Status of the Layer-Based SVG Engine in WebKit

Web Engines Hackfest 2024
A Coruña, 3rd June 2024

Nikolas Zimmermann nzimmermann@igalia.com
A short introduction

The LBSE work is a joint effort:

- Nikolas Zimmermann, Igalian since 2019 - located in Germany
- Rob Buis, Igalian since 2017 - located in Ireland

- Founders of kdom, kcanvas, ksvg2/khtml2
- khtml contributors since 2001
- WebKit contributors since 2005
- WebKit reviewers since February 2007
Topics

1) Introduction to LBSE

2) Upstreaming status

3) Outlook
1) Introduction to LBSE
What is LBSE?

Layer-Based SVG Engine

- Codename for new SVG engine in WebKit
- Started as Proof-Of-Concept in September 2019
- Goal: resolve architectural issues present since 15+ years
- Developed by Igalia, funded by Igalia, Vorwerk and Wix.
What is LBSE?

Layer-Based SVG Engine

- Enable hardware-accelerated compositing / hardware-accelerated transform animations
- Enable 3D transform support for arbitrary SVG content, unlock perspective transformations
- Unify HTML/SVG rendering pipelines, which are mutually exclusive at present
- **Proof-Of-Concept** patch passed all existing layout tests in **October 2021** (1)
- Performance comparable to legacy engine in e.g. MotionMark (2)

(1) Compositing/z-index/will-change/... was only tested in most basic scenarios for SVG.
(2) MotionMark suffers from layer overhead → LBSE ~2-4% slower.
How is it achieved?

- Let SVG participate in the layer tree, which handles compositing, 3D transformations, etc. for HTML/CSS.
- Remove knowledge about transform handling out of the SVG renderers.
- Remove SVG specific clipping/marker/masking/filter code
- Redesign SVG render tree to be convenient for the existing CSS code. (Coordinate system decisions, etc.)
- Reuse as much as code possible in RenderLayer - without SVG specific changes.
Evolution since 2021

The "final" version of the prototype was a drop-in replacement for the old SVG engine.

During and after the WebKit contributors meeting 2021, a plan was established how LBSE can be integrated into WebKit, without violating the usual high standards with respect to reviewability.

As a consequence...

- No way to use any existing patch as-is from LBSE downstream
- Lots of manual work necessary.
- It is equal to yet another rewrite.
Upstreaming plan

It was decided to bring-up LBSE in small, reviewable atomic pieces in parallel to the legacy SVG engine, share code where it makes sense, and split elsewhere.

All the code is behind a compile-time flag `ENABLE(LAYER_BASED_SVG_ENGINE)` and an additional setting `LayerBasedSVGEEngineEnabled` that can be used to toggle between LBSE and the legacy SVG engine at runtime.

⟶ Upstreaming started Dec 2021
Let's have a look at the current status...
2) Upstreaming status
Progress tracking

The bug report "#90738 - Harmonize HTML & SVG rendering" tracks the upstreaming status in WebKit Bugzilla; individual commits are tracked on "GitHub WebKitIgalia #1".

- 161 patches so far directly related to LBSE (first: 29. November 2021, current: 10. April 2024).
- Overall status ~ 87%
Achievements

Since the last WebKit Contributors Meeting in 2023, two major tasks were finished. This work was funded by Wix, bringing LBSE closer to a shipable state.

For more details, see the blog post [https://wpewebkit.org/blog/status-of-lbse-in-webkit.html](https://wpewebkit.org/blog/status-of-lbse-in-webkit.html).

1) Re-design resource invalidation logic
2) Implement all SVG resources for LBSE
What is a SVG resource?

```xml
<defs>
  <linearGradient id="aGradient">
    <stop offset="10%" stop-color="blue"/>
    <stop offset="90%" stop-color="green"/>
  </linearGradient>

  <clipPath id="aClip">
    <circle cx="50" cy="50" r="50"/>
  </clipPath>
</defs>

<rect width="100" height="100" fill="url(#aGradient)" clip-path="url(#aClip)"/>
```
Status of resources in October 2023

- LBSE support solids fill / stroke operations for shapes / text ✔
- **No support** for gradients / patterns / markers
- **No support** for clipping / masking
- **No support** for filters

Our plan was to redesign the resource handling, avoiding **design mistakes** from the legacy SVG engine, and offering support for all SVG painting features in an unified way for both HTML elements and SVG elements.

⚠ Caveat: We can not **easily** re-use the resource logic from the legacy SVG engine... ⚠
Legacy SVG engine design issues

Resource invalidation is **fundamentally broken** in the legacy SVG engine.

Example: A `<mask>` is applied to a `<path>`, and a child of that `<mask>` references a `<clipPath>`.

```
<defs>
  <clipPath id="clip">
    <circle r="10"/>
  </clipPath>
  <mask id="mask">
    <rect x="5" clip-path="url(#clip)"/>
  </mask>
</defs>
<path mask="url(#mask)"/>
```

```
RenderSVGHiddenContainer ("defs")
  RenderSVGResourceClipper ("clip")
    RenderSVGEllipse ("circle", r=10)
  RenderSVGResourceMasker ("mask")
    RenderSVGRect ("rect", x=5) | uses "clip" resource

RenderSVGPath ("path") | uses "mask" resource
```
Legacy SVG engine design issues

How to handle nested invalidations?

