A Detailed Look at Cairo's OpenGL Spans Compositor Performance

Bryce Harrington – Senior Open Source Developer

Samsung Research America (Silicon Valley)

b.harrington@samsung.com

What is Cairo?

cairo

2D pen-based drawing model

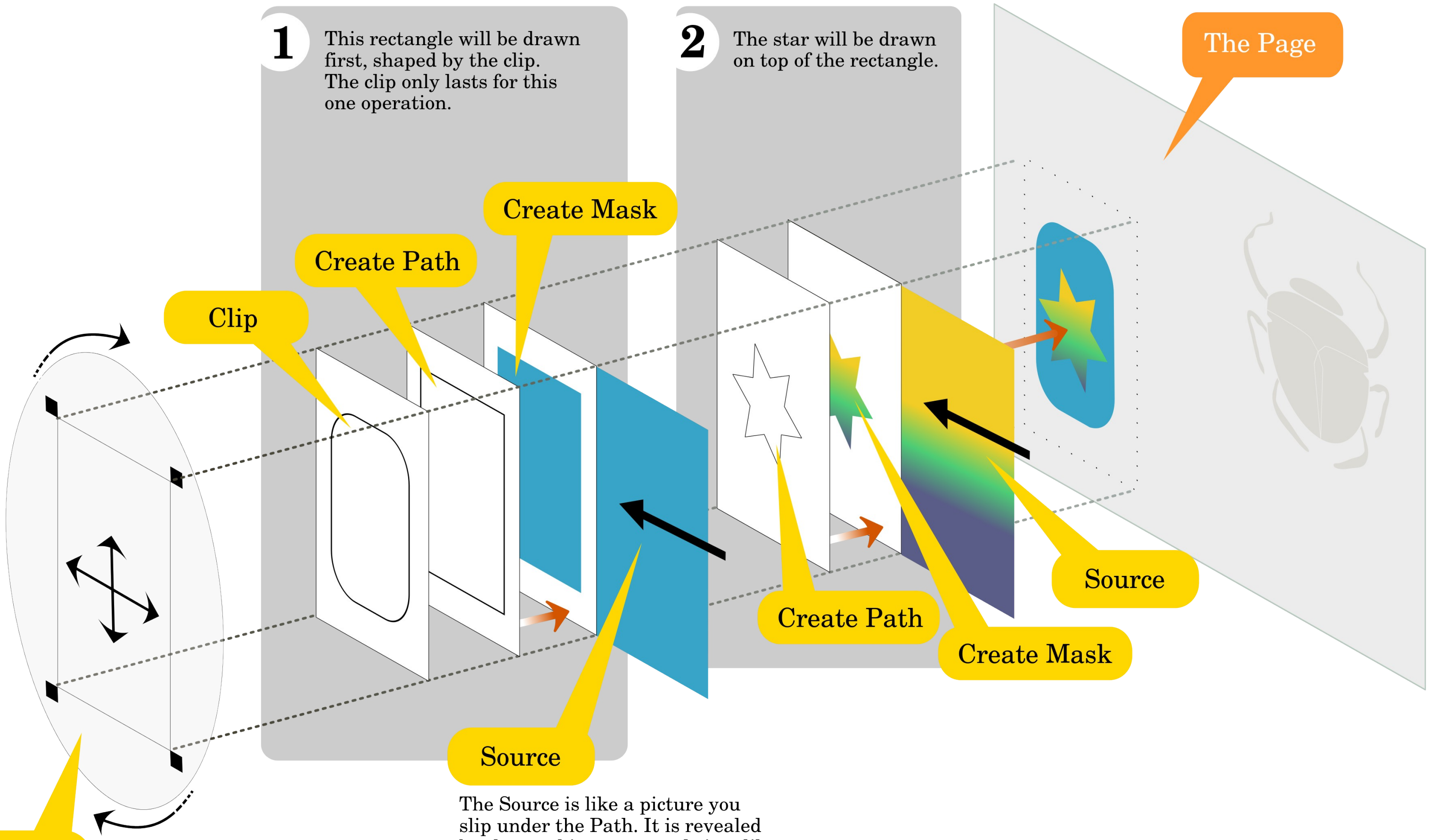
For both display and print

Includes backends for acceleration
and for vector output formats

python Cairoglyphics vers 2.0

Overview

This diagram works from left to right. It shows the drawing of two shapes which each go to the page one after the other. All the commands to do this come from a *toolbox* called the Context. On the next page (Summary) I list many of the commands that you can use.


1 This rectangle will be drawn first, shaped by the clip. The clip only lasts for this one operation.
2 The star will be drawn on top of the rectangle.
The Page
Create Mask
Create Path
Clip
Source
Create Path
Create Mask
Source
Transforms

Transforms

Before you start drawing, you can set up a rotation and scale just like in a vector program like Inkscape.

Source

The Source is like a picture you slip under the Path. It is revealed by the masking commands just like scratching away the surface to see what's underneath.

See *Sources, Surfaces & Patterns* diagram for details.

Diagram (c) 2007 Donn Ingle - donn.ingle@gmail.com. Kubuntu 7.10 & Inkscape

http://www.tortall.net/mu/wiki/CairoTutorial

Where is Cairo Used on the Linux Desktop?

GTK+/Pango

GNOME, XFCE4

Gnuplot

Gnucash

Mozilla

Evince (xpdf)

Scribus

Inkscape

:

:

:

```
$ apt-cache rdepends libcairo2 | wc -l
712
```

Cairo Backends

Format backends

- ps
- pdf
- svg

Platform backends

- image
- xlib
- xcb
- **cairo-gl**
- quartz
- win32
- beos

Cairo-gl on the Linux Desktop

Cairo-gl is not enabled for some distros (e.g. Ubuntu):

- --enable-gl links cairo to libgl
- NVIDIA's libgl gets linked to every client app
- Enormous RAM increase per app running (300%)
- See Launchpad #725434

Several GL backends supported

- cairo-gl (OpenGL) - EGL, GLX, WGL
- glesv2 (OpenGL ES 2.0) - EGL
- glesv3 (OpenGL ES 3.0) - EGL
- vg (OpenVG) - EGL, GLX
- cogl - experimental

Cairo-gl Compositors

Compositing combines visual elements into a single scene

The cairo-gl backend has multiple compositors:

- MSAA
- Spans
- Mask
- Traps

cairo-gl heuristically selects best compositor for operation.

