NVISION 08
THE WORLD OF VISUAL COMPUTING

gDEBugger - Advanced OpenGL Debugger and profiler

Avi Shapira, Graphic Remedy

NVIDIA

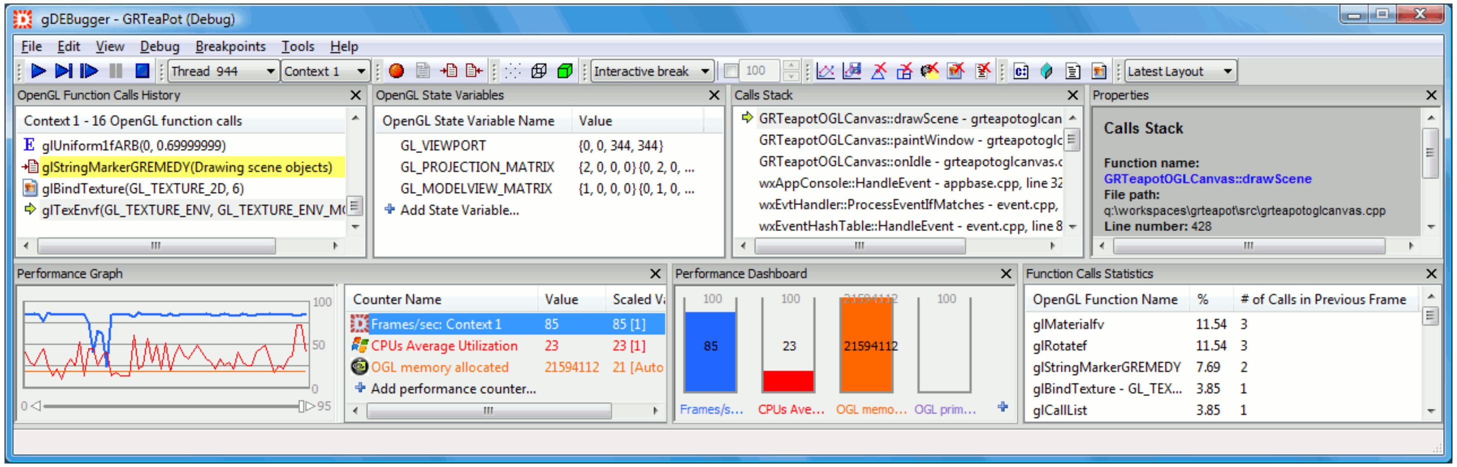
The problem

- Debugging and profiling 3D applications are both hard and time consuming tasks
- Companies find it extremely hard to deliver a robust and bug-free 3D application
- It is almost impossible to optimize a 3D application to fully utilize the graphic system performance

Graphic Remedy offers a unique solution

gDEBugger

- OpenGL and OpenGL ES Debugger and Profiler
- Exposes internal graphic system information needed to find bugs
- Locates graphic system performance bottlenecks


gDEBugger - GRTeaPot (Debug)
File Edit View Debug Breakpoints Tools Help
Thread 944
Context 1
Interactive break
100
Latest Layout

OpenGL Function Calls History

Context 1 - 16 OpenGL function calls
- glUniform1fARB(0, 0.69999999)
- glStringMarkerGREMEDY(Drawing scene objects)
- glBindTexture(GL_TEXTURE_2D, 6)
- glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_M

OpenGL State Variables

OpenGL State Variable Name	Value
GL_VIEWPORT	{0, 0, 344, 344}
GL_PROJECTION_MATRIX	{2, 0, 0, 0} {0, 2, 0, ...
GL_MODELVIEW_MATRIX	{1, 0, 0, 0} {0, 1, 0, ...
Add State Variable...	

Calls Stack

- GRTeapotOGLCanvas::drawScene - grteapotoglcan
- GRTeapotOGLCanvas::paintWindow - grteapotoglc
- GRTeapotOGLCanvas::onIdle - grteapotoglcanvas.c
- wxAppConsole::HandleEvent - appbase.cpp, line 32
- wxEvtHandler::ProcessEventIfMatches - event.cpp,
- wxEventHashTable::HandleEvent - event.cpp, line 8

Properties

Calls Stack

Function name:
GRTeapotOGLCanvas::drawScene
File path:
q:\workspaces\grteapot\src\grteapotoglcanvas.cpp
Line number: 428

Performance Graph

Counter Name	Value	Scaled V
Frames/sec: Context 1	85	85 [1]
CPUs Average Utilization	23	23 [1]
OGL memory allocated	21594112	21 [Auto
Add performance counter...		

Performance Dashboard

Frames/s...	CPUs Ave...	OGL memo...	OGL prim...
85	23	21594112	

Function Calls Statistics

OpenGL Function Name	%	# of Calls in Previous Frame
glMaterialfv	11.54	3
glRotatef	11.54	3
glStringMarkerGREMEDY	7.69	2
glBindTexture - GL_TEX...	3.85	1
glCallList	3.85	1

Customer Benefits

- Reduces the time needed to debug and optimize 3D graphic applications
- Improves application robustness and quality
- Optimizes application performance

Current markets segments

- Our products are being used in various fields such as:
  - Graphic chips manufacturing
  - Games industry
  - Movies production
  - CAD
  - Medical imaging
  - Aerospace
  - Construction engineering
  - Industrial and scientific visualization
  - Telecom industry
  - Electronic devices
  - Defense
  - Being taught and used by universities worldwide

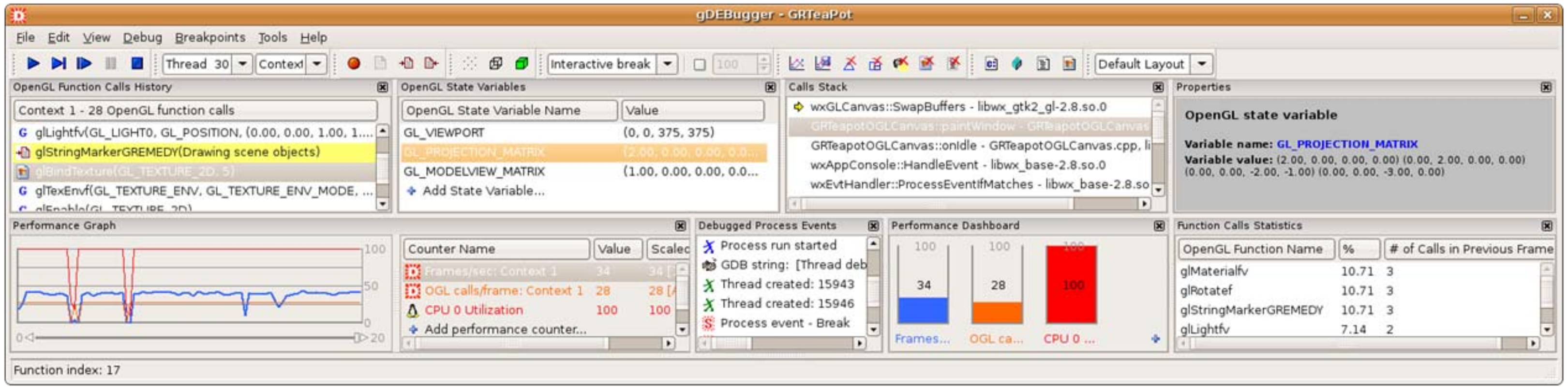
gDEBugger ES (for OpenGL ES)

