Autodesk®
Scaleform®

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:	3.05
Last Edited:	February 24, 2014

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.4

How to Contact Autodesk Scaleform:

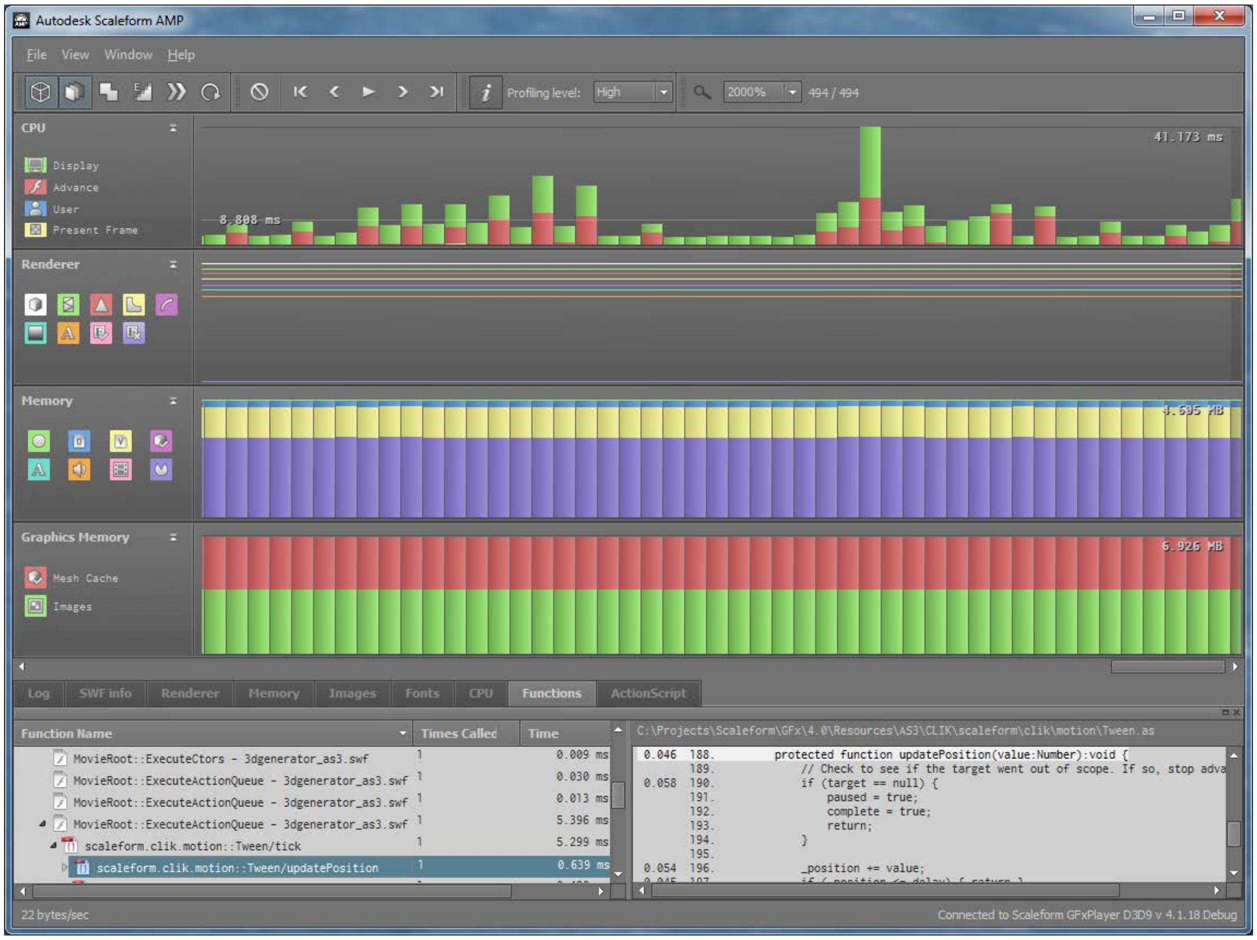
Document	AMP User Guide

Address	Autodesk 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 Scaleform 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.



Profiling level: High
2000% 494 / 494

41.173 ms
8.808 ms
4.695 MB
6.926 MB

Function Name	Times Called	Time
MovieRoot::ExecuteCtors - 3dgenerator_as3.swf	1	0.009 ms
MovieRoot::ExecuteActionQueue - 3dgenerator_as3.swf	1	0.030 ms
MovieRoot::ExecuteActionQueue - 3dgenerator_as3.swf	1	0.013 ms
MovieRoot::ExecuteActionQueue - 3dgenerator_as3.swf	1	5.396 ms
scaleform.clik.motion::Tween/tick	1	5.299 ms
scaleform.clik.motion::Tween/updatePosition	1	0.639 ms

C:\Projects\Scaleform\GFx\4.0\Resources\AS3\CLIK\scaleform\clik\motion\Tween.as

```
0.046        protected function updatePosition(value:Number):void {
                 // Check to see if the target went out of scope. If so, stop adva
0.058            if (target == null) {
                     paused = true;
                     complete = true;
                     return;
                 }

0.054            _position += value;
```

22 bytes/sec

Connected to Scaleform GFxPlayer D3D9 v 4.1.18 Debug

Figure 1: Scaleform AMP

Every Scaleform 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 Scaleform 3.2 and higher versions.

1.1 AMP Server

A Scaleform application being profiled (the AMP server) sends a certain amount of information over the network to the profiler. Depending on the level of function profiling, each ActionScript function and some important C++ functions may be timed. If per-line source code profiling is enabled in ActionScript 2, each ActionScript bytecode execution is timed (ActionScript 3 source code profiling does not require timing each execution). 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. Scaleform Shipping builds do not act as AMP servers, and AMP support may be easily disabled for any build within an application, if so desired.

Performance statistics on the server are maintained per Movie. 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 Movies 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 Scaleform Player, some of it requires application-specific implementation, and may be performed as part of a Scaleform integration. To graciously handle the case where some of these capabilities have not been implemented in a Scaleform 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 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 (ActionScript2) or AS file (ActionScript3) should be in a location where AMP can find it.

