

# SVG In Motion

- Freelance front-end dev.
- Wrote the Codrops CSS Reference.
- Co-authored Smashing Book 5
- SVG advocate

# What's with all the vowel letters?

- Swaïdaan
- Swaydein
- Sw- "eye"-daan

## What we will cover (not in this particular order):

1. How to Embed SVG
2. The SVG viewBox
3. Animating the viewBox
4. SVG Animation Techniques: CSS, SMIL, JavaScript
5. SVG Sprite Animation Techniques
6. SVG Line Drawing Effects

# 6 Embedding Techniques

```

```

- 1) Image can be cached (Requires HTTP request. But: HTTP/2!).**
- 2) No CSS interactions.**
- 3) No scripting.**
- 4) CSS animations work only if defined inside <svg>**

```
<picture>
  <source type="image/svg+xml" srcset="path/to/image.svg">
    
</picture>
```

**Same as `<img>`**

**Fallback is included inside `<picture>`**

```
background-image: url(path/to/mySVG.svg);
```

**Same as <img>, but the SVG *can be* cached as part of the style sheet if it is inlined using data URIs.**

```
<object type="image/svg+xml" data="mySVG.svg">
  <!-- fallback here; beware of multiple requests -->
</object>
```

- 1) Image cached.**
- 2) Scripting.**
- 3) Default fallback mechanism.**
- 4) CSS animations and interactions work only if defined inside <svg>**

```
<iframe src="mySVG.svg">  
  <!-- fallback here -->  
</iframe>
```

**Same as `<object>`**

```
<svg xmlns="http://www.w3.org/2000/svg" ...>
  <!-- svg content -->
</svg>
```

- 1) Image is not cached.**
- 2) No extra HTTP requests.**
- 3) Scripting.**
- 4) CSS animations and interactions.**

Need to provide Fallback? Read this guide: <https://css-tricks.com/a-complete-guide-to-svg-fallbacks/>

# Which embedding technique should you choose?!

- Is the SVG animated?
- Is it interactive?
- What kind of animation? (Does it require JS?)
- What browser support do I need (for the animation)?
- What kind of content and fallback do you need? (e.g. infographics)

# Quick Animation Recap

## Embedding Technique

## Animations

## Interactions?

<img>

CSS (, SMIL)

No

url();

CSS (, SMIL)

No

<picture>

CSS (, SMIL)

No

<iframe>

CSS, JavaScript (, SMIL)

Yes

<object>

CSS, JavaScript (, SMIL)

Yes

<svg>

CSS, JavaScript (, SMIL)

Yes

Embedding Technique	CSS Animations Location	JS Animations Location
<img>	Inside <svg>	-
url();	Inside <svg>	-
<picture>	Inside <svg>	-
<iframe>	Inside <svg>	Anywhere
<object>	Inside <svg>	Anywhere
<svg>	Anywhere	Anywhere

## Accessing an embedded SVG document (<object>):

```
window.onload=function() {
    // Get the Object by ID
    var a = document.getElementById("svgObject");

    // Get the SVG document inside the Object tag
    var svgDoc = a.contentDocument;

    // Get one of the SVG items by ID;
    var svgItem = svgDoc.getElementById("svgItem");

    // Set the colour to something else
    svgItem.setAttribute("fill", "lime");
};
```

\*script accessing the SVG from main page must be CORS compatible

# To Optimize or Not To Optimize

# SARA SOUEIDAN

[SPEAKING](#)    [WORKSHOPS](#)    [ARTICLES](#)    [ABOUT ME](#)    [HIRE ME](#)

## Useful SVGO[ptimization] Tools

Published January 26, 2015 |

Estimated Reading Time: 7 min

One of the steps you need to do when working with SVG is optimizing the SVG code after exporting it from the editor and before embedding in on your web page. For that, several standalone optimization tools exits. The two tools I usually mention in my [articles](#) and [talks](#) are [Peter Collingridge's online editor](#), and [SVGO](#). In this article, I'm going to introduce you to a new SVGO Tool that provides us with everything Peter's tool does, and a bit more.

@SaraSoueidan [SaraSoueidan.com](http://SaraSoueidan.com)

This is not to say that peter's tool is no longer useful—it certainly is. But if you use SVGO,

Optimization tools usually change the SVG document structure, and can break any animations you've set up.

# CSS, SMIL or JavaScript?

- SMIL has been deprecated.
- Use CSS only for simple animations.
- Use JavaScript for complex animations.

**Re SMIL: You can use [a polyfill](#) if you need to make it work. Follow [this guide](#) for details on how to use it.**

Animation	Technique/Tool
Transforms (scale, rotate, translate, skew)	CSS, JavaScript Recommended: JavaScript
Path Morphing	JavaScript
Line Drawing	CSS, Javascript Recommended: JavaScript
Color and other simple animations and transitions	CSS, JavaScript Recommended: CSS whenever possible

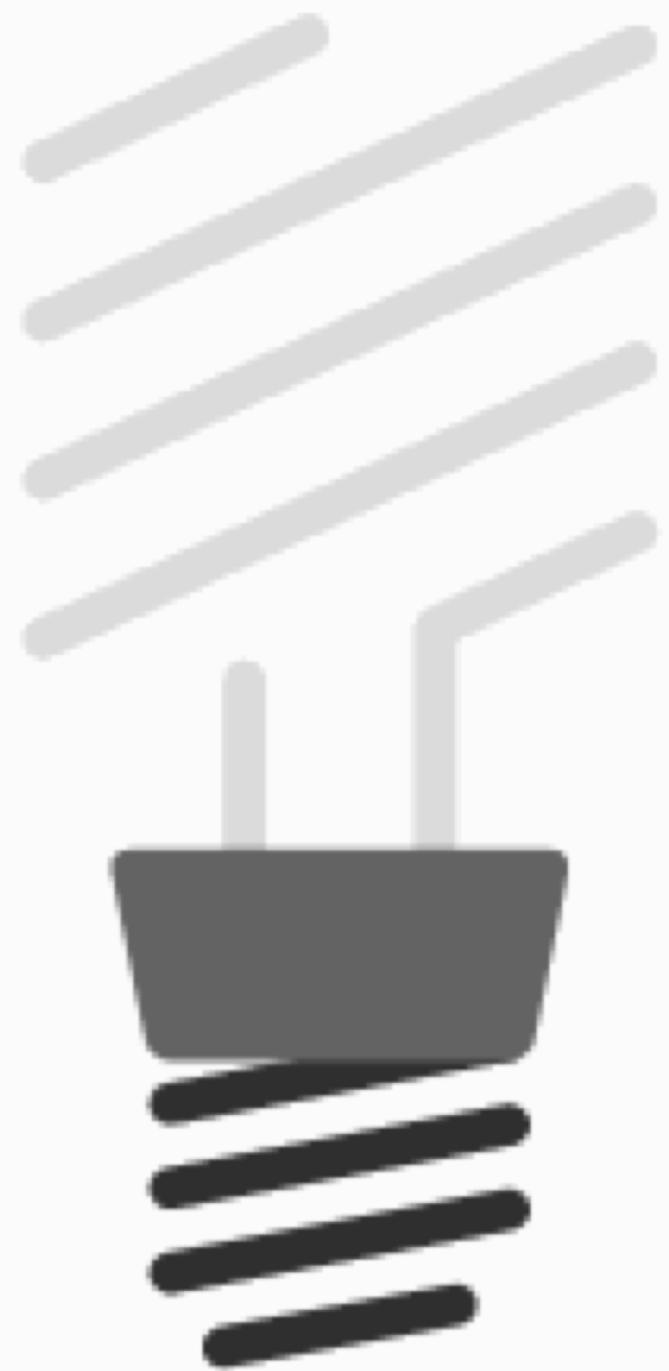
# Animating SVG with CSS: The Good, the Bad and the Ugly

- 1) The [list of SVG attributes that can be animated with CSS](#) is limited (in SVG 1.1).
- 2) In SVG2, the [list](#) is extended.
- 3) Most properties are animated on SVG elements just like they are animated on HTML elements.

