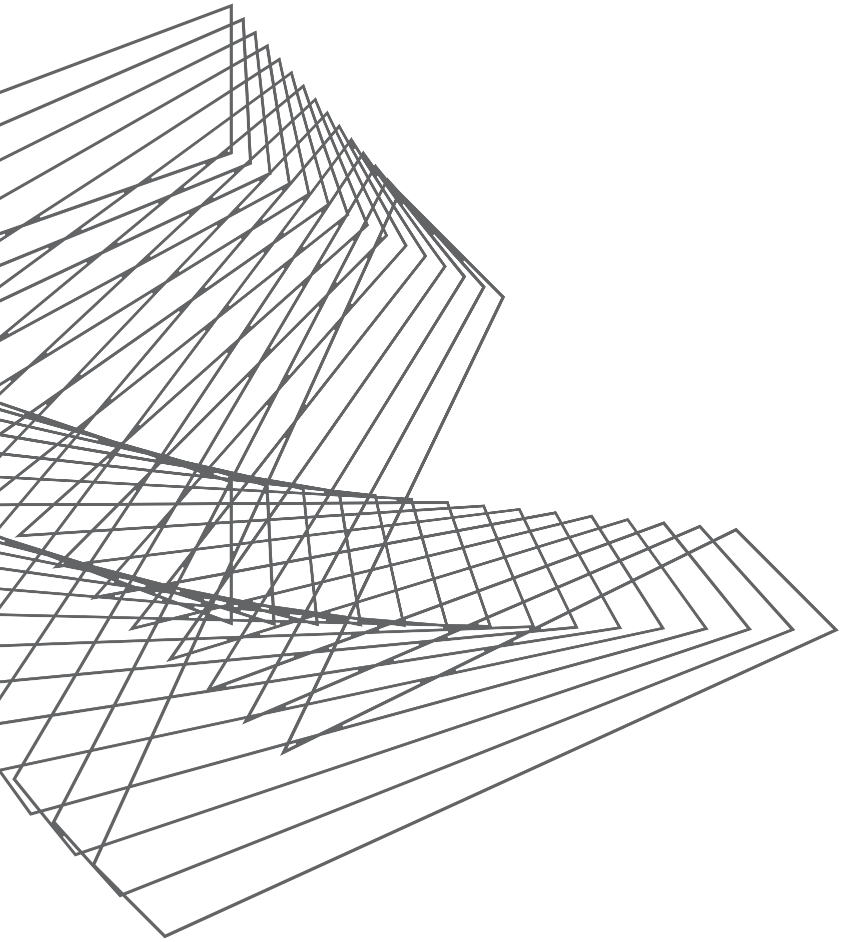
Emailzine

issue #01



design—dev

Designed by *Anna Yeaman*
Contact: anna@stylecampaign.com
Demos: http://stylecampaign.com/SVG/SMIL/
Resources: https://bit.ly/2JhLwcO

After writing about email design on the *stylecampaign* blog for eleven years, I wanted to create something you could take offline. Attending *lazinefest.com* inspired me to try and put together an email zine. It's nice to make something by hand, and I like how easy-going the format is. As with any zine, I wrote about stuff I'm into or currently learning about. There's not a theme but I realized as I was writing it, that much of the content overlaps a bit. Variable fonts touch on accessibility, dark mode and film-strips; and are essentially the same as SVG shape morphing in that a bunch of outlines are being shifted around. SVG filmstrips and flipbooks are just a bit of fun, some R&D I'd thought to share. Smart Invert brings us back to accessibility, and a loose tie-in with SVG in that we might see more of it in email as a result of dark mode. There's also a bunch of online demos and resources if you want to have a play (*see left*), though I tried to write it in a way that makes them optional.
I hope you enjoy it–*Anna*

Contents

Accessibility feature *Smart Invert* as a pseudo iOS dark mode

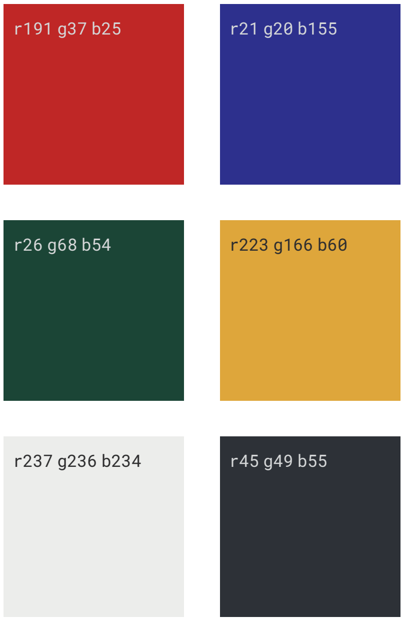
I came across the article, ‘Dark mode as iOS accessibility feature’ written back in 2014 on cgpgrey.com. Grey talks about how he has floaters in his vision, little dots that get in the way of his field of view. Black text on white aggravates it, whereas dark mode would make reading more comfortable. He goes on to mention a pseudo dark mode that people have been using for years called Invert Colors.

Invert Colors is an accessibility feature that reverses all the colors on the screen, including HTML email creative. On iOS 12 it’s found under Settings/General/Accessibility/Display accommodations/Invert Colors. It can aid people with poor vision, sensitivity to light or color blindness. In combination with other settings like Color Filters, it can address a number of visual impairments.

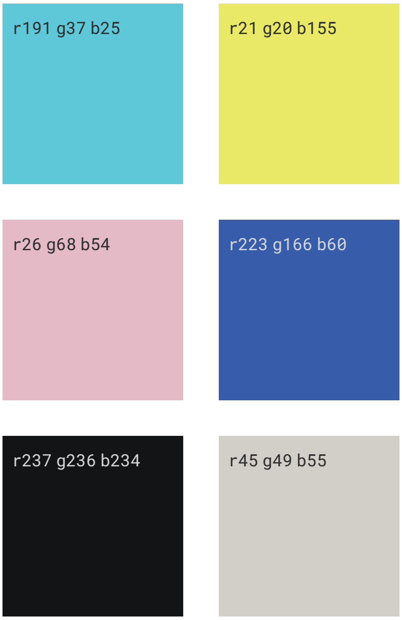
When Invert Colors is activated black text on white becomes white on black. Chris Coyier jokingly referred to it as “halloween mode” years ago on Twitter, as blue on white interface elements become orange on black. It’s a bit heavy-handed for some people as everything is inverted including images. Even so it’s been in use for over five years as a general purpose dark mode. Part of the appeal is that you can map accessibility shortcuts to the home or side button, making it easy to switch modes for more situational use. Apple were aware that Invert Colors was being used as a dark mode stand in, so with the release of iOS 11 they fine tuned it by adding a Smart Invert option. The original Invert Colors was renamed Classic Invert. The difference between the two is that Smart Invert doesn’t alter images, media and some apps that already use dark color styles. This made it a lot more practical for casual users. I imagine a number of people have been viewing emails under these settings, as it was the closest iOS had to a real dark mode. It’s more likely when you consider mobile opens tend to spike late at night and early morning, while people read email in bed.

With the release of iOS 13 and an official dark mode Smart Invert will become obsolete for many, but it might still function as a back up. During WWDC 2019, Apple posted the video ‘Supporting dark mode in your web content’ which also has a section on email messages. In the video they show how plain text emails will automatically be adjusted but HTML emails will not auto-darken (just like macOS Mojave). They explain that developers are in control of how their content appears, and as such have to adopt *color-scheme* and *prefers-color-scheme* in order to customize their emails for dark mode. So Smart Invert might still be in limited play, as it’s a known and reliable way to view HTML emails in ‘dark mode’. At least until we get up to speed and more emails contain dark styles.

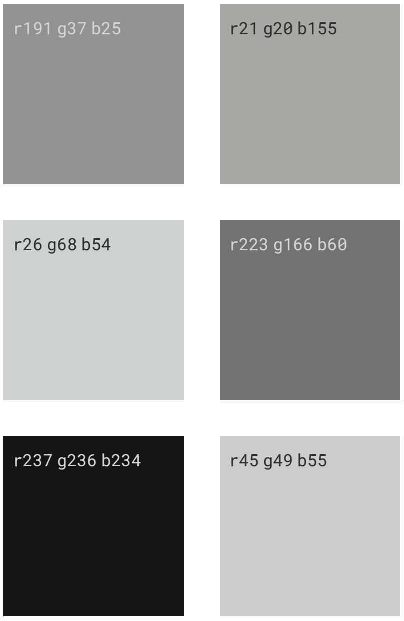
This page shows the default colors, and (*p9*) under Smart Invert. Shifts such as red–blue, can lead to mixed up color cues.



