LEHRSTUHL FÜR RECHNERTECHNIK UND RECHNERORGANISATION

Aspekte der systemnahen Programmierung bei der Spieleentwicklung

Projektaufgabe – Aufgabenbereich Bildverarbeitung (A203)

1 Organisatorisches

Auf den folgenden Seiten finden Sie die Aufgabenstellung zu Ihrer Projektaufgabe für das Praktikum. Die Rahmenbedingungen für die Bearbeitung werden in der Praktikumsordnung [1] festgesetzt, die Sie auch über die Praktikumshomepage [2] aufrufen können.

Der **Abgabetermin** ist der **1. Februar 2016 (23:59 MEZ)**. Die Abgabe erfolgt per Git in das im Gitlab für Ihre Gruppe eingerichtete Projektrepository. Bitte beachten Sie unbedingt die in der Praktikumsordnung angebene Liste von abzugebenden Dateien!

Wie in der Praktikumsordnung beschrieben, sind die Aufgaben relativ offen gestellt. Besprechen Sie diese innerhalb Ihrer Gruppe und konkretisieren Sie die Aufgabenstellung. **Der erste Teil Ihrer Projektpräsentation** ist eine kurze Vorstellung Ihrer Aufgabe in einer Tutorübung in der Woche **12.12.2016-16.12.2016**. Wählen Sie bitte eine Übung, in der mindestens ein Team-Mitglied angemeldet ist.
Erscheinen Sie bitte **mit allen Team-Mitgliedern** und bereiten Sie einen Kurzvortrag mit folgenden Inhalten vor:

- 1 Folie: Vorstellung der Team-Mitglieder
- 2 Folien: Zusammenfassung der Aufgabenstellung

In der Übung wird eine Beamer vorhanden sein, an den Sie Ihr eigenes Notebook anschließen können. Alternativ können Sie auch Ihre Präsentation als PDF Datei auf einem USB-Stick mitbringen. Der Kurzvortrag wird von einer Person gehalten.

Bei Fragen/Unklarheiten in Bezug auf den Ablauf und die Aufgabenstellung wenden Sie sich bitte an Ihren Tutor.

Mit freundlichen Grüßen
Die Übungsleitung

PS: Vergessen Sie nicht, sich rechtzeitig in TUMonline zur Prüfung anzumelden. Dies ist Voraussetzung für eine erfolgreiche Teilnahme am Praktikum im laufenden Semester.

[1] `http://wwwi10.lrr.in.tum.de/~buettned/GEP-ASP/GEP-ASP-Praktikumsordnung.pdf`
[2] `http://www.lrr.in.tum.de/lehre/`

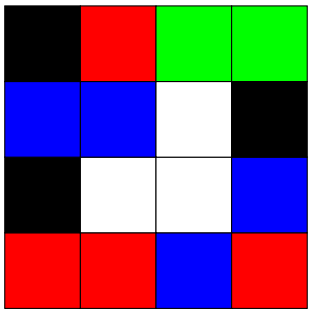
2 Bildkompression

2.1 Überblick

Im Zuge Ihrer Projektaufgabe werden Sie theoretisches Wissen aus der Mathematik im Anwendungszusammenhang verwenden, um einen Algorithmus auf Ihrem Raspberry Pi 3 implementieren. Sie konzentrieren sich dabei auf das Feld des *Image Processing*, in welchem Pixelbilder, wie sie typischerweise Digitalkameras produzieren, als Eingabe für bestimmte Algorithmen verwendet werden und mathematische Überlegungen dadurch sichtbar gemacht werden.

2.2 Funktionsweise

Die Lauflängencodierung (Run-Length-Encoding) ist ein einfaches Verfahren zur Datenkompression. Betrachten Sie beispielsweise die 16 Pixel des folgenden Bildes:



Anstatt die Pixel einzeln abzuspeichern, kann man auch *Folgen* von gleichen Pixeln speichern. Im konkreten Fall scannt man das Bild zeilenweise von links oben nach rechts unten und speichert Tupel $T = (L, V)$ mit L hintereinander auftretenden Werten $V \in \{R, G, B, W, S\}$:

$$RLE = ((1,S),(1,R),(2,G),(2,B),(1,W),(2,S),(2,W),(1,B),(2,R),(1,B),(1,R)) \tag{1}$$

Die Werte lassen sich dann wiefolgt lesen:

- $(1, S)$ Ein Pixel Schwarz
- $(1, R)$ Ein Pixel Rot
- $(2, G)$ Zwei Pixel Grün
- ...

So lässt sich aus den komprimierten Daten das Bild Stück für Stück wieder rekonstruieren.
Ihre Anwendung findet die Lauflängencodierung beispielsweise im BMP-Format für Pixelgrafiken. Ihr Ziel ist es, ein Programm zum Komprimieren von BMP-Grafiken zu erstellen.

