Scaleform GFx

AMP User Guide

This document describes the Scaleform Analyzer for Memory and Performance (AMP) and how to use the system for profiling applications.

Author:	Alex Mantzaris
Version:	1.05
Last Edited:	June 27, 2011

Scaleform®

Copyright Notice

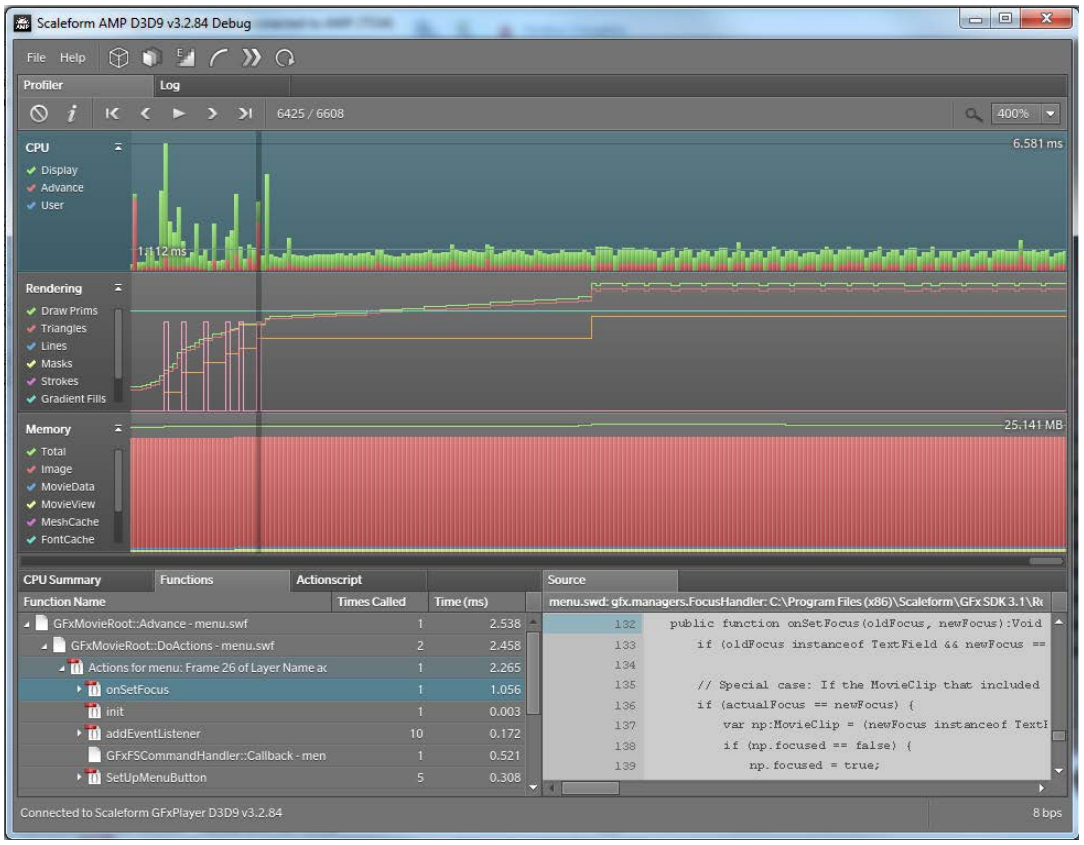
How to Contact Autodesk Scaleform:

Document	AMP User Guide
Address	Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
Website	www.scaleform.com
Email	info@scaleform.com
Direct	(301) 446-3200
Fax	(301) 446-3199

Table of Contents

1 Introduction

AMP™ is the Scaleform® remote profiling system, capable of monitoring CPU usage, graphics rendering, and memory consumption of a GFx application. Real-time graphs provide a method of quickly identifying memory and performance bottlenecks, and per-frame statistics allow an in-depth analysis and determination of the causes. AMP can also help find script problem areas, using its per-frame ActionScript function and source code timings.



Function Name	Times Called	Time (ms)
GFxMovieRoot::Advance - menu.swf	1	2.538
GFxMovieRoot::DoActions - menu.swf	2	2.458
Actions for menu: Frame 26 of Layer Name ac	1	2.265
onSetFocus	1	1.056
init	1	0.003
addEventListener	10	0.172
GFxFSCommandHandler::Callback - men	1	0.521
SetUpMenuButton	5	0.308

menu.swd: gfx.managers.FocusHandler: C:\Program Files (x86)\Scaleform\GFx SDK 3.1\R

```
public function onSetFocus(oldFocus, newFocus):Void
    if (oldFocus instanceof TextField && newFocus ==

    // Special case: If the MovieClip that included
    if (actualFocus == newFocus) {
        var np:MovieClip = (newFocus instanceof TextF
        if (np.focused == false) {
            np.focused = true;
```

Figure 1: Scaleform AMP

Every GFx application can act as an AMP server, which means that it listens for incoming connection requests by the profiler, gathers information every frame, and sends it to the profiler over the network. The profiler acts as an AMP client, which means that it connects to a specific AMP server, processes the information sent from the server every frame, and sends requests to control the profiled application.

AMP is available with GFx 3.2 and higher versions.

1.1 AMP Server

A GFx application being profiled (the AMP server) sends a certain amount of information over the network to the profiler. Each ActionScript function and some important C++ GFx functions are timed, and their calling functions are recorded. If per-line source code profiling is enabled, each ActionScript bytecode execution is timed. In addition, rendering statistics, detailed memory heap and image information, flash file information, and ActionScript markers are recorded. Note that this extra work per frame can slow down the application and increase its memory consumption, especially if per-line timings are enabled.

In order to keep AMP from interfering with the application being profiled, all AMP server information is kept on a separate memory heap. If the size of this heap exceeds a pre-set limit, the application is paused until the pending frame information is sent over the network and subsequently deleted. GFx Shipping builds do not act as AMP servers, and AMP support may be disabled for any build within an application, if so desired.

