DATA VISUALIZATION OF THE
GRAPHICS PIPELINE:
TRACKING STATE WITH THE STATEVIEWER
ARAM HOETZLEIN, DEVELOPER TECHNOLOGY, NVIDIA
“Data Visualizations assist humans with data analysis by representing information visually. These mechanisms rely on human perception to help understand data.”

Human Factors in Visualization Research, Melanie Tory & Torsten Moller
GRAPHICS PIPELINE

- Input Assembler
- Vertex Shader
- Geometry Shader
- Rasterizer
- Pixel Shader
- Output Merger
- Stream Output
- Video Memory (Buffer, Texture, Constant Buffer)
Graphics State is complex.
CAD/Workstation Applications solve *complex*, real world problems
CPU Bound: Traversal of CPU scene graph, or drawing setup, outweighs GPU rendering.

Many CAD/Professional Workstation applications are CPU Bound.

These are ideal candidates for next-gen APIs.
PROFILING PRACTICE
PROFILING PRACTICE

APP → Modify → Profile → Driver → GTX970
PROFILING PRACTICE

CPU Timing ↔ State Profiling (Sequence) ↔ Driver Timing ↔ GPU Timing

APP → API → Driver

replay → capture → StateViewer → visualize → Driver

see the sequence
**GOALS**

API Tracing
Identify named buffers at the time of API calls.

Value Tracing
Identify which state arbitrary buffers belong to.
Identify *values* inside named buffers.
Identify *values* transferred by memcpy/map

Value-Delta Tracing
Identifies changes in values *in the same buffer*.
Identifies when switching buffers with *same value*.

*We want tools that identify all of the above.*
### Example

<table>
<thead>
<tr>
<th>Simple State Tracking</th>
<th>Value Tracking</th>
<th>Value-Delta Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B A</td>
<td>A(0) B(0) A(0)</td>
<td>Created Created</td>
</tr>
<tr>
<td>B B B</td>
<td>B(3) B(5) B(9)</td>
<td>Same State (0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Changed Buf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Same Buf (B)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Same Buf (B)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Changed Buf</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>A</th>
<th>B</th>
<th>VBO State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **A(0)**: Same State (0)
- **B(3)**: Changed Buf
- **B(5)**: Same Buf (B)
- **B(9)**: Same Buf (B)
- **B(2)**: Changed Buf
Colored rectangles map state values.

Colored flags map state value changes.

Create/write – app is allocating a new buffer, or rewriting it.

Switch – app is switching to another buffer.

Reuse – app is reusing buffer from last draw, no switch.
PASS #1

Replay all API calls to determine state bins.

Example:
- DXCreateBuffer
- IASetVertexBuffer

How will it be used? Unknown until later.
Now, we know it is a VBO.

PASS #2

Replay all API calls again, and record both input and output values.
Compress all values using a 128-bit hash.
Assign colors and track deltas based on the hash.

Every API call specifies a unique state bin, named object, and value.
ALGORITHM

Map buffer memcpy

... some data ...

Map buffer memcpy

... other data ...

Assign hash

128-bit hash

Assign color

Compare to previous state

Different, give red flag (due to Map)

time
<table>
<thead>
<tr>
<th></th>
<th>WHAT STATES TO TRACK?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Shader</td>
</tr>
<tr>
<td>1</td>
<td>Render Target</td>
</tr>
<tr>
<td>2</td>
<td>Viewport</td>
</tr>
<tr>
<td>3</td>
<td>Rasterizer State</td>
</tr>
<tr>
<td>4</td>
<td>Depth State</td>
</tr>
<tr>
<td>5</td>
<td>Blend State</td>
</tr>
<tr>
<td>6</td>
<td>Sampler State</td>
</tr>
<tr>
<td>7</td>
<td>Input</td>
</tr>
<tr>
<td>8</td>
<td>Texture</td>
</tr>
<tr>
<td>9</td>
<td>Vertex Buffer (IA Slot 0)</td>
</tr>
<tr>
<td>10</td>
<td>Vertex Buffer (IA Slot 1)</td>
</tr>
<tr>
<td>11</td>
<td>Vertex Buffer (IA Slot 2)</td>
</tr>
<tr>
<td>12</td>
<td>Vertex Buffer (IA Slot 3)</td>
</tr>
<tr>
<td>13</td>
<td>Vertex Buffer (IA Slot 4)</td>
</tr>
<tr>
<td>14</td>
<td>VS Const Buffer 0</td>
</tr>
<tr>
<td>15</td>
<td>VS Const Buffer 1</td>
</tr>
<tr>
<td>16</td>
<td>VS Const Buffer 2</td>
</tr>
<tr>
<td>17</td>
<td>PS Const Buffer 0</td>
</tr>
<tr>
<td>18</td>
<td>PS Const Buffer 1</td>
</tr>
<tr>
<td>19</td>
<td>PS Const Buffer 2</td>
</tr>
<tr>
<td>20</td>
<td>Index Buffer</td>
</tr>
</tbody>
</table>
STATEVIEWER

Contributed to *apitrace*, open source.

