Echtzeit-Röntgenbildverarbeitung mit Standardhardware

Martin Mack[1], Dietrich Paulus[1], Joachim Hornegger[2], Stefan Böhm[3] und Adam Galant[4]

[1]Institut für Computervisualistik, Universität Koblenz-Landau
[2]Lehrstuhl für Mustererkennung, Universität Erlangen-Nürnberg
[3] AX, Siemens Medical Solutions, Forchheim
[4] AX, Siemens Medical Solutions, Hoffman Estates, IL, USA

Zusammenfassung. In modernen Röntgensystemen unterliegen die verwendeten Bildverarbeitungsalgorithmen strengen Laufzeitbedingungen. Der hier vorgestellte Ansatz zeigt, dass sich Standardgrafikkarten zur Realisierung dieser Algorithmen eignen. Für die Implementierung der gesamten Bildverarbeitungsalgorithmen einer Kardiologieanlage wird der DirectX 9 Development Kit verwendet. Die experimentelle Evaluierung erfolgt mit realen Bilddaten, und die resultierende Bildqualität wird mit den Ergebnissen kommerziell verfügbarer Systeme abgeglichen. Die gemessenen Rechenzeiten für die gesamte Bildkette sowie die erzielte Bildqualität belegen die Konkurrenzfähigkeit des gewählten Ansatzes. Aufgrund der Ergebnisse ist zukünftig damit zu rechnen, dass in vielen Systemen auf teure und aufwändig zu pflegende Spezialhardware verzichtet werden kann.

1 Einleitung

Kardiologieanlagen akquirieren bis zu 30 Bilder pro Sekunde mit einer Auflösung von 1024×1024 Bildpunkten. Jeder Bildpunkt wird mit bis zu 16 Bit quantisiert [1]. Folglich müssen bis zu 60 MB Bilddaten pro Sekunde vorverarbeitet, visualisiert und gespeichert werden. Die Vorverarbeitung besteht aus bis zu 20 einzelnen Algorithmen. In allen kommerziell verfügbaren Systemen wird heute für diese Aufgabe Spezialhardware eingesetzt, was mit einem hohen Entwicklungs- und Kostenaufwand verbunden ist.

Bisher konnten keine Standardgrafikkarten für die Bildverarbeitung im medizinischen Umfeld eingesetzt werden, da die verfügbare Bit-Tiefe, nämlich 8 Bit pro Farbkanal, die notwendige Bildqualität nicht sicherstellen konnte. Die jüngste Generation handelsüblicher Grafikkarten verfügt nun über eine Genauigkeit von 16 Bit pro Farbkanal und ermöglicht damit erstmals deren Einsatz für die Echtzeitbildverarbeitung im medizinischen Umfeld. Dieser Beitrag stellt die Bildkette vor, wie sie in Röntgenanlagen eingesetzt wird, und zeigt, wie diese auf einer handelsüblichen Grafikkarte realisiert werden kann und welche Laufzeiten erreicht werden.

2 Übersicht über die Bildkette

Die realisierte Bildkette wird in der Form von allen großen Geräteherstellern für die digitale Subtraktionsangiografie eingesetzt [1]. Eine Standardisierung der Algorithmen ist insofern erfolgt, als die Parametrierung bereits Bestandteil der DICOM-Header in der Bilddatenbank ist, die unabhängig vom Hersteller interpretiert werden können. Das Eingabebild der Bildkette ist das aktuell zu filternde Röntgenbild, wie es vom Detektor geliefert wird. Der erste Vorverarbeitungsalgorithmus führt eine *Multiskalenzerlegung* durch, bei der einzelne Frequenzbänder unterschiedlich gewichtet aufaddiert werden. Der zweite Algorithmus der Bildkette ist die *Grauwertfensterung*. Diese skaliert die Grauwerte so, dass der Kontrast im Bild für das perzeptive System des Menschen optimiert wird. Im dritten Schritt können mit der *Zoom-Funktion* beliebige Bereiche des Bildes vergrößert dargestellt werden. Mittels der *Pan-Funktion* können diese Bereiche über das Gesamtbild beliebig verschoben werden. Die finale *Kollimator-Funktion* erlaubt es, unwichtige Bereiche am Bildrand auszublenden.

