Adobe Scout – Getting Started Guide 0.7 – Prerelease 7

What's new in Prerelease 7

- **Adobe Monocle is now named Adobe® Scout.**
- **Memory limit is now global, not per session.**

What is Scout?

Scout is the next-generation profiler for Flash content running on desktop (**Flash Player**) and mobile devices (**Adobe AIR**). It allows developers to see things developers could not see in the past when profiling their Flash content.

Before Scout, ActionScript 3 developers using Flash Builder profiler could see what was happening at the ActionScript level, but the rest was not visible. Things like network, rendering, etc. were not exposed and could not be profiled easily out of the box.

How does Scout work?

Scout relies on the **Telemetry** feature introduced in Flash Player 11.4 and Adobe AIR 3.4. The Telemetry feature works deep inside the internals of the Flash runtime and sends data to Scout, which parses it and displays it in a clear and concise way.

The figure 1.1 illustrates the idea:

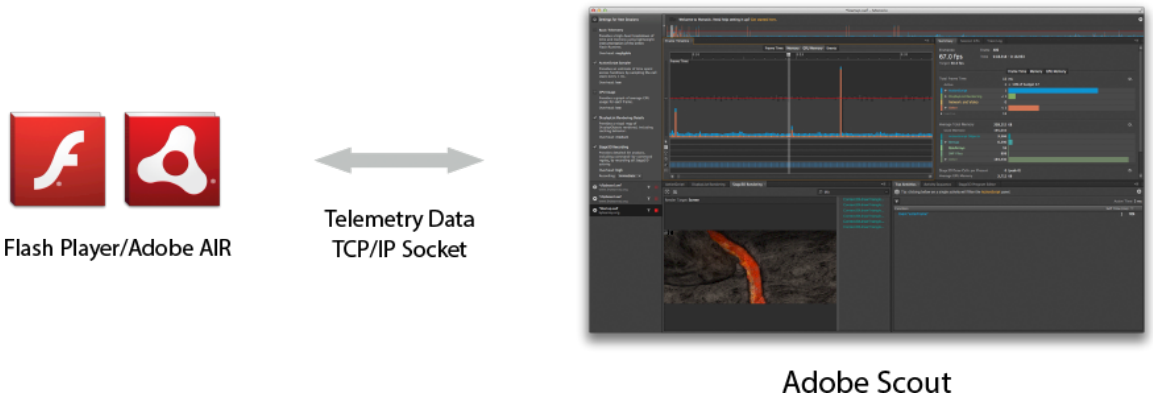

Figure 1.1
How Scout works.

The beauty of Scout is that it does not require the debugger versions of the Flash Player. This allows you to profile your content in the context of a page, even as a release build. This has the benefit of profiling your content at full speed rather than inside a debugger player also running slower than the release player.

Enable advanced Telemetry on SWFs

This is the first thing you need to know and do. By default SWFs will provide only limited metrics to Scout. To access all the advanced metrics like Stage3D or ActionScript, you need to enable *advanced telemetry* on your SWFs. This behavior was introduced to let developers decide if they want to expose all the advanced metrics to other developers using Scout.

A few options are possible to enable advanced Telemetry:

Using Flash Builder 4.7:

Recommended way, build your project using **Flash Builder 4.7 Game Development Beta 2** with Project Scout Support available here:
http://labs.adobe.com/technologies/flashbuilder4-7/

Flash Builder 4.7 comes with a new option in the compiler settings allowing you to enable Telemetry. This is useful for new and existing projects, just go to:

1. Your project Properties
2. ActionScript Compiler Tab
3. Then, check the *Enable detailed telemetry* option.

The figure 1.2 below illustrates the UI:

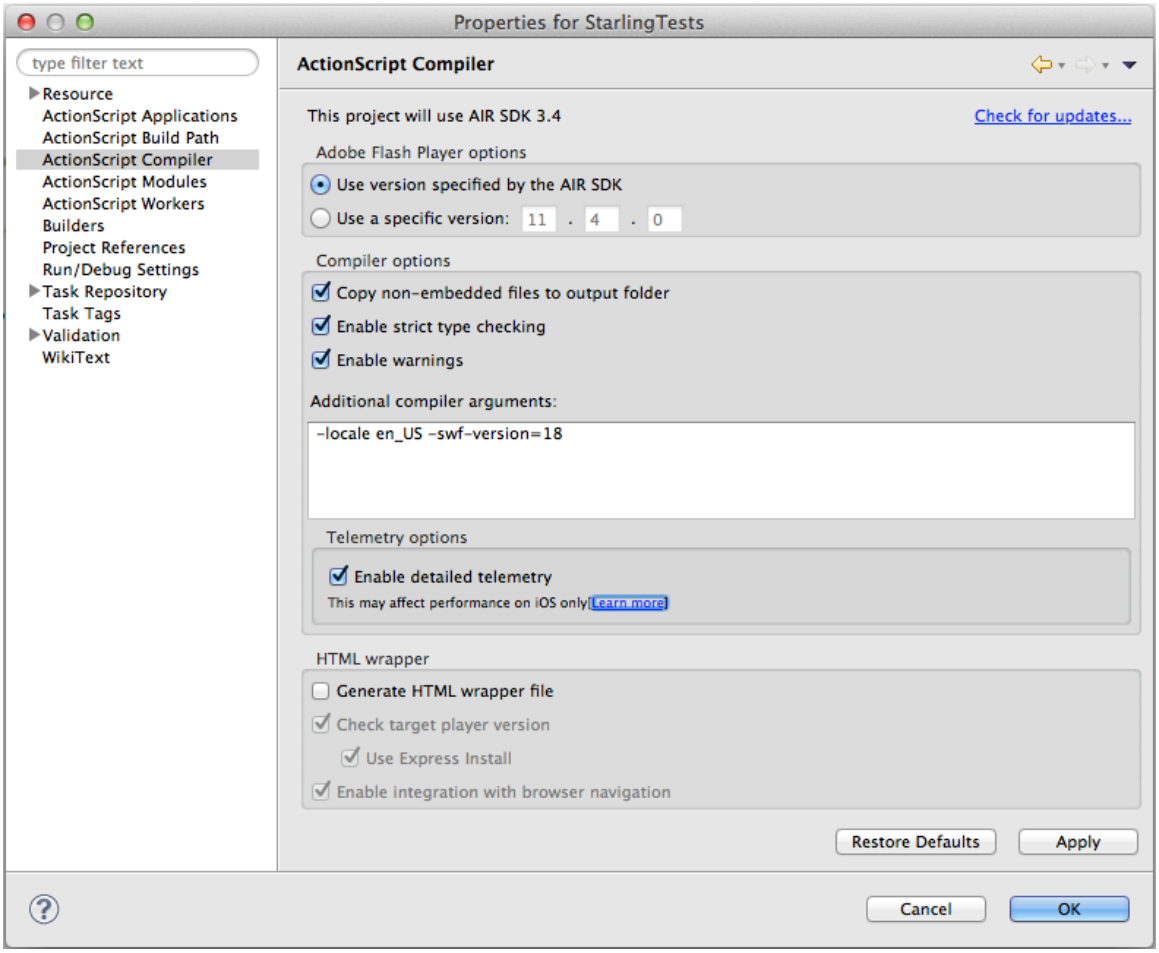

Figure 1.2
Enable detailed telemetry option in Flash Builder 4.7.

Once enabled, your content should automatically work with Scout. The figure below shows an example:

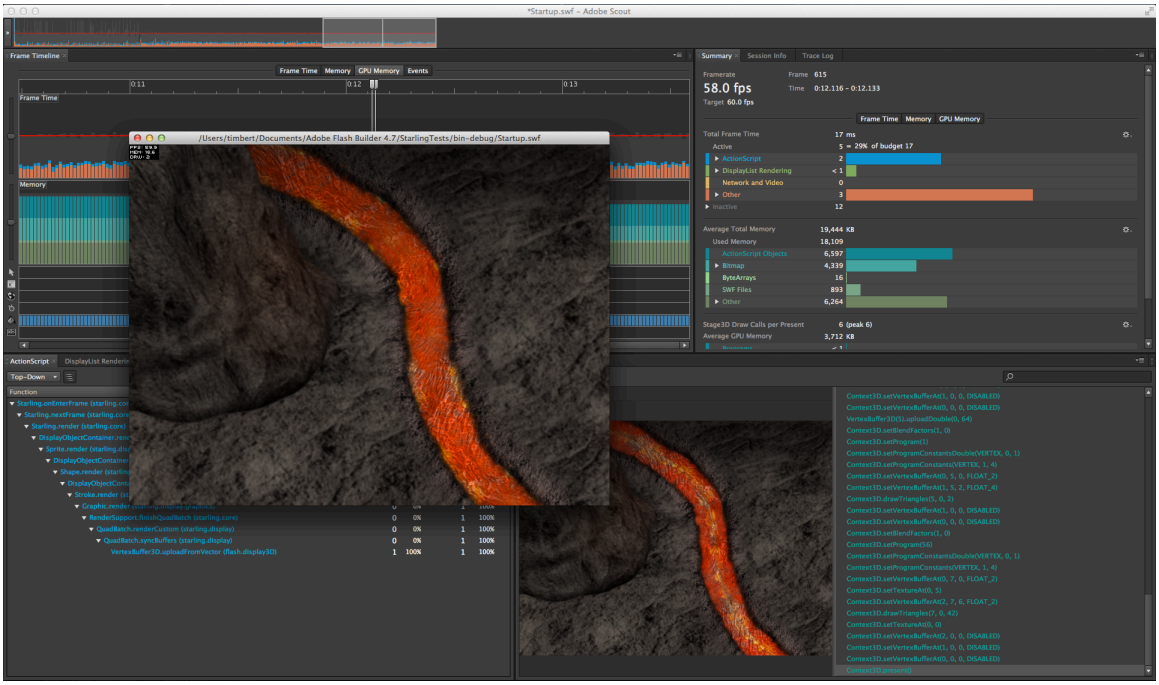

Figure 1.3
Scout profiling Starling content.

So before trying to run Scout on your mobile or desktop content, make sure your SWFs have advanced telemetry turned on.

Using Flash Professional:

If you are using Flash Professional, there is no UI like in Flash Builder 4.7. This will be added in an upcoming version of Flash Professional. In the mean time, we provide a little script in the Scout package (advanced-telemetry.py).

Go to python.org and install Python if necessary and run the add-advanced-telemetry.py script (provided in the package) on your SWF.

This post-processing approach can be also useful if you have existing SWFs you need to profile but don't necessarily want to recompile them.

How to configure Scout

To use Scout, the following components are required:

1. Flash Player 11.4 plugin or standalone or Adobe AIR 3.4.
2. Scout to present the profiling data.

On mobile devices, use the **companion applications** (iOS and Android). No config mobile files are created or needed, way simpler. Settings are read from the desktop configuration of Scout.

Note that if Scout gives you an error, there may be more information in the Scout log, which is located at:

1. **Windows**:
2. C:\Users\{user.name}\AppData\Roaming\Adobe\Scout\1.0\logs
3. **OSX**: /Users/<username>/Library/Preferences/Adobe/Scout/1.0/logs

Note: To previous users still relying on telemetry.cfg, check the Appendix 1 (at the end of this document) for more details.

Remote profiling

One powerful feature of Scout is called remote profiling. This allows you to run the content to be profiled on a different machine than the machine running Scout. A few scenarios can be imagined in the future:

1. A desktop machine is running the content (with Flash Player), with another computer running Scout. This can be useful to debug a remote project.

2. A mobile device is running the content (with Adobe AIR), with another computer running Scout. This is a typical workflow when developing for mobile devices.

You can direct Telemetry to any accessible machine by entering a hostname or IP address. So if the address of the computer running Scout is 10.0.1.1, add the following parameter to the .telemetry.cfg file:

TelemetryAddress=10.0.1.12

Start the content on the machine; the data is automatically sent to Scout, which is running on the other computer. Sweet!

AIR on mobile support

Scout supports profiling of Adobe AIR content on mobile devices through the Scout Companion App.

The Scout Companion App 1.0 allows you:

1. Link easily your mobile device with Scout.
2. Configure dynamically which metrics you are interested in.

The Scout Companion App for iOS is delivered through TestFlight, to have access to it, please enroll here: http://tflig.ht/LIVmSt

We will then receive your device IDs and distribute a build installable on your devices. Note: The Android version of the Scout Companion App will be made available soon.

Make sure your device is connected to the same Wifi network as your computer running Scout.

Once launched:

1. The companion App will search for computers having Scout running.
2. Once found, sync them.
3. Select the metrics you want to gather.
4. Launch your AIR mobile content.
5. Session should start in Scout.