Or:

```
export CAIRO_GL_COMPOSITOR=spans
```

Cairo-gl compositing fallbacks

MSAA - Multisample anti-aliasing

- Composites OpenGL primitives directly to the GPU

Spans

- Scanline compositing – rows of identical pixels inside regular polygonal shapes

Mask

- Renders the mask using spans on CPU rather than geometry

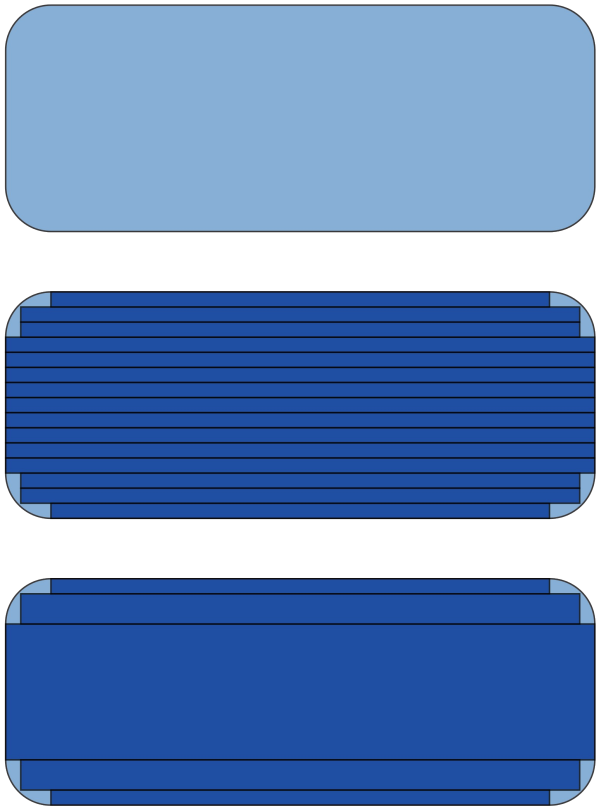
Traps

- Traps is the original Cairo 1.0 compositor, based on Xrender
- Only used for glyph rendering fallback now

Image backend

- Software rendering

Spans Compositor



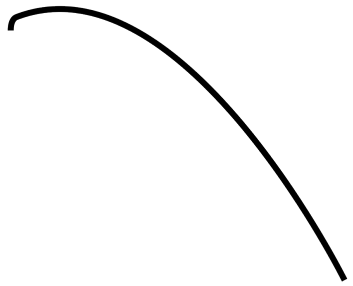
Identifies horizontal lengths that will render as identical pixels.

Spans are drawn as GL_LINES or as GL_QUADS where possible.

