Performance Gains Achieved Through Modern OpenGL in the Siemens DirectModel Rendering Engine

Jeremy Bennett [Senior Software Engineer, Siemens PLM Software]

Michael Carter [Senior Key Expert, Siemens PLM Software]

DirectModel: History

- Developed as joint venture between EAI and HP as large model visualization in 1997
- Now the graphics engine underlying all Siemens Teamcenter Visualization products

- Originally implemented against OpenGL 1.0 and Starbase (who remembers this?)
- Now pushing the envelope into OpenGL 4.5 features

DirectModel: Support

- Platforms: Windows, Linux, Mac, iOS, Android
- GPUs: Nvidia Quadro & Grid, AMD FireGL & FirePro, Intel HD 4500>
- Support variety of OpenGL levels

OpenGL 1.1	
OpenGL 1.5	Vertex Buffer Objects
OpenGL 2.1	Shaders
OpenGL 3.1	Uniform Buffer Objects
OpenGL 4.3	Multi Draw Elements Indirect
OpenGL 4.5	Direct State Access

Presentation

State Architecture

- Current architecture and how it maps to GL

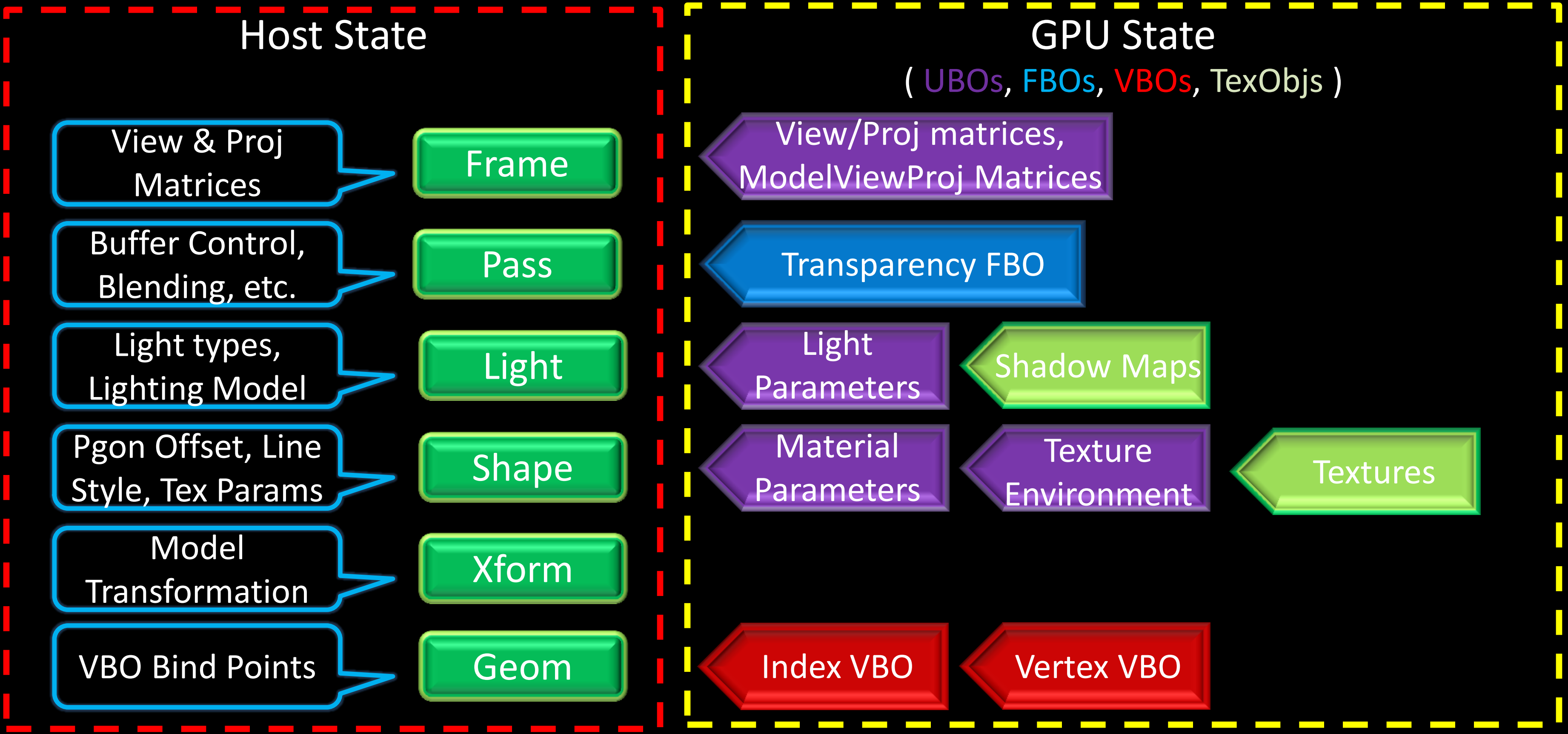
Pipeline Optimizations

- No single magic bullet but rather a whole continuum
- Motivated by
  - Real World Experiences
  - GTC S3032: Advanced SceneGraph Rendering Pipeline
  - GTC S4379: OpenGL Scene-Rendering Techniques
  - GDC ‘14: Approaching Zero Driver Overhead

