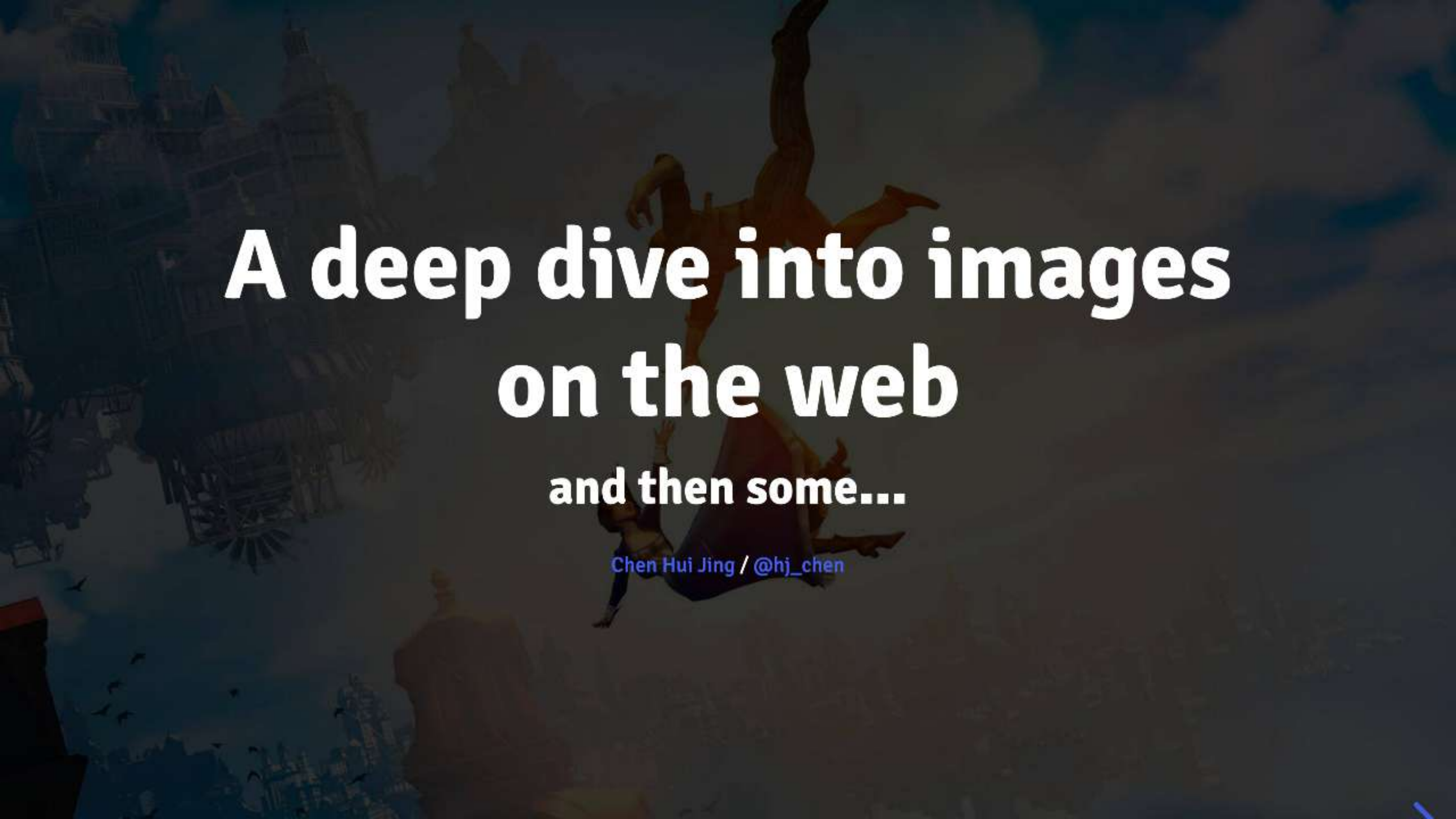
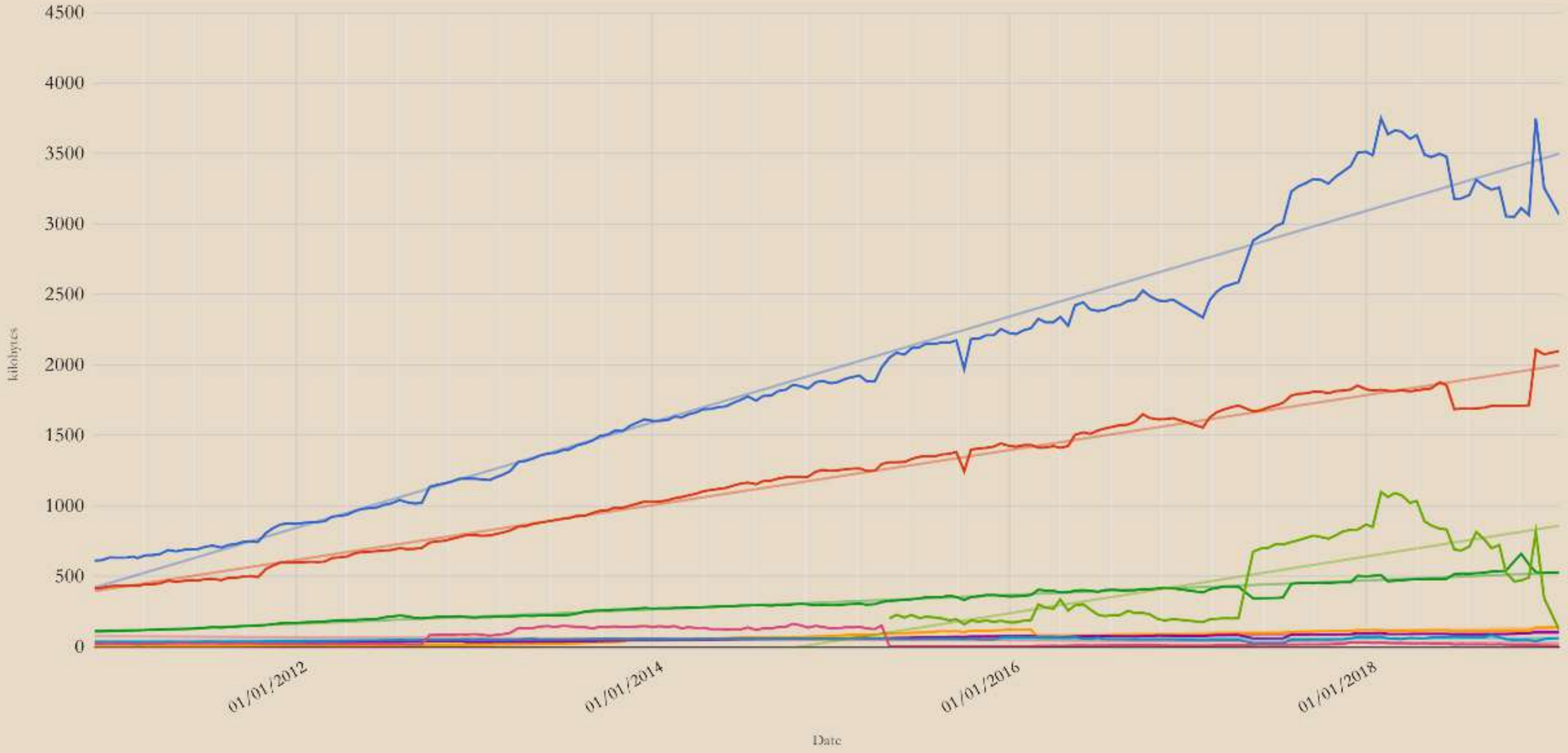


A deep dive into images on the web

and then some...

