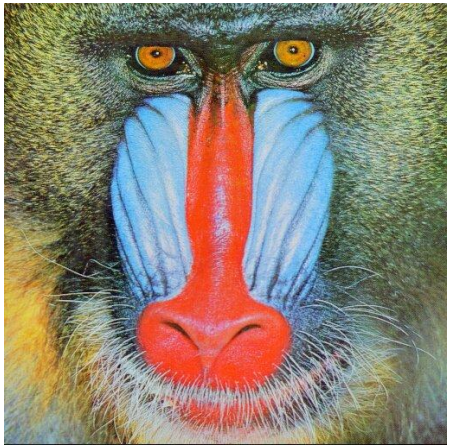
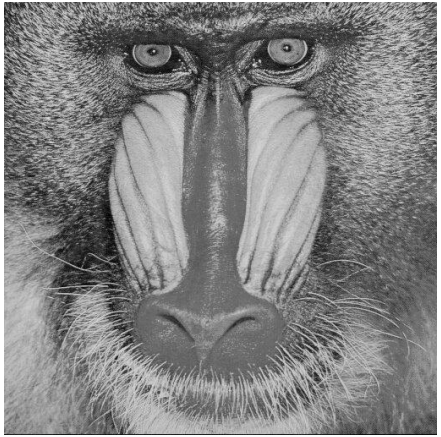
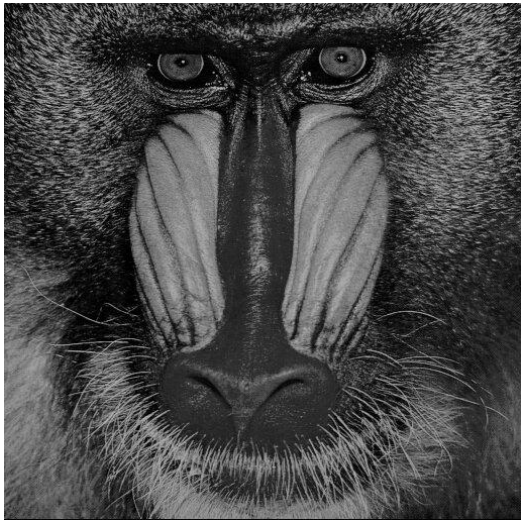
Team 104

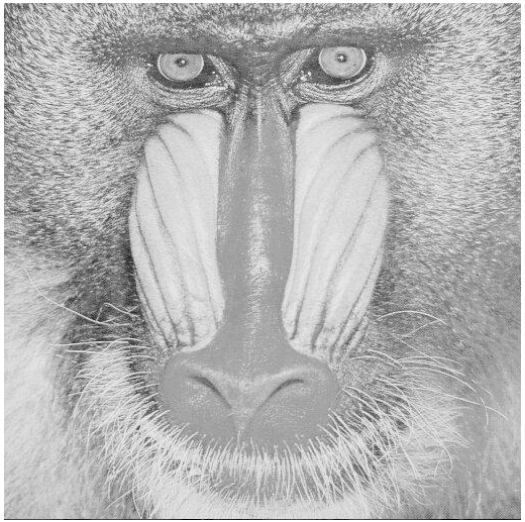
Maryna Redka, Yevhenii Karpushchenko, Maksym Floria

Problemstellung: Gammakorrektur









Problemstellung

Theoretischer Teil

- Eingabeformat: PPM2 (24bpp, P6)
- Ausgabeformat: Netpbm3

- Parameter a, b, c auf HVS basieren
- Exponentialfunktion nur mit einfachen Rechenoperationen berechnen

- **Algorithmus entwickeln**
  - Graustufen umwandeln
  - Gammakorrektur anwenden

Funktionsweise

$$D = \frac{a \cdot R + b \cdot G + c \cdot B}{a + b + c}$$

$$Q_{(x,y)} = D$$

$$Q'_{(x,y)} = \left(\frac{Q_{(x,y)}}{255} \right)^{\gamma} \cdot 255$$

Problemstellung

Praktischer Teil

I/O-Operationen in C

- PPM-Datei einlesen
- Pointer auf Bilddaten an Assembly übergeben
- Ausgabedatei im Netpbm-Format erstellen

Funktion power in Assembly

- Nur einfachen Rechenoperationen
- Genauigkeit behalten
- Rechenzeit minimieren

Funktion gamma_correct in Assembly

- Parameter: RGB-Pixel, Breite, Höhe, a, b, c, Gamma
- Ergebnis: Graustufen + Gammakorrektur
- Puffer im Rahmenprogramm allokieren