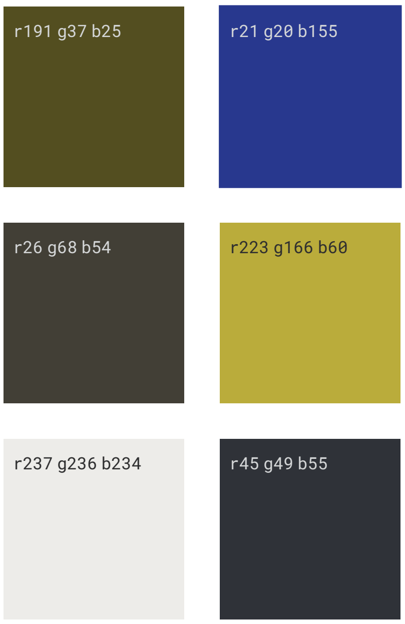
Colors under Smart Invert; some of the text in our test—*stylecampaign.com/SVG/invert/*—became unreadable as the tone had also shifted.



People combine Smart Invert & the grayscale filter as it’s *“soothing”* and a *“battery saver mode”*. Monochromatic vision is rare — 1 in 33,000 — but more could be viewing email in grayscale than we realized.



How the colors from (*p8*) look under PS’ protanopia red/green color blindness simulator: *View / Proof setup / Color blindness / Protanopia*



“Try the smart invert feature–imagine dark mode email.
really neat.” *@rstevens October* 2017

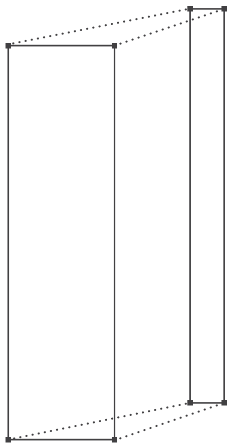
I looked at Mail on iOS 12 using Smart Invert, to see how HTML emails appear to someone who isn't colorblind. The majority of emails are light themed to begin with so they convert well. Though you'll still want to run your creative through something like contrast-ratio.com. HTML emails that use live text and background colors etc. provide a better experience than image-based emails, as images are left untouched and are often bright white. Emojis are still inverted under Smart Invert, and the unfamiliar colors can make them harder to decipher in subject lines.

When inverted, black image-based headings and logos are overlaid on a black background. If you're using transparent PNGs they'll disappear. In my 2018 small text study, 27.7% used image-based headings alongside live text. This is from 50 emails so it's not statistically sound, but it's still a fairly common practice. Looking ahead to 'prefers-color-scheme', we might see more use of SVG in email design for icons and illustrations. As you can adjust the colors of an inline SVG with CSS, so you only need the one set of assets. Another common occurrence using Smart Invert is if you've cropped an image assuming a white background it can look messy against black. Rémi Parmentier mentioned similar issues in 'Dealing with Outlook.com's dark mode', the same group of problems show up here.

It can get a bit confusing using iOS invert modes if you're not colorblind. As Grey mentioned, "Invert colors makes the less important parts of the screen pop and subdues the parts that are supposed to draw your attention". Mixed up color cues, and unfamiliar brand colors can be disorientating. Though it's a useful exercise for designers, as it can uncover any confusing aspects of a design.

Because not everyone views color in the way the designer intended, WCAG 2.1 states that color shouldn't be the only visual means of conveying information. A new iOS 13 feature called 'Differentiate without color' addresses this. It allows designers to replace interface items that rely only on color with an alternative. For cases where it's difficult to have those colorblind adjustments shown by default. Color vision deficiencies affect 8% of men worldwide, and 0.5% of women according to colourblindawareness.org, with red-green blindness being the most common. You can simulate what this looks like in PS, under View / Proof setup / Color blindness and pick protanopia or deuteranopia. I went into this to see what email looks like under 'dark mode', but it's left me wanting to learn more about color accessibility. Besides all the color and contrast tools which you're probably familiar with, Geri Coady's book, Color Accessibility Workflows looks like a good introduction.

Variable fonts generate multiple styles from a single font file, this is possible as character outlines can change their form



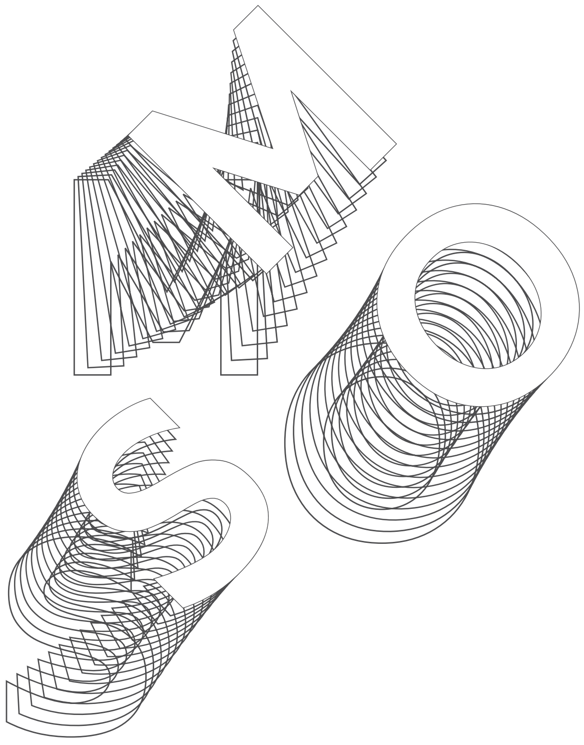
Glyph outlines like the letter 'I' can shapeshift into different weights

Currently when you purchase a webfont you choose from a range of individual styles such as condensed or bold, and each of these is a separate file. The advantage of OpenType variable fonts is that they can generate multiple styles from a single font file. This is possible as character outlines can morph between extremes, such as a thin to black weight. If you squint and imagine the image to your left is a letter 'I', its outline can shapeshift across a range of weights; all drawn from the same glyph and with the same number of points.

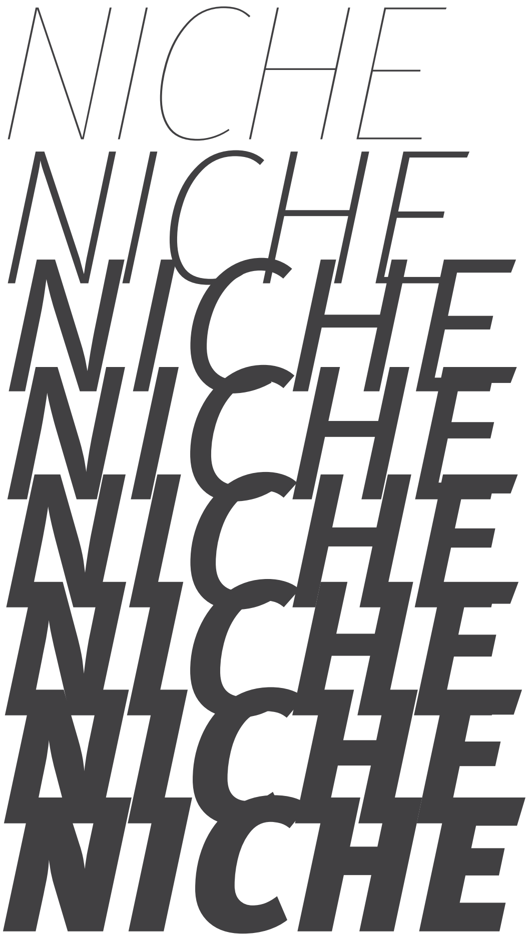
The full range of weight options falls along a design-variation axis, from which the type designer is able to select different locations and assign names e.g. thin, regular or bold. These pre-defined options are called *named instances*. If you require a weight that falls inbetween a named instance you can define it yourself, allowing for more control and choice. As type designer DJR explains,"Variable fonts break down that wall between type designer and type user, because now you have access to stuff that previously only I had access to."

Weight is just one of five registered design axes introduced with the OpenType 1.8 font specification. There's also width, slant, optical size and italics. Type designers can also add their own custom axes. For instance WHOA by Scribble Tone has an appropriately named '*Radness*' axis, and Jabin by Frida Medrano has a '*Swash*' axis. Design axis can be combined to create a multi-axis font, offering a vast array of potential combinations all rendered from the same font file. This increase in choice comes at a smaller file-size—assuming you need more than a couple of styles to begin with—and fewer requests.

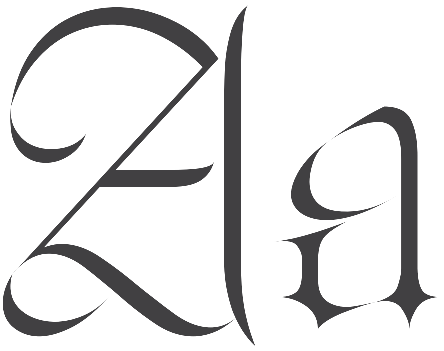
OpenType font variations were introduced at the ATypI conference in 2016 in a collaboration between Adobe, Google, Apple and Microsoft. In a recording of the OpenType session Simon Daniels shared,"The fact that we have four major companies here, should be a statement that we are expecting this to be supported in all major platforms, OS platforms and major content creation platforms."



Variable font *WHOA* designed by Scribble Tone, originally had the one Radness axis slider. Version 0.3 now has four axes giving you separate control of the horizontal offset, vertical offset, rotation and zoom.



