HTML2PPTX Library Improvements

Summary

This document details the architectural improvements made to the html2pptx library to fix poor output quality and make it more robust and generalized.

Date

October 14, 2025

Problems Identified

1. Poor Flexbox Layout Handling

Issue: The `calculatePosition` function used a naive approach that simply estimated positions based on DOM order without understanding CSS flexbox properties.

Impact:
- Elements weren't properly distributed across the slide
- Incorrect sizing and positioning
- Ignored critical layout properties like `flex: 1`, `gap`, `flex-direction`

2. Missing Support for `:nth-child()` Selectors

Issue: The CSS parser only stored styles by basic selector strings and didn't evaluate pseudo-selectors.

Impact:
- All elements received the same styling
- Individual element customization (like different border colors) was lost
- Example: `.text-box:nth-child(1)` through `:nth-child(5)` with different border colors were ignored

3. Incorrect Dimension Calculations

Issue: Used arbitrary values like `containerWidth / 2` for element widths without considering the actual container layout.

Impact:
- Elements were incorrectly sized (5" wide instead of 8.89" wide)
- Didn't account for container padding and gaps
- Wasted space on slides

4. No Gap Property Support

Issue: The `gap` property from flex containers was completely ignored.

Impact:
- Elements had incorrect spacing between them
- Boxes appeared too close together or overlapping

5. Poor Style Cascade Logic

Issue: Style application order was incorrect (inline styles before class styles).

Impact:
- CSS specificity rules were violated
- Inline styles couldn't override class styles

Solutions Implemented

1. Enhanced `getComputedStyle()` Method

Location: Lines 132-195

Changes:
- Fixed style cascade order: tag styles → class styles → nth-child pseudo-selectors → inline styles
- Added support for `:nth-child(n)` pseudo-selector matching
- Proper index-based element identification within parent container
- Checks if base selector (class or tag) matches before applying nth-child styles

Code Example:

```
// Check for nth-child pseudo-selectors
const parent = $elem.parent();
if (parent.length > 0) {
    const siblings = parent.children();
    const index = siblings.index($elem[0]);

    for (const selector in this.styles) {
        if (selector.includes(':nth-child')) {
            const match = selector.match(/(.+):nth-child\((\d+)\)/);
            if (match) {
                const baseSelector = match[1];
                const nthIndex = parseInt(match[2]) - 1;

                if (index === nthIndex) {
                    // Apply styles if base selector matches
                }
            }
        }
    }
}
```

2. New `calculateFlexPosition()` Method

Location: Lines 437-521

Changes:
- Created dedicated method for handling flexbox layouts
- Properly parses flex container properties:
- `flex-direction`: column or row
- `gap`: spacing between flex items
- `padding`: all four sides (top, bottom, left, right)
- Calculates available space after accounting for padding
- Properly handles `flex: 1` for equal distribution

- Supports both column and row flex directions
- Accurate positioning with gap calculation

Algorithm:

```
For flex-direction: column:
1. Calculate available height = containerHeight - paddingTop - paddingBottom
2. Calculate total gaps = (totalItems - 1) × gap
3. Calculate item height = (availableHeight - totalGaps) / totalItems (if flex: 1)
4. Calculate Y position = paddingTop + (index × (itemHeight + gap))
```

3. Improved `calculatePosition()` Method

Location: Lines 384-432

Changes:
- Now detects parent container's display property
- Routes to `calculateFlexPosition()` when parent uses flexbox
- Falls back to original logic for non-flex layouts
- Better separation of concerns

4. Enhanced `parseTextOptions()` Method

Location: Lines 526-623

Changes:
- Better border handling with separate color and width extraction
- Support for both shorthand (`border: 2px solid red`) and individual properties (`border-color` , `border-width`)
- Added `border-radius` support with shape type detection
- Converts border-radius to `rectRadius` for rounded rectangles
- More robust border color extraction