- Brings all of gDEBugger's Debugging and Profiling abilities to the OpenGL ES developer’s world

- Acts as an OpenGL ES implementation when working on Windows PC

gDEBugger Linux

- Brings all of gDEBugger's Debugging and Profiling abilities to the Linux developers' world



OpenGL Function Calls History

Context 1 - 28 OpenGL function calls
- glLightfv(GL_LIGHT0, GL_POSITION, (0.00, 0.00, 1.00, 1....
- glStringMarkerGREMEDY(Drawing scene objects)
- glBindTexture(GL_TEXTURE_2D, 5)
- glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, ...
- glEnable(GL_TEXTURE_2D)

OpenGL State Variables

OpenGL State Variable Name	Value
GL_VIEWPORT	(0, 0, 375, 375)
GL_PROJECTION_MATRIX	(2.00, 0.00, 0.00, 0.0...
GL_MODELVIEW_MATRIX	(1.00, 0.00, 0.00, 0.0...
Add State Variable...	

Calls Stack

- wxGLCanvas::SwapBuffers - libwx_gtk2_gl-2.8.so.0
- GRTeapotOGLCanvas::paintWindow - GRTeapotOGLCanvas
- GRTeapotOGLCanvas::onIdle - GRTeapotOGLCanvas.cpp, li
- wxAppConsole::HandleEvent - libwx_base-2.8.so.0
- wxEvtHandler::ProcessEventIfMatches - libwx_base-2.8.so

Properties

OpenGL state variable

Variable name: GL_PROJECTION_MATRIX
Variable value: (2.00, 0.00, 0.00, 0.00) (0.00, 2.00, 0.00, 0.00) (0.00, 0.00, -2.00, -1.00) (0.00, 0.00, -3.00, 0.00)

Performance Graph

Counter Name	Value	Scaled
Frames/sec: Context 1	34	34 [
OGL calls/frame: Context 1	28	28 [A
CPU 0 Utilization	100	100
Add performance counter...		

Debugged Process Events

- Process run started
- GDB string: [Thread deb
- Thread created: 15943
- Thread created: 15946
- Process event - Break

Performance Dashboard

Function Calls Statistics

OpenGL Function Name	%	# of Calls in Previous Frame
glMaterialfv	10.71	3
glRotatef	10.71	3
glStringMarkerGREMEDY	10.71	3
glLightfv	7.14	2

Function index: 17

The Problem...

The developer sees OpenGL as a “Black box”!

Application is sending API calls

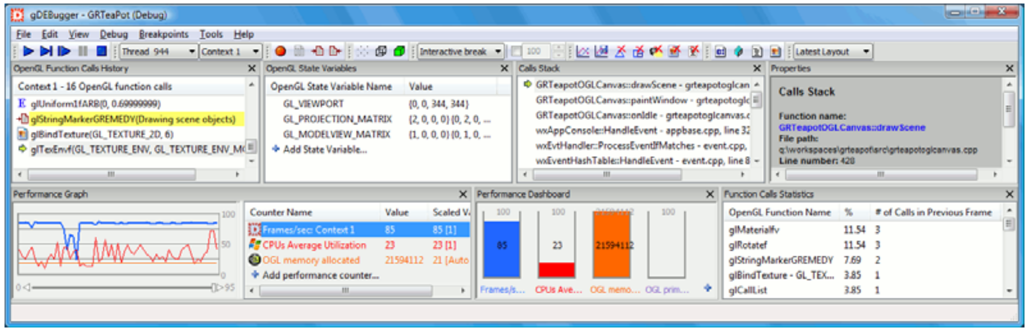
↓

Something happens here!

↓

gDEBugger Debugging & Profiling model

Application is sending API calls


gDEBugger - GRTeaPot (Debug)
File Edit View Debug Breakpoints Tools Help
Thread 944
Context 1
Interactive break
Latest Layout

OpenGL Function Calls History

Context 1 - 16 OpenGL function calls

- glUniform1fARB(0, 0.69999999)
- glStringMarkerGREMEDY(Drawing scene objects)
- glBindTexture(GL_TEXTURE_2D, 6)
- glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_M

OpenGL State Variables

OpenGL State Variable Name	Value
GL_VIEWPORT	{0, 0, 344, 344}
GL_PROJECTION_MATRIX	{2, 0, 0, 0} {0, 2, 0, ...
GL_MODELVIEW_MATRIX	{1, 0, 0, 0} {0, 1, 0, ...
Add State Variable...	

Calls Stack

```
GRTeapotOGLCanvas::drawScene - grteapotoglcan
GRTeapotOGLCanvas::paintWindow - grteapotoglc
GRTeapotOGLCanvas::onIdle - grteapotoglcanvas.c
wxAppConsole::HandleEvent - appbase.cpp, line 32
wxEvtHandler::ProcessEventIfMatches - event.cpp,
wxEventHashTable::HandleEvent - event.cpp, line 8
```

Properties

Calls Stack

Function name: GRTeapotOGLCanvas::drawScene
File path: q:\workspaces\grteapot\src\grteapotoglcanvas.cpp
Line number: 420

Performance Graph

Counter Name	Value	Scaled V
Frames/sec: Context 1	85	85 [1]
CPUs Average Utilization	23	23 [1]
OGL memory allocated	21594112	21 [Auto
Add performance counter...		