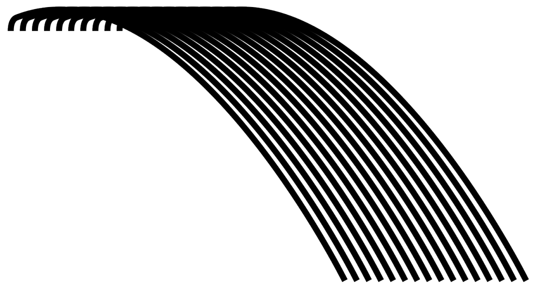
Cairo Testing

Cairo testing

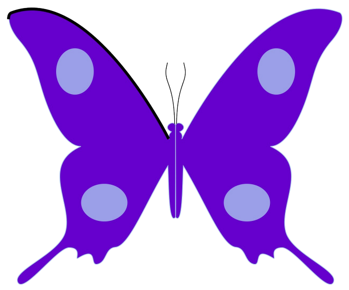
- Functional tests



- Micro-benchmarks



- Macro-benchmarks

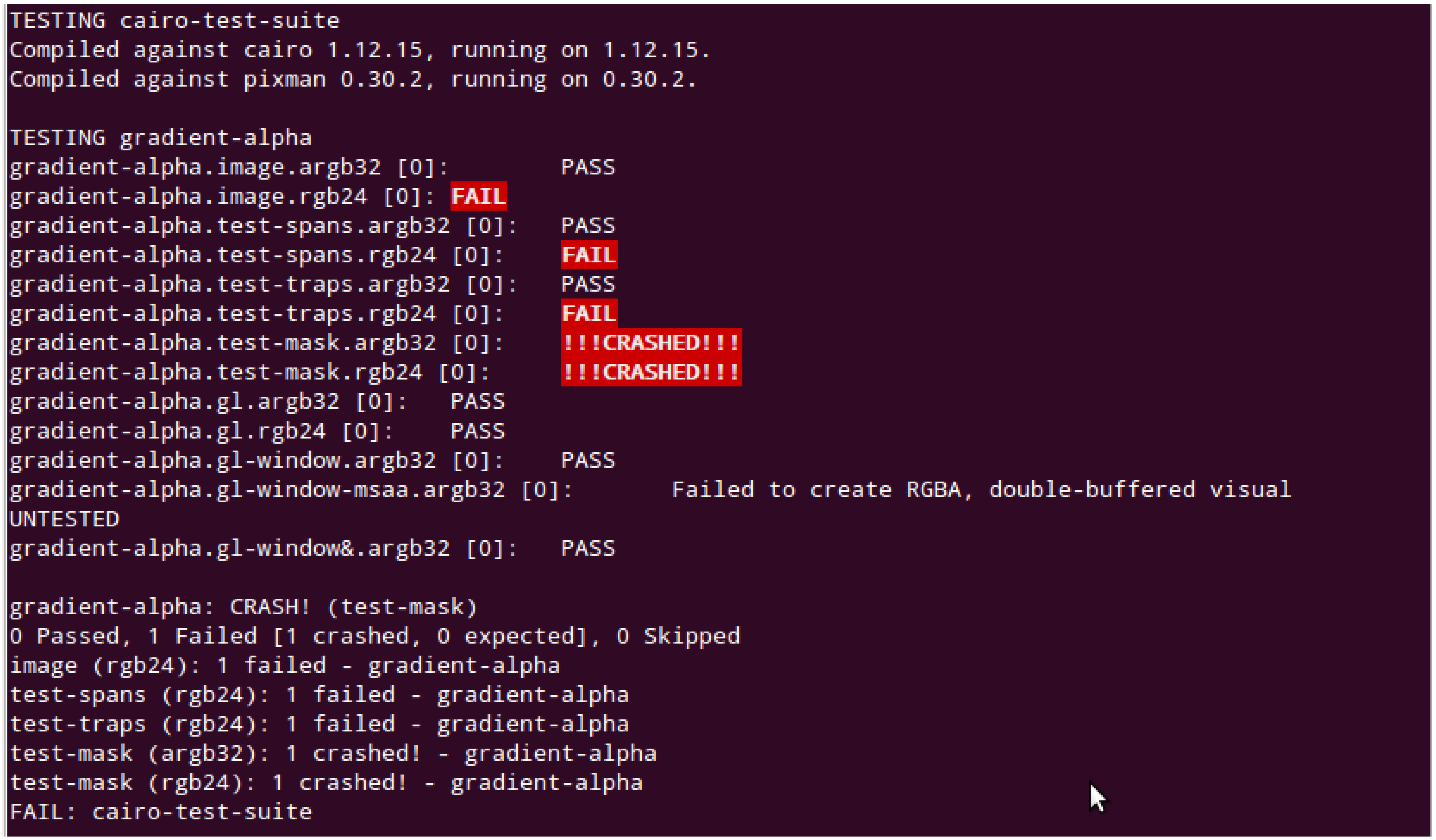


- Other (manually run) benchmarks

Functional Tests

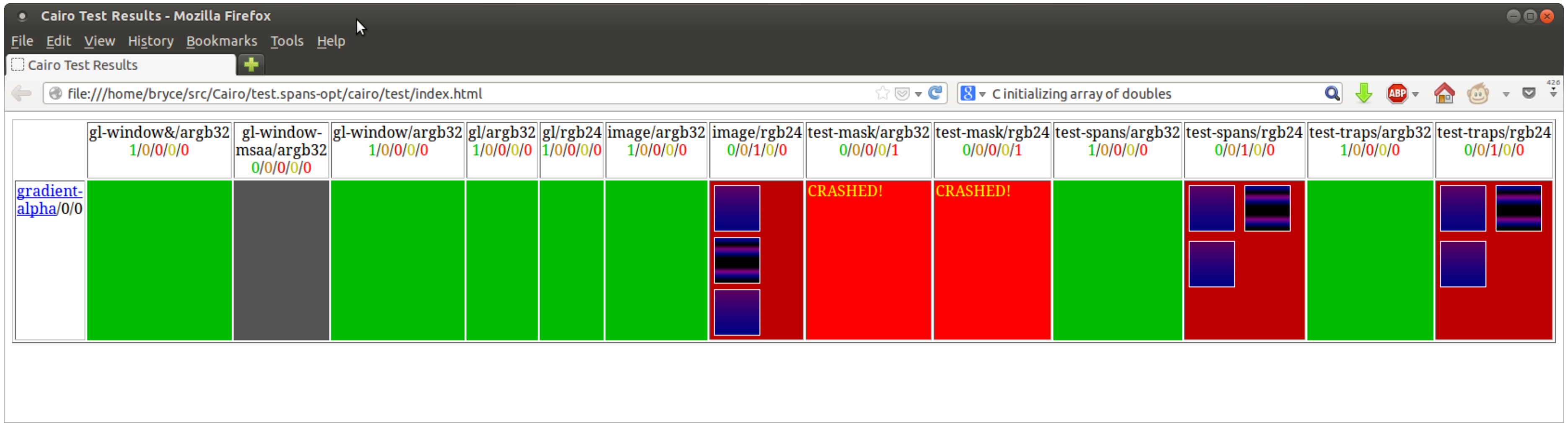
Cairo functional test suite

```
$ export CAIRO_TESTS="gradient-alpha"
$ make test TARGETS=image,test-traps,test-mask,test-spans,gl
```



```
TESTING cairo-test-suite
Compiled against cairo 1.12.15, running on 1.12.15.
Compiled against pixman 0.30.2, running on 0.30.2.

TESTING gradient-alpha
gradient-alpha.image.argb32 [0]:        PASS
gradient-alpha.image.rgb24 [0]: FAIL
gradient-alpha.test-spans.argb32 [0]:   PASS
gradient-alpha.test-spans.rgb24 [0]:    FAIL
gradient-alpha.test-traps.argb32 [0]:   PASS
gradient-alpha.test-traps.rgb24 [0]:    FAIL
gradient-alpha.test-mask.argb32 [0]:    !!!CRASHED!!!
gradient-alpha.test-mask.rgb24 [0]:     !!!CRASHED!!!
gradient-alpha.gl.argb32 [0]:   PASS
gradient-alpha.gl.rgb24 [0]:    PASS
gradient-alpha.gl-window.argb32 [0]:    PASS
gradient-alpha.gl-window-msaa.argb32 [0]:       Failed to create RGBA, double-buffered visual
UNTESTED
gradient-alpha.gl-window&.argb32 [0]:   PASS

gradient-alpha: CRASH! (test-mask)
0 Passed, 1 Failed [1 crashed, 0 expected], 0 Skipped
image (rgb24): 1 failed - gradient-alpha
test-spans (rgb24): 1 failed - gradient-alpha
test-traps (rgb24): 1 failed - gradient-alpha
test-mask (argb32): 1 crashed! - gradient-alpha
test-mask (rgb24): 1 crashed! - gradient-alpha
FAIL: cairo-test-suite
```

