MARCH 5-9
2007
SAN FRANCISCO

TAKE CONTROL

www.gdconf.com

MOSCONE CENTER

CMP

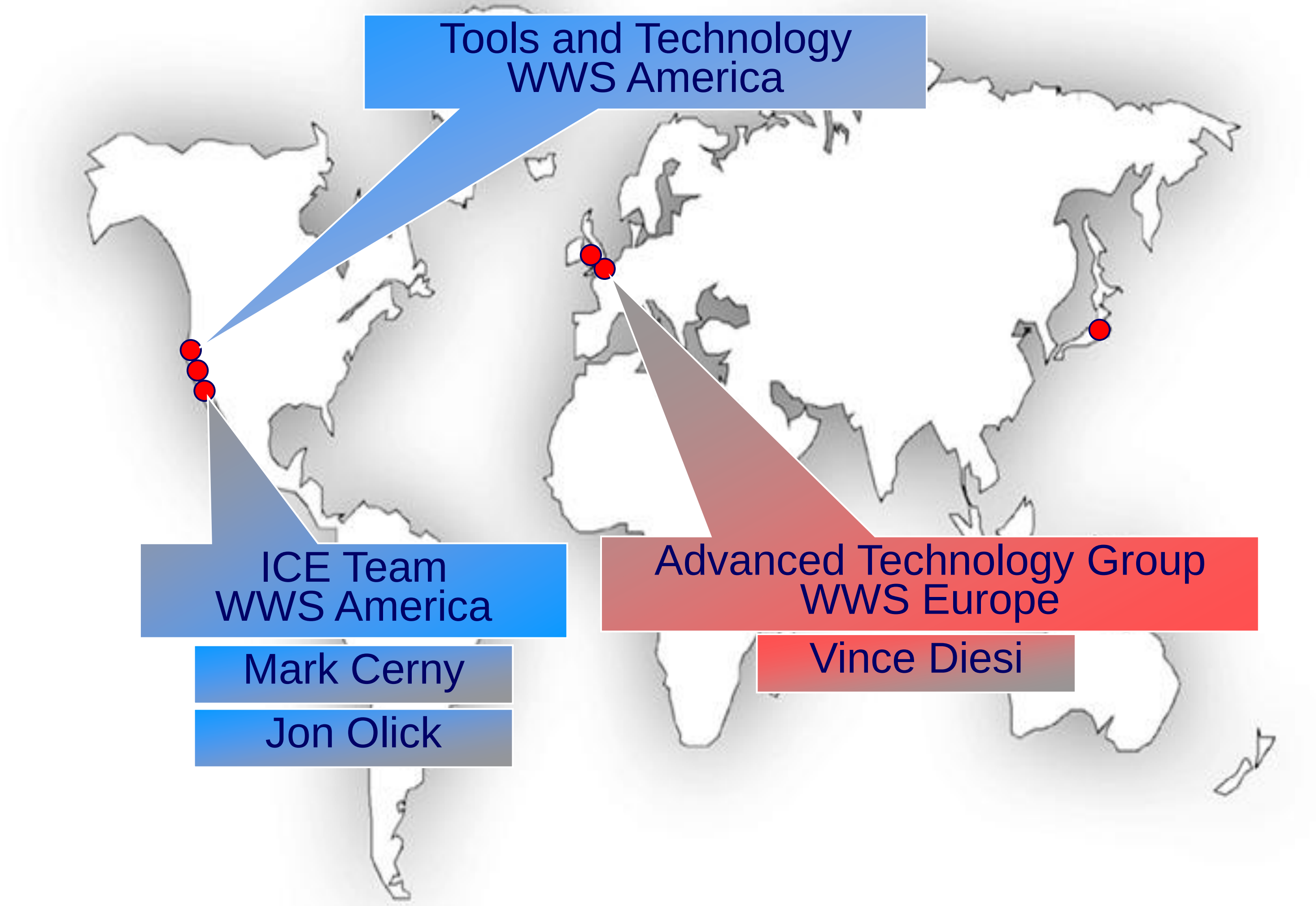
PLAYSTATION®Edge

PLAYSTATION®Edge

Mark Cerny

Jon Olick

Vince Diesi



Tools and Technology
WWS America

ICE Team
WWS America

Mark Cerny

Jon Olick

Advanced Technology Group
WWS Europe

Vince Diesi

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

GCM Replay

- March release
- RSX Performance Analysis
- *Speculative* Performance Analysis

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

PLAYSTATION®Edge Philosophy

- Discrete pieces of technology
  - Targeted for easy adoption
  - Show first party best practices

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

PLAYSTATION®Edge Component Overview

- Animation System
- Geometry Processing
- Compression
- GCM Replay

PLAYSTATION®Edge Component Overview

- Animation System
  - Blend trees of arbitrary depth
  - Several layers of compression
  - High performance
  - Very flexible

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

PLAYSTATION®Edge Component Overview

- Geometry Processing
  - Skinning on SPUs
    - Offload the RSX
  - Triangle Culling on SPUs
    - Remove unnecessary RSX processing
  - Blend Shapes on SPUs
    - Offload the PPU
  - Compressed Data formats
    - SPUs can use better data compression than the RSX

PLAYSTATION®Edge Component Overview

- Compression
  - Fast zlib decompression implemented for the SPUs
  - Increases effective bandwidth from BD-ROM
  - Useful for high speed streaming
  - 40MB/sec with ~25% of an SPU

PLAYSTATION®Edge Component Overview

- GCM Replay
  - New tool for use with the RSX
    - Analysis
    - Debugging
    - Profiling

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

PLAYSTATION®Edge Component Overview

- Full source code available
  - SPU code
    - Runs as SPURS jobs
    - C with Intrinsics
  - PPU and tools code written in C with some C++
  - libGCM used as RSX interface

PLAYSTATION®Edge Component Overview

- Offline Tools Pipeline
  - Generates binary data used by animation and geometry runtime
  - Collada compatible pipeline
  - Multi-layered approach

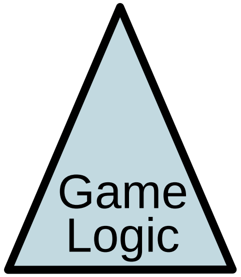
GameDevelopers Conference 07
TAKE CONTROL
March 5-9, 2007 in San Francisco
CMP

PLAYSTATION®Edge Component Overview

- Will be released as part of the PlayStation 3 SDK to all licensed developers

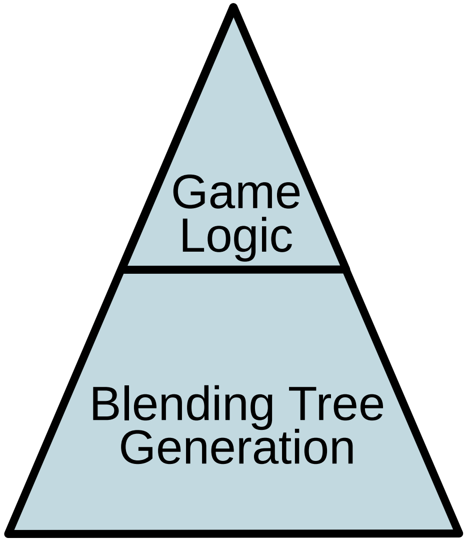
PLAYSTATION®Edge Animation

Animation Processing

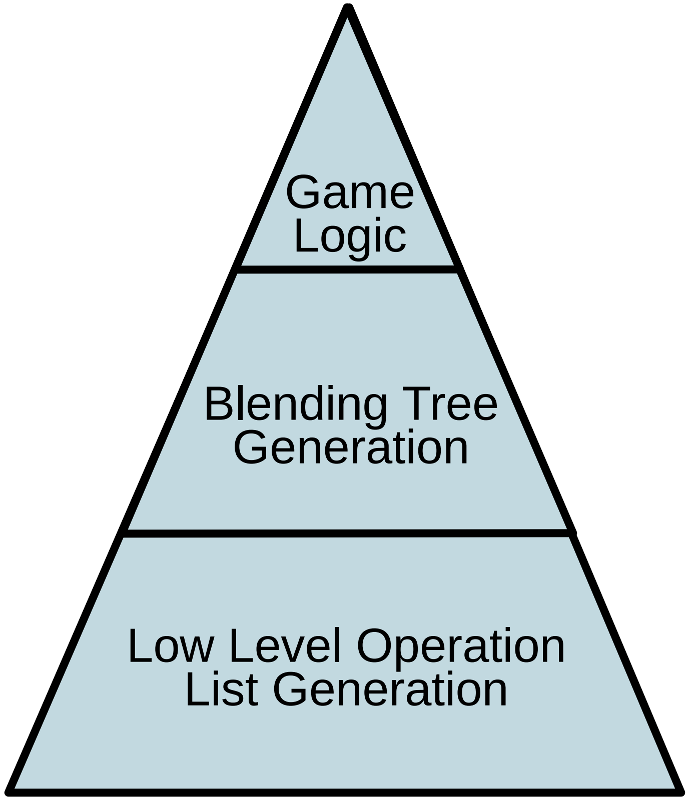


Game Logic

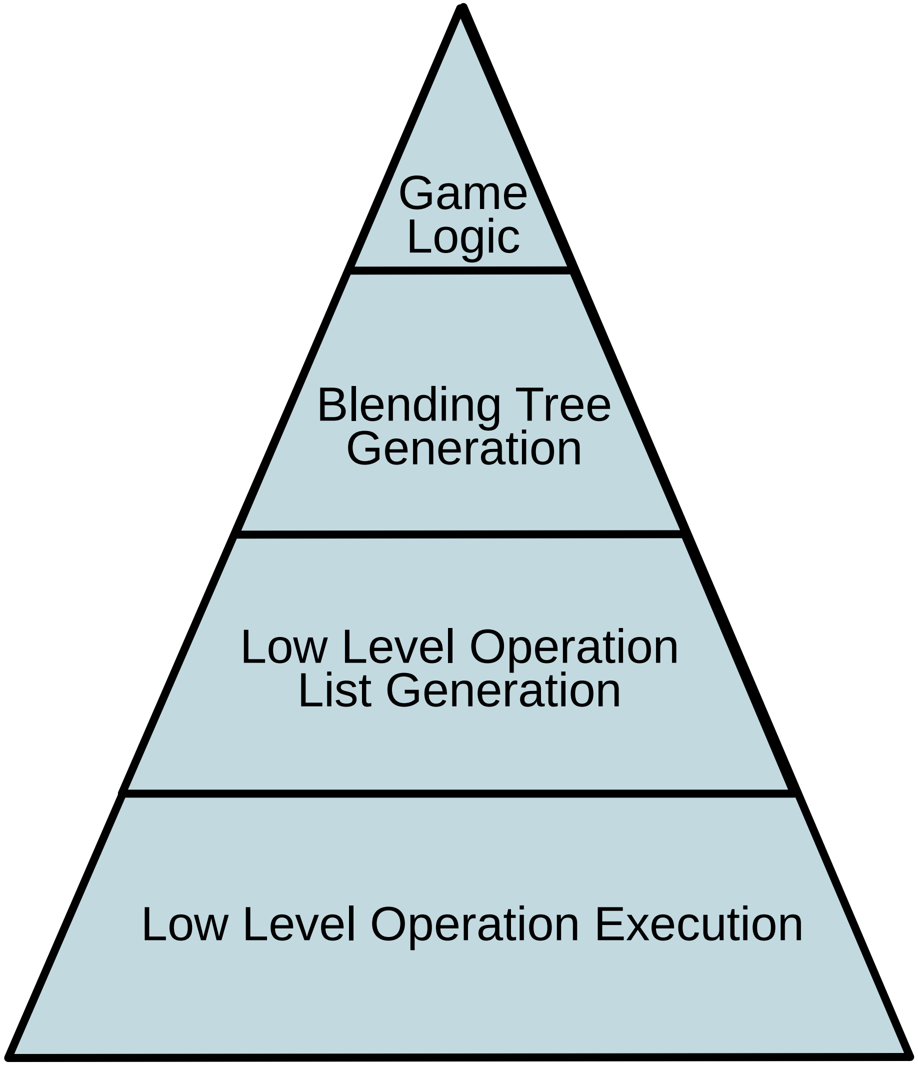
Animation Processing



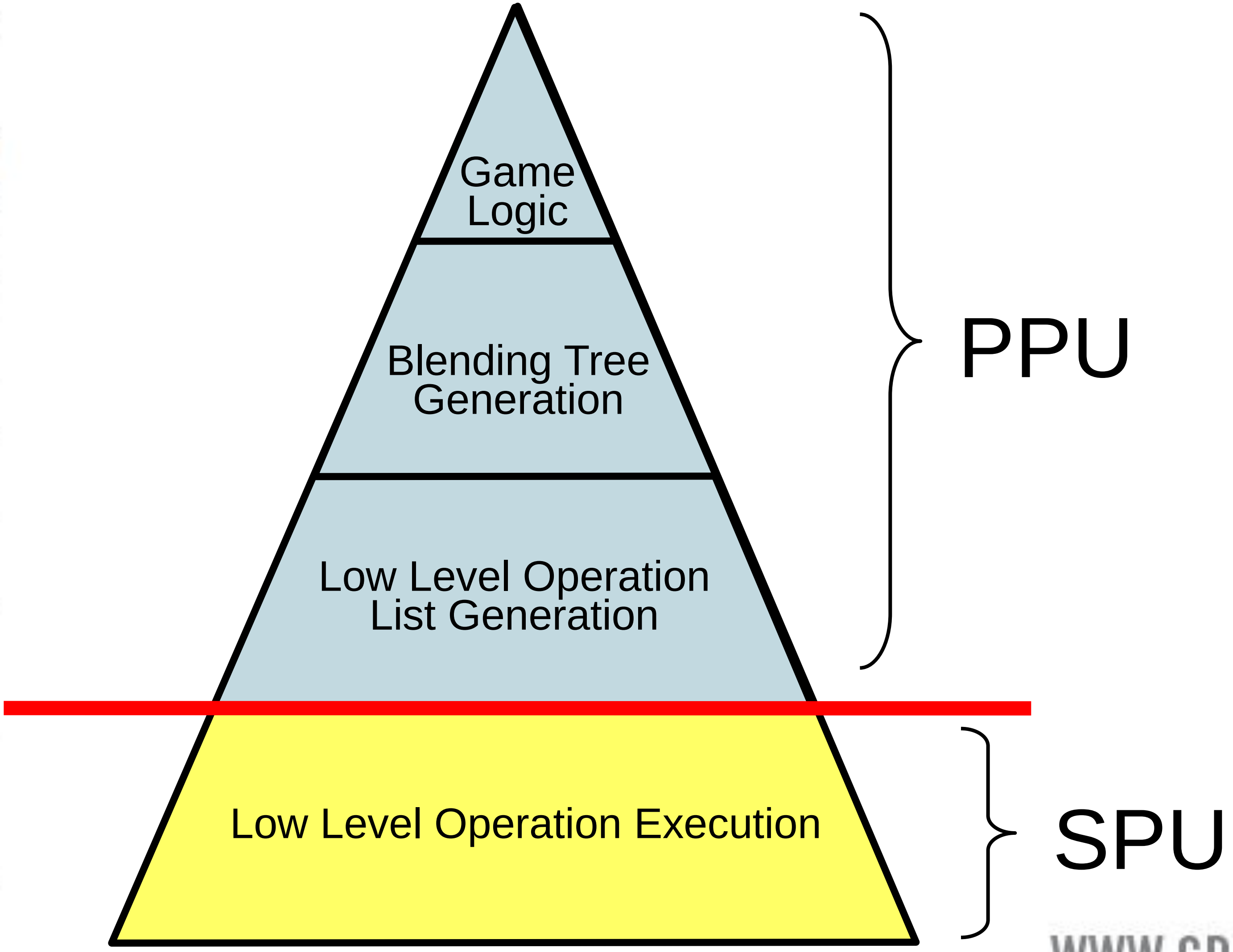
Animation Processing



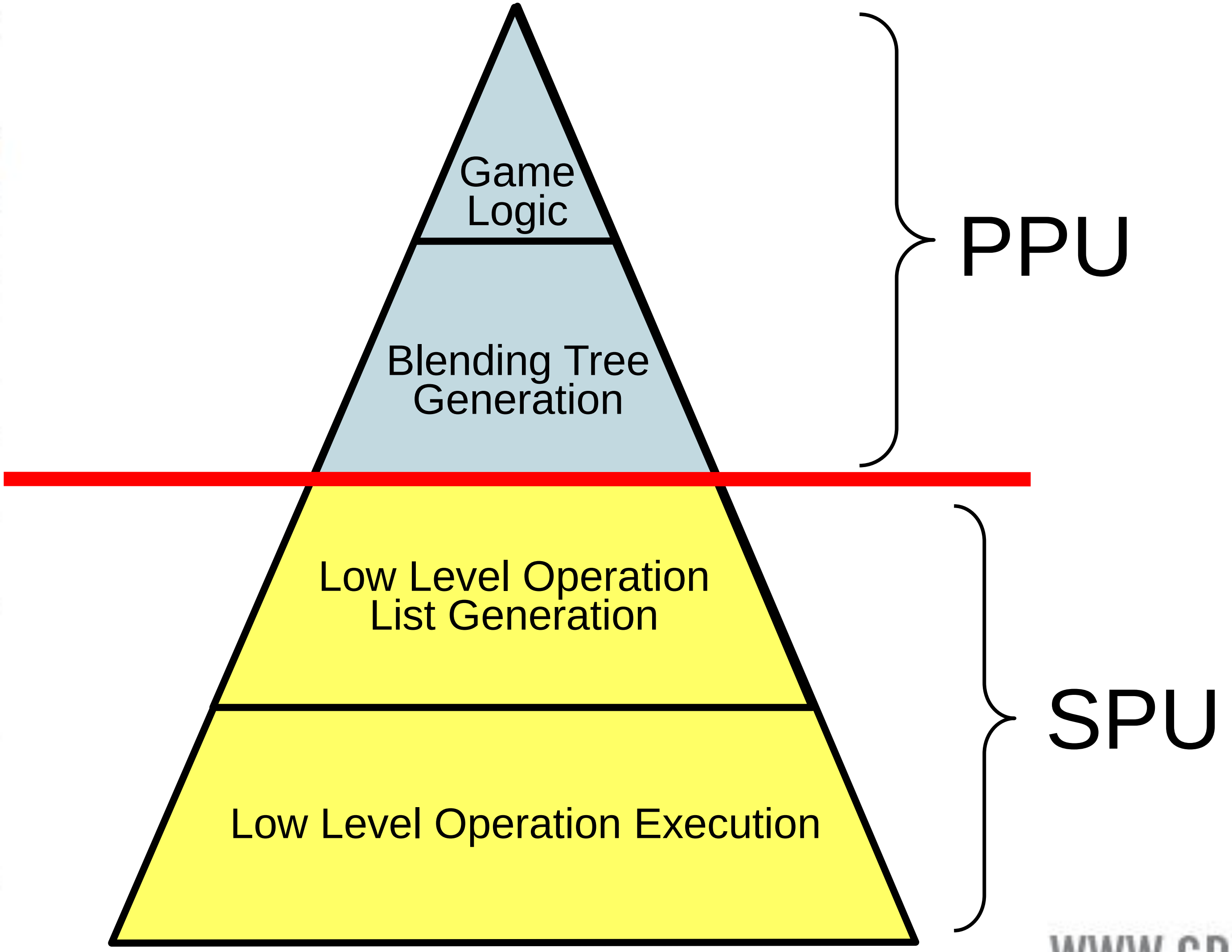
Animation Processing


Game Logic
Blending Tree Generation
Low Level Operation List Generation
Low Level Operation Execution

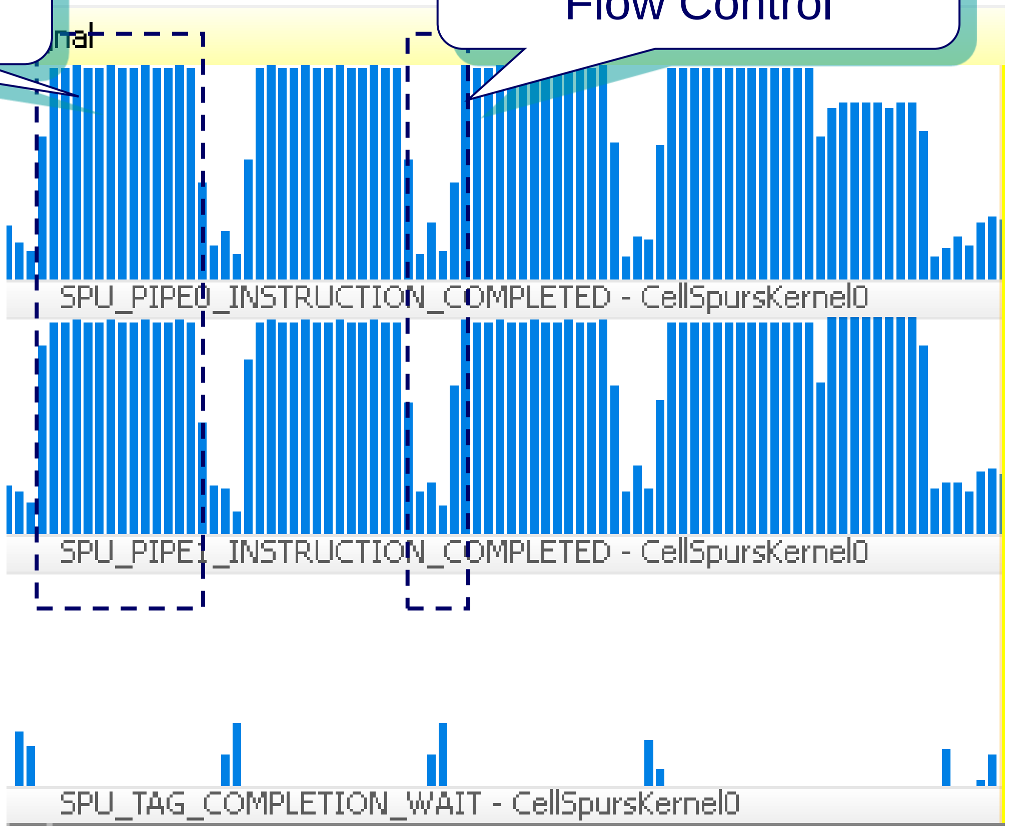
Animation Processing


Game Logic
Blending Tree Generation
Low Level Operation List Generation
Low Level Operation Execution
PPU
SPU

Animation Processing


Game Logic
Blending Tree Generation
Low Level Operation List Generation
Low Level Operation Execution
PPU
SPU

SPU Capture


nal
SPU_PIPE0_INSTRUCTION_COMPLETED - CellSpursKernel0
SPU_PIPE1_INSTRUCTION_COMPLETED - CellSpursKernel0
SPU_TAG_COMPLETION_WAIT - CellSpursKernel0

Low Level Functions

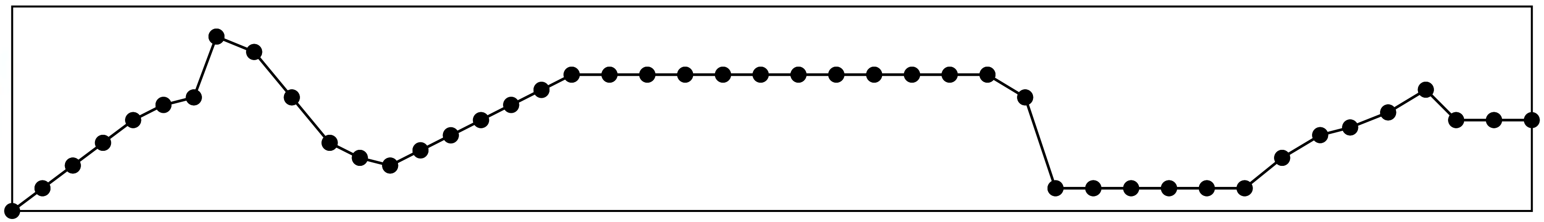
Blend Tree Parsing, Flow Control

no significant DMA stall

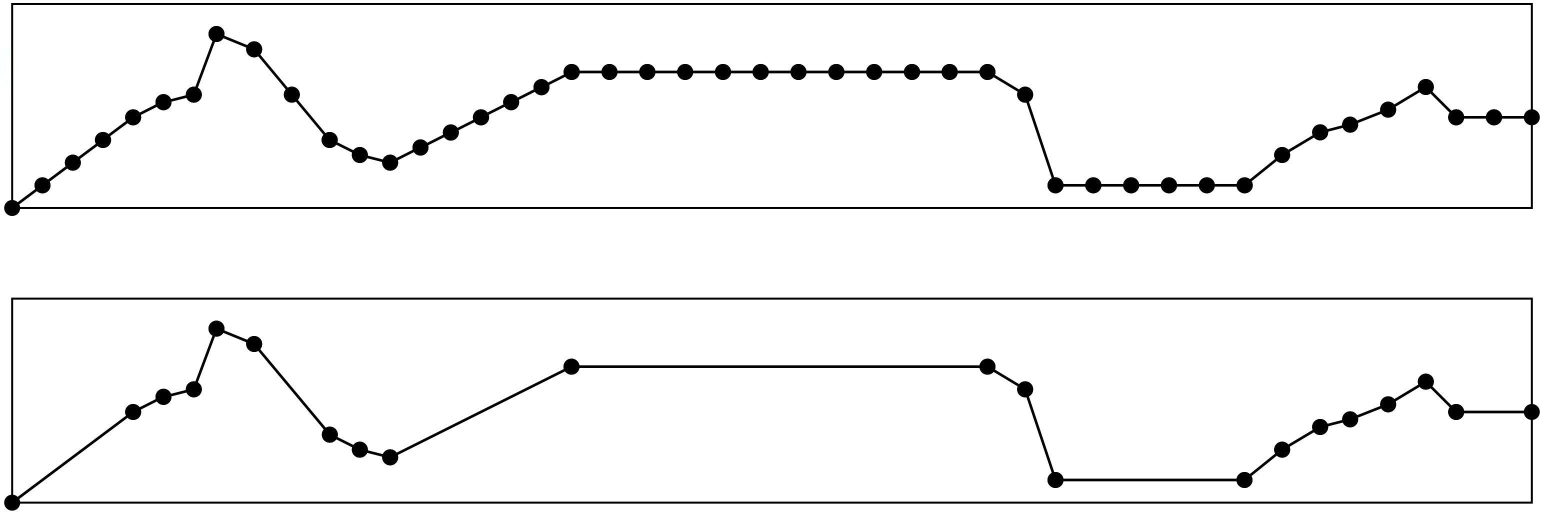
Additional Features

- Additive Blending
- Partial Animations
  - Per-joint weight
- Compression
  - Static joint parameters removed
  - Varying joint parameters expressed as sparse keyframes

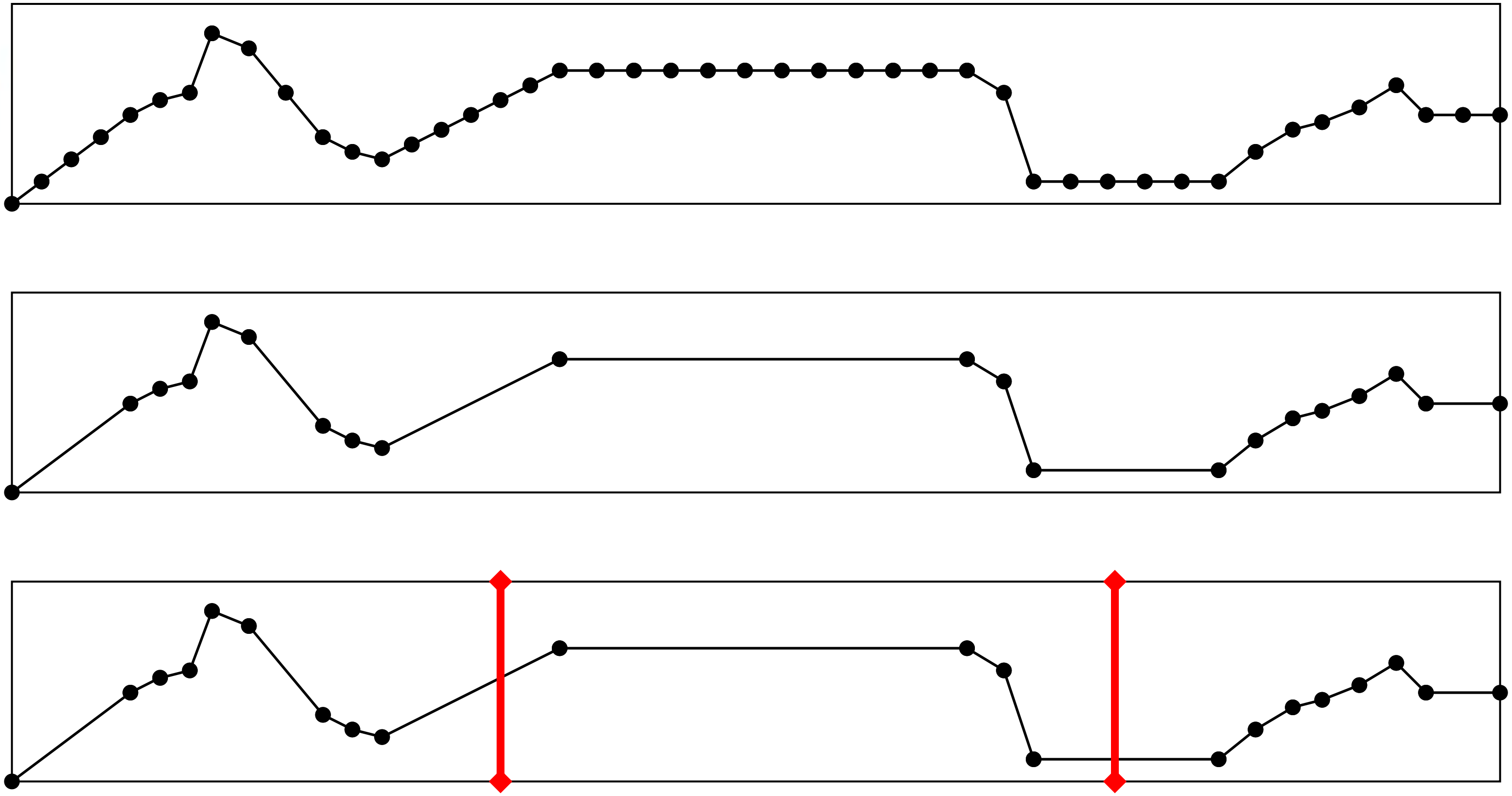
Varying parameter treatment



Varying parameter treatment



Varying parameter treatment



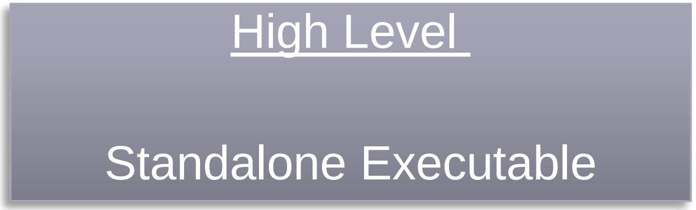
Offline Tools Layout

Offline Tools Layout

- Tools generate
  - Joint hierarchy
  - Compressed animation data

Offline Tools Layout

- Tools generate
  - Joint hierarchy
  - Compressed animation data

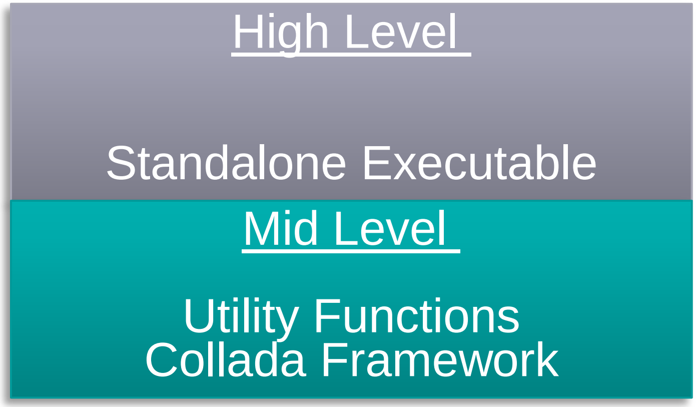


High Level

Standalone Executable

Offline Tools Layout

- Tools generate
  - Joint hierarchy
  - Compressed animation data



High Level

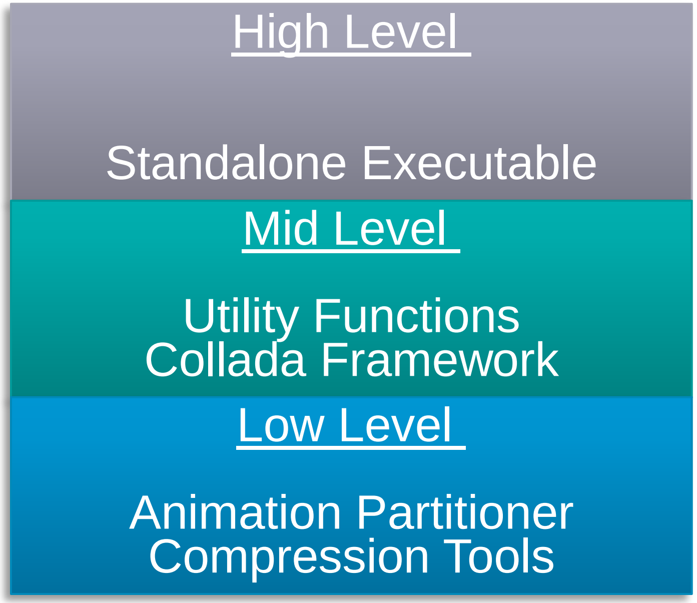
Standalone Executable

Mid Level

Utility Functions
Collada Framework

Offline Tools Layout

- Tools generate
  - Joint hierarchy
  - Compressed animation data



High Level

Standalone Executable

Mid Level

Utility Functions
Collada Framework

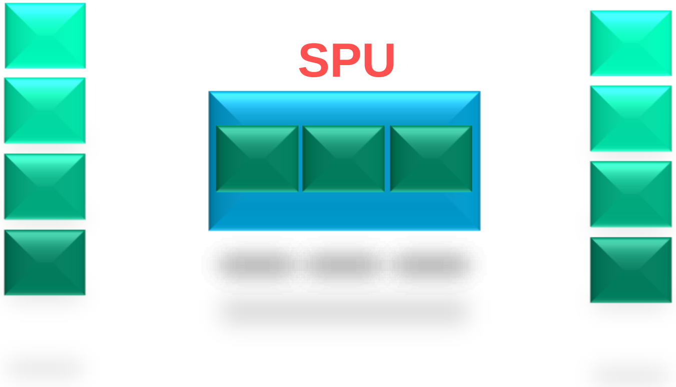
Low Level

Animation Partitioner
Compression Tools

PLAYSTATION®Edge Geometry

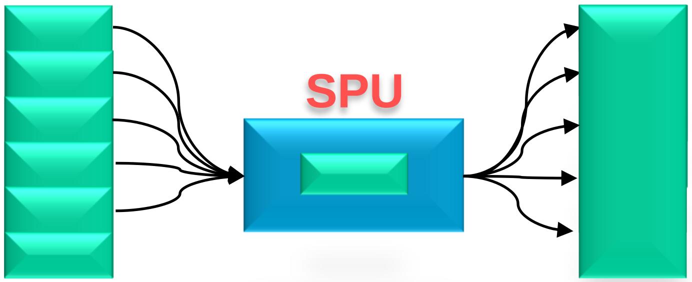
Two modes of usage

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

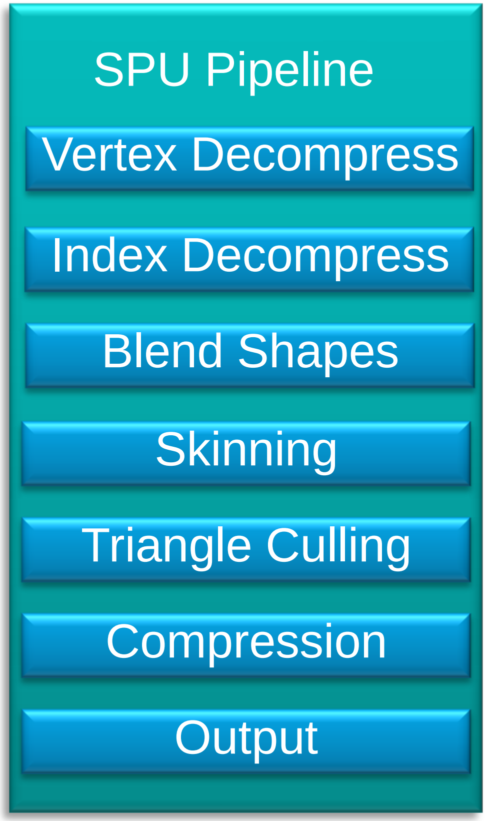


Two modes of usage (cont)