Variable font *FF Meta* designed by Erik Spiekermann, has a weight axis for both the roman and italic (*above*) versions. Jason Pamental built out a demo site at *codepen.io/jpamental/pen/MGEPEL*



Frida Medrano's variable font Jabin (*above & right*) has two axes: weight and swash. Weight goes from light to bold, whereas swash is a custom axis that controls the '*swashiness*' of the uppercase letters. Left shows a light weight with no swash, while right is a bold weight with a heavy swash. Everything between those extremes is available: *v-fonts*.com/fonts/*jabin*

Google Fonts have a few variable fonts on their early access page, and appear to be considering how best to implement them. This Twitter poll is from @googlefonts, January 2019:

If you know what Variable Fonts are, what benefits are the most important to you?
18% Custom expressive fonts
36% Finer text typography
27% Same fonts, faster
19% Animated text effects

Performance gains received the second highest number of votes. Variable fonts are smaller in size compared to a font family of individual styles, as they interpolate or morph between outlines. The type designer draws the letter 'G' for example at its lightest and heaviest—the two extremes on a weight variation axis—and the in-betweens are dynamically generated. So only the data from those two or three key outlines make up the file size, even though we have access to a full-range of weights. This is just like SVG shape morphing on p52. You only need the SVG data that describes say a circle and a target shape like a square, and it tweens between the two.

Although variable fonts are more performant, they can still contain bloat. In the write up 'How to use variable fonts in the real world' Clearleft created a subset of the variable font Source Sans (essentially ditching parts of the font they didn't need), and converted from TTF to WOFF2 which took them from 491Kb down to 29Kb. In an effort to make these types of optimizations readily available, Jason Pamental a member of the W3C Web Fonts Working Group, shared that they're exploring something along the lines of font streaming, or 'progressive font enrichment'. This involves only serving up parts of the font as and when needed; across multiple page views or sites. Google Fonts have built out a proof of concept at: *fonts.gstatic.com/experimental/incxfer_demo*.

‘Finer text typography’ or responsive typography was listed as the top benefit in the poll. More control over the typesetting for improved readability and accessibility is a strong feature. In Isabel Lea’s variable font experiment, the word ‘Loud!’ gets bolder as she claps. Bram Stein showed how a variable font can aid justification. While Jason Pamental demonstrated how increasing the grade axis — *Grade* changes the weight of a typeface without altering glyph widths, so the text doesn’t reflow — can improve the readability of reversed type in dark mode. Other demos show variable fonts responding to device width, human gestures, ambient light and distance.

Keeping with the more experimental side of variable fonts, there are many interesting new typeface designs such as those found on futurefonts.com, and through David Jonathan Ross’ fontofthemonth.club. Kinetic type, colored variable fonts, Laurence Penney’s horse animation that’s a variable font, Toshi Omagari’s variable font converted from an animated Gif. People are still exploring what this new format can do.

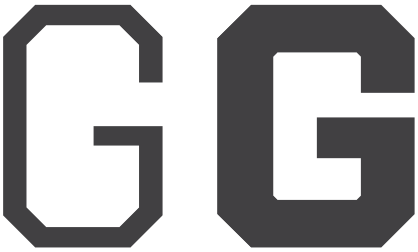
There is a slight risk that the ability to adjust everything might add complexity to a type system; and maintaining that along with a bunch of unfamiliar custom axes could be challenging for some clients. Type is often the first thing to breakdown in a modular email system, so it could require some extra guidance. Adidas built out a custom Illustrator panel to aid designers working with their variable font Adineue PRO. It even has a fit to width button that Jeremy Mickel, the font designer said, “Turns just some text into something that feels like a sign”. The tooling around variable fonts is still being figured out, and existing typefaces reworked. So there’s time to get familiar before we start using them in production.

If you want to try out some variable fonts yourself, there are a number available on v-fonts.com and axis-praxis.org. Illustrator and Photoshop CC already support variable fonts, with InDesign support coming soon. In PS look for a VAR to the right in your font menu, then check the properties panel.

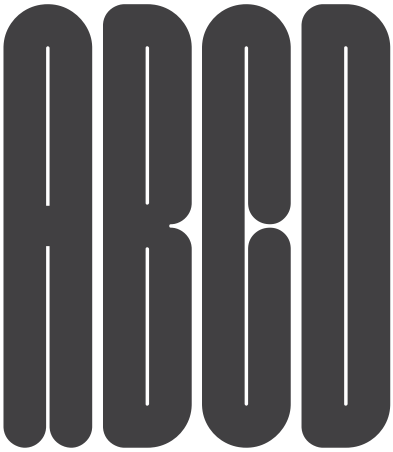
G G

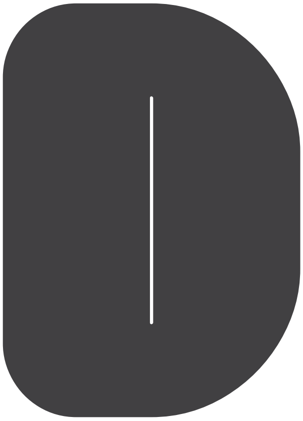
Variable font *Graduate* by Eduardo Tunni (*above & right*) has 12 axes. Left shows the Grade axis set to a range of 0–20, useful for reversed type when switching to dark mode.

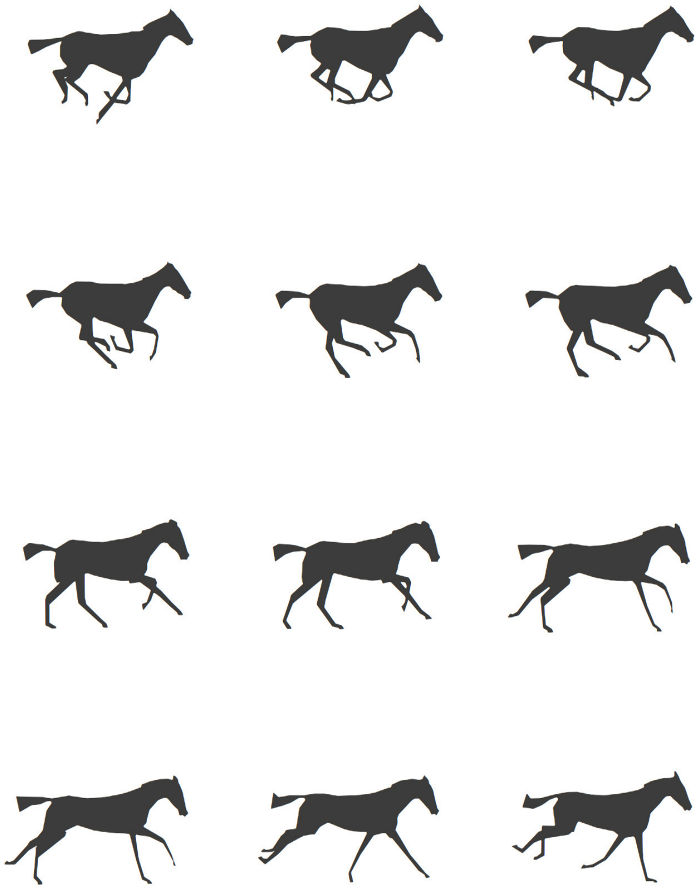
Right features the X Opaque axis set to a range of 40–140, this varies the horizontal (x-direction) thickness of the strokes. You can play with all 12 axes at: *v-fonts.com/fonts/graduate*



Variable font UT Morph, by Wete and Oscar Cobo. Inspired by Wim Crouwel, with a Positive and a Negative axis. Letter ‘D’ (*left*) Iso and (*right*) Hyper weight.