Make sure you mobile device is connected to the same wifi as your computer running Scout, once the application is started on your mobile device, you should see automatically the profiling session start in Scout:

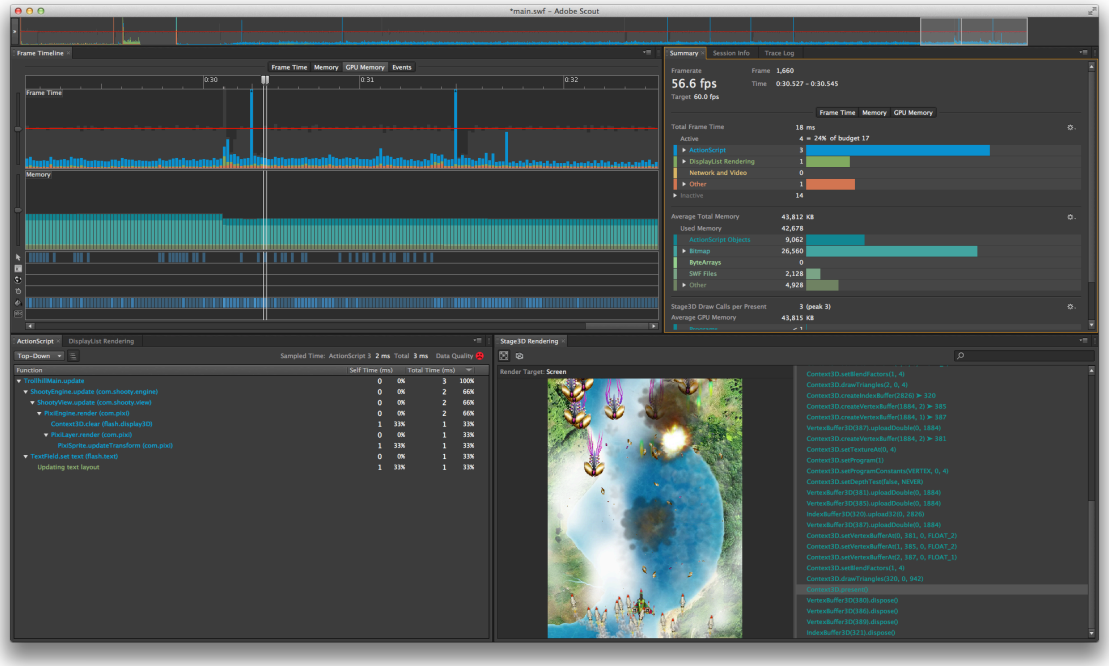

Figure 1.4
Scout profiling an AIR mobile game.

Start a session

Leave Scout running and start any Flash content on your machine, either inside a browser, through the standalone Flash Player or through Adobe AIR. If everything is configured correctly, profiling data should start to appear inside Scout.

Flash Player or Adobe AIR only checks for telemetry when starting Flash content, so make sure to load or reload the Flash content after you start Scout. During profiling, you can stop the session anytime by clicking the red stop button next to the SWF being profiled:

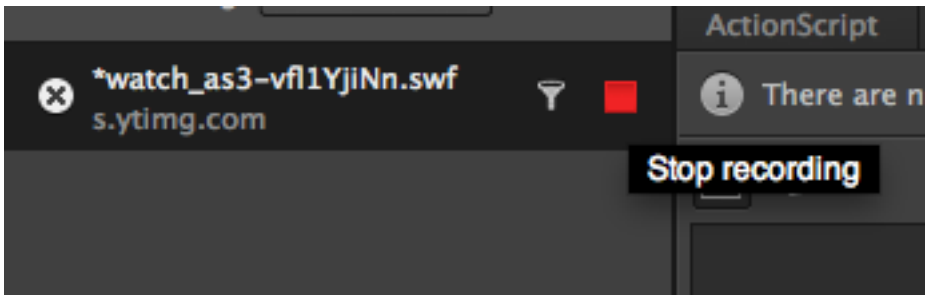

Figure 1.5
Disconnect button in Scout.

Note that we also introduced a filtering option, allowing:

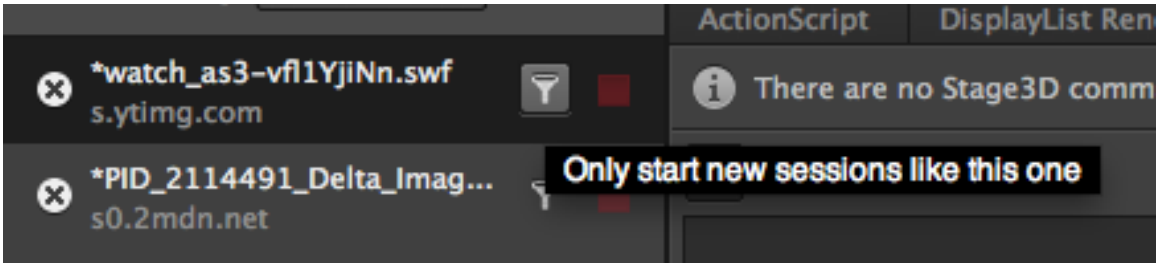

Figure 1.6
Filtering option.

A pretty neat feature of Scout is that you can also do remote profiling. Let's dig a little bit into this feature.

Scout Panels

The Scout interface is designed to be simple and efficient at displaying performance data. Scout's interface is made of 10 different panels illustrated below.

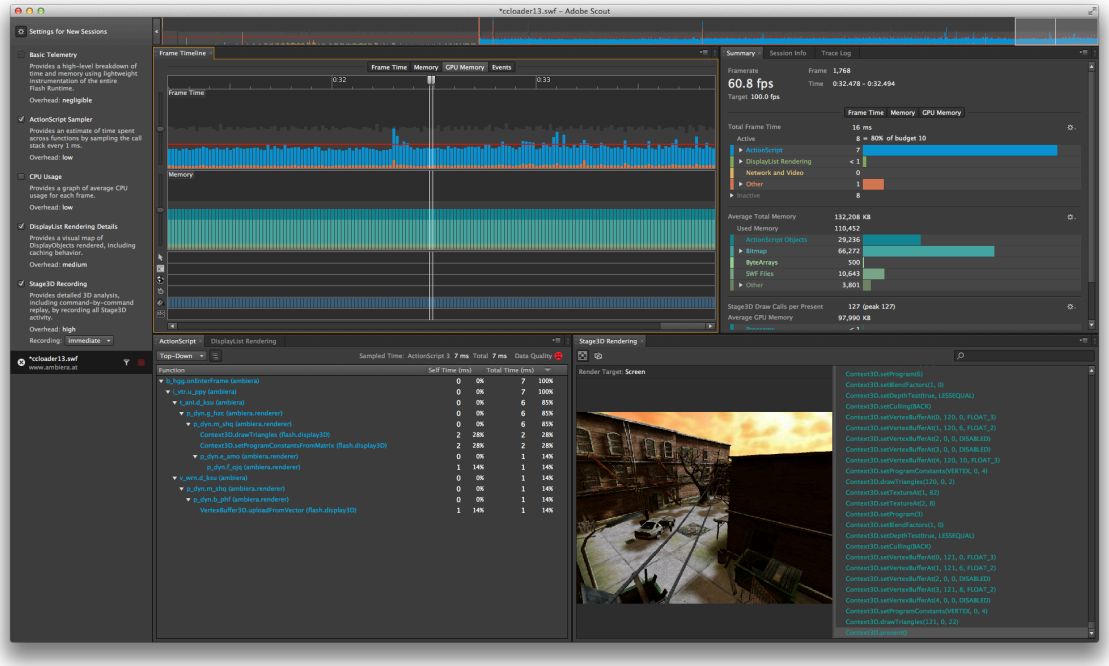

Figure 1.7
Scout's interface.

Each panel can be positioned the way you want, and workspaces can be also saved. (A workspace being a specific layout for the panels). The figure above illustrates a custom horizontal layout. Let's now take a closer look now at the different panels available in Scout.

Settings for new sessions

The first panel you will be working with is the Sidebar for settings for new sessions, giving you a way to specify which metrics you are interested in:

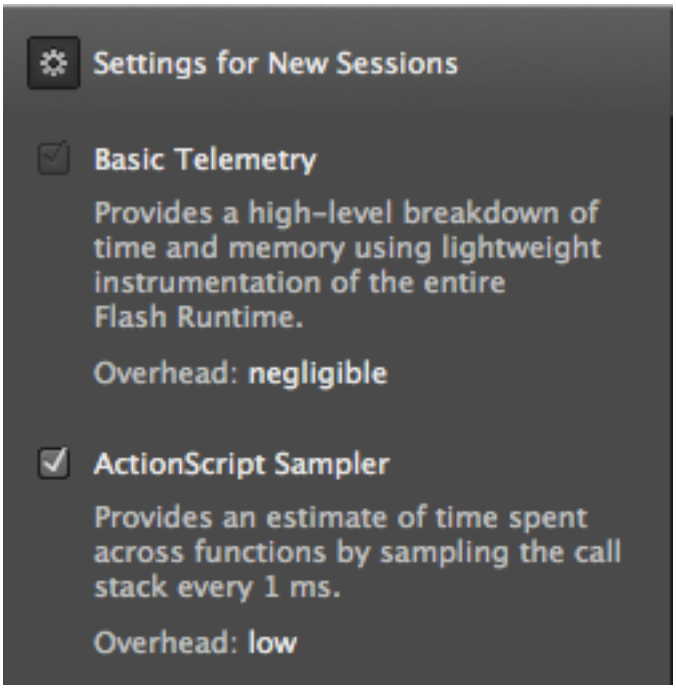

Figure 1.8
Session summary.

Note that SWF's will show as a separate a list in the same panel:

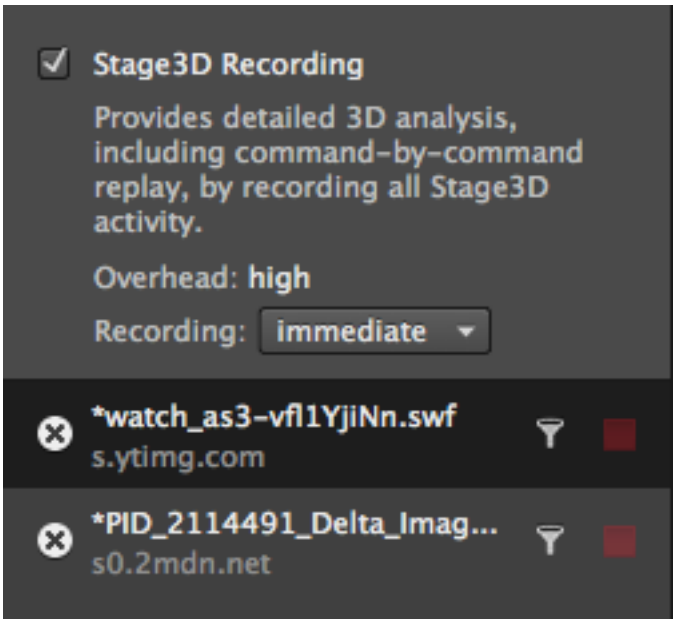

Figure 1.9
List of SWF's profiled.

By clicking on top right collapse button, the menu can be minimized:

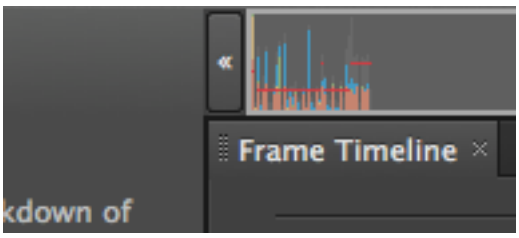

Figure 1.10
Collapse button.

Let's spend some time now on the Session summary.

Session summary

To jump to any frame, we can use the Session summary. This panel is located at the top taking the entire width of the tool. By right clicking on it, a few options are available:

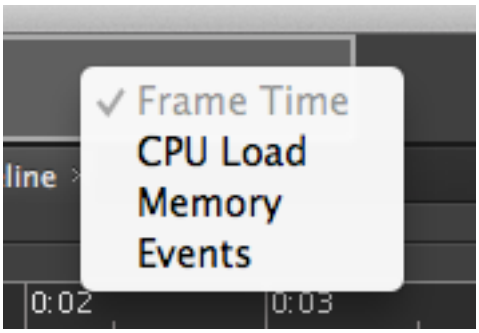

Figure 1.11
Session summary options.

By clicking on these options, you will be able to see the overall trend of the session:

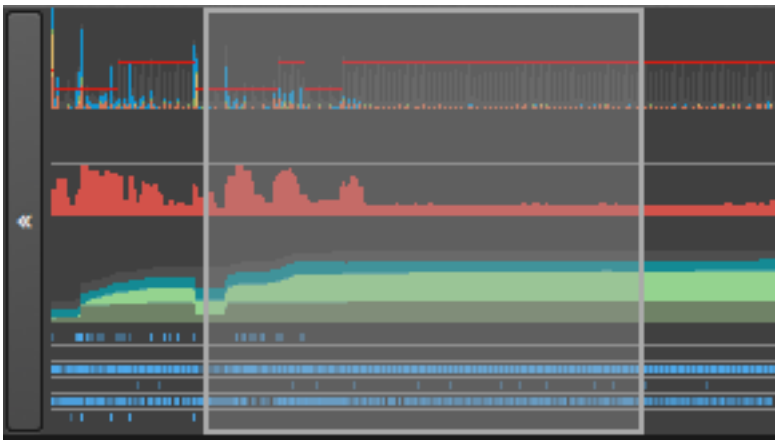

Figure 1.12
Session Summary (CPU Load, Memory and Events).

