HAVE YOUR SAY.

SUBMIT YOUR WORK, ESSAY,
ARTICLE OR RANDOM PIECE.
YOU'RE THE COMMUNITY,
YOU'RE OUR FEATURE.

SUBMISSIONS@LIBREGRAPHICSMAG.COM

WE WANT YOUR FEEDBACK.
TELL US WHAT YOU THINK.
WHERE WE COULD DO BETTER.
WHERE YOU COULD HELP.

FEEDBACK@LIBREGRAPHICSMAGAZINE.COM

DISTRIBUTE THE MAGAZINE.
GET YOUR LIBRARY TO ORDER A SUBSCRIPTION.
SPREAD THE WORD.
HELP US REACH OUT.

ENQUIRIES@LIBREGRAPHICSMAG.COM

New releases

Reclaim your tools

by Jakub Szypulka

http://vimeo.com/18568225

Documents the slow beauty and diversity of activity to be found at even the most hectic meeting of software contributors. In this case, documenting Libre Graphics Meeting 2010. Made using Kdenlive and Audacity.

AdaptableGIMP

http://adaptablegimp.org

A new version of GIMP, which allows users to make easy customizations. Read more about it on pages 46-50.

ArtistX 1.0

http://www.artistx.org/site3

A version of GNU/Linux which bills itself as able to turn a 'common computer into a full multimedia production studio.' Based on Ubuntu and designed for multimedia artists.

CrunchBang 10 Statler

http://crunchbanglinux.org

CrunchBang is version of GNU/Linux notable for its community of users who actively share screenshots of their modifications to the desktop. They share not only screenshots of their modifications, but also instructions for replicating their results.

What's new with you? We're always eager to find out what designers, artists and others using and working with F/LOSS are up to. Tell us what you've done lately at ***enquiries@libregraphicsmag.com***

Upcoming events

We're very pleased to present a calendar of upcoming events which encompass all things graphic design, media art and F/LOSS. Given that there are few events which tackle all three subjects, we aim to offer you events where you can be the agent of change: the F/LOSS designer at a traditional design event, or maybe the designer at a predominantly software developer event.

1-3 APRIL

Flourish 2011

CHICAGO

http://flourishconf.com/2011

30 APRIL 1 MAY

Linux Fest Northwest

BELLINGHAM, WASHINGTON

http://linuxfestnorthwest.org

10-13 MAY

Libre Graphics Meeting

MONTREAL

http://libregraphicsmeeting.org/2011

9-13 MAY

Icograda design week

VILNIUS

http://icograda.org/events/events/calendar738.htm

1-3 MAY

Pica 2011
BANFF, ALBERTA

http://picaconference.ca

2 MAY

agIdeas 2011: International design research lab
MELBOURNE

http://agideas.net/agideas-2011/design-research-lab

7-21 MAY

CHI 2011
VANCOUVER

http://chi2011.org

19-21 MAY

Typo Berlin
BERLIN

http://typoberlin.de/2011

19-22 MAY

Live Performers Meeting
ROME

http://liveperformersmeeting.net

Copyleft Business

Dave Crossland

Copyleft has a scary reputation among business people because they often do not understand it.

Copyright is easy — it's about what restrictions and freedoms you have to use and redistribute a work. Copyleft is a "pay it forward" feature of copyright licenses that says if you redistribute the work, you must pass it along on the same terms. You are free to take a libre work and improve it. You can take it as a part and combine it with your own parts to make a new, and hopefully better, thing. What makes copyleft powerful — and scary — is that if you choose to do this, the whole thing must be libre. You can stand on the shoulders of others but others can also stand on yours — or you can start from scratch and set your own terms.

Copyleft has been smeared as "viral" and "a cancer" because creators of proprietary software much prefer libre licenses without this bargain. Those licenses allow people to have their cake and eat it by exercising their freedom while denying others that freedom. Including libre parts in a proprietary whole defeats the original point of setting the work free, and copyleft is a good defense against this abuse. Copyleft is central to the most popular libre licenses for programs and creative works, in the GNU GPL and the Creative Commons Attribution-ShareAlike licenses respectively. Copyleft powers the explosive, exponential growth of share-and-share-alike culture. And, as always, fonts are special.