Performance statistics on the server are maintained per MovieView. This allows users to easily identify problem spots in the case where there are several views associated with a particular SWF file. It also allows AMP to produce meaningful call graphs for ActionScript execution of multiple MovieViews on separate threads.

In addition to reporting memory and statistics to the client, the server can be controlled by the AMP client in several ways to assist profiling. For example, the profiler can send requests for wireframe mode rendering, for pausing, fast-forwarding, or restarting the flash movie, for toggling the anti-aliasing and stroke modes, for changing the localization fonts, and for changing the vector graphics' curve tolerance. Furthermore, the client can toggle special AMP rendering modes to show display performance bottlenecks, such as mask, filter, and pixel overdraw highlighting.

While all the above functionality has been implemented in GFxPlayer, some of it requires application-specific implementation, and may be performed as part of GFx integration. To graciously handle the case where some of these capabilities have not been implemented in a GFx application, the AMP server reports the set of supported application control functionality to the client on connection. It also reports its current state (wireframe mode, for example) to the client for visual feedback.

Finally, the AMP server has the ability to send Flash debug information (SWD files) over the network to the profiler, so that source code and per-line timings can be displayed if the client does not have local access to that information. In order for this transfer to occur, the SWD file should be in the same location as the SWF file being profiled.

1.2 AMP Client

The remote profiler (AMP client) is a stand-alone application that connects to an AMP server, receives profile information every frame, and displays this information in a user-friendly way to enable effective analysis of performance and memory bottlenecks. The profiler is itself a GFx application, implemented using Scaleform CLIK.

The AMP client has several features that allow developers to view multiple aspects of their Flash assets and identify problem areas:

- **CPU graph**: Shows how much time was spent per game frame in ActionScript (Advance), GFx rendering (Display), and in the Direct Access API (User). This graph is especially useful to identify spikes in CPU usage due to GFx. When a frame with a spike is selected, the cause can be identified by examining the detailed statistics for that frame. When the CPU graph is selected, the following tabs become visible in the bottom panel of the application:
    - **CPU Summary:** Breaks down the Advance, Display, and User times into sub-categories to provide a more in-depth overview of where the time was spent for the selected frame(s). If multiple frames are selected, then the values will be the average over those frames.
    - **Functions:** Shows a tree control with the amount of time spent inside each function, inclusive of the functions that were called from it. When multiple frames are selected the time and number of calls are an average per game frame. ActionScript and C++ functions are mixed in the same call graph because the code path may move between the ActionScript VM and the rest of the application. Expanding each function reveals the functions that were called from that function. Note that the time displayed in each item is the sum of time spent in a function when called from a given caller for that frame. This gives intuitive results in most cases. However, in the case of recursion, the times reported do not add up to the time spent in the calling function, since the timer corresponding to the same caller-callee pair is incremented multiple times in the same call stack.
    - **Actionscript:** This tab shows similar information as the Functions tab, but the information is not hierarchical, and C++ functions are excluded. It is useful to quickly identify the most expensive script functions.
    - **Source:** Shows the amount of time spent in each line of the code for the selected function in the Functions tab. In order to see per-line timings, which are off by default, the corresponding button needs to be toggled on the toolbar. Instruction profiling is time-consuming, and can slow down the GFx application being profiled, so it is best done after a problem area has already been identified. The timings shown next to each line do not add up to the amount of time reported in the function call graph in the case where the function was called by more than one other function. This is because the per-line timings are the total time spent in the function regardless of from where it was

called. Also, line timings do not include the time spent inside function calls, whereas the call graph timings include time spent in called functions. In order to display source code and per-line timings, any flash debug files *not found on the AMP server (SWF location)* need to be in the working directory of the AMP client executable.

Rendering graph: The GFx Best Practices Guide document identifies several renderingrelated areas that may cause performance hits, such as an excessive number of draw primitives, triangles, and lines, or the presence of masks, filters, strokes, or gradient fills. Counters for each of these areas are shown in the rendering graph, so that developers can gain more insight to what is happening inside GFx. When the Rendering graph is selected, the **Render Summary** tab becomes visible at the bottom panel of the application, which shows the same information as the graph, but in numeric form. This tab also includes some additional rendering statistics, such as Mesh Thrashing and current number of Font Textures.

- **Memory graph:** The memory consumption of GFx is tracked in the memory graph, allowing the developer to quickly monitor memory usage by category such as rendering, images, sound, fonts, movie data, etc, and includes all loaded files. When the Memory graph is selected, the following tabs become visible at the bottom panel of the application:
    - **Memory Summary:** Allows the user to determine how memory is allocated within GFx, by purpose. It is a more in-depth view of the graph information, and is available only when detailed profiling is enabled (see the profiling toolbar, below).
    - **Heaps:** Shows how much memory is allocated to each memory heap. This view is particularly useful for determining the memory consumption of each SWF file.
    - **Images:** Shows the memory consumed by each image, along with its dimensions.

- **File Menu:** This menu includes the following choices:
    - **Connection:** Brings up the connection dialog, for connection to a different server.
    - **Load Profile Frames:** Loads a previously-saved profile run for reexamination.
    - **Save Profile Frames:** Saves the current profile run to disk. The run can be reloaded by selecting the “Load Profile Frames” option, above.
    - **Exit:** Closes the application**.**

- **Help Menu:** This menu currently has only one option, “About,” which brings up a screen with the AMP version and credits.
- **Application Control Toolbar:** This UI element contains a series of buttons for controlling the application being profiled.
    - **Wireframe:** Sends a request to the profiled application to render content in wireframe mode. When the application is already in wireframe mode, clicking the button takes it out of that mode.
    - **Performance Highlighting:** Sends a request to the profiled application to enter a rendering mode that highlights areas of pixel overdraw, masks, and filters. The AMP

server renders areas of pixel overdraw with green (color intensity is proportional to number of overdraws), masks with red, and filters with blue. When the application is already in performance highlighting mode, clicking the button takes it out of that mode.

