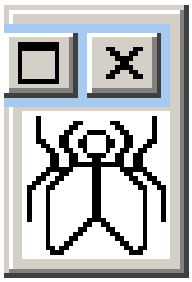
Übung Nr. 1:

Das Programm `DIPintro.c` (aus: Digital Image Processing) soll in mehreren Etappen zu einem einfachen Anwendungsprogramm mit wichtigen Techniken der digitalen Bildverarbeitung komplettiert werden. Seine Bedienung geht weitestgehend aus dem Menü (Abb. 1) und der ausführbaren Programm-Kopie (`DIPintro.exe` im `Demo`-Verzeichnis) hervor. Im Rahmen der laufenden Lehrveranstaltung wird die interne Arbeitsweise des Programms untersucht und die volle Funktionalität der ausgelieferten unvollständigen Version hergestellt. Die Stellen, in denen Code eingefügt oder geändert werden muß, sind mit Präprozessor-Direktiven (hier: `#ifdef MORE_INTRO` etc.) gekennzeichnet (insg. ca. 12 Zeilen Code).



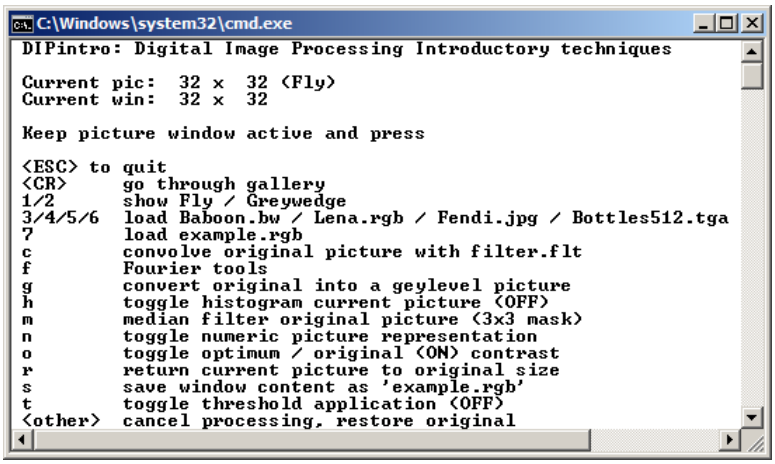


Abb. 1 `DIPintro.exe`

Zum Compilieren des unter http://homepages.thm.de/christ/ angebotenen VS-Projekts wird die Installation der darin verwendeten (für die nächsten Schritte benötigten) Version von OpenCV vorausgesetzt. Eine Anleitung hierzu findet sich auf derselben Seite der Internet-Präsenz des Autors.

1. Teilaufgabe: Installation von OpenCV 2.1

In der ausgelieferten Version ist `DIPintro.c` fehlerfrei compilierbar. Das Programm arbeitet aber noch nicht korrekt: Statt dem pixelweise (in `DIPintro.h`) eingegebenen Fliegen-Muster aus der Funktion `flyPic()` (Abb. 1) wird lediglich ein schwarzes Fenster in der erwarteten Größe dargestellt. Bei näherer Betrachtung wird der Grund dafür klar:

Nach dem Programmstart oder nach entsprechendem Tastendruck (also in `main()` oder in der Funktion `Keyboard()`) wird ein Bild zunächst in die Struktur `img` geladen.[1] Anschließend wird durch Aufruf von `copyDIPimg()` eine Kopie des Bildes erstellt (Struktur `aoi`), die zur Laufzeit verarbeitet und den bei jedem Aufruf der Callback-Funktion `Display()` dargestellt wird.[2]

[1] Die Struktur `DIPimg` ist in `LoadImg.h` deklariert; sie enthält Angaben zu Bildnamen, -breite, -höhe, Anzahl Byte pro Pixel und Format, sowie die Bilddaten selbst.

[2] Dieses Vorgehen stellt sicher, daß bei Veränderungen des Bildes, die wiederholt auf das Original zurückgreifen müssen (z.B.: digitale Erstellung einer Mehrfachbelichtung) das Original stets unverändert verfügbar bleibt. Der Name der Kopie (`aoi`, aus: Area Of Interest) weist darauf hin, daß bei Ausbau des Programms auch Bildausschnitte statt dem gesamten Bild behandelt werden könnten.

2. Teilaufgabe: Der Code der Funktion `copyDIPimg()` ist zu erstellen.[3]

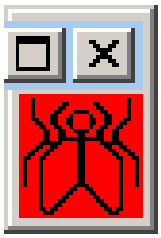


Abb. 2
Fliegenmuster in Rot

Nach erfolgreicher Behandlung erscheint das Fliegenmuster nicht wie erwartet (Abb. 1), sondern vor rotem Hintergrund (Abb. 2). Das liegt daran, daß der Präprozessor, der im Muster den Unterstrich (_) als die Zahl 255 compiliert, damit Rot statt Weiß im Bild einsetzt.

Mit der vorgenannten Korrektur kann nun das Fliegen-Muster wie geplant dargestellt werden.