The Session Summary panel is very useful, it allows you to jump directly to any hotspot you may see.

Frame Timeline

The Frame Timeline panel is the main window in Scout to get into the details of what is happening in your application by selecting the frames that you are interested in. Each column is a different frame and the colors have different meanings:

1. Blue: ActionScript
2. Green: DisplayList Rendering
3. Orange: Network and Video
4. Dark orange: Other

The figure below illustrates the Frame Timeline panel:

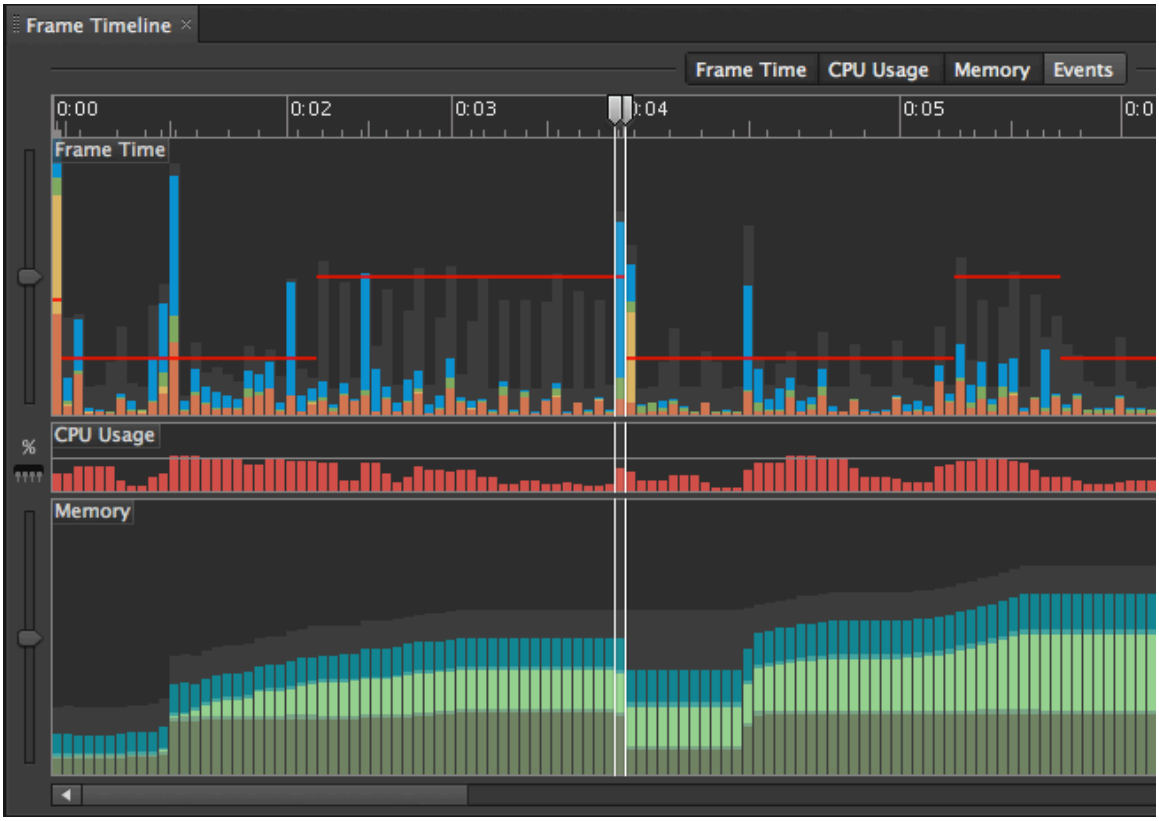

Figure 1.13

Frame Timeline panel.

Notice the red bar in this panel, informing you about the budget you have to spend on each frame. So what is the concept of budget?

It is really simple, it represents, the maximum time you have per frame to execute your operations. On a SWF running at 30fps, the budget you have is 33ms per frame (1000ms / 30fps) = around 33ms. If you spend more than 33ms per frame, your application will start stuttering and skip frames, which you do not want to happen!

Now, by looking at the figure 1.11, you can quickly see which frames are over budget and require attention. Note that the budget line is varying here, this is due to the fact that the framerate of the SWF is changed dynamically, hence why the budget changes.

Each bar in the graph is clickable and displays information about the data:

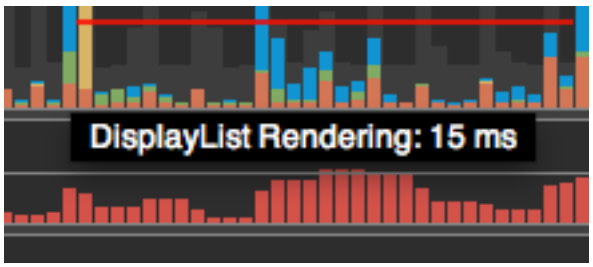

Figure 1.14
Tooltips in the Frame Timeline panel.

Below the graph, is a timeline of events:

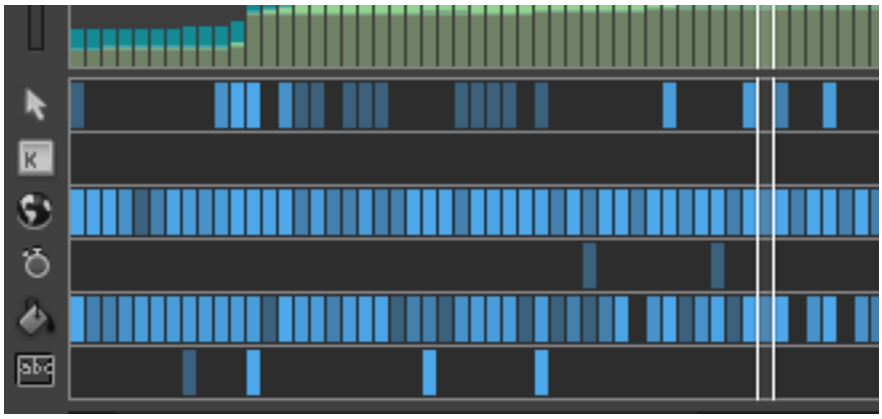

Figure 1.15
Events (Frame Timeline Panel).

Each horizontal line describes a specific event:

1. **Mouse**: Are there any mouse events dispatched and when?
2. **Keyboard**: Are there any keyboard events dispatched and when?
3. **Network**: Are there any file I/O operations happening and when?
4. **Timer**: Is the Timer class being used and when?
5. **Rendering**: Is there any rendering happening.
6. **Trace Events**: Trace calls.

As you can see, the Frame Timeline panel allows you to quickly find hotspots in your application and see where too much time is spent and why.

Summary

We saw previously the different colors and their meanings. The Summary panel (illustrated below) works with the Frame Timeline panel we just saw. Each frame selected in the Frame Timeline panel updates the Summary panel:

Figure 1.16
Summary panel.

The Summary panel provides details on specific frames selected in the Frame Timeline panel, by selecting multiple frames in the Frame Timeline, the Summary panel aggregates the data from all of the frames.

Very easily, you can see how much time ActionScript 3 takes to execute on this frame, how much time rendering took, but also advanced details about memory usage at this specific frame. We can see that for those 9 frames we are actually

spending 95% of the budget, which means we are almost over budget and we may want to Scout that.

The figure 1.15 below illustrates the Summary panel with 2 other frames selected in the Frame Timeline panel:

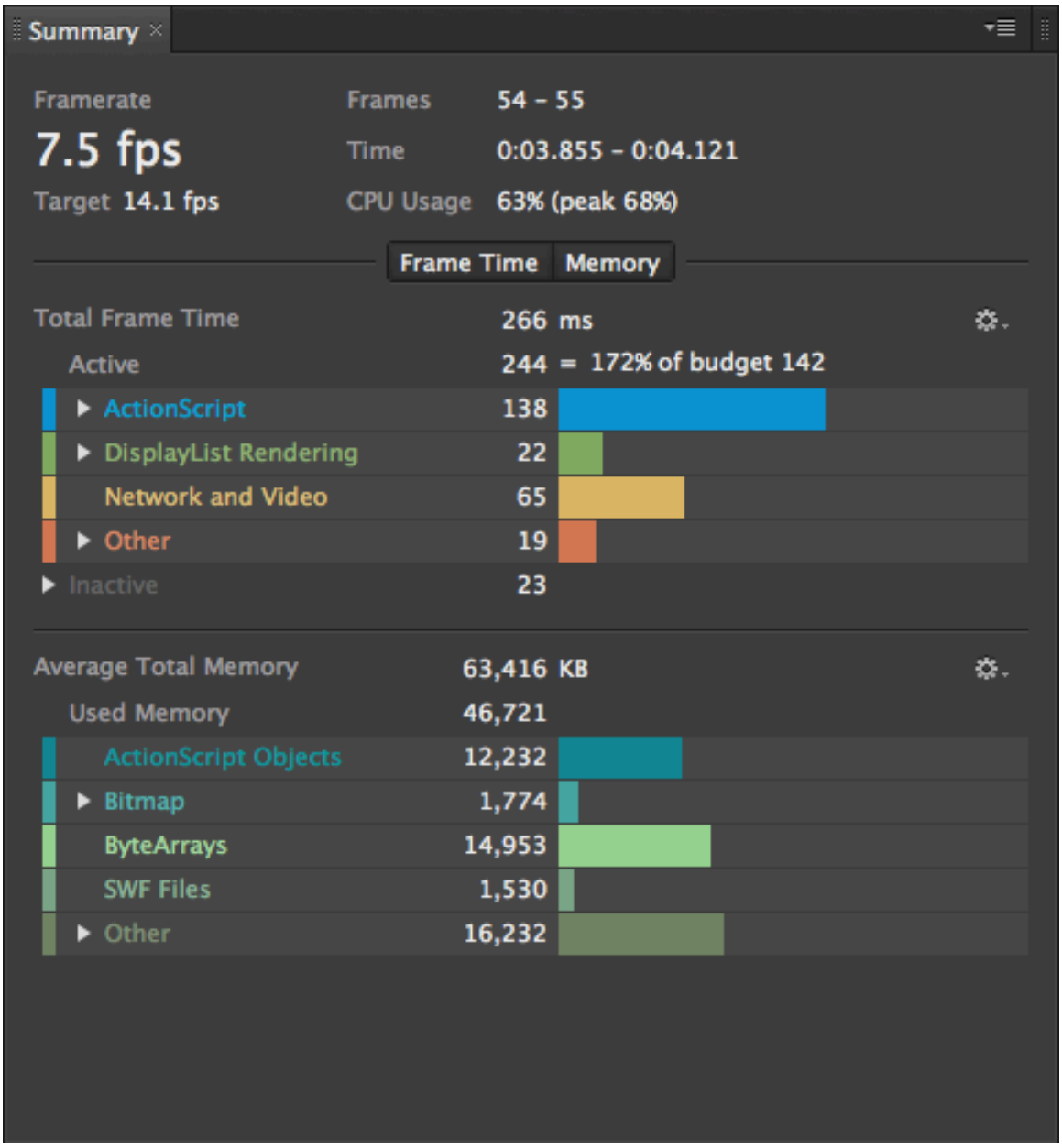

Figure 1.17
Summary panel.

We can see here that during those 2 frames, the average frame rate was 7.5fps out of a target frame rate of 14 and we are way over budget (172%). We can see that our

application does not perform smoothly and we may want to know why. The next Scout panels will help us to figure this out.

At any time, you can click on each category:

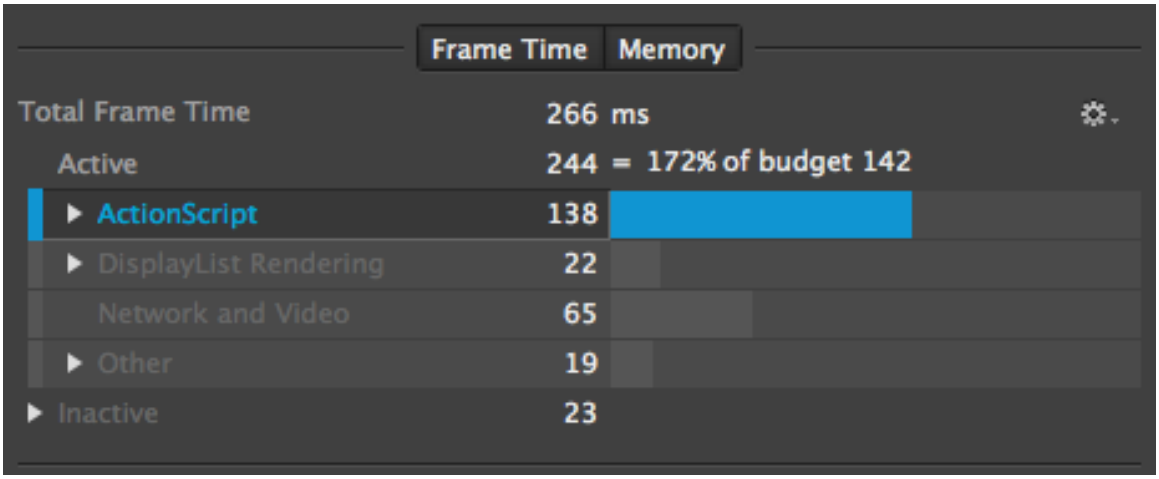

Figure 1.18
Summary panel filtering.

