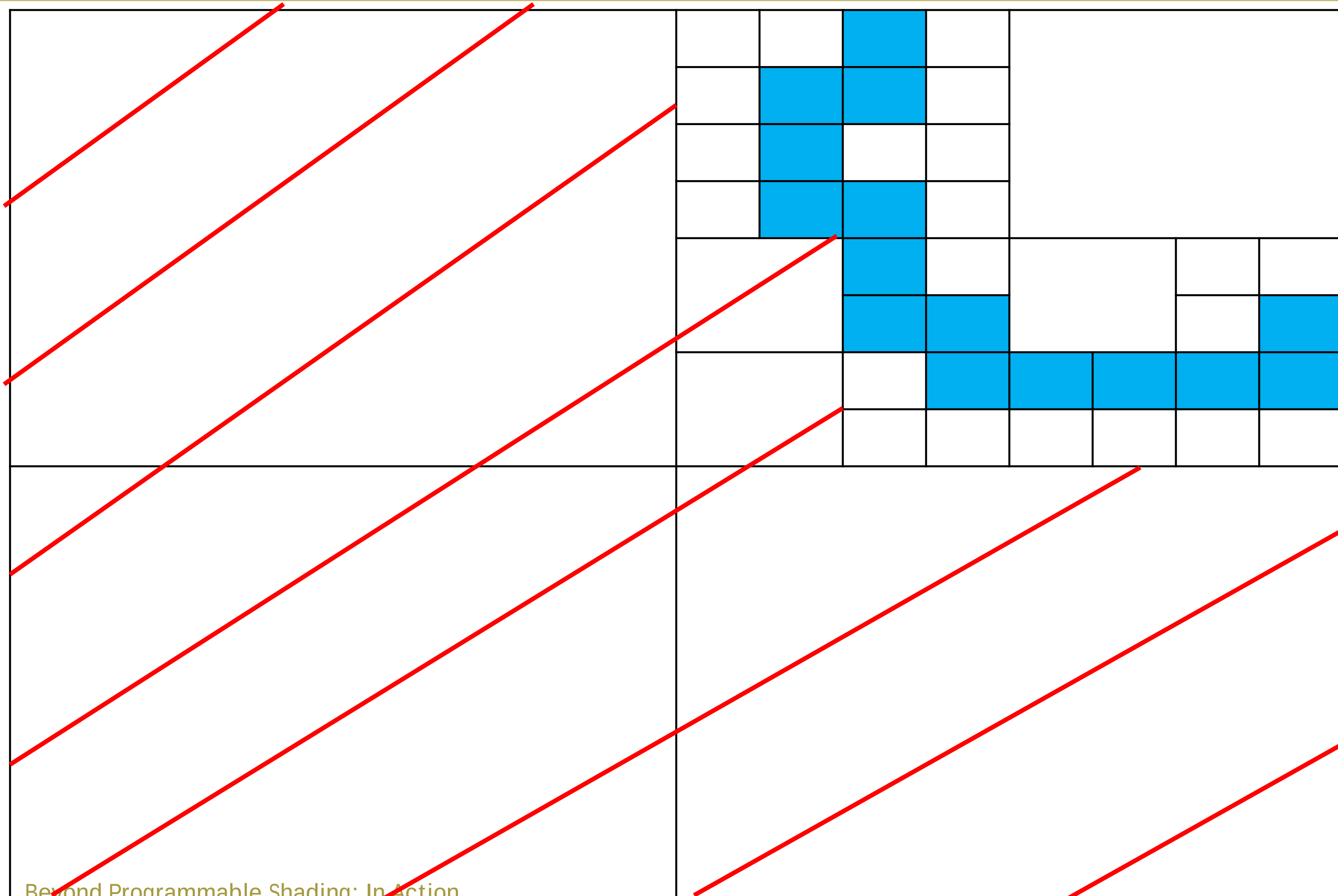


Why is the control flow efficient?



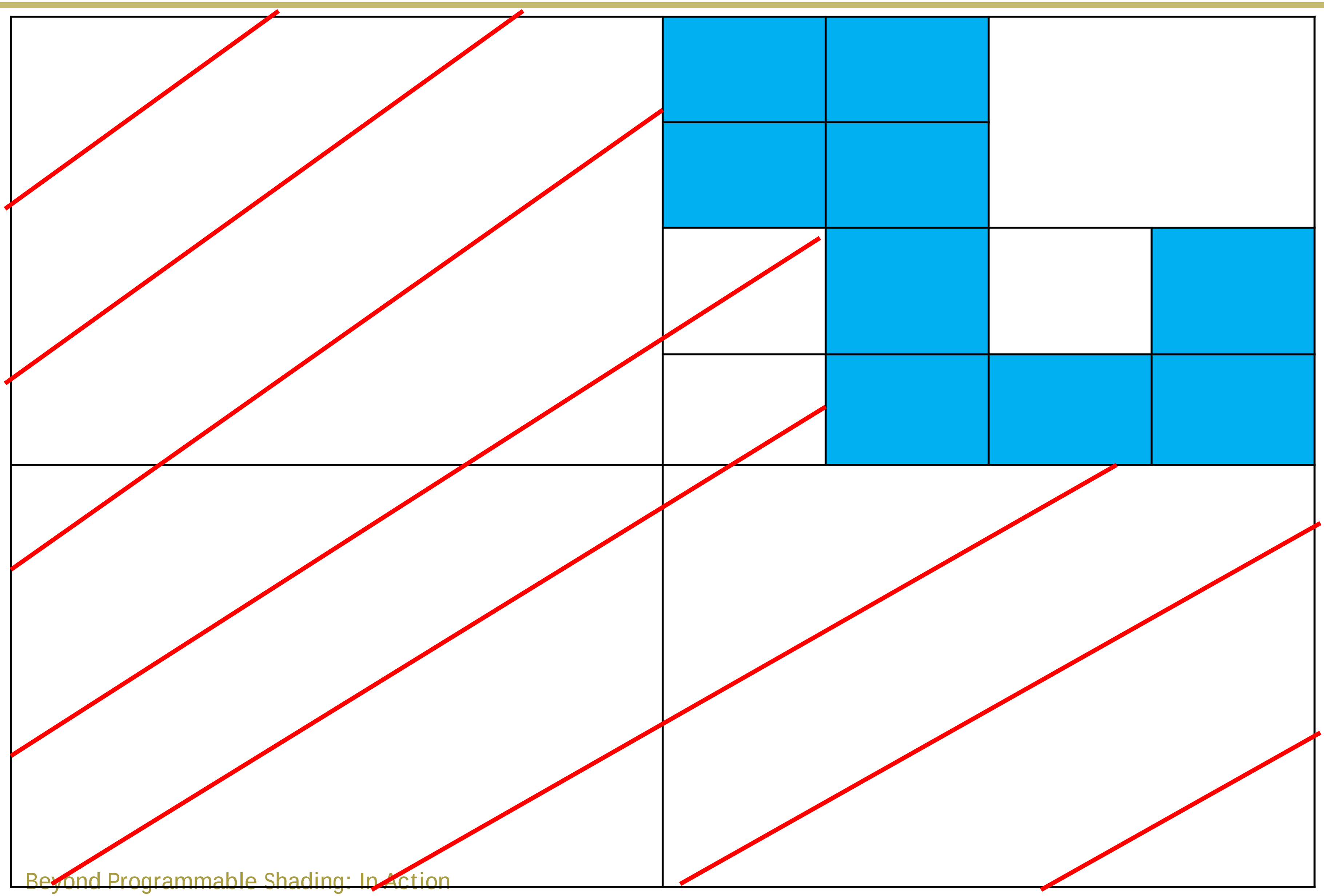
Why is the control flow efficient?