  - **Anti-aliasing:** Sends a request to the profiled application to cycle through anti-aliasing modes. The three modes available are Scaleform's edge anti-aliasing technique (EdgeAA), Hardware- full-scene anti-aliasing (HW FSAA), or no anti-aliasing (None).
  - **Stroke:** Sends a request to the profile application to cycle through stroke modes. The three modes available are Normal, Correct, and Hairline.
  - **Fast-forward:** Sends a request to the profiled application to play the flash content in fast-forward. This causes the SWFs to advance as fast as the rendering frames and disregard its document FPS setting. When the application is already in fast-forward mode, clicking the button takes it out of that mode.
  - **Restart:** Sends a request to the profiled application to restart the flash content.

- **Profiling Toolbar:** This element consists of a series of buttons that are used for profiling.
  - **Clear:** Discards the current profile run.
  - **Detailed Profiling (Source line timing and detailed memory breakdown):** When this option is selected, the time spent in each line of ActionScript code is displayed in the Code panel for the selected frame. A debug file (SWD) is required in order to show source line timings. In addition, the option gathers more detailed memory statistics per frame, which is displayed in the memory summary and heaps tabs. This option may considerably slow down the application being profiled.
  - **First Frame:** Selects the first frame in the current profile run.
  - **Previous Frame:** Selects the previous frame from the currently-selected one.
  - **Pause:** Sends a request to the profiled application to stop sending memory and performance data. This operation does not pause the application itself.
  - **Next Frame:** Selects the next frame from the currently-selected one.
  - **Last Frame:** Selects and locks onto the last frame in the current profile run. As information for new frames is received, the selection is changed to always be the last received frame.

    Zoom: This drop-down control on the far-right alters the horizontal scale of the graphs. Zooming out allows identification of problem areas of the run, and zooming in allows more precise examination of those areas. The mouse wheel can also be used for zooming in and out, and provides a higher resolution not available via the drop-down.

- **Log:** Clicking on the Log tab displays the GFx log that has been produced by the profiled application over the course of the connection. Clicking back on the Profile Tab brings back the display with the memory and performance graphs and detail tabs.

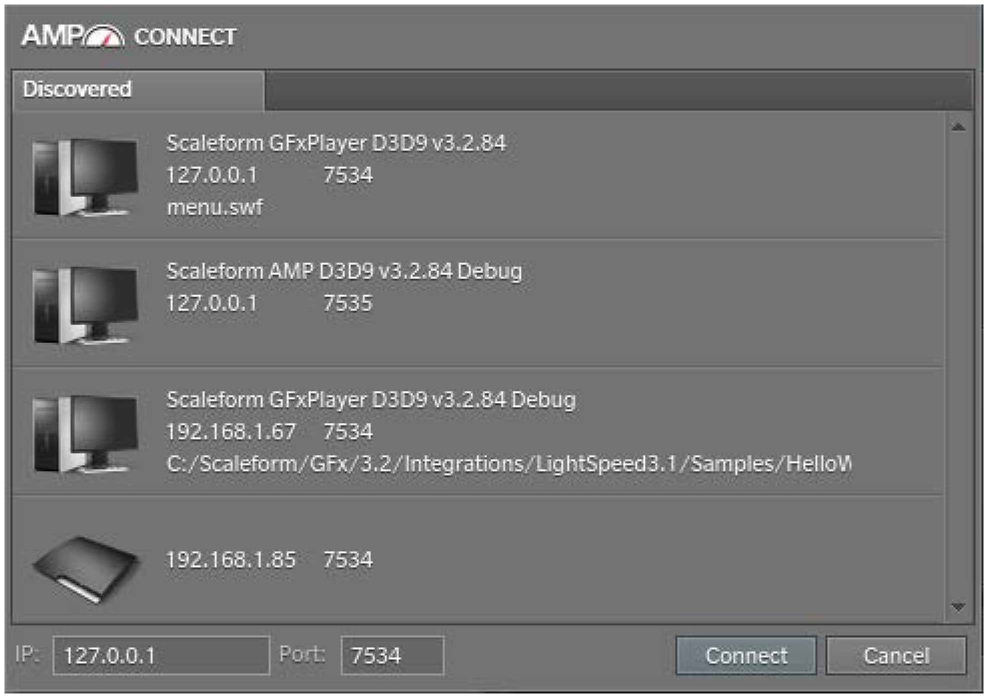
- **Status:** At the bottom of the profiler is a Status Bar that shows the current connection and bit rate of the received information over the network.

2 Getting Started with AMP

To start using the AMP profiler, run the *GFxAmpClient.exe* executable that is located in Bin/AMP under the main Scaleform installation directory. The application starts, and the connection dialog appears.

2.1 Making a Connection

A connection dialog appears automatically when the client starts up. It prompts the user to either select from a list of discovered AMP servers, or manually type the IP address and port to connect. Select the second approach if the AMP server of interest is not yet running, as would be the case if there is a need to profile an application as it is starting up.



```
Scaleform GFxPlayer D3D9 v3.2.84
127.0.0.1        7534
menu.swf

Scaleform AMP D3D9 v3.2.84 Debug
127.0.0.1        7535

Scaleform GFxPlayer D3D9 v3.2.84 Debug
192.168.1.67    7534
C:/Scaleform/GFx/3.2/Integrations/LightSpeed3.1/Samples/HelloW

192.168.1.85    7534

IP: 127.0.0.1    Port: 7534
```

Figure 2 : AMP connection dialog

The Connection item under the File menu can be used to bring up the connection dialog at any time. Each AMP server in the discovered list has a distinct icon identifying its platform (PC, Xbox360, PS3, etc). Next to the icon is the title of the application, the IP address and port where the server is listening, and the Flash content currently being played. If some of this information is missing, it is because the corresponding AMP server is not sending it. A description of how to add full AMP support to any GFx application, including broadcasting the application and content information, is found later in this document. A connection is established either by double-clicking a discovered server, or by selecting a server and clicking the “Connect” button.