Performance Dashboard

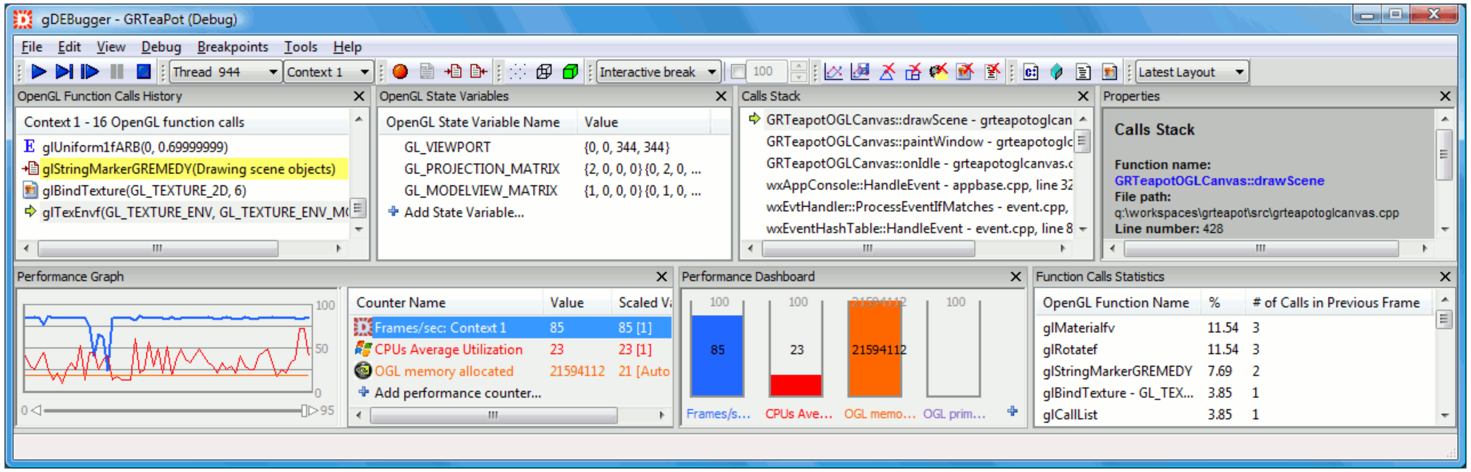
Function Calls Statistics

OpenGL Function Name	%	# of Calls in Previous Frame
glMaterialfv	11.54	3
glRotatef	11.54	3
glStringMarkerGREMEDY	7.69	2
glBindTexture - GL_TEX...	3.85	1
glCallList	3.85	1

Debugging - Demo

GUI Overview

- GUI layouts
- Toolbars and commands
- Views
- Viewers



OpenGL Function Calls History

Context 1 - 16 OpenGL function calls
glUniform1fARB(0, 0.69999999)
glStringMarkerGREMEDY(Drawing scene objects)
glBindTexture(GL_TEXTURE_2D, 6)
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MO

OpenGL State Variables

OpenGL State Variable Name	Value
GL_VIEWPORT	{0, 0, 344, 344}
GL_PROJECTION_MATRIX	{2, 0, 0, 0} {0, 2, 0, ...
GL_MODELVIEW_MATRIX	{1, 0, 0, 0} {0, 1, 0, ...
Add State Variable...	

Calls Stack

GRTeapotOGLCanvas::drawScene - grteapotoglcan
GRTeapotOGLCanvas::paintWindow - grteapotoglc
GRTeapotOGLCanvas::onIdle - grteapotoglcanvas.c
wxAppConsole::HandleEvent - appbase.cpp, line 32
wxEvtHandler::ProcessEventIfMatches - event.cpp,
wxEventHashTable::HandleEvent - event.cpp, line 8

Properties

Calls Stack

Function name: GRTeapotOGLCanvas::drawScene
File path: q:\workspaces\grteapot\src\grteapotoglcanvas.cpp
Line number: 428

Performance Graph: 100, 50, 0, 0, 95

Counter Name	Value	Scaled V
Frames/sec: Context 1	85	85 [1]
CPUs Average Utilization	23	23 [1]
OGL memory allocated	21594112	21 [Auto
Add performance counter...		

Performance Dashboard

Frames/s...	CPUs Ave...	OGL memo...	OGL prim...
100	100	21594112	100
85	23	21594112	

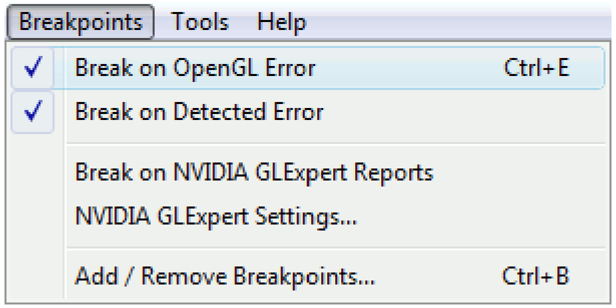
Function Calls Statistics

OpenGL Function Name	%	# of Calls in Previous Frame
glMaterialfv	11.54	3
glRotatef	11.54	3
glStringMarkerGREMEDY	7.69	2
glBindTexture - GL_TEX...	3.85	1
glCallList	3.85	1

Automatic errors detection mechanism

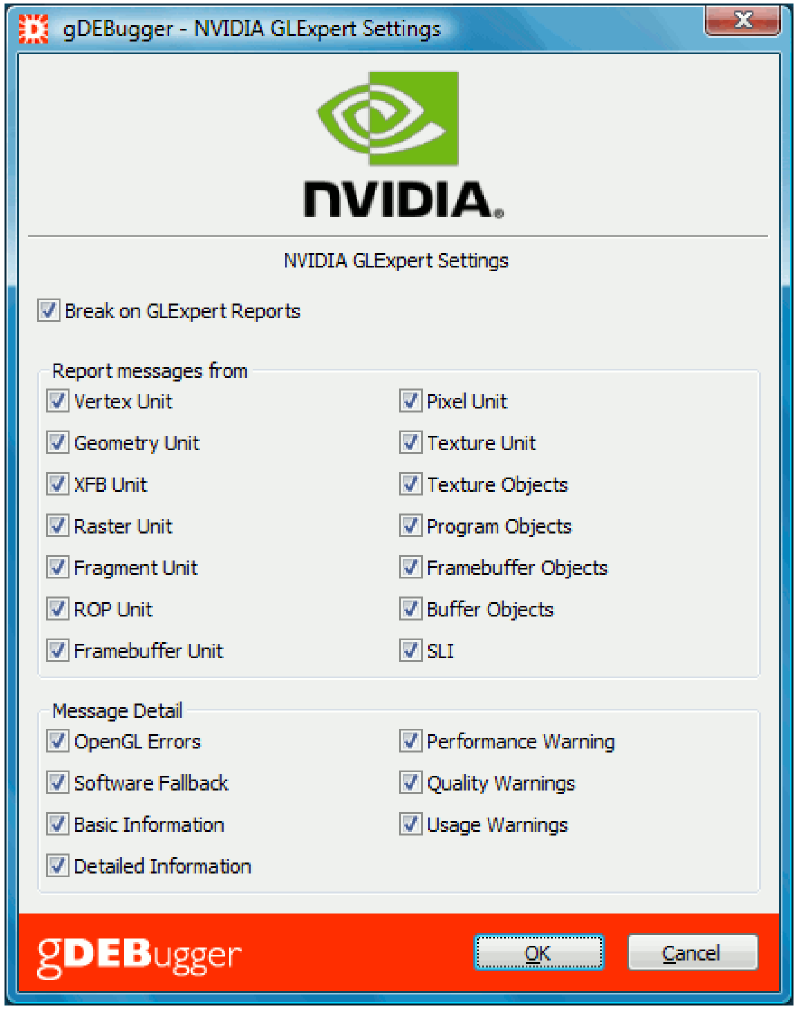
gDEBugger can automatically break on:

- OpenGL errors
- gDEBugger detected errors: a unique and more comprehensive errors detection mechanism
- NVIDIA GLExpert driver reports


Breakpoints
Tools
Help
✓ Break on OpenGL Error Ctrl+E
✓ Break on Detected Error
Break on NVIDIA GLExpert Reports
NVIDIA GLExpert Settings...
Add / Remove Breakpoints... Ctrl+B

gDEBugger and GLExpert driver reports

- Breaks the application run on GLExpert reports
- Views the call stack and source code that led to the GLExpert report
- Sets the GLExpert report areas and message details from within gDEBugger



gDEBugger - NVIDIA GLExpert Settings

NVIDIA GLExpert Settings

☑ Break on GLExpert Reports

Report messages from

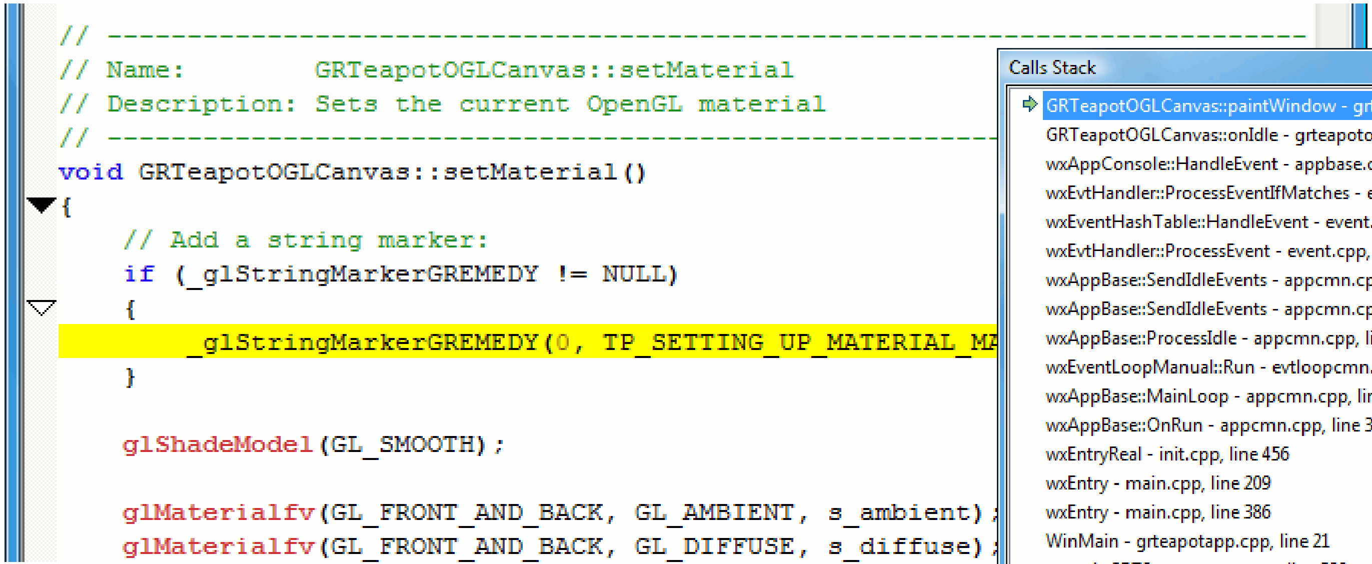
☑ Vertex Unit	☑ Pixel Unit
☑ Geometry Unit	☑ Texture Unit
☑ XFB Unit	☑ Texture Objects
☑ Raster Unit	☑ Program Objects
☑ Fragment Unit	☑ Framebuffer Objects
☑ ROP Unit	☑ Buffer Objects
☑ Framebuffer Unit	☑ SLI

Message Detail

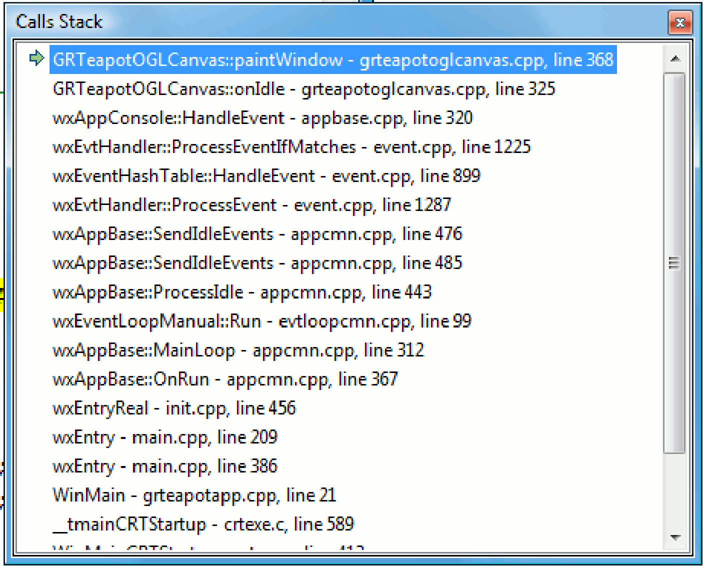
☑ OpenGL Errors	☑ Performance Warning
☑ Software Fallback	☑ Quality Warnings
☑ Basic Information	☑ Usage Warnings
☑ Detailed Information	

Call stack and Source code views

- View the call stack and source code that led to the error / OpenGL function call



```
// ---------------------------------------------------------------------------
// Name:        GRTeapotOGLCanvas::setMaterial
// Description: Sets the current OpenGL material
// ---------------------------------------------------------------------------
void GRTeapotOGLCanvas::setMaterial()
{
    // Add a string marker:
    if (_glStringMarkerGREMEDY != NULL)
    {
        _glStringMarkerGREMEDY(0, TP_SETTING_UP_MATERIAL_MA
    }

    glShadeModel(GL_SMOOTH);

    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, s_ambient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, s_diffuse);
```