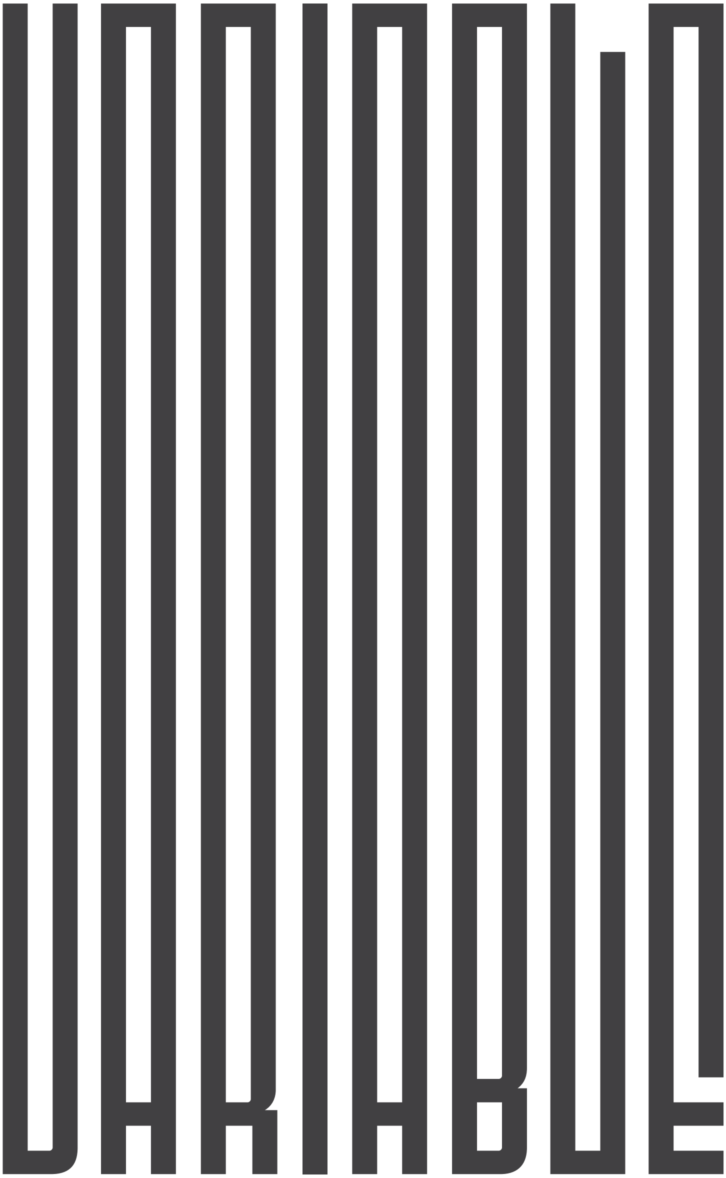






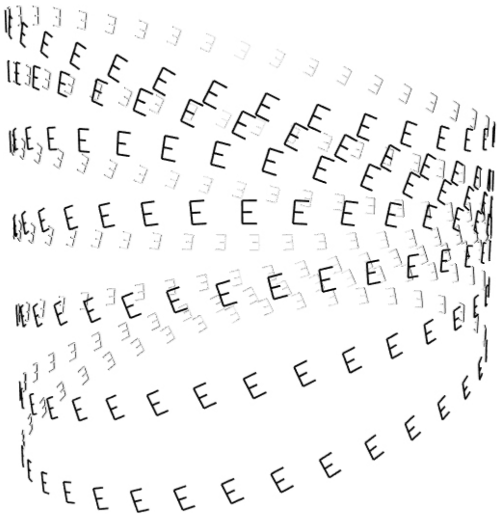
Instead of an axis that adjusts e.g. the weight of a variable font, *Laurence Penney* is using the axis as a filmstrip to create animations: *axis-praxis.org/playground/horse*

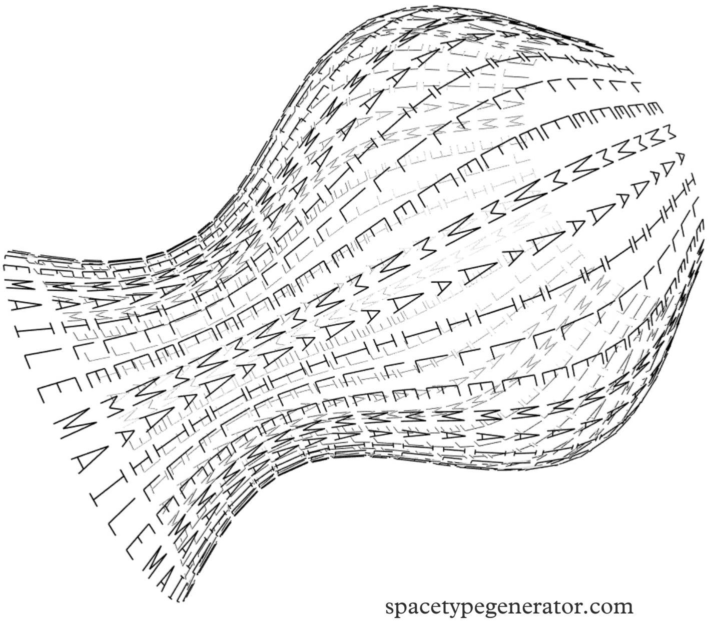
```
/* Code by Laurence Penney, you end up with a
   horse animation that runs in Mail */

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<style>
@font-face {
    font-family: Muybridge;
    src: url(MuybridgeGX.woff2);
}
@keyframes Gallop {
    from {
        font-variation-settings: "TIME" 0;
    }
    to {
        font-variation-settings: "TIME" 1000;
    }
}
body {
     font: 320px Muybridge;
     animation: Gallop 0.5s linear 0s infinite;
}
</style>
</head>
<body>🐎</body>
</html>
```


VARIABLE

Variable font *Fit* by David Jonathan Ross
was designed to fill up any space

In the *Google Fonts* Twitter poll, 19% cited animated text effects as the main benefit of variable fonts. Kinetic typography has been increasingly popular over the last few years, and the ability to animate variable fonts adds more fuel to that trend. It might be worth exploring in conjunction with *prefers-reduced-motion* as some animations, though mesmerizing, can be a bit hectic. The image on the left is by Mitch Paone of @DIA_Studio, who specialize in kinetic identity systems. You can hear him talk about their work at: youtube.com/watch?v=qTynk4wmXD8





spacetypegenerator.com
by *Kiel Mutschelknaus*

Email support

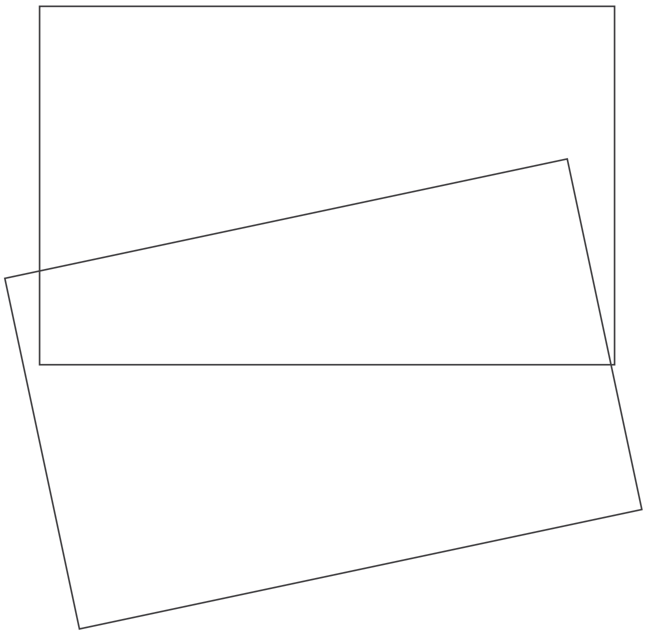
As of July 2019, Caniuse reports 85% global support for variable fonts. In order to test email client support we choose Amstelvar by David Berlow. It's listed on the Google Fonts early access page and can be downloaded from GitHub. Our test file—*stylecampaign.com/SVG/fonts/variable/cm/*—includes five paragraphs of text. The top two use semantic markup like H1, so the client determines the styles of each element. The next two place the styles inline which is how most email designers still work, and the bottom example is inline with an @supports switch. We can determine from those cases what's supported, as if the two paragraphs differ it should only be a result of using font-variation-settings.

Implementing variable fonts isn't that different from how we currently handle webfonts. The CSS property *font-variation-settings* is the only real change, as it allows you to assign properties beyond what we're familiar with. Everything is defined under the umbrella of *font-variation-settings*, with the parameters/axes in a comma delineated list. Each axis has a four letter tag associated with it, and you only need to list the ones in use. The five registered axes such as *wdth* and *wght* are in lowercase, whereas custom axis are in caps. The value beside the axis name maps to a position on that variation axis. You can get those range values from Wakamaifondue.com or axis-praxis.org, both break down the available features in a variable font such as the number of axes etc.

Email support for variable fonts is ~42% and includes iOS Mail and Apple Mail (at least based on this TTF demo). You might come across the odd client that supports webfonts but not variable fonts. It would show the custom font but not in the VF styles. The fallback that we used was Helvetica, and it came in fine everywhere. So I think variable fonts are viable, it just needs a proper run through with a real-world project, as there's bound to be caveats somewhere.

```
/* We built out the demo working from sample code on
developers.google.com/web/fundamentals/design-and-ux/typog-
raphy/variable-fonts */

/* Link to the variable font in the head as usual */

@font-face {
font-family: ‘AmstelvarAlpha’;
src: url(‘http://www.stylecampaign.com/SVG/fonts/Amstel-
varAlpha-VF.ttf’);
font-weight: normal;
font-style: normal;
}

/* If font-variation-settings are supported it serves up the
variable font, if not the inline styles are rendered */

@supports (font-variation-settings: ‘wdth’ 200) {
*[class=font-amstelvar-sup-a]{font-family: ‘AmstelvarAlpha’
!important;
font-variation-settings: ‘wdth’ 320, ‘wght’ 90, ‘opsz’ 19,
‘GRAD’ 88;
}
}
```

Animated images as SVG/SMIL *filmstrips and flipbooks*



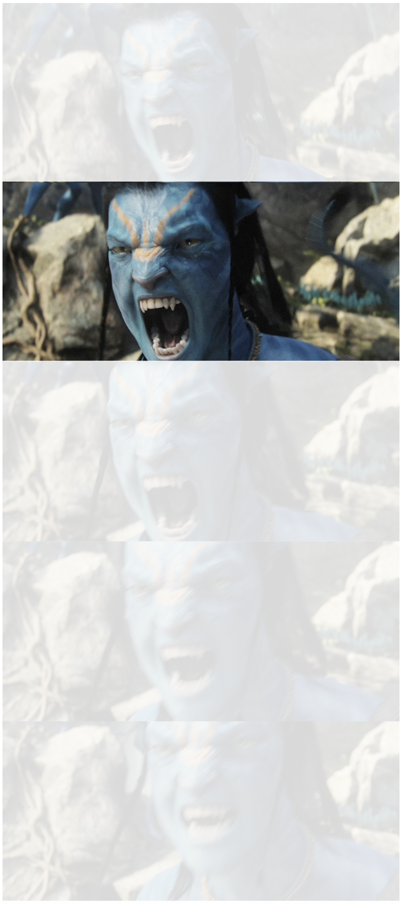
Support for real video in email has been non-existent or spotty over the years, so there's a long tradition of fake video workarounds. Starting with video Gifs in 2008, JPEG push, cinemagraphs, and more recently Kristian Robinson's faux-video, and Alice Li's rollover Gifs. It's as if all email R&D eventually leads to a video hack, and SVG is no different.

