

Live-Pixel für einen Röhrenfernseher

Wie eine Demoszene-Show für einen PAL-Röhrenfernseher auf einem Raspberry Pi 1 von 2012 läuft, und wie eine KI sie gebaut hat, ohne den Bildschirm je zu sehen.

bitcircuit101, assisted by Claude

2026

Kurz gesagt

- Der alte Röhrenfernseher im bitcircuit101 zeigt eine Demoszene-Show, live berechnet auf einem Raspberry Pi 1 von 2012: 17 Szenen und zwei Folien-Decks, 50 Bilder pro Sekunde auf einem Kern mit 700 MHz, ohne Hilfe einer GPU, ohne eine einzige Bilddatei.
- Das Schwierige waren nicht die Effekte, sondern der Videoweg: PAL statt NTSC, 288 progressive Zeilen statt Zeilensprung-Flackern, und ein Overscan, der mit einer Testkarte vermessen wurde.
- Das Ganze ist ein C-Programm: ein Canvas von 360×288 Pixeln in 16 Bit Farbe, doppelt gepuffert in den Linux-Framebuffer, jedes Pixel auf dem Weg verdoppelt, ein Sequencer mit Überblendungen, und Szenen, die Dateien mit zwei Funktionen sind, `init()` und `frame(t)`.
- Gemessen, nicht geraten: Der Sequencer protokolliert auf dem Pi die Bildrate und das langsamste Bild jeder Szene, ein Bench-Werkzeug tut dasselbe offline.
- Die Show hat eine KI per SSH geschrieben, ohne den Bildschirm je zu sehen. Die Werkzeuge, die das möglich machten, ein Offscreen-Renderer, Bewegungsmessungen, ein eigenständiges Flipbook und ein Deploy, das prüft, ob die Show läuft, sind der Teil, der sich zum Nachmachen lohnt.
- Derselbe C-Code läuft im Browser, übersetzt nach WebAssembly, auf der Website des Projekts.
- Mitglieder steuern die Show per SSH: `ssh zapper` ist eine Fernbedienung im Terminal, jede Szene hat eine Programmnummer, und Szenen in C lassen sich hochladen, werden gebaut, offscreen durchgespielt und mit Rollback eingetauscht. Die Show läuft in einer Sandbox, als Nutzer, der sonst nichts darf.

1. Ein Röhrenfernseher und ein Pi von 2012

bitcircuit101 ist ein Technik- und Kreativraum in Bonn. Irgendwo darin steht ein alter Röhrenfernseher, ein Toshiba C-500, ein 5-Zoll-Kofferfernseher aus den frühen 1980ern mit Teleskopantenne und Senderskala auf der Front, daneben ein Raspberry Pi 1 Model B, das Board von 2012 mit einem ARM1176-Kern bei 700 MHz und 512 MB RAM. Ein Composite-Kabel verbindet die beiden. Der Fernseher ist kein Monitor, er ist ein Fernseher, mit allem, was dazugehört: PAL, 50 Halbbilder pro Sekunde, ein Overscan, der die Ränder verschluckt, und eine leichte Weichheit, die viel verzeiht.

Die Idee war einfach und ein bisschen stur: Der Fernseher soll etwas Lebendiges zeigen, und alles, was er zeigt, soll aus Code kommen. Keine Videodatei, keine Diaschau aus PNGs, kein Browser-Kiosk. Ein C-Programm berechnet jedes Pixel in Echtzeit und schreibt es in den Framebuffer, so wie es eine Demo auf einem Heimcomputer der 90er tat. Siebzehn Szenen laufen in Schleife: Plasma, ein Tunnel, Feuer, ein jonglierender Clown, Flower Power, ein Synthwave-Horizont, eine Bildröhre in 3D, die sich selbst erklärt, Bernd das Brot, das wie das DVD-Logo durchs Bild springt. Zwei Folien-Decks erklären, wie alles funktioniert, mit Live-Demos am unteren Rand jeder Folie.

Der sture Teil ist die Regel, dass nichts ein Bild ist. Das Flipbook auf der Website, die Abbildungen in diesem Text, die Show im Browser: Alles wird beim Bauen aus demselben Quellcode gerendert. Ändert sich eine Szene, ändert sich jedes Bild von ihr mit.

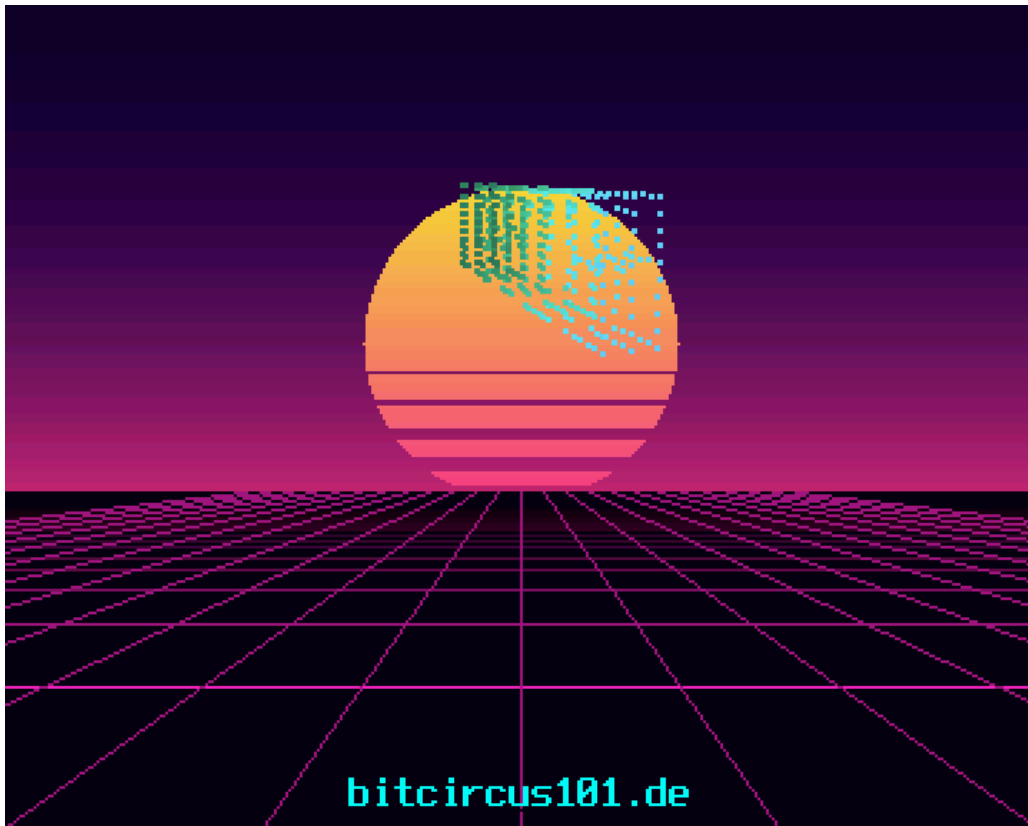


Abbildung 1: Die Show im Browser: derselbe C-Code, übersetzt nach WebAssembly, mit 50 Schritten pro Sekunde wie auf dem Pi.