1.2 AMP Client

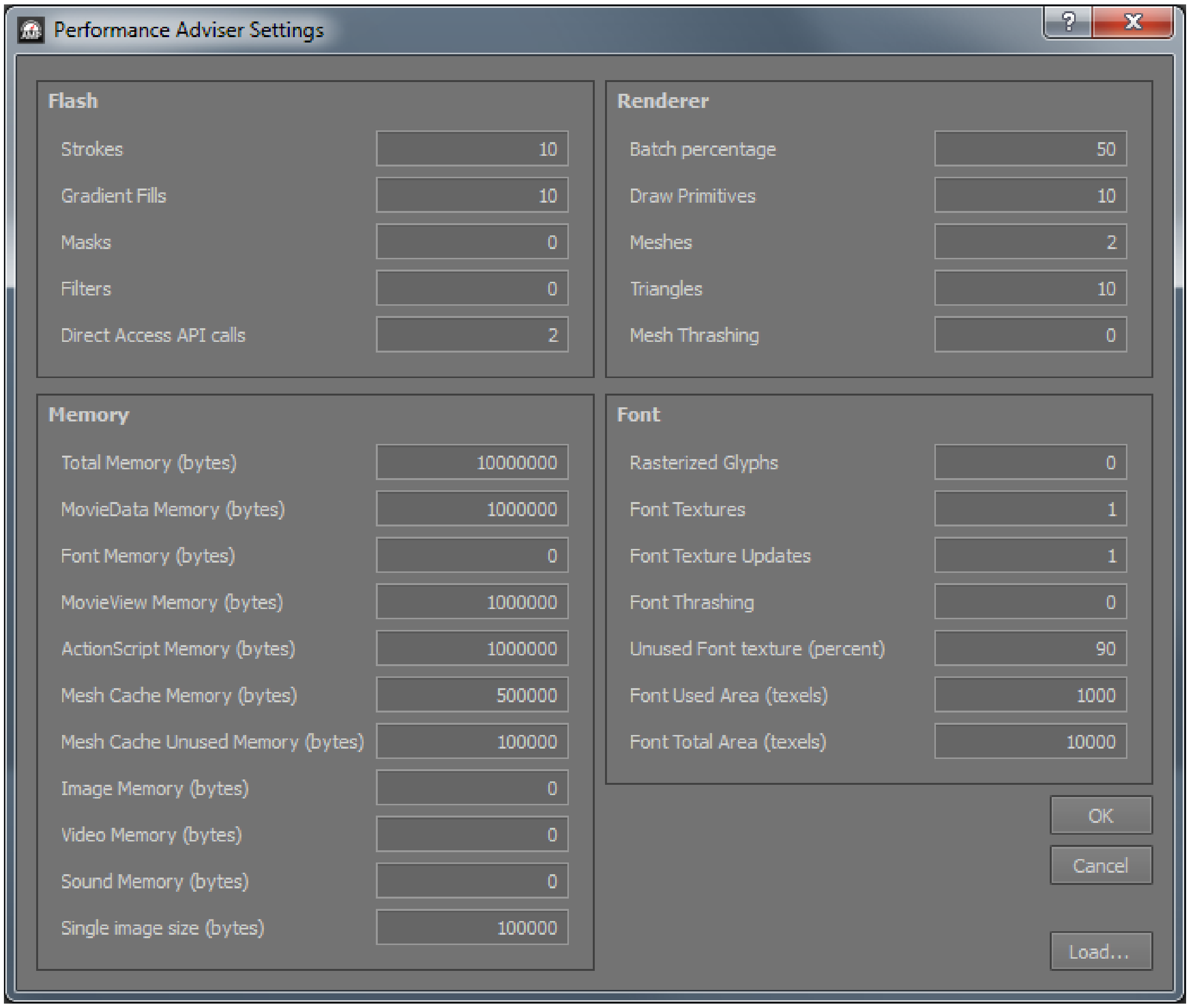
The remote profiler (AMP client) is a stand-alone application that connects to an AMP server, receives profile information every frame, and displays this information in a user-friendly way to enable effective analysis of performance and memory bottlenecks.
The AMP client has several features that allow 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), Scaleform rendering (Display), the Direct Access API (User), and graphics buffer swapping (PresentFrame). This graph is especially useful to identify spikes in CPU usage due to Scaleform. When a frame with a spike is selected, the cause can be identified by examining the detailed statistics for that frame. The user may select multiple frames by left-clicking on the graph and dragging the mouse until the desired range of frames has been selected. Right-clicking and dragging the graph scrolls the graph window. Each of the components of this graph may be shown or hidden by clicking on its corresponding toggle button in the legend box.
- **Rendering graph**: The Scaleform Best Practices Guide document identifies several rendering-related 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 Scaleform.
- **Memory graph:** The memory consumption of Scaleform is tracked in the memory graph, allowing the developer to quickly monitor system memory usage by category such as rendering, images, sound, fonts, movie data, etc, and includes all loaded files.
- **Graphics memory graph:** Most of the Scaleform graphics memory consumption is also tracked by AMP, which reports GPU memory consumed for images and for the mesh cache.
- **Log tab:** Clicking on the Log tab displays the Scaleform log that has been produced by the profiled application over the course of the connection.
- **Performance Adviser:** This tab provides guidance on how to improve the performance of Flash content. It is a list of warnings that fire when a user-defined threshold of some content-related property has been exceeded. The performance-related properties are:
    - *Text Fields optimized for readability*: If you use "Anti-alias for readability" for large text fields then the font cache contains separate glyph entries for each font size of each character. If a text field uses "Anti-alias for animation" then there will be only one entry, regardless of font size.
    - *Ratio of draw primitives to number of meshes*: This is a measure of batching efficiency. In the worst case of no batching, there will be one mesh for each draw primitive (ratio of 1.0). The ratio drops when some meshes are part of the same draw primitive, which is an indication of improved batching efficiency.

- *Number of draw primitives*: A high number of draw primitives is an indication of a complex scene that takes longer to render.
- *Number of meshes*: A high number of meshes is also an indication of a complex scene.
- *Number of triangles*: A high number of triangles hurts both memory and performance.
- *Mesh cache evictions*: This is the number of times per frame that an item is removed from the cache to make room for a new entry. Ideally, this number is zero, which means that everything needed by the renderer is found in the cache.
- *Number of strokes*: As indicated in the "Best Practices" guide, strokes in flash content result in a performance hit.
- *Number of gradient fills*: As indicated in the "Best Practices" guide, gradient fills also result in a performance hit.
- *Number of masks*: As indicated in the "Best Practices" guide, the use of masks results in a performance hit.
- *Number of filters*: As indicated in the "Best Practices" guide, the use of filters results in a performance hit.
- *Number of DirectAccess API calls*: Too many DirectAccess API calls can result in a performance hit.
- *Number of rasterized glyphs*: Too many glyphs can hurt memory consumption.
- *Number of font textures*: Some platforms may have more than one font texture.
- *Number of font texture allocations*: The number of times per frame a new font texture is created. This can hurt performance.
- *Font cache evictions*: This is the number of times per frame a glyph is removed from the cache to make room for a new glyph. Ideally, this number is zero, which means that the font cache is large enough to hold all rasterizations.
- *Font misses*: This is number of times per frame glyphs are rasterized because they are not found in the font cache. This number can be high when text has not been encountered before, and this is normal. Sometimes, there may not be enough space left in the glyph cache to store additional glyphs when new text is encountered. When this happens, older glyphs have to be evicted from the cache to make space for new ones. This means that when the old text is encountered again, it will have to be re-rasterized and rendered to the cache, incurring a performance penalty. This type of font cache eviction is bad and should be optimized whenever possible.
- *Unused font texture percentage*: A high percentage indicates that there is space reserved for the font cache that is not being used.
- *Used font cache area*: Too many different font and styles can result in excessive memory consumption.
- *Total font cache area*: The size of all the font textures. A font cache improves performance at a memory consumption cost.
- *Total Memory*: The total Scaleform memory footprint.
- *MovieData Memory*: This is directly related to the size of the imported movie files.
- *Font Memory*: Font memory can be reduced by having text fields share fonts.

- *MovieView Memory*: The memory used for GFx::Movie, containing its timeline and instance data. MovieView memory is subject to internal garbage collection.
- *ActionScript Memory*: Too much ActionScript content can result in excessive memory consumption.
- *Mesh Cache Memory*: The mesh cache improves performance, but hurts memory consumption.
- *Unused Mesh Cache Memory*: Unused mesh cache hurts memory consumption without improving performance.
- *Image Memory*: Both system and graphics card memory used for images.
- *Video Memory*: This is memory used for Scaleform video, not GPU memory.
- *Sound Memory*: Memory used for sound.
- *Number of images exceeding a certain size*: The user can set the image threshold, and AMP reports the number of images that exceed that threshold.

To set values for all of the above thresholds, AMP reads an XML file, called "adviser.xml" in its working directory. If it finds no such file, all thresholds are set to zero. The user can modify any threshold by clicking the Settings button on the right of the Performance Advisor tab. This brings up the settings dialog:


Performance Adviser Settings

Flash

Strokes	10
Gradient Fills	10
Masks	0
Filters	0
Direct Access API calls	2

Renderer

Batch percentage	50
Draw Primitives	10
Meshes	2
Triangles	10
Mesh Thrashing	0

Memory

Total Memory (bytes)	10000000
MovieData Memory (bytes)	1000000
Font Memory (bytes)	0
MovieView Memory (bytes)	1000000
ActionScript Memory (bytes)	1000000
Mesh Cache Memory (bytes)	500000
Mesh Cache Unused Memory (bytes)	100000
Image Memory (bytes)	0
Video Memory (bytes)	0
Sound Memory (bytes)	0
Single image size (bytes)	100000

Font

Rasterized Glyphs	0
Font Textures	1
Font Texture Updates	1
Font Thrashing	0
Unused Font texture (percent)	90
Font Used Area (texels)	1000
Font Total Area (texels)	10000

Clicking OK saves the settings to an XML file that AMP reads next time it is executed. The user may create different adviser settings for different content types, and select the one to be used by loading it from the settings dialog. Launching AMP with a command-line argument of "–adviser <AdviserFilePath>" makes the program read an adviser file other than the default one located in the working directory.

