



Developer Tools

Tim Purcell
NVIDIA

GP GPU

Programming Soap Box

- Successful programming systems require at least three 'tools'
 - High level language compiler
 - Cg, HLSL, GLSL, RTSL, Brook...
 - Debugger
 - Profiler

Debugging

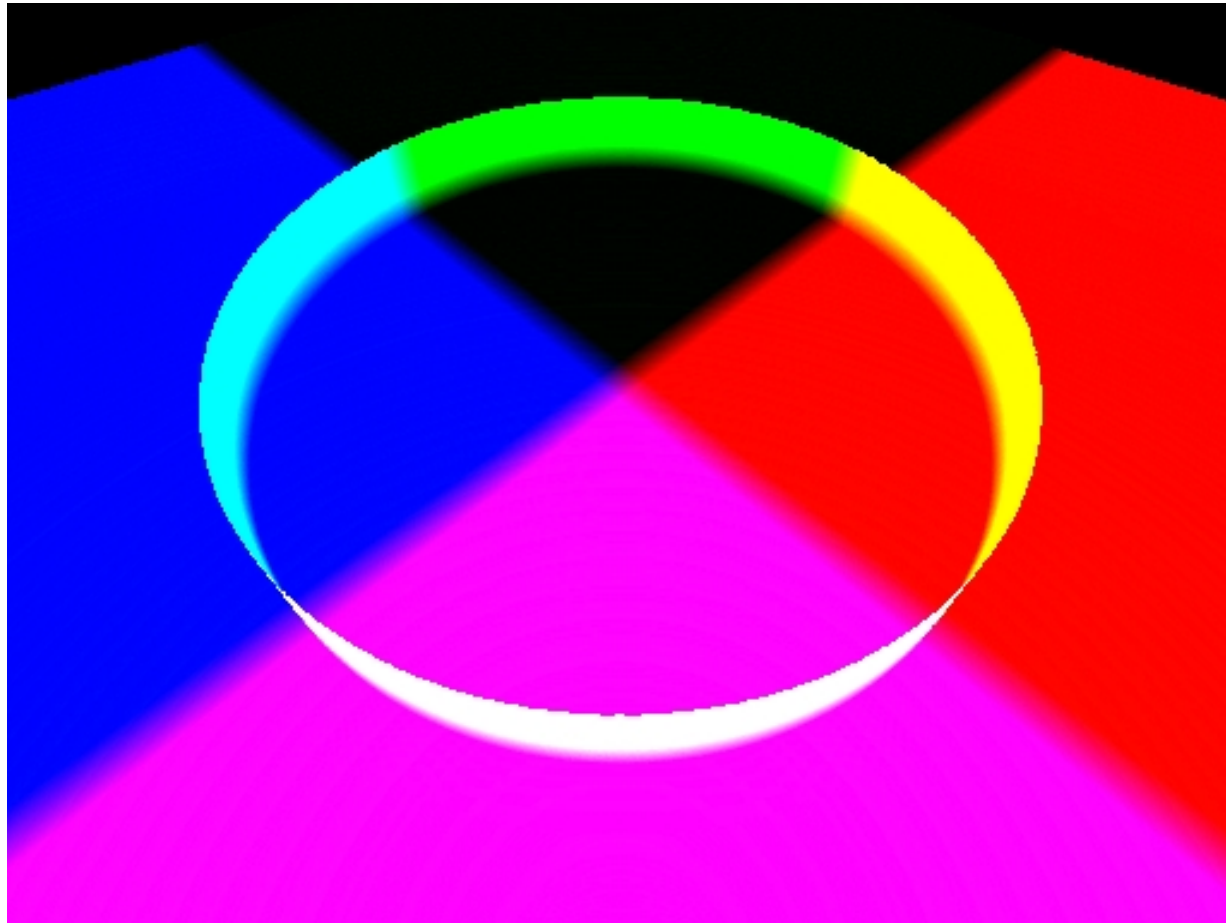
Ideal Fragment Program Debugger

- Automate 'printf' debugging

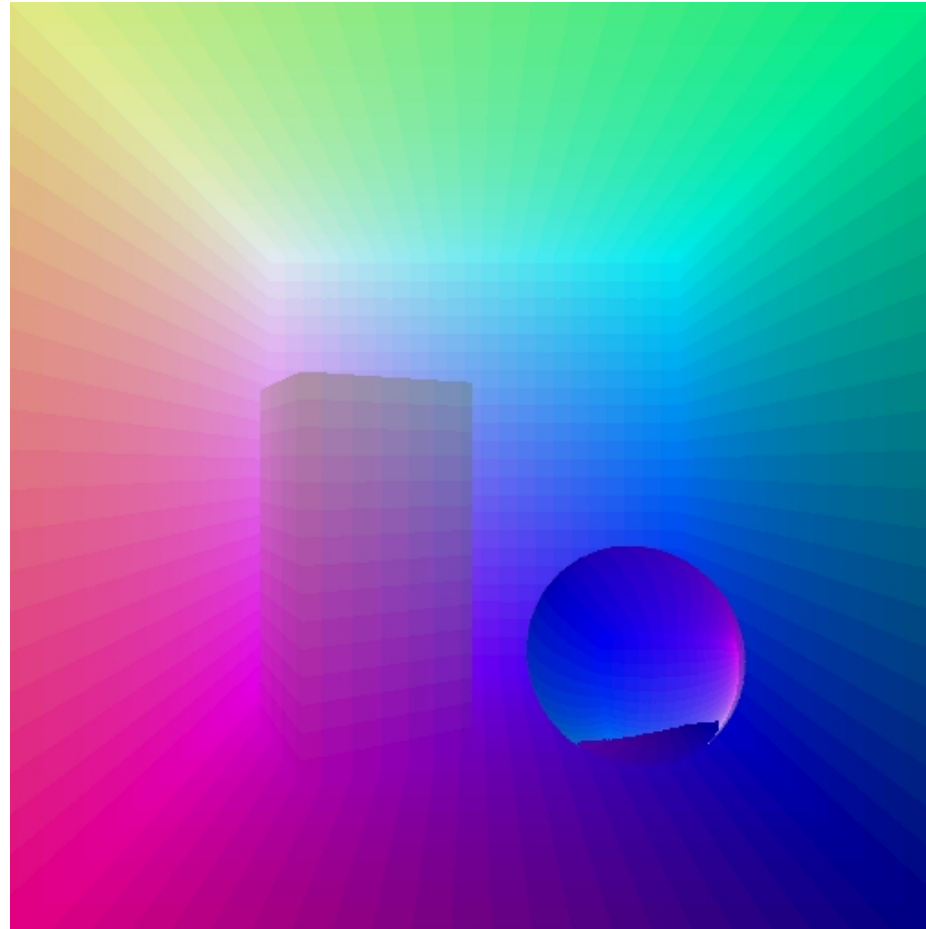
'printf' Debugging

- MOV suspect register to output
 - Comment out anything else writing to output
 - Scale and bias as needed
- Recompile
- Display/readback frame buffer
- Check values
- Repeat until error is (hopefully) found

