MARCH 5-9
2007
SAN FRANCISCO

MOSCONE
CENTER

TAKE CONTROL

www.gdconf.com

GameDevelopers Conference 07
TAKE CONTROL
March 5-9, 2007 in San Francisco
CMP

RSX™ Best Practices

Mark Cerny, Cerny Games

David Simpson, Naughty Dog

Jon Olick, Naughty Dog

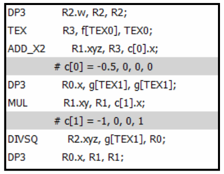
RSX™ Best Practices

- About libgcm
- Using the SPUs with the RSX™
- Brief overview of GCM Replay

December 7th, 2004

Sony Computer Entertainment and NVidia announce joint development of RSX™

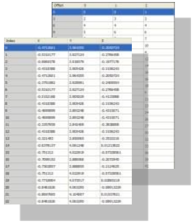
Fragment Programs



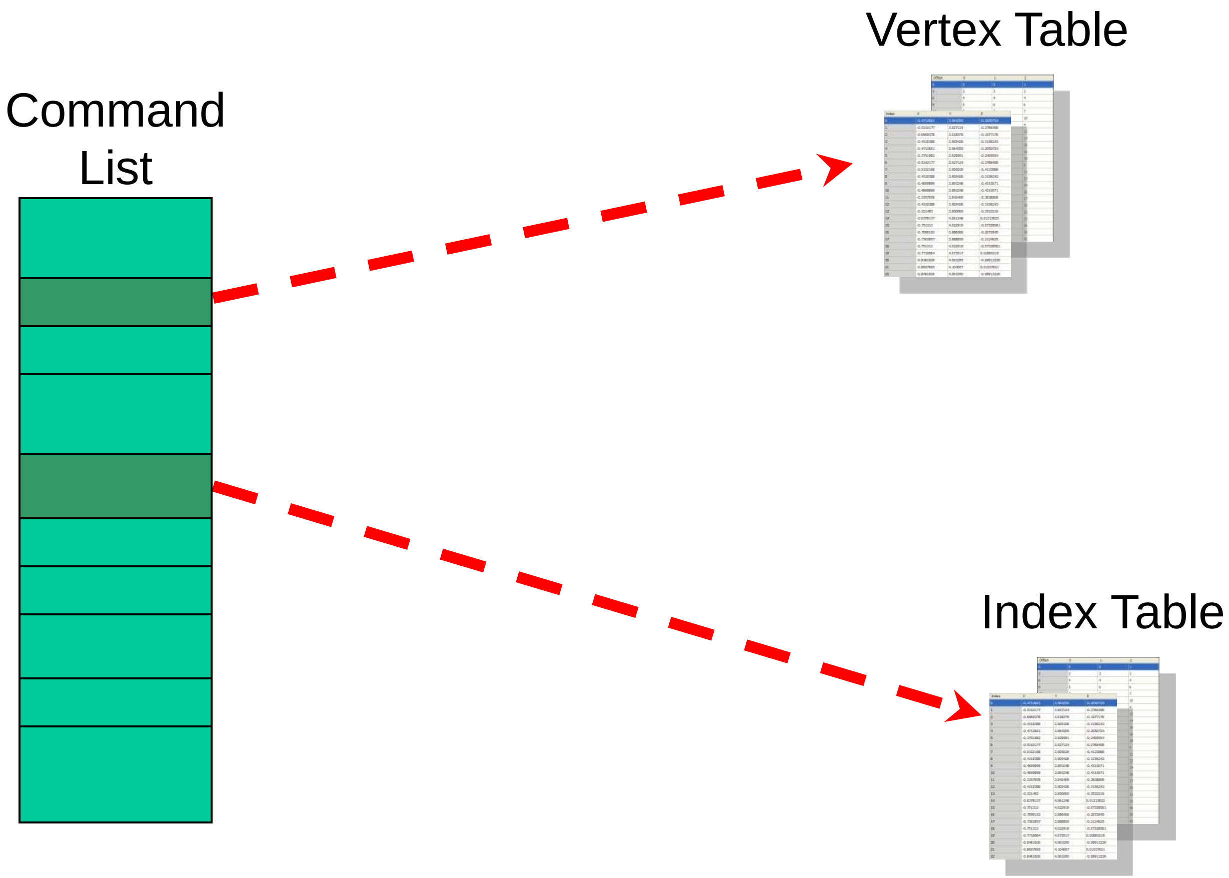
```
DP3       R2.w, R2, R2;
TEX       R3, f[TEX0], TEX0;
ADD_X2    R1.xyz, R3, c[0].x;
          # c[0] = -0.5, 0, 0, 0
DP3       R0.x, g[TEX1], g[TEX1];
MUL       R1.xy, R1, c[1].x;
          # c[1] = -1, 0, 0, 1
DIVSQ     R2.xyz, g[TEX1], R0;
DP3       R0.x, R1, R1;
```

Textures

Vertex & Index Buffers



PLAYSTATION®2 Command List

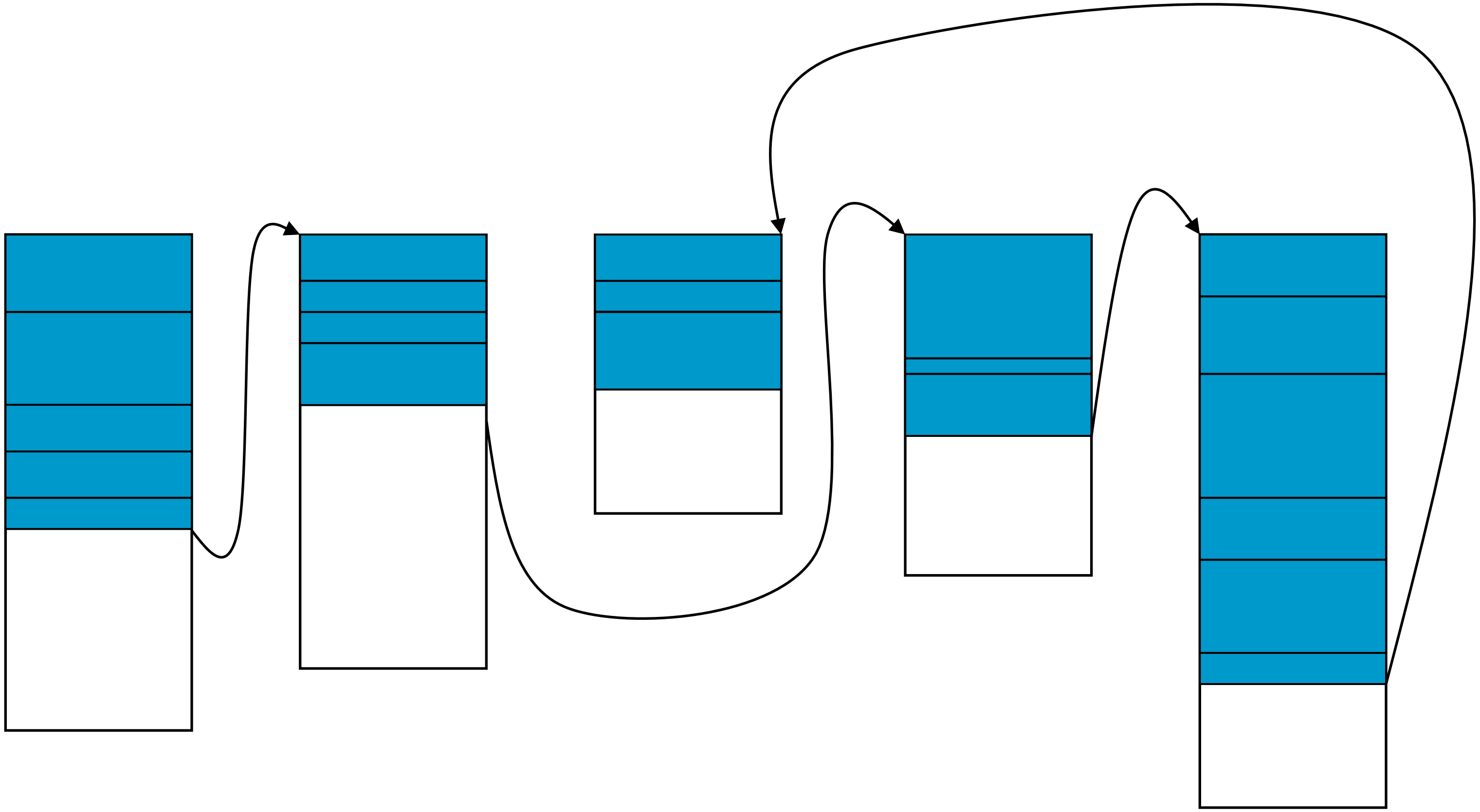


Vertex Table

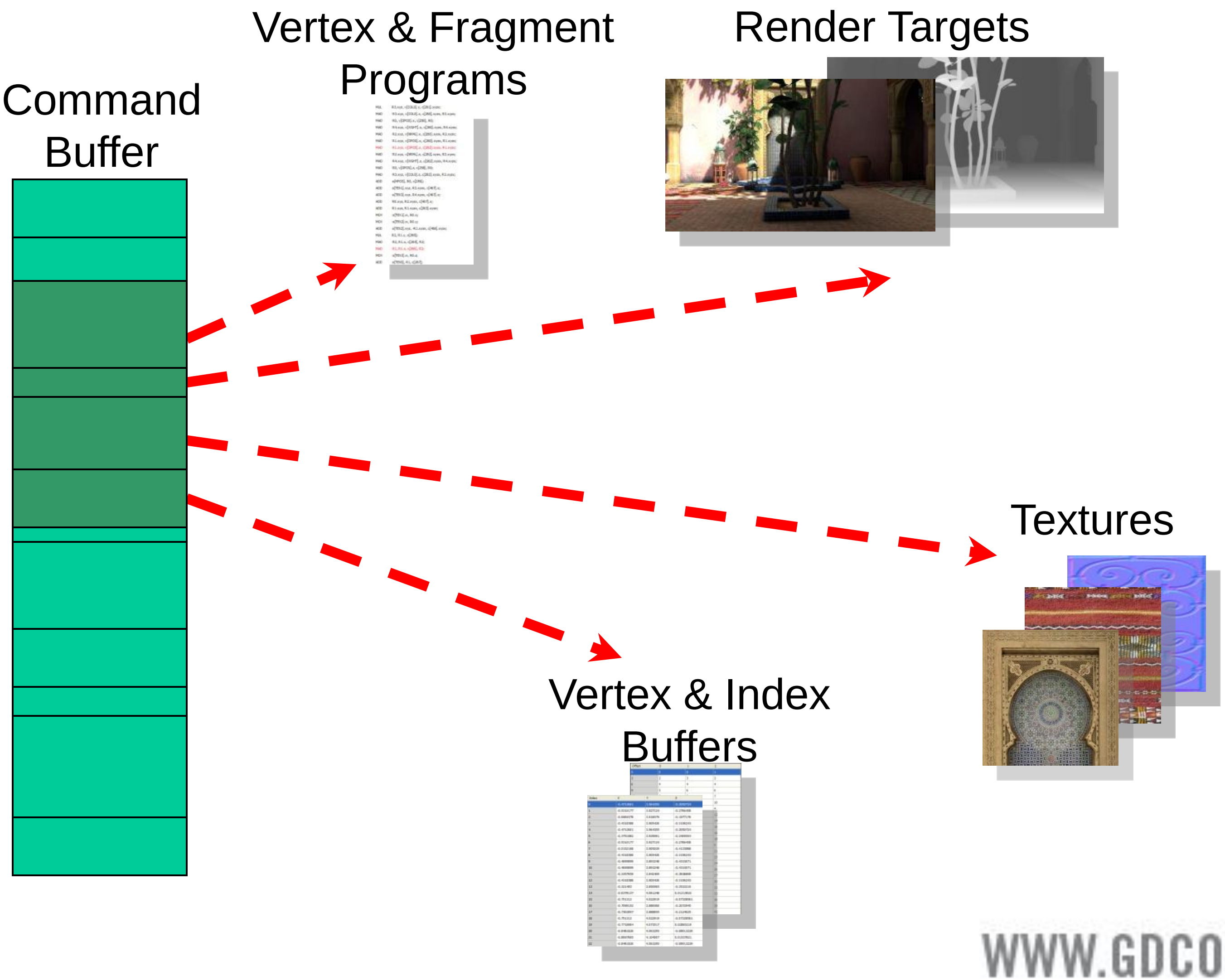
Command List

Index Table

PLAYSTATION®2 Command List Construction



PLAYSTATION®3 Command Buffer



Command Buffer

Vertex & Fragment Programs

Render Targets

Textures

Vertex & Index Buffers

State in OpenGL

```
glActiveStencilFaceEXT(GL_BACK);
glStencilOp(GL_KEEP,GL_KEEP,GL_DECR_WRAP_EXT);
glActiveStencilFaceEXT(GL_FRONT);
glStencilOp(GL_KEEP,GL_KEEP,GL_INCR_WRAP_EXT);
```

More State in OpenGL

```
glClearDepth(depth);
glClearStencil(s);
glClear(GL_DEPTH | GL_STENCIL)
```

Goals with libgcm

Goals with libgcm

- Support multiple buckets

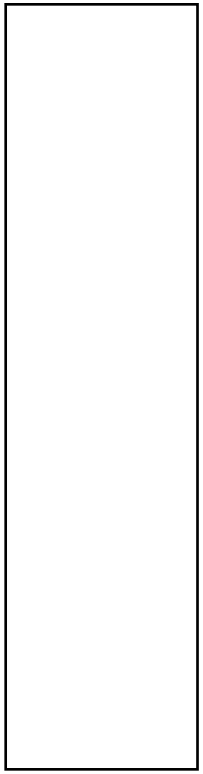
Goals with libgcm

- Support multiple buckets
- Remove state

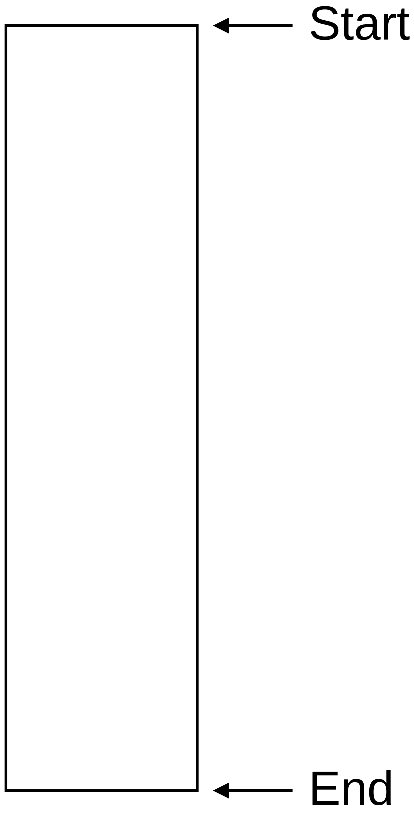
Goals with libgcm

- Support multiple buckets
- Remove state
  - glClearDepth and glClearStencil become a single function

