A Hardware-Aware Debugger for the OpenGL Shading Language

Magnus Strengert, Thomas Klein, Thomas Ertl

Institute for Visualization and Interactive Systems,
University of Stuttgart

Motivation

"Turn around time for debugging and tuning shaders is too long."
(NVIDIA GDC'07, Performance Tools slides)

"GPU programmers have just a small handful of languages to choose from, and few if any full-featured debuggers and profilers."
(Owens et al., A Survey of General-Purpose Computation on Graphics Hardware, COMPUTER GRAPHICS forum, 2007)

Motivation

Limited debug interface to GPUs

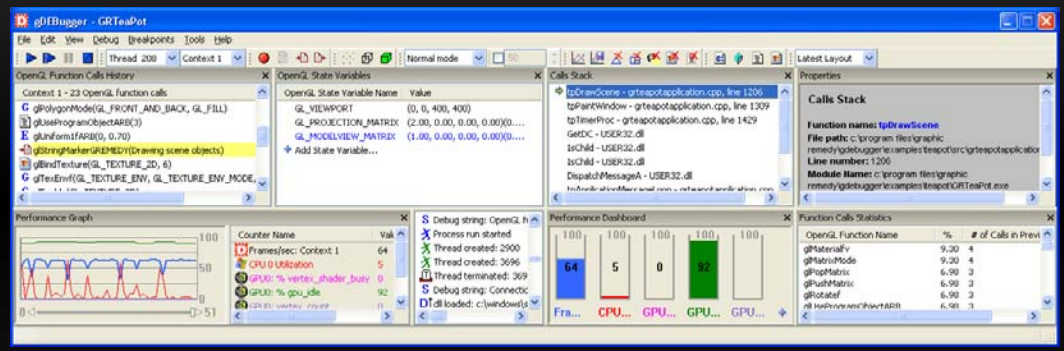
- Performance counters
- No register content, no single stepping

Shaders tend to become very long, complex

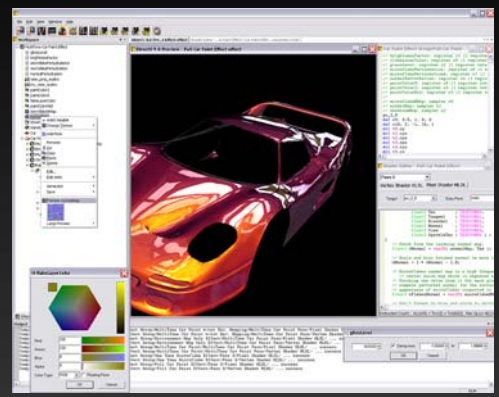
- Printf debugging increasingly difficult
- How to *printf* Vertex, Geometry shaders?

Related Work:

- **OpenGL State Debugging:**
  - spyGLass, BuGLe, GLIntercept
  - gDEBugger *(Graphic Remedy)*



- **Shader Development:**
  - Shader Designer *(TyphoonLabs)*
  - RenderMonkey *(AMD)*
  - FX Composer *(NVIDIA)*



