Aufgabe 1: Algorithm. Bildverarbeitung

Werkzeuge: *Microsoft Visual Studio 2008*

Zwischenstand: *20.04.2011, 13:00 Uhr*
Abgabetermin: *02.05.2011, 13:00 Uhr*

Bevor Sie beginnen

Stellen Sie eine Verbindung mit *\\nas2\MMS* her. Falls Sie über SFTP zugreifen, verbinden Sie sich mit *serv9.inf.tu-dresden.de* und wechseln Sie dann in das Verzeichnis */zbv/mms*. Legen Sie dort unter */1_Bildverarbeitung/* ein neues Unterverzeichnis an und benennen Sie dieses nach Ihrer Matrikelnummer. Legen Sie Ihre Ergebnisdateien in diesem Verzeichnis ab.

Die für diese Aufgabe benötigten Quelldateien finden Sie auf der Übungswebsite. Der Quellcode ist in C# geschrieben und basiert auf dem .Net3.5-Framework. Die Beispielanwendung ist eine Anwendung auf Basis von WPF[1], die mit XAML[2] spezifiziert wurde. Sie benötigen jedoch keine umfassenden Kenntnisse in den genannten Technologien, sondern lediglich ein algorithmisches Grundverständnis.

Die Anwendung besteht aus dem eigentlichen Hauptprogramm (**Window1.xaml** und **Window1.xaml.cs**) und einer Reihe weiterer Klassen (**ExerciseX.cs**). Am Hauptprogramm müssen Sie keine Änderungen vornehmen. Für Sie ist lediglich die entsprechende Klasse einer Teilübung (für Aufgabe 1.1 die Klasse **Exercise1.cs**, usw.) relevant – Sie müssen hier nur das markierte Codestück (*#region*) implementieren.

Allgemeine Hinweise

Sie sollen im Laufe der Übung die algorithmische Verarbeitung von Bilddaten erlernen. Dazu greifen Sie pixelweise (z.B. mit *GetPixel*) auf die Bilder zu. Zwar bietet das .Net Framework fertige Mechanismen zur Lösung der Aufgaben an, diese dürfen Sie jedoch *nicht* verwenden. Sie sollen die entsprechenden Algorithmen selbst implementieren, um den Umgang mit Pixeldaten zu erfahren. Dokumentieren Sie bitte ihren Quellcode, indem Sie in kurzen eigenen Worten noch einmal im Code selbst beschreiben, was der Algorithmus leistet, wie er arbeitet und wofür welche Variablen gewählt wurden. Dies hilft bei der Bewertung, Ihre Lösung nachzuvollziehen. Wenn Sie den Algorithmus zunächst in Pseudocode skizzieren, gibt dieser eine gute Vorlage für die Kommentare.

Pro Aufgabe können Sie jeweils ein Bild für die Abgabe auswählen, das durch Ihren Filter verändert werden soll. Es ist also nicht erforderlich, alle Bilder zu filtern.

Alle Ergebnisse müssen folgende Anforderungen erfüllen:

- Der Quellcode sollte so strukturiert und kommentiert sein, dass er für die Bewertung nachvollziehbar ist.
- *Alle verwendeten Bestandteile müssen entweder Public Domain sein oder die erforderlichen Nutzungsrechte müssen schriftlich vorliegen.*

[1] Windows Presentation Foundation
[2] Extensible Application Markup Language

Aufgabe 1.1

Warmup: Bildspiegelung

Eine der Basisoperationen in der Bildverarbeitung ist die einfache Spiegelung. Ihre Aufgabe ist es, ein vorgegebenes Bild *vertikal* zu spiegeln. Lesen Sie dazu die Pixeldaten einzeln aus und erzeugen Sie ein neues, gespiegeltes Bild, das Sie anschließend speichern (**1_sample_0X.jpg**). Abbildung 1 zeigt, wie es aussehen sollte:

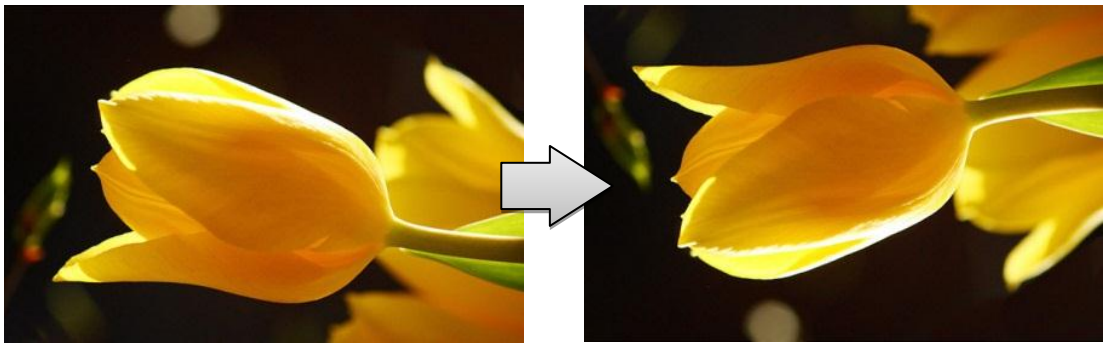

Abbildung 1: Vertikal gespiegeltes Bild

Quelldateien
- Exercise1.cs

Abgabedateien
- 1_sample_0X.jpg
- Exercise1.cs

Bewertung
20 P

Aufgabe 1.2

Farbverlust: Dithering mit dem Floyd-Steinberg-Algorithmus

Mittels Dithering können für den Betrachter Zwischentöne auf einem Ausgabemedium erzeugt werden, die von dem Medium selbst nicht unterstützt würden, z.B. bei GIF-Animationen. In dieser Aufgabe sollen Sie den Floyd-Steinberg-Algorithmus auf ein Graustufenbild anwenden, um eine Schwarz-Weiß-Repräsentation dieses Bildes zu erhalten. Farbbilder müssen Sie zunächst in ein Graustufenbild überführen. Speichern Sie das geditherte Bild anschließend für die Abgabe ab (**2_sample_0X.jpg**). Abbildung 2 zeigt, wie es aussehen sollte:

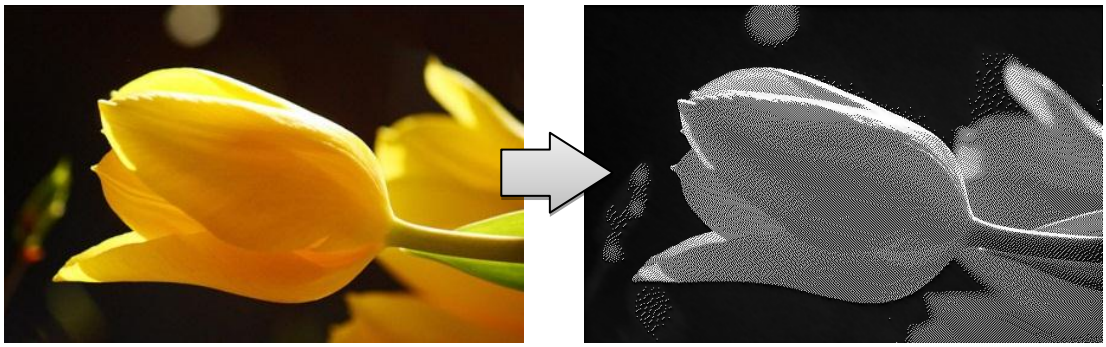

Abbildung 2: Auf zwei Farben (Schwarz/Weiß) gedithertes Bild

Quelldateien
- Exercise2.cs

Abgabedateien
- 2_sample_0X.jpg
- Exercise2.cs

Bewertung
35 P

Aufgabe 1.3

Tunnelblick: Automatisches Zuschneiden mit dem Sobelfilter

Der Sobelfilter erhält die Kanten in einem Bild und kann damit ideal zur Kantenerkennung eingesetzt werden. Ihre Aufgabe ist es, zunächst einen 2D-Sobelfilter zu implementieren, um die Kanten auf dem vorgegebenen Bild zu ermitteln. Recherchieren Sie die grundlegende Arbeitsweise des Algorithmus und implementieren Sie anschließend den Filter. Nutzen Sie dazu die vorgegebenen 3x3 Sobel-Filtermatrizen für die horizontale (S_x) sowie vertikale (S_y) Filterung. Vereinen Sie beide Filtereffekte und speichern Sie die Kanteninformationen als Graustufenbild ab (**3a_sample_0X.jpg**).

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}, \quad S = \sqrt{{S_x}^2 + {S_y}^2}$$

In einem zweiten Schritt sollen Sie dieses Kantenbild nutzen, um das Bild so rechteckig zuzuschneiden, dass der freie Platz um das dargestellte Objekt verschwindet. Da das Kantenbild Graustufen aufweist, müssen Sie zunächst noch einen geeigneten Grenzwert für die Bestimmung des neuen Rands festlegen. Nutzen Sie nun diese Informationen, um das *farbige Originalbild* zu beschneiden. Speichern Sie anschließend diesen Bildausschnitt ebenfalls ab (**3b_sample_0X.jpg**).

Lösungshinweis: Das Ergebnis der Filterung (S) ist nicht wieder ein einfacher Grauwert, sondern (je nach Bild und Filterposition) ein *Kantenwert* zwischen 0 und 1082. Je nach Anwendung können daraus verschiedene Bilder generiert werden. Für diese Übung soll der Kantenwert gleichmäßig auf den Grauwertbereich von 0 bis 255 abgebildet werden. Der Einfachheit halber empfiehlt es sich, hohe diagonale Kantenwerte (S > 1020) vorher abzuschneiden und auf den maximalen Kantenwert 1020 abzubilden. Hierbei kommt es nicht auf 100%ige Genauigkeit an. Abbildung 3 zeigt das Ergebnis des Filters am Tulpenbeispiel.

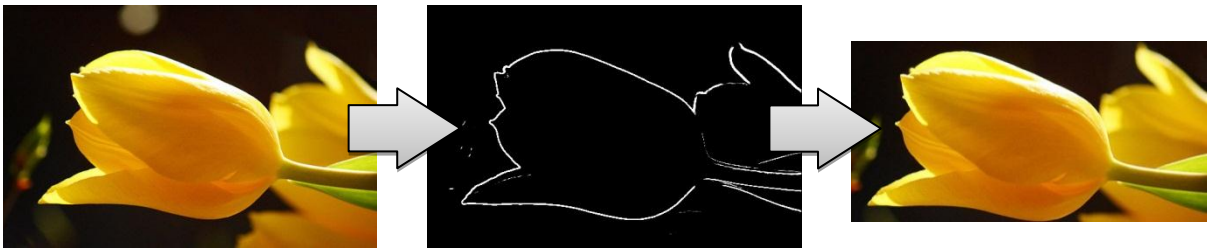

Abbildung 3: Mit Sobelfilter erstelltes Kantenbild und entsprechend zugeschnittenes Originalbild

Quelldateien
- Exercise3.cs
- Exercise3b.cs

Abgabedateien
- 3a_sample_0X.jpg
- 3b_sample_0X.jpg
- Exercise3.cs
- Exercise3b.cs

Bewertung

45 P

Zusatz: Um weitere Erfahrungen zu sammeln, können Sie versuchen, Ihren Algorithmus performant zu gestalten. *GetPixel* ist nämlich eine vergleichsweise langsame Funktion zum Zugriff auf die Bildinformationen. Als schnellere Alternative können Sie auf *LockBits* zurückgreifen und die Informationen direkt im Hauptspeicher durch Pointer (Code als *unsafe* markieren) verarbeiten. Diese Erweiterung hat keinen Einfluss auf die Bewertung und ist zur Vertiefung aus persönlichem Interesse gedacht.