2.3 Aufgabenstellungen

Ihre Aufgaben lassen sich in die Bereiche Konzeption (theoretisch) und Implementierung (praktisch) aufteilen. Sie können (müssen aber nicht) dies bei der Verteilung der Aufgaben innerhalb Ihrer Arbeitsgruppe ausnutzen. Alle Antworten auf konzeptionelle Fragen sollten in Ihrer Ausarbeitung erscheinen. Besprechen Sie nach eigenem Ermessen außerdem im Zuge Ihres Vortrags einige der konzeptionellen Fragen. Die Antworten auf die Implementierungsaufgaben werden durch Ihrem Code reflektiert.

2.3.1 Theoretischer Teil

- Erarbeiten Sie anhand geeigneter Sekundärliteratur die genaue Funktionsweise der Lauflängenkodierung.
- Das Bitmap-Format ist gut dokumentiert. Erarbeiten Sie mithilfe einer geeigneten Quelle die Dateiformate für unkomprimierte und RLE-komprimierte Bitmaps mit indizierten 8 Bit pro Pixel. (Eine mögliche Quelle für das komprimierte Datenformat ist folgender MSDN-Artikel: `http://msdn.microsoft.com/en-us/library/windows/desktop/dd183383.aspx`) Ihr Programm muss explizit nur das 8bpp-BMP-Format abdecken! Sie haben das beim Image-Processing üblicher Weise verwendete Testbild in komprimierter und unkomprimierter Fassung zur Dokumentation erhalten.
- Berechnen Sie Kompressionsrate und Laufzeit Ihres Algorithmus'. Verifizieren Sie dessen Korrektheit, indem Sie das Ergebnis in einem Bildbetrachter (z.b. GIMP) öffnen.

2.3.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, mit welchen Sie Ihrem Assemblerprogramm die Inhalte einer eingelesenen Datei als Pointer übergeben können. Parsen Sie dazu die nötigen BMP-Header-Informationen, um das Offset der Daten in der Datei zu finden.
- Implementieren Sie in der Datei mit dem Assemblercode eine Funktion

  ```
  int compress(char *data, char *result, unsigned int result_size)
  ```

  welche einen Pointer auf die unkomprimierten Eingabedaten `data` übergeben bekommt und die nach BMP-Spezifikation lauflängenkodierten Daten in das Array `result` speichert. Der Parameter `result_size` gibt die maximale Größe des für `result` reservierten Speicherbereichs an. Der Rückgabewert soll im Erfolgsfall die Länge der kodierten Daten sein. Wenn die Funktion fehlschlägt (beispielsweise, weil das Ergebnisarray nicht groß genug gewählt wurde) soll -1 zurückgegeben werden. Speichern Sie das komprimierte Bild mit angepasstem BMP-Header in einer vom Benutzer wählbaren Datei mithilfe des Rahmenprogramms (d.h. in C, das Speichern muss nicht in Assembler erfolgen).

2.3.3 Bonusaufgabe

Implementieren Sie in der Datei mit dem Assemblercode eine Funktion

```
int decompress(char *data, char *result, unsigned int result_size)
```

welche einen Pointer auf die lauflängenkodierten Eingabedaten `data` übergeben bekommt und die unkomprimierten Daten in das Array `result` speichert. Der Parameter `result_size` gibt die maximale Größe des für `result` reservierten Speicherbereichs an. Der Rückgabewert soll im Erfolgsfall die Länge der unkomprimierten Daten sein. Wenn die Funktion fehlschlägt (beispielsweise, weil das Ergebnisarray nicht groß genug gewählt wurde) soll -1 zurückgegeben werden. Speichern Sie das unkomprimierte Bild mit angepasstem BMP-Header in einer vom Benutzer wählbaren Datei mithilfe des Rahmenprogramms (d.h. in C, das Speichern muss nicht in Assembler erfolgen).

2.4 Checkliste

Die folgende Liste soll Ihnen als Gedächtnisstütze beim Bearbeiten der Aufgaben dienen. Sollten Sie eine Reverse-Engineering-Aufgabe erhalten haben, sind diese Punkte für Sie weitestgehend hinfällig.

- Verwenden Sie keinen Inline-Assembler.
- I/O-Operationen dürfen grundsätzlich in C implementiert werden.
- Sie dürfen die Signatur der in Assembler zu implementierenden Funktion nur dann ändern, wenn Sie dies (in Ihrer Ausarbeitung) rechtfertigen können.
- Denken Sie daran, das Laufzeitverhalten Ihres Codes zu testen (Sichere Programmierung, Performanz).
- Verwenden Sie NEON/SIMD-Befehle, wenn möglich.
- Fügen Sie Ihrem fertigen Quelltext Anweisungen hinzu, wie das Projekt kompiliert werden kann.
- Eingabedateien, welche Sie generieren, um Ihre Implementierungen zu testen sollten mit abgegeben werden.
- Bonusaufgaben (sofern vorhanden) müssen nicht implementiert werden.