Please note that connection to a Wii development kit (NDEV) does not use the above server discovery method. Instead, the Wii is physically connected to a PC using the USB interface (COM port) built into the NDEV. Connect to a Wii AMP server from the connection dialog by specifying “wii” in the IP address field.

2.2 Analyzing Performance Metrics

Once a connection has been established, profiling information begins to flow into the client, and the graphs start getting populated. The top graph shows how much CPU time is spent per frame rendering GFx (Display), executing ActionScript (Advance), and in Direct Access API (User).

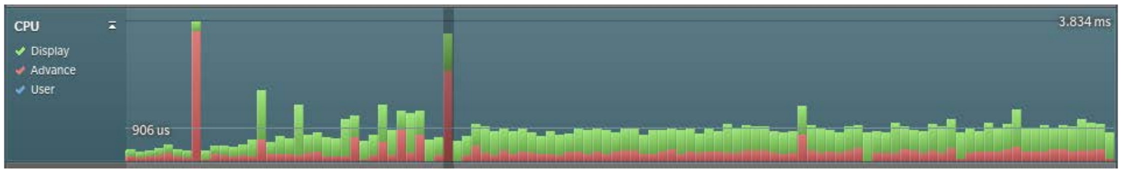
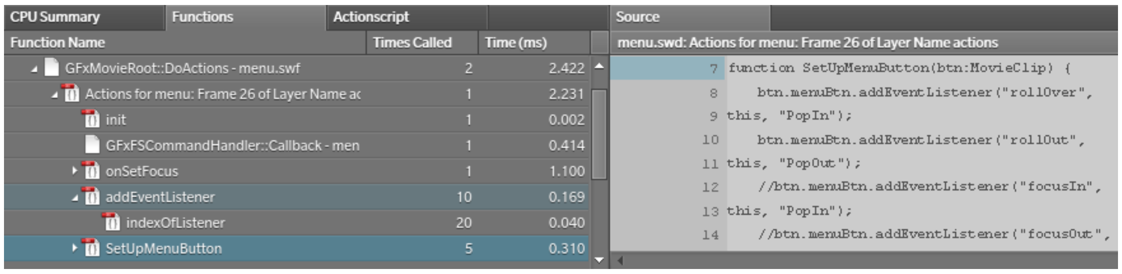


Figure 3: CPU Graph

If the CPU graph is not of interest at some point during the analysis, it may be minimized so that more room is available for other UI elements. If one of the graphed quantities is not of interest, it may be removed by clicking on the checkmark next to its legend.

With a frame on the CPU graph selected, the CPU Summary, Functions, and ActionScript tabs become visible at the bottom of the profiler. The CPU Summary tab shows the graph information in more detail, the Functions tab shows the call graph for the selected frame, and the ActionScript tab shows a non-hierarchical view of all script functions called during the selected frame.



Function Name	Times Called	Time (ms)
GFxMovieRoot::DoActions - menu.swf	2	2.422
Actions for menu: Frame 26 of Layer Name ac	1	2.231
init	1	0.002
GFxFSCommandHandler::Callback - men	1	0.414
onSetFocus	1	1.100
addEventListener	10	0.169
indexOfListener	20	0.040
SetUpMenuButton	5	0.310

menu.swd: Actions for menu: Frame 26 of Layer Name actions

```
function SetUpMenuButton(btn:MovieClip) {
   btn.menuBtn.addEventListener("rollOver",
this, "PopIn");
   btn.menuBtn.addEventListener("rollOut",
this, "PopOut");
   //btn.menuBtn.addEventListener("focusIn",
this, "PopIn");
   //btn.menuBtn.addEventListener("focusOut",
```

Figure 4 : CPU Summary

By examining the CPU detail tabs for a frame with a spike in the CPU graphs, the cause of script performance bottlenecks may be identified. On the other hand, rendering performance bottlenecks are more easily found by examining the rendering graph.

2.3 *Analyzing Rendering Metrics*

The rendering graph shows number of draw primitives, number of triangles, number of lines, number of masks used, number of strokes, number of gradient fills, number of glyphs in the mesh cache, and number of font updates that occurred during rendering for the selected frame. The Best Practices Guide explains in detail why minimizing the above quantities is important for performance.

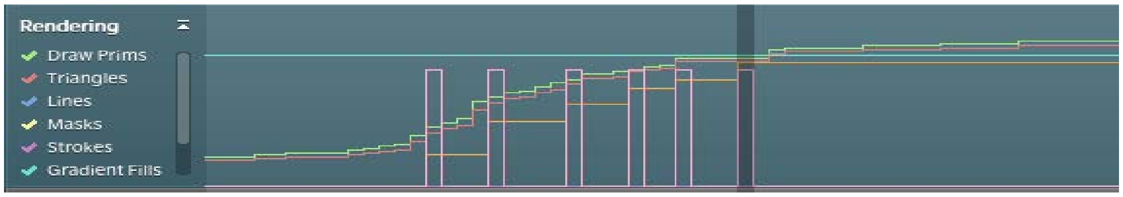
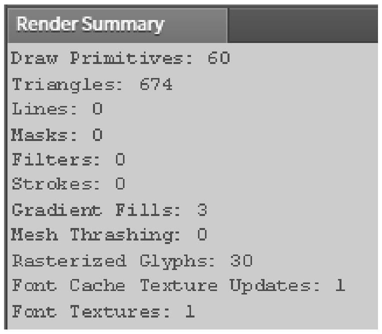