3 Implementierung

Die Implementierung der oben skizzierten Algorithmen erfolgt auf dem Grafikkartenprozessor über Pixel Shader Programme. Ein Pixel Shader Programm legt am Ende der Rendering Pipeline fest, wie die einzelnen Bildpunkte der Zieloberfläche einzufärben sind. Die Zieloberfläche ist dabei entweder die Oberfläche einer Textur oder die Bildschirmoberfläche. Für jeden zu berechnendem Bildpunkt wird das Pixel Shader Programm einmal durchlaufen. Da die Pixel Shader Programme als Teil der 3D-Rendering Pipeline ausgeführt werden, muss die Zieloberfläche auf ein zur Bildschirmoberfläche paralleles Rechteck abgebildet werden, das über vier Eckpunkte definiert wird. Die Eingabe-Bilder müssen in Texturen der selben Größe und ausreichender Bit-Tiefe pro Kanal gespeichert werden. Die Farbwerte des Ausgabebildpunktes werden mittels arithmetischer Operationen aus Textur- und Vertex-Farbwerten berechnet. Die Architektur der Grafikkarte beschränkt die Anzahl der arithmetischen Operationen und der Texturzugriffe pro Pixel Shader Programm. Komplexe Algorithmen müssen daher auf mehrere Pixel Shader Programme aufgeteilt werden. Zur Speicherung der Zwischenergebnisse werden Texturen verwendet.

3.1 DirectX 9, Grafikkartenzustände und Texturen

Die Implementierung unter Direct3D gliedert sich in vier Abschnitte [2]: die Initialisierung der Direct3D Umgebung, das Setzen der Grafikkartenzustände, die Initialisierung der Texturen und der Pixel Shader Programme und die Schleife für das Rendering. Mittels des Direct3D Objektes wird ein Direct3D Device erzeugt und geeignet initialisiert. Im Direct3D Device steckt die komplette Funktionalität zur Ansteuerung der Grafikkarte. Wenn das Direct3D Device verfügbar ist, werden die Grafikkartenzustände gesetzt, die benötigten Quell- und Zieltexturen, Transformationsmatrizen und der Vertex-Puffer initialisiert sowie die HLSL

Pixel Shader Dateien geladen und kompiliert. Die Schleife für das Rendering wird nach der Initialisierung bis zum Ende der Applikation ausgeführt.

Die Grafikkartenzustände sind in „Render States“ und „Sampler States“ unterteilt. Die „Render States“ legen das Verhalten des Direct3D Rasterisierungsmoduls fest. Die „Sampler States“ legen fest, wie Texturen abgetastet und wie ihre Koordinaten behandelt werden.

Zwei Arten von Texturen werden werden für die Implementierung der Algorithmen benötigt: Quelltexturen und Zieltexturen. Quelltexturen dienen als Eingabe für die Pixel Shader Berechnungen. Zieltexturen stellen den Speicher für die Ausgabe der Pixel Shader Programme. Für die Zieltexturen wird ein 32 Bit Format bestehend aus zwei vorzeichenlosen 16 Bit Kanälen verwendet. Für die Quelltexturen wird ein vorzeichenloses 16 Bit Einkanal Format verwendet. Die Zieltexturen werden als „Rendertarget“ erzeugt. Sie werden im Grafikkartenspeicher angelegt. Ein Zugriff auf diese Texturen aus dem Systemspeicher ist sehr langsam. Beispielsweise dauert unter DirectX 9 ein Zugriff auf eine 1024×1024 Textur mit 32 Bit Quantisierung 60 ms. Daher wird der Textur, in der das eingegebene Röntgenbild gespeichert wird, der Verwendungsparameter „Dynamic“ zugewiesen und wird im AGP-Port-Speicher angelegt. Damit wird sichergestellt, dass vom Systemspeicher aus effizient auf die Textur zugegriffen werden kann und dass die Textur zur Bearbeitung im Pixel Shader schnell in den Grafikkartenspeicher transferiert werden kann.