Shared with CSS	SVG-Only
font, font-family, font-size, font-size-adjust, font-stretch, font-style, font-variant, font-weight, direction, letter-spacing, text-decoration, unicode-bidi, word-spacing, visibility, text-rendering, writing-mode, clip-path, mask-opacity, filter, pointer-events, image-rendering, clip, color, cursor, display, overflow	clip-rule, flood-color, flood-opacity, stop-opacity, kerning, text-anchor, color-profile, color-rendering, fill, fill-opacity, fill-rule, marker, marker-end, marker-mid, marker-start, stroke, stroke-width, stop-color, lighting-color, enable-background, dominant-baseline, color-interpolation-filters, color-interpolation, glyph-orientation-horizontal, glyph-orientation-vertical, shape-rendering, baseline-shift, alignment-baseline, stroke-miterlimit, stroke-linejoin, stroke-linecap, stroke-dashoffset, stroke-dasharray, stroke-opacity

Using these properties, you can apply animations and transitions to SVG elements just like you would with HTML elements, using the main CSS selectors:

```
rect {  
}  
  
#myPath {  
}  
  
.circle {  
}
```



## PROOF OF CONCEPT

## Light up your icons

You love CSS. So does Iconic. Each icon is designed with styling in mind. Individual elements of each and every icon can be given a completely custom look. So, by all means, go crazy.



S | Elements Network Sources Timeline Profiles Resources Audits Console Grunt

```
<article>
  <div class="container">
    ::before
    <div class="row">
      ::before
      <div class="col-md-6 col-md-offset-3">
        <h4>Light up your icons</h4>
        ><p>...</p>
        <p class="text-center" style=" margin-bottom: 80px; ">
          <svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" style=" width: 100%; height: 100%; >
            <circle class="iconic-lightbulb-screws" cx="357" cy="357" r="33" />
            <g class="iconic-lightbulb-screws">...</g>
            <g class="iconic-lightbulb-light">
              <defs>...</defs>
              <g clip-path="url(#iconic-lightbulb-light-clip)">...</g>
              <line x1="478" y1="129" x2="254" y2="233" />
              <line x1="354" y1="125" x2="254" y2="173" />
              <line x1="478" y1="189" x2="254" y2="293" />
              <line x1="478" y1="249" x2="254" y2="353" />
            </g>
            <path class="iconic-lightbulb-base" d="M292.431,505c0.431,0 85.431,0 85.431,0" />
          </svg>
        </p>
      </div>
    </div>
  </div>
</article>
```

html body section article div div div p.text-center svg.svg-inject.iconic.iconic-lightbulb

Styles Event Listeners DOM Breakpoints Properties

:active       :hover  
 :focus       :visited

```
element.style {  
}  
.iconic-lightbulb:hover .iconic-lightbulb-light * {  
  stroke: #rgb(255, 204, 0);  
  stroke-width: 32;  
}  
.iconic-lightbulb .iconic-lightbulb-light * {  
  stroke-width: 16;  
  stroke: #rgb(221, 221, 221);  
  fill: none;  
  stroke-linecap: round;  
}
```

index.html:58  
index.html:50

# Fullscreen Overlay Styles & Effects

[HUGE INC](#)[CORNER](#)[SLIDE DOWN](#)[SCALE](#)[DOOR](#)[CONTENT PUSH](#)[CONTENT SCALE](#)[CORNER SHAPE](#)[LITTLE BOXES](#)[SIMPLE GENIE](#)[GENIE](#)

Genie effect with SVG animation.

[OPEN OVERLAY](#)

```
path {  
  transition: d .6s ease-in-out;  
}  
  
path.open {
```

```
  d: "M20.308,...";  
}
```

```
<path>
  <animate
    attributeName="d"
    dur="1440ms"
    repeatCount="indefinite"
    keyTimes=".."
    calcMode="spline"
    keySplines=".."
    values="M 0,0
      C 50,0 50,0 100,0
      100,50 100,50 100,100
      50,100 50,100 0,100
      0,50 0,50 0,0
    Z;

    M 0,0
    C 50,0 50,0 100,0
    100,50 100,50 100,100
    50,100 50,100 0,100
    0,50 0,50 0,0
    Z;
  ...
  ..."/>
```



<https://css-tricks.com/guide-svg-animations-smil/>

# CSS Transformations on SVG elements:

	transform-origin (default value)
HTML Elements (div, ::before, etc.)	<b>50% 50%</b> <i>(the center of the element itself, calculated relative to its box model)</i>
SVG Elements (circle, rect, etc.)	<b>0 0</b> <i>(top left corner of the SVG canvas, not of the element itself)</i>

```
<!DOCTYPE html>
<div style="width: 100px; height: 100px; background-color: orange"> </div>
<svg style="width: 150px; height: 150px; background-color: #eee">
    <rect width="100" height="100" x="25" y="25" fill="orange" />
</svg>
```

