Heinrich Heine
Universität Düsseldorf

**UNIVERSITÄT DÜSSELDORF**
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Prof. Dr. Michael Schöttner

# Hinweise zum Grafikmodus

**„Betriebssystem-Entwicklung“**

## 1. Textausgaben

### Allgemeines

Die Vorgabe enthält eine Schriftart sowie Funktionen, um damit rudimentär Text im Grafikmodus ausgeben zu können. Es handelt sich um eine nicht-proportionale Schrift, bei der alle Zeichen die gleiche Breite haben. Die Schriftart ist als monochrome Rastergrafik gespeichert und die Rastergrafik für jedes Zeichen hat neben der gleichen Breite auch die gleiche Höhe. Dies vereinfacht die Handhabung sehr stark, im Vergleich zu True-Type-Schriftarten, bietet aber im Gegenzug weniger Möglichkeiten für die Darstellung.

### Schriftarten

Für die Schriftart gibt es eine Header-Datei (`devices/fonts/font_8x8.rs`) welche die Rastergrafiken für alle Zeichen beinhaltet. Die Schrift wurde ursprünglich mit dem Werkzeug `cpi2fnt` erzeugt, welches noch vom Amiga-Betriebssystem (Amiga ist ein Home-Computer aus den 80er Jahren) stammt und welches im Netz nicht mehr auffindbar ist. Solche Schriften wurden teilweise auch in Linux von X11 verwendet und es finden sich weitere solche Schriften im Internet, die man in sein hhuTOS auch integrieren kann.

Weitere Schriftarten gibt es im Linux-Kernel: https://github.com/torvalds/linux/tree/master/lib/fonts

### Ausgabefunktionen

Für die Ausgabe von Zeichenketten mit einem bestimmten Font in einer gegebenen Farbe gibt es in `vga` die Funktion `draw_string`.

*Hinweis: eine vergleichsweise einfache Möglichkeit, um True-Type-Fonts als Rastergrafiken in hhuTOS nutzen zu können geht über das BDF-Format (Glyph Bitmap Distribution Format von Adobe), wofür es kostenlose Werkzeuge im Internet gibt. Eine echte Unterstützung von True-Type-Fonts bei Zeichen dynamisch berechnet und dargestellt werden ist sehr komplex.*

## 2. Einbinden und Ausgaben von Bitmaps

In der Vorgabe befindet sich das Logo der HHU als Bitmap in der Datei `bmp_hhu.rs`. Das Format der Datei ist unkomprimiert und pro Pixel werden 4 Bytes gespeichert, je 1 Byte für Rot, Grün und Blau und 1 Byte für den Alpha-Wert (0 - 255 Deckkraft).

In GIMP kann eine Grafik als C-Sourcecode (*.c) exportiert werden. In den Optionen beim Export sollten **keine** Glib Typen verwendet werden und das Bild sollte **mit** Alpha-Kanal exportiert werden. Außerdem darf keine Kompression aktiviert werden. Im Header der Datei steht als Name für die `struct "gimp_image"`, was man in einen geeigneten Namen abändern sollte. Diese Datei kann mit dem kleinen C-Programm `cbmp2rs` in eine Rust-Datei konvertiert werden, die dann eingebunden werden kann. Wie das funktioniert sieht man in der Vorgabe zu Aufgabe 7; dort ist ein fertiges Bitmap-Beispiel dabei.

*Hinweis: Man kann mit dem Makro `include_bytes!` auch einfach in Dateien in Rust einbinden. Weitere Infos siehe hier:* https://doc.rust-lang.org/core/macro.include_bytes.html