Related Work: Shader Debugging

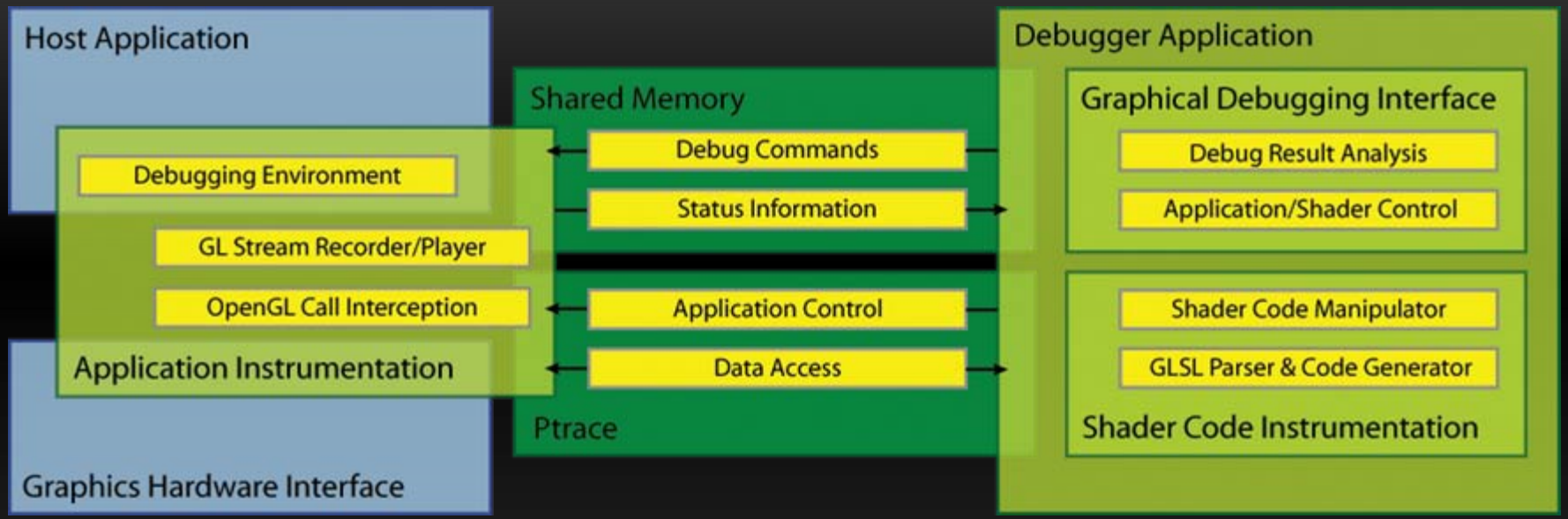
- **Shadesmith** *(Purcell et al., 2003)*
  - ARB fragment programs, interactive deepening
- **A Relational Debugging Engine for the Graphics Pipeline** *(Duca et al., Siggraph 2005)*
  - CG vertex and fragment programs
  - GQL: Graphical Query Language
  - Never publicly available
- **Software Rasterization:**
  - Microsoft PIX: HLSL Shader Debugger
  - Mesa 7.0: GLSL 1.2 Software Emulation

Goal

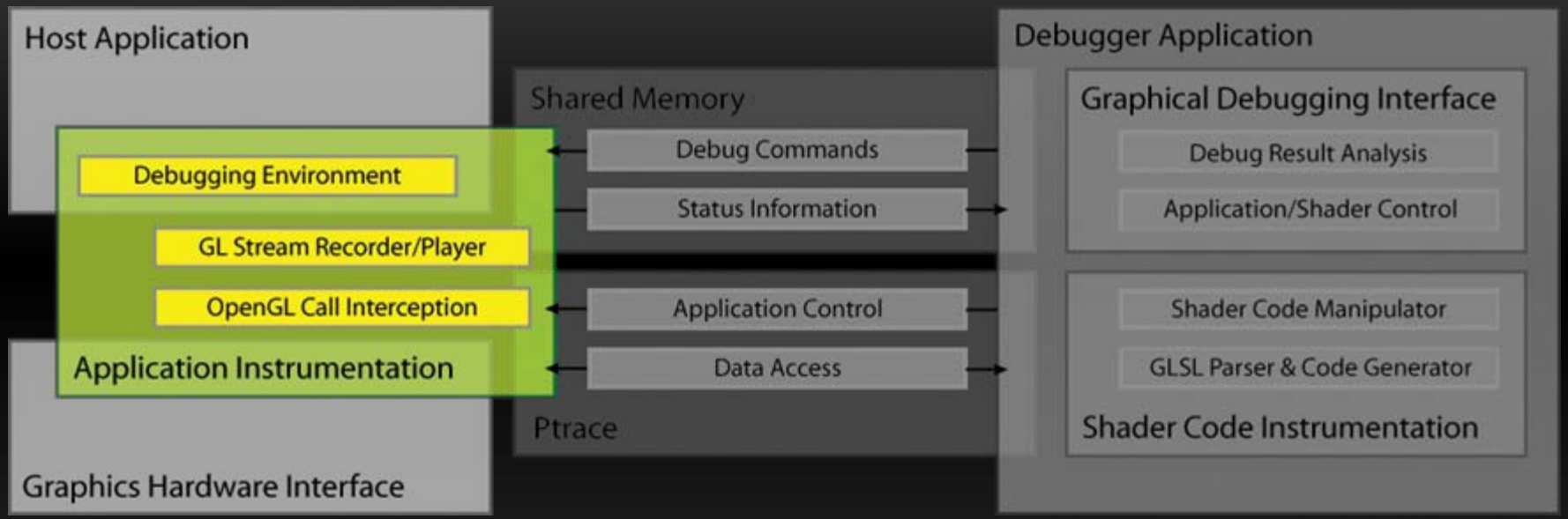
GPU-Debugging as easy as CPU-Debugging

- Application transparent
  - OpenGL call interception (Dll-Hooking/Pre-Loading)
- No software emulation, real hardware values
  - Shader Instrumentation
- Support for Vertex, Geometry, and Fragment shaders
  - Readback Vertex and Fragment data