SIGGRAPH2008

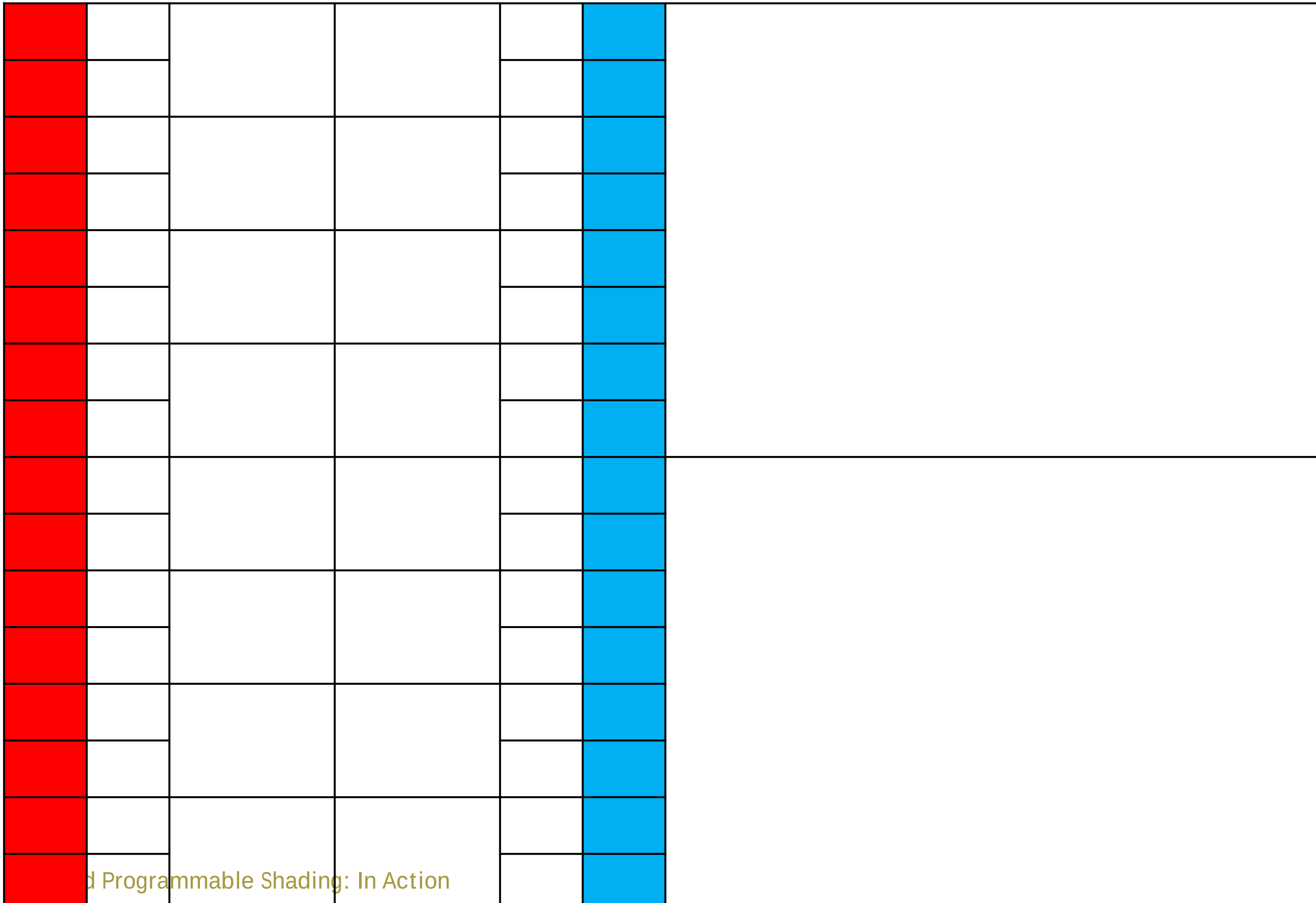


Why is the control flow efficient?