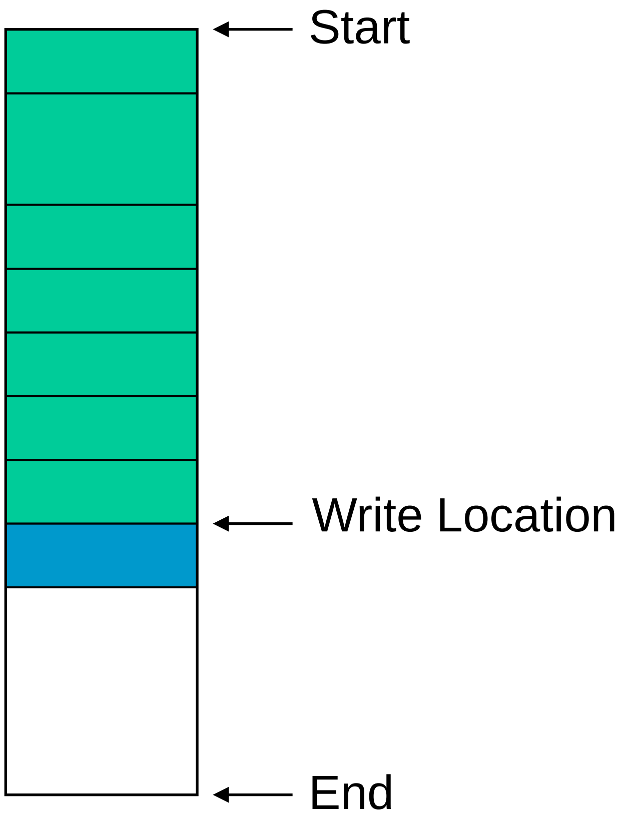
libgcm Context Structure



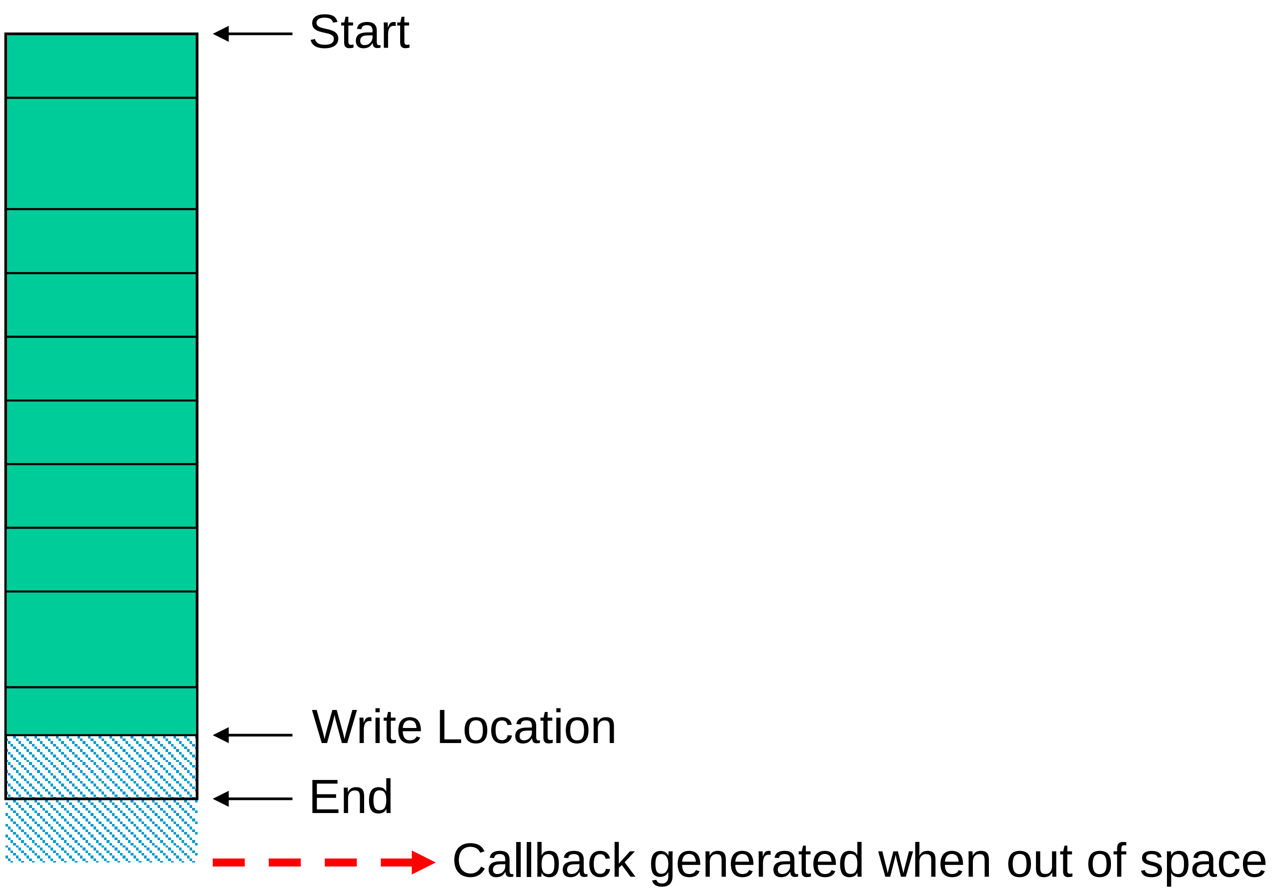
libgcm Context Structure



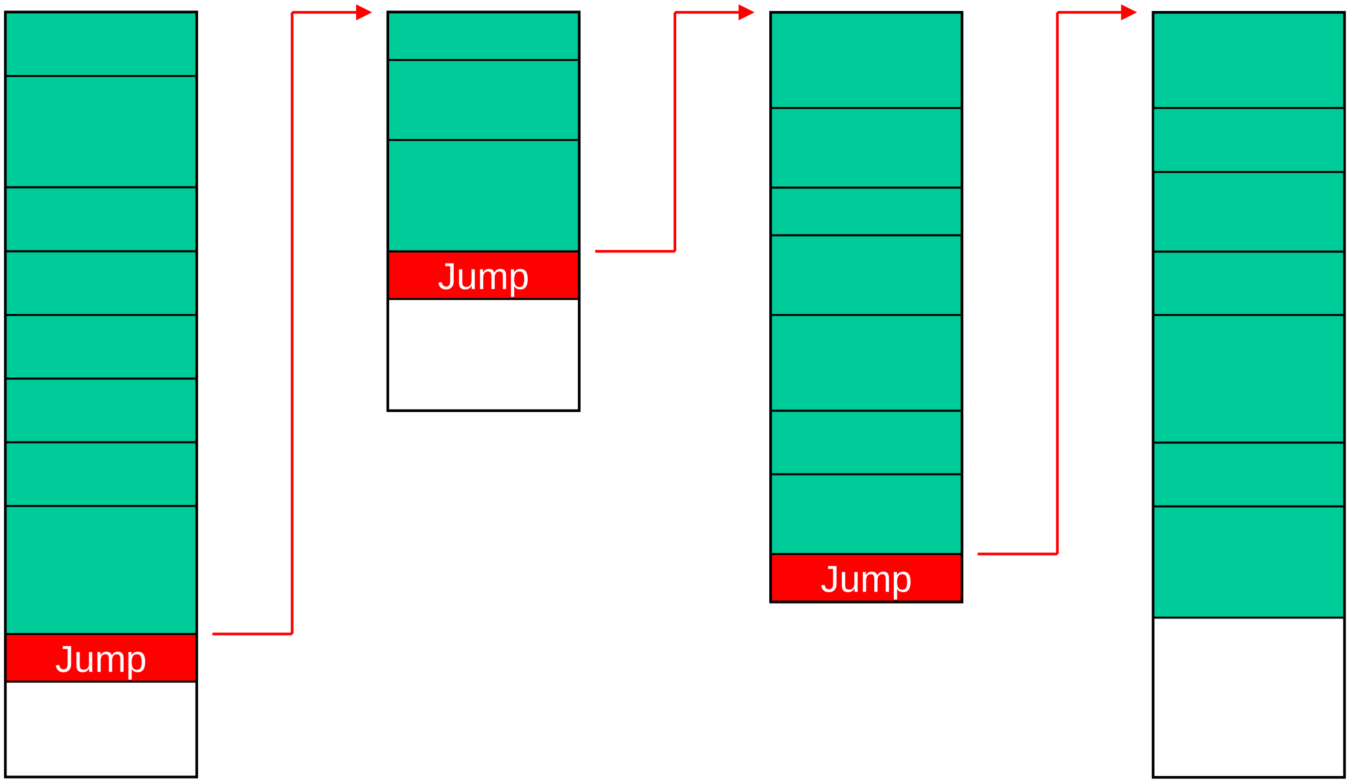
libgcm Context Structure



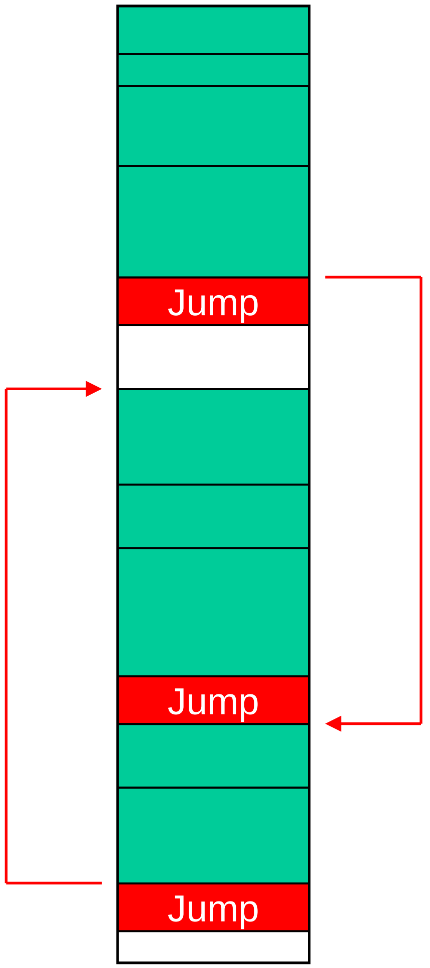
libgcm Context Structure



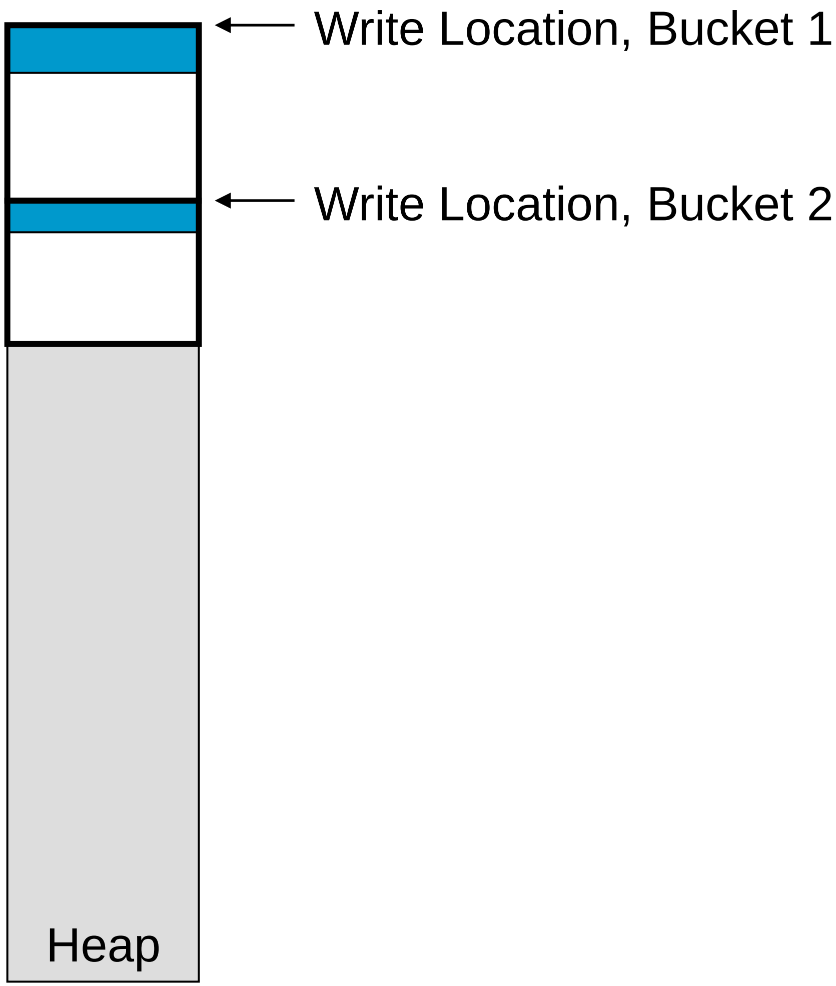
Multiple Buffers



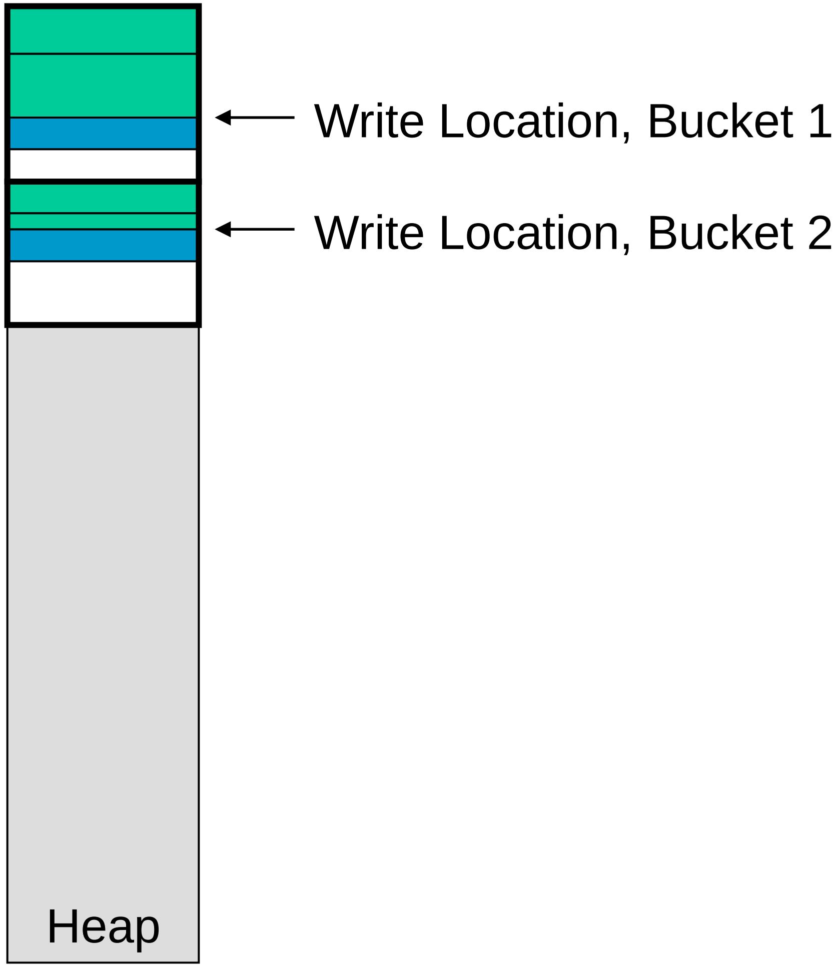
Single Large Buffer



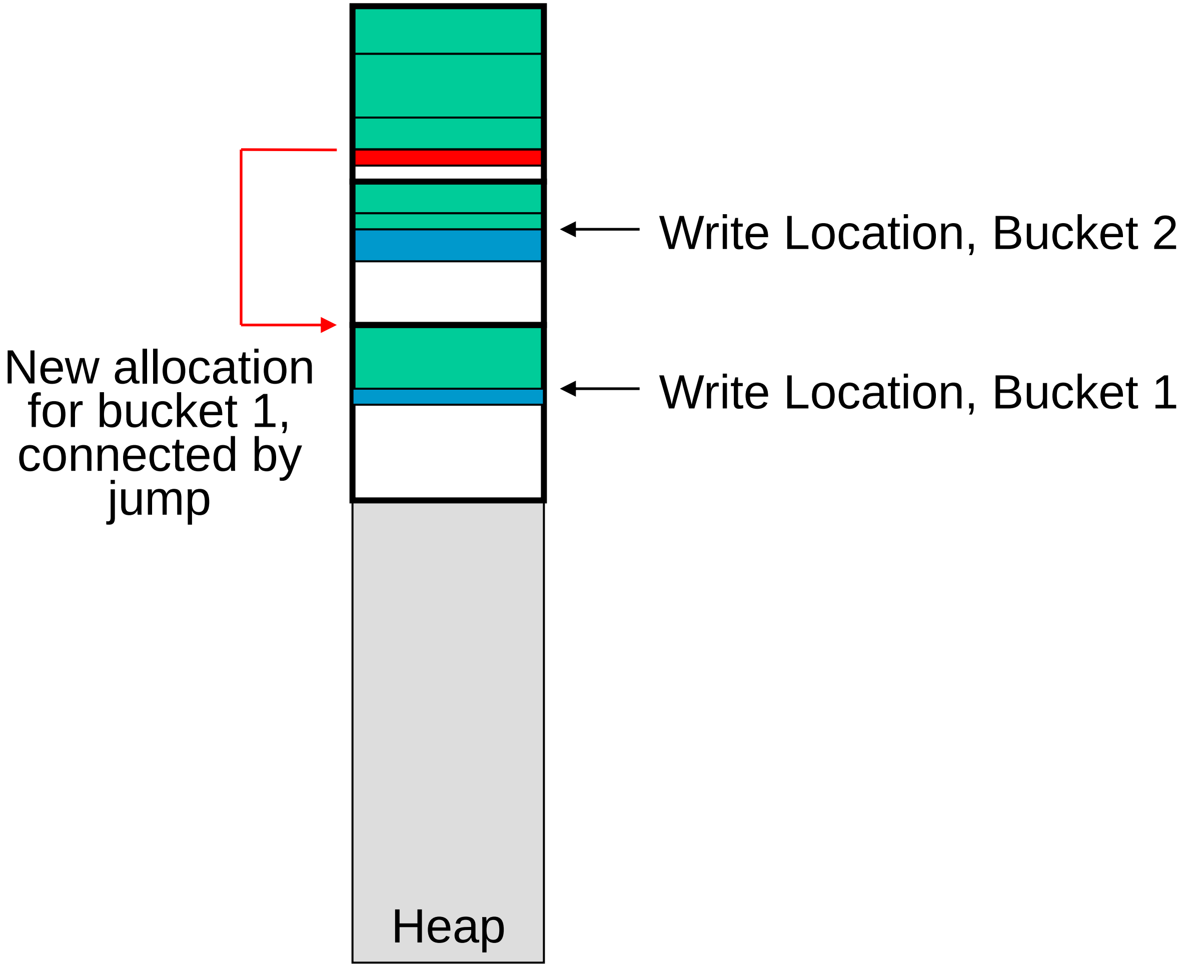
Virtual Buckets



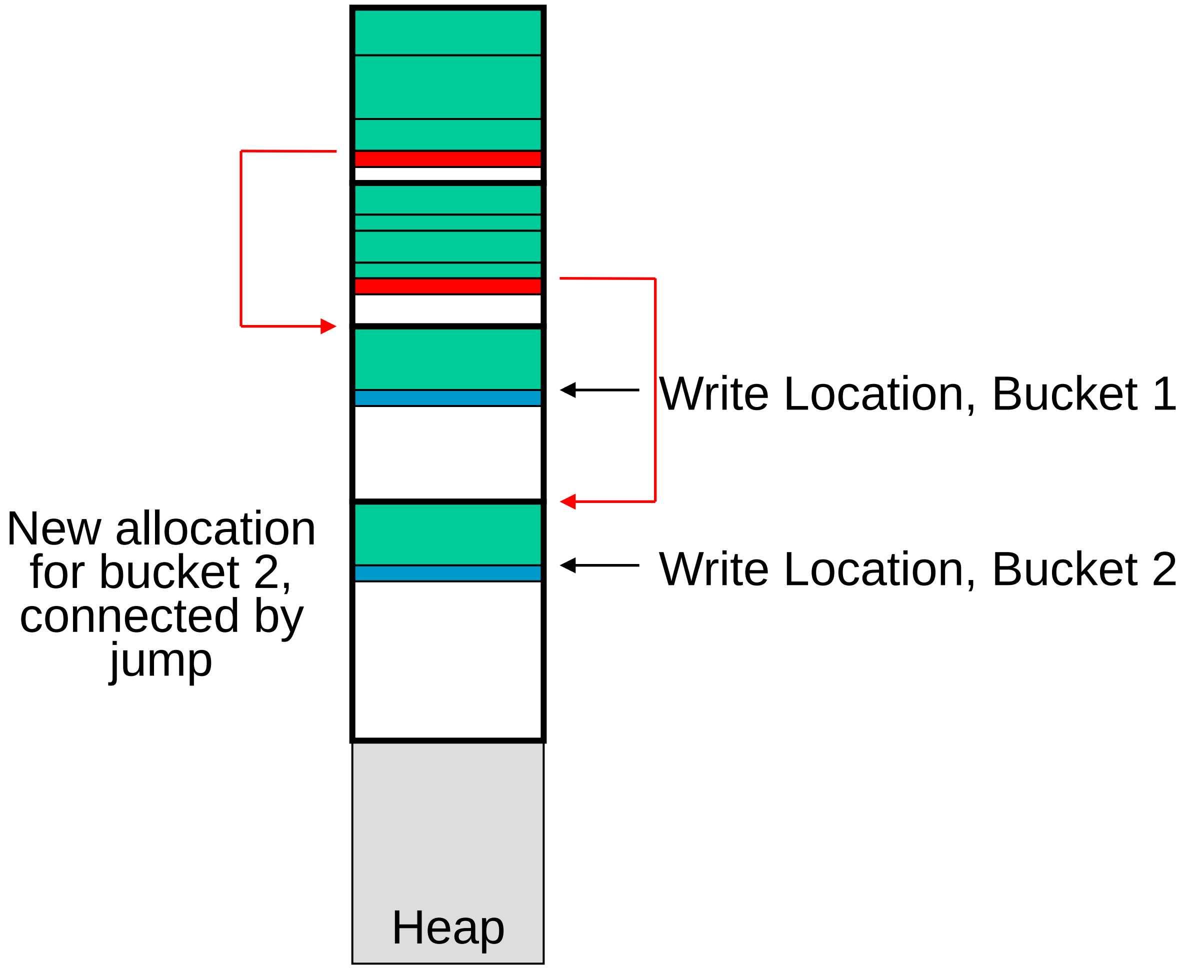
Virtual Buckets



Virtual Buckets



Virtual Buckets



Context Types

- Space Checking
- No Space Checking

Set Alpha Blend

Space Checking	5.0x
No Space Checking	1.1x

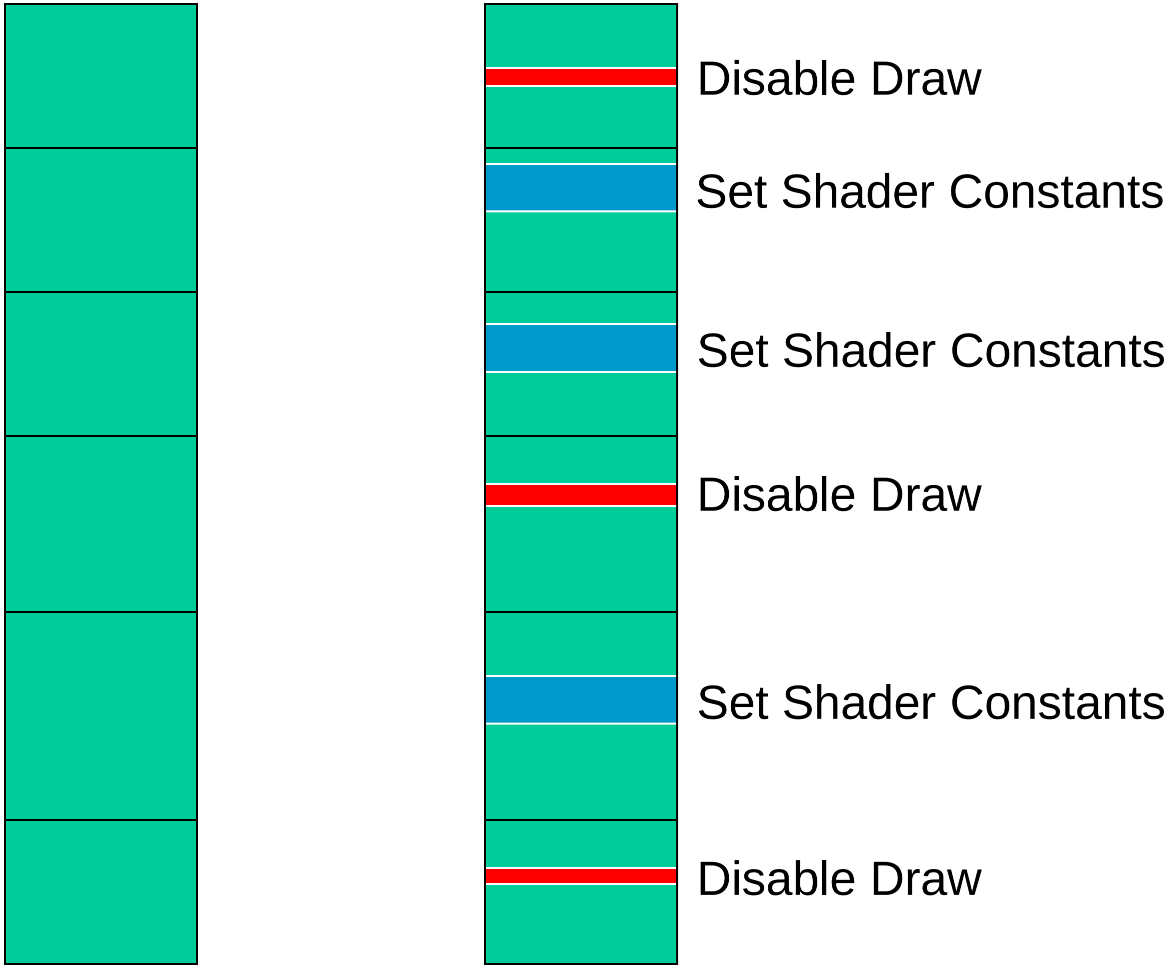
Setup Texture for Shader

Space Checking	1.8x
No Space Checking	1.8x

Space Checking

- Set up Alpha Blend
  - 1.1x
- Set up Texture for Shader
  - 1.05x

Patch Static Command Buffer



Disable Draw

Set Shader Constants

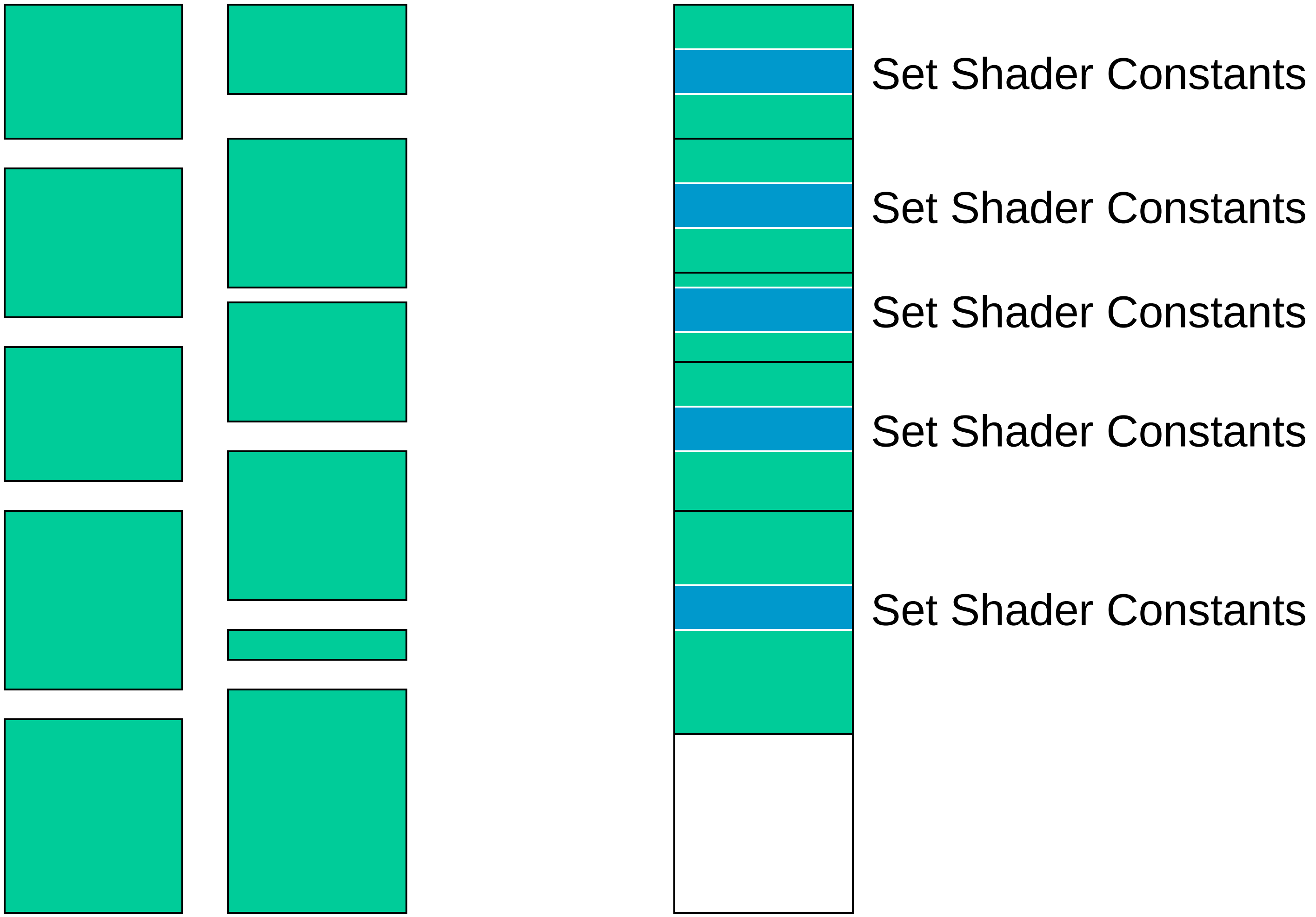
Set Shader Constants

Disable Draw

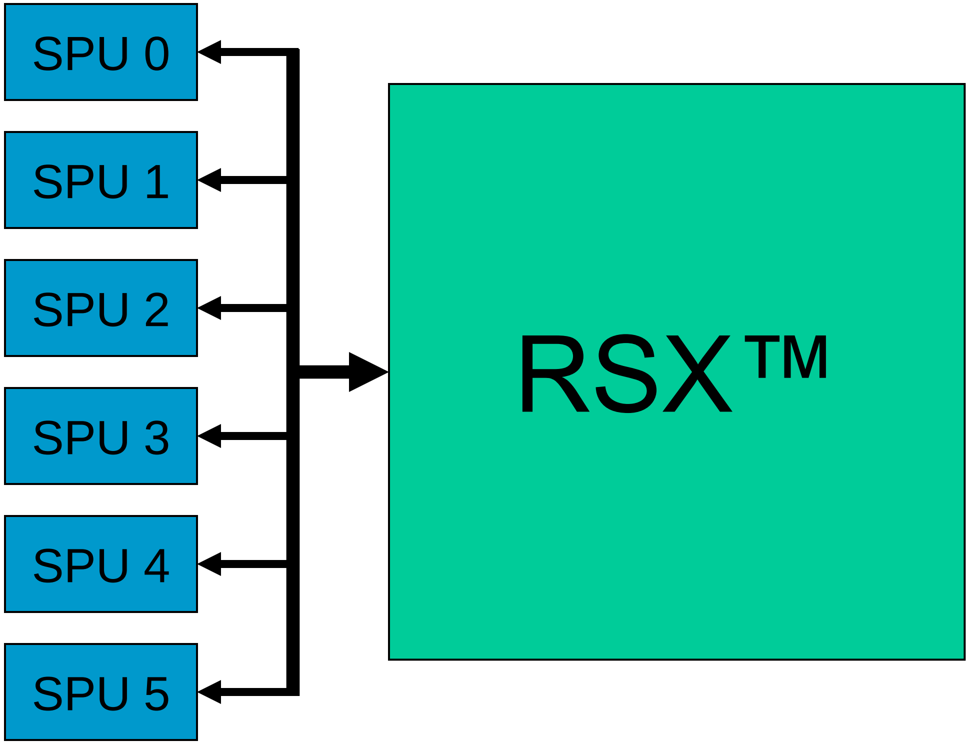
Set Shader Constants

Disable Draw

Concatenate Static Command Buffers



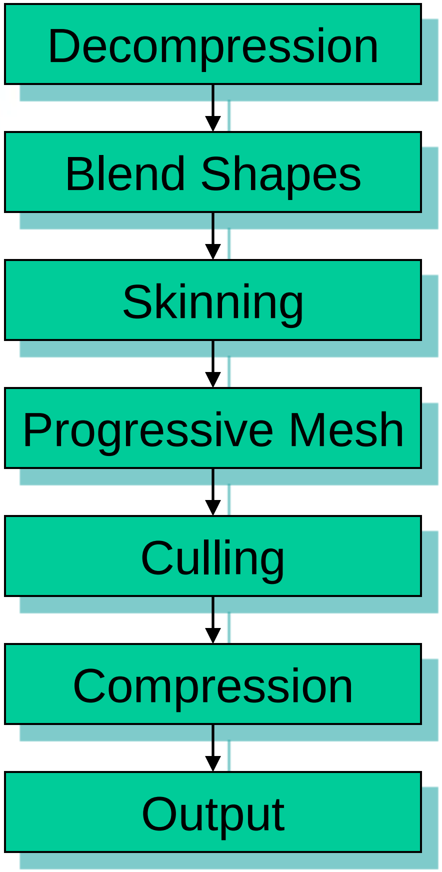
Using the RSX™ with the SPUs



Using the RSX™ with the SPUs

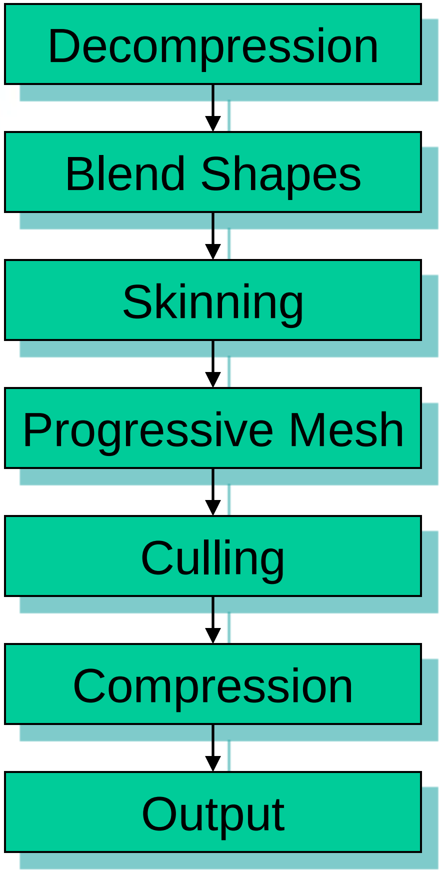
- SPUs can be used to supercharge vertex processing on the RSX™
- SPUs can perform triangle and mesh operations that cannot be performed on the RSX™

Geometry Processing Pipeline



- Runs on SPUs
- Modular
  - Need only to use some pieces
- Outputs index and vertex data which is directly read by the RSX™

Geometry Processing Pipeline



Decompression

Blend Shapes

Skinning

Progressive Mesh

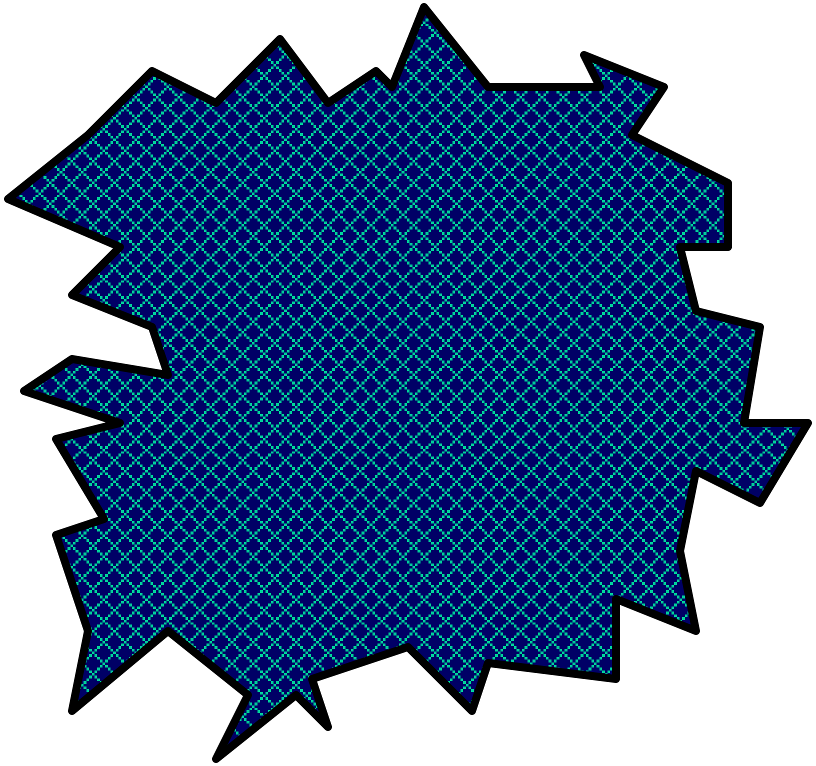
Culling

Compression

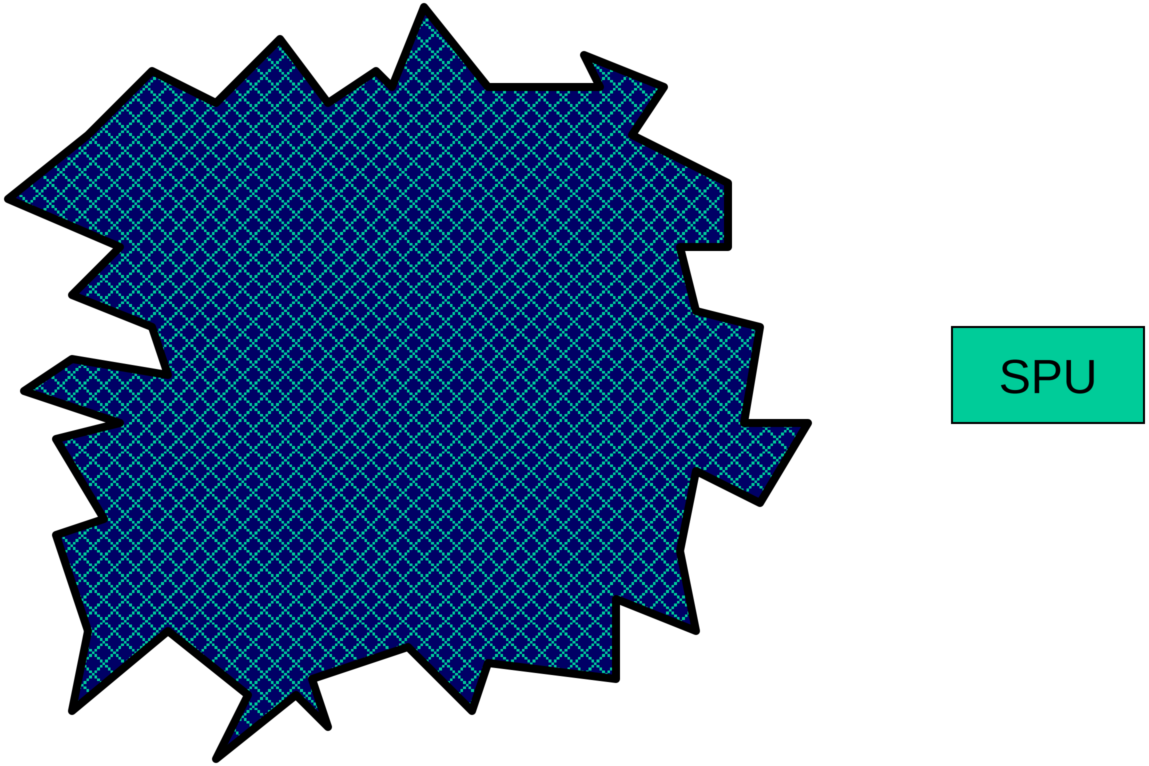
Output

- SPU processes one vertex set at a time
  - One or more vertex sets are generated per mesh in an offline tools processing step called partitioning

The RSX™ can process vertices in large chunks

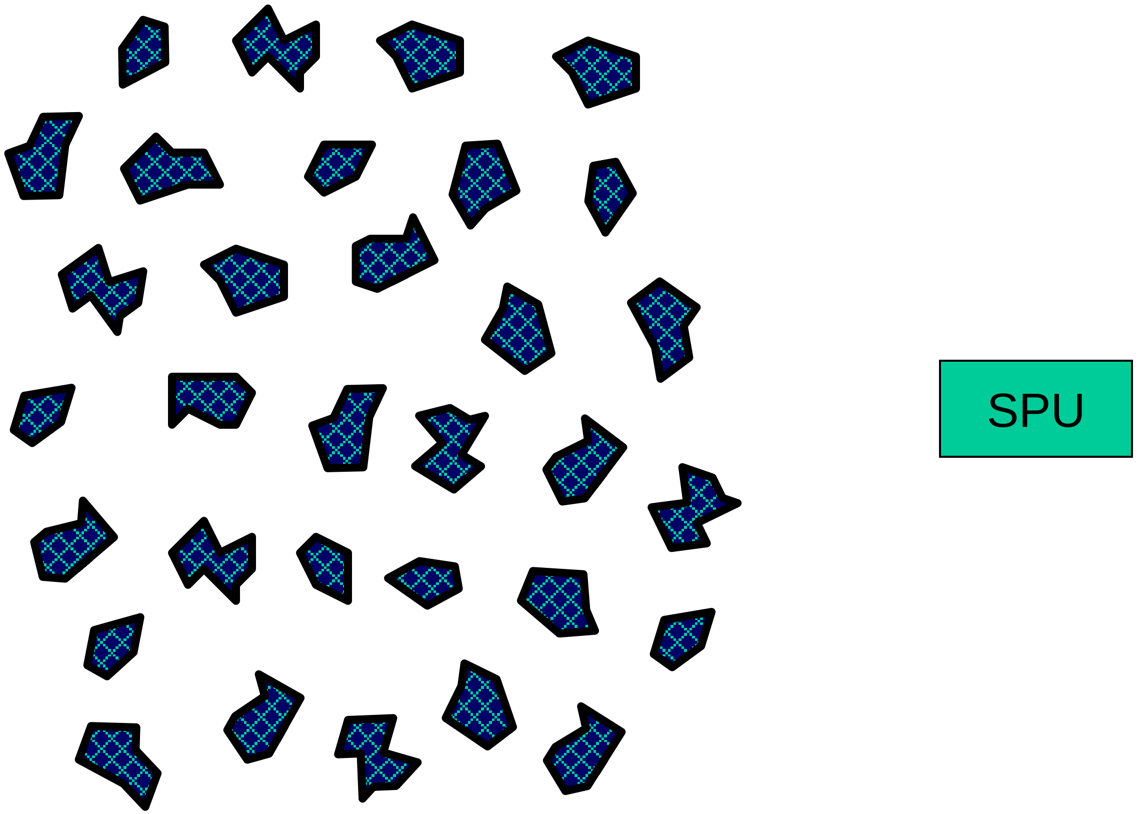


But a 50,000 vertex object won't fit in an SPU!

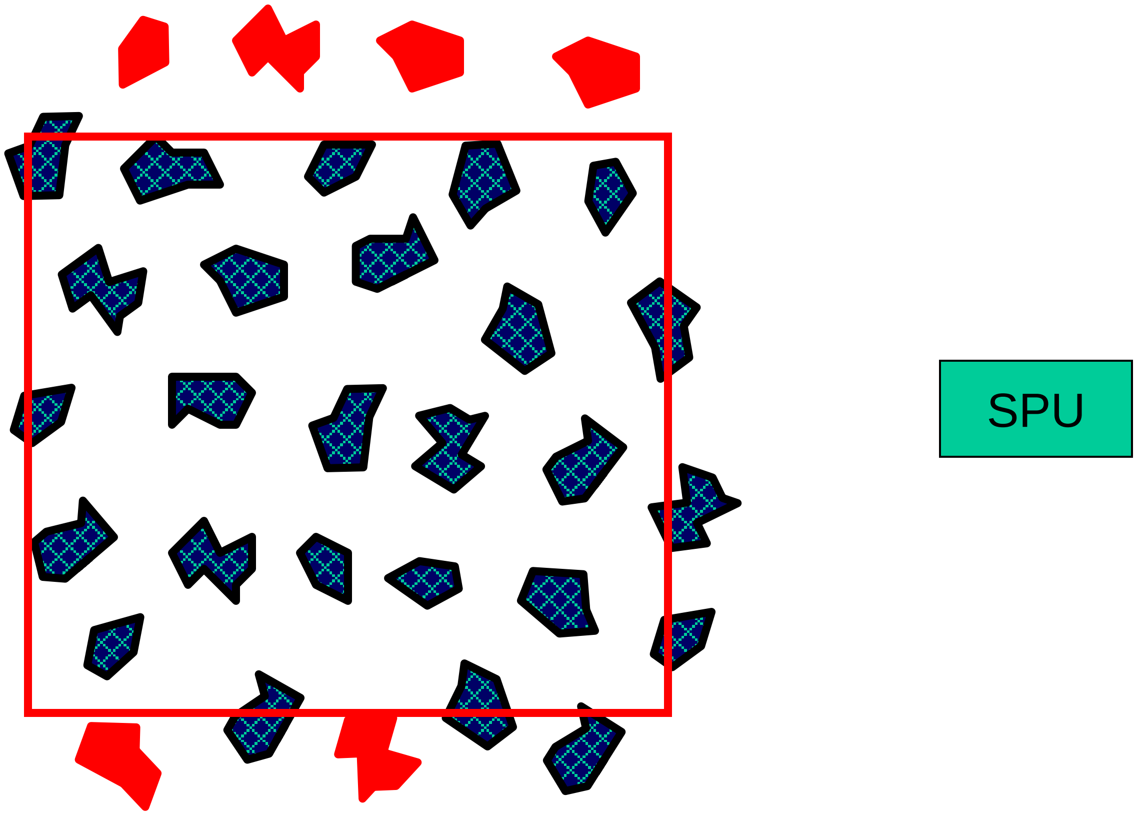


SPU

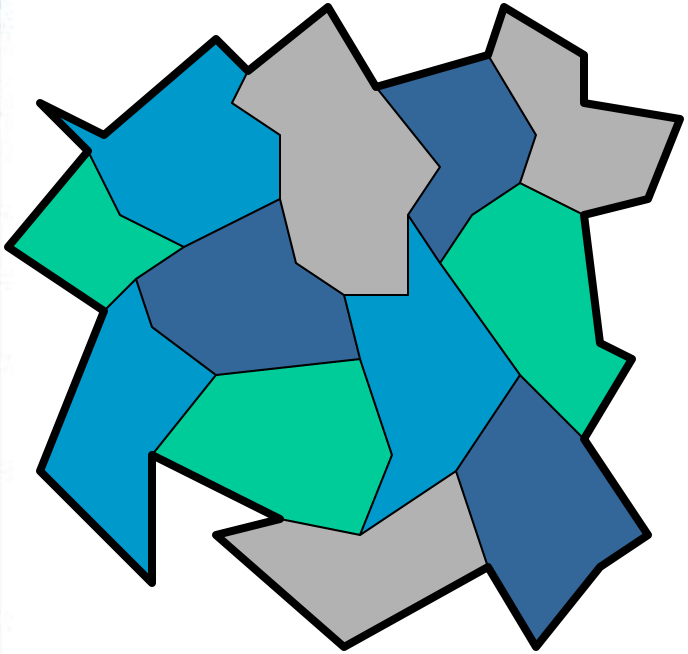
The object needs to be partitioned into smaller pieces, called vertex sets



Culling is much better with smaller pieces too

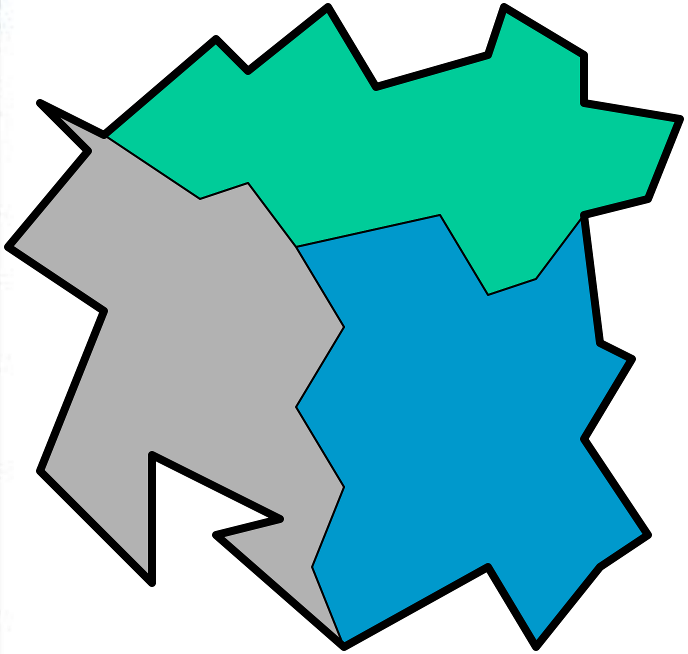


On the PLAYSTATION®2 we used vertex sets with about 64 vertices



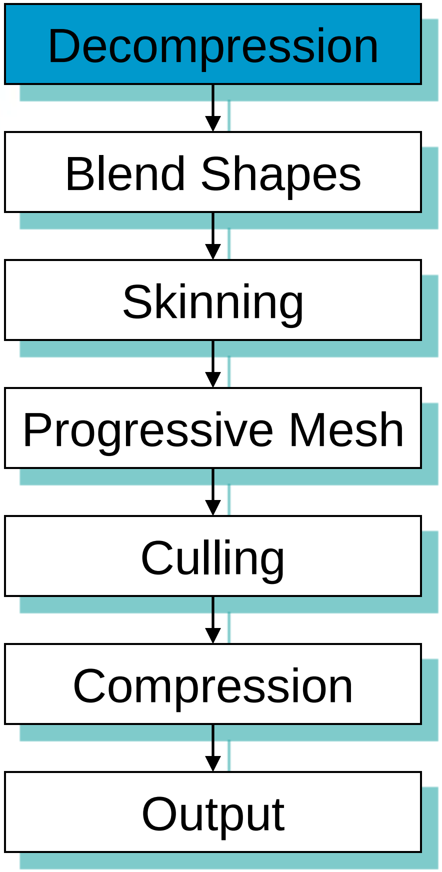
- Many repeated vertices
  - Data increase of about 30%

An SPU can handle vertex sets with between 500-1500 vertices



- Still some repeated vertices
  - Data increase of about 7-10%
- Vertex data is ultimately smaller due to increased compression

Decompression



- SPUs free to use any type of compressed data – not restricted to 8, 16, 32 bit or the like
- Vertex data is decompressed into full floats, as they are easiest for the SPU to use
- Triangle index data can also be decompressed at this time

N-Bit Stream Decompression

- Each vertex attribute is an N-bit stream
  - Each component of that attribute has its own number of bits, integer offset, scale, and bias
- Each component is decompressed with the following equation:
  - **out** = float(**in** + intOffset) * scale + bias
    - Scale and bias need to be constant across an entire object to prevent cracks
    - The number of bits and integer offset need not be