System Overview



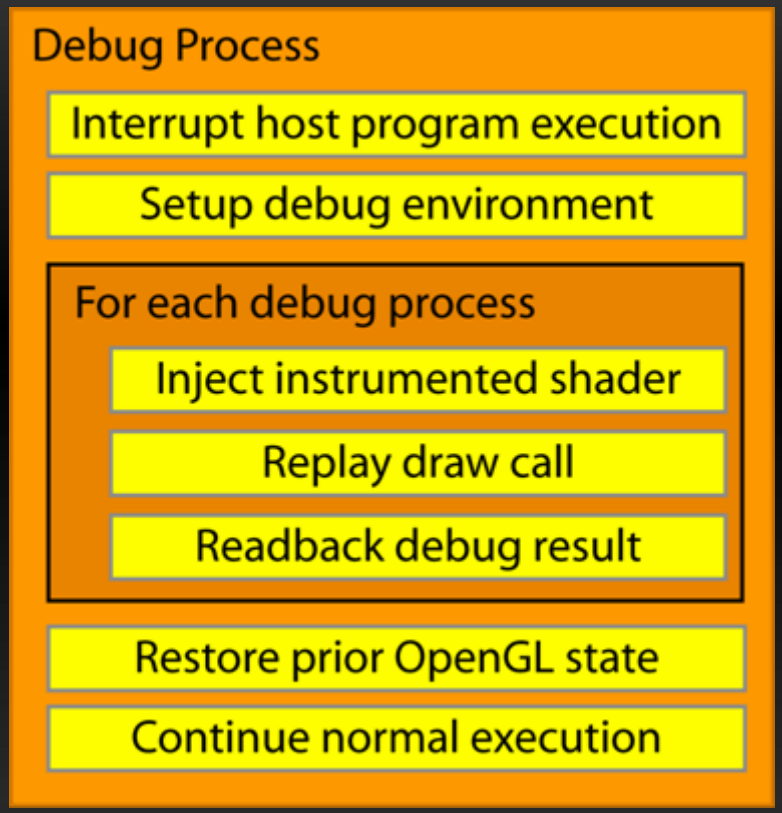
System Overview



Application Instrumentation

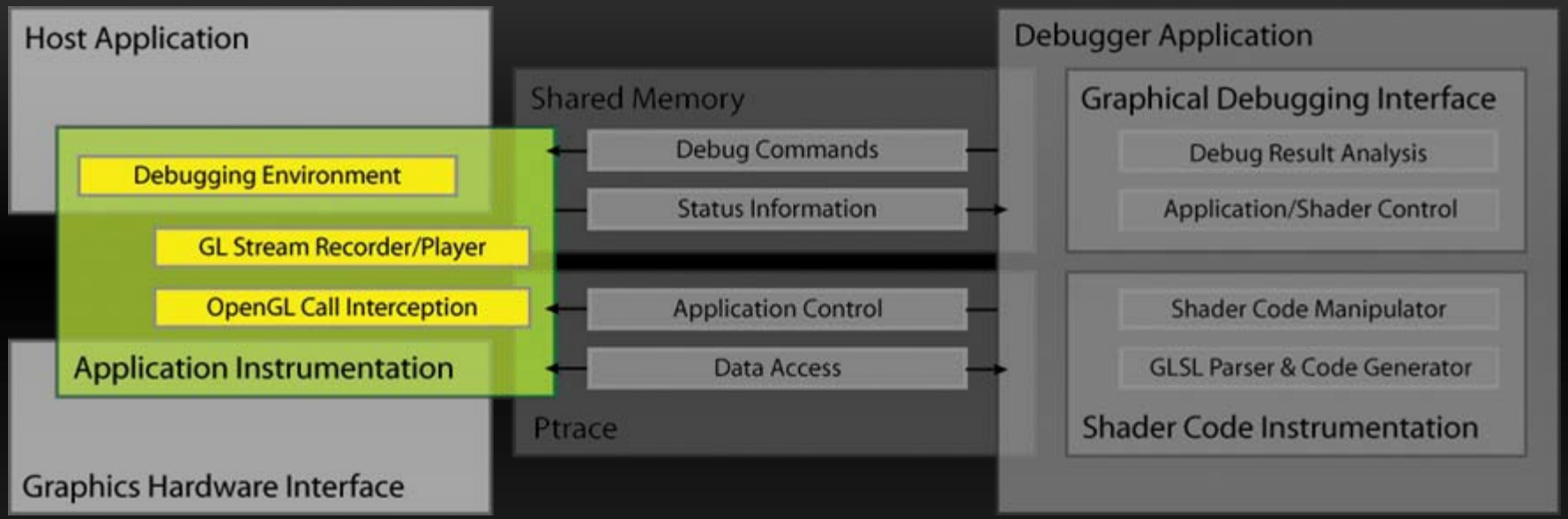
- Control execution of debugged application
  - Execute, Run, Interrupt
  - Single stepping through OpenGL calls
  - Edit OpenGL function call parameters
- Debug shader invocation of interest
  - Retrieve/Inject shader code
  - Provide contained environment for debugging

