

# gDEBugger

## 그래픽 응용프로그램 성능 최적화 가이드

GDEBUGGER는 OPENGL 기반 응용프로그램의 성능을 분석, 최적화하고 소프트웨어의 신뢰성을 더욱 높여 줍니다.

### 이 가이드를 통해 :

**정밀 분석을 통한 프로파일로 개발 시간을 단축 하십시오.**

그래픽 파이프라인 동작 상태를 정확히 파악 하여 병목현상 가능성을 미리 예측할 수 있습니다.

**제품의 가치를 높이십시오. 제품의 성능을 높이십시오.**

gDEBugger가 제공하는 정밀한 정보를 통해 응용프로그램의 병목 지점을 찾고 성능을 최적화 할 수 있습니다.

성능 저하 요인과 불필요한 중복 OpenGL 호출을 제거 합니다.

**업그레이드 된 OpenGL 개발자가 되십시오.**

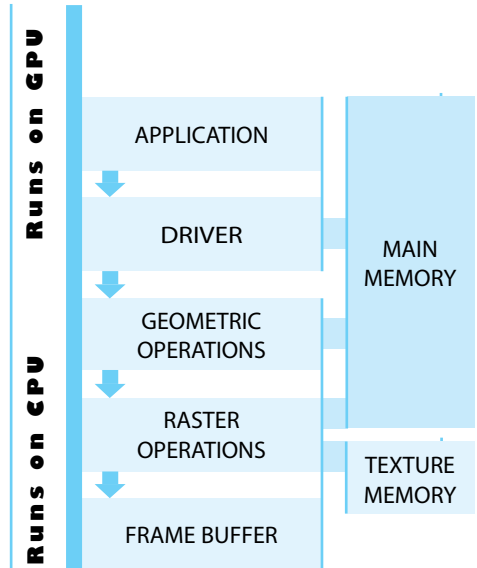
개발자가 변경한 내용이 OpenGL 사용상태, 비주얼, 성능, 정밀도에 어떤 영향을 주는지 바로 확인 할 수 있습니다.

# 그래픽 파이프라인, 사용하는 하드웨어를 먼저 파악하십시오

**1 그래픽 파이프라인의 병목현상** 그래픽 시스템은 일련의 연산을 파이프라인으로 수행하여 이미지를 생성하게 됩니다. 전체 파이프라인의 성능은 파이프라인에서 가장 낮은 성능의 스테이지 보다 빨라질 수 없습니다. 일반적으로 이 가장 낮은 성능의 스테이지를 파이프라인의 병목지점이라 할 수 있습니다. 단순하고 간단한 그래픽 요소(예를 들면 삼각형)의 렌더링의 경우에는 하나의 그래픽 파이프라인 병목지점이 있겠지만 다양한 종류의 그래픽 요소를 포함하는 프레임의 렌더링의 경우 병목지점이 다양하게 변할 수 있습니다. 예를 들면, 처음에 많은 선을 그리고 나서 조명효과와 셰이딩이 효과가 있는 삼각형들을 그린다면 병목지점이 변하게 됩니다.

## 2 The OpenGL Pipeline

OpenGL 파이프라인은 그래픽 시스템 파이프라인의 추상화 된 개념입니다. 아래 그림은 OpenGL 파이프라인의 개략적인 그림입니다.



OPENGL GRAPHIC PIPELINE (ROUGH SKETCH)

일부 파이프라인 스테이지는 CPU에서 실행되고 나머지는 GPU상에서 실행됩니다. GPU에서 실행되는 거의 모든 연산은 병렬로 처리됩니다.

**Application** - 그래픽 응용은 CPU에서 실행되며, OpenGL API 함수를 호출합니다.

**Driver** - 그래픽 시스템 드라이버는 CPU에서 실행되며 OpenGL API 호출을 CPU 또는 GPU에서 수행되는 명령으로 변환합니다.

**Geometric operations** - 기하 연산은 Vertex 속성과 위치에 관한 각종 계산을 수행하고 2D 공간으로 이미지를 렌더링하는 역할을 합니다. 여기에는 모델좌표계를 스크린 좌표계로 변환하는 행렬 곱셈, Vertex 조명모델 계산, Vertex Shader 등을 포함합니다.