Automatically the Frame Timeline panel gets filtered to represent only the metrics selected. In our case, we selected ActionScript:

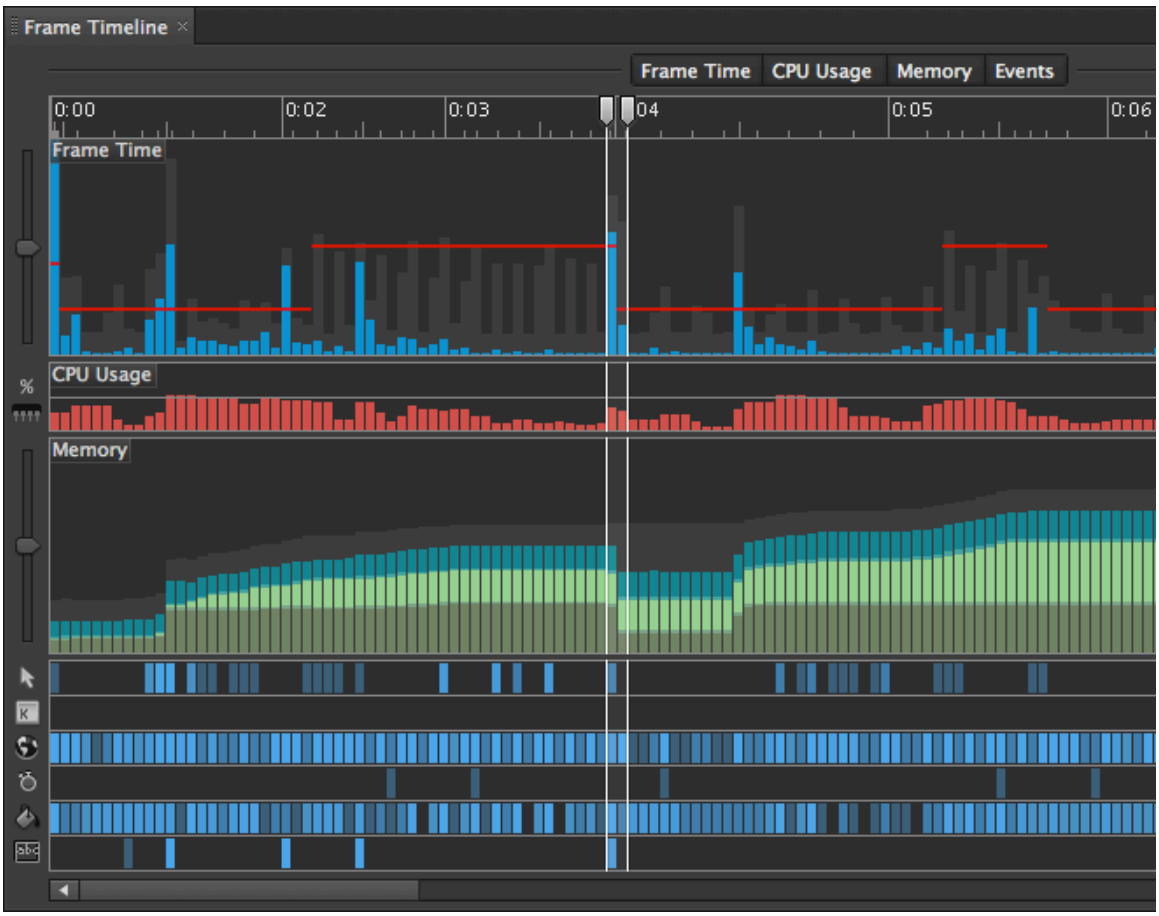

Figure 1.19
Frame Timeline panel filtered by Summary panel.

Notice now that we have an Average Total Memory section now giving more granular details on memory. The figure below illustrates that functionality:

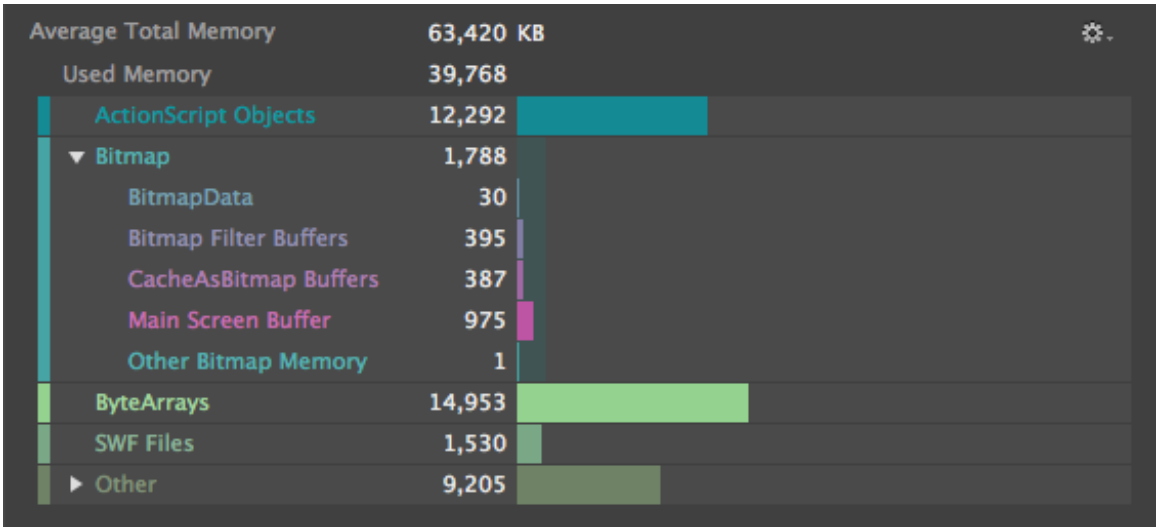

Figure 1.20
Summary panel filtered.

Let's have a look now at the Top Activities panel.

Top Activities

The Top Activities panel goes into lot more detail about what is happening inside Flash Player or Adobe AIR. In contrast, the previous panels give us details regarding the time spent on things like ActionScript, rendering and other general areas. The Top Activities goes one level deeper by telling us for instance, that for rendering, most time was spent rendering cached surfaces, rasterizing them or rendering text, and more…

The figure below illustrates the Top Activities Panel:

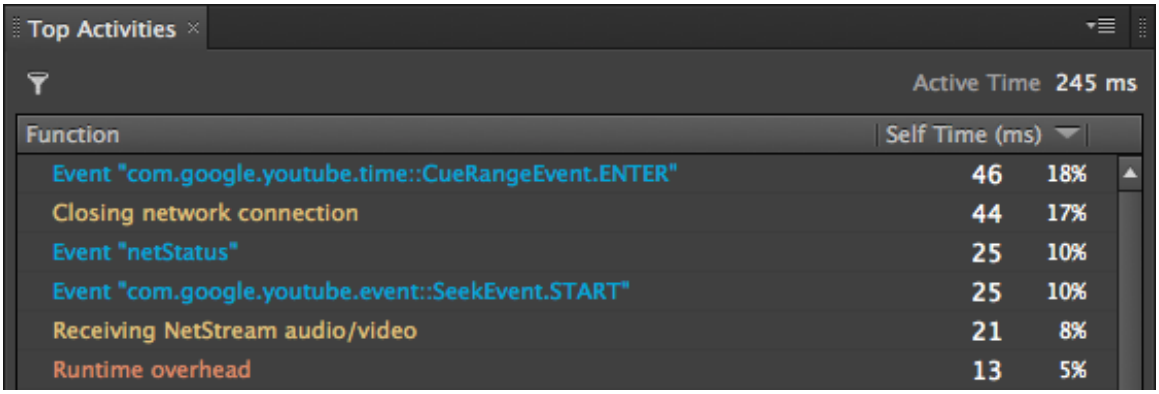

Figure 1.21
Top Activities panel.

On the right side of the panel, two columns are available indicating the time spent on each task and how much work it did, represented as a percentage. Of course, these numbers allow you to filter the data accordingly. In this example, we see that most of the time is spent (18% - 46ms) in the CueRangeEvent.ENTER event being dispatched.

By selecting one of the metric in the summary panel, the ActionScript sampler panel will be filtered automatically to show you which code is related to this metric. Let's click for instance on the first item, the CueRangeEvent.ENTER event, automatically the ActionScript sampler is filtered and shows only the code triggered by this event:

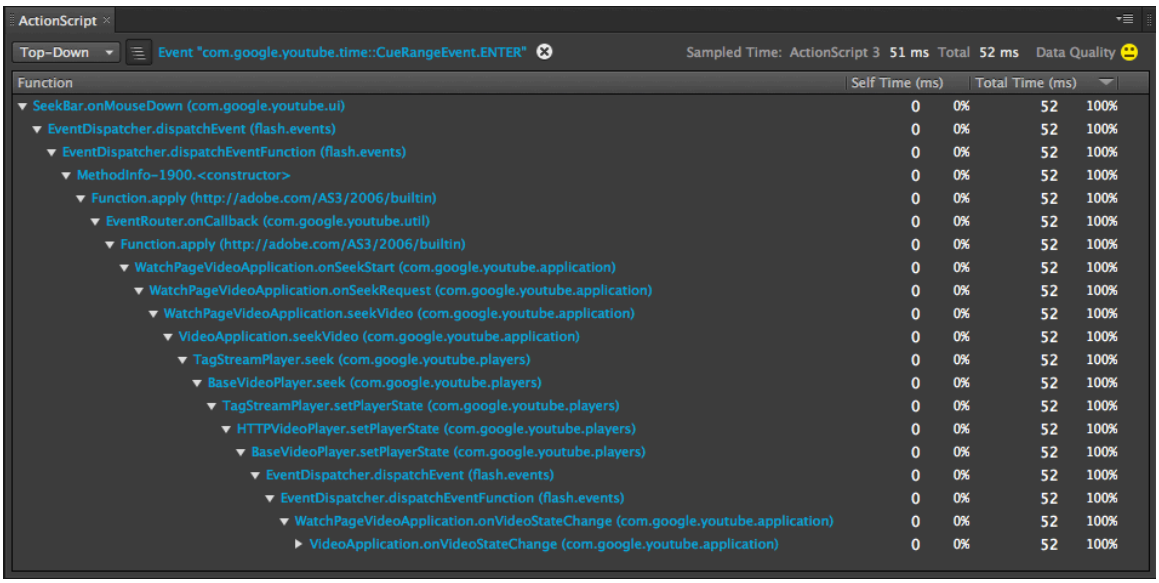

Figure 1.22
ActionScript sampler panel.

Note that the Top Activities panel also highlights very critical things like garbage collection. The figure below illustrates such a scenario:

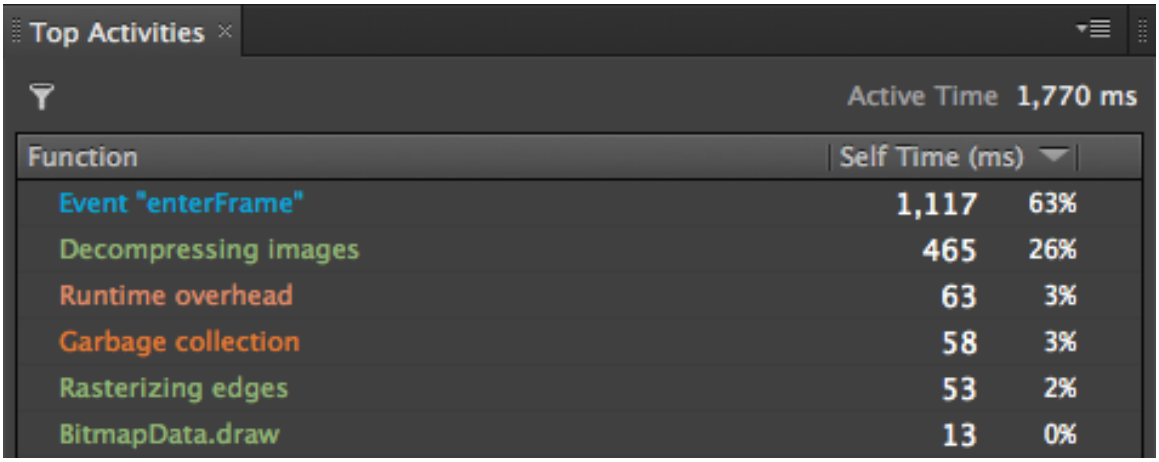

Figure 1.23
Garbage Collection triggered.

