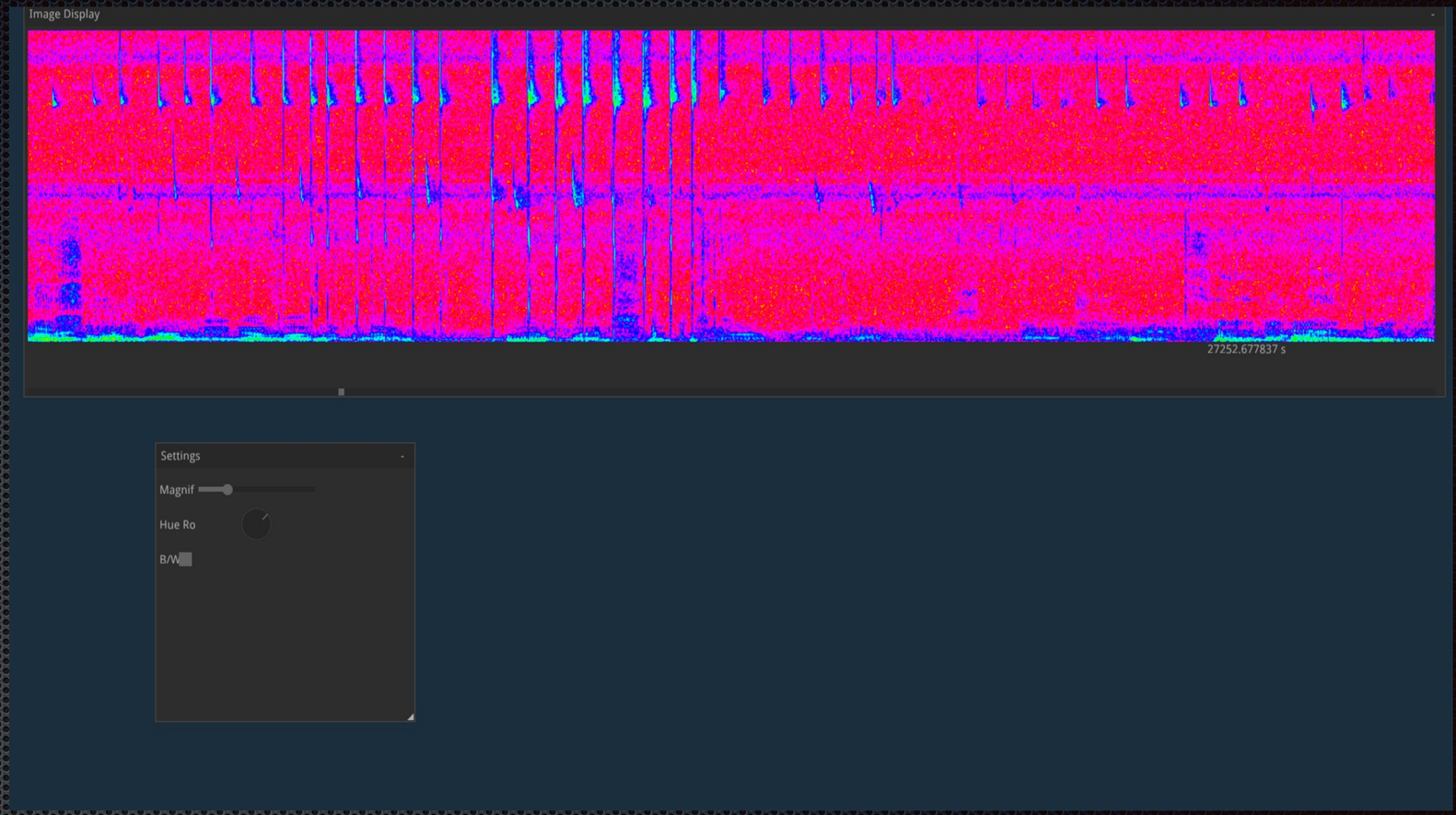
Dr. Olaf Flebbe

Fledermäuse Desktop



About me

PhD in computational physics
(Theor. Astrophysik Tübingen)

Former projects: Minix68k (68k FP Emulation), Linux libm.so.5 (High Precision FP), perl and python for epoc, flightgear, msktutil

Member Apache Software Foundation, PMC Apache Bigtop

Lead Developer Navigation System Specialist
Bosch eBike

Motivation

- Fledermausprofis:
  - Die Auswerteprogramme sind alle Windows basiert
- „Das kann doch nicht so schwer sein“
- Kann ich dann ja mal gleich als Vergleichsstudie für Programmiersprachen verwenden