**Raster operations** - fragment 또는 스크린 픽셀에 대한 연산으로, 여기에는 색상의 읽고 쓰기, Z-buffer나 Stencil 버퍼에 대한 읽고 쓰기, 알파 블렌딩, 텍스처 기능, fragment shader 의 사용 등이 포함됩니다.

**Frame buffer** - 렌더링된 2D 이미지를 저장하는 메모리

# GDEBUGGER를 사용하여 그래픽 응용프로그램을 최적화 하십시오

## 1 단계

### "정확한" OpenGL의 사용

응용프로그램의 최적화에 앞서 먼저 해야할 일은 OpenGL 사용상의 오류가 없도록 하는 일이다. 정확한 OpenGL 사용법에 따랐는지, 정확히 수행하기 원하는 OpenGL API 가 호출되어 수행되는지 점검합니다.

#### a. OpenGL 에러의 제거:

OpenGL 에러가 발생한 경우 대부분 OpenGL은 에러가 발생한 API 호출을 무시하고 진행합니다. 이 경우 응용프로그램이 사용자가 원하는 기능중 에러가 발생한 기능을 수행하지 않았음을 의미합니다. 따라서 최적화를 시작하기 전에 이러한 OpenGL 에러를 제거 하는 것이 중요합니다.

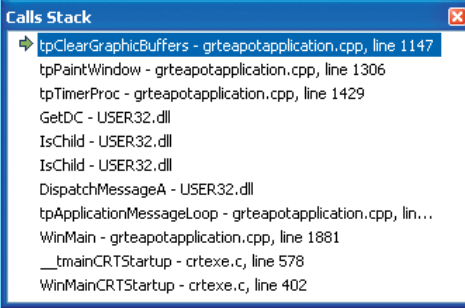
gDEBgger는 이를 위한 두가지 메커니즘을 제공 합니다.

**1. BREAK ON OPENGL ERRORS:** OpenGL 에러가 발생한 곳마다 응용프로그램의 수행을 정지 시킵니다.

**2. BREAK ON DETECTED ERRORS:** OpenGL 드라이버가 정상적으로 실행(성능측면에서도)하는가에 대한 테스트를 gDEBgger가 수행하도록 합니다. 이 기능은 gDEBgger가 감지한 에러가 발생했을 때마다 실행을 정지시키도록 하는 메커니즘을 작동 시킵니다.

Breakpoints	Tools	Help
Break on OpenGL Error		Ctrl+E
<input checked="" type="checkbox"/> Break on Detected Error		
<input checked="" type="checkbox"/> Break on NVIDIA GLExpert Reports		
NVIDIA GLExpert Settings		
<input checked="" type="checkbox"/> Enable Breakpoints		Ctrl+I
Add / Remove Breakpoints...		Ctrl+B

응용프로그램의 수행이 중단된 후에는 Call Stack View 를 사용하여 Call Stack과 에러를 유발한 소스코드를 검토할 수 있습니다.



### b. 중복된 OpenGL 호출의 제거:

대다수의 OpenGL 기반 응용프로그램은 OpenGL API를 중복 호출하는 경우가 많습니다. 이러한 중복 호출은 렌더링 성능에 큰 영향을 미치게 됩니다. OpenGL 호출에 대한 log 를 검토하여 렌더링 속도에 영향을 주는 것으로 보이는 중복 호출을 제거하는 것은 중요합니다. 대표적인 중복호출은 상태 변경을 중복하여 하는 경우, OpenGL의 같은 기능을 반복적으로 On/Off를 하는 경우, immediate mode 렌더링을 사용하는 경우 등이 있습니다.

gDEBbugger는 이를 위한 몇 가지 메커니즘을 제공 합니다.

**1. OPENGL FUNCTION CALLS HISTORY VIEW:** OpenGL, OpenGL ES, extensions, WGL과 EGL의 각 Render Context 내에서 수행된 함수 호출에 대한 log 를 보여 줍니다.

**2. OPENGL FUNCTION CALLS STATISTICS VIEW:** 직전 프레임의 렌더링에서 각각의 OpenGL 함수가 얼마나 자주 호출되었는지 그 횟수와 전체 함수 호출에서 차지하는 비율을 볼수 있도록 해 줍니다.