2. Der Videoweg: Composite, PAL, 288 Zeilen

Erst Farbe

Das erste Bild war schwarzweiß. Der Composite-Ausgang des Pi sendet standardmäßig NTSC, der Fernseher erwartet PAL, und ein PAL-Gerät, das NTSC-Farbe bekommt, zeigt klaglos ein sauberes graues Bild. Ein Kernel-Parameter hat es gelöst, angehängt an die eine Zeile in `/boot/firmware/cmdline.txt`:

```
vc4.tv_norm=PAL
```

Flackern

Das zweite Bild flackerte. Ein PAL-Fernseher zeichnet 576 Zeilen als zwei verschränkte Halbbilder, 288 ungerade Zeilen, dann 288 gerade, 50 Halbbilder pro Sekunde. Feiner Text, der in nur einem Halbbild liegt, flackert mit 25 Hz, und genau solchen Text zeigt eine Konsole. Die Lösung ist, das zweite Halbbild aufzugeben: Der Pi steuert den Fernseher mit 288 progressiven Zeilen bei 50 Hz an, jedes Bild vollständig, kein Flackern, und ein Bild, das immer noch so scharf ist, wie eine Röhre es zeigen kann.

```
video=Composite-1:720x288@50
```

Die Ränder

Eine Röhre versteckt ihre Ränder hinter der Blende, und keine zwei Geräte verstecken gleich viel. Die Szene `testcard` zeichnet nummerierte Linien an jedem Rand: Links und rechts zählen die Zahlen Framebuffer-Pixel, oben und unten Zeilen. Jemand stellt sich vor den Fernseher, liest an jedem Rand die erste ganz sichtbare Zahl ab, und diese vier Zahlen werden die Ränder:

```
margin_left=56,margin_right=40,margin_top=16,margin_bottom=20
```

Die GPU skaliert dann das ganze Bild, die Linux-Konsole eingeschlossen, in den sichtbaren Bereich. Das Programm muss vom Overscan gar nichts wissen.



Abbildung 2: Die Testkarte: nummerierte Linien an jedem Rand, von einem Menschen an der Röhre abgelesen.

In den Framebuffer

Es gibt keinen X-Server und kein Fenster. Eine systemd-Unit startet das Programm beim Booten auf tty1. Das Programm öffnet `/dev/fb0`, prüft, dass der Framebuffer wirklich 720×288 bei 16 Bit pro Pixel ist, und verweigert sonst den Start; dann blendet es ihn in den Speicher ein und schaltet die Konsole mit `KDSETPMODE` in den Grafikmodus. Beim Beenden löscht es den Bildschirm und stellt den Textmodus wieder her, die Konsole ist also zurück, wenn die Show endet.

3. Architektur: Canvas, Puffer, Sequencer

Eine Szene sind zwei Funktionen

```
typedef struct {
    const char *name;
    float dur;
    void (*init)(void);
    void (*frame)(float t, float dt);
    float (*length)(void); // optional, ersetzt dur nach init
} Scene;
```

Das ist die ganze Schnittstelle. `init()` läuft einmal beim Start der Szene und rechnet vor, was teuer ist. `frame(t, dt)` malt das Bild für die Szenenzeit `t` ins Canvas. Eine Szene ist eine Datei in `src/scenes/`, und eine Tabelle in `src/scenes.c` führt die siebzehn Szenen der Schleife in Spielreihenfolge auf, dazu die Extras, die nur auf Abruf laufen: die zwei Folien-Decks und die Testkarte. Die Schleife dauert knapp drei Minuten, 176 Sekunden, dann beginnt sie von vorn.

Fünfzigmal pro Sekunde

Der Sequencer in `main.c` ist eine Schleife mit vier Schritten: Die Szene malt ins Canvas, die Blende skaliert das Bild am Anfang und Ende einer Szene, der Blit kopiert das Canvas in den Framebuffer, und ein Schlaf wartet auf den nächsten 20-ms-Takt. Der Takt ist ein absoluter Zeitpunkt, keine relative Pause: Das Programm berechnet, wann das nächste Bild fällig ist, und schläft mit `clock_nanosleep` und `TIMER_ABSTIME` bis dahin, ein zu früh fertiges Bild verschiebt den Takt also nicht. Auf ein zu spätes Bild folgt das nächste sofort, und erst wenn das Programm mehr als

100 ms in Rückstand gerät, gibt es auf und setzt den Takt neu auf jetzt. Der Zeitschritt, den eine Szene bekommt, ist auf 50 ms gedeckelt, ein Schluckauf lässt also nie ein Teilchen quer über den Bildschirm springen.

Jede Szene bekommt am Ende eine Zeile in `/run/bc101/anim.log`: ihre mittlere Bildrate und ihr langsamstes Bild in Millisekunden. Diese zwei Zahlen sind die ganze Performance-Geschichte der Show, und Kapitel 5 handelt von ihnen.

Ein Pixel wird zwei

Das Canvas ist 360×288 Pixel groß, halb so breit wie der Framebuffer. Auf dem Weg hinaus wird jedes Pixel zweimal geschrieben:

```
dst[x] = src[x] | ((uint32_t)src[x] << 16);
```

Zwei 16-Bit-Pixel passen in ein 32-Bit-Wort, die Verdopplung kostet also einen Schreibzugriff pro Paar, und die Szenen haben halb so viel Arbeit. Die verdoppelten Pixel sind auf einer 4:3-Röhre fast quadratisch, was das Auge erwartet.

Die Szene malt nie direkt in den Framebuffer. Sie malt ins Canvas, und nur ein fertiges Bild wird kopiert, in einem Durchgang von oben nach unten. Das ist Double Buffering, und deshalb erreicht nie ein halbfertiges Bild die Röhre.

Sechzehn Bit Farbe

Ein Pixel ist ein `uint16_t` in RGB565: fünf Bit Rot, sechs Bit Grün, fünf Bit Blau, 65.536 Farben. Grün bekommt das zusätzliche Bit, weil das Auge dort am feinsten unterscheidet. Das Canvas ist ein flaches Array, `buf[y * 360 + x]`, und die Hilfsfunktionen darum sind der übliche kleine Satz: ein Pixel mit Grenzprüfung, Rechtecke, Linien, Kreise, Ringe, Text, und zwei Farbhelfer, einer addiert zwei Farben mit Sättigung, einer skaliert eine Farbe mit einem Faktor von 0 bis 256. Die Blenden sind dieser zweite Helfer, auf jedes Pixel angewandt.