- **SWF info tab:** This tab provides some static information about the all the flash files being profiled. This information includes file name, flash version, movie dimensions, and movie frame rate.
- **Renderer tab:** This tab shows the same information as the graph, but in numeric form. It also includes some additional rendering statistics, such as mesh and font cache evictions, font fill rate (the area of all the glyphs as a percentage of the total area of the font cache textures), font cache failure rate (how many glyphs failed to allocate during the frame, and were replaced by vector shapes), number of font cache texture allocations during the frame, and current number of font textures.
- **Memory tab:** This tab allows the user to determine in detail the Scaleform footprint, how much memory claimed by Scaleform is actually in use, and how memory is allocated by purpose.

- **Images tab:** Shows the memory consumed by each image, along with its dimensions and format. Clicking on each item in the list displays a thumbnail of that image in the preview pane. Note that the image data are not sent as part of the profiling information, but are sent upon request. Therefore, it is possible that no image is displayed if that image has already been unloaded by Scaleform. AMP currently does not support preview for all image formats, such as DXT compressed images.
- **Fonts tab:** Shows the total memory consumed by the font cache, as well as the font list for each SWF. When an item in this tab is selected, an image representing the current font cache is displayed, in similar manner as for the image preview in the images tab.
- **CPU tab:** 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 tab:** 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.
- **ActionScript tab:** This tab shows similar information as the Functions tab, but the information is not hierarchical, C++ functions are excluded, and the time for each function excludes the time spent in other functions in the list. The tab is useful to quickly identify the most expensive script functions.
- **Source pane:** This window is part of the Functions and ActionScript tabs. It shows the amount of time spent in each line of the code for the selected function in the Functions tab. For AS3, per-line timings are always enabled for code that resides in AS files. For AS3 code embedded in the SWF, no source code is showm. For AS2, in order to see per-line timings, which are off by default, the "high" level of function profiling needs to be selected from the drop-down control on the toolbar. Instruction profiling in AS2 is time-consuming, and can slow down the Scaleform 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, or in the case of AS3 source files, in the same relative path to the loaded SWF.
- **Movie Objects tab:** The left pane shows the display object tree for each movie. The right pane shows a report about current ActionScript objects and the links between them. This report may be used to determine how objects in the selected MovieView reference each other and why they have not been garbage collected. Note that the information shown is updated only when a MovieView is selected in the list control on the left pane, and is therefore unrelated to the currently selected frame on the graphs of the profiler.
- **File menu: T**his menu includes the following choices:
    - **Connection:** Brings up the connection dialog, for connection to a different server.

  - **Disconnect:** Terminates the current connection.
  - **Compress Net Msgs:** Toggles network message compression. Turn this off to connect to an AMP server that does not support zlib compression.
  - **Debug Info Paths:** Brings up a dialog listing all the paths in the file system where AMP searches for SWD files (for AS2) or AS files (for AS3). Paths may be added or removed using the Add and Remove buttons, respectively.
  - **Load Profile Frames:** Loads a previously-saved profile run for reexamination. Selecting this menu item brings up a file load dialog, which is used to locate the profile data file.
  - **Save Profile Frames:** Saves the current profile run to disk. The run can be reloaded by selecting the "Load Profile Frames" option, above. Selecting this menu item brings up a file save dialog, which is used to select the path and name of the file containing the profile data.
  - **Dump Memory Report:** Saves the current profile frames to a file named "AmpMemReport.txt" in the working directory of AMP.
  - **Exit:** Closes the application.

- **View menu:** This menu includes the following choices:
  - **Graph Tooltips:** Selecting this menu toggles context-sensitive popup help when placing the mouse over the graphs in AMP.
  - **Time Units:** Allows the user to select the units for reporting all time quantities, such as display and advance times. The choices are milliseconds and microseconds.
  - **Memory Units:** Allows the user to select units for the Memory tab and the Memory graph. The choices are bytes, kilobytes, and megabytes.
- **Window menu:** This menu allows the user to show and hide any of the tabs (Log, SWF Info, Renderer, Memory, Images, Fonts, CPU, Functions, ActionScipt) described above. It also features a Restore UI choice, which removes any window customization the user has previously made to the AMP application.
- **Help menu:** This menu currently has two options: User Guide, which links to this document in the Scaleform.com developer center, and 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.
  - **Overdraw mode:** 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 overdraw mode, clicking the button takes it out of that mode.

- **Previous Batch:** This button is enabled only when batch profiling mode is active. Clicking in sends a request to the profiled application to change the highlighted batch.
- **Batch profiling mode:** Sends a request to the profiled application to ender a rendering mode that draws each render batch in a different color. When the application is already in batch profiling mode, clicking the button takes it out of that mode.
- **Next Batch:** This button is enabled only when batch profiling mode is active. Clicking in sends a request to the profiled application to change the highlighted batch.
- **Texture Density:** Sends a request to the profiled application to enter a rendering mode that shows the screen-space density of texels within a primitive. Primitives that do not use textures appear in blue with low alpha. Primitives with a texel density above two texels per pixel are shown in red, and those with a texel density of zero are shown in green. The color of texel density values between zero and two is linearly interpolated (green, yellow, red).
- **Blending view mode:** Sends a request to the profiled application to enter a rendering mode that shows primitives that use blending. Primitives that use blending are shown in green, while primitives that not use blending are shown in red.
- **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).
- **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.

- **Frame control toolbar:** This toolbar consists of the following buttons:
  - **Clear:** Discards the current profile run. The profile frames are discarded, thereby freeing up AMP client memory for new data.
  - **First Frame:** Selects the first frame in the current profile run, thereby updating the information displayed on the bottom panel tabs.
  - **Previous Frame:** Selects the previous frame from the currently-selected one, thereby updating the information displayed on the bottom panel tabs.

- **Pause/Play:** Sends a request to the profiled application to stop sending memory and performance data. This operation does not pause the application itself. Clicking the button again resumes the profiling.
- **Next Frame:** Selects the next frame from the currently-selected one, thereby updating the information displayed on the bottom panel tabs.
- **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, and the information displayed on the bottom panel tabs is continuously updated.

- **Profiling toolbar:** This element consists of a series of buttons that are used for profiling.
  - **Detailed Profiling** (detailed memory breakdown): When this option is selected, AMP gathers more detailed memory statistics per frame, which are displayed in the memory tab. This option also causes AMP to display the full list of images in the Images tab, rather than a maximum of ten.
  - **Profiling Level:** This drop-down control allows three levels of function profiling: "Low," "Medium," and "High." A low level of profiling causes AMP to time only some high-level C++ functions. A medium level of profiling results in ActionScript functions being timed as well as the functions from the previous level. A high level of profiling also gathers data from some lower-level C++ functions. In addition, the high level of profiling keeps track of the time spent in each line of ActionScript 2 code, which is displayed in the Code panel for the selected frame. A debug file (SWD) is required in order to show source line timings. In the case of ActionScript 3, these timings are always shown.
- **Zoom toolbar:** Consists of a drop-down control that 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. The toolbar also displays the currently selected frame number.
- **Status bar:** 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.