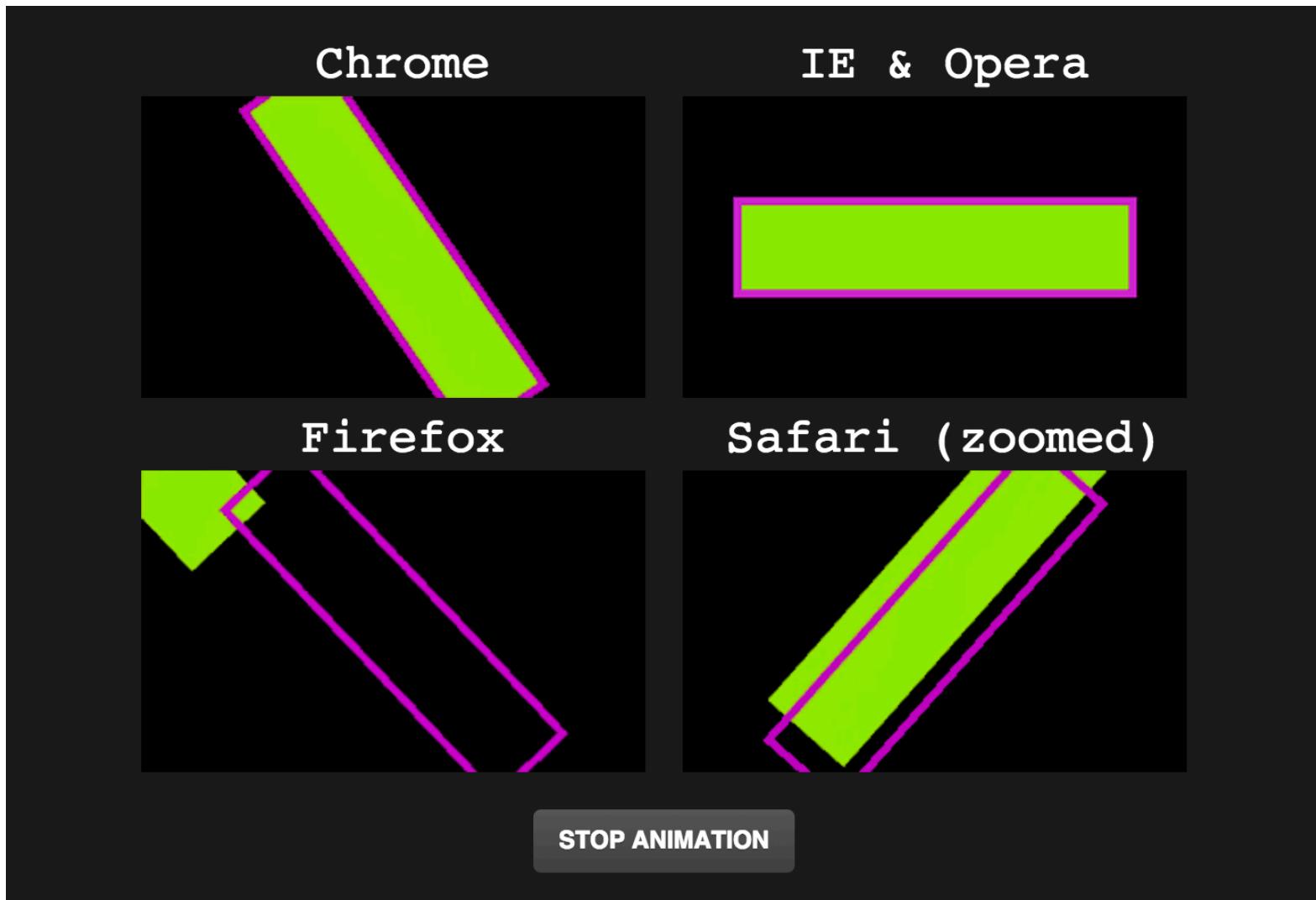


```
div { transform: rotate(45deg); }
```

```
rect { transform: rotate(45deg); }
```

# Change the transform-origin value:

1. Using percentage values: The value is set relative to the element's bounding box, which includes the stroke used to draw its border.
2. Using absolute values: The origin is set **relative to the entire SVG canvas**.



<http://greensock.com/svg-tips>

\*Firefox bug has been fixed in FF44.

`smoothOrigin:true`  
element doesn't jump when origin changes.



`smoothOrigin:false`  
element jumps when origin changes.



RESTART

<http://greensock.com/svg-tips>

CSS transforms don't work in IE/Edge. 😳

# CSS 3D Transforms on SVG Elements?

Don't count on them yet.

You can use standard sprite animation to achieve 3D animation effects.

# What about performance?

Some useful (possibly outdated) benchmarks: <https://css-tricks.com/weighing-svg-animation-techniques-benchmarks/>

# The SVG viewBox

The viewBox is made up of four numerical values: the first two determine the coordinates of the origin of the system, and the other two determine the system's finite dimensions.

```
viewBox = <min-x> <min-y> <width> <height>
```

These values are animatable.



By changing the value of the viewBox, you change the area of the canvas that is visible inside the SVG viewport. This enables you to zoom in to specific areas or objects.

But how do you determine the value for the viewBox to use, to zoom into a particular area of choice?

# SVG Bounding Boxes



The bounding box (or "bbox") of an element is the tightest fitting rectangle aligned with the axes of that element's user coordinate system that entirely encloses it and its descendants.

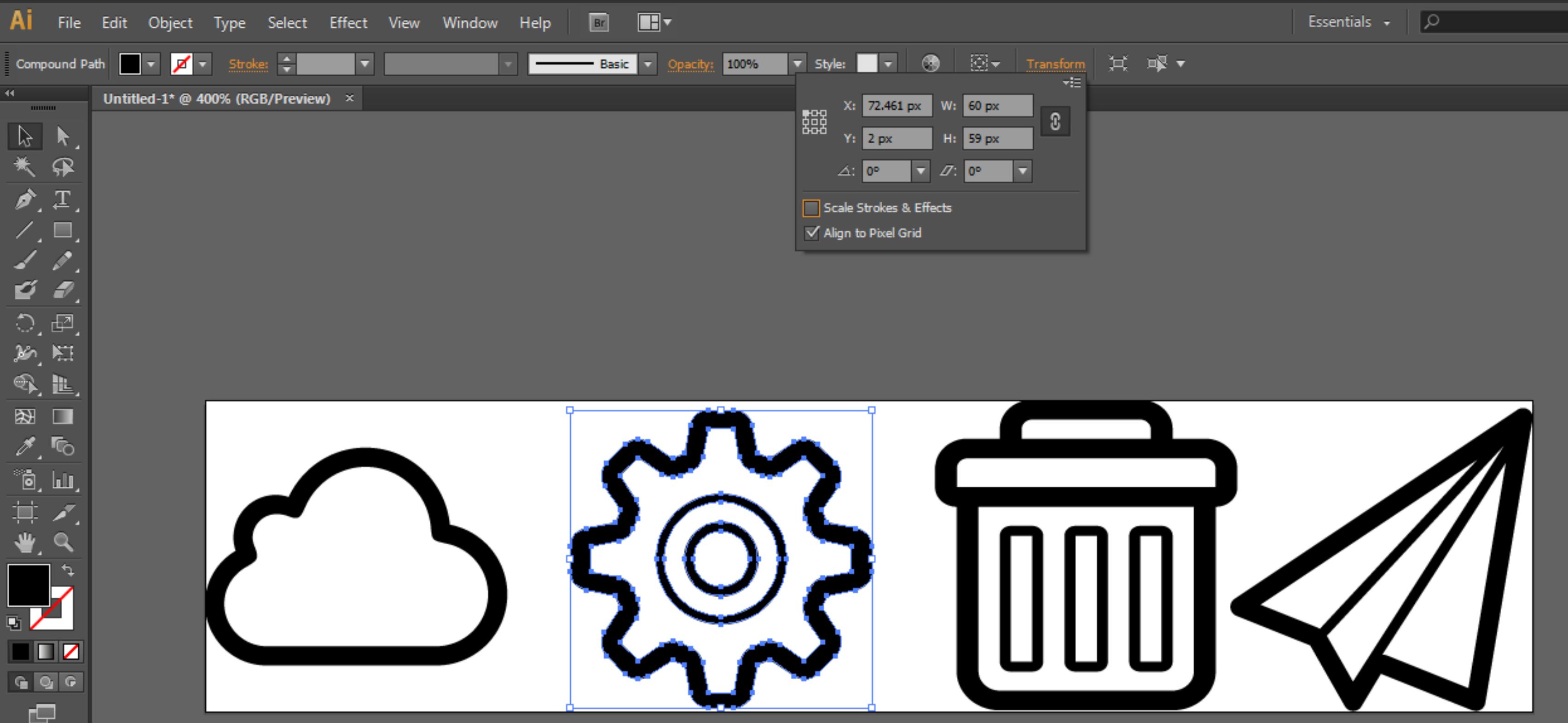
An SVG BBox has the same properties as those defining the SVG viewBox: x, y, width and height...