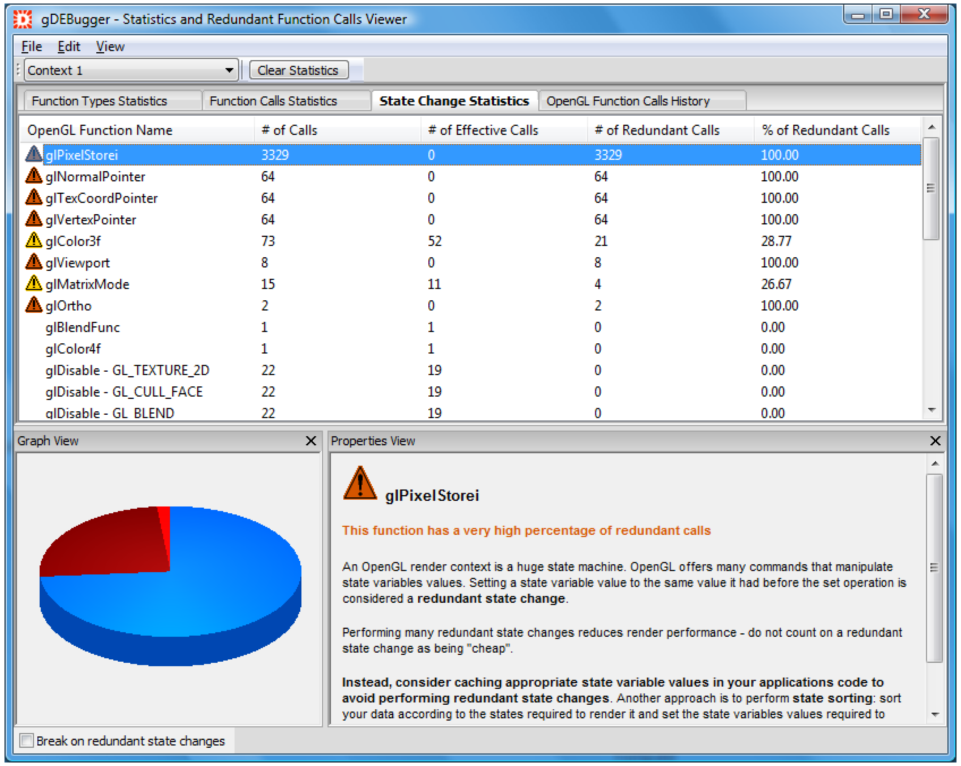


Calls Stack

```
GRTeapotOGLCanvas::paintWindow - grteapotoglcanvas.cpp, line 368
GRTeapotOGLCanvas::onIdle - grteapotoglcanvas.cpp, line 325
wxAppConsole::HandleEvent - appbase.cpp, line 320
wxEvtHandler::ProcessEventIfMatches - event.cpp, line 1225
wxEventHashTable::HandleEvent - event.cpp, line 899
wxEvtHandler::ProcessEvent - event.cpp, line 1287
wxAppBase::SendIdleEvents - appcmn.cpp, line 476
wxAppBase::SendIdleEvents - appcmn.cpp, line 485
wxAppBase::ProcessIdle - appcmn.cpp, line 443
wxEventLoopManual::Run - evtloopcmn.cpp, line 99
wxAppBase::MainLoop - appcmn.cpp, line 312
wxAppBase::OnRun - appcmn.cpp, line 367
wxEntryReal - init.cpp, line 456
wxEntry - main.cpp, line 209
wxEntry - main.cpp, line 386
WinMain - grteapotapp.cpp, line 21
__tmainCRTStartup - crtexe.c, line 589
```

Statistical and Redundant calls viewer

- Displays the number of times each OpenGL function was called
- Display redundant state changes
- Break on redundant state change


gDEBugger - Statistics and Redundant Function Calls Viewer
File Edit View
Context 1
Clear Statistics
Function Types Statistics
Function Calls Statistics
State Change Statistics
OpenGL Function Calls History
Graph View
Properties View

OpenGL Function Name	# of Calls	# of Effective Calls	# of Redundant Calls	% of Redundant Calls
glPixelStorei	3329	0	3329	100.00
glNormalPointer	64	0	64	100.00
glTexCoordPointer	64	0	64	100.00
glVertexPointer	64	0	64	100.00
glColor3f	73	52	21	28.77
glViewport	8	0	8	100.00
glMatrixMode	15	11	4	26.67
glOrtho	2	0	2	100.00
glBlendFunc	1	1	0	0.00
glColor4f	1	1	0	0.00
glDisable - GL_TEXTURE_2D	22	19	0	0.00
glDisable - GL_CULL_FACE	22	19	0	0.00
glDisable - GL_BLEND	22	19	0	0.00

glPixelStorei

This function has a very high percentage of redundant calls

An OpenGL render context is a huge state machine. OpenGL offers many commands that manipulate state variables values. Setting a state variable value to the same value it had before the set operation is considered a **redundant state change**.

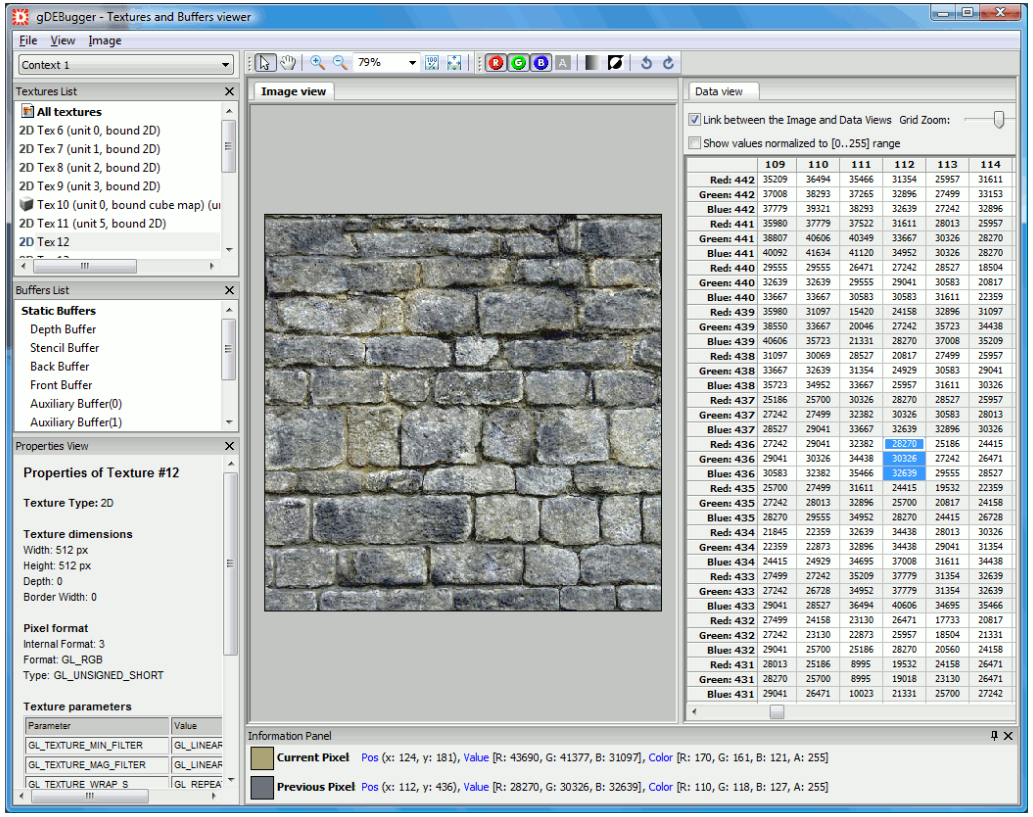
Performing many redundant state changes reduces render performance - do not count on a redundant state change as being "cheap".