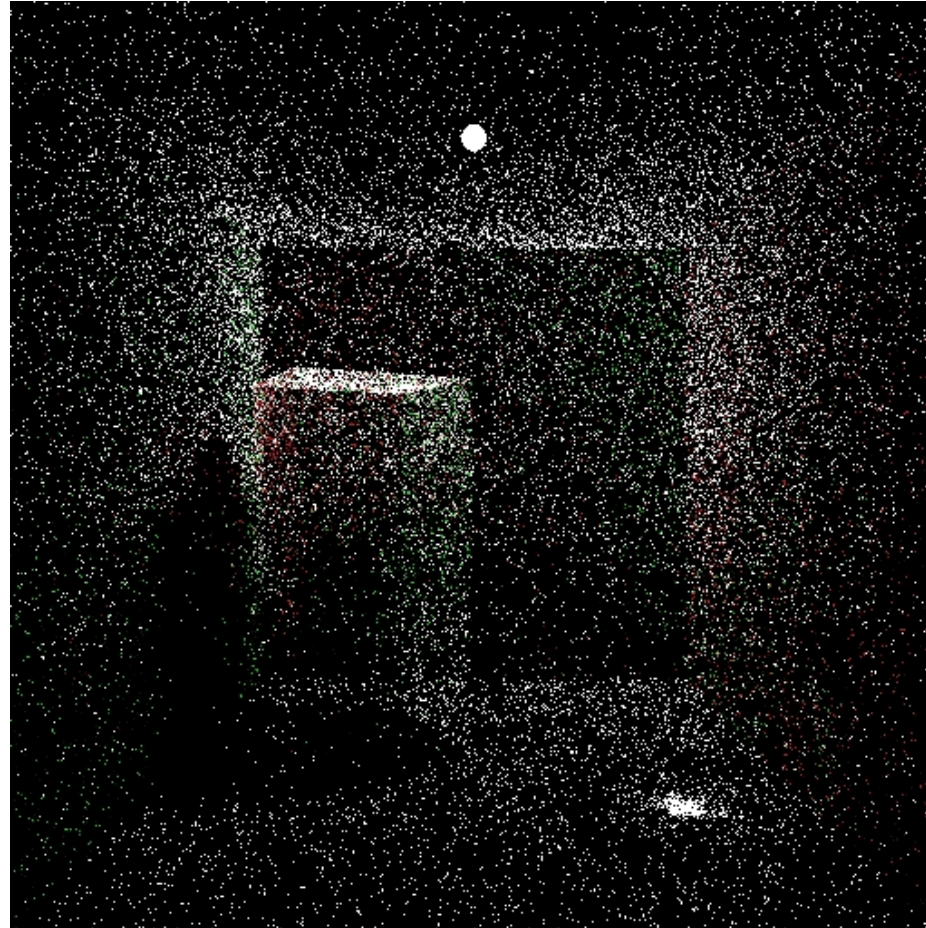
'printf' Debugging Examples



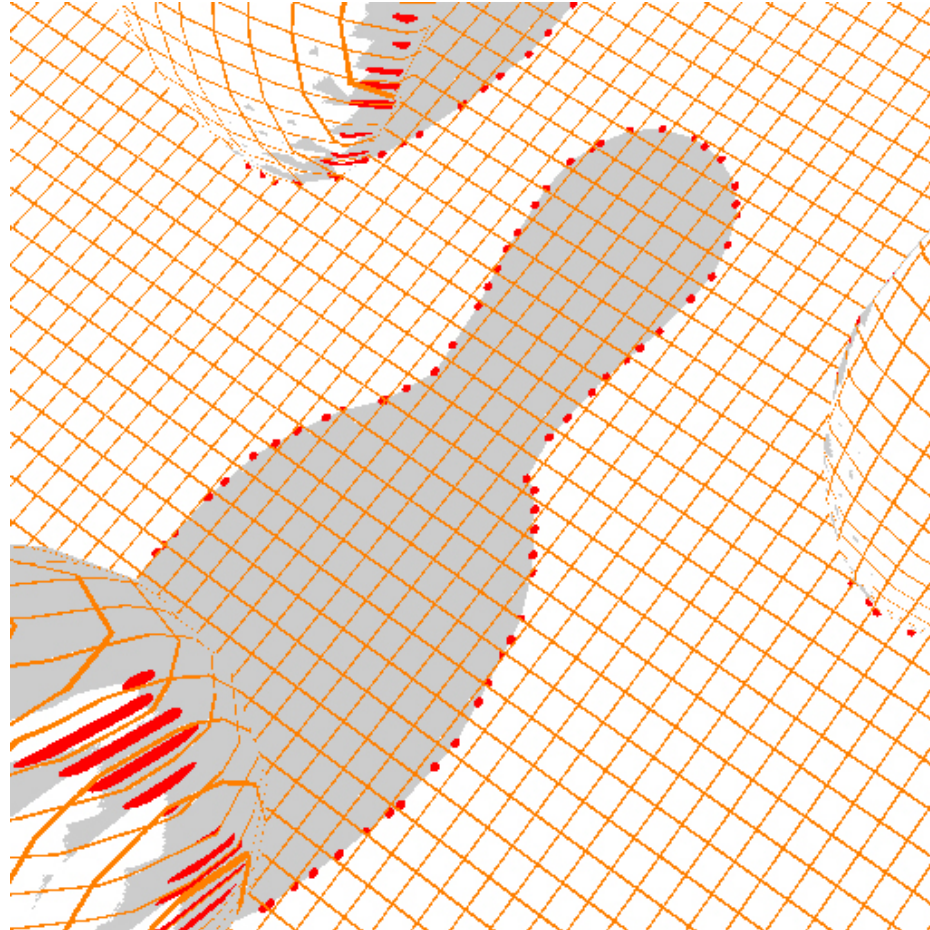
'printf' Debugging Examples



'printf' Debugging Examples



'printf' Debugging Examples



Ideal Fragment Program Debugger

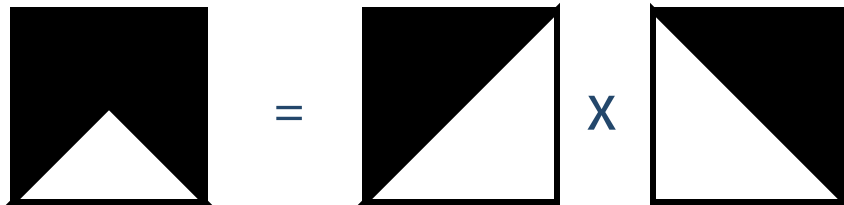
- Automate 'printf' debugging
- Intuitive and easy to use interface
- Features over performance
 - Debuggers don't need to be fast
 - Should still be interactive
- Easy to add to existing apps
 - And remove
- Touch as little GPU state as possible
- Report actual hardware values
 - No software simulations!

Debugger Features

- **Per-pixel register watch**
 - Including interpolants, outputs, etc.
- **Breakpoints**
- **Fragment program interpreter**
 - Single step forwards or backwards
 - Execute modified code on the fly
 - And save it

Debugger Features - Visualization

- **Display register value at each pixel**
 - Color channel masking
 - Arbitrary code for visualization
 - Allows scale and bias of values
- **Multiple visualization windows**
 - Visualize each source register and result
 - MUL R2, R1, R0;



Debugging Options Today

- Graphic Remedy gDebugger
- GLIntercept - [D. Trebilco]
- Microsoft Shader Debugger Tool
- Apple OpenGL Shader Builder
- Imdebug - The Image Debugger [B. Baxter]
- Shadersmith - [T. Purcell, P. Sen]
- Relational Debugging Engine [Duca et al. 2005]