By clicking on the Garbage Collection metric, the ActionScript panel is automatically updated and filtered to highlight where exactly the collection got triggered:

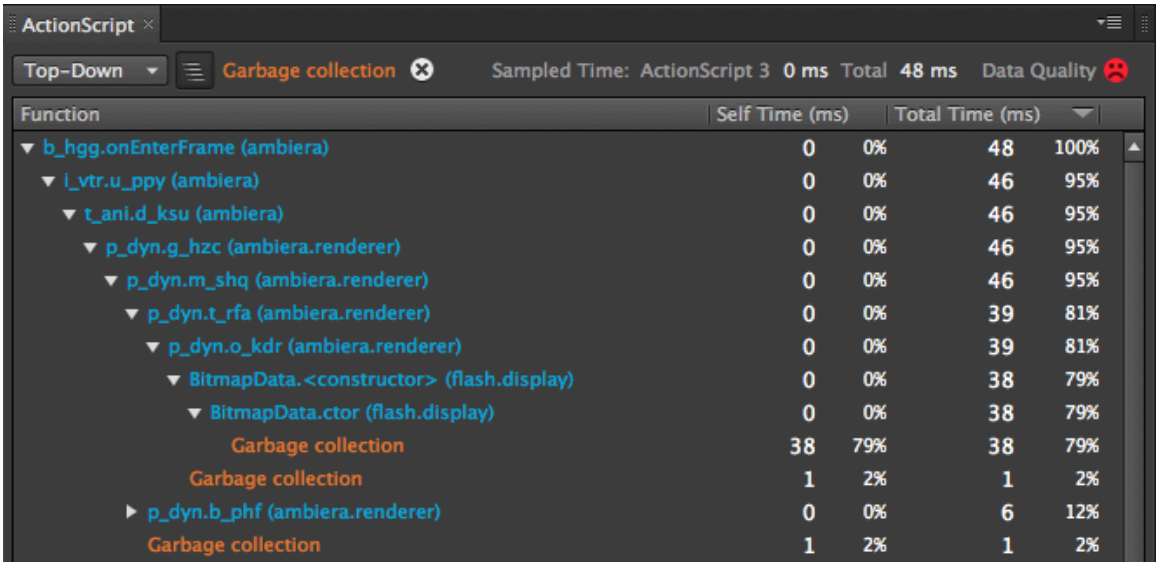

Figure 1.24
Garbage Collection triggered.

Did you notice the Garbage Collection metric inlined here? This tells you which function triggered the garbage collector. Pretty cool right?

Let's have a look now at the Activity Sequence panel, slightly different from the Top Activities panel.

Activity Sequence

The Activity Sequence panel describes what happened for the frame specified. Note that in contrast to the Summary or Top Activities panels, the Activity Sequence panel only describes the data in a single frame, not multiple ones.

In the figure below you can see an example of what this panel can display for our frame 45:

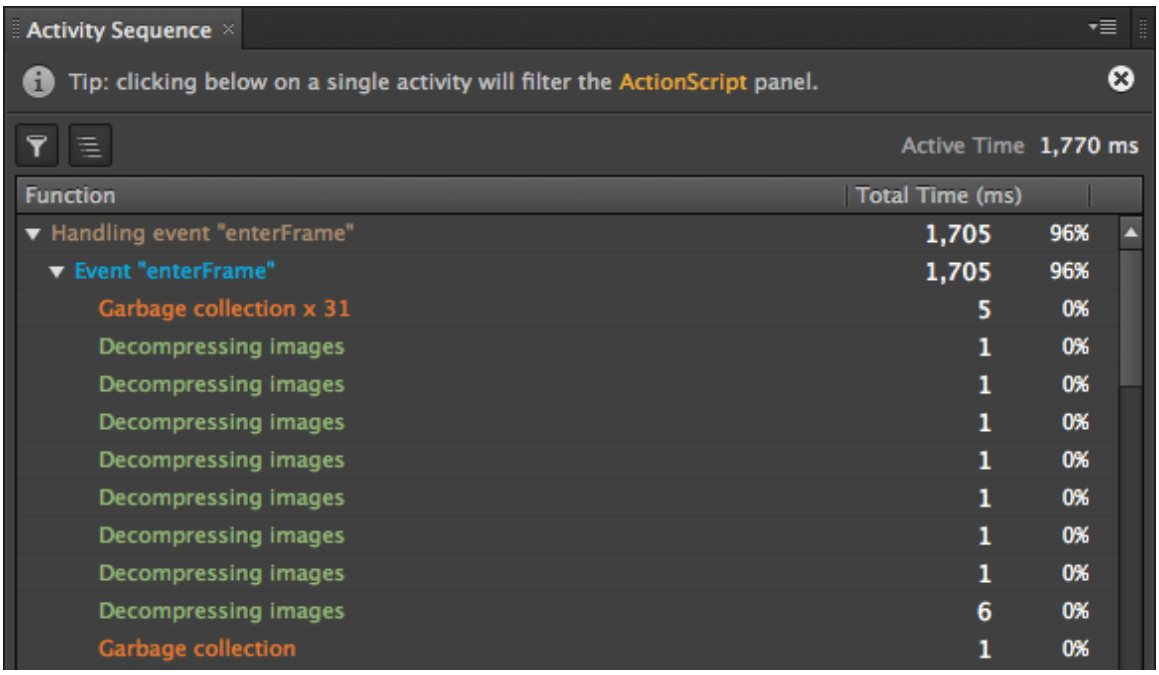

Figure 1.25
Activity Sequence panel.

The data displayed here is much more granular than in the previous panels we saw. And rather than listing the top activities, it shows each individual activity in time order for that frame whereas top activities aggregates the times and lets you sort by name or time. Note that through all the panels, the color theme is consistent, helping you to easily focus on the area you are interested in: ActionScript, rendering, video/network, etc.

For example, in this scenario, we see that most of the time is spent in the logic triggered by the *"enterFrame"* event.

ActionScript

With another frame selected, here is what the ActionScript panel exposes:

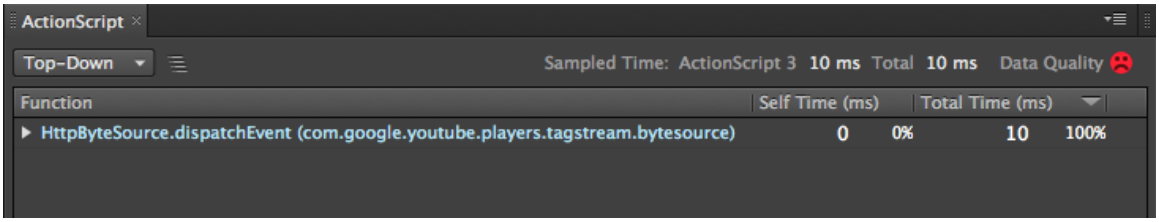

Figure 1.26
ActionScript panel.

Not much right? Well, make sure you expand all, by using the expand all button on the left. Once expanded, we have much more details:

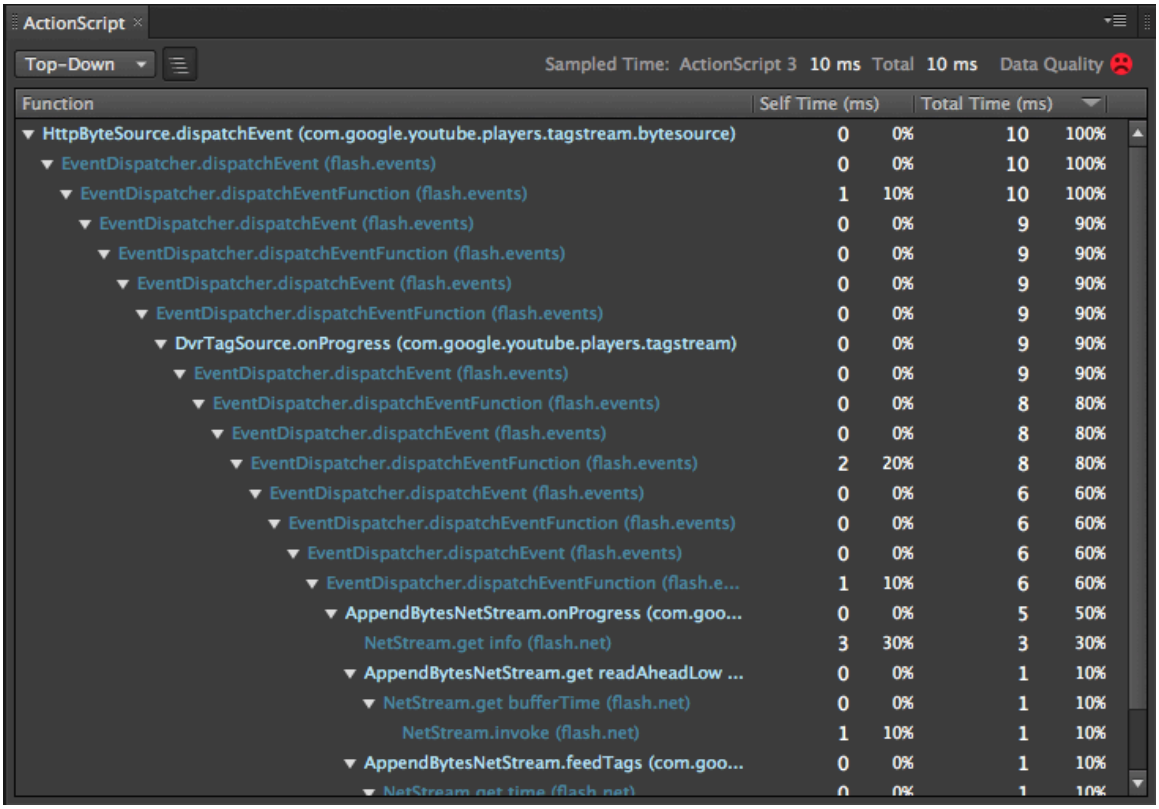

Figure 1.27
ActionScript panel (Expand-all option selected).

Note the slight color difference between native calls (dark blue) and custom ActionScript 3 code (light blue), this proves to be really useful to easily differentiate between the two.

By default, we have a bottom-up view, where the top parent node is the underlying native call, and we see in order which native calls are triggered and the time it took to execute each of them, really useful.

Let's scroll a little bit to see which native API we are calling see if time is spent there:

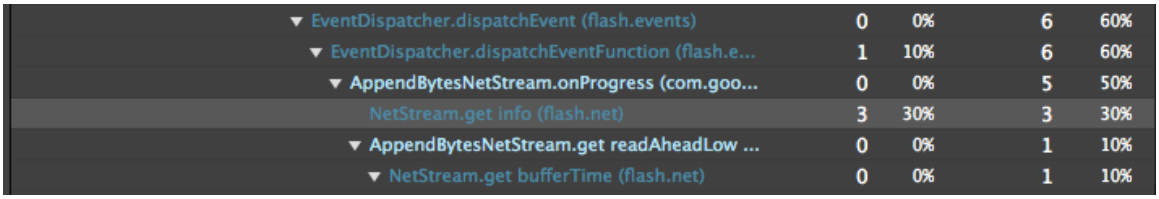

Figure 1.28
ActionScript panel detailing time spent on *NetStream*.

Nice! We see that the *getter* call is actually the most expensive here, we may want to change the frequency of calls or for instance change how our application is architected to reduce the number of calls.

In case you want to get straight to the point, and see a reversed order, just go in the options of the panel and choose the bottom-up view.