PostScript powered the early days of desktop publishing and it required the redistribution of complete fonts with documents. PostScript (PS) document files *linked* to font files. That was intensely annoying for proprietary font vendors because fonts were endlessly copied all without license fees being paid. Font Digital Rignts Management (DRM) schemes were cooked up over the years and found to be more trouble than they were worth. Being unable to print documents correctly is perhaps only slightly less annoying for designers than having an application crash and taking the work with it.

PDF solved this by combining fonts with documents or, ideally, the minimum parts of fonts needed for that particular document to print reliably. (Scribus has had faultless PDF export as a top priority since the beginning.) But this makes the story for copyleft fonts complicated. A copyleft font may overreach into the documents that use it, unless an exception is made to the normal terms — an additional permission to allow people to combine parts of a font with a document without affecting the license of texts, photographs, illustrations and designs. Most libre fonts today have such a copyleft license — the SIL OFL or GNU GPL with the Font Exception described in the GPL FAQ.

Web fonts return the world to linking documents to fonts. This is extremely unfortunate for the proprietary business world because people can see a font, like it, and figure out how to download and save it without paying for a proprietary license. It is, however, extremely fortunate for those doing business with copyleft works, because copyleft distribution is a wealth creation engine for those who know how to drive it. More distribution means more money.

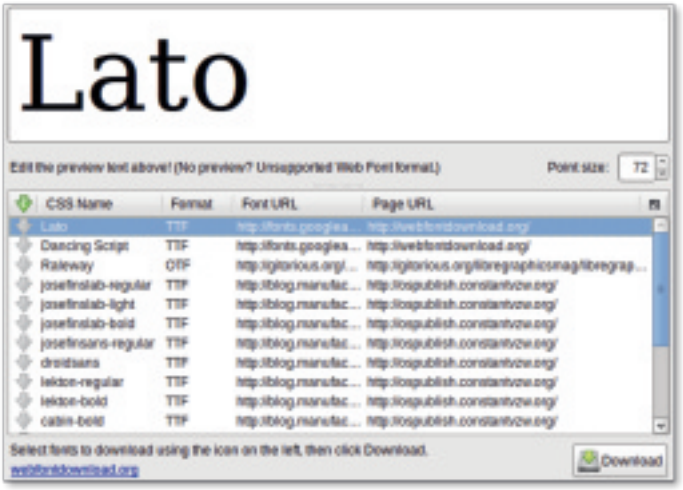
The business of libre fonts is open for designers who can take a libre font and combine it with their own parts to make a custom typeface design for their clients — customers who could not afford to commission totally new typefaces, but who still desire

The Web Font Downloader Firefox Add-On delivers the dream, making it easy to download libre web fonts.

fresh typographic identities. Since all businesses will want to use their fonts on their websites, participation in free culture is guaranteed by copyleft. If you see a great typeface on a web page and it has a libre license, you can download and save it and improve it further.

The Web Font Downloader Firefox Add-On delivers this dream, making it easy to download libre web fonts. The next step, improving the font further, highlights the issue of font sources. OpenType has two flavours, one with PostScript-style cubic outlines and the other with TrueType-style quadratic outlines. The PostScript flavor is superior as a font format and looks great on computers using FreeType and on Mac OS X, but lacks the pixel-level control of TrueType needed to look good on most Microsoft Windows computers. This means almost all web fonts are distributed in a format that is a long way from "the preferred form of the work for making modifications to it." That is the definition of source code in the GNU GPL, and it works very well for programs. I hope one day it will be a tradition for fonts too.

Get the Web Font Downloader Firefox Add-On now from
WWW.WEBFONTDOWNLOAD.ORG



Dave Crossland believes anyone can learn to design great fonts. He is a type designer fascinated by the potential of software freedom for graphic design, and runs workshops on type design around the world.
http://understandingfonts.com

The heritage of our pixels

Eric Schrijver

When John Whitney made his pioneering computer art films as an artist in residence for IBM in 1960, the computer screen he used did not use pixels. Rather, it was a single beam which could be instructed to move across the screen, much in the same way that postscript instructions tell a vector to move.[1]

> When in doubt, look at your predecessors. Most of our historic design for the computer screen is bitmap-based.