Recap

- Fledermäuse können im Ultraschallbereich (20kHz – 200kHz) rufe aussenden, die spezifisch für die jeweilige Art sind
- Die Detektoren nehmen Audio Daten als Raw Data auf (typisch 10Bit sampling rate 256 kHz)

Signalverarbeitung

- 12 Bit Daten (0 – 4096)
- 512 Werte auswählen

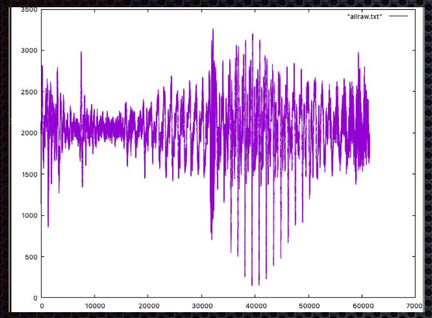
- 512 real FFT

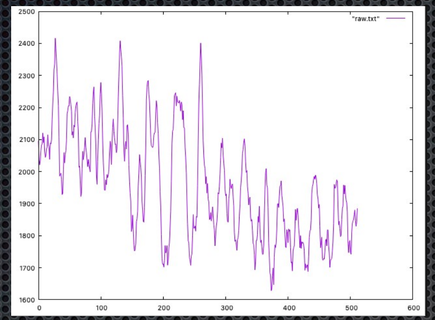
- Hamming Window

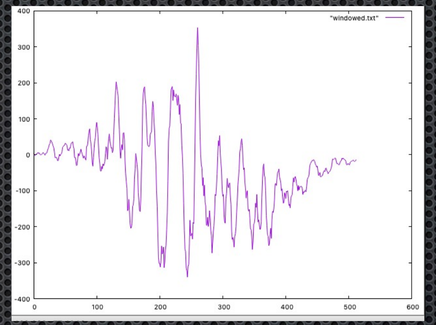
- Spektrogramm

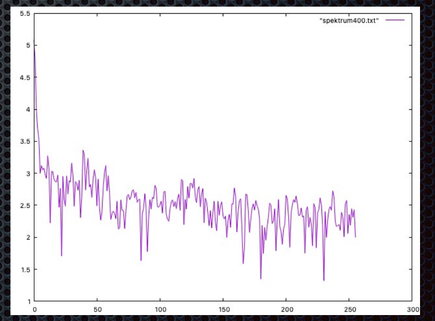
- Logarithmisch darstellen

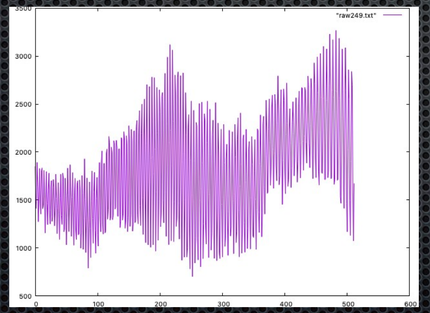
Signalverarbeitung

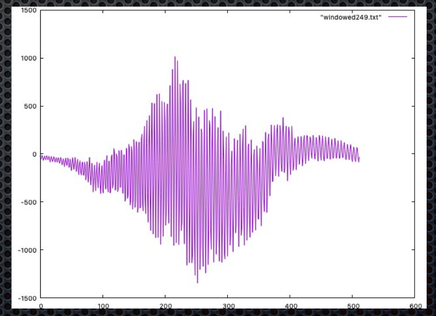


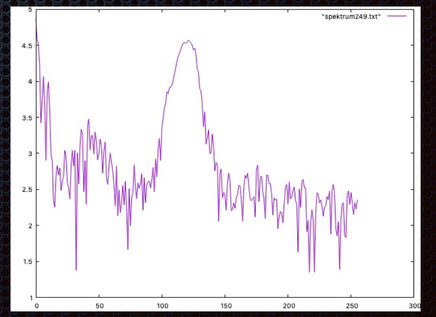




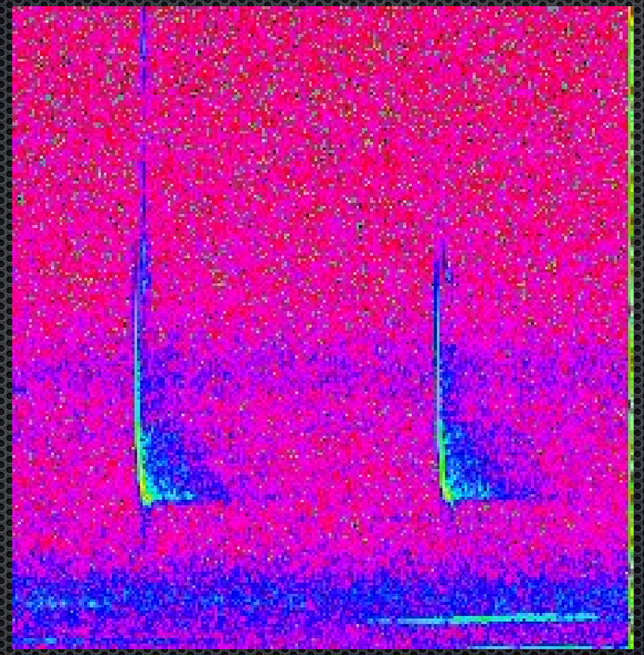








HSL Kodiert
(und C Programmierfehler am rechten Bildrand)



K(r)ampf der Programmiersprachen

- Fangen wir mal mit Qt/C++ an

~~Qt~~ Olaf Sucks

```
QImage image(width, height, QImage::Format_RGB888);
for (int i = 0; i < width; i++) {
  int index = ((len - SIZE) * i) / (width - 1);
  for (int j = 0; j < SIZE; j++) {
    assert(index + j < len);
    in[j] = (buf[index + j] - 2048) * window[j];
  }
  fftwf_execute(plan);

  int count_more_05 = 0;
  for (int j = 0; j < height; j++) {
    const std::complex<float> *outf =
        reinterpret_cast<std::complex<float> *>(out[j]);