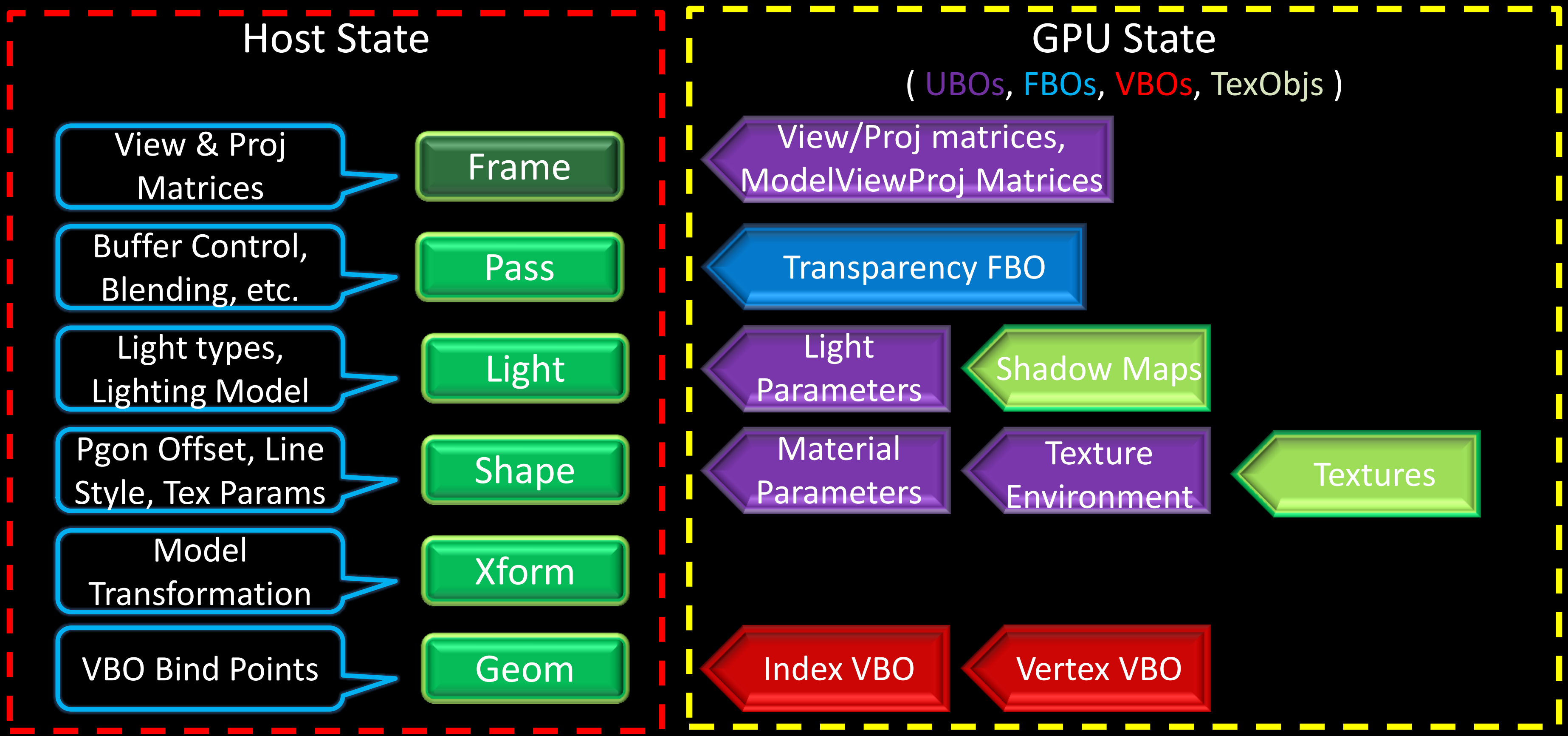
State Architecture: Motivation

- Design priorities are flexibility, high performance, and maintainability (slightly different from a game engine; must be able to gracefully cope with unexpected situations)
- Previous architecture based on managing discrete OpenGL state changes incrementally
- New State object represents comprehensive state for rendering a single object – including the geometry
- Important for the middleware architecture to match the underlying underlying GAPI architecture