Tabellen statt Rechnen

Ein Sinus pro Pixel, 100.000-mal pro Bild, würde das 20-ms-Budget allein aufbrauchen. Die Show macht deshalb, was Demos in den 90ern taten. Eine Tabelle hält 256 Sinuswerte; ein Winkel ist ein Byte, eine volle Umdrehung hat 256 Schritte, und `& 255` läuft kostenlos im Kreis. Zwei weitere Tabellen, einmal beim Start gebaut, halten für jedes Pixel den Winkel und den Abstand zur Bildmitte. Ein Tunnel oder ein Kaleidoskop braucht dann pro Bild gar keine Trigonometrie mehr: Es schlägt Winkel und Abstand nach und benutzt sie als Koordinaten in einer Textur.

Paletten sind der zweite alte Trick. Eine Szene wie das Plasma oder das Feuer berechnet keine Farben; sie berechnet eine kleine Zahl pro Pixel, und eine Palette übersetzt Zahlen in Farben. Verschiebt man die Palette, ändern sich alle Farben auf dem Bildschirm auf einmal, und das Bild scheint sich zu bewegen, obwohl sich kein einziger Pixelwert geändert hat. Das Folien-Deck zeigt das mit einem Streifen Zahlen, der stehen bleibt, während die Farben wandern.

Schrift und Text

Die Buchstaben sind Terminus, die Schrift der Linux-Konsole, in zwei Größen: 16×32 fett für Titel und 8×16 fett für Fließtext. `make` extrahiert sie auf dem Pi aus den Konsolenfonts des Systems, das Repository trägt also keine Schriftdateien. Das Programm liest die PSF-Dateien neben der ausführbaren Datei und bildet Latin-1-Zeichen auf Glyphen ab; die Folien-Decks sind UTF-8 und werden beim Laden nach Latin-1 gewandelt, was für Deutsch bequem reicht.

Die Folien sind Daten

Die zwei Decks liegen in `slides/*.txt`. Eine Zeile, die mit `#` beginnt, eröffnet eine Folie, eine Zeile mit `@` wählt den Live-Demostreifen am unteren Rand, eine Zeile mit `>` ist Code, alles andere ist Fließtext, der sich selbst tippt. Die Demostreifen sind kleine Szenen für sich: eine Pipeline aus Kästen, ein Pixelpuffer, der zeigt, wie ein eindimensionales Array zu einem zweidimensionalen Bild wird, ein Hitzepuffer mit sichtbaren Zahlen, eine Palette, die rotiert, und ein CPU-Messer, der die eigene Auslastung des Programms einmal pro Sekunde aus `/proc/self/stat` liest.

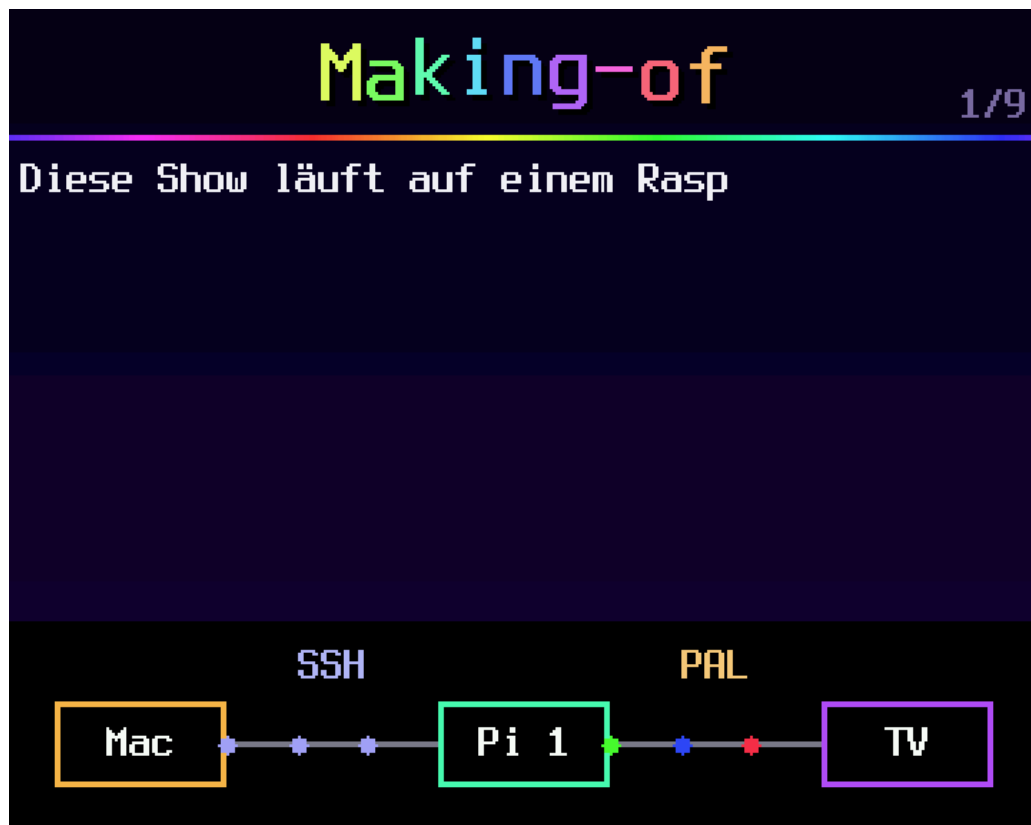


Abbildung 3: Eine Folie aus dem Making-of-Deck mit ihrem Demostreifen: der Signalweg vom Mac über den Pi zum Fernseher.

4. Szenentechniken

Plasma. Vier Sinuswellen, jede eine Funktion von x, y und Zeit, werden addiert; die Summe wählt eine Farbe aus einer Palette, die sich dreht. Ein riesiges „bc101“ ist aus dem Plasma ausgeschnitten, mit umgekehrter Palette, Kontur und Schatten.



Abbildung 4: Plasma: vier Sinuswellen, eine rotierende Palette, das Logo herausgeschnitten.

Tunnel und Kaleidoskop. Die Winkel- und Abstandstabellen machen aus dem Bildschirm Polarkoordinaten. Im Tunnel wird der Winkel das u einer Textur und der Abstand das v , und v mit der Zeit vorzuschieben fliegt die Kamera durch die Röhre; „BC101“ ist in die Wandtextur gestempelt. Das Kaleidoskop faltet den Winkel in acht Keile und spiegelt jeden zweiten, mit einer sich drehenden Blume in der Mitte.

