SVG Text NG

Author: Kent Tamura <tkent@chromium.org>
Date: 2021-09-29
Visibility: Public
Tracking Bug: crbug.com/1179585

Background

As of February 2021, text rendering code for SVG in Blink depends on the legacy layout classes such as `LayoutBlockFlow`, `RootInlineBox`, `InlineFlowBox`, and `InlineTextBox`. We should stop using them and should switch to LayoutNG in order to remove the legacy layout.

The current SVG text rendering has many issues. We'd like to resolve them by rewriting the text handling algorithm while switching to LayoutNG

Summary of the benefit by LayoutNG SVG text:

- We can remove the legacy layout in the future
- Reduce memory consumption by avoiding the legacy text shaping
- Improve the correctness of BiDi resolving by using LayoutNG BiDi resolver

Understanding text in SVG

Text in SVG must be in `<svg:text>`. `<svg:text>` can contain the following rendered nodes:

- `Text`
- `<svg:tspan>`
- `<svg:textPath>`
- `<svg:a>`

`<svg:text>` can contain only inline boxes.

We can specify absolute/relative positions and angles of each character by `x/y/dx/dy/rotate` attributes. e.g. `<svg:text rotate="0, 90, 180, 270, 0">Hello</svg:text>`

We can put text on an `<svg:path>` by `<svg:textPath>`.

We can specify the length in which text should be drawn, by `textLength` attribute.

In SVG 1.1, SVG text doesn't have line wrapping. SVG 2.0 defines that `inline-size` property triggers line wrapping though no browsers implement it yet.

Legacy text processing

Layout

`<svg:text>` is represented by `LayoutSVGText`, which is a subclass of `LayoutBlockFlow`. It can have only inline children. `LayoutSVGText` lays out children as a normal inline formatting context except for `CreateLineBoxesFromBidiRuns()`, which calls `SVGRootInlineBox::ComputePerCharacterLayoutInformation()` instead of `LayoutBlockFlow::ComputeInlineDirectionPositionsForLine()`.

`SVGTextLayoutEngine` is called by `SVGRootInlineBox::ComputePerCharacterLayoutInformation()`, and it creates `SVGTextFragment` instances, and pass them to `SVGInlineTextBox`.

`SVGTextFragment` represents a group of characters that can be positioned together. It is owned by an `SVGInlineTextBox`.

Text scaling

`FontDescription::ComputedSize()` returns the computed size based on `font-size` content attribute or `font-size` CSS property. In SVG text layout, `LayoutBlockFlow` lays out with the

`ComputedSize()`, however geometries in inline boxes are not used by `SVGTextLayoutEngine`. It recomputes text geometries with scaled font, then stores unscaled values to `SVGTextFragment`'s data members as `float`, not `LayoutUnit`. On the paint stage, we draw text with scaled fonts.
For example, for `<svg:svg width="480" height="360" viewBox="0 0 160 120"> <svg:text font-size="0.4" transform="scale(20)">`, `ComputedSize()` is 0.4px, but we draw text with a font of which size is 0.4 * 20 * 3 == 24 screen pixels. 'Scale factor' is 60, and scaled font size is 24.