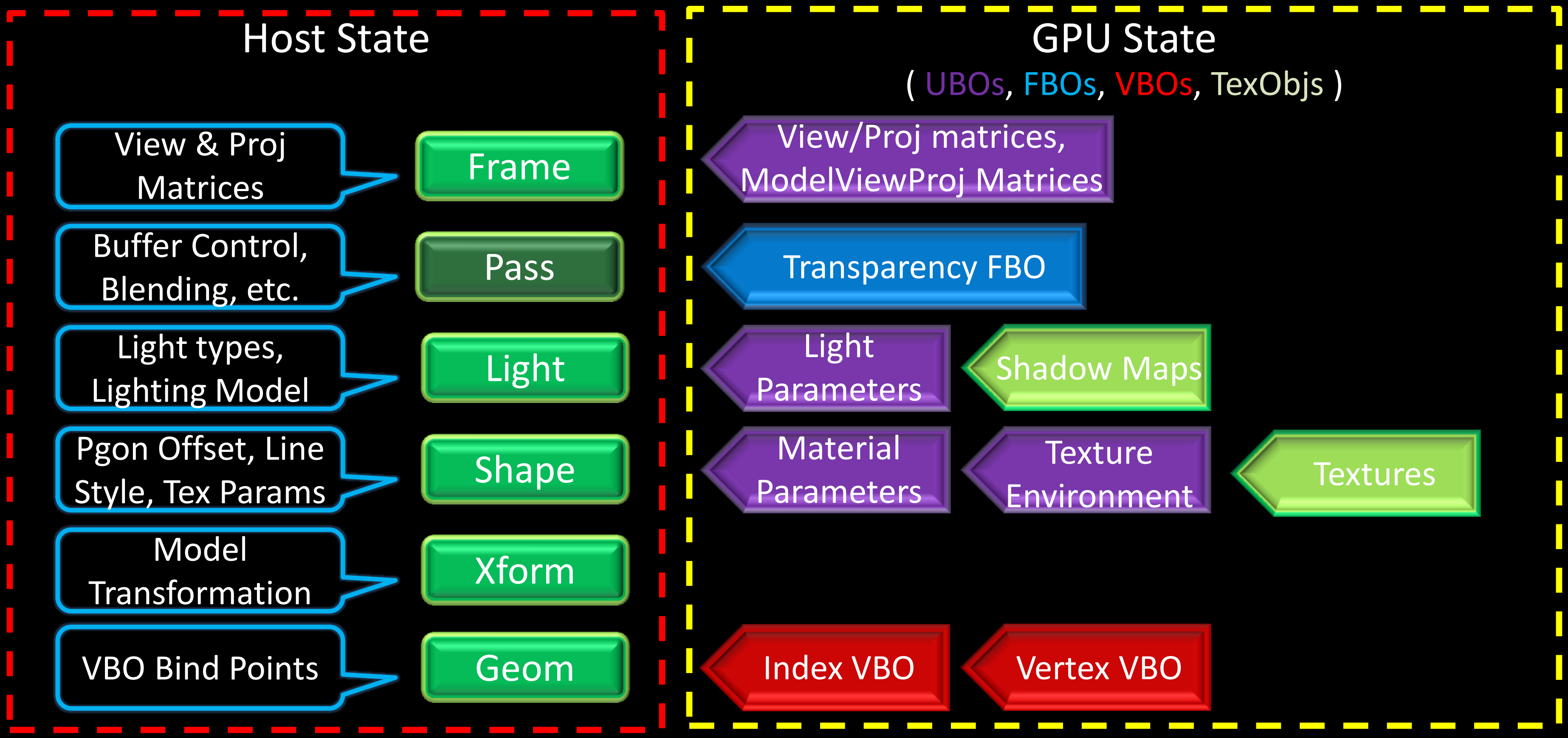
State Architecture: Block Diagram



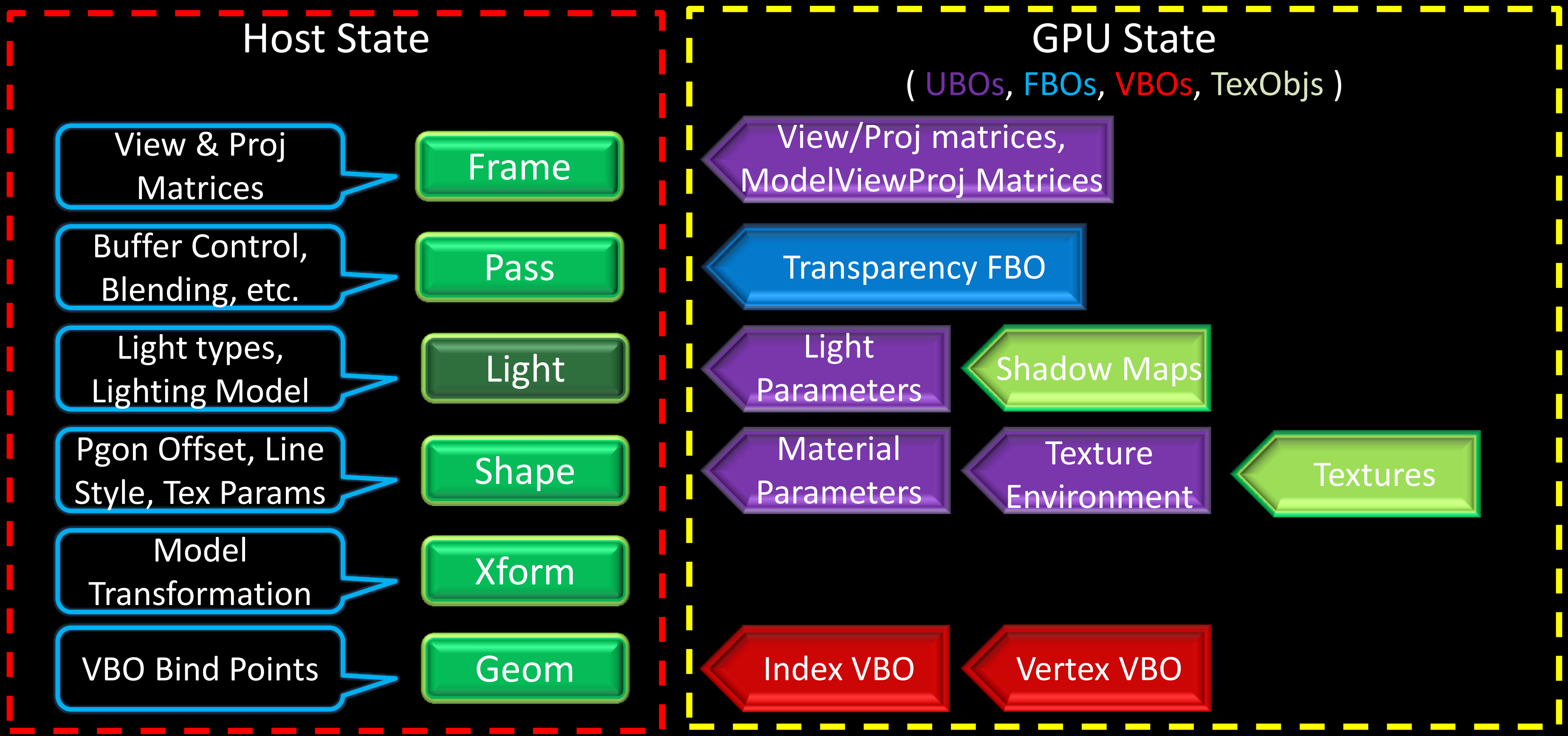
State Architecture: Frame State



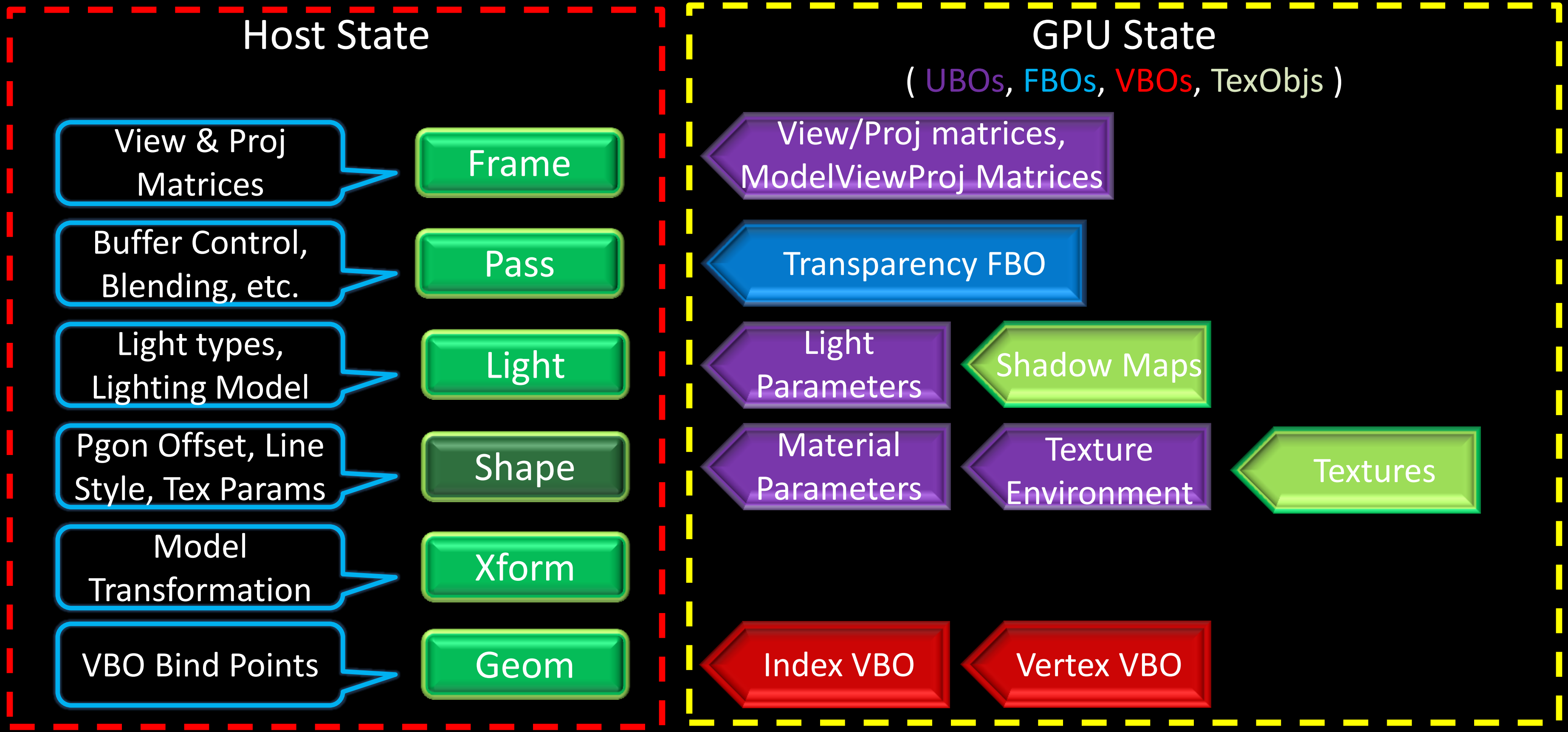
State Architecture: Pass State



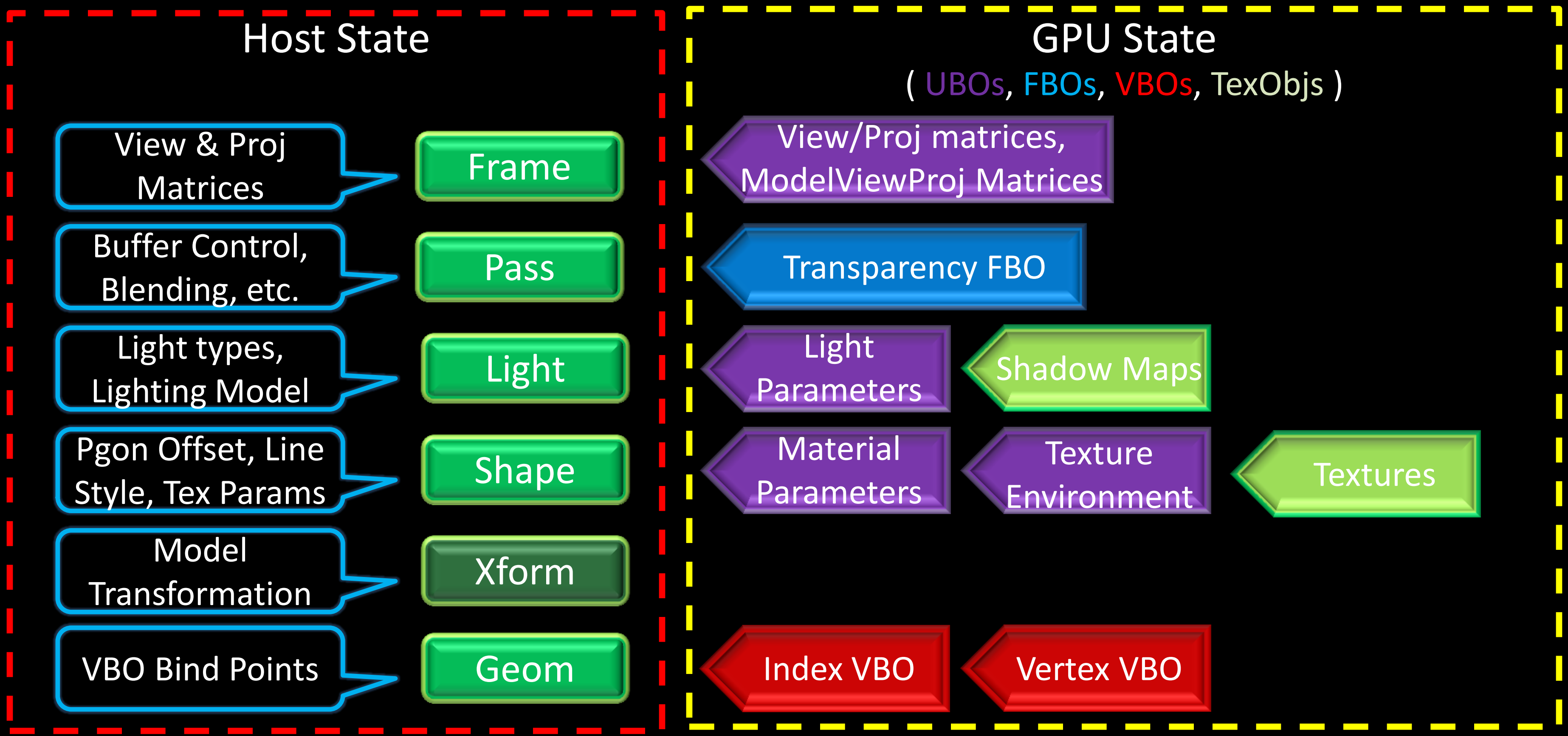
State Architecture: Light State