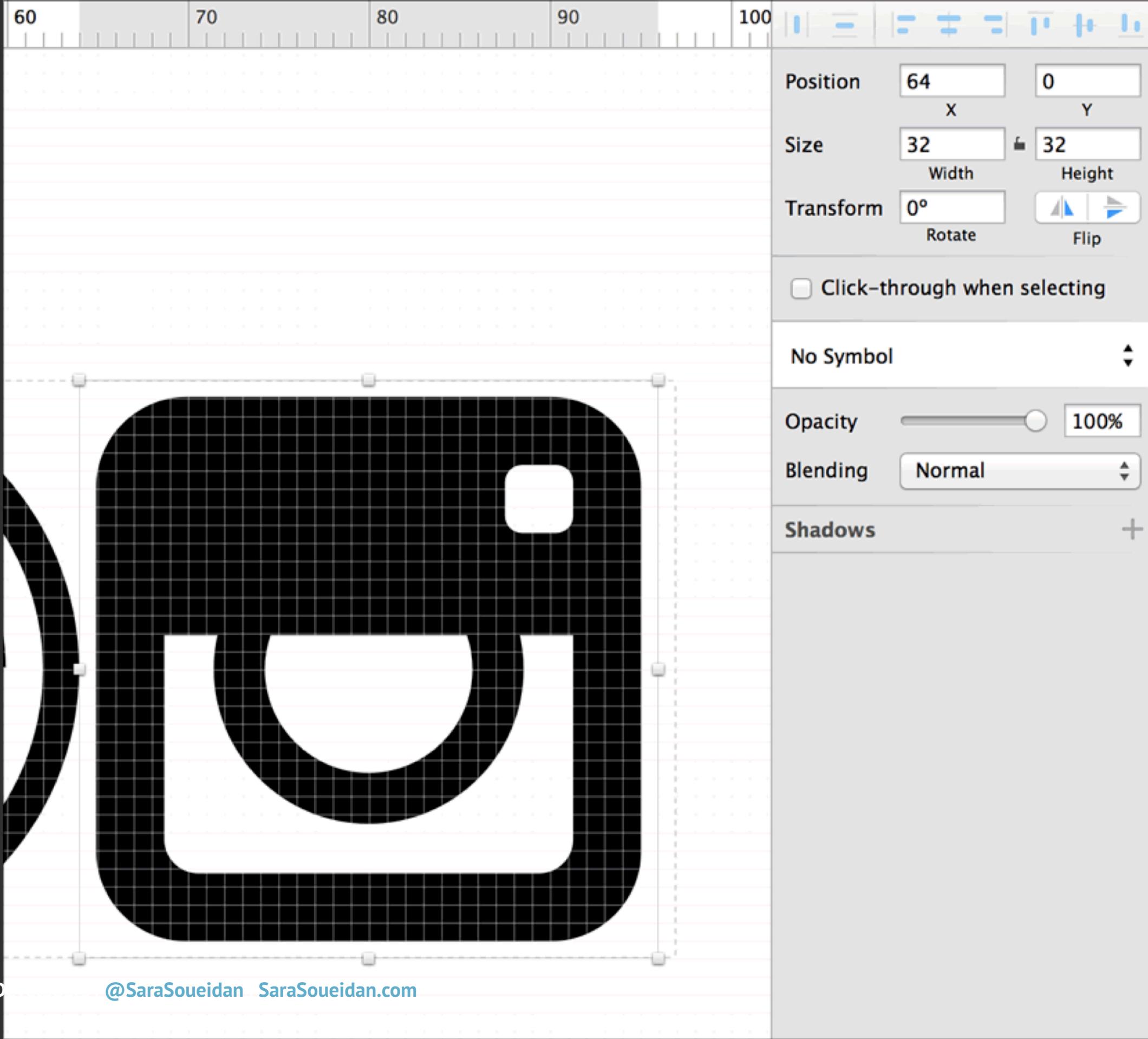
You can retrieve these properties using the `getBBox()` method:

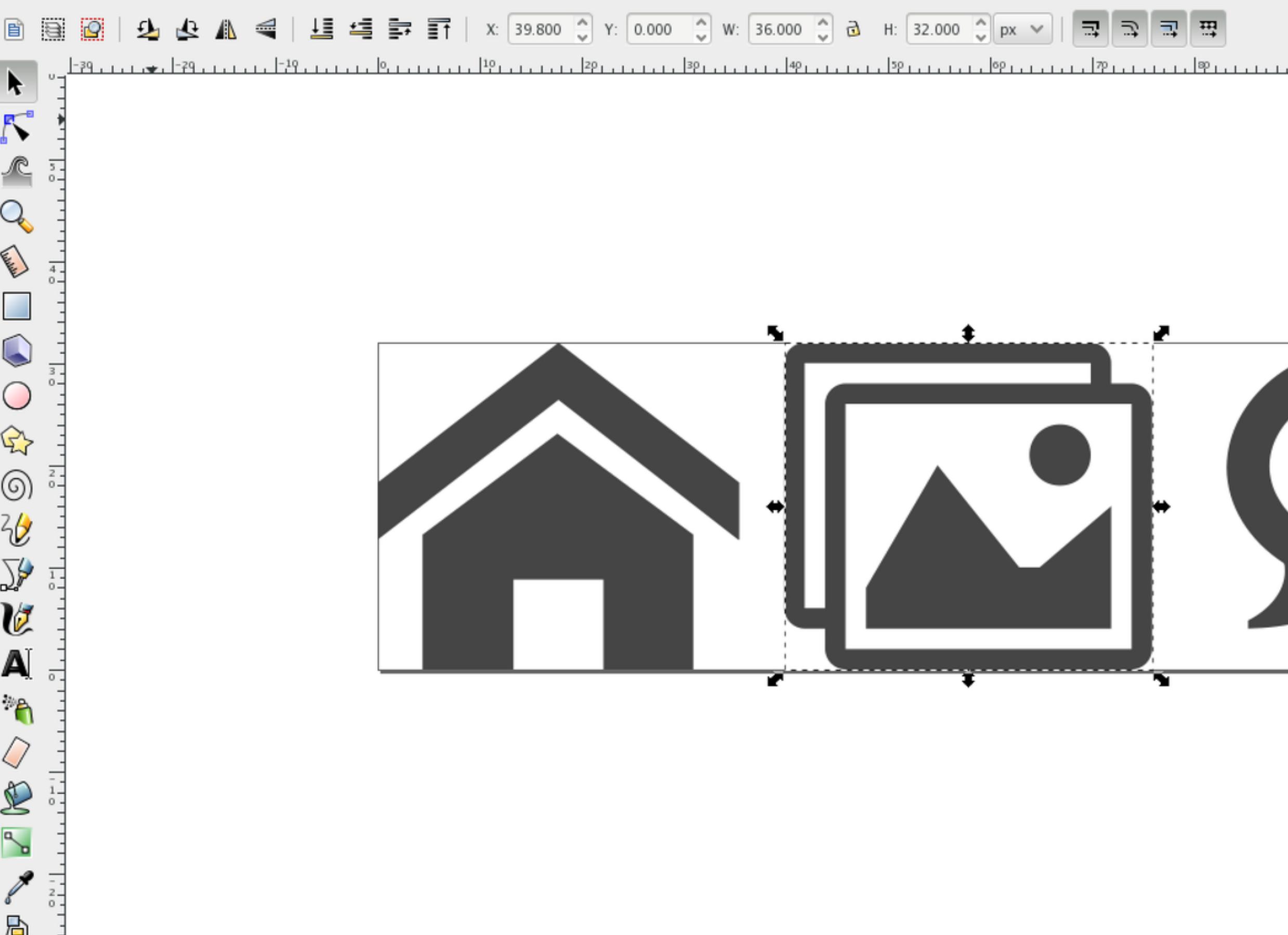
```
var svgElement = document.getElementById('el');
bbox = svgElement.getBBox();

console.log( bbox.x ) ;
console.log( bbox.y ) ;
console.log( bbox.width ) ;
console.log( bbox.height ) ;
```