Application Instrumentation

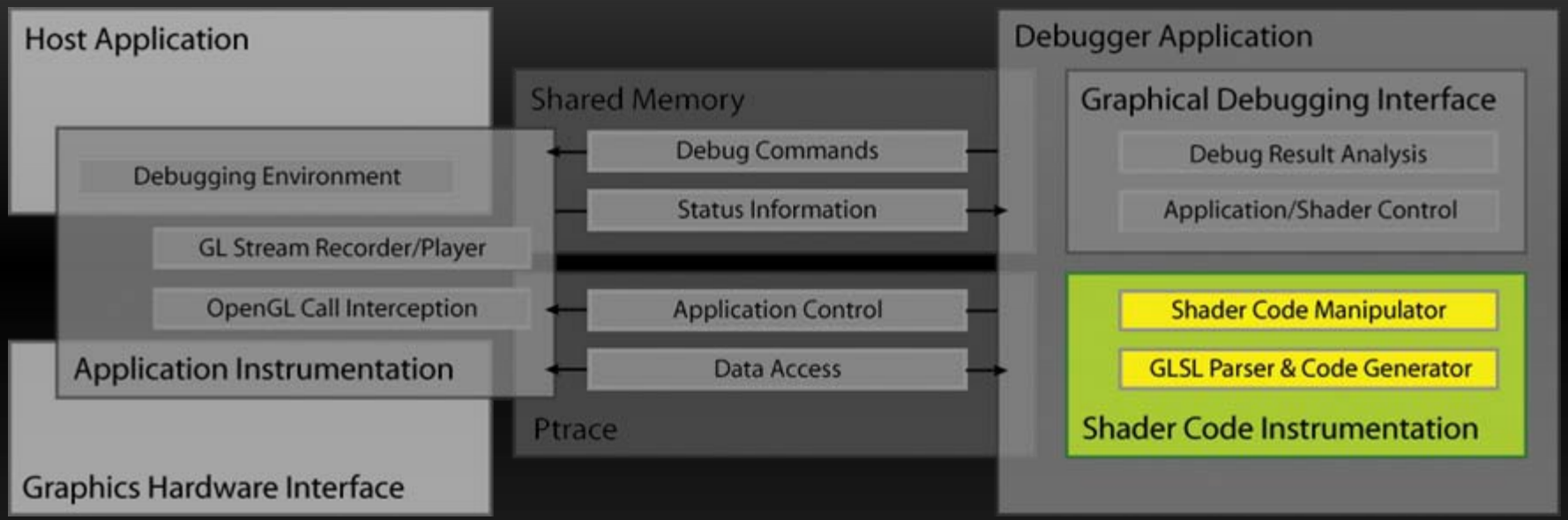


- Debug environment
  - Framebuffer object
  - Transform feedback
- Transparent for host
  - OpenGL States
  - Buffer content
  - Queries

System Overview



System Overview



Shader Instrumentation

Manipulate GLSL shader code

- Add debug code
  - Output variable content at per-statement level
- Changes should be minimal
- Program semantic must remain unchanged
  - Except debug output (additional varying or color.r)
  - Respect per-fragment tests (alpha, depth)

Debug Code Insertion

Use sequence (,) operator

- can be used in place for any single expression
- operation order from left to right
- return type and value defined by right-most operand

```
float dbgResult;
void main() {
   gl_FragColor = (dbgResult = gl_Color.x, gl_Color *  2.0f);
   gl_FragDepth = gl_FragColor.x;
   gl_FragColor.x = dbgResult;
}
```