Instead, consider caching appropriate state variable values in your applications code to avoid performing redundant state changes. Another approach is to perform **state sorting**: sort your data according to the states required to render it and set the state variables values required to

☐ Break on redundant state changes

Textures and Buffers viewer

Displays all texture types, FBOs, pBuffers, Depth, Stencil, etc.



Textures List

All textures
2D Tex 6 (unit 0, bound 2D)
2D Tex 7 (unit 1, bound 2D)
2D Tex 8 (unit 2, bound 2D)
2D Tex 9 (unit 3, bound 2D)
Tex 10 (unit 0, bound cube map) (u
2D Tex 11 (unit 5, bound 2D)
2D Tex 12

Buffers List

Static Buffers
Depth Buffer
Stencil Buffer
Back Buffer
Front Buffer
Auxiliary Buffer(0)
Auxiliary Buffer(1)

Properties View

Properties of Texture #12

Texture Type: 2D

Texture dimensions
Width: 512 px
Height: 512 px
Depth: 0
Border Width: 0

Pixel format
Internal Format: 3
Format: GL_RGB
Type: GL_UNSIGNED_SHORT

Texture parameters

Parameter	Value
GL_TEXTURE_MIN_FILTER	GL_LINEAR
GL_TEXTURE_MAG_FILTER	GL_LINEAR
GL_TEXTURE_WRAP_S	GL_REPEA

Data view

[✓] Link between the Image and Data Views Grid Zoom:
[ ] Show values normalized to [0..255] range

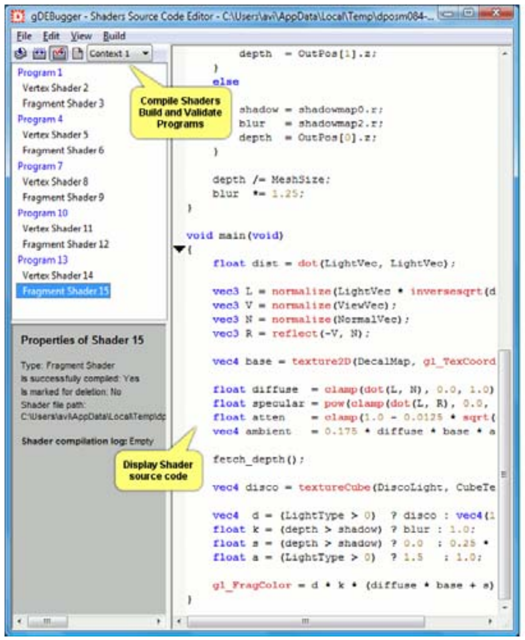
	109	110	111	112	113	114
Red: 442	35209	36494	35466	31354	25957	31611
Green: 442	37008	38293	37265	32896	27499	33153
Blue: 442	37779	39321	38293	32639	27242	32896
Red: 441	35980	37779	37522	31611	28013	25957
Green: 441	38807	40606	40349	33667	30326	28270
Blue: 441	40092	41634	41120	34952	30326	28270
Red: 440	29555	29555	26471	27242	28527	18504
Green: 440	32639	32639	29555	29041	30583	20817
Blue: 440	33667	33667	30583	30583	31611	22359
Red: 439	35980	31097	15420	24158	32896	31097
Green: 439	38550	33667	20046	27242	35723	34438
Blue: 439	40606	35723	21331	28270	37008	35209
Red: 438	31097	30069	28527	20817	27499	25957
Green: 438	33667	32639	31354	24929	30583	29041
Blue: 438	35723	34952	33667	25957	31611	30326
Red: 437	25186	25700	30326	28270	28527	25957
Green: 437	27242	27499	32382	30326	30583	28013
Blue: 437	28527	29041	33667	32639	32896	30326
Red: 436	27242	29041	32382	28270	25186	24415
Green: 436	29041	30326	34438	30326	27242	26471
Blue: 436	30583	32382	35466	32639	29555	28527
Red: 435	25700	27499	31611	24415	19532	22359
Green: 435	27242	28013	32896	25700	20817	24158
Blue: 435	28270	29555	34952	28270	24415	26728
Red: 434	21845	22359	32639	34438	28013	30326
Green: 434	22359	22873	32896	34438	29041	31354
Blue: 434	24415	24929	34695	37008	31611	34438
Red: 433	27499	27242	35209	37779	31354	32639
Green: 433	27242	26728	34952	37779	31354	32639
Blue: 433	29041	28527	36494	40606	34695	35466
Red: 432	27499	24158	23130	26471	17733	20817
Green: 432	27242	23130	22873	25957	18504	21331
Blue: 432	29041	25700	25186	28270	20560	24158
Red: 431	28013	25186	8995	19532	24158	26471
Green: 431	28270	25700	8995	19018	23130	26471
Blue: 431	29041	26471	10023	21331	25700	27242

Information Panel

Current Pixel Pos (x: 124, y: 181), Value [R: 43690, G: 41377, B: 31097], Color [R: 170, G: 161, B: 121, A: 255]
Previous Pixel Pos (x: 112, y: 436), Value [R: 28270, G: 30326, B: 32639], Color [R: 110, G: 118, B: 127, A: 255]

Shader's Source Code Editor

- Displays shader's source code
- Displays programs active uniform values
- Edit shader's source code, re-compile, link and validate "on the fly"