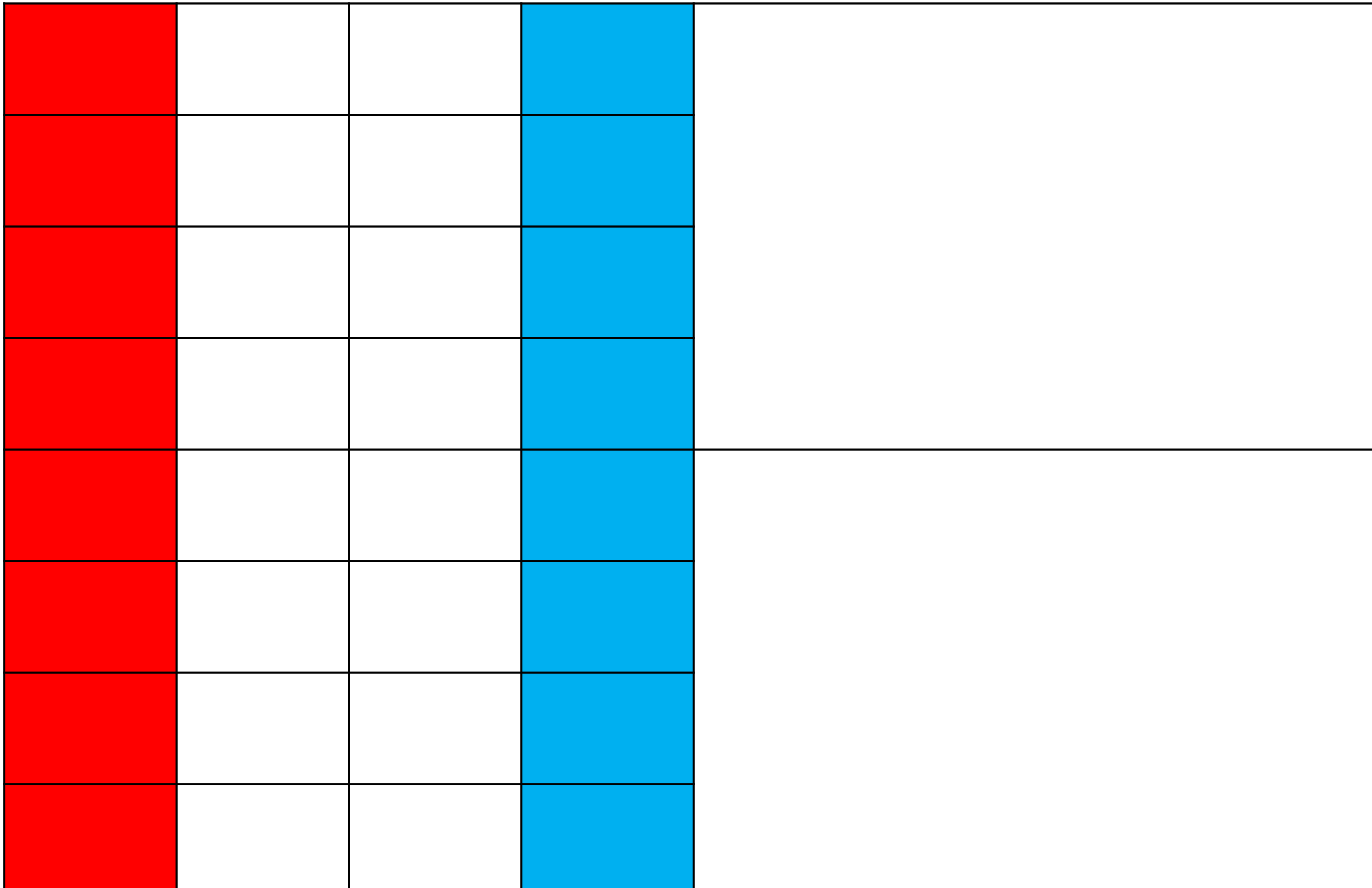
SIGGRAPH2008



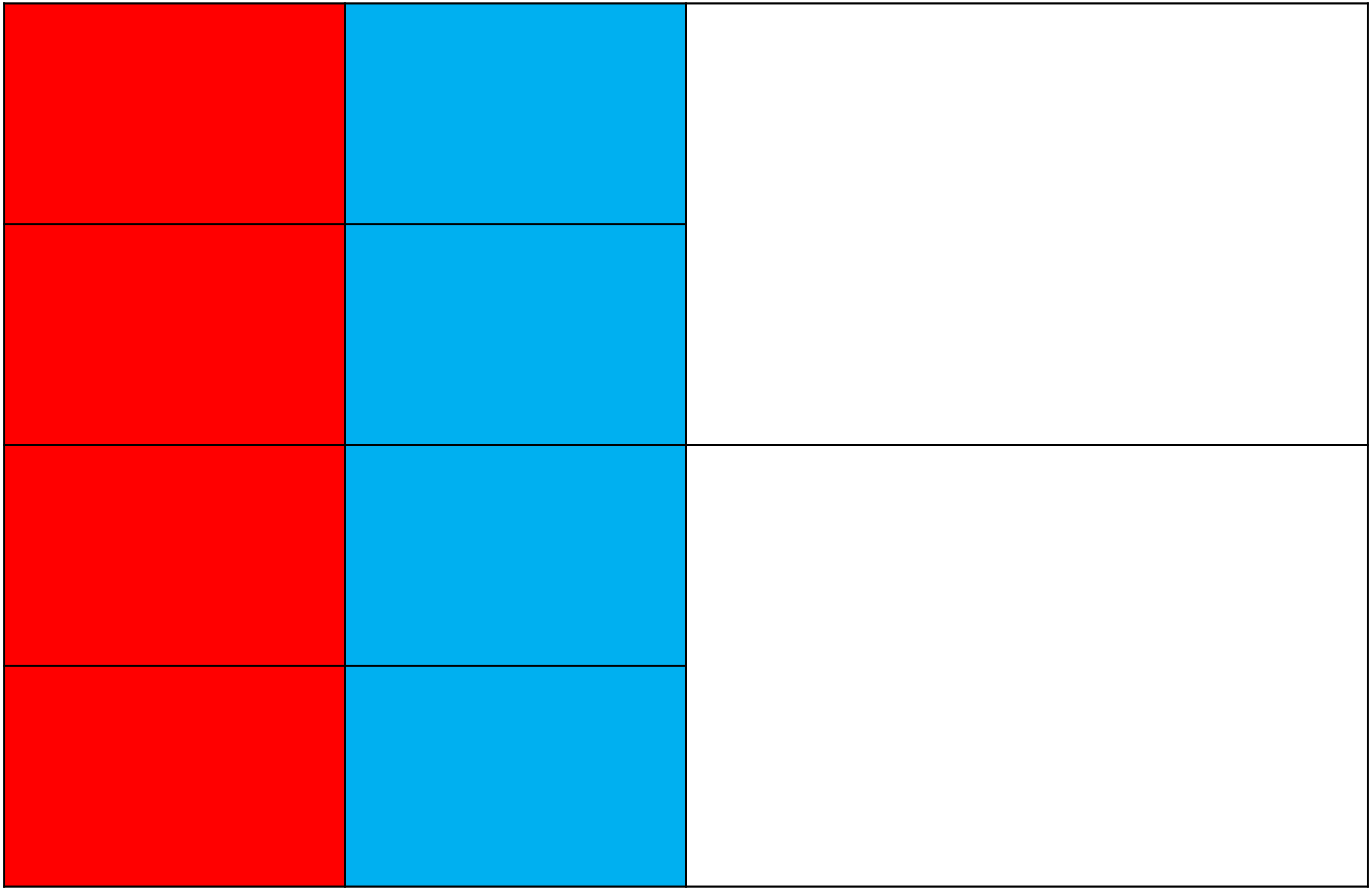
Voxel Mip Mapping - Thin Walls



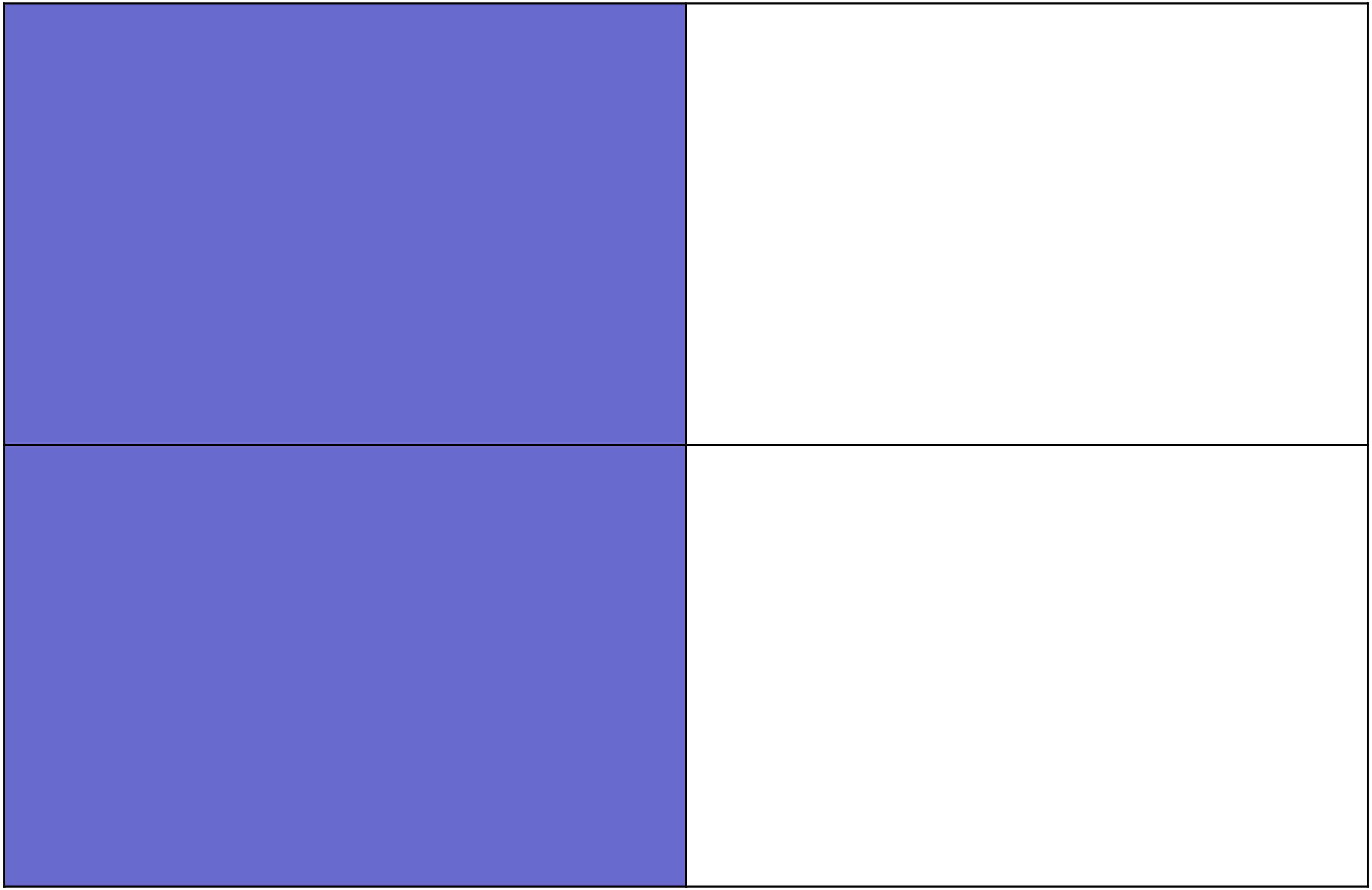
Voxel Mip Mapping - Thin Walls



Voxel Mip Mapping – Thin Walls



Voxel Mip Mapping - Thin Walls



Caveats of Ray-Tracing?

- **“Primary rays cache, secondary rays thrash”™**
  - Importance sampling to the rescue!
- **Ray Tracing != Ray Casting**

Sparse Voxel Oct-trees

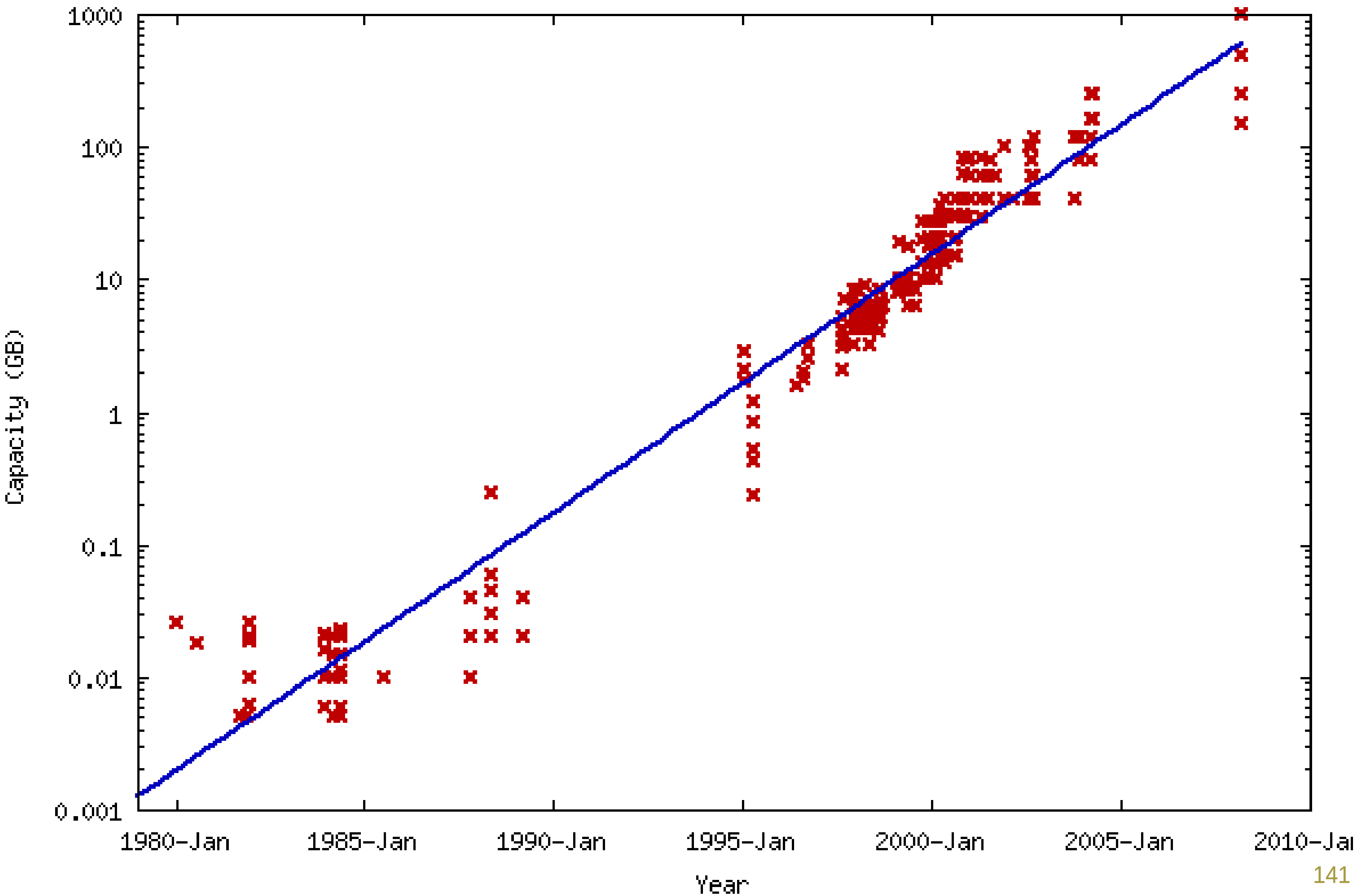
- **Oct-trees as collection of maximal blocks.**
  - Related to run-length encoding.
  - Variable splitting planes

Data Structure

- **Disk Caching with Virtual and Physical Pages**

Is Disk Caching a Valid Lever?

SIGGRAPH2008



Hot Data Structure

- **Disk Caching with Virtual and Physical Pages**

Hot Data Structure

- **Disk Caching with Virtual and Physical Pages**
  - Start out with a single virtual page.

Hot Data Structure

- **Disk Caching with Virtual and Physical Pages**
  - Start out with a single virtual page.
  - Render some voxels into the tree until page capacity is reached.

Hot Data Structure

- **Disk Caching with Virtual and Physical Pages**
  - Start out with a single virtual page.
  - Render some voxels into the tree until page capacity is reached.
  - Split page into 8 sub-pages and attempt to add the overflow voxel again.

Hot Data Structure

- **Disk Caching with Virtual and Physical Pages**
  - Start out with a single virtual page.
  - Render some voxels into the tree until page capacity is reached.
  - Split page into 8 sub-pages and attempt to add the overflow voxel again.
  - Store out virtual pages to disk.
  - Load/Unload each page’s levels as necessary at runtime.

Hot Data Structure

- **Page capacity can be based on…**
  - CUDA's shared memory size
  - SPU local store size
  - Optimum disk streaming performance
  - Minimum physical page memory

Virtual Page Fragmentation

- **Traverse indexing oct-tree**
  - Write out pages according to optimal layout (breadth first, depth first, etc…)

Physical Page Fragmentation

- Constantly loading / unloading data fragments memory over time
- Bucket memory into sections and assign each page to a section.