Chen Hui Jing / @hj_chen

Page Weight Chart



Surname First name

陈 慧 晶

Chen Hui Jing

@hj_chen

🥑 Developer Advocate 🥑

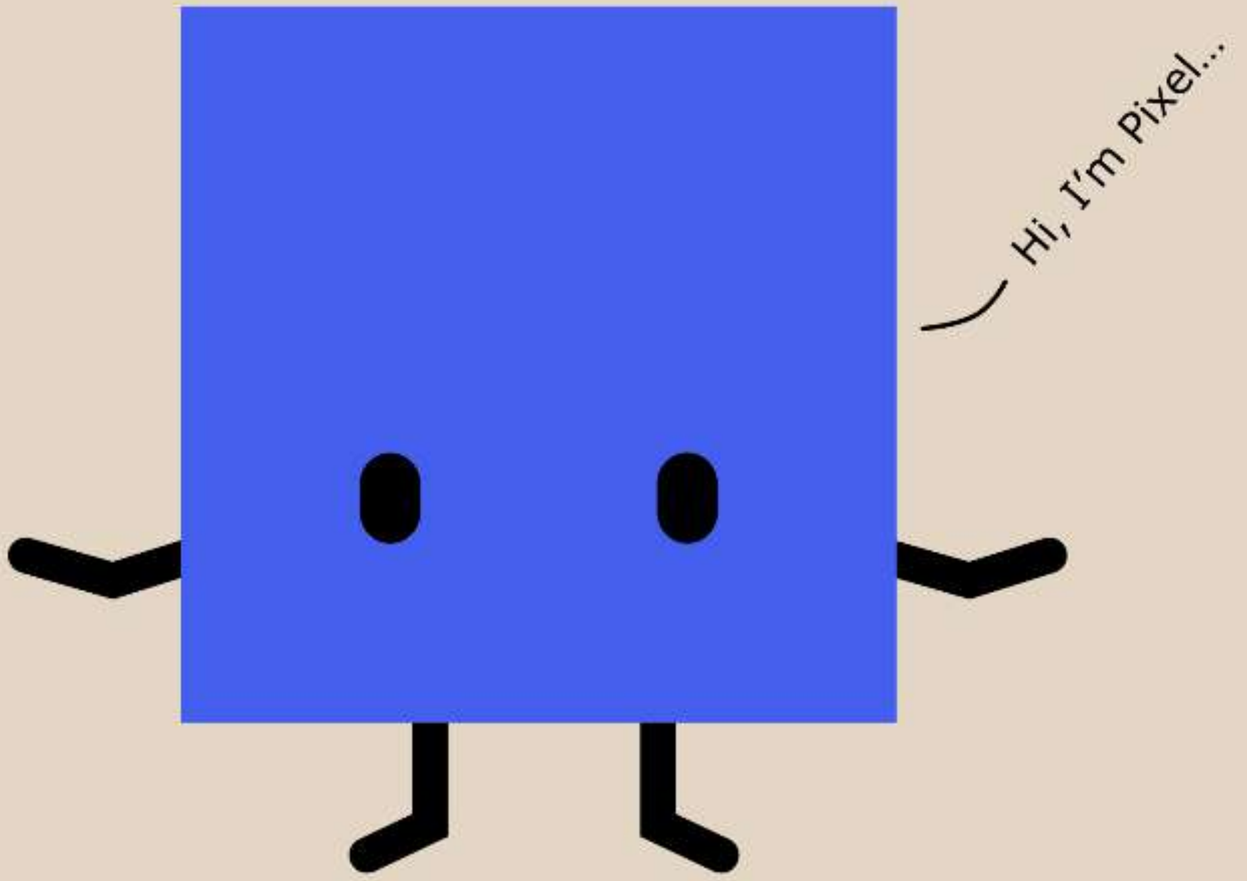
nexmo®

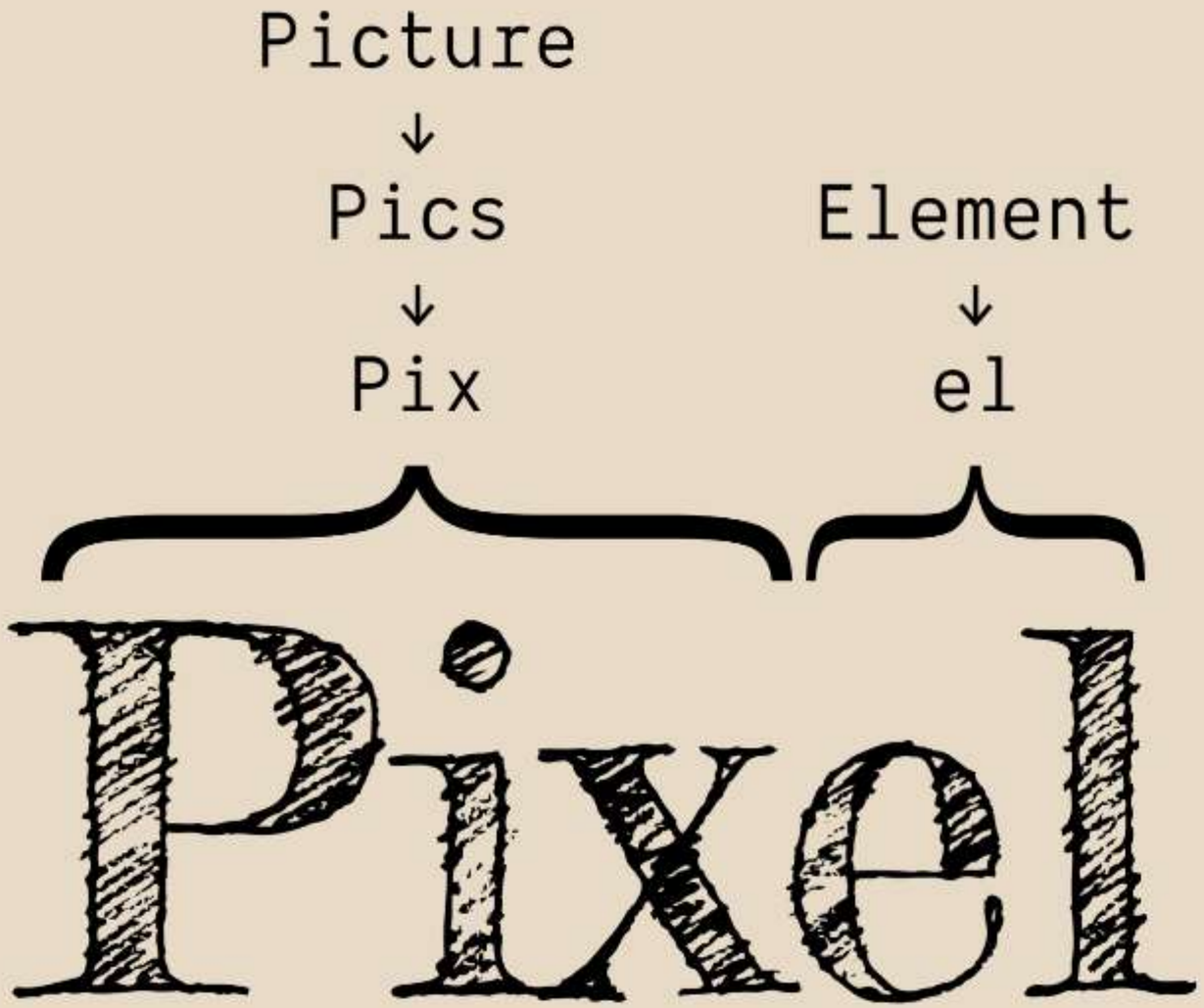
The Vonage® API Platform

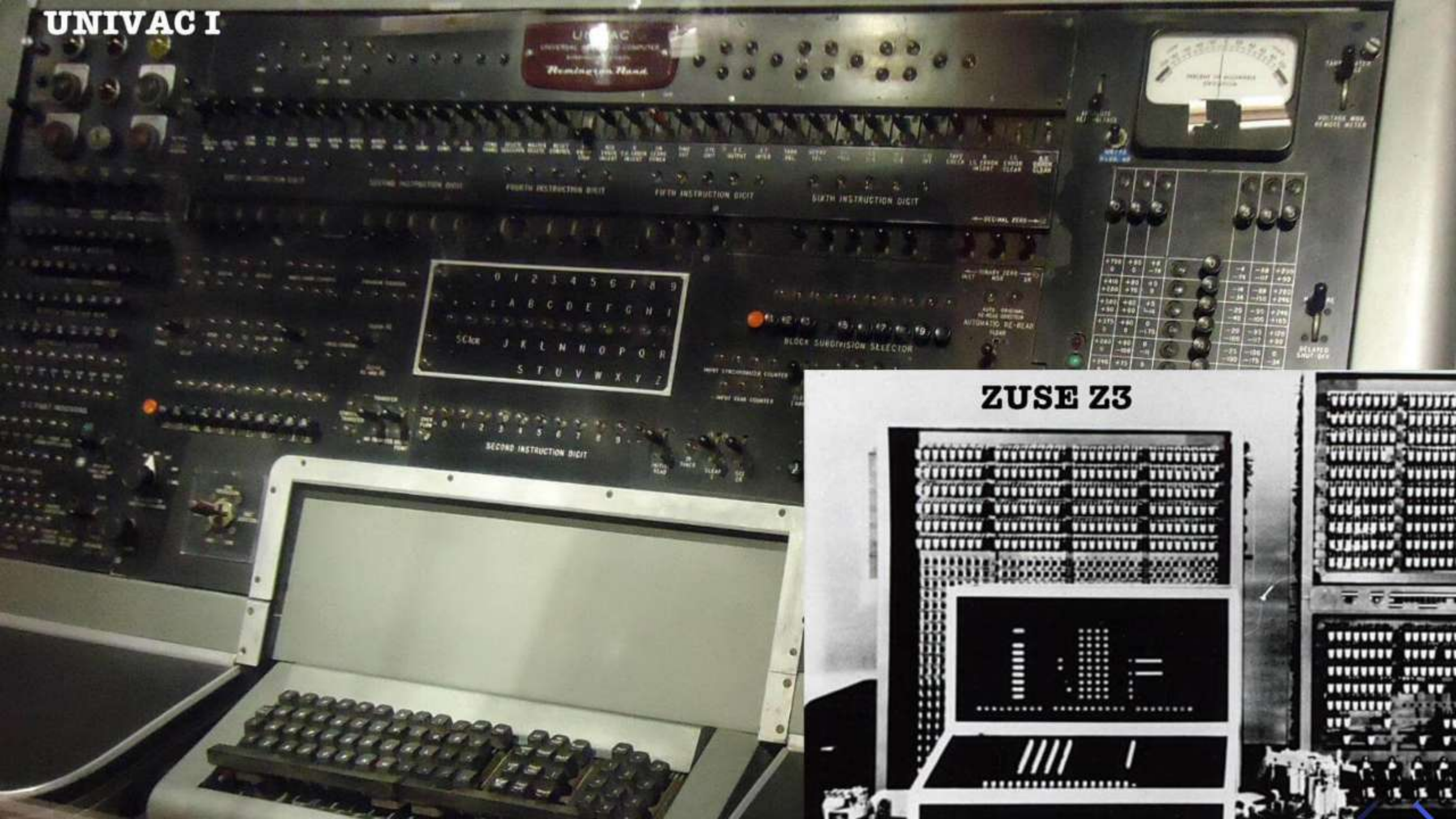

Primate Research Institute


TROUT
APPLES

ELECTRONIC SIGNALS


Hi, I'm Pixel...

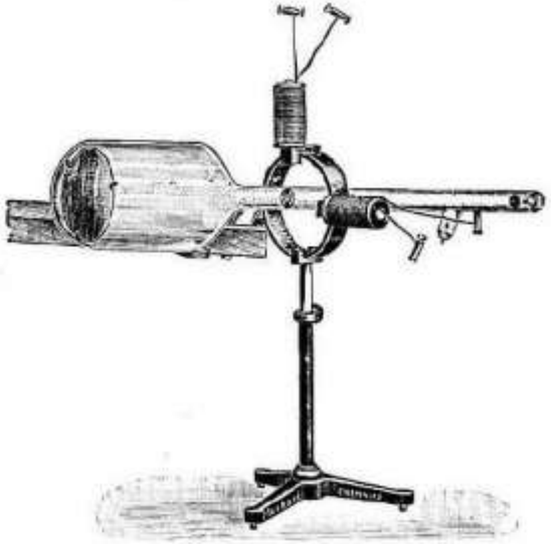

Picture
↓
Pics
↓
Pix
Element
↓
el
Pixel


UNIVAC I
Remington Rand
FOURTH INSTRUCTION DIGIT
FIFTH INSTRUCTION DIGIT
SIXTH INSTRUCTION DIGIT
SECOND INSTRUCTION DIGIT
BLOCK SUBDIVISION SELECTOR
ZUSE Z3

94 195. 1 : 9.

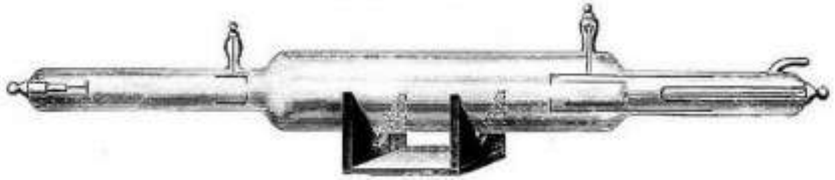
94 194. 1 : 6.

94 196. 1 : 6.



94 200 und 94 194. 1 : 8.

94 197. 1 : 5.



94 198. 1 : 5.

94 194. **Kathodenstrahlenröhre** nach Braun, Figur (Wied. Ann. 1897, Seite 552). Beschreibung

94 195. — dieselbe, größer, Figur, 1 m lang, mit Schirm von 130 mm Durchmesser

94 196. **Vakuumröhre** nach Thompson, Figur, zum Studium der Ablenkbarkeit der Kathodenstrahlen durch statische Elektrizität und durch Magnet . . . . . .

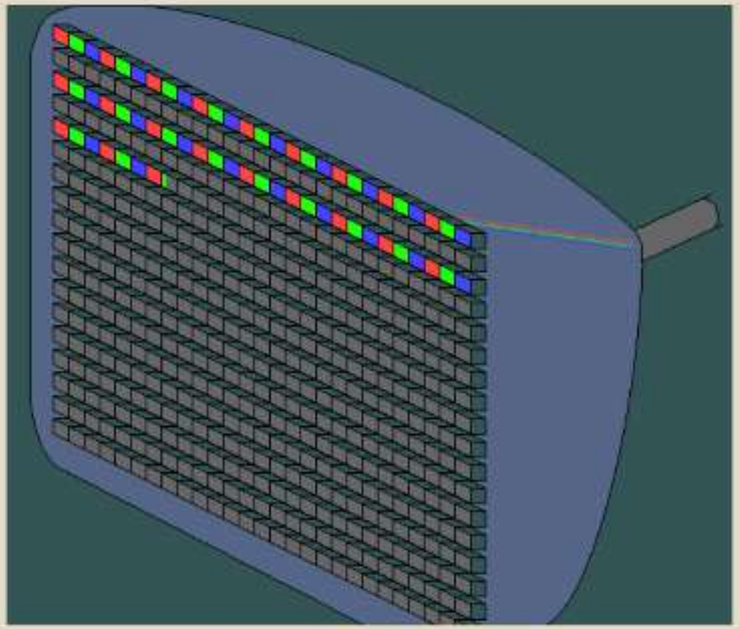
94 197. — desgl. nach Braun-Wehnelt, Figur, für elektrostatische Ablenkungen, mit Faradayschem Käfig als Diaphragma . . . . . .

94 198. **Vakuumröhre** nach Perrin, zur Vorführung der durch Kathodenstrahlen hervorgerufenen negativen Ladung, Figur, mit Fuß . . . . . .

94 199. **Universal-Stativ** nach Simon und Reich, für Versuche mit der Braunschen Kathodenstrahlenröhre, Figur . . . . . .

CRTs sold by Müller-Uri, Source: The Cathode Ray Tube site

Raster scanning



Source: M-SYS MV

	Raster scan	Random scan
Electron beam	Swept across entire screen, one row at a time, from top-to-bottom	Only directed to parts of the screen where image is drawn
Resolution	Poor, due to plotting as discrete point sets	Good, as CRT beam directly follows line path
Picture definition	Stored as set of intensity values (pixels) in refresh buffer area	Stored as set of line drawing instructions in display file
Realism	Variable intensity values allow for realistic shadow and colour patterns	Most suited for line drawing
Drawing method	Screen points (pixels)	Mathematical functions

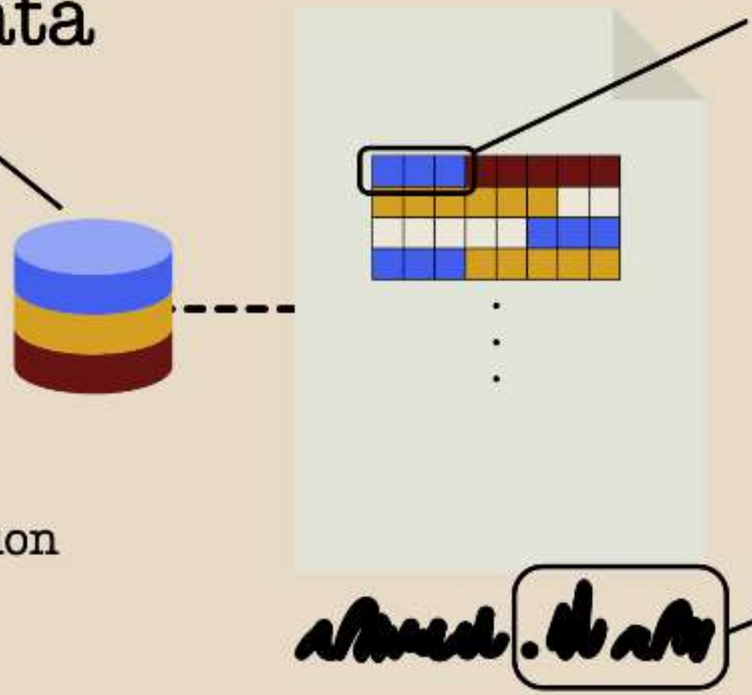
Source: Prof. Vijay M. Shekhat, CE Department, Computer Graphics


SVG
JS
PNG
JPG
DOC
DOC
XML
PDF
ZIP
MP3
MP4
EPS
AVI
AI
FLA
PSD
BIN
EXE
ICO
MKV
WMV
MOV
PPT
ISO
GIF
JAR
VCF
OBJ
HTM
DLL
ASP
DWG
EML
TXT
3DS
INI
OTF
TTF
PKG
COM
NFO
WAV
RTF
XLS
CSV
DBF
CSS

File formats

External metadata

Mac OS type-codes
Mac OS X UTIs
OS/2 extended attr.
POSIX extended attr.
PUIDs
MIME types
FFIDs
File content-based identification



Internal metadata

File header / Magic number

File extension

Colour depth

1-bit PNG
(2 colours)

2-bit PNG
(4 colours)

4-bit PNG
(16 colours)

8-bit PNG
(256 colours)

24-bit PNG
(16,777,216 colours)

Source: Wikipedia, Color depth

MacPaint

ST Picture Formats

Edited by:

David Baggett

Internet:
dmb@ai.mit.edu

(Please report errors or additions.)

Introductory Information

word = 2 bytes
long = 4 bytes
palette = Hardware color palette, stored as 16 words. First word is color register zero (background), last word is color register 15. Each word has the form:

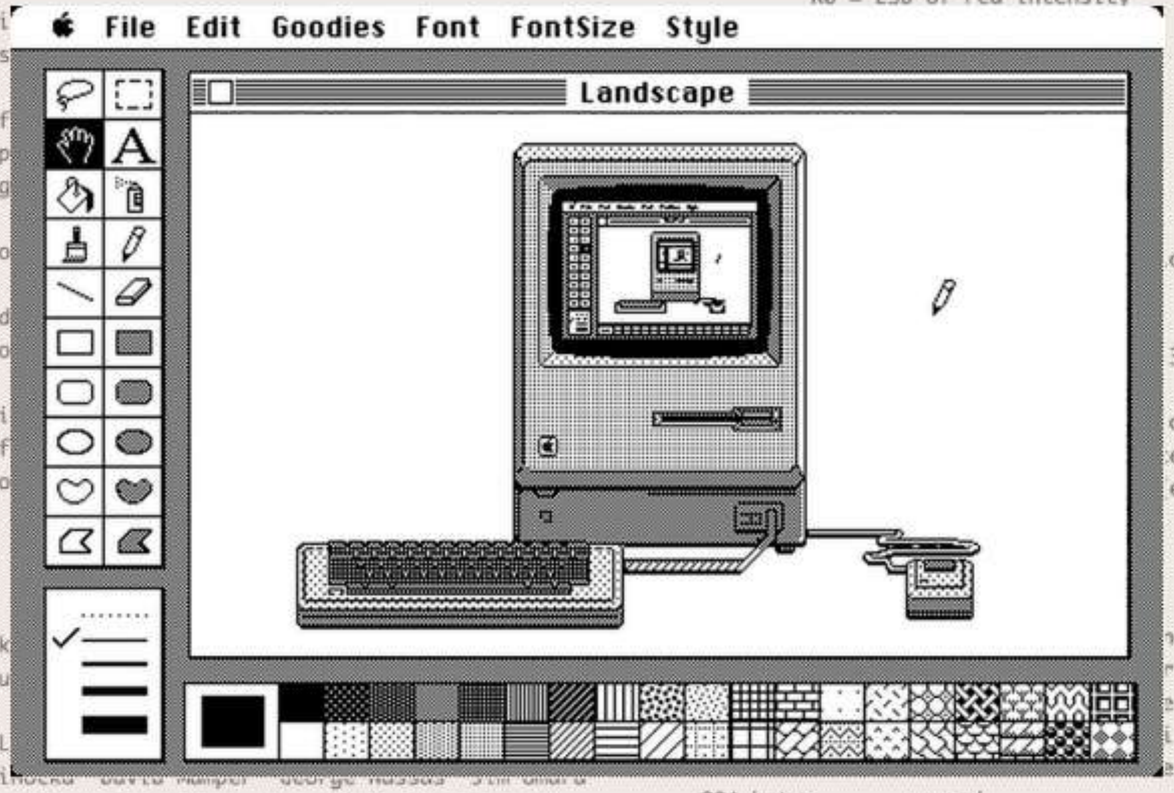
Bit: (MS 01 00 (LSB)
-- --
B1 B0

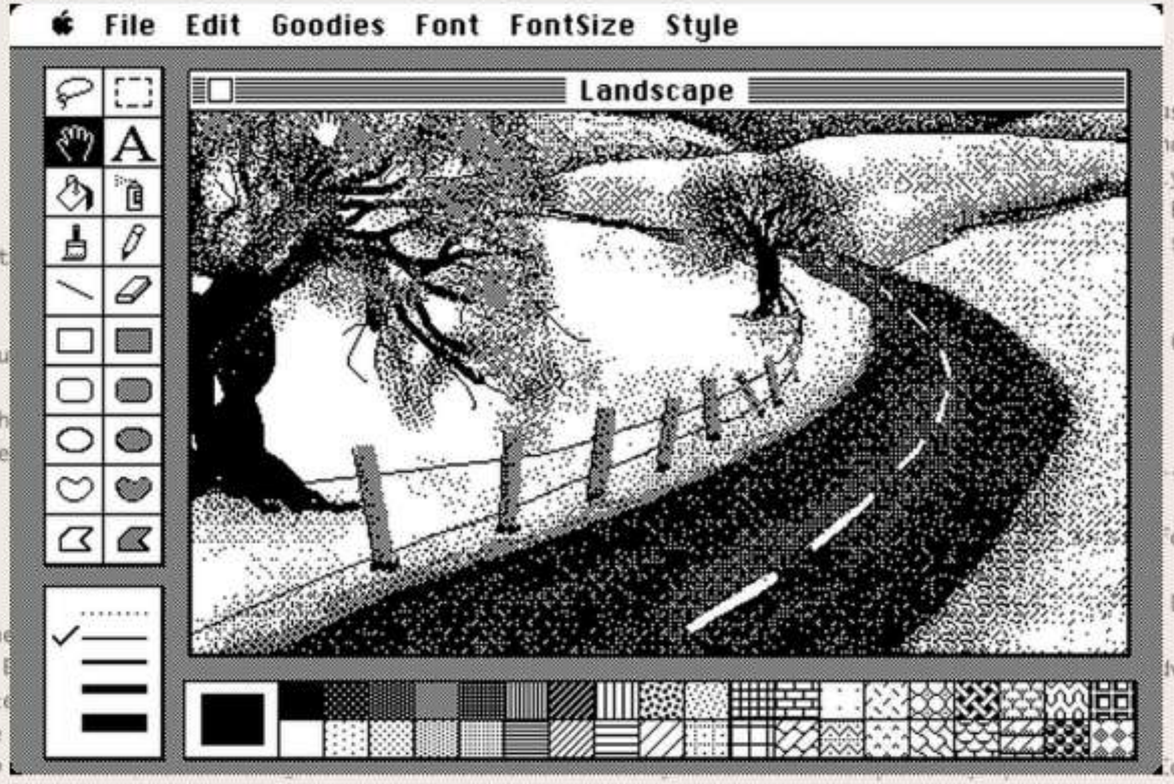
R2 = MSB of red intensity
R0 = LSB of red intensity

NOTE: The version number is actually a flag to MacPaint to ind the brush/fill patterns are present in the file. If the version the default patterns are used. Therefore you can simply save a file by writing a blank header (512 $00 bytes), followed by the image data.

Bitmap compression:

The bitmap data is for a 576 pixel by 720 pixel monochrome im The packing method is PackBits (see below). There are 72 bytes scan line. Each bit represents one pixel; 0 = white, 1 = black.





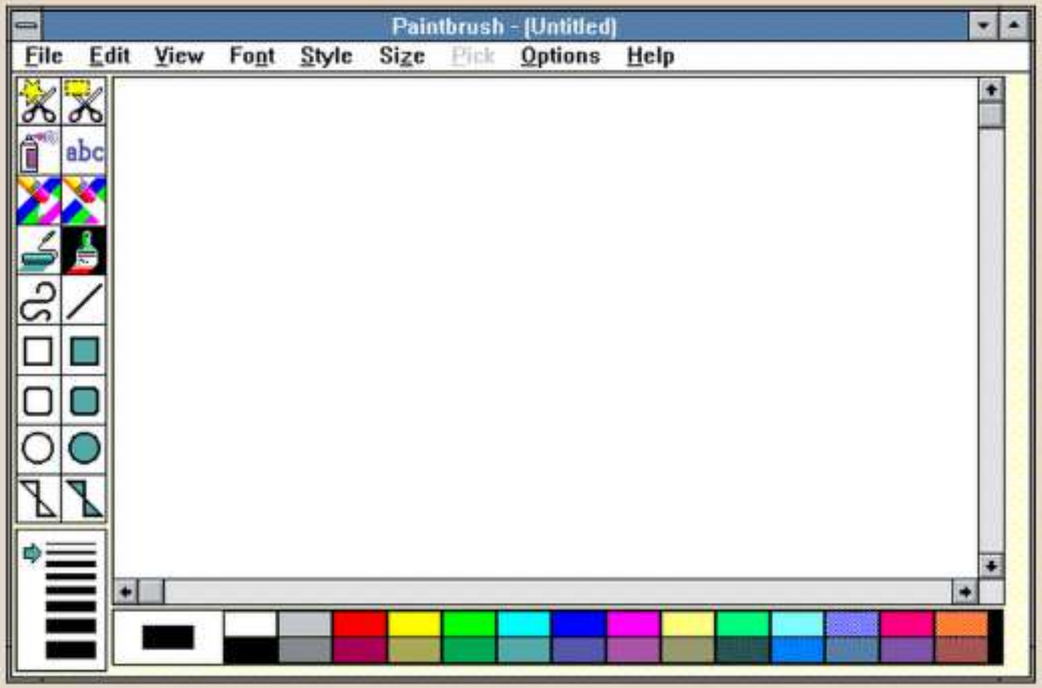
Chris Ridd George Seto Joe Smith Greg Wageman
Roland Waldi* Gerry Wheeler

204 bytes	unused
512 bytes	total for header
< 51200 bytes	compressed bitmap data
< 51712 bytes	total

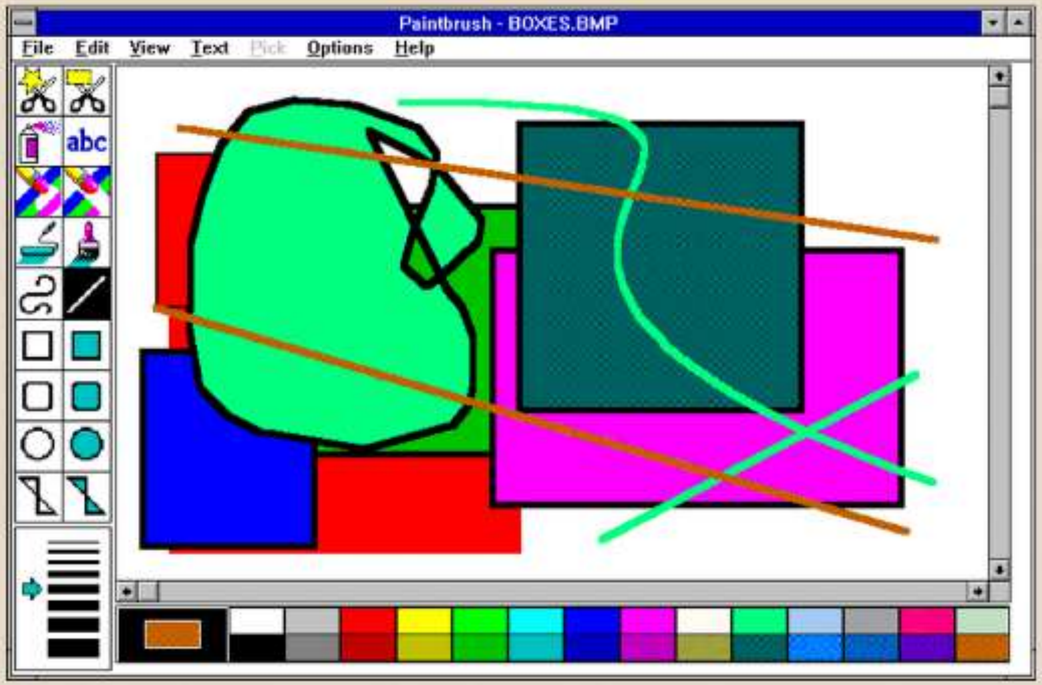
GEM Bit Image, IFF, MacPaint

Version of.....Sun Oct 30 12:40:13 EST 1994
(Last change: Extended GEM .IMG format updated)

MS Paintbrush

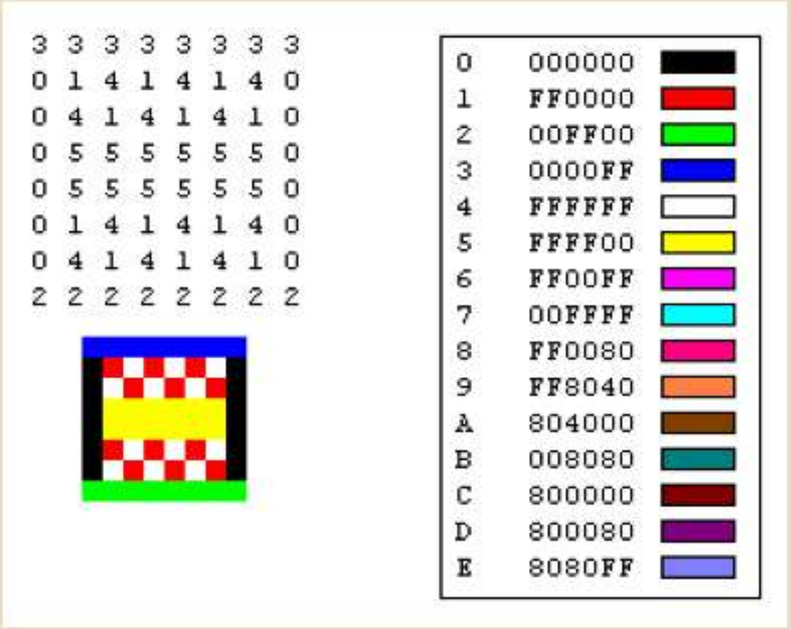


Paintbrush for Windows 3.0



Paintbrush for Windows 3.1

Bitmap (BMP)



```
3 3 3 3 3 3 3 3
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
0 5 5 5 5 5 5 0
0 5 5 5 5 5 5 0
0 1 4 1 4 1 4 0
0 4 1 4 1 4 1 0
2 2 2 2 2 2 2 2
```

0	000000
1	FF0000
2	00FF00
3	0000FF
4	FFFFFF
5	FFFF00
6	FF00FF
7	00FFFF
8	FF0080
9	FF8040
A	804000
B	008080
C	800000
D	800080
E	8080FF

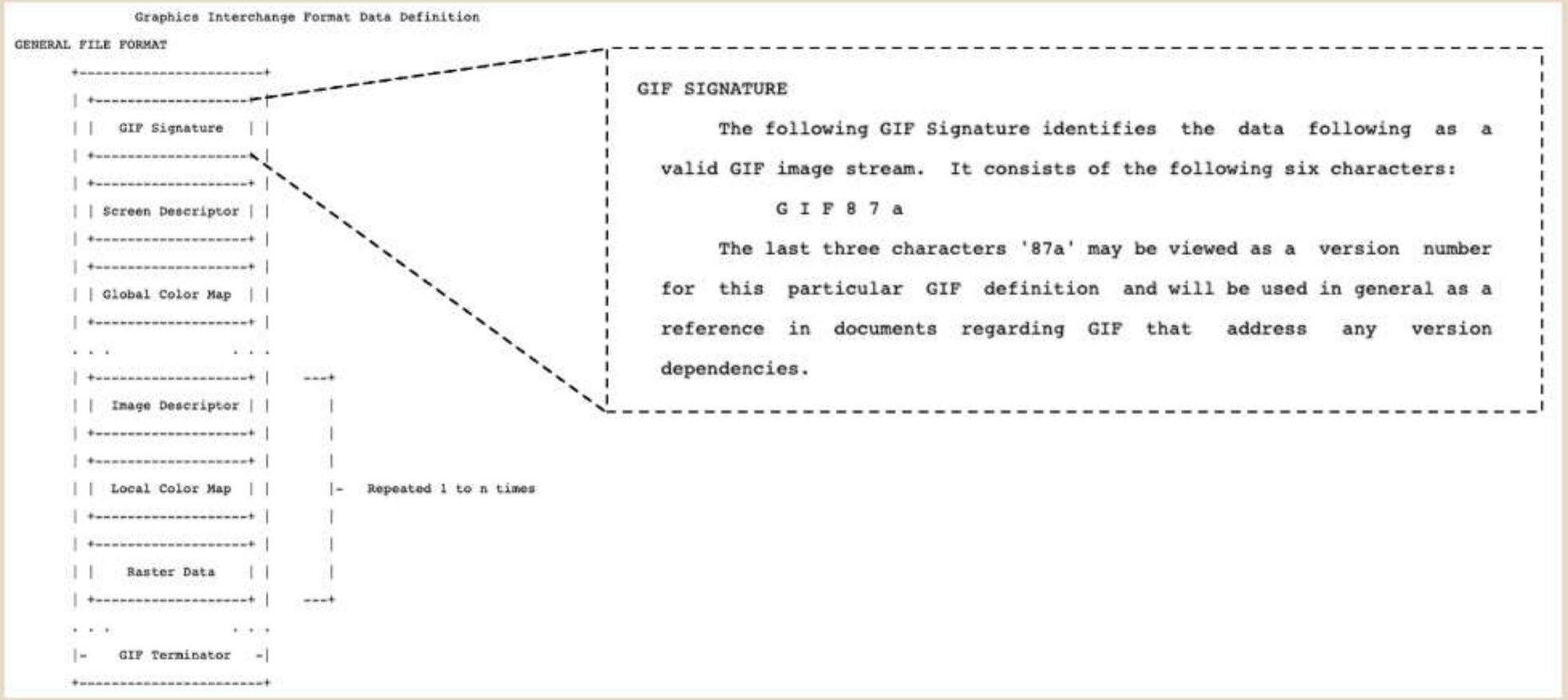
Bitmap and its corresponding colour table

```
0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
00FF00 FF0000 FFFFFF FF0000 FFFFFF FF0000 FFFFFF 00FF00
00FF00 FFFFFF FF0000 FFFFFF FF0000 FFFFFF FF0000 00FF00
00FF00 FF0000 FFFFFF FF0000 FFFFFF FF0000 FFFFFF 00FF00
00FF00 FFFFFF FF0000 FFFFFF FF0000 FFFFFF FF0000 00FF00
00FF00 FF0000 FFFFFF FF0000 FFFFFF FF0000 FFFFFF 00FF00
00FF00 FFFFFF FF0000 FFFFFF FF0000 FFFFFF FF0000 00FF00
0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF 0000FF
```

