

Shader Driven Compilation of Rendering Assets

Paul Lalonde

Eric Schenk

Electronic Arts (Canada) Inc.



Motivation

- We want high level API: models more than vertices
- We would like a cross-platform system
- We also want low-level features that are *not* cross platform
- High performance requires out-of-API knowledge
- Less programmer involvement in shader selection



Target Hardware

- Sony PS2, Microsoft Xbox, Nintendo GC, Microsoft DX8 PC
- Common Characteristics:
 - CPU and GPU connected by narrow bus
 - GPU is programmable
 - Texture combining is a post-GPU step
(our system does not address pixel to vertex shader feedback)
- Avoid read/compute/write/submit-to-GPU
 - Avoid touching data twice



Display Lists and Scene Graphs

- OpenGL and DirectX display lists can be efficient, but must be constructed by a program
 - More efficient because they save many function calls for subsequent submissions.
- Iris Performer forces direct-mode geometry submission in exchange for hardware state change reduction.



Shader Compilers

- Might produce cross-platform shaders
- Address specification and construction of efficient shaders
- Do not address the layout of rendering data
 - No automatic system to connect model data to shaders
- Do not address offline data transforms



Let's Compile Art

- Encode out-of-API hardware performance issues in the Compiler
 - Cache sizes
 - Cost of state changes
 - Memory bandwidth
- Be aware of user constructs, such as models
- Perform offline transforms
- Generate code to hook model data to shaders



Properties of Game Art

- Fixed topology
- Come in atomic bundles: Models
 - Multiple shaders per model
- Game AI controls many attributes
 - Skeleton pose
 - Morph weighting
 - Shader variables



Mapping Art to Shaders

- Three kinds of shader inputs
 - Source art that is opaque
 - Coordinates, normals, colours
 - Source art modified by the user at runtime
 - Some coordinates, colours
 - Often default parameters (lighting, ...)
 - State information (shading modes, textures)
 - Runtime values used for control
 - Transformation matrices, matrix palettes
- How do we link art to shaders?



The Render Method

- Extend a shader to include:
 - Input specifications
 - Offline transformations
- Make it re-usable with different art
- Use the Render Method both in runtime and in the Compiler
- Unfortunately platform specific



Render Method – Inputs

- Lists data elements required from the source art
- Inputs are platform independent

```
Rendermethod gouraud {  
    inputs {  
        Coordinate4 Coordinates;  
        ColourARGB Colour;  
        Char GeometryName;  
        int nVerts;  
    }  
    ...  
}
```



Render Method – Variables

- Transforms input data to shader data

```
Rendermethod gouraud {  
    ...  
    variables {  
        extern volatile Matrix Xf = View::Xf;  
        Coord3Colour coords[nVerts]  
            = Pack3(Coordinates, Colours);  
        export modifiable State  
            GeometryName::state;  
        noupload RenderBuffer output[nVerts];  
    }  
    ...  
}
```

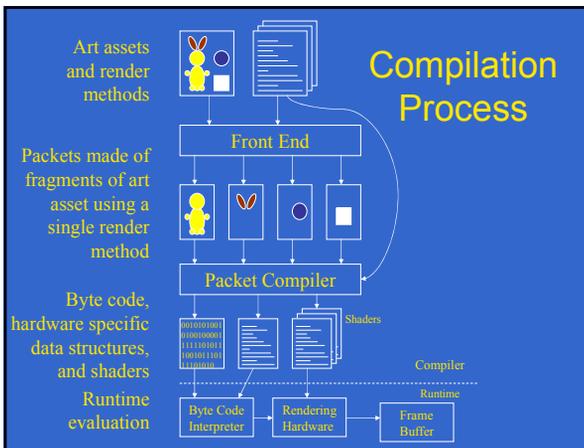


Render Method – Computations

- Computations are macro expanded to make shaders

```
Rendermethod gouraud {  
    ...  
    Computations {  
        XCProject(nVerts, coords, Xf, output);  
        XGKick(GeometryName::state);  
        XGKick(output);  
    }  
}
```





Compiler Front End

- We compile from an art package agnostic intermediate form
- FE breaks models into per-render method classes
- Vertices may be reordered for cache optimization (tri-strips, meshes, etc)
- Classes are broken to fit hardware and rendermethod constraints (eg, skinning palettes), into streams of packets



Packet Compiler

- For each packet stream
 - Reorder to minimize state changes
 - Model draw is atomic
- Per packet
 - Execute data transforms
 - Generate platform specific data structures
 - Generate platform specific byte code



Code Generation

- Generate a byte-code program to render each packet
 - Byte code used to exploit ICache coherence
 - Program attaches data elements to shader
 - Instructions relate to accumulating hardware data structures and setting state

Global Optimizations

- Optimizations encode much platform specific out-of-API knowledge
 - Remove redundant state settings
 - Remove redundant instructions
 - Remove redundant data transfers
 - Merge DMA chains
- After Optimization, Emit.
 - We generate ELF files to facilitate linkage



Results

- Easy to port
 - 3 Man-months to a new platform
 - Games port in 1-2 man-months
- Render Methods have been used for many custom features



Results II

				
Platform	Gouraud	Lit	Skinned	Gouraud
PS2	17.0/22.6	10.9/14.7	8.5/11.5	25.2/31.8
Xbox	47.2/91.4	22.4/43.4	14.2/30.3	63.9/93.8
NGC	18.7/NA	10.3/NA	7.2/NA	NA/NA
PC	24.1/46.1	15.9/20.9	10.3/10.9	26.3/36.2

Millions of Polygons per sec/Millions of Vertices per sec
Figures date from November 2001

Future Directions

- Runtime/Just in time compilation
- Ease of use for dynamic topology
- Incremental compilation for rapid art iteration
- Cross-platform Render Methods

Video Not Available

- 2 Mins