GPU Vendor Debugging Options

- ATI RenderMonkey
- Nvidia FXComposer

- Won't talk about these today

Graphic Remedy gDebugger

- **Pros**

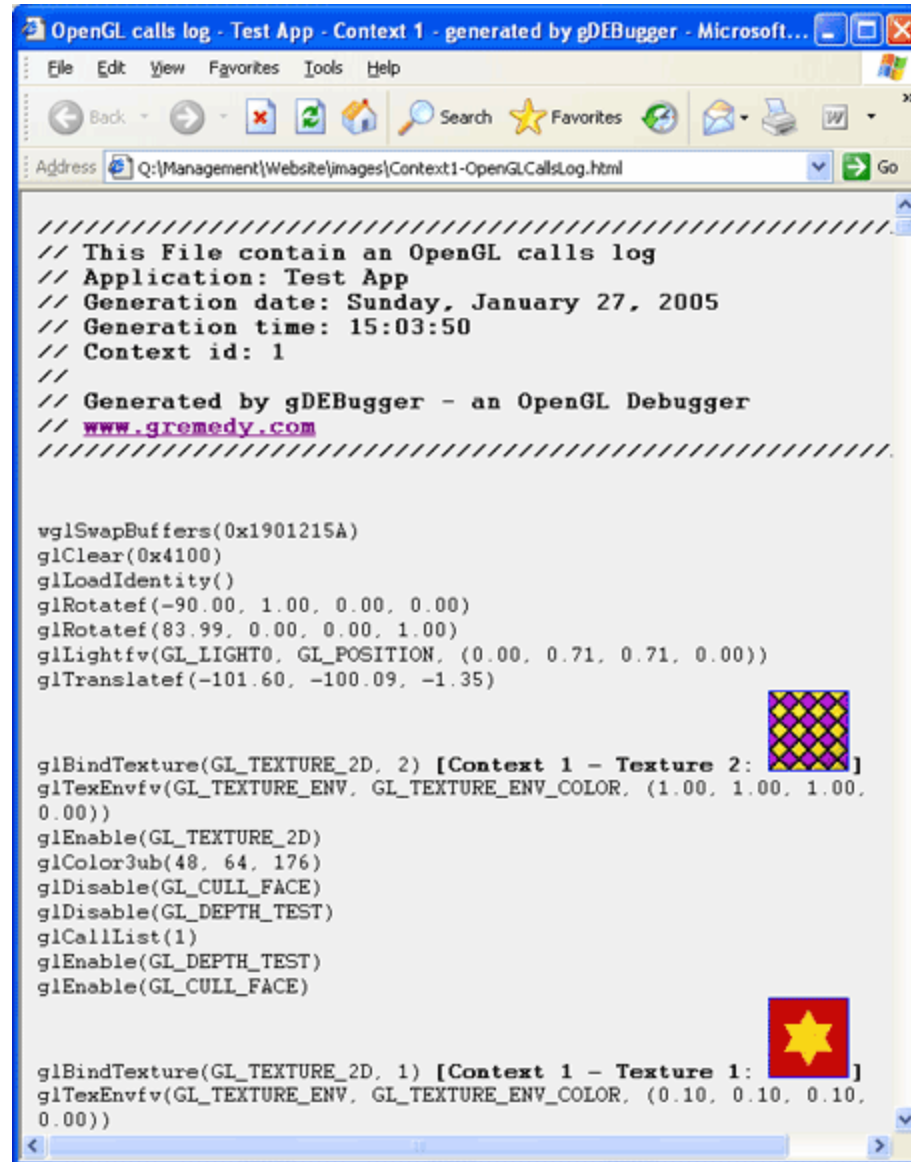
- Breakpoints, stepping
- Watch windows
- Performance analysis
- OpenGL 2.0

- **Cons**

- OpenGL only
- Currently no program debugging


- <http://www.gremedy.com/>


gDebugger



```
OpenGL calls log - Test App - Context 1 - generated by gDebugger - Microsoft...
File Edit View Favorites Tools Help
Address Q:\Management\Website\images\Context1-OpenGLCallsLog.html
////////////////////////////////////
// This File contain an OpenGL calls log
// Application: Test App
// Generation date: Sunday, January 27, 2005
// Generation time: 15:03:50
// Context id: 1
//
// Generated by gDebugger - an OpenGL Debugger
// www.gremedy.com
////////////////////////////////////

wglSwapBuffers(0x1901215A)
glClear(0x4100)
glLoadIdentity()
glRotatef(-90.00, 1.00, 0.00, 0.00)
glRotatef(83.99, 0.00, 0.00, 1.00)
glLightfv(GL_LIGHT0, GL_POSITION, (0.00, 0.71, 0.71, 0.00))
glTranslatef(-101.60, -100.09, -1.35)

glBindTexture(GL_TEXTURE_2D, 2) [Context 1 - Texture 2: ]
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, (1.00, 1.00, 1.00, 0.00))
glEnable(GL_TEXTURE_2D)
glColor3ub(48, 64, 176)
glDisable(GL_CULL_FACE)
glDisable(GL_DEPTH_TEST)
glCallList(1)
glEnable(GL_DEPTH_TEST)
glEnable(GL_CULL_FACE)

glBindTexture(GL_TEXTURE_2D, 1) [Context 1 - Texture 1: ]
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, (0.10, 0.10, 0.10, 0.00))
```