3.2 HLSL Pixel Shader Programme

Pixel Shader Programme, die in der Microsoft Hochsprache High Level Shader Language (HLSL) geschrieben sind, werden mit Hilfe des im DirectX 9 SDK enthaltenen Compiler in Maschinencode übersetzt. Ein HLSL Programm besteht aus globalen Variablen, einer Funktionsdeklaration und dem Funktionsrumpf. Globale Variablen können vor Beginn eines Renderprozesses mit Werten belegt werden. So können Funktionsparameter zur Laufzeit aus der Applikation heraus verändert werden. Pixel Shader Programme mit HLSL Syntax können nur unter DirectX verwendet werden.

3.3 Implementierung des Multiskalen-Filter

Das implementierte Multiskalen-Filter hat sieben Auflösungsstufen. Die horizontale und vertikale Auflösung wird in jeder Stufe halbiert. Eine Auflösungsstufe wird in die nächst niedrigere Auflösungsstufe nach Anwendung eines Tiefpassfilters transformiert. Für jedes Filter werden dazu zwei Pixel Shader Programme ausgeführt, eines für die horizontale und eines für die vertikale Filterung. Bei der horizontalen Filterung wird das Ergebnis in eine Textur mit halber horizontaler Auflösung gespeichert. Entsprechend wird beim vertikalen Filtern verfahren. Wenn alle sieben Auflösungsstufen erzeugt sind, werden diese auf Texturen doppelter Auflösung skaliert. Dabei wird linear interpoliert, um die Grauwerte der fehlenden Bildpunkte zu berechnen. Die skalierten Bilder werden dann, um die

Frequenzbänder zu extrahieren, von der entsprechenden Auflösungsstufe abgezogen. Hierbei ist zu beachten, dass negative Werte geklippt werden. Positive Werte werden dabei in Kanal eins und negative Werte ohne Vorzeichen in Kanal zwei gespeichert. Vor der Benutztung dieser Textur in folgenden Berechnungen wird innerhalb des jeweiligen Pixel Shader Programms Kanal zwei von Kanal eins subtrahiert. Welches Filter bei der Abbildung einer Textur auf eine größere oder kleinere Auflösung angewendet wird, bestimmt der eingestellte Grafikkartenzustand. Die unterste Auflösungsstufe wird mit einem Faktor kleiner als eins gewichtet, um das Rauschen kantenerhaltend zu unterdrücken. Ausgehend von dieser Auflösungsstufe wird zuerst die Auflösung ihrer Textur verdoppelt und dann die Textur mit der extrahierten Frequenz der nächst höheren Auflösungsstufe gewichtet addiert. Die Auflösung der Ergebnistextur wird erneut verdoppelt und die Textur mit der extrahierten Frequenz der nächst höheren Auflösungsstufe gewichtet aufaddiert, bis die ursprüngliche Auflösung erreicht ist.

3.4 Implementierung der Subtraktion und der Grauwertfensterung

Die Subtraktion, die beiden Algorithmen Algorithmen zur Grauwertfensterung und das Mischen der subtrahierten Farbwerte mit den Originalen können in einem Pixel Shader Programm implementiert werden. Zuerst wird die Subtraktion berechnet, dann die originalen Farbwerte und die subtrahierten Farbwerte mit jeweils unterschiedliche einstellbaren Parametern mit der Funktion zur Grauwertfensterung bearbeitet und beide Bilder gewichtet überlagert. Die Gewichtung ist dabei parametrierbar.