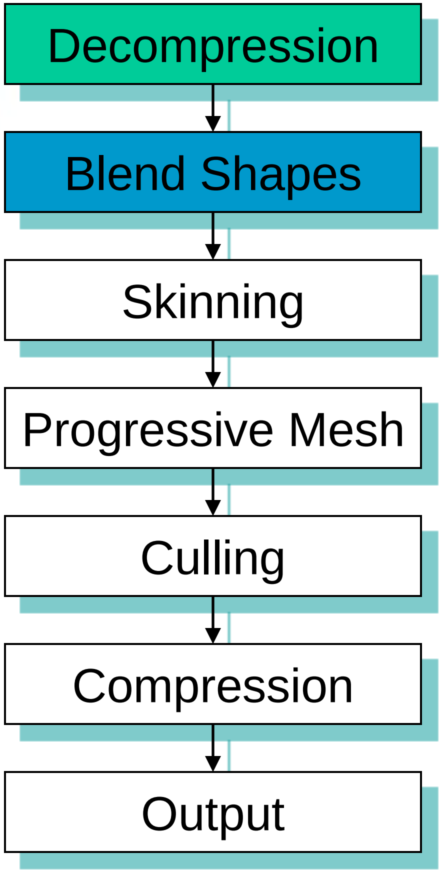
Integer Offset Example

Object

List 1	List 2
1	17
5	14
12	20
0	16
8	13
3	19
14	18
9	

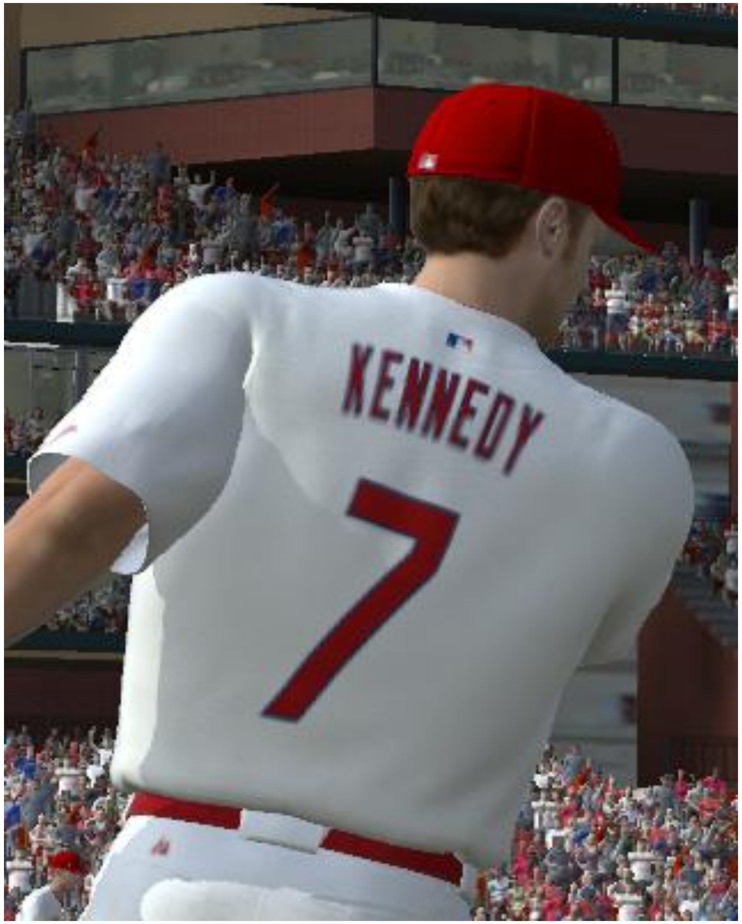
- The total range of this object is 21 units
  - Requires 5 bits
- The range of the first list is only 15 units
  - Requires only 4 bits
- The range of the second list is 8 units
  - When intOffset is set to 13, entries in the second list require only 3 bits

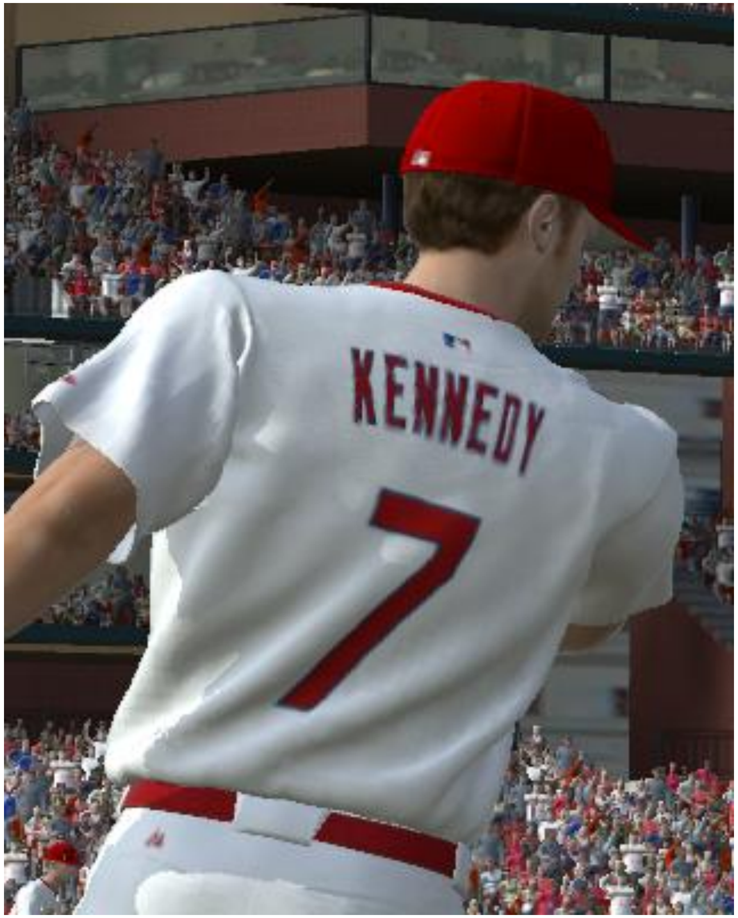
Blend Shapes


Decompression
Blend Shapes
Skinning
Progressive Mesh
Culling
Compression
Output

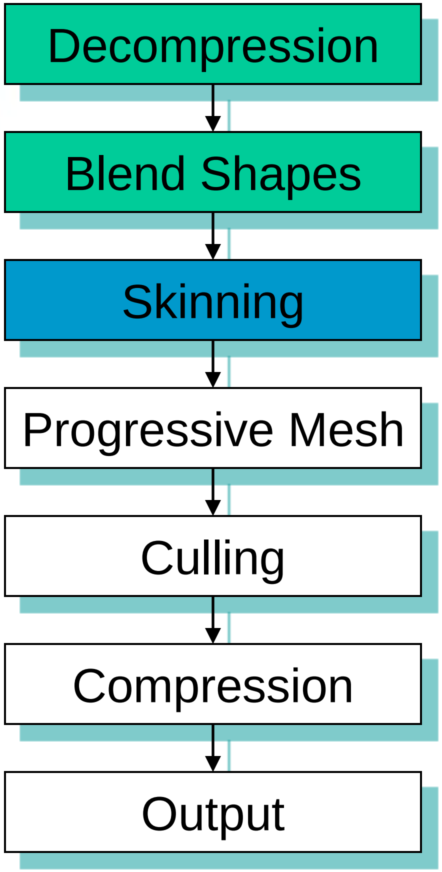
- Not really possible to do on a GPU
- SPU can blend any number of shapes and any number of vertex attributes
- Large data savings
  - Store only deltas
  - Use highly compressed data formats, like N-bit compression
  - Only store data for changing vertices

Blend Shape Use in MLB 07: The Show





Skinning



Decompression

Blend Shapes

Skinning

Progressive Mesh

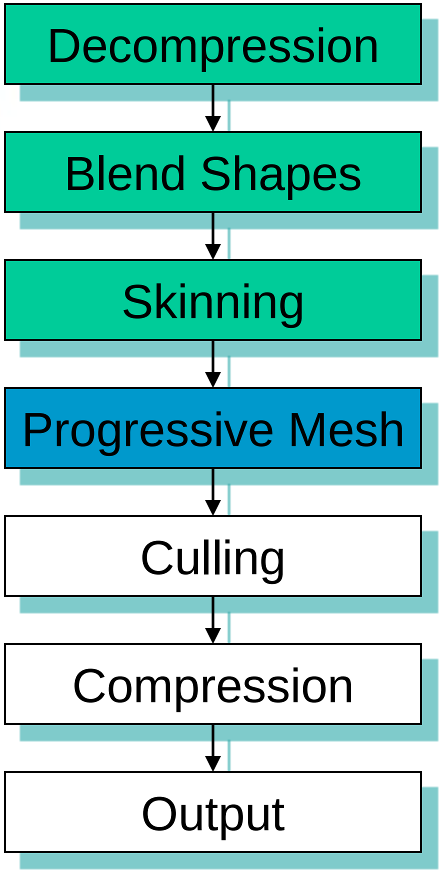
Culling

Compression

Output

- Huge offload of vertex processing from the RSX™
- No need to set large number of vertex program constants with matrix data
- SPU can handle vertices with an arbitrary number of influences
  - Number of influences can vary across the mesh, resulting in data and computational savings

LOD Systems



Decompression

Blend Shapes

Skinning

Progressive Mesh

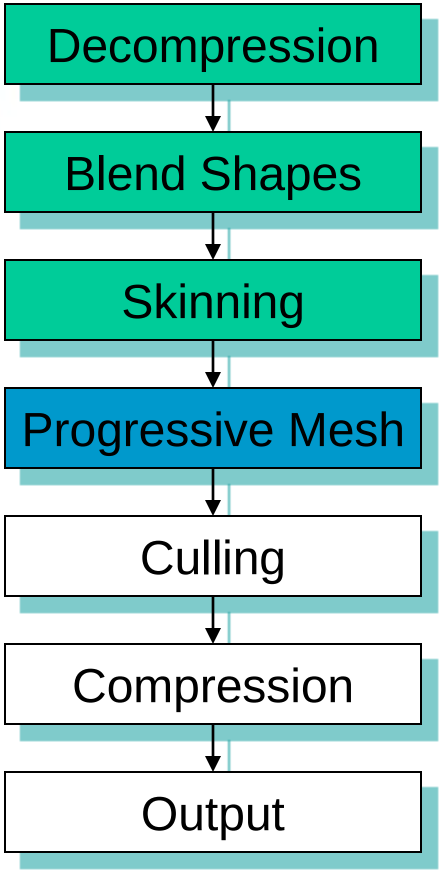
Culling

Compression

Output

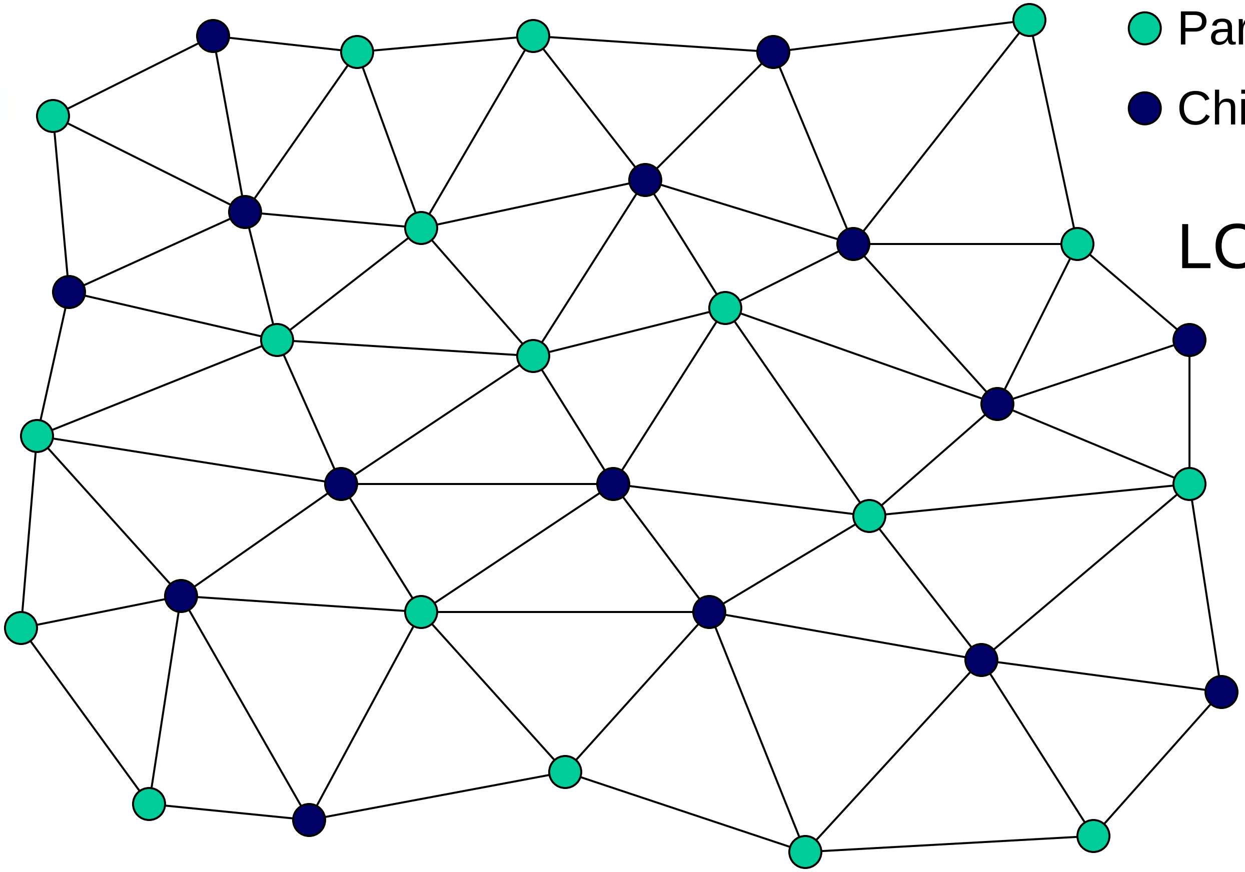
- Reduces processing time as an object moves into the distance
- Many LOD systems are not a good match for a GPU
  - Often operate upon an entire mesh at a time
- Good match for the SPUs

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
- An entire object is processed at once by the tools to avoid cracks between vertex sets

At an LOD there are two types of vertices

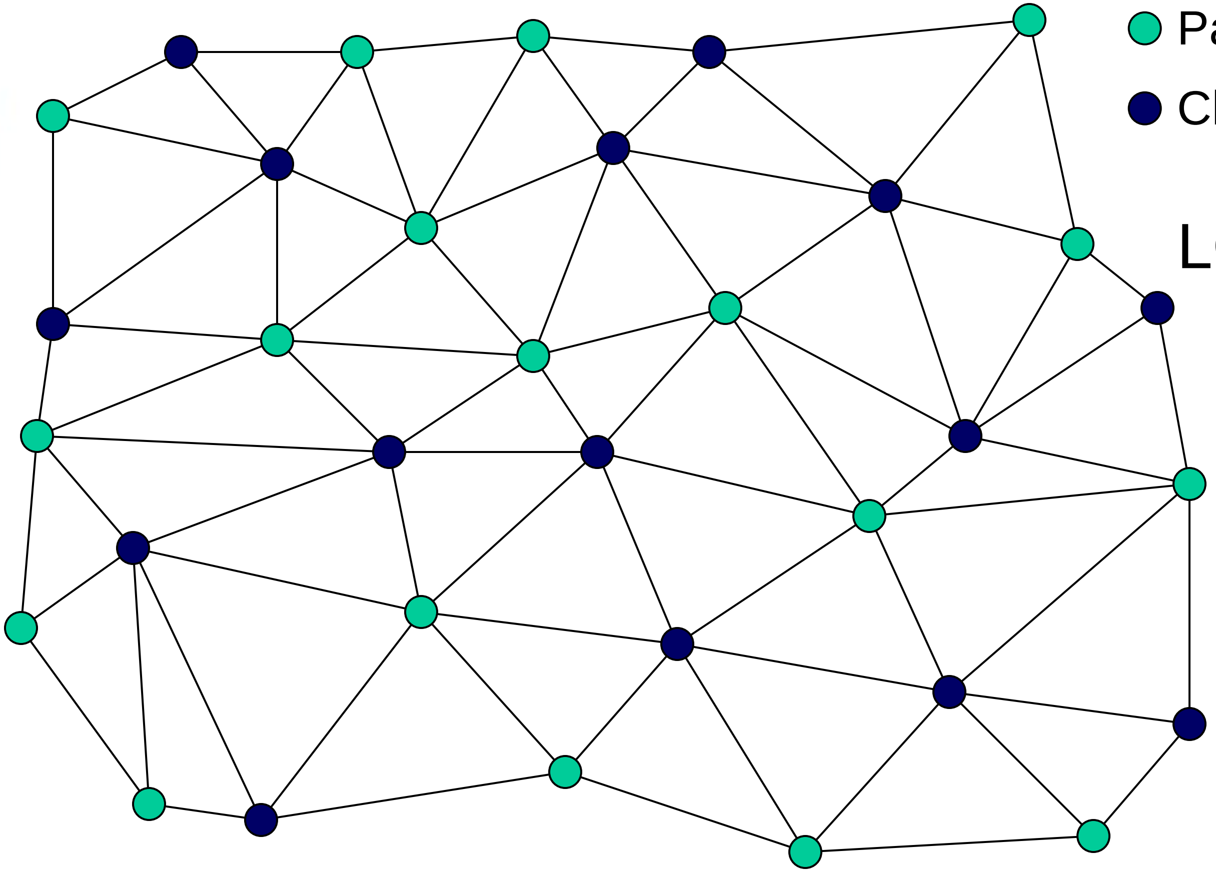


🟢 Parent Vertex

● Child Vertex

LOD = 0.0

As the LOD level decreases, the children “slide” towards their parents

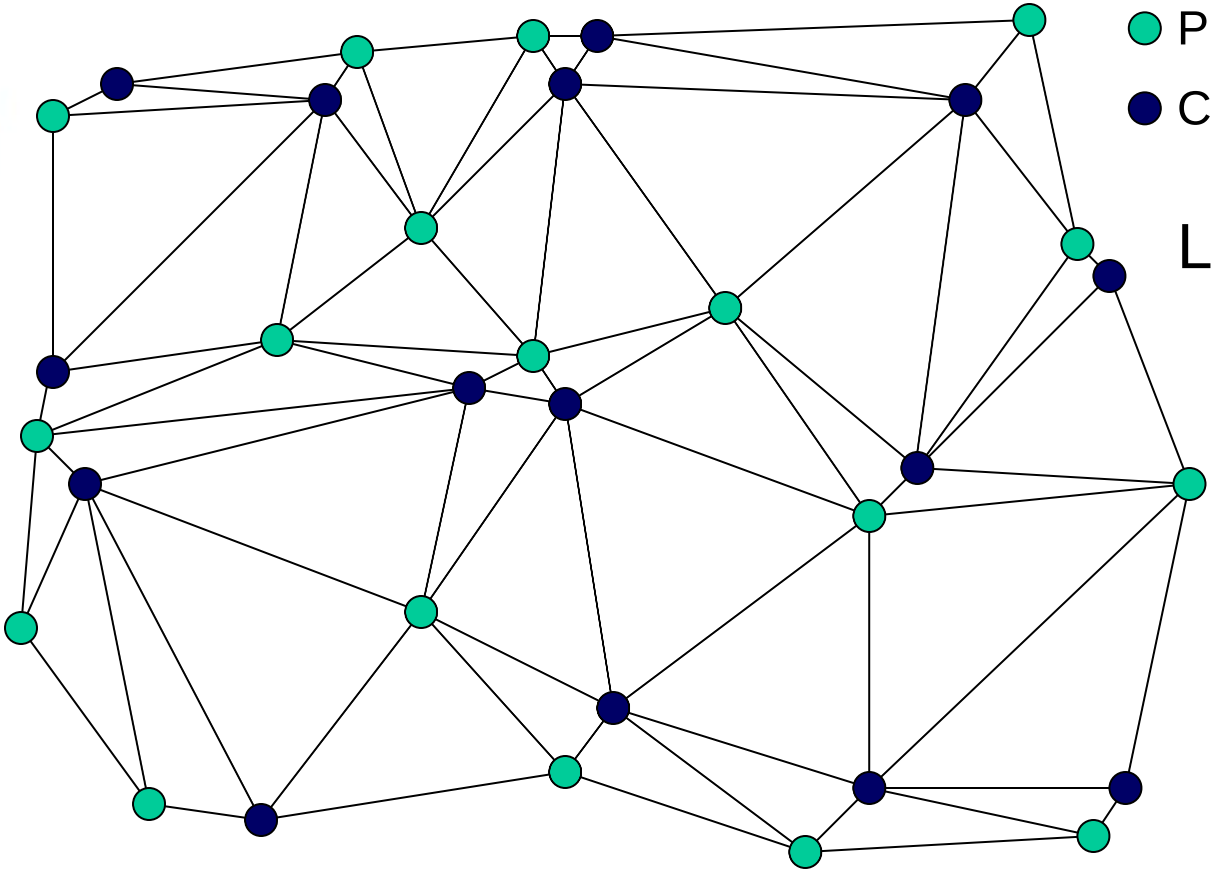


● Parent Vertex

● Child Vertex

LOD = 0.2

The children continue to move towards their parents

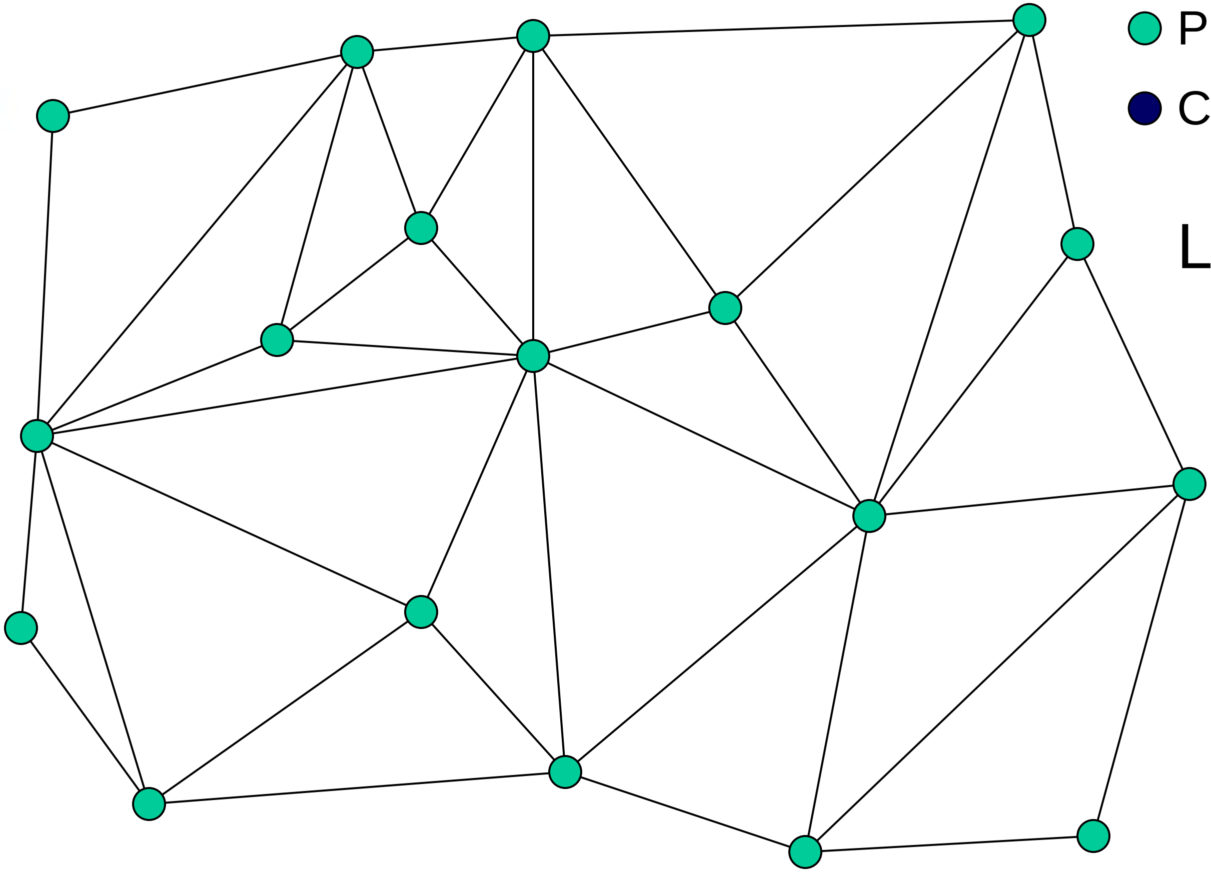


🟢 Parent Vertex

● Child Vertex

LOD = 0.7

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

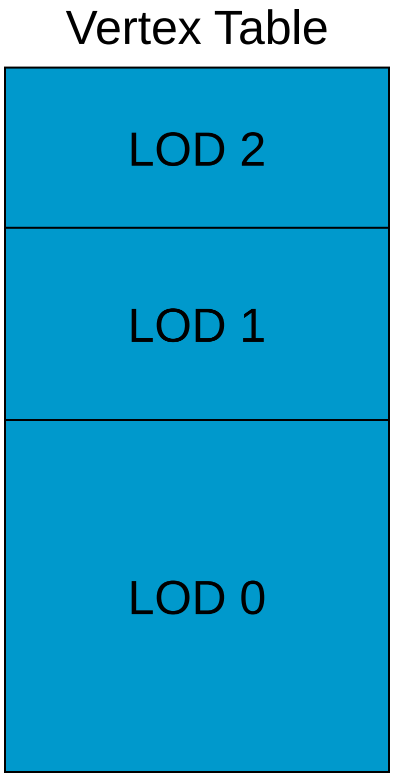


🟢 Parent Vertex

● Child Vertex

LOD = 1.0

Vertices are arranged from lowest LOD to highest LOD



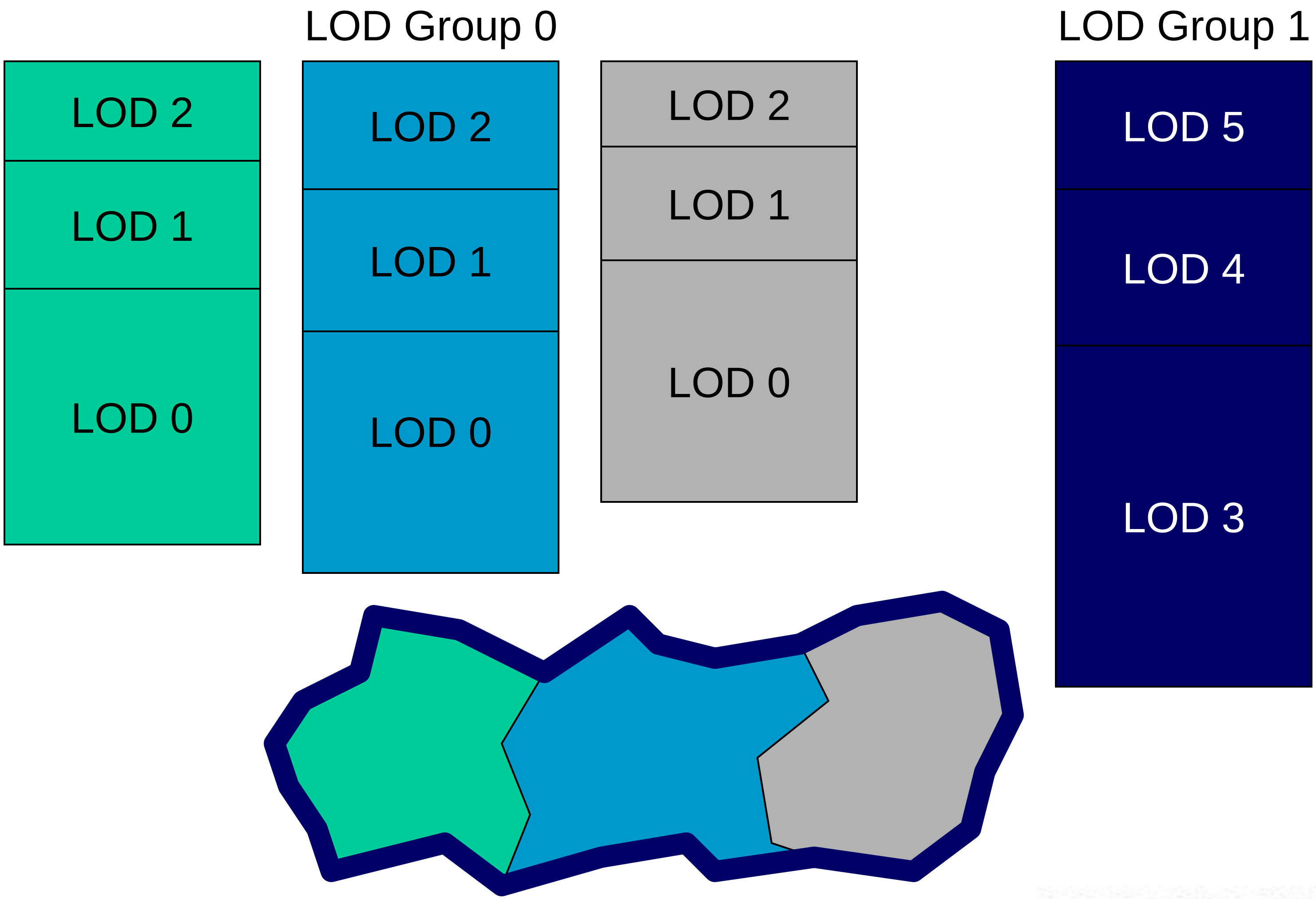
- At LOD 0 all vertices are needed.
- At LOD 1, child vertices from LOD 0 are no longer needed
- At LOD 2, child vertices from LOD 1 are also removed
- This saves bandwidth to the SPUs

Every LOD has its own index table of triangles and parent table

Indexes for LOD 0	Parent Table for LOD 0
Indexes for LOD 1	Parent Table for LOD 1
Indexes for LOD 2	Parent Table for LOD 2

- The parent table contains an index to the parent for every child vertex

LODs are arranged in LOD groups to avoid small vertex sets



LOD Group 0

LOD 2
LOD 1
LOD 0

LOD 2
LOD 1
LOD 0

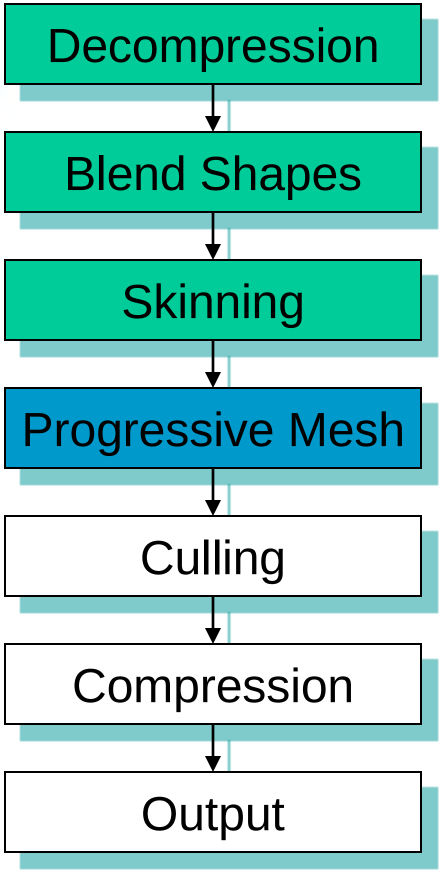
LOD 2
LOD 1
LOD 0

LOD Group 1

LOD 5
LOD 4
LOD 3

MLB 07 THE SHOW™

Continuous Progressive Mesh


Decompression

Blend Shapes

Skinning

Progressive Mesh

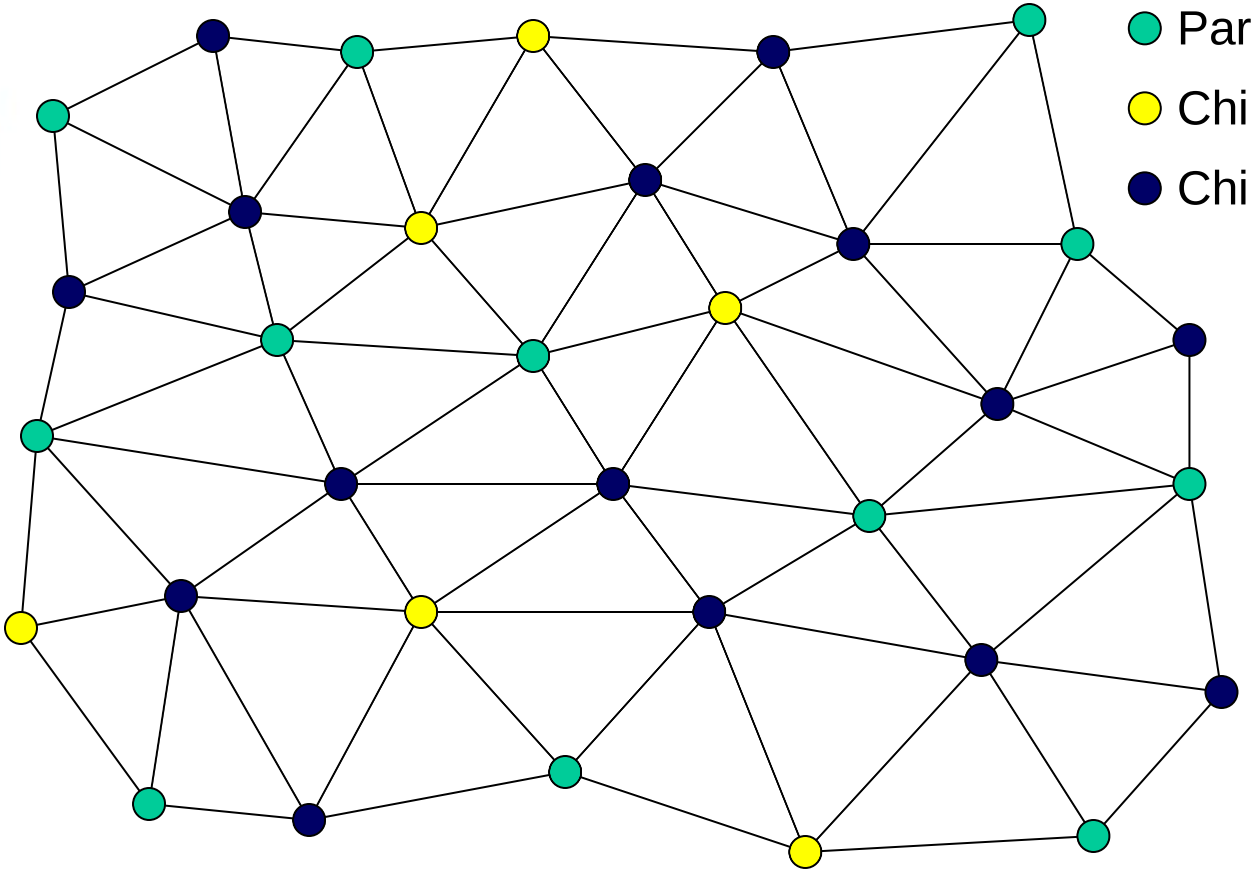
Culling

Compression

Output

- Like discrete progressive mesh, child vertices 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