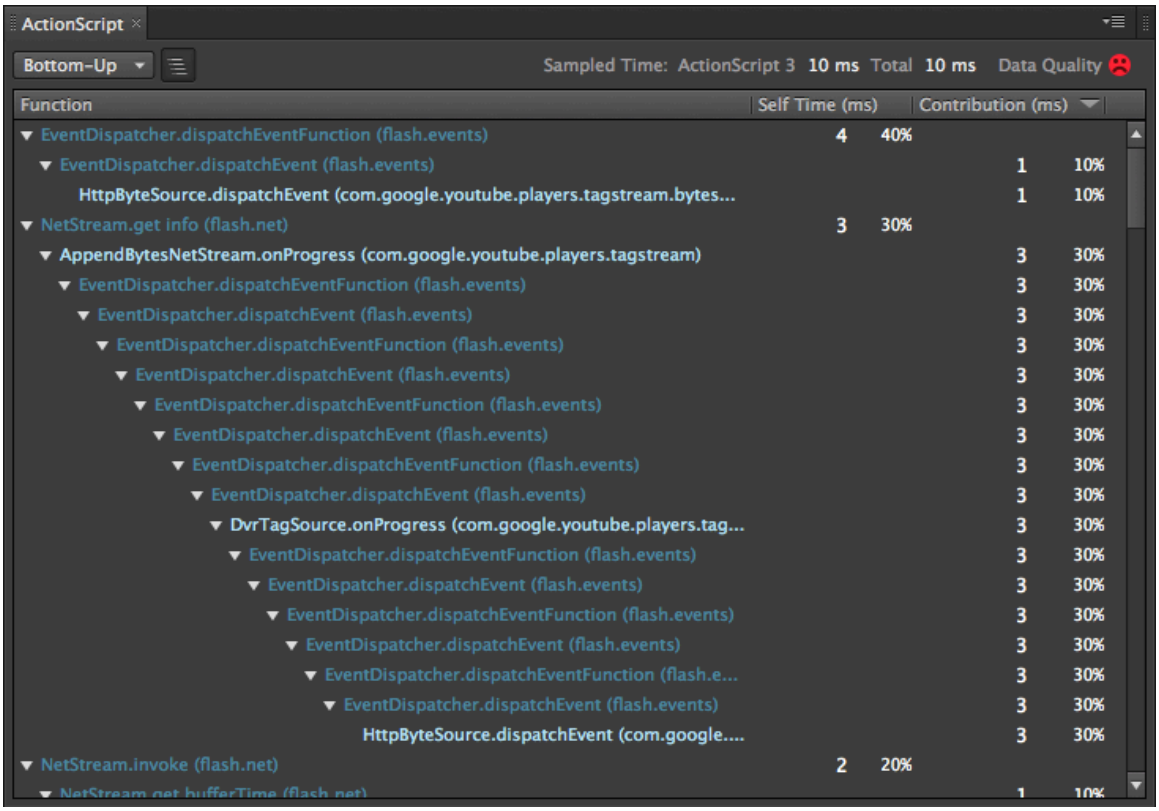

Figure 1.29
ActionScript panel (Bottom-up option selected).

Very easily we can see which native calls are being called at the end of the execution flow. This allows you to detect which native calls are expensive and how to workaround any potential slowdown cause by them.

Trace logs

A new very useful panel has been added, to log the trace calls. If you look at the figure 1.13, you will see an event for traces. By just clicking on a frame with a trace call, the content will be output in the Trace Logs panel:

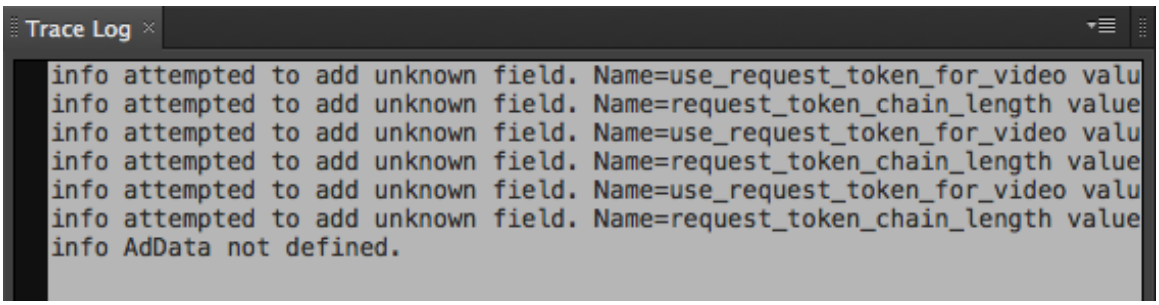

Figure 1.30
Trace Logs panel.

We just covered lots of things related to ActionScript, memory but we did not really spend time on rendering profiling. Let's fix that by jumping to the next section!

DisplayList Rendering

The DisplayList Rendering panel is very powerful and goes into very low-level details. It exposes the regions being redrawn when using the Display List (CPU based).

The figure below illustrates the panel in action with the Heat Map feature:

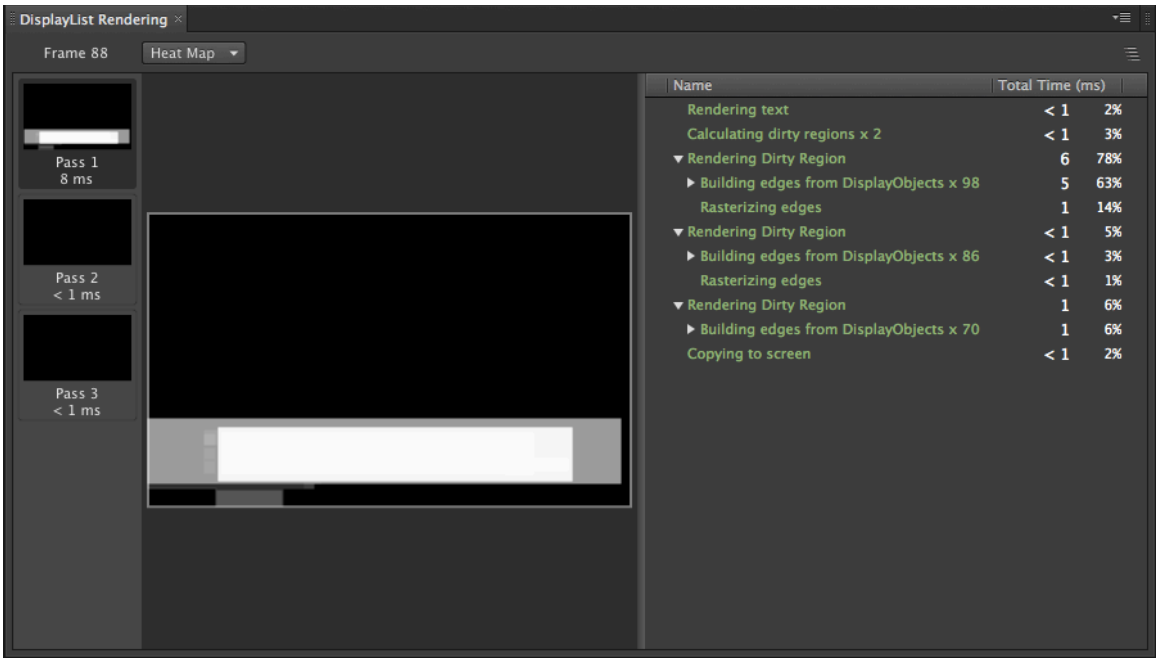

Figure 1.31
DisplayList Rendering panel (Heat Map).

The Heat Map feature allows you to select multiple frames and get an idea of where you are spending most of your time rendering things on screen. Very useful.

Like any other Scout panel, the DisplayList Rendering panel works with any frame selected from the Frame Timeline. In this example we are profiling the YouTube video player, we can see which areas are being computed by the Flash Player, and you can clearly see the most actively rendered areas caused by the video player controls and ads.

We just saw the Heat Map feature, we can also get specific details on what is being rendered and what is it and how much time it takes. Let's switch to the Regions mode, the panel now looks like the following:

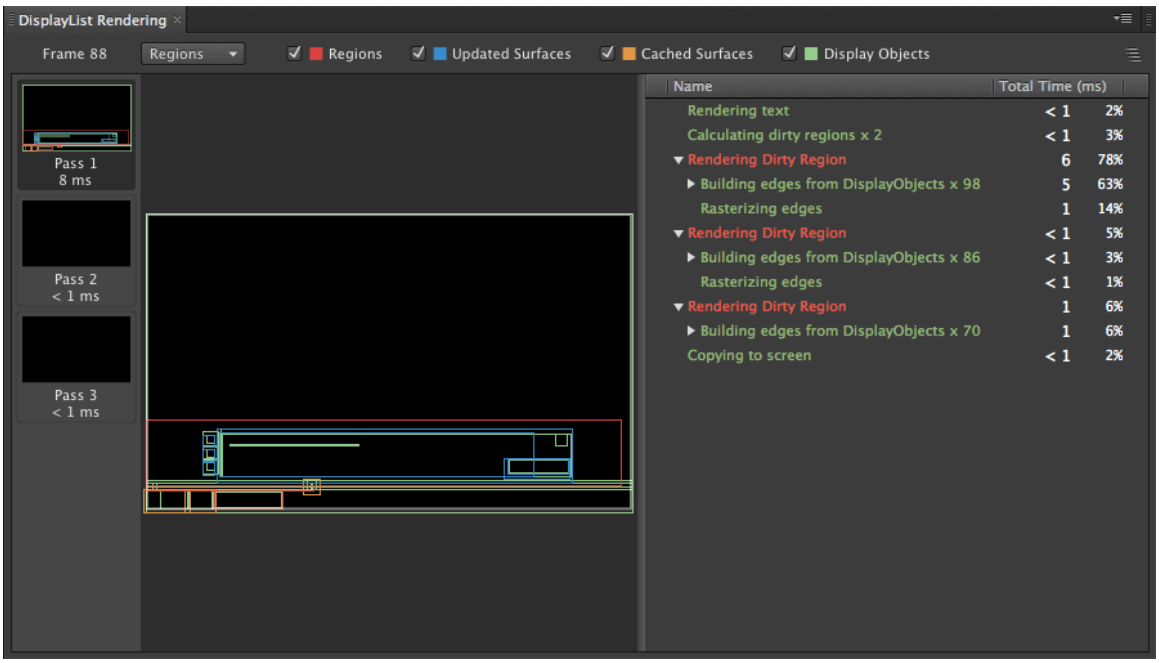

Figure 1.32
DisplayList Rendering panel (Regions).

At the top of the panel we can see the following four categories:

- Regions (red): Same behavior as the DisplayList Rendering feature from the Flash Player (debugger). A redrawn region is actually the bounding box containing the pixels being redrawn, in other words, the recomputed region.
- Updated Surface (blue): Surfaces are bitmaps created internally by the Flash Player when using things like filters, blend modes, bitmap caching and of course bitmaps. This represent the updated surfaces.
- Cached Surface(yellow): This represent the cached surfaces, being reused over frames.
- Display Objects (green): Vector content.

Also, by clicking in a specific area, a graphical element can be selected. Automatically, details about the underlying internals and how much time it took for the Flash Player to render that element will be provided. The figure below illustrates the selection in action:

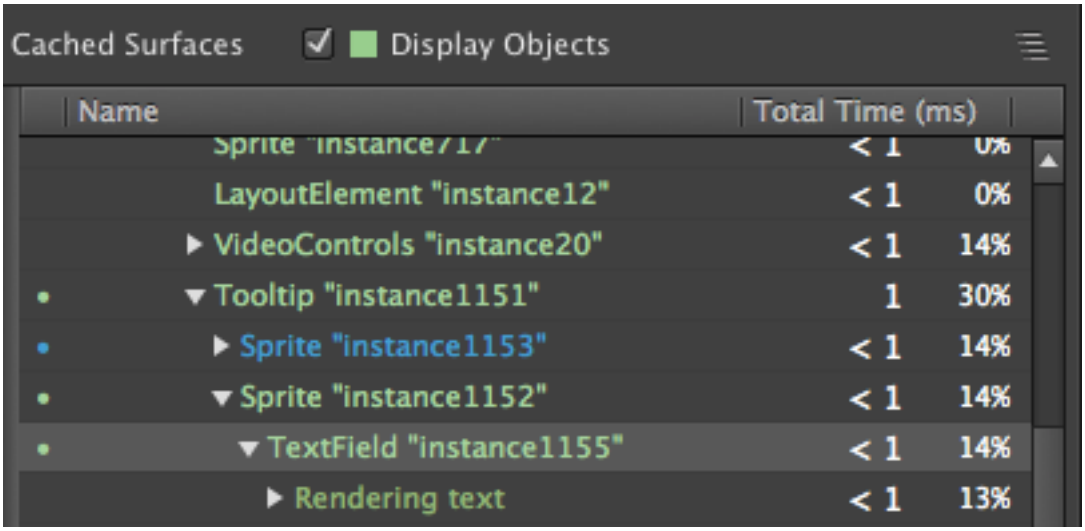

Figure 1.33
Graphical object selected.

That's it for the CPU rasterizer, now two other panels will delight GPU developers. Let's have a look at these.

Stage3D Commands

When producing GPU accelerated content, you will be using the Stage3D APIs. Stage3D allows you to leverage the GPU horsepower to accelerate rendering on desktop with Adobe AIR (3.0 and above) and Flash Player (11 and above) and on mobile devices through Adobe AIR (3.2 and above).