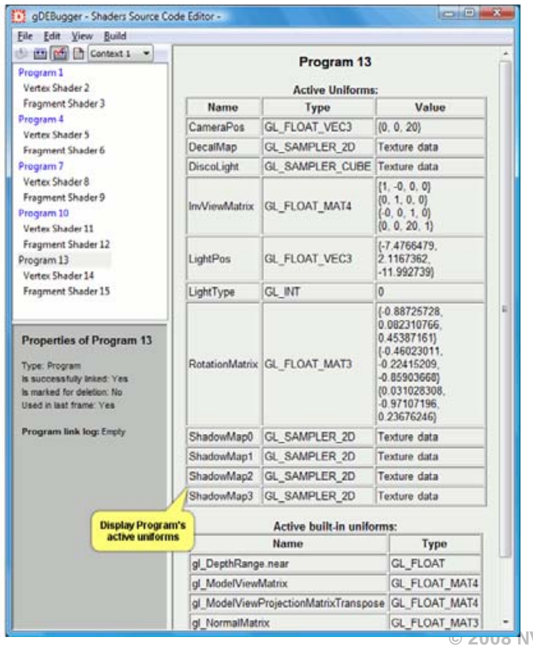


Properties of Shader 15

Type: Fragment Shader
Is successfully compiled: Yes
Is marked for deletion: No
Shader file path:
C:\Users\avi\AppData\Local\Temp\dp

Shader compilation log: Empty

```
        depth  = OutPos[1].z;
    }
    else
    {
        shadow = shadowmap0.r;
        blur   = shadowmap2.r;
        depth  = OutPos[0].z;
    }

    depth /= MeshSize;
    blur  *= 1.25;
}

void main(void)
{
    float dist = dot(LightVec, LightVec);

    vec3 L = normalize(LightVec * inversesqrt(d
    vec3 V = normalize(ViewVec);
    vec3 N = normalize(NormalVec);
    vec3 R = reflect(-V, N);

    vec4 base = texture2D(DecalMap, gl_TexCoord

    float diffuse  = clamp(dot(L, N), 0.0, 1.0)
    float specular = pow(clamp(dot(L, R), 0.0,
    float atten    = clamp(1.0 - 0.0125 * sqrt(
    vec4 ambient   = 0.175 * diffuse * base * a

    fetch_depth();

    vec4 disco = textureCube(DiscoLight, CubeTe

    vec4  d = (LightType > 0)  ? disco : vec4(1
    float k = (depth > shadow) ? blur : 1.0;
    float s = (depth > shadow) ? 0.0  : 0.25 *
    float a = (LightType > 0)  ? 1.5  : 1.0;

    gl_FragColor = d * k * (diffuse * base + s)
}
```



Properties of Program 13

Type: Program
Is successfully linked: Yes
Is marked for deletion: No
Used in last frame: Yes

Program link log: Empty

Program 13

Active Uniforms:

Name	Type	Value
CameraPos	GL_FLOAT_VEC3	{0, 0, 20}
DecalMap	GL_SAMPLER_2D	Texture data
DiscoLight	GL_SAMPLER_CUBE	Texture data
InvViewMatrix	GL_FLOAT_MAT4	{1, -0, 0, 0} {0, 1, 0, 0} {-0, 0, 1, 0} {0, 0, 20, 1}
LightPos	GL_FLOAT_VEC3	{-7.4766479, 2.1167362, -11.992739}
LightType	GL_INT	0
RotationMatrix	GL_FLOAT_MAT3	{-0.88725728, 0.082310766, 0.45387161} {-0.46023011, -0.22415209, -0.85903668} {0.031028308, -0.97107196, 0.23676246}
ShadowMap0	GL_SAMPLER_2D	Texture data
ShadowMap1	GL_SAMPLER_2D	Texture data
ShadowMap2	GL_SAMPLER_2D	Texture data
ShadowMap3	GL_SAMPLER_2D	Texture data

Active built-in uniforms:

Name	Type
gl_DepthRange.near	GL_FLOAT
gl_ModelViewMatrix	GL_FLOAT_MAT4
gl_ModelViewProjectionMatrixTranspose	GL_FLOAT_MAT4
gl_NormalMatrix	GL_FLOAT_MAT3

Profiling Graphic applications

Remove redundant OpenGL calls

Find and remove:

- Redundant state changes
- Repeatedly turning on and off the same OpenGL mechanisms
- Redundant OpenGL API calls
- Immediate mode rendering
- etc.

➡ Do not invest time in removing calls that do not seem to have significant impact on render performance!

Steps for optimizing render performance

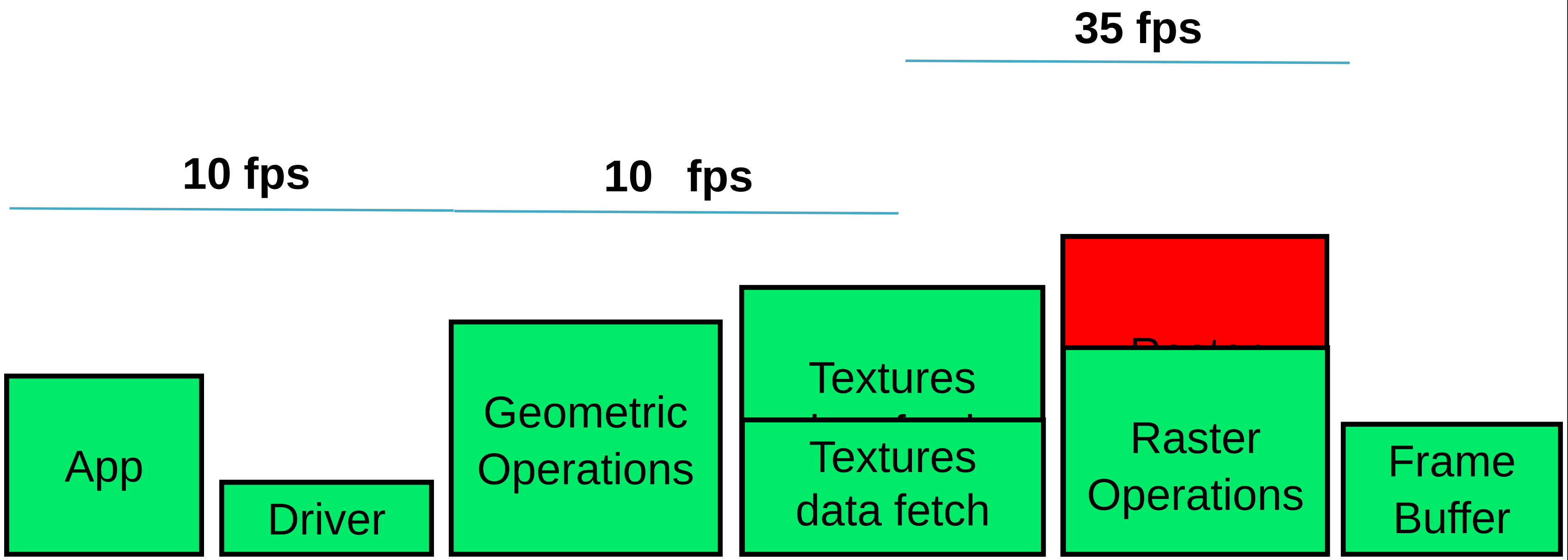
1. Remove redundant calls and state changes
2. Identify a performance bottleneck
3. Optimize the pipeline stage that causes the bottleneck

Repeat 2 and 3 until reaching the desired performance level or until the performance cannot be improved anymore

➢ If you cannot improve performance anymore, you can add workload to pipeline stages that are not fully utilized (to improve visual effects / visual fidelity)

Pipeline speed

- The pipeline runs as fast as its slowest stage!

