Median Begriffserklärung

Der Median oder Zentralwert ist ein Mittelwert für Verteilungen in der Statistik. Der Median einer Auflistung von Zahlenwerten ist der Wert, der an der mittleren (zentralen) Stelle steht, wenn man die Werte der Größe nach sortiert. Beispielsweise ist für die Werte 4, 1, 37, 2, 1 die Zahl 2 der Median, nämlich die mittlere Zahl in 1, 1, 2, 4, 37.

Allgemein teilt ein Median einen Datensatz, eine Stichprobe oder eine Verteilung so in zwei (gleich große) Hälften, dass die Werte in der einen Hälfte nicht größer als der Medianwert sind, und in der anderen nicht kleiner.

Ablauf der Medianfilterung

Liest man in C# mittels „OpenFileDialog“ ein Bild in das Programm, erhält man ein „BitmapImage“.

Dieses „BitmapImage“ lässt sich mit Hilfe der Klassenfunktion „CopyPixels“ in ein Byte-Array umwandeln. An Stelle eines 2-dim arrays erhält man eine lineare Folge von Bytes, in denen abwechselnd die Luminiszenz, der Blau, der Grün und der Rot -Wert stehen. Die ersten 4 Bytes gehören zum Pixel (1,1) also das Pixel links ober, die Bytes 5 bis 8 zum Pixel (1,2) als dem 2.ten Pixel von links in der obersten Zeile usw.

Filtern kann man allerdings nicht einfach die gesamte Farbinformation die man aus den 4 Werten berechnen kann, weil der Farbraum keinen linearen Anstieg hat (wie man es bei einer Anordnung nach Regenbogenfarben vermuten könnte). zB ist der Farbwert 255 helles reines Rot, der Farbwert 256 ist dunkles Grün und hat gar keinen Rotanteil. Man muss also jeden Farbanteil für sich filtern.

Aus dem Byte Array muss man also die einzelnen Rot-, Grün- und Blau-Werte des Bildes auslesen. Nachdem alle einzelnen Farbwerte jedes Pixels in ein eigenes byte-Array gespeichert wurden, kann man mit der Filterung beginnen.

Zuerst wird die Grenze des Medianfilters ermittelt, also die Anzahl an Werten am Rand des Bildes, die nicht gefiltert werden können, da ihnen auf mindestens einer Seite mindestens ein zur Berechnung des Medians benötigter Nachbar fehlen.  
Die Grenze ermittelt man, indem man den Median halbiert. Z.B. sind die Grenzen für den 3 Punkt Median 1 und für den 5 Punkt Median lauten sei 2. Beim 5 Pkt. Median müssen alle Wert auf allen Seiten von zumindest zwei Werten umgeben sein.

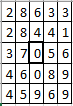
Aus all diesen Werten wird nun eine aufsteigend sortierte Liste erstellt und deren Mittelwert ausgelesen. Der Wert im byte-Array wird danach durch den berechneten Mittelwert ersetzt. Dieser Ablauf wird sowohl für das byte-Array mit allen Blauwerten als auch für die byte-Arrays mit den Grün- und Rotwerten angewendet.

Abbildung : Filterbereich 5 Pkt. Median

Die Arbeitsschritte müssen für jeden einzelnen Pixel bzw. Wert in den RGB-Byte-Arrays wiederholt werden, deswegen ist die Filterung sehr **rechen- und zeitaufwändig,** mit einer hohen Anzahl von Sortiervorgängen.

Zu beachten ist außerdem, dass man für die Berechnung der nachfolgenden Werte nicht die bereits bearbeiteten Werte sondern die originalen verwenden darf.

Wenn alle drei byte-Arrays gefiltert sind, kann man sie wieder zu einem großen zusammenfügen, indem man abwechselnd einen blauen, einen grünen und einen roten Wert in das große Array einfügt und diese wiederum in ein 2-dim Image zurückkopiert.

# Optimiermöglichkeiten:

***Parallelisierung:***Codeabschnitte einer Schleife, die eine sehr aufwendige Funktion funk berechnen können unter Umständen parallelisiert werden:

zB

for(i=1;i<1000;i++)  
{

x[i]=funk(y);

}

Kann auch in 4 Schleifen zerlegt werden   
for(i=1;i<250;i++)  
{

x[i]=funk(y);

}

for(i=250;i<500;i++)  
{

x[i]=funk(y);

}

usw.

Eine 4Kern CPU könnte in jedem ihrer Kerne eine der Schleifen erledigen und so 4mal so schnell alle x[i] berechnen.

Im Gegensatz dazu kann

for(i=1;i<1000;i++)  
{

x[i]=funk(x[i-1]);

}

Nicht parallelisiert werden, weil jedesmal der vorhergehende Wert benötigt wird, bevor also x[250] berechnet werden kann muss man x[249] berechnet haben usw.

Da immer nur ein rechteckiger Teilbereich des Bildes bearbeitet werden muss, um ein gefiltertes Pixel im Zentrum zu berechnen, und dieser Teilbereich keine Informationen seiner Nachbarrechtecke benötigt lässt sich die Berechnung sehr gut parallelisieren. Üblicherweise erkennen moderne Compiler durch Analyse des Quellcodes der Funktionen bzw Methoden ob dies möglich ist und verteilen bei entsprechender OS und Compiler-Einstellung die Rechenlast auf mehrere Prozessoren.