Cairo functional test suite



Cairo Test Results - Mozilla Firefox

file:///home/bryce/src/Cairo/test.spans-opt/cairo/test/index.html

	gl-window&/argb32 1/0/0/0/0	gl-window-msaa/argb32 0/0/0/0/0	gl-window/argb32 1/0/0/0/0	gl/argb32 1/0/0/0/0	gl/rgb24 1/0/0/0/0	image/argb32 1/0/0/0/0	image/rgb24 0/0/1/0/0	test-mask/argb32 0/0/0/0/1	test-mask/rgb24 0/0/0/0/1	test-spans/argb32 1/0/0/0/0	test-spans/rgb24 0/0/1/0/0	test-traps/argb32 1/0/0/0/0	test-traps/rgb24 0/0/1/0/0
gradient-alpha/0/0	🟩	⬛	🟩	🟩	🟩	🟩	🟥	CRASHED!	CRASHED!	🟩	🟥	🟩	🟥

Micro Benchmarks

Cairo micro benchmarks

$ make perf

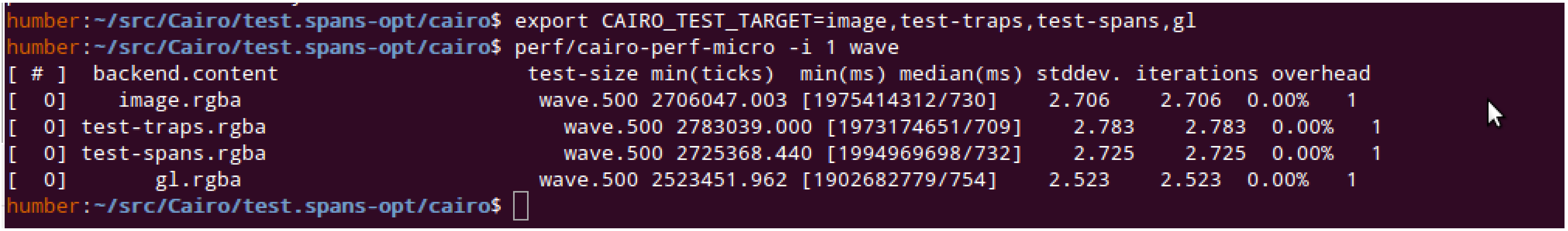
$ sudo taskset -cp 0 $(pidof X)

$ taskset -cp 1 $$

crashes

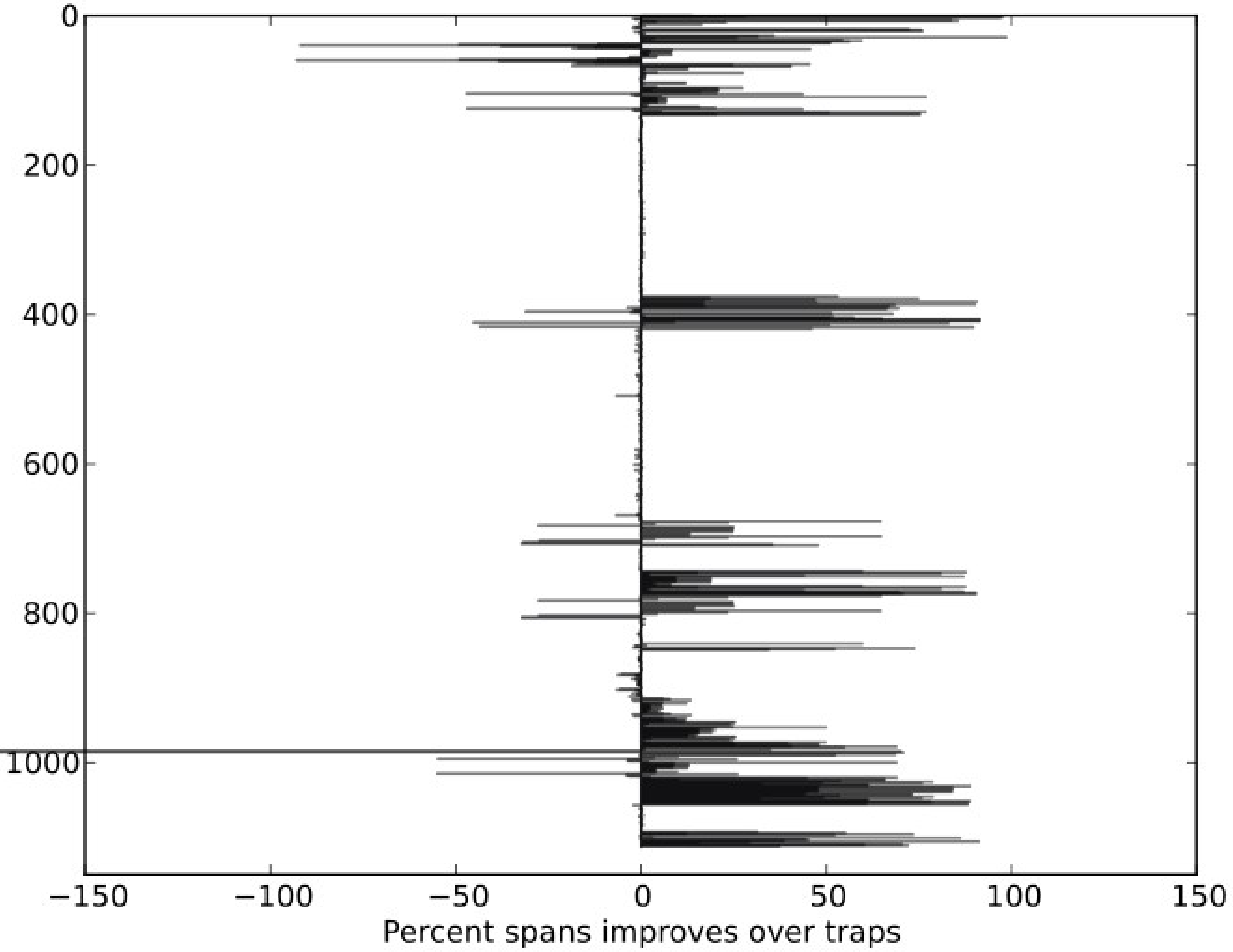
$ export CAIRO_TEST_TARGET=image,test-traps,~~test-mask~~,test-spans,gl