OpenGL Function Na...	%	# of Calls in Previous Frame
glNormal3fv	33.29	166395
glTexCoord2f	33.29	166395
glVertex3fv	33.29	166395
glTexEnvi	0.04	195
glMaterialfv	0.01	52
glMatrixMode	0.01	35
glMultMatrixd	0.01	26
glPopMatrix	0.00	20
glPushMatrix	0.00	20
glLoadIdentity	0.00	17
glBegin - GL_TRIANGLES	0.00	13
glBindTexture - GL_T	0.00	13

## 2 단계

### 병목 현상의 제거

앞서 "그래픽 파이프라인의 병목 현상"에서 언급한 바와 같이 그래픽 시스템은 일련의 그래픽 연산을 파이프라인 상에서 수행함으로써 이미지를 생성합니다. 이때 전체 파이프라인은 "병목지점"이라 불리는 가장 성능이 느린 스테이지보다 빨리 수행될 수는 없습니다.

다음 단계를 통해 성능 병목 현상을 제거할 수 있습니다.

1. 병목 지점의 확인 (IDENTIFYING THE BOTTLENECK) - 현재 그래픽 파이프라인에서 병목지점의 위치를 찾아 냅니다.
  2. 최적화 (OPTIMIZING) - 현재 병목 지점이 더이상 병목 지점이 되지 않거나 원하는 성능을 얻을 때까지 해당 스테이지의 작업 부하를 줄여 줍니다.
- \* 전체 성능이 원하는 수준에 다달을 때까지 1, 2 단계를 반복하여 수행합니다.

성능의 최적화가 완료되거나 더이상 병목지점의 작업부하를 줄일 수 없는 단계에서는, 파이프라인의 완전히 사용되고 있지 않은 다른 스테이지에 추가로 작업 부하를 주어도 전체 렌더링 성능에 영향을 주지 않게 됩니다. (예를 들면 좀더 정밀한 텍스처나 좀더 복잡한 Vertex Shader 기능, 등을 추가적으로 사용할 수 있게 됩니다)

The screenshot displays the gDBDebugger - GRTeaPot interface. The top menu includes File, Edit, View, Debug, Breakpoints, Tools, and Help. The main window is divided into several panels:

- OpenGL Function Calls History:** Lists recent OpenGL calls for Context 1, including `glMaterial`, `glPolygonMode`, `glUseProgramObjectARB`, `glUniform1fARB`, `glStringMarkerGREMEDII`, and `glBindTexture`.
- OpenGL State Variables:** Shows current state variables such as `GL_VIEWPORT`, `GL_PROJECTION_MATRIX`, and `GL_MODELVIEW_MATRIX`.
- Performance Graph:** A line graph showing performance metrics over time. The Y-axis ranges from 0 to 1.00. The X-axis shows a time range from 0 to 5.5. A sharp dip is visible in the graph.
- Counter Name Table:** A table with columns for Counter Name, Value, and Scaled Value.

Counter Name	Value	Scaled Value
Frames/sec: Context 1	64	64 [1]
vertex_shader_busy	0	0 [1]
gpu_idle	89	89 [1]
ogl_vdmem_bytes	329152	52 [Auto scale]
CPU 0 Utilization		

Additional panels on the right show a list of calls and a debug string log with entries like "Process run started", "Thread created: 3836", and "Debug strings: Connection with the".