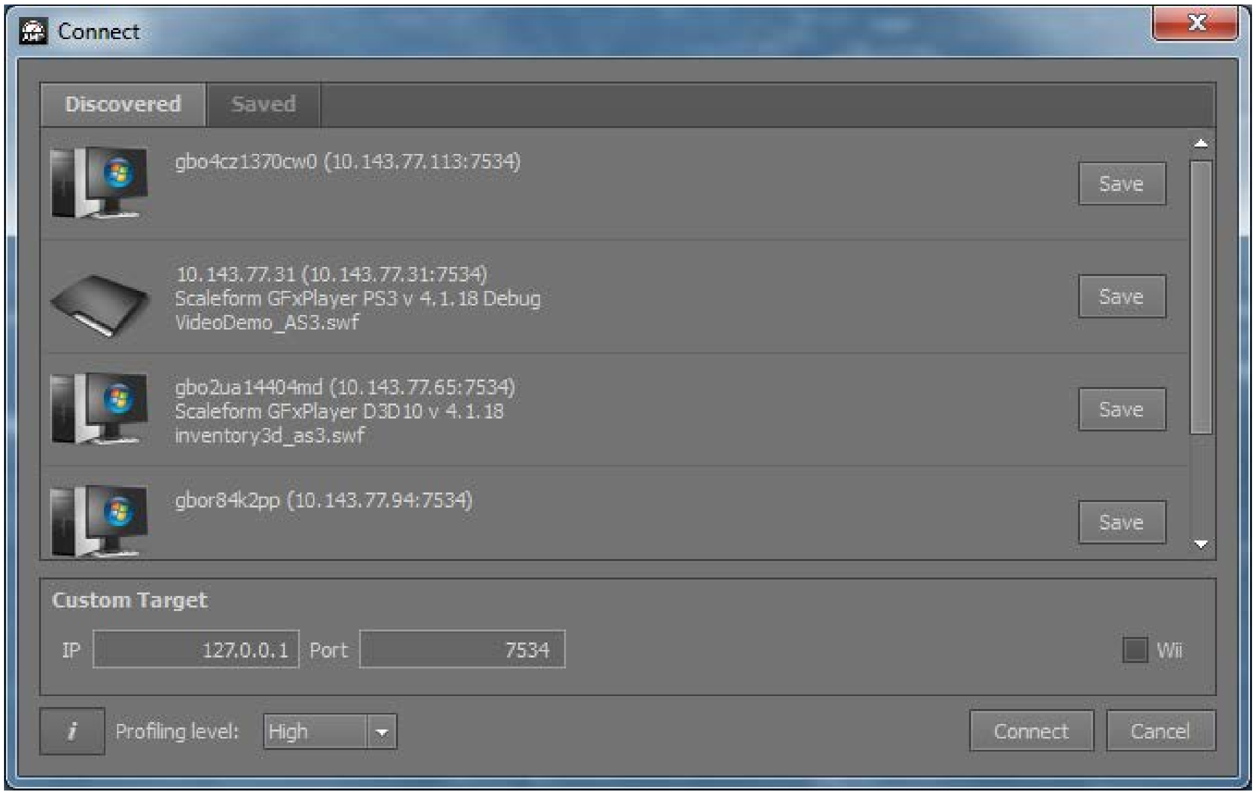
AMP allows the user to customize its appearance. The locations where the toolbars and tabs are docked may be changed by dragging them to their new locations. The tabs and toolbars may be undocked from the application. The sizes of the graphs and the tabs may be adjusted, and the graphs may be either partially or totally collapsed. The state of these UI changes is persistent over AMP profiling sessions, but the user can reset the geometry to its default state from the Reset UI selection in the Window menu.

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.



gbo4cz1370cw0 (10.143.77.113:7534)

10.143.77.31 (10.143.77.31:7534)
Scaleform GFxPlayer PS3 v 4.1.18 Debug
VideoDemo_AS3.swf

gbo2ua14404md (10.143.77.65:7534)
Scaleform GFxPlayer D3D10 v 4.1.18
inventory3d_as3.swf

gbor84k2pp (10.143.77.94:7534)

IP 127.0.0.1 Port 7534

Profiling level: High

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 Scaleform 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.

From the connection dialog it is possible to choose the profiling level and detailed memory reporting on connection, so that these are guaranteed to be set immediately on connection. This is especially important if the startup of a Scaleform application is being profiled.

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 Scaleform (Display), executing ActionScript (Advance), and in Direct Access API (User).

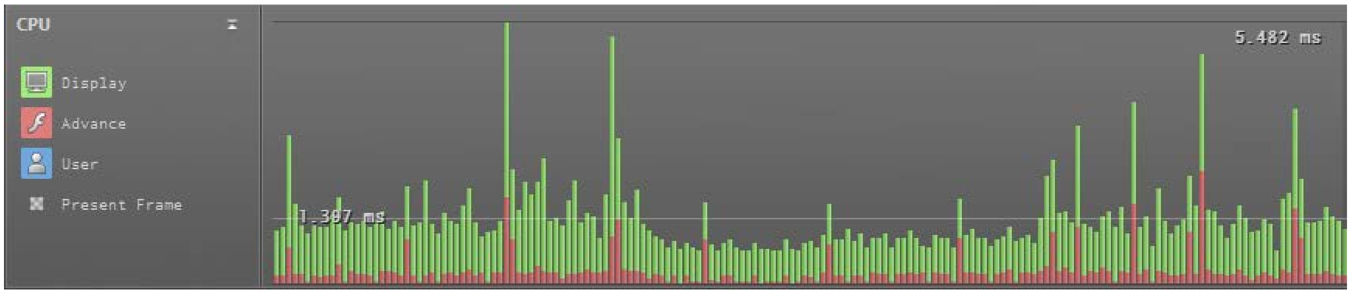
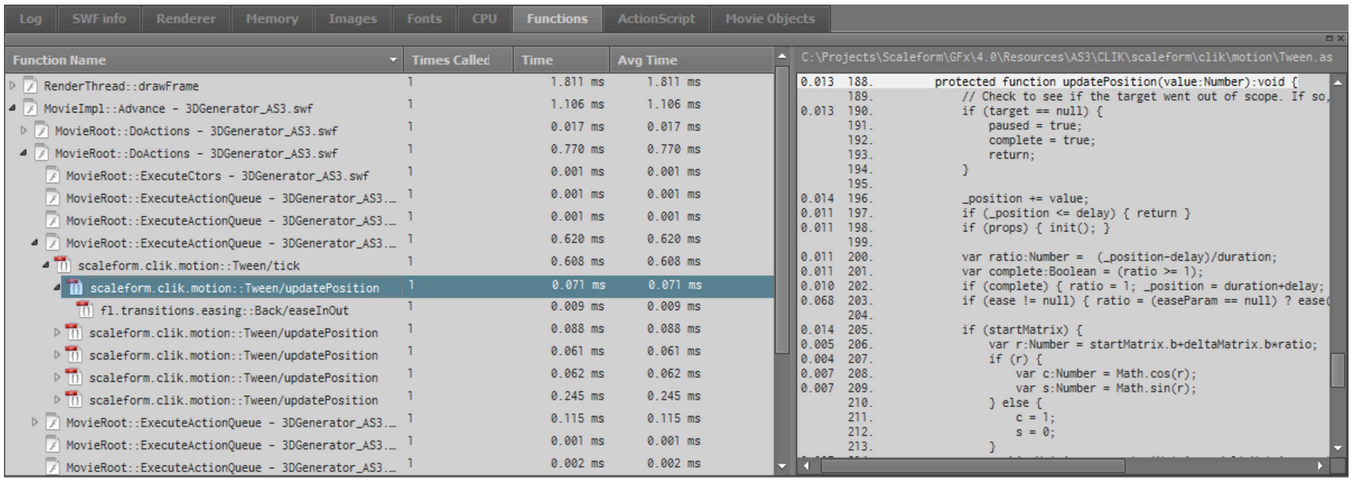


Figure 3: CPU Graph

If the CPU graph is not of interest at some point during the analysis, it may be minimized or resized 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 icon next to its legend.

Related to the CPU graph, are the CPU Summary, Functions, and ActionScript tabs 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	Avg Time
RenderThread::drawFrame	1	1.811 ms	1.811 ms
MovieImpl::Advance - 3DGenerator_AS3.swf	1	1.106 ms	1.106 ms
MovieRoot::DoActions - 3DGenerator_AS3.swf	1	0.017 ms	0.017 ms
MovieRoot::DoActions - 3DGenerator_AS3.swf	1	0.770 ms	0.770 ms
MovieRoot::ExecuteCtors - 3DGenerator_AS3.swf	1	0.001 ms	0.001 ms
MovieRoot::ExecuteActionQueue - 3DGenerator_AS3.…	1	0.001 ms	0.001 ms
MovieRoot::ExecuteActionQueue - 3DGenerator_AS3.…	1	0.001 ms	0.001 ms
MovieRoot::ExecuteActionQueue - 3DGenerator_AS3.…	1	0.620 ms	0.620 ms
scaleform.clik.motion::Tween/tick	1	0.608 ms	0.608 ms
scaleform.clik.motion::Tween/updatePosition	1	0.071 ms	0.071 ms
fl.transitions.easing::Back/easeInOut	1	0.009 ms	0.009 ms
scaleform.clik.motion::Tween/updatePosition	1	0.088 ms	0.088 ms
scaleform.clik.motion::Tween/updatePosition	1	0.061 ms	0.061 ms
scaleform.clik.motion::Tween/updatePosition	1	0.062 ms	0.062 ms
scaleform.clik.motion::Tween/updatePosition	1	0.245 ms	0.245 ms
MovieRoot::ExecuteActionQueue - 3DGenerator_AS3.…	1	0.115 ms	0.115 ms
MovieRoot::ExecuteActionQueue - 3DGenerator_AS3.…	1	0.001 ms	0.001 ms
MovieRoot::ExecuteActionQueue - 3DGenerator_AS3.…	1	0.002 ms	0.002 ms

```
C:\Projects\Scaleform\GFx\4.0\Resources\AS3\CLIK\scaleform\clik\motion\Tween.as
0.013       protected function updatePosition(value:Number):void {
                // Check to see if the target went out of scope. If so,
0.013           if (target == null) {
                    paused = true;
                    complete = true;
                    return;
                }

0.014           _position += value;
0.011           if (_position <= delay) { return }
0.011           if (props) { init(); }

0.011           var ratio:Number =  (_position-delay)/duration;
0.011           var complete:Boolean = (ratio >= 1);
0.010           if (complete) { ratio = 1; _position = duration+delay;
0.068           if (ease != null) { ratio = (easeParam == null) ? ease(

0.014           if (startMatrix) {
0.005               var r:Number = startMatrix.b+deltaMatrix.b*ratio;
0.004               if (r) {
0.007                   var c:Number = Math.cos(r);
0.007                   var s:Number = Math.sin(r);
                    } else {
                        c = 1;
                        s = 0;
                    }
```