$ perf/cairo-perf-micro -i 1 wave

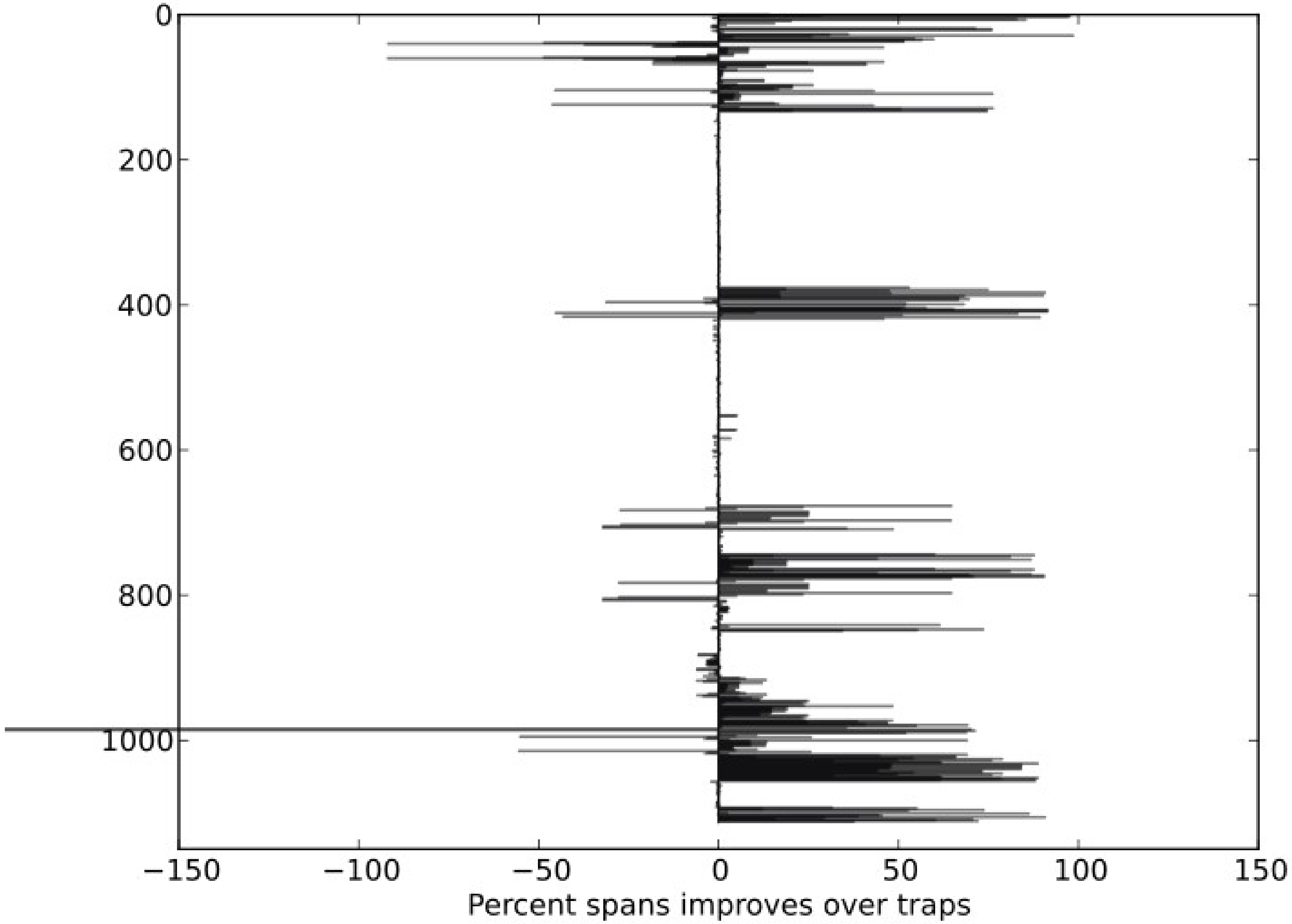


```
humber:~/src/Cairo/test.spans-opt/cairo$ export CAIRO_TEST_TARGET=image,test-traps,test-spans,gl
humber:~/src/Cairo/test.spans-opt/cairo$ perf/cairo-perf-micro -i 1 wave
[ # ]  backend.content                    test-size min(ticks)  min(ms) median(ms) stddev. iterations overhead
[  0]    image.rgba                        wave.500 2706047.003 [1975414312/730]    2.706    2.706  0.00%   1
[  0] test-traps.rgba                        wave.500 2783039.000 [1973174651/709]    2.783    2.783  0.00%   1
[  0] test-spans.rgba                        wave.500 2725368.440 [1994969698/732]    2.725    2.725  0.00%   1
[  0]       gl.rgba                        wave.500 2523451.962 [1902682779/754]    2.523    2.523  0.00%   1
humber:~/src/Cairo/test.spans-opt/cairo$
```