The graphics in Atari's arcade games, like Battlezone, were also drawn with vector lines on an oscilloscope.[2] In the long run, a matrix of points became the preferred method to describe screen output. And it still is today. In fact, we have a more rigid matrix now that we use LCD displays: they have a "native" resolution determined by the number of pixel elements — whereas the phosphor dots in a color CRT display bear no relation to pixels or subpixels displayed on them.[3]

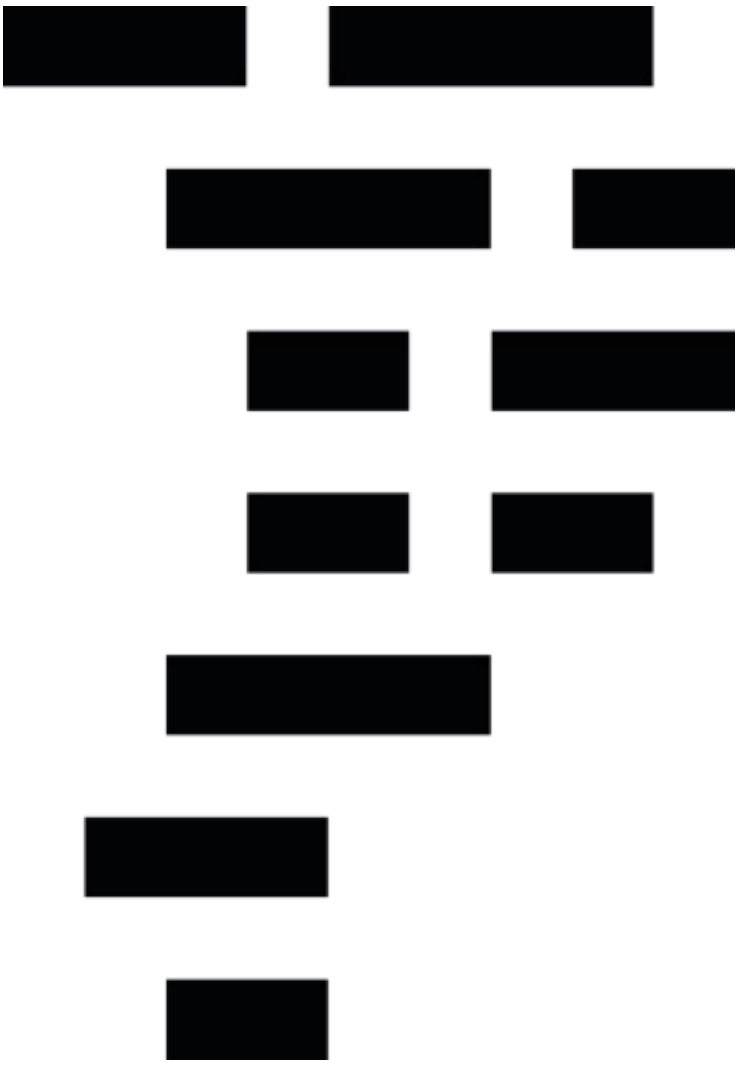
So, to get your digital resources out on a computer screen, you have to describe them as a matrix of points. That's easiest when you work with data that itself is a matrix of points. It's even easier when you map the matrix of points directly in the data to the matrix of the points on the screen.

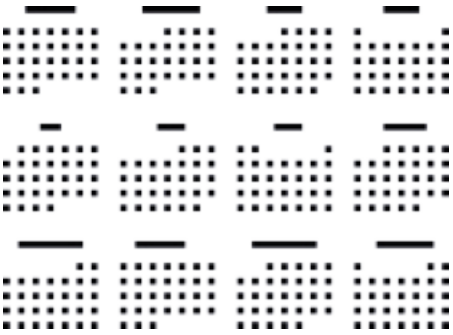
The easiest solution is not the best, in this case. Try to browse the internet on a 24 inch screen and, by default, it will look rather awkward: singular columns of 960 pixels, with huge swaths of background image on either side. That is because the layouts are specified in CSS pixels and, by default, the browser makes them correspond with "device pixels".[4] Where this used to be a predicament, now it's just a convention. All modern browsers support zooming in on the content. They're able to make pixel-based layouts smaller or bigger.