We inadvertently came across this SVG/SMIL option while building our SVG Xmas slots game, as the mechanism is pretty much the same. Slots consisted of three animated filmstrips, with a mask showing the visible area. A PHP script then generated random offsets which moved each of the strips vertically by a set amount, giving you a unique spin every time. Looking at those long strips of slots it was easy to imagine replacing them with strips of video. We knew from our SVG slider experiment that you could incorporate raster images into an SVG layout, so we more or less just switched the vector assets out and adjusted the animation.

Just like slots the interaction and animation is driven by SMIL, and we're using inline SVG with an external image reference. The filmstrip is a jpeg and embedded into the SVG. While working on jpeg push, we converted some video clips into individual frames. As they were handy we used those same images to create one vertical filmstrip in PS; you could automate this step if you were to use it often enough. While playing with some long clips, we hit a 30,000px height limit on jpeg files so we added the option of multiple strips just to be safe.

There's one SMIL keyframe for each frame of video, so hand-coding is slow going and prone to error. If you want to make adjustments to say the frame rate, you need to update the whole thing manually. After doing a couple by hand Graeme gave up and wrote a PHP script that generates the SVG code for the keyframes. Using a form you plug in the resolution, number of strips, frame rate, whether to read filmstrips from the top or bottom—can also run your video backwards—and it spits out the SVG code.

The SMIL keyframes make up the bulk of the code, so automating it saves you a lot of work. A 3.5 sec. animation contains ~53 frames at 15 fps. So the number of keyframes can get big, really quick. File-size-wise you'll want to juggle resolution, length of clip, image compression, frame rate, etc. jpegs are more efficient than Gifs and better quality, but you can use whatever image format you like, or stick with vector artwork.



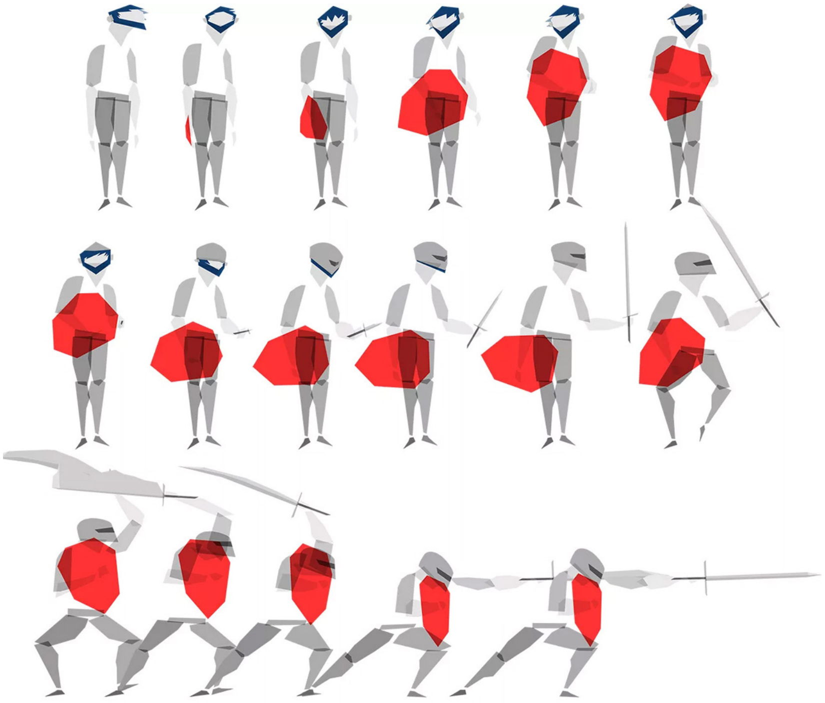
```
/* Example code for a 3.5 sec SVG/SMIL filmstrip animation */
<svg xmlns="http://www.w3.org/2000/svg" xmlns:link="http://
www.w3.org/1999/xlink"/* define SVG parameters */>

<defs> /* viewport is size of a frame with mask as backup */
<clipPath id="clipPath">
<rect id="clip" x="100" y="0" width="1280" height="544" />
</clipPath>
</defs>

<defs> /* defining an image object to use later */
<g id="filmstrip">
<image x="0" y="0" width="1280" height="29376"
xlink:href="filmstrip.jpg" />
</g>
</defs>

<g id="main-viewport"> /* group containing keyframe data */
<g style="clip-path: url(#clipPath);">
<use xlink:href="#filmstrip" x="100" y="-28832">
/* first keyframe set y position at this time */
<set attributeName="y" attributeType="XML" to="-28288"
begin="sr01.click+0.0666666666666667s" fill="freeze"/>

/* next two keyframes update y position at this time */
<set attributeName="y" attributeType="XML" to="-27744"
begin="sr01.click+0.133333333333333s" fill="freeze"/>
<set attributeName="y" attributeType="XML" to="-27200"
begin="sr01.click+0.2s" fill="freeze"/>

/* ... additional keyframe definitions go here
...  */
```

```
/* final keyframe */
<set attributeName=”y” attributeType=”XML” to=”0”
begin=”sr01.click+3.5333333333333s” fill=”freeze”/>
</use>
</g>
</g>

/* play button (circle and triangle) */
<g id=”control”>
<polygon points=”730,246 760,266 730,286” fill=”#d0d0d0”
stroke=”none” style=”fill-opacity:0.75;”>
<set attributeName=”points” attributeType=”XML”
to=”730,2046 760,2066 730,2086” begin=”sr01.click”
fill=”freeze”/>
<set attributeName=”points” attributeType=”XML” to=”730,246
760,266 730,286” begin=”sr01.click+4.1s” fill=”freeze”/>
</polygon> /* end of polygon that draws a triangle */

/* click circle with id ‘sr01’ that triggers an event which
is then used to animate the image (click + a time = trigger
for each keyframe) */
<circle id=”sr01” cx=”740” cy=”266” r=”40” stroke=”#d0d0d0”
stroke-width=”6” style=”fill:#d0d0d0;fill-opaci-
ty:0.0;stroke-opacity:0.75;cursor:pointer”>
<set attributeName=”cy” attributeType=”XML” to=”2066”
begin=”click” fill=”freeze”/>
<set attributeName=”cy” attributeType=”XML” to=”266”
begin=”click+4.1s” fill=”freeze”/>
</circle>
</g>

</svg>
```

3 of 14 SVG models from our flipbook

We originally went with a filmstrip as that's how slots was configured, but you could use individual frames instead like a flipbook. Jonathan Ingram shared a SVG/SMIL flipbook back in 2012 (*bifter.co.uk/issue/17*). In 2014 we shared a few SVG/SMIL flipbooks as part of a 3D to SVG tool write up. The flipbooks were created by exporting a series of SVG vector assets that made up a run cycle e.g. 14 frames in different positions. We then turned each SVG model on and off, setting the timing for each keyframe. Our models were a bit too big, but with more performant artwork you'd have no problem.

I think it was around five years ago that I saw *Playground Inc's* vector animations and wondered if we could do something similar in email. Graeme used to work with Flash artists years ago as a games programmer, and had seen some really slick vector animations. Illustration is always a trend in email design and SVGs have many benefits over raster images. But it's a lot of effort — plus fallbacks — so I get why we don't see more SVG animations like this. Interestingly *Playground* were using a "timer" powered by Raphaël so that each SVG frame gets drawn at a certain time. A JavaScript solution, so not something we could use in email, but a similar premise to the SMIL flipbook.