Colours directly in bitmap itself

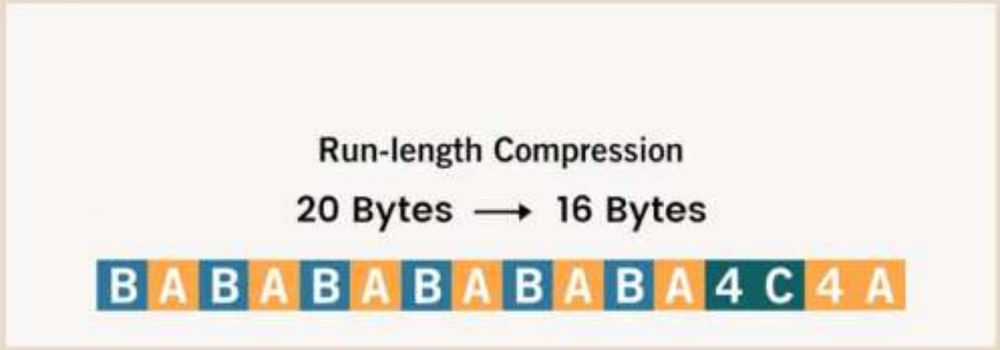
Source: Microsoft, Types of Bitmaps

Graphics Interchange Format (GIF)

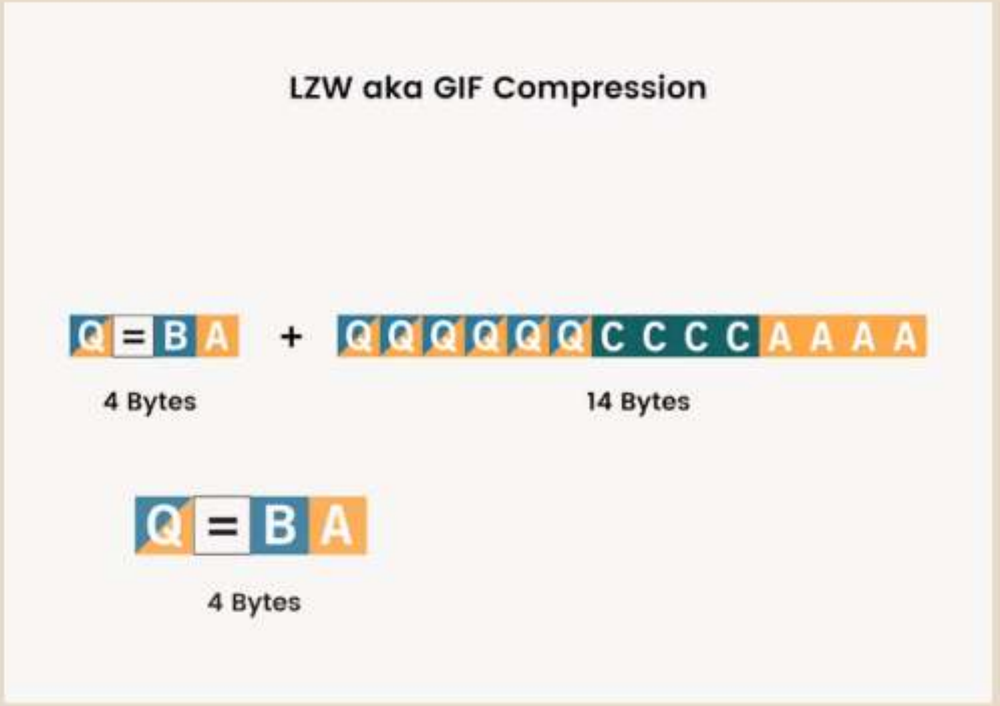


Source: Graphics Interchange Format (tm)

Compression algorithms



Run length compression used by MacPaint



Lempel–Ziv–Welch (LZW) compression

Motion graphics by Crystal Law

First band photo on the web

Source: The Cernettes

Joint Photographic Experts Group (JPEG)

JPG compression

Source RGB image

Colourspace conversion

Image split into blocks

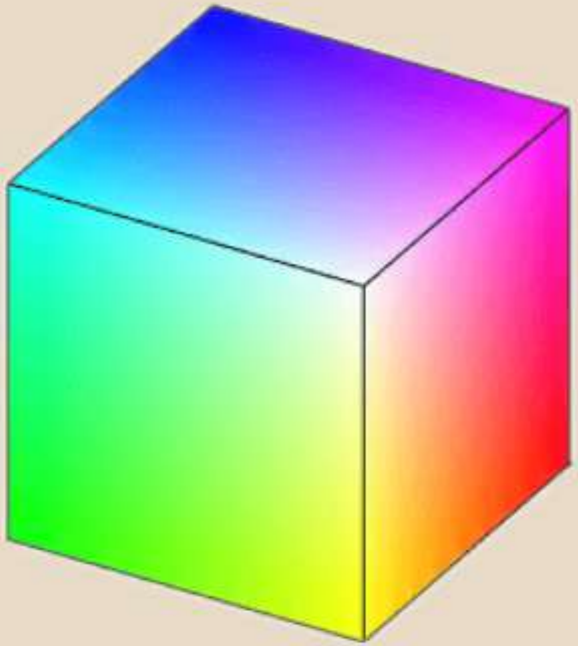
Forward Discrete Cosine Transform

Quantisation

Statistical encoding

.JPG file

Reference: How JPG works



Colourspace conversion

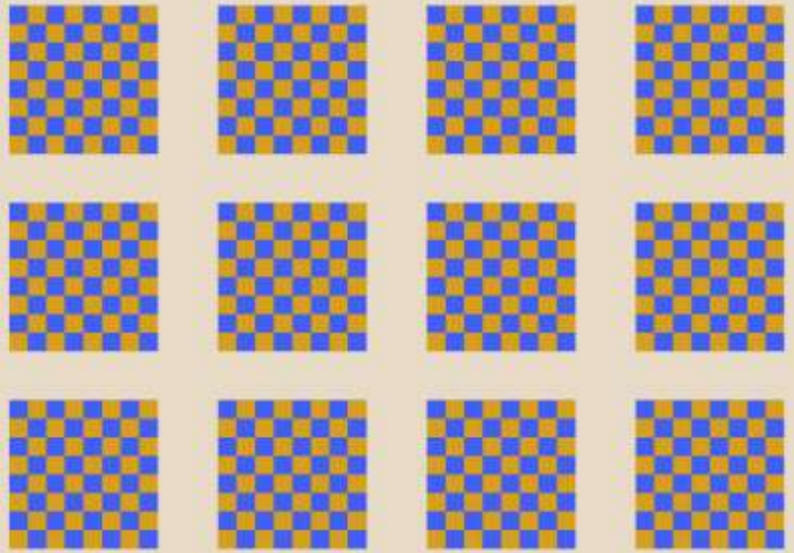


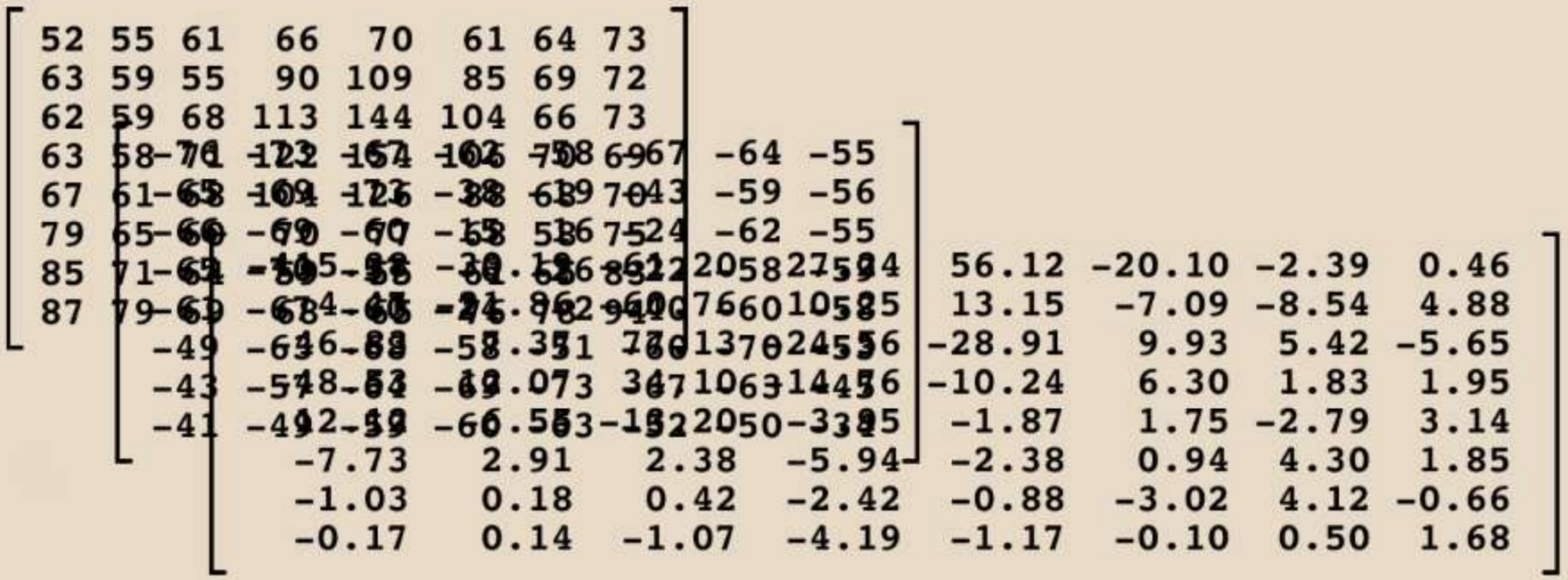
Image split into blocks

Reference: How JPG works

$$
\begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99
\end{bmatrix}
\begin{bmatrix}
-26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\
0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