Figure 5: Rendering Graph

When a frame on the rendering graph is selected, the Render Summary tab becomes visible at the bottom of the application.



```
Draw Primitives: 60
Triangles: 674
Lines: 0
Masks: 0
Filters: 0
Strokes: 0
Gradient Fills: 3
Mesh Thrashing: 0
Rasterized Glyphs: 30
Font Cache Texture Updates: 1
Font Textures: 1
```

Figure 6: Rendering Summary

2.4 *Analyzing Memory Consumption*

The third graph from the top shows GFx memory usage:

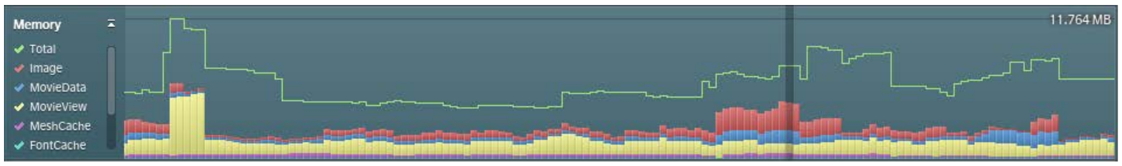


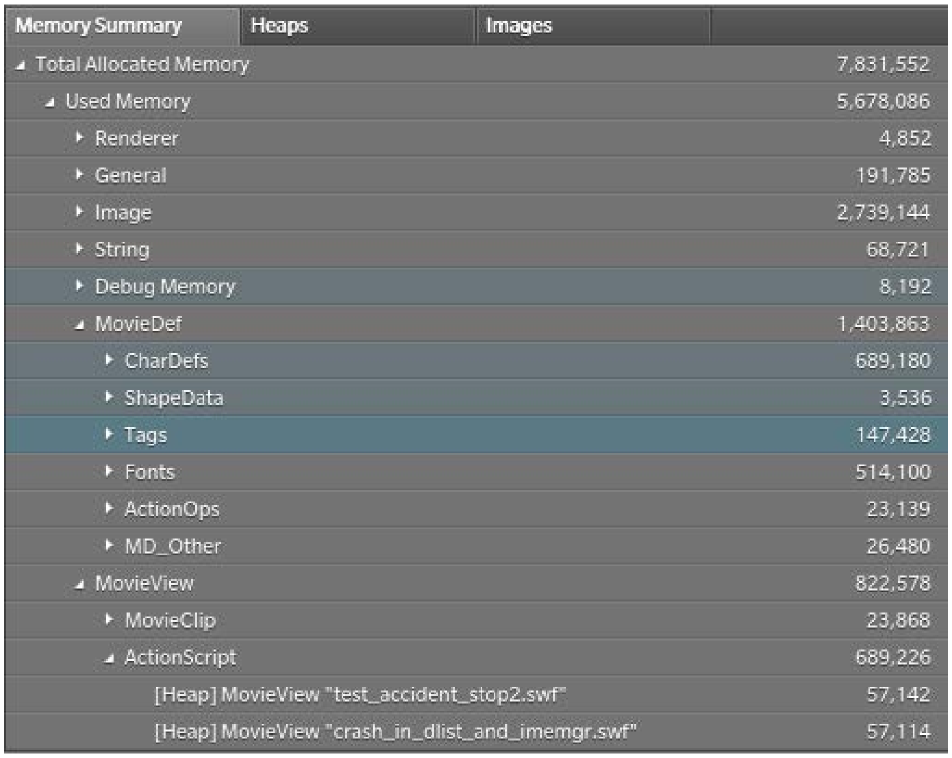
Figure 7: Memory graph

- **Total:** The total memory claimed by GFx (footprint). The total memory is drawn as a line graph, and includes all the categories listed below, as well as unused memory due to overhead, granularity in requested allocations, and fragmentation. Note that the total memory is greater than the sum of the used memory categories due to unused memory not being displayed in any category.

- **Image:** Memory used for images (loaded DDS textures). This will often be a major source of memory use.
- **MovieData:** Memory occupied by all loaded SWF or GFX files. Complex vector graphics objects, embedded fonts, and timeline animations affect this number.
- **MovieView**: Runtime memory occupied by GFxMovieView instances, allocated for timeline animation support, on-screen objects, and ActionScript.
- **MeshCache:** Memory occupied for the cached shape mesh data, generated by the vector tesselator and edge anti-aliasing
- **Font Cache:** Memory used by the font cache.
- **Sound**: Memory used for embedded sound data. It is affected by the number, length, or quality of embedded sound samples.
- **Video:** Memory used for video memory playback buffers. These buffers are allocated when videos start, and released when they stop.
- **Other:** Memory that is in use by GFx, but not part of the above categories.

See the Memory System Overview document for more details on the memory system and how to configure, optimize, and manage it.

When the memory graph is selected, the Memory Summary, Heaps, Images, and Fonts tabs become visible at the bottom of the application. The Memory Summary tab shows the above memory categories in more detail. All values in the memory tabs are in bytes, unless noted otherwise.

