Übung Nr. 3:

Das Programm `DIPintro` unter http://homepages.thm.de/christ/ soll um Punkt- und Lokaloperationen erweitert werden. Das Menü in der Datei `DIPintro.c` ist hierfür vorbereitet. Die nun folgenden Aufgaben betreffen Funktionen in der Datei `DIPops.c`; die dazu vorgesehenen Stellen sind mit dem Bezeichner `MORE_OPS` und den dazugehörigen Präprozessor-Anweisungen (s.a. Datei `DIPops.h`) gekennzeichnet.

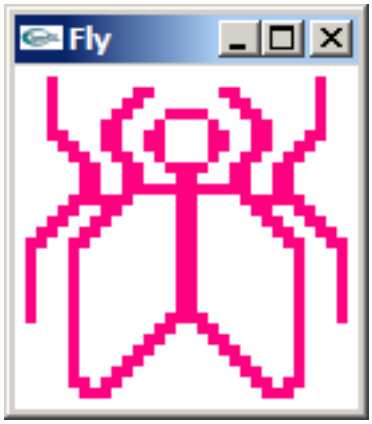
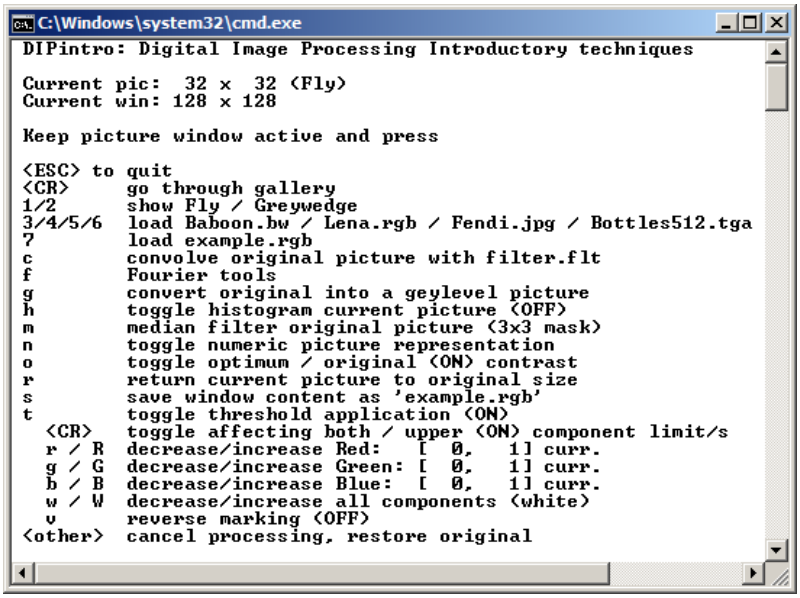


Abb. 1 `DIPintro.exe`



Als erstes soll die Funktion `threshPic()` ergänzt werden (Abb. 1).

Wie man dem Kommentar und dem kurzen Code entnehmen kann, ist diese Funktion für die Ausführung von Schwellenwertoperationen zuständig. Im Menü und im Code wird kurz erläutert, daß wahlweise die Grau- und Farbwerte innerhalb oder außerhalb des im Menü eingestellten Wert-Intervalls markiert werden können. Der Code für die Markierung von Pixeln außerhalb des Intervalls ist schon vorhanden (Menüpunkt „`reverse marking`“); es fehlt lediglich die zu dieser (umgekehrten) Markierung entgegengesetzte Grundeinstellung.

1. Teilaufgabe: Vervollständigung des Codes der Funktion `threshPic()`

Ein nützliches Werkzeug, um die genaue Belegung und die Veränderung eines Bildes zu verfolgen, ist die numerische Ausgabe seiner Farbwerte. Hierfür ist in `DIPops.c` die Funktion `numberPic()` implementiert. Sie gibt wahlweise die Werte der roten, grünen oder blauen Farbkomponente jedes Pixels in einem Bildausschnitt aus, während sie im Bild selbst grafisch den betrachteten Ausschnitt mit einem Rechteck markiert. Die Wirkungsweise von `numberPic()` sollte aus dem Bedienungsmenü und der Kommentierung im Code ersichtlich sein; man kann sie sich auch vergegenwärtigen, wenn man sich in Abb. 2 die Ausgabe der blauen Pixelkomponenten zu Abb. 1 genauer anschaut.