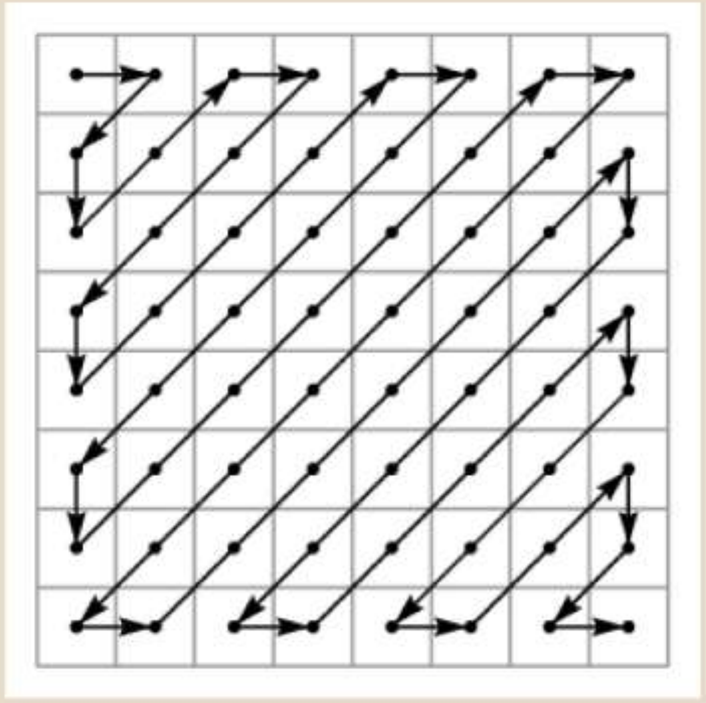
Quantisation

Reference: How JPG works



Forward Discrete Cosine Transform

Reference: How JPG works



$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

```
-26,-3,0,-3,-2,-6,2,-4,1,-3,1,1,5,
1,2,-1,1,-1,2,0,0,0,0,0,-1,-1,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Statistical encoding

Reference: How JPG works

Progressive JPGs

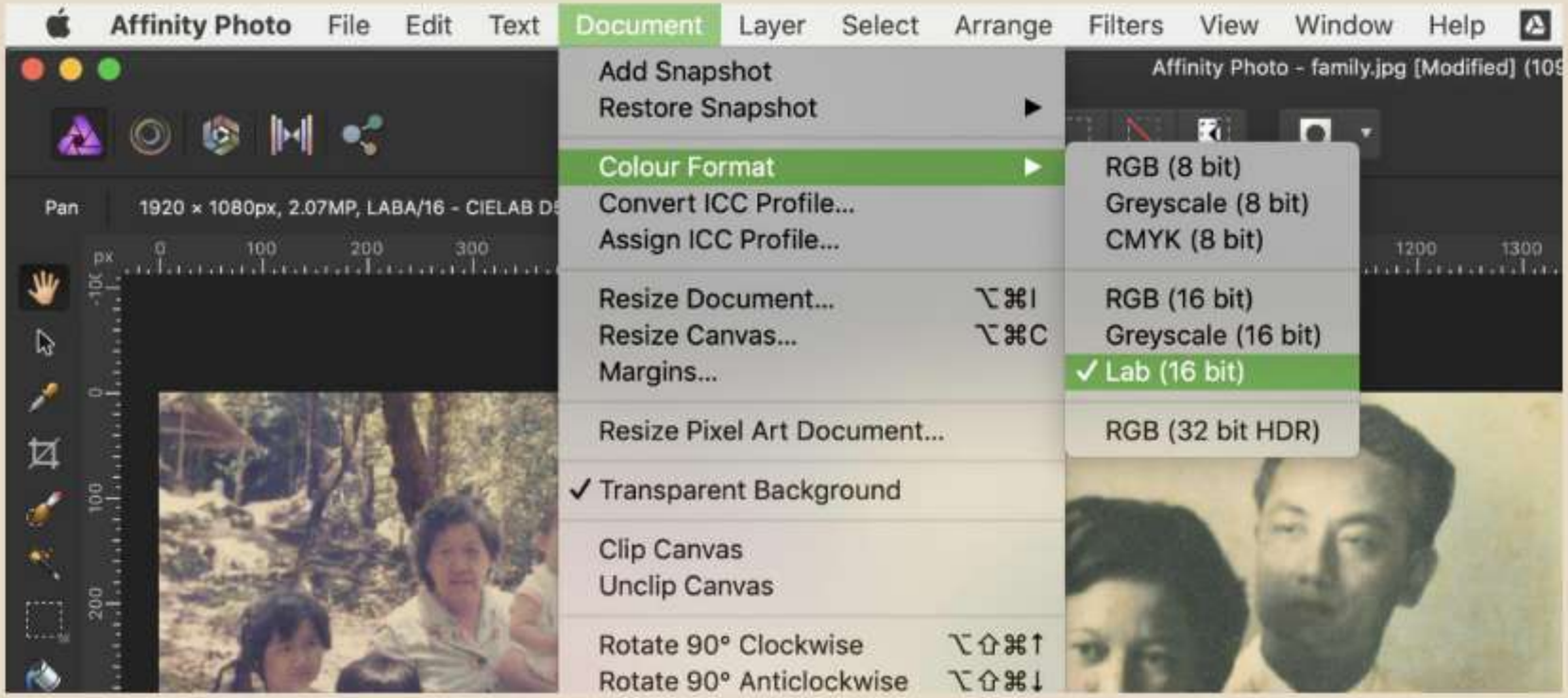
Image source: What is a progressive JPEG?

JPG optimisation tips

1. Use high quality source material
2. Alignment on the 8x8 pixel grid
3. Reduce contrast and saturation
4. Sepia images
5. Slight blurring

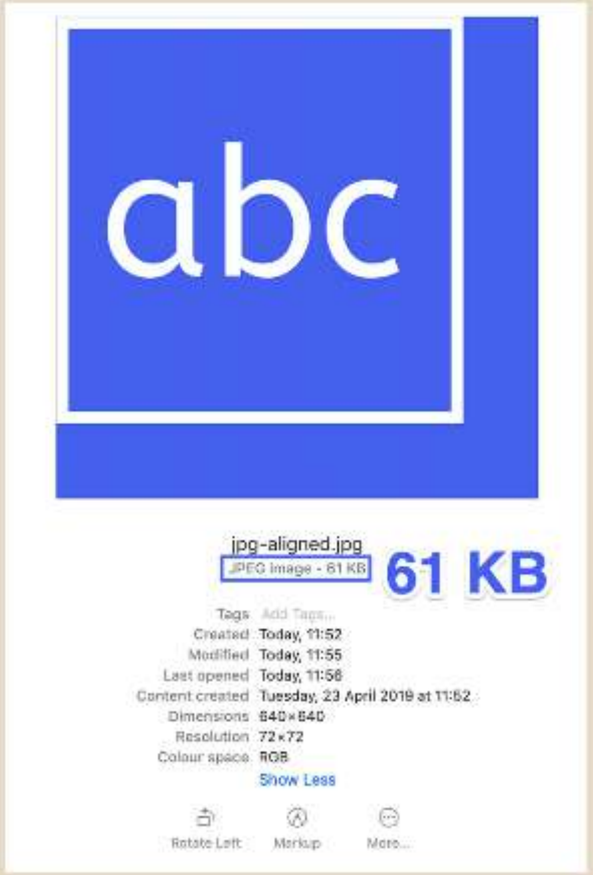
Reference: Finally understanding JPG

3. Reduce contrast and saturation



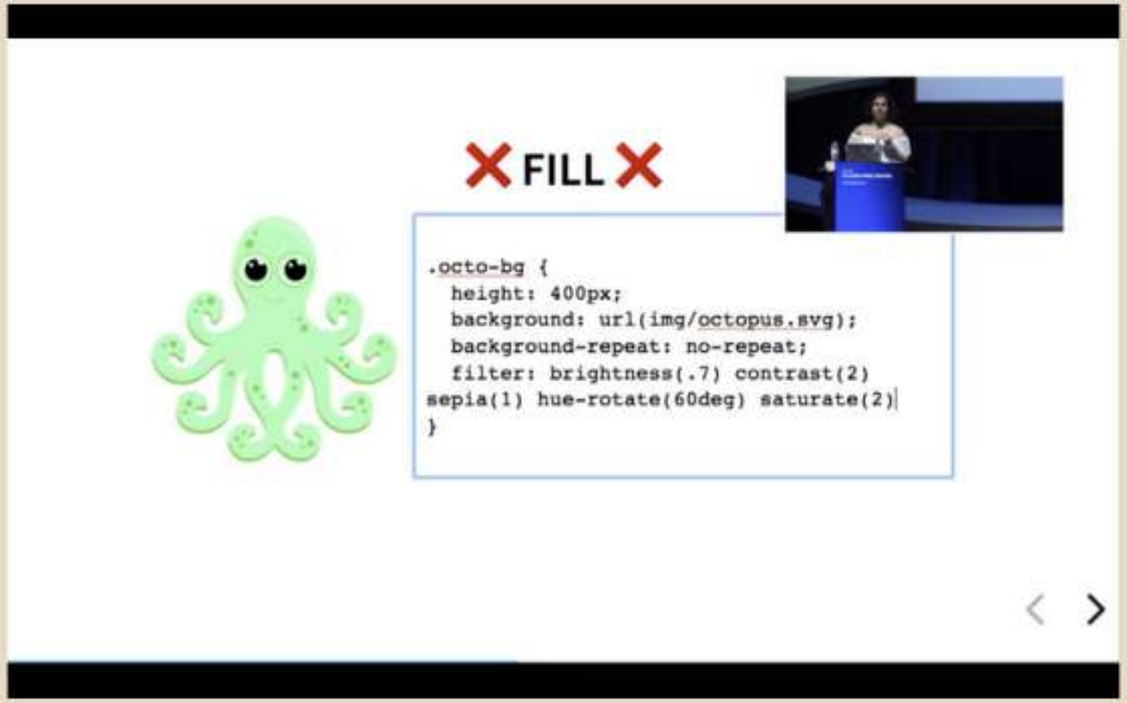
Reference: Finally understanding JPG

2. Alignment on the 8x8 pixel grid



Reference: Finally understanding JPG

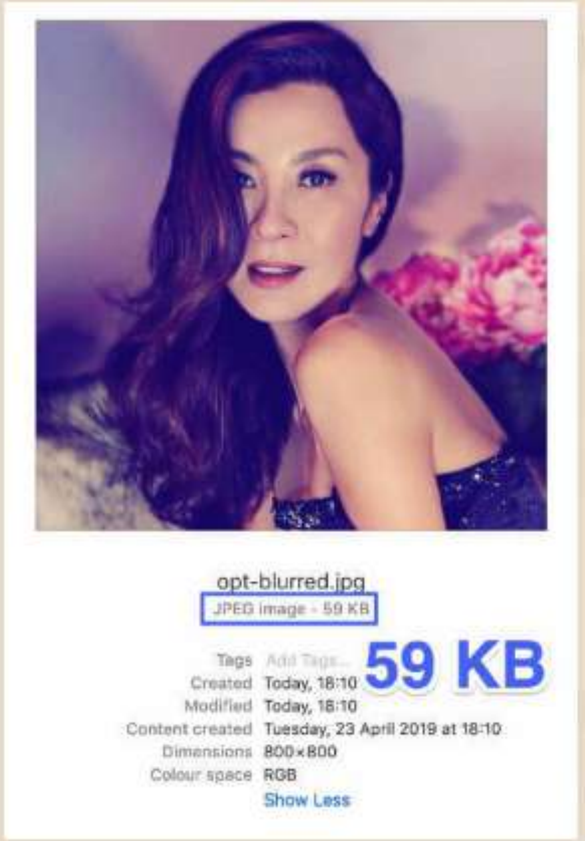
4. Sepia images



Una Kravets: CSS Blend Modes, Because···

Reference: Finally understanding JPG

5. Slight blurring



Reference: Finally understanding JPG

Speed, Quality, Size

Can't have 'em all

JPG encoders, there are many

libjpeg

mozJPEG

Guetzli

libjpeg-turbo

https://libjpeg-turbo.org/

GIF89a

Burn All GIFs day

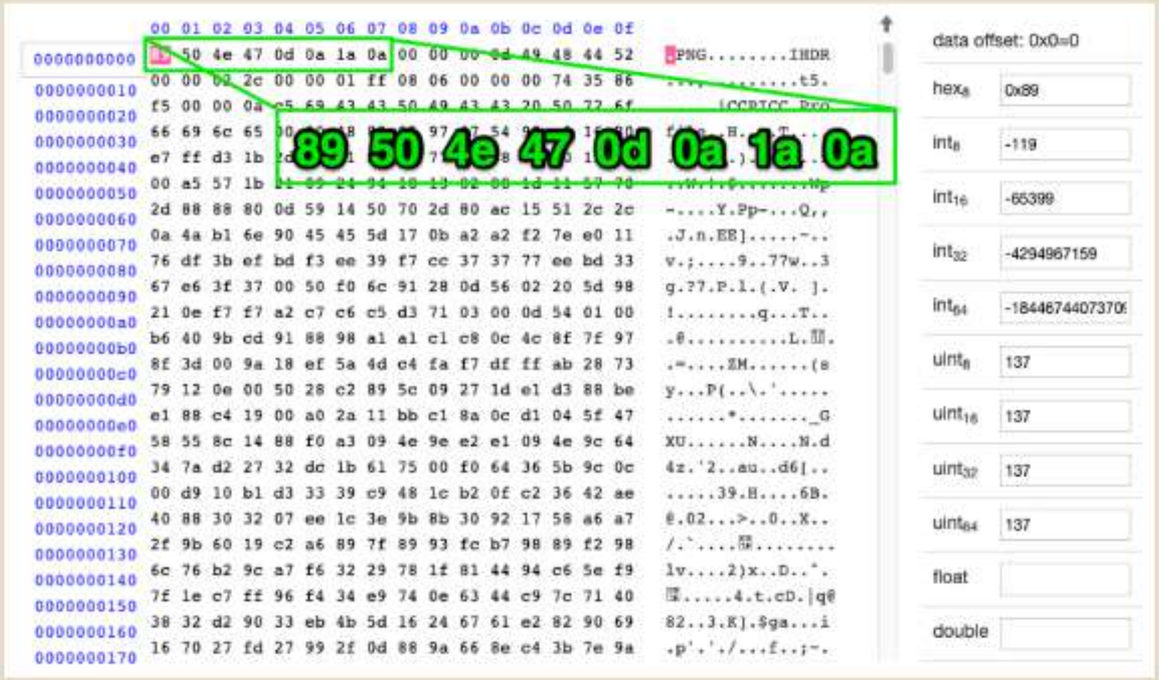
November 5, 1999

https://burnallgifs.org/archives/

PNG filter algorithms

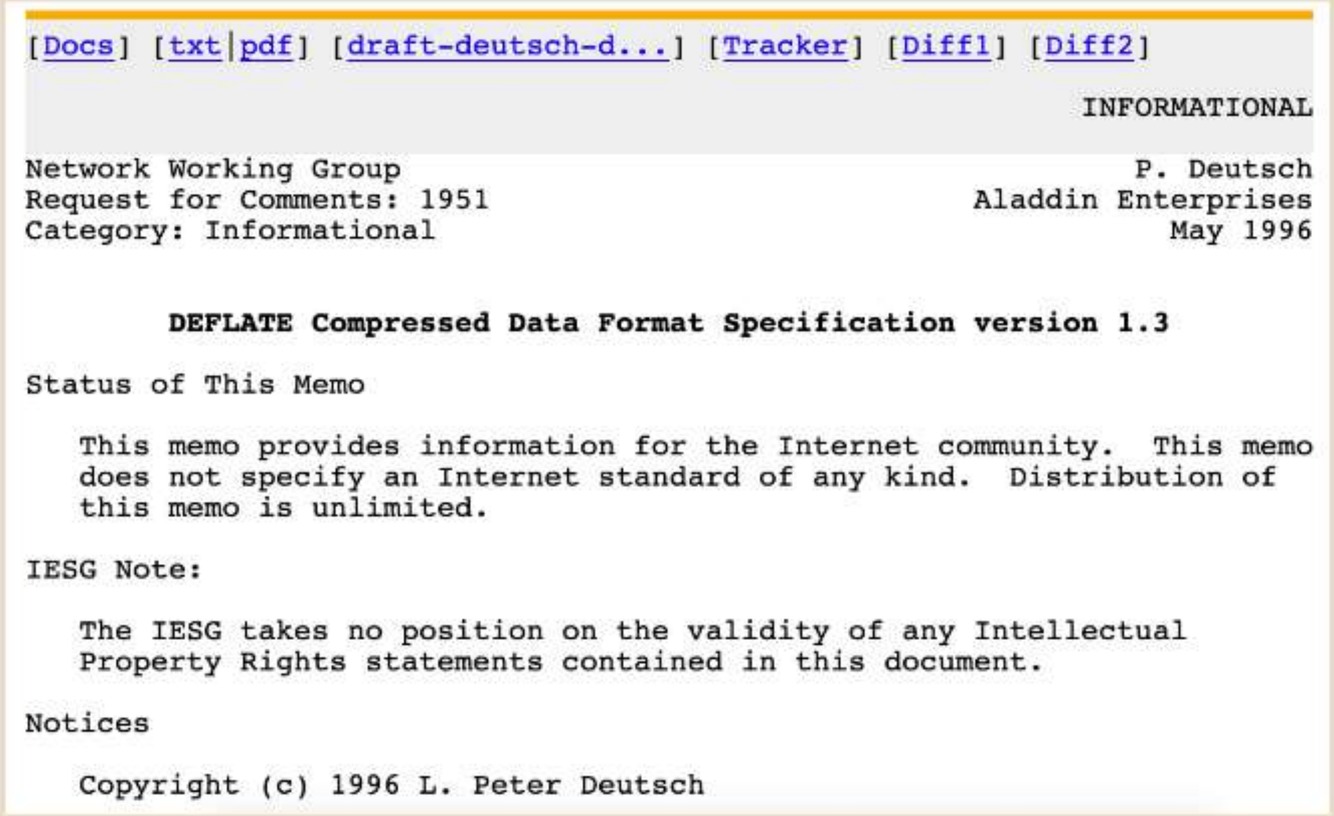
None	—	Zero	52 55 61 66 70 63 59 55 90 109 → 52 55 61 66 70 63 59 55 90 109
Sub	`Sub(x) = Raw(x) - Raw(x-bpp)`	Byte A (to the left)	52 55 61 66 70 63 59 55 90 109 → 52 55 61 66 70 63 -4 -4 35 19
Up	`Up(x) = Raw(x) - Prior(x)`	Byte B (above)	52 55 61 66 70 63 59 55 90 109 → 52 55 61 66 70 63 4 -6 24 39
Average	`Average(x) = Raw(x) - floor((Raw(x-bpp)+Prior(x))/2)`	Mean of bytes A and B, rounded down	52 55 61 66 70 63 59 55 90 109 → 52 55 61 66 70 63 0 -5 30 29
Paeth	`Paeth(x) = Raw(x) - PaethPredictor(Raw(x-bpp), Prior(x), Prior(x-bpp))`	A, B, or C, whichever is closest to `p = A + B - C`	52 55 61 66 70 63 59 55 90 109 → 52 55 61 66 70 63 -4 -4 35 19

Portable Network Graphics (PNG)



8-byte signature of a PNG file

DEFLATE compression algorithm



[Docs] [txt|pdf] [draft-deutsch-d...] [Tracker] [Diff1] [Diff2]

INFORMATIONAL

Network Working Group — P. Deutsch
Request for Comments: 1951 — Aladdin Enterprises
Category: Informational — May 1996

DEFLATE Compressed Data Format Specification version 1.3

Status of This Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

IESG Note:

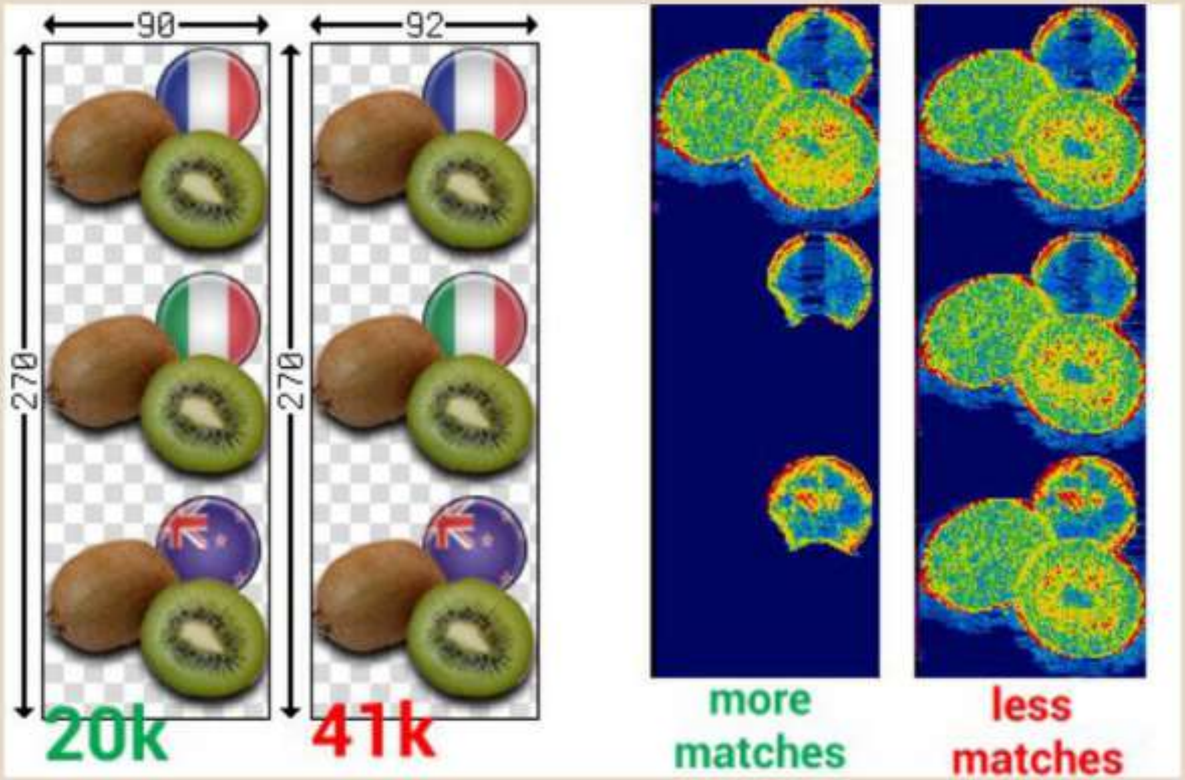
The IESG takes no position on the validity of any Intellectual Property Rights statements contained in this document.

Notices

Copyright (c) 1996 L. Peter Deutsch

DEFLATE Compressed Data Format Specification version 1.3

What a difference 2 pixels can make



Reference: pngthermal

Optimising PNG files

1. Reduce number of colours
2. Choose the right pixel format
3. Use indexed images, if possible
4. Optimise fully transparent pixels

Reference: Reducing PNG file Size

3. Use indexed images, if possible

Full colour PNG

Indexed colour PNG

Reference: Reducing PNG file Size

4. Optimise fully transparent pixels

Masked portion of image filled with single colour

Masked portion of image untouched

Reference: Reducing PNG file Size

libpng

libpng