Feuer. Das Feuer ist das aus der PlayStation-Fassung von Doom (1995), wie Fabien Sanglard es beschrieben hat: Die unterste Zeile eines Hitzepuffers glüht, jeder Wert wandert pro Bild eine Zeile nach oben, kühlt um einen zufälligen Betrag ab und driftet um ein zufälliges Pixel zur Seite. Eine Palette mit 37 Einträgen, von Schwarz über Rot und Gelb bis Weiß, macht aus Hitze Farbe. Hier läuft das Feuer in halber Auflösung, die Buchstaben des Titels sind selbst Hitzequellen, der Schriftzug brennt also, und am Ende wird die unterste Zeile gelöscht und das Feuer geht aus.

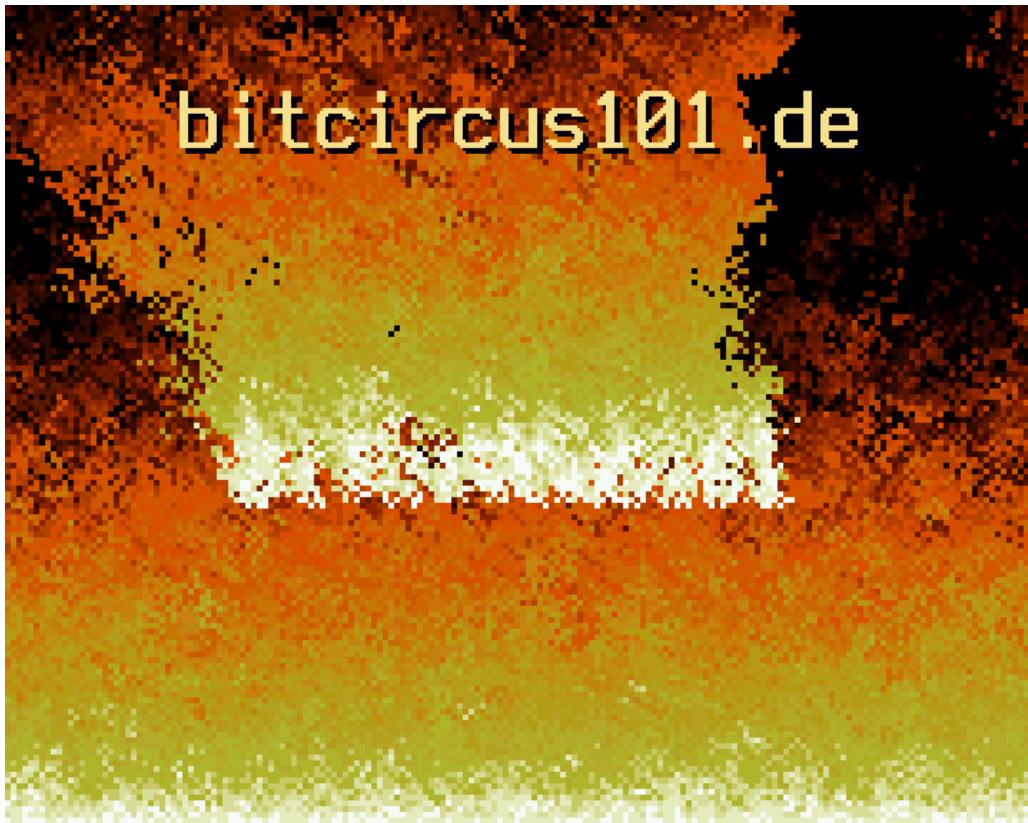


Abbildung 5: Feuer: ein Hitzepuffer, der aufsteigt, abkühlt und driftet; der Titel brennt.

Teilchen. In der Explosion wird jedes Pixel des Titels ein Teilchen mit Geschwindigkeit und Schwerkraft. Der Titel zittert, zerplatzt nach außen hinter einer Druckwelle, die Teilchen prallen vom Boden ab, und dann zieht eine Feder jedes einzelne an seinen Platz zurück, bis der Titel wieder steht.

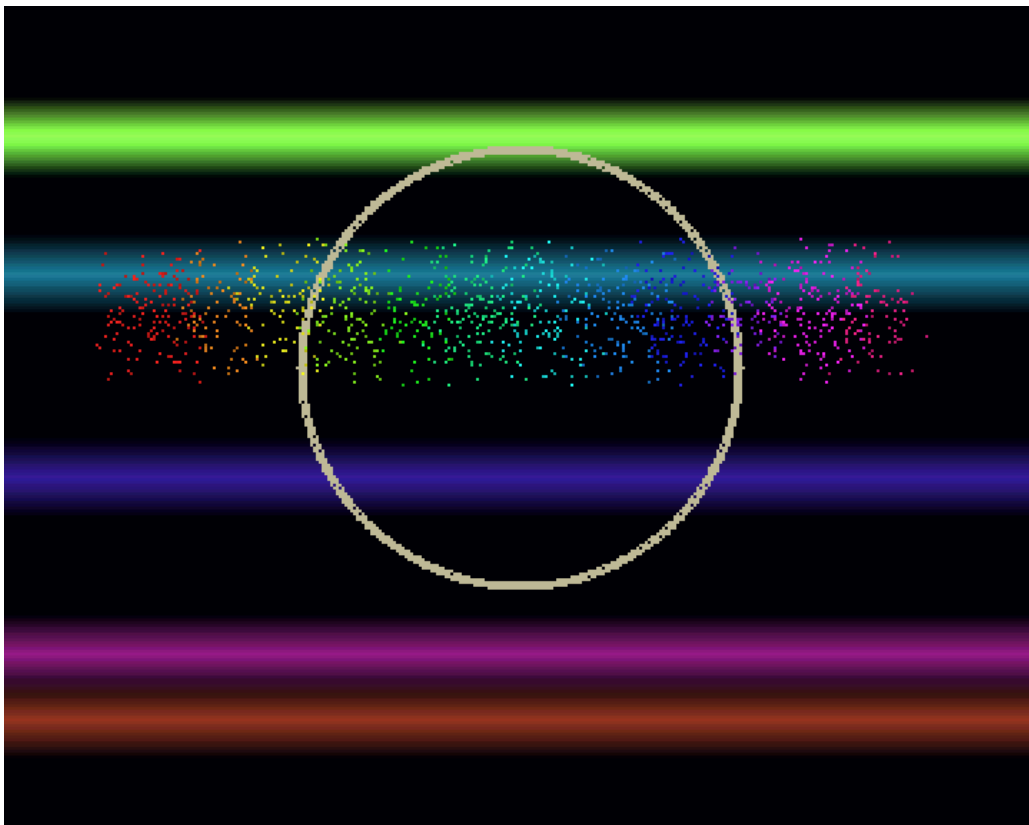


Abbildung 6: Die Explosion: jedes Pixel des Titels ist ein Teilchen.