Rahmenprogramm in C

- Optionen verarbeiten
- Randfälle abfangen
- Fehlermeldungen ausgeben

- Benchmarking ausführen
- Implementierungsversionen verwalten
- Die Daten dem Assembly weitergeben

Funktion gamma_correct in Assembly

1. Die RGB-Werte lesen
2. Die Formeln richtig implementieren
3. Die Graustufen-Werte schreiben

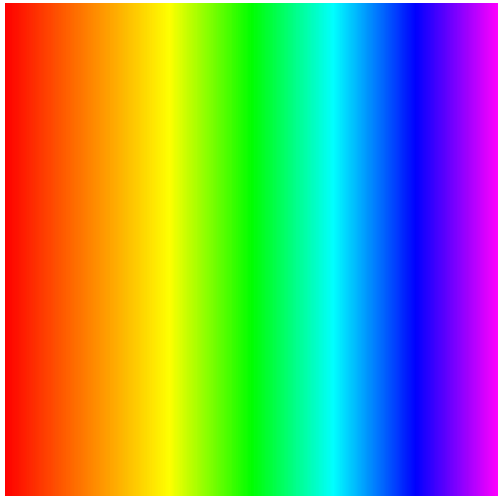
$$D = \frac{a \cdot R + b \cdot G + c \cdot B}{a + b + c}$$

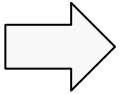
Versionen:

- V2: einfacher Ansatz
- V1: SIMD Instruktionen
- V0: Optimierung von power

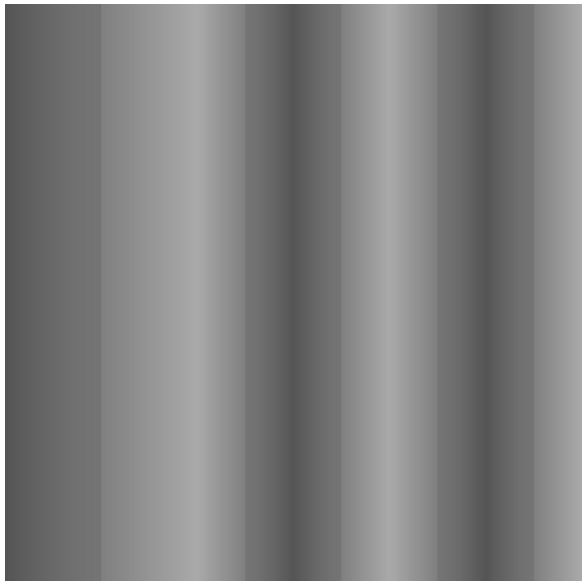
$$Q'_{(x,y)} = \left(\frac{Q_{(x,y)}}{255} \right)^{\gamma} \cdot 255$$

Werte für a,b,c

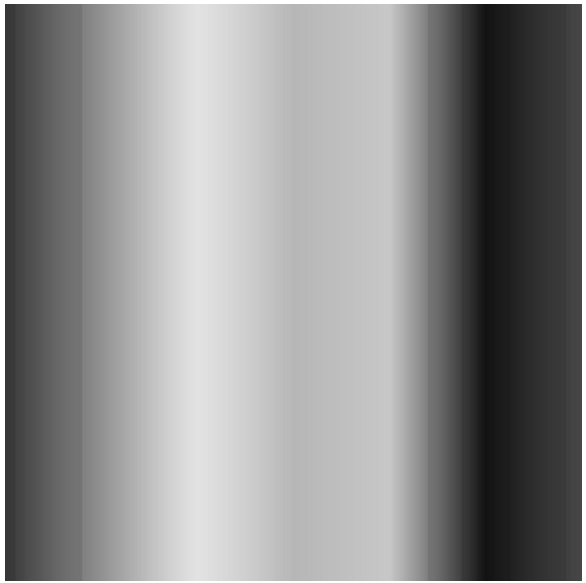




a = b = c = 1



Adobe RGB Werte:
a ≈ 0.2, b ≈ 0.7, c ≈ 0.07



Für beide Fälle gamma = 1

Funktion power in Assembly

- Umwandeln c = a^b
- e^b(ln(a))

  a als d*2^d darstellen

- e^b(ln(m*2^d))
- e^b(ln(m) + d*ln(2))

- ln(m) als ln(1+x) approximieren als summe von minimax Polynom
- ln(1+x) = a1*x + a2*x^2 + …