Page Optimization

- **Execution time proportional to number of blocks.**

Page Optimization

- **Execution time proportional to number of blocks.**
- **Number of blocks can be reduced through translation.**

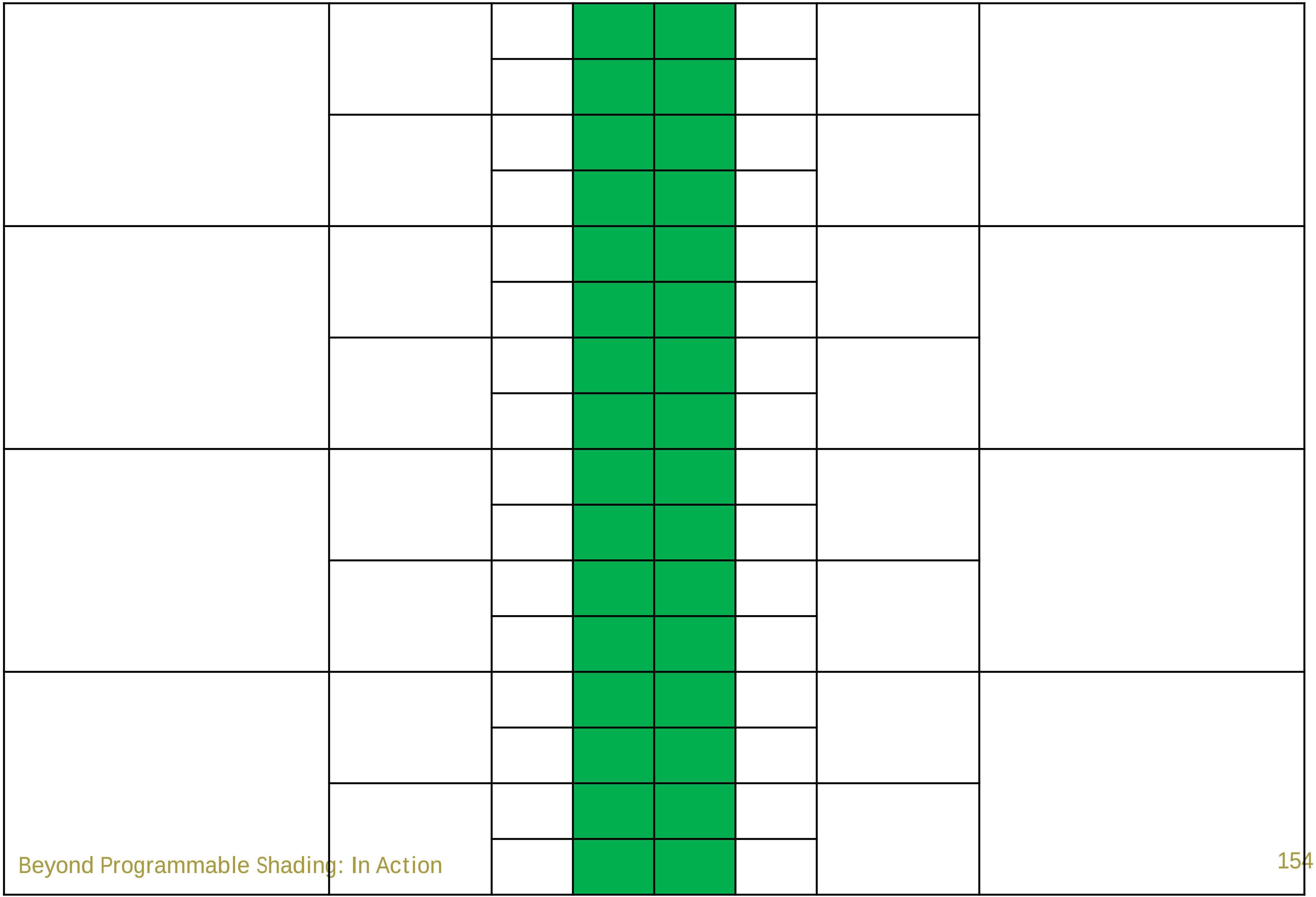
Page Optimization

- Execution time proportional to number of blocks.
- Number of blocks can be reduced through translation.
- Translating by 2^n doesn't affect any oct-tree level smaller than 2^n

Page Optimization

- **Create scratch page with enlarged region**
  - $2^{n+1} \times 2^{n+1} \times 2^{n+1}$
- **Apply successive translations of magnitude power of 2 in the x, y, & z directions and keep track of the number of nodes.**
- **Store off total translation for ray casting adjustment.**
- **$O(n * 2^{2n})$**
  - n is the number of levels in the oct-tree