GLIntercept

- **Pros**
 - Track all OpenGL state
 - Runtime shader edit and recompile
 - Resource tracking and management
- **Cons**
 - OpenGL only
 - No shader debugging
 - No direct support for visualizations
- <http://glintercept.nutty.org/>

GLIntercept

Flash the selected shader in the render window

Re-compile the shader into the application

The last frame the shader was used

Full tool tip support for GLSL

Error line reporting

Use editor as a stand alone shader editor

```
#define lerp

uniform sampler2D Base;
uniform sampler2D Bump;

uniform float invRadius;
uniform float ambient;

varying vec2 texCoord;
varying vec3 lightVec;
varying vec3 viewVec;

void main() {
    vec4 base = texture2D(Base, texCoord);
    vec3 bump = texture2D(Bump, texCoord).xyz * 2.0 - 1.0;

    bump = normalize(bump);

    float distSqr = dot(lightVec, lightVec);
    vec3 lVec = lightVec * inversesqrt(distSqr);

    float atten = clamp(1.0 - invRadius * sqrt(distSqr), 0.0, 1.0);
    float diffuse = clamp(dot(lVec, bump), 0.0, 1.0);

    float specular = pow(clamp(dot(reflect(normalize(-viewVec), bump), bump), 0.0, 1.0), 2.0);
    vec4 foobar;
    gl_FragColor = ambient * base + cos(diffuse * base + 0.6 * specular) * foobar;
}
```

| Shader UID | OpenGL ID | Type | Frame Num |
|------------|-----------|------|-----------|
| 5 | 1 | GLSL | 48 |

[1 of 4] cos(float angle) = float;
The standard trigonometric cosine function.

ERROR: 0:64: 'foobar' : undeclared identifier
ERROR: 0:65: 'gl_FragColor' : syntax error parse error

Compiling: 5_1.glsl
ll=65 co=39 INS (CR+F)

Microsoft Shader Debugger Tool

- **Pros**

- Direct3D debugging integrated into Visual Studio IDE
- Full featured
 - Assembly and high level debugging
 - Vertex and fragment programs
 - Watches, breakpoints, etc.

- **Cons**

- Only works with software rasterizer
 - Slow and painful
- D3D only
- Shader changes require recompilation
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/Tools/ShaderDebugger.asp

Apple OpenGL Shader Builder

- **Pros**

- Integrated development environment
- Handy reference guide, resource manager
- Texture editor
- On the fly edit and display of shader changes

- **Cons**

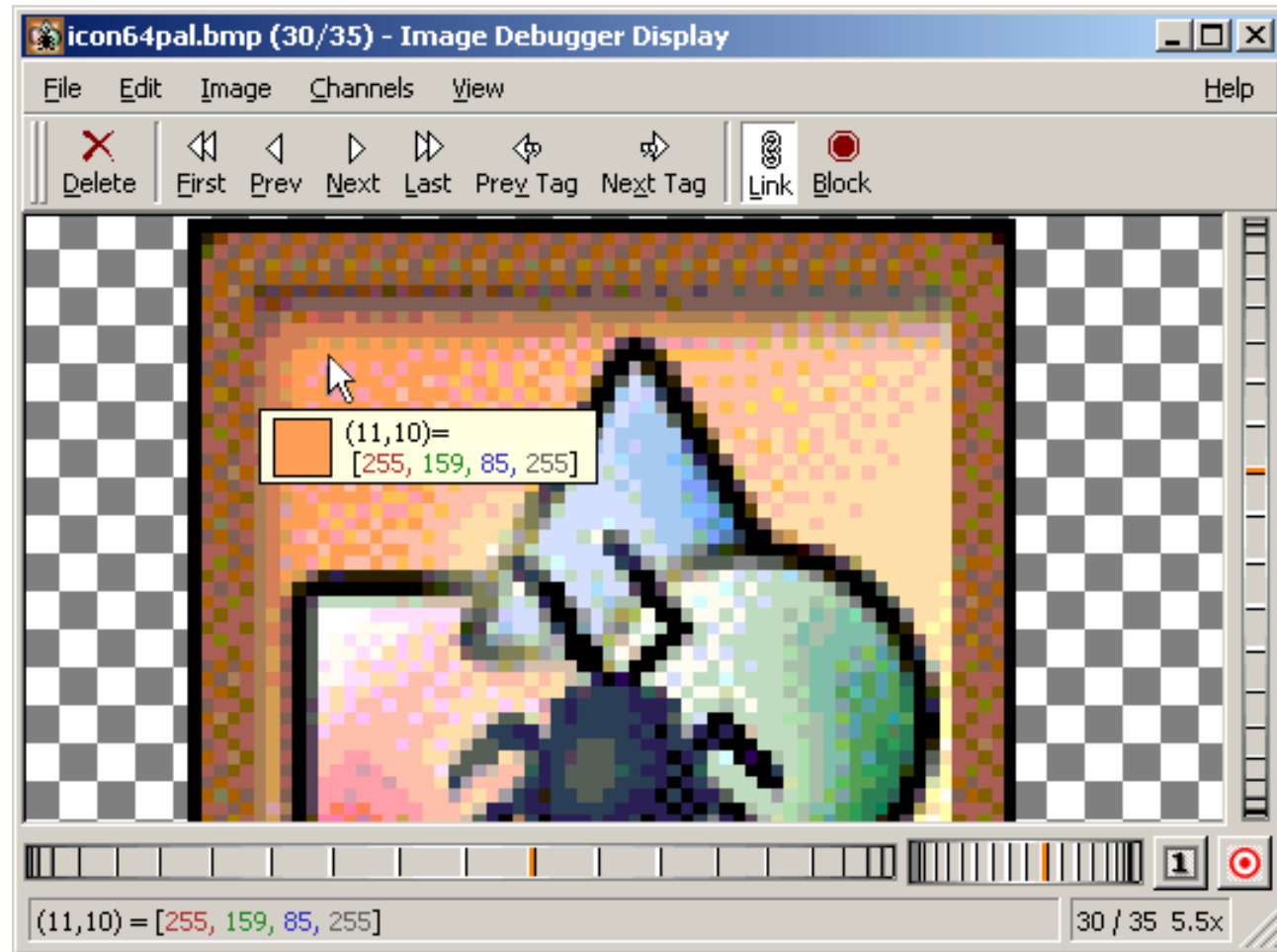
- Canned geometry
 - Basically a shader tool (in the traditional sense)
 - not GPGPU debugger
- ARB vertex/fragment programs only
 - No vendor specific GL extensions support?
- http://developer.apple.com/graphicsimaging/opengl/shader_image.html