- e^k als e^(n^ln(2) + r) darstellen als
- 2^n * e^r

- e^r approximieren mit Taylor\Mcloren Reihe

Genauigkeit

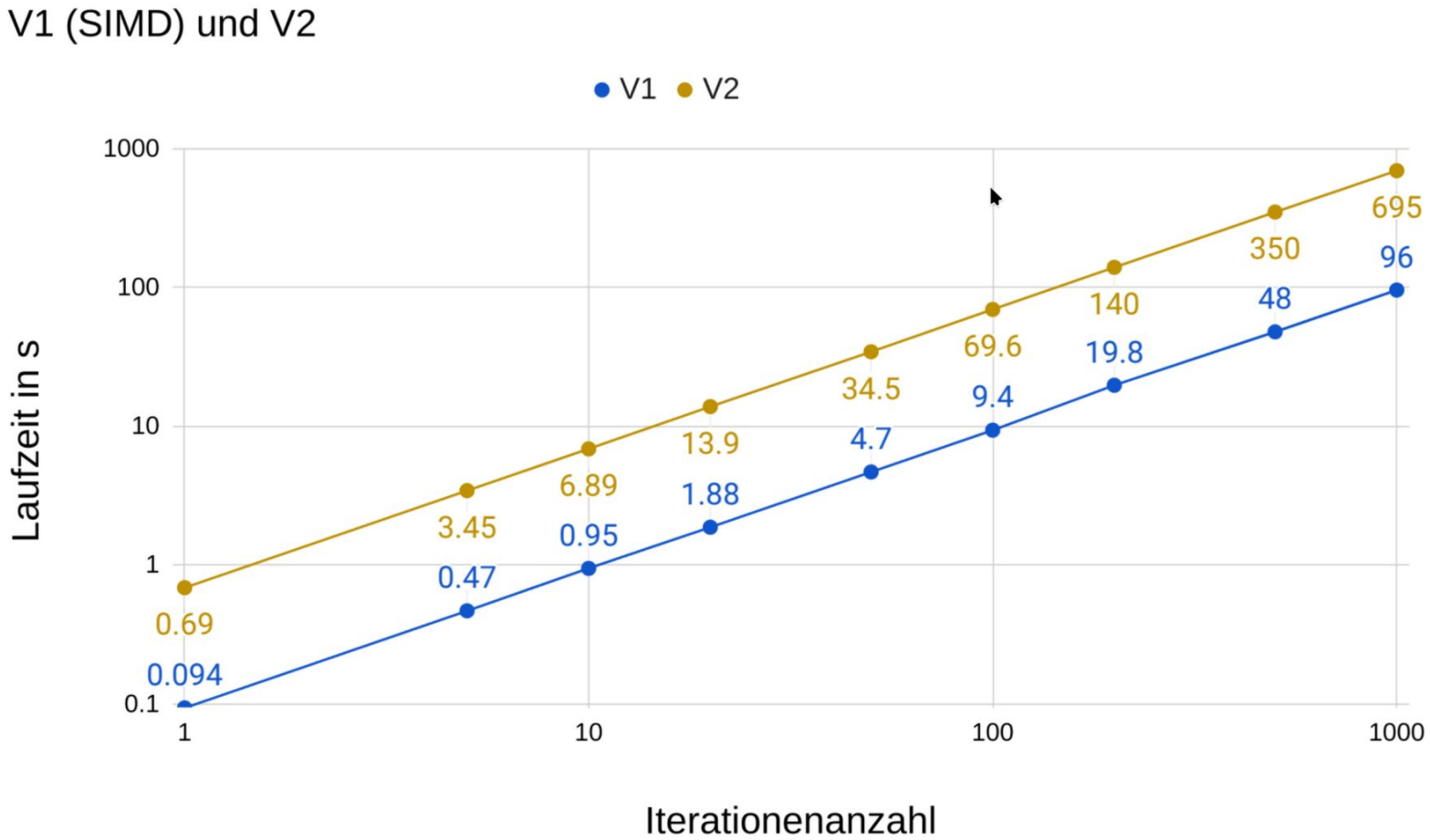
V1 durchschnittliche Genauigkeit:

- 10^-7
- 0.0001% zur math.h pow()