Figure 4 : Functions tab

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 cache allocations that occurred during rendering for the selected frame. The Best Practices Guide explains in detail why minimizing the above quantities is important for performance.

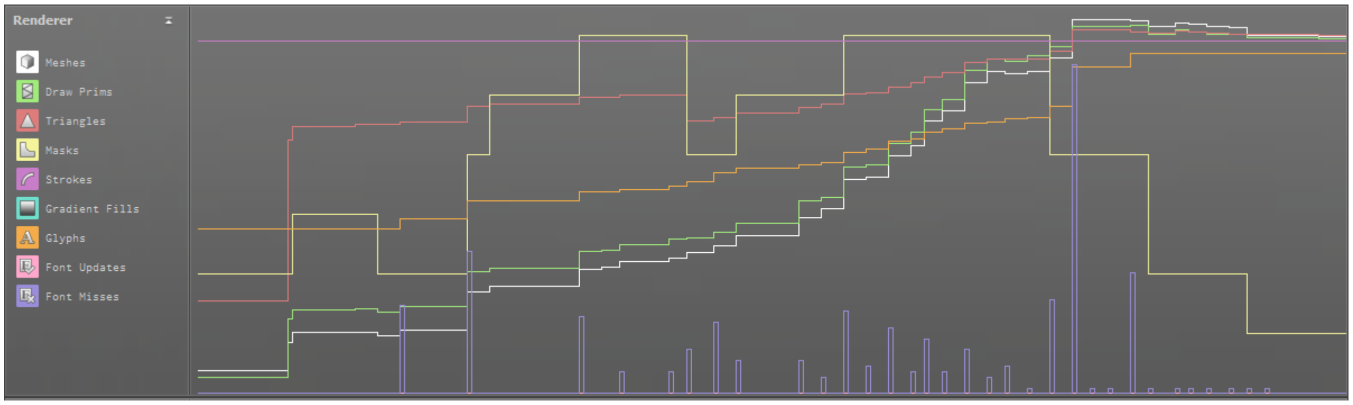
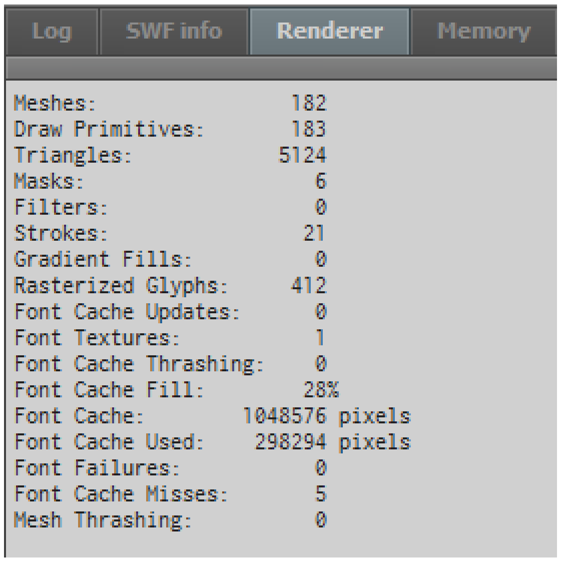


Figure 5: Rendering Graph

Related to the rendering graph, is the Render Summary tab at the bottom of the application.



```
Meshes:                182
Draw Primitives:       183
Triangles:            5124
Masks:                   6
Filters:                 0
Strokes:                21
Gradient Fills:          0
Rasterized Glyphs:     412
Font Cache Updates:      0
Font Textures:           1
Font Cache Thrashing:    0
Font Cache Fill:        28%
Font Cache:        1048576 pixels
Font Cache Used:    298294 pixels
Font Failures:           0
Font Cache Misses:       5
Mesh Thrashing:          0
```

Figure 6: Rendering Summary

2.4 Analyzing Memory Consumption

The third and fourth graphs from the top show Scaleform system and graphics memory usage, respectively:

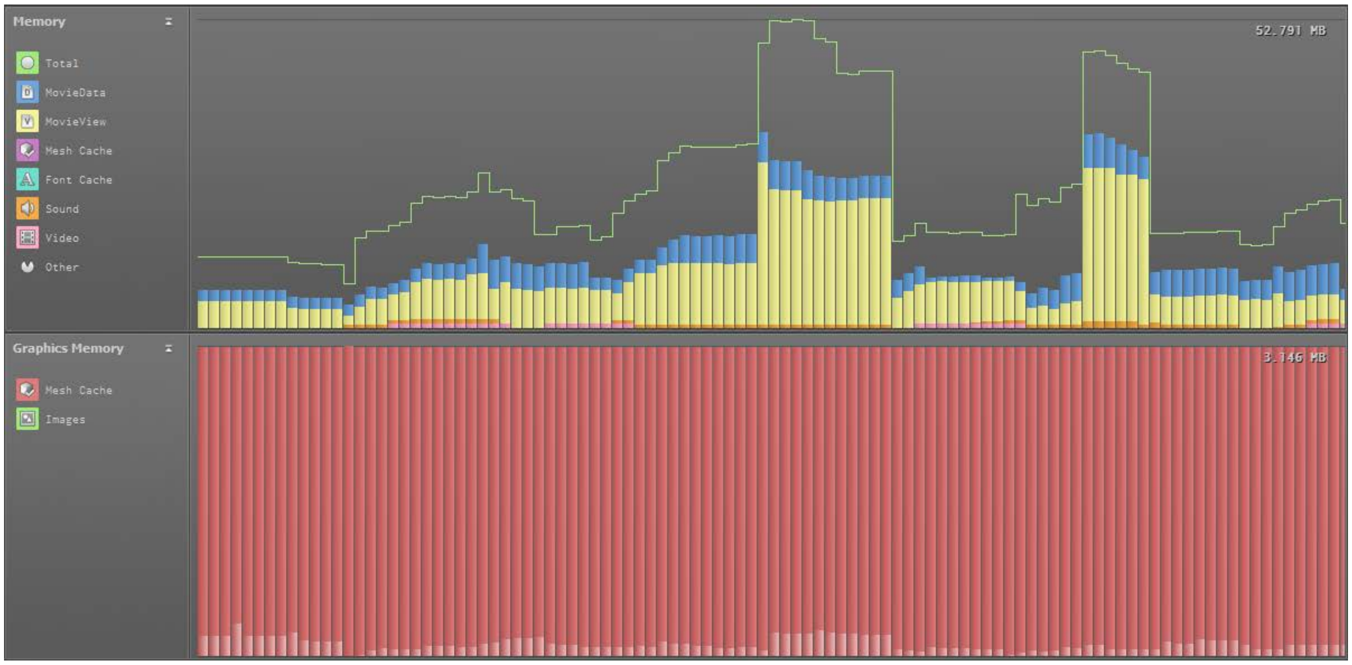


Figure 7: Memory graphs

- **Total:** The total memory claimed by Scaleform (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.