Debug Code Insertion

Logical-and (&&) operator for conditional code

- Used for debugging in a loop body
- Check for name collisions when adding debug variables

```
int dbgIter0;
...
dbgIter0 = 0;
for ( i = 10; i > 0; i--, dbgIter0++) {
   (dbgIter0 == 5 && (dbgResult = f, true )) , f += f;
}
...
```

Debug Code Generation

Temporary debug registers

- For function parameters or conditionals

Duplicate functions and rename

- To debug function calls at single invocation

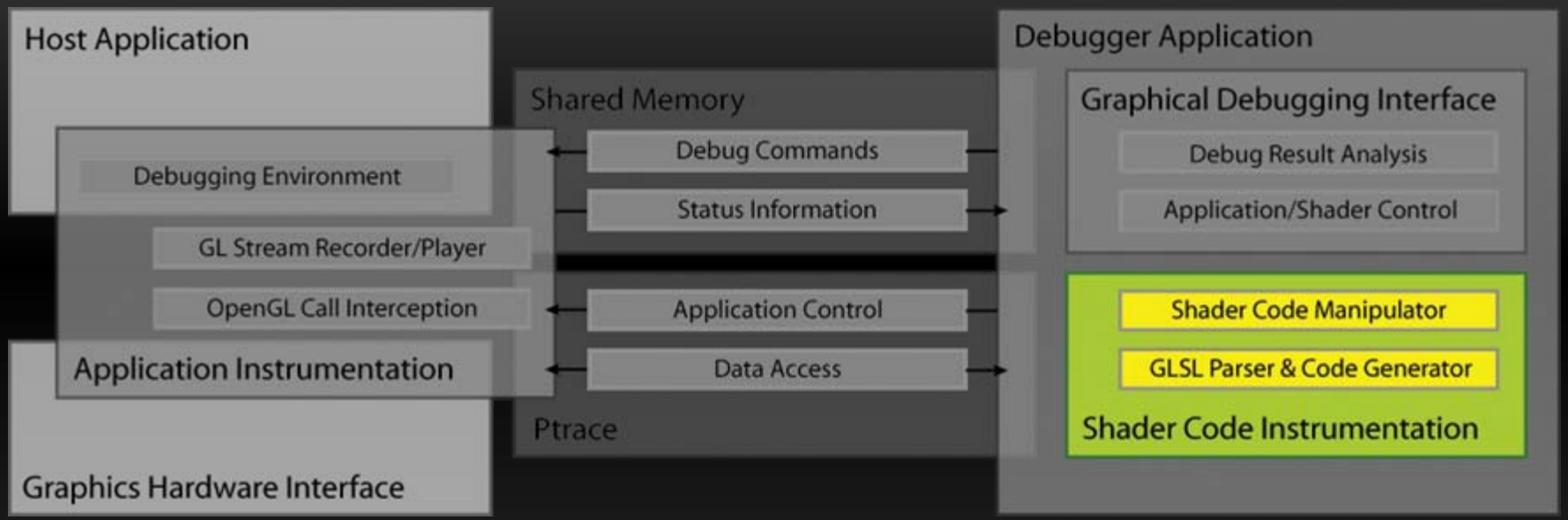
```
void F(inout int p1, int p3, out int p4);
…
int dbgParam;
F (i, (dbgParam = float(k + = j) ,dbgResult = k , dbgParam), k );
…
```