Page Optimization

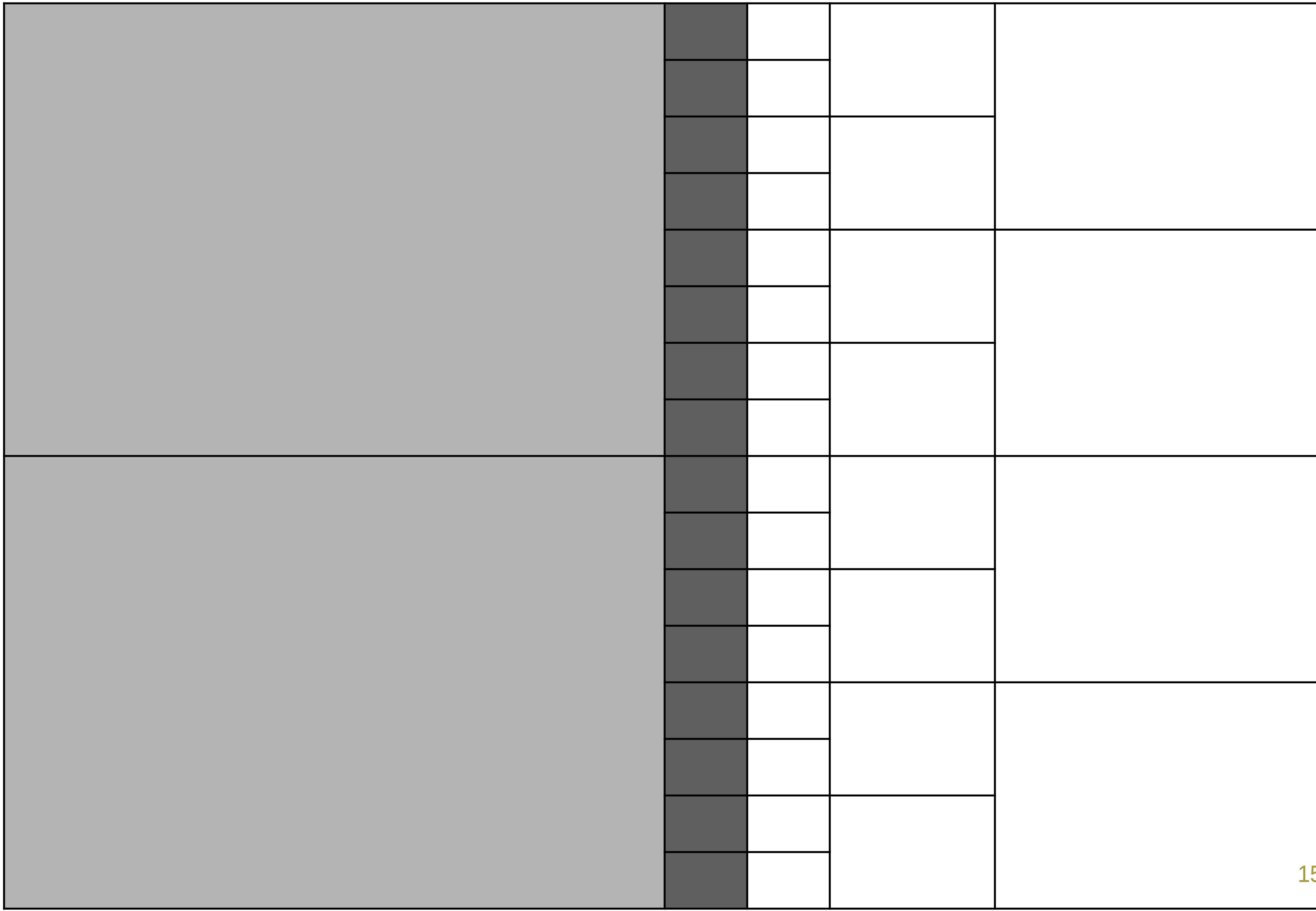


Page Optimization

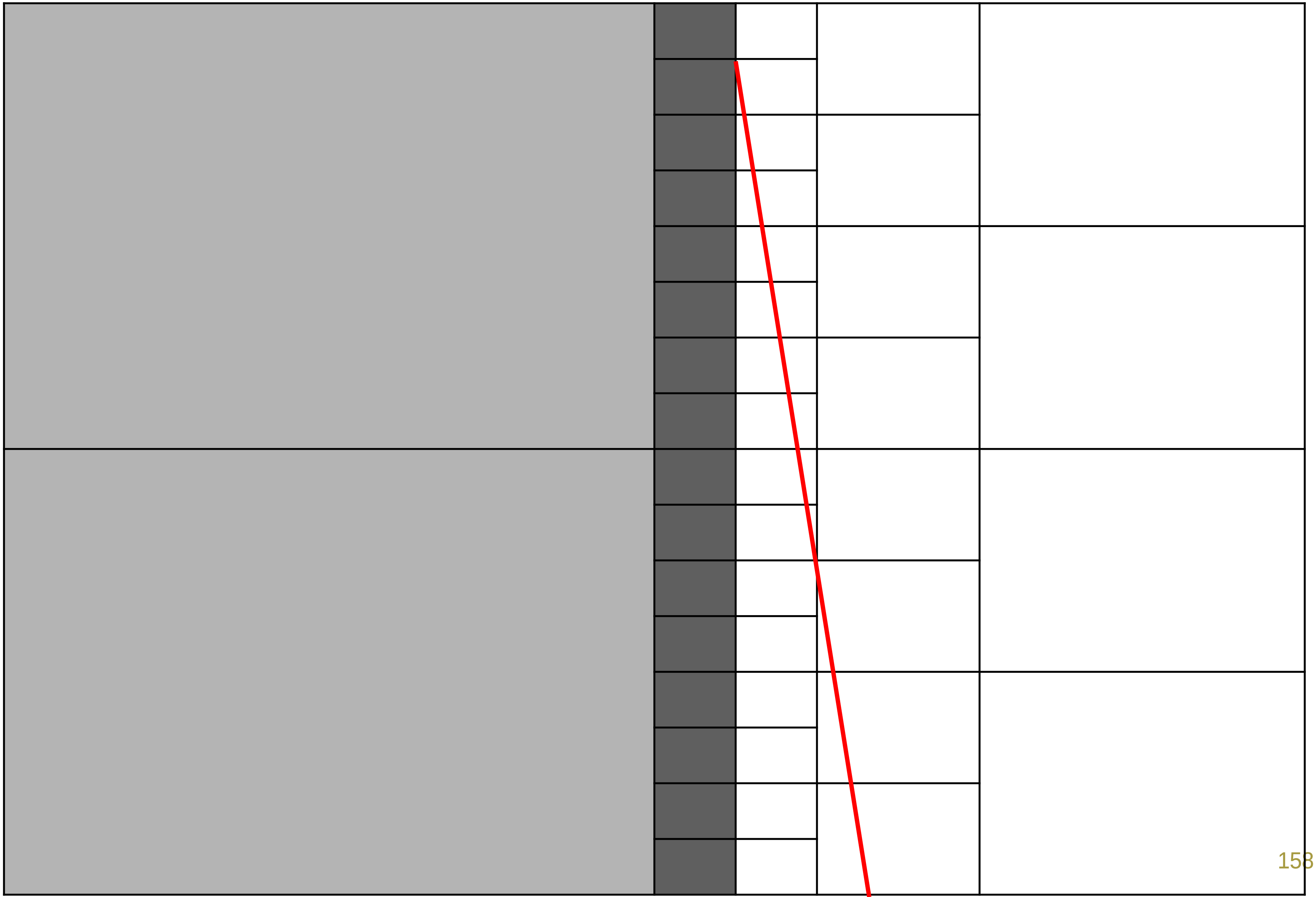
Page Optimization

- Minimize outside nodes for faster casting

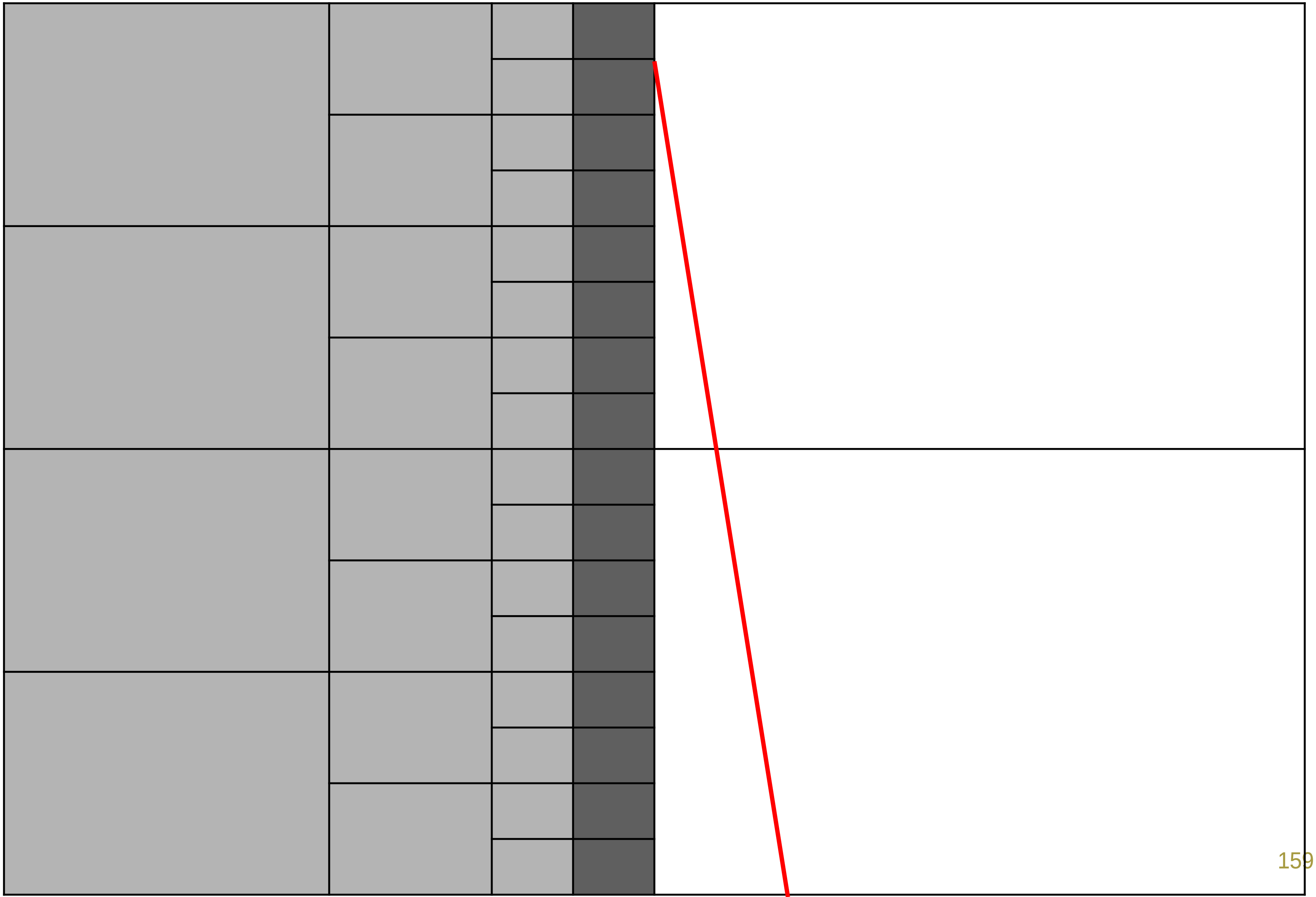
Page Optimization



Page Optimization



Page Optimization



Data Structure

- Different structures for editing, runtime and storage.

Runtime Data Structure

- child offsets : 32
- diffuse rgb : 3
- specular scale/power : 1
- planes : 12
- normal xyz : 3
- pad : 1
- total : 52 bytes per node

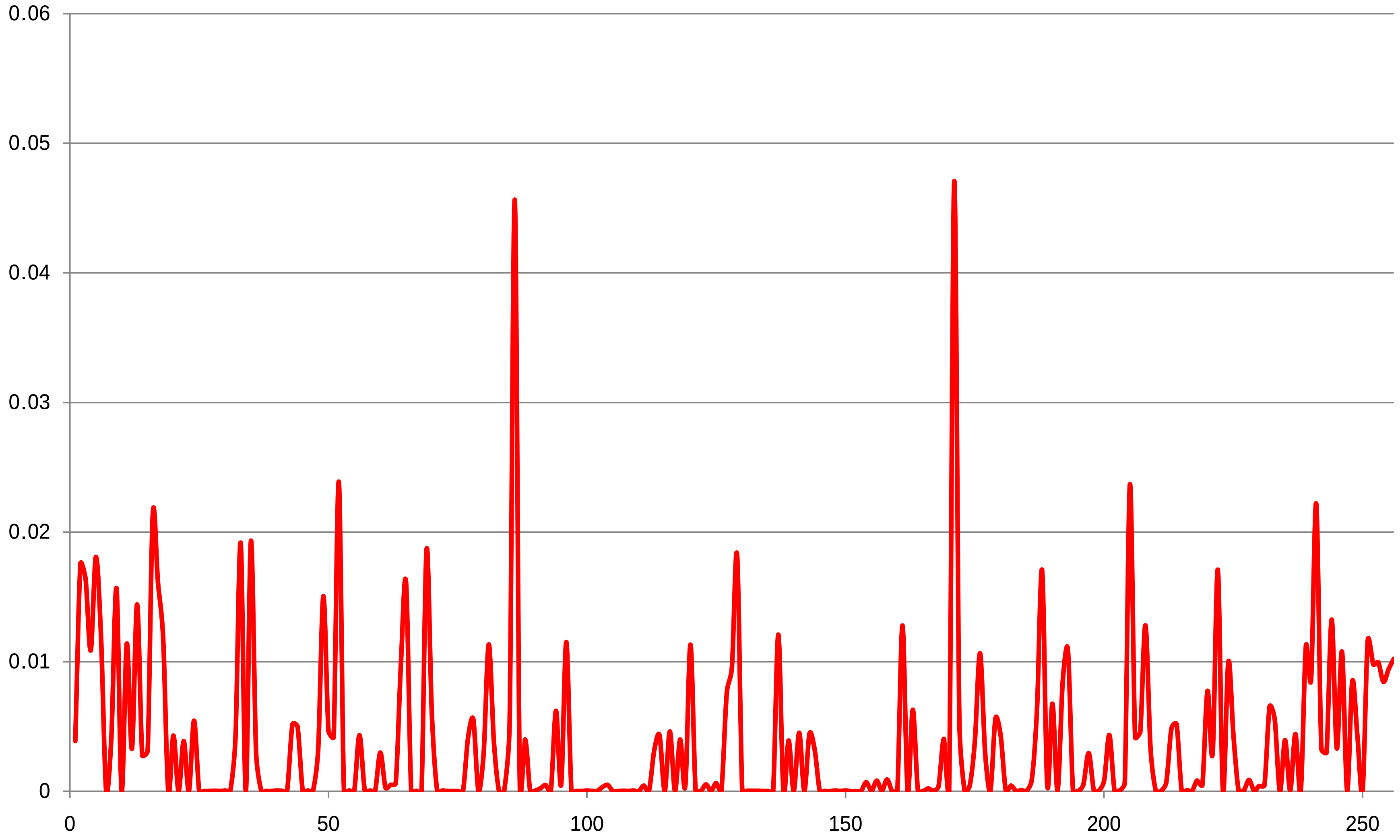
Storage Data Structure

- children bit mask : 1
- diffuse rgb : 3
- specular scale/power : 1
- normal xyz : 3
- total : 8 bytes per node