Vector animations by *Playground Inc*
from a sequence of 72 frames

```
/* Example code for SMIL flipbook using SVG artwork */
<svg version="1.1" id="svglayer" xmlns="http://www
w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
viewBox="0 0 800 600" x="0px" y="0px" width="800px"
height="600px" xml:space="preserve"/* define SVG parame-
ters */>

/* define SVG group for frame one, model pose one */
<g stroke-width="1" stroke-miterlimit="1">
<animate
id="frame1"
attributeName="display"
/* 14 values corresponding to the 14 SVG polygon models in
the run cycle. inline = visible and none = not visible */
values="inline;none;none;none;none;none;none;none;none;none
;none;none;none;none"

/* 14 keyframes for the 14 SVG models, values between 0-1 as
proportion of animation duration */
keyTimes="0;0.076;0.152;0.228;0.304;0.38;0.456;0.532;0.608
;0. 684;0.76;0.836;0.912;1"
dur="1s"
begin="0s"
repeatCount="indefinite" />

/* Polygon model data for first of the 14 poses */
<polygon fill="#39210b" stroke="#39210b"
points="454.3,386.5 440.7,387.2 434.8,388.9" />
/* ... additional polygon data goes here
...  */
<polygon fill="#474318" stroke="#474318"
points="407.0,224.9 408.9,212.8 394.7,224.0" />
</g> /* end of polygon data for first model pose */
```

```
/* repeat for the second model pose (frame 2) */
<g fill="#000000" stroke="#000000" stroke-width="1"
stroke-miterlimit="1" opacity="0.4" filter="url(#s-blur)">
<animate
id="frame2s"
attributeName="display"

/* note how 'inline' is now second in the list as model two
is now visible and model one is 'none' or not visible */
values="none;inline;none;none;none;none;none;none;none;none
;none;none;none;none"

/* Same 14 keyframe timings as easy to keep track of */
keyTimes="0;0.076;0.152;0.228;0.304;0.38;0.456;0.532;0.608
;0.684;0.76;0.836;0.912;1"
dur="1s"
begin="0s"
repeatCount="indefinite" />

/* Polygon model data for second of the 14 poses */
<polygon fill="#1e1306" stroke="#1e1306"
points="461.1,434.5 438.1,416.6 445.3,436.7" />
/* ... additional polygon data goes here
...  */
<polygon fill="#474218" stroke="#474218"
points="394.3,230.0 394.2,217.3 379.8,227.7" />
</g> /* end of polygon data for second frame pose */

/* repeat for the remaining 12 frames or poses, adjusting
'none' and 'inline' values to turn them on and off */

<polygon fill="#31261f" stroke="#31261f"
points="388.9,200.3 388.3,198.3 378.4,191.5" />
</g></svg> /* end of last model (frame 14) close SVG */
```

'Video' flipbook

If you take this basic flipbook technique you can rework it using raster images. We took the code from one of the flipbook examples and swapped out the SVG polygon models for jpeg images, keeping the rest of the code the same.

```
/* replace SVG polygon model data with an image url */
<g>
<animate
id="frame1"
attributeName="display"
values="inline;none;none;none;none;none;none;none;none;none
;none;none;none;none;none;none;none;none;none;none"
/* values between 0-1 as proportion of animation duration */
keyTimes="0;0.052;0.105;0.157;0.21;0.263;0.315;0.368;0.42
1;0.473;0.526;0.578;0.631;0.684;0.736;0.789;0.842;0.894;0.
94;1"
dur="1.333s"
begin="0s"
repeatCount="indefinite" />
<image x="0" y="0" width="480" height="204" xlink:href="av/
av1197.jpg"/>
</g>
```

You end up with a 1.3 sec video sequence, containing 20 frames at ~15 fps. Single frames are less complicated as there's no need to calculate filmstrip offsets, just the keyframe timing. It really only needs three per frame for off/on/off, it's just easier to keep track of this way. For longer sequences you'd want to optimize it, and possibly write a script to generate the keyTimes. As it's a sequence of individual frames, you don't need to piece together a filmstrip, though that's something else you could automate. I'd also add a slight delay at the start to give it time to buffer a few frames.

Eight jpeg frames—from a series of 20—each is turned on & off at different *keyTimes* that fall between 0–1 to create a flipbook.

Frame #1 keyTime 0

Frame #2 keyTime 0.052

Frame #3 keyTime 0.105

Frame #4 keyTime 0.157

Frame #5 keyTime 0.21

Frame #6 keyTime 0.263

Frame #7 keyTime 0.315

Frame #8 keyTime 0.368

The downside of the flipbook technique is you'd end up with more code and HTTP requests. As instead of bringing in one big image, you'd need to request multiple and they'd all be listed in the code. If using raster images it would be inline—which would count towards Gmails ~100K limit—but you can externally reference the frames if using SVGs. It's not so bad for a 14 frame sequence, but with a 10sec video clip the frame count would start to climb.

Adding controls to a flipbook also comes with different considerations. With a filmstrip a click triggers an event, and the event drives the animation. Whereas a flipbook contains a series of keyTimes which drives the animation. You could use 'begin' to control a flipbook animation via a click event, but it's not something we've looked into thoroughly. For now our flipbooks loop indefinitely. Another consideration, is with SMIL it's one button per event so it only plays once. If you want to press play multiple times without refreshing the page, you need to layer up the buttons. For example SVG slots has five identical buttons, one for each spin. Most of our demos have the one button, but if you were to use this in production I'd add a couple more just to be safe.

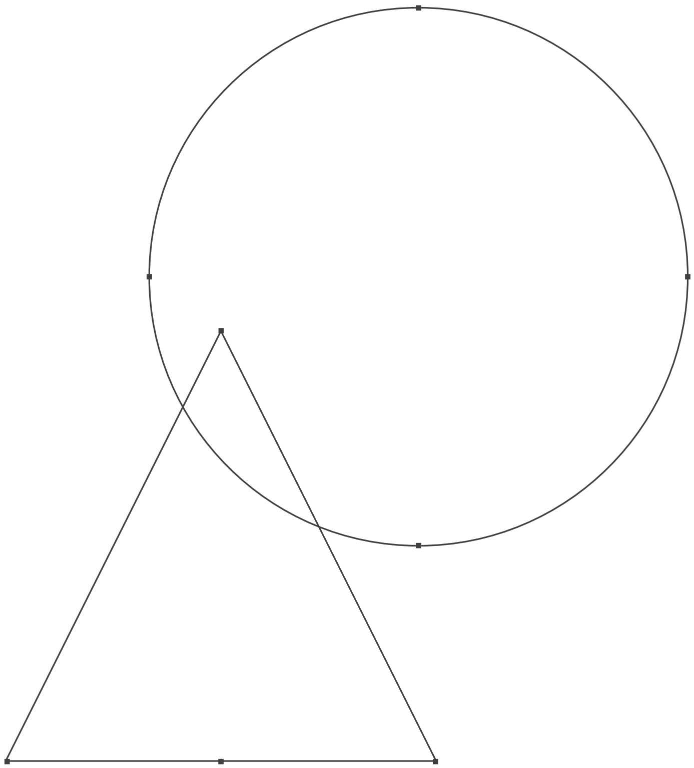
We haven't dug into all the nuances of animating sequences of images with SMIL, this was just a two-for-one back when we were building slots. There's bound to be other ways to go about this, as flipbooks and filmstrips are both old techniques. Depending on what you're trying to do, it might be worth comparing the two if you want to explore further.

Creative layouts

One interesting aspect to all this, is the creative possibilities that being part of an SVG brings. You can incorporate your 'video' into elaborate layouts. As SVG has many useful features like clip paths, filters, overlapping live text, responsive design, interaction and spline animation.

It’s strange to consider SVG has been around since 2000-ish, we could have been using this in place of those tiny Gifs a decade ago. Though I’m not sure what SVG support would’ve looked like back then. Today inline SVG has ~43% support, based on the top ten email clients from emailclient-marketshare.com for June 2019. This includes iOS, Mac Mail, Outlook for Mac, Native Android and Samsung Mail. Though everyone’s support numbers will differ. IE doesn’t support SMIL animation, so only the first frame would show. You could serve up an animated video Gif fallback in IE and elsewhere, if you really wanted to get into it. If you’re working with SVG and looking for a native “animated images” solution for either vector or raster artwork, you may find this useful.

I wanted to take another look at SVG *shape morphing*



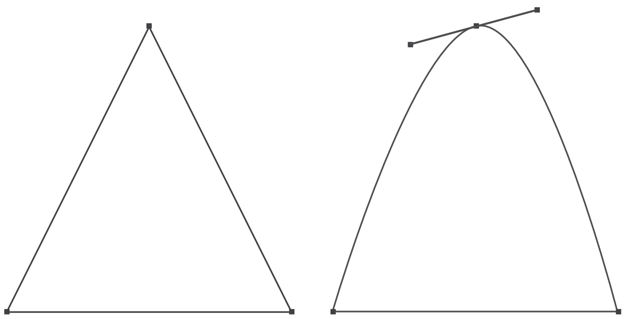
I wanted to revisit SVG shape morphing to see where we stand with support. Shape morphing is typically more lightweight than filmstrips or flipbooks. As you have a chunk of code that draws say a circle, and another chunk of code that repositions the circle's points into a triangle. It then interpolates between the two shapes, which means it generates the inbetween states for you. Whereas a flipbook or filmstrip needs all the extra data for the inbetween frames.