Jonglage. Ein Strichmännchen-Clown jongliert die Buchstaben B, C, 1, 0, 1 in einer Fünf-Ball-Kaskade unter dem Zirkuszelt. Jeder Wurf ist eine Parabel, und die Kaskade ist das klassische Muster, in dem jede Hand im selben Takt zur anderen wirft, die Buchstaben kreuzen sich also in der Luft.

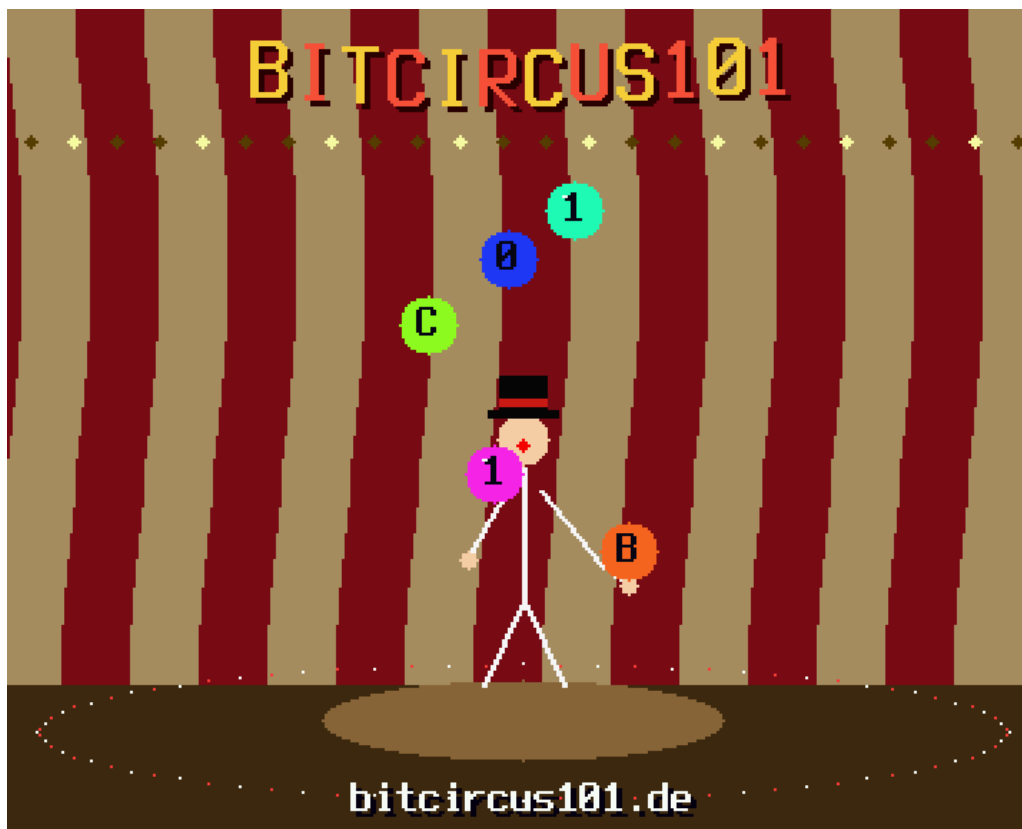


Abbildung 7: Der Clown jongliert die Buchstaben des Namens in einer Fünf-Ball-Kaskade.

Szenen mit Gedächtnis. Das Sternefeld, der Matrix-Regen, die Blumen und die Explosion berechnen kein Bild allein aus t ; sie führen einen Zustand von Bild zu Bild weiter. Sterne beschleunigen und die Kamera rollt; Tropfen des digitalen Regens, die das verborgene Logo kreuzen, rasten ein, bis „BC101“ steht; Blumen sprießen und blühen. Für die Werkzeuge ist das wichtig: Ein Standbild einer solchen Szene stimmt nur, wenn auch jedes Bild davor gerendert wurde.

Rasterbalken und Scroller. Der hüpfende Titel über Rasterbalken mit einem Sinus-Scroller am unteren Rand, und der Regenbogen, der sich selbst über den Himmel malt, während das Logo mit Schweif hindurchfliegt, sind das Heimcomputer-Erbe in Reinform: Farbbalken, die auf und ab laufen, Text, der welkt.

Fälschungen. Die Boot-Szene ist ein BIOS, das sich selbst tippt, Speicher zählt, drei Clowns und fünf Jonglierbälle erkennt und CIRCUS.SYS lädt. Die Glitch-Szene ist ein VHS- und Datenschaden-Look: ein RGB-versetztes Logo, Streifen, die reißen, Rauschschübe, eine getippte URL. Die Synthwave-Szene hat einen gestreiften Sonnenuntergang, ein scrollendes Neongitter und eine rotierende Punktwolke, die den Namen in 3D schreibt.

Die Bildröhre. Die Szene, die den Fernseher erklärt, auf dem sie läuft, zeichnet eine Röhre in 3D: flach schattierte Polygone mit Maler-Sortierung, eine additive Glasglocke, Beschriftungen an Hinweislinien. Der Strahl läuft von der Kanone durch die Ablenkspulen zum Schirm und tastet ein Raster aus 16 Zeilen ab, drei Bildschritte je Zeile und sechs für den Rücklauf, mit einem Nachleuchten, das wie Phosphor verblasst. Die Kamera bewegt sich absichtlich in Schritten von 12,5 Bildern pro Sekunde: Das ist das Ruckeln eines Renderings der 90er, und darunter läuft die Show mit 50 Hz weiter. Die gesamte Geometrie steht fest im Raum und wird in `init()` gebaut.