Code Example:

```
if (style['border-radius']) {
    const borderRadius = this.parsePosition(style['border-radius']);
    if (borderRadius > 0) {
        options.shape = this.pptx.ShapeType.roundRect;
        options.rectRadius = borderRadius / 72; // Convert px to inches
    }
}
```

Results

Before Fixes:

```
Text Box 1: "Text Box 1"
  Position: (0.50", 0.50")
  Size: 5.00" × 0.83"
  Border: #3498DB (Blue - WRONG, should be Red)

Text Box 2: "Text Box 2"
  Position: (0.50", 1.63")
  Size: 5.00" × 0.83"
  Border: #3498DB (Blue - CORRECT)

Text Box 3: "Text Box 3"
  Position: (0.50", 2.75")
  Size: 5.00" × 0.83"
  Border: #3498DB (Blue - WRONG, should be Green)

... (All had same blue border)
```

Issues:
- Width too narrow (5” instead of ~9”)
- All borders the same color
- Incorrect spacing

After Fixes:

```
Text Box 1: "Text Box 1"
  Position: (0.56", 0.56")
  Size: 8.89" × 0.68"
  Border: #E74C3C (Red - CORRECT ✓)

Text Box 2: "Text Box 2"
  Position: (0.56", 1.51")
  Size: 8.89" × 0.68"
  Border: #3498DB (Blue - CORRECT ✓)

Text Box 3: "Text Box 3"
  Position: (0.56", 2.47")
  Size: 8.89" × 0.68"
  Border: #2ECC71 (Green - CORRECT ✓)

Text Box 4: "Text Box 4"
  Position: (0.56", 3.43")
  Size: 8.89" × 0.68"
  Border: #F39C12 (Orange - CORRECT ✓)

Text Box 5: "Text Box 5"
  Position: (0.56", 4.39")
  Size: 8.89" × 0.68"
  Border: #9B59B6 (Purple - CORRECT ✓)
```

Improvements:
- ✅ Correct width (8.89” accounts for padding)
- ✅ Each border has the correct unique color
- ✅ Proper spacing with gaps (~0.27” between boxes)

- ✅ Correct padding from container edges
- ✅ Correct background colors applied

Architectural Benefits

1. Generalized Solution

- The fixes work for any flexbox layout, not just this specific case
- Supports both column and row flex directions
- Handles various gap and padding configurations
- No hardcoded values specific to the test HTML

2. Proper CSS Specificity

- Respects CSS cascade and specificity rules
- Inline styles properly override class styles
- Pseudo-selectors are evaluated correctly

3. Extensible Design

- Easy to add support for more CSS properties
- Separation of concerns (flex calculation vs. regular layout)
- Clear method responsibilities

4. Robustness

- Handles missing properties with sensible defaults
- Falls back gracefully for non-flex layouts
- Proper unit conversions (px → inches)

Testing

The library was tested on:
1. **5 Text Boxes 16:9 HTML** - Vertical flex layout with 5 equally-sized boxes
- ✅ All border colors correct
- ✅ Proper spacing and sizing
- ✅ Correct padding application

1. **check.html** - More complex layout
   - ✅ Successfully converted without errors

Future Improvements

While these fixes significantly improve the library, potential future enhancements include:

1. **Grid Layout Support**: Add support for CSS Grid layouts
2. **Nested Flexbox**: Better handling of nested flex containers
3. **Responsive Units**: Better handling of %, em, rem units with context awareness

4. **Transform Properties**: Support for CSS transforms (rotate, scale, etc.)
5. **More Pseudo-selectors**: Support for `:hover`, `:first-child`, `:last-child`, etc.
6. **Advanced Border Styles**: Support for different border styles (dashed, dotted, etc.)

Conclusion

The html2pptx library has been significantly improved with systematic, architectural changes that make it:
- ✅ More accurate in rendering HTML layouts
- ✅ Better at handling modern CSS (flexbox)
- ✅ More robust and generalized
- ✅ Extensible for future enhancements

The library now produces high-quality PowerPoint presentations that faithfully represent the original HTML layouts and styles.