Five years ago we were trying to create character animations by morphing between a series of polygon meshes. We chose to work with polygons as Graeme had written a tool which converted 3D polygons into SVG polygons. Polygons have straight edges, in our case made up of 3-sided triangles or 4-sided-quads. They're constructed out of interconnecting points — like join the dots — each defined by their x/y coordinates within the SVG viewport. The more point positions you need to store, the more code it generates increasing the file-size. With enough polygons you could mimic curves, but the mesh would likely be dense. So even though I like that low-poly aesthetic, it's not going to be suitable for all projects.

If you want to morph between organic, curved edged vector shapes it's best to use splines instead of polygons. Spline data is slightly different in that you store the point coordinates, along with their control points. If you've used the pen tool in PS, it's those handles you can drag to change the flex of the spline between points. Besides drawing organic shapes, you can also place text along a spline, animate an object along it, or use it as a mask or clipPath. So for my testing this time around, I decided to shift from polygons to splines.

Shape morphing works by essentially shifting a bunch of points around, and telling it where you want to reposition them. Like bait balls, those same points reform into different shapes. You start out with a base shape. As each shape has to have the same number of points in the same order — at least when working with SMIL — you want it to be the most complicated. In our demo the star requires the most points to draw compared with a circle or square, so it's the base. You can then manipulate your base shape to form new target shapes in a vector editor, possibly by tracing over some drawn keyframes. You don't need a target shape for every frame, as shape morphing smoothly tweens between keyframes.

In order to get the point data you can export the different model positions or 'keyframes' from e.g. Illustrator as individual SVGs, view source and then copy and paste the control point data into your HTML. When you first define the base shape in the code it can be any one of your keyframe



Polygon with three points and straight edges (*left*), spline with additional control points which allows you to create curved edges (*right*) shapes, as they're all made up of the same number of points by then. You are just saying here's the chunk of data we'll be shifting around.

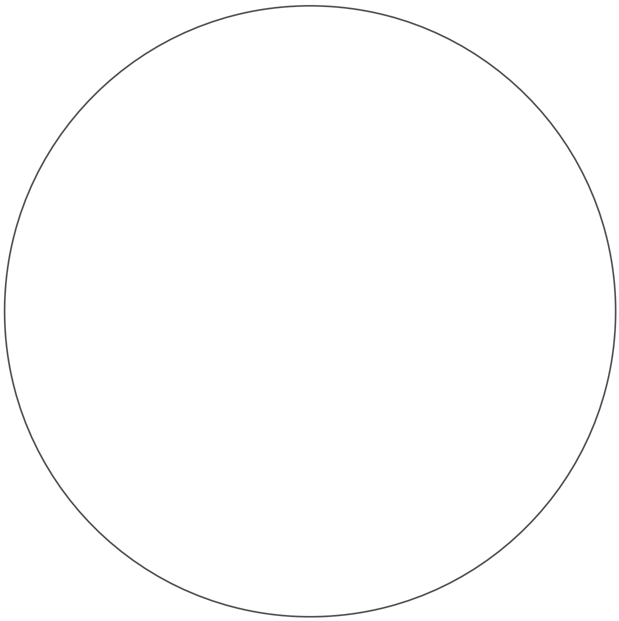
You then define the animation, by setting a duration and telling it how you want to position all that base point data on frame one. Again your first keyframe can be any one of your shapes, you're just saying on the first frame place that group of points like so. You then set a target shape or series of target shapes for it to morph between. If you go from a square to a circle like on p57, it will just pop from a square to a circle. You'll need to add a third keyframe — a new set of data points — for it to loop from a square to a circle, then back to a square again. I just wanted a pared-down example as it's easier to follow.

There are a number of ways to add an SVG to an HTML email, and each has varying levels of support. In order to test shape morphing — effectively a test of SMIL support — I stuck with inline SVG and SVG image. Image has the broadest support of all the embedding techniques at ~62%, based on emailclientmarketshare.com for May 2019. It includes iOS, Apple Mail, Outlook for Mac, Native Android, Samsung, Android Outlook, Gmail App IMAP, Windows 10 Mail, Yahoo Mail!, Outlook.com, Office 365, AOL Mail and Thunderbird. SVG images support SMIL animation, so shape morphing will also run in those clients. That's not bad coverage, and you could fallback to a raster image elsewhere. Shape morphing when placed inline has less support at ~43%. It includes iOS, Apple Mail, Outlook on Mac, Thunderbird and Samsung Mail. Browser support is solid except for IE which doesn't support SMIL. IE displays a static SVG — like Gifs in Outlook — which for many illustrations or backgrounds would be fine. Or you could use UA targeting to serve up different content. In 2015 SMIL was briefly deprecated in Chrome 45 and then reinstated, I guess it's in a holding pattern for now.

Support isn't the only consideration though, as different techniques have varying capabilities. SVG image supports SMIL animation but not interaction, whereas inline SVG supports both. You can manipulate the parts of an inline SVG using CSS, you can't with an SVG image. Images are external, so everything we've come to know about dynamic images in email also applies here. Inline SVGs aren't as suited to dynamic content as you're limited to what gets sent out in the HTML. It's these types of project requirements that usually determine how you'll embed the SVG.

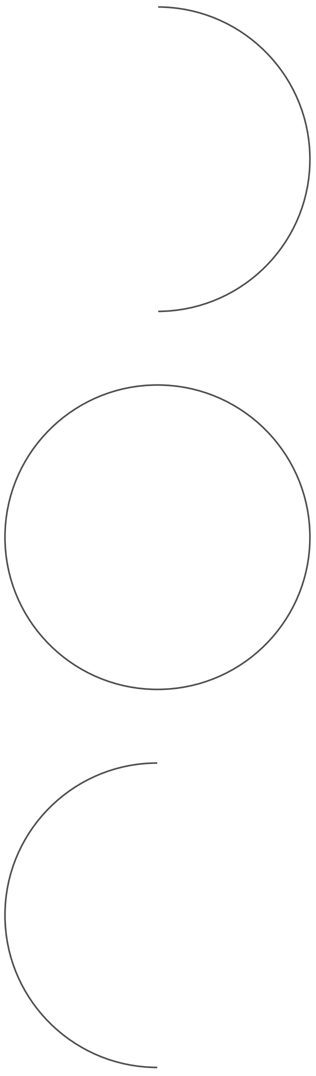
Basic SVG image support is inching up in email clients, and SMIL animation is built in as a native solution. This means shape morphing is more viable than five years ago, as long as SMIL browser support stays current.

Morphing square—circle



```
<svg xmlns="http://www.w3.org/2000/svg" /* define SVG parameters */>
/* base shape data (square but could be circle) */
<path fill="none" stroke="#114460" stroke-width="2" d="M398.01,80.04
4c1.282,0.015,2.128,1.142,2.143,2.424 c0.071,7.839,0.525,303.622,0.2
87,313.027c-0.076,2.911,0.388,3.277-2.411,3.284c-8.695,0.02-306.306-
0.824-314.243-0.888c-3.621-0.028-3.549-0.087-3.602-3.555c-0.089-
5.803,0.279-303.958,0.573-311.282c0.071-1.779,1.341-3.3,2.728-3.314
C92.983,79.645,389.112,79.938,398.01,80.044z"> /* end point data */

<animate attributeName="d" /* animation parameters */
dur="2s" /* animation duration */
repeatCount="indefinite"
/* keyframes follow */
values="M398.01,80.044c1.282,0.015,2.128,1.142,2.143,2.424 c0.
71,7.839,0.525,303.622,0.287,313.027c-0.076,2.911,0.388,3.277-
2.411,3.284c-8.695,0.02-306.306-0.824-314.243-0.888 c-3.621-
0.028-3.549-0.087-3.602-3.555c-0.089-5.803,0.279-303.958,0.573-
311.282c0.071-1.779,1.341-3.3,2.728-3.314 C92.983,79.645,389.112,7
.938,398.01,80.044z; /* data for starting shape (square) */

M354.385,111.599c23.167,21.333,59.115,72.234,57.382,124.901
c2.066,71-29.927,107.005-41.875,123.214c-21.597,21.106
59.392,62.413-136.483,62.413c-71.241,0-109.702-39.962-121.952-49.212
c-20.083-21.417-52.289-53.081-56.45 125.081c1.495-76.667,29.877-
107.214,56.277-135.263 c19.935-18.074,55.885-45.071,122.104-47.269C2
96.833,65.303,337.885,97.933,354.385,111.599z" /* data for target
shape (circle) */
/* end of keyframe data */
fill="freeze"
calcMode="spline"
keySplines="0.4 0.8 0.5 1" /* easing & faring on one morph */
keyTimes="0; 1"/> /* two keyframes as proportion of duration */

</path>
</svg>
```