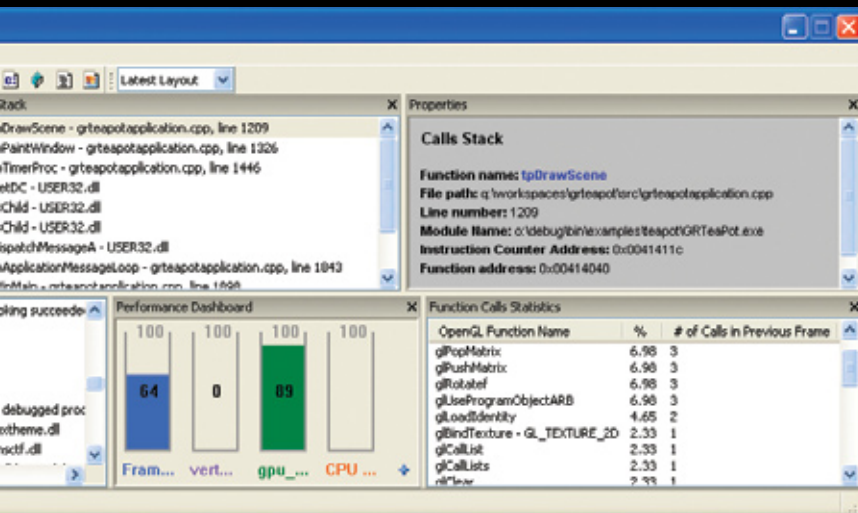
gDEBbugger는 응용프로그램의 그래픽 파이프라인의 병목지점을 확인 할 수 있도록 도와주는 뷰와 Toolabar를 제공 합니다.

### a. 성능 그래프 뷰

#### (Performance Graph view):

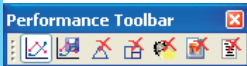
gDEBbugger의 성능 그래프 뷰는 FPS, 프레임당 OpenGL 함수 호출 횟수, vertex와 fragment 프로세서의 성능, 비디오 메모리의 사용량, GPU Idle 시간, 드라이버 Idle 시간, 텍스처 데이터의 사용 상태와 현재 그래픽 파이프 라인에 적재된 텍셀의 크기 등 그래픽 시스템의 성능 정보를 실시간에 표시해 줍니다. 이를 위해서 사용자의 소스코드를 수정하거나 재컴파일할 필요가 전혀 없습니다. 또한 성능 지표를 성능 그래프 뷰 내에 표시할 수 있습니다.

gDEBbugger는 nVIDIA의 NVPerfKit을 통한 성능 지표, ATI의 Performance metrics 그리고 3DLabs의 Hardware Profiling Technology를 통한 성능 지표를 지원합니다.



## b. 성능 분석 툴바 (Performance Analysis Toolbar):

Performance Analysis Toolbar는 응용프로그램의 성능 병목지점을 지정 할 수 있는 명령들을 제공합니다. 이러한 명령은 원하는 그래픽 파이프라인 스테이지를 사용하지 않도록 설정할 수 있습니다. 만약 성능 지표가 상승 한다면, 바로 설정된 스테이지가 병목지점이 되는 것입니다.



이 명령들에는 다음 기능을 포함합니다.

### 모든 그래픽 명령 비활성화(Eliminate all draw commands)

모든 OpenGL Draw 명령을 비활성화 함으로써 CPU/BUS 성능이 병목지점인지를 확인할 수 있습니다. 즉 Draw 명령을 모두 제거 하고 남아 있는 OpenGL 명령은 vertex 또는 텍스처 데이터를 OpenGL에 전송하는 함수만 남게 됩니다. 따라서 CPU/BUS 부하는 그대로 남아 있지만 남은 명령이 삼각형, 선 등의 기본 입력에 의해서만 GPU가 활동하게 되어 거의 모든 GPU 부하는 없어지게 됩니다.

### 모든 래스터 명령 비활성화 (Eliminate all raster op's)

뷰 포트를 1x1 픽셀 크기로 작동 시켜 봄으로써 래스터 명령이 병목지점인지 확인할 수 있습니다. 래스터 명령은 fragment 또는 픽셀 단위로 수행됩니다. 따라서 뷰포트의 크기는 1x1 픽셀로 설정함으로써 거의 대부분의 래스터 명령을 비활성화 할 수 있습니다.



### 모든 고정 광원 관련 명령 비활성화

#### (Eliminate all fixed light op's) -

"fixed pipeline lights" 관련 계산이 병목지점인지 확인하기 위해 OpenGL 의 모든 고정광원을 꺼 봅니다. 주목할 점은 고정 광원을 모두 사용하지 않는다 하더라도 fragment shader에는 영향을 주지 않는다는 점입니다.