Realization

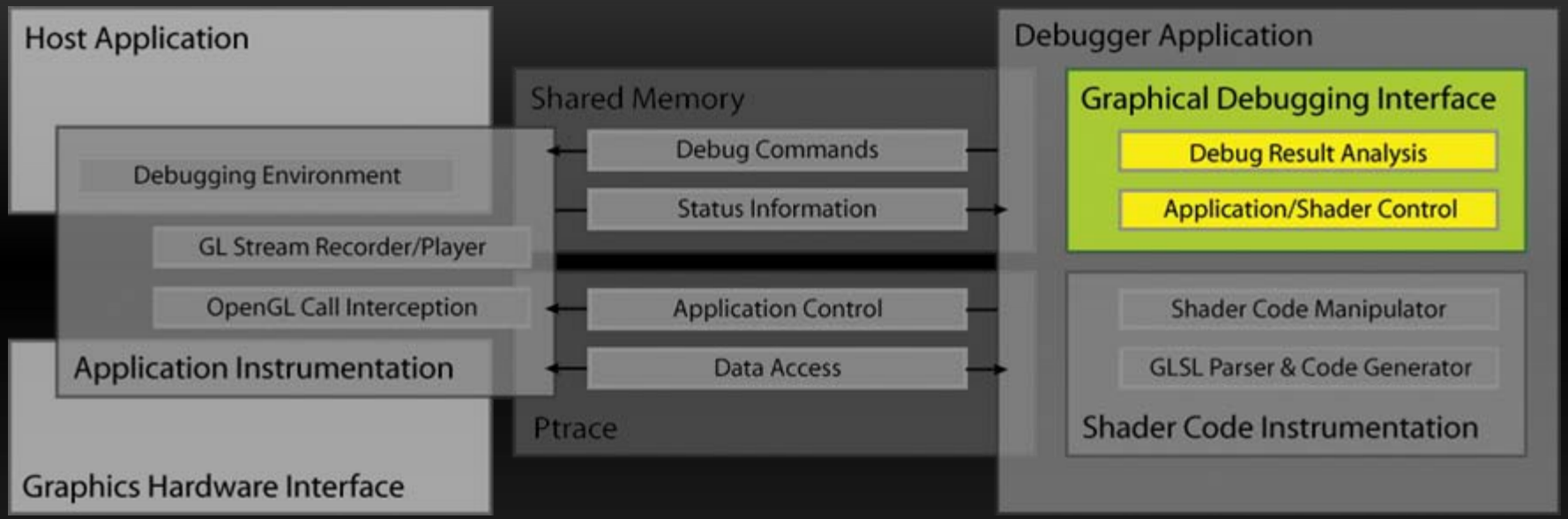
Built intermediate shader representation

- GLSL compiler build upon *3DLabs GLSL Compiler Frontend*
  - Added support for GLSL 1.20
  - Includes extension *EXT_gpu_shader4*
- Debug Code Generator Backend