Traps vs. Spans with Intel Driver



Traps vs. Spans with Fglrx



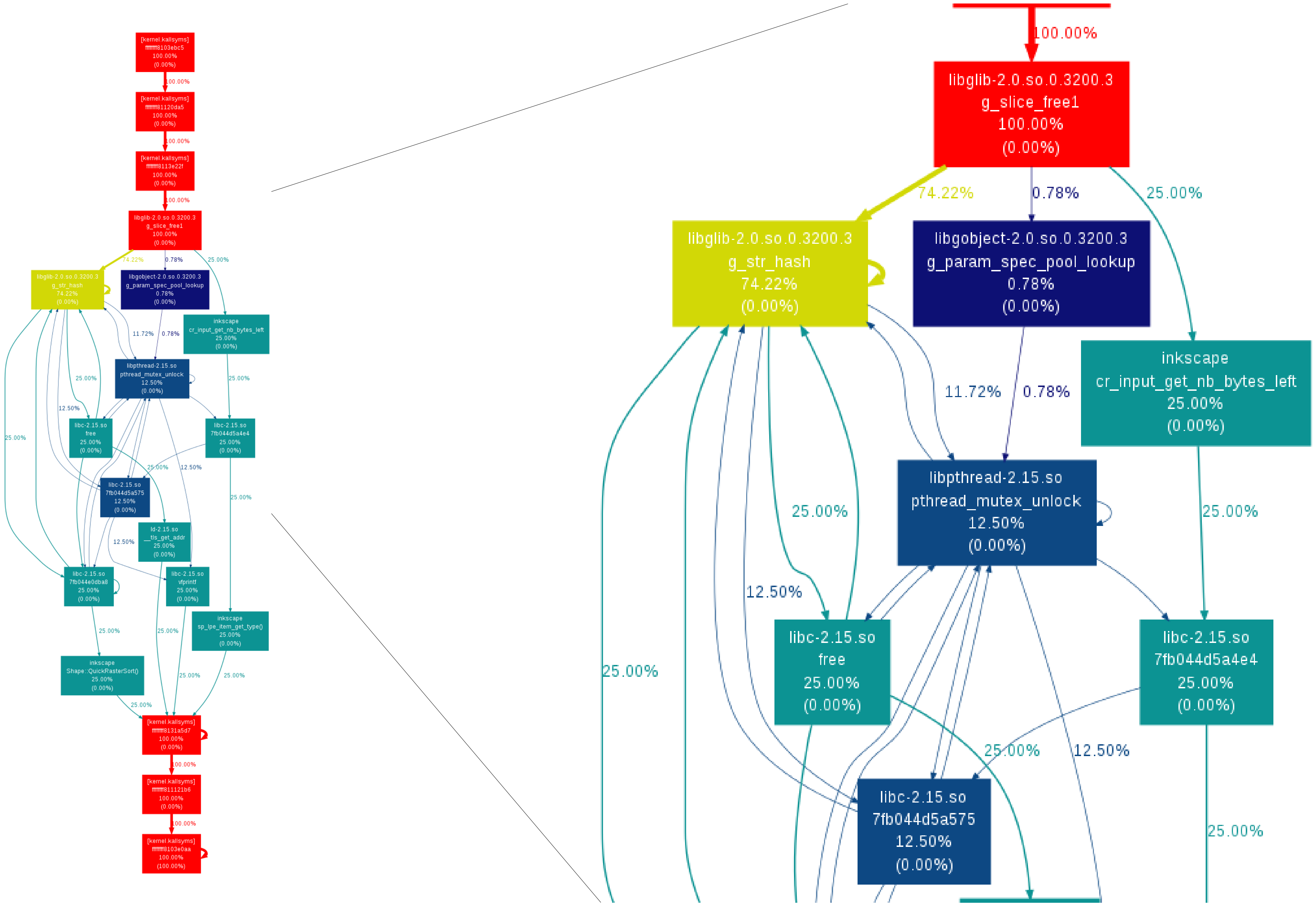
Spans Performance Regressions

Intel	Fglrx	Test Case
-49%	-49%	fill-annuli_image-rgb_source
-92%	-92%	fill-annuli_image-rgba-mag_over
-38%	-37%	fill-annuli_image-rgba-mag_source
-49%	-49%	fill-annuli_similar-rgb_source
-93%	-92%	fill-annuli_similar-rgba-mag_over
-38%	-37%	fill-annuli_similar-rgba-mag_source
-47%	-45%	fill_image-rgba-mag_over
-47%	-46%	fill_similar-rgba-mag_over
-31%	-31%	line-nhh
-45%	-45%	many-fills-horizontal
-43%	-43%	many-strokes-horizontal
-28%	-27%	mask-solid_image-rgba_source
-27%	-27%	mask-solid_similar-rgba_source
-32%	-32%	mask-solid_solid-rgb_source
-32%	-32%	mask-solid_solid-rgba_source
-28%	-28%	paint-with-alpha_image-rgba_source
-28%	-28%	paint-with-alpha_similar-rgba_source
-32%	-32%	paint-with-alpha_solid-rgb_source
-32%	-32%	paint-with-alpha_solid-rgba_source
-418%	-417%	spiral-diag-nonalign-nonzero-fill
-208%	-209%	spiral-diag-pixalign-nonzero-fill
-55%	-55%	stroke_image-rgba-mag_over
-55%	-56%	stroke_similar-rgba-mag_over