On the mobile phone, the rapport between the pixel of the design and the pixel of the screen has been cut entirely. The webpage is initially displayed fully, and subsequently the user can zoom, continuously, in and out on the design.

Scalable user interfaces benefit from vector graphics. After all, vector graphics are supposed to shine in the world of scalable.[5] There's even a vector format that was named after this inherent property: Scalable Vector Graphics. But does that mean we can't use the model of the bitmap in our new designs and interfaces? Not necessarily.

A city
Beatiful abstractions in Anthony's icons





Above: **A calendar.**
Left: **A tornado (from Nethack).**

When in doubt, look at your predecessors. Most of our historic design for the computer screen is bitmap-based. I like to look at early pixel-based GUIs for inspiration. There's a library of icons and user interface elements for the X window system, collected by Anthony Thyssen, available online.[6] Because of the limitations inherent in early systems, many of them are really simple, black and white, 16x16 bitmaps. Through tight constraints, they attain a very evocative kind of abstraction. In this they resemble Susan Kare's icon designs for the original Macintosh, which are much better executed than current iterations.

These designs don't deserve to stay locked to the grid of display pixels growing ever tinier. They also don't have to: you could paint these designs with square meter pixels on your wall, with even that rendering making them look great.

But what better way to reinterpret these designs than to convert them to vectors?

Traditional tracing algorithms do no justice to these designs. Looking for the curves underlying the designs ignores that the pixel grid is constitutive of the design. We are not looking for the platonic ideal. In this case, there's nothing to do but make vector pixels: a vector square for every pixel! It doesn't even have to be a square. After all, a bitmap is a collection of points, and points have no predefined shapes. It could be circles or any arbitrary shape. You could make the pixels come together in horizontal scanlines, or vertical ones. You could distort the grid on which they are based.

There are many possibilities in the future of rendering and the further we go in exploring them, the closer we come to keeping alive the heritage of our pixels.

---

1. Thanks to Joost Rekveld for his classes, introducing these works amongst others
2. Form and Code, In Design Art and Architecture: Casey Reas, Chandler McWilliams, LUST; Princeton Architectural Press 2010
3. http://en.wikipedia.org/wiki/Pixel
4. http://webkit.org/blog/55/high-dpi-web-sites
5. Actually, there are quite some pixel based scaling algorithms too: http://en.wikipedia.org/wiki/Pixel_art_scaling_algorithms
6. My reissue available at https://github.com/codingisacopingstrategy/AIcons

Eric Schrijver (Amsterdam, 1984) is a visual artist who makes installations and performances. Eric teaches Design for new media at the Royal Academy of Art in The Hague. He is inspired by open source and programming culture. http://ericschrijver.nl

Want to make your own vector pixels? Follow these (relatively easy) steps to generate your own vector pixel icons.

The following instructions should work just fine on either Linux or Mac.

Grab the code: Either type it in by hand, copying the code [on the right] or go to the assets sections of our website (HTTP://LIBREGRAPHICSMAG.COM/ASSETS) and download the vector pixel pack we've prepared for you.

If you're copying out the code manually, enter it into a text editor and call the file `vectorpixel.py`.

Find an image: If you're doing it on your own (instead of using the assets we've provided), find a simple image. Make sure it has very few colours (you're going to have to strip all the colour out of it). Simple logos, warning signs and similar types of images work well. Open it up in your favourite raster image editor (we used GIMP).

Strip out the colour by doing things like increasing the contrast as much as possible and posterizing. You're aiming to have an image with only black and white. While you're at it, resize the image to a very small size. 50px by 50px works well.

WARNING! We're serious about the small image size. If it's too big, the resulting SVG will be very, very big and may just crash your image viewer.

Save your image (as something like a PNG, JPG or other basic raster format). Make sure to flatten while you're at it. Layers will only cause trouble in this case. Make sure you save it in the same directory as your `vectorpixel.py` file.

Point the script: Take another look at `vectorpixel.py`. On the 8th line, you'll find something that looks like this: `SOURCEIMAGE = 'city.png'`. If you've made an image of your own, you'll want to change `city.png` to whatever the name of your file is. Then save `vectorpixel.py` again. Now, when you run it, it'll be looking for the right image.