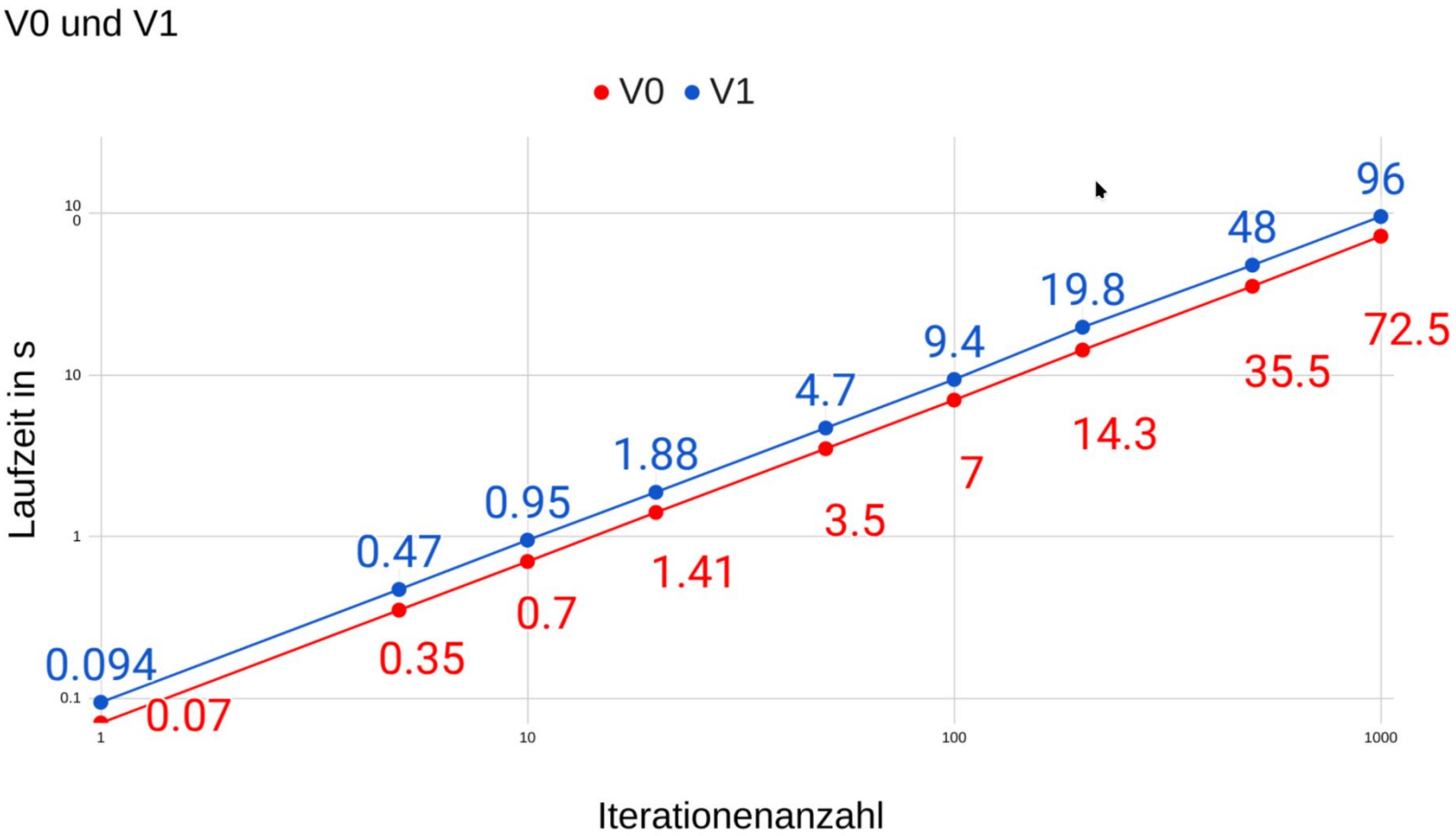
V0:

- 25% schneller
- 2.8% error zur math.h pow()

Performanzanalyse



Performanzanalyse



Performanzanalyse

Getestet wurde auf einem System:

- Intel i5-1135G7 Prozessor, 4.20GHz,
- 8 GB Arbeitsspeicher,
- Arch Linux x86_64,
- Linux-Surface Kernel 6.12.7.

Kompiliert wurde mit:

- GCC 14.2.1 mit der Option -O2.

Die Berechnungen wurden durchgeführt mit:

- PPM-Eingabedatei der Größe 5184x3456.

Zusammenfassung und Ausblick

- Farbbilder → Graustufen + Gammakorrektur
  - Implementierung in C und Assembler
- Optimiert für Genauigkeit & Performanz
  - Erfolgreich getestet und validiert