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



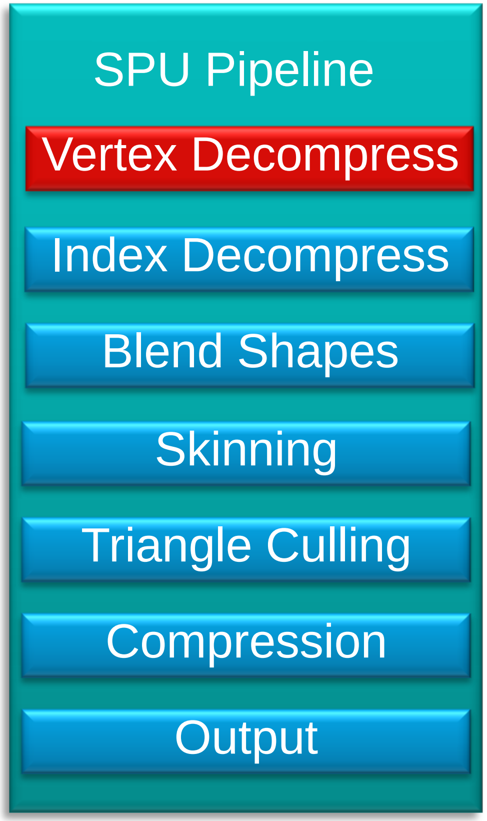
SPU Geometry Pipeline Stages



SPU Pipeline

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

Vertex Decompression


SPU Pipeline

Vertex Decompress

Index Decompress

Blend Shapes

Skinning

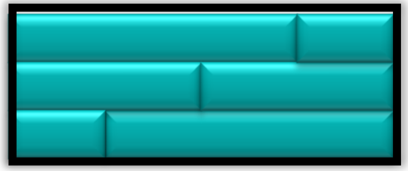
Triangle Culling

Compression

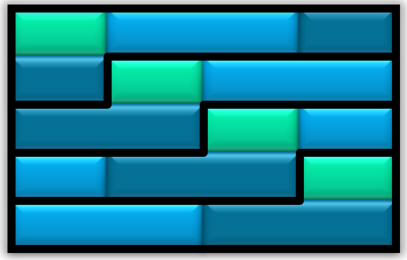
Output

Vertex attributes can be input into the SPUs in multiple arrays

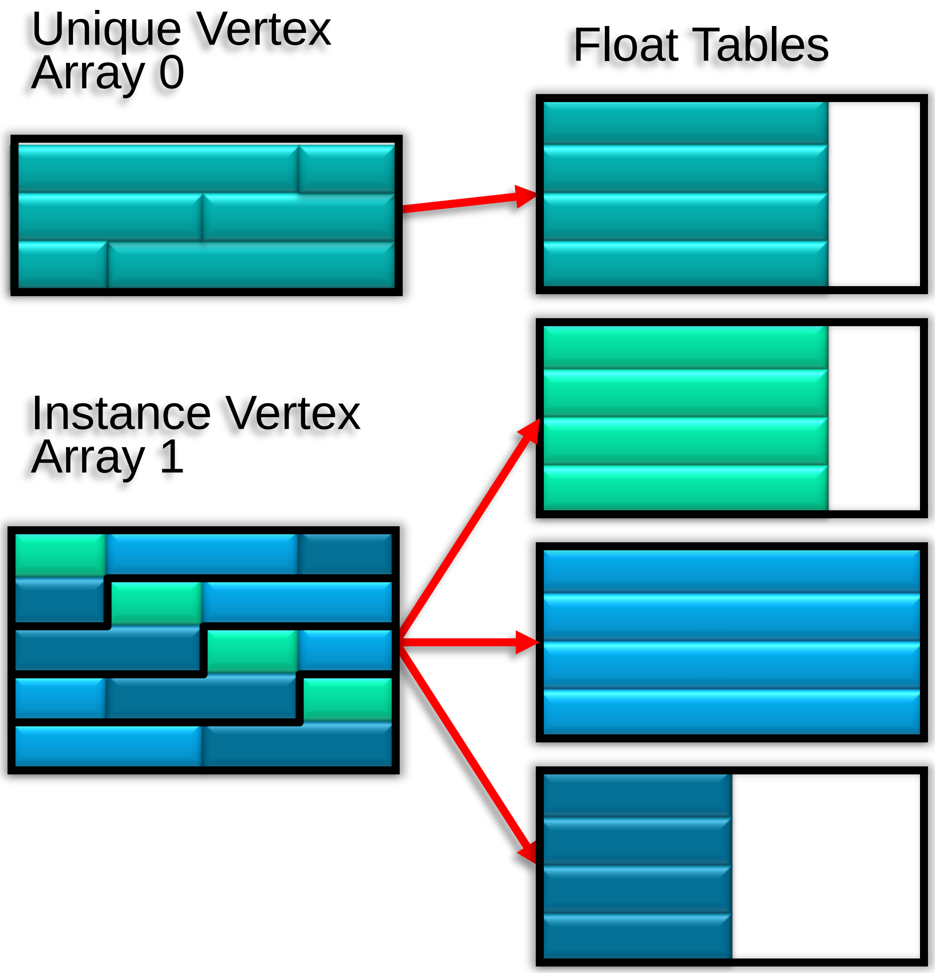
Unique Vertex Array 0



Instance Vertex Array 1



Vertex information is decompressed into tables of floats



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)

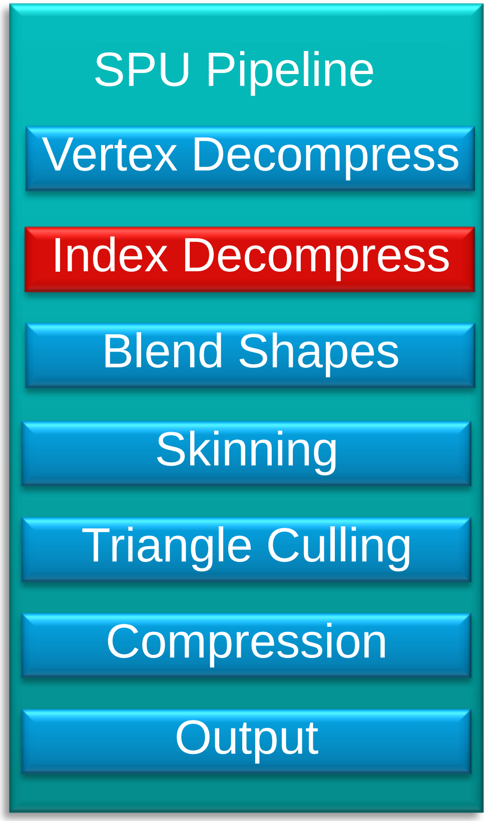
24bit Unit Vector

- 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 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

Triangle Culling

Compression

Output

Index Table Construction

- Index table is created by a vertex cache optimizer
  - Supplied in PlayStation 3 SDK
- First party research
  - Importance of mini-cache

RSX Best Practices

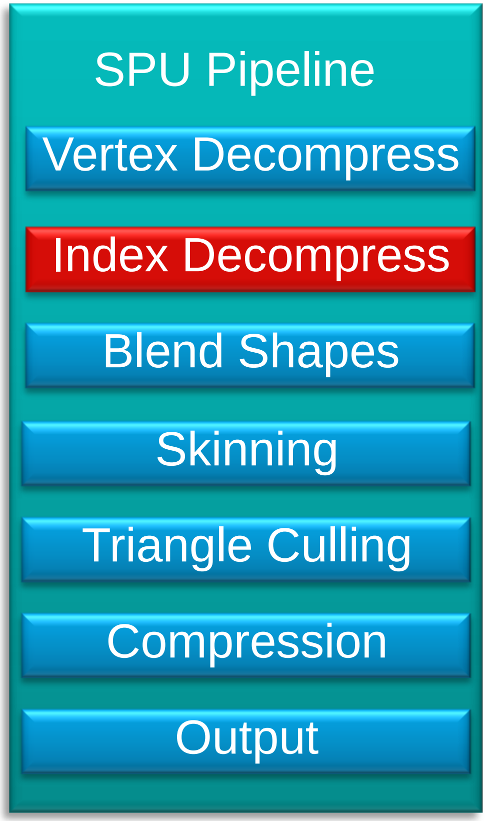
Thursday 2:30 pm – 3:30 pm

Room 3001, West Hall

Index Buffer Cache Optimizer

	Scene #1	Scene #2
Standard VCO	1495599 cycles	14292754 cycles
PLAYSTATION Edge	1449833 cycles	14058953 cycles
	(45766) 3.1%	(233801) 1.6%

Index Decompression



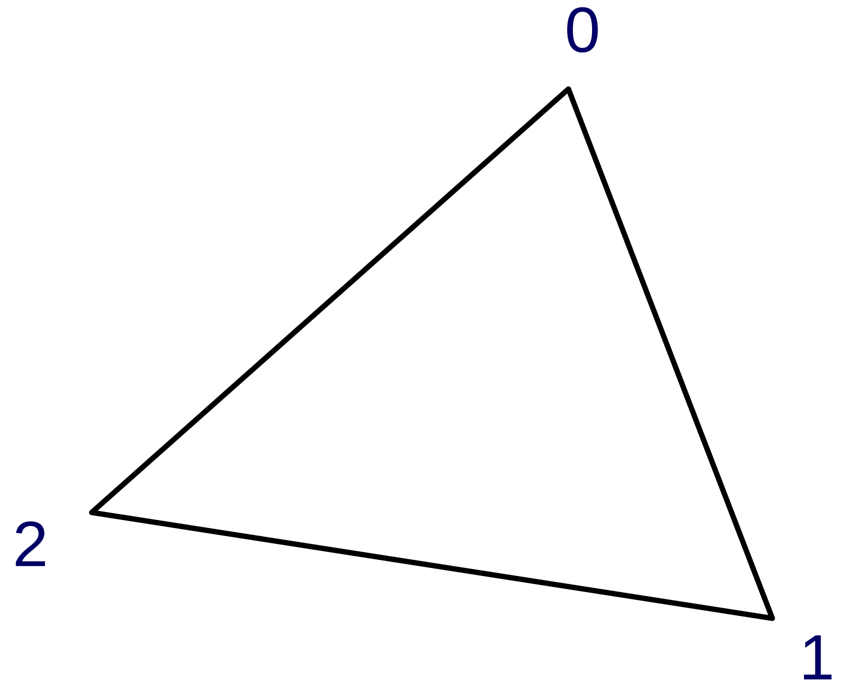
SPU Pipeline

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

Index Decompression

- Provided vertex cache optimizer produces very regular index data
    - Index patterns are easily compressed

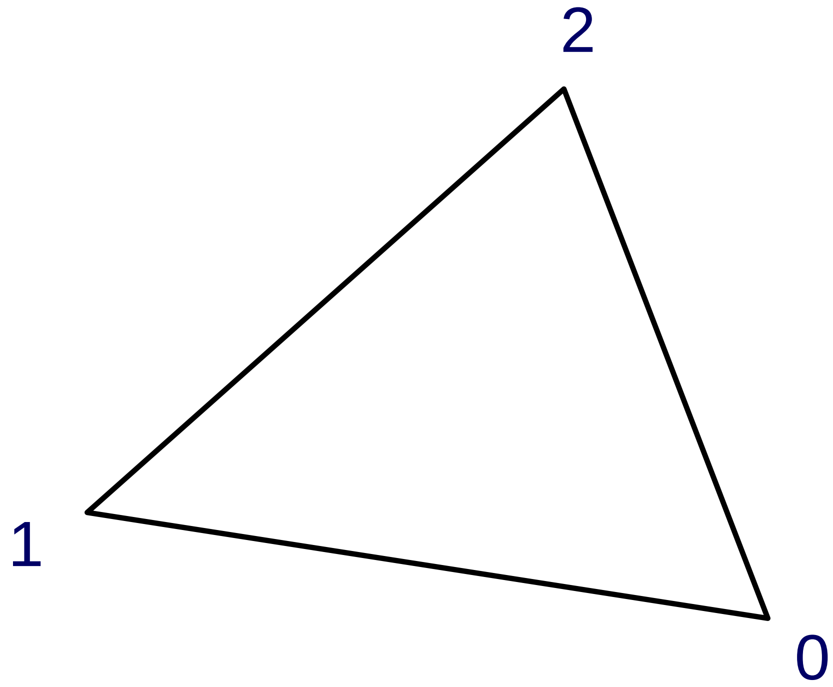
Index Decompression



Triangle Indexes

0	1	2

Index Decompression



Triangle Indexes

2	0	1

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

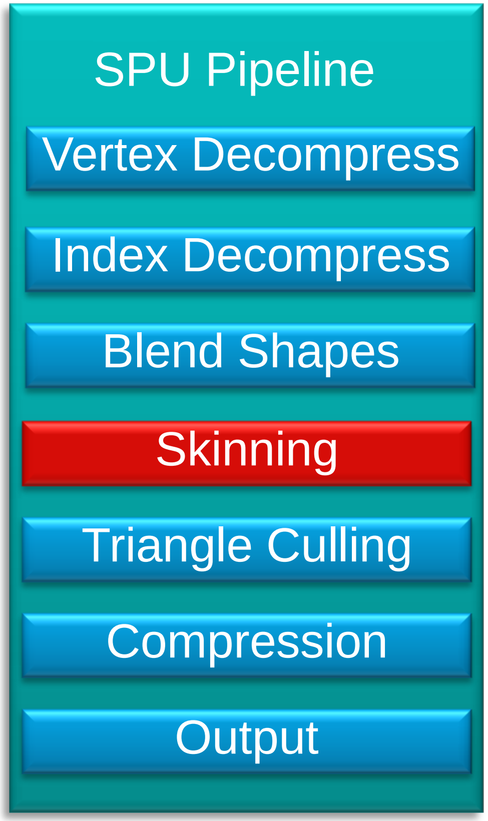
85% compression
6.5x more triangles

Blend Shapes

SPU Pipeline

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

Skinning



SPU Pipeline

Vertex Decompress

Index Decompress

Blend Shapes

Skinning

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

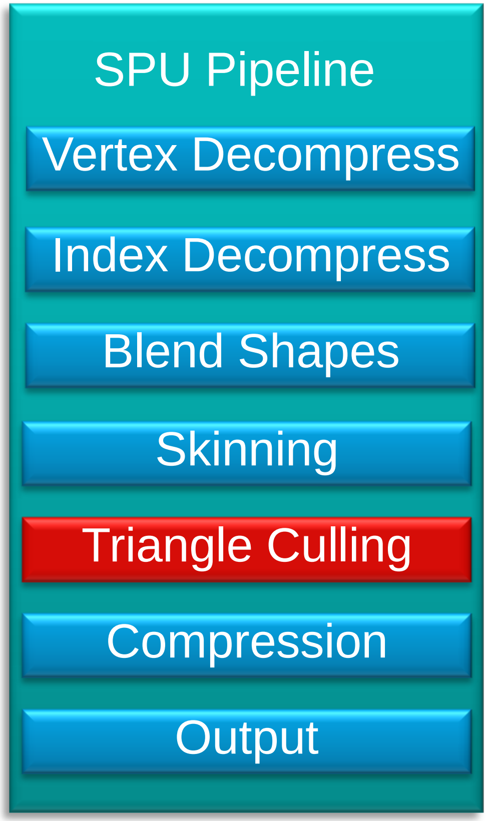
30% Performance Improvement

Skinning on SPUs

30% Performance Improvement

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

Triangle Culling



SPU Pipeline

Vertex Decompress

Index Decompress

Blend Shapes

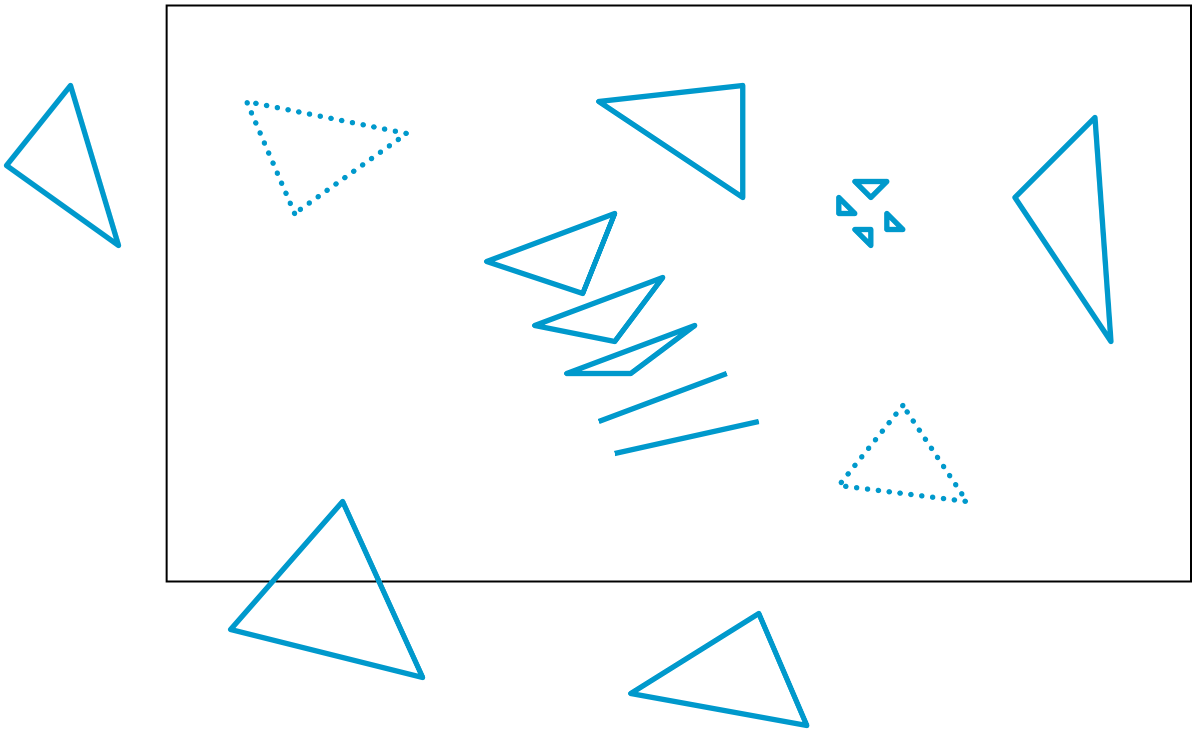
Skinning

Triangle Culling

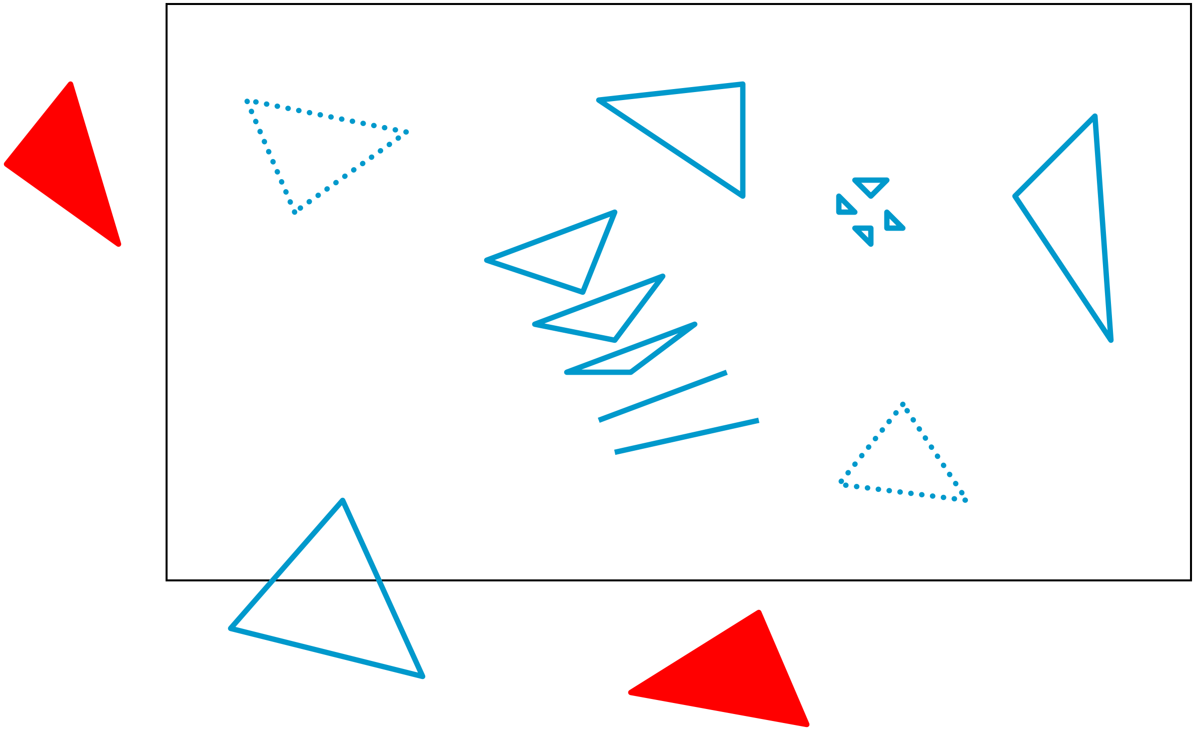
Compression

Output

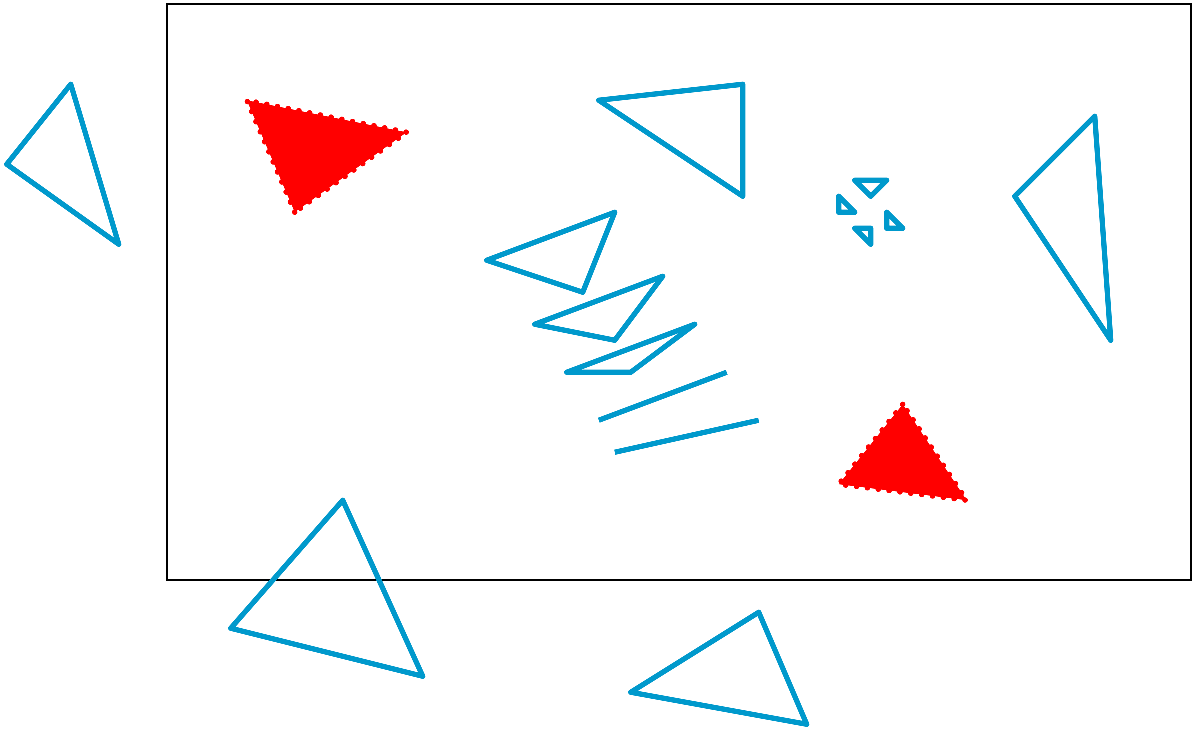
Up to ***70%*** of triangles do not contribute to final image.