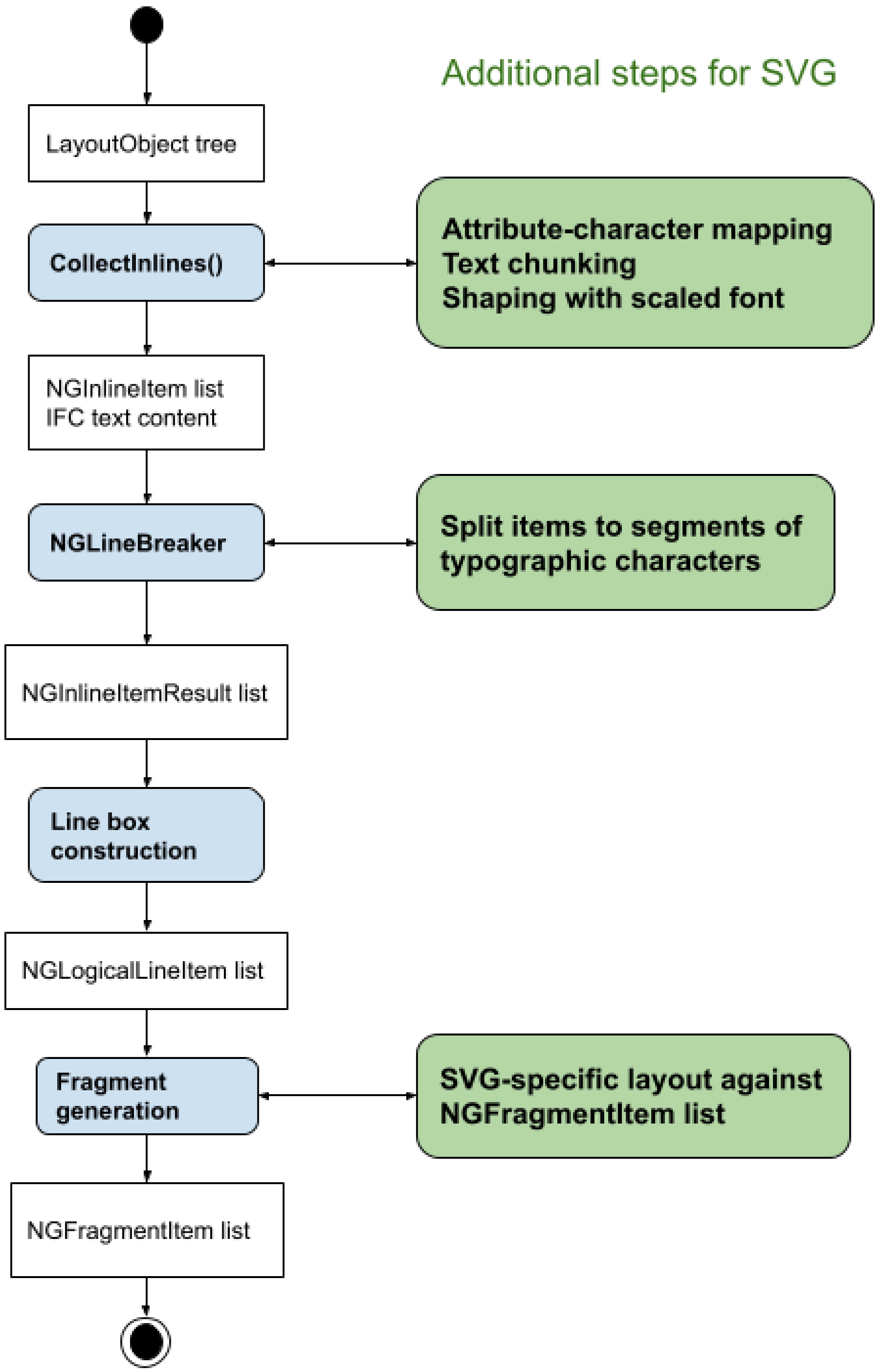
Paint

`SVGInlineTextBoxPainter` is the core part. It paints `SVGTextFragment` instances in an `SVGInlineTextBox`. It supports the paint server feature, paint-order attribute, and has its own text-decoration painting code.

Hit-Testing

`SVGInlineTextBox::NodeAtPoint()` ⇨ `SVGInlineTextBox::HitTestFragments()`

Proposed LayoutNG text processing



Additional steps for SVG

LayoutObject tree

CollectInlines()

Attribute-character mapping
Text chunking
Shaping with scaled font

NGInlineItem list
IFC text content

NGLineBreaker

Split items to segments of typographic characters

NGInlineItemResult list

Line box construction

NGLogicalLineItem list

Fragment generation

SVG-specific layout against NGFragmentItem list

NGFragmentItem list

Layout

`CollectInlinesInternal()` in ng_inline_node.cc and/or `NGInlineItemsBuilder` should take care of text chunks. We should add BiDi control characters so that each of the text chunks are handled as `unicode-bidi:isolate` before running `SegmentBidiRuns()`. It will fix crbug.com/341932 and crbug.com/1034464.

- Making text chunks at this timing would be the best because text chunks affect font shaping results. We do `ShapeText()` just after `SegmentText()`, which includes `SegmentBidiRuns()`.
- A single text chunk should be represented by one or more `NGInlineItem`s.
- We need to compute the addressable character index in this step because `x/y` attribute values are mapped to addressable characters, and the attribute values produce text chunks.
- Examples:
  `<svg:text>foo עִבְרִית bar</svg:text>` ➪ a single text chunk
  `<svg:text>foo <svg:tspan>עִבְרִית</svg:tspan></svg:text>` ➪ a single text chunk
  `<svg:text y="0 -10"><svg:tspan>foo</svg:tspan></svg:text>` ➪ two text chunks; "f" and "oo"
  `<svg:text>foo <svg:tspan y="10 20">bar</svg:tspan> עִבְרִית</svg:text>` ➪ 3 text chunks; "foo ", "b", and "ar עִבְרִית"
- `NGSvgTextLayoutAttributesBuilder` is responsible for this step. It maps attributes to addressable characters, and we store its result to `NGInlineNodeData`.

Fonts referred to in `NGInlineNode::ShapeText()` should be scaled fonts. This is necessary to use text metrics computed by the inline layout in the SVG specific text layout algorithm. Of course `NGFragmentItem::rect_` will have scaled metrics, and we need to adjust it later or stop using it in SVG. `ShapeResult` and `ShapeResultView` referred from <svg:text> are always for scaled fonts.

`NGLineBreaker::HandleText()` ~~creates `NGInlineItemResult` for each of typographic characters of an input `NGInlineItem`.~~ splits an input `NGInlineItem` into multiple segments of typographic characters. A single segment consists of characters which we can paint together. `x/y/dx/dy` attributes create a new segment from the corresponding character, `rotate` attribute and <svg:textPath> make a new segment for each of the characters.
Note: The definition of "segment" is the same as `SVGTextFragment`.
Note: Initially we thought split by characters was simple and enough. However we found that the split-by-character didn't work well for text selection painting and negative text-shadow.