- **MovieData:** Memory occupied by all loaded SWF or GFX files. Object data, embedded fonts, and timeline animations affect this number.
- **MovieView**: Runtime memory occupied by GFx::Movie instances, allocated for timeline animation support, on-screen objects, and ActionScript.
- **Mesh Cache:** Memory occupied for the cached shape mesh data, generated by the vector tesselator and edge anti-aliasing. The mesh cache memory may be allocated from system or graphics memory, depending on the platform. The memory actually used is shown in a darker shade, so that the user may easily identify if a smaller mesh cache could be set, thereby saving memory.
- **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 Scaleform, but not part of the above categories. For a more detailed breakdown of memory categories, please toggle detailed memory reports, as described below
- **Images:** Graphics memory used for images (textures).

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

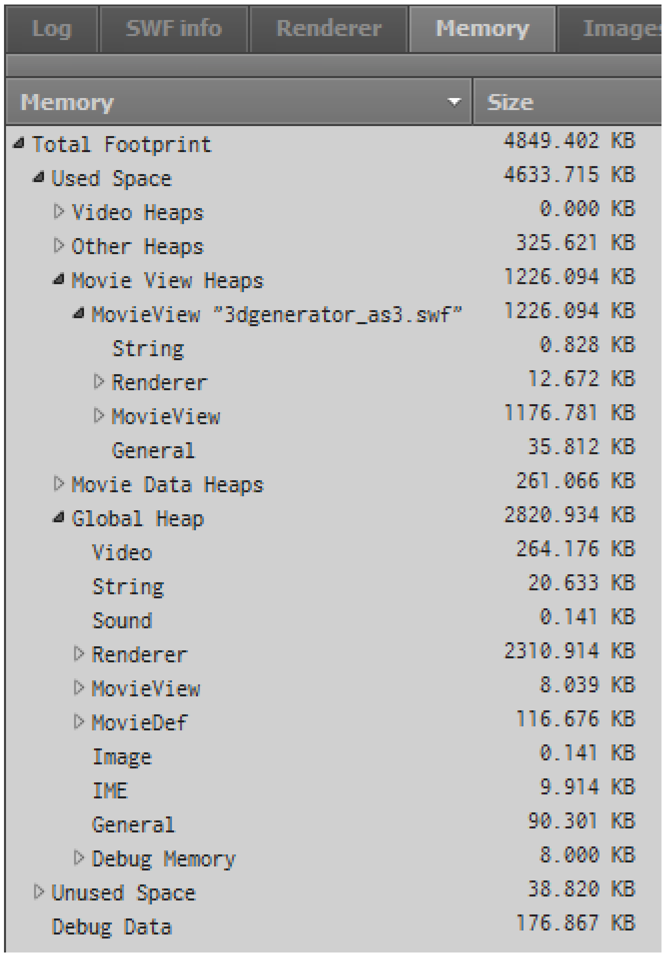
The Memory tab shows the above memory categories in more detail. All values in the memory tabs are in the same units (bytes, kilobytes, or megabytes), as specified in the memory units selection, under the view menu. Please note that “Used Space” is memory in active use by GFx. “Unused Space” is memory that has been reserved by GFx but is not being used, mostly due to fragmentation. Debug Data denotes memory used internally by GFx for tracking memory and performance statistics.

When profiling in full memory report mode, which is toggled by the “i” button on the AMP toolbar, the memory report becomes more detailed, showing a breakdown by heap, by file, and by purpose. The debug version of the Scaleform libraries separates heap allocations into memory categories, from which the user can obtain a better understanding of how Scaleform is using the memory that is being consumed. The memory categories reported are as follows:

- **Renderer:** Memory allocated by the Scaleform rendering engine. This category is further subdivided into the following subcategories:
  - **Buffers:** Memory allocated for the mesh staging buffer, which acts as a short-term cache that stores generated meshes before they are instanced and copied into the vertex and index buffers.

    - **RenderBatch:** Memory allocated for collections of entities such as meshes, primitives, or rendering commands associated with nodes in the rendering tree that are to be drawn together.
    - **Primitive:** Memory used to hold collections of vertices that are to be drawn together.
    - **Fill:** Memory used to store and manage data describing a hardware fill.
    - **Mesh:** Memory allocated for the mesh cache. The mesh cache is used to feed to the renderer with vertex and index buffer data.
    - **MeshBatch:** Memory allocated for the individual chunks of data cached in vertex and index buffers of the Mesh Cache.
    - **Context:** Memory consumed by the rendering support system that manages the lifetime of the rendering trees. The context maintains multiple logically independent snapshots of the rendering tree so that threads can work on the rendering data without interfering with each other.
    - **NodeData:** Memory for frame-state information, such as the current matrix or effects applied to the objects being rendered.
    - **TreeCache:** Memory allocated for the renderer's scene graph, which is a tree structure of objects that are being rendered.
    - **TextureManager:** Memory allocated to managing texture images. The actual image data are normally not stored in system memory, and are not, therefore, part of this memory category.
    - **MatrixPool:** Memory consumed by a pool of matrices maintained by the renderer.
    - **MatrixPoolHandles:** Memory consumed by allocation of matrix handles.
    - **Text:** Memory allocated for text management and rendering, such as formatting information, style and color information, html parsing, rich text data, and line buffers.
    - **Font:** Memory related to font rendering, such as glyph rasterization and font cache storage.
- **MovieDef:** Memory allocated for the immutable data representing the template of a movie element. It consists of the following subcategories:
    - **CharDefs:** Memory used for character definitions, such as buttons and text.
    - **ShapeData:** Memory used for shape definitions.
    - **Tags:** Memory used for ActionScript tags
    - **Fonts:** Memory used for font resources.
    - **Sounds:** Memory used for sound data.
    - **ASBinaryData:** Memory consumed by the ActionScript buffer data.
    - **MD_Other:** Other MovieDef data, not included in the above categories.
- **MovieView:** Memory allocated for timeline animation support, on-screen objects, and ActionScript. It is further divided into the following subcategories:
    - **MovieClip:** Memory used for on-screen animated objects.
    - **ActionScript:** Memory consumed by ActionScript code and data.
        - **ASString:** Memory consumed by ActionScript strings
    - **Text:** Memory consumed for text-related display objects.

        - **XML:** Memory consumed for XML objects.
        - **MV_Other:** Other Movie View memory that does not fit in the above categories.
        - **VM:** Memory allocated for the ActionScript virtual machine. This category is further broken down as follows:
            - **AS3 VM:** Memory consumed by the ActionScript 3 virtual machine, but not part of the following subcategories.
            - **CallFrame:** Memory allocated for each entry in the call stack during ActionScript execution.
            - **VTable:** Memory consumed by the ActionScript virtual table
            - **SlotInfo:** Memory used for structures that describe ActionScript Class members. Each ClassTraits and InstanceTraits contain SlotInfo structures in order to describe the Class or Instance memory layout. See below for more information on ClassTraits and InstanceTraits.
                - **SlotInfoHash:** Memory used by hash tables that speed up property lookup for ActionScript.
            - **ClassTraits:** Memory allocated to hold the memory layout of ActionScript Classes
            - **Class:** Memory allocated to hold static members of ActionScript objects
            - **InstanceTraits:** Memory allocated to hold the non-static memory layout of ActionScript Objects.
            - **Instance:** Memory allocated to hold instance variables and methods of ActionScript objects.
            - **AbcFile:** Memory allocated for data structures representing byte code and its corresponding data.
                - **AbcConstPool:** Memory used for storing constants, such as numbers, strings, namespaces, and multinames. This is just byte code, so it is part of AbcFile, above.
            - **VMAbcFile:** Memory holding an internal data structure that stores the optimized byte code generated from the AbcFiles, dependencies among files, and other information.
            - **Tracer:** Memory allocated for code optimization.
    - **IME:** Memory consumed for Input Method Editor (IME) support.



Memory	Size
◢ Total Footprint	4849.402 KB
◢ Used Space	4633.715 KB
▷ Video Heaps	0.000 KB
▷ Other Heaps	325.621 KB
◢ Movie View Heaps	1226.094 KB
◢ MovieView "3dgenerator_as3.swf"	1226.094 KB
String	0.828 KB
▷ Renderer	12.672 KB
▷ MovieView	1176.781 KB
General	35.812 KB
▷ Movie Data Heaps	261.066 KB
◢ Global Heap	2820.934 KB
Video	264.176 KB
String	20.633 KB
Sound	0.141 KB
▷ Renderer	2310.914 KB
▷ MovieView	8.039 KB
▷ MovieDef	116.676 KB
Image	0.141 KB
IME	9.914 KB
General	90.301 KB
▷ Debug Memory	8.000 KB
▷ Unused Space	38.820 KB
Debug Data	176.867 KB

Figure 8: Memory

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 Scaleform memory use in any particular frame.