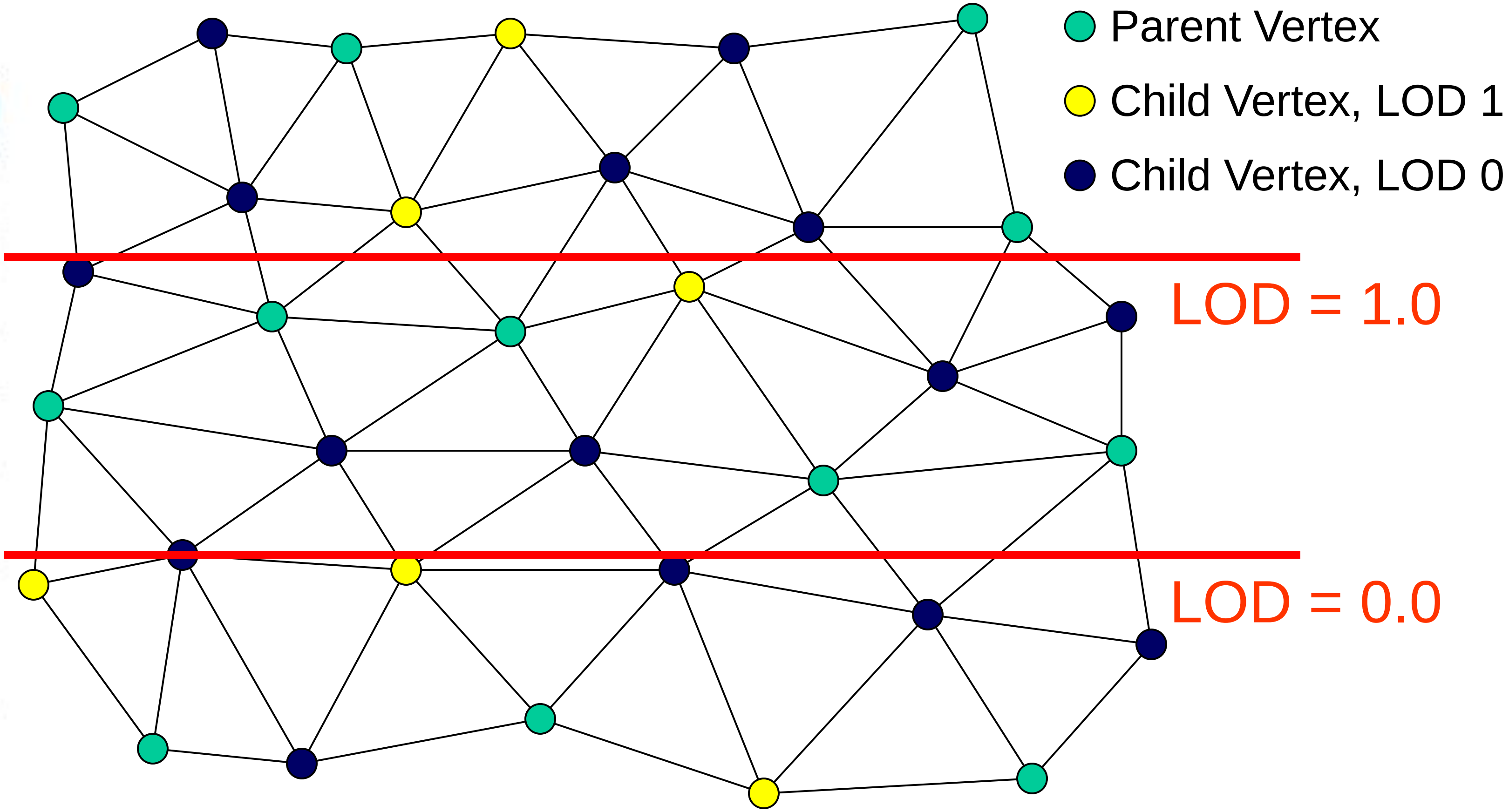


🟢 Parent Vertex

🟡 Child Vertex, LOD 1

● Child Vertex, LOD 0

A single vertex set can straddle several LOD ranges

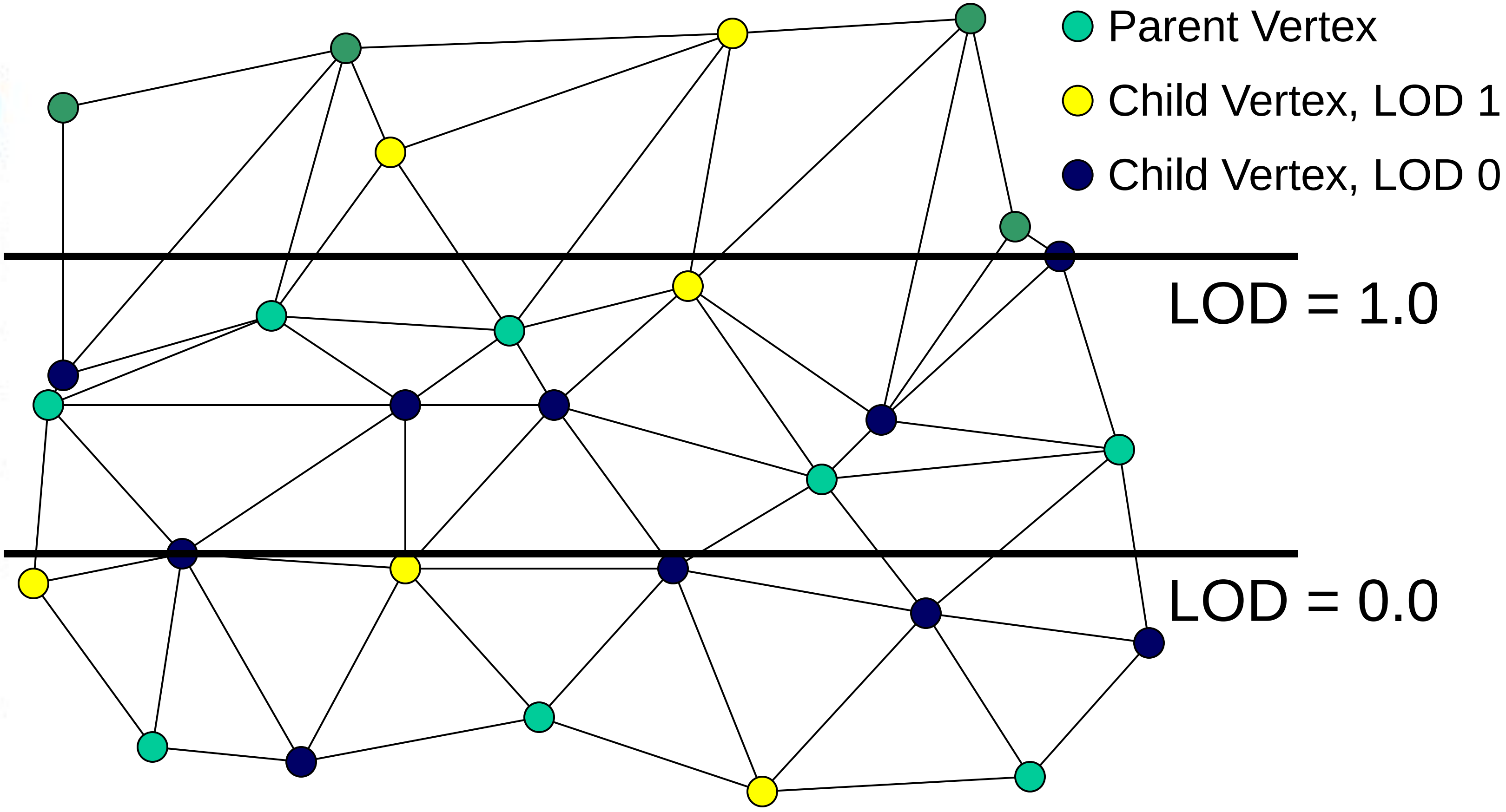


🟢 Parent Vertex

🟡 Child Vertex, LOD 1

● Child Vertex, LOD 0

Vertices move depending on their distance



🟢 Parent Vertex

🟡 Child Vertex, LOD 1

● Child Vertex, LOD 0

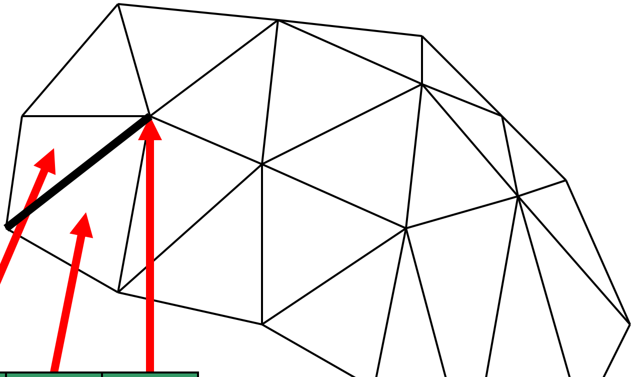
LOD = 1.0

LOD = 0.0

L·A·I·R

FACTOR 5

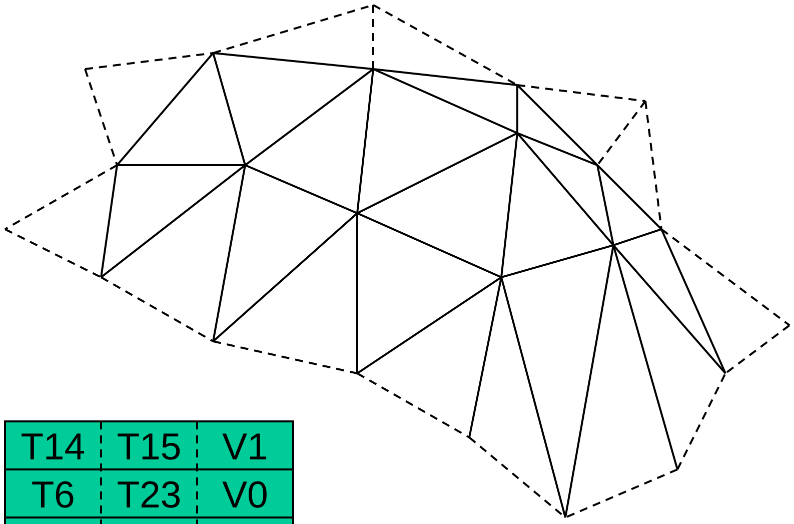
Stencil Shadows



Edge Table

T14	T15	V1
T6	T23	V0
T10	T18	V2
T0	T8	V0
T2	T10	V0
T5	T6	V1
T5	T26	V2
T4	T13	V0
T15	T17	V2
T11	T25	V1

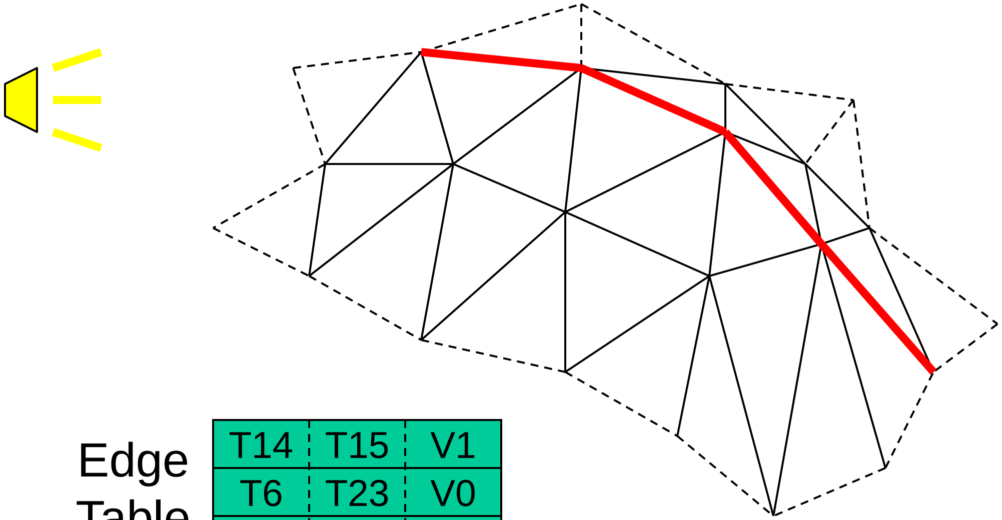
Also need adjoining triangles and vertices from neighboring vertex sets



Edge Table

T14	T15	V1
T6	T23	V0
T10	T18	V2
T0	T8	V0
T2	T10	V0
T5	T6	V1
T5	T26	V2
T4	T13	V0
T15	T17	V2
T11	T25	V1

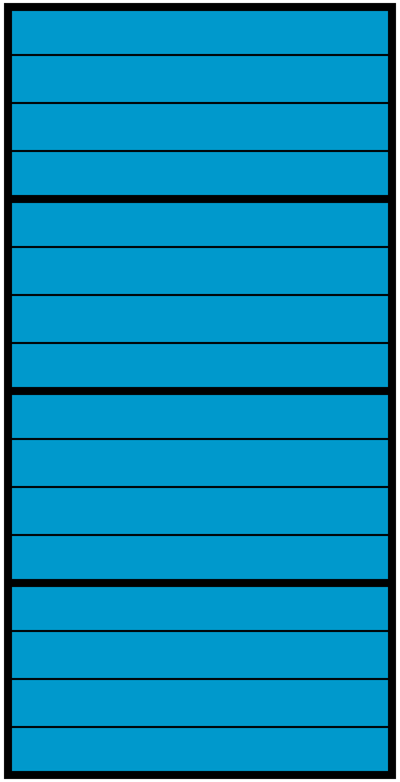
Find the profile edges and generate a new vertex table of extruded edges



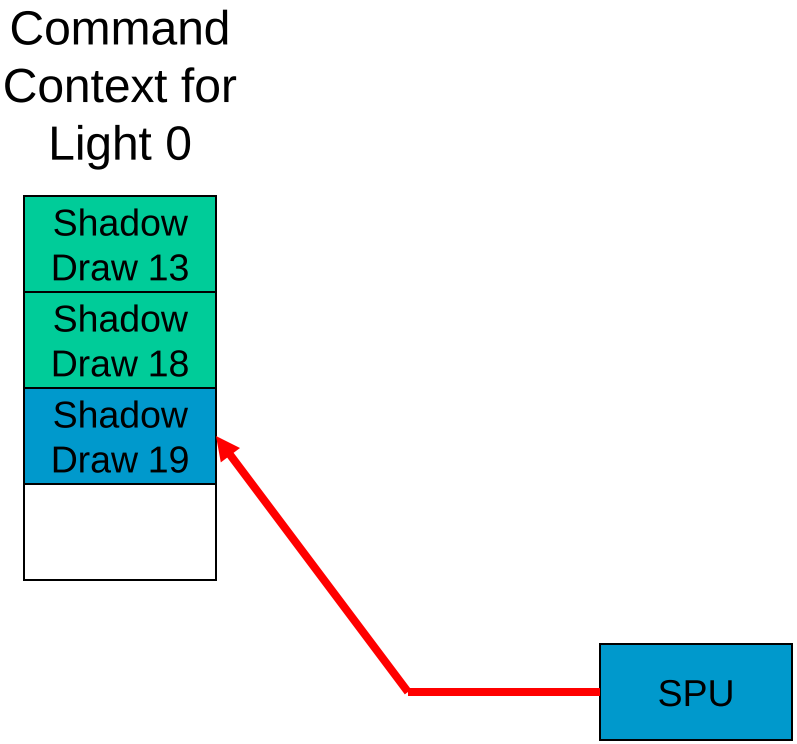
Edge Table

T14	T15	V1
T6	T23	V0
T10	T18	V2
T0	T8	V0
T2	T10	V0
T5	T6	V1
T5	T26	V2
T4	T13	V0
T15	T17	V2
T11	T25	V1

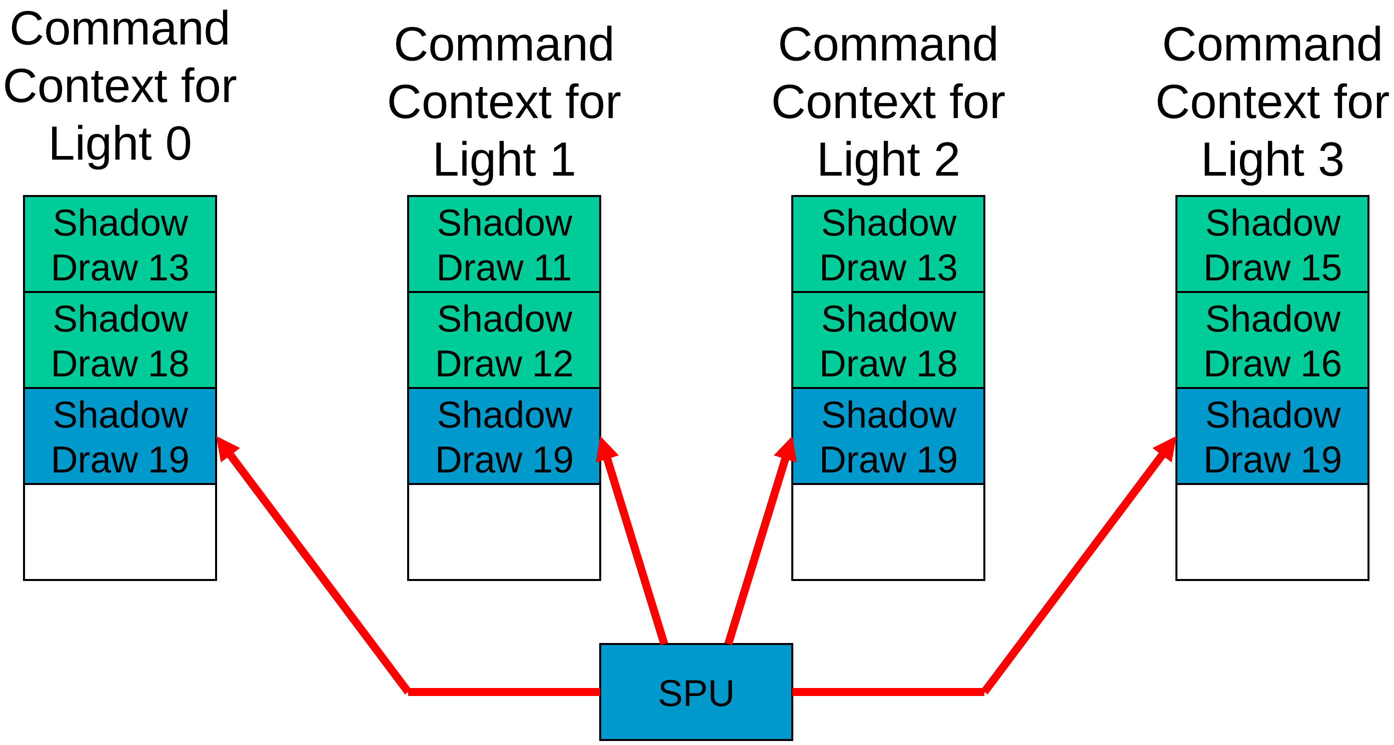
Extruded Edges



Output the new vertex data and draw commands to a shadow context



May as well do multiple lights at the same time



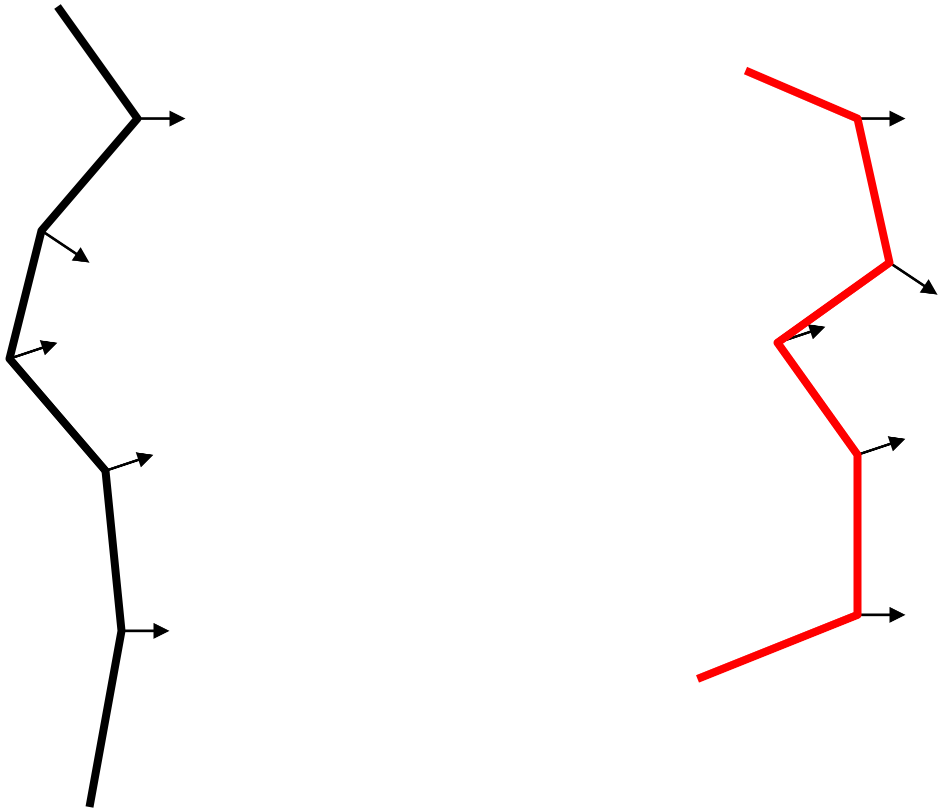
Command Context for Light 0	Command Context for Light 1	Command Context for Light 2	Command Context for Light 3
Shadow Draw 13	Shadow Draw 11	Shadow Draw 13	Shadow Draw 15
Shadow Draw 18	Shadow Draw 12	Shadow Draw 18	Shadow Draw 16
Shadow Draw 19	Shadow Draw 19	Shadow Draw 19	Shadow Draw 19

SPU

Normal and Tangent Calculation

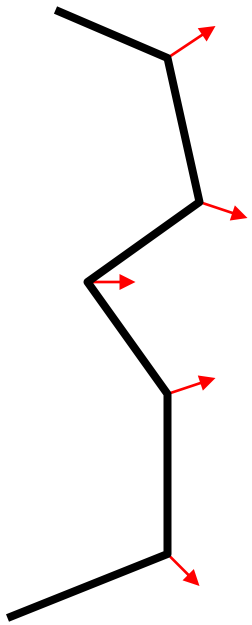
- Typically having normals and tangents included in the vertex data is a good thing
- However, some operations can move the positions so much that the included normals and tangents are no longer correct
- Solution: Recalculate the normals and tangents on the SPU!

Blend shapes can move the positions quite a lot!

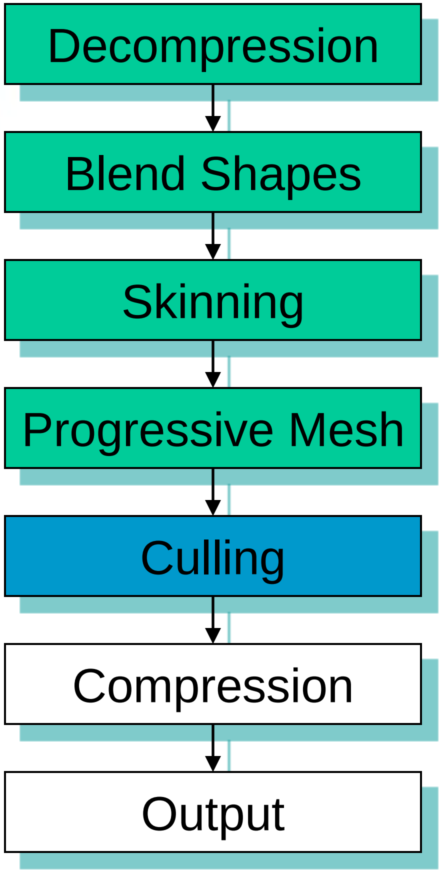


Recalculate the normals!

- Like stencil shadows, calculating normals and tangents requires information about adjoining triangles and vertices from neighboring vertex sets
- Only worth the cost in limited situations

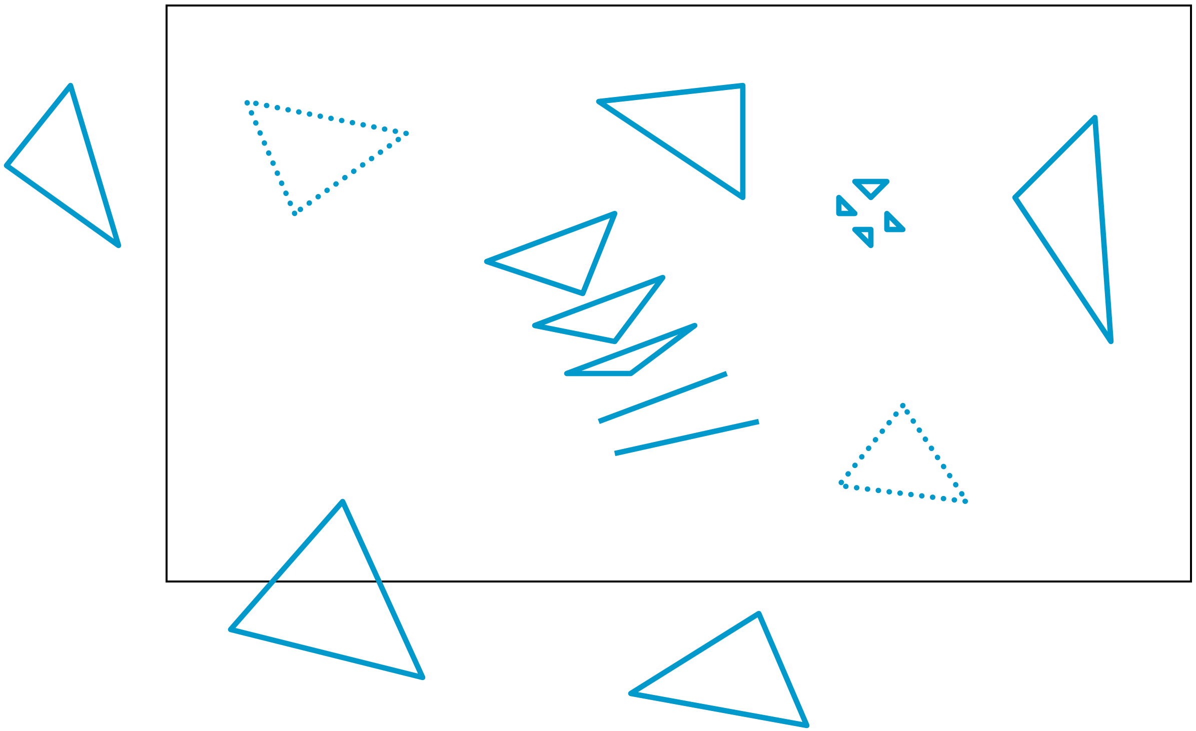


Triangle Culling

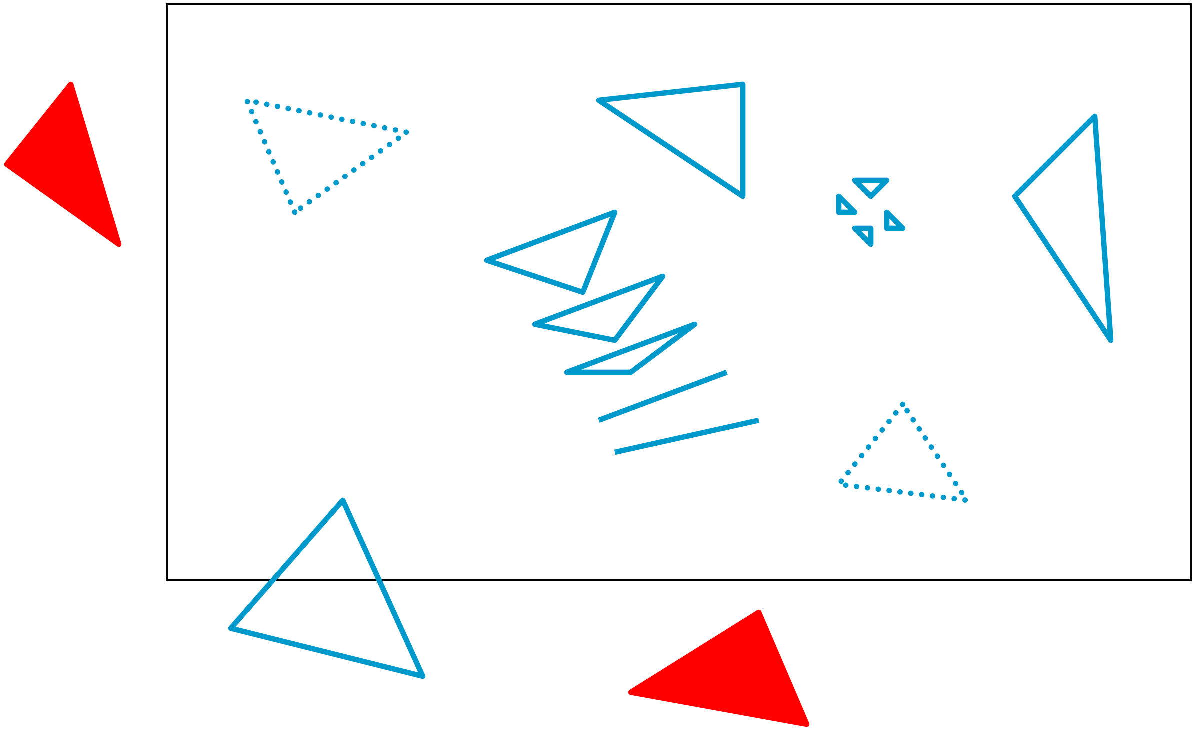


- Many triangles in a scene will ultimately have no renderable area
- Culling these triangles on the SPU removes the burden of the RSX™ processing triangles which do not contribute to the final image
  - This leaves the RSX™ with more time to process relevant triangles

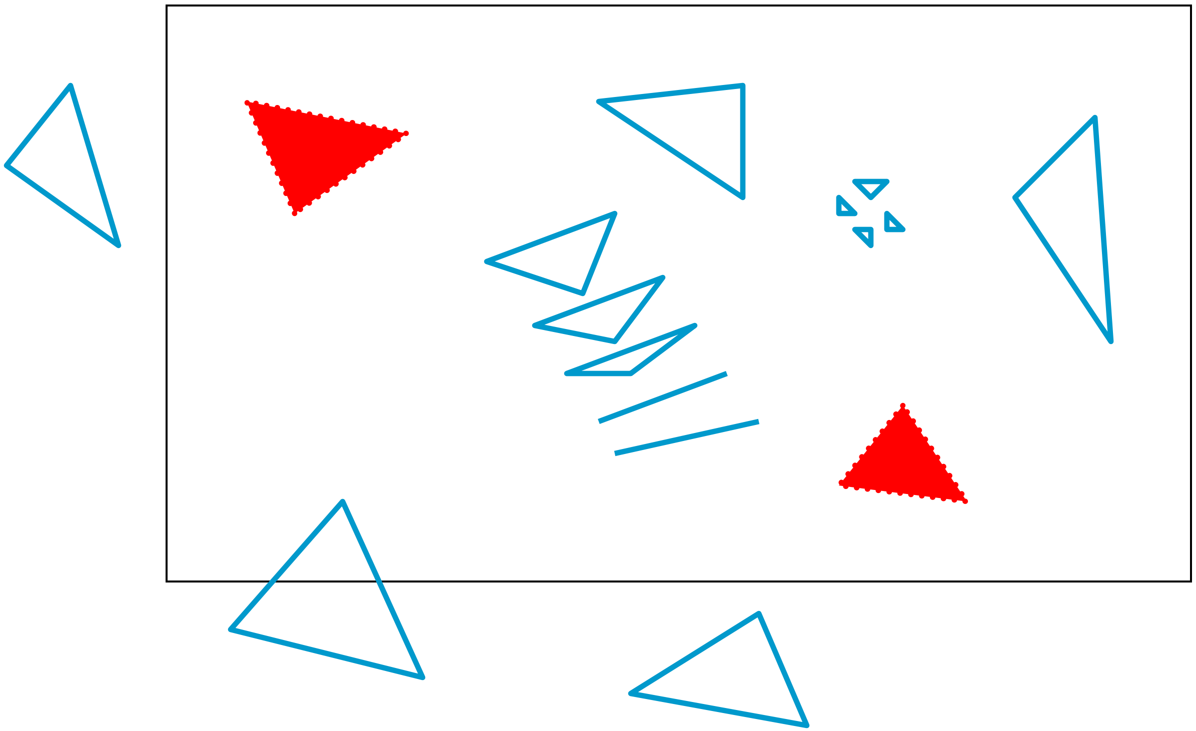
What types of triangles can be culled?