Macro Benchmarks

Analyzing performance using linux-perf

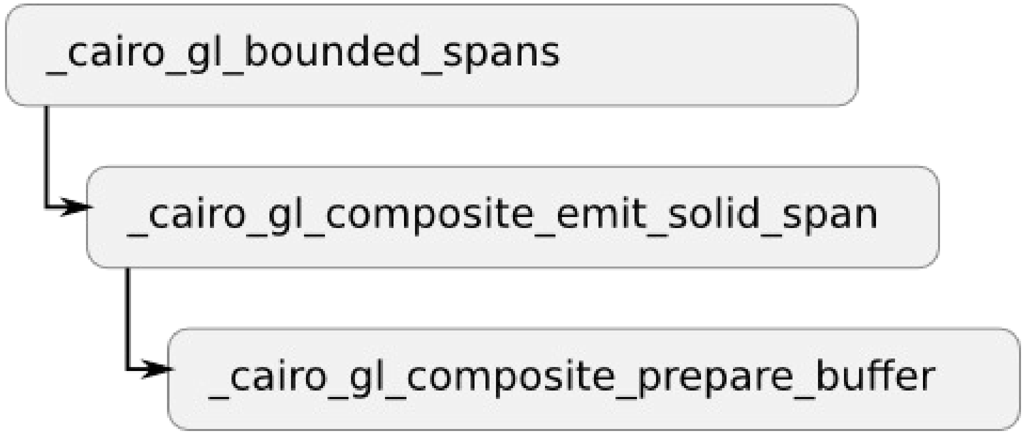
```
$ git clone git://anongit.freedesktop.org/cairo-traces
$ cairo-traces && make && cd ../cairo

$ export CAIRO_TRACE_DIR="../cairo-traces"
$ export CAIRO_TEST_TARGET_EXCLUDE=""
$ export CAIRO_TEST_TARGET="gl image xlib xcb"
$ export CAIRO_GL_COMPOSITOR="msaa"

$ benchmark=firefox-fishbowl
$ iterations=20

$ perf record -g -- ./perf/cairo-perf-trace -i ${iterations} ${benchmark}
$ perf script | gprof2dot.py -f perf | dot -Tpng -o output.png
```



[kernel.kallsyms] ffffffff8103ebc5 100.00% (0.00%)
[kernel.kallsyms] ffffffff81120da5 100.00% (0.00%)
[kernel.kallsyms] ffffffff8113e22f 100.00% (0.00%)
libglib-2.0.so.0.3200.3 g_slice_free1 100.00% (0.00%)
libglib-2.0.so.0.3200.3 g_str_hash 74.22% (0.00%)
libgobject-2.0.so.0.3200.3 g_param_spec_pool_lookup 0.78% (0.00%)
inkscape cr_input_get_nb_bytes_left 25.00% (0.00%)
libpthread-2.15.so pthread_mutex_unlock 12.50% (0.00%)
libc-2.15.so free 25.00% (0.00%)
libc-2.15.so 7fb044d5a4e4 25.00% (0.00%)
libc-2.15.so 7fb044d5a575 12.50% (0.00%)
ld-2.15.so __tls_get_addr 25.00% (0.00%)
libc-2.15.so 7fb044e0dba8 25.00% (0.00%)
libc-2.15.so vfprintf 25.00% (0.00%)
inkscape sp_lpe_item_get_type() 25.00% (0.00%)
inkscape Shape::QuickRasterSort() 25.00% (0.00%)
[kernel.kallsyms] ffffffff8131a5d7 100.00% (0.00%)
[kernel.kallsyms] ffffffff811121b6 100.00% (0.00%)
[kernel.kallsyms] ffffffff8103e0aa 100.00% (100.00%)
100.00%
74.22%
0.78%
25.00%
11.72%
12.50%

Analyzing performance using linux-perf

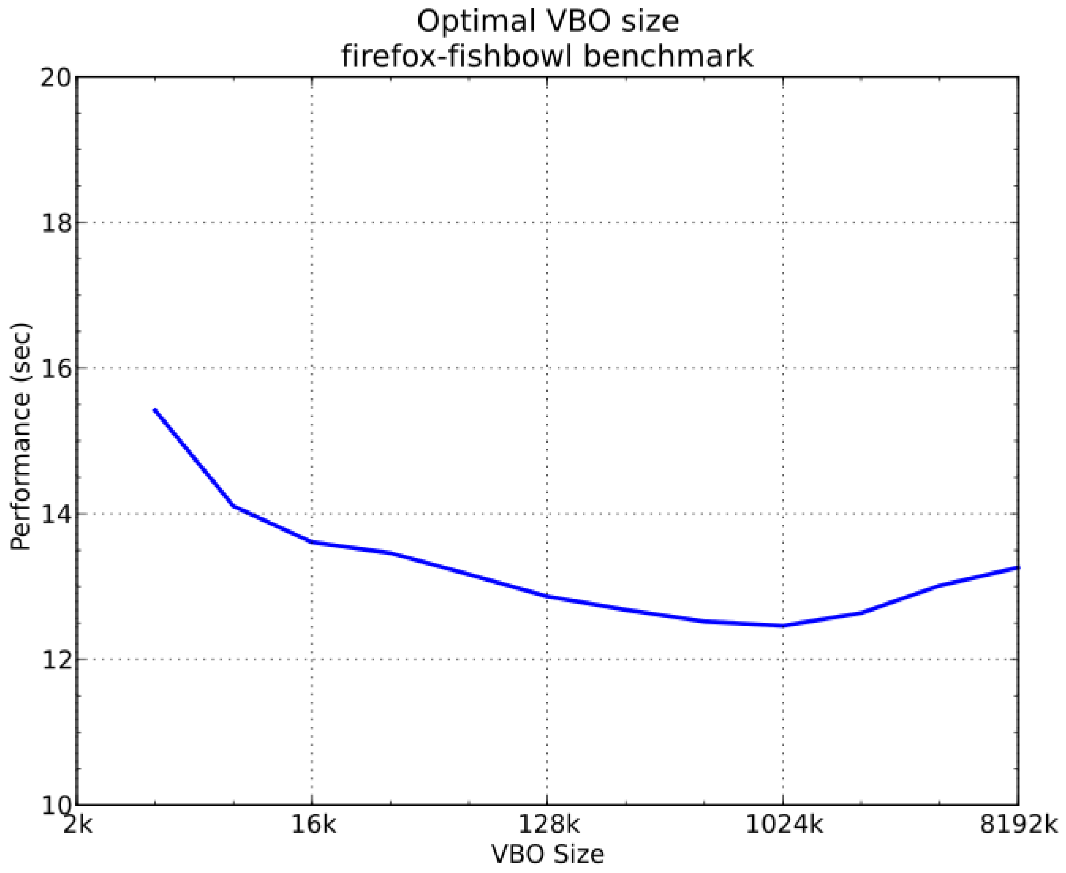
```
$ perf report

+  11.56%  lt-cairo-perf-t  libcairo.so.2.11200.15                      [.] _cairo_tor_scan_converter_generate
+  11.27%  lt-cairo-perf-t  libc-2.15.so                                [.] 0x80f31
+   9.32%  lt-cairo-perf-t  libcairo.so.2.11200.15                      [.] _cairo_gl_composite_emit_solid_span
+   6.78%  lt-cairo-perf-t  libcairo.so.2.11200.15                      [.] cell_list_render_edge
+   5.68%  lt-cairo-perf-t  libcairo-script-interpreter.so.2.11200.15   [.] _csi_hash_table_lookup
+   5.06%  lt-cairo-perf-t  libcairo-script-interpreter.so.2.11200.15   [.] _scan_file.5939
+   3.60%  lt-cairo-perf-t  libcairo.so.2.11200.15                      [.] _cairo_gl_bounded_spans
+   3.24%  lt-cairo-perf-t  [kernel.kallsyms]                           [k] 0xffffffff8103e0aa
+   2.35%  lt-cairo-perf-t  libcairo-script-interpreter.so.2.11200.15   [.] _csi_parse_number
+   2.25%  lt-cairo-perf-t  libcairo-script-interpreter.so.2.11200.15   [.] csi_file_getc
+   1.32%  lt-cairo-perf-t  libcairo.so.2.11200.15                      [.] _cairo_gl_composite_prepare_buffer
```