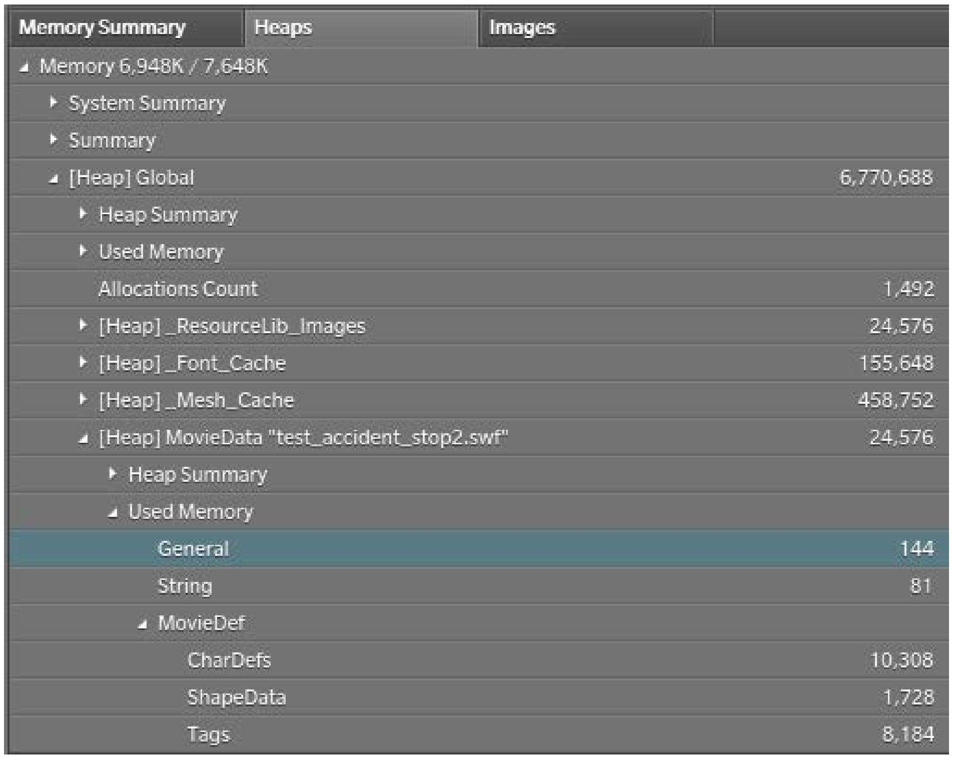


Total Allocated Memory	7,831,552
Used Memory	5,678,086
Renderer	4,852
General	191,785
Image	2,739,144
String	68,721
Debug Memory	8,192
MovieDef	1,403,863
CharDefs	689,180
ShapeData	3,536
Tags	147,428
Fonts	514,100
ActionOps	23,139
MD_Other	26,480
MovieView	822,578
MovieClip	23,868
ActionScript	689,226
[Heap] MovieView "test_accident_stop2.swf"	57,142
[Heap] MovieView "crash_in_dlist_and_imemgr.swf"	57,114

Figure 8: Memory Summary

Expanding each category reveals subcategories or memory heaps responsible for direct allocations in that category, if any. Drilling down in this manner, it is possible to gain a deeper understanding of GFx memory use in any particular frame.

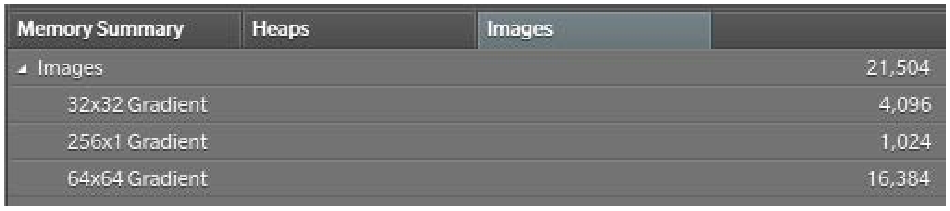
The Heaps tab offers a different view of the per-frame memory breakdown. The breakdown is essentially by heap first and then by category, rather than by category first and then by heap, as in the Memory Summary tab. It includes some extra information, and, therefore, this format may be more daunting for a developer without a thorough understanding of the GFx memory system. However, it is identical to the format used by the AMP memory HUD embedded in GFxPlayer, so it should be familiar to veteran GFx users.



```
Memory 6,948K / 7,648K
  System Summary
  Summary
  [Heap] Global                                        6,770,688
    Heap Summary
    Used Memory
    Allocations Count                                      1,492
    [Heap] _ResourceLib_Images                            24,576
    [Heap] _Font_Cache                                   155,648
    [Heap] _Mesh_Cache                                   458,752
    [Heap] MovieData "test_accident_stop2.swf"            24,576
      Heap Summary
      Used Memory
        General                                              144
        String                                                81
        MovieDef
          CharDefs                                        10,308
          ShapeData                                        1,728
          Tags                                             8,184
```

Figure 9: Heaps memory tab

The third Memory tab displays the images loaded in the selected frame(s), along with their sizes.

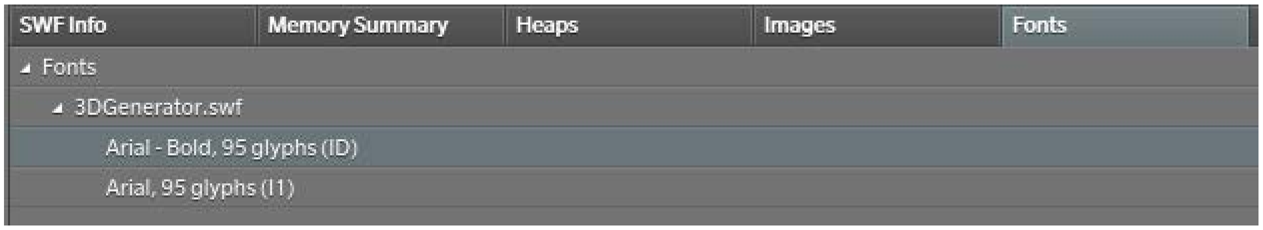


```
Images                         21,504
  32x32 Gradient                4,096
  256x1 Gradient                1,024
  64x64 Gradient               16,384
```

Figure 10: Images memory tab

Use this tab to quickly check whether a large image is present that increases the memory consumption of the target GFx application.

The fourth Memory Tab displays font information for the selected frame(s).



Fonts
3DGenerator.swf
Arial - Bold, 95 glyphs (ID)
Arial, 95 glyphs (I1)

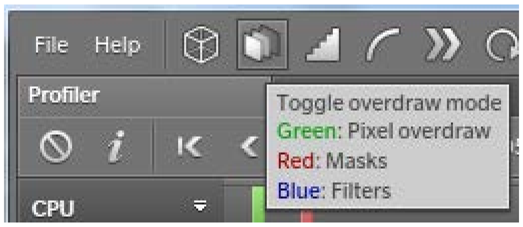
Figure 11: Fonts memory tab

Use this tab to examine the number of embedded glyphs for each font type, per SWF file.