    float p = abs(*outf);
    const float z = log10(p);

    float ang = (z + 0.4) / (6 + 0.4);
    QColor c;
    c.setHslF(ang, 1.f, ang);
    image.setPixelColor(i, j, c.toRgb());
  }
```

C++ Ade

So gehts nicht:
Qimage, Qcolor etc sind viel zu langsam ... Reference Counting, Locking uvam.

Müsste alles lowlevel selbst implementieren oder ganz anders (mehr später dazu)

K(r)ampf der Programmiersprachen

- Dann halt C
- Go verwende ich in $Arbeit
- Rust, weil …

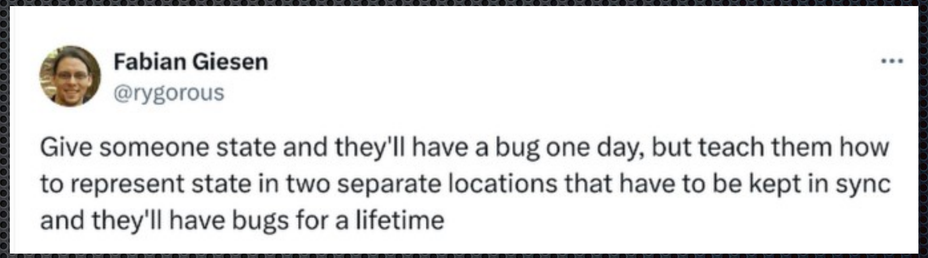
K(r)ampf der UI Technologien

- Qt als typische Retained Mode UI
  - UI sind widgets, objektorientiert, State im Widget getrennt von Business Logic. (MVC)

(Nicht verwechseln Retained / Immediate Mode Graphics)

Was macht denn die Spieleindustrie?

- Dear ImGui protagonist der Immediate Mode GUIs



- Kaum State
- Rendert in CommandQueue
- Verschiedenste Backends die CommandQueue mittels OpenGL, Vulkan, Metal, Framebuffer …
- „60 fps“

Randbedingungen für Projekte

- Immediate Mode UI, native in der Sprache
- Memory Mapped I/O (kein „Einlesen“)
- Problem ist embarrassingly parallel
  - (Keine Datenabhängigkeit): Viele 1-D FFT, also Parallel
- Keine Compiler Warnungen: Sicher/Korrekt soweit möglich
- MacOS ohne deprecated libs unterstützen