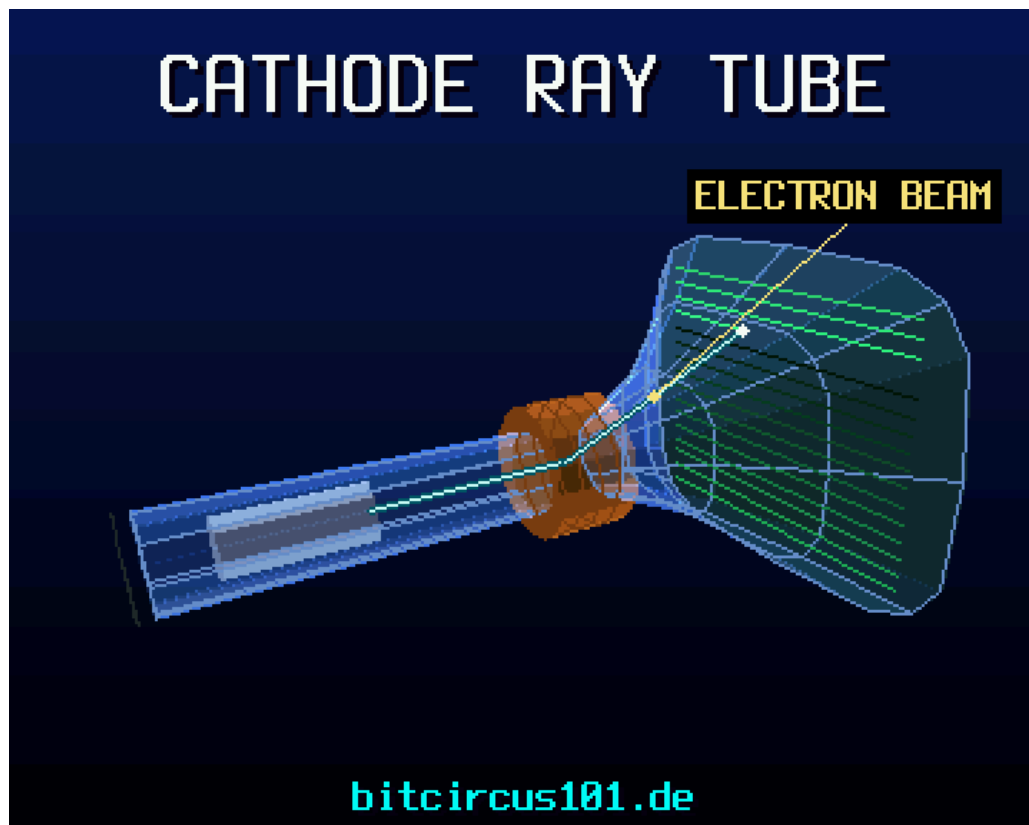


Abbildung 8: Die Bildröhre erklärt sich selbst: Kanone, Ablenkspulen, Strahl, Phosphor.

Bernd. Bernd das Brot springt wie das DVD-Logo über den Bildschirm, 115 Pixel pro Sekunde waagerecht und 118 senkrecht, und wechselt an jeder Wand zur nächsten von acht Farben. Die Überquerungszeiten, 2,6 s quer und 1,9 s hinunter, sind so gewählt, dass er bei $t = 10,0$ s die Ecke oben links exakt trifft, Pixel 0,0, den Moment, auf den jeder Betrachter eines DVD-Bildschirmschoners gewartet hat.

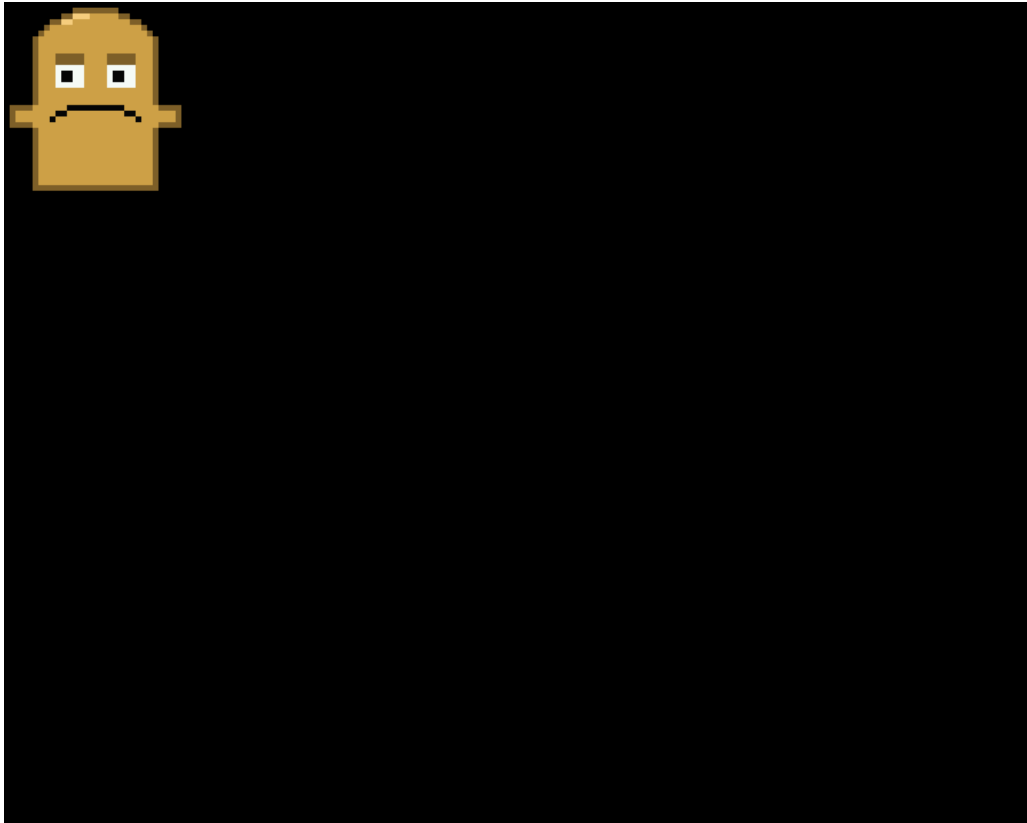


Abbildung 9: Bernd das Brot, im Augenblick des Eckentreffers.

5. Messungen auf dem Pi 1

Das Budget sind 20 Millisekunden pro Bild, und der Pi 1 hat einen Kern, um sie auszugeben; das Programm ist das Einzige, das malt, aber der Kernel, der WLAN-Stick und die gelegentliche Log-Zeile teilen sich diesen Kern mit ihm. Zwei Instrumente passen auf.

Das erste ist das Protokoll, das der Sequencer auf dem Pi schreibt: mittlere Bilder pro Sekunde und das langsamste Bild je Szene, gemessen auf der echten Hardware, den echten Framebuffer-Blit eingeschlossen. Die Szene, die die Bildröhre erklärt, die schwerste in der Schleife, hält in der laufenden Show 50,0 fps. Neben der Show auf demselben Kern mit dem Bench-Werkzeug gemessen, brauchen ihre Bilder 9 bis 13 ms, es bleibt also Platz für Blit und Kernel.

Das zweite Instrument ist `build/preview ... bench`, das eine gegebene Zahl von Bildern offline rendert und Mittelwert und Maximum meldet. Auf einem Arbeitsrechner sagen seine Zahlen nichts über den Pi, sie sind 20- bis 50-mal kleiner, aber als Verhältnis zwischen zwei Fassungen einer Szene sagen sie alles: Verdoppelt eine Änderung die Bildzeit auf dem Mac, verdoppelt sie sie auch auf dem Pi.

Zwei weitere Zahlen beschreiben das Programm. Das Ganze sind rund 2500 Zeilen C, die zu einer 95 KB großen ausführbaren Datei werden, gegen nichts gelinkt als die C-Bibliothek, die Mathematikbibliothek und die Speicher-routinen des Pi. Ein voller Build dauert auf dem Pi 1 etwas mehr als eine Minute; eine geänderte Szene ist in etwa sechs Sekunden neu gebaut, weil `make` nur übersetzt, was sich geändert hat.