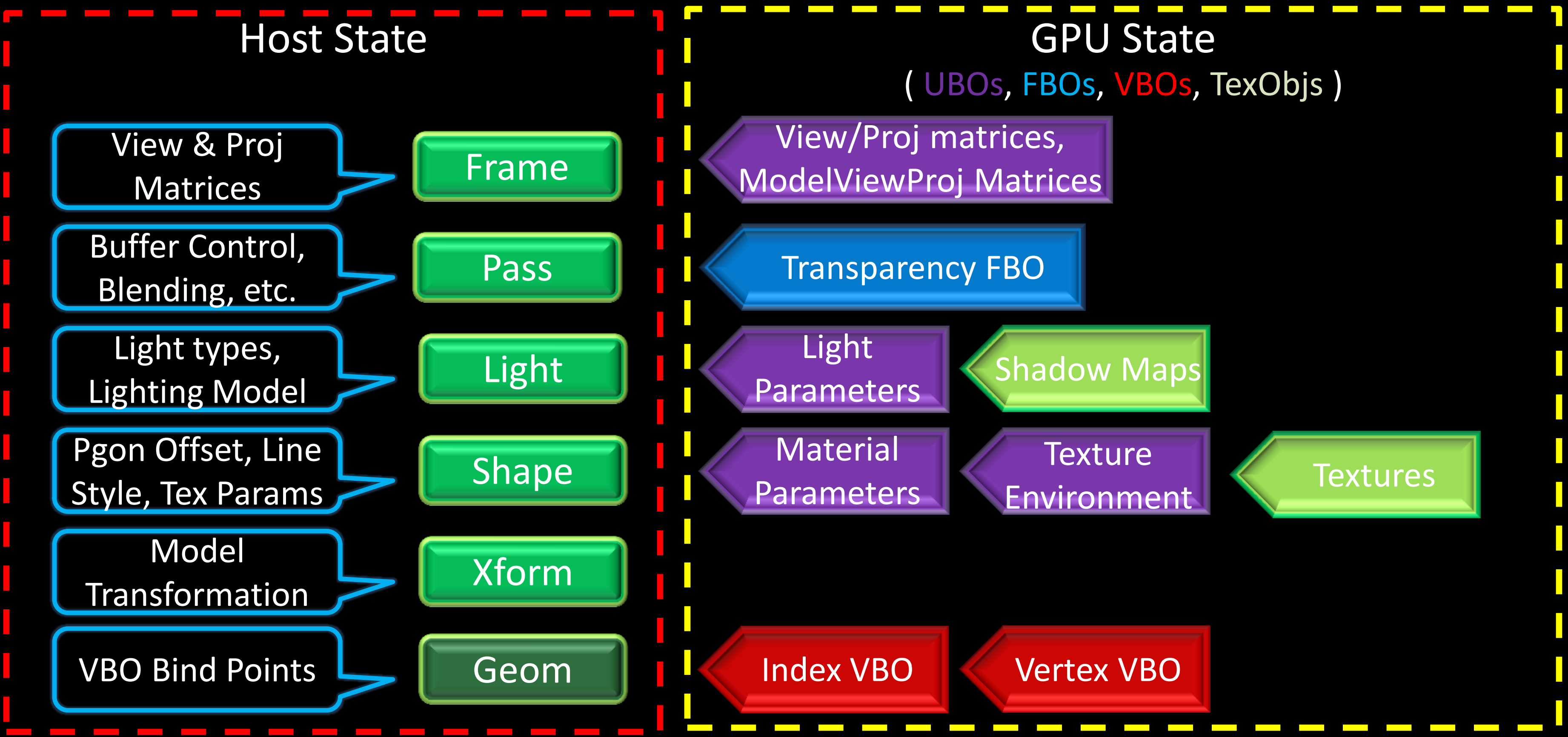
State Architecture: Shape State



State Architecture: Xform State



State Architecture: Geom State



Optimization: Strategy

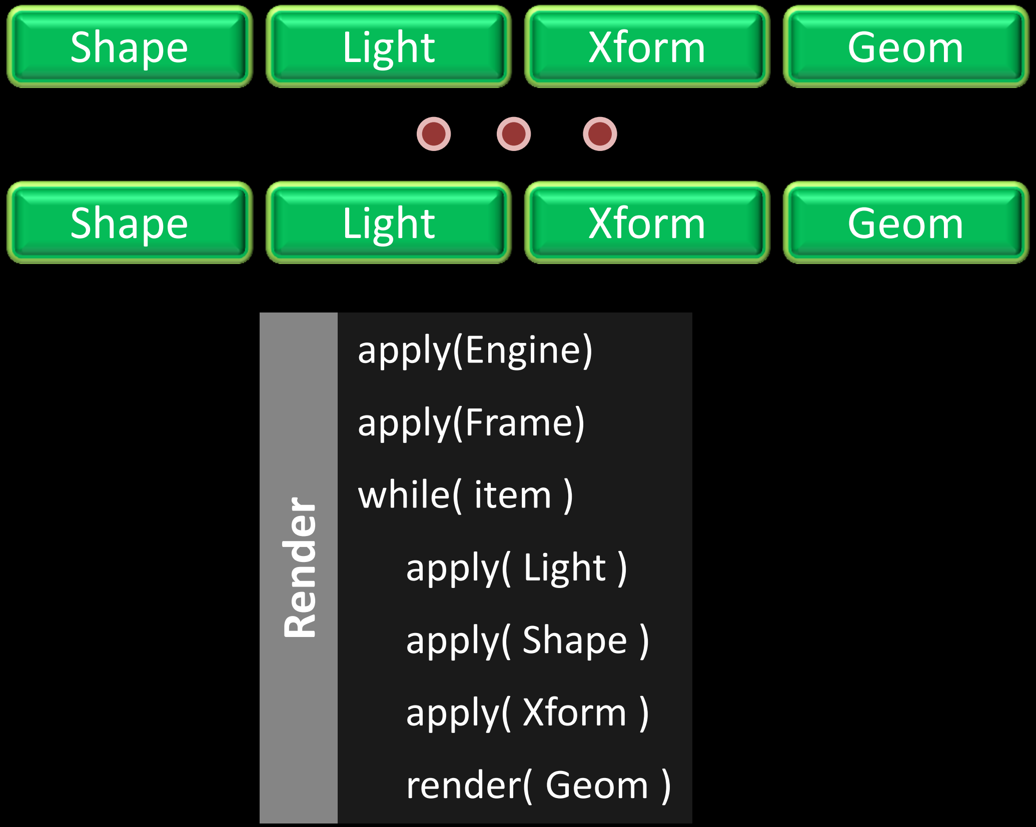
- Reduce CPU Overhead
  - Minimize OpenGL Calls
  - Minimize State Updates
- Increase GPU Performance
  - Use faster APIs
  - Prevent Stalls

Areas of Exploration

- Index | Display Lists | VBOS
- Fixed Function Pipeline | Shaders
- State Calls | Uniforms | Uniform Buffer Objects
- DrawRangeElements | MultiDrawElementsIndirect | CommandList
- Buffers | Persistently Mapped | Bindless