imdebug

- **Printf-style debugger**
 - `imdebug("rgb w=%d h=%d %p", 16, 17, testRGB);`
 - Readback memory (texture/frame buffer) and display in image window

- <http://www.billbaxter.com/projects/imdebug/>

imdebug



imdebug

- **Pros**

- Simple addition of single printf-style statement to programs
- Displays hardware computed values - not software generated values
- Scale and bias
- Source available for download

- **Cons**

- Can't breakpoint shaders
- Can only watch what shader outputs

Shadesmith

- **Debugger in the spirit of imdebug**
 - Simply add a debug statement when binding shaders
 - Display window with scale, bias, component masking
- **Advanced features**
 - Can watch any shader register contents without recompile
 - Shader single stepping (forward and backward), breakpointing
 - Shader source edit and reload without recompile
- <http://graphics.stanford.edu/projects/shadesmith/>

Shadersmith Implementation Insight

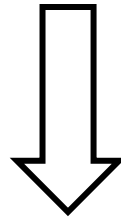
- Only one register modified per instruction
 - Ignore CC for now
- Decompose fragment program into several smaller programs
 - One program per assembly instruction
 - Save register state on host
 - Iterative deepening decomposition

Iterative Deepening

```
...  
ADD R0, R1, f[WPOS];  
MAD R1, R0, R2, R3;  
TEX R2, R1, TEX0, RECT;  
...
```

Iterative Deepening

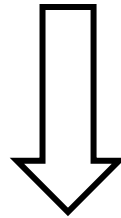
```
...  
ADD R0, R1, f[WPOS];  
MAD R1, R0, R2, R3;  
TEX R2, R1, TEX0, RECT;  
...
```



```
...  
ADD R0, R1, f[WPOS];  
MAD R1, R0, R2, R3;  
MOV o[COLR], R1;  
END
```

Iterative Deepening

```
...  
ADD R0, R1, f[WPOS];  
MAD R1, R0, R2, R3;  
TEX R2, R1, TEX0, RECT;  
...
```



```
...  
ADD R0, R1, f[WPOS];  
MAD R1, R0, R2, R3;  
TEX R2, R1, TEX0, RECT;  
MOV o[COLR], R2;  
END
```