_cairo_gl_bounded_spans
_cairo_gl_composite_emit_solid_span
_cairo_gl_composite_prepare_buffer

Optimizing VBO size to improve performance

Vertex Buffer Objects (VBOs) store vertex data (position, vector, color, etc.) in video device memory for rendering

- Small VBO means more flushes
- Large VBO can cause trouble for embedded devices
- Currently is 16k



WIP: http://cgit.freedesktop.org/~bryce/cairo/?h=vbo-size

Analysis of other benchmarks - intel

swfdec-giant-steps

```
18.26%     _cairo_tor_scan_converter_generate
 4.35%     cell_list_render_edge
 3.70%     _fill_xrgb32_lerp_opaque_spans
 2.41%     _cairo_bentley_ottmann_tessellate_polygon
```

firefox-canvas

```
11.49%    _cairo_tor_scan_converter_generate
 3.50%    _cairo_bentley_ottmann_tessellate_polygon
 1.86%    cell_list_render_edge
 1.75%    _cairo_polygon_intersect
```

ocitysmap

```
 8.69%    _cairo_tor_scan_converter_generate
 2.86%    _cairo_bentley_ottmann_tessellate_polygon
 2.79%    _fill_xrgb32_lerp_opaque_spans
 0.78%    _cairo_tor_scan_converter_add_polygon
 0.67%    cell_list_render_edge
```

firefox-scrolling

```
 1.85%    _cairo_hash_table_lookup
 1.11%    _cairo_scaled_font_glyph_device_extents
```

evolution

```
 1.06%    _fill_xrgb32_lerp_opaque_spans
 0.97%    _cairo_tor_scan_converter_generate
 0.88%    _cairo_hash_table_lookup
 0.71%    _cairo_scaled_font_glyph_device_extents
```

firefox-talos-svg

```
 5.88%    _cairo_tor_scan_converter_generate
 2.94%    _cairo_bentley_ottmann_tessellate_polygon
```

Analysis of other benchmarks - fglrx

swfdec-giant-steps

```
13.25%      _cairo_tor_scan_converter_generate
 3.14%      cell_list_render_edge
 2.76%      _fill_xrgb32_lerp_opaque_spans
 1.89%      _cairo_bentley_ottmann_tessellate_polygon
```

firefox-canvas

```
10.45%    _cairo_tor_scan_converter_generate
 3.57%    _cairo_bentley_ottmann_tessellate_polygon
 1.73%    cell_list_render_edge
 1.63%    _cairo_polygon_intersect
```

ocitysmap

```
 8.23%    _cairo_tor_scan_converter_generate
 2.78%    _cairo_bentley_ottmann_tessellate_polygon
 1.88%    _fill_xrgb32_lerp_opaque_spans
 0.81%    _cairo_tor_scan_converter_add_polygon
 0.79%    cell_list_render_edge
```

firefox-scrolling

```
 0.82%    _cairo_hash_table_lookup
 0.52%    _cairo_scaled_font_glyph_device_extents
```

evolution

```
 0.70%    _cairo_tor_scan_converter_generate
 0.65%    _cairo_hash_table_lookup
 0.50%    _cairo_scaled_font_glyph_device_extents
```

firefox-talos-svg

```
 5.59%    _cairo_tor_scan_converter_generate
 2.82%    _cairo_bentley_ottmann_tessellate_polygon
```

Generating new traces

To record a trace:

```
 $ cairo-trace --profile inkscape <args>
```

Generates an inkscape.1234.trace file.

Please document exact steps to re-generate the trace, for future reference!

Thank you.

Bryce Harrington – Senior Open Source Developer

Samsung Research America (Silicon Valley)

B.Harrington@Samsung.com

Further Reading

http://cworth.org/tag/cairo/

http://www.mattfischer.com/blog/?p=375

http://ssvb.github.io/2012/05/04/xorg-drivers-and-software-rendering.html

http://mgdm.net/talks/dpc10/cairo.pdf