OR, you can do it visually in your graphics editor of choice:







You can use the properties of an element's bounding box as values for the viewBox. This will make the element zoom in to the viewport. Example: zooming into specific areas/countries on a map...

```
var svg = document.getElementById('map'),  
    s = Snap(svg);  
  
var leb = document.getElementById('LB');  
var bbox = Snap(leb).getBBox();  
  
var clicked = false;  
function zoomIn(ev) {  
    if(clicked == false) {  
        clicked = true;  
        s.animate({viewBox: bbox.vb}, 1000);  
    }  
    else {  
        clicked = false;  
        s.animate({viewBox: "0 0 1100 700"}, 2000);  
    }  
}  
  
leb.addEventListener('click', zoomIn, false);
```

## Element.getBBox()

Returns the bounding box descriptor for the given element

object

**Returns:** bounding box descriptor:

{

cx:	number	x of the center,
cy:	number	y of the center,
h:	number	height,
height:	number	height,
path:	string	path command for the bo
r0:	number	radius of a circle that fully
r1:	number	radius of the smallest circl
r2:	number	radius of the largest circle
vb:	string	box as a viewbox commar
w:	number	width,
width:	number	width,
x2:	number	x of the right side,
x:	number	x of the left side,
y2:	number	y of the bottom edge,
y:	number	y of the top edge

Snap.svg is “the jQuery of SVG”.

The getBBox() method returns a *string*, making the bounding box values available as a viewBox command!

You can also use GreenSock—particularly useful for when yo have nested or sequenced animations:

```
tl
.to(svg, 2, {delay: 1, attr: {viewBox: "450 350 252 178"}})
.to(svg, 4, {attr: {viewBox: "60 350 252 178"}})
.to(svg, 2, {attr: {viewBox: "60 210 252 178"}}, "-=0.25")
.to(svg, 4, {attr: {viewBox: "444 210 252 178"}})
.to(svg, 2, {attr: {viewBox: "0 0 757.8 534.8"}})
```

Source: <http://codepen.io/dbj/pen/RWmQNJ?editors=1010>

Ideally, we'd be able to do this in CSS:

```
svg {  
  viewBox: 0 0 200 200;  
  transition: viewBox .4s ease-in-out;  
}  
  
svg:hover {  
  viewBox: 0 0 100 100;  
}  
  
@keyframes zoom {  
  from { viewBox: ....; }  
  to { viewBox: ...; }  
}
```

# SVG Sprite Animation Techniques

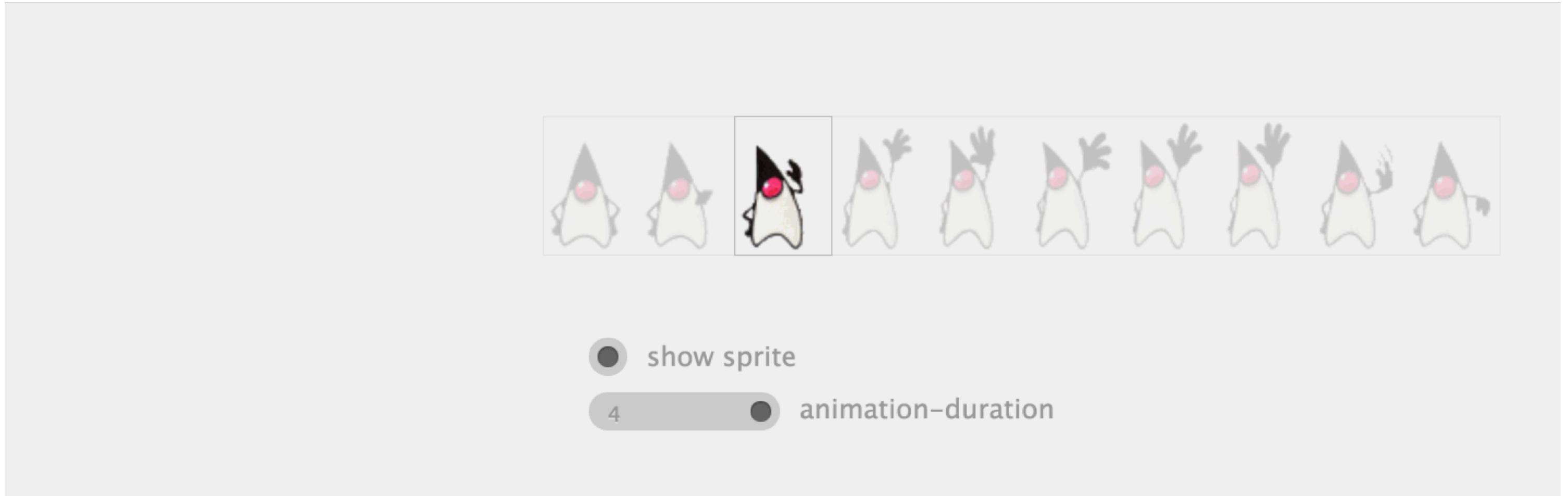
# Technique #1: SVG-as-Background Sprite Animation

This technique treats an SVG image just like a PNG sprite image.