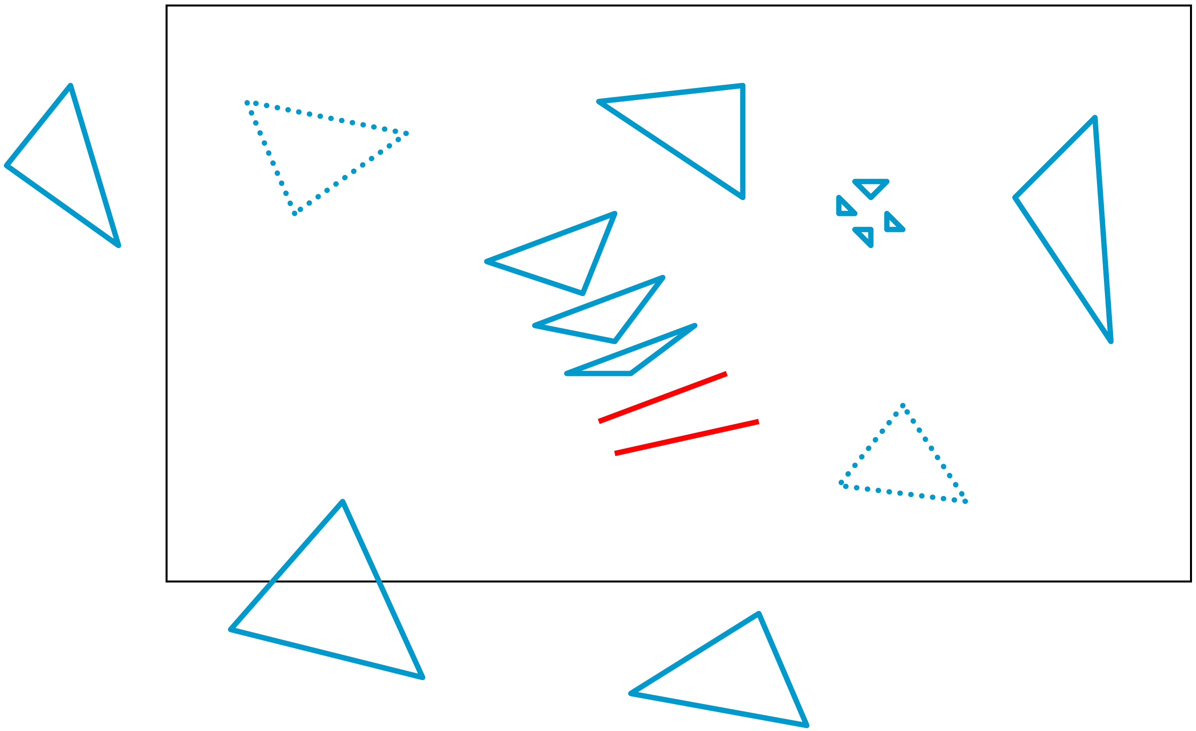
Off Screen Triangles



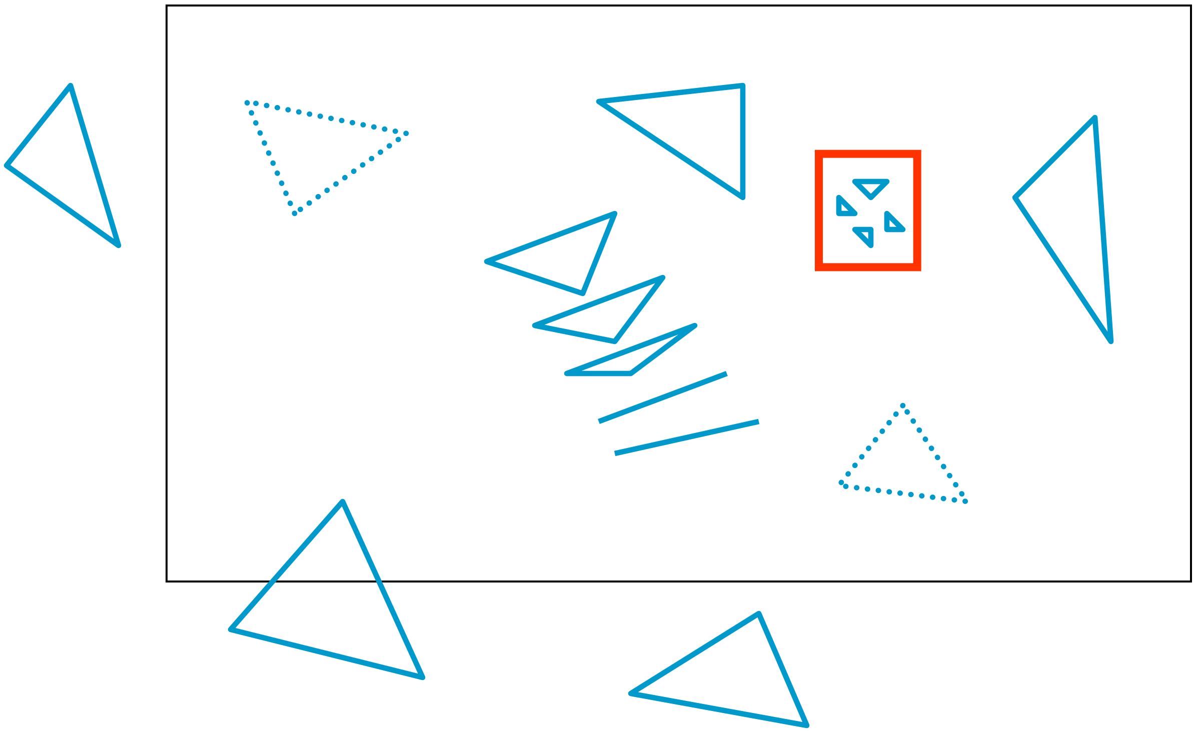
Back Facing Triangles



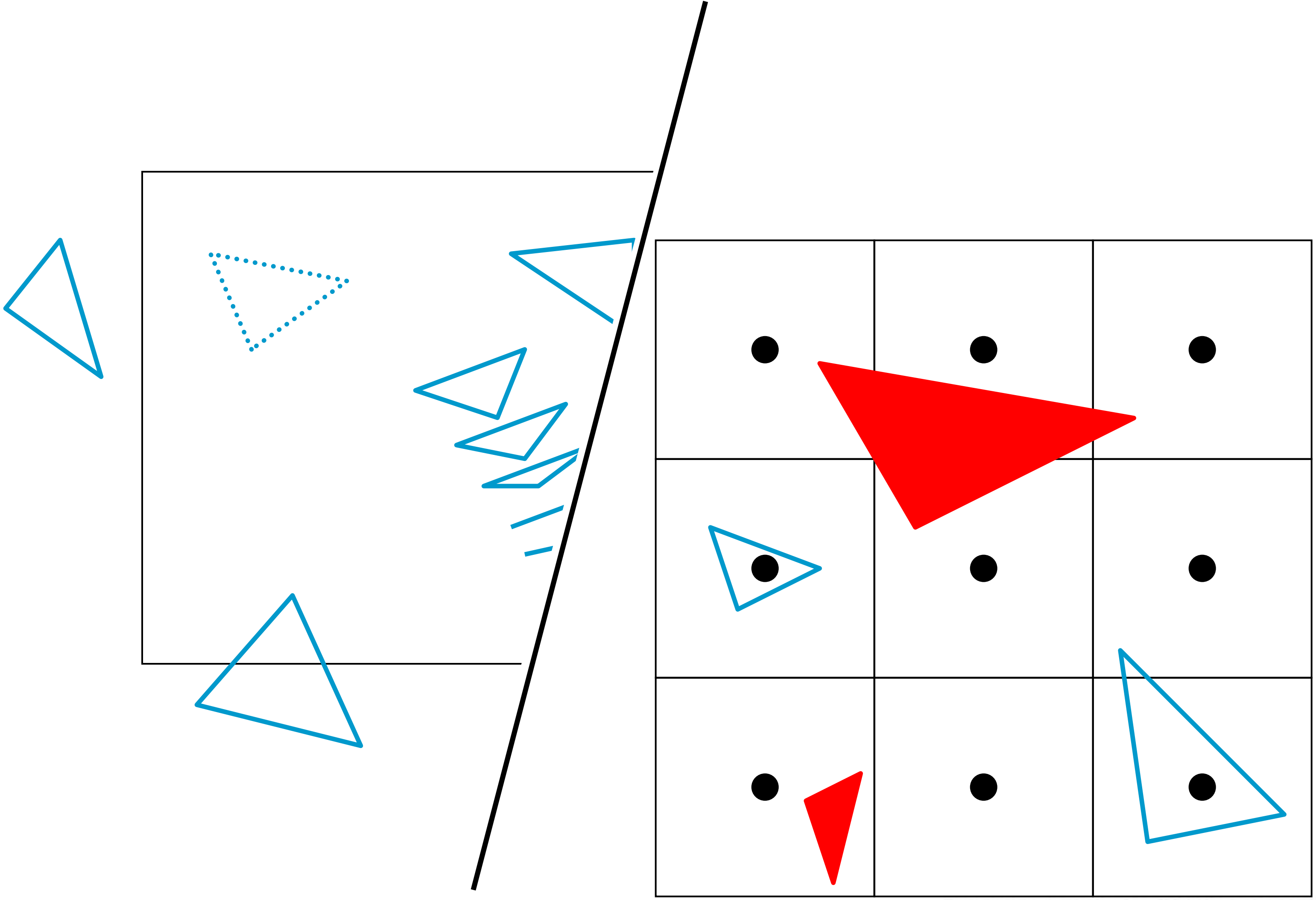
Zero Area Triangles



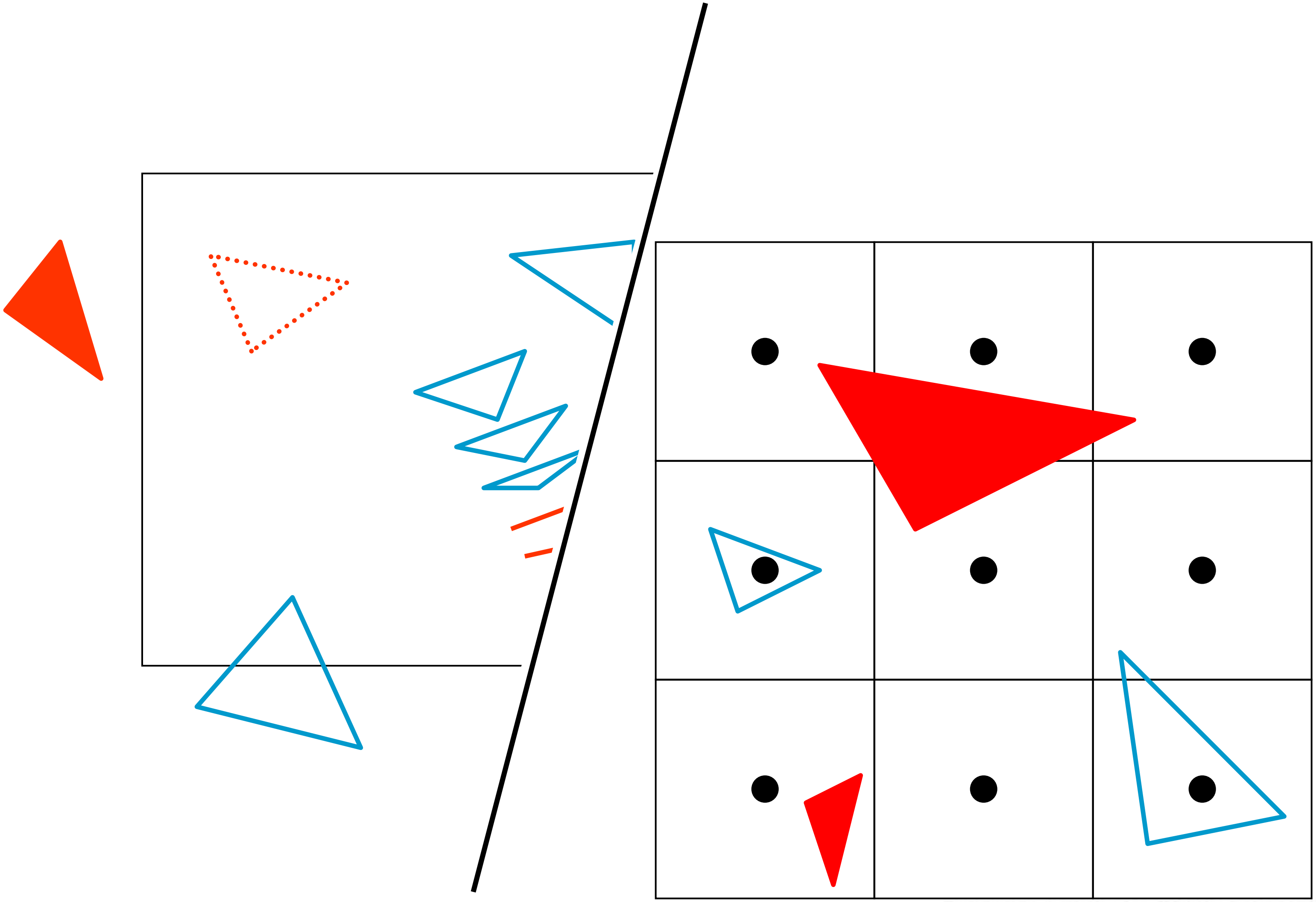
Zero Area Triangles



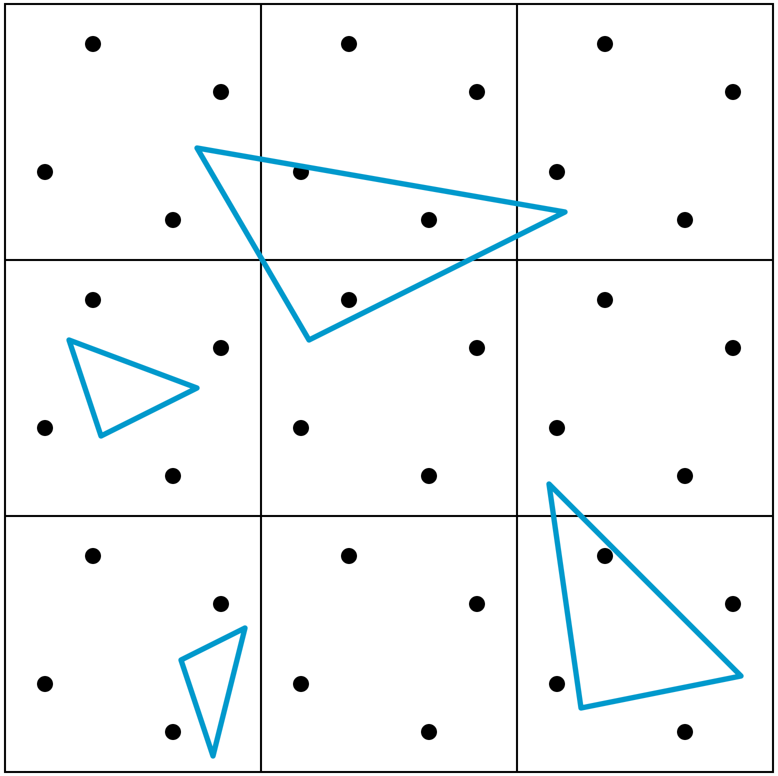
No Pixel Triangles



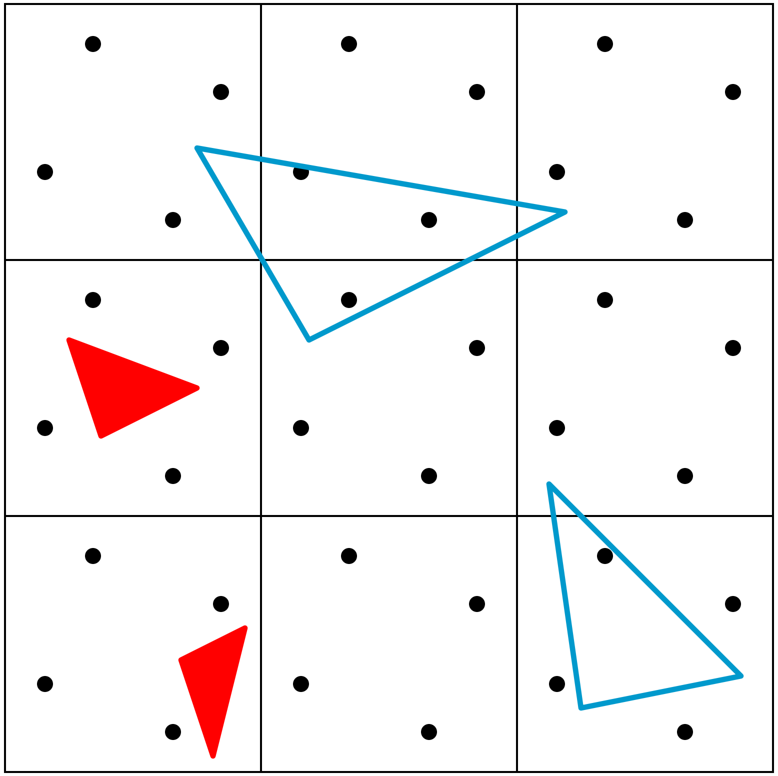
Triangle Culling



Multisampling adds some complications…



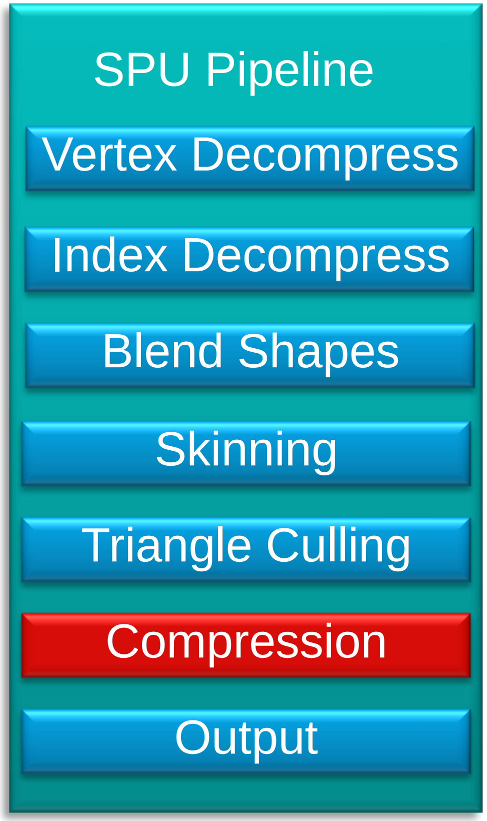
Culled



Triangle Culling

10% to 20%
Performance Improvement

Compression for Output



SPU Pipeline

Vertex Decompress

Index Decompress

Blend Shapes

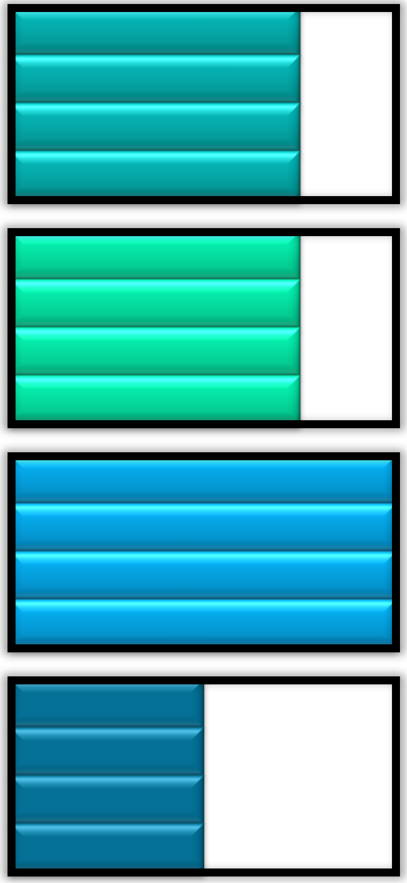
Skinning

Triangle Culling

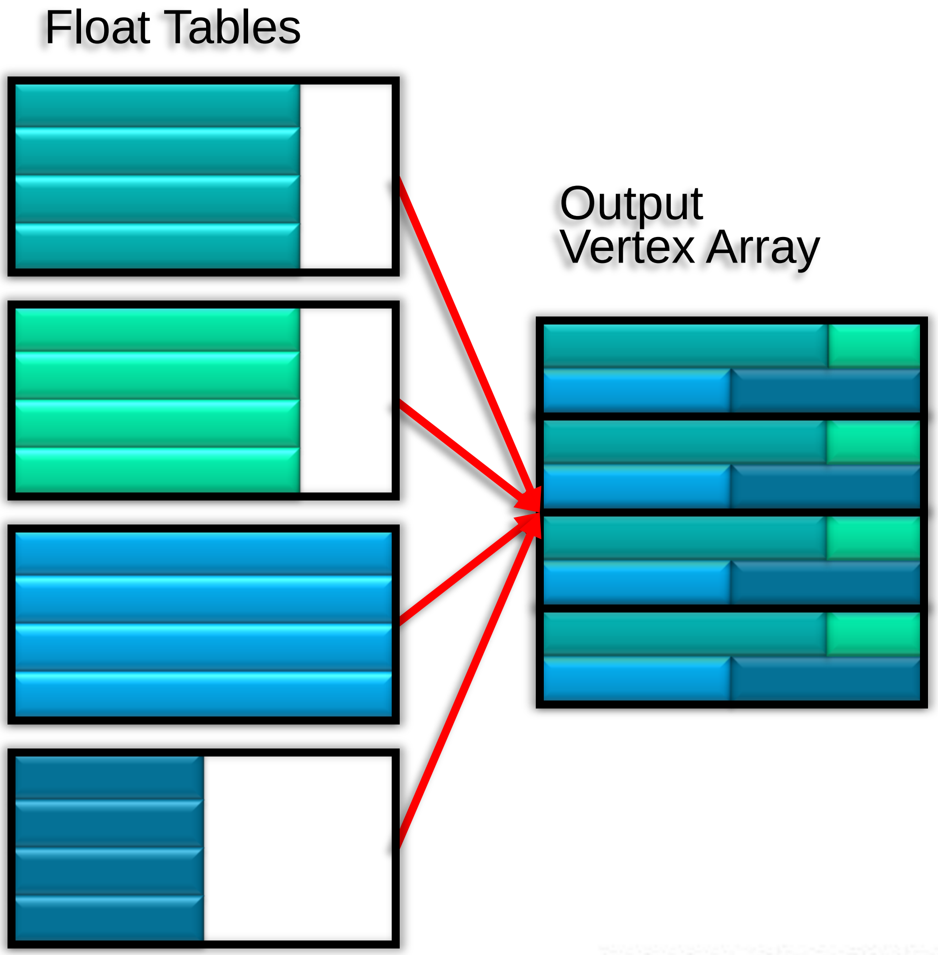
Compression

Output

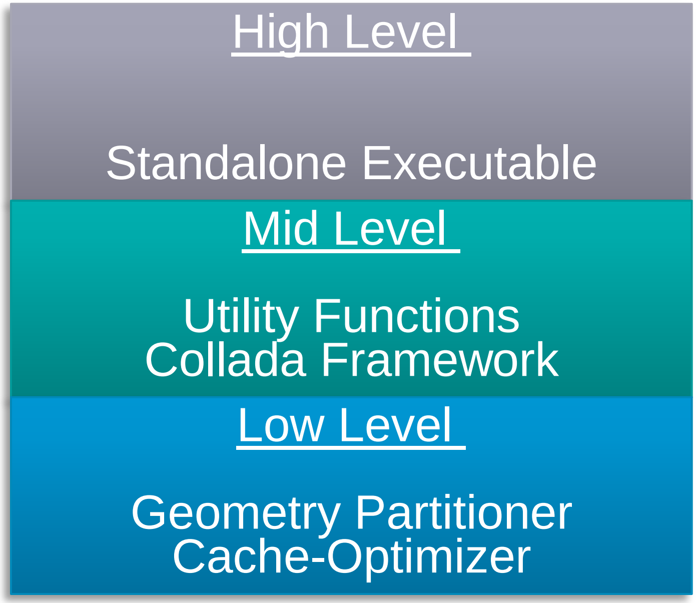
Float Tables



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



Offline Tools Layout



High Level

Standalone Executable

Mid Level

Utility Functions
Collada Framework

Low Level

Geometry Partitioner
Cache-Optimizer

Geometry Runtime Details

“Just in Time” Single Buffer Strategy

- SPUs generate data in same frame as RSX consumes it
- System tuned so that the RSX rarely waits on SPUs
- SPU ←→ RSX synchronization in place to handle rare cases

Geometry System Rendering Sequence

- On the PPU
  - Create a SPURS job
  - Place most RSX commands in the command buffer
  - Leave space in the RSX command buffer for the SPU to fill in later
- On the SPU
  - Process geometry
  - Write final commands to RSX command buffer

Synchronization Techniques

RSX ←→ SPU synchronization
by manipulation of put pointer

⇩

RSX ←→ SPU synchronization
through “local stalls”

RSX Best Practices
Thursday 2:30 pm – 3:30 pm
Room 3001, West Hall

```
void cellSpursJobMain(CellSpursJobContext* stInfo,
   CellSpursJob256 *job)
{
   edgeInitialize(...);
   edgeDecompressVertexes(...);
   edgeProcessBlendShapes(...);
   edgeSkinVertexes(...);
   edgeDecompressIndexes(...);
   edgeTransformVertexesForCull(...);
   edgeCullTriangles(...);
   if(!edgeAllocateOutputSpace(...))
         return;
   edgeOutputIndexes();
   edgeCompressVertexes();
   edgeOutputVertexes();
   edgeFillPushBufferHole(...);
}
```

```
void WaveVertexes(float *positions, unsigned numVertexes,
float t)
{
    for(unsigned i = 0; i < numVertexes; ++i)
        positions[i*4+0] +=
                 sinf(t + positions[i*4+0] +
                 positions[i*4+1]) * 10.f;
}
```

```
void cellSpursJobMain(CellSpursJobContext* stInfo,
   CellSpursJob256 *job)
{
   edgeInitialize(...);
   edgeDecompressVertexes(...);
   edgeProcessBlendShapes(...);
   edgeSkinVertexes(...);
   WaveVertexes(…);
   edgeDecompressIndexes(...);
   edgeTransformVertexesForCull(...);
   edgeCullTriangles(...);
   if(!edgeAllocateOutputSpace(...))
         return;
   edgeOutputIndexes();
   edgeCompressVertexes();
   edgeOutputVertexes();
   edgeFillPushBufferHole(...);
}
```

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);
```

20x faster than straight C/C++

```
EDGE_DECOMPRESS_INIT_GLOBAL(...);
EDGE_DECOMPRESS_INIT_F32(EDGE_ATTRIBUTE_USAGE_POSITION,...);
EDGE_DECOMPRESS_INIT_F16(EDGE_ATTRIBUTE_USAGE_GENERIC,...);
EDGE_DECOMPRESS_LOAD_COMMON();
do
{
    EDGE_DECOMPRESS_LOOP_START();
    EDGE_DECOMPRESS_LOOP_F32(...);
    EDGE_DECOMPRESS_LOOP_F16(...);
    EDGE_DECOMPRESS_LOOP_END();
} while (! EDGE_DECOMPRESS_LOOP_DONE());
EDGE_DECOMPRESS_FINALIZE_F32(...);
EDGE_DECOMPRESS_FINALIZE_F16(...);
```

PLAYSTATION®Edge 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

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>

PLAYSTATION®Edge

Beta Release

MARCH 2007

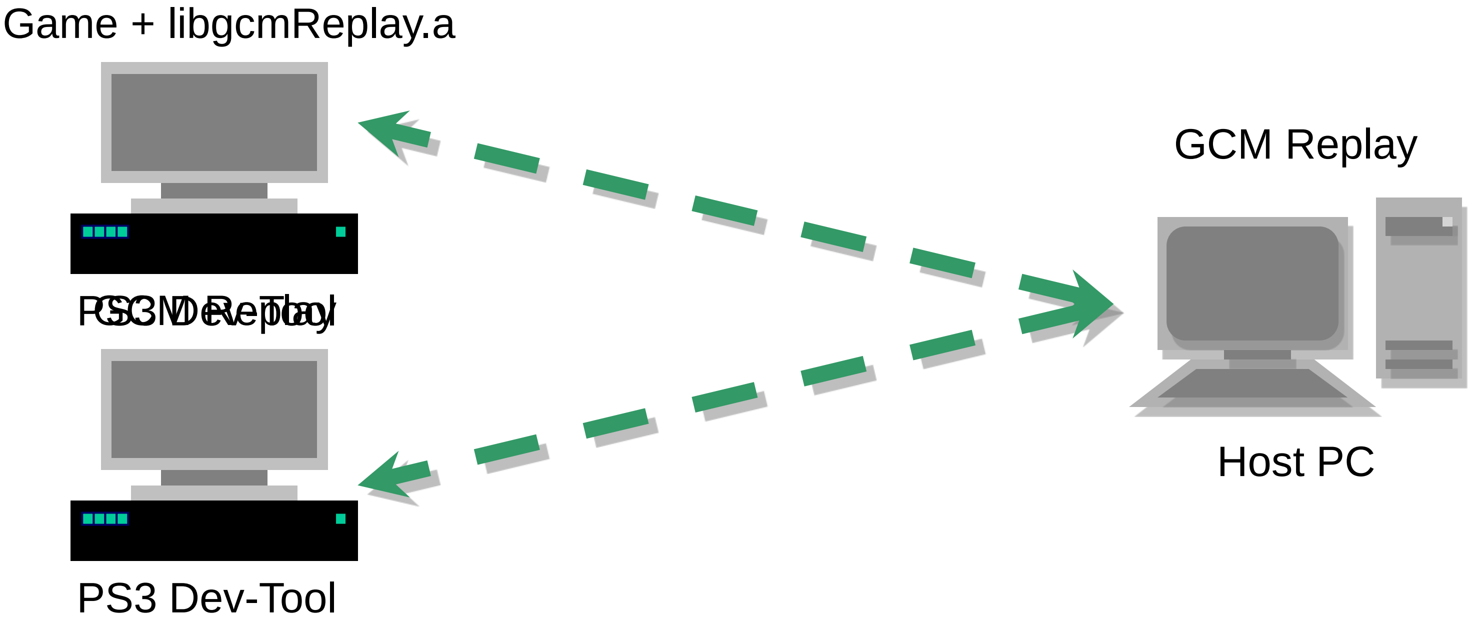
GCM Replay

SCE World Wide Studios

GCM Replay

- GCM Replay is a new tool for RSX
  - Analysis
  - Debugging
  - Profiling

GCM Replay - Overview



- GCM Replay consists of two parts
  - Small PS3 runtime library
  - Main Application - runs on a Windows PC + PS3 Dev-Tool

GCM Replay - Overview

- Uses RSX rather than simulation
- Supports highly detailed analysis
  - Far greater than a typical real-time profiler would allow
  - Supporting scene-wide analysis
  - To the analysis of individual draw calls, vertices and pixels
- Focus on off-line performance analysis features
  - Many of which have never been available before

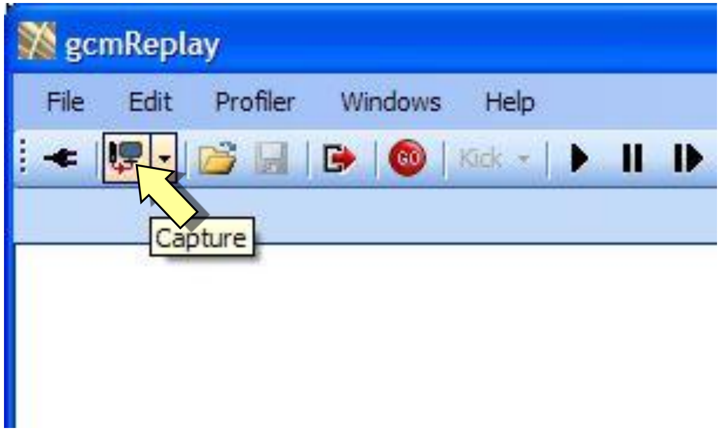
GCM Replay

Workflow and Behind the Scenes

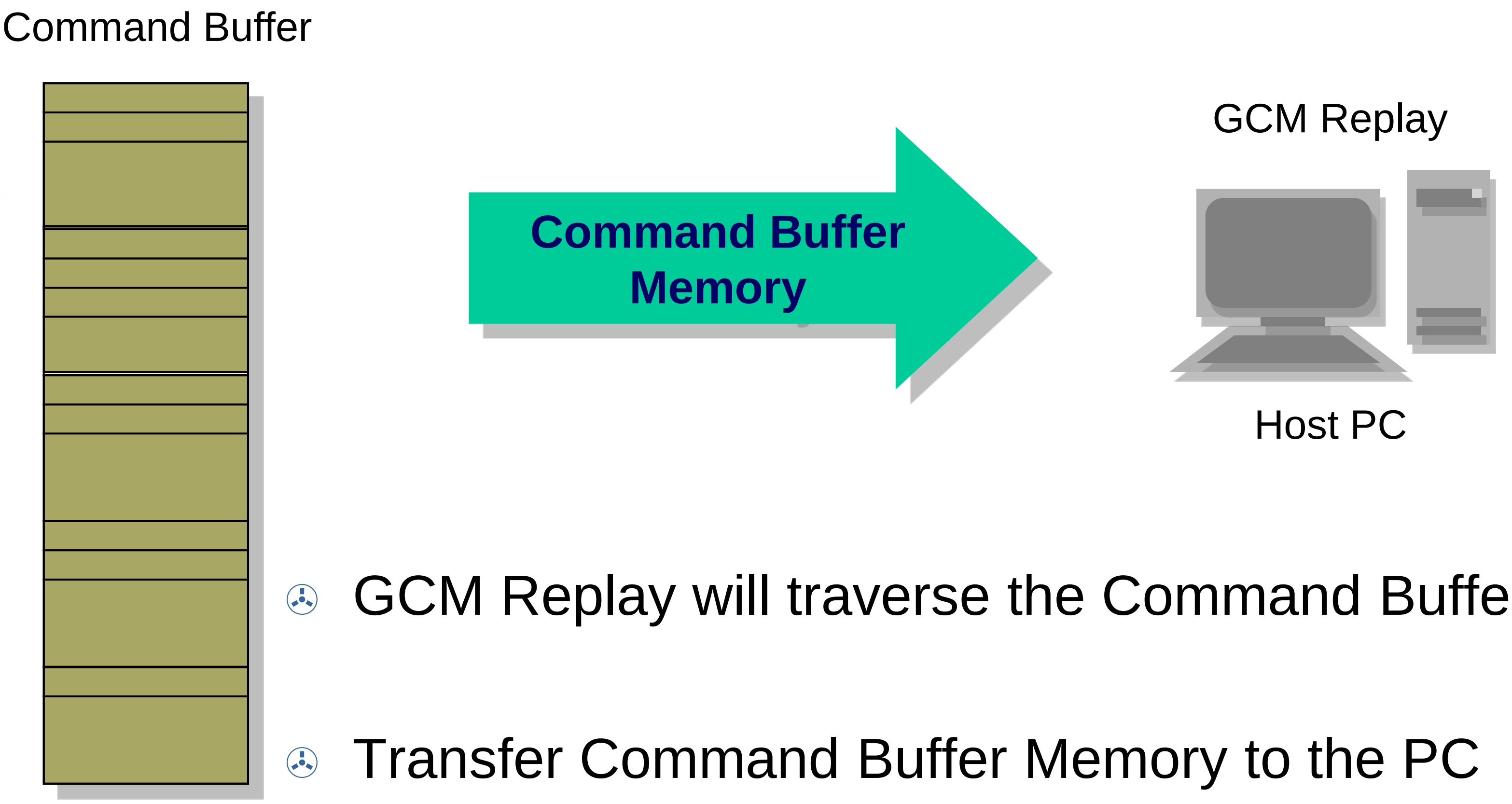
GCM Replay - Overview

- Run your game with the GCM Replay runtime linked in
- Once you reach a point of interest…

Hit Capture

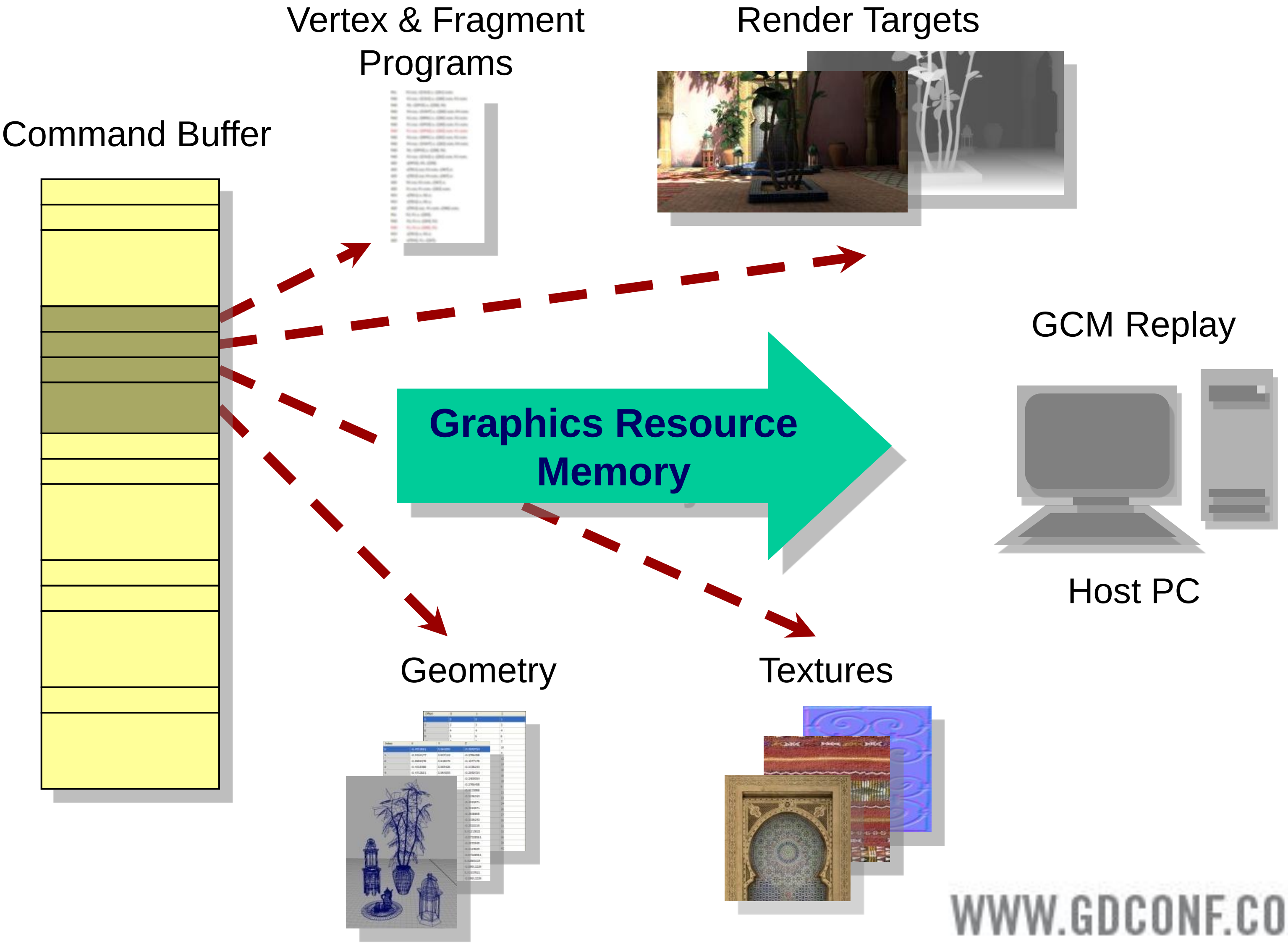


Capturing the Command Buffer



- GCM Replay will traverse the Command Buffer
- Transfer Command Buffer Memory to the PC
- Each command is analysed

Capturing the Command Buffer



Vertex & Fragment Programs

Render Targets

Command Buffer

GCM Replay

Graphics Resource Memory

Host PC

Geometry

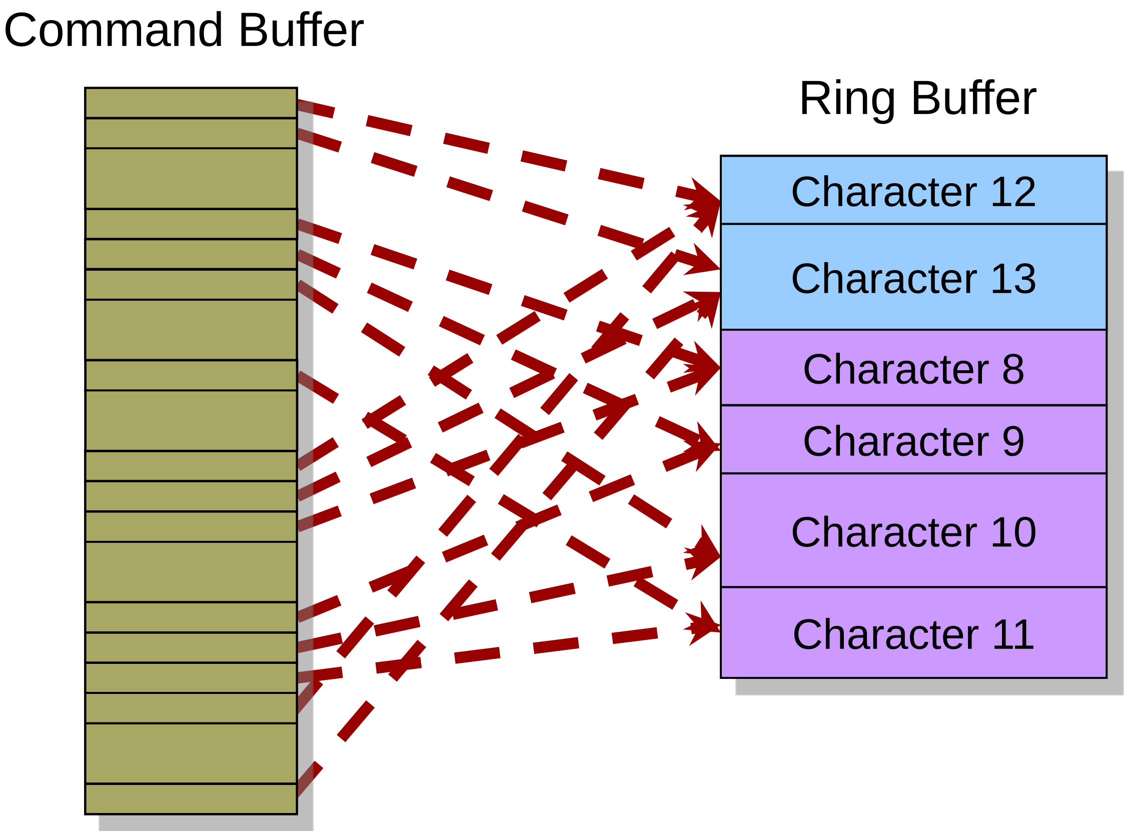
Textures

Capturing the Command Buffer

- Once the process is complete - GCM Replay has all the data it needs to

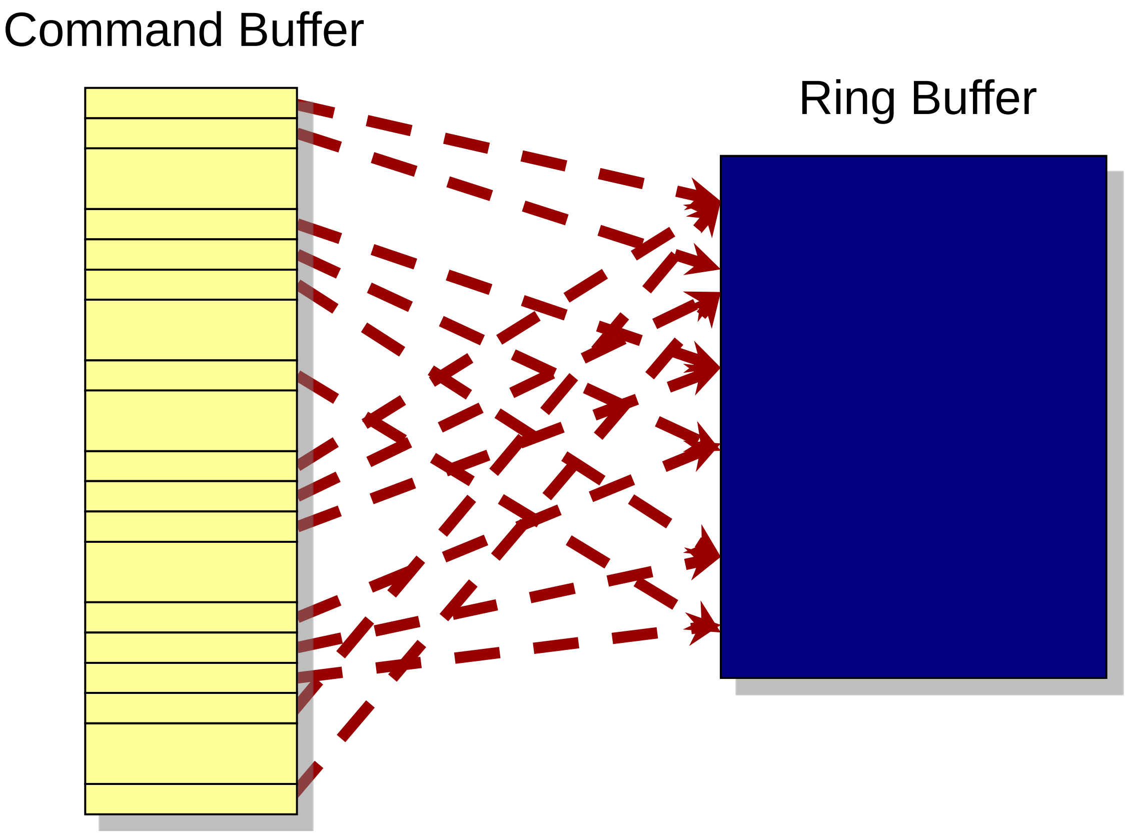
REPLAY your Command Buffer

Capturing the Command Buffer



- Capturing the Command Buffer can be very complex

Capturing the Command Buffer



- All RSX usage models can be captured with GCM Replay

GCM Replay Integration

- Only takes a few minutes
- At initialisation

```
// Initialise the capture API
cellGcmReplay::Network::Init();
cellGcmReplay::Capture::Init();
```

GCM Replay Integration

- Then every frame…

```
// Call a single heartbeat function
cellGcmReplay::Heartbeat(&yourContext);
```

- Add optional annotations