Later code more expensive than early code

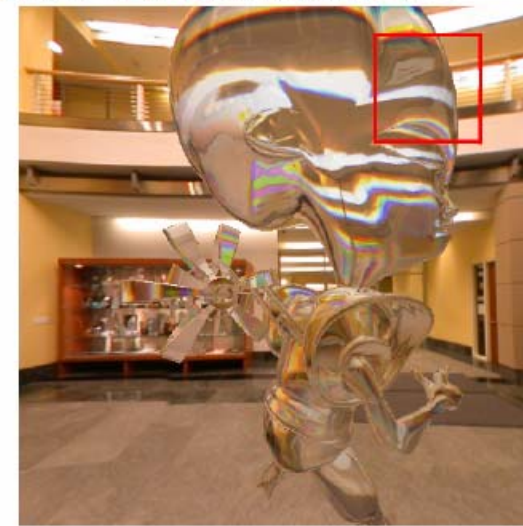
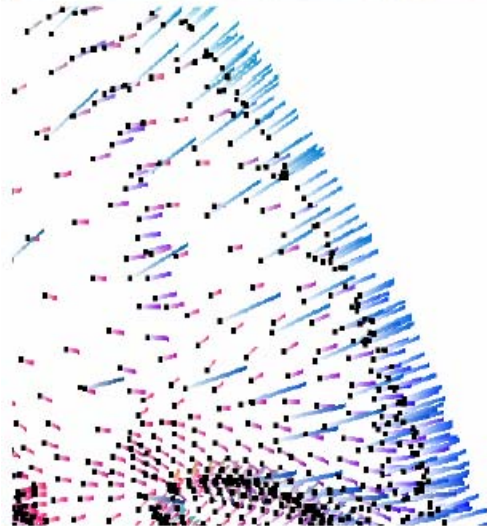
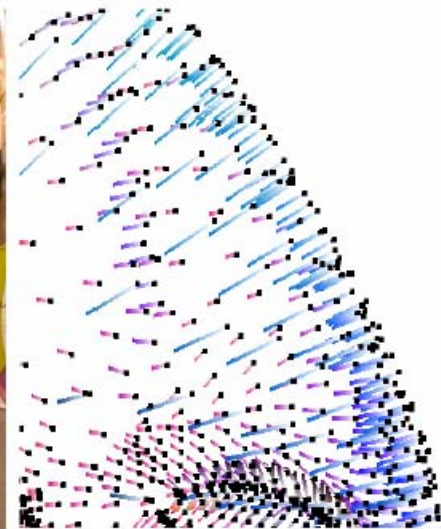
Basic Shadersmith Flow

- **Decompose program into smaller programs**
 - Use iterative deepening approach
- **Run programs required to determine watch values**
 - One program per value watched
 - Readback modified register to host
 - Via `glReadPixels()`
- **Display register values per pixel**
 - Visualization windows
 - Per-pixel values on mouseover

Relational Debugging Engine

- **Build database of GPU state**
 - Including pipeline state, shader state
- **SQL-like queries generate visualizations**
 - Including vertex programs
 - Raw text output available as well
- **Built on top of Chromium**
 - Can debug any OpenGL application without recompilation
- **Current system assumes Cg shaders**
 - Approach is applicable to other GPU languages

