An Incremental Rendering VM

Georg Haaser, Harald Steinlechner, Stefan Maierhofer, and Robert F. Tobler

VRVis Research Center
Vienna, Austria
Motivation

Rendering huge scenes with dynamism

- Editor applications / strategy games
- Everything is changeable (add / remove / modify)
- Changes are relatively “rare”

Problems

- Many driver calls
- Driver calls impose cost (e.g. redundancy, validation)
- Hardware not utilized
Typical approaches

Static
- Re-organize the scene (e.g. grouping)
- Merge geometries
- Render caches (e.g. Durbin et. al. 1995)
- Difficult to update

Dynamic
- Filter redundant API calls
- State sorting (e.g. sort by shaders)
- Runtime overhead
Ultimate Goal

Dynamic flexibility
without compromising performance
Minimal frontend

Expected input
- Set of Render Objects
- Order independent

Render Objects
- Draw call arguments
- Stateless

<table>
<thead>
<tr>
<th>Render Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
</tr>
<tr>
<td>- Shaders</td>
</tr>
<tr>
<td>- Modes (BlendMode,...)</td>
</tr>
<tr>
<td>- IndexBuffer</td>
</tr>
<tr>
<td>- Vertex Attributes</td>
</tr>
<tr>
<td>- Instance Attributes</td>
</tr>
<tr>
<td>- Uniforms</td>
</tr>
<tr>
<td>- DrawCall Arguments</td>
</tr>
</tbody>
</table>
How to render those objects?

track changes

sort by state

prune & execute commands

per frame

{\{s_1, t_2, b_2\}, \{s_1, t_1, b_1\}, \{s_2, t_3, b_3\}}
State Sorting

**Optimal order**
- Costs for all transitions (shaders, textures, etc.)
- Minimize total cost

**Incremental changes?**
- Cannot maintain optimality
- State-trie turns out to be effective
- Easy to update
An Incremental Rendering VM

shaders

textures

buffers

\{s_1, t_1, b_1\} \rightarrow \{s_2, t_3, b_3\}
An Incremental Rendering VM
An Incremental Rendering VM

Georg Haaser
An Incremental Rendering VM

- **Shaders**: \{s_1, t_2, b_2\}, \{s_2, t_3, b_3\}
- **Textures**: \{s_1, t_1, b_1\}
- **Buffers**: \{s_1, t_2, b_2\}, \{s_2, t_3, b_3\}

Georg Haaser
An Incremental Rendering VM

shaders

textures

buffers

\{s_1, t_1, b_1\}

\{s_2, t_3, b_3\}
An Incremental Rendering VM
Reduce runtime costs

track changes

per frame

sort by state

prune & execute commands

{\{s_1, t_2, b_2\}, \{s_1, t_1, b_1\}, \{s_2, t_3, b_3\}}
Reduce runtime costs

{\{s_1, t_2, b_2\}, \{s_1, t_1, b_1\}, \{s_2, t_3, b_3\}\}

sort by state

prune & execute commands

Success: less stuff at runtime!

track changes per frame
Idea: Graphics commands are data too

Instructions describe graphics commands

Decouple creation and execution

Examples

- SetShaderInstruction(5)
- DrawInstruction(Lines, 0, 100)
- ...

Georg Haaser

An Incremental Rendering VM
Idea: Instructions are data too

prune & execute commands

1. generate instructions
2. optimize
3. execute
\[ \{s_1, t_1, b_1\} \]

- SetShader(s₁)
- SetTexture(t₁)
- SetBuffer(b₁)
- Draw()

\[ \{s_1, t_2, b_2\} \]

- SetShader(s₁)
- SetTexture(t₂)
- SetBuffer(b₂)
- Draw()
Reduce runtime costs

- generate abstract instructions
- redundancy removal
- execute

per frame
Reduce runtime costs

- Generate abstract instructions
- Redundancy removal
- Execute per frame

Success: less stuff at runtime!
But...

How to efficiently execute optimized instructions?
Interpreter techniques

Switch interpreter
- Managed / Unmanaged
- Employed by Wörister et al. 2013

Indirect threaded code
- Array of function pointers

Direct threaded [Bell 1973]
- Compile the whole list of fragments
- Baseline for performance (changes are expensive)
No overhead! Native compilation

SetShader(s₁)
SetTexture(t₁)
SetBuffer(b₁)
Draw()

SetTexture(t₂)
SetBuffer(b₂)
Draw()

Just-in-time compilation

Executable memory

0xEAFL

MOV EXC, 0xb44
MOV RAX, 0x7f
CALL RAX
JMP 0xFEED

0xFEED

MOV EXC, 0xe44
MOV RAX, 0x5a
CALL RAX
JMP 0xDEAD
Benchmark: fake driver

![Graph showing the benchmark results for different instruction counts and time per instruction. The graph compares Direct Threaded, Ours (sequential), C# switch, and C++ switch. The x-axis represents the instruction count, ranging from 0k to 250k, while the y-axis represents the time per instruction, ranging from 0 ns to 45 ns. The graph indicates that Ours (sequential) and Direct Threaded have the lowest time per instruction, followed by C# switch and C++ switch.]
Benchmark: real driver

(Average relative framerate)
Benchmark: other engines

Architecture (7022 objects)
Benchmark: other engines

Sponza24 (9408 objects)
Benchmark: other engines

HugeCity (6580 objects)
## Benchmark: other engines

<table>
<thead>
<tr>
<th>Platform</th>
<th>Scene</th>
<th>Frame Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>Sponza24</td>
<td>&gt; 200 ms</td>
</tr>
<tr>
<td>Linux</td>
<td>HugeCity</td>
<td>&gt; 200 ms</td>
</tr>
<tr>
<td>Linux</td>
<td>Architecture</td>
<td>&gt; 200 ms</td>
</tr>
<tr>
<td>Windows</td>
<td>Sponza24</td>
<td>&gt; 200 ms</td>
</tr>
<tr>
<td>Windows</td>
<td>HugeCity</td>
<td>&gt; 200 ms</td>
</tr>
<tr>
<td>Windows</td>
<td>Architecture</td>
<td>&gt; 200 ms</td>
</tr>
</tbody>
</table>

Frame Time

- **Ours**
- **Unity 4.6**
- **Open Scenegraph**
What we achieved

incremental changes

per frame

- State trie
- Instruction
- JIT Compiler
- Execution
Costs

- Compilation overhead still more than 2000 changes/s
- Platform dependent compiler

Outlook

- NV_command_lists, Mantle, DirectX 12, etc.
- SceneGraph frontend
- Driver integration anyone? 😊
- Order dependent rendering
Thank you for your attention!

Please visit us at

http://www.VRVis.at/