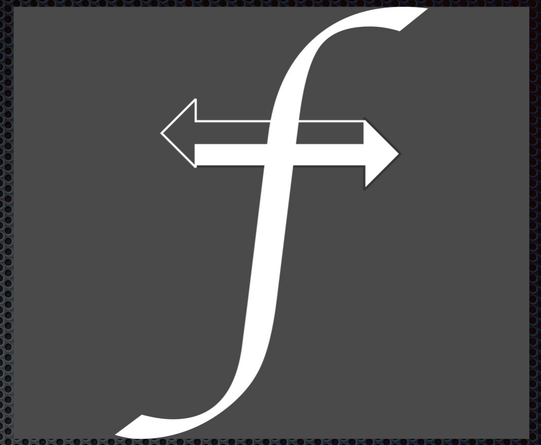
Objective:

- Laufzeit FFT
- Anzahl Dependencies
- Support MacOS, Linux, Windows
- Crosscompiling von Linux
- Größe des Executables

Golang

- Schwierig UIs zu finden
  - Die meisten Referenzen sind obsolete Bindings
  - Gioui: Sehr spärliche Dokumentation, Go nativ, technisch extrem anspruchsvoll zu verstehen
  - Fyne: Komischen Nachgeschmack: UI hat polling loop Entwickler verneint dies (Dauernde CPU Last im Idle zustand)
  - Nukular: (reimplementation von Nuklear in go)..
    zu spät gefunden

Bat Detect Go Fyne



- https://github.com/oflebbe/bat-desktop-go-fyne

- Stats:
  - 25 external dependencies, Cgo,
  - 107 sec compile time
  - Executable 30 MB
  - OpenGL auf Mac Deprecated, Shader ???
- 2% CPU Zeit bei idle, obwohl retained UI Framework

Bat Detect Go gioui

- https://github.com/oflebbe/bat-desktop-go-gioui

- Stats:
  - 9 external dependencies, Cgo,
  - 15 sec compile time
  - Executable 9.8 MB
- Windows Crosscompile OOB (!)
- Verwendet Metal auf MacOSX, Shader ???
- Extrem schwierig zu verstehen, wenig Doku

Bat Desktop in Go

- Langsam, Keine schnelle FFT Implementierung (CGO benötigt)
  - Keine Real FFT
  - Go builtin Complexer Datentyp
  - Memory Mapping nicht wirklich unterstützt, tut aber.
  - Memory Management / vs CGO

Eingestellt

Rust

- Wenig native UI Support
  - Egui: immediate Mode, Dokumentation so ok, gute Beispiele
  - Egui: Metal auf MacOSX möglich, Shader nicht verstanden sehr aktives/volatiles Projekt: LLM haben lustig halluziniert
  - Nur rustFFT für FFT gefunden: Extrem schnell, aber lässt ohne Real FFT viel Leistung liegen.
  - Mmap mit memmap2, schwierig zu finden.
  - Impedanz Mismatch rustFFT und rayon (oder gehts ganz anders) =>Viel Aufwand „wie sag ich es dem Compiler“: Project stalled

Bat Detect Rust egui

- https://github.com/oflebbe/bat-desktop-rs

- Stats:
  - 305 (!) external dependencies, (incl. div. C libraries)
  - 49 sec compile time
  - Executable 15.7 MB

- Doku so ok, Windows Crosscompiling funktioniert gut, MacOS wegen des SDK mit Tricks.

Batdetect in C:
Threads

- C17 Threads nicht verwendet da MacOS diese nicht implementiert
- POSIX threads native auf MacOS, Windows praktisch immer dabei

- Aber Workers sind aufwändig zu programmieren:

- Deswegen: OpenMP: gcc, clang implementiert seit Äonen wird im HPC verwendet:
- Auf Mac muss man halt llvm von Homebrew verwenden, dann gibts OpenMP auf Mac, MSVC supported OpenMP
- Tipp: OpenMP so Verwenden, das single threaded programm auch korrekt ist!

Batdetect in C
Single Header Libs

- Shawn Barrets single header libraries:
- Created „flo“ headers for sharing with the bat detector
- Dependency Management trivial
- Verwendet:
  - Glad OpenGL Bindings (Generator!)
  - Meow FFT
  - Nuklear
    - Zusammen mit SDL2/OpenGL3 backend

Batdetect C
Neueres C99

- Flexible Array Member

```
// 16bit color flo_pixmap_t (565)

typedef struct

{

  size_t len;
  unsigned int width;
  unsigned int height;
  // flo_pixmap_format_t format;
  uint16_t buf[];

const flo_pixmap_t
*flo_pixmap_create_str(unsigned int len,
const char str[len], ...
```

- Variable Length Arrays
- Compound Literals , Designated initialization

```
return (textures_t){.left = left, .right = right, .correlation = correlation};
```

- Pass structs by Value (like rust, go...)
- Try to avoid Pointers
- Static Arrays (VSCode Argh!)

size_t flo_filesize(FILE fp[static 1]);

<complex.h>

<stdint.h>: uint16_t, ...