- The SVG image would have all the "frames" drawn inside of it.
- The SVG is used as a background image on an element.
- The position of the SVG in the background positioning area is animated using `steps()`, thus showing only one frame at a time, over a specified period of time.

```
#element-with-animated-background {  
    /* ... */  
    background-image: url(path/to/image.svg)  
    width: width-of-each-frame;  
    height: height-of-the-image;  
    animation: frame-animation 1s steps(number-of-frames) infinite;  
}  
  
@keyframes frame-animation {  
    0% { background-position: 0px 0; }  
    100% { background-position: -(width-of-the-image) 0; }  
}
```

**Where 500px is the width of the image used as a background**



<http://simurai.com/blog/2012/12/03/step-animation/>

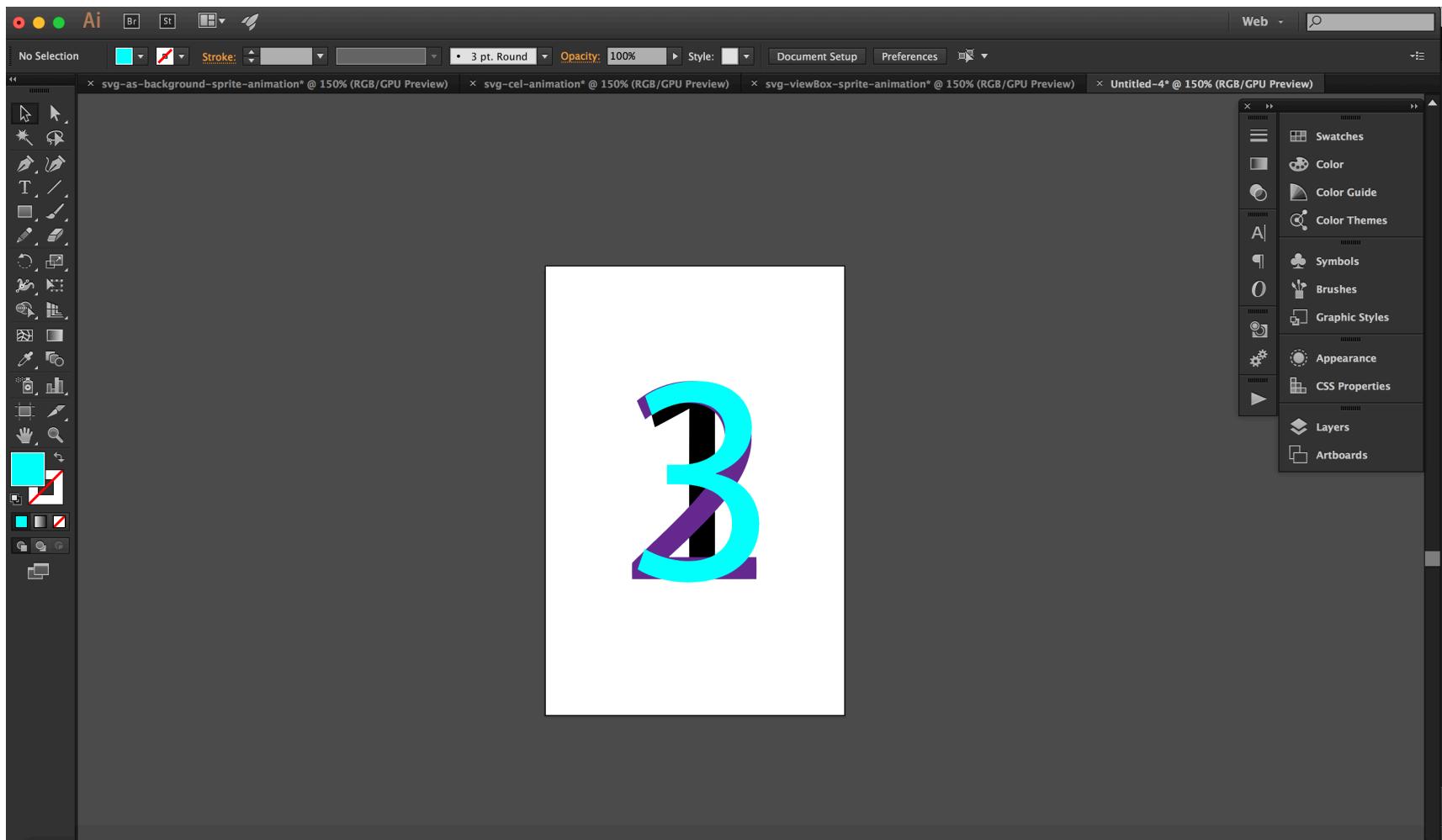
# Technique #2: (Independent) SVG Sprite Animation

# How is this different from Technique #1?

- The SVG is used as a foreground image, not an element's background image, so the frame animation happens *inside* the SVG.
- The frames inside the SVG are positioned on top of each other.
- Each frame is **animated into view** by changing its opacity.
- Frame animation uses `steps()` to animate each frame in *independently but concurrently*.