The Images tab is also related to memory profiling, as it displays the images loaded in the selected frame(s), along with their sizes.

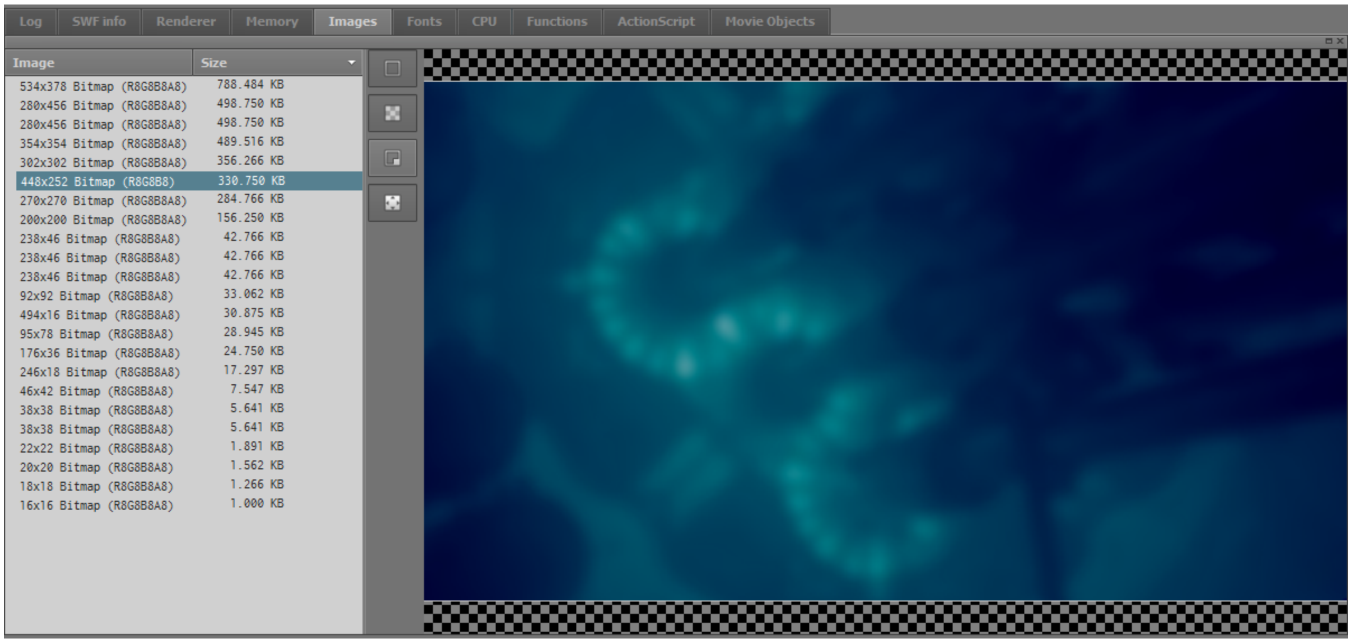
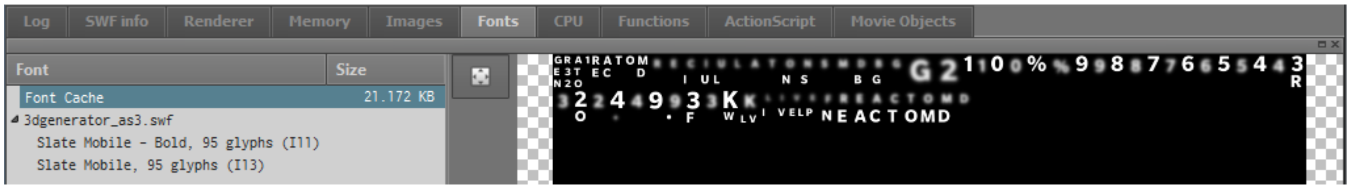


Image	Size
534x378 Bitmap (R8G8B8A8)	788.484 KB
280x456 Bitmap (R8G8B8A8)	498.750 KB
280x456 Bitmap (R8G8B8A8)	498.750 KB
354x354 Bitmap (R8G8B8A8)	489.516 KB
302x302 Bitmap (R8G8B8A8)	356.266 KB
448x252 Bitmap (R8G8B8)	330.750 KB
270x270 Bitmap (R8G8B8A8)	284.766 KB
200x200 Bitmap (R8G8B8A8)	156.250 KB
238x46 Bitmap (R8G8B8A8)	42.766 KB
238x46 Bitmap (R8G8B8A8)	42.766 KB
238x46 Bitmap (R8G8B8A8)	42.766 KB
92x92 Bitmap (R8G8B8A8)	33.062 KB
494x16 Bitmap (R8G8B8A8)	30.875 KB
95x78 Bitmap (R8G8B8A8)	28.945 KB
176x36 Bitmap (R8G8B8A8)	24.750 KB
246x18 Bitmap (R8G8B8A8)	17.297 KB
46x42 Bitmap (R8G8B8A8)	7.547 KB
38x38 Bitmap (R8G8B8A8)	5.641 KB
38x38 Bitmap (R8G8B8A8)	5.641 KB
22x22 Bitmap (R8G8B8A8)	1.891 KB
20x20 Bitmap (R8G8B8A8)	1.562 KB
18x18 Bitmap (R8G8B8A8)	1.266 KB
16x16 Bitmap (R8G8B8A8)	1.000 KB

Figure 99: Images memory tab

Use this tab to quickly check whether a large image is present that increases the memory consumption of the target Scaleform application. Clicking on an item in the image list displays a thumbnail of that image in the preview pane. Note that the image data are not sent as part of the profiling information, but are sent upon request. Therefore, it is possible that no image is displayed if that image has already been unloaded by the connected Scaleform application.

The final tab related to memory profiling is the Fonts tab, which displays font information for the selected frame(s).



Font	Size
Font Cache	21.172 KB
◢ 3dgenerator_as3.swf	
Slate Mobile - Bold, 95 glyphs (I11)	
Slate Mobile, 95 glyphs (I13)	

Figure 10: 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 tabs, respectively. Font memory may be reduced by sharing fonts over several SWF files.

2.5 Using Overdraw Profiling

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.

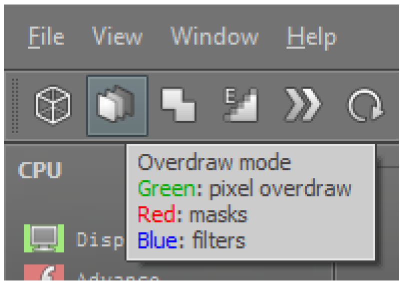


Figure 11: Toggle Button

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.

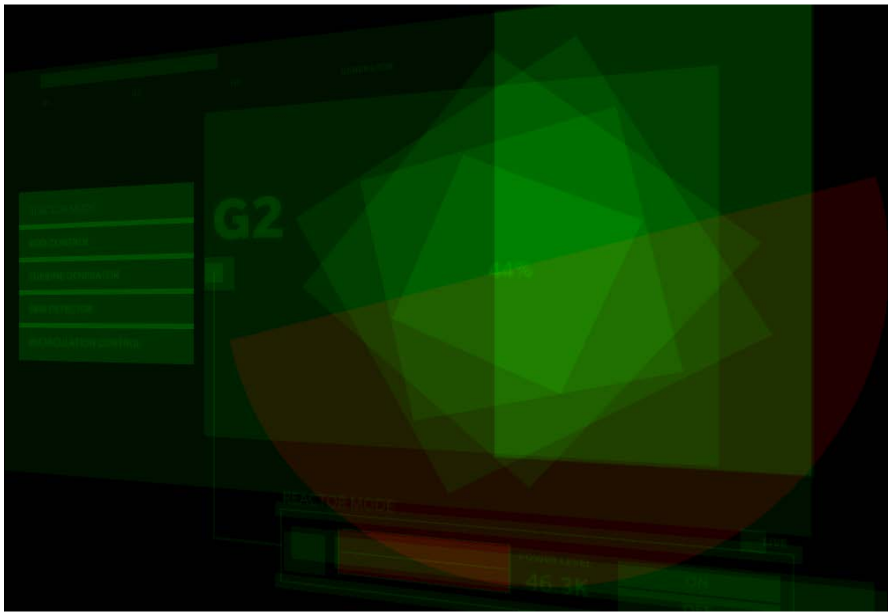


Figure 12: Rendering mode for Overdraw profiling

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

2.6 Using Batch Profiling

Another special rendering mode that AMP supports is batch profiling. To toggle this rendering mode, click on the appropriate button located on the application control toolbar.