A free tool for deep state tracking /w value deltas.

Simple trace and view workflow.

* Now available on github! *

```
Application        App Replay
run              replay    track state
apitrace .trace  d3dretrace .raw     stateviewer
glretrace .raw   visualize data
```
Example:

Draw instanced spheres with some GUI controls.
SECTIONVIEWER: SIMPLE EXAMPLE

Observe:

Frames separate by white bars.

Each column is one draw call.

First draw uses different shader, VBO, and VS constant. This draws instanced spheres.

Eight other calls use same shader, and VBO. These draw the GUI bars.

PS Const1 flip-flops between 2 states. This is the grey and green bars in the GUI of the app.
The Flip-Flop

Bars oscillate between values. Indicates potentially unnecessary switch between two states.

Example:
Draw faces, then edges, then faces, then edges.
Flatliner

Set of draw calls which use the same shader, VBO and number of primitives. Draw may be unnecessarily repeated.

Example: Drawing multiple copies of an object in the different locations.
The Repeater

A set of states that is similar to an earlier group.
Strongly suggests candidates for grouping.

Example:
Draw legs, arms, back and seat of a chair.
Then draw whole chair again!
## All About the (Data) Patterns

<table>
<thead>
<tr>
<th>Worst Case</th>
<th>Bad Case</th>
<th>Better Case</th>
<th>Great Case</th>
</tr>
</thead>
</table>
| 1. Recreating buffers  
2. Using single buffer w/ different data | 1. Good - Write once  
2. Lots of switching | 1. Good - Write once  
2. State sorting  
3. Draw without switch. | 1. Good - Write once  
2. State sort!  
3. Make global buffers for other state  
4. Draw ONCE with MultiDraw |

### StateViewer Output

- **Create**
- **Map / Unmap**
- **IASetVB / Draw**
- **StateSort**

### Results

- **Worst Case:** All red flags
- **Bad Case:** Mostly orange
- **Better Case:** Mostly green
- **Great Case:** All green (except first frames)

### Flags

- **Red Flags:** Too many
- **Orange Flags:** Too many
- **Green Flags:** Green!
## Real-World Applications

![GPU Technology Conference](image)

### Frame Rates
- Frame draws per second:
- 9800: 3201000
- 1990: 3200

### Frame Statistics:
- Shader
- Rasterizer State
- Viewport
- Depth State
- Blend State
- Sampler State
- Input Assembler
- Texture

### Vertices
- Vertex 0
- Vertex 1
- Vertex 2
- Vertex 3

### Outputs
- VS Output 0
- VS Output 1
- VS Output 2
- WS Output 0
- WS Output 1
- WS Output 2
- WS Output 3
- WS Output 4

### Index Buffer
- Index count: 345714
- Number of prims: 345714

### Transfer (bytes)
- 1

---

*Image of a data visualization from the GPU Technology Conference showing real-world applications.*
REAL-WORLD APPLICATIONS

Good use of const buffers.
(Multiple buffers, mostly green flags)

Shader is switched frequently.
Candidate for shader-based sorting.

Drawing many small object.
Candidate for geometry binning.
REAL-WORLD APPLICATIONS
REAL-WORLD APPLICATIONS

Shader re-assigned on each draw.

Vertex buffer rewritten on every other draw.

Constant buffers rewritten, often with repeatedly used value. (Flip-flop)

Repetitive pattern suggests duplicated geometry. (Repeater)
GPU Timing:
Gives valuable information about what the graphics API and GPU are doing.
*Good for GPU-bound apps. Use NSight.*

e.g. Does GPU spend more time in vertex or pixel shader?

CPU Function Profiling:
Gives valuable information about which are the slowest functions.
*Good for Algorithm-bound apps.*

e.g. Which specific part of a CPU algorithm is slowest?

StateViewer:
Gives systematic information about design patterns in the application.
*Good for Data-bound apps.*
*Tells us *why* the app is slow, without access to code!*

e.g. How could the data be better organized for submission to graphics pipeline?
StateViewer has identified unknown issues in several large CAD/Workstation applications.

Provides an overall picture of the application’s *systematic* behavior.

Gives feedback with direct indicators on areas of improvement.
Thank You!

DATA VISUALIZATION OF THE GRAPHICS PIPELINE

JOIN THE CONVERSATION
#GTC15