Relational Debugging Engine

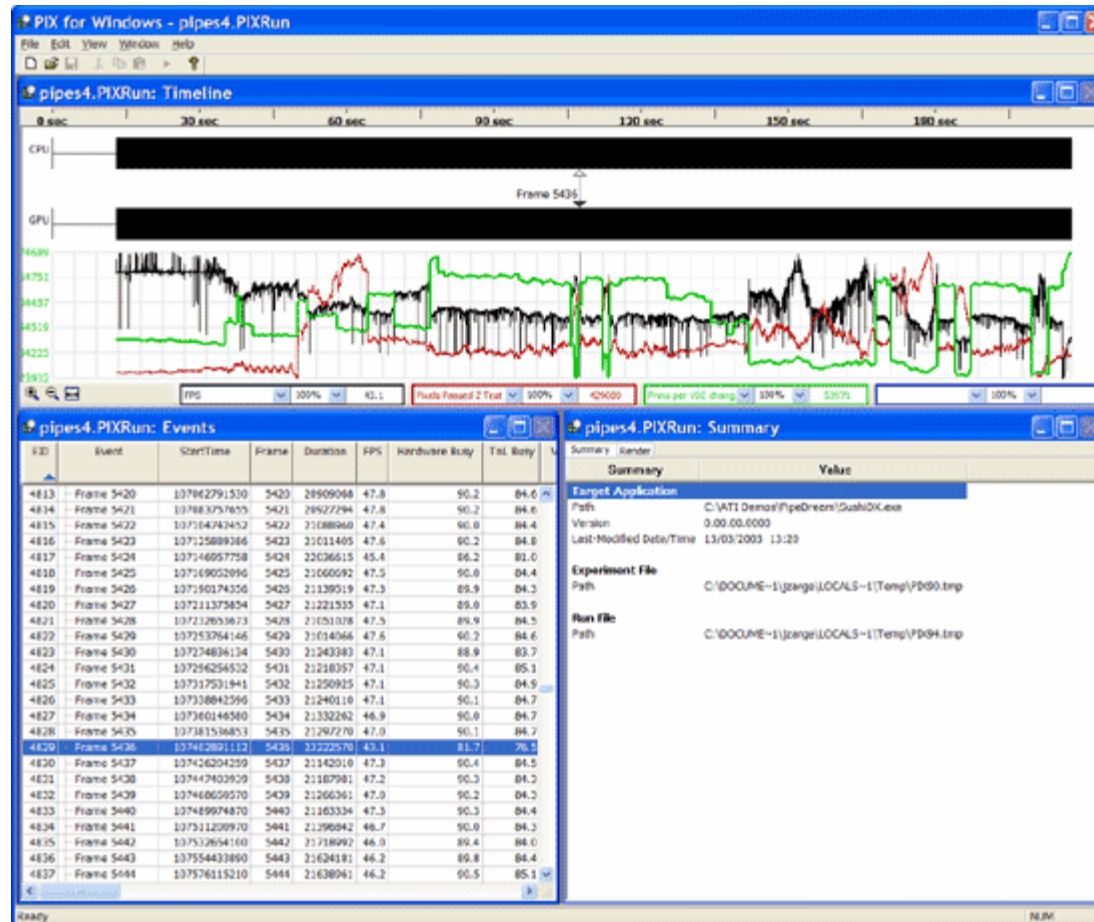


Profiling

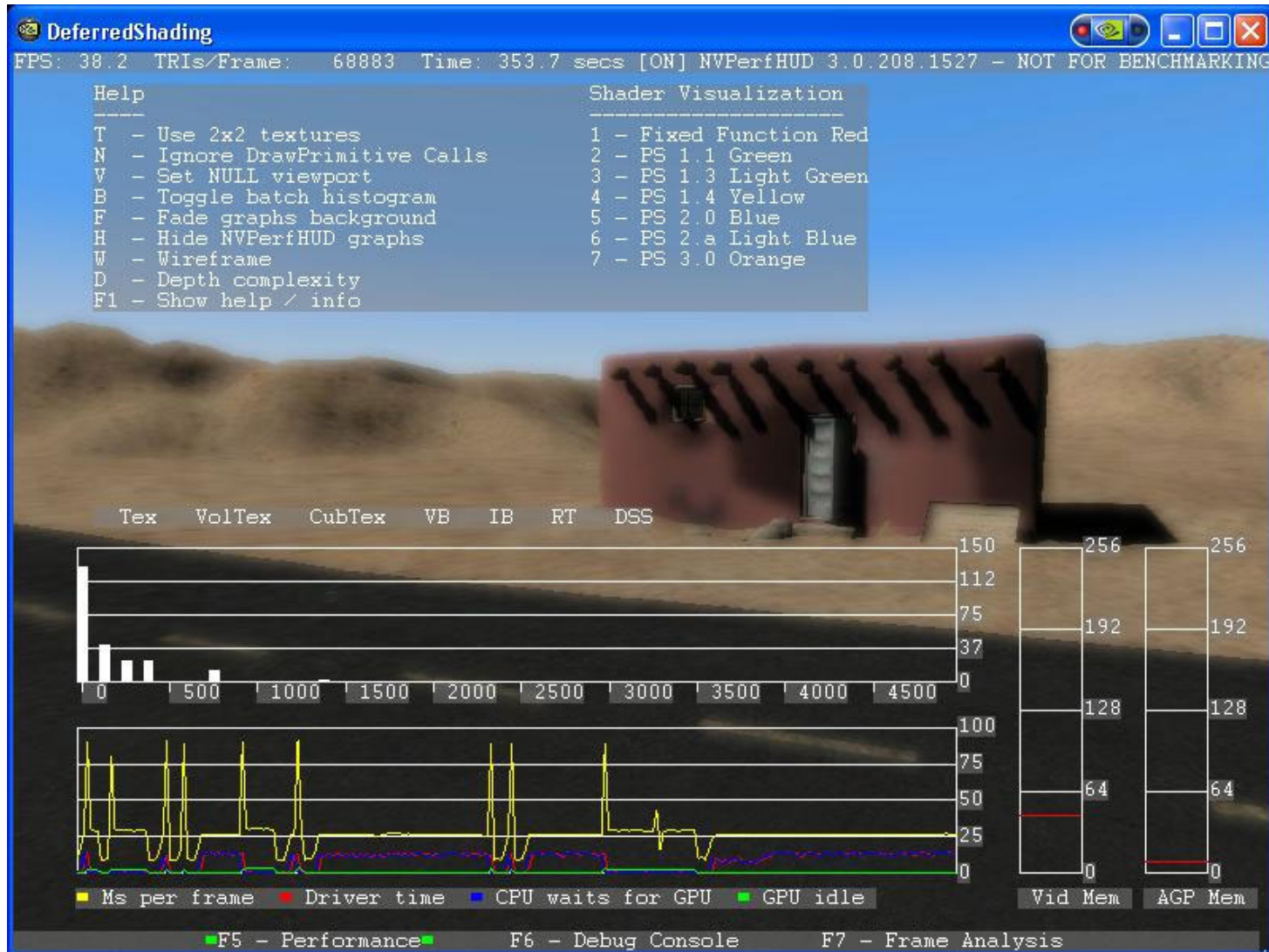
Profiling

- Not much from 3rd parties
 - GPU Architecture internals mostly 'secret'
- ATIPixPlugin
- NVPerfHUD
- NVPerfKit
 - Supported by gDebugger 2.0 release

ATIPIXPlugin



NVPerfHUD



NVPerfKit

- **Instrumented Driver**
 - OpenGL and Direct3D support
- **NVIDIA Developer Control Panel**
- **NVIDIA Plug-in for Microsoft PIX for Windows**
- **Access to performance counters via PDH**
 - Support for PerfMon, VTune, gDEDebugger, and more

NVPerfKit