Data Compression

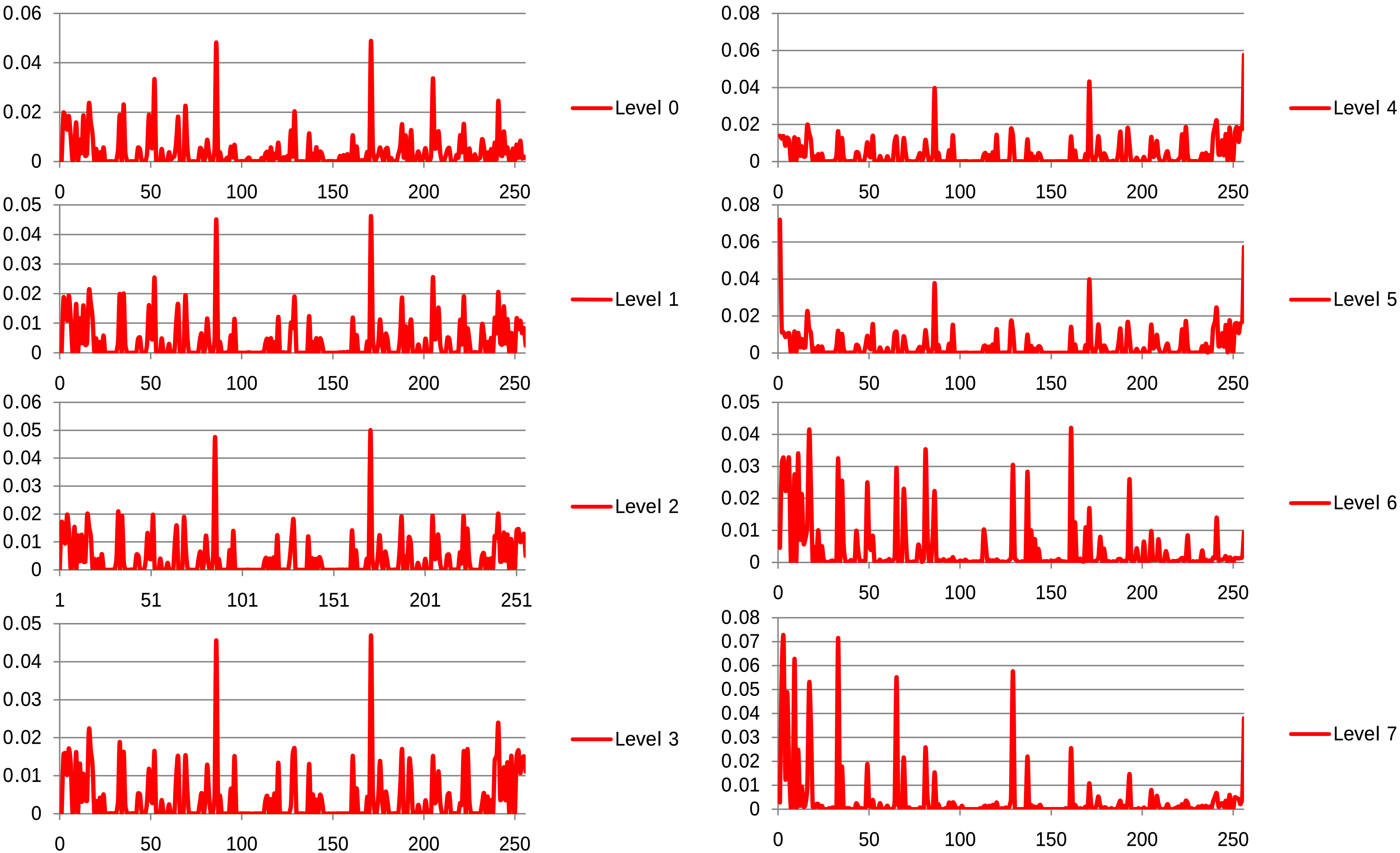
- Compressing child bits
- Compressing Colors

Compressing Child Bits



Compressing Child Bits

SIGGRAPH2008

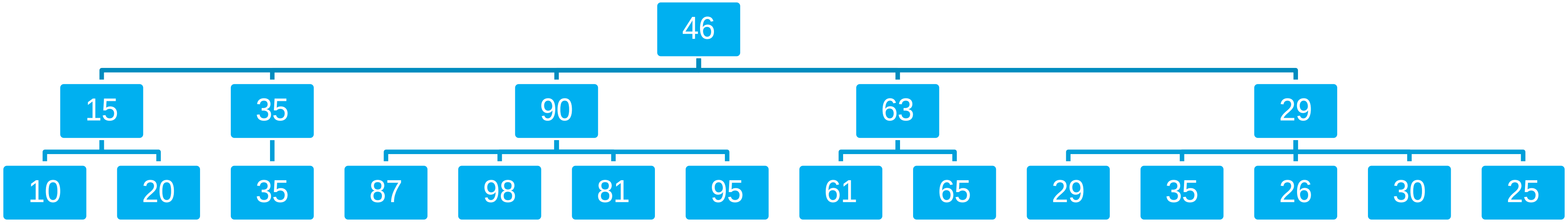


Compressing Child Bits

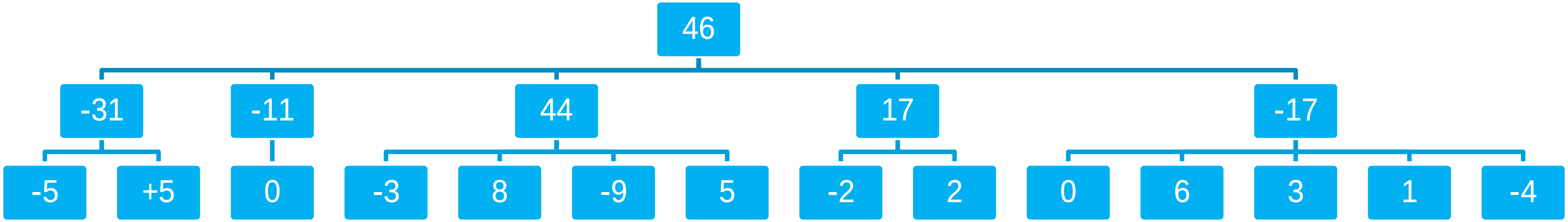
SIGGRAPH2008

- Split by oct-tree level.
- Entropy Encoding

Compressing Color Data



Compressing Color Data



Compressing Color Data

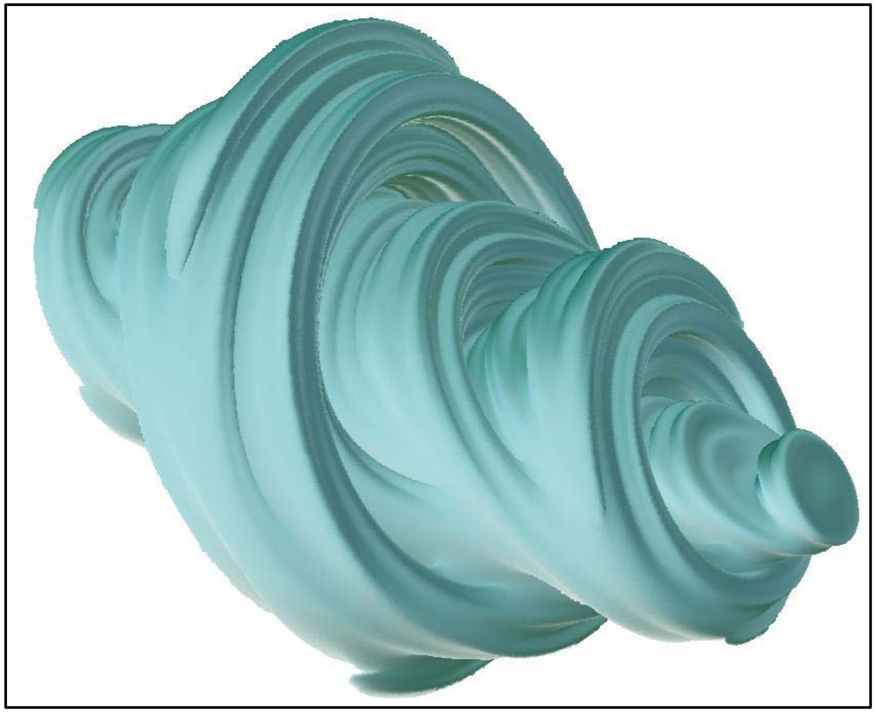
- Split by oct-tree level.
- Quantization
- Entropy Encoding
- 8:1 expected compression ratio