Embedded fonts can consume a significant amount of memory, which shows up in AMP by increased sizes of the MovieData heap and the CharDefs category. These may be examined in the Heaps and Memory Summary tabs, respectively. Font memory may be reduced by sharing fonts over several SWF files.

2.5 Using Performance Highlighting

AMP may be used to remotely trigger a special rendering mode that highlights potential problem areas in the profiled application. To toggle this rendering mode, click on the appropriate button located on the application control toolbar.



Toggle overdraw mode
Green: Pixel overdraw
Red: Masks
Blue: Filters

Figure 12: Toggle Button

Once this mode is toggled, all objects are rendered in green. Green indicates pixel overdraw (16 layers for full brightness), masks are red, and filters are blue (8 layers of masks or filters for full brightness). These colors blend together, so a bright yellow area is high overdraw with nested masks.

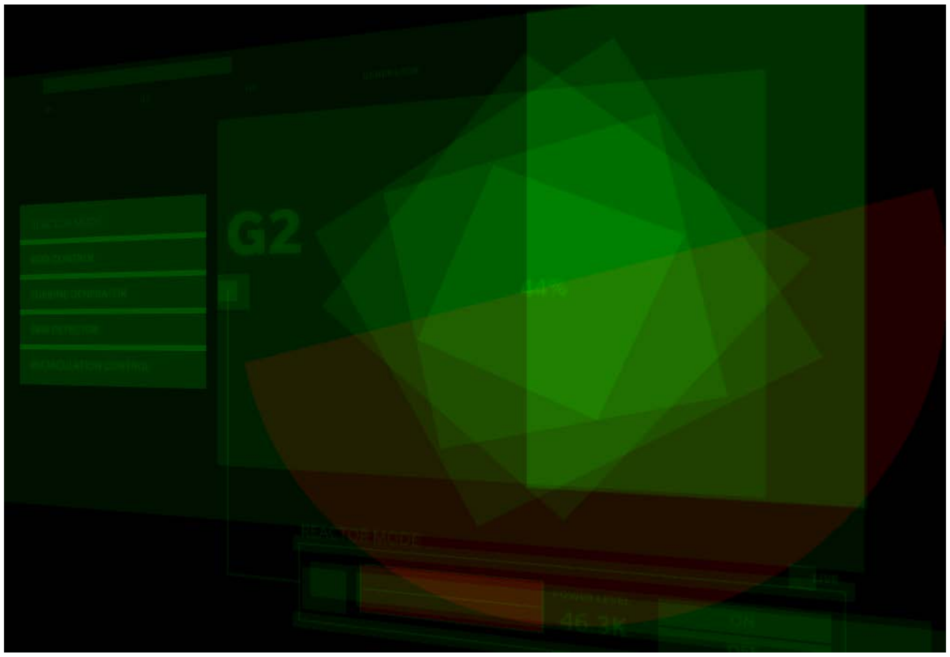


Figure 13: Rendering mode for Performance Highlighting

Overdraw, masks, and filters all affect performance, and should be minimized whenever possible.

3 Platform and Integration Notes

1. AMP server is currently supported for Windows, Xbox360, PS3, Linux, Mac, and Wii. Using the automatic discovery feature of the AMP client, connecting is straightforward. However, connecting without automatic discovery requires knowledge of the IP address for the AMP server, which can cause confusion because some platforms have multiple IP addresses.

    - For the Xbox360, please make sure to specify the title IP address, not the debug IP address. The title IP address can be determined by viewing the properties of the development or test kit in the Xbox 360 neighborhood.
    - For the PS3, if the development kit is configured to use two IP addresses, please make sure to specify the application IP address, not the IP address used for debugging.
    - The Wii does not use network sockets for communication with AMP. Instead, the console is physically connected to a PC using the USB interface (COM port) built into the NDEV. Users may connect to a Wii AMP server from the connection dialog by specifying “wii” in the IP address field. **Important:** Please make sure *RVL_SDK/X86/bin* is in the PATH environment variable, so that hio2.dll may be loaded by AMP.

2. Some game engines, such as Unreal Engine 3, package their assets in such a way that makes it impossible for the AMP server to locate the SWD files corresponding to the flash content being profiled. If that is the case, please make sure to place the SWD files in the AMP client working directory, rather than relying on the AMP server to find them in the same directory as the SWF and send them over the network.

4 Build Notes

1. AMP is part of all non-shipping GFx builds by default. To remove AMP from GFx, make sure GFX_AMP_SERVER is undefined in GConfig.h, and rebuild the GFx libraries. If GFx source access is not available, then AMP may be disabled at run-time by calling: `GFxAmpServer::GetInstance().SetState(AMP_Disabled, true)`

2. AMP uses network sockets for communication with the profiler. Therefore, make sure the application links with the appropriate platform-specific network library (Ws2_32.lib for Windows, Xnet.lib for Xbox360, libnetctl for PS3, socket for Linux, etc).

3. If AMP is the only system using network sockets, it will initialize the socket library, and release it when it is done. On some platforms, releasing the library has no effect if it has been initialized more times than it has been released. In that case, all is fine. On other platforms, however, releasing the network library takes immediate effect, even if that library has been initialized more times than has been released, and AMP may interfere with socket connections used in other parts of the application. In that case, AMP may be forced to skip socket initialization and use a previously-initialized socket library by calling: `GFxAmpServer::GetInstance().SetInitSocketLib(false).` This call needs to be performed before AMP has already been initialized.