```
// Useful for adding semantics
cellGcmReplay::InsertDebugString(“Bloom Pass”);
```

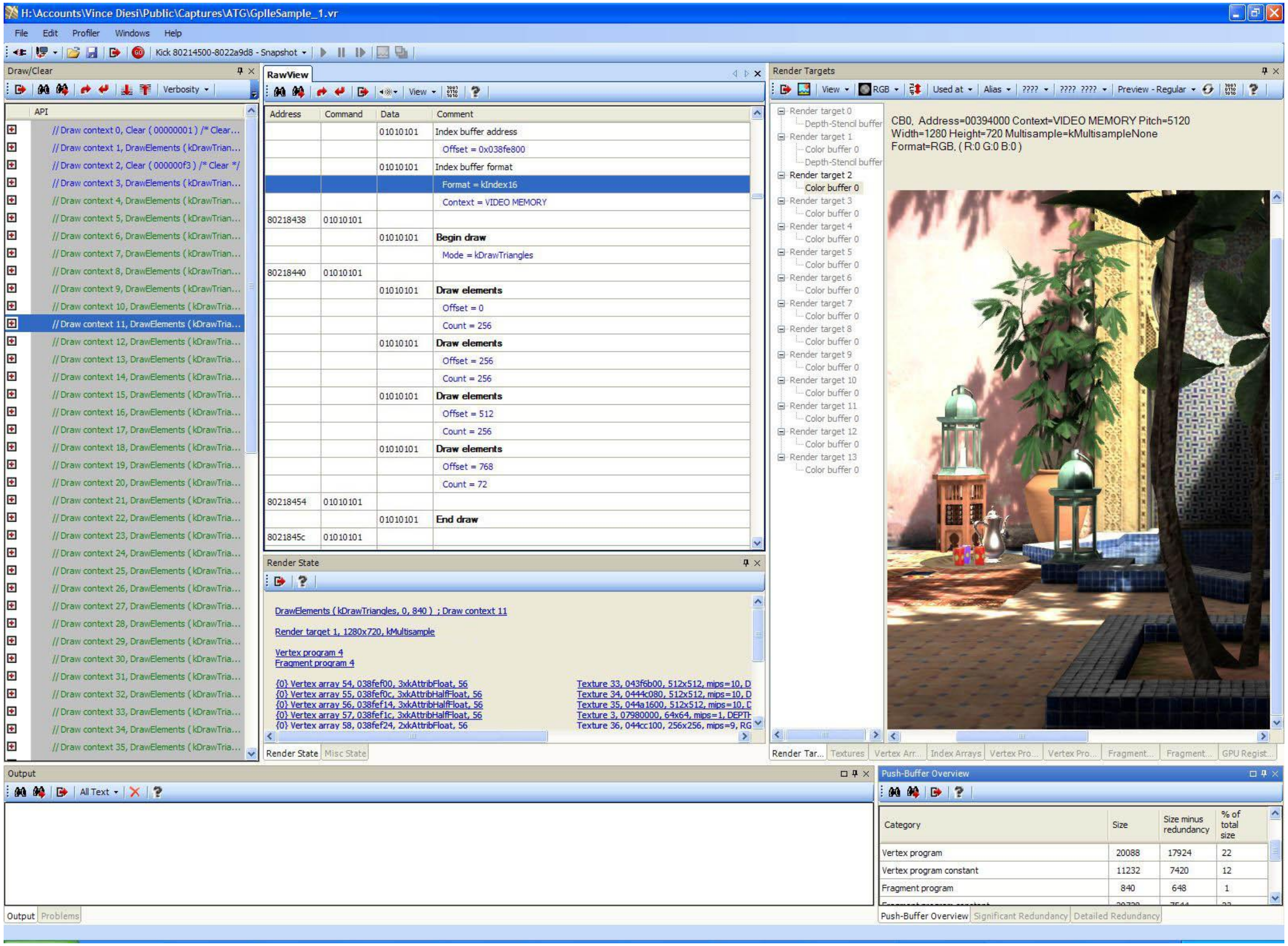
GCM Replay Captures

Analyse it immediately

OR

Save for analysis later

GCM Replay Application



API

// Draw context 0, Clear (00000001) /* Clear...
// Draw context 1, DrawElements (kDrawTrian...
// Draw context 2, Clear (000000f3) /* Clear */
// Draw context 3, DrawElements (kDrawTrian...
// Draw context 4, DrawElements (kDrawTrian...
// Draw context 5, DrawElements (kDrawTrian...
// Draw context 6, DrawElements (kDrawTrian...
// Draw context 7, DrawElements (kDrawTrian...
// Draw context 8, DrawElements (kDrawTrian...
// Draw context 9, DrawElements (kDrawTrian...
// Draw context 10, DrawElements (kDrawTria...
// Draw context 11, DrawElements (kDrawTria...
// Draw context 12, DrawElements (kDrawTria...
// Draw context 13, DrawElements (kDrawTria...
// Draw context 14, DrawElements (kDrawTria...
// Draw context 15, DrawElements (kDrawTria...
// Draw context 16, DrawElements (kDrawTria...
// Draw context 17, DrawElements (kDrawTria...
// Draw context 18, DrawElements (kDrawTria...
// Draw context 19, DrawElements (kDrawTria...
// Draw context 20, DrawElements (kDrawTria...
// Draw context 21, DrawElements (kDrawTria...
// Draw context 22, DrawElements (kDrawTria...
// Draw context 23, DrawElements (kDrawTria...
// Draw context 24, DrawElements (kDrawTria...
// Draw context 25, DrawElements (kDrawTria...
// Draw context 26, DrawElements (kDrawTria...
// Draw context 27, DrawElements (kDrawTria...
// Draw context 28, DrawElements (kDrawTria...
// Draw context 29, DrawElements (kDrawTria...
// Draw context 30, DrawElements (kDrawTria...
// Draw context 31, DrawElements (kDrawTria...
// Draw context 32, DrawElements (kDrawTria...
// Draw context 33, DrawElements (kDrawTria...
// Draw context 34, DrawElements (kDrawTria...
// Draw context 35, DrawElements (kDrawTria...

Address	Command	Data	Comment
		01010101	Index buffer address
			Offset = 0x038fe800
		01010101	Index buffer format
			Format = kIndex16
			Context = VIDEO MEMORY
80218438	01010101		
		01010101	**Begin draw**
			Mode = kDrawTriangles
80218440	01010101		
		01010101	**Draw elements**
			Offset = 0
			Count = 256
		01010101	**Draw elements**
			Offset = 256
			Count = 256
		01010101	**Draw elements**
			Offset = 512
			Count = 256
		01010101	**Draw elements**
			Offset = 768
			Count = 72
80218454	01010101		
		01010101	**End draw**
8021845c	01010101		

Render State

DrawElements (kDrawTriangles, 0, 840) ; Draw context 11

Render target 1, 1280x720, kMultisample

Vertex program 4
Fragment program 4

{0} Vertex array 54, 038fef00, 3xkAttribFloat, 56
{0} Vertex array 55, 038fef0c, 3xkAttribHalfFloat, 56
{0} Vertex array 56, 038fef14, 3xkAttribHalfFloat, 56
{0} Vertex array 57, 038fef1c, 3xkAttribHalfFloat, 56
{0} Vertex array 58, 038fef24, 2xkAttribFloat, 56

Texture 33, 043f6b00, 512x512, mips=10, D
Texture 34, 0444c080, 512x512, mips=10, D
Texture 35, 044a1600, 512x512, mips=10, D
Texture 3, 07980000, 64x64, mips=1, DEPTH
Texture 36, 044cc100, 256x256, mips=9, RG

Render target 0
Depth-Stencil buffer
Render target 1
Color buffer 0
Depth-Stencil buffer
Render target 2
Color buffer 0
Render target 3
Color buffer 0
Render target 4
Color buffer 0
Render target 5
Color buffer 0
Render target 6
Color buffer 0
Render target 7
Color buffer 0
Render target 8
Color buffer 0
Render target 9
Color buffer 0
Render target 10
Color buffer 0
Render target 11
Color buffer 0
Render target 12
Color buffer 0
Render target 13
Color buffer 0

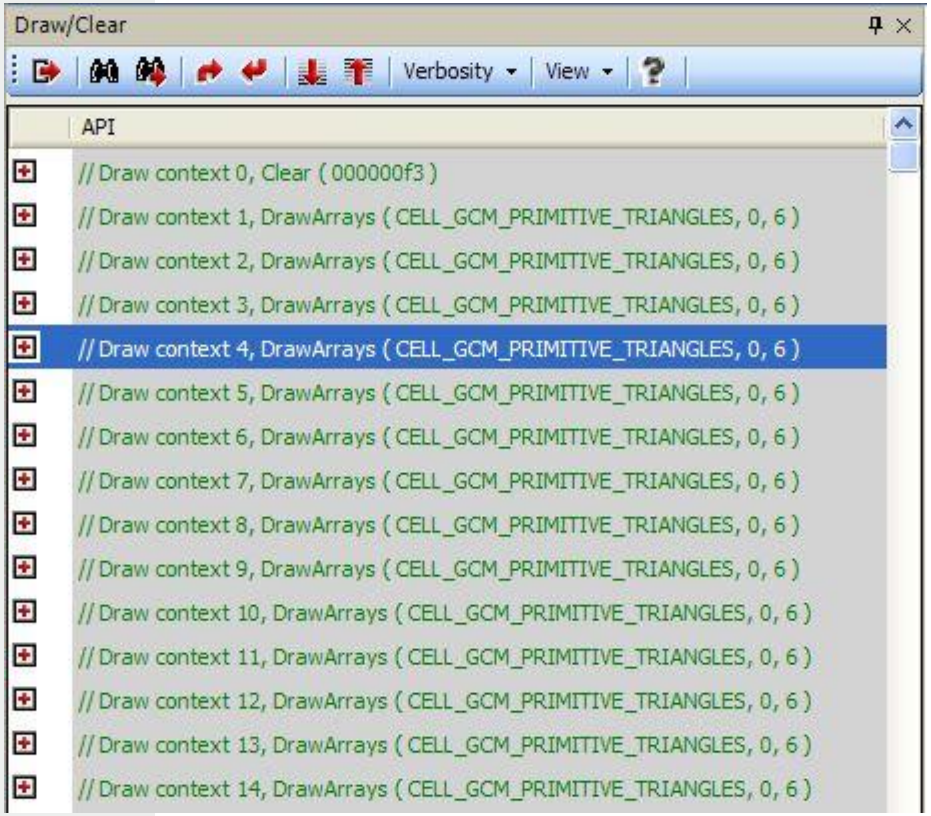
CB0, Address=00394000 Context=VIDEO MEMORY Pitch=5120
Width=1280 Height=720 Multisample=kMultisampleNone
Format=RGB, (R:0 G:0 B:0)

Category	Size	Size minus redundancy	% of total size
Vertex program	20088	17924	22
Vertex program constant	11232	7420	12
Fragment program	840	648	1

GCM Replay

Analysis and Debugging Views

Draw Context View

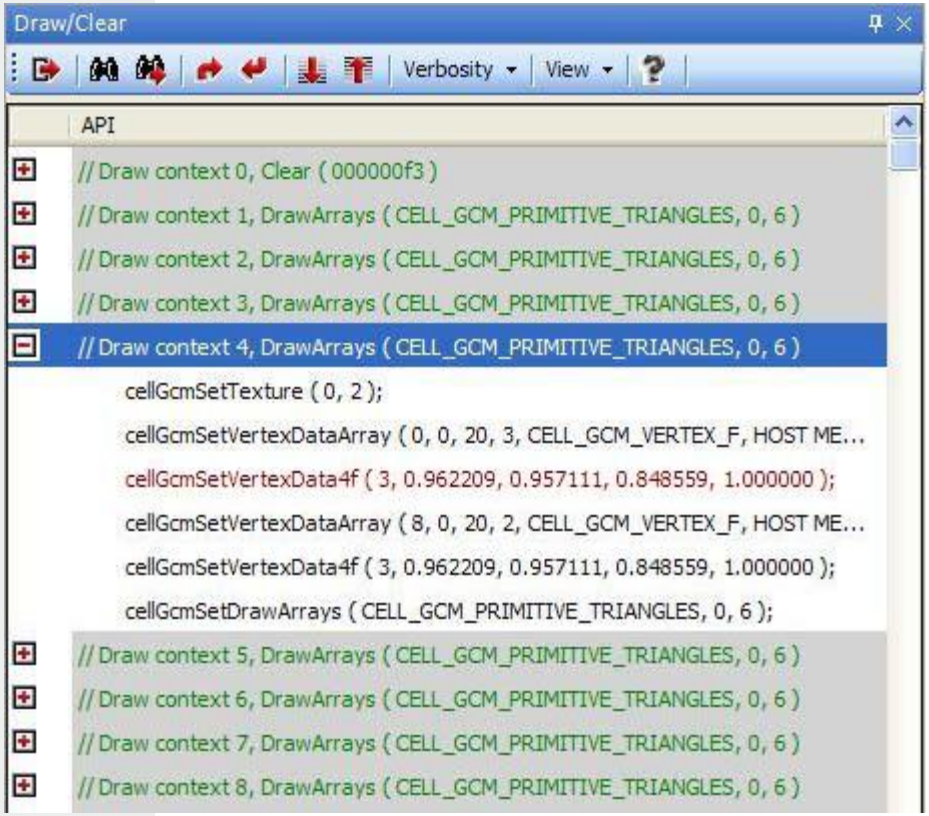


API

- // Draw context 0, Clear (000000f3)
- // Draw context 1, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 2, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 3, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 4, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 5, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 6, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 7, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 8, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 9, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 10, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 11, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 12, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 13, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)
- // Draw context 14, DrawArrays (CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6)

- Provides primary means of Command Buffer navigation
- Lists all Draw / Clear calls
- Plus their associated setup state

Draw Context View

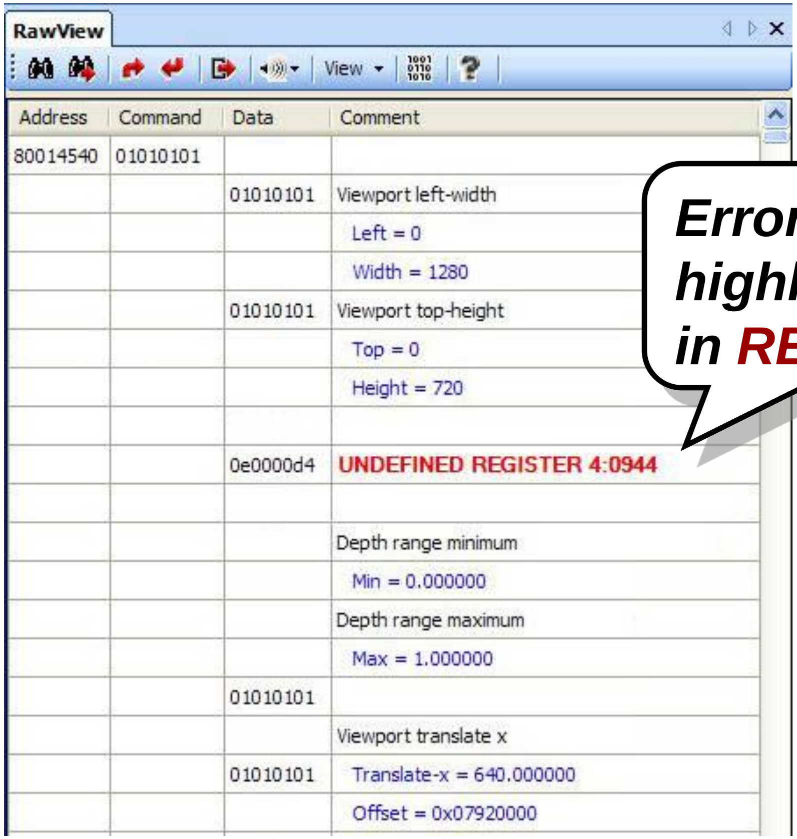


```
// Draw context 0, Clear ( 000000f3 )
// Draw context 1, DrawArrays ( CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6 )
// Draw context 2, DrawArrays ( CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6 )
// Draw context 3, DrawArrays ( CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6 )
// Draw context 4, DrawArrays ( CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6 )
    cellGcmSetTexture ( 0, 2 );
    cellGcmSetVertexDataArray ( 0, 0, 20, 3, CELL_GCM_VERTEX_F, HOST ME...
    cellGcmSetVertexData4f ( 3, 0.962209, 0.957111, 0.848559, 1.000000 );
    cellGcmSetVertexDataArray ( 8, 0, 20, 2, CELL_GCM_VERTEX_F, HOST ME...
    cellGcmSetVertexData4f ( 3, 0.962209, 0.957111, 0.848559, 1.000000 );
    cellGcmSetDrawArrays ( CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6 );
// Draw context 5, DrawArrays ( CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6 )
// Draw context 6, DrawArrays ( CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6 )
// Draw context 7, DrawArrays ( CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6 )
// Draw context 8, DrawArrays ( CELL_GCM_PRIMITIVE_TRIANGLES, 0, 6 )
```

- Expand to see your original Gcm API calls
- Full source-level disassembly
- Parameter Annotations
- User Annotations

Raw Command Buffer View

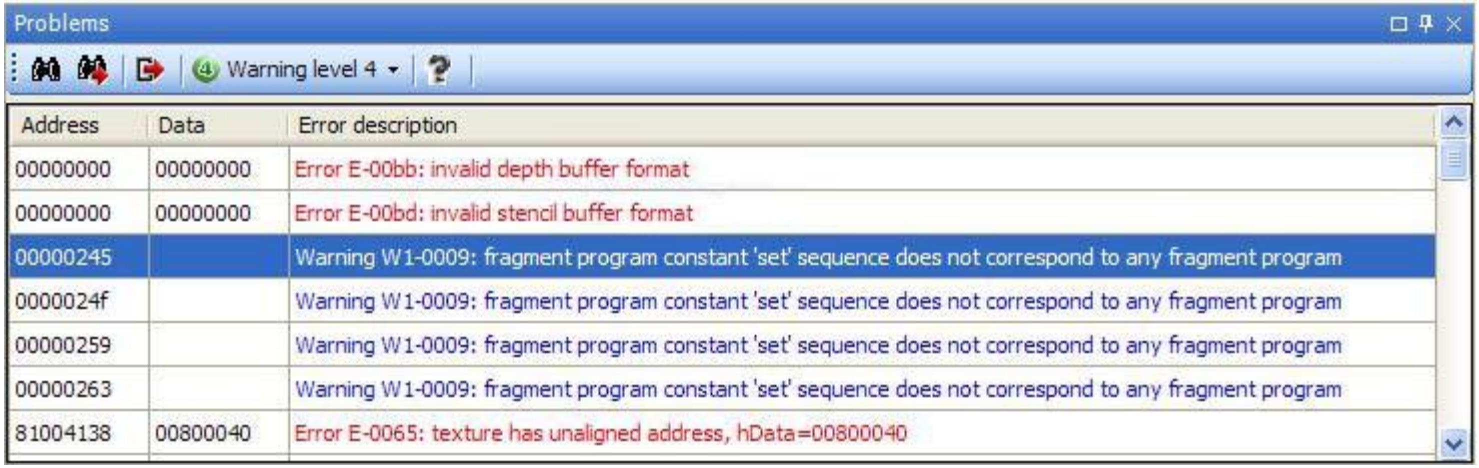
- Displays full Command Buffer disassembly



Address	Command	Data	Comment
80014540	01010101		
		01010101	Viewport left-width
			Left = 0
			Width = 1280
		01010101	Viewport top-height
			Top = 0
			Height = 720
		0e0000d4	UNDEFINED REGISTER 4:0944
			Depth range minimum
			Min = 0.000000
			Depth range maximum
			Max = 1.000000
		01010101	
			Viewport translate x
		01010101	Translate-x = 640.000000
			Offset = 0x07920000

Errors highlighted in RED

Problems View



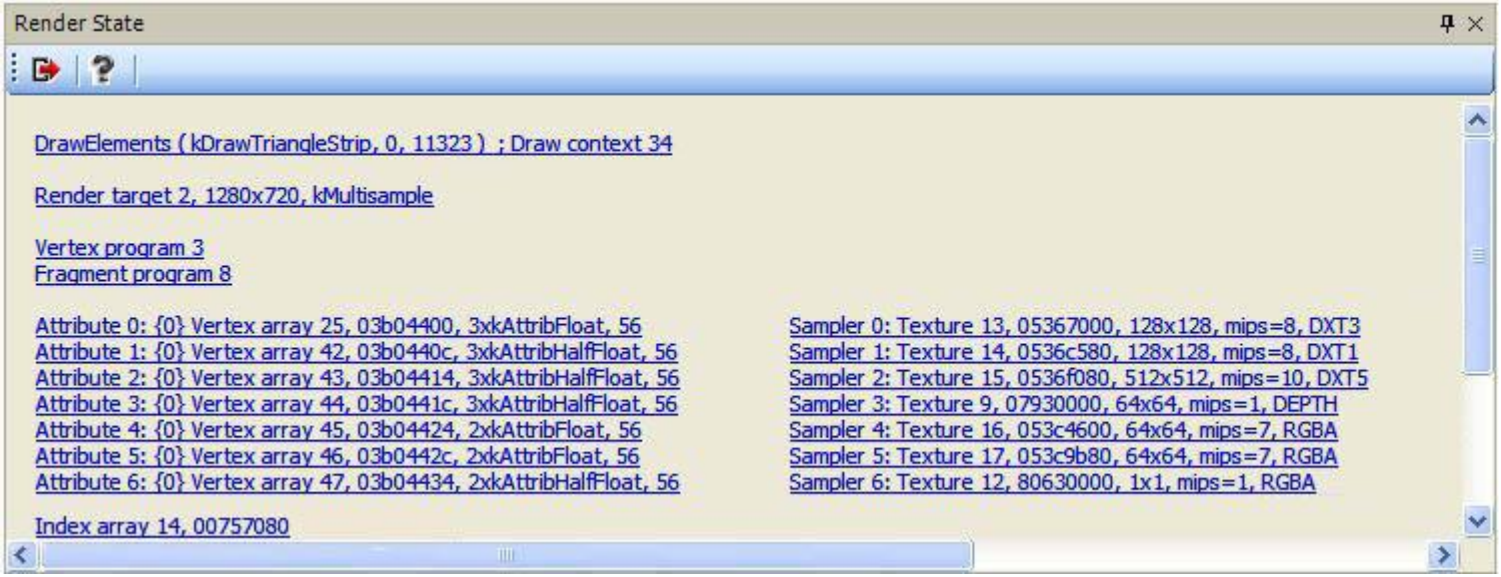
Problems

Warning level 4

Address	Data	Error description
00000000	00000000	Error E-00bb: invalid depth buffer format
00000000	00000000	Error E-00bd: invalid stencil buffer format
00000245		Warning W1-0009: fragment program constant 'set' sequence does not correspond to any fragment program
0000024f		Warning W1-0009: fragment program constant 'set' sequence does not correspond to any fragment program
00000259		Warning W1-0009: fragment program constant 'set' sequence does not correspond to any fragment program
00000263		Warning W1-0009: fragment program constant 'set' sequence does not correspond to any fragment program
81004138	00800040	Error E-0065: texture has unaligned address, hData=00800040

- Summarises all warnings and errors
- Click on problem - jump to offending Draw Context

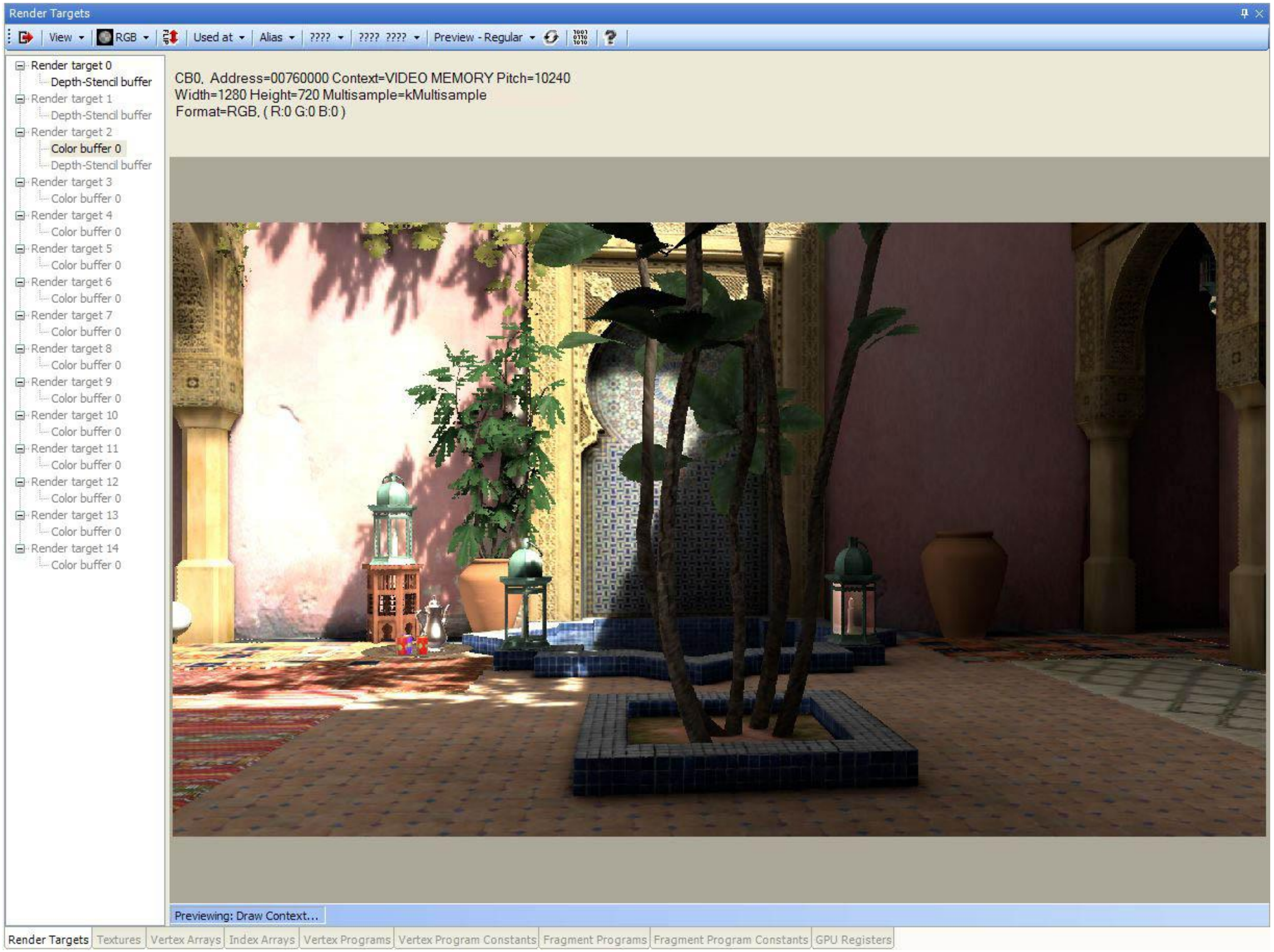
Render State View



```
DrawElements ( kDrawTriangleStrip, 0, 11323 )  ; Draw context 34

Render target 2, 1280x720, kMultisample

Vertex program 3
Fragment program 8

Attribute 0: {0} Vertex array 25, 03b04400, 3xkAttribFloat, 56        Sampler 0: Texture 13, 05367000, 128x128, mips=8, DXT3
Attribute 1: {0} Vertex array 42, 03b0440c, 3xkAttribHalfFloat, 56    Sampler 1: Texture 14, 0536c580, 128x128, mips=8, DXT1
Attribute 2: {0} Vertex array 43, 03b04414, 3xkAttribHalfFloat, 56    Sampler 2: Texture 15, 0536f080, 512x512, mips=10, DXT5
Attribute 3: {0} Vertex array 44, 03b0441c, 3xkAttribHalfFloat, 56    Sampler 3: Texture 9, 07930000, 64x64, mips=1, DEPTH
Attribute 4: {0} Vertex array 45, 03b04424, 2xkAttribFloat, 56        Sampler 4: Texture 16, 053c4600, 64x64, mips=7, RGBA
Attribute 5: {0} Vertex array 46, 03b0442c, 2xkAttribFloat, 56        Sampler 5: Texture 17, 053c9b80, 64x64, mips=7, RGBA
Attribute 6: {0} Vertex array 47, 03b04434, 2xkAttribHalfFloat, 56    Sampler 6: Texture 12, 80630000, 1x1, mips=1, RGBA

Index array 14, 00757080
```

- See all Resources referenced by current Draw Context
- Click on Link Label to select that Resource for previewing

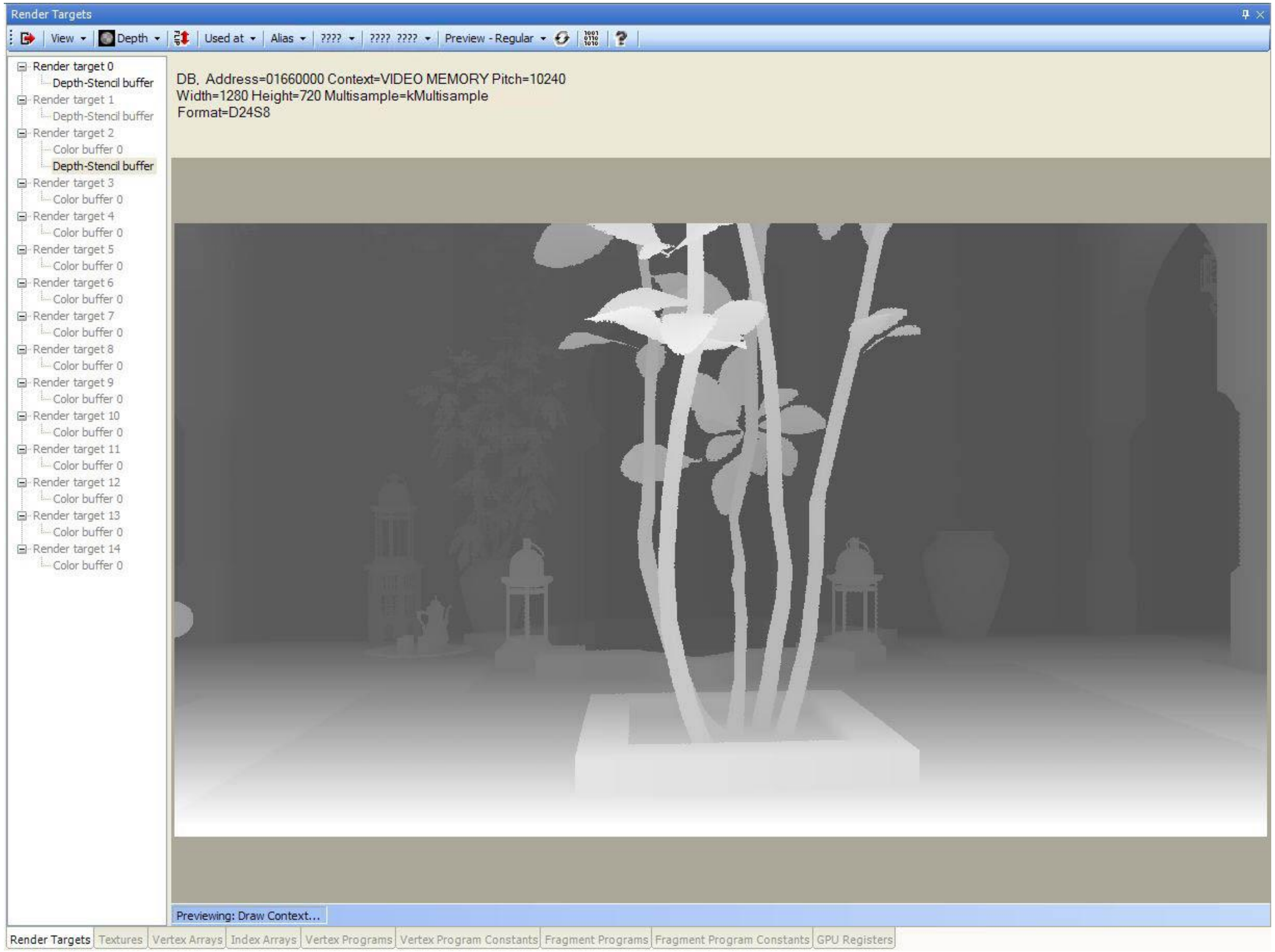
Render Targets View



```
CB0, Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, ( R:0 G:0 B:0 )
```

Previewing: Draw Context...

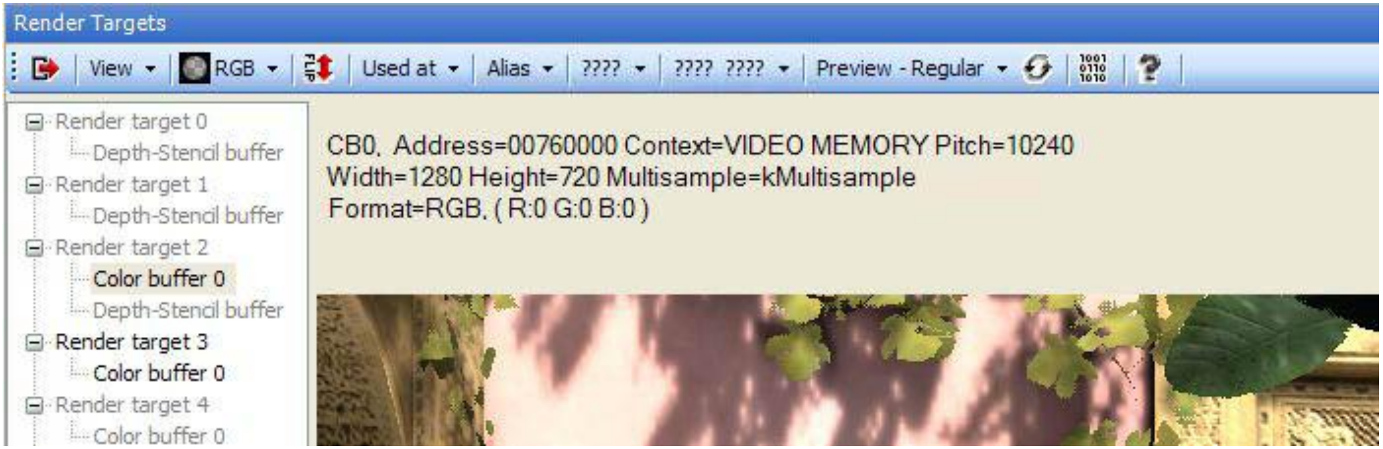
Render Targets View



DB, Address=01660000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=D24S8

Previewing: Draw Context...

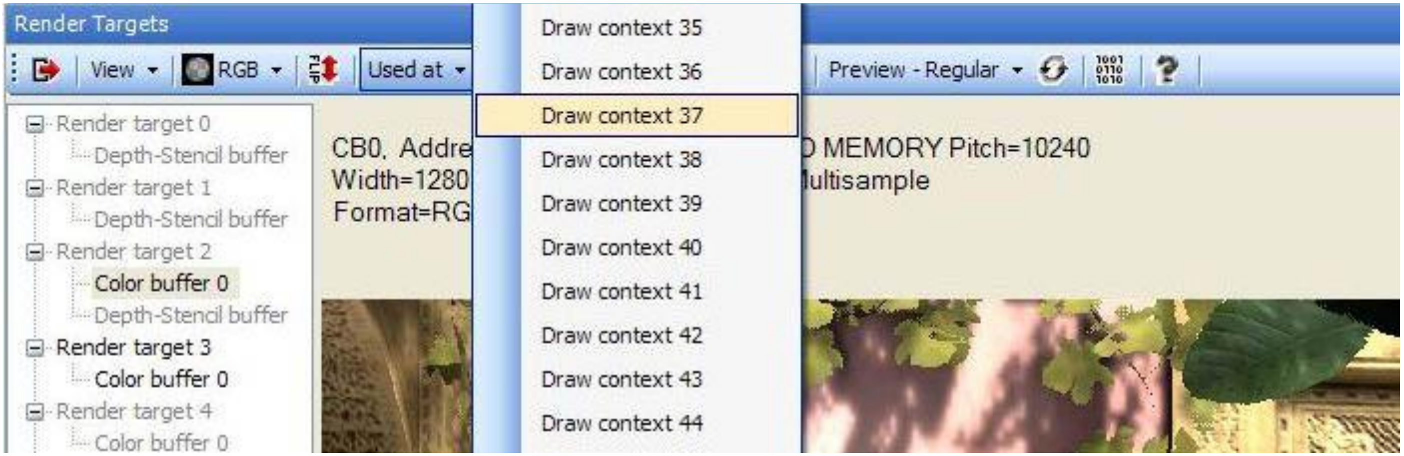
Render Targets View



CB0, Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, (R:0 G:0 B:0)

- Puts internal analysis results at your finger tips
- So you can instantly answer…

Render Targets View

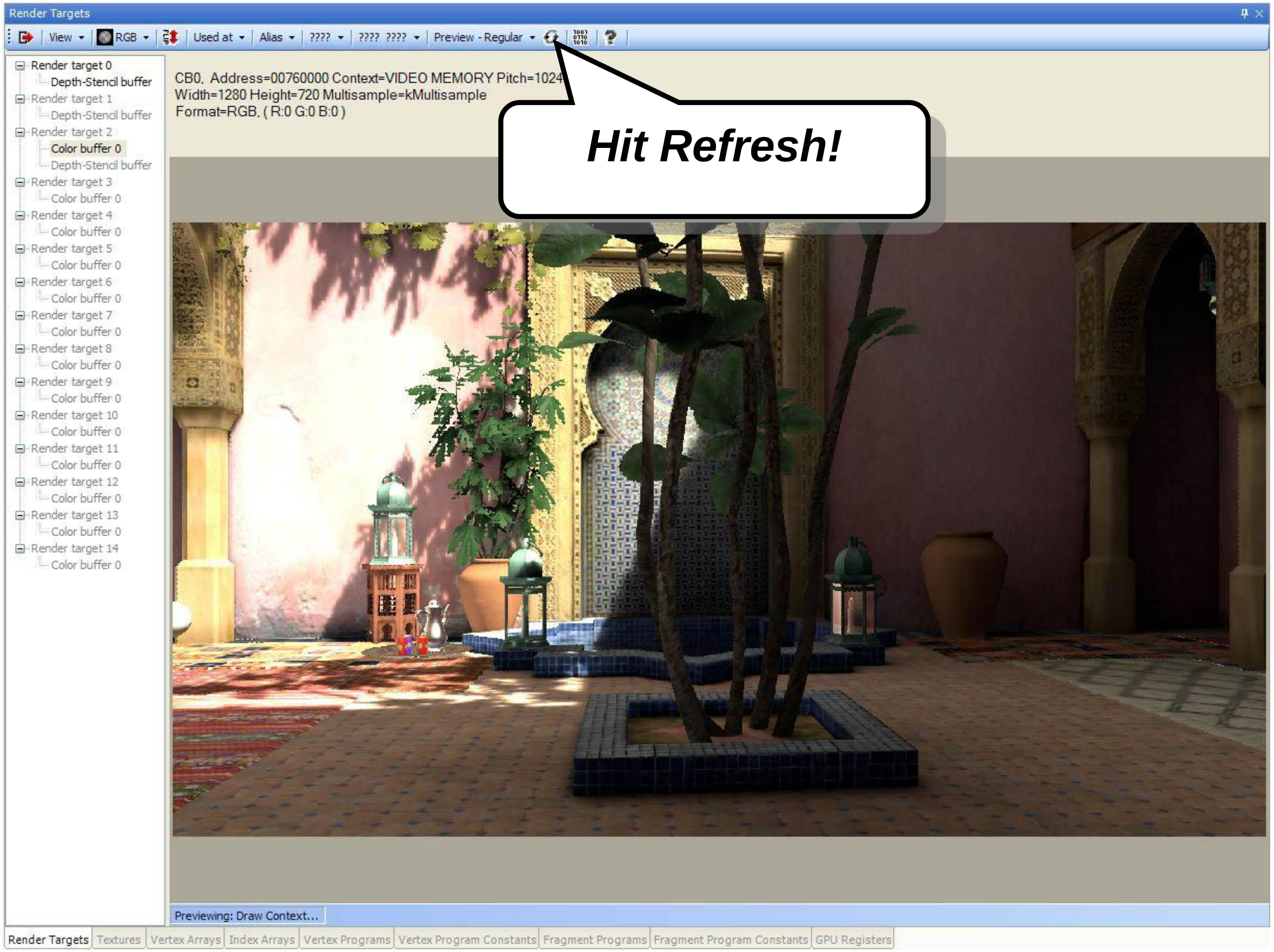


CB0, Addre
Width=1280
Format=RG

D MEMORY Pitch=10240
ultisample

- What Draw Contexts write to this Render Target?
- Is this Render Target aliased as a Texture?
- Is this Render Target setup for
  - Double-Speed rendering?
  - Early-Z optimisation?

Render Target Refresh



- Render target 0
  - Depth-Stencil buffer
- Render target 1
  - Depth-Stencil buffer
- Render target 2
  - Color buffer 0
  - Depth-Stencil buffer
- Render target 3
  - Color buffer 0
- Render target 4
  - Color buffer 0
- Render target 5
  - Color buffer 0
- Render target 6
  - Color buffer 0
- Render target 7
  - Color buffer 0
- Render target 8
  - Color buffer 0
- Render target 9
  - Color buffer 0
- Render target 10
  - Color buffer 0
- Render target 11
  - Color buffer 0
- Render target 12
  - Color buffer 0
- Render target 13
  - Color buffer 0
- Render target 14
  - Color buffer 0