Data Storage Size

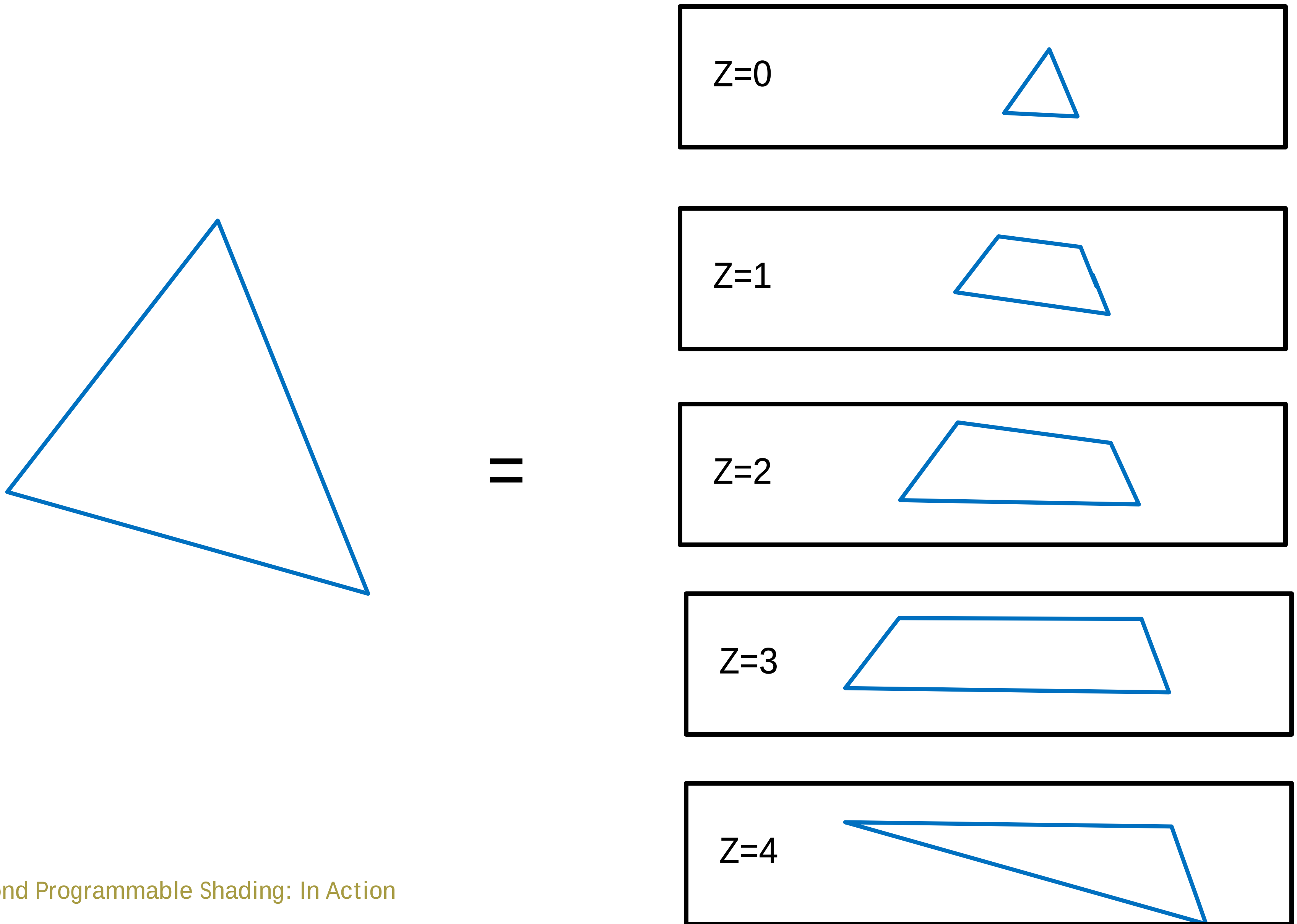
- **1.15 bits of positional data per voxel**
- **Cost savings improves as triangle size decreases.**
- **160 bits per triangle in traditional format**
  - x,z,y,s,t all 32-bits
  - 2.2:1 compression ratio
- **80 bits per triangle in compressed format**
  - x,y,z,s,t all 16-bits
  - 1.1:1 compression ratio
- **72 bits equivalent per triangle in oct-tree**
  - (for next generation)

Generating the Data

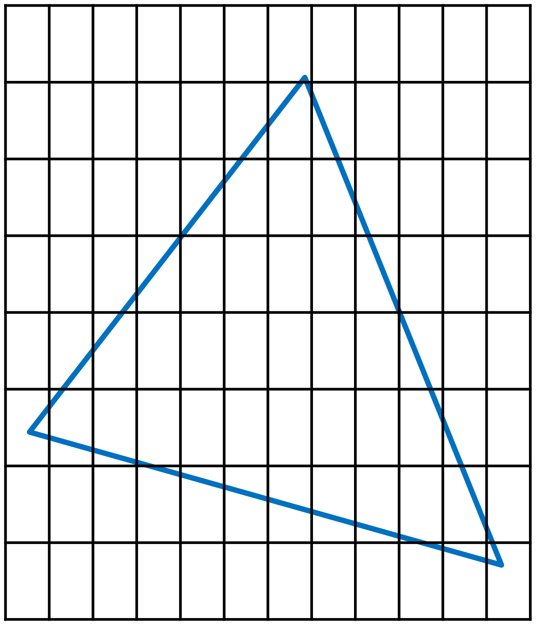
- **Every surface can enumerate into voxels.**
  - Triangles
    - 3D Scan Conversion, Volume Projection, Subdivision