libpng is the official PNG reference library. It supports almost all PNG features, is extensible, and has been extensively tested for over 23 years. The home site for development versions (i.e., may be buggy or subject to change or include experimental features) is https://libpng.sourceforge.io/, and the place to go for questions about the library is the **png-mng-implement** mailing list.

libpng is available as ANSI C (C89) source code and requires **zlib 1.0.4** or later (**1.2.5** or later recommended for performance and security reasons). The current public release, **libpng 1.6.36**, fixes some build issues, adds a couple of small optimizations (ARM `png_do_expand_palette()`, Intel SSE2 `memcpy()`), and updates the license (identical terms to the zlib license, with the old license appended in the manner of the Python Software Foundation License version 2, and the list of contributing authors moved to a separate AUTHORS file).

Portability Note

The libpng 1.5.x, 1.6.x, and upcoming 1.7.x series continue the evolution of the libpng API, finally **hiding the contents** of the venerable and hoary `png_struct` and `png_info` data structures inside **private (i.e., non-installed) header files**. Instead of direct struct-access, applications should be using the various `png_get_xxx()` and `png_set_xxx()` accessor functions, which have existed for almost as long as libpng itself.

The portability notice should not come as a particular surprise to anyone who has added libpng support to an application this millenium; the manual has warned of it since at least July 2000. (Specifically: *"Starting with version 2.0.0, both structures are going to be hidden, and the contents of the structures will only be accessible through the png_get/png_set functions."* OK, so the version number was off a bit...and the grammar, too, but who's counting?) Those whose apps depend on the older API need not panic, however (for now); libpng 1.2.x continues to get security fixes, as has 1.0.x for well over a decade. (Greg no longer bothers to list either series here; enough's enough, folks. Update those apps now!)

The 1.5.x and later series also include a new, more thorough test program (`pngvalid.c`) and a new `pnglibconf.h` header file that tracks what features were enabled or disabled when libpng was built. On the other hand, they no longer internally include the `zlib.h` header file, so applications that formerly depended on `png.h` to provide that will now need to include it explicitly. Complete differences relative to libpng 1.4.x are detailed here.

See the bottom of this page for warnings about **security and crash bugs** in versions up through **libpng 1.6.31**.

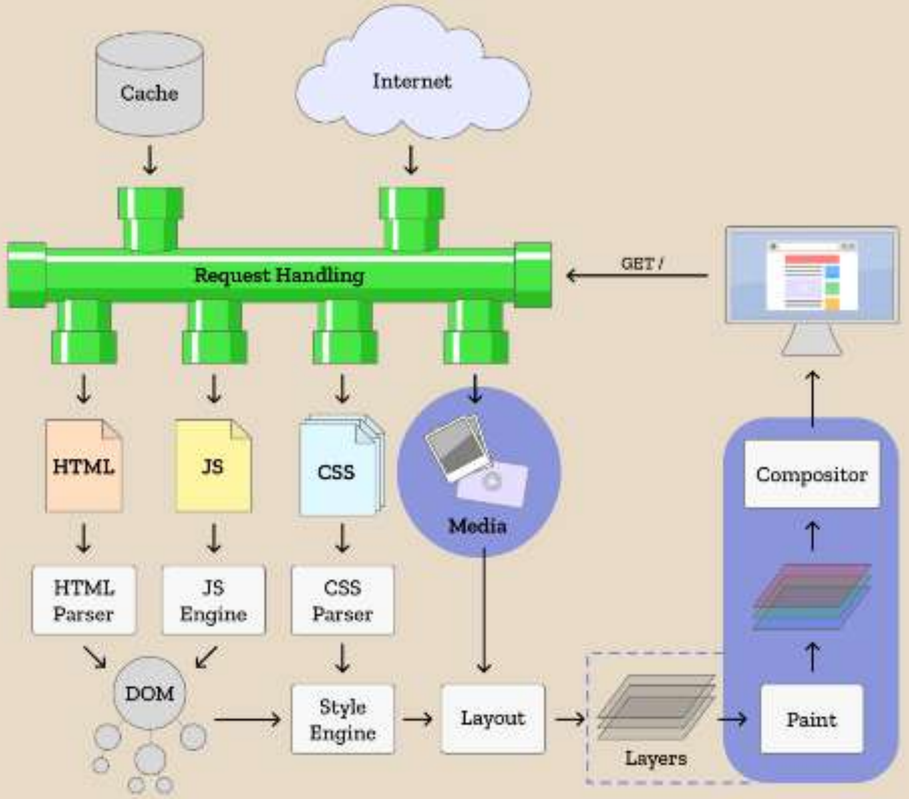
In addition to the main library sources, all of the 1.2.x/1.4.x/1.5.x/1.6.x/1.7.x series include the rpng, rpng2 and wpng demo programs, the pngminus demo program, a subset of Willem van Schaik's PngSuite test images, and Willem's VisualPng demo program.

http://www.libpng.org/pub/png/libpng.html

What is a browser engine? 🤔

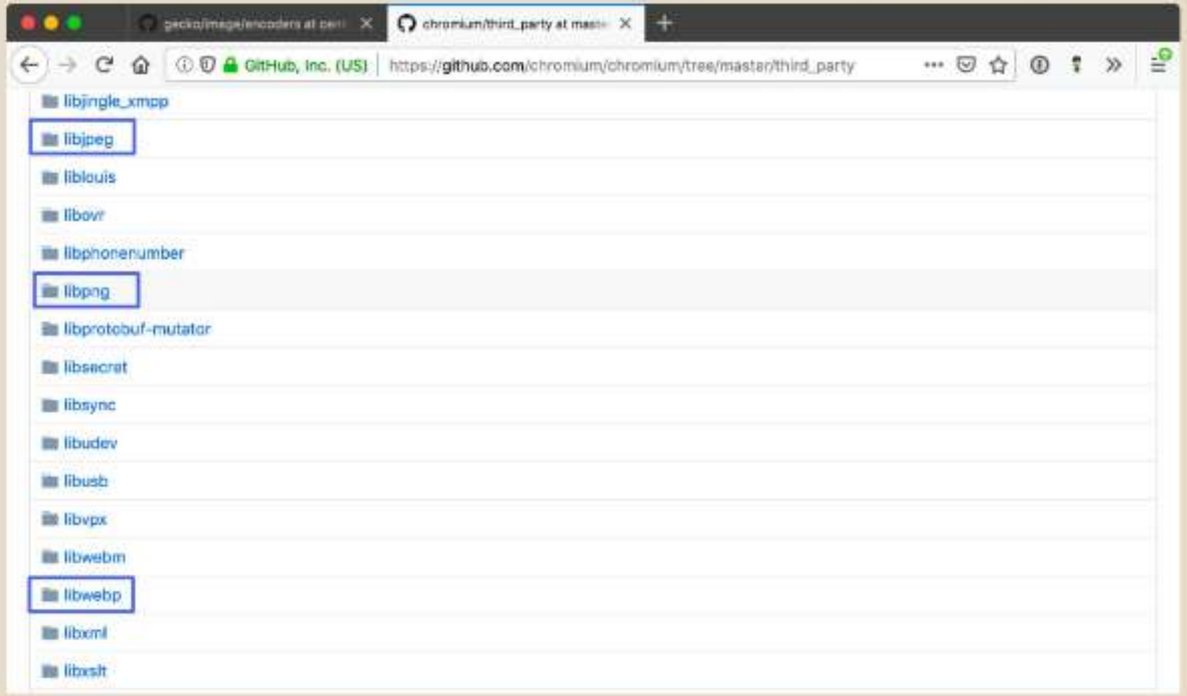
Source: Quantum Up Close: What is a browser engine?

What is a browser engine? 🤓

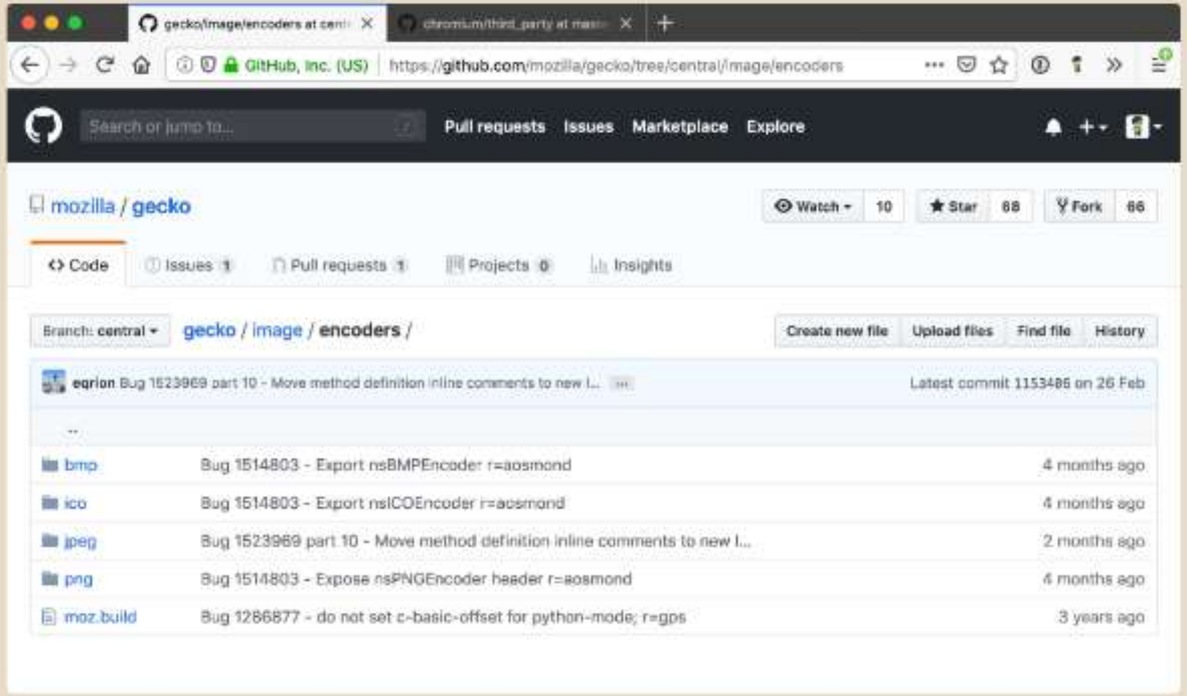


Source: Quantum Up Close: What is a browser engine?

Image encoders in browsers

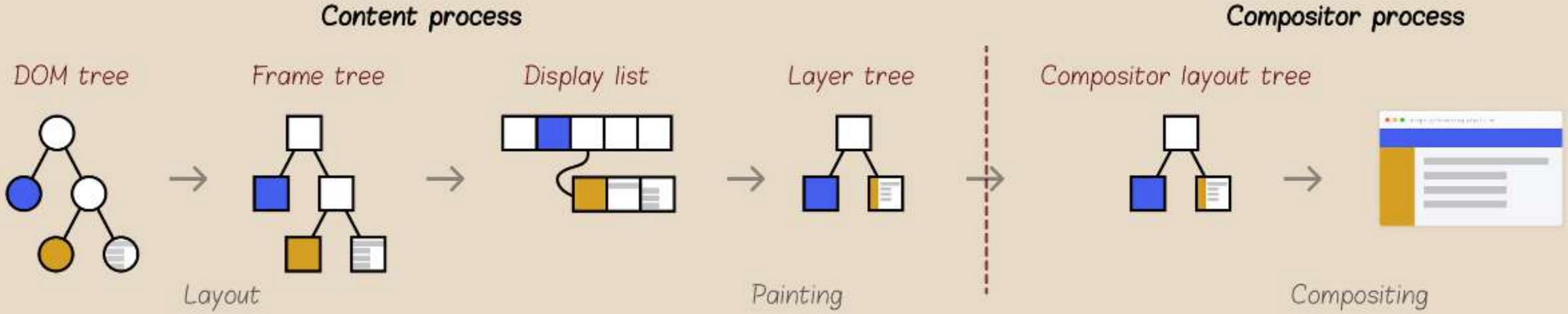


Chromium

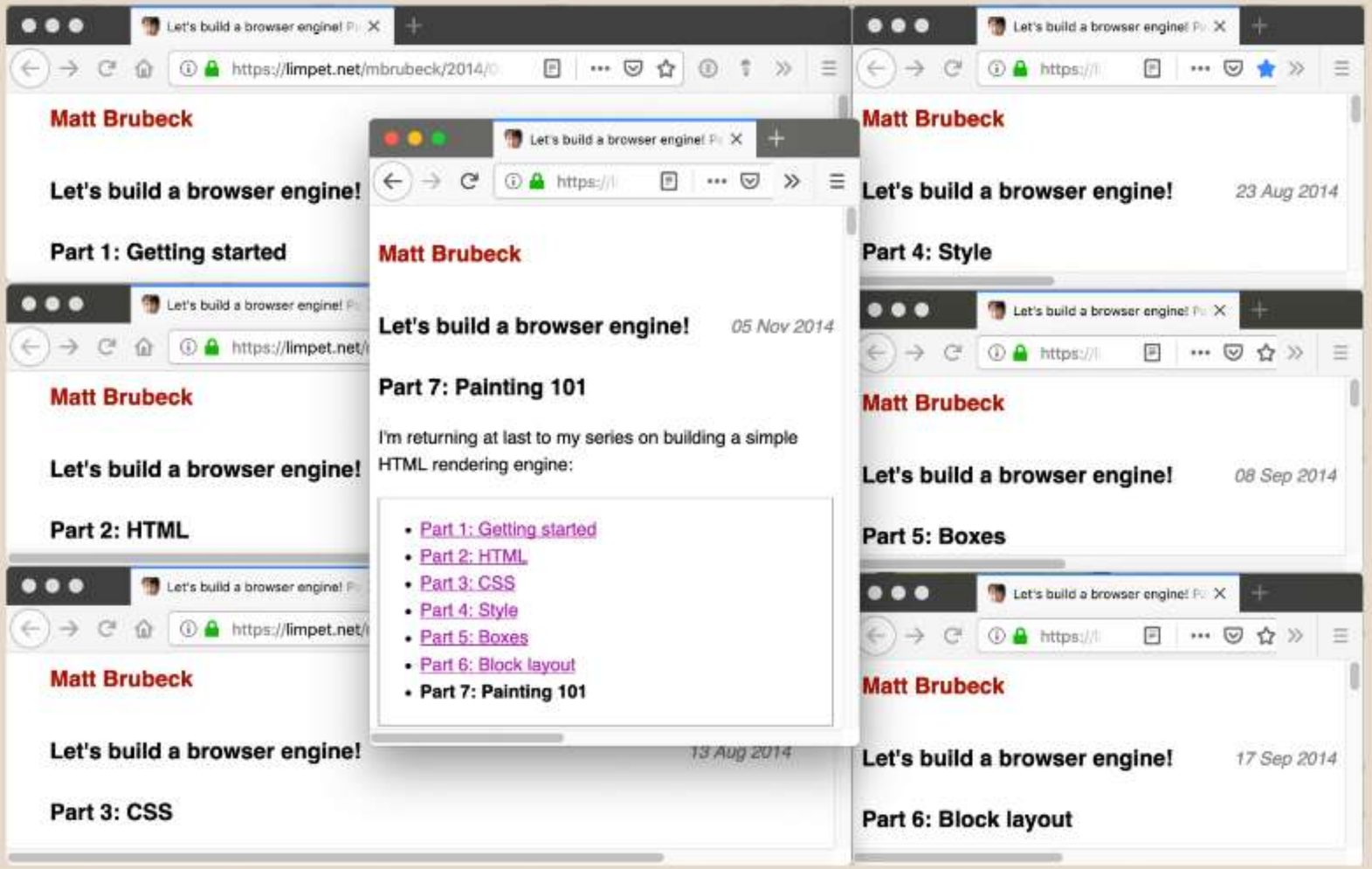


Gecko

Browser rendering pipeline



Reference: Introduction to WebRender – Part 1 – Browsers today



Reference: Let's build a browser engine!

Rasterisation (1/3)

Simple rasteriser for painting rectangles