For example, suppose we have three frames in our animation, wrapped in a

```
<g class="frames"></g>
```



```
@keyframes frame-1-animation {
  0% {
    opacity: 1;
  }
  33.3333% {
    opacity: 0;
  }
}

.frames > :nth-child(1) {
  animation-name: frame-1-animation;
}

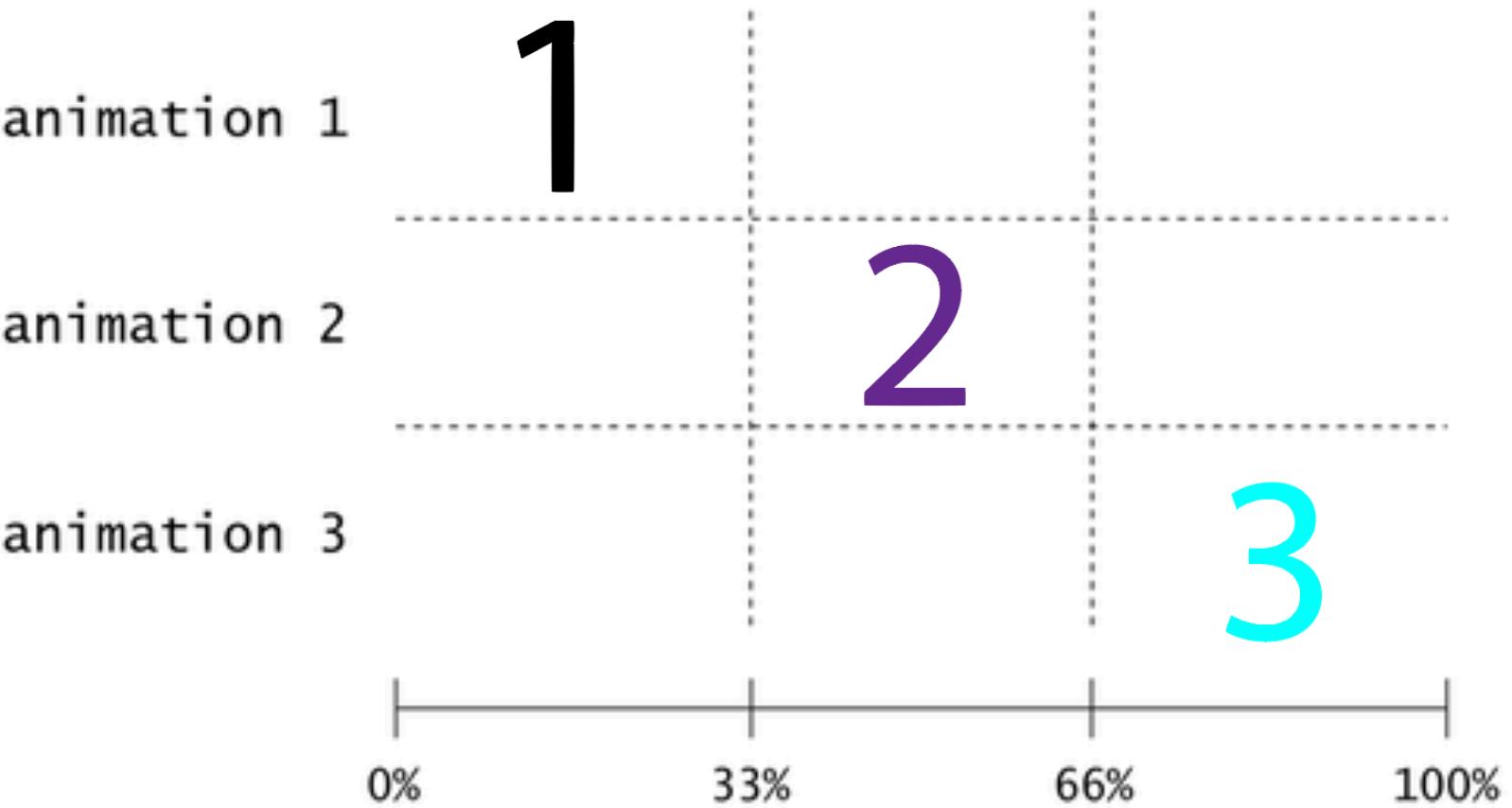
@keyframes frame-2-animation {
  33.3333% {
    opacity: 1;
  }
  66.6667% {
    opacity: 0;
  }
}

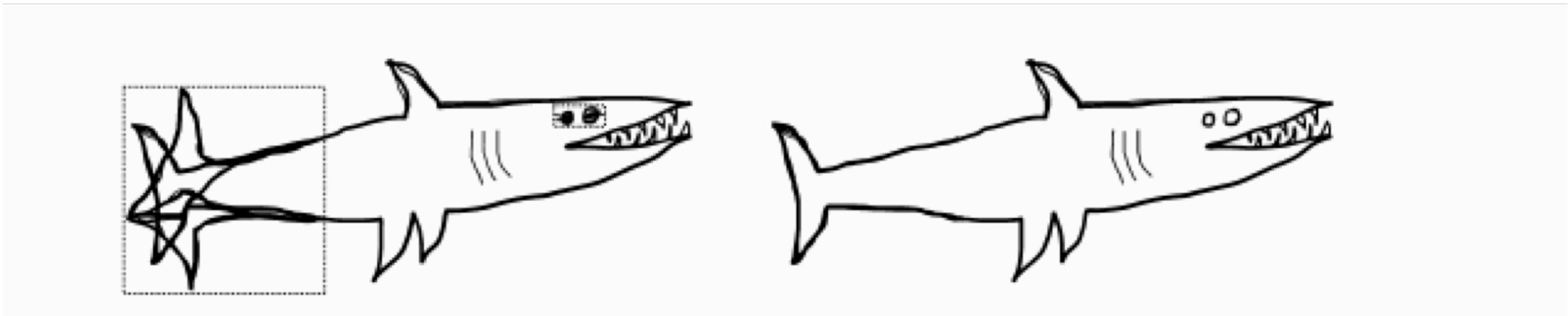
.frames > :nth-child(2) {
  animation-name: frame-2-animation;
}

@keyframes frame-3-animation {
  66.6667% {
    opacity: 1;
  }
  100% {
    opacity: 0;
  }
}

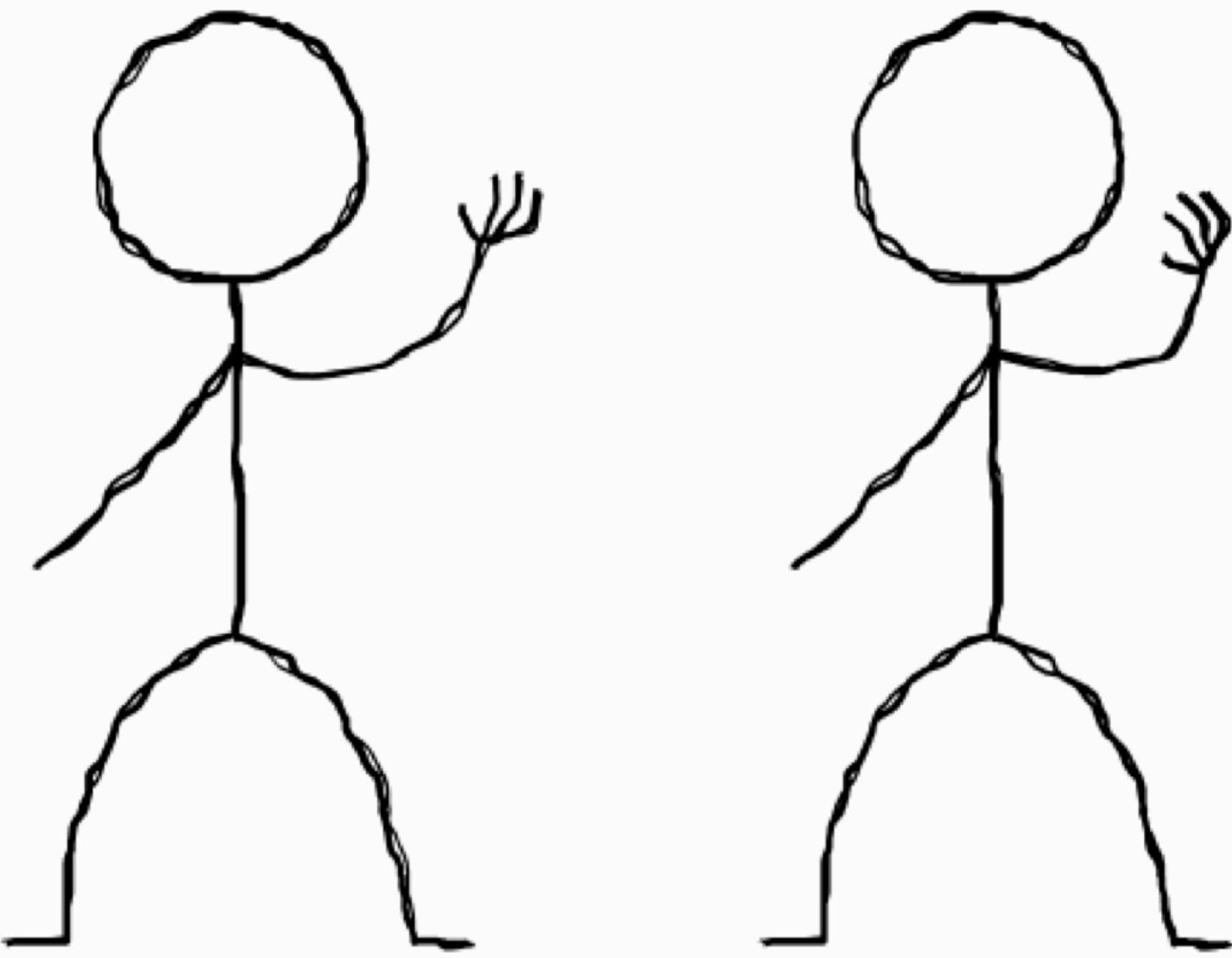
.frames > :nth-child(3) {
  animation-name: frame-3-animation;
}
```

In the animation timeline: each animation starts before the previous one finishes. The sum of all animation is the duration of the entire animation, so the **duration of visibility of each frame** is = total animation duration / 3.