When using Stage3D, all calls done on the Context3D object can be capture by Scout through the Stage3D commands, the figure below illustrates the panel:

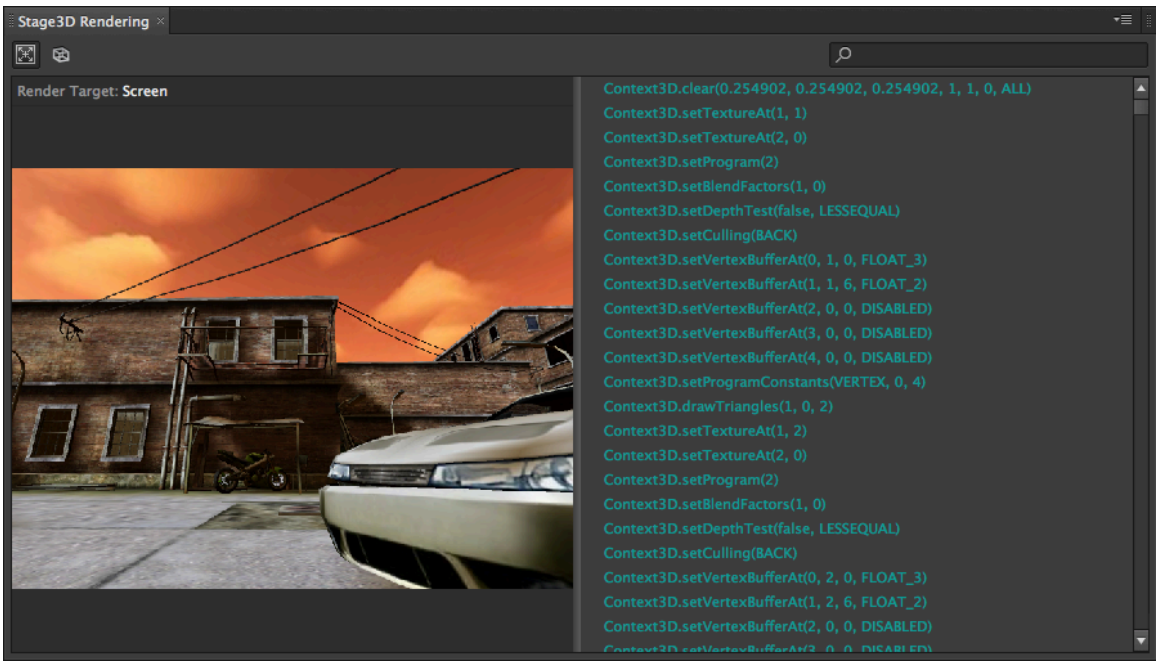

Figure 1.34
Stage3D Commands panel.

By just selecting each draw call (drawTriangles) in the list, we can reissue each draw call and reconstruct the scene step by step. The figure below illustrates the scene half rendered by stepping through the draw calls manually:

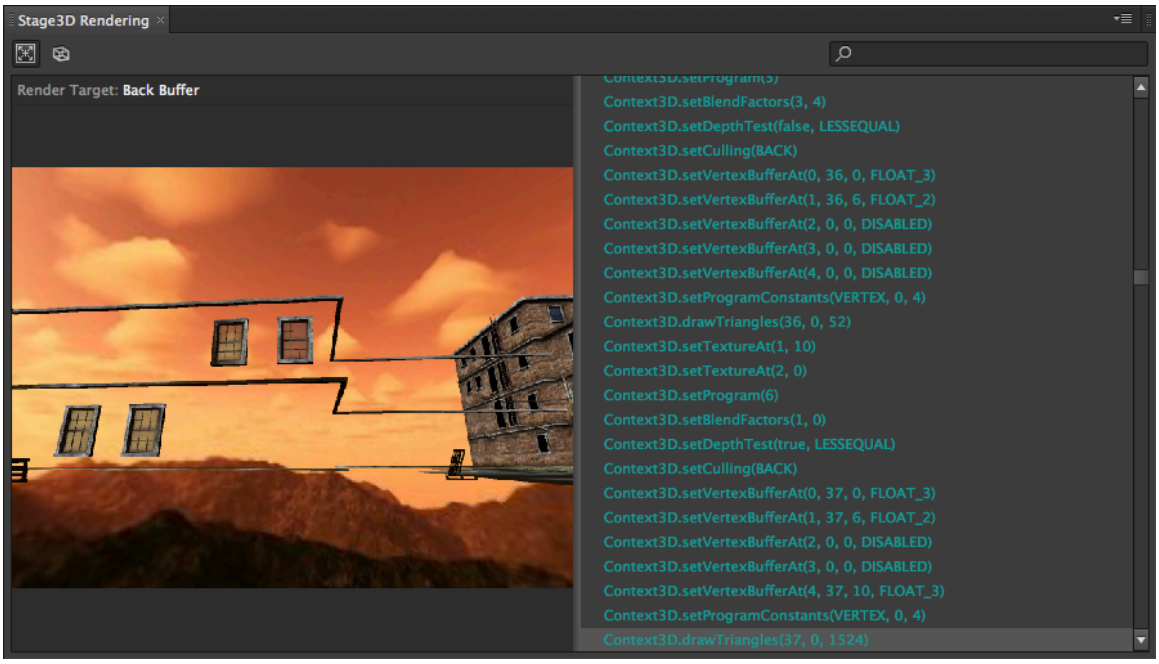

Figure 1.35
Stepping through draw calls.

Note that to automatically jump to the next drawTriangles call, just press SPACE.

You can then recompose any frame of your choice and study Stage3D commands issued and identify possible optimizations. You can also select multiple frames from the Frame Timeline and go through each frame of your choice from inside Scout, exciting!

Note that the panel also informs you which render target is being utilized for this draw call, in this case illustrates by the figure 1.33, it is obviously the back buffer.

So, how does it work?

We have isolated the Stage3D engine inside Scout, so we are not sending bitmap images, video or doing any other trick. The Stage3D engine is running inside Scout and re-rendering your scene live, with perfect accuracy.

Also by clicking on a frame in the Frame timeline, we get details on the resources being utilized through the Summary panel:

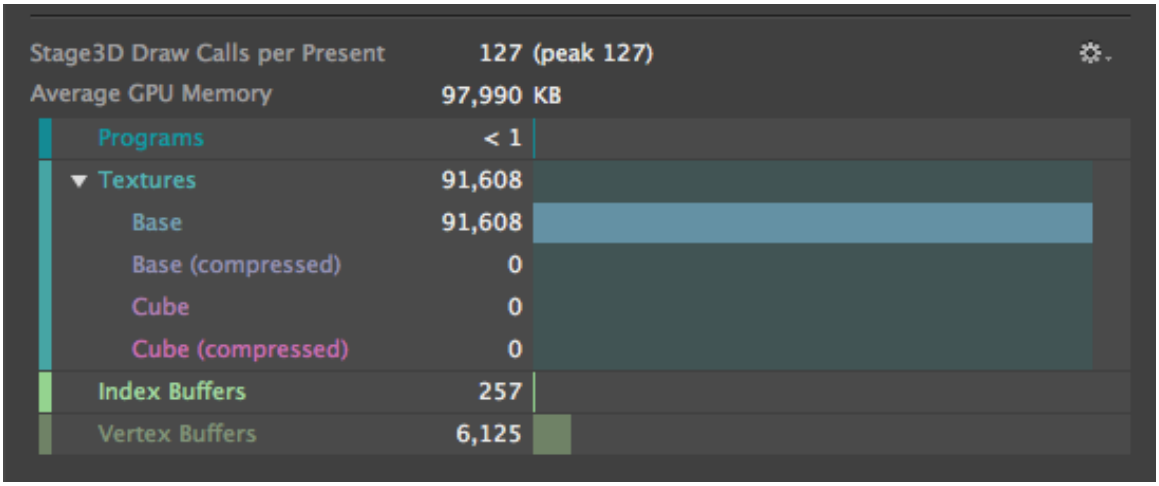

Figure 1.36
GPU memory usage per frame.

But, wait there is more!

Wireframe mode

The Stage3D Rendering panel also integrates a nice feature, the wireframe mode. This allows you to preview the scene in wireframe to see the triangles untextured to look through objects:

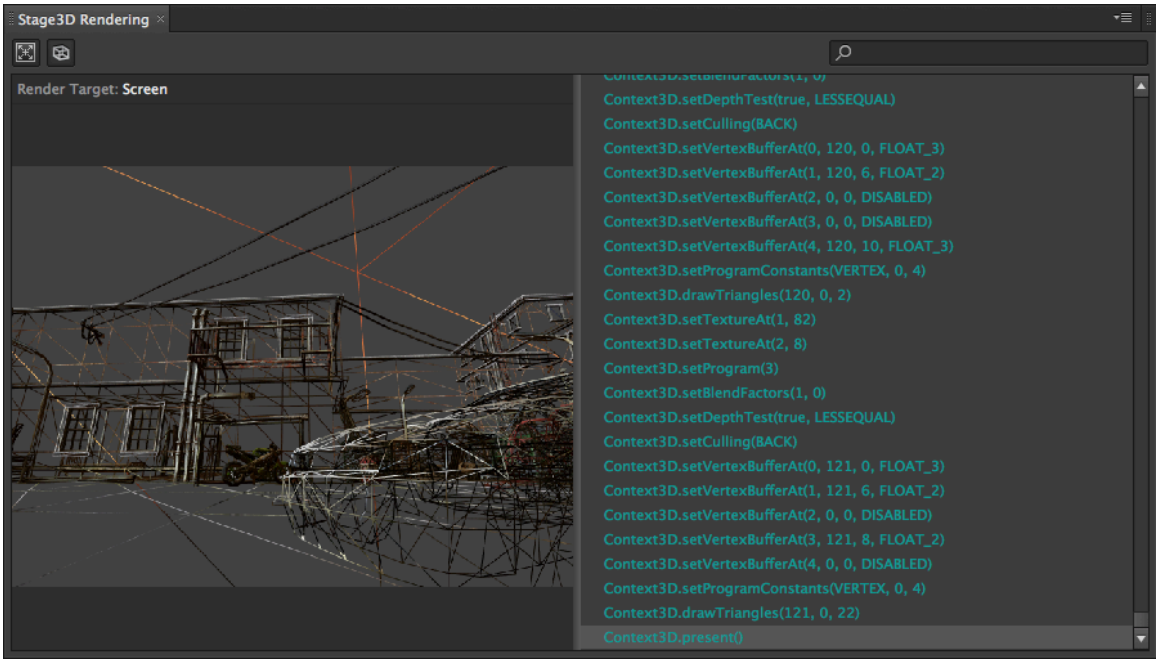

Figure 1.37
Stage3D wireframe option.

This feature can be useful to see which objects are covering others, and see if some drawing calls could have been avoided. Of course, this behavior is preserved as you select multiple frames in the Frame Timeline.

Stage3D Program Editor

Now here is something you are going to love, the Stage3D Program Editor panel. As you may know,each draw call (*drawTriangles*) has shaders related to, dictating how the geometry is placed and how the pixels are colored. By selecting any *drawTriangles* command like in the figure below:

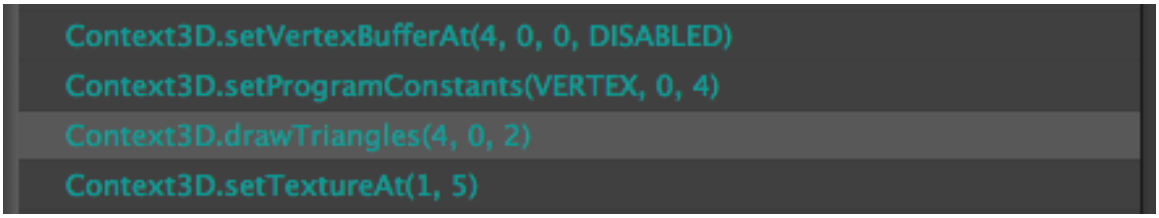

Figure 1.38
Stage3D Commands panel, with *drawTriangles* call selected.

The Stage3D Program Editor panel now exposes the shader program (vertex and fragment) associated to each draw call:

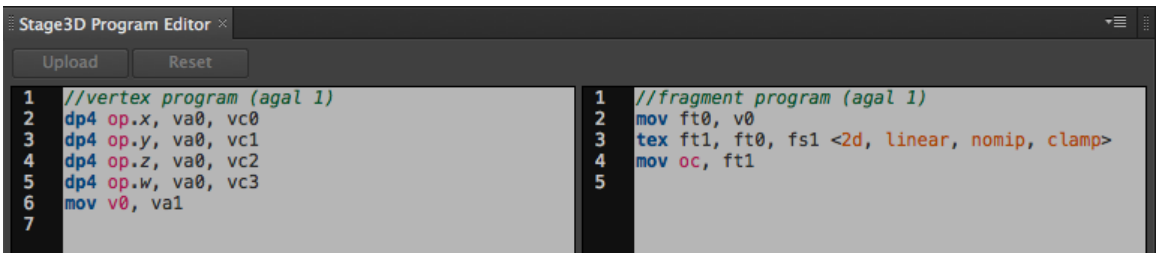

Figure 1.39
Stage3D Program Editor.

Now here is the beauty of it, we can edit the vertex shader (on the left) or the fragment shader (on the right) from Scout and upload the new shader and see the changes reflected in the Stage3D Rendering.