```
pub struct Canvas {
    pub pixels: Vec<Color>,
    pub width: usize,
    pub height: usize,
}

impl Canvas {
    /// Create a blank canvas
    fn new(width: usize, height: usize) -> Canvas {
        let white = Color { r: 255, g: 255, b: 255, a: 255 };
        Canvas {
            pixels: vec![white; width * height],
            width: width,
            height: height,
        }
    }
  // …
}
```

Source: Let's build a browser engine!

Rasterisation (2/3)

Simple rasteriser for painting rectangles

```
fn paint_item(&mut self, item: &DisplayCommand) {
    match *item {
        DisplayCommand::SolidColor(color, rect) => {
            // Clip the rectangle to the canvas boundaries.
            let x0 = rect.x.clamp(0.0, self.width as f32) as usize;
            let y0 = rect.y.clamp(0.0, self.height as f32) as usize;
            let x1 = (rect.x + rect.width).clamp(0.0, self.width as f32) as usize;
            let y1 = (rect.y + rect.height).clamp(0.0, self.height as f32) as usize;

            for y in y0 .. y1 {
                for x in x0 .. x1 {
                    self.pixels[y * self.width + x] = color;
                }
            }
        }
    }
}
```

Source: Let's build a browser engine!

Rasterisation (3/3)

Simple rasteriser for painting rectangles