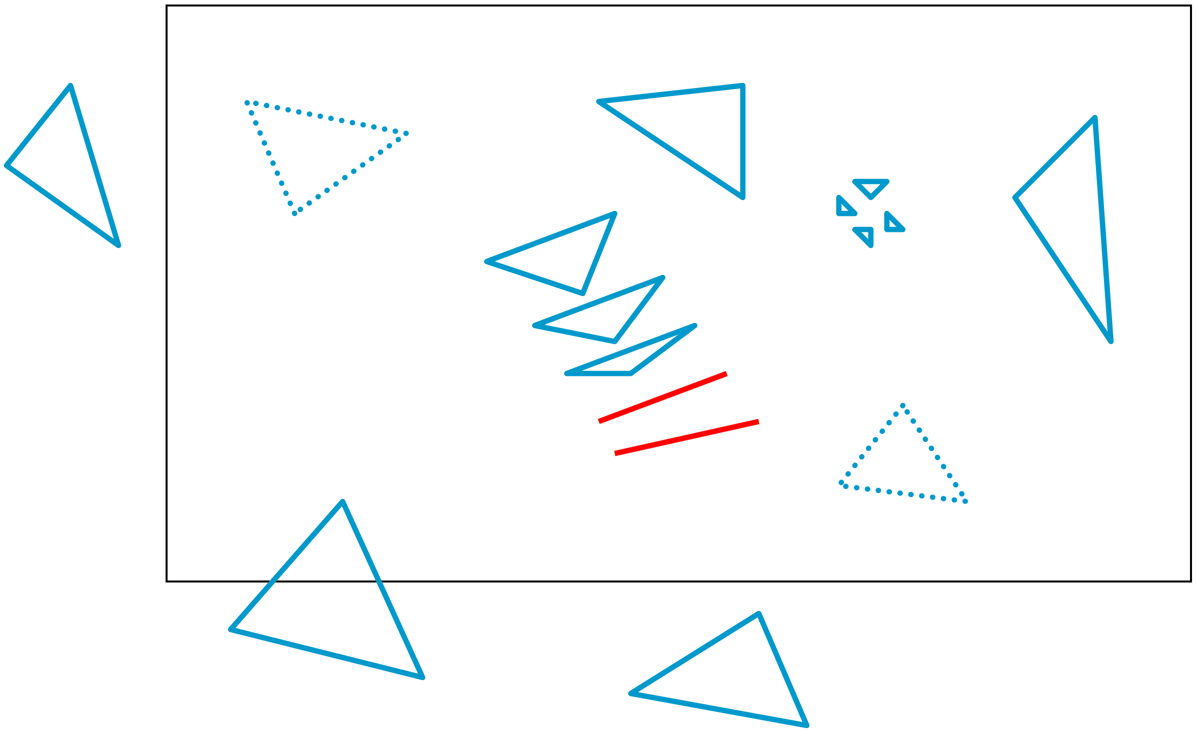
Off screen triangles can be culled



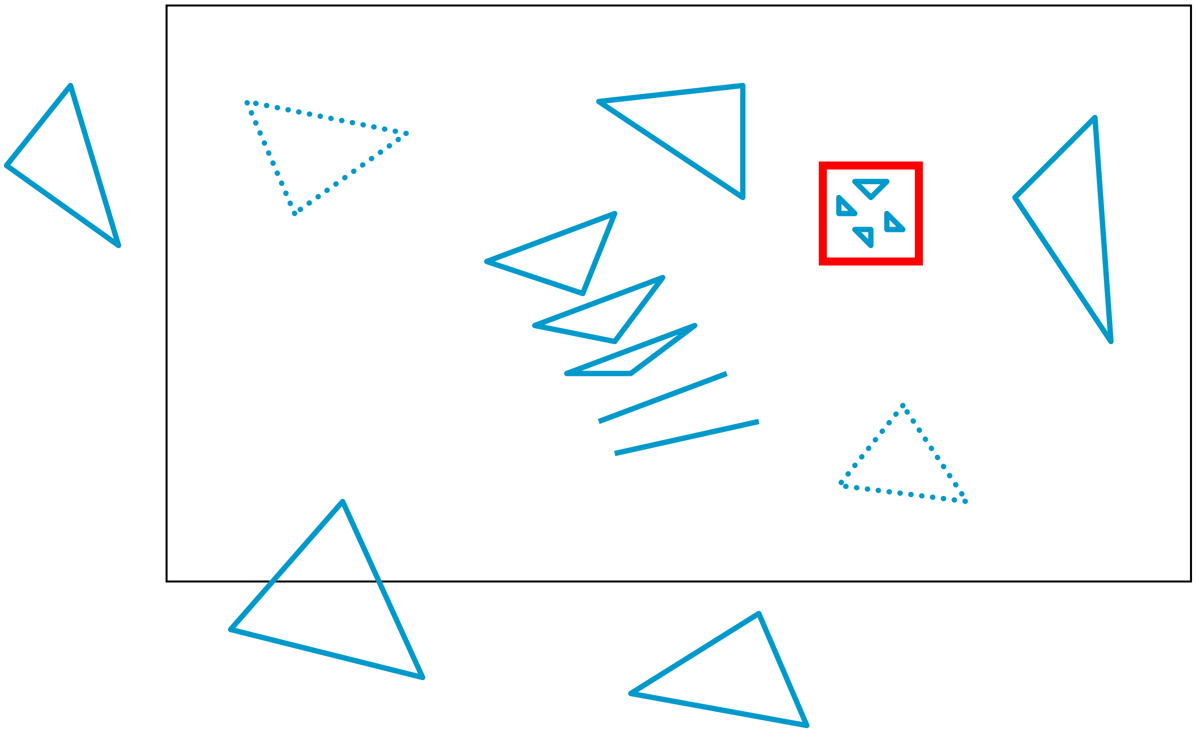
Back facing triangles can be culled, but be sure to use error bars



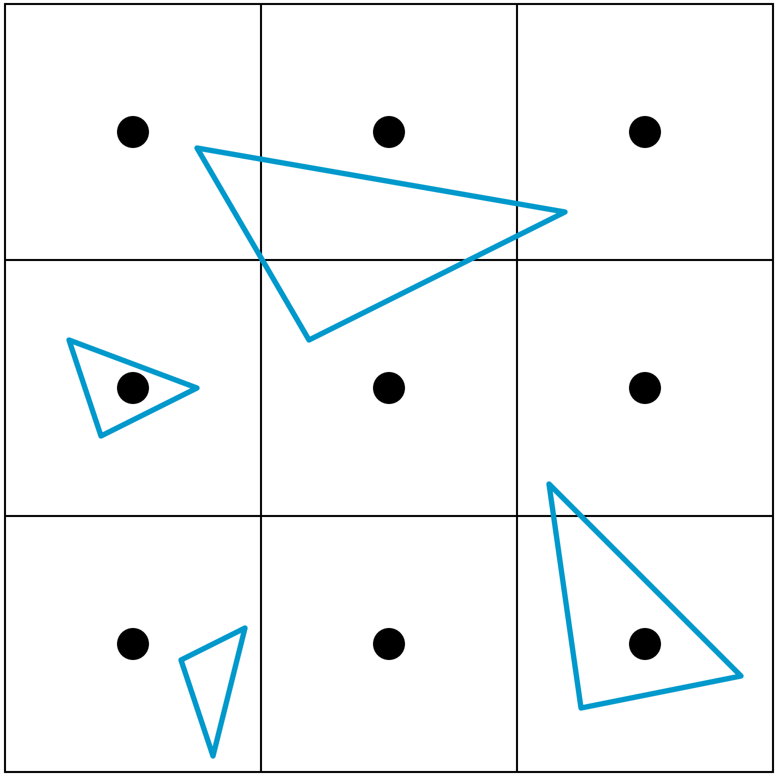
Degenerate triangles can be culled



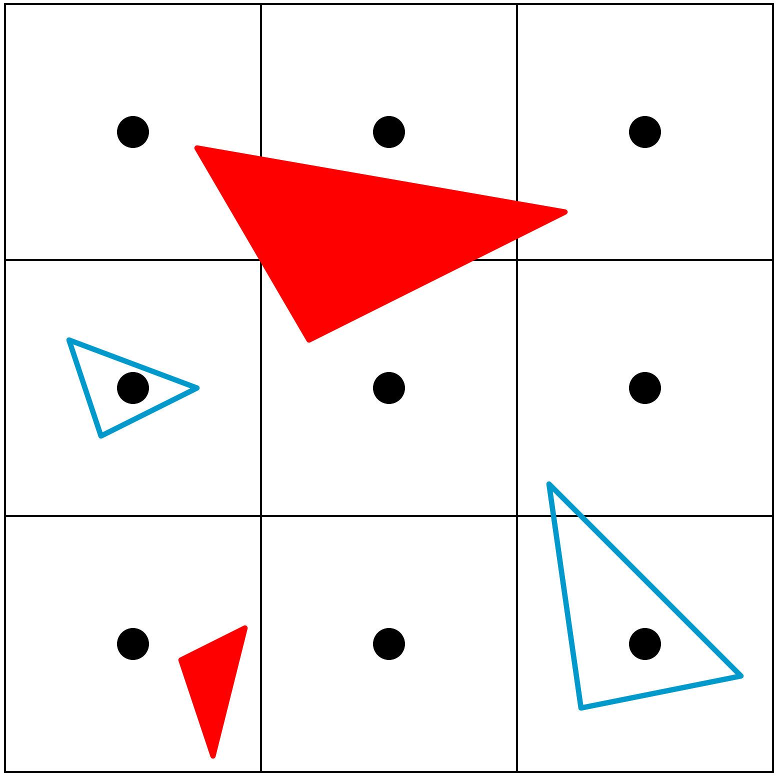
Some triangles are very small



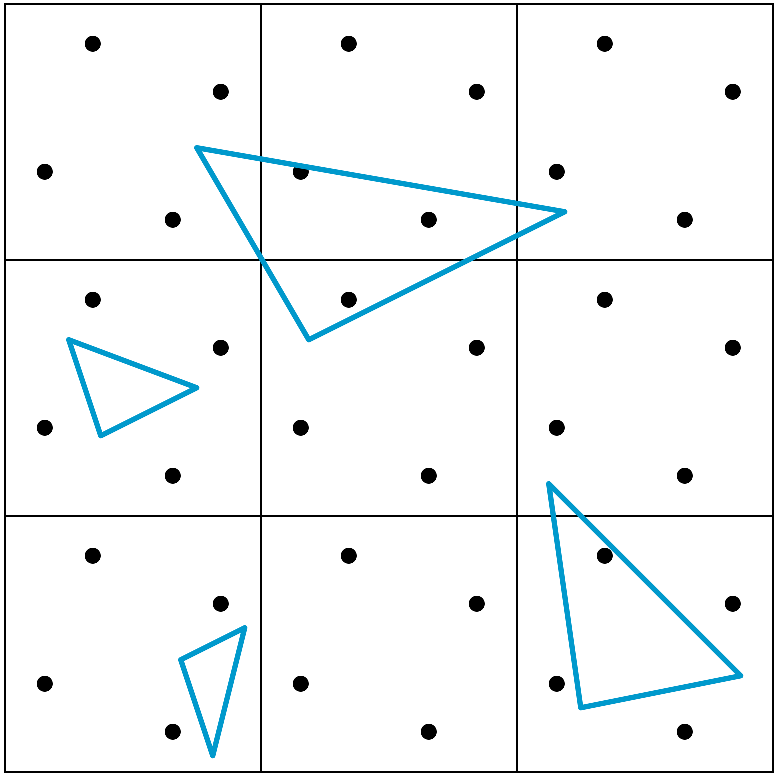
Some triangles are so small that they do not cover a pixel center



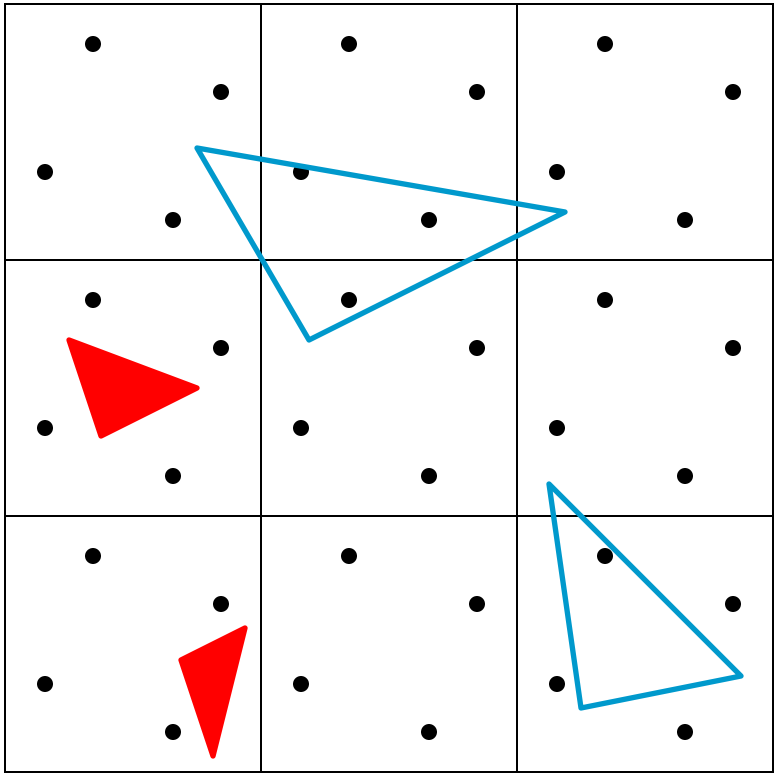
These triangles can be culled



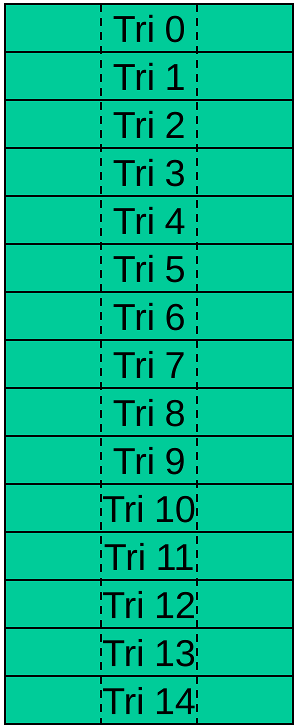
Multisampling adds some complications…



But these triangles can still be culled



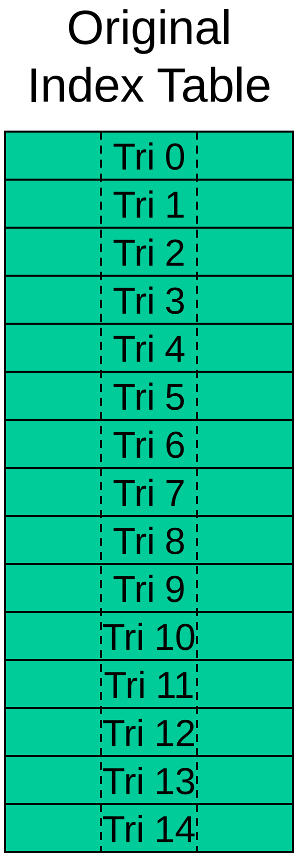
The SPU starts with the input triangle index table



Original Index Table

Tri 0
Tri 1
Tri 2
Tri 3
Tri 4
Tri 5
Tri 6
Tri 7
Tri 8
Tri 9
Tri 10
Tri 11
Tri 12
Tri 13
Tri 14

The culling algorithm determines which triangles are to be kept



Original Index Table

Tri 0
Tri 1
Tri 2
Tri 3
Tri 4
Tri 5
Tri 6
Tri 7
Tri 8
Tri 9
Tri 10
Tri 11
Tri 12
Tri 13
Tri 14

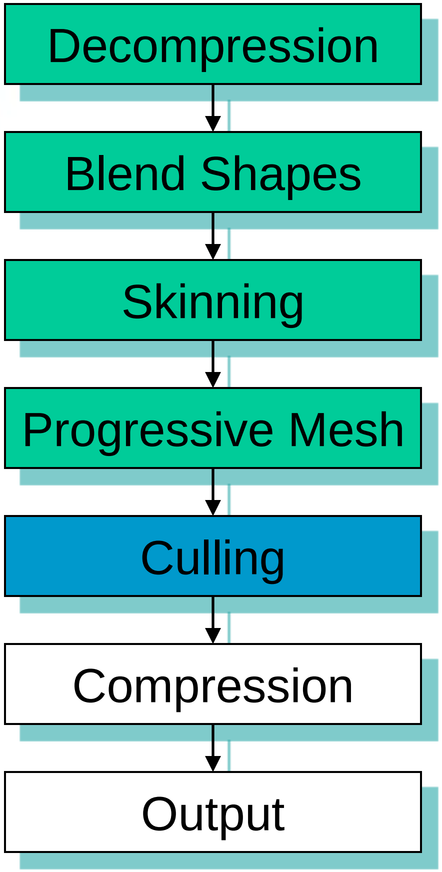
And a new index table is created from these triangles

Original Index Table
Tri 0
Tri 1
Tri 2
Tri 3
Tri 4
Tri 5
Tri 6
Tri 7
Tri 8
Tri 9
Tri 10
Tri 11
Tri 12
Tri 13
Tri 14

Culled Index Table
Tri 1
Tri 4
Tri 6
Tri 7
Tri 11
Tri 14

Culling can remove 60-70% of all triangles!

Vertex Culling



- After triangles are culled, some vertices are no longer used in any triangle
- These vertices can be removed from the vertex table
  - This is done by first building a vertex renaming table which contains the new vertex index for each vertex

Start with an empty vertex renaming table

Index Table		
0	2	5
1	4	10
1	8	13
2	5	7
5	7	8
7	8	9
8	10	13
10	13	14

Vertex Table	Renaming Table
0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	-1
8	-1
9	-1
10	-1
11	-1
12	-1
13	-1
14	-1
15	-1

Add a new index for each used vertex in the index table

Index Table		
0	2	5
1	4	10
1	8	13
2	5	7
5	7	8
7	8	9
8	10	13
10	13	14

Vertex Table
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Renaming Table
0
3
1
-1
4
2
-1
8
6
9
5
-1
-1
7
10
-1

Using the renaming table, build a new vertex table with only used vertices

Index Table		
0	2	5
1	4	10
1	8	13
2	5	7
5	7	8
7	8	9
8	10	13
10	13	14

Vertex Table
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Renaming Table
0
3
1
-1
4
2
-1
8
6
9
5
-1
-1
7
10
-1

New Vertex Table
0
2
5
1
4
10
8
13
7
9
14

Finally, replace the old indices in the index table with the new indices

Index Table

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

Vertex Table

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

Renaming Table

0
3
1
-1
4
2
-1
8
6
9
5
-1
-1
7
10
-1

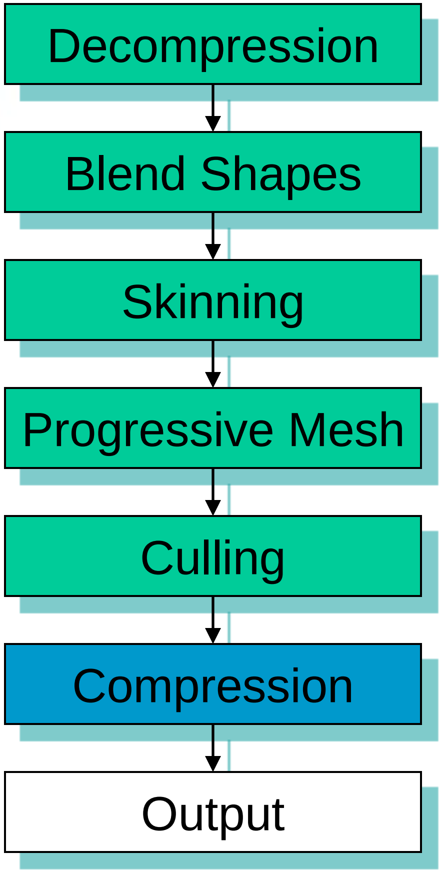
New Vertex Table

0
2
5
1
4
10
8
13
7
9
14

Only minor performance gains on the RSX™, if any

- Removes about 30% of the vertex data
- Better use of the pre-transform cache, but not much else

Vertex Stream Combining



Decompression

Blend Shapes

Skinning

Progressive Mesh

Culling

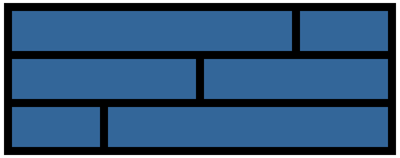
Compression

Output

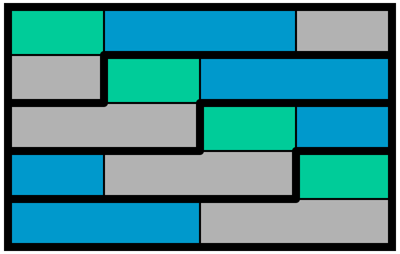
- A vertex stream is an interleaved set of vertex attributes, which is used natively by the RSX™
- Fewer vertex streams results in better performance
- Easy to combine streams while compressing vertex attributes into RSX™ formats

Vertex attributes can be input into the SPUs in multiple streams

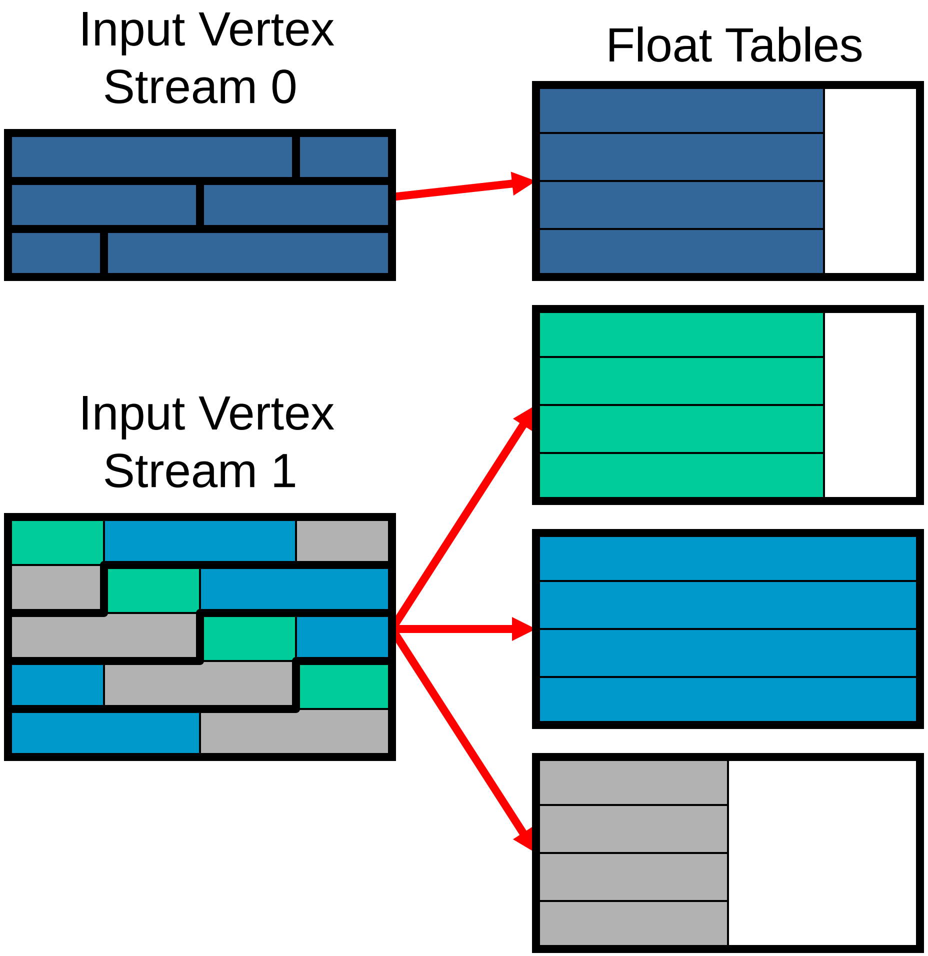
Input Vertex Stream 0



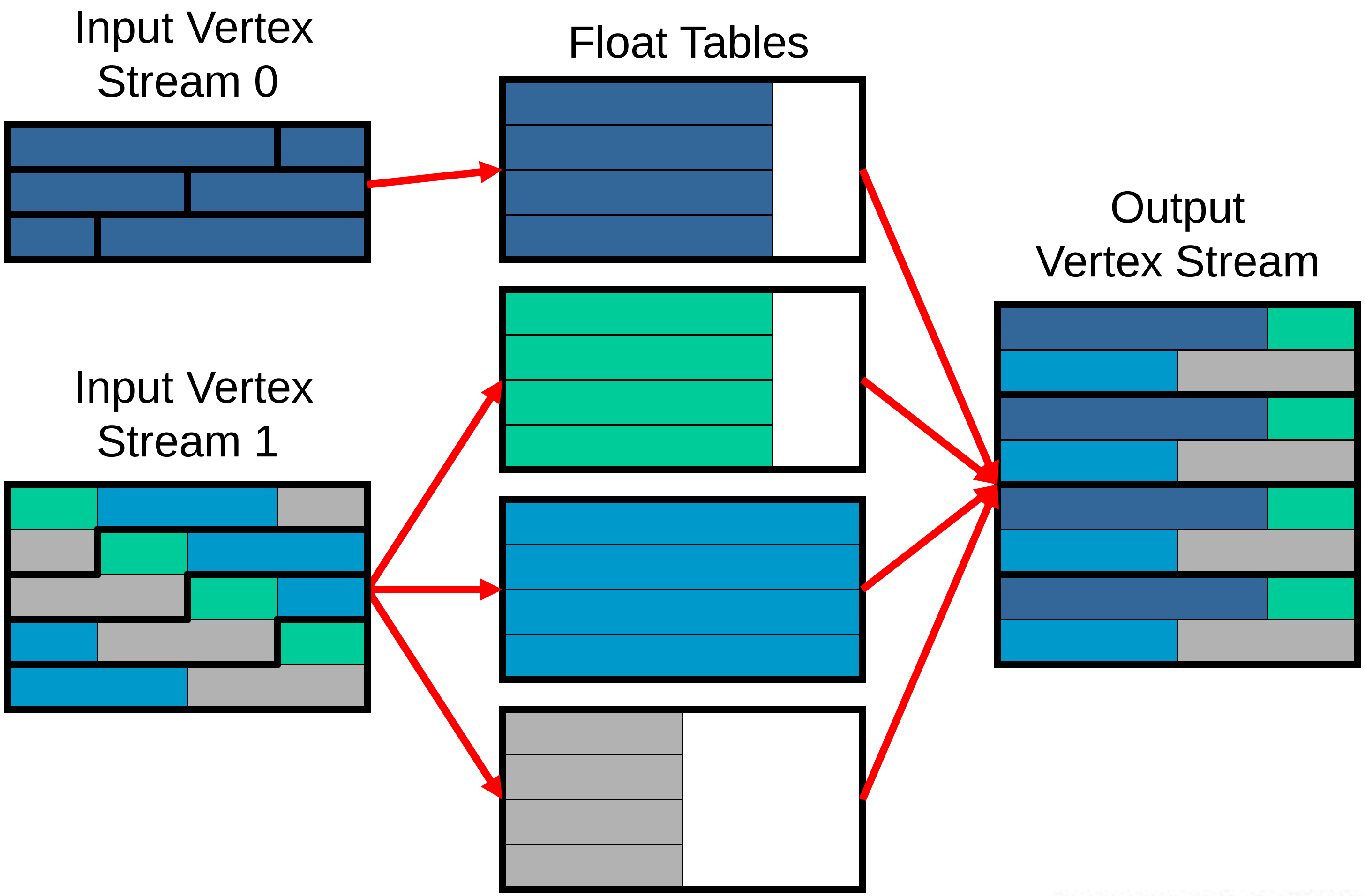
Input Vertex Stream 1



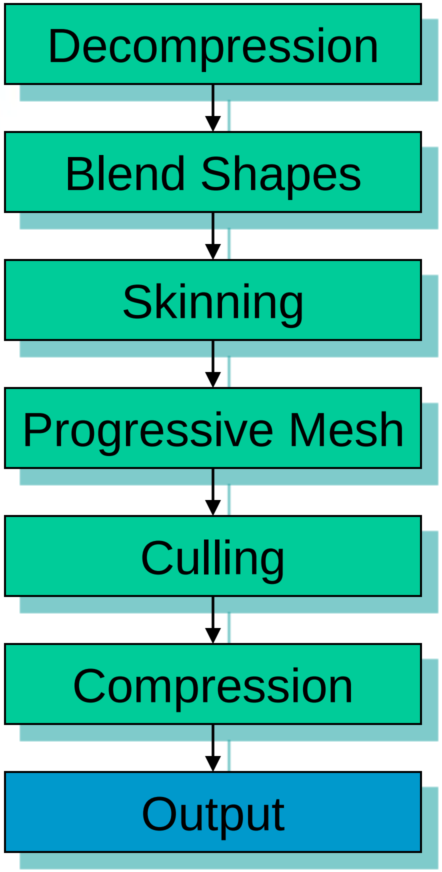
The vertex streams are decompressed into tables of floats



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

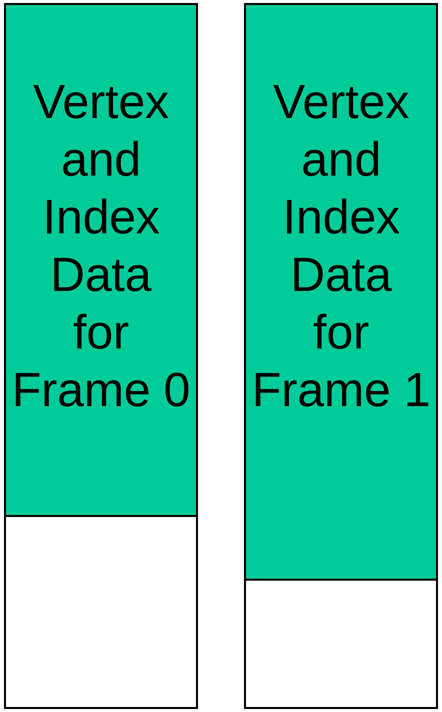


Output Buffering Schemes



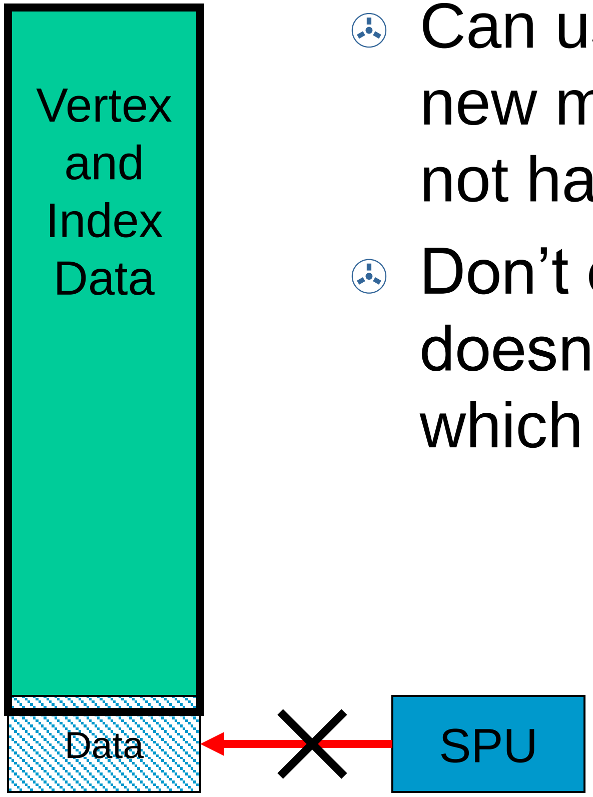
- Vertex and index data constructed by the SPUs is output from SPU local store
- Holes in the command buffer are patched with pointers to the vertex and index data as well as the draw commands

Double Buffer


Vertex and Index Data for Frame 0
Vertex and Index Data for Frame 1

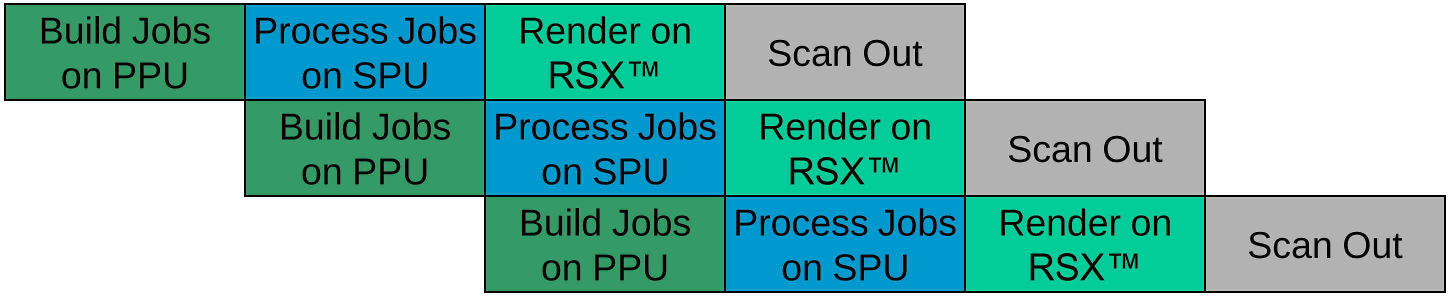
- Each buffer stores vertex and index data for an entire frame
- SPUs atomically access a mutex which is used to allocate memory from a buffer
- Easy synchronization with the RSX™ once a frame
- Uses lots of memory

It is possible to completely fill a buffer



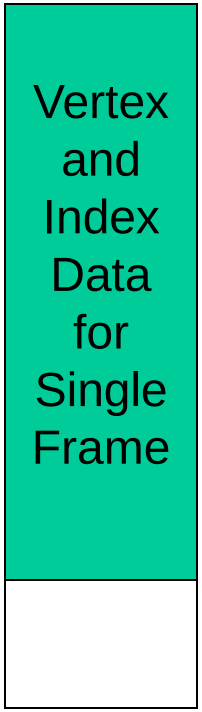
- Can use a callback to allocate new memory (which you may not have)
- Don’t draw geometry that doesn’t fit (difficult to pick which geometry not to draw)