Let's try something. We are going to change the fragment shader, and see the impact. Here is the fragment shader we have on this draw call:

```
//fragment program (agal 1)
tex ft0, v0, fs1 <2d, linear, miplinear, repeat>
tex ft1, v1, fs2 <2d, linear, miplinear, repeat>
mul oc, ft0, ft1
```

And here is the frame being drawn and represented in the Stage3D Rendering:

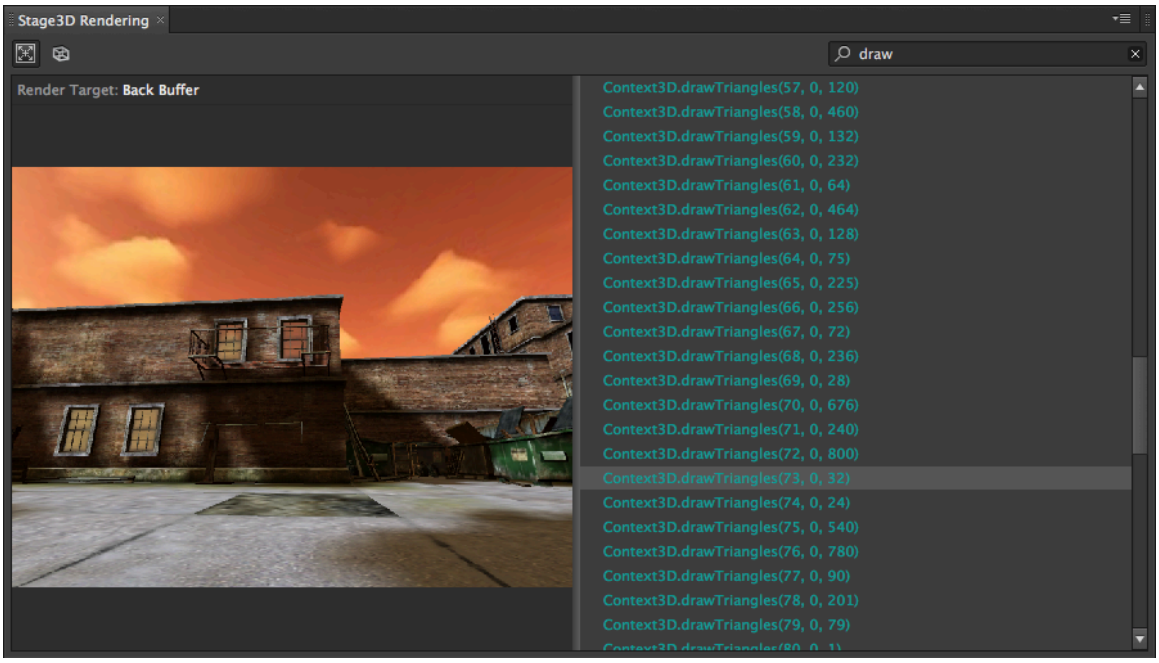

Figure 1.40
Stage3D Program Editor.

We see that we have here two texture sampling (the first 2 lines), then a multiply is done with the two textures, to blend them together. Let's disable the multiply and see what happens. In the Shader Editor we change our shader to the following:

```
//fragment program (agal 1)
tex ft0, v0, fs1 <2d, linear, miplinear, repeat>
tex ft1, v1, fs2 <2d, linear, miplinear, repeat>
mov oc, ft0
```

Note that we did not remove the two "texture sampling", but we do not multiply anymore, we simply output the color related to the first texture sampling.

By pressing the upload button, we see the changes live inside the Stage3D Rendering:

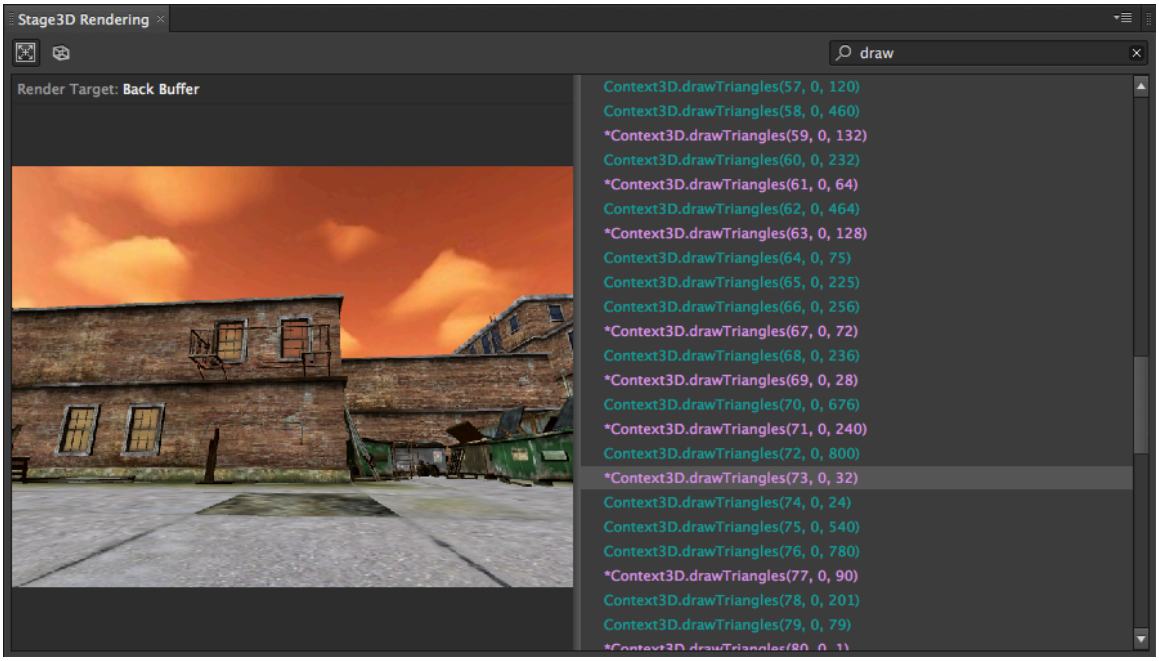

Figure 1.41
Our scene without shadows.

Did you see the shadow disappeared? Nice! The multiply was used to blend the baked shadows. Let's modify the fragment shader again, this time we will just output the color related to the second texture sample:

```
//fragment program (agal 1)
tex ft0, v0, fs1 <2d, linear, miplinear, repeat>
tex ft1, v1, fs2 <2d, linear, miplinear, repeat>
mov oc, ft1
```

By uploading this new shader, we get the following result:

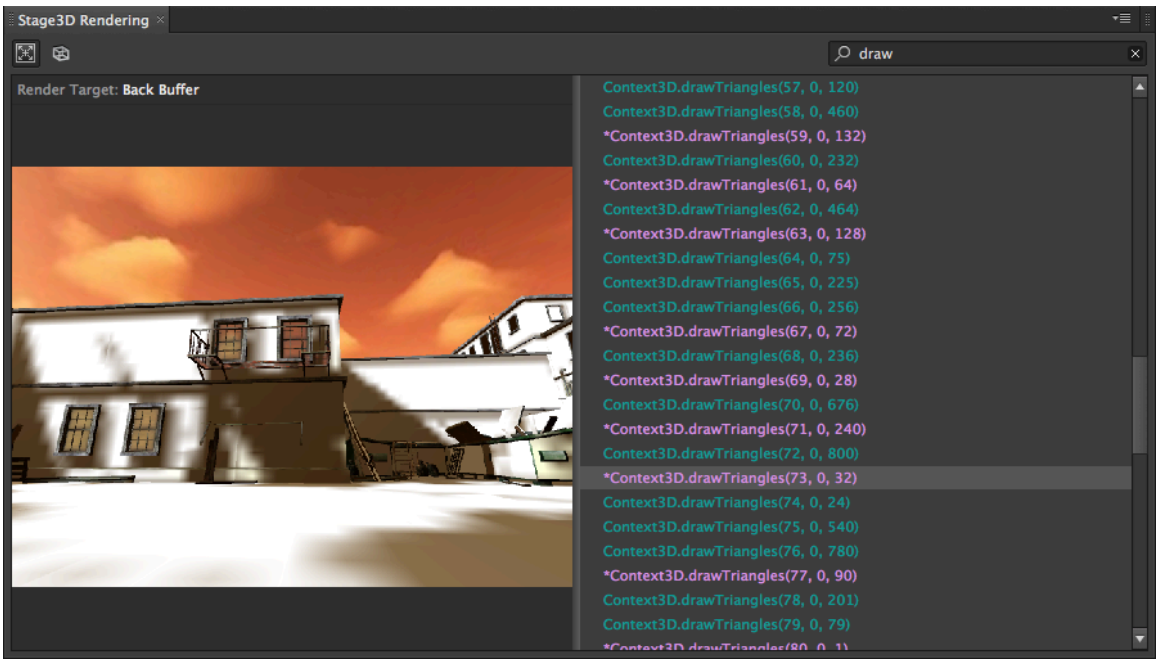

Figure 1.42
Our scene with the shadows only.

And voila! Only our baked shadows are shown. As a result, the Stage3D Program Editor will be a very useful tool to debug, but also learn how to program AGAL shaders.

Feedback

We are excited that you are part of the Customer Advisory Board of Scout. Please share your feedback about Scout so that you can help to make it a great invaluable product to all ActionScript 3 developers.

Thanks for trying Scout!
The Scout team.

Appendix 1.

To previous users, the mms.cfg file is no longer used for any telemetry configuration. Any telemetry settings saved in that file should be removed and will be ignored by current runtimes.

The rules have changed:

1. We have changed "telemetry.cfg" to ".telemetry.cfg" (with a starting dot) on the desktop so that it will be hidden in OSX user folders.
2. The new version of Scout will read and write only to ".telemetry.cfg" in the current user directory. Please delete any copy of "telemetry.cfg" (without starting dot) that might have in your desktop user folder.

If needed, it is still possible to configure mobile manually without the companion apps by packaging a configuration file with the app if needed for special cases.

On mobile, the rules have changed too:

1. Android no longer supports reading "telemetry.cfg" from the sdcard root, but you can add a ".telemetry.cfg" to the application package.
2. iOS will read from "telemetry.cfg" packaged in the application. Since iOS does not allow packaging names that start with a ".", you must use "telemetry.cfg" (without starting dot).
3. Mobile configuration files override the companion apps, so remove any configuration files before using the companion applications.

The runtime on all platforms will first look for ".telemetry.cfg" and if not found will look for "telemetry.cfg". We are providing a project (Waste Invaders.zip - http://gaming.adobe.com/getstarted/) containing already everything you need to test AIR profiling with Scout.

The .telemetry.cfg file should look like this:

TelemetryAddress=<your ip>
SamplerEnabled=true
Stage3DCapture=true
DisplayObjectCapture=true

Then use the following commands to package the application for iOS:

```
./adt -package -target ipa-ad-hoc -sampler -provisioning-profile <full path to YourProvision.mobileprovision> -keystore <full path to YourCert_cert.p12> -storetype PKCS12 -storepass YourPass <appName>.ipa <appName>-app.xml <appName>.swf <asset1> <asset2> <appIconName.png> telemetry.cfg
```

FAQ

Connection troubleshooting

No session gets started

- Is your version of Adobe Scout new enough?
- Is your version of Flash Player new enough?
    - http://get.adobe.com/flashplayer/.
- For mobile projects, did you build with a new version of the SDK?
    - http://labs.adobe.com/technologies/flashbuilder4-7/

- For mobile projects, is the companion app connected? (see below if not)
- Is ~/.telemetry.cfg there? Does it have the right contents?
- Is the Scout server listening?
    - Open Scout, then type "localhost:7934" into your web browser.
    - It should return very quickly with something like "server dropped connection" or "no data received" (it's not an HTTP server).
    - Scout should show the error "can't start a session because the telemetry data isn't valid".
    - If the browser stalls for a long them, then comes back with "server not found" or similar, the socket isn't working. Your firewall/security gunk may be in the way.
    - Try changing the Scout port (Preferences > Listen for new session on port).

iOS/Android Companion app can't connect

- Is your Scout companion app new enough?
- Does the companion app find your machine? If not, can it connect when you choose "Other" and type the IP?
- Can connect to your machine from the device using another app? Try setting up a web server on your machine and connecting to it
    - Mac: Enable System Prefs > Sharing > Web sharing.
    - Win: Enable Internet Information Services (IIS)
    - Test that you can get to your machine's website from itself
    - Now try from your device

What is "Waiting for GPU"?

In a Stage3D app, your CPU and GPU have to run in lock-step in order to get the framerate you want. If either one is overloaded, the framerate will drop. "Waiting for GPU" means that your GPU is overloaded. There are a few possibilities:

1. GPUs will not go faster than 60fps. If you try to go faster, they simply block. If you're getting 60fps, don't worry about the gray bars - ship it!
2. If you're getting a consistently low framerate, you're asking your GPU to do too much. You'll have to optimize your 3D code (or run on a better device) - less triangles, simpler shaders, smaller textures, etc. Unfortunately, Scout can't show you directly what was slow on the GPU, because it can't measure time on the GPU. However, if you use the Stage3D Recording feature (turn it

on in the settings area), you can see exactly which 3D commands you executed, step through them, and see how they impacted the scene. You may be able to find some inefficiencies that way.

3. If your framerate is jittering, or oscillating, or doing a sawtooth pattern, you may be running into a Flash Player problem relating to the synchronization between the CPU and GPU. We're working on this.