System Overview

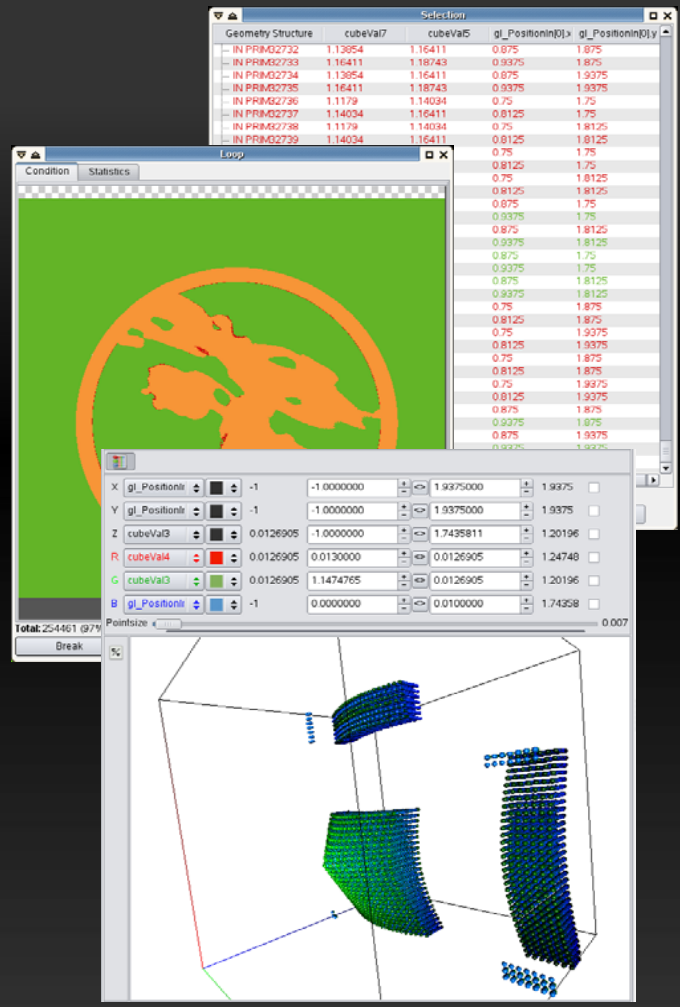


System Overview



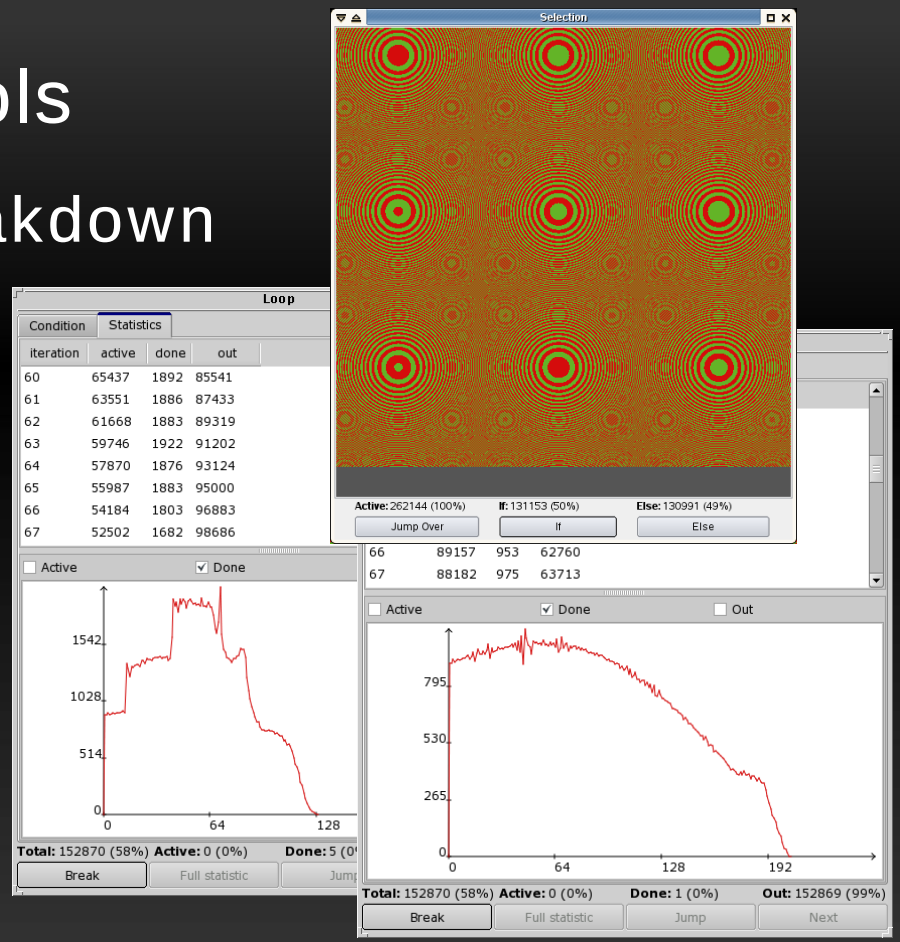
Application Interface

- Typical debugger concepts
  - Step In, Step Over
  - Watch Variables
- Parallel target hardware
  - Millions of threads in parallel
    - Flow Control Decisions
    - Data Inspection