Convert it: Open up your terminal (for more on using the terminal, check out the detailed instructions and explanation on pages 22-23). Navigate to the directory containing `vectorpixel.py` and your image.

At the prompt, type: `python vectorpixel.py > city.svg`

If you've provided your own image, you can change that last bit. For example, if your source file is called `attention.png`, you can sub in `attention.svg`. All this does is set up a destination file.

Hit enter. It'll look a little like nothing has happened. However, if you go and take a look in your folder, you'll find a new file, called `city.svg` (or whatever you've named it). Take a look at it. It should be made up of lots of little vector pixels.

You've just made a vector pixel icon!

```
#!/usr/bin/env python
""" Generates vectorpixels based on 2-bitmaps (2 color pictures).

    TODO: use element tree for XML; implement Floyd-Steinberg
    dithering for color and greyscale images; implement vertical
    and horiontal scanlines """

import Image

SOURCEIMAGE = 'city.tiff'

class vectorpixel:
    def __init__(self, image):
        self.i = Image.open(image)
        self.px = self.i.load()
        self.constructed = False

    def construct(self, grid=24, line=1, rounded=4, test=(lambda x: x == 0)):
        self.grid = grid
        self.line = line
        self.rounded = rounded
        self.width = self.height = self.grid - 2 * self.line
        self.test = test
        self.fill = '#000000'
        self.constructed = True

    def _yieldlocations(self):
        for x in range(self.i.size[0]):
            for y in range(self.i.size[1]):
                if self.test(self.px[x,y]):
                    yield (x,y)

    def _mkelements(self):
        for l in self._yieldlocations():
            yield "<rect x='%s' y='%s' width='%s' height='%s' rx='%s' fill='%s'/>" % (
    self.grid * l[0] + self.line, self.grid * l[1] + self.line, self.width, self.height, self.rounded, self.fill)

    def _format(self):
        output = '<svg xmlns="http://www.w3.org/2000/svg" width="%s" height="%s">\n' % (self.i.size[0] * self.grid, self.i.size[1]
* self.grid)
        for e in self._mkelements():
            output += e
            output += '\n'
        output += '</svg>'
        return output

    def generate(self):
        if not self.constructed:
            self.construct()
        return self._format()

if __name__ == "__main__":
    v = vectorpixel(SOURCEIMAGE)
    print v.generate()
```

Coding pictures

Ricardo Lafuente

At the Fine Arts Faculty of Porto University, we built up an introdutory class focusing on procedural strategies inside a graphic design context. In less stuffy terms, the purpose was to introduce design students to code. However, this required some thought on what subjects to teach (and which to leave out), which pitfalls to avoid, and which approach would work best to introduce an alien, cold and logical subject such as programming to creative people.

Designers are inevitably involved with computers, which are present in most stages of a graphic designer's workflow, from initial sketches to printing finished pieces. Yet there's a dearth of formal education on the technical and social workings of computers and digital media in general.

Nevertheless, attention has been paid to this field during the last decade, which saw the birth and growth of creative-oriented applications, spearheaded by its most popular example, Processing. Among other creative coding tools, we find Pure Data, Context Free, Drawbot, Nodebox, Shoebot, Supercollider and Fluxus. The overwhelming majority of these tools are F/LOSS.

Learning to code is becoming more and more of an obvious choice for designers. The rising popularity of the web has created a huge demand for skilled coders, but designers are also a key part of any serious venture. A designer who can implement his own ideas, instead of just handing over mockups to a web developer, ends up with a big advantage. Becoming acquainted with digital logic, the workings of computers and their bells and whistles is also a way to liberate oneself from being a software operator, and be able to think for and with the machine.

TOOLS AND STRATEGIES

In our semester-long class, we focused solely on still, static output, meaning that animation and interactivity were left out. This gave room to explore the basic commands and features, as well as combining them with creative strategies that the digital medium enables, such as repetition and randomness.

Processing was considered as the application for this class, but Nodebox/Shoebot were picked because they work natively with vector graphics, which was a crucial factor when considering that the created designs should be meant for print. The fact that they're based on Python, whereas Processing is based on Java, also played a part. Python is one of the most appropriate