```
CB0, Address=00760000 Context=VIDEO MEMORY Pitch=1024
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, ( R:0 G:0 B:0 )
```

Previewing: Draw Context...

Hit Refresh!

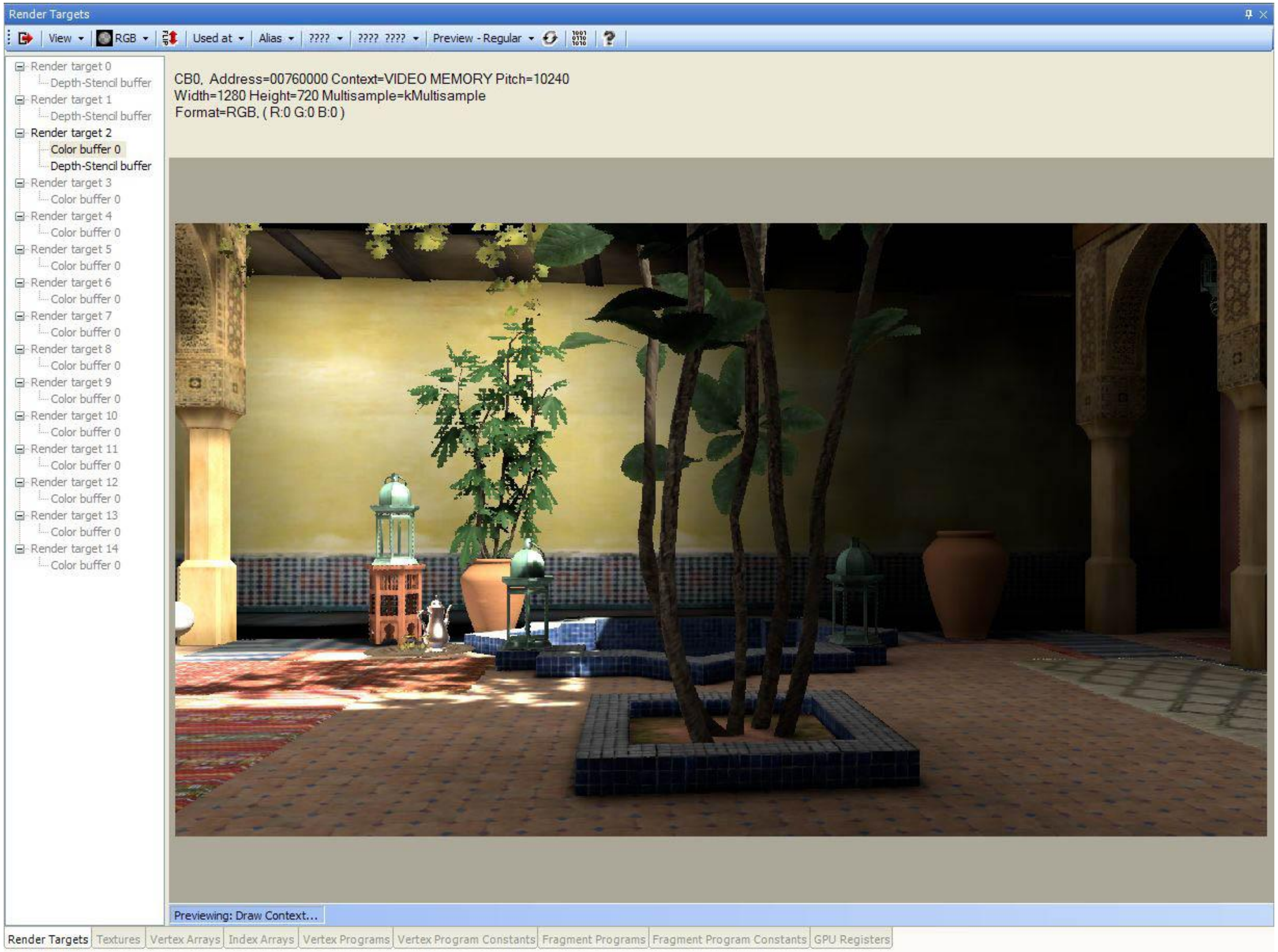
Render Target Refresh

1. Transfer Command Buffer and Resources
2. Kick Command Buffer up to the current Draw Context

Allows you to single step your rendering process

Both forwards and backwards in time

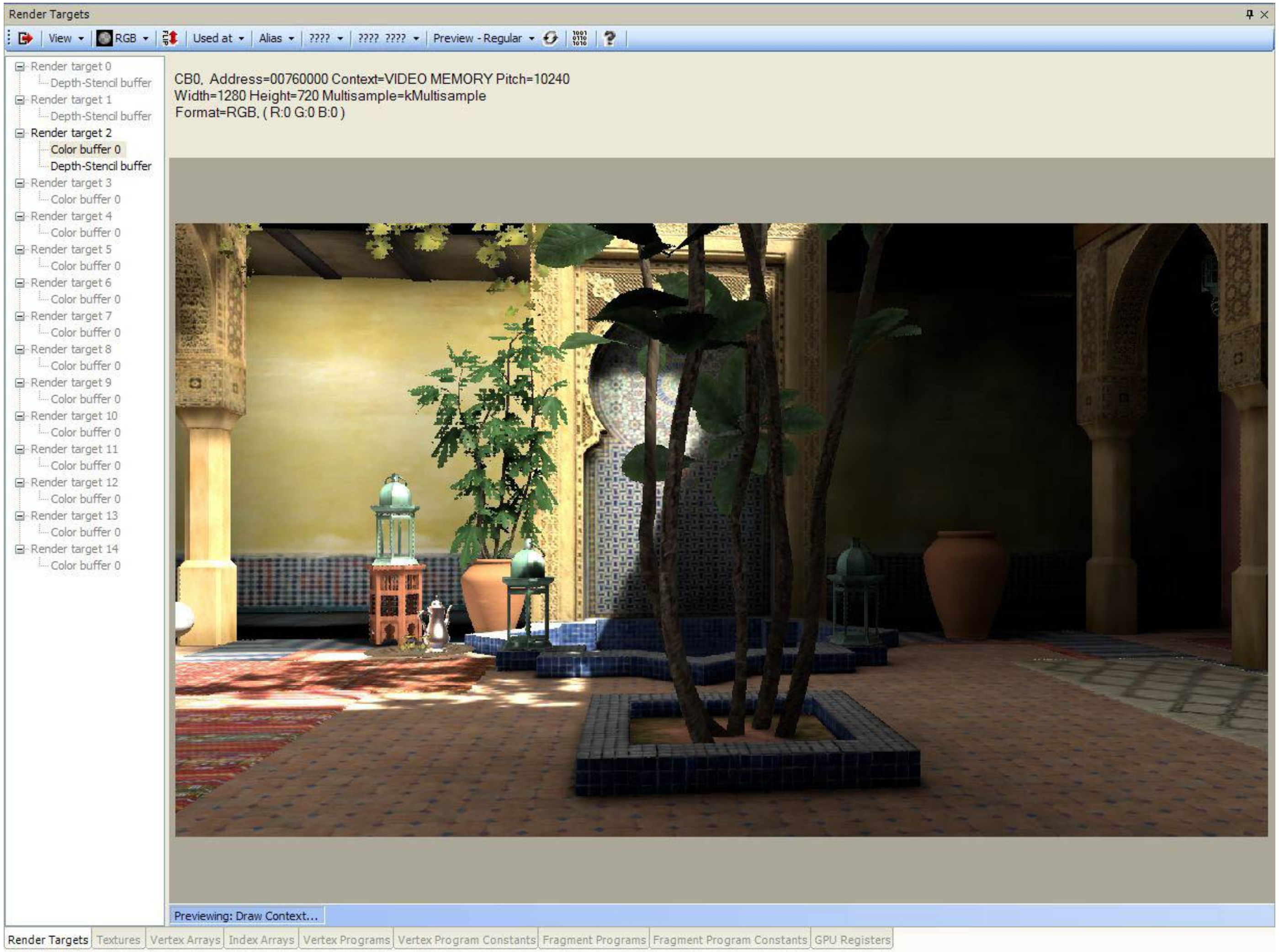
Render Target Refresh



CB0, Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, (R:0 G:0 B:0)

Previewing: Draw Context...

Render Target Refresh

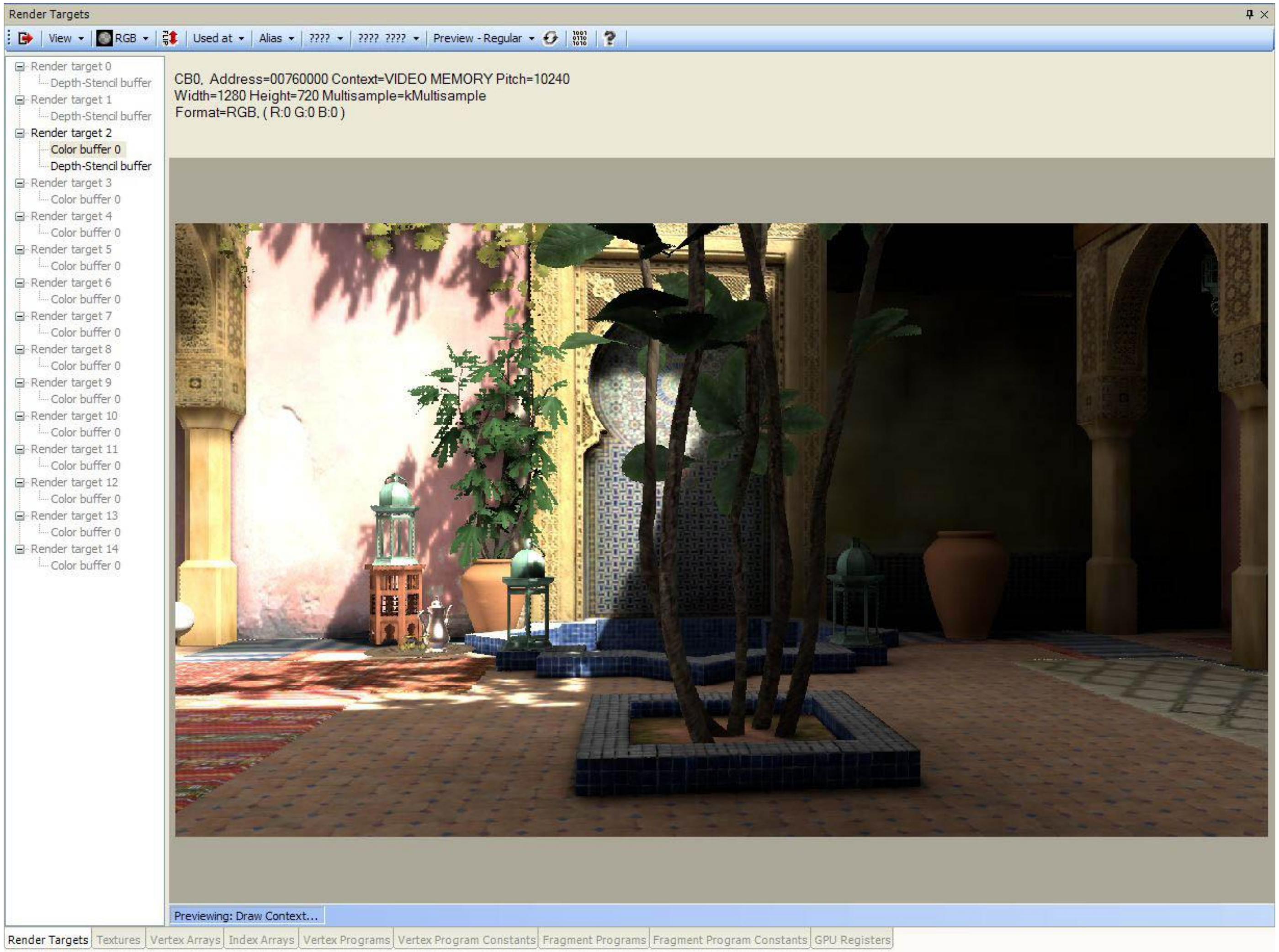


- Render target 0
  - Depth-Stencil buffer
- Render target 1
  - Depth-Stencil buffer
- Render target 2
  - Color buffer 0
  - Depth-Stencil buffer
- Render target 3
  - Color buffer 0
- Render target 4
  - Color buffer 0
- Render target 5
  - Color buffer 0
- Render target 6
  - Color buffer 0
- Render target 7
  - Color buffer 0
- Render target 8
  - Color buffer 0
- Render target 9
  - Color buffer 0
- Render target 10
  - Color buffer 0
- Render target 11
  - Color buffer 0
- Render target 12
  - Color buffer 0
- Render target 13
  - Color buffer 0
- Render target 14
  - Color buffer 0

```
CB0,  Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, ( R:0 G:0 B:0 )
```

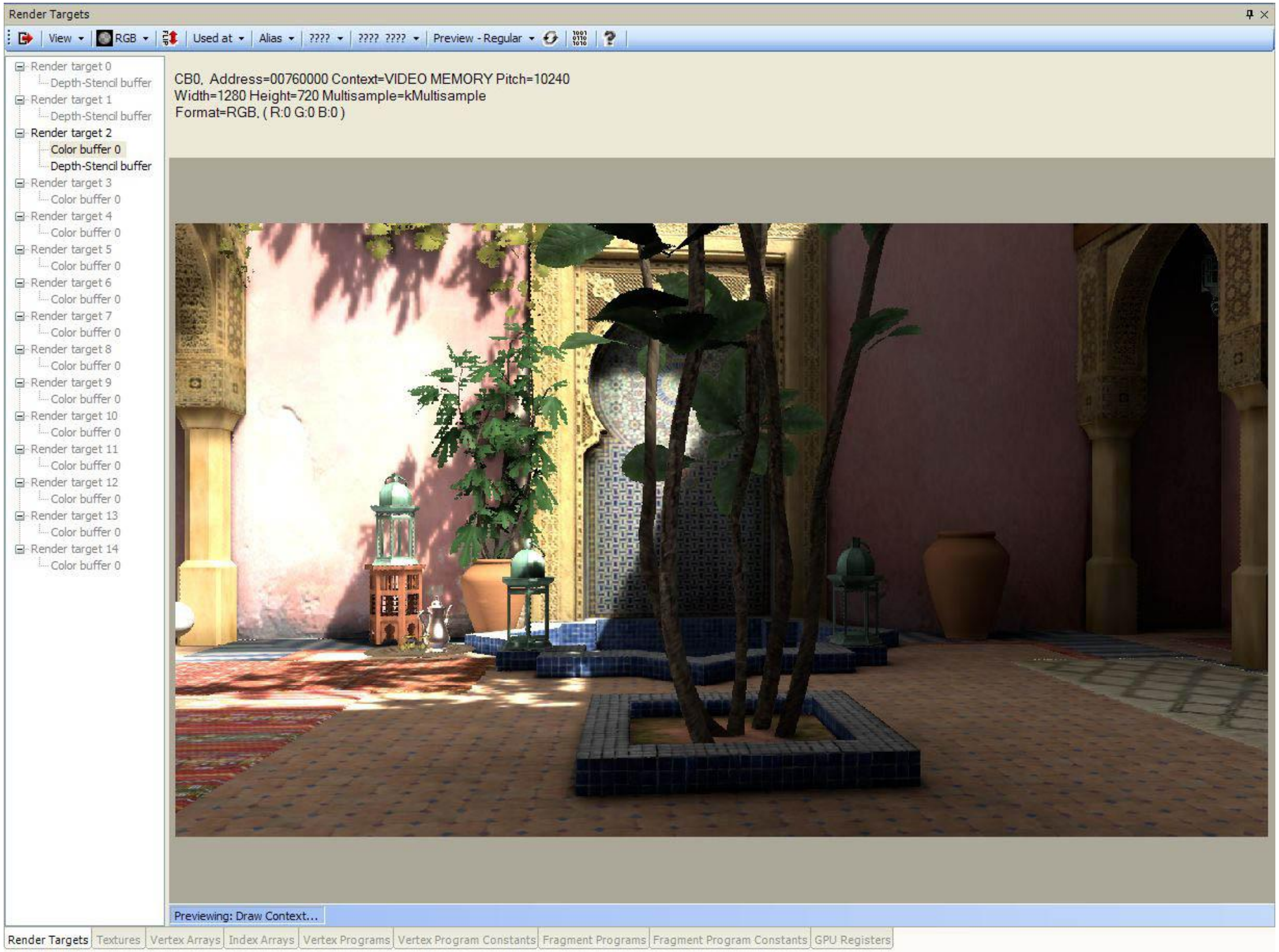
Previewing: Draw Context...

Render Target Refresh



CB0, Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, (R:0 G:0 B:0)

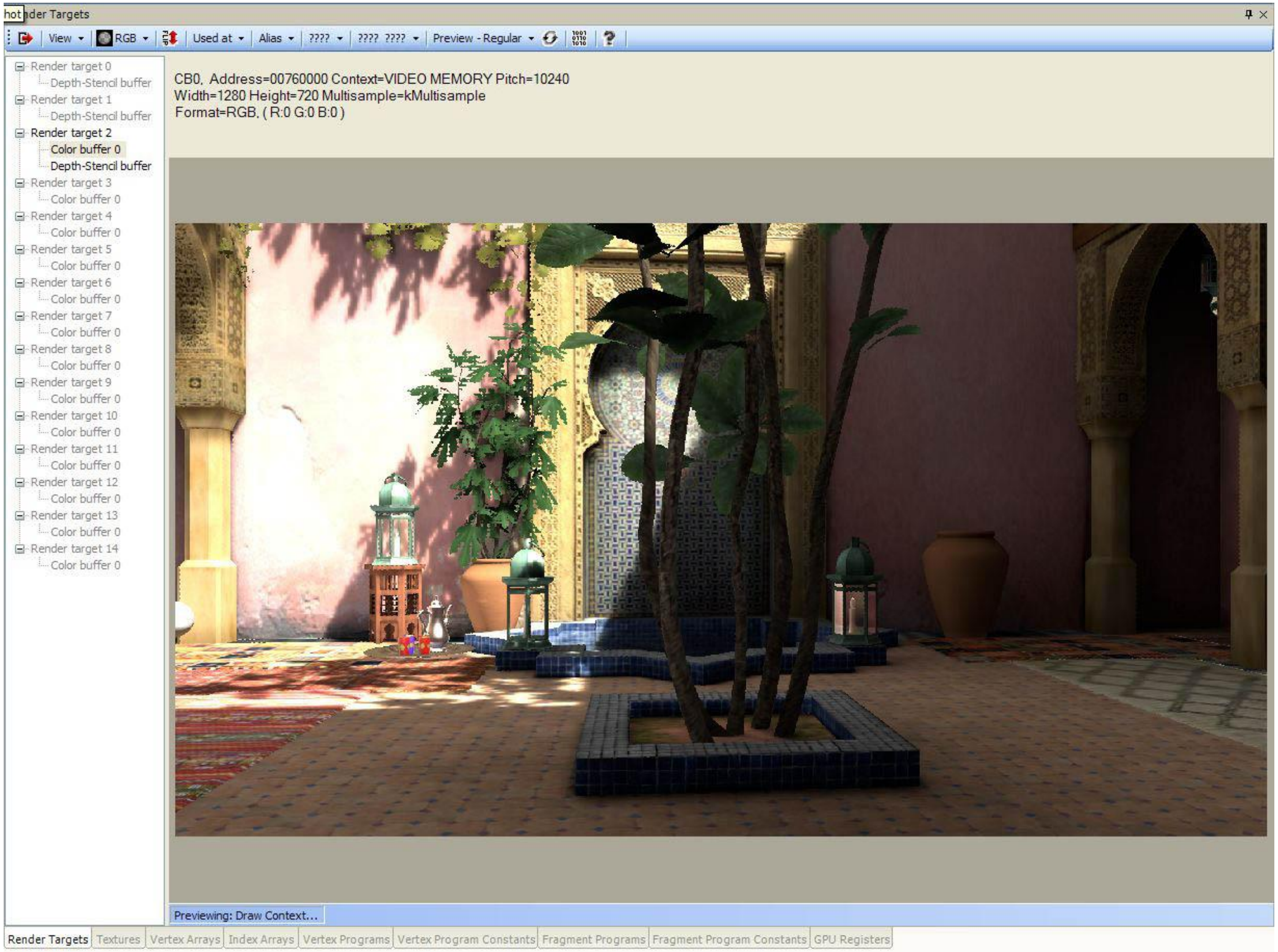
Render Target Refresh



CB0, Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, (R:0 G:0 B:0)

Previewing: Draw Context...

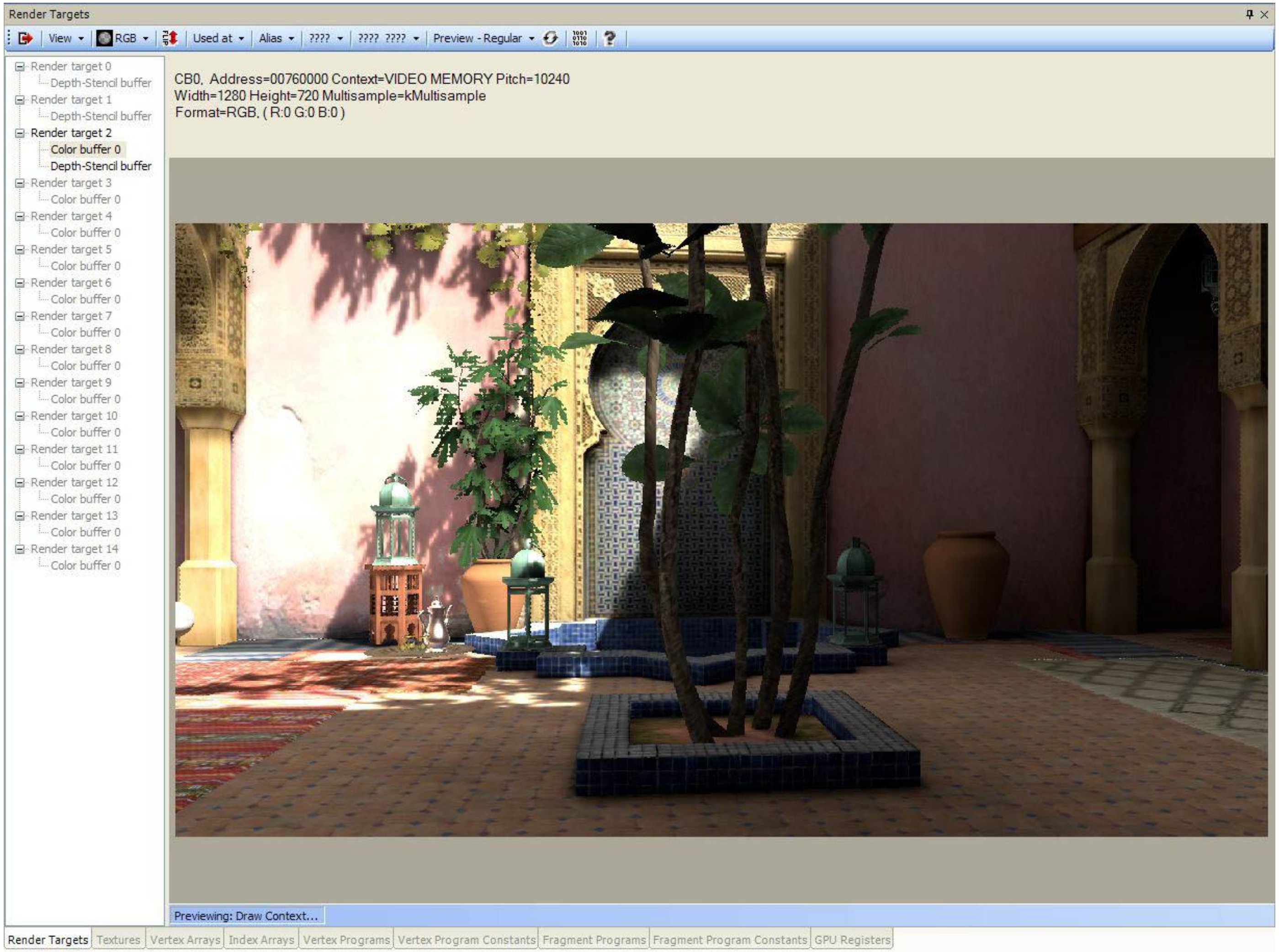
Render Target Refresh



```
CB0, Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, ( R:0 G:0 B:0 )
```

Previewing: Draw Context...

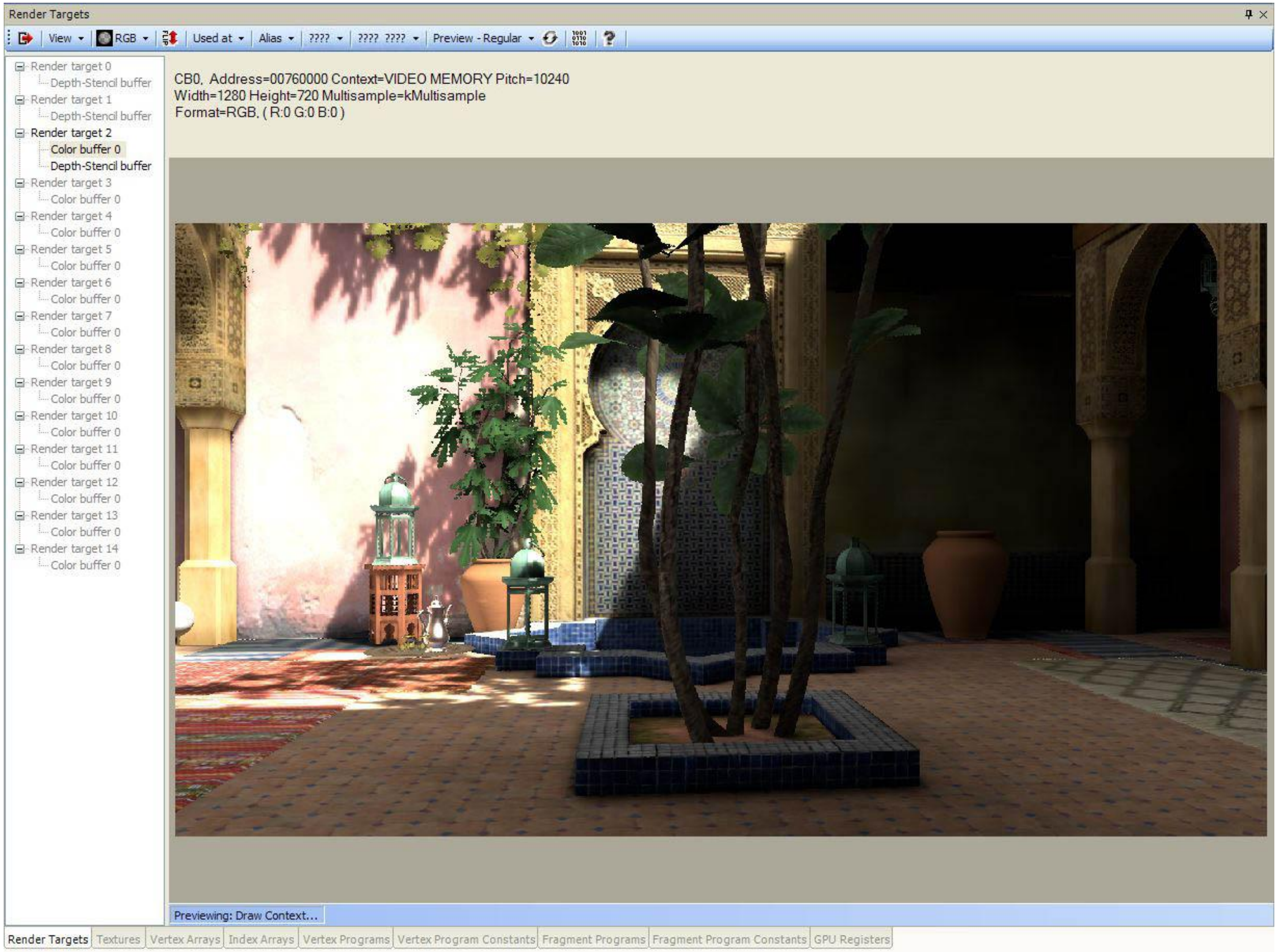
Render Target Refresh



CB0, Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, (R:0 G:0 B:0)

Previewing: Draw Context...

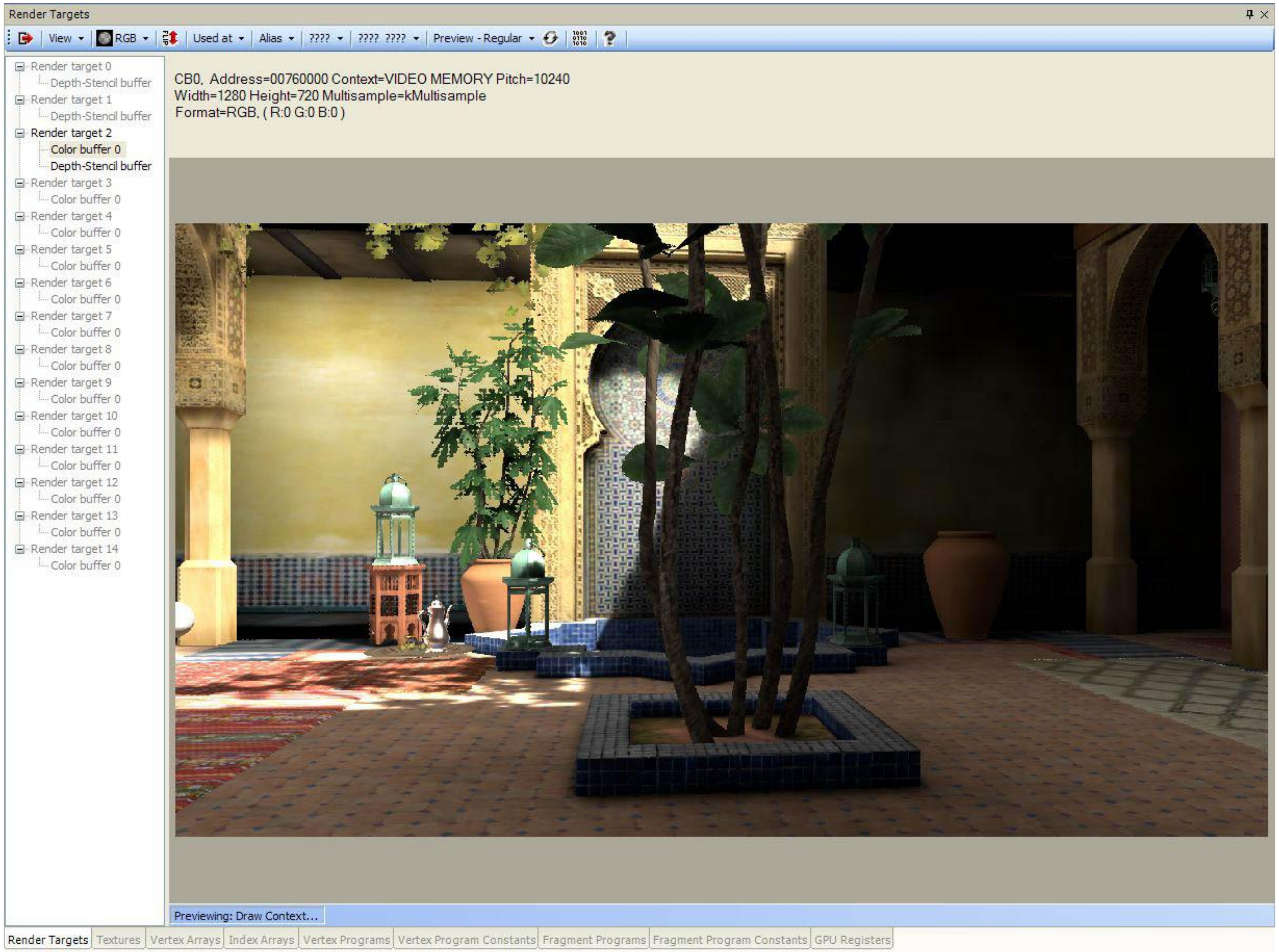
Render Target Refresh



CB0, Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, (R:0 G:0 B:0)

Previewing: Draw Context...

Render Target Refresh

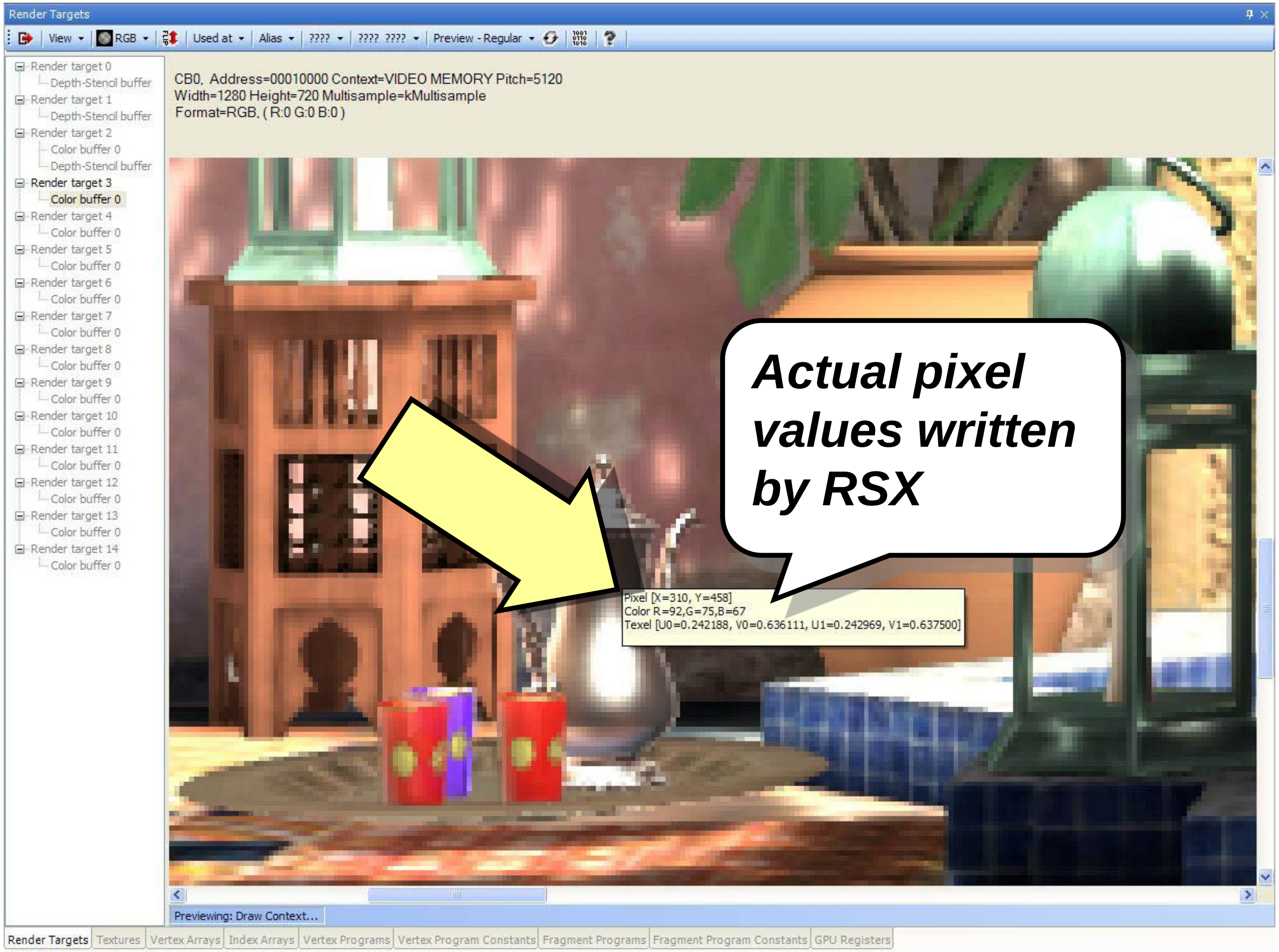


- Render target 0
  - Depth-Stencil buffer
- Render target 1
  - Depth-Stencil buffer
- Render target 2
  - Color buffer 0
  - Depth-Stencil buffer
- Render target 3
  - Color buffer 0
- Render target 4
  - Color buffer 0
- Render target 5
  - Color buffer 0
- Render target 6
  - Color buffer 0
- Render target 7
  - Color buffer 0
- Render target 8
  - Color buffer 0
- Render target 9
  - Color buffer 0
- Render target 10
  - Color buffer 0
- Render target 11
  - Color buffer 0
- Render target 12
  - Color buffer 0
- Render target 13
  - Color buffer 0
- Render target 14
  - Color buffer 0

```
CB0, Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, ( R:0 G:0 B:0 )
```

Previewing: Draw Context...

Render Target Pixel Analysis



Render Targets

- Render target 0
  - Depth-Stencil buffer
- Render target 1
  - Depth-Stencil buffer
- Render target 2
  - Color buffer 0
  - Depth-Stencil buffer
- Render target 3
  - Color buffer 0
- Render target 4
  - Color buffer 0
- Render target 5
  - Color buffer 0
- Render target 6
  - Color buffer 0
- Render target 7
  - Color buffer 0
- Render target 8
  - Color buffer 0
- Render target 9
  - Color buffer 0
- Render target 10
  - Color buffer 0
- Render target 11
  - Color buffer 0
- Render target 12
  - Color buffer 0
- Render target 13
  - Color buffer 0
- Render target 14
  - Color buffer 0

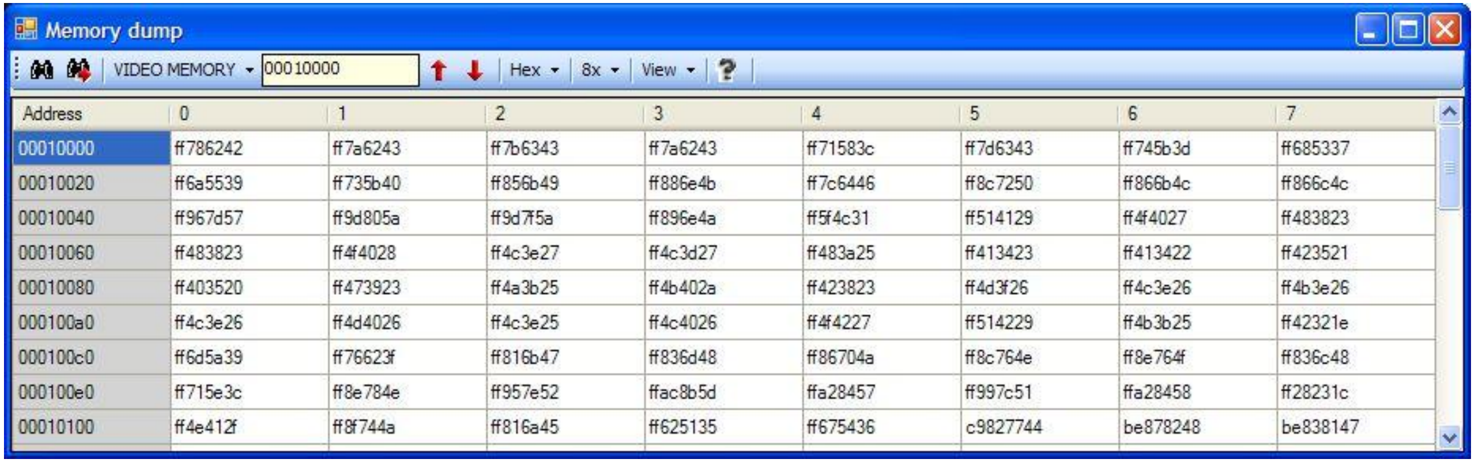
```
CB0,  Address=00010000 Context=VIDEO MEMORY Pitch=5120
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, ( R:0 G:0 B:0 )
```