Double buffer requires an extra frame of lag in the rendering pipeline



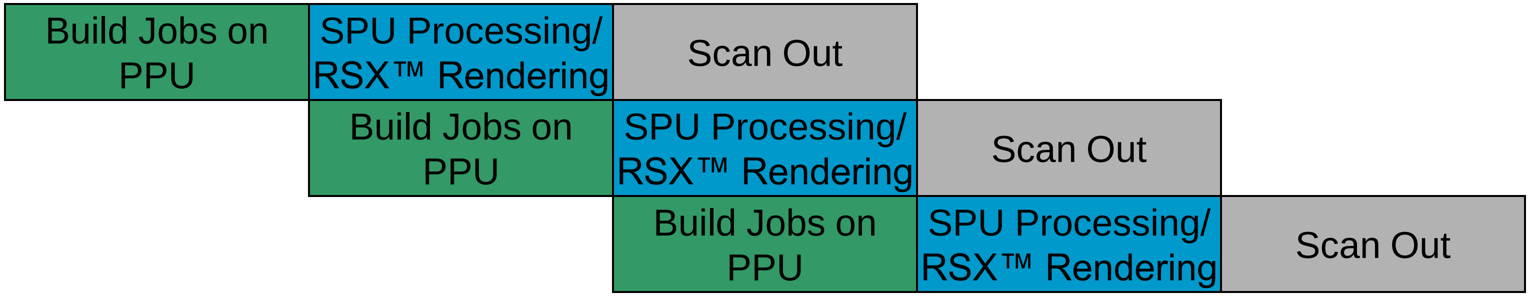
Build Jobs on PPU	Process Jobs on SPU	Render on RSX™	Scan Out			
	Build Jobs on PPU	Process Jobs on SPU	Render on RSX™	Scan Out		
		Build Jobs on PPU	Process Jobs on SPU	Render on RSX™	Scan Out	

Single Buffer



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

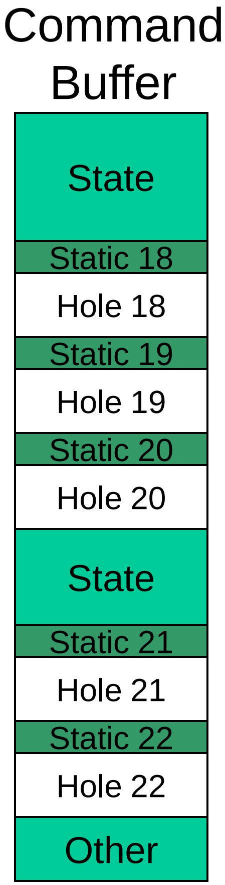
Single buffer uses a shorter rendering pipeline



Build Jobs on PPU	SPU Processing/ RSX™ Rendering	Scan Out		
	Build Jobs on PPU	SPU Processing/ RSX™ Rendering	Scan Out	
		Build Jobs on PPU	SPU Processing/ RSX™ Rendering	Scan Out

- 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

Command Buffer Holes



Command Buffer

State

Static 18

Hole 18

Static 19

Hole 19

Static 20

Hole 20

State

Static 21

Hole 21

Static 22

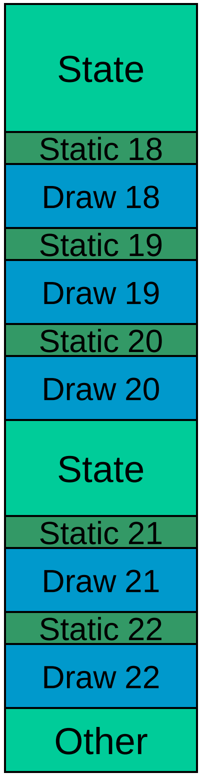
Hole 22

Other

- SPU processing requires some setup by the PPU
  - Some job data is required for each vertex set
  - Static portions of the command buffer are built on the PPU
    - Static vertex attribute pointers
    - Index table pointer and draw commands when not performing triangle culling on the SPU
  - “Holes” are left for the dynamic portion built by the SPU

Filling the Holes

Command Buffer



State

Static 18

Draw 18

Static 19

Draw 19

Static 20

Draw 20

State

Static 21

Draw 21

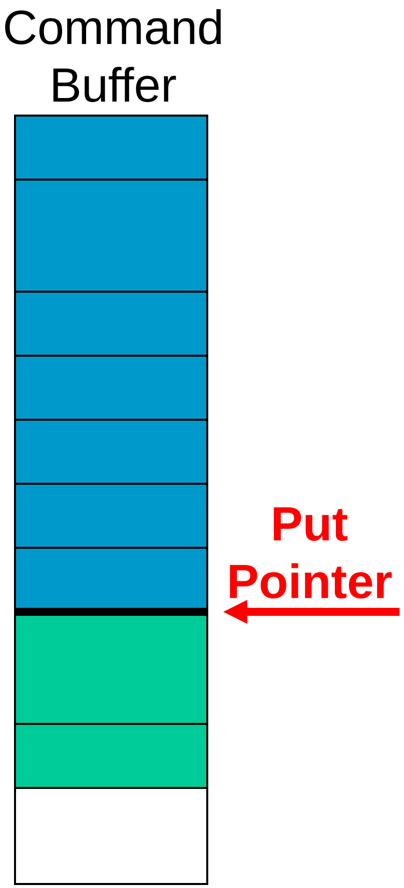
Static 22

Draw 22

Other

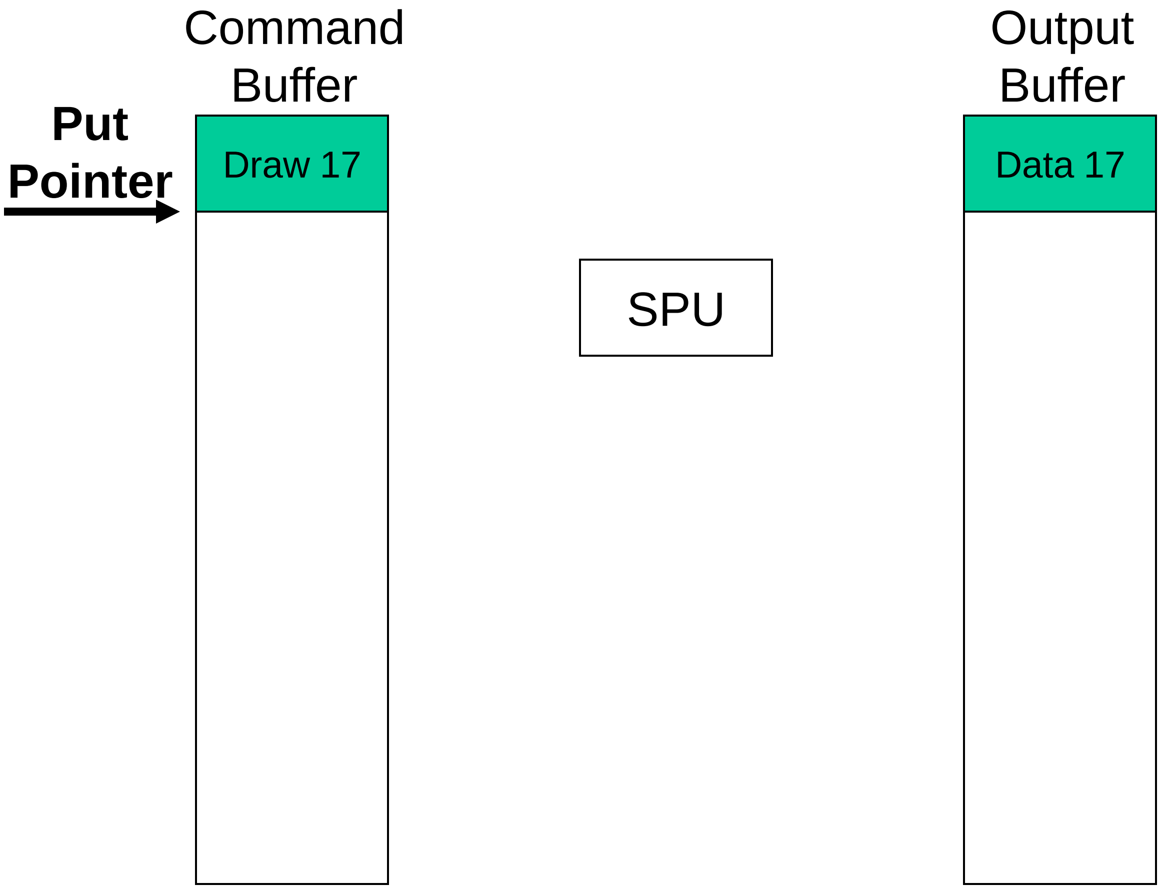
- Dynamic portions are built on the SPU
  - Vertex attribute pointers for any attributes output by the SPU
  - Draw commands when performing triangle culling
  - Commands necessary for ring buffer synchronization
- For brevity, we will not show the PPU generated commands going forward

RSX™ Put Pointer



- RSX™ has an internal “Put Pointer”
- Commands in the command buffer are executed up to the Put Pointer
- Commands after the Put Pointer are not executed

SPU↔RSX™ Synchronization Using the Put Pointer



Command Buffer

Put Pointer

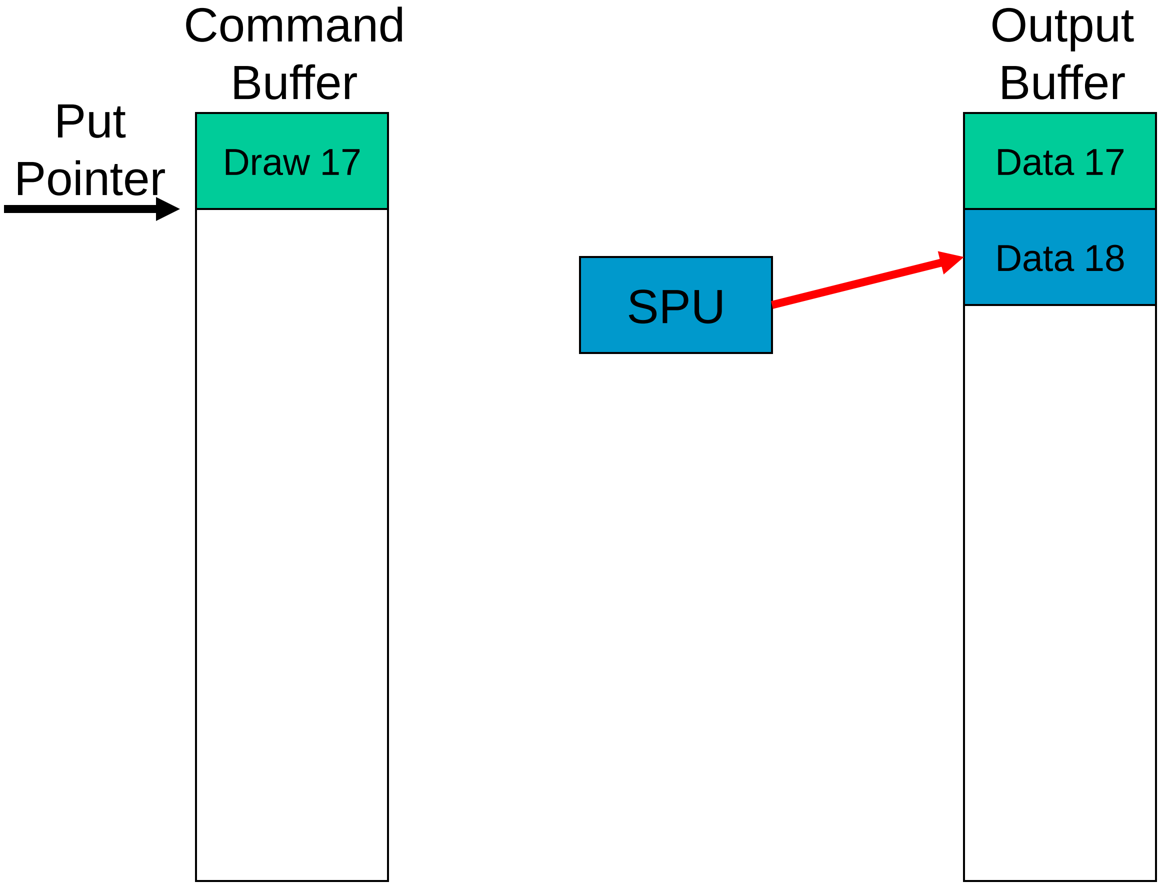
Draw 17

SPU

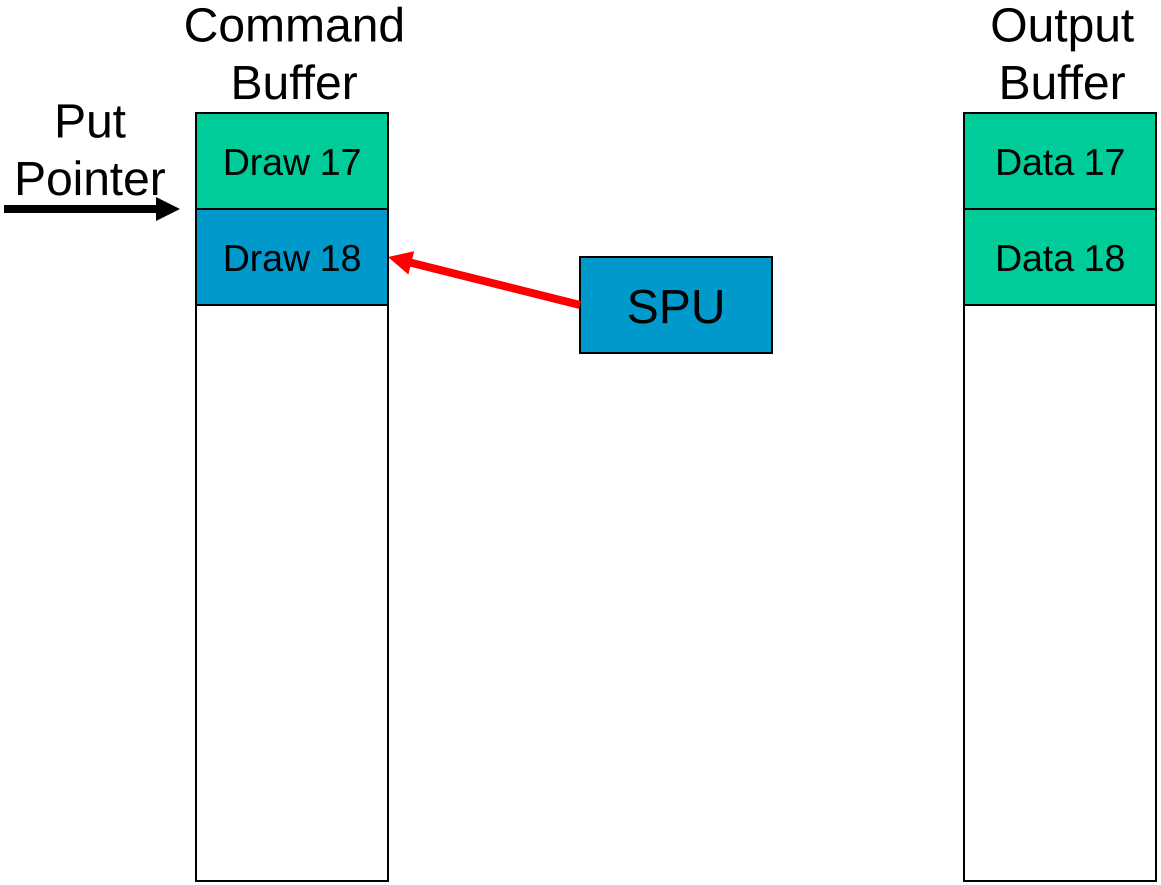
Output Buffer

Data 17

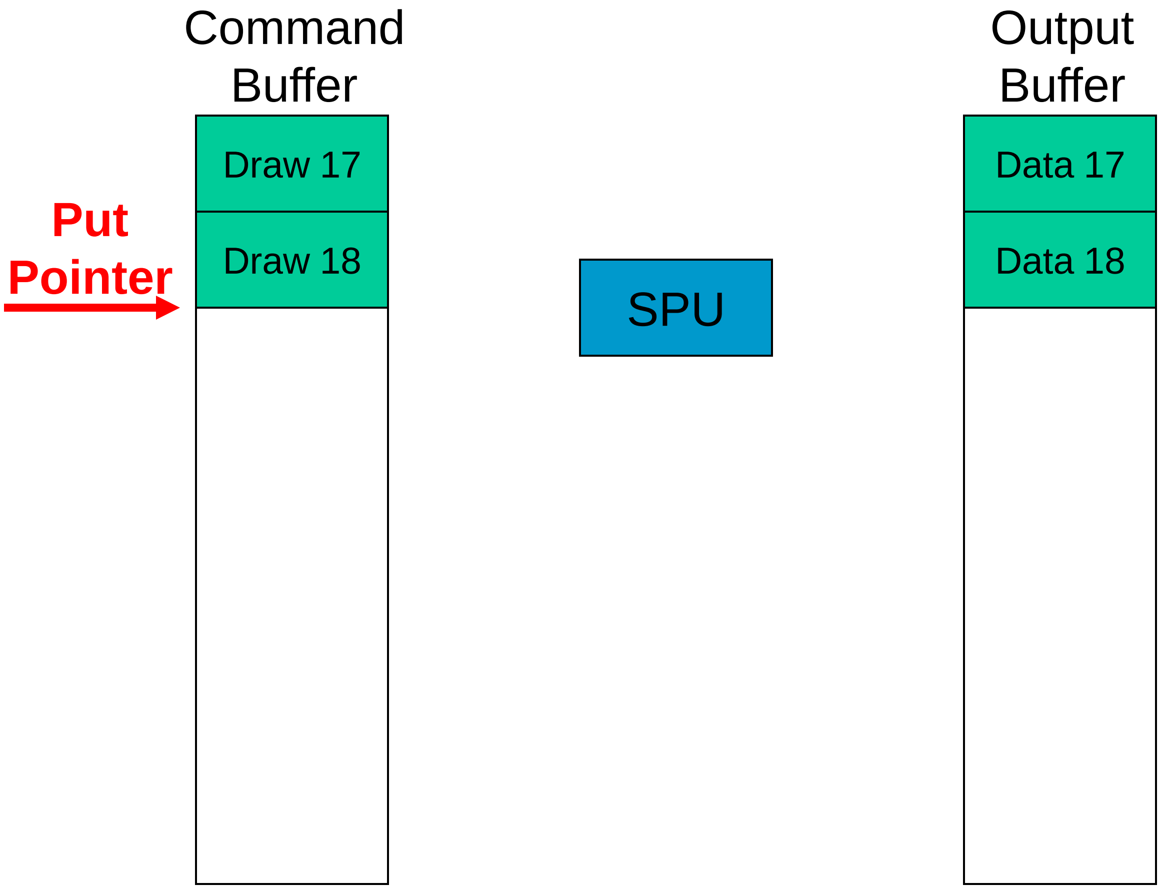
SPU outputs vertex and index data



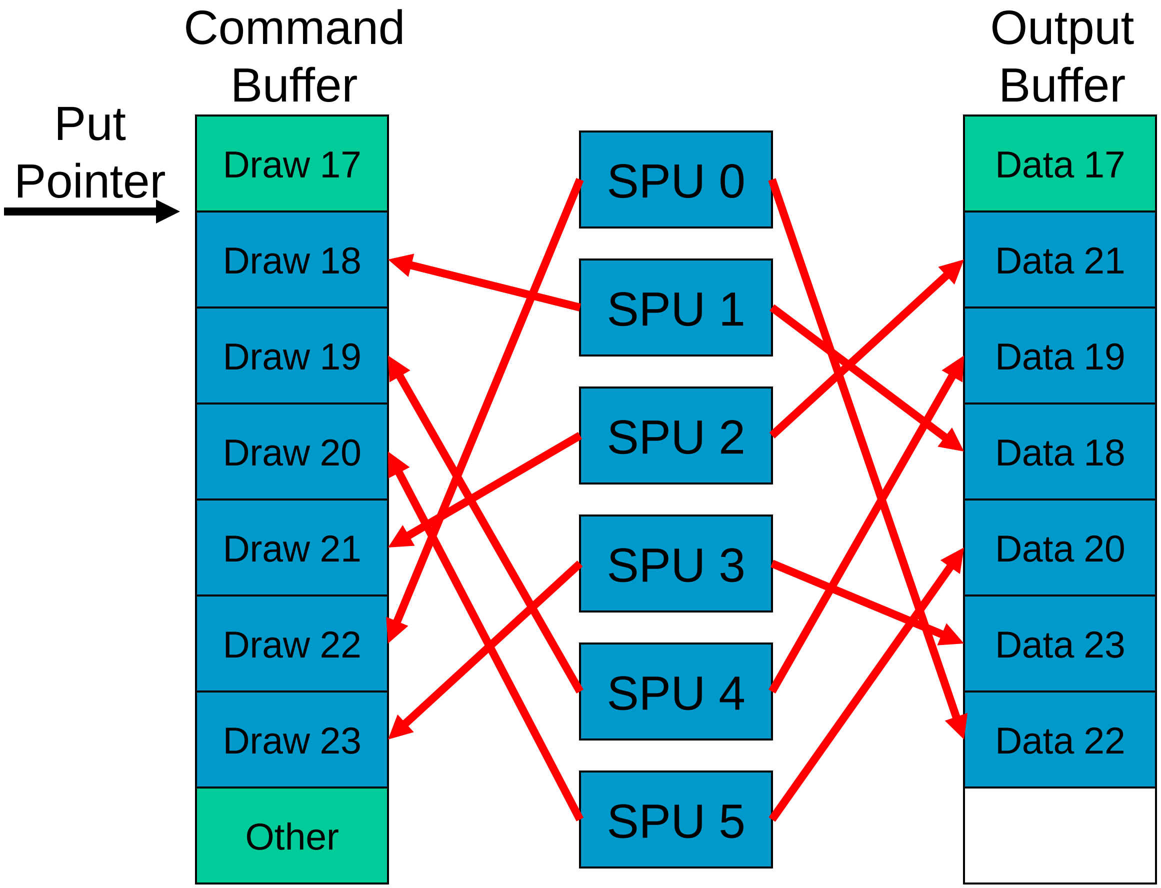
SPU outputs vertex attribute pointers and draw commands



SPU updates the Put Pointer



But there are six SPUs, so who updates the Put Pointer?



Command Buffer

Put Pointer

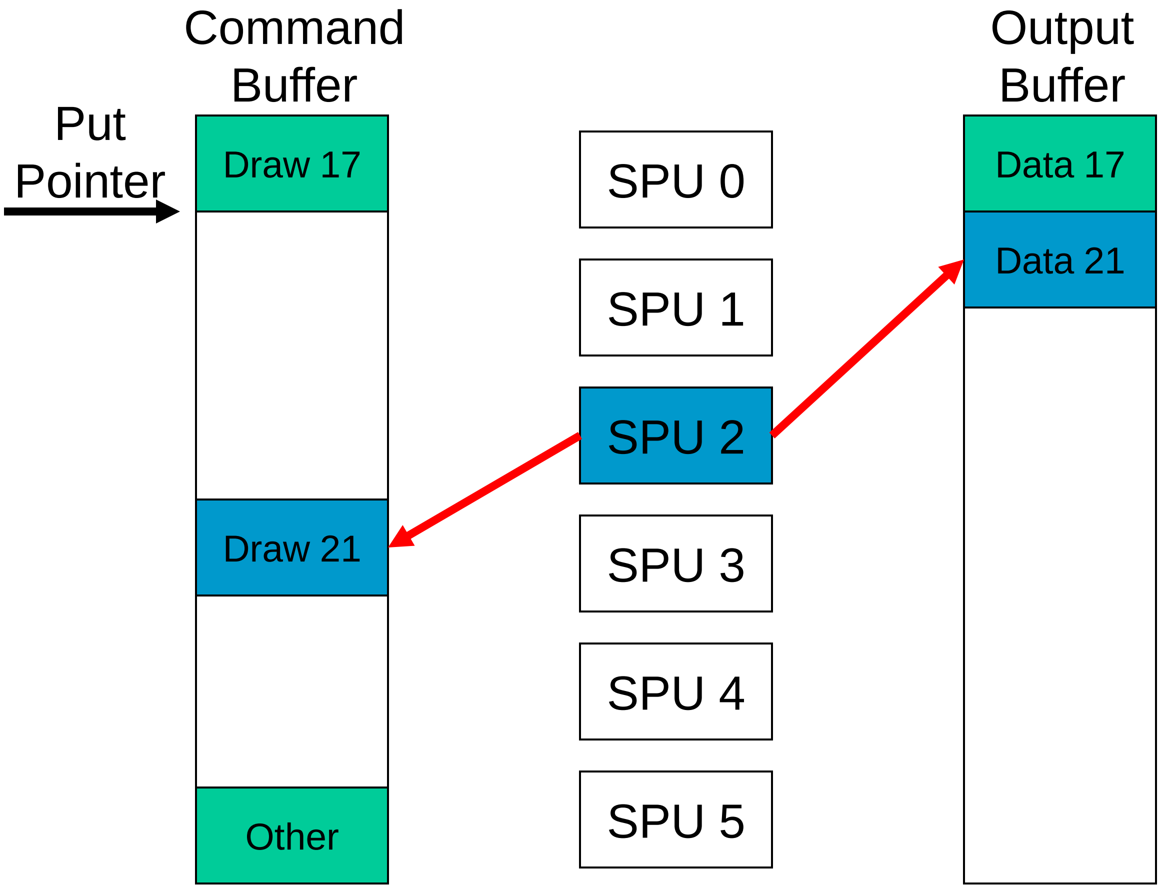
Draw 17
Draw 18
Draw 19
Draw 20
Draw 21
Draw 22
Draw 23
Other

SPU 0
SPU 1
SPU 2
SPU 3
SPU 4
SPU 5

Output Buffer

Data 17
Data 21
Data 19
Data 18
Data 20
Data 23
Data 22

SPUs are asynchronous, so they can finish in any order!



Command Buffer

Put Pointer

Draw 17

Draw 21

Other

SPU 0

SPU 1

SPU 2

SPU 3

SPU 4

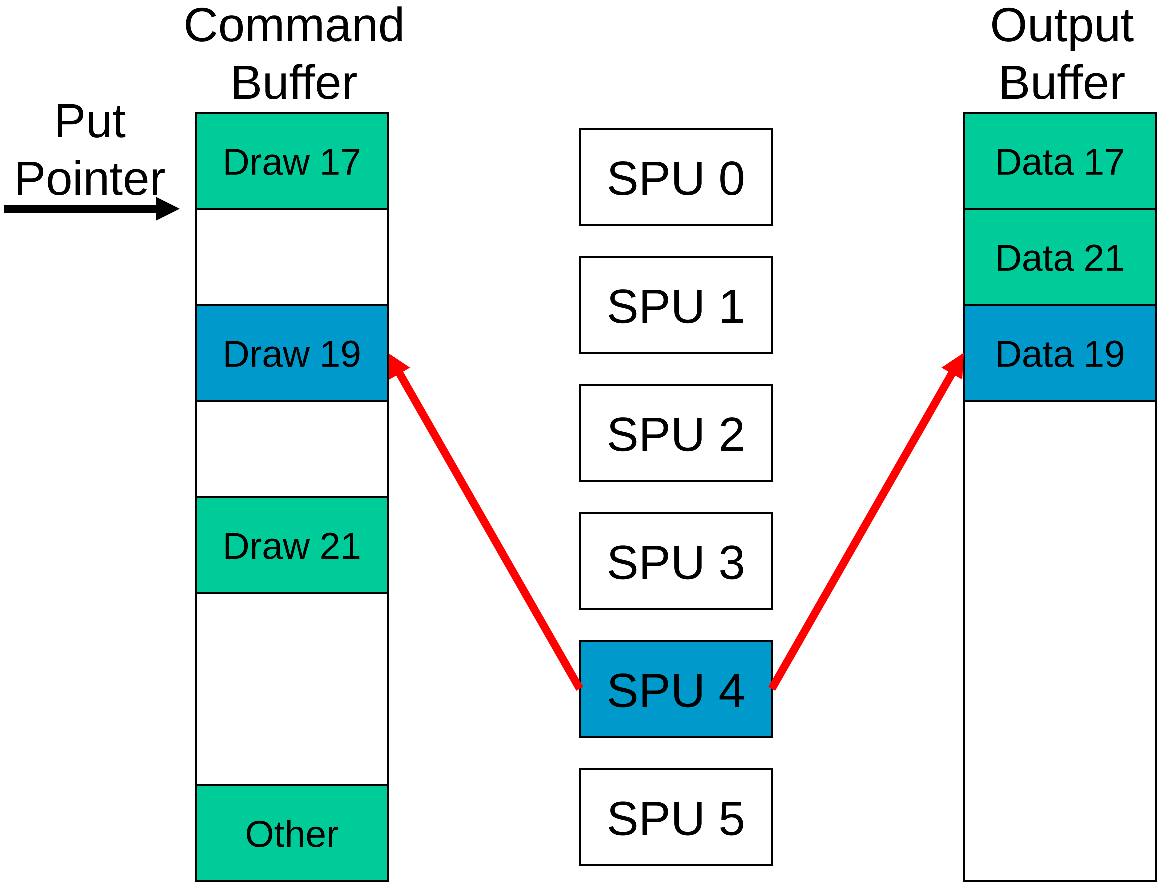
SPU 5

Output Buffer

Data 17

Data 21

So, the SPUs must synchronize with each other!