```
/// Paint a tree of LayoutBoxes to an array of pixels.
pub fn paint(layout_root: &LayoutBox, bounds: Rect) -> Canvas {
    let display_list = build_display_list(layout_root);
    let mut canvas = Canvas::new(bounds.width as usize, bounds.height as usize);
    for item in display_list {
        canvas.paint_item(&item);
    }
    canvas
}
```

Source: Let's build a browser engine!

> *The graphics backend is an array of bytes and a for loop that goes and says: go from the left edge of the rectangle to the right edge and write the same colour over and over again into this array.*
>
> *—Matt Brubeck, Bay Area Rust Meetup (Nov 2014)*

¯_(ツ)_/¯

Graphics libraries

Skia

WebRender

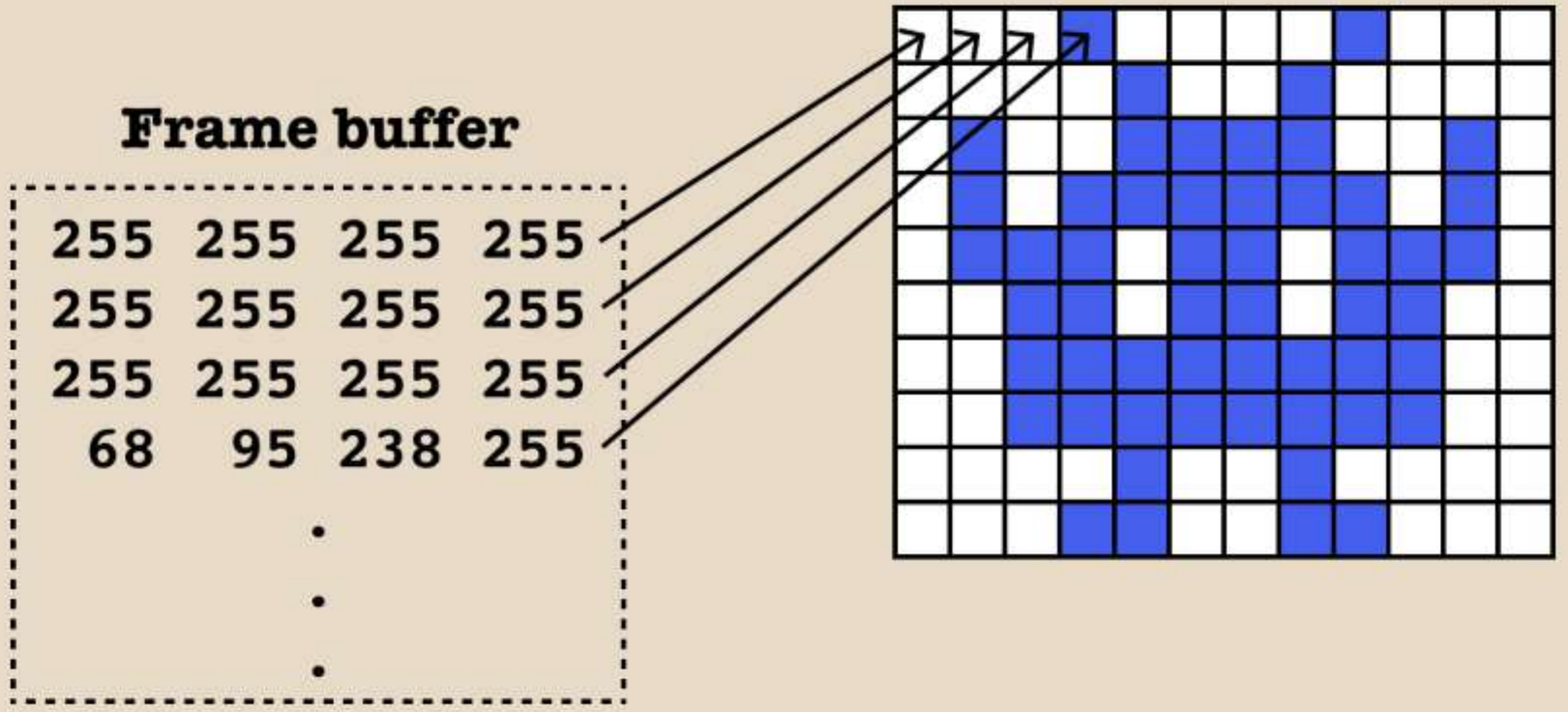
Core Graphics

Cairo

Παν語

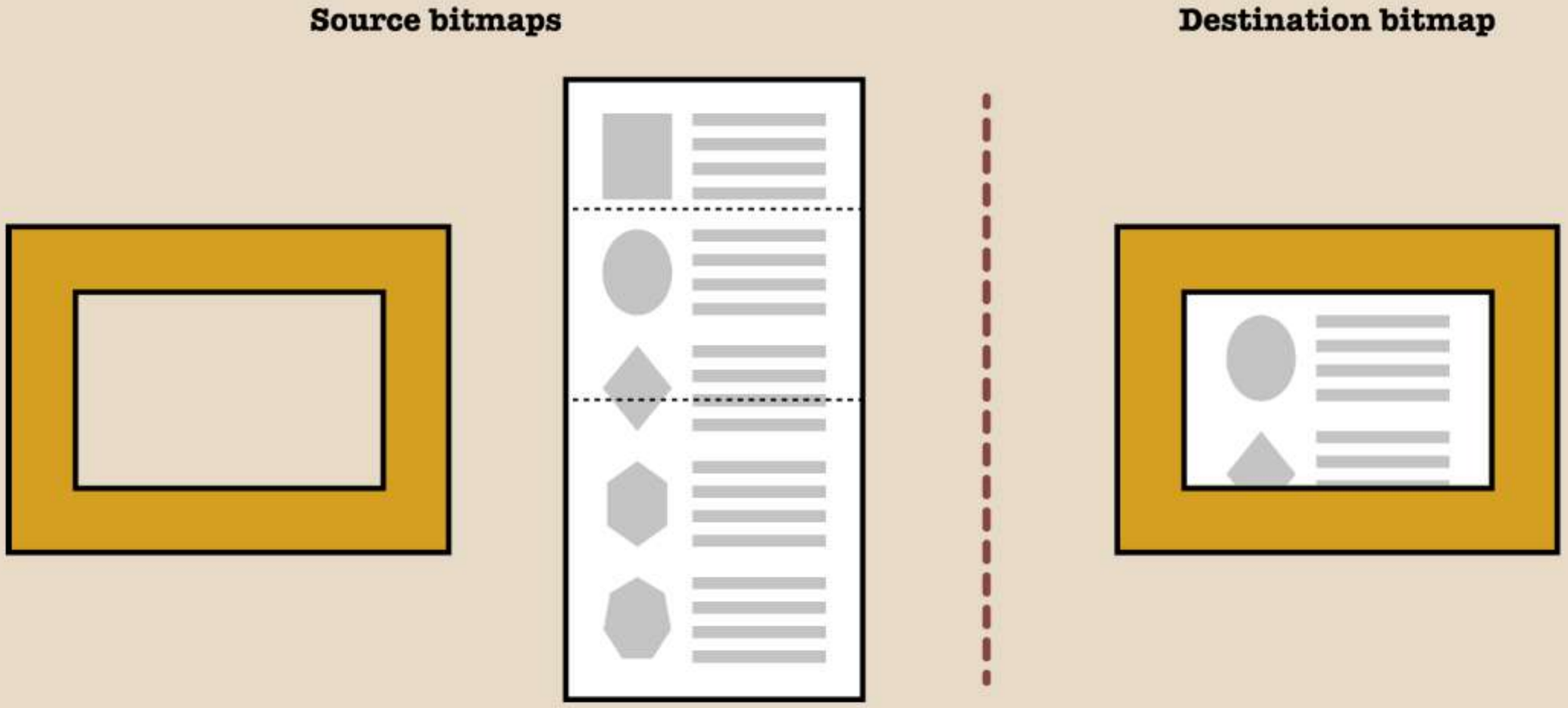
Pango

Painting



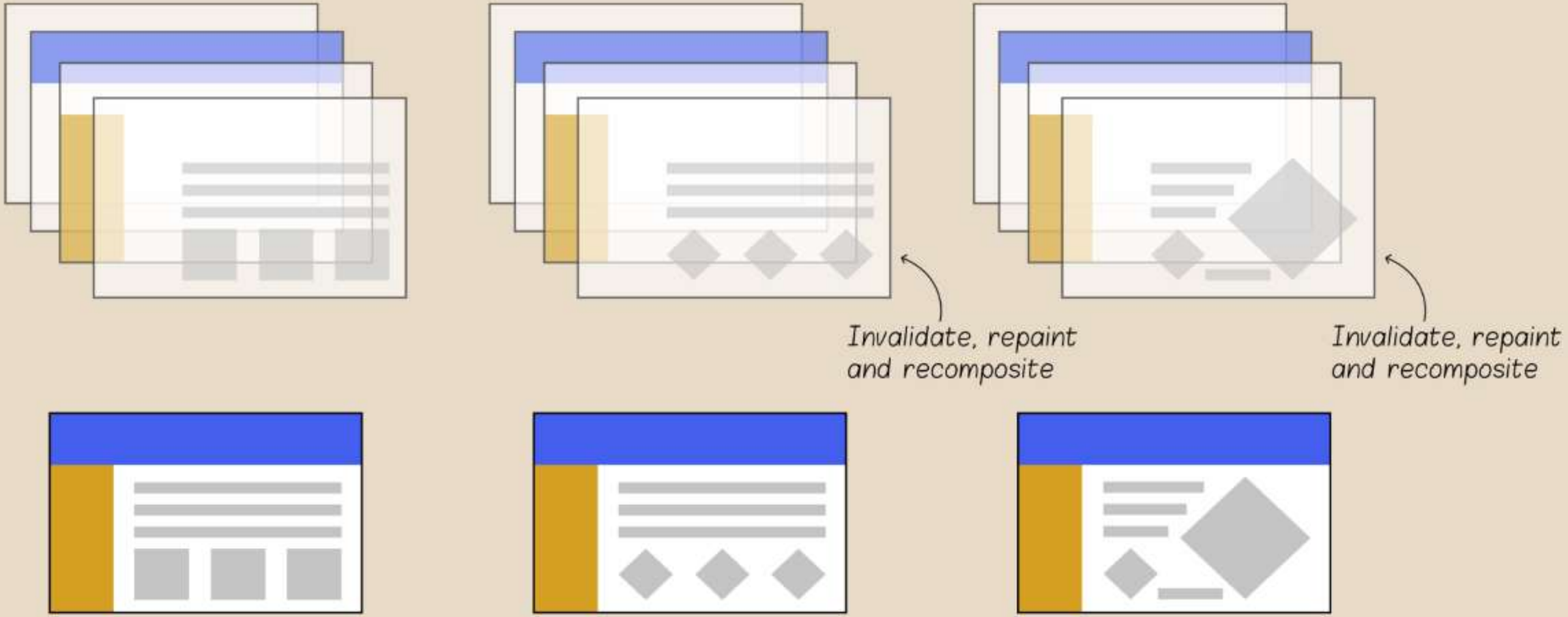
Reference: The whole web at maximum FPS: How WebRender gets rid of jank

Compositing (1/2)



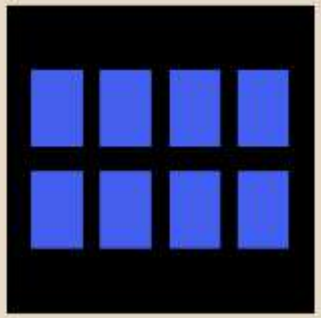
Reference: The whole web at maximum FPS: How WebRender gets rid of jank

Compositing (2/2)

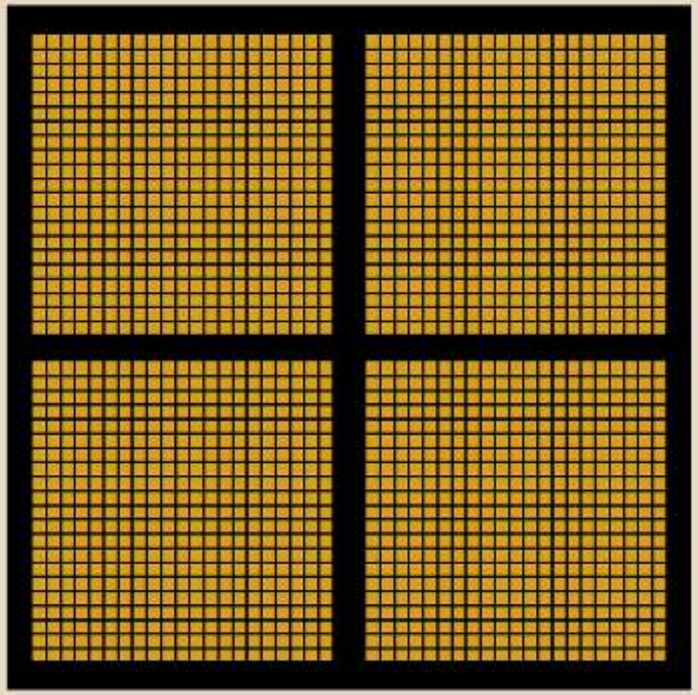


Reference: GPU Accelerated Compositing in Chrome

Making use of the GPU (1/2)



CPU
(multiple cores)

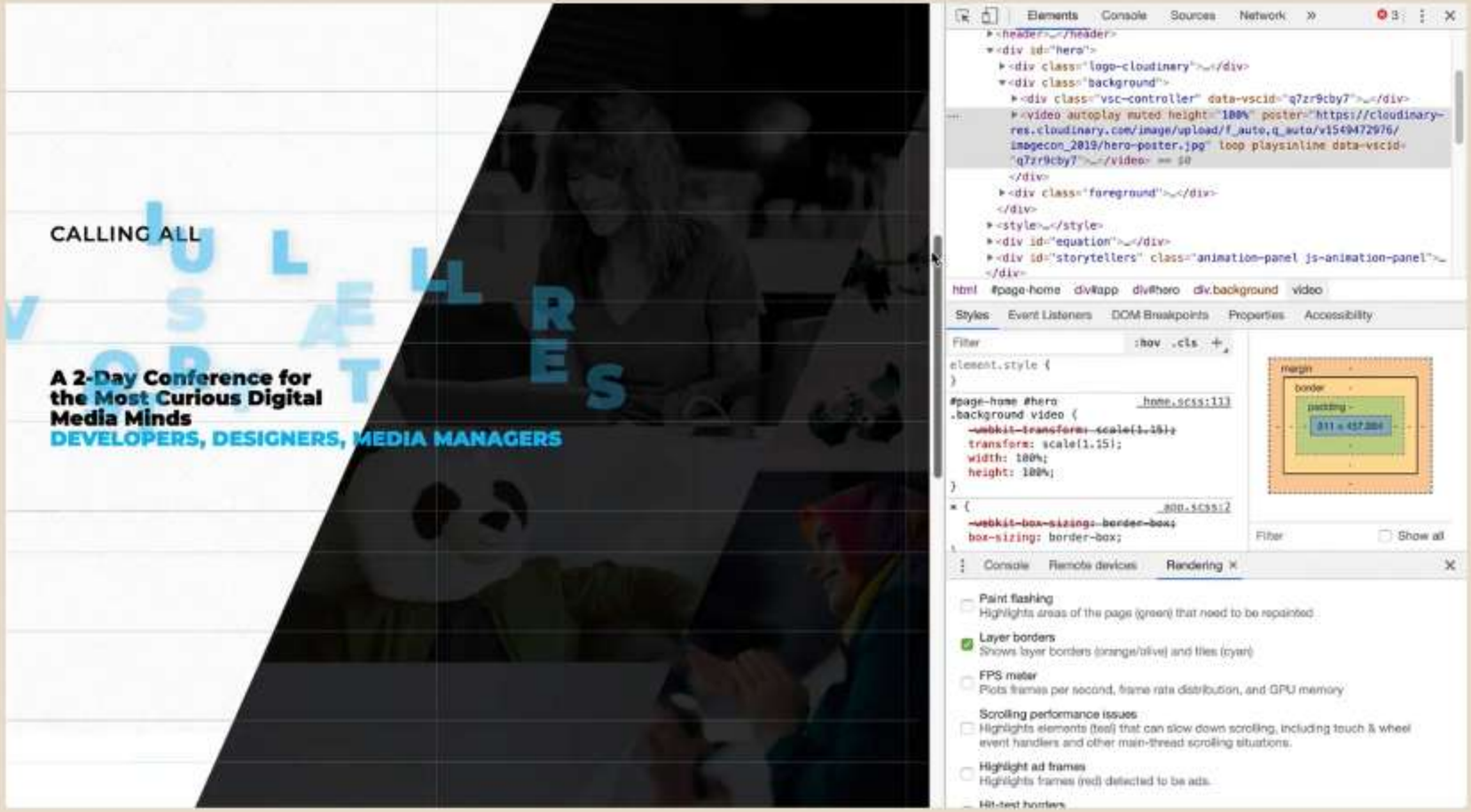


GPU
(thousands of cores)

Making use of the GPU (2/2)

Reference: Hardware acceleration and compositing

GPU rasterisation in Chromium



Reference: Software vs. GPU rasterization in Chromium

If you could have a do-over...

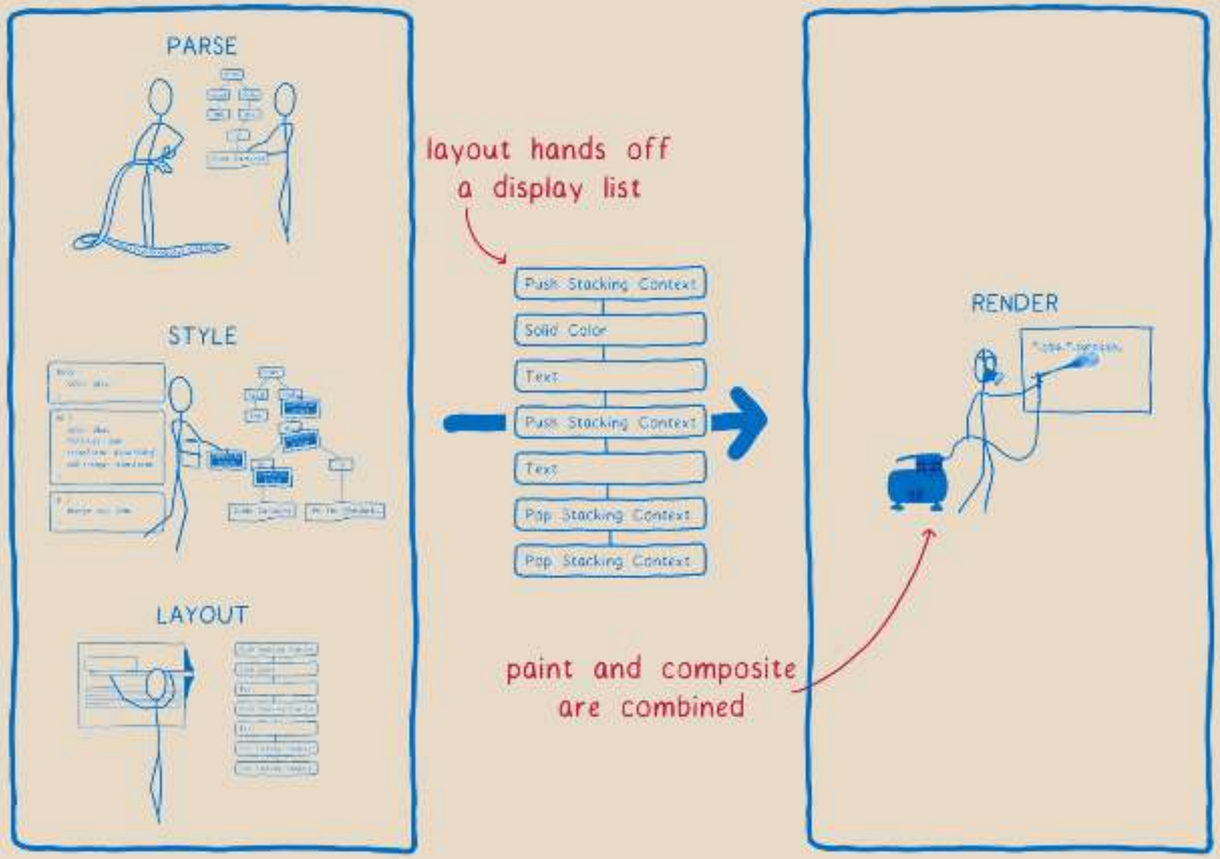






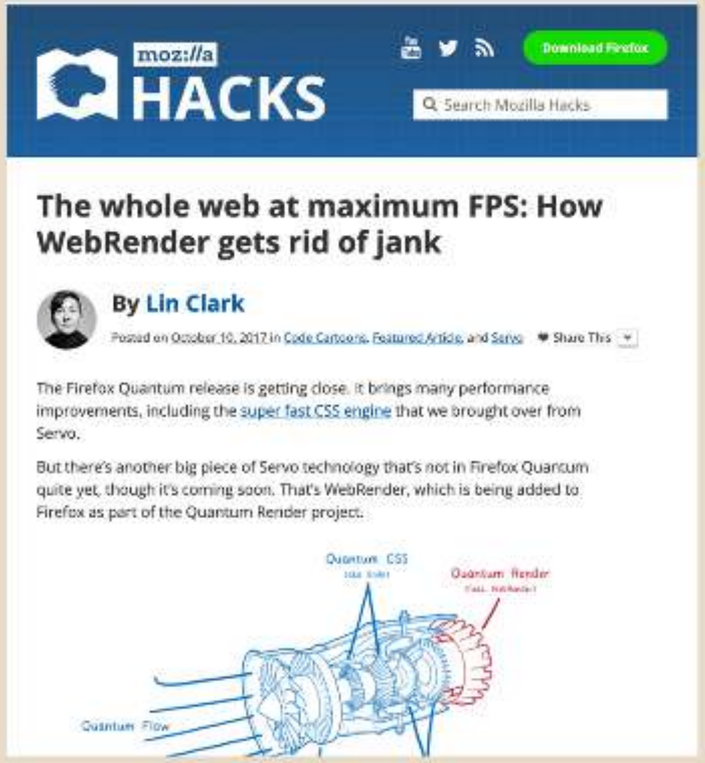
What if we actually need a butterfly instead?

WebRender (1/2)

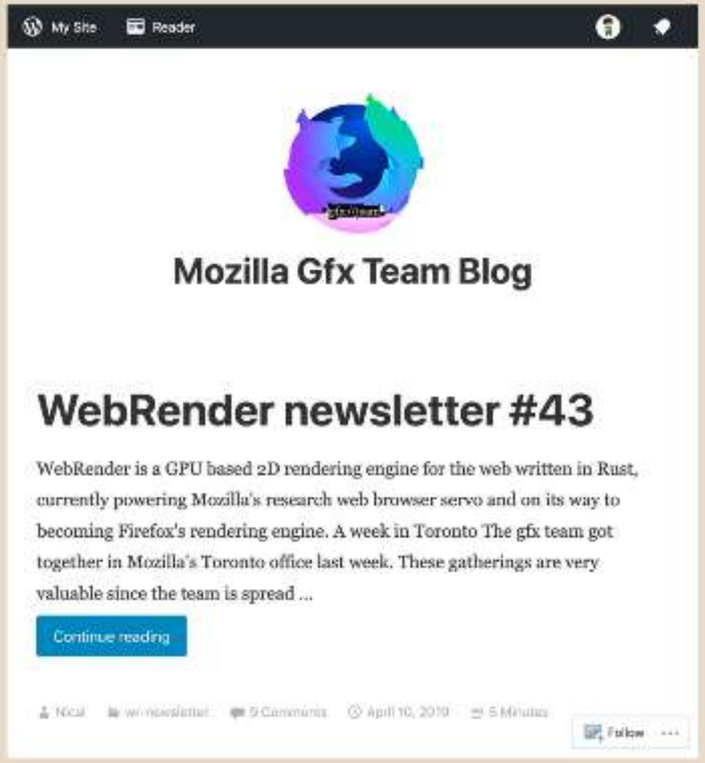


The whole web at maximum FPS: How WebRender gets rid of jank by Lin Clark

WebRender (2/2)



https://hacks.mozilla.org/2017/10/the-whole-web-at-maximum-fps-how-webrender-gets-rid-of-jank/



https://mozillagfx.wordpress.com/

Acknowledgements

🙏 Big thank you to these beautiful human beings who answered my noob questions 🙏


@bmeurer


@callahad


@slsoftworks


@linclark


@g33konaut


@mathias


@nicalsilva


@potch

References

- Davis, W. (1986). *The Origins of Image Making*. Current Anthropology, 27(3), 193-215. doi:10.1086/203422
- The GIF Is Dead. Long Live the GIF.
- Types of Bitmaps
- Why do we need JPG compression and how it's technically working? by Steven Hansen
- Progressive JPEGs and green Martians by Jon Sneyers
- Finally understanding JPG by Christoph Erdmann
- How JPG Works, How PNG Works, Reducing PNG file Size by Colt McAnlis
- Thoughts on a GIF-replacement file format
- Quantum Up Close: What is a browser engine? by Matt "Potch" Claypotch
- Let's build a browser engine! by Matt Brubeck
- On rendering engines and graphic libraries by Kilian Valkhof
- Following up on the 2d graphics in Rust discussion by Nicolas Silva
- Introduction to WebRender – Part 1 – Browsers today by Nicolas Silva
- The whole web at maximum FPS: How WebRender gets rid of jank by Lin Clark
- Software vs. GPU rasterization in Chromium* by Martina Kollarova
- GPU Accelerated Compositing in Chrome by Tom Wiltzius, Vangelis Kokkevis & the Chrome Graphics team

This talk is dedicated to browser engineers everywhere.

I owe my career to you.

Thank you!

 https://www.chenhuijing.com

 @hj_chen

 @hj_chen

 @huijing

Font used is Signika by Anna Giedryś