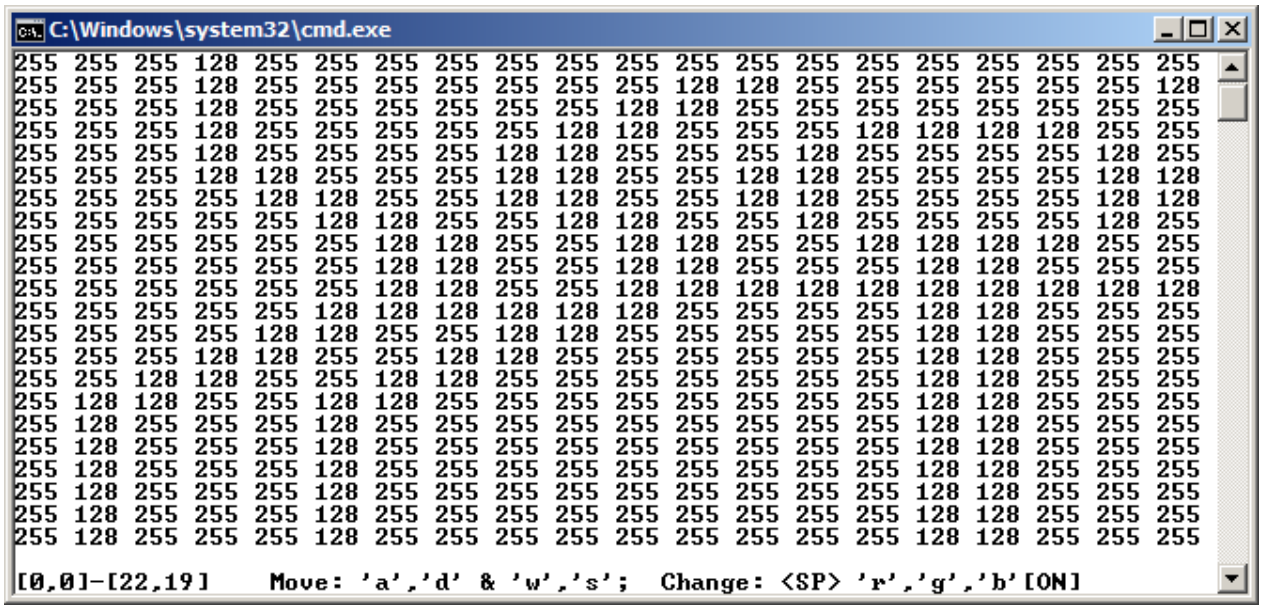


Abb. 2 Numerische Farbwert-Ausgabe der blauen Komponenten zu Abb. 1

Da die Zahlenwert-Darstellung von Bildern unübersichtlich sein kann, soll `numberPic()` um die Bilddarstellung mit Pseudograuwerten erweitert werden, damit interessante Bildstellen leichter identifiziert werden können (Abb. 3). Als Beispiel für einen Vorrat an ASCII-Zeichen, die hierzu verwendet werden können, dient in `DIPops.c` eine erste Belegung der globalen Feldvariablen `pseudo[]`.

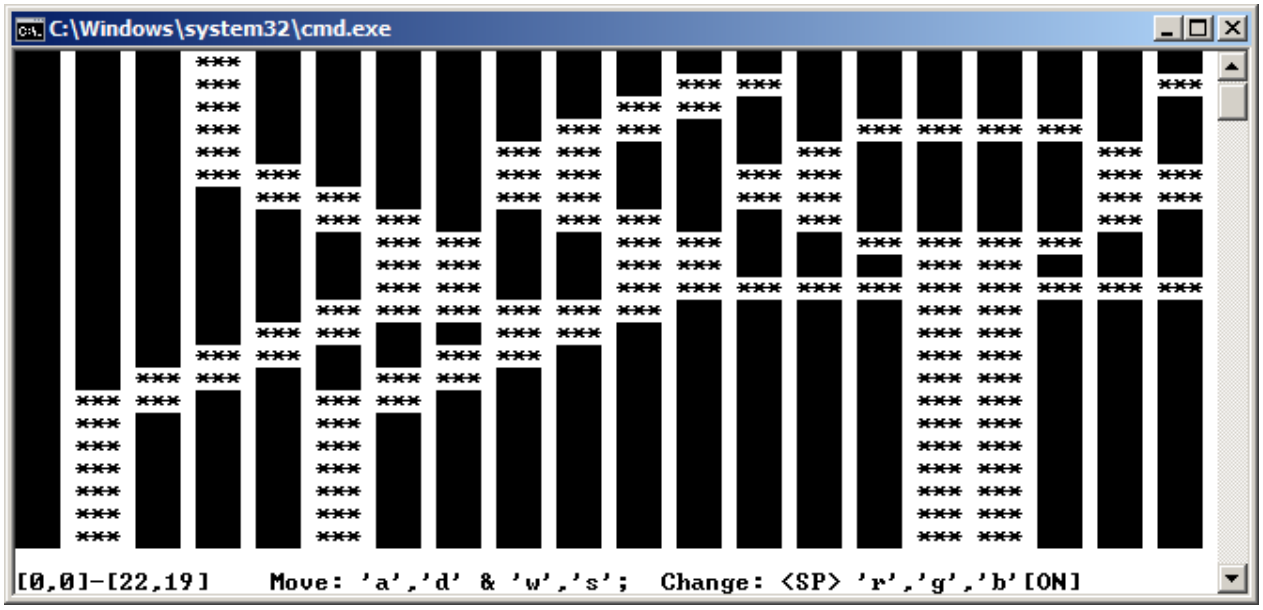


Abb. 3 Farbwert-Ausgabe der blauen Komponenten zu Abb. 2 mit Pseudo-Grauwerten

2. Teilaufgabe: Vervollständigung der Funktion `numberPic()` durch Einsatz von Pseudograuwerten. Anzahl und Auswahl der verwendeten Zeichen sind frei wählbar.