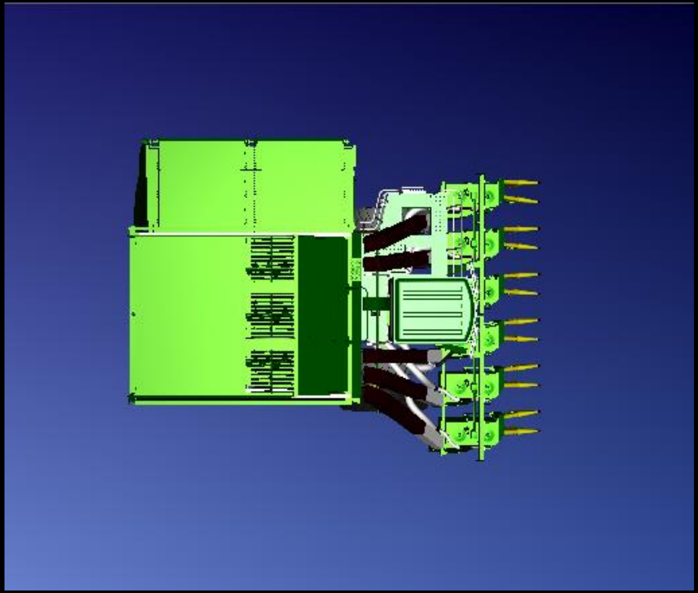
Optimization: Rendering Pipeline

- Generate Render List
  - Use CPU or GPU
- Iterate over Render List
  - Apply State
  - Render Geometry



Optimization: Test Procedure

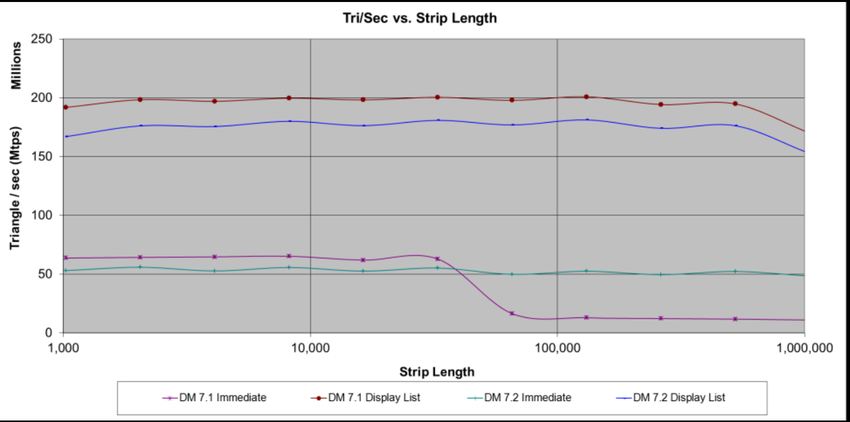
- Load model into test application
- Rotate model until stable state is reach
- Capture statistics for rotating the model 360 degree in 1 degree increments



- 16 Million Triangles
- 12,699 Occurrences

Optimization: Vertex Data Layout

- How are your vertices stored relative to how they are referenced?



Quadro 4500

  - Collocation: Sorts along random axis in order to eliminate duplicated vertices
  - Simple Fix: Sort in order of first reference
  - Advanced Fix: Vertex Cache Optimization (e.g. Tipsify, …)

Optimization: Vertex Buffer Objects

- Upload vertex data to buffer on the GPU and render straight from the buffer
  - Data on GPU does not have to match Data on CPU
  - Similar performance as GL Display Lists

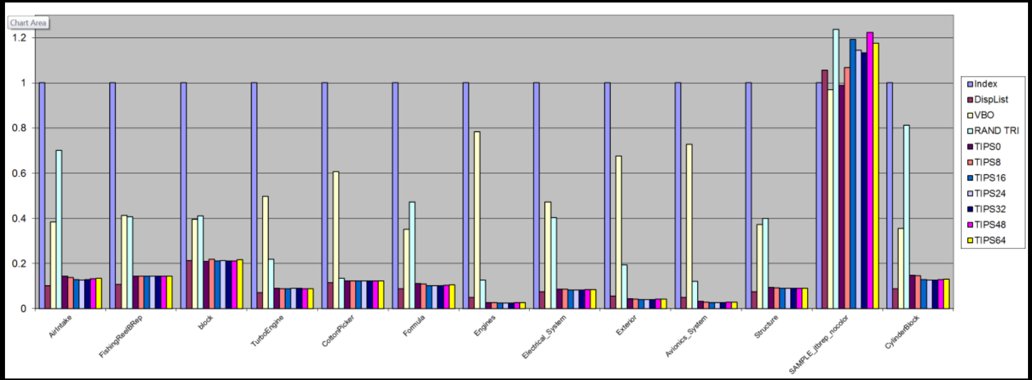
Poor Performance on certain GPUs

- glMultiDrawArrays

Optimum Performance

- glDrawRangeElements - Triangles
- glDrawRangeElements - PrimRestart

Render Time FireGL 7350 (*Relative to Index*)



Performance
15x | 2.6x

K2100M	
IDX	65 fps
VBO	13 fps
VCO	25 fps

Optimization: Unified Vertex Buffer Objects

- Create VBOs of a fixed size and populate sections with data from multiple render items
  - Significantly reduce the number of vertex bind calls
  - Increase cache coherency of data on the GPU, especially during render

Performance		
VBO	122 fps	27%
UVBO	155 fps	

Optimization: State Sorting

- Significant amount of GL calls can be attributed to applying the state updates
  - Sorting the state and only applying if it changes allows for the number of state update to be reduced

Performance		
Unsorted	120.40 fps	23%
Sorted	161.43 fps	

Render

```
apply(Engine)
apply(Frame)

while( item )  {

  if ( bNewL ) apply( Light )
  if ( bNewS ) apply( Shape )
  if ( bNewX )  apply( Xform )

  bind(geom)
  render( Geom )

}
```

Optimization: Uniform Buffer Objects

- Still a significant amount of state to be set
- Shaders complicate matters as they require state passed in through uniforms
- Uniform buffer objects allows for large blocks of state to be uploaded to the GPU and then set using a single bind call

Top 10 DM 7.3	Count
glBindBuffer	63560
glClientActiveTexture	12721
glVertexPointer	12712
glNormalPointer	12712
glDrawRangeElements	12712
glPushMatrix	12461
glPopMatrix	12461
glMultMatrixd	12451
glDisable	421
glMaterialfv	79