4. AMP needs to receive information every game frame in order to display memory and performance statistics. If the application does not use a custom renderer, then no modification are needed for AMP, as the information is sent from within GRenderer::EndFrame. If the application does not call GRenderer::EndFrame, then the following call should be inserted to explicitly update AMP: `GFxAmpServer::GetInstance().AdvanceFrame()`

5. A limit may be set for the AMP memory heap. When that limit is exceeded, GFx is paused until any pending messages have been sent to the profiler. The limit is set by calling `GFxAmpServer::GetInstance().SetHeapLimit().`

6. The AMP server will create a listening network socket on port 7534 by default. A different port may be set by calling: `GFxAmpServer::GetInstance().SetListeningPort().`

7. AMP can be configured to halt execution until a connection has been established with the profiler client. This is useful for obtaining statistics during startup, and is accomplished on startup by calling: `GFxAmpServer::GetInstance().SetConnectionWaitTime.`

5 Adding AMP Support

Any GFx application may be configured to support remote profiling by AMP by following the steps outlined in this section. Also, the GFxPlayer source can be examined for an example of a fully-configured AMP server.

5.1 SWD File Generation

AMP has the ability to use Flash debug information (SWD files) to display source code and per-line timings. Generate SWD files by running the debugger (Ctrl+Shift+Enter) from within Flash CS3 or CS4. Place the SWD file either in the same location as the corresponding SWF, or in the working directory of the AMP client.

5.2 Application Control

AMP has the ability to remotely control settings in the profiled application, such as wireframe mode, or performance highlighting mode. Many of the supported settings are application-specific, and may be implemented by the application. Any such functionality not implemented will simply not be available via the AMP client.

Follow the steps outlined below to handle app-control messages sent from AMP:

- Implement a custom app-control callback class that derives from `GFxAmpAppControlInterface`, and override the `HandleAmpRequest` method to handle `GFxAmpMessageAppControl` messages from AMP.
- Install the custom handler by calling `GFxAmpServer::GetInstance().SetAppControlCallback` on startup.
- Inform AMP of the supported functionality by calling `GFxAmpServer::GetInstance().SetAppControlCaps`. The argument to this method is a `GFxAmpMessageAppControl` message, with the supported functionality set to true. For example:

```
GFxAmpMessageAppControl caps;
caps.SetCurveToleranceDown(true);
caps.SetCurveToleranceUp(true);
caps.SetNextFont(true);
caps.SetRestartMovie(true);
caps.SetToggleAaMode(true);
caps.SetToggleAmpRecording(true);
```

```
            caps.SetToggleFastForward(true);
            caps.SetToggleInstructionProfile(true);
            caps.SetToggleOverdraw(true);
            caps.SetToggleStrokeType(true);
            caps.SetTogglePause(true);
            caps.SetToggleWireframe(true);
            GFxAmpServer::GetInstance().SetAppControlCaps(&caps);
```

5.3 Connection Status

AMP client has the ability to display the name of the connected application in its status bar. This information is communicated by calling `GFxAmpServer::GetInstance().SetConnectedApp`.
Also, the AMP client can display the current application state, but for this it needs to be informed whenever the application state changes. Use the following methods to update the state:

- `GFxAmpServer::GetInstance().SetState`, with the first argument one of the following AmpServerStateType enumeration types:
    - `Amp_RenderOverdraw`
    - `Amp_App_Wireframe`
    - `Amp_App_Paused`
    - `Amp_App_FastForward`
- `GFxAmpServer::GetInstance().SetAaMode`
- `GFxAmpServer::GetInstance().SetStrokeType`
- `GFxAmpServer::GetInstance().SetCurrentLocale`
- `GFxAmpServer::GetInstance().SetCurveTolerance`
- Alternatively, a `GFxAmpCurrentState` object may be filled in with all the above information and passed to `GFxAmpServer::GetInstance().UpdateState`

5.4 Performance Highlighting

To use AMP performance highlighting in the application, provide a `GAmpRenderer` instance to the `GFxLoader` instead of the existing `GRenderer` instance. `GAmpRenderer` is a template that wraps the actual renderer and adds support for the performance highlighting features. Construct the instance as follows, replacing `GRendererOGL` with the appropriate renderer type:

```
GAmpRendererImpl<GRendererOGL>* pamprenderer = new
                              GAmpRendererImpl<GRendererOGL>(pRenderer);
Loader.SetRenderer(pamprenderer);
GFxAmpServer::GetInstance().AddAmpRenderer(pamprenderer);
```

5.5 Markers

AMP can display special indicators, markers, on the CPU graph that correspond to C++ or ActionScript calls made within the profiled application. Add a marker in ActionScript as follows:

```
Amp.addMarker("markerName")
```

Note: Compiling .as files that contain the Amp.addMarker() code may produce a compilation error. To avoid this, make sure to use the intrinsic Amp class provided under the Resources/CLIK directory.

Add a marker in C++ as follows:

```
GFxMovieRoot:: AdvanceStats->AddMarker("markerName")
```

AMP will then display a marker on the frame where the call was made. The string argument distinguishes different marker types, and is displayed on the SWF Info tab.

6 FAQs Related to AMP

This section contains few of the questions that developers may have when using the AMP tool.

1. Why is AMP pausing one second or more to update (GFxAmpServer::WaitForAmpThread)?

Normally, AMP server is paused for a second if it consumes too much memory, so that any accumulated data may be sent to the client. If this continues to occur, it is possible that your content results in so much profiling information that AMP does not have time to send it all to the client without pausing GFx.

You can reduce the amount of profiling information being sent, by toggling off source line timing and detailed memory reports (the 'i' button on AMP client). There is also a 'profile level' dropdown in the AMP client, which can be lowered as well. Finally, you can increase the memory threshold that triggers the problem. The default amount is set to 1MB, but you could modify it by calling GFxAmpServer::SetHeapLimit.

7 Additional Information

For more information on AMP, please note the following resources:

- AMP Use Cases document.
- GFx Reference document – for details on the code/classes.
- AMP Forum – for community questions and support.