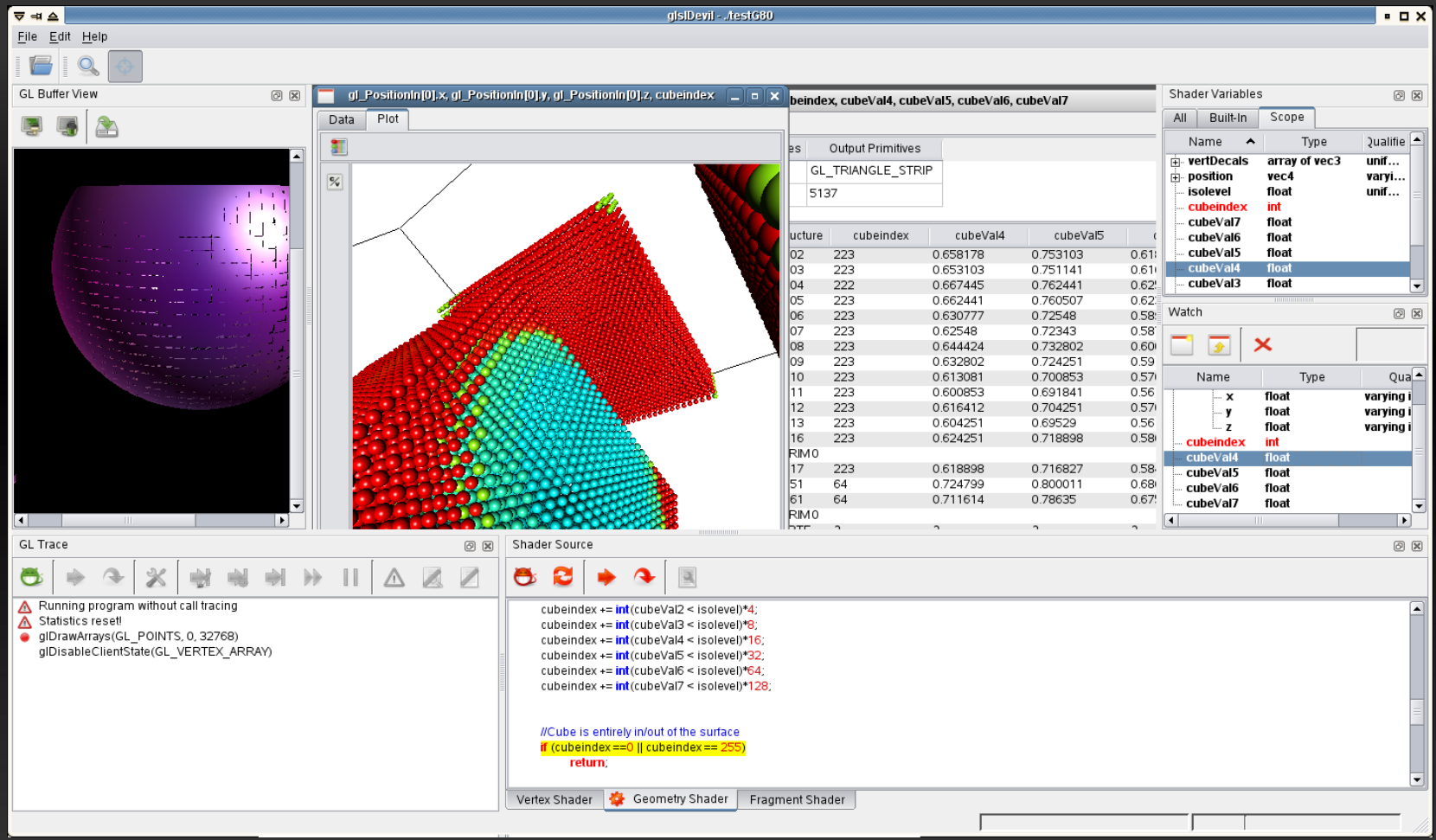


More than finding bugs?

- Advanced Analysis Tools
  - Conditional branch breakdown
    - Level of divergence
  - Loop iteration analysis
    - Active/Finished fragments per iteration
    - Loop graphs



Demo


glslDevil - ./testG80
File Edit Help
GL Buffer View
gl_PositionIn[0].x, gl_PositionIn[0].y, gl_PositionIn[0].z, cubeindex
Data Plot
Output Primitives
GL_TRIANGLE_STRIP
5137
cubeindex, cubeVal4, cubeVal5, cubeVal6, cubeVal7
Shader Variables
All Built-In Scope
Name Type Qualifie
vertDecals array of vec3 unif...
position vec4 varyi...
isolevel float unif...
cubeindex int
cubeVal7 float
cubeVal6 float
cubeVal5 float
cubeVal4 float
cubeVal3 float
Watch
Name Type Qua
x float varying i
y float varying i
z float varying i
cubeindex int
cubeVal4 float
cubeVal5 float
cubeVal6 float
cubeVal7 float
GL Trace
Running program without call tracing
Statistics reset!
glDrawArrays(GL_POINTS, 0, 32768)
glDisableClientState(GL_VERTEX_ARRAY)
Shader Source
cubeindex += int(cubeVal2 < isolevel)*4;
cubeindex += int(cubeVal3 < isolevel)*8;
cubeindex += int(cubeVal4 < isolevel)*16;
cubeindex += int(cubeVal5 < isolevel)*32;
cubeindex += int(cubeVal6 < isolevel)*64;
cubeindex += int(cubeVal7 < isolevel)*128;
//Cube is entirely in/out of the surface
if (cubeindex ==0 || cubeindex == 255)
return;
Vertex Shader Geometry Shader Fragment Shader

Conclusion

- Debugging solution for the whole shader pipeline
  - Fits well in the development pipeline
  - More than just *printf* debugging
- Limitations
  - Relies on correctness and reliability of drivers
  - No vendor specific GLSL spec. enhancements
  - No breakpointing

Thank you!

Project webpage and download:

http://www.vis.uni-stuttgart.de/glsldevil