An early design decision was to re-use the `layout()` logic of the render tree to handle resource invalidations. Consider changing the radius to 20 dynamically from JavaScript.

What happens?

Render tree

- RenderSVGHiddenContainer ("defs")
  - RenderSVGRResourceClipper ("clip")
    - RenderSVGEllipse ("circle", r=10)
  - RenderSVGRResourceMasker ("mask")
    - RenderSVGRect ("rect", x=5) | uses "clip" resource
- RenderSVGPath ("path") | uses "mask" resource
1. Parse attribute in `SVGCircleElement`, update presentational style and trigger style invalidation.

2. During style resolving `RenderSVGEllipse` receives a new style with a changed radius requiring a re-layout, which is triggered asynchronously.

3. The resources in the ancestor chain are notified about the style change, clients of resources are invalidated recursively. `RenderSVGRect` is marked for layout, followed by `RenderSVGPath`.

4. After style resolving finished, a render tree re-layout is triggered asynchronously.

→ Unnecessary `layout()` for subtrees containing resources, expensive tree walks, etc.
Resource handling redesign

Relevant call graphs:

1. JS calling `setAttribute(...)` , triggering the invalidation chain.
Resource handling redesign

2. **RenderSVGEllipse** is marked for layout (as consequence of the previous style invalidation)

```plaintext
frame #0: WebCore::RenderObject::setNeedsLayout(this=0x00000001640056a0)
frame #1: WebCore::RenderObject::setNeedsLayoutAndPrefWidthsRecalc(this=0x00000001640056a0)
frame #2: WebCore::RenderElement::styleDidChange(this=0x00000001640056a0, diff=Layout, oldStyle=...)
frame #3: WebCore::LegacyRenderSVGMModelObject::styleDidChange(this=0x00000001640056a0, diff=Layout, old...
frame #4: WebCore::RenderElement::setStyle(this=0x00000001640056a0, ...)
frame #5: WebCore::RenderTreeUpdater::updateRendererStyle(this=0x000000016f374b50, ...)
frame #6: WebCore::RenderTreeUpdater::updateElementRenderer(this=0x000000016f374b50, ...)
frame #7: WebCore::RenderTreeUpdater::updateRenderTree(this=0x000000016f374b50, ...)
frame #8: WebCore::RenderTreeUpdater::commit(this=0x000000016f374b50, ...)
frame #9: WebCore::Document::updateRenderTree(this=0x0000000116145a00, ...)
frame #10: WebCore::Document::resolveStyle(this=0x0000000116145a00, ...)
frame #11: WebCore::Document::updateStyleIfNeeded(this=0x0000000116145a00)
```
Resource handling redesign

Relevant call graphs:

3. `RenderSVGRect` is marked for layout (as it uses mask, which uses clip, which got modified)
Calling `setNeedsLayout` from within `layout` is evil and inefficient. ⚠️

Furthermore the correctness of the invalidation chain depends on the element order in DOM.

This is known since a long time, dating back to at least 11 years ago:

- Bug #81515 - SVG Resources layout needs refactoring
- Bug #179788 - Make `RenderSVGResource::markForLayoutAndParentResourceInvalidation()` more robust
- Bug #207451 - `RenderSVGShape` invalidates all its resources when it needs layout, but is that necessary?
- Bug #208903 - [SVG] `RenderSVGResourceContainer`'s style invalidation should be a pre-layout task
- Bug #242420 - avoid invalidating SVG resources when referencing element changes layout

... 

Plan presented during WebKit Contributors Meeting in October 2023:

1) Re-design resource invalidation logic
2) Implement all SVG resources for LBSE
Status 2024

1) Re-design resource invalidation logic

A new approach, re-using similar logic from CSS invalidation, was implemented. All code landed upstream.

2) Implement all SVG resources for LBSE

- LBSE support solids fill / stroke operations for shapes / text
- Complete support for gradients / patterns / markers
- Complete support for clipping / masking
- Complete support for filters (under review)
3) Outlook
Short-term

- ⚠ Ensure funding for LBSE work. ⚠
  We need more partners that help us to get LBSE across the finish line.

- Finish `<filter>` work (Apple).

- Finish `<text>`/*`<tspan>` related repainting issues

- Verify invalidation is complete for all SVG resources, used in HTML/SVG contexts.

- Perform security audit: fuzzing, ASAN -- try the whole arsenal.
Long-term

- Ensure funding for LBSE work.

- Finish LBSE implementation, such that all layout tests pass.

- Ensure LBSE is at least as fast as the legacy engine in any standard benchmark.
  - Reduce RenderLayer overhead.
  - Selectively construct RenderLayers only if necessary.
  - ... (there are more ideas in the pipeline, that need testing)

- Final task: Turn on LBSE by default. Remove legacy SVG engine.
Thanks for your attention!