3. Teilaufgabe: Der Makro für '_' in `DIPintro.h` ist von Rot auf Weiß zu ändern.

Drücken auf '2' soll einen Graukeil erzeugen; stattdessen erscheint jedoch ein kleiner roter Streifen im sonst schwarzen Fenster. Aber auch eine Rückkehr zum Fliegen-Muster (Drücken von '1') erzeugt wieder ein schwarzes Fenster.

Der Grund dafür ist, daß das Programm nach jeder Wahl eines neuen Bildes es versäumt, das Fenster zu aktualisieren. Dies kann noch eingepflegt werden, indem am Ende der `switch`-Anweisung zur Registrierung von Tastatur-Eingaben (Funktion `Keyboard()`) die Meldung ans GLUT-Fenstersystem eingesetzt wird, daß das Fenster aktualisiert werden muß (sobald die Funktion verlassen wird und GLUT wieder die Kontrolle über die Programmausführung erlangt). Der Platz für den Befehl `glutPostRedisplay()` ist am Ende der Funktion mit Präprozessor-Direktiven gekennzeichnet.

4. Teilaufgabe: Die Fenster-Aktualisierung ist in die Funktion `Keyboard()` einzufügen.

Abb. 3 Graukeil in Rot

Erfolgreiche Erledigung dieser Teilaufgabe läßt nun zwischen Bildern wechseln. Aber auch der Graukeil erscheint in roter Farbe (Abb. 3). Die Ursache hierfür ist am Anfang der Quellcode-Datei `DIPintro.c` zu suchen: Der Makro `GREYLEVEL` erzeugt Rot- statt Grautöne.

5. Teilaufgabe: Der Makro `GREYLEVEL` ist auf die Grün- und Blau-Farbkomponenten auszudehnen.

Mit `DIPintro` können auch weitere Bilder (über die Programmbibliotheken OpenGL und OpenCV) geladen und mit Hilfe von OpenGL-Funktionen gezeigt werden. Die hierzu verwendeten Laderoutinen sind im Projekt enthalten. Es fällt jedoch auf, daß Bilder, die mit OpenCV-Funktionen geladen werden (u.a. JPG-Dateien) „auf dem Kopf stehen“ – genauer: die Bildzeilen werden in umgekehrter Reihenfolge dargestellt (Abb 4).

[3] Zur Vermeidung verfrühter oder unterlassener Freigabe des Speichers für die Bildkopie empfiehlt sich zur Speicherplatzreservierung die Verwendung von `realloc(NULL, ...)`. Näheres ist der empfohlenen Literatur zu entnehmen.

Abb. 4 Bild vor `ocvBottomUp()`-Korrektur

Abb. 5 Bild unter `DOITYOURSELF`

Dieser Effekt markiert den Unterschied zwischen OpenGL und GLUT einerseits, die, als Programm-Bibliotheken für Grafik, y-Koordinaten von unten nach oben gerichtet annehmen, während OpenCV andererseits, als Bibliothek für Bilder (und ggf. Schrift), die Zeilen (und somit die y-Werte) von oben nach unten numeriert. Zuständig dafür ist die Funktion `ocvBottomUp()` in der Datei `ShowAux.c`. Zur Behebung dieses Fehlers muß dem Index der Zeilen für OpenGL (Variable `oglIdx`) eine Nummer zugewiesen werden, die in umgekehrter Richtung als der Zeilenzähler für OpenCV (Variable `ocvIdx`) wächst.

6. Teilaufgabe: In der Funktion `ocvBottomUp()` soll der Zeilen-Index für OpenGL (`oglIdx`) in umgekehrter Reihenfolge zugewiesen werden als der Zeilenzähler für OpenCV (`ocvIdx`).

Ein weiterer Konflikt zwischen den beiden Systemen offenbart sich, wenn man den Identifier `DOITYOURSELF` aktiviert (Abb. 5). Dies setzt den OpenCV-Aufruf `cvCvtColor(…, CV_BGR2RGB)` außer Kraft, was einer Vertauschung der Farbauszüge durch Wechsel zwischen `RGBA` und `ABGR`. Bei Interesse lohnt es sich, den Komponentenzähler (`jcomp`) entsprechend rückwärts laufen zu lassen (was sonst der OpenCV-Aufruf erledigt).

Es ist vorgesehen, jedes ausgegebene Bild oder Verarbeitungsergebnis über den Menüpunkt 's' als Datei `'example.rgb'` zu sichern. Die hierfür zuständige Funktion `SaveAoI()` speichert jedoch vorerst nur den Bild-Header, während die Schleifen zur Speicherung der Bilddaten leer sind.

7. Teilaufgabe: Die Funktion `SaveAoI()` ist um die Sicherung der Bilddaten in den vorbereiteten Schleifen zu ergänzen.

Der Erfolg des letzten Schrittes läßt sich u.a. dadurch verifizieren, daß man das gespeicherte Bild lädt (Menüpunkt '7').