6. Die Werkzeugkette: Sehen ohne Bildschirm

Die Show hat jemand per SSH geschrieben, der den Fernseher nicht sehen konnte, und selbst der Mensch, der es konnte, stand nicht immer davor. Also wuchs dem Projekt ein Satz Werkzeuge, die die Frage „Wie sieht das aus?“ ohne Bildschirm beantworten, und am Ende sind diese Werkzeuge das, worum es in dem Projekt geht.

Der Offscreen-Renderer

`build/preview` linkt jede Szene ohne Framebuffer und baut auf Linux und macOS. Es schreibt einzelne Bilder als BMP zu gewählten Zeitpunkten, nummerierte Bildfolgen mit gegebener Bildrate, misst eine Szene, und es sondiert sie: Für eine Liste von Zeitpunkten gibt es aus, wo die rein weißen Pixel liegen, als Schwerpunkt, und die Box um alles, was nicht schwarz ist. Der Schwerpunkt reicht, um zu prüfen, ob sich eine Marke so bewegt, wie sie soll.

Diesen letzten Modus gibt es wegen eines Fehlers, den Standbilder nicht zeigen konnten. Der Strahl in der Bildröhren-Szene schreitet mit 12,5 Bildern pro Sekunde, und an einer Stelle überlagerte sich seine Bewegung mit diesem Raster: Jedes einzelne Bild sah richtig aus, und in Bewegung lief der Strahl rückwärts. Erst eine Reihe von Schwerpunkten über aufeinanderfolgende Schritte hat es gezeigt. Die Lehre steht in der README: Standbilder übersehen Bewegungsfehler, Positionen über aufeinanderfolgende Schritte messen, Bilder hashen, die identisch sein müssen.

Das Flipbook

Ein Standbild ist eine Behauptung, ein Flipbook ein Beleg. `tools/flipbook.py` rendert ganze Szenen in ihrem eigenen Takt und bündelt jedes Bild in eine HTML-Datei, abspielbar und scrubbar, mit Zeitleiste, Marken und Notizen je Szene. Die Datei lädt nichts aus dem Netz, nicht einmal eine Schrift, man kann sie also verschicken oder irgendwo ablegen. Sie entsteht mit nichts als der Python-Standardbibliothek: Die BMP-Bilder werden mit `zlib` und einem handgeschriebenen Chunk-Writer nach PNG gewandelt und als Data-URLs eingebettet.

Ein Detail zeigt, wie wörtlich solche Werkzeuge sein müssen. Der Renderer tastet jedes Bild eine Millisekunde nach seinem nominellen Zeitpunkt ab, denn eine Szene, die genau im Abtasttakt schreitet, die Röhre mit 12,5 fps, darf nicht auf ihren Schrittgrenzen abgetastet werden, wo die Gleitkomma-Rundung den vorigen Schritt wählt und das Flipbook jedes Bild doppelt zeigt.

Ein Deploy, das sich selbst prüft

Das Repository ist die Quelle der Wahrheit; der Pi bekommt immer nur Kopien. `scripts/deploy.sh` gleicht die Quellen mit `rsync` ab, installiert die `systemd`-Unit, die es selbst erzeugt, und startet dann `tools/install.sh` auf dem Pi, das als Dienstanutzer baut, den Dienst aktiviert und neu startet, und ihn dann drei Sekunden lang beobachtet. Läuft die Show danach nicht mehr mit derselben Prozess-ID, schlägt das Deploy fehl. Eine Szene, die auf dem Pi abstürzt, aber nicht auf dem Mac, fällt beim Deploy auf, nicht am nächsten Morgen.

Der Build selbst hat eine Lektion auf die harte Tour gelernt. Ein Neustart mitten in `make` hinterließ einmal eine leere Objektdatei mit frischem Zeitstempel; `make` hielt sie für aktuell, und das Linken scheiterte an einer fehlenden Referenz. Seitdem wird jedes Ziel unter einem temporären Namen geschrieben und umbenannt, und `.DELETE_ON_ERROR` räumt halb geschriebene Dateien weg, wenn ein Rezept fehlschlägt.

Die Show im Browser

Die Website spielt die Show mit dem echten Szenencode. Ein kleiner Shim ersetzt den Framebuffer durch einen RGBA-Puffer und den Sequencer durch vier exportierte Funktionen, Emscripten übersetzt die Szenen, die Fonts und die Folien-Decks in ein 121 KB großes WebAssembly-Modul, und ein Seitenskript kopiert jedes Bild auf ein Canvas. Der Browser schreitet die Show in festen 20-ms-Takten voran, egal mit welcher Bildwiederholrate der Bildschirm läuft, die Szenen, die ihren Zustand Bild für Bild aufbauen, sehen also genau so aus wie auf der Röhre. Ein Rauchtest unter Node spielt eine Sekunde jeder Szene und lässt den Build scheitern, wenn eine nichts zeichnet.

Continuous Integration

Jeder Push baut die ganze Website auf GitHubs Runnern mit demselben `make site`, das ein Entwickler lokal ausführt: das WebAssembly-Modul, beide Flipbooks, diesen Text als HTML und PDF, die Abbildungen. Ein Secret-Scan über die ganze Historie und `shellcheck` über die Skripte vervollständigen den Satz. Pushes auf `main` werden auf GitHub Pages veröffentlicht.

7. Die Fernbedienung: eine Programmnummer für jede Szene

Eine Show in einem gemeinsam genutzten Raum braucht einen Weg zu sagen: die da, jetzt. Die Fernbedienung ist die Mitgliederseite von `tv`, und sie ist bewusst keine App, sondern `ssh zapper@<der pi>`. Mit einem Terminal öffnet das den Zapper, eine Fernbedienung aus Text: LED-Ziffern für das Programm, das Programm raster, ein Fortschrittsbalken, die Tasten, die ein Fernseher hätte. Mit einem Befehl ist es ein einzelner Schuss: `ssh zapper 7` hält Programm 7, `next` springt weiter, `auto` kehrt zur Schleife zurück, `play testcard 30` zeigt etwas ein einziges Mal. Das funktioniert aus der SSH-App eines Handys genauso wie vom Laptop.