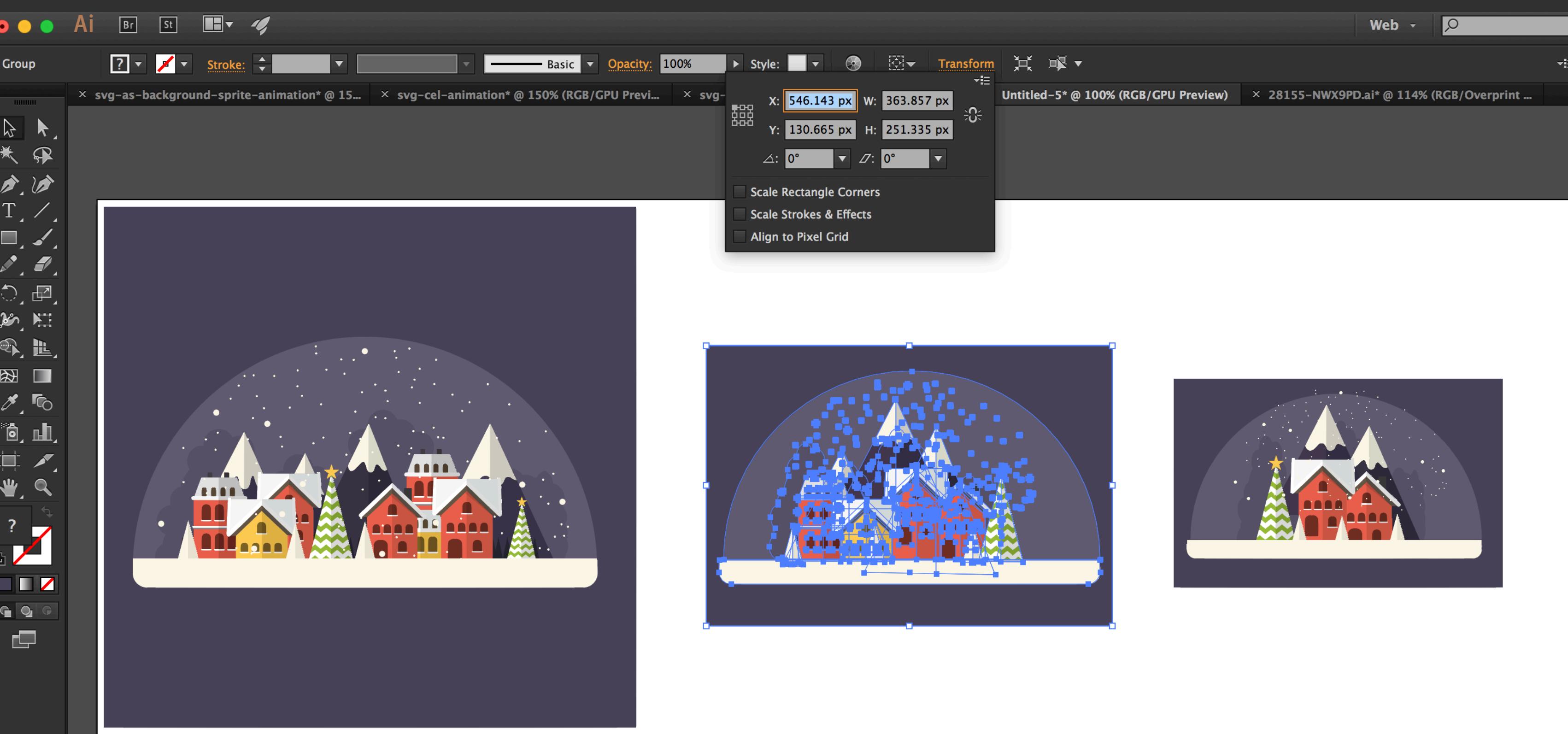
<http://www.smashingmagazine.com/2015/09/creating-cel-animations-with-svg/>



*A friendly wave: the left animation uses a smooth, keyframe-based transform, while the right animation plays three independently drawn pictures (or cels) one after another.*

# Technique #3: Responsive SVG Sprite Animation using viewBox

- The SVG is independent.
- Instead of multiple frames, the SVG contains multiple "scenes".
- Each scene is brought into view / shown inside the viewport depending on the size of the viewport.
- OR: The SVG is cropped so that only one scene is visible inside the viewport.
- viewBox is used to crop the SVG to each scene.



```
var svg = document.getElementById("svg");

// media query event handler
if (matchMedia) {
    var mq = window.matchMedia("(min-width: 500px)");
    mq.addListener(WidthChange);
    WidthChange(mq);
}
// media query change
function WidthChange(mq) {
    if (mq.matches) {
        svg.setAttribute("viewBox", "...");
    }
    else {
        svg.setAttribute("viewBox", "...");
    }
};
```

Read more about this technique: <http://www.smashingmagazine.com/2015/03/different-ways-to-use-svg-sprites-in-animation/>

# Animating SVG with JavaScript

# Popular SVG JavaScript animation libraries:

1. GreenSock Animation Platform (GSAP)
2. Snap.svg (“*The jQuery of SVG*”)
3. Velocity.js
4. D3.js

## GSAP's strengths:

- The ability to stack tweens
- Creating precise timelines
- Control tween delays
- Specify moments in time to start animations
- Start animations relative to each other using relative labels
- Nest timelines
- Create time lapses and Slow-Mo scenes
- Morphing shapes without point number restrictions: any shape can be morphed into any shape
- Line drawing, text animation, and much more

# GSAP animation capabilities include:

- Motion Along A Path
- Advanced Text Animation
- Draggable
- Shape Morphing
- Line Drawing
- Relative Color Tweening (See [this pen](#))

# Example: Line Drawing Effect

# Line Drawing with SVG:

The line drawing effect is essentially an animated *stroke offset*.

The stroke is a perfect even line, with perfect line edges and the equal amount of distance between the edges along the entire path/stroke length.

## Article

Jake Archibald wrote...

## Animated line drawing in SVG

Posted 29 July 2013 using tired fingers



I like using diagrams as a way of showing information flow or browser behaviour, but large diagrams can be daunting at first glance. When I gave talks about [the Application Cache](#) and [rendering performance](#) I started with a blank screen and made the diagrams appear to [draw themselves bit by bit](#) as I described the process. Here's how it's done:

# Thank You

# Credits:

1. CSS SVG transform origin bugs / behavior GIFS from GreenSock
2. Snow globe illustration designed by Freepik
3. Grumpicon screenshot from <http://grumpicon.com>