```
Pixel [X=310, Y=458]
Color R=92,G=75,B=67
Texel [U0=0.242188, V0=0.636111, U1=0.242969, V1=0.637500]
```

Previewing: Draw Context...

Actual pixel values written by RSX

Render Target Memory Dump

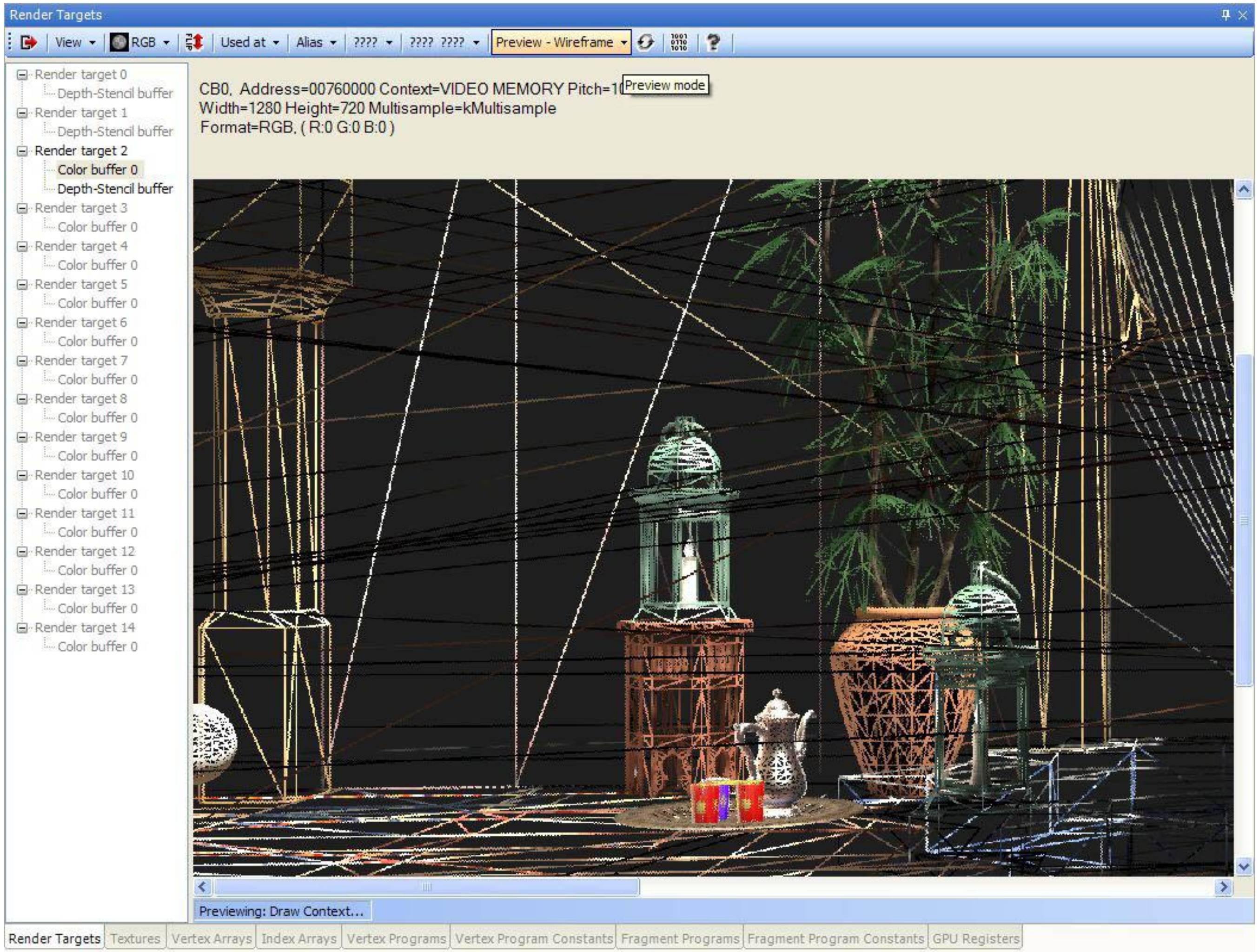


Address	0	1	2	3	4	5	6	7
00010000	ff786242	ff7a6243	ff7b6343	ff7a6243	ff71583c	ff7d6343	ff745b3d	ff685337
00010020	ff6a5539	ff735b40	ff856b49	ff886e4b	ff7c6446	ff8c7250	ff866b4c	ff866c4c
00010040	ff967d57	ff9d805a	ff9d7f5a	ff896e4a	ff5f4c31	ff514129	ff4f4027	ff483823
00010060	ff483823	ff4f4028	ff4c3e27	ff4c3d27	ff483a25	ff413423	ff413422	ff423521
00010080	ff403520	ff473923	ff4a3b25	ff4b402a	ff423823	ff4d3f26	ff4c3e26	ff4b3e26
000100a0	ff4c3e26	ff4d4026	ff4c3e25	ff4c4026	ff4f4227	ff514229	ff4b3b25	ff42321e
000100c0	ff6d5a39	ff76623f	ff816b47	ff836d48	ff86704a	ff8c764e	ff8e764f	ff836c48
000100e0	ff715e3c	ff8e784e	ff957e52	ffac8b5d	ffa28457	ff997c51	ffa28458	ff28231c
00010100	ff4e412f	ff8f744a	ff816a45	ff625135	ff675436	c9827744	be878248	be838147

Preview Render Target

Modify *the Command Buffer before its kicked*

Preview Render Target as Wireframe



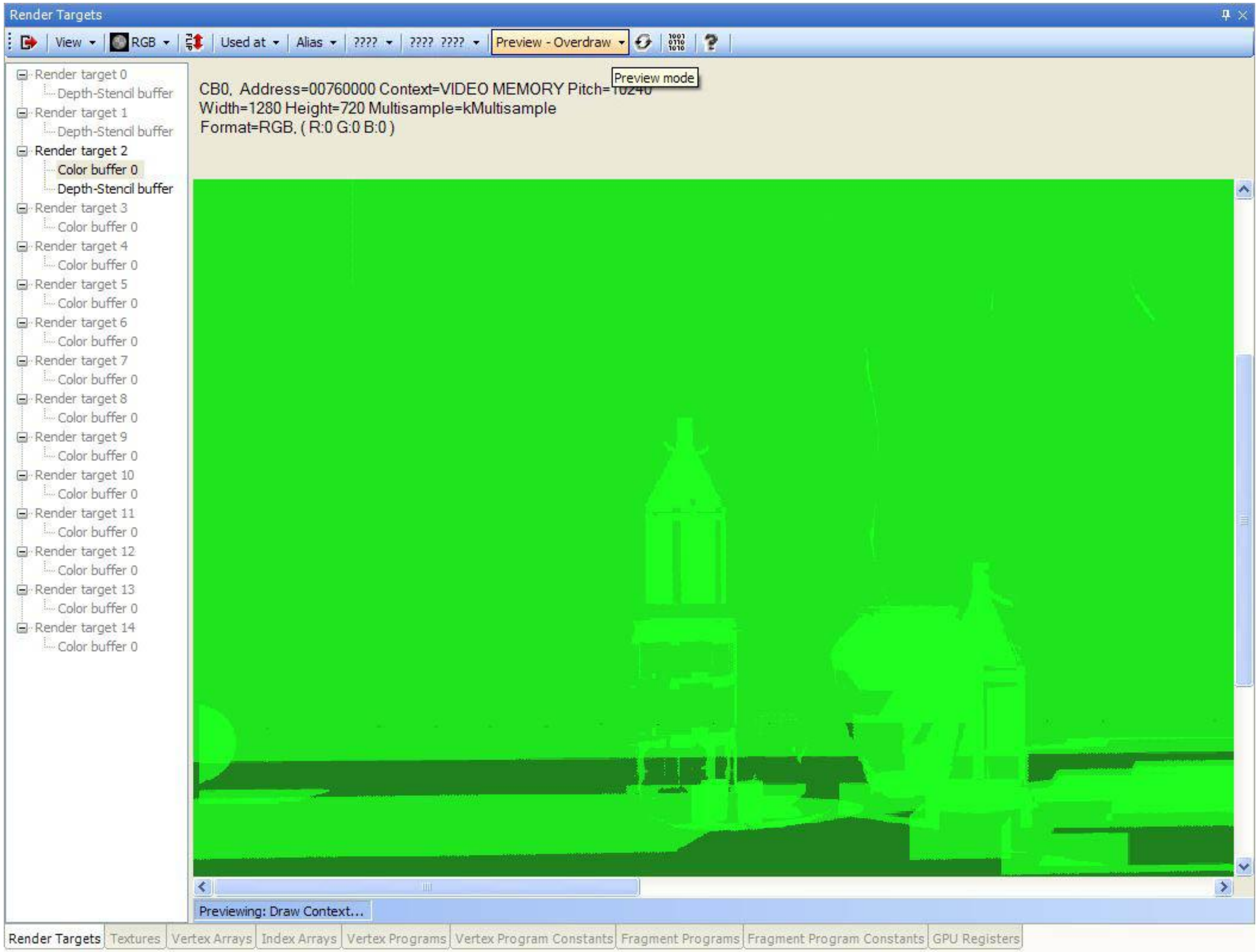
CB0, Address=00760000 Context=VIDEO MEMORY Pitch=1

Width=1280 Height=720 Multisample=kMultisample

Format=RGB, (R:0 G:0 B:0)

Previewing: Draw Context...

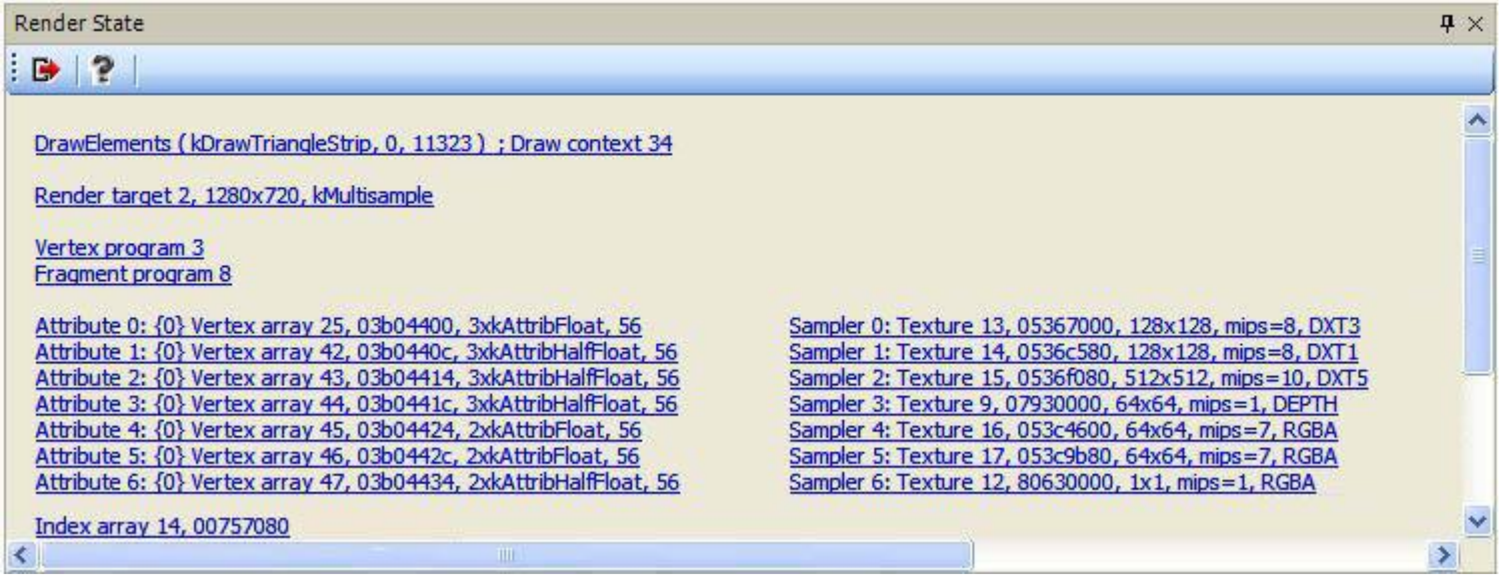
Preview Render Target Overdraw



```
CB0, Address=00760000 Context=VIDEO MEMORY Pitch=10240
Width=1280 Height=720 Multisample=kMultisample
Format=RGB, ( R:0 G:0 B:0 )
```

Previewing: Draw Context...

Render State View



DrawElements (kDrawTriangleStrip, 0, 11323) ; Draw context 34

Render target 2, 1280x720, kMultisample

Vertex program 3
Fragment program 8

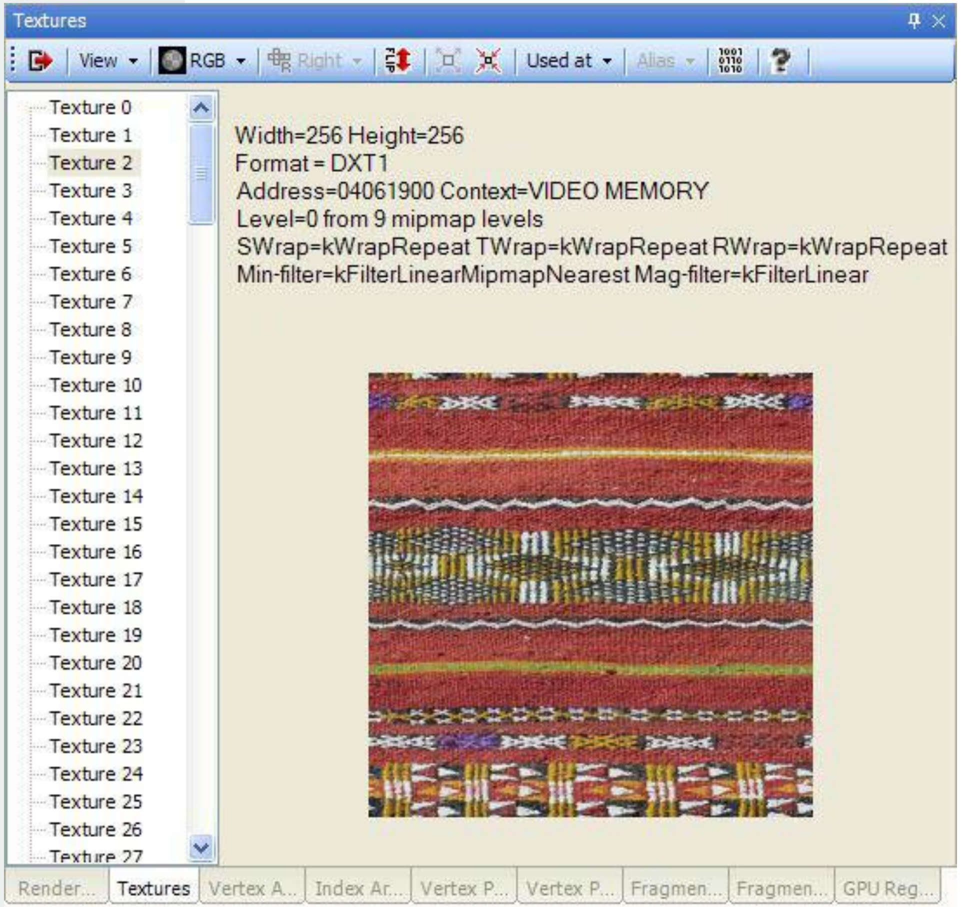
Attribute 0: {0} Vertex array 25, 03b04400, 3xkAttribFloat, 56
Attribute 1: {0} Vertex array 42, 03b0440c, 3xkAttribHalfFloat, 56
Attribute 2: {0} Vertex array 43, 03b04414, 3xkAttribHalfFloat, 56
Attribute 3: {0} Vertex array 44, 03b0441c, 3xkAttribHalfFloat, 56
Attribute 4: {0} Vertex array 45, 03b04424, 2xkAttribFloat, 56
Attribute 5: {0} Vertex array 46, 03b0442c, 2xkAttribFloat, 56
Attribute 6: {0} Vertex array 47, 03b04434, 2xkAttribHalfFloat, 56

Sampler 0: Texture 13, 05367000, 128x128, mips=8, DXT3
Sampler 1: Texture 14, 0536c580, 128x128, mips=8, DXT1
Sampler 2: Texture 15, 0536f080, 512x512, mips=10, DXT5
Sampler 3: Texture 9, 07930000, 64x64, mips=1, DEPTH
Sampler 4: Texture 16, 053c4600, 64x64, mips=7, RGBA
Sampler 5: Texture 17, 053c9b80, 64x64, mips=7, RGBA
Sampler 6: Texture 12, 80630000, 1x1, mips=1, RGBA

Index array 14, 00757080

- What other Resource types can we select for analysis?

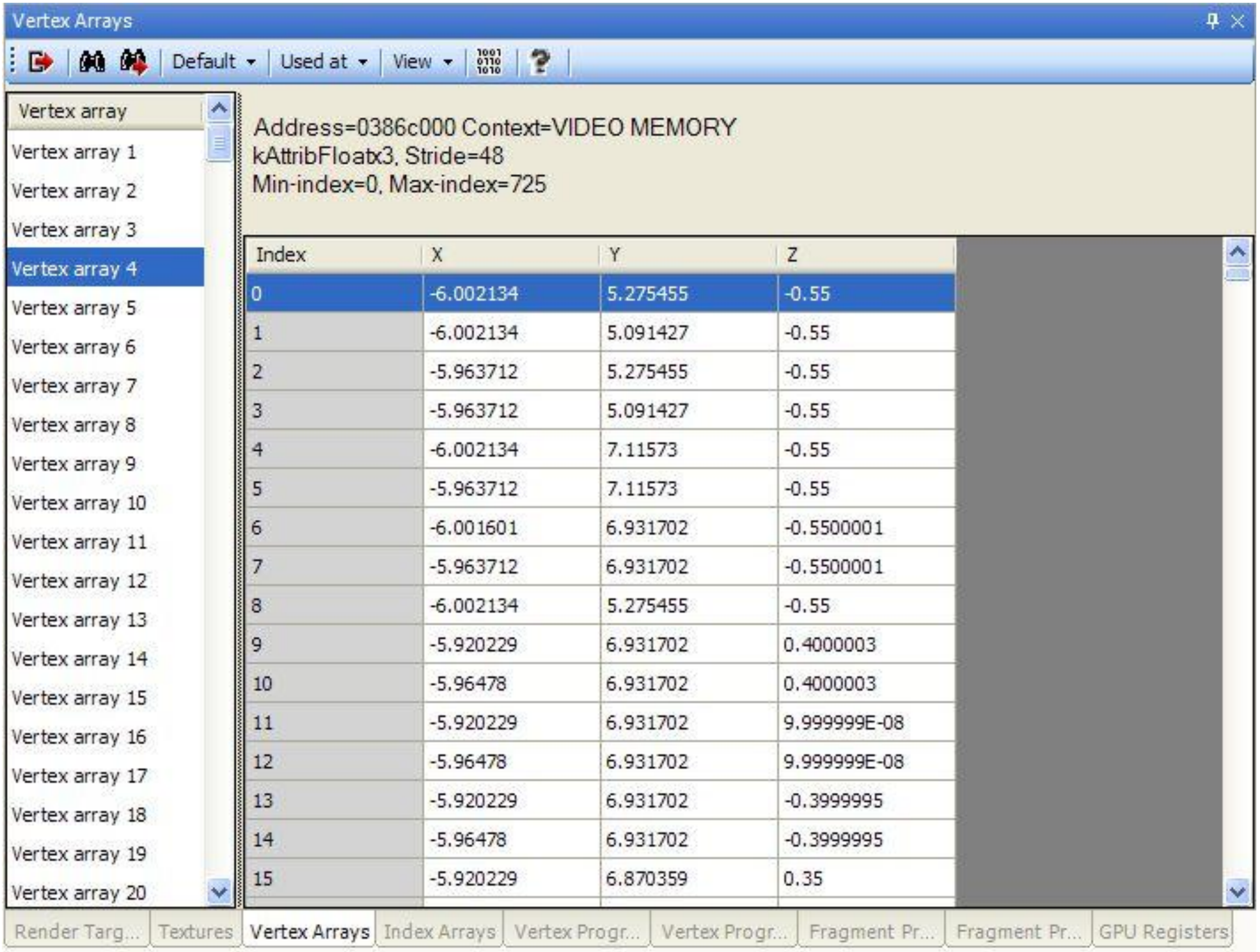
Textures View