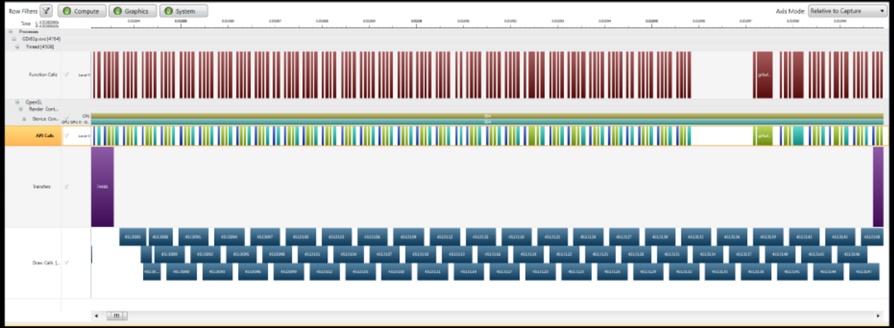
Top 10 DM 8.0	Count
glBindBuffer	30681
glVertexAttribPointer	20318
glVertexAttrib4fv	12715
glDrawRangeElements	12713
glBufferSubData	200
glPolygonMode	33
glBindBufferBase	31
glDisableClientState	11
glMatrixMode	9
glClientActiveTexture	9

Performance		
Uniforms	16.49 fps	11.5x
UBO	189.47 fps	

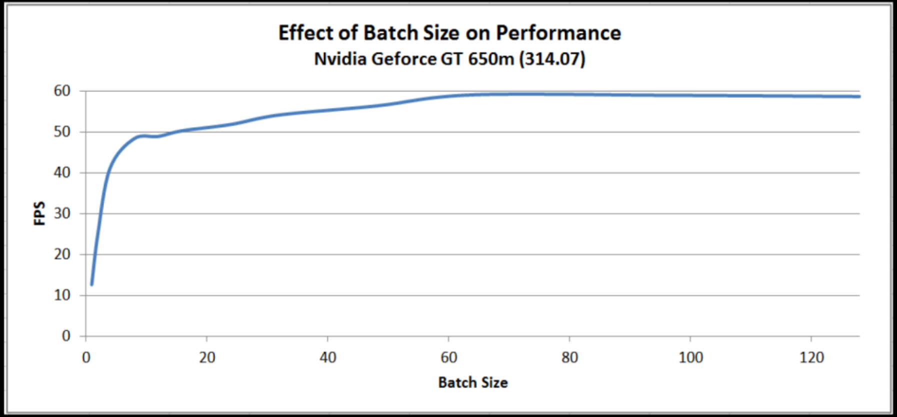
Optimization: Xform Batching

- GPU stalls due to data transfer can significantly impeded render performance

GPU Transfers as a result of xform updates



Increased concurrency as the result of batching



Optimization: MultiDrawElementsIndirect

- Allows for multiple draw calls to be combined into a single call
  - Offloads traditionally CPU work to the GPU
  - Biggest benefit will be seen by application that are CPU bound and render lots of small shapes

```
void MultiDrawElementsIndirect(enum mode,
                               enum type,
                               const void *indirect,
                               sizei primcount,
                               sizei stride);
```

```
struct DrawElementsIndirectBuffer
{
    DrawElementsIndirectBuffer()
      : count(0),
        instanceCount(0),
        firstIndex(0),
        baseVertex(0),
        baseInstance(0) {}

    JtUInt32 count;         // Number of elements to be drawn
    JtUInt32 instanceCount; // Number of instance to be drawn
    JtUInt32 firstIndex;    // Offset into index array
    JtInt32  baseVertex;    // Offset to to vertex records
    JtUInt32 baseInstance;  // Under GL 4.2 specifies the base instance for fetching
                            // instanced vertex attributes , other wise 0
};
```

Optimization: MultiDrawElementsIndirect

- Verify your application is a good fit
  - Use system timers to calculate system time
  - Use glQuery objects to measure GPU time

General	Redraw	0	2.760
	PreFrameCB	1	0.000
	PreFrame		0.003
	Strategy		1.844
	Render		0.910
	PostFrame		0.000
	PostFrameCB		0.000
	RenderMMVStategy	2	1.822
	ActivateRC		0.037
	ShadowMap		0.000
	MirrorRender		0.000
	MirrorPlanes		0.000
	PreRender		0.001
	PrepareLMVRendList		0.001
	RenderLMVRendList		0.007
	RenderMMVRendList		0.711
	PostRender		0.001
	PostFramePreSwapCB		0.000
	Swap		0.106
	PauseStrategy		0.001
	TransferGPU		0.000
	RenderGPU		0.538
	MainFrames		360

MMV	OccResult	0.533
	OccPropagate	0.021
	OccPopulate	0.089
	OccDepth	0.063
	OccTrav	0.132
	OccZPrime	0.539
	OccQuery	0.539
	OccRenderList	0.142
	OccDepthGPU	0.359
	OccZPrimeGPU	0.006
	OccQueryGPU	0.551
	OccResultGPU	
	OccConvertGPU	0.000
	OccRenderItems	120502
	OccDepRenderItems	119416
	OccZPrRenderItems	162027
	OccQueRenderItems	215197
	StategyFrames	180

Is your application CPU bound?

D	FPS	135.644
	CPU	2.654
	GPU	1.454
MT	CPU	0.832
	GPU	0.538
ST	CPU	1.822
	GPU	0.916

Are there a significant number of draw calls?

MT	BufferBinds	5800
	BufferSets	7614
	VertAttribBinds	7292
	IdxAttribBinds	360
	DrawCalls	432498
ST	BufferBinds	540
	BufferSets	3807
	VertAttribBinds	3260
	IdxAttribBinds	0
	DrawCalls	651776