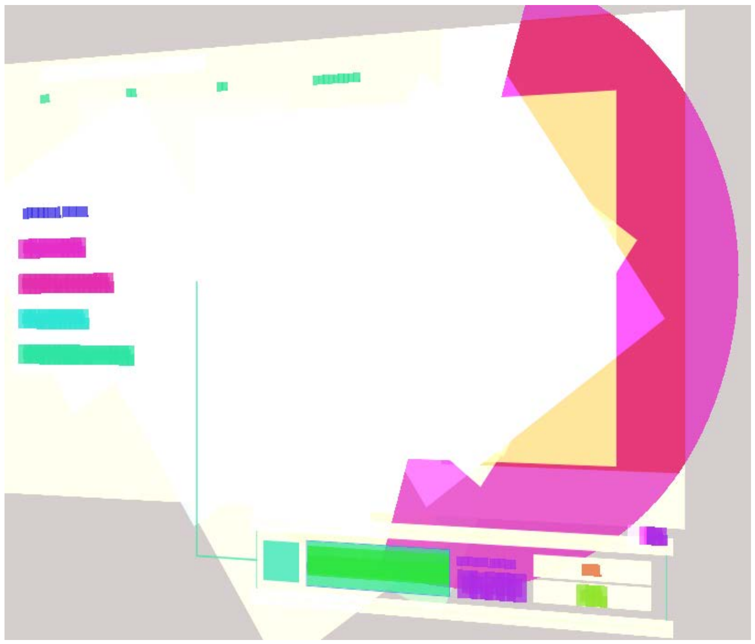


Figure 13: Batch profiling mode

Once this mode is toggled, each rendering batch is rendered in a different color. Best performance is achieved by minimizing the number of batches, whenever possible.

3 Platform and Integration Notes

1. AMP is normally initialized during the GFx::System object construction, or during the GFx::System::Init function call. Alternatively, if a Scaleform::System is instead used to initialize Scaleform, the programmer needs to explicitly initialize and shut down AMP by calling `GFx::AMP::Server::Init()`and `GFx::AMP::Server::Uninit(),` respectively.

2. AMP needs to receive information every game frame in order to display memory and performance statistics. The following call should be inserted in the rendering thread to explicitly update AMP: `AmpServer::GetInstance().AdvanceFrame()`

3. The AMP client application is currently available only for Windows.

4. AMP server is currently supported for Windows, Xbox360, PS3, Linux, Mac, Wii, Android, iPhone/iPad, 3DS, PS Vita, WiiU, and XboxOne.. 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.
   - For Android, internet permission needs to be enabled at the application level. To do this, the following needs to be added to AndroidManifest.xml for the application:

     ```
     <uses-permission android:name="android.permission.INTERNET" />
     ```

   - Please connect to the WiiU through the host PC bridge, not through WiFi connection. The host PC detects the console by automatic discovery of port 6003. Once the connection request is made, communication is switched from TCP/IP sockets to the host bridge.
   - 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.
   - For the XboxOne, AMP works only on development consoles. Applications that want to use AMP need to specify a SecureDeviceSocketUsage of SendDebug and ReceiveDebug in the XboxNetworkingManifest. For example:

```
<mx:Extension Category="windows.xbox.networking">
```

```
   <mx:XboxNetworkingManifest>
       <mx:SocketDescriptions>
          <mx:SocketDescription Name="AMP" SecureIpProtocol="Tcp" BoundPort="7534">
             <mx:AllowedUsages>
                <mx:SecureDeviceSocketUsage Type="SendDebug" />
                <mx:SecureDeviceSocketUsage Type="ReceiveDebug" />
             </mx:AllowedUsages>
          </mx:SocketDescription>
       </mx:SocketDescriptions>
   </mx:XboxNetworkingManifest>
</mx:Extension>
```

5. 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 AS2 SWD or AS3 AS files corresponding to the flash content being profiled. If that is the case, please make sure to place the SWD or AS 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 Scaleform builds by default. To remove AMP from Scaleform, make sure SF_AMP_SERVER is undefined in GFxConfig.h, and rebuild the Scaleform libraries. If Scaleform source access is not available, then AMP may be disabled at run-time by calling:
   ```
   Ptr<GFx::AMP::ServerState> serverState = *SF_NEW GFx::AMP::ServerState();
   serverState->StateFlags |= GFx::AMP::Amp_Disabled;
   AmpServer::GetInstance().UpdateState(serverState);
   ```

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: `AMP::Server::GetInstance().SetInitSocketLib(false).` This call needs to be performed before AMP has already been initialized.

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

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

6. 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: `AMP::Server::GetInstance().SetConnectionWaitTime.`

5 Adding AMP Support

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

5.1 Debug File Generation

AMP has the ability to use ActionScript 2 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.

For ActionScript 3, there are no SWD files, and debug information is embedded directly into the SWF. AS files are needed to display the source code. Generate debug SWF files by running the debugger from within Flash Studio.

5.2 Application Control

AMP has the ability to remotely control settings in the profiled application, such as wireframe mode, or batch profiling 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 `AMP::AppControlInterface`, and override the `HandleAmpRequest` method to handle `AMP::MessageAppControl` messages from AMP.
- Install the custom handler by calling `AMP::Server::GetInstance().SetAppControlCallback` on startup.
- Inform AMP of the supported functionality by calling `AMP::Server::GetInstance().SetAppControlCaps`. The argument to this method is a `AMP::MessageAppControl` message, with the supported functionality set to true. For example:

```
AMP::MessageAppControl 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.SetToggleBatch(true);
        caps.SetToggleStrokeType(true);
        caps.SetTogglePause(true);
        caps.SetToggleWireframe(true);
        AMP::Server::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 `AMP::Server::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:

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

5.4 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(1)
```

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/AS2/CLIK directory.

Add a marker in C++ as follows:

```
GFx::MovieImpl:: AdvanceStats->AddMarker(1)
```

AMP will then display a marker on the frame where the call was made. The integer argument will be used in the future to display more than one type of marker.

6 FAQs Related to AMP

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

1. **What are the bare minimums I need game side to have AMP working?**

You need an AMP-enabled build (i.e. with GFX_AMP_SERVER defined). By default, all non-shipping builds of GFx have AMP enabled. Also, if your game does not call Platform::RenderThread::drawFrame, you need to make sure AMP is updated every frame, by calling AmpServer::GetInstance().AdvanceFrame

2. **When we used AmpClient to connect our game, the frame rate of our game became very slow. We found AmpServer::AdvanceFrame() takes a lot of time. Is this normal?**

As with most profilers, it is normal for there to be a performance penalty when connected to AMP. In order to minimize this performance impact, please make sure you start at a low profiling level, which can be set by selecting from the appropriate dropdown control on the AMP client toolbar. Then, once you have located the bottleneck, you may increase the profiling level at the bottleneck to determine the cause in greater detail. Requesting full memory reports at each frame may also have a significant performance impact, so please make sure these reports are produced only when you need to see them. You may toggle detailed memory reports on and off via the "i" (lowercase I) on the AMP client toolbar.

3. **I'm running the game on Windows/PS3/Xbox, and I'm able to connect with the AMP client after it finds it in the discovery window. However, all the stats are blank.**

AMP needs to receive information every frame in order to display memory and performance statistics. If your game does not call Platform::RenderThread::drawFrame, you need to make sure AMP is updated every frame, by calling AmpServer::GetInstance().AdvanceFrame

4. **Why is AMP pausing one second or more to update (AMP::Server::WaitForAmpConnection)?**

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 Scaleform.

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 GFx::AMP::Server::SetHeapLimit.

5. How much memory overhead can we expect when using the profiler?

By default, the AMP heap limit is 1MB, although this is not actually a hard limit, so it can be exceeded by a little. You can change this limit in your game by calling AmpServer::SetHeapLimit. The greater the limit, the less likely it will be that GFx will have to be paused while data is being sent to the AMP client.

7 Additional Information

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

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