

# Antialiasing-Techniken für Linien

Eduard Angold

Student der Informatik (1. Sem)

Dusiplatz 1a

21035 Hamburg

+49 4073590250

5angold@informatik.uni-hamburg.de

Sebastian Ulm

Student der Informatik (1. Sem)

Wördemannsweg 111d

22527 Hamburg

+49 406792876-0

5ulm@informatik.uni-hamburg.de

## ABSTRACT

Antialiasing findet zunehmend Verbreitung in der aktuellen Computertechnik. Auch wenn es viele bewährte Algorithmen gibt, haben wir uns dazu entschlossen, für das Zeichnen von Linien, einen angepassten Algorithmus zu entwickeln. Dabei bedienen wir uns der Vektormathematik und der Technik des Supersamplings.

## General Terms

Algorithms, Performance, Antialiasing, Lines

## 1. ANTIALIASING

### 1.1. Motivation

Da das Arbeiten mit Computern in der heutigen Zeit maßgeblich über grafische Ein- und Ausgabegeräte erfolgt, verlangen diese Geräte neue Bibliotheken, die oftmals auf Vektor-orientierten Formaten basieren. Beispiele sind hier CAD-Anwendungen, Computerspiele und natürlich auch Gebrauchssoftware. Gleichzeitig verfügen moderne Rechner über ausreichend Ressourcen, um nicht nur den notwendigen Anforderungen gerecht zu werden, sondern auch ergonomischen und nicht zuletzt auch ästhetischen Gesichtspunkten Rechnung zu tragen. Außerdem kann durch klare Linien die Kantenerkennung der Retina unterstützt, oder der Lesefluss verbessert werden.

### 1.2. Antialiasing

Antialiasing stellt ein technisches Mittel dar, mit dem die Qualität von grafischen Ausgaben stark gesteigert werden kann ohne dass der Aufwand den Nutzen übersteigt.

Die interne Darstellung von Bildern im Computer erfolgt meist als sogenanntes Bitmap, also einer Menge aus Pixeln in einem Koordinatensystem. Für viele darzustellende Objekte ist diese diskrete Form aber ungeeignet. Beispiele sind hier einfache Objekte wie Kreise oder Kurven oder komplizierte wie beispielsweise Schriftzeichen.

Klassische Ansätze zur Qualitätsverbesserung sind das Erhöhen der Auflösung, sprich mehr Pixel pro darzustellendes Objekt. Antialiasing ist hierbei eine moderne Möglichkeit mit vorhandenen Ressourcen die Qualität signifikant zu erhöhen. Hierbei wird, nur während der Berechnungsphase, ein Pixel in Untereinheiten, sogenannte Sub-Pixel, aufgeteilt. Dies kommt einer lokalen Erhöhung der Auflösung gleich. Weiterhin kann der durch die Darstellungsfunktion berechnete Farbwert eines Pixels seine Nachbarpixel beeinflussen. Dies ist auch eine Näherung an das Darstellungsprinzip von CRT-Monitoren die einen runden Farbpunkt statt einem rechteckigen Pixel darstellen.

## 1.3. Technische Voraussetzungen

Die technische Umsetzung basiert auf einem Framebuffer und Funktionen zum Lesen und Schreiben von Pixeln in 32 Bit Farbtiefe unter Verwendung eines Alphakanals (Transparenzwert). Außerdem kommt eine Bibliothek für allgemeine Vektorrechnung zum Einsatz.

Für die Entwicklung verwenden wir die Programmiersprache C++ mit einem Compiler der Gnu Compiler Collection, Version 3.4.4, getestet. Für den vollständigen Quellcode aller Beispiele, die unter der GPL veröffentlicht sind, kontaktieren sie bitte die Autoren.

## 2. TECHNIKEN

Die folgenden drei Algorithmen stellen eine Auswahl dar, bauen aber auch aufeinander auf. Vektorbasiertes Antialiasing stellt einen vollständigen Algorithmus auf Basis der Vektorrechnung dar, der qualitativ hochwertige Ergebnisse liefert. Er berechnet nahezu unabhängig von einem Pixelrastrer was eine hohe Skalierbarkeit zur Folge hat.

Ein weiterer Ansatz ist die Methode des Supersampling bei der der klassische Weg der höheren Auflösung gegangen wurde. Am Ende der Berechnung wird die Auflösung auf die Anforderung gesenkt.

Der dritte Algorithmus ist eine optimierte Version des Vektorbasierten Antialiasing, bei dem das Augenmerk auf die Geschwindigkeit gelegt wurde.

### 2.1. Vektorbasierendes Antialiasing

Für das Vektorbasierende Antialiasing wurde eine Funktion entwickelt die fünf Parameter benötigt. Die ersten zwei Parameter sind der Start- und Endpunkt der Linie, der dritte Parameter ist der Kern-Parameter, der vierte der Korona-Parameter und als fünften die Kernfarbe. Die Angabe der Geometrie der Linie erfolgt über zwei Vektoren, während der Kernbereich und die Korona als Fließkommazahl angegeben werden.

Zuerst wird eine Geradengleichung mithilfe von Vektoren gebildet, dann wird mit der Parameterform weiter gerechnet.

Um die Parameterform für die Gerade zu bilden benötigen wir einen Stützvektor ( $\vec{s}$ ) und einen Aufpunktvektor ( $\vec{a}$ ). Mithilfe dieser Gleichung können wir jeden Punkt auf der Linie bestimmen.

Parameterform:  $r \cdot \vec{a} + \vec{s} = \vec{c}$

Um den Stützvektor ( $\vec{s}$ ) zu berechnen müssen wir vom Startpunkt den Ursprungspunkt subtrahieren.

$$P_1(x_1, y_1)$$

$$P_U(x_U, y_U)$$

$$\vec{s} = P_1 - P_U = \begin{pmatrix} x_1 - x_U \\ y_1 - y_U \end{pmatrix}$$

$$P_U(0,0)$$

$$\vec{s} = P_1 - P_U = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

Um den Richtungsvektor zu bestimmen, ist es notwendig, den Startpunkt vom Endpunkt zu subtrahieren.

$$P_1(x_1, y