Optimization: MultiDrawElementIndirect

- Define MDEI Buffers per State

```
// Prototype for rendering multiple items in a single draw call
class MultiRenderItem
{
  public:
    MultiRenderItem()  {}
    ~MultiRenderItem() {}

    SharedPtr<MultiDrawElementsIndirectBuffer> _pMDEIState;   ///< Multi geometry state
    SharedPtr<GeomState>                       _pGeomState;   ///< Geometry state
    SharedPtr<ShapeState>                      _pShpState;    ///< Shape state
    SharedPtr<LightState>                      _pLightState;  ///< Light state
};
```

```
// Prototype
class MultiDrawElementsIndirectBuffer : public JtRefCounted
{
  public:
    MultiDrawElementsIndirectBuffer()  {}
    ~MultiDrawElementsIndirectBuffer() {}

    BufferHdl                        _hMDEIBuf;     ///< DM Handle to Indirect Buffer
    BufferHdl                        _hMatBuf;      ///< DM Handle to Matrix Buffer
    TexObjHdl                        _hMatTex;      ///< DM Handle to Matrix Texture
    BufferHdl                        _hIdxBuf;      ///< DM Handle to Index Buffer

    JtVec<DrawElementsIndirectBuffer> _vDEIBuf;     ///< Vector of Indirect Buffer Elements
    SharedPtrVec<XformState>         _vpXformState; ///< XForm state
};
```

- Pass xforms in through texture buffer
- Use the glBaseInstanceID to specify Matrix
- Use an additional vertex attribute with glVertexDivisor for better performance

- MDEI and Index Buffer created once and then bound per each state transition
- Xforms buffer initialized with other buffers, however the matrices are recalculated before binding
  - Model*View
  - Model*View*Projection

Optimization: MultiDrawElementIndirect

- Define MDEI Buffers per State
  - Results in worse performance

		1	2
D	FPS	135.644	116.392
	CPU	2.654	3.093
	GPU	1.454	2.366
MT	CPU	0.832	0.446
	GPU	0.538	0.613
ST	CPU	1.822	2.647
	GPU	0.916	1.753

		1	2
MT	BufferBinds	5800	6902
	BufferSets	7614	10637
	VertAttribBinds	7292	7782
	IdxAttribBinds	360	360
	DrawCalls	432498	9557
ST	BufferBinds	540	900
	BufferSets	3807	13008
	VertAttribBinds	3260	3755
	IdxAttribBinds	0	180
	DrawCalls	651776	441580

MDEI generation is expensive on both CPU and GPU

Draw calls are significantly reduced

MMV	OccResult	0.533	0.127
	OccPropagate	0.021	0.036
	OccPopulate	0.089	1.062
	OccDepth	0.063	0.140
	OccTrav	0.132	0.324
	OccZPrime	0.539	0.672
	OccQuery	0.539	0.672
	OccRenderList	0.142	0.286
	OccDepthGPU	0.359	0.308
	OccZPrimeGPU	0.006	0.004
	OccQueryGPU	0.551	0.619
	OccResultGPU		
	OccConvertGPU	0.000	0.822
	OccRenderItems	120502	216590
	OccDepRenderItems	119416	215311
	OccZPrRenderItems	162027	163061
	OccQueRenderItems	215197	274944
	StategyFrames	180	180

Performance		
Orig	135.64	17%
MDEI | State	116.32	

Optimization: MultiDrawElementsIndirect

- MDEI Buffer Per Render List

```
/// Prototype
class MultiRenderList : public JtRefCounted
{

    BufferHdl                           _hMDEIBuf;     ///< DM Handle to Indirect Buffer
    BufferHdl                           _hMatBuf;      ///< DM Handle to Matrix Buffer
    TexObjHdl                           _hMatTex;      ///< DM Handle to Matrix Texture
    BufferHdl                           _hIdxBuf;      ///< DM Handle to Index Buffer

    JtVec<MDEIGroup>                    _vMDEIGroup;

    JtVec<DrawElementsIndirectBuffer>   _vDEIBuf;      ///< Vector of Indirect Buffer Elements
    SharedPtrVec<XformState>            _vpXformState; ///< XForm state

    JtUInt32                            _nAllocItems;
};
```

```
struct MDEIGroup
{
    MDEIGroup()
      : _pGeomState(),
        _pShpState(),
        _pLightState(),
        _iOffset(0),
        _nElements(0) {}

    SharedPtr<GeomState>    _pGeomState;   ///< Geometry state
    SharedPtr<ShapeState>   _pShpState;    ///< Shape state
    SharedPtr<LightState>   _pLightState;  ///< Light state
    JtUInt32                _iOffset;      ///< Offset into DEI
    JtUInt32                _nElements;    ///< Numbr of elements in DEI
};
```

		1	2	3
D	FPS	135.644	116.392	167.442
	CPU	2.654	3.093	2.150
	GPU	1.454	2.366	1.253
MT	CPU	0.832	0.446	0.382
	GPU	0.538	0.613	0.449
ST	CPU	1.822	2.647	1.768
	GPU	0.916	1.753	0.804

		1	2	3
MT	BufferBinds	5800	6902	6520
	BufferSets	7614	10637	1800
	VertAttribBinds	7292	7782	8682
	IdxAttribBinds	360	360	360
	DrawCalls	432498	9557	9850
ST	BufferBinds	540	900	540
	BufferSets	3807	13008	1800
	VertAttribBinds	3260	3755	4074
	IdxAttribBinds	0	180	0
	DrawCalls	651776	441580	440554

		1	2	3
MMV	OccResult	0.533	0.127	0.117
	OccPropagate	0.021	0.036	0.039
	OccPopulate	0.089	1.062	0.380
	OccDepth	0.063	0.140	0.103
	OccTrav	0.132	0.324	0.294
	OccZPrime	0.539	0.672	0.562
	OccQuery	0.539	0.672	0.562
	OccRenderList	0.142	0.286	0.233
	OccDepthGPU	0.359	0.308	0.218
	OccZPrimeGPU	0.006	0.004	0.004
	OccQueryGPU	0.551	0.619	0.582
	OccResultGPU			
	OccConvertGPU	0.000	0.822	0.000
	OccRenderItems	120502	216590	216228
	OccDepRenderItems	119416	215311	214943
	OccZPrRenderItems	162027	163061	162523
	OccQueRenderItems	215197	274944	274135
	StategyFrames	180	180	180

Significantly improves time to render on CPU

Performance	
Default	135.64
MDEI |State	116.39
MDEI | RL	167.44

23%

Optimization: Summary

Discussed

- Vertex Data Layout
- VBOs | Unified VBOs
- UBOs
- Batching of Data Updates
- MDEI

Future

- Bindless
- Culling
- CommandLists

Questions:

Jeremy Bennett [Senior Software Engineer, Siemens PLM Software]
jeremy.bennett@siemens.com

Michael Carter [Senior Key Expert, Siemens PLM Software]
michael.b.carter@siemens.com

Please complete the Presenter Evaluation sent to you by email or through the GTC Mobile App. Your feedback is important!