The inline layout handles `alignment-baseline`, `baseline-shift`, and `dominant-baseline` properties.

Then, we proceed the inline layout just before creating `NGFragmentItems`. In `NGFragmentItemsBuilder::ToFragmentItems()`, we kick the SVG specific text layout algorithm starting with "1. Setup". It updates `NGFragmentItem`s stored in `NGFragmentItemsBuilder::item_`.

- `NGSvgTextLayoutAlgorithm` implements the algorithm.
- The algorithm uses the result of `NGSvgTextLayoutAttributesBuilder`.
- The outcome of the algorithm is an array of per-character information. See 1.3. of the algorithm in the specification. A new `struct SvgPerCharacterInfo` represents it.

```
// Rough idea of what SvgPerCharacterInfo looks like.
struct SvgPerCharacterInfo {
  base::optional<float> x_;
  base::optional<float> y_;
  base::optional<float> rotate_;
  bool hidden_ = false;
  bool addressable_ = false;
  bool middle_ = false;
  bool anchor_chunk_ = false;

  // Should have a pointer to a NGFragmentItem / ShapeResultView?
};
```

- Introduce `kSVGText` type of `NGFragmentItem`. It's similar to kText, but it has SVG-specific data.

```
struct NGSvgFragmentData {
  scoped_refptr<const ShapeResultView> shape_result;
  NGTextOffset text_offset;
  FloatRect rect;
  AffineTransform transform;  // represents 'rotate'.
};

class NGFragmentItem {
  …
  struct SVGTextItem {
    std::unique_ptr<NGSVGFragmentData> data;
  };
```

`kSVGText` `NGFragmentItem` is created from a `kText` `NGFragmentItem` and an array of `SvgPerCharacterInfo`. A single `NGFragmentItem` represents a segment of typographic characters.

As for svgwg:537, we should follow the current specification for compatibility with the current implementation for now. That is to say, index attributes such as x/y/dx/dy/rotate are mapped to Unicode code points, not grapheme clusters. We think it's not so hard to adopt grapheme clusters in the future.

Note: As for connected glyphs such as ligatures, the specification defines how to handle an attribute value pointing to a middle of a connected glyph. Firefox follows it, and the current Chrome and Safari break connected glyphs.

Note: Firefox supports `white-space:pre*`. Supporting it would be very easy. (crbug.com/366558) However, it's not in the scope of the project.

Paint and HitTesting

We should handle `kSVGText` `NGFragmentItem` in addition to `kText` `NGFragmentItem` in the paint phase and the hittesting phase. We need to take care of scaled fonts, the paint server feature, `paint-order`, and per-character transform.

Estimated workload

Probably we can't reuse existing SVG text code such as `SVGTextLayoutEngine`.
We guess this would need 2 quarters * engineers. We can proceed the first paragraph in the Layout section and the remaining part in parallel though the former is much easier than the latter.

Known differences from the legacy SVG text

- Bidi reordering is scoped within text chunks split by x/y attributes.
- dx/dy/textLength attributes and <textPath> don't break a single glyph consisted of multiple code points, such as ligatures and Emoji ZWJ sequences
- Text without positioning just after <textPath> starts on the end of the path, not the end of the text in the <textPath>.
- Underlines and overlines are thicker than the legacy SVG text, but same as HTML.
- textLength attribute doesn't affect getComputedTextLength() result.
- The list of bugs fixed by the project
    - Issue 245618: SVG bidi rtl behaves incorrectly when there are multiple spans
    - Issue 341932: tspan with x/y attributes should create text chunks
    - Issue 347126: SVGTextElement.getStartPositionOfChar and getCharNumAtPosition disagree
    - Issue 360314: [SVG] Incorrect alignment of the first tspan in text element when RTL direction is set
    - Issue 360315: [SVG] Incorrect alignment of incrementally shifted tspan elements when RTL direction is set
    - Issue 374526: Complex text is broken on text path
    - Issue 375258: textLength not handled correctly for tspan elements

    - Issue 597055: Vertical SVG text lays out rtl text backwards
    - Issue 622336: SVGTextContentElement.getSubStringLength() returns non-zero width for zero-width non-joiner character
    - Issue 631903: SVG text appearing in wrong location
    - Issue 917770: svg tspan: diagonal-fractions and letter-spacing do not work together
    - Issue 927214: Thai vowel character placement incorrect on SVG Text Path.
    - Issue 936382: SVG textlength not properly inherited from text element to textPath
    - Issue 967655: text svg elements with high letter-spacing are not underlined properly
    - Issue 973581: letter-spacing CSS property breaks bidi in SVG
    - Issue 1034464: SVG file formatting affects rendering
    - Issue 1083726: SVG text element with writing-mode="tb" text-anchor="middle" not positioned correctly
    - Issue 1132249: Devanagari combining characters become unjoined when using dx attribute on SVG text (wrong appearance)
    - Issue 1155114: SVGTextContentElement.getComputedTextLength() doesn't account for letter spacing.
    - Issue 1180484: Text does not render correctly in SVG file

---

[EOF]