***Vorsortierung berücksichtigen***

Eine weitere Idee ist folgende:  
Annahme man verwendet einen Medianfilter der Größe 9x9. Dann müssen für die Berechnung des Pixels in der Mitte (an der Stelle 5,5) 81 Werte sortiert werden. Für das Pixel an der Stelle 5,6 wieder usw für alle Pixel im Bild. Für ein 1 Bild in XGA Auflösung (1024\*768) werden dafür schon ca 15 Milliarden (81\*81\*1024\*768\*3) Rechenschritte benötigt, bei einem 15\*15 sind es schon über 120 Milliarden usw. . Es spielt natrülich eine große Rolle ob der Sortieralgorithmus von quadratischer Ordnung ist oder nur mit zB log(anzahl)\*anzahl eingeht, wie ein quicksort es machen würde.

Wenn man ein Teilrechteck gefiltert hat, hat man alle Werte dieses Bereich bereits der Größe nach sortiert, beim nächsten Pixel kommt rechts eine Reihe von Werten dazu, während links eine Reihe wegfällt. Mit einem schlauen Algorithmus könnte man sich das Sortieren der bereits berücksichtigen Pixel sparen und nur die links nicht mehr benötigten entfernen und die rechts neu hinzugekommen an der richtigen Stelle einfügen. Diese Operation ist von niedrigerer Ordnung als das Sortieren, allerdings verliert man dadurch auch die Möglichkeit des Parallelisierens, weil ein Schleifendurchgang auf Werte des vorhergehenden Durchgangs benötigt.

**Verteilung einzelner Bildausschnitte auf mehrere Rechner**

Umso höher der ausgewählte Medianfilter desto länger dauert die Berechnung. Um den Zeitaufwand zu verringern, kann man das Bild aufteilen und von verschiedenen Rechnern bearbeiten lassen.

Dazu dividiert man das Byte-Array des Bildes durch die Anzahl der verfügbaren Clients und erhält somit gleich große „Puzzleteile“. Die Abschnitte müssen mit einer Positionsnummer versehen werden, um sie nach der Berechnung wieder richtig zusammensetzen zu können.

Die Clients berechnen ihren Teil dann einfach wie ein eigenes Bild und senden das Ergebnis zurück an den Hauptrechner.

Teilt man das Bild in genau gleich große Teile, können die neu entstandenen Grenzen nicht gefiltert werden, da jeder Client die „Grenzwerte“, also die Werte ohne genügend Nachbarn nicht berechnen kann. Muss das verhindert werden, kann man jedes Puzzleteil um die fehlenden Nachbarn vergrößern. Dann kann jeder Client alle ihm eigentlich zugewiesenen Pixel bearbeiten. Beim Zusammenfügen der einzelnen Stücke müssen die addierten Nachbarn aber wieder entfernt werden.

Die Verteilung der Arbeit auf verschiedene Rechner oder Prozessorkerne macht jedoch nicht mit unbegrenzt vielen Stellen Sinn. Sind zu viele Stellen beteiligt, kann das Bild in so kleine Happen zerleget werden, dass die Zeit für die Datenübertragung und der Overhead durch das Zerteilen und Zusammenfügen den Zeitgewinn durch das verteilte Filtern aufhebt.

Arbeitseinteilung:  
Server:

* kennt Anzahl der Clients die Arbeiten übernehmen können
* besitzt GUI (diese könnet aber auch auf einem Client sein)
* nimmt von der GUI Filterauftrag entgegen: Bild, Filtergröße
* verteilt Aufträge an die Clients (Bild, Bildausschnitt, Filtergröße)
* erhält gefilterte Teilbereiche und setzt sie zusammen
* zeigt Ergebnis in der GUI

Client:

* Wartet auf Aufträge (per polling)
* Erhält Infos vom Server (Bild, Filtergröße)
* Filtert seinen Bereich
* Sendet Ergebnis an den Server

Bei den Aufträgen gibt es mehrere Varianten:

1. Jeder Klient erhält das ganze Bild, zerlegt es, bearbeitet seinen Teilbereich und sendet das Ergebnis an den Server.
2. Der Server zerlegt das Bild und sendet nur den Teilbereich an den Klient, der es bearbeitet.

Welche der Methoden die bessere ist, hängt von der Leistungsfähigkeit des Servers (soll er das Zerlegen machen), des Netzwerkes (bei langsamen ist es besser nur Teilbereiche zu übertragen) und der Art des Bildes ab (jpegs sind optimal komprimiert und eignen sich für die Übertragung besser als raw Files).  
Wenn jeder Klient das gesamte Bild erhält wird viel Umwandlungsarbeit mehrfach erledigt (copypixels, bytearray erstellen, in RGB-Matrix zerlegen).

## Datenübertragung zu den Clients

Am günstigsten erscheint ein Senden über sockets (siehe Ausarbeitung 9 und 24) als Datagramm.

Das zu bearbeitende Bild könnte aber auch auf einem freigegeben Netzlaufwerk liegen.  
Der Server könnte den Clients sogar schon im voraus, wenn der Benutzer in der GUI das Bild gewählt hat, den Clients den Auftrag erteilen das Bild einzulesen und als bytearray vorzubereiten. Beim starten des Filtervorganges aus der GUI müssen nur noch die Filtergröße und Koordinaten der Eckpunkte des Bildausschnittes an die Clients gesendet werden, so dass diese gleich mit dem Filtern beginnen können.