Batdetect C
C23

- <stdbool.h> : true und false sind keywords (VSCode argh!)
- nullptr
- constexpr

Speed: Shader

- Statt Bilder zu erzeugen, lade Matrizen in die Grafikkarte
- Und kontrolle das Rendering
- HSL oder B/W Kodierung
- Größe

OpenGL 3 / GLES 3 shader

C Security

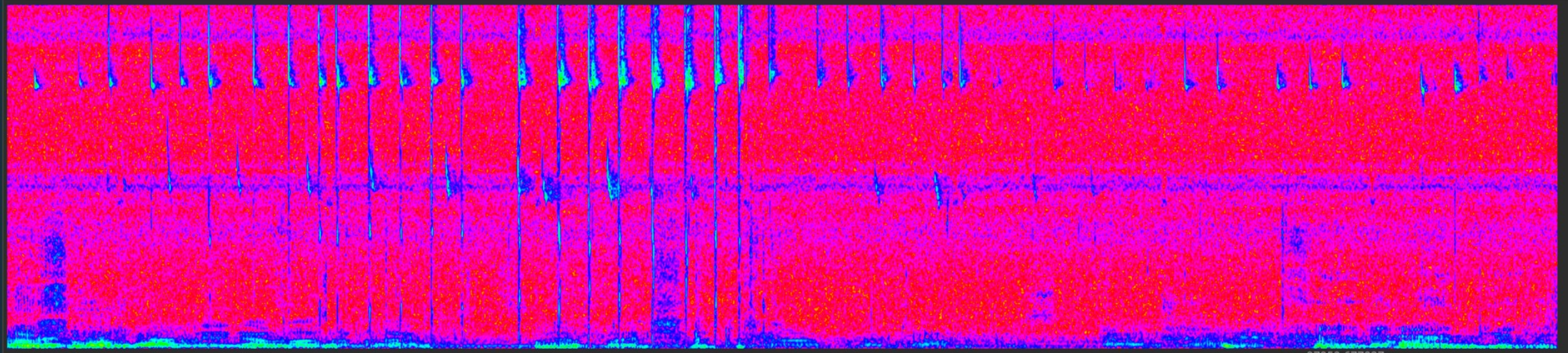
- OpenSSF Recommendations
- Interessant: Man wird plötzlich in C dazu gezwungen sich um unsigned und signed ints zu kümmern, wie in Rust/Go
- Viele Rangechecks mit assert(), wenig overhead, zT nicht messbar
- Mit Typen auf Stack arbeiten (das lernt man bei TinyGo auch -> Keine GC nötig, bzw. Kein Malloc/Free Probleme)
- Cmake Release definiert NDEBUG: WTF?
- ASAN sehr nützlich beisst sich aber mit OpenMP

Batdesktop in C mit Nuklear

- https://github.com/oflebbe/bat-desktop

- Stats:
  - 4 external dependencies, (SDL2, meowfft, nuklear, GLAD)
  - Nur SDL2 muss installiert sein
  - 7 sec compile time
  - Executable 5.7 MB
  - Gdb loads tons of runtime dependencies :)
  - Builds easily on the platforms : Crosscompiling with Mingw, Container MacosX SDK

Image Display



27252.677837 s

Settings

Magnif

Hue Ro

B/W

Erfahrungen

- Es ist wirklich grenzwertig:
  - Rangecheck mal übersehen (hier kann ja nichts passieren): Bäng (Bilder enthielten erratische Pixel)
  - Tippfehler: Falschen Typ beim sizeof Allozieren: Bäng
  - Array Decay ist echt ein Problem
  - Strings als char * war eine schlechte idee
  - CMake kann bei Tippfehlern sehr ungnädig sein

Sinnkrise

- Boomer Probleme
  - Performance
  - Size
  - Zu Kompliziert?
- IT Probleme gelöst statt eine funktionsfähige SW entwickelt

Fragen?

- https://github.com/oflebbe/batdetect
- https://www.oflebbe.de
- Fosstodon https://fosstodon.org/@0x01af