```
Width=256 Height=256
Format = DXT1
Address=04061900 Context=VIDEO MEMORY
Level=0 from 9 mipmap levels
SWrap=kWrapRepeat TWrap=kWrapRepeat RWrap=kWrapRepeat
Min-filter=kFilterLinearMipmapNearest Mag-filter=kFilterLinear
```

- View referenced Textures
- Including
  - State
  - Complete Mip-chain
  - Cube Faces
  - Volume Slices

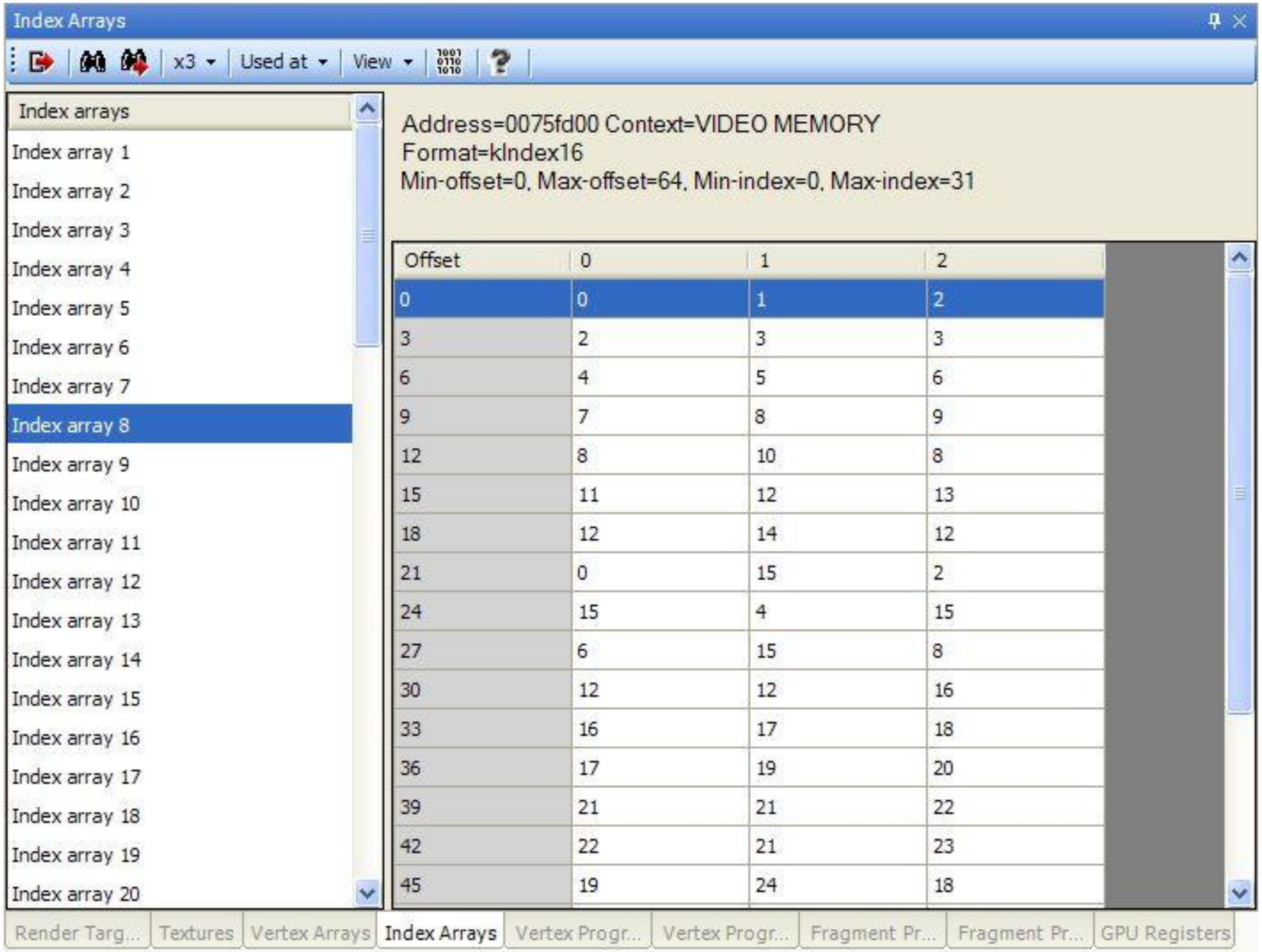
Vertex Arrays View



Address=0386c000 Context=VIDEO MEMORY
kAttribFloatx3, Stride=48
Min-index=0, Max-index=725

Index	X	Y	Z
0	-6.002134	5.275455	-0.55
1	-6.002134	5.091427	-0.55
2	-5.963712	5.275455	-0.55
3	-5.963712	5.091427	-0.55
4	-6.002134	7.11573	-0.55
5	-5.963712	7.11573	-0.55
6	-6.001601	6.931702	-0.5500001
7	-5.963712	6.931702	-0.5500001
8	-6.002134	5.275455	-0.55
9	-5.920229	6.931702	0.4000003
10	-5.96478	6.931702	0.4000003
11	-5.920229	6.931702	9.999999E-08
12	-5.96478	6.931702	9.999999E-08
13	-5.920229	6.931702	-0.3999995
14	-5.96478	6.931702	-0.3999995
15	-5.920229	6.870359	0.35

Index Arrays View



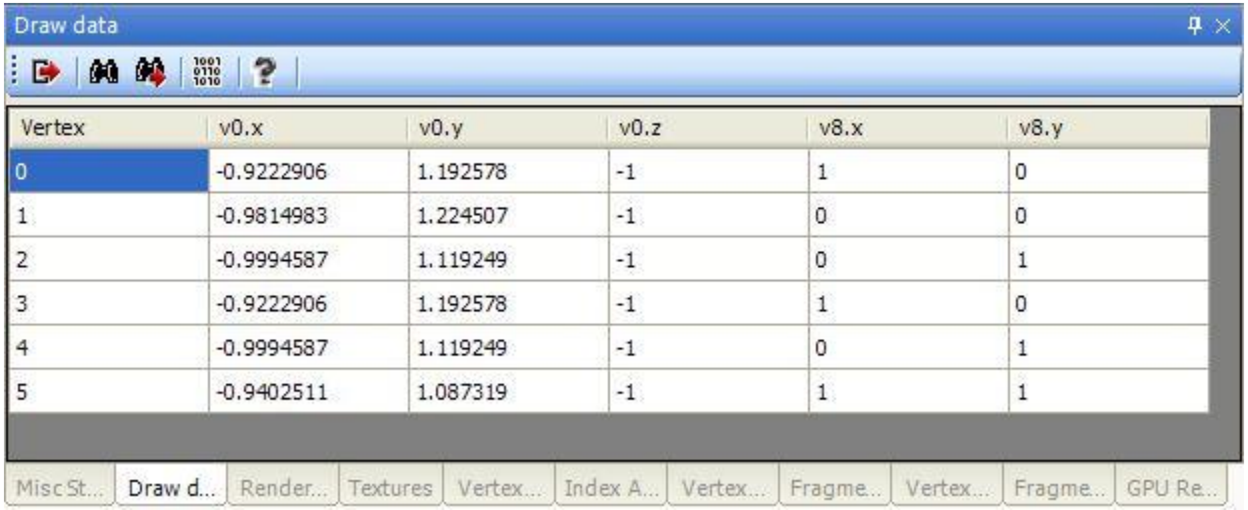
Index arrays

Index array 1
Index array 2
Index array 3
Index array 4
Index array 5
Index array 6
Index array 7
Index array 8
Index array 9
Index array 10
Index array 11
Index array 12
Index array 13
Index array 14
Index array 15
Index array 16
Index array 17
Index array 18
Index array 19
Index array 20

Address=0075fd00 Context=VIDEO MEMORY
Format=kIndex16
Min-offset=0, Max-offset=64, Min-index=0, Max-index=31

Offset	0	1	2
0	0	1	2
3	2	3	3
6	4	5	6
9	7	8	9
12	8	10	8
15	11	12	13
18	12	14	12
21	0	15	2
24	15	4	15
27	6	15	8
30	12	12	16
33	16	17	18
36	17	19	20
39	21	21	22
42	22	21	23
45	19	24	18

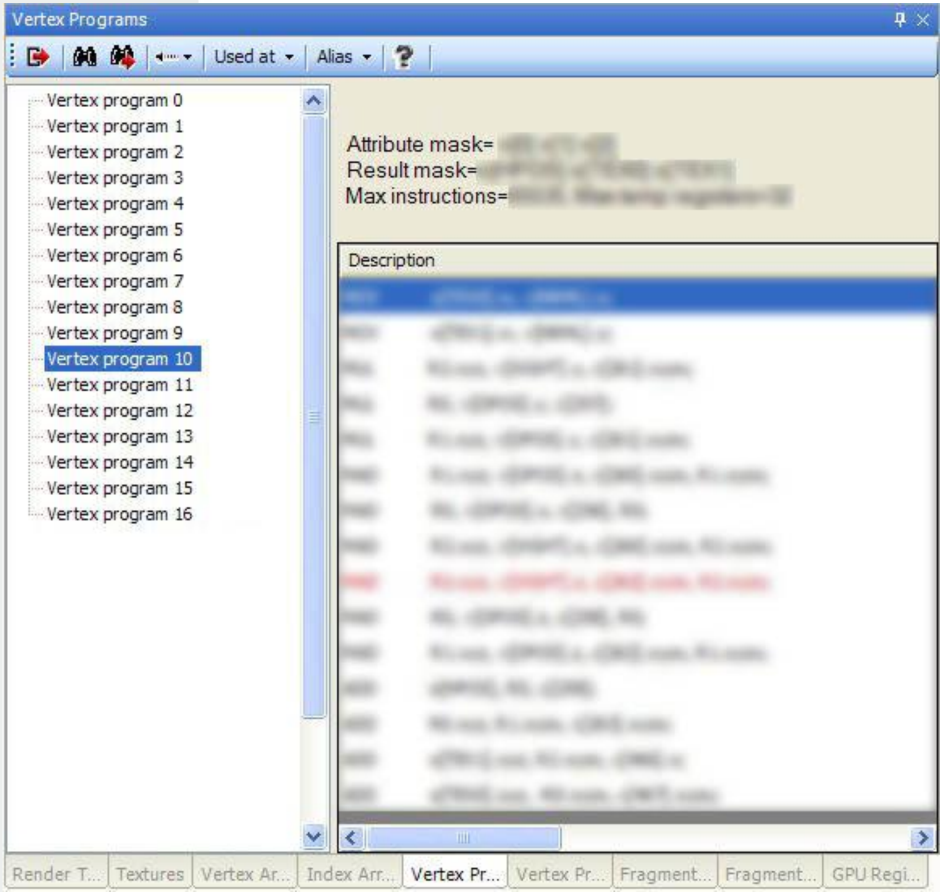
Draw Data View



Vertex	v0.x	v0.y	v0.z	v8.x	v8.y
0	-0.9222906	1.192578	-1	1	0
1	-0.9814983	1.224507	-1	0	0
2	-0.9994587	1.119249	-1	0	1
3	-0.9222906	1.192578	-1	1	0
4	-0.9994587	1.119249	-1	0	1
5	-0.9402511	1.087319	-1	1	1

- See vertices kicked by current Draw Context
- Each element of each referenced attribute

Vertex Programs View

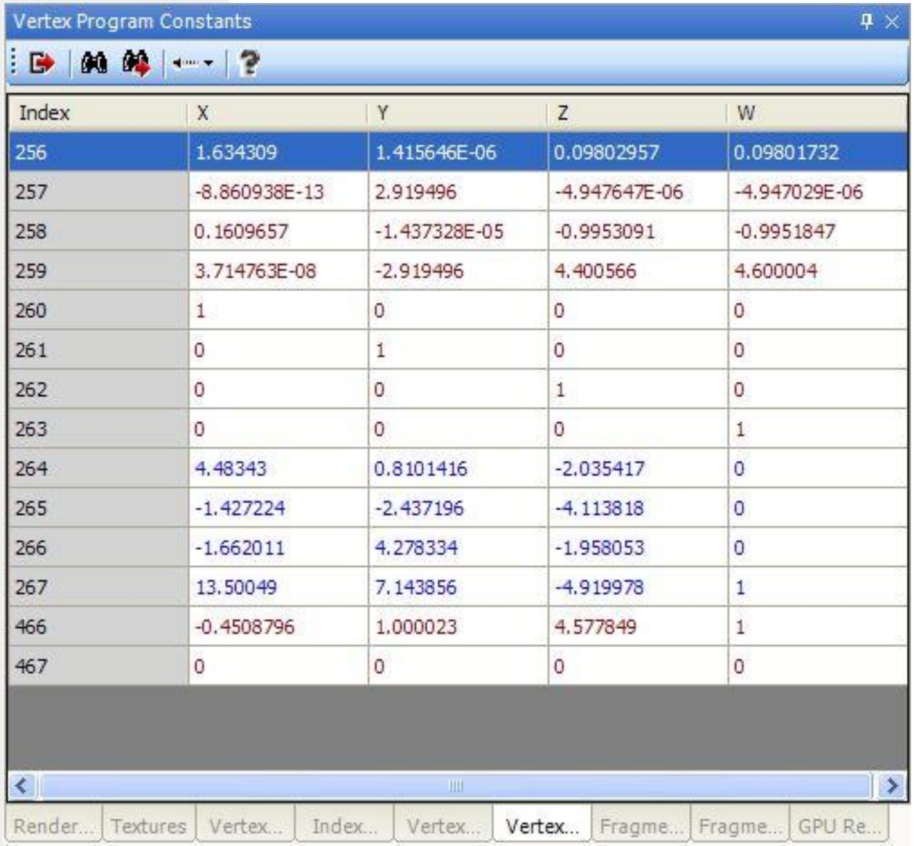


Vertex program 0
Vertex program 1
Vertex program 2
Vertex program 3
Vertex program 4
Vertex program 5
Vertex program 6
Vertex program 7
Vertex program 8
Vertex program 9
Vertex program 10
Vertex program 11
Vertex program 12
Vertex program 13
Vertex program 14
Vertex program 15
Vertex program 16

Attribute mask=[illegible]
Result mask=[illegible]
Max instructions=[illegible]

- Full disassembly
- Stalls highlighted in ***Red***
- Optionally show
  - Instruction latencies
  - Dual issue

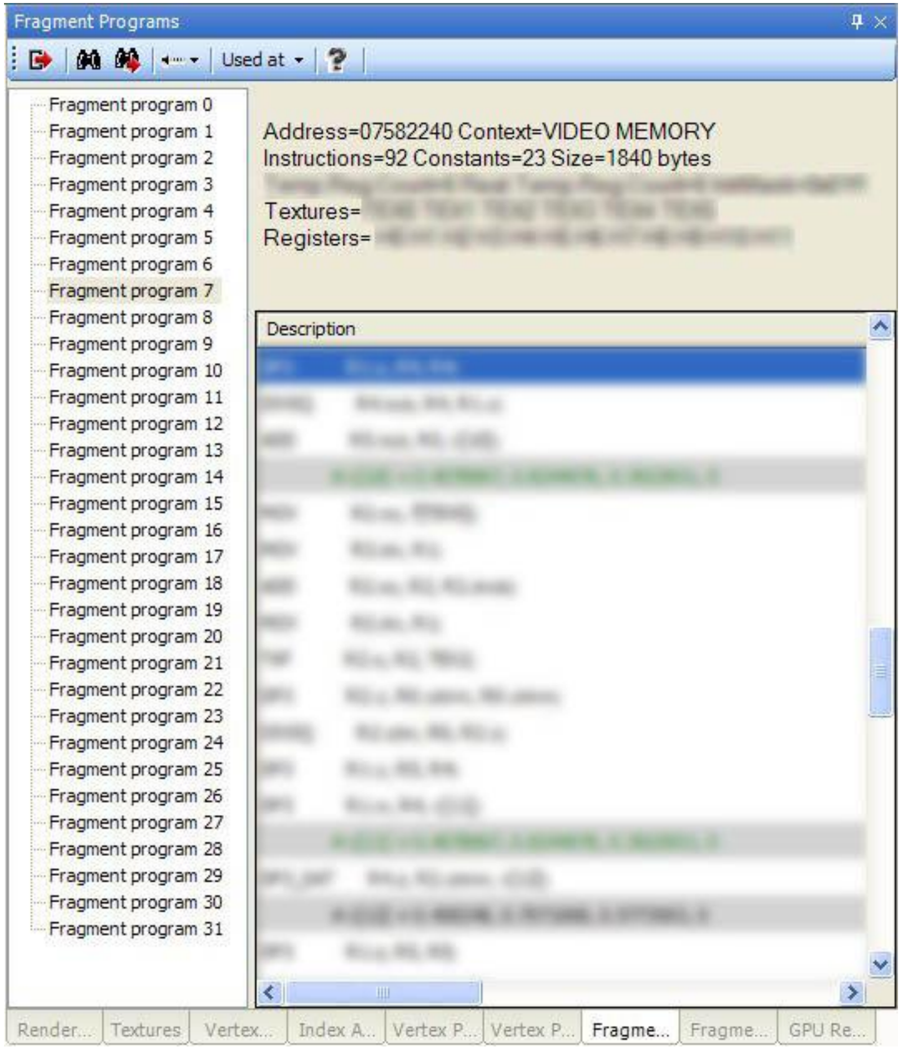
Vertex Program Constants View



Index	X	Y	Z	W
256	1.634309	1.415646E-06	0.09802957	0.09801732
257	-8.860938E-13	2.919496	-4.947647E-06	-4.947029E-06
258	0.1609657	-1.437328E-05	-0.9953091	-0.9951847
259	3.714763E-08	-2.919496	4.400566	4.600004
260	1	0	0	0
261	0	1	0	0
262	0	0	1	0
263	0	0	0	1
264	4.48343	0.8101416	-2.035417	0
265	-1.427224	-2.437196	-4.113818	0
266	-1.662011	4.278334	-1.958053	0
267	13.50049	7.143856	-4.919978	1
466	-0.4508796	1.000023	4.577849	1
467	0	0	0	0

- See Vertex Program Constants
- Used by current Draw Context
- Colour-coded by analysis passes
  - ***Blue*** - newly modified
  - ***Green*** - inherited from previous
  - ***Red*** – redundant sets

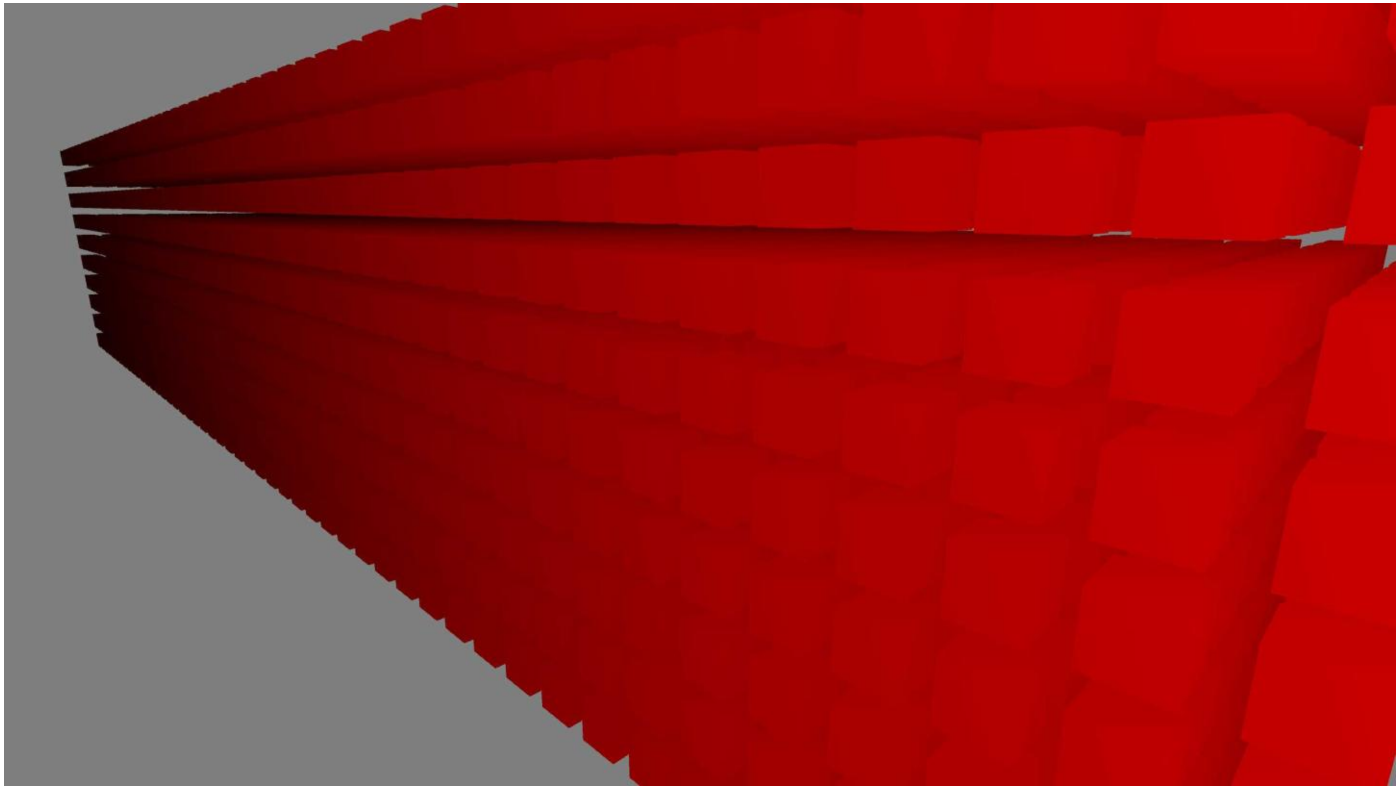
Fragment Programs View



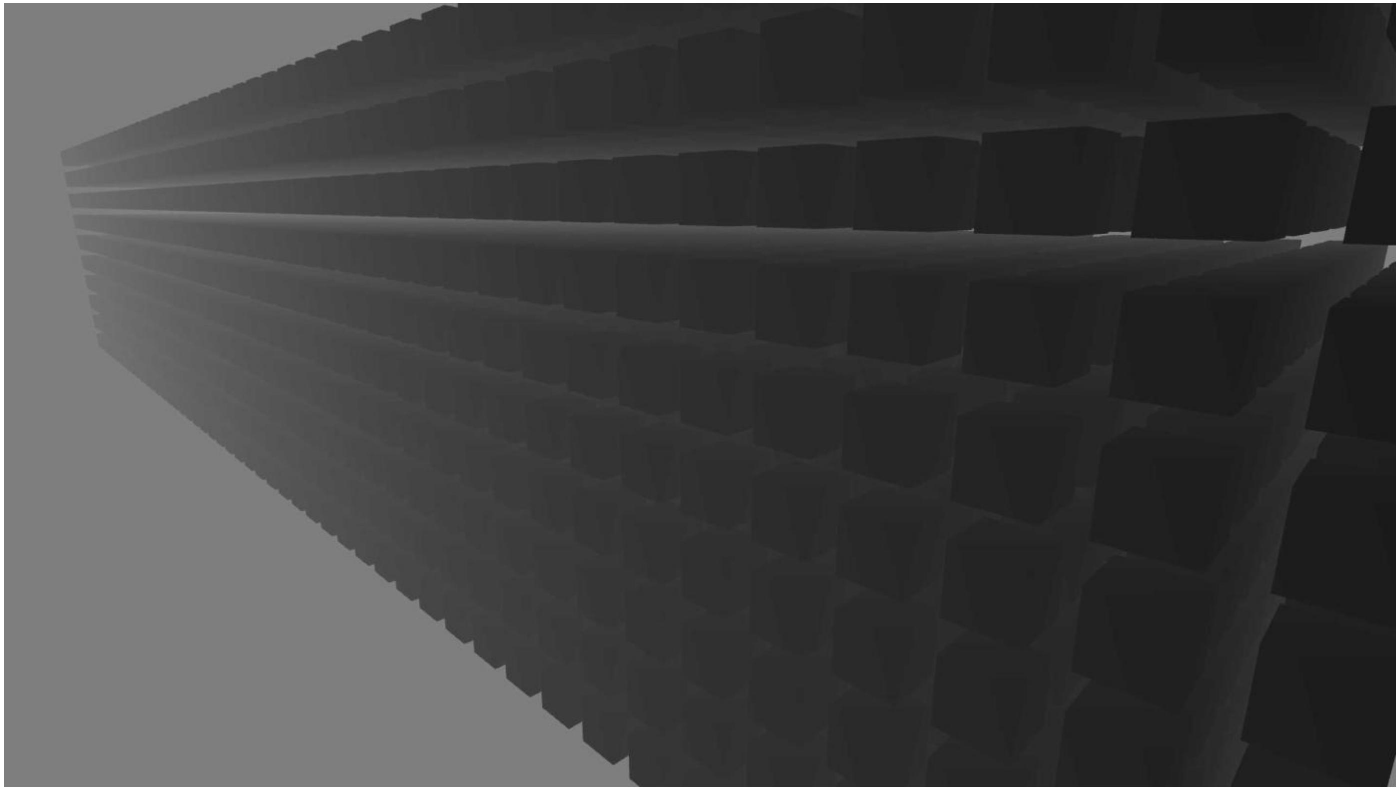
Fragment program 0
Fragment program 1
Fragment program 2
Fragment program 3
Fragment program 4
Fragment program 5
Fragment program 6
Fragment program 7
Fragment program 8
Fragment program 9
Fragment program 10
Fragment program 11
Fragment program 12
Fragment program 13
Fragment program 14
Fragment program 15
Fragment program 16
Fragment program 17
Fragment program 18
Fragment program 19
Fragment program 20
Fragment program 21
Fragment program 22
Fragment program 23
Fragment program 24
Fragment program 25
Fragment program 26
Fragment program 27
Fragment program 28
Fragment program 29
Fragment program 30
Fragment program 31

Address=07582240 Context=VIDEO MEMORY
Instructions=92 Constants=23 Size=1840 bytes
[illegible]
Textures= [illegible]
Registers= [illegible]

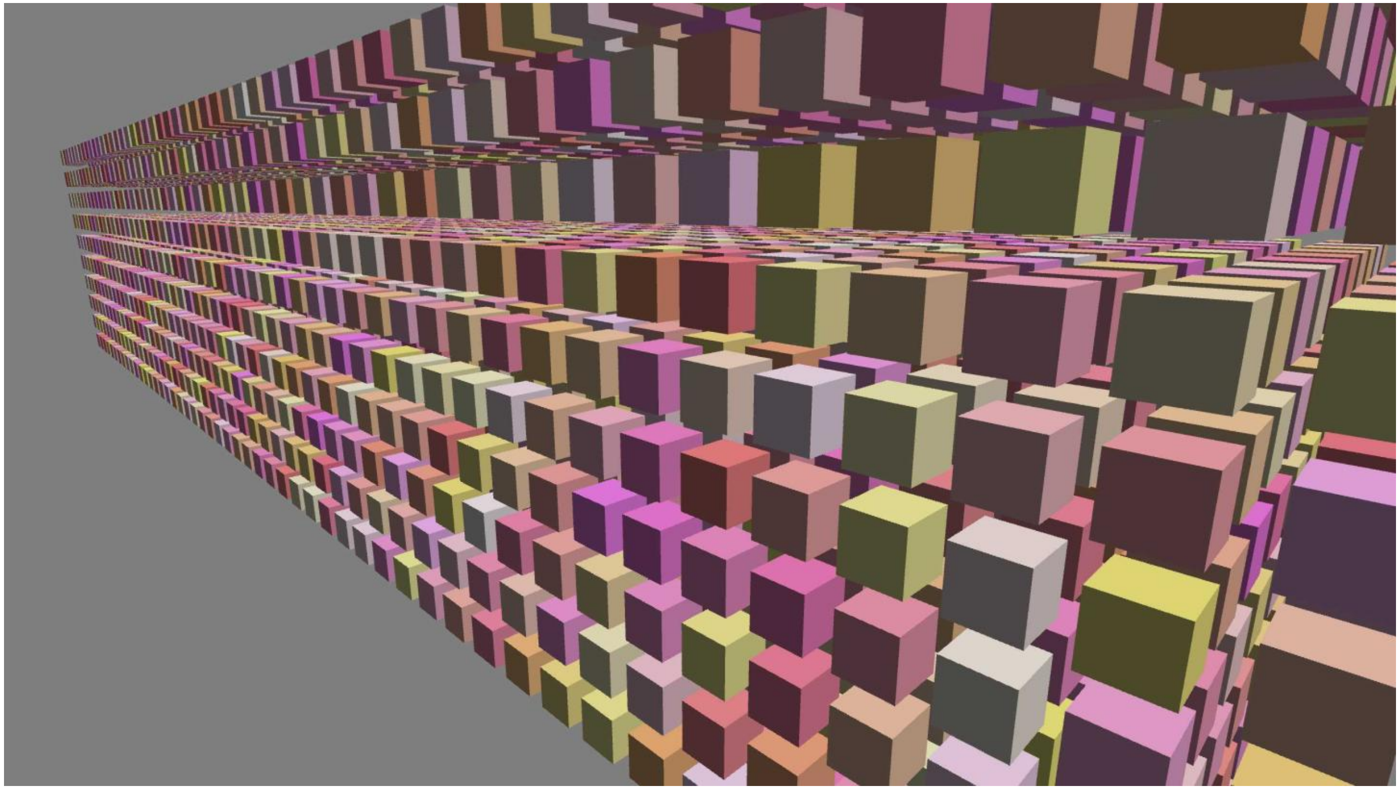
Fragment Program Debugging



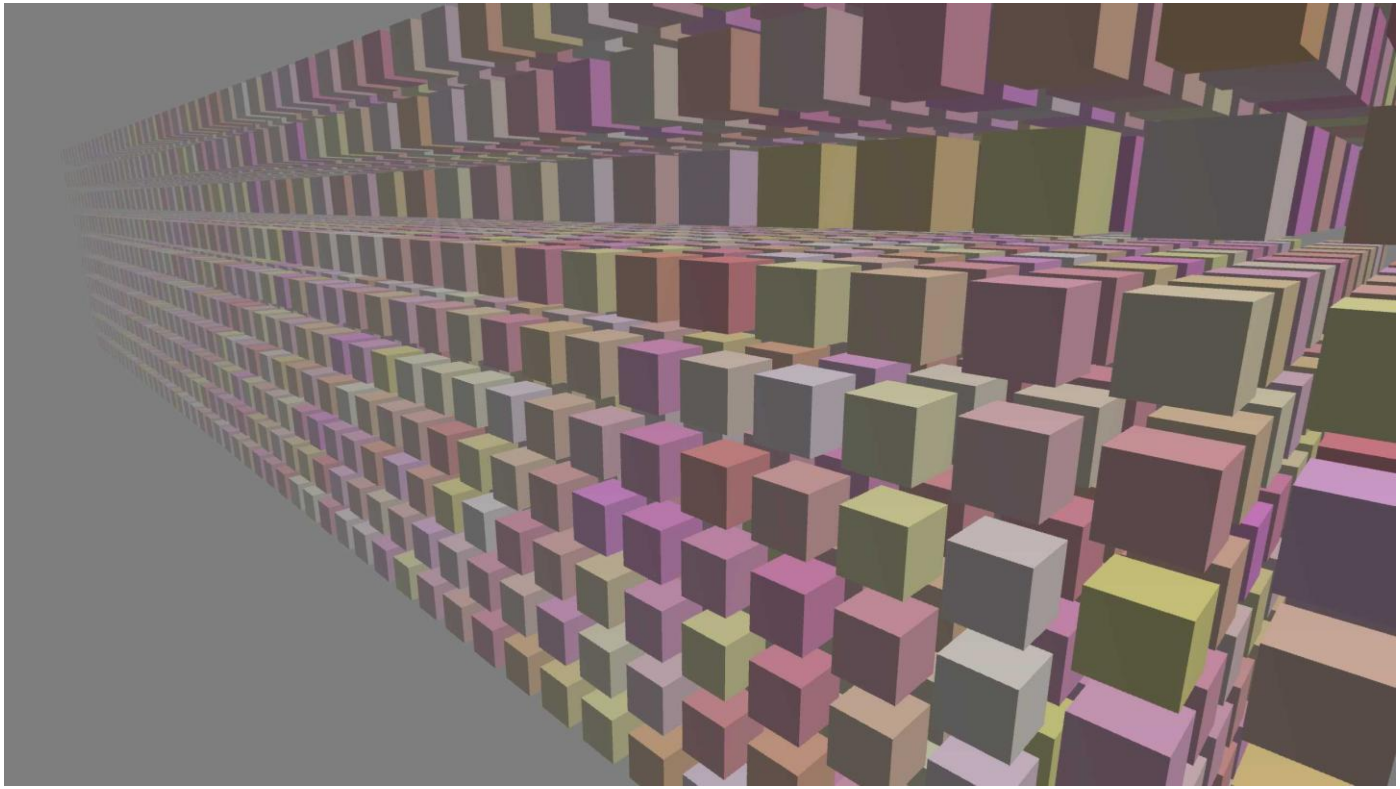
Fragment Program Debugging



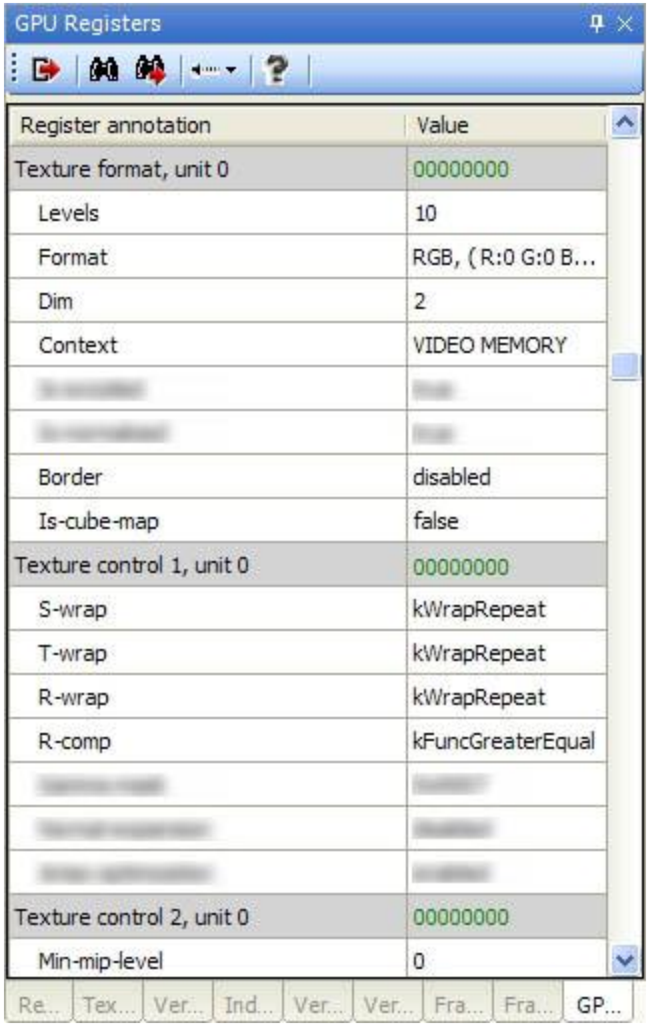
Fragment Program Debugging



Fragment Program Debugging



GPU Registers View



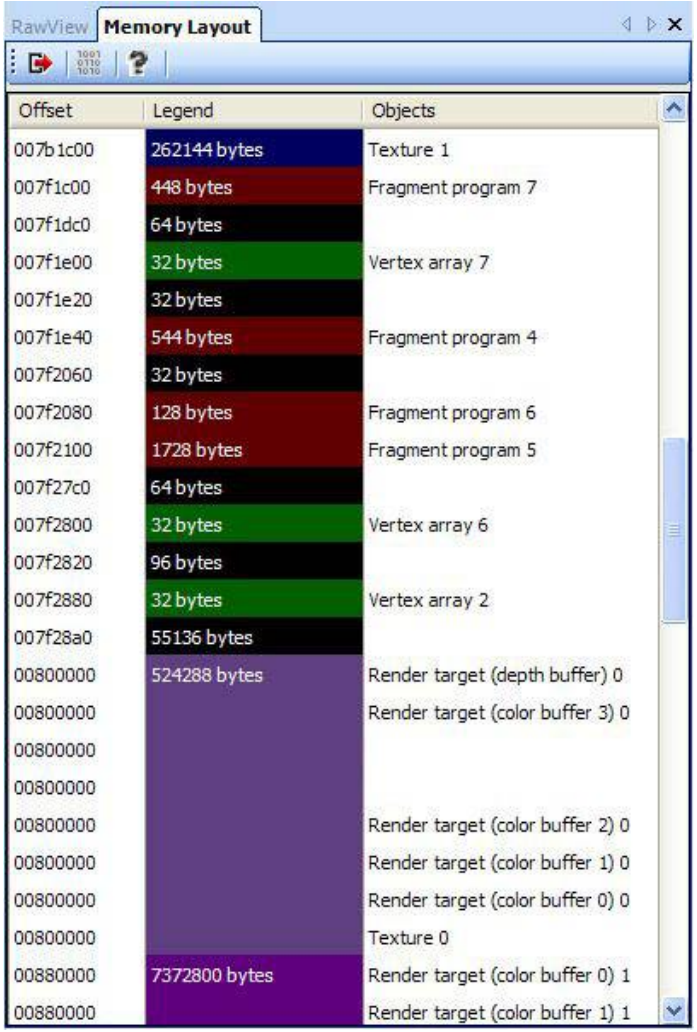
Register annotation	Value
Texture format, unit 0	00000000
Levels	10
Format	RGB, (R:0 G:0 B...
Dim	2
Context	VIDEO MEMORY
[redacted]	[redacted]
[redacted]	[redacted]
Border	disabled
Is-cube-map	false
Texture control 1, unit 0	00000000
S-wrap	kWrapRepeat
T-wrap	kWrapRepeat
R-wrap	kWrapRepeat
R-comp	kFuncGreaterEqual
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
Texture control 2, unit 0	00000000
Min-mip-level	0

- ***Brief Mode*** - registers set in current Draw Context
- ***Descriptive Mode*** - entire register state
- *Verify RSX state is what you expect*

Preview Resource Views

- Exist for all Resource types
- Analyse
  - Draw Context
  - Entire Scene
- All Preview Views have unique features
- Share common functionality - including
  - Cross-referencing
  - Search
  - Memory Dump
  - Export

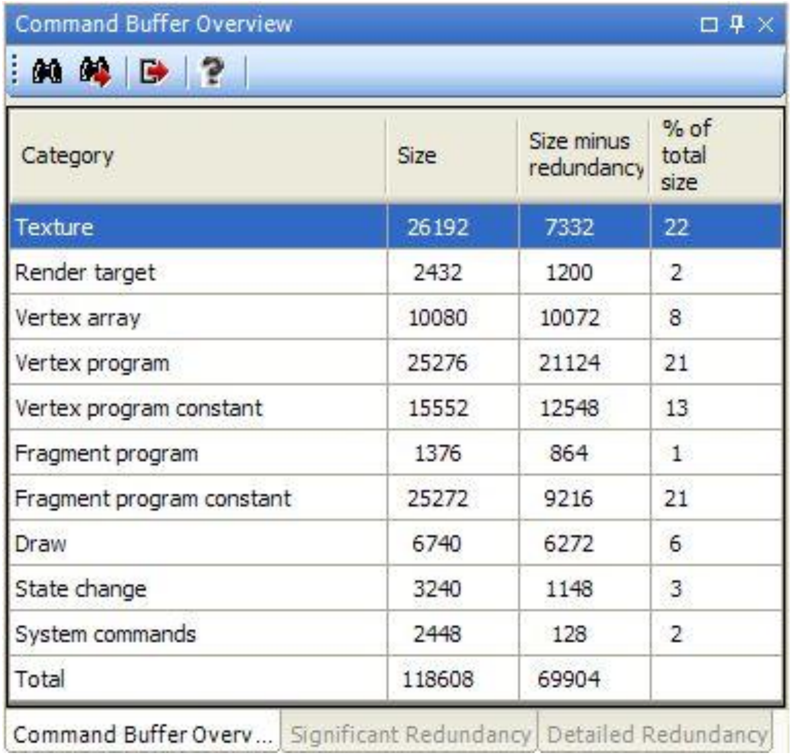
Memory Layout View



Offset	Legend	Objects
007b1c00	262144 bytes	Texture 1
007f1c00	448 bytes	Fragment program 7
007f1dc0	64 bytes	
007f1e00	32 bytes	Vertex array 7
007f1e20	32 bytes	
007f1e40	544 bytes	Fragment program 4
007f2060	32 bytes	
007f2080	128 bytes	Fragment program 6
007f2100	1728 bytes	Fragment program 5
007f27c0	64 bytes	
007f2800	32 bytes	Vertex array 6
007f2820	96 bytes	
007f2880	32 bytes	Vertex array 2
007f28a0	55136 bytes	
00800000	524288 bytes	Render target (depth buffer) 0
00800000		Render target (color buffer 3) 0
00800000		
00800000		
00800000		Render target (color buffer 2) 0
00800000		Render target (color buffer 1) 0
00800000		Render target (color buffer 0) 0
00800000		Texture 0
00880000	7372800 bytes	Render target (color buffer 0) 1
00880000		Render target (color buffer 1) 1

- See your Memory map
- Resource locations in
  - Local Memory
  - Host Memory

Command Buffer Overview



Category	Size	Size minus redundancy	% of total size
Texture	26192	7332	22
Render target	2432	1200	2
Vertex array	10080	10072	8
Vertex program	25276	21124	21
Vertex program constant	15552	12548	13
Fragment program	1376	864	1
Fragment program constant	25272	9216	21
Draw	6740	6272	6
State change	3240	1148	3
System commands	2448	128	2
Total	118608	69904	

- Command Buffer breakdown
- Categorised by Command type

GCM Replay

Profiling Features

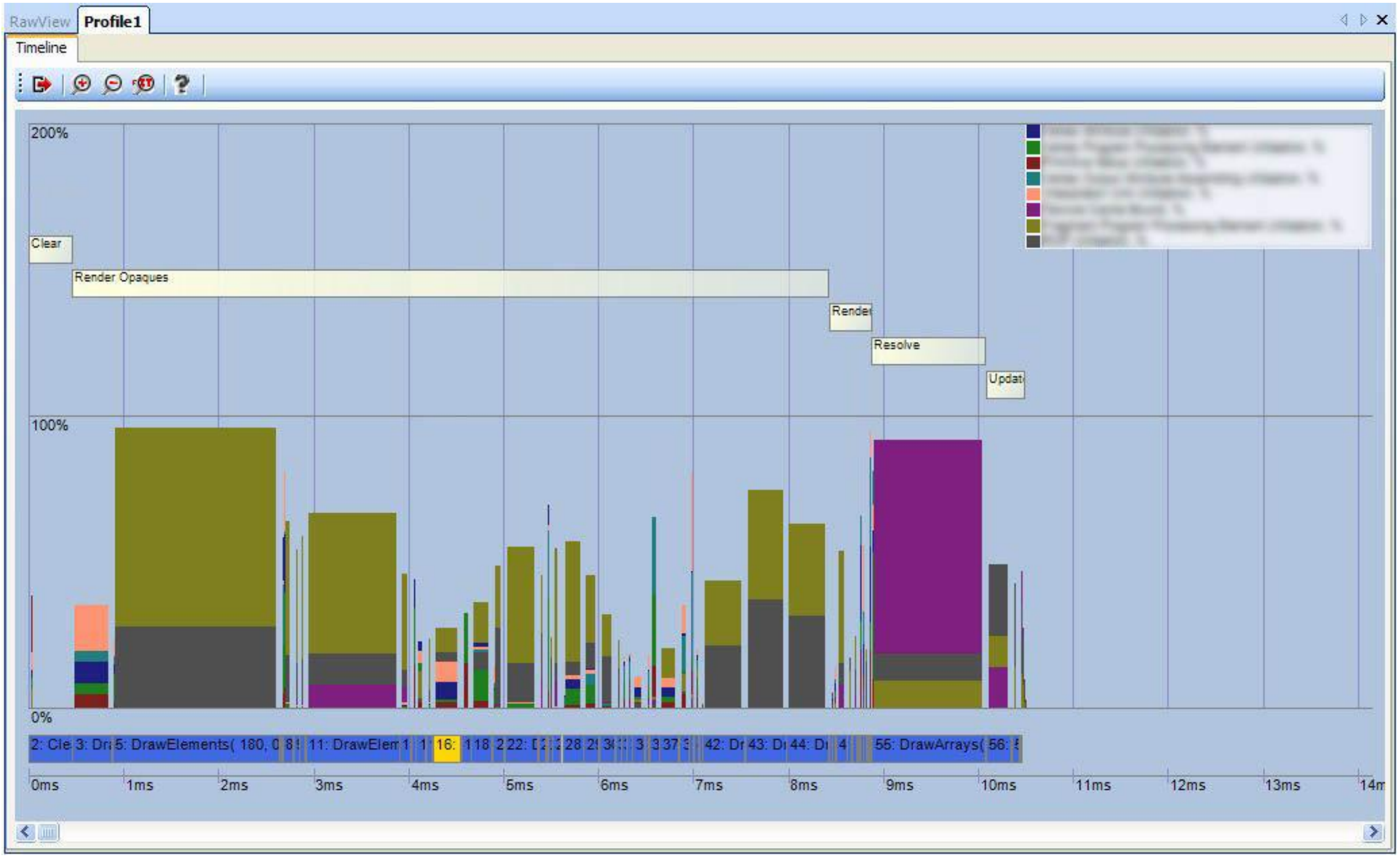
GameDevelopers Conference 07
TAKE CONTROL
March 5-9, 2007 in San Francisco
CMP

GCM Replay Profiling

- Supports a number of flavours
- RSX executes your Command Buffer many times
- Use of RSX hardware counters and timing facilities

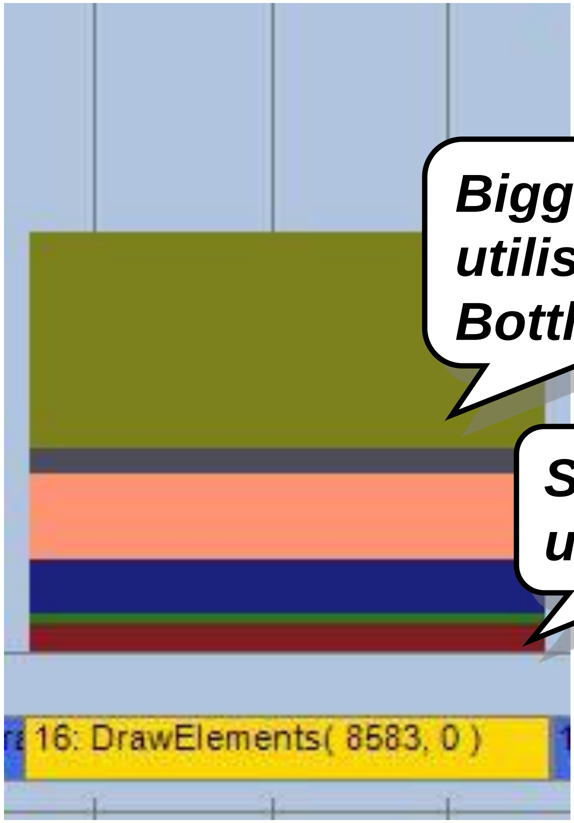
Ensures timings and event counts are accurate

Sub-Unit Utilisation



- See % utilisation of each major pipeline unit

Sub-Unit Utilisation

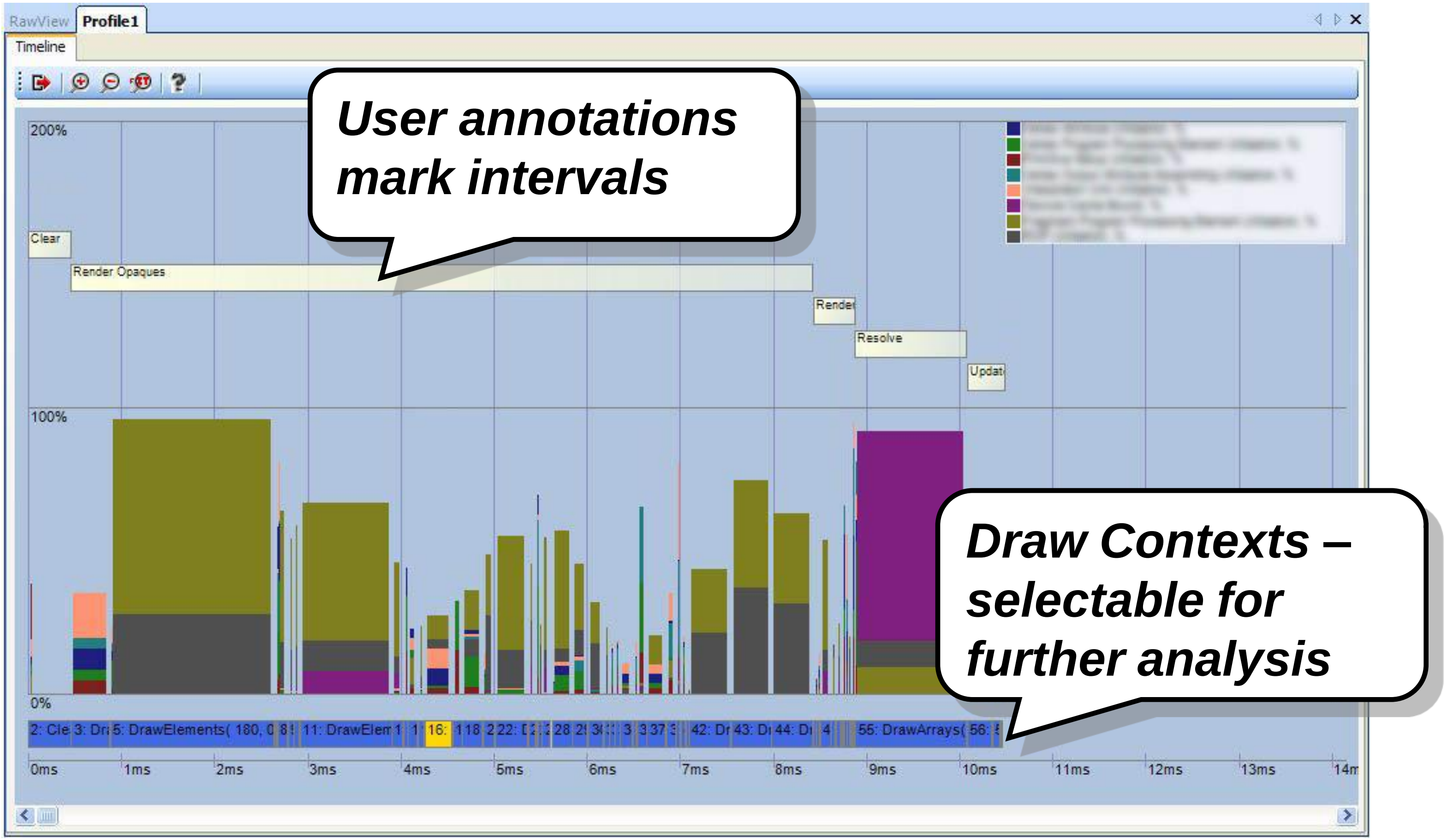


Biggest utilisation - Bottleneck!

Smallest utilisation

- On a Draw Context basis
- Each utilisation is drawn overlapped *stacked)*
- Result - a sorted list of optimisation targets
- indication of the ***bottleneck***

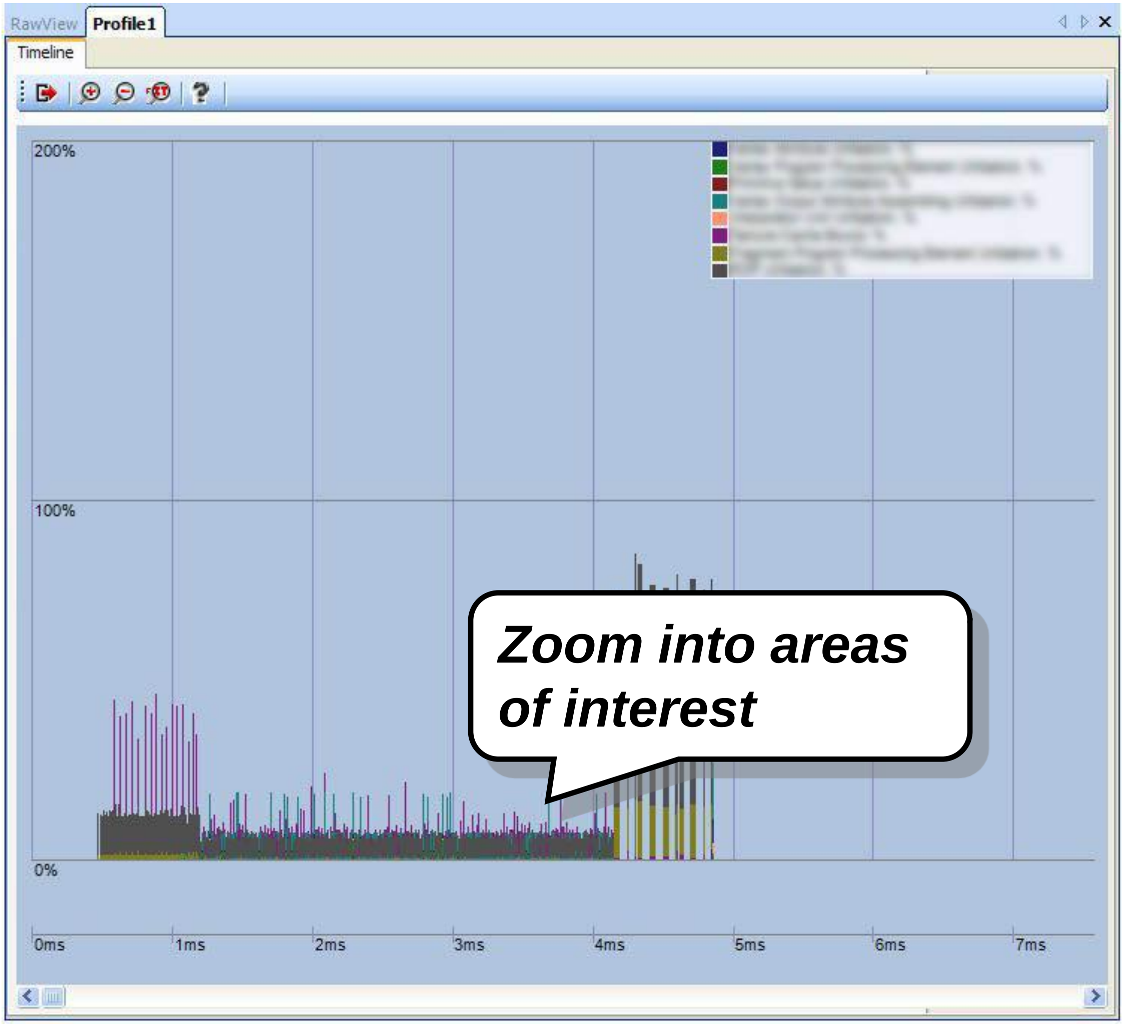
Sub-Unit Utilisation


RawView
Profile1
Timeline
200%
Clear
Render Opaques
Render
Resolve
Updat
100%
0%
2: Cle 3: Dr 5: DrawElements(180, 0 8 11: DrawElem 1 1 16: 1 18 2 22: 2 28 2 3 3 37 42: Dr 43: D 44: D 4 55: DrawArrays(56:
0ms
1ms
2ms
3ms
4ms
5ms
6ms
7ms
8ms
9ms
10ms
11ms
12ms
13ms
14m

User annotations mark intervals

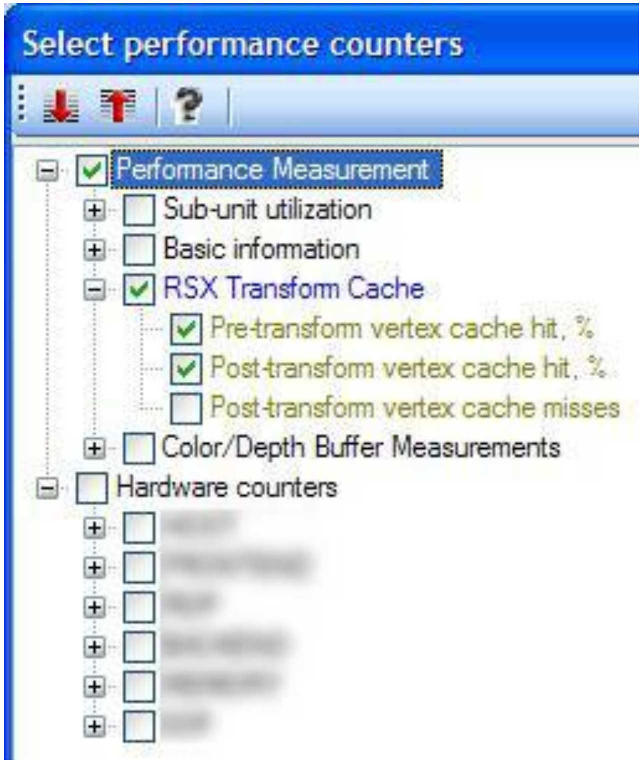
Draw Contexts – selectable for further analysis

Sub-Unit Utilisation



Zoom into areas of interest

Performance Counters

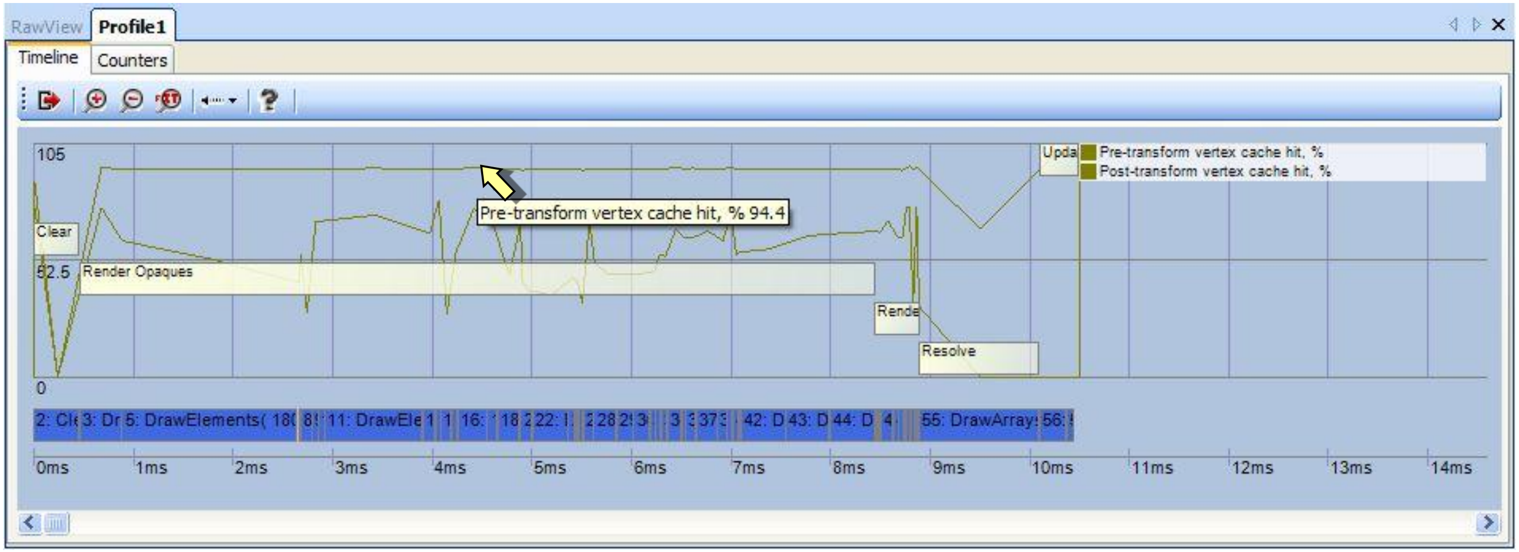


Select performance counters

- ☑ Performance Measurement
  - ☐ Sub-unit utilization
  - ☐ Basic information
  - ☑ RSX Transform Cache
    - ☑ Pre-transform vertex cache hit, %
    - ☑ Post-transform vertex cache hit, %
    - ☐ Post-transform vertex cache misses
  - ☐ Color/Depth Buffer Measurements
- ☐ Hardware counters
  - ☐ [redacted]
  - ☐ [redacted]
  - ☐ [redacted]
  - ☐ [redacted]
  - ☐ [redacted]
  - ☐ [redacted]

- Provide more detail
- Profile additional events
- GCM Replay exposes
  - ***Hardware counters***
  - ***Derived counters***
- Workflow
  - Select counters
  - ***Hit Profile***
  - Analyse *results*

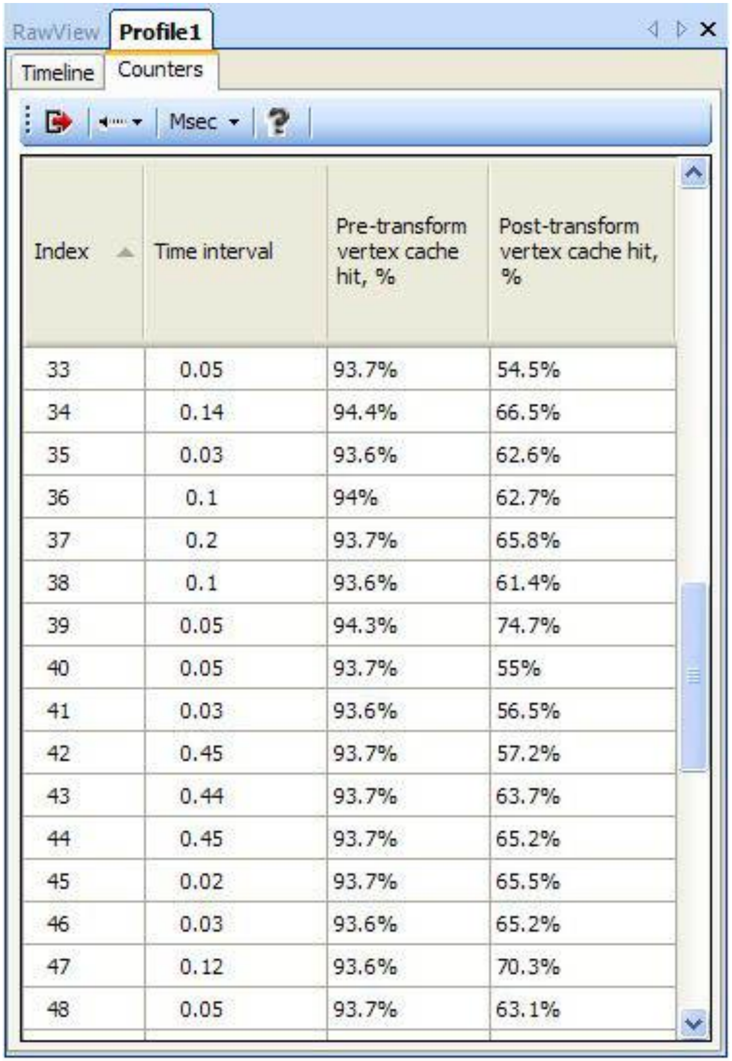
Performance Counters


RawView
Profile1
Timeline
Counters
105
52.5
0
Clear
Render Opaques
Rende
Resolve
Upda
Pre-transform vertex cache hit, %
Post-transform vertex cache hit, %
Pre-transform vertex cache hit, % 94.4