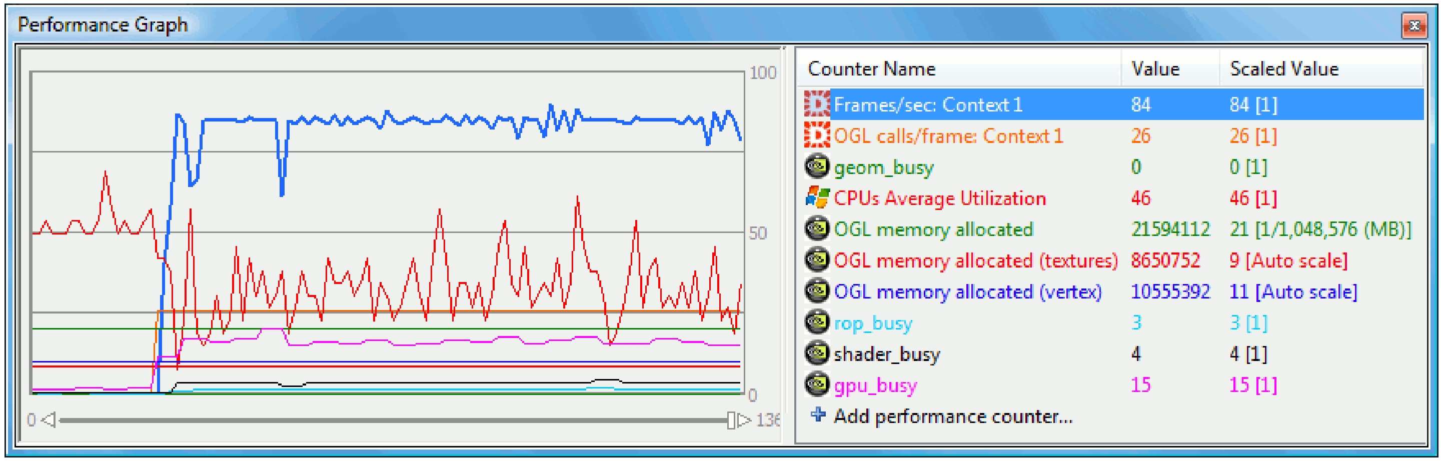


Profiling Demo

Performance graph

Displays performance counters graphs of

- Windows and Linux OS
- NVIDIA's GPUs and drivers through NVPerfKit
- gDEBugger OpenGL Server



Counter Name	Value	Scaled Value
Frames/sec: Context 1	84	84 [1]
OGL calls/frame: Context 1	26	26 [1]
geom_busy	0	0 [1]
CPUs Average Utilization	46	46 [1]
OGL memory allocated	21594112	21 [1/1,048,576 (MB)]
OGL memory allocated (textures)	8650752	9 [Auto scale]
OGL memory allocated (vertex)	10555392	11 [Auto scale]
rop_busy	3	3 [1]
shader_busy	4	4 [1]
gpu_busy	15	15 [1]

nvision 08
THE WORLD OF VISUAL COMPUTING

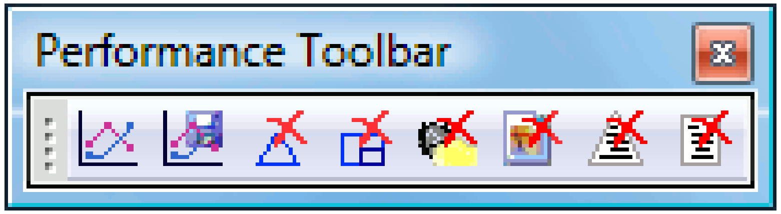
NVIDIA.

Available performance counters

gDEBugger	NVIDIA	OS (windows & Linux)
Frames / sec	% GPU Idle	CPU utilization
# OGL function calls	% Driver Waiting	CPU user mode utilization
# Texture objects	% Vertex Shader Busy	CPU privilege mode utilization
# Loaded texels	% Pixel Shader Busy	Available physical memory
* All counters are "per render context" and available on all graphic hardware's	% ROP Busy	Virtual memory usage
	% Shader Waits for Texture	Virtual memory pages / sec
	% Shader Waits for ROP	Drivers virtual memory usage
	Video Memory Usage	And more…
	AGP / PCI-E Memory Usage	
	# Vertex Count	
	# Frame Batch	
	# Frame Vertex	
	# Frame Primitive	
	And more…	

Performance Analysis Toolbar

- Turn off the graphic pipeline stages one after the other
- If the performance improves when turning off a certain stage, you have found a graphic pipeline bottleneck



These commands include

- Enter profiling mode
- Eliminate draw commands
- Eliminate raster operations
- Eliminate fixed pipeline lights
- Eliminate texture data fetch operations
- Eliminate geometry shader operations
- Eliminate fragment shader operations

gDEBugger offers many other features

- Launches any OpenGL or OpenGL ES application for debug or profile session. Instrumentation or recompilation are not needed!
- Adds breakpoints for any OpenGL, OpenGL ES or extensions entry point
- Views a list of active and deleted OpenGL render contexts
- Display debugged process events such as: thread created, dll loaded and unloaded, breakpoint hit, etc.
- Forces OpenGL to render directly into the front buffer and controls the rendering speed
- Forces the OpenGL Polygon Raster mode to see the rendered geometry and test your applications culling
- gDEBugger ES is being used as an “out of the box” OpenGL ES implementation for the embedded systems on a Windows PC machine
- Displays implementation-specific OpenGL run-time information such as pixel formats and available extensions
- Saves textures as image files to disk
- Views OpenGL state variables values in a watch view and comparison viewer
- Supports applications that render using multiple threads and multiple render contexts
- Saves performance counters data into a file (.csv). This enables performing performance and regression tests using different hardware and driver configurations
- Output an OpenGL calls log into a formatted HTML file, containing texture imaged, vertex and fragment shaders source code
- Supports GL_GREMEDY_string_marker extension that allows adding string markers into the reported and recorded calls log, making the calls log much clearer
- Support OpenGL 2.1 standard and many additional extensions (OpenGL 3.0 will be supported soon)
- And more…

Questions?

Avi Shapira
www.gremedy.com
info { a t } gremedy [.] com