Jede Szene hat eine Programmnummer: 00 ist Standby, ein schwarzes Bild, bei dem der Dienst weiterläuft; 01 bis 17 sind die Schleife in Spielreihenfolge; 20 bis 22 die Extras; ab 30 kommen die *Visuals*, Szenen, die Mitglieder hochladen. Hinter den Nummern steckt ein kleiner Steuerkanal im Sequencer. Einmal pro Bild fragt er einen Unix-Socket nach einer Textzeile und antwortet mit `ok` oder `err`; `status`, `list` und `log` lesen, `tune`, `play`, `next` und `auto` ändern, was

nach der laufenden Blende passiert. Nichts wird mitten im Bild unterbrochen: Eine Änderung ist ein Wunsch, den der Sequencer bei der nächsten Blende erfüllt, und so sieht das Bild auf der Röhre immer beabsichtigt aus.

Ein Visual ist eine Szene wie jede andere: eine C-Datei, die `const Scene scene_<name>` definiert. `ssh zapper upload foo < foo.c` schickt sie. Auf dem Pi wird sie geprüft (Text, Größe, die Definition), mit den Flags der Show kompiliert und mit dem Preview-Werkzeug offscreen in voller Länge durchgespielt, sodass ein Absturz oder eine absurde Bildzeit dort endet und nicht auf der Röhre. Dann wird ein neues Binary gelinkt, die laufende Show wird nach ihrer Blende per `exec` darauf umgesetzt und nimmt Programm und Modus mit, sodass niemand im Raum einen Neustart sieht. Antwortet der neue Build nicht binnen Sekunden, kehrt das vorherige Binary zurück, das als Hardlink aufbewahrt wurde, und `systemd` startet es. Ein Compile-Fehler kommt als Meldung des Compilers zurück zu dem, der hochgeladen hat.

Code von Mitgliedern bleibt Code. Der Schutz ist kein Filter über den Quelltext, sondern die Kiste, in der die Show läuft. Der Dienstanutzer `zapper` hat keine Shell: `sshd's ForceCommand` gibt jedem Schlüssel genau ein Programm, einen Dispatcher, der den Befehl gegen eine Whitelist aus Zeichen und Wörtern prüft, bevor irgendetwas passiert, und Schlüssel tragen eine Rolle, `op` zappt, `admin` lädt auch hoch und löscht. Die Show selbst läuft als dieser Nutzer in einer `systemd`-Sandbox: kein Netz, ein schreibgeschütztes Dateisystem samt eigenem Verzeichnis, als Geräte nur Framebuffer und Konsole, kein Weg zu mehr Rechten, ein `Syscall-Filter`; `systemd-analyze security` bewertet sie mit 0,8 von 10. Der Client des Dispatchers, `tvctl`, ist ohne Szenencode gebaut, damit ein Visual, das beim Start stirbt, nicht die Hand mitreißt, die es steuert. Ein Admin-Schlüssel bedeutet damit genau dies: Code läuft als `zapper`, mit diesen Rechten und keinen anderen. Der Deploy-Weg hält dasselbe Versprechen in die andere Richtung: Die Unit-Datei kommt vom Arbeitsrechner, und der Deploy-Login darf nur als `zapper` bauen und abgleichen plus ein paar feste `systemctl`-Befehle, sodass nichts, was `zapper` schreiben kann, je `root` erreicht.

Der eine Preis ist Rechenzeit. Der Pi hat einen Kern, und die Show will ihn ganz; eine offene SSH-Sitzung verbraucht etwas davon für die Verschlüsselung, und der Zapper fragt die Show zweimal pro Sekunde. Ein paar lange Bilder, während jemand zappt, sind der Preis der Fernbedienung.

8. Making-of mit einer KI

Programmiert hat die Show Claude, das KI-Modell von Anthropic, per SSH vom Mac aus. Es hat den Fernseher nie gesehen. Ein Mensch im Raum schon: Er meldete das schwarzweiße Bild, das Flackern, las die Zahlen von der Testkarte ab und sagte, wenn etwas falsch aussah. Alles andere, der C-Code, die Werkzeuge, die Folien, die Skripte, kam aus diesem Gespräch.

Drei Dinge haben das möglich gemacht, und es sind dieselben drei, die jede Grafikarbeit aus der Ferne handhabbar machen.

Erst die Augen bauen. Das Preview-Werkzeug, die Sonde und das Flipbook kamen früh und haben sich vielfach bezahlt gemacht. Eine KI, die ein Bild rendern und messen kann, wo die Dinge liegen, braucht weit weniger Schleifen über einen Menschen als eine, die fragen muss: „Sieht es jetzt richtig aus?“ Den Bewegungsfehler aus Kapitel 6 hat die Sonde gefunden, kein Mensch.

Das Gerät ist die Wahrheit. Zahlen vom Arbeitsrechner sind ein Modell des Pi, nicht der Pi. Bildzeiten sind nur im Protokoll auf der Hardware echt, das Bild ist nur auf der Röhre echt, und ein Deploy zählt erst, wenn der Dienst drei Sekunden später noch läuft. Die Werkzeuge sind so gebaut, dass die billige lokale Prüfung zuerst kommt und die echte zuletzt, und keine von beiden ausfällt.

Ein Repository, einfache Dateien, kleine Werkzeuge. Szenen sind Dateien mit zwei Funktionen. Folien sind Text. Die Flipbook-Spezifikation ist JSON. Die Werkzeuge sind ein C-Programm und ein Python-Skript ohne Abhängigkeiten. Nichts hier muss zweimal erklärt werden, und das zählt, wenn der Mitarbeiter zwischen zwei Sitzungen alles vergisst und das Repository liest, statt sich zu erinnern.

Was das nicht ist: die Behauptung, die Bilder seien neu. Plasma, Tunnel, Feuer und Teilchen sind das gemeinsame Erbe der Demoszene, und die Show sagt das auf ihren eigenen Folien. Neu, oder zumindest selten, ist die Kombination: ein Einkern-Board von 2012, das einen Fernseher in einer Form ansteuert, die nicht flackert, eine Show, die sich selbst erklärt, während sie läuft, ein Build, in dem nirgends eine Bilddatei existiert, und eine Werkzeugkette, mit der ein Programmierer, der den Bildschirm nicht sehen kann, mit Zuversicht Bilder dorthin liefert.

9. Lizenzen und Dank

Der Code ist GPL-3.0-or-later; die Texte, dieser Text, die Folien-Decks und die Flipbook-Notizen sind CC-BY-4.0, sie dürfen also mit einer Nennung von bitcircus101 in Vorträge, Artikel und Newsletter wandern. Die Buchstaben sind Terminus von Dimitar Zhekov, das die Website unter der SIL Open Font License 1.1 bündelt. Das Feuer folgt dem PlayStation-Doom-Effekt, wie Fabien Sanglard ihn beschrieben hat. Der Rest gehört der Demoszene, mit Dank.

Quellcode, Folien und Werkzeuge: <https://github.com/bmmmm/tvtv>. Die Show, das Flipbook und dieser Text: <https://bmmmm.github.io/tvtv/index.de.html>.

created with ♥ in bitcircus101 · assisted by Claude