3.5 Zoom, Pan und Kollimator

Die beiden Funktionen Zoom und Pan nutzen die frei veränderbaren Texturkoordinaten in Direct3D. Die Texturkoordinaten sind an die Eckpunkte des Rechteckes gebunden, auf das die Textur abgebildet wird. Verändert man die Texturkoordinaten der vier Eckpunkte proportional so wird die Textur vergrößert, verkleinert oder verschoben. Im letzten Pixel Shader Programm wird der Kollimator implementiert. Eine bei der Initialisierung erzeugte Kollimator Textur enthält an den zu verdeckenden Stellen den Wert Null und an den restlichen Stellen, den Wert Eins. Im Pixel Shader Programm wird der Grauwert des gefilterten Röntgenbildes mit dem Wert der Kollimator Textur multipliziert. So werden die Stellen, die vom Kollimator verdeckt sein sollen schwarz und die anderen unverändert dargestellt.

4 Experimentelle Evaluierung und Diskussion

Die beschriebene Bildkette ist in Pixel Shader Programmen realisiert und die Laufzeitmessungen haben ergeben, dass die zeitaufwändigsten Operationen Texturzugriffe und Flusssteuerungsanweisungen sind. Die Bildkette benötigt pro

Abb. 1. Ergebnisse der Bildvorverarbeitung: Originalaufnahme (links), die Resultate der Multiskalenfilterung mit unterschiedlicher Parametrierung (beide Aufnahmen in der Mitte), Grauwertfensterung (links)

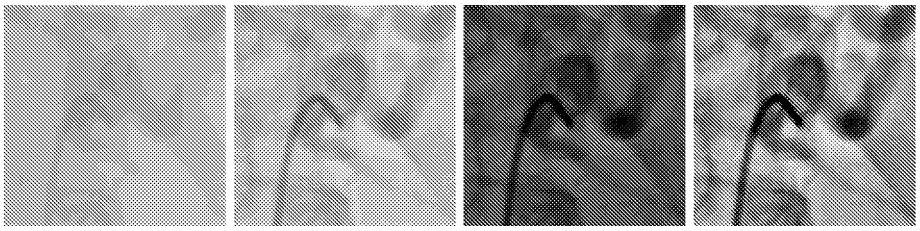

Bild 30 ms Rechenzeit auf einem System mit einer ATI Radeon 9700 Pro Grafikkarte, einem Athlon XP 2400+ Prozessor und 512MB PC 2700 RAM. Im ersten Bild der Abb. 1 ist ein Ausschnitt des originalen Röntgenbildes zu sehen; im zweiten wurde dieser mit dem Multiskalen-Filter bearbeitet. Die vier höchsten Frequenzbänder sind mit dem Faktor 3,6 gewichtet. Im dritten Bild der Abb. 1 sind zusätzlich die niedrigsten Restfrequenzen mit dem Faktor Null gewichtet, um eine kantenerhaltende Glättung zu implementieren und im vierten Bild werden die Grauwerte zusätzlich durch die Grauwertfensterung optimiert.

5 Zusammenfassung

In diesem Betrag wurde die Implementierung der Bildkette einer Röntgenanlage auf einer Standardgrafikkarte beschrieben. Die gemessenen Laufzeiten belegen, dass die gewählte Technologie im Stande ist, die klinischen Anforderungen zu erfüllen. Es ist zu erwarten, dass in zukünftigen Bildsystemen Grafikkarten teure Spezialhardware ablösen werden und dass Standardgrafikkarten Einzug in Befundungsstationen halten werden, deren Leistungsfähigkeit heutzutage noch für viele Anwendungen unzureichend ist.

Literaturverzeichnis

1. Morneburg, H. (Herausgeber), Bildgebende Systeme für die medizinische Diagnostik, Siemens Publicis MCD Verlag, Erlangen, 1995.
2. Walsh, P; Perez, A.: Advanced 3D Game Programming with DirectX 9.0. Wordware Publishing, Plano, 2003.