#### 텍스처 가져오기 명령 비활성화(Eliminate all texture fetch operations) -

"texture memory"의 성능이 병목지점인지 확인하기 위해 사용자가 응용프로그램에서 정의한 텍스처 대신에 크기가 2x2인 빈 텍스처를 대신 사용합니다. 이렇게 작은 빈 텍스처를 사용하면 텍스처를 텍스처 메모리로 가져오는데 사용되는 부하는 거의 완벽하게 제거됩니다.



#### Fragment Shader 명령 비활성화

#### (Eliminate fragment shader op's)

"fragment shader"가 병목지점인지 확인하기 위해 사용자가 응용프로그램에서 정의한 fragment shader 명령 대신 매우 간단한 빈 fragment shader를 사용합니다.

# 그외 GDEBUBGGER 기능들

다음은 앞에서 언급하지 못한 gDEBubger의 기능들 입니다.

어떤 OpenGL 혹은 OpenGL ES 응용 프로그램이라도 debug 또는 profile을 위한 실행을 할 수 있습니다.

어떤 OpenGL, OpenGL ES 또는 extension에도 breakpoint를 추가 할 수 있습니다.

텍스처와 관련 파라미터 그리고 관련 데이터를 이미지 형태로 볼 수 있습니다.

텍스처를 디스크에 이미지 파일로 저장할 수 있습니다.

Shader의 파라미터, 활성화된 uniform 변수 값, Shader 소스 코드를 볼 수 있습니다.

Shader 프로그램을 "실행중"에 편집, 저장, 컴파일, 링크, 검증할 수 있습니다.

현재 활성화 되어 있거나 삭제된 OpenGL render context의 목록을 볼 수 있습니다.

OpenGL 에러를 감지하여 자동으로 응용프로그램의 수행을 중지 하도록 할 수 있습니다.

응용프로그램의 Call Stack 쓰레드와 소스코드를 볼 수 있습니다.

OpenGL 상태 변수의 값을 watch view에서 볼 수 있습니다.

OpenGL 상태 Snapshot을 파일로 저장할 수 있습니다.

저장된 OpenGL 상태 Snapshot을 파일과 현재의 상태를 자동으로 비교할 수 있습니다.

렌더링을 Front buffer에서만 수행하도록 강제하고 렌더링 속도를 조절 할 수 있습니다.

OpenGL의 Polygon Raster mode (shade, wireframe 등) 을 강제하여 현재 렌더링되는 기하 오브젝트를 확인 할 수 있습니다.

---

성능지표 데이터를 파일로 저장 할 수 있습니다.

이 기능을 통해서 사용자 응용프로그램을 다양한 하드웨어와 드라이버 설정 환경에서 성능과 비교분석 테스트를 수행할 수 있습니다.

---

다수의 쓰레드에서 다수의 렌더링 context를 사용하는 응용프로그램을 지원합니다.

---

OpenGL ES 응용프로그램의 debug와 profile을 지원합니다.

---

gDEBugger ES는 "out of the box" 형태, 즉 PC 상에서 수행되는 OpenGL ES 애플레이터로 사용될 수 있습니다.

---

Display의 구현에 따라 변경되는 픽셀 포맷이나 가용한 extension과 같은 run-time 정보를 볼 수 있습니다.

---

NVIDIA의 GLExpert 드라이버 리포트 정보를 표시해 주고 자동으로 응용프로그램을 중단 할 수 있습니다.

---

그 밖에 많은 기능 들이 있습니다...

---

gDEBugger는 모든 상용 그래픽 하드웨어를 지원합니다.

NVIDIA GPUs performance counters via NVPerfKit, NVIDIA

GLExpert driver reports, ATI Performance Metrics, 또한 가장 최신 버전의OpenGL과 많은 extensions을 지원합니다.

REALIZE  
YOUR  
OpenGL  
POTENTIAL

[www.gremedy.com](http://www.gremedy.com)

[info@gremedy.com](mailto:info@gremedy.com)

한국총판 (주)휴원

[gdebugger@hu1.com](mailto:gdebugger@hu1.com)

Tel (053)325-4956 Fax (053)325-4951

*graphic***REMEDY**

SOFTWARE SOLUTIONS FOR  
THE 3D GRAPHICS INDUSTRY