Command Buffer

Put Pointer

Draw 17

Draw 19

Draw 21

Other

SPU 0

SPU 1

SPU 2

SPU 3

SPU 4

SPU 5

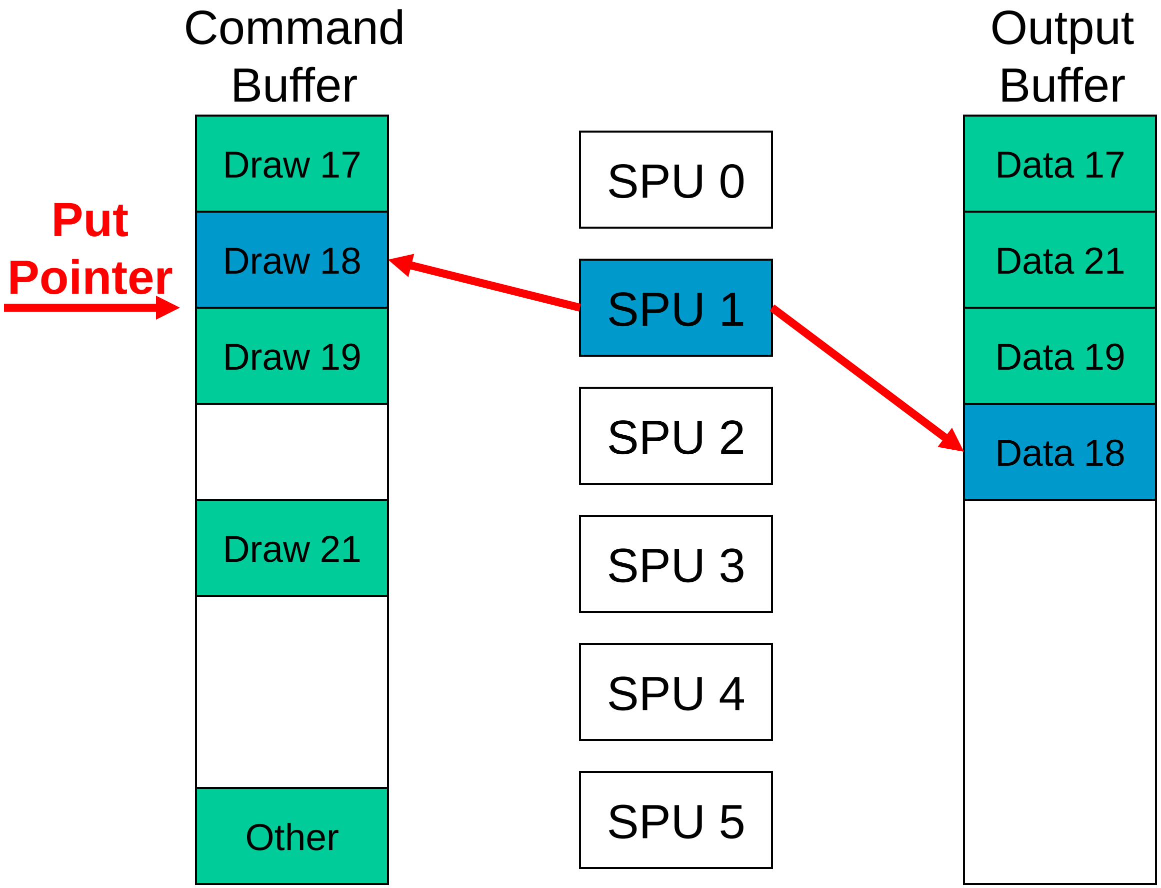
Output Buffer

Data 17

Data 21

Data 19

The Put Pointer is updated only when ALL previous jobs are done…



Command Buffer

Draw 17

Draw 18

Draw 19

Draw 21

Other

Put Pointer

SPU 0

SPU 1

SPU 2

SPU 3

SPU 4

SPU 5

Output Buffer

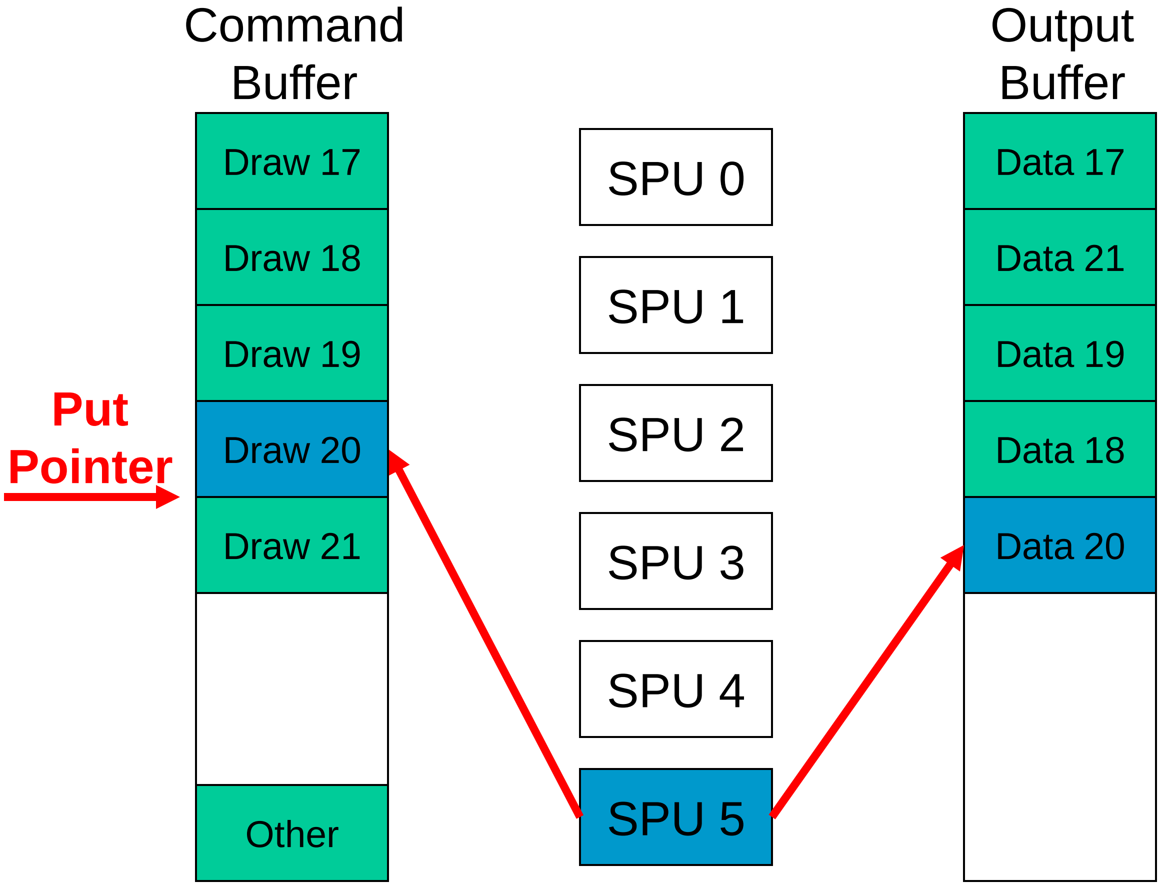
Data 17

Data 21

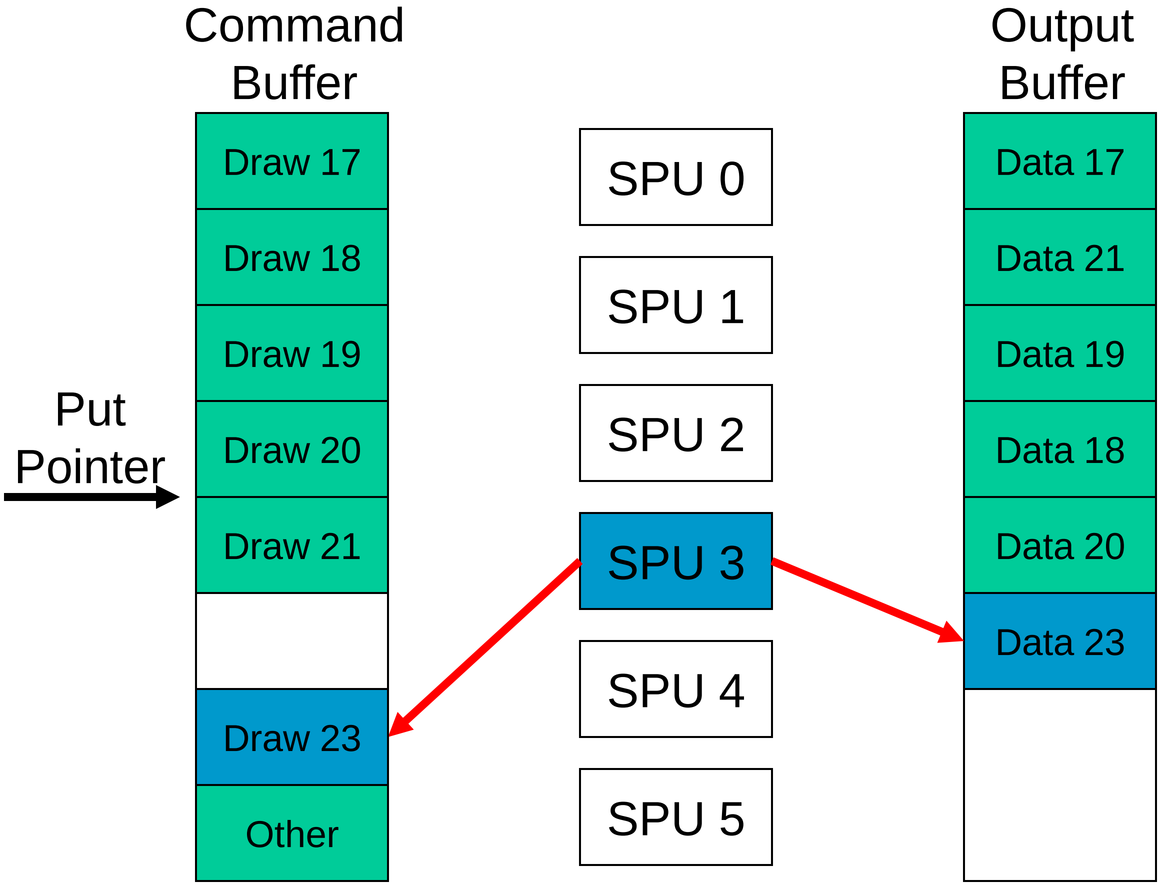
Data 19

Data 18

But can only be moved to the end of this job's draw commands



Remember to guarantee progress!



Command Buffer

Draw 17
Draw 18
Draw 19
Draw 20
Draw 21

Draw 23
Other

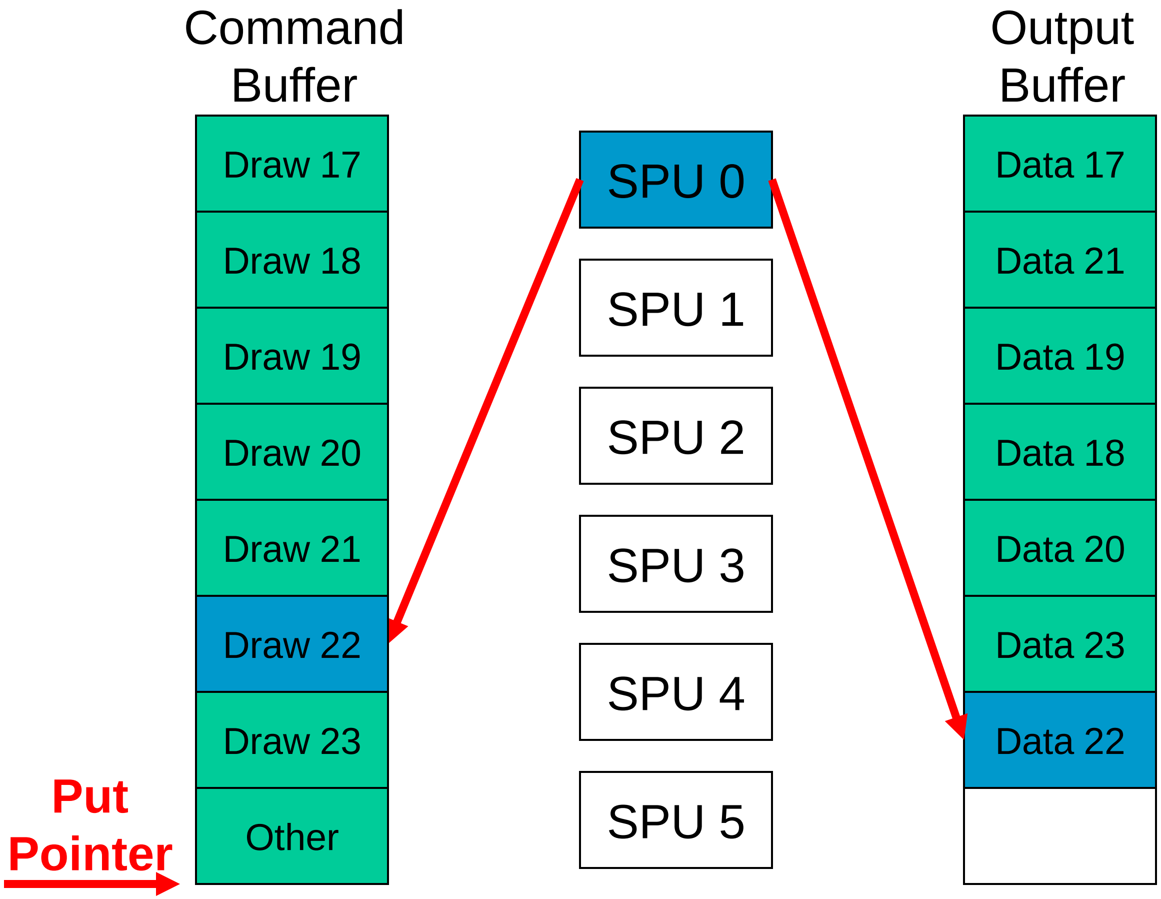
Put Pointer

SPU 0
SPU 1
SPU 2
SPU 3
SPU 4
SPU 5

Output Buffer

Data 17
Data 21
Data 19
Data 18
Data 20
Data 23

The last job finished moves the Put Pointer to the end of the buffer



Command Buffer

Draw 17

Draw 18

Draw 19

Draw 20

Draw 21

Draw 22

Draw 23

Other

Put Pointer

SPU 0

SPU 1

SPU 2

SPU 3

SPU 4

SPU 5

Output Buffer

Data 17

Data 21

Data 19

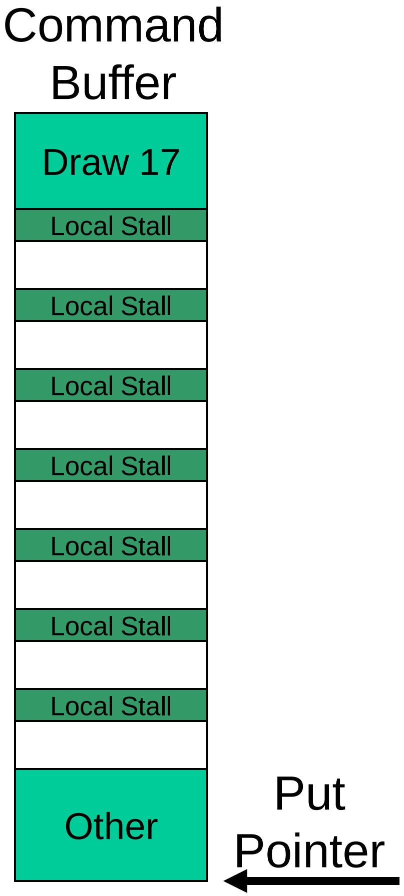
Data 18

Data 20

Data 23

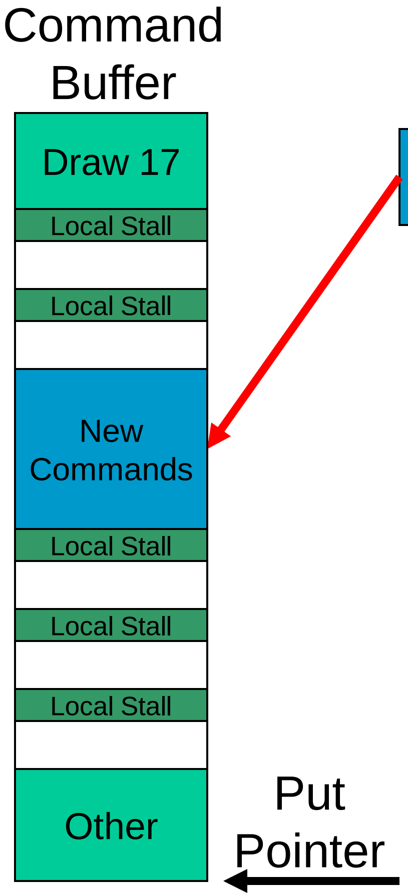
Data 22

SPU↔RSX™ Synchronization Using Local Stalls



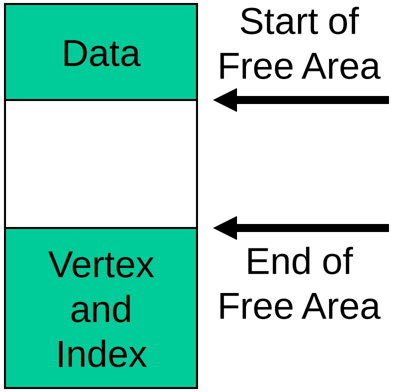
- Easier and faster than Put Pointer synchronization
- Place local stalls in the command buffer where necessary
- RSX™ will stop processing at a local stall until it is overwritten by new commands
- SPUs will generally stay ahead of the RSX™, so stalls rarely occur

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



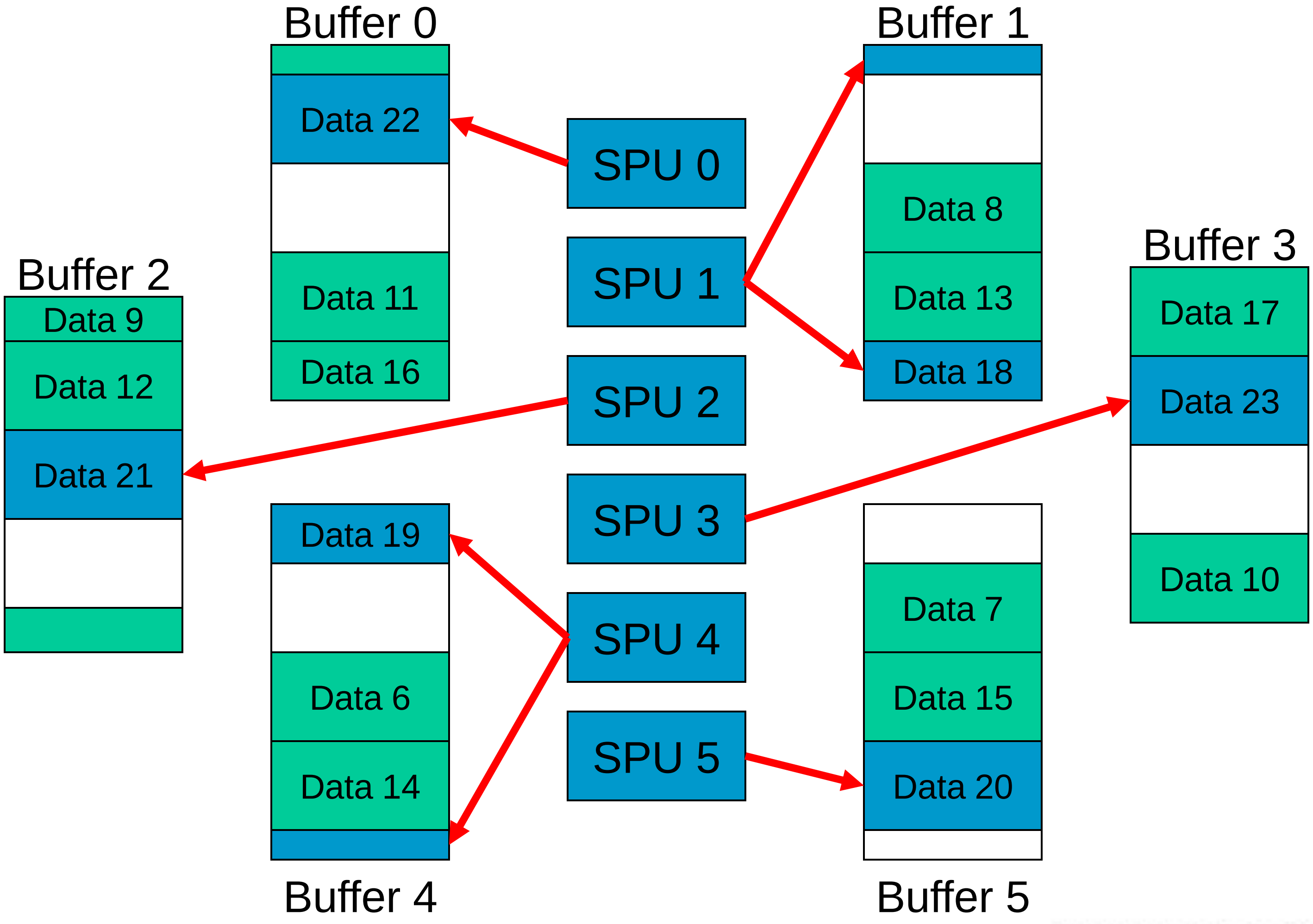
- No SPU↔SPU synchronization required!
- Please see the document regarding this technique on the PS3 Developer's Support website for crucial details

Ring Buffers



- Small memory footprint
- Will not run out of memory
- Can stall the SPUs if buffers become full
- Objects need to be processed in the same order the RSX™ renders them to prevent deadlock

Each SPU has its own buffer to prevent deadlock



Buffer 0
Data 22
Data 11
Data 16

Buffer 1
Data 8
Data 13
Data 18

Buffer 2
Data 9
Data 12
Data 21

Buffer 3
Data 17
Data 23
Data 10

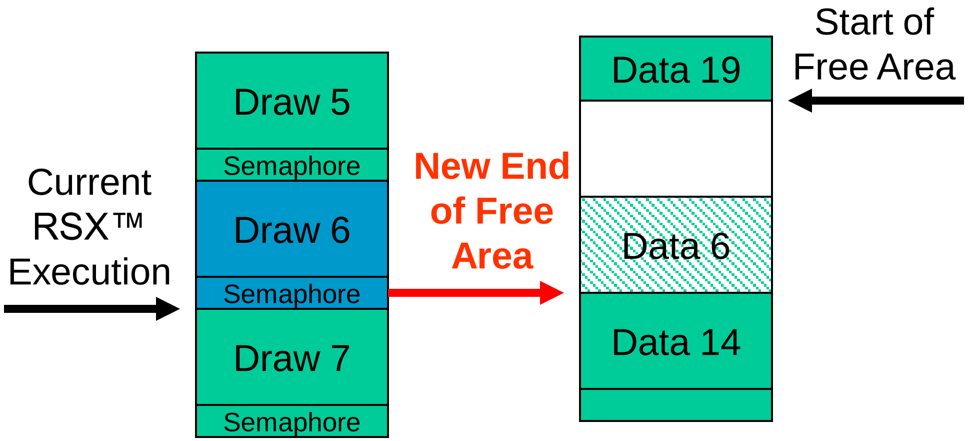
Buffer 4
Data 19
Data 6
Data 14

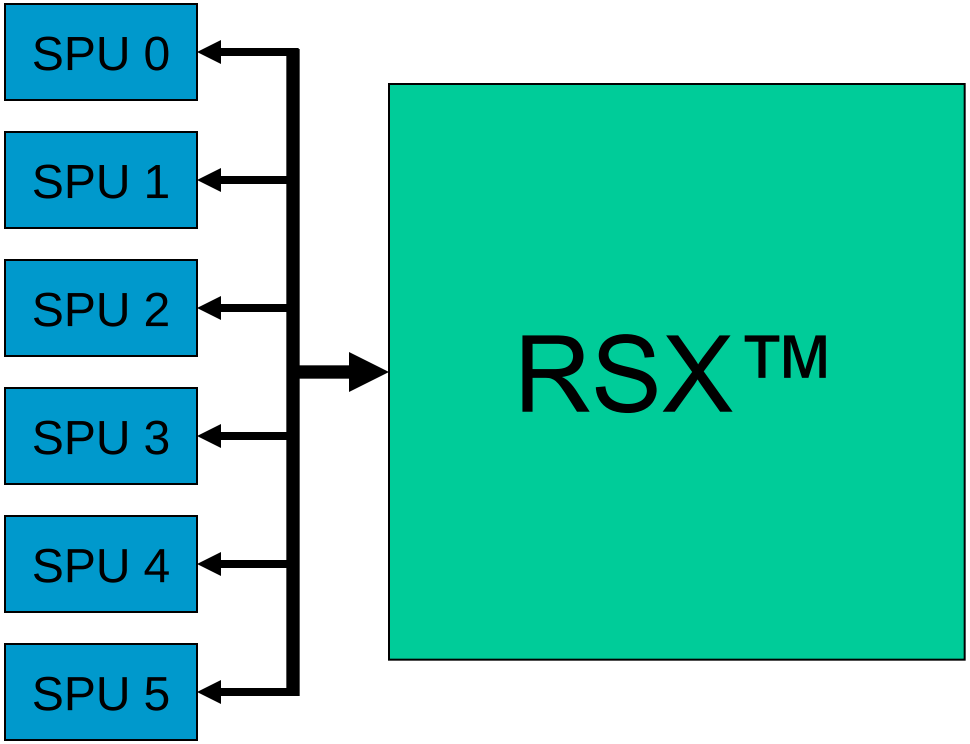
Buffer 5
Data 7
Data 15
Data 20

SPU 0
SPU 1
SPU 2
SPU 3
SPU 4
SPU 5

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




SPU 0
SPU 1
SPU 2
SPU 3
SPU 4
SPU 5
RSX™

Future Work

- Cg compiler for SPUs
  - Complicated vertex programs could be run on the SPUs instead of the RSX™
  - Can’t have too many outputs otherwise the RSX™ will take longer loading them than it would have to run the program

GameDevelopers Conference 07
TAKE CONTROL
March 5-9, 2007 in San Francisco
CMP

Future Work

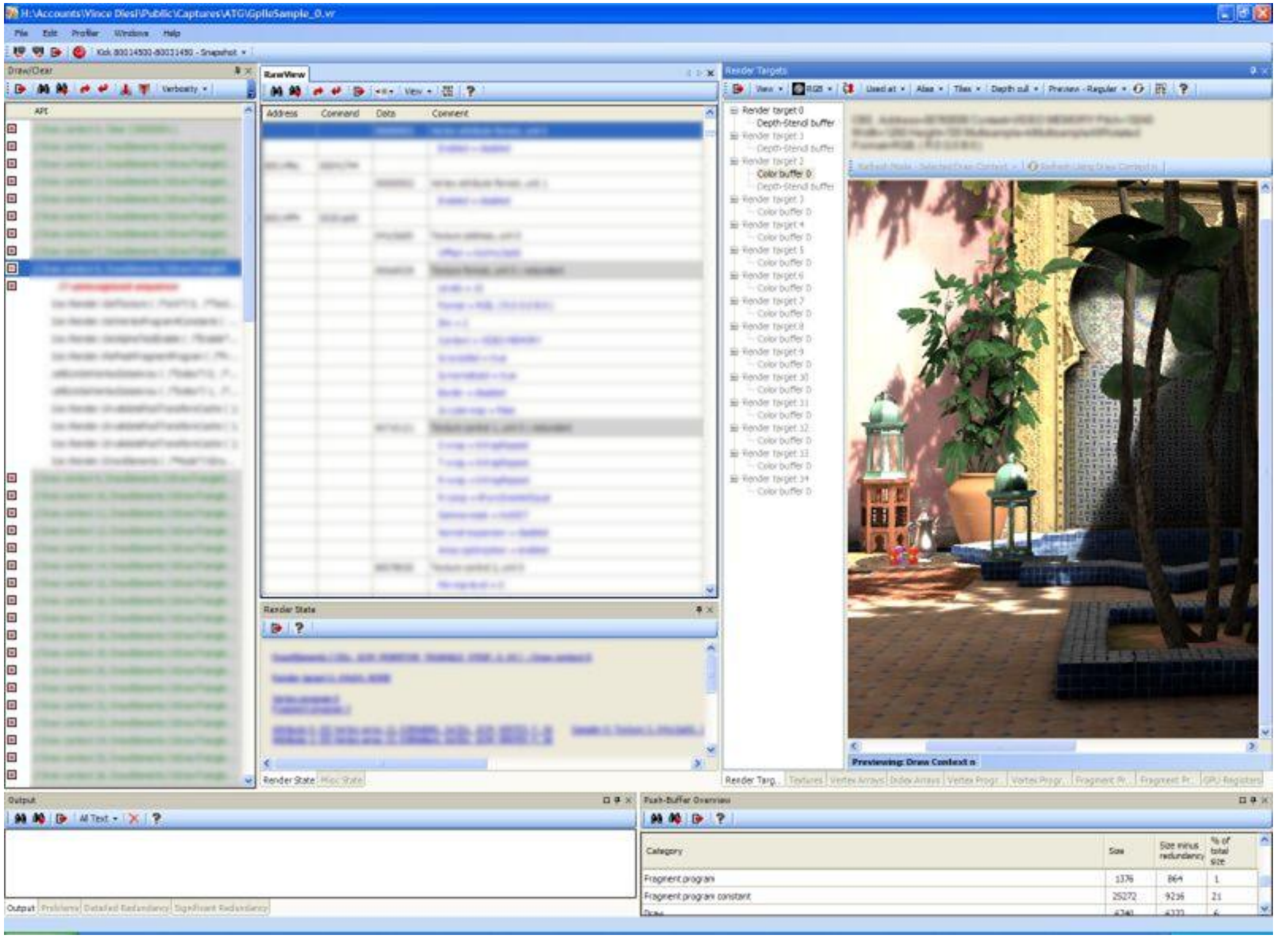
- Shadow map generation on SPUs
  - Large load removed from RSX™
  - Very doable
    - Much more complicated if you have alpha cutouts in your textures

GCM Replay

GCM Replay

- New tool for use with the RSX™
  - Analysis
  - Debugging
  - Profiling
- Will be released soon to all licensed developers as part of PLAYSTATION®Edge
- Main tool runs on the PC
- Integration into your title is simple and easy

Capture a Snapshot

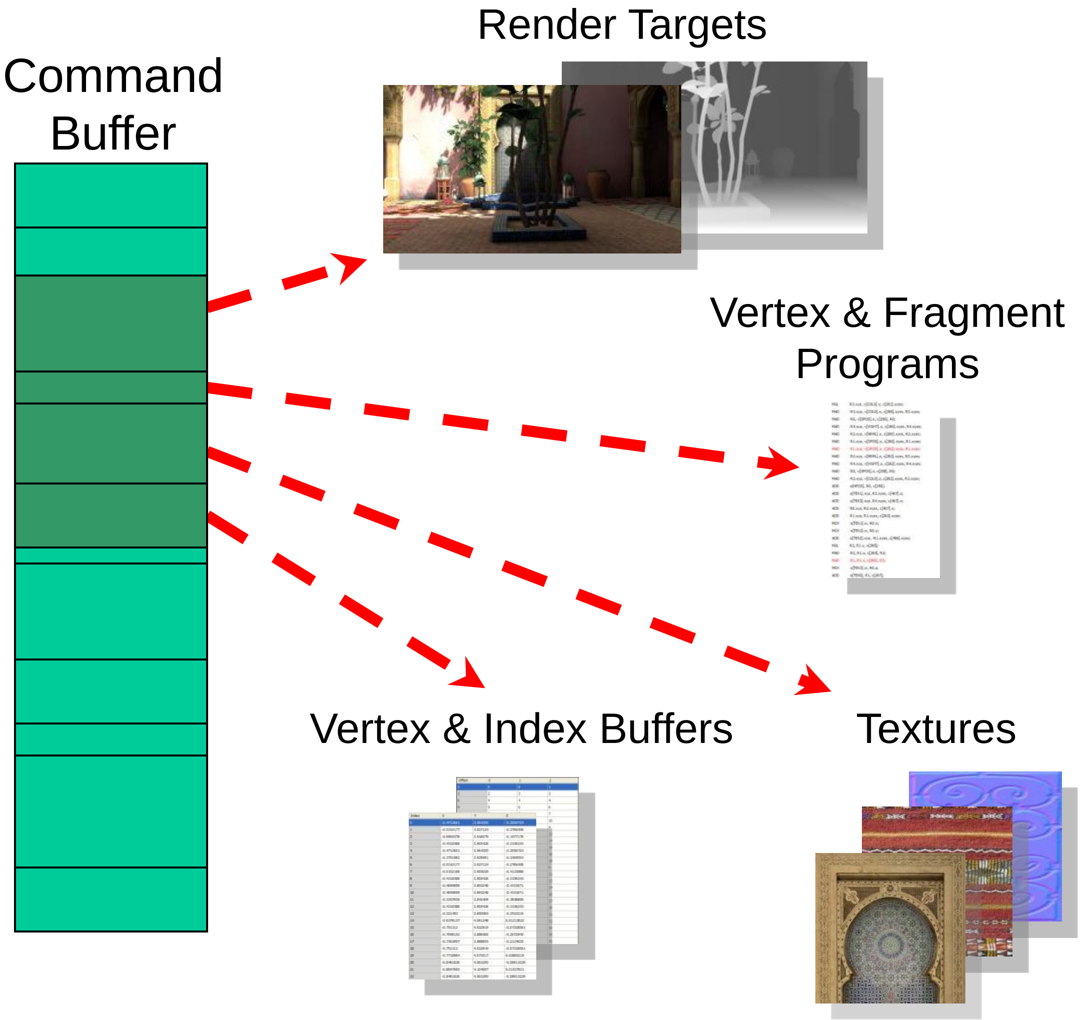


H:\Accounts\Vince Diesi\Public\Captures\ATG\GpllsSample_0.vr

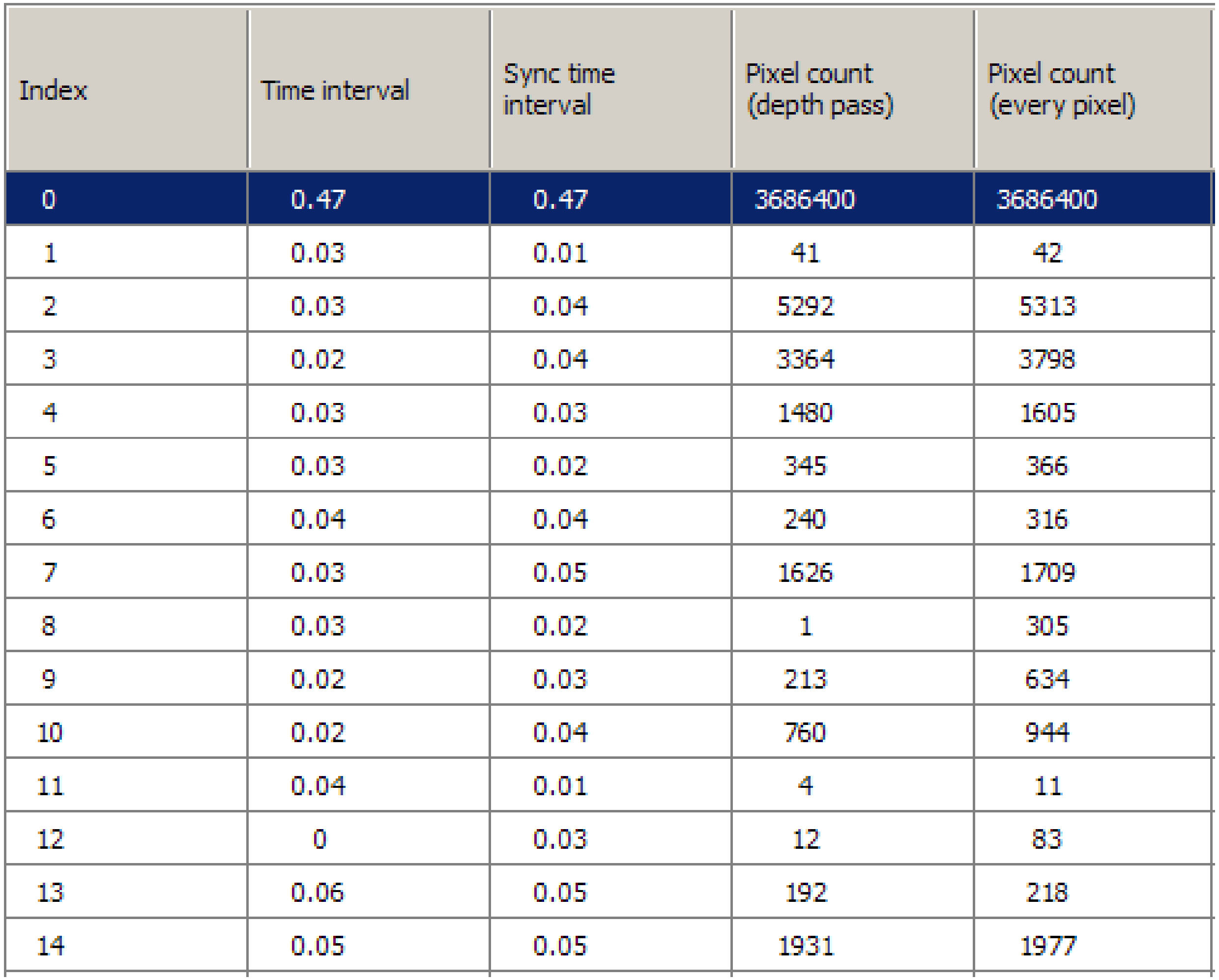
Kick 80014500-80031450 - Snapshot

Category	Size	Size minus redundancy	% of total size
Fragment program	1376	864	1
Fragment program constant	25272	9216	21
Draw	6740	6272	6