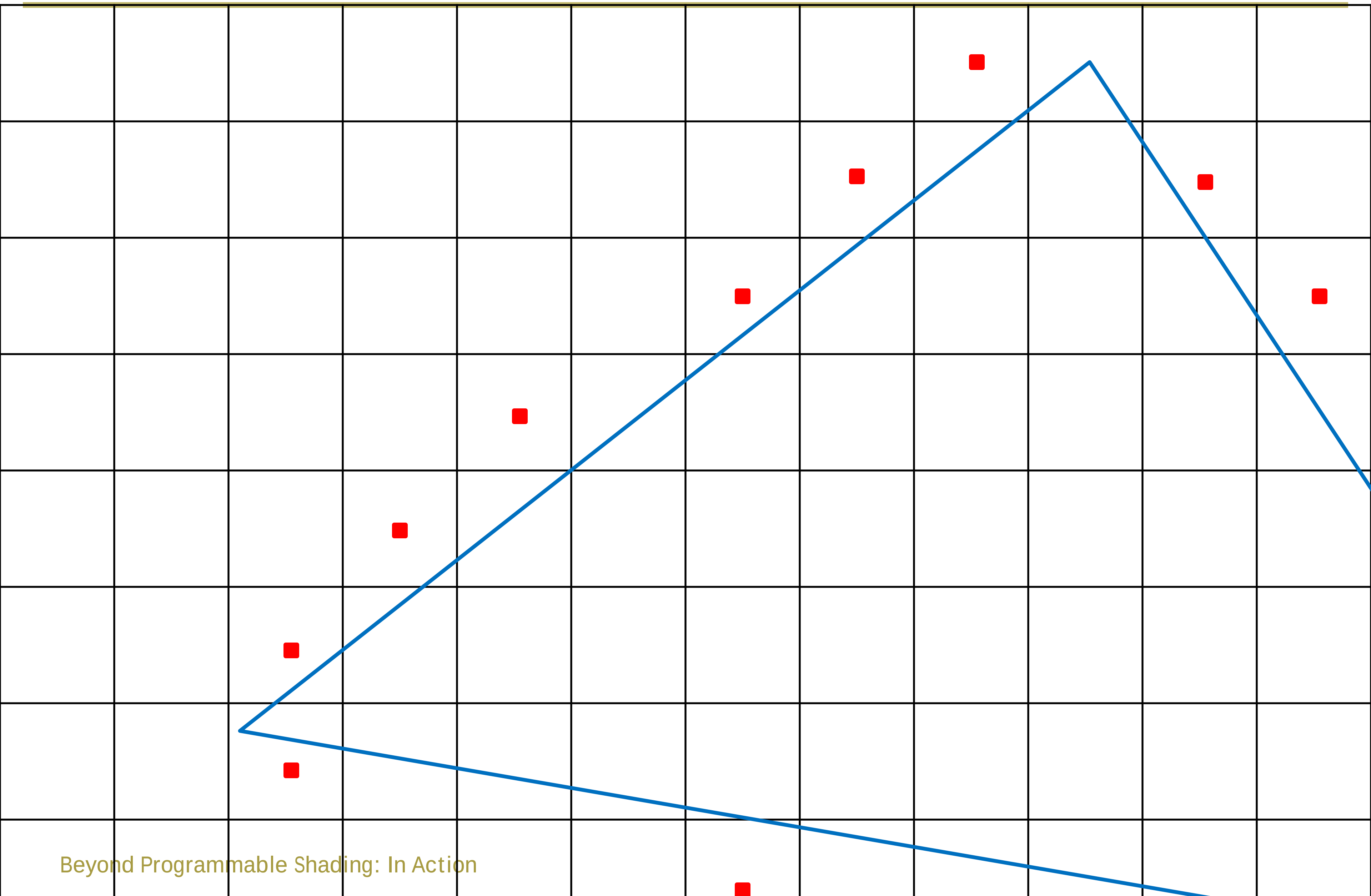
3D Scan Conversion



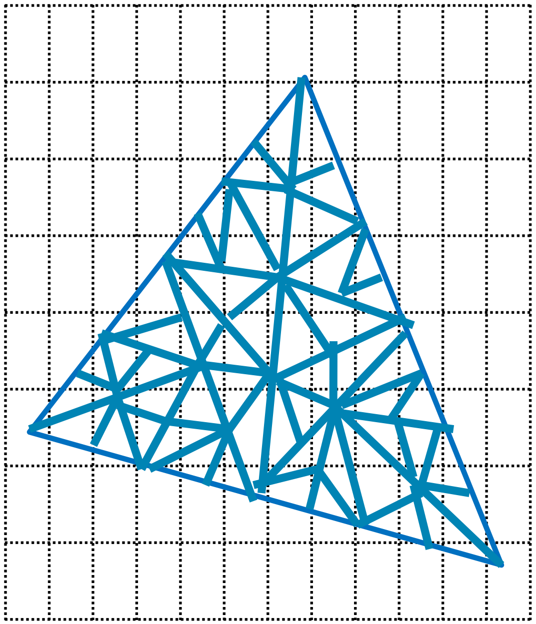
Volume Projection



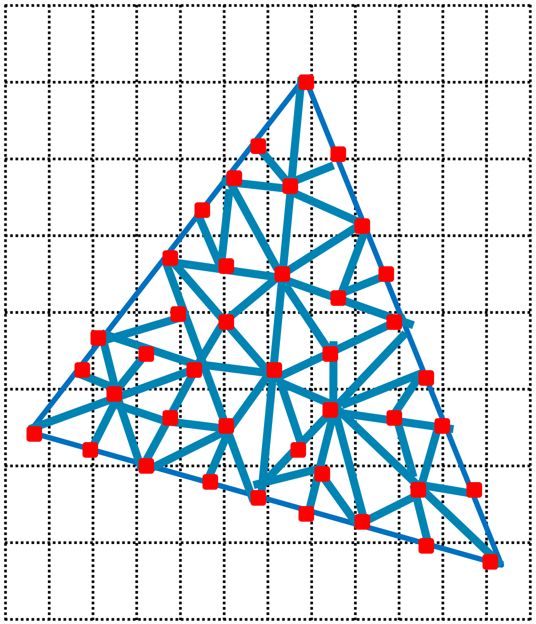
Volume Projection



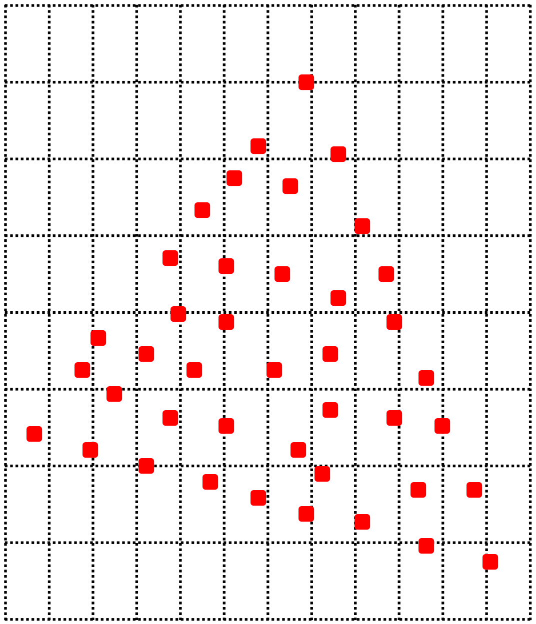
Subdivision



Subdivision



Subdivision



Generating the Data

- **Every surface can enumerate into voxels.**
  - 3D Scan Conversion, Volume Projection, Subdivision
- **Thick surfaces are unnecessary**

Generating the Data

- **Every surface can enumerate into voxels.**
  - 3D Scan Conversion, Volume Projection, Subdivision
- **Thick surfaces are unnecessary**
  - Flood fill world and remove unnecessary voxels.

Generating the Data

- **Every surface can enumerate into voxels.**
  - 3D Scan Conversion, Volume Projection, Subdivision
- **Thick surfaces are unnecessary**
  - Flood fill world and remove unnecessary voxels.
- **Generate geometry mip-maps**

Generating the Data

SIGGRAPH2008

- **Every surface can enumerate into voxels.**
  - 3D Scan Conversion, Volume Projection, Subdivision
- **Thick surfaces are unnecessary**
  - Flood fill world and remove unnecessary voxels.
- **Generate geometry mip-maps**
- **Perform ray-tracing to light the voxels.**

Using the Data

SIGGRAPH2008

- **For each pixel on the screen**
  - Shoot out a ray into the oct-tree and write out the node number (and depth)

Oct-tree Ray Traversal

- **Similar to KD-tree traversal. Clip the line with the mid-planes only.**
- **Tree traversal with two lookup tables.**
  - One to find which nodes to intersect with a given ray direction in a worst-case scenario.
  - The other to determine the order of intersection.
- **Faster than most stackless traversal methods for CUDA.**

LOD

- **How to handle oct-tree mip-mapping and when is it necessary to load additional detail levels?**

LOD - Stop Depth

- **Stop Depth based on pixel and voxel size**
  [Wald07]

LOD - Stop Depth

- **Stop Depth based on pixel and voxel size [Wald07]**
  - Oblique surfaces have unnecessary extra detail
    - Hurts casting performance by traversing detail that you won’t see
    - Hurts streaming performance by loading unnecessary data

LOD - Post Process

- **Ray casting outputs node indexes**

LOD - Post Process

- **Ray casting outputs node indexes**
- **A post process which looks at ratios of nodes to pixels.**
  - Small feedback buffer (320x180) contains list of pages which require additional detail.

LOD - Post Process

- **Ray casting outputs node indexes**
- **A post process which looks at ratios of nodes to pixels.**
  - Small feedback buffer (320x180) contains list of pages which require additional detail.
- **Up to 20% performance improvement**

Post Process Blur

- **Fixes the “Jam your head into a wall” scenario.**

Post Process Blur

- **Fixes the "Jam your head into a wall" scenario.**
- **Width of blur kernel related to size of voxel on screen.**

Rendering Dynamic Geometry

- **With voxels**
  - Option 1
    - Ray cast or rasterize a triangle mesh
    - Transform to base pose
    - Trace with local oct-tree
    - Allows instancing of geometry

Rendering Dynamic Geometry

- **With voxels**
  - Option 1
    - Ray cast or rasterize a triangle mesh
    - Transform to base pose
    - Trace with local oct-tree
    - Allows instancing of geometry
  - Option 2
    - Have two different oct-trees: static & dynamic
    - Render both and merge results together with depth information

Rendering Dynamic Geometry

- **With voxels**
  - Option 1
    - Ray cast or rasterize a triangle mesh
    - Transform to base pose
    - Trace with local oct-tree
    - Allows instancing of geometry
  - Option 2
    - Have two different oct-trees: static & dynamic
    - Render both and merge results together with depth information
- **With triangles**
  - Hybrid rendering via rasterization and deferred shading.

Depth Advance Optimization

SIGGRAPH2008

- **Render a coarse hull of the geometry into a depth-buffer.**
  - Automatically calculate from voxel geometry.
- **Start the ray casting at the depth-buffer values.**

Depth Advance Optimization

- **Render a coarse hull of the geometry into a depth-buffer.**
  - Automatically calculate from voxel geometry.
- **Start the ray casting at the depth-buffer values.**
- **Skips most of the traversal process.**
  - Up to 2x speed improvement

Depth Advance Optimization

- **Render a coarse hull of the geometry into a depth-buffer.**
  - Automatically calculate from voxel geometry.
- **Start the ray casting at the depth-buffer values.**
- **Skips most of the traversal process.**
  - Up to 2x speed improvement
  - Less sensitive to scene complexity

Adaptive Sub-Sampling

SIGGRAPH2008

- After rendering the scene, perform a Sobel edge filter over the frame buffer to figure out where additional rays would improve the quality of the image.
- Cast additional rays.
- Repeat until 16 ms.

Adaptive Sub-Sampling Problems

- Inherently always sampling the most divergent parts of the scene
- Can manage performance hit by sampling highly aliased to less aliased in chunks

Infinite Surface Detail

- Oct-tree node's recursively point back in on themselves to create an infinite amount of detail
- Create detail octree sub-segments to simulate rough, smooth, porous, sharp edges, etc..
- Programatically simulate virtual detail levels.

How much time to innovate?

- 1 year tools
- 3 months runtime

Expected Runtime Performance

- 33% of the time rendering characters / etc
- 66% of the time rendering world
- Ray-casting the world must complete in ~20ms for 30 FPS
- Theoretically possible on today's technology at 720p and 30 fps (GeForce 8800 Series)

Expected Runtime Performance

- **33% of the time rendering characters / etc**
- **66% of the time rendering world**
- **Ray-casting the world must complete in ~20ms for 30 FPS**
- **Theoretically possible on today's technology at 720p and 30 fps (GeForce 8800 Series)**
  – Theory falls a little short of reality.

How would this affect a platform launch?

- Generational skip in geometric complexity
- Next gen platforms 4 times better at least
- 60 FPS at 1080p with Anti-aliasing

Special Thanks

- **Paul Debevec**
  - Light probes used with permission
- **Dimitry Parkin**
  - www.parkparkin.com
- **John Carmack**
- **Cass Everitt**
- **Mark Harris**
- **Nathaniel Duca**
- **Aaron Lefohn**
- **Mike Houston**
- **Tom Forsyth**
- **Sony**
- **Intel**
- **Nvidia**

Questions

Jon Olick (jon.olick@gmail.com)

id Software

References

- http://www.sci.utah.edu/~wald/Publications/2007/MROct/download/mroct.pdf
  - [Wald07]