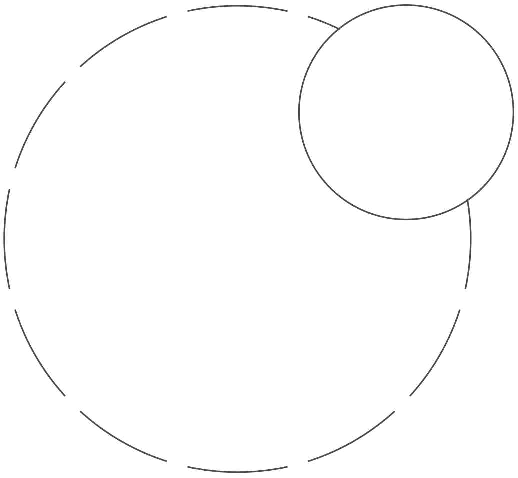
Moon path—morph



Mask (*above*) reveals & obscures circle underneath

```
/* Moon phase animation using one spline as a mask (left) to reveal
   the moon, can also do this with two splines see demo */

<svg xmlns="http://www.w3.org/2000/svg" /* define SVG parameters */>

<circle cx="200" cy="200" r="188" /* circle revealed behind mask */
stroke="#606060" fill="#606060"/> /* give circle a color */

/* define the spline (path) */
<path d="M200,10 C200,10,400,10,400,200 S200,390, 200,390
S400,390,400,200 C400,10,200,10,200,10z" stroke="#202020"
fill-rule="nonzero" fill="#202020"> /* color it */
<animate attributeName="d"
dur="10s" /* animation duration */
repeatCount="indefinite"

/* Three keyframes (Mxxx the start of each) */
values="M200,10 C200,10,400,10,400,200 S200,390, 200,390
S400,390,400,200 C400,10,200,10,200,10z;
 M200,10C200,10,400,10,400,200 S200,390,200,390 S0,390,0,200
C0,10,200,10,200,10z;
 M200,10 C200,10,0,10,0,200S200,390,200,390 S0,390,0,200C0,10,200,10
,200,10z;
fill="freeze"
calcMode="spline"
keySplines="1 1 1 1;1 1 1 1"/>
</path>

</svg>
```

Moon path—follow



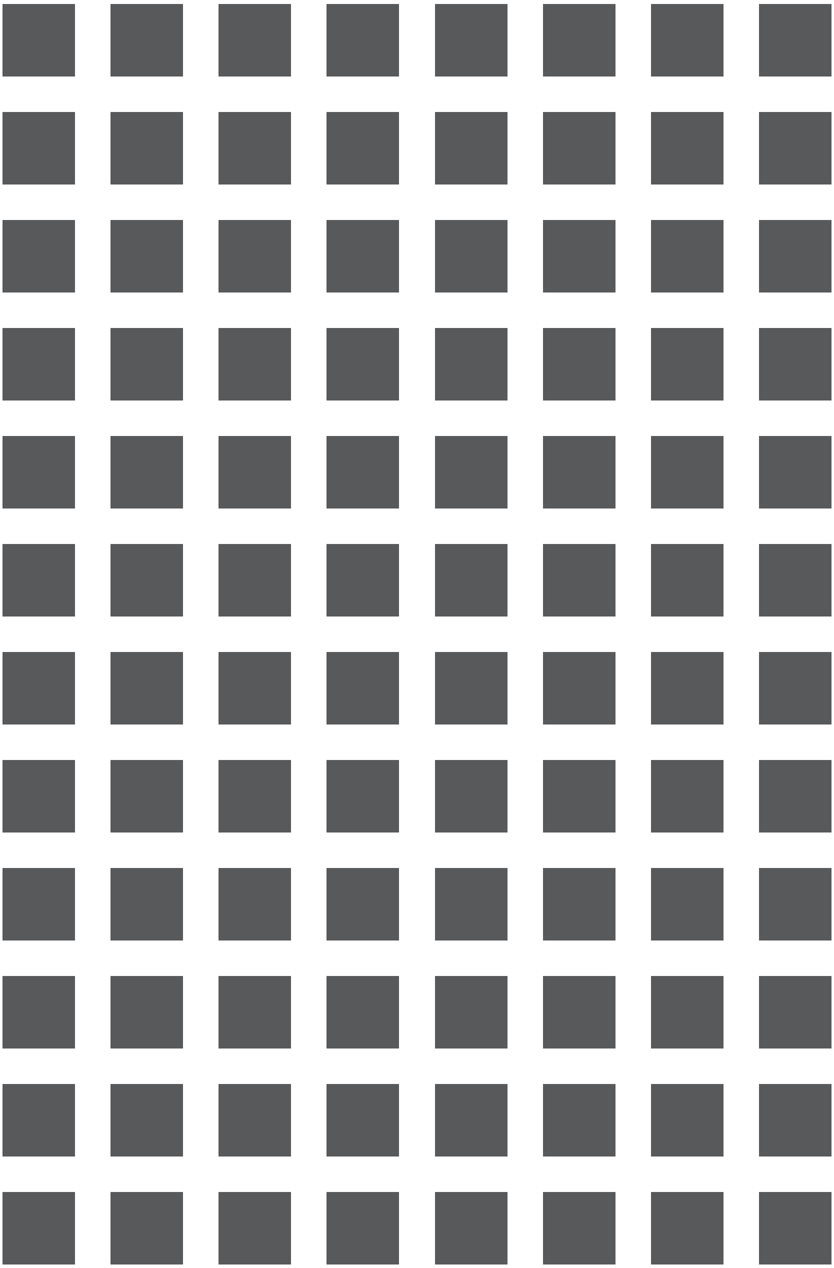
Animate an object along any shaped spline

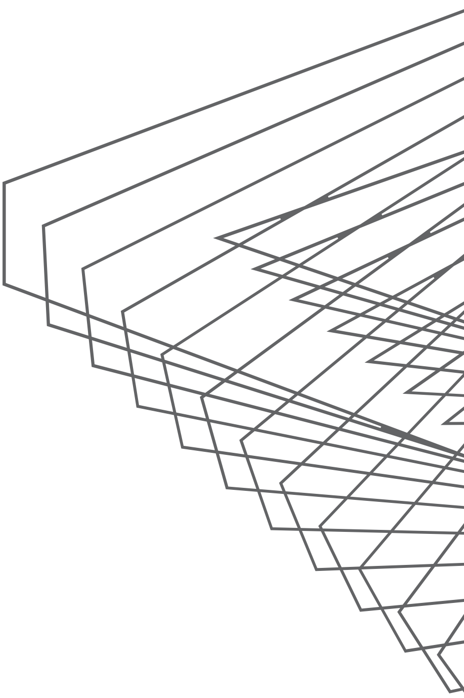
```
/* Takes moon phases and animates it along a path */

<svg width="360" height="360" viewBox="0 0 400 400" xmlns="http://
www.w3.org/2000/svg" /* define SVG parameters */>

/* define path named moonPath */
<path d="M240 130 C386,130 386,350 240,350 S 94,130 240,130 Z"
stroke="none" fill="transparent" id="moonPath"/>
<g id="moon">
<circle cx="-40" cy="-40" r="84" fill="#606060"/> /* moon */

<g transform="translate(-130,-130) scale(0.45)">
<path d="M200,0c0,0,200,0,200,200S200,400,200,400 s200,0,200-
200 C400,0,200,0,200,0z" stroke="#202020" fill-rule="nonzero"
fill="#202020">
<animate attributeName="d"
dur="10s"
repeatCount="indefinite"
values="M200,10 C200,10,400,10,400,200 S200,390, 200,390
S400,390,400,200 C400,10,200,10,200,10z; M200,10
C200,10,400,10,400,200 S200,390, 200,390 S0,390,0,200
C0,10,200,10,200,10z; M200,10 C200,10,0,10,0,200 S200,390,200,390
S0,390,0,200C0,10,200,10,200,10z;" /* moon spline animation */
fill="freeze"
calcMode="spline"
keySplines="1 1 1 1;1 1 1 1"/>
</path>
</g>

<animateMotion begin="0.0s" dur="12.s" repeatCount="indefinite">
<mpath xlink:href="#moonPath"/> /* sends moon around moonPath */
</animateMotion>
</g>

</svg>
```

Whenever I read a magazine I'm always curious about their grid, especially if it's about design or type. So for what it's worth here's mine. I'm using the typeface *Plantin MT Pro* for body copy and titles, and *Roboto Mono* for the code samples and folio. My primary body copy is 9.9/13.5 points, giving me my base unit of 13.5 pt. My grid modules are 2×2 units or 27×27 pt each, with one unit inbetween. Excluding the margins, my grid is 8×12 modules (*right*).The full-width is 310.5 pt, though I'm only using seven modules 270 pt/22p6 for single column text.

One early question I had was whether this small A5 format could accommodate a two column layout. I'd read that ~13 picas per column was decent, and there's also the 7–10 words per line guide. So I reduced the two column text to 8.9 pt, and it takes up all eight modules. Each 4-module column is 148.5 pt / 12.37 picas. I later came across a few A5 sized magazines that all used two column as their main layout, so that reassured me.

The typeface used on the cover is *WHOA* by Scribble Tone, which you can license from *www.futurefonts.xyz*. You can send me any feedback: *anna@stylecampaign.com*





Issue #01