Als weitere nützliche Punktoperation ist die Umwandlung von Farb- in Schwarzweißbilder vorgesehen. Dafür ist die Funktion `cv2grey()` vorbereitet. Ihre Wirkungsweise und die dazu benötigte Code-Ergänzung sollten aus dem Theorieteil dieses Faches (ggf. unter Zuhilfenahme des Autors) hervorgehen.

3. Teilaufgabe: Vervollständigung der Funktion `cv2grey()`

Es wird empfohlen, sich als nächstes der Funktion `median()` zuzuwenden. Die Routine ist vorbereitet, damit die gewünschte Maske ausreichenden Speicherplatz reserviert bekommt. Es fehlt noch das Sortieren der von der Maske umfaßten Pixel.

4. Teilaufgabe: Vervollständigung der Funktion `median()` mit einem Sortierverfahren der eigenen Wahl.

Den Schwerpunkt dieser Übung bildet die Faltung im Ortsbereich. In der vorbereiteten Funktion `convol()` ist dafür gesorgt worden, daß das zu filternde Bild, das mit der untersten Zeile zuerst gespeichert ist, zu Beginn umgedreht wird und am Ende der Bearbeitung zurück in seine programm-interne Aufstellung gebracht wird[1]. `BottomUp()` ist die dafür zuständige Routine; sie hat starke Ähnlichkeit zu entsprechenden Teilen der Ladefunktionen.

5. Teilaufgabe: Ergänzung der Funktion `BottomUp()`.

In `convol()` wird die Verschmierungsmaske zunächst geladen und normiert. Die hierzu benötigten Routinen `loadMask()` und `normMask()` sind „gebrauchsfertig“. Die für die Bildung des Faltungsintegrals notwendige Spiegelung der Maske an der x- und der y-Achse ist jedoch nicht vollständig.

6. Teilaufgabe: Codierung der Funktion `mirrMask()`.

Bei der Codierung der Faltungsfunktion `convol()` ist anhand der Beschreibung in der Vorlesung und der Kommentare im vorhandenen Teil des Codes (und bei Bedarf mit Hilfe des Autors) zu beachten, daß die geforderte Masken-Operation aus einer zweimaligen Anwendung derselben Maske bestehen kann. Dies wird über die Variable `purpose` gesteuert; in diesem Fall wird beim zweiten Durchlauf die transponierte Maske verwendet. Im Umgang mit unterschiedlichen Masken-Größen kann es interessanter sein, die Transposition nicht explizit (als eigenständige Funktion), sondern eher als gezielte Auslese von Indizes der eindimensional gespeicherten Masken-Matrix zu realisieren.

[1] Die Maßnahme ist unnötig aufwendig und daher keineswegs praxisnah; sie soll lediglich den Umgang mit den Pixel-Indizes erleichtern.

Die Maske selbst (hinter der Adreßvariablen `mask`) wird in `loadMask()` aus der formatierten (Text-) Datei `filter.flt` geladen (Verzeichnis `_Data`). Das schafft die Möglichkeit, zur Laufzeit des Programms die Maske in der Datei nach Belieben zu setzen. Die ausgelieferte Datei enthält bereits die wichtigsten der in der Vorlesung besprochenen Filter; so braucht nur die jeweils interessierende Filterung an den Datei-Anfang gesetzt zu werden.

7. Teilaufgabe: Codierung der Funktion `convol()`

[Tip: Speichert man das Bild während der Histogramm-Darstellung samt Markierung ab (Menüpunkt 's‘) und lädt es erneut (Menüpunkt '7‘), so erkennt man in `numberPic()` mit Hilfe des Debuggers anhand der Markierungspixel, deren Farbe in `threshPic()` dreidimensional mit `glColor3ub(255,255,0)` gesetzt wurde, welchen Zahlenwert OpenGL als 4. Dimension für die Transparenz (Alpha) hinzufügte, aber auch (über Vergleich der Variablen `ipixel` und `pixel`), in welcher Reihenfolge Bytes bei der eingesetzten Hardware numeriert sind:

Beim Intel-PC führen die gesetzten Werte (`255,255,0`) zur Variablen-Belegung:

```
pixel[0]=255; pixel[1]=255; pixel[2]=0; pixel[3]=255;
```

Der Wert von `ipixel` erfährt man gleichzeitig vom Debugger als 4.278.255.615, also binär:

```
1111 1111   0000 0000   1111 1111   1111 1111.
```

(In welchen Bytes sind MSB und LSB enthalten?)]