Snapshot Contents

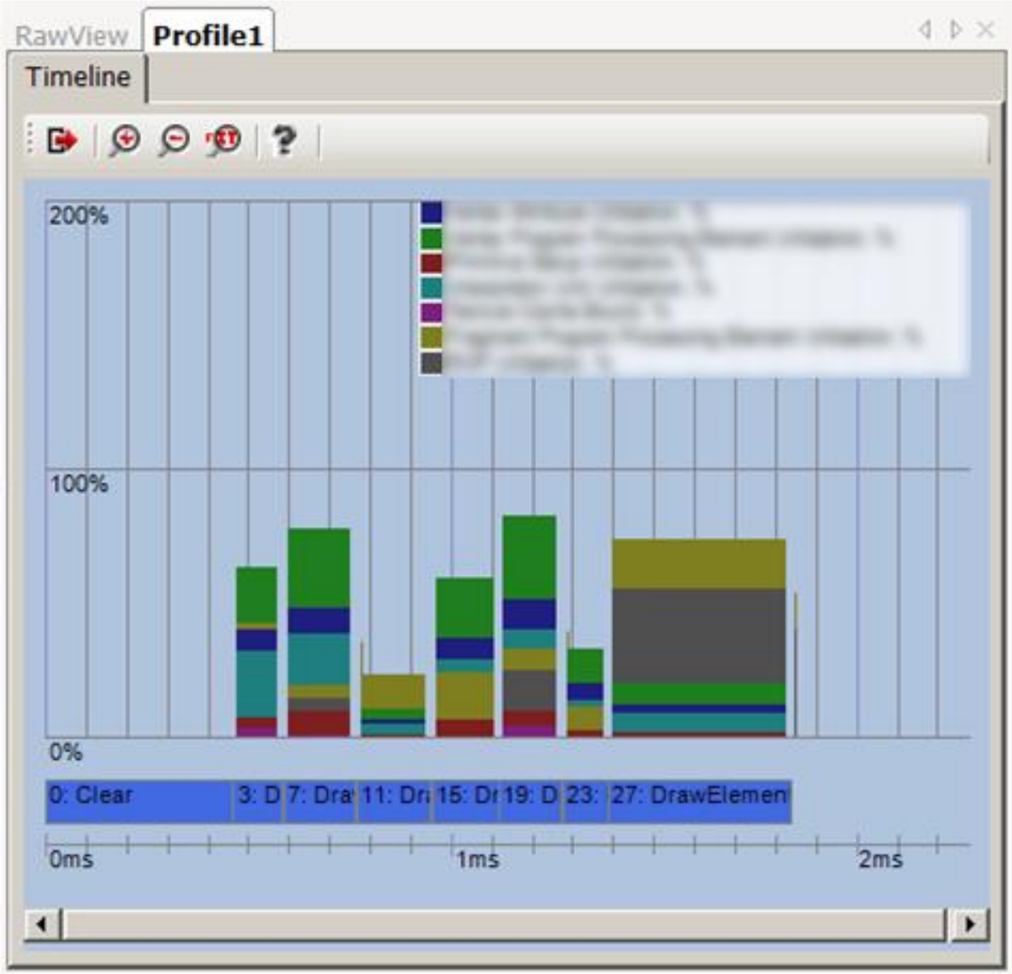


Performance Analysis

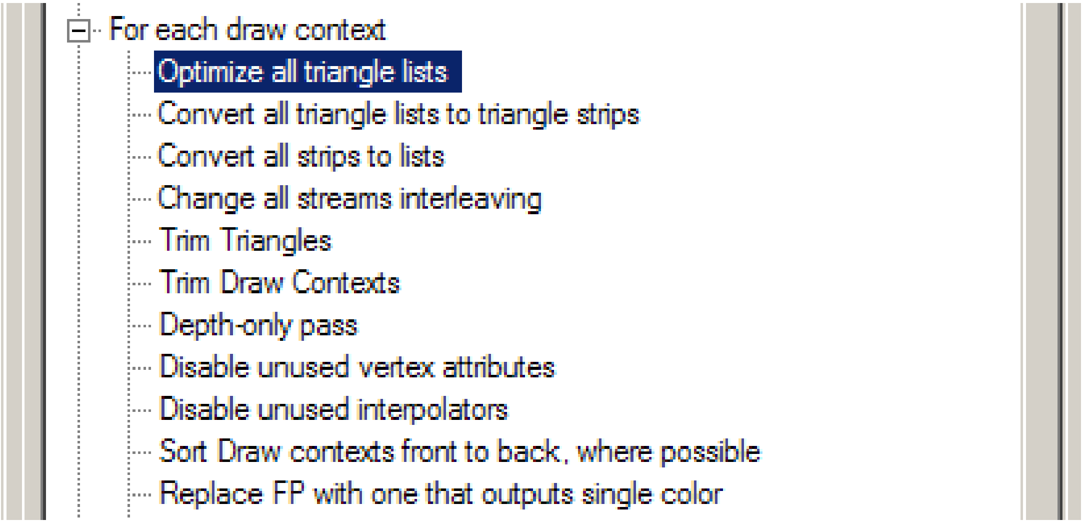


Index	Time interval	Sync time interval	Pixel count (depth pass)	Pixel count (every pixel)
0	0.47	0.47	3686400	3686400
1	0.03	0.01	41	42
2	0.03	0.04	5292	5313
3	0.02	0.04	3364	3798
4	0.03	0.03	1480	1605
5	0.03	0.02	345	366
6	0.04	0.04	240	316
7	0.03	0.05	1626	1709
8	0.03	0.02	1	305
9	0.02	0.03	213	634
10	0.02	0.04	760	944
11	0.04	0.01	4	11
12	0	0.03	12	83
13	0.06	0.05	192	218
14	0.05	0.05	1931	1977

Find Bottlenecks



“What Ifs”

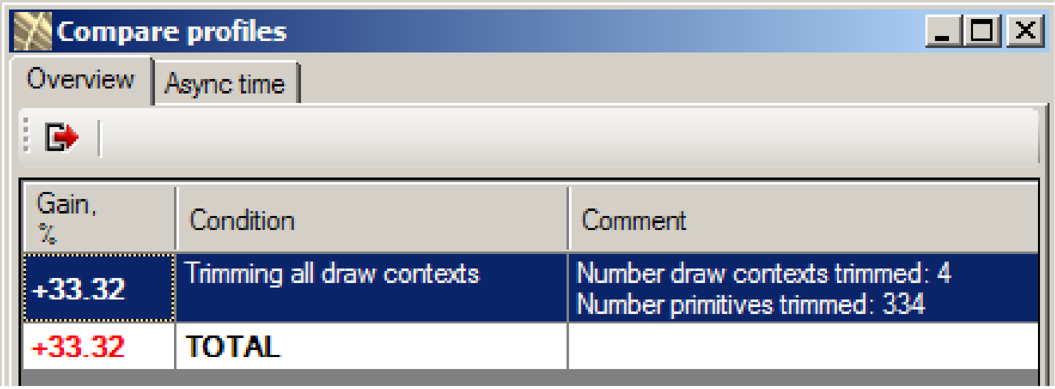


- For each draw context
  - Optimize all triangle lists
  - Convert all triangle lists to triangle strips
  - Convert all strips to lists
  - Change all streams interleaving
  - Trim Triangles
  - Trim Draw Contexts
  - Depth-only pass
  - Disable unused vertex attributes
  - Disable unused interpolators
  - Sort Draw contexts front to back, where possible
  - Replace FP with one that outputs single color

Q: What If I Had Efficient Triangle Culling?

What would the performance gain be?

- GCM Replay can remove all draw calls to triangles which never write a pixel
- Once this is done, GCM Replay can reprofile the snapshot and compute the speed increase



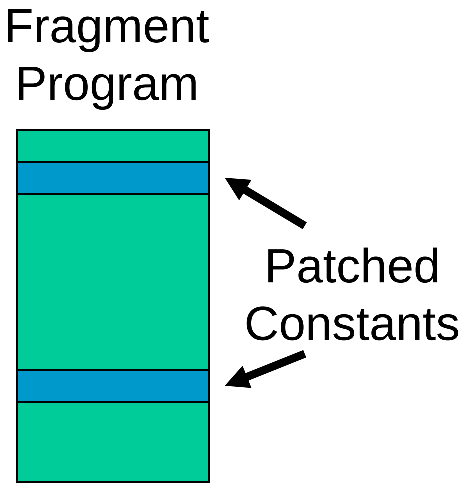
Gain, %	Condition	Comment
+33.32	Trimming all draw contexts	Number draw contexts trimmed: 4 Number primitives trimmed: 334
+33.32	TOTAL	

A: Cull Triangles Using an SPU!

- Triangle culling techniques shown earlier can dramatically increase performance

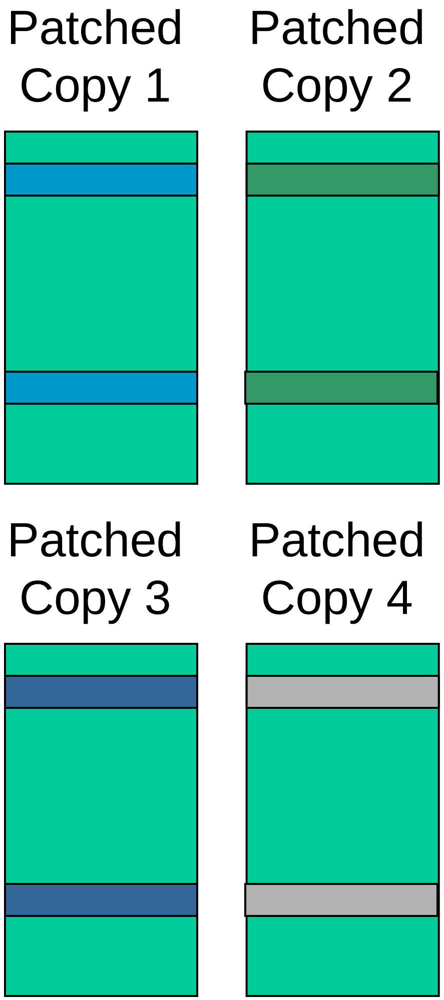
Q: What If My Setting of Fragment Program Constants Was Done Externally to the Command Buffer?

Conventional Patch Technique



- One copy of each fragment program is kept in memory
- Individual fragment program constants are patched by placing draw commands in the command buffer in the appropriate locations

A: Patch Using the PPU or SPU!

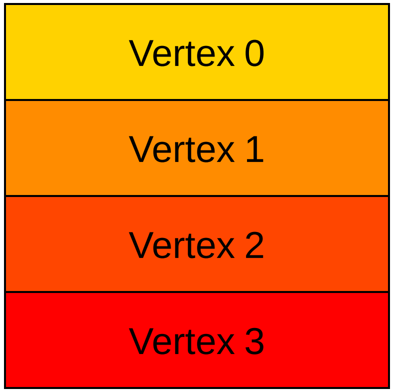


- Multiple copies of fragment programs can be patched with the appropriate constants either on the PPU or an SPU
  - Removes 100% of the RSX™ load for patching fragment programs
  - If done as part of SPU processing of a vertex set, synchronization will be already be taken care of

Q: What If I Had More Optimal Indexed Triangle Lists?

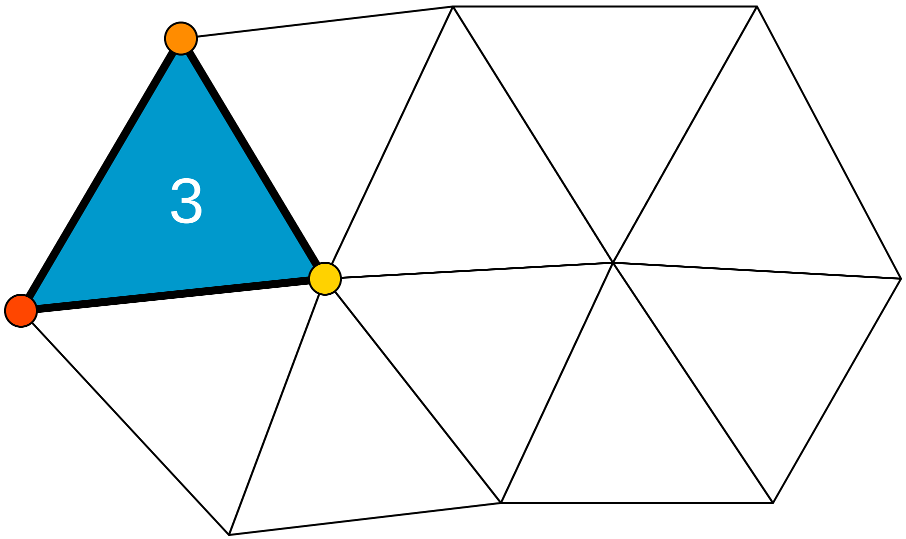
A: Optimize for the Four Vertex Mini-cache!

Four Vertex Mini-cache



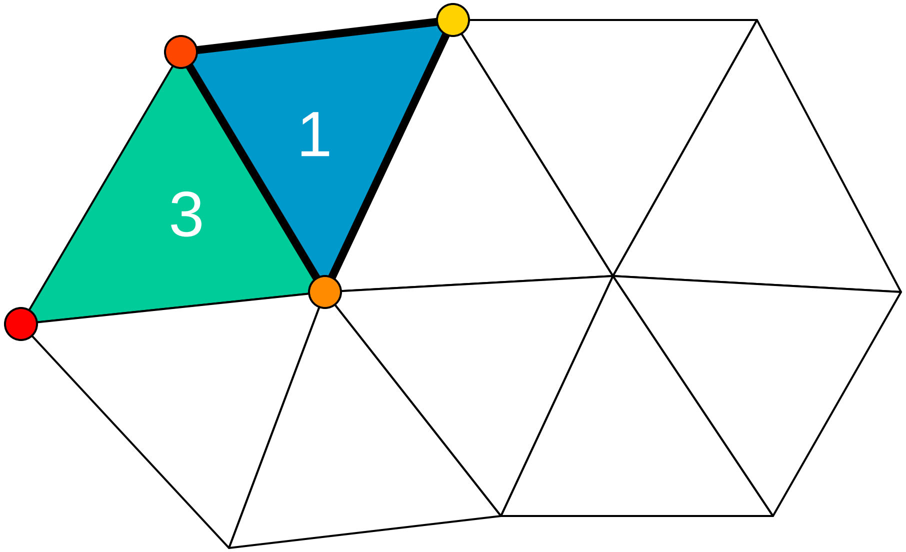
- GCM Replay contains an optimizer for indexed triangle list ordering
- Corresponding offline indexed triangle list optimizer available as part of PLAYSTATION®Edge

Strip Example



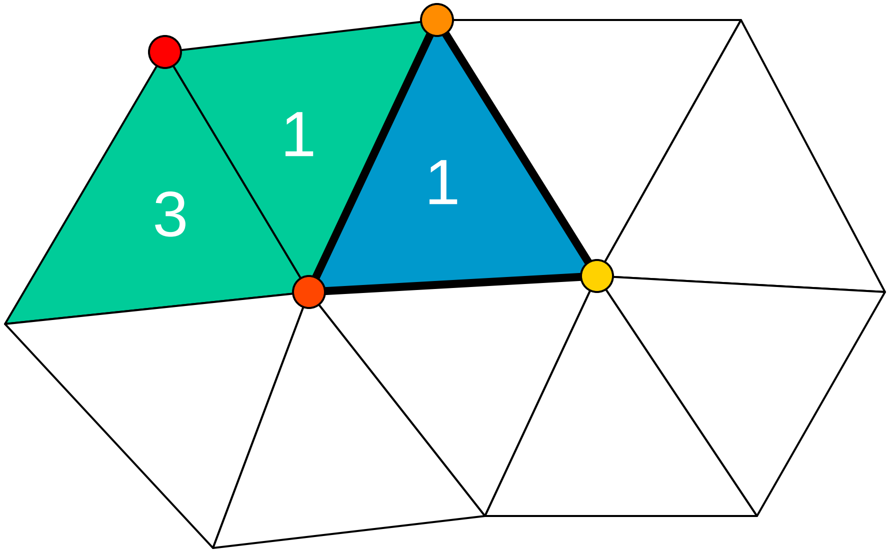
3 new vertices

Strip Example



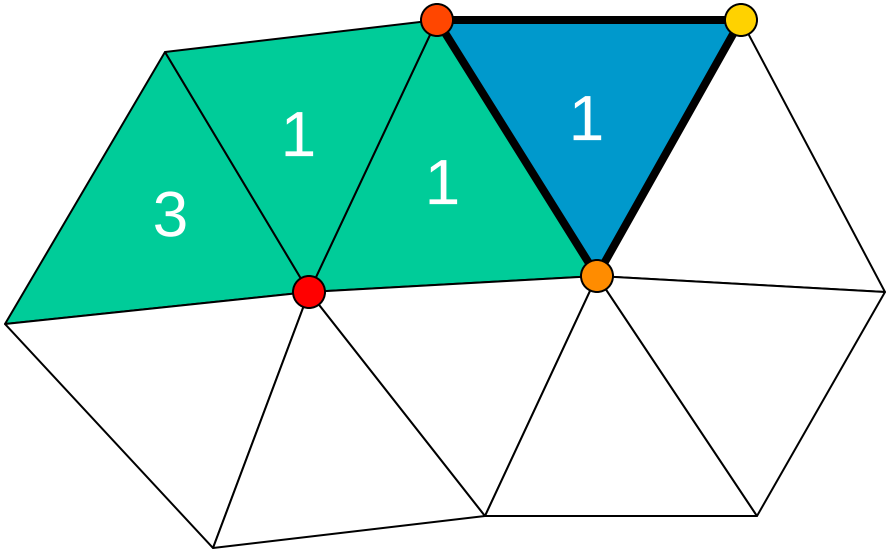
1 new vertex

Strip Example



1 new vertex

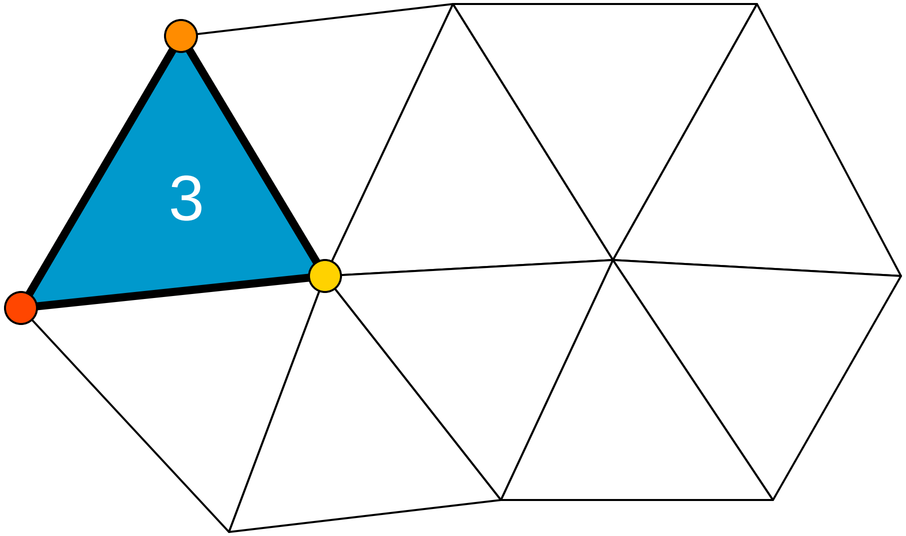
Strip Example



1 new vertex…

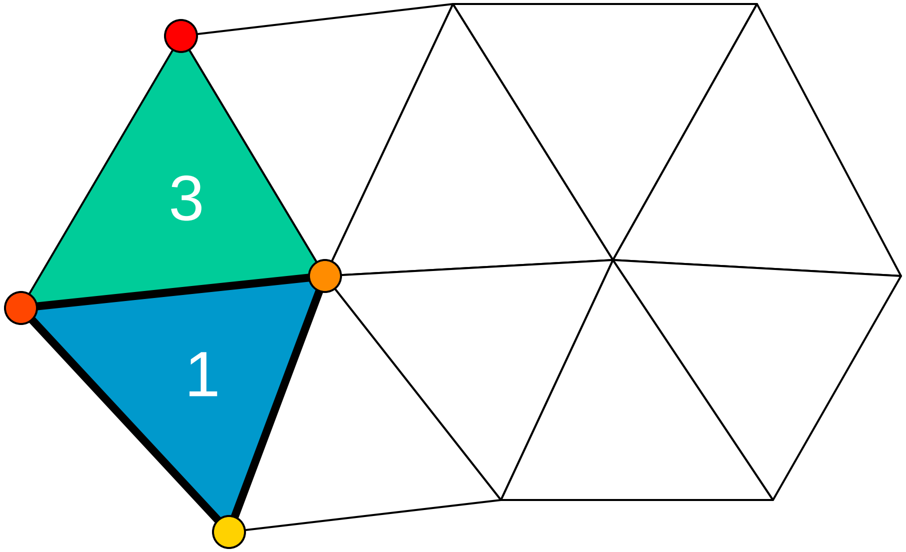
2 vertices + 1 per triangle in total

Free Form Example



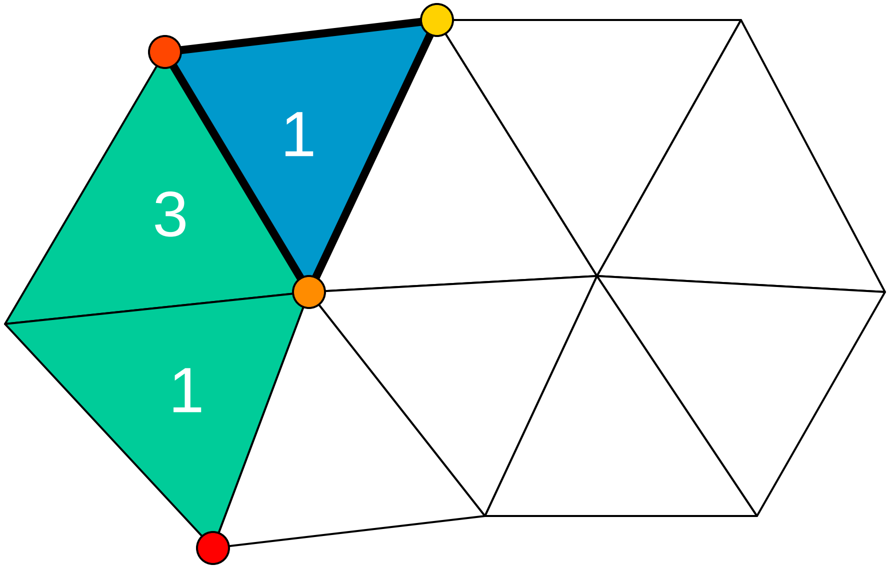
3 new vertices

Free Form Example



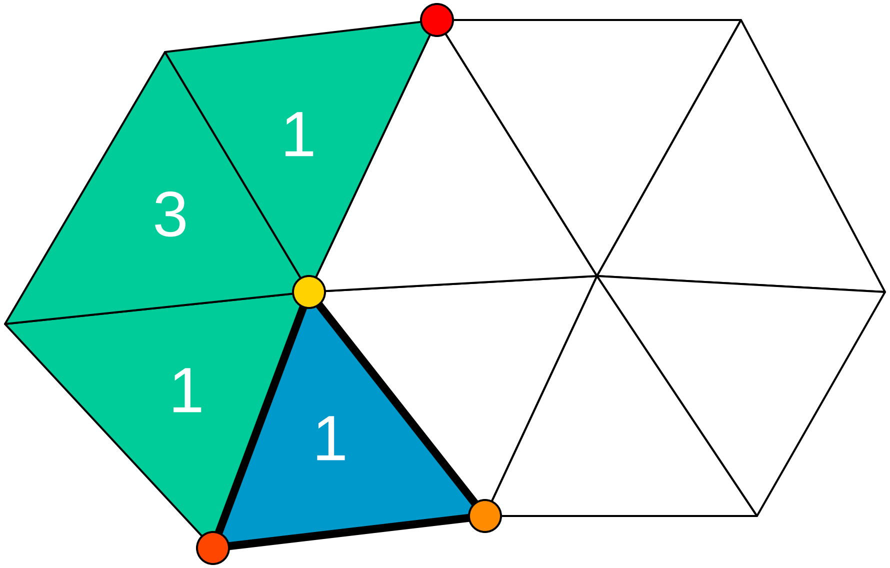
1 new vertex

Free Form Example



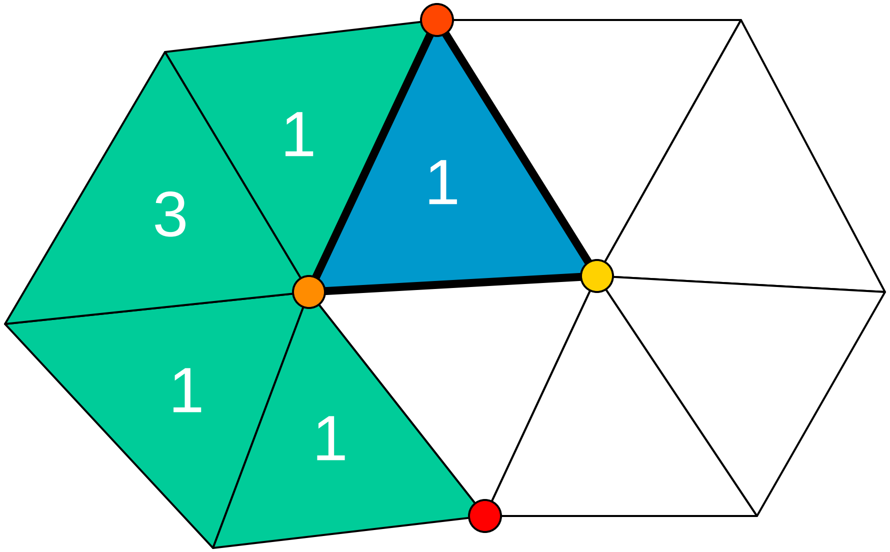
1 new vertex

Free Form Example



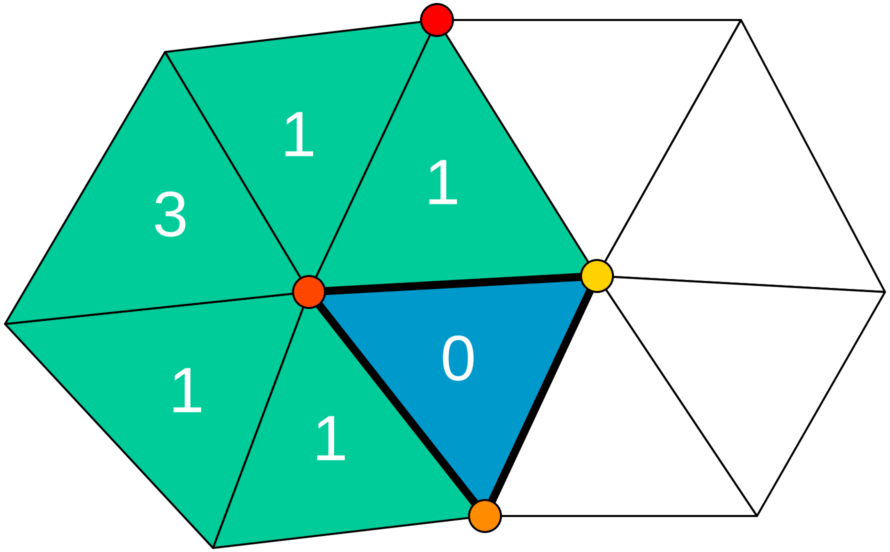
1 new vertex

Free Form Example



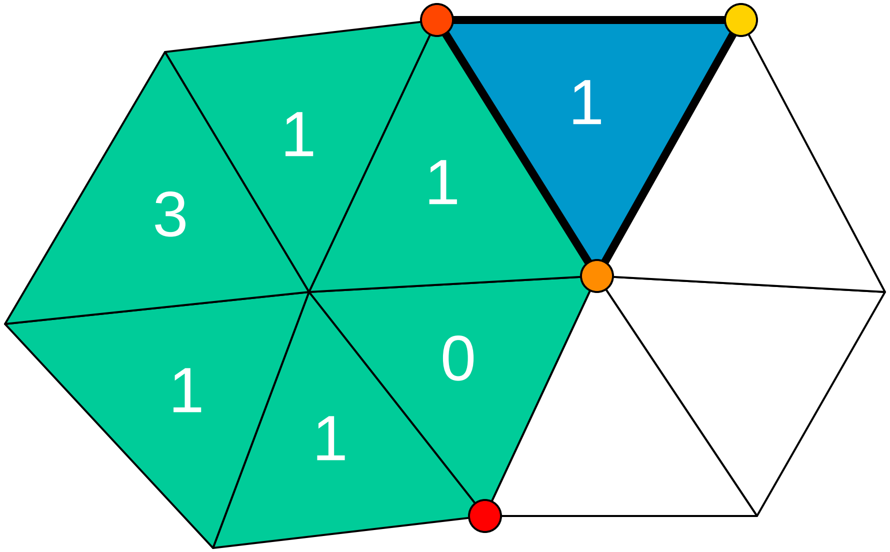
1 new vertex

Free Form Example



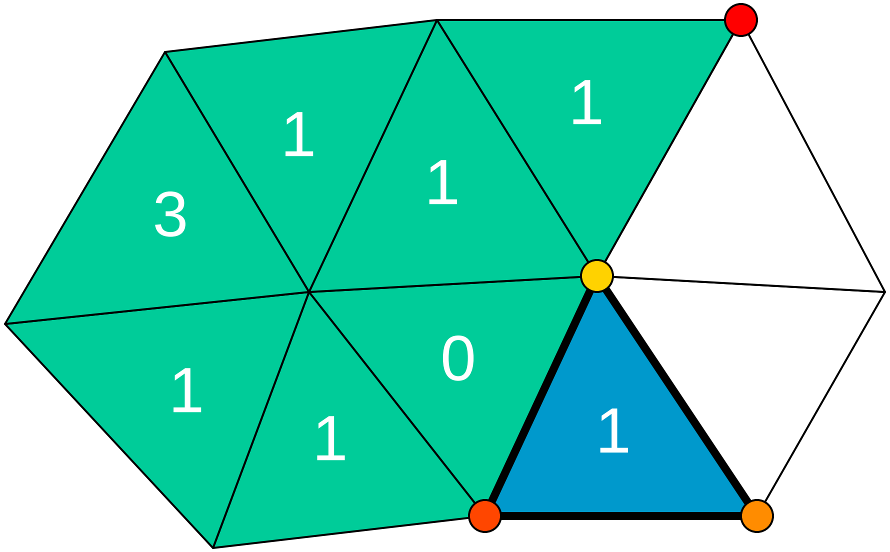
0 new vertices!

Free Form Example



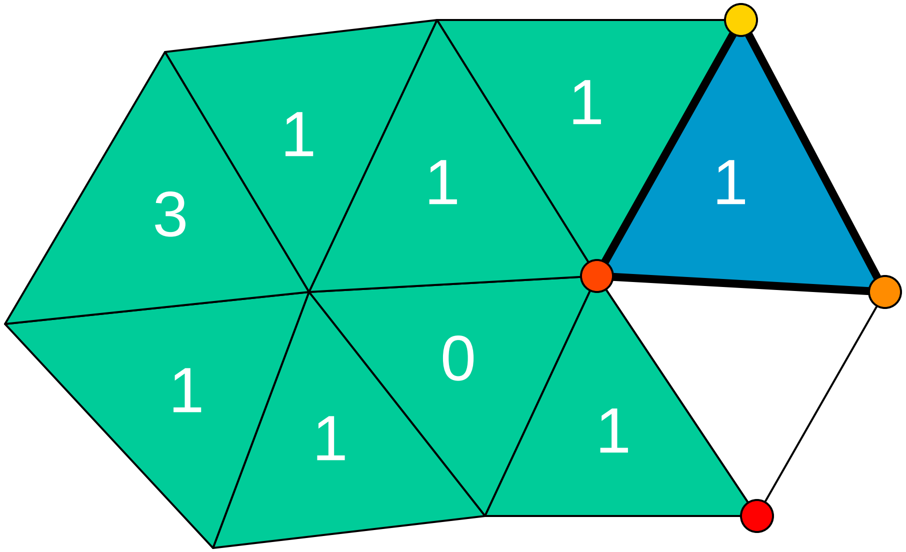
1 new vertex

Free Form Example



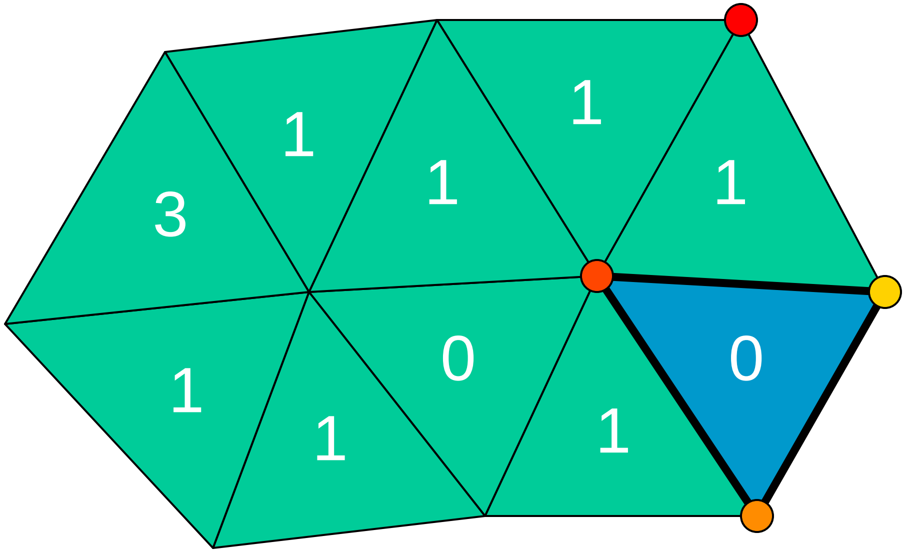
1 new vertex

Free Form Example



1 new vertex

Free Form Example



0 new vertices…

2 vertices + 3 per 4 triangles in total

Q: What If I Had Perfect Object Z-Culling?

- Some objects will not contribute to the final scene because they are entirely blocked by other objects
- GCM Replay will soon be able to show the performance difference if good object Z-culling was performed

A: Object Z-Culling on SPUs

- Write an SPU rasterizer
- Render the depth values of a low polygon version of the environment
- Rasterize and check bounding volumes of objects

WARHAWK™

INCOGNITO™
entertainment

MARCH 5-9
2007
SAN FRANCISCO

MOSCONE
CENTER

TAKE
CONTROL

www.gdconf.com

CMP