2: Cl 3: Dr 5: DrawElements(18 8 11: DrawEle 1 1 16: 18 2 22: 2 28 2 3 3 3 37 42: D 43: D 44: D 4 55: DrawArray 56:
0ms
1ms
2ms
3ms
4ms
5ms
6ms
7ms
8ms
9ms
10ms
11ms
12ms
13ms
14ms

- Multiple counters graphed simultaneously

Highlight counters of interest

Performance Counters



Index	Time interval	Pre-transform vertex cache hit, %	Post-transform vertex cache hit, %
33	0.05	93.7%	54.5%
34	0.14	94.4%	66.5%
35	0.03	93.6%	62.6%
36	0.1	94%	62.7%
37	0.2	93.7%	65.8%
38	0.1	93.6%	61.4%
39	0.05	94.3%	74.7%
40	0.05	93.7%	55%
41	0.03	93.6%	56.5%
42	0.45	93.7%	57.2%
43	0.44	93.7%	63.7%
44	0.45	93.7%	65.2%
45	0.02	93.7%	65.5%
46	0.03	93.6%	65.2%
47	0.12	93.6%	70.3%
48	0.05	93.7%	63.1%

- Raw counters in tabulated form
- Sort on individual keys
- Select Draw Contexts of interest

Story so far…

- Using GCM Replay you can
  - ***Capture***
  - ***Analyse and Debug***
  - ***Modify and Replay***
  - ***Profile***

What’s the next BIG question?

How do I make it run faster?

RSX

- Deep and complex pipeline
- Large array of rendering options
- Difficult to predict
    - *Effects of your engine changes?*
    - *What optimisations matter most?*

What-If… *I change the anisotropic filtering level on my race track?*

- *How much time would that buy back?*
- *What would it look like?*
- *What's the best compromise?*

What-If… *I re-optimise my meshes?*

For example

- *Convert triangle strips to triangle lists*
- *Interleave all attribute streams*
- *Optimise index tables for all vertex caches*

- *How would these effect RSX performance?*

What-If… *If I write a near **perfect** visibility culler?*

- *That culls on a per triangle basis*
- *Removes all triangles outside the viewport*
- *All back-facing triangles*
- *Zero-area degenerate triangles*
- *Micro-triangles that miss all pixel centres*

And…

It runs on the SPUs and removes all triangles before they hit the RSX

How much time would that buy back?

With GCM Replay you can answer all these questions…

Without touching a single line of code

GCM Replay ***What-Ifs***

- A new form of *Conditional Profiling*

- Make fundamental changes to your
  - Command Buffer
  - Resources

- All from within GCM Replay

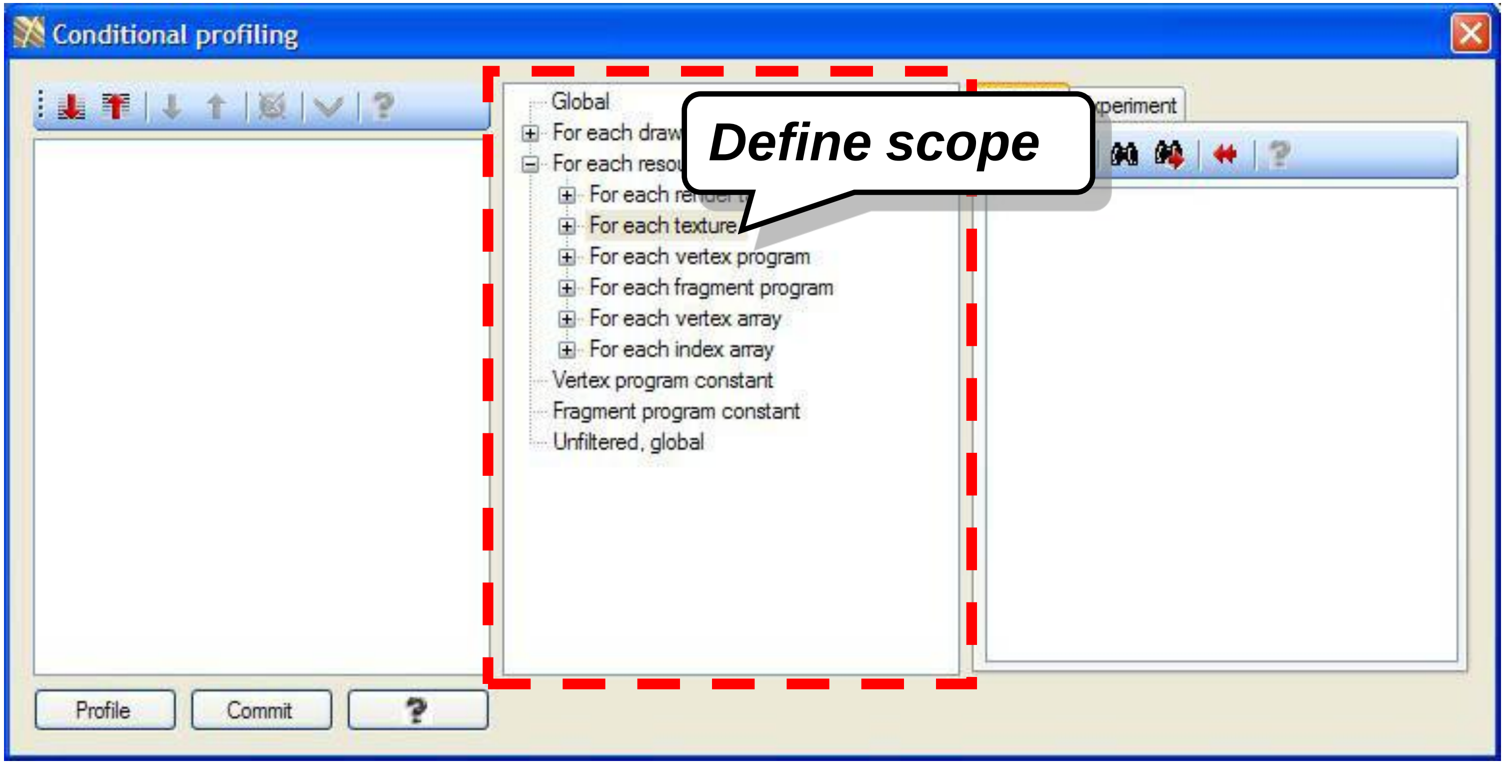
- Measure observed performance difference
  - See % ***Gain*** or ***Loss***

What-if… Example

My Scene

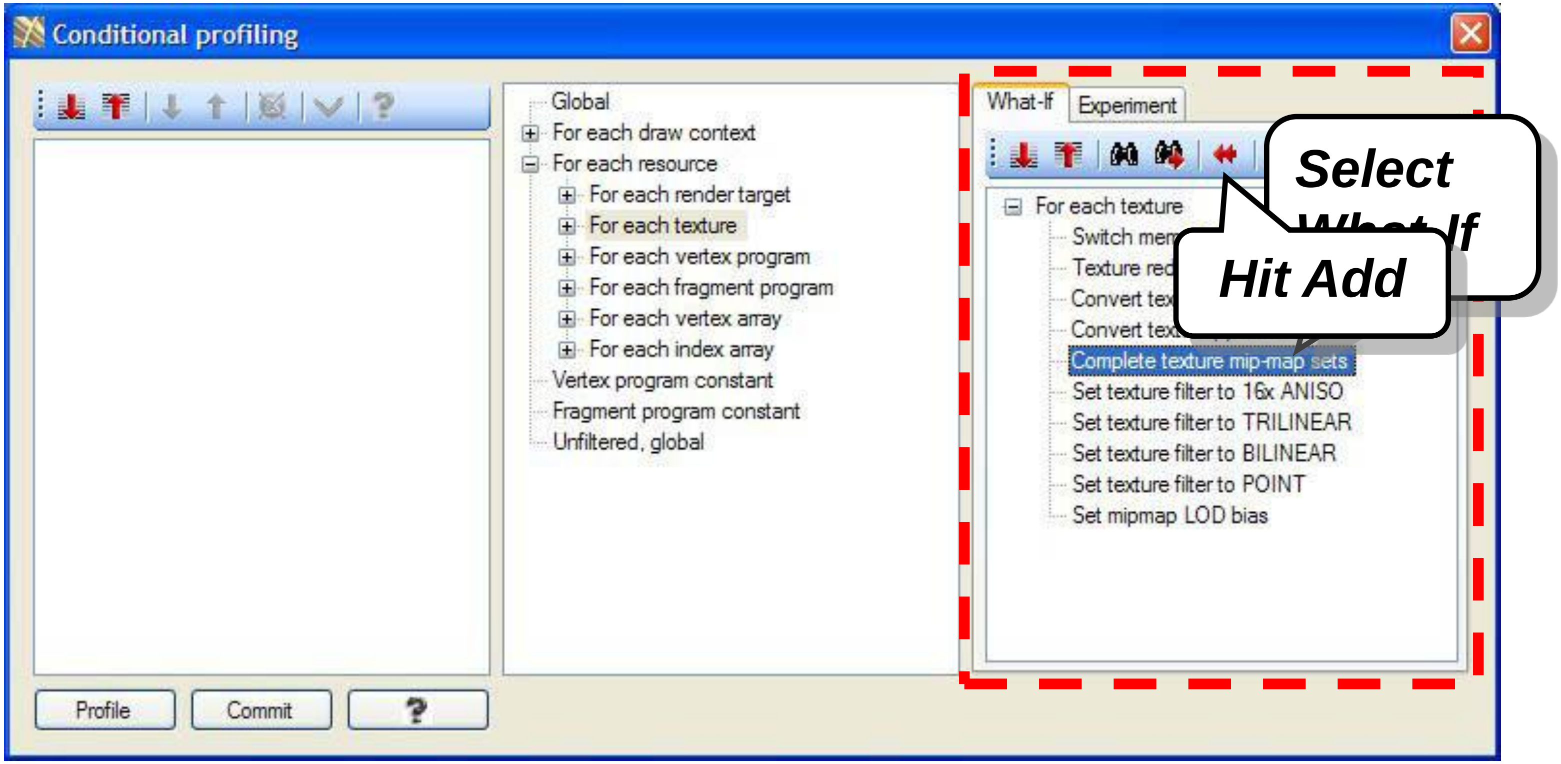
- Through analysis I realise…
- *Not all my textures are compressed*
- *Not all my textures have a mip-chain*

What-If… Workflow



- Apply to
  - A Draw Context ***OR*** all Draw Contexts
  - A Resource ***OR*** all Resources of that type

What-If… Workflow



- Global
- For each draw context
- For each resource
  - For each render target
  - For each texture
  - For each vertex program
  - For each fragment program
  - For each vertex array
  - For each index array
- Vertex program constant
- Fragment program constant
- Unfiltered, global

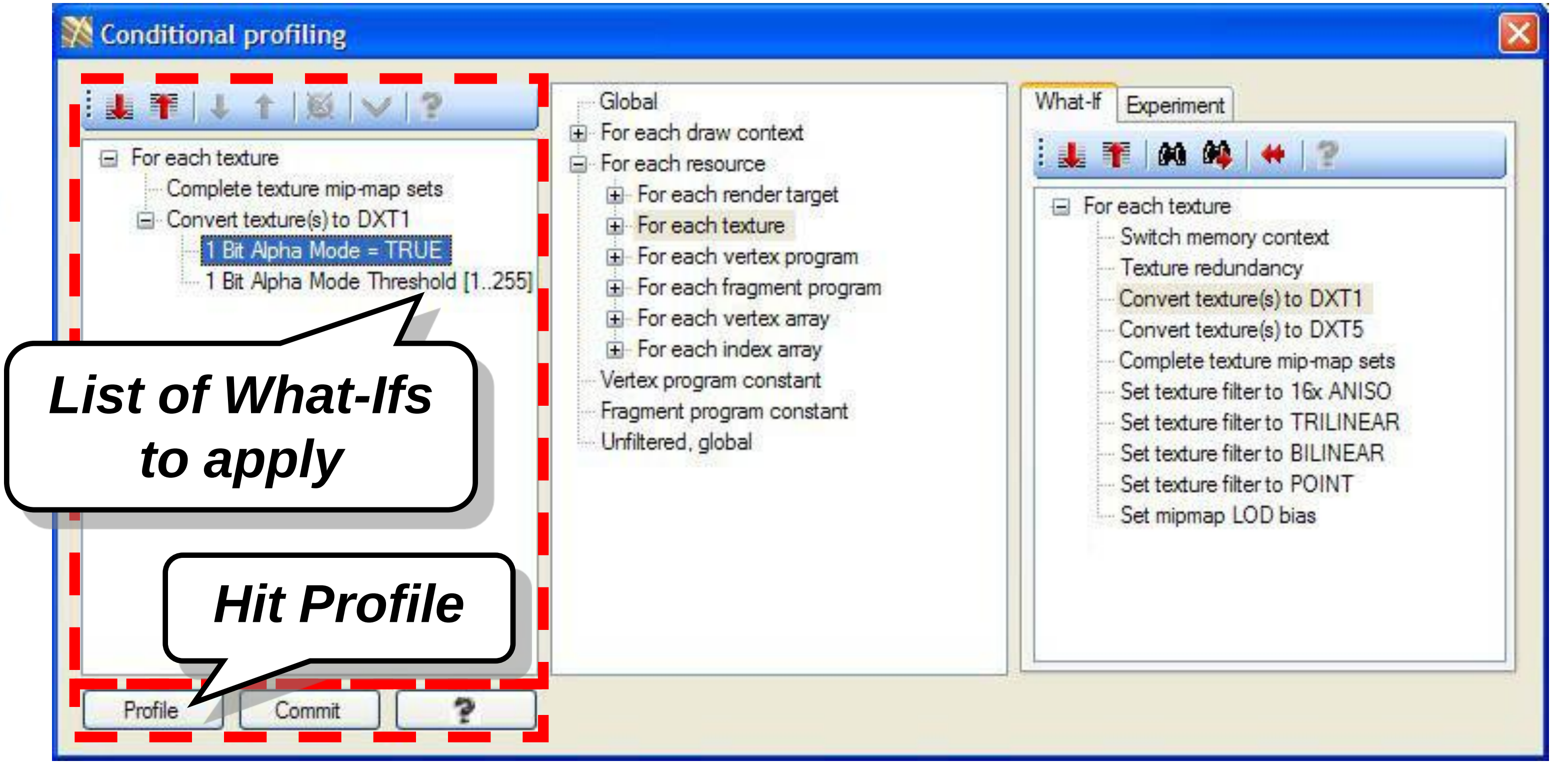
- For each texture
  - Switch mem
  - Texture red
  - Convert tex
  - Convert text
  - Complete texture mip-map sets
  - Set texture filter to 16x ANISO
  - Set texture filter to TRILINEAR
  - Set texture filter to BILINEAR
  - Set texture filter to POINT
  - Set mipmap LOD bias

Select What-If

Hit Add

- Select ***What-Ifs*** from a filtered set

What-If… Workflow



Conditional profiling

For each texture
- Complete texture mip-map sets
- Convert texture(s) to DXT1
  - 1 Bit Alpha Mode = TRUE
  - 1 Bit Alpha Mode Threshold [1..255]

Global
- For each draw context
- For each resource
  - For each render target
  - For each texture
  - For each vertex program
  - For each fragment program
  - For each vertex array
  - For each index array
- Vertex program constant
- Fragment program constant
- Unfiltered, global

What-If | Experiment

For each texture
- Switch memory context
- Texture redundancy
- Convert texture(s) to DXT1
- Convert texture(s) to DXT5
- Complete texture mip-map sets
- Set texture filter to 16x ANISO
- Set texture filter to TRILINEAR
- Set texture filter to BILINEAR
- Set texture filter to POINT
- Set mipmap LOD bias

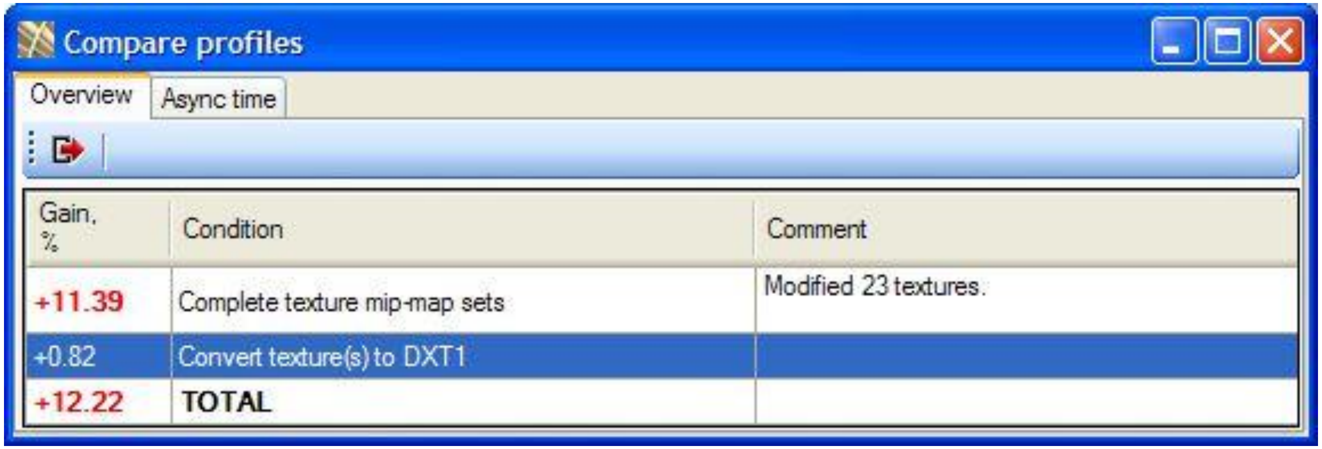
- Optionally tweak ***What-If*** parameters

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

What-Ifs – Behind the Scenes

- Profile baseline
- For each ***What-If*** condition
  - Make modifications
  - Profile
  - Save results
- In this example
  - Generate mip-chains
  - Compress Textures
  - Modify Texture state in Command Buffer

What-If… Results



Gain, %	Condition	Comment
+11.39	Complete texture mip-map sets	Modified 23 textures.
+0.82	Convert texture(s) to DXT1	
+12.22	TOTAL	

- Summarise
  - % gain for each ***What-If***
  - Comments on actual modifications made
- Total % gain for all ***What-Ifs***

Instantly see the change in performance

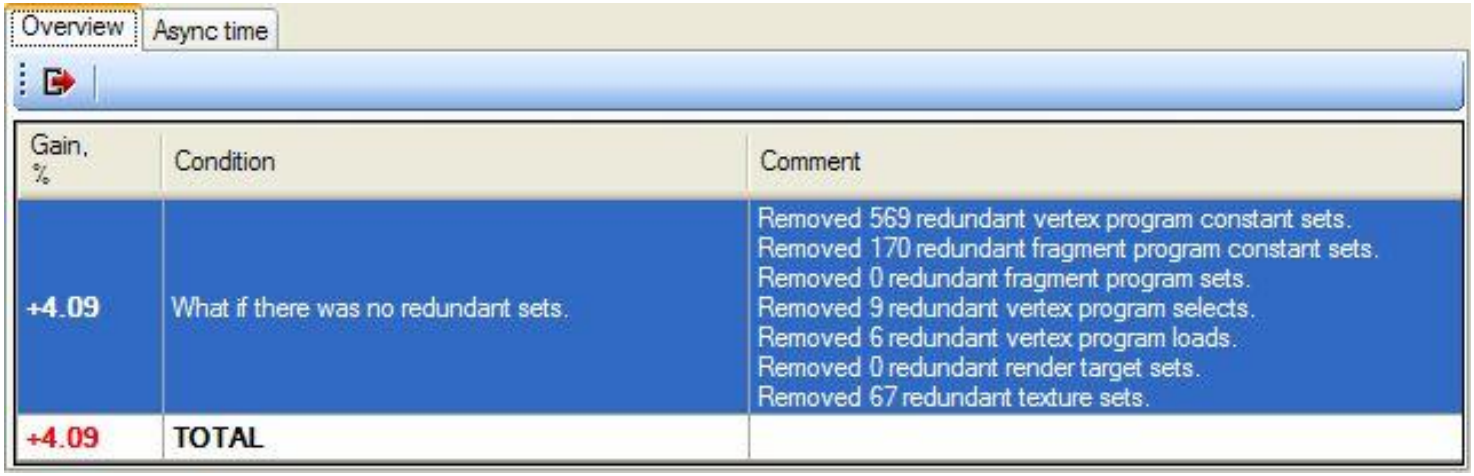
What-If… Workflow

- Two options
  - ***Profile*** – with new ***What-Ifs,*** same baseline
  - ***Commit*** **–** make new baseline
- Iterate Process
  - Performance target reached
  - We’re as close as we can get

Incorporate the optimisations into your game

What-If… Remove Redundancy

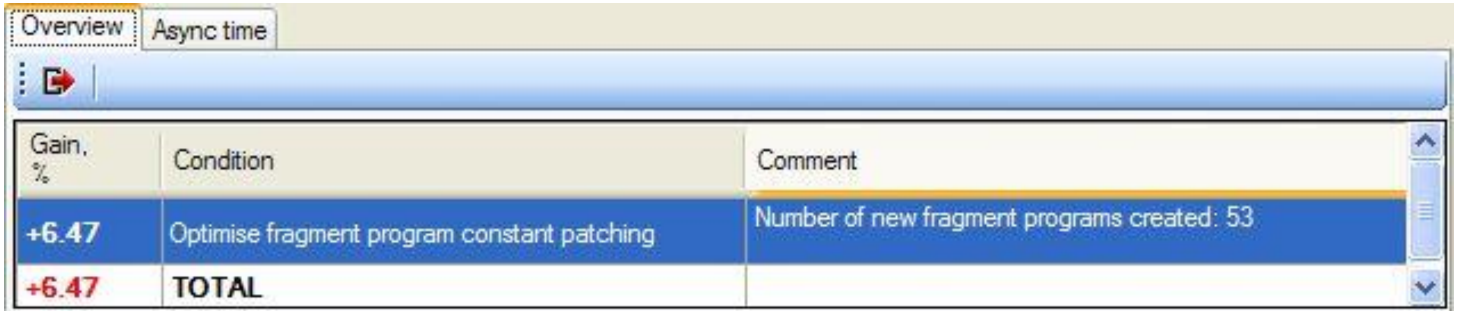
- What if we remove all redundant commands?



Gain, %	Condition	Comment
+4.09	What if there was no redundant sets.	Removed 569 redundant vertex program constant sets. Removed 170 redundant fragment program constant sets. Removed 0 redundant fragment program sets. Removed 9 redundant vertex program selects. Removed 6 redundant vertex program loads. Removed 0 redundant render target sets. Removed 67 redundant texture sets.
+4.09	TOTAL	

What-If… Optimise FP Constant Patching

- What if all constants are set externally?
- Directly patched by PPU or SPUs



Gain, %	Condition	Comment
+6.47	Optimise fragment program constant patching	Number of new fragment programs created: 53
+6.47	TOTAL	

Moroccan Scene Results

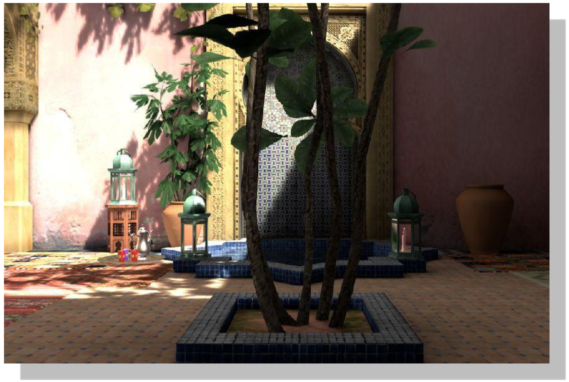
- Applying all four ***What-Ifs…***

Complete Texture Mip-chains	*+11.39%*
Compress Textures	*+0.82%*
Remove Global Redundancy	*+4.09%*
Optimise Fragment Constant Patching	*+4.73%*

- *Total Performance Gain*

~21%

faster than original scene



PLAYSTATION®Edge Demo

sweek
The Real League
SAMSUNG
TDK
SANY
OFFICIAL BEER PREMIER LEAGUE
TICKETS

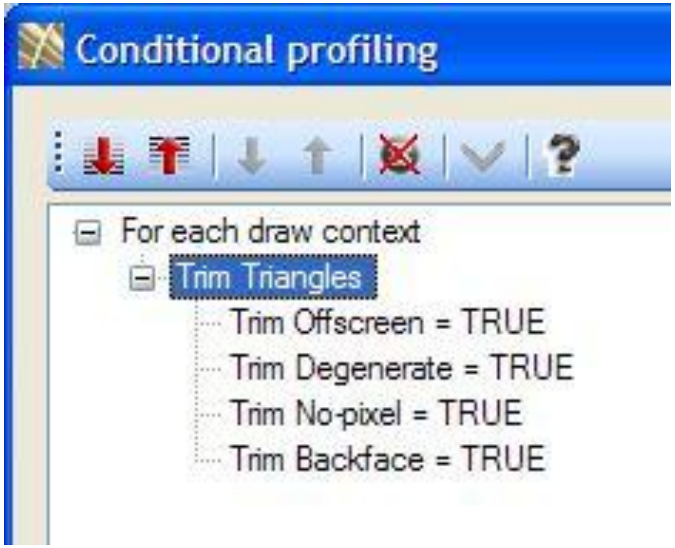
What-If… Trim Triangles

- What if we trim all triangles
  - Off-screen
  - Back facing
  - Degenerate
  - Don’t hit any pixel centres

- Set scope to all Draw Contexts

- Enable all triangle tests

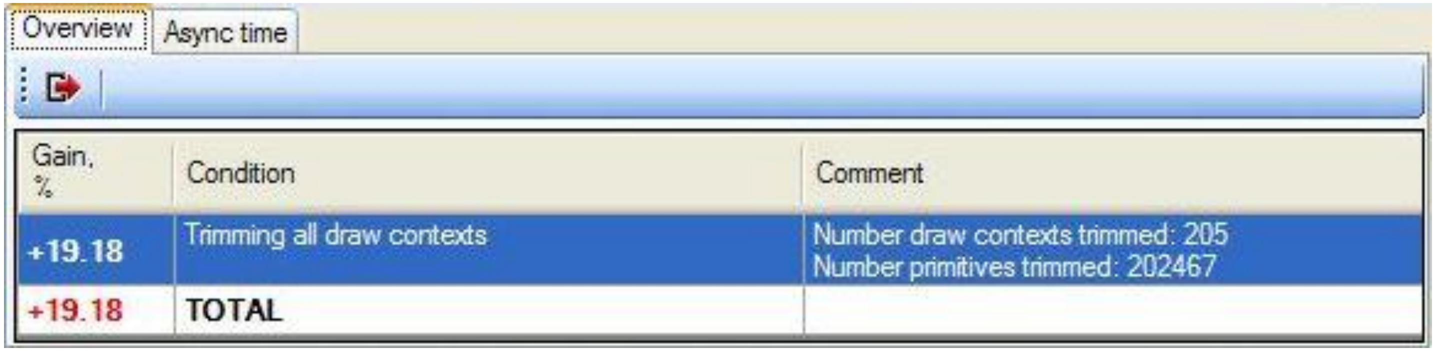
- ***Hit Profile***



For each draw context
- Trim Triangles
  - Trim Offscreen = TRUE
  - Trim Degenerate = TRUE
  - Trim No-pixel = TRUE
  - Trim Backface = TRUE

What-If… Trim Triangles

- *19%* performance gain from Triangle Culling alone



Gain, %	Condition	Comment
+19.18	Trimming all draw contexts	Number draw contexts trimmed: 205 Number primitives trimmed: 202467
+19.18	TOTAL	

GCM Replay *What-Ifs*

- Evaluate fundamental engine changes
  - Without actually having to make them
- Provide rapid feedback
  - *See **What-If…** results within minutes*
- Help you make informed decisions
  - *What optimisations matter most?*
  - *How close are we to theoretical maximums?*
- Help avoid wasting time on fruitless changes
  - *Save precious development time*

The ***What-Ifs***

- ***Global***
  - Remove Redundancy
- ***For each Draw Context***
  - Optimise all Triangle Lists
  - Convert Strips to Lists
  - Change Stream Interleaving
  - Trim Triangles
  - Trim Batches
  - Depth-only Pass
  - Disable unused Attributes
  - Disable unused Interpolators
  - Sort Batches Front to Back
  - Replace with Single Colour FP
  - Non-disclosed x3
  - Perfect Early-Z Settings
  - Convert to Indexed Drawing
  - Disable unused Clear Components
  - Remove redundant Clears
  - Remove completely filled Clears
  - Remove non-varying Attributes
- ***For each Render Target***
  - Remove redundancy
  - *Non-disclosed*
- ***For each Texture***
  - Switch Memory Context
  - Remove redundancy
  - *Convert to DXT1*
  - *Convert to DXT5*
  - *Complete mip-chain*
  - *Override filtering modes*
  - *Override LOD bias*
- ***For each Vertex Program***
  - Remove redundancy
  - Non-disclosed
- ***For each Fragment Program***
  - Optimise Constant Patching
  - Remove redundancy
  - Non-disclosed x2

GCM Replay ***Experiments***

- Extend the ***What-If*** concept
- ***Experiments*** – automatically replay selected ***What-Ifs*** with many times
- Finds the optimal settings for your game
- Example – Texture Placement ***Experiment***

GCM Replay – The Future

- More ***What-Ifs***
- More ***Experiments***
- Extend ***Edit-and-Continue***
  - Modify all Resource types
  - Hot-load replacement Resources
- ***Vertex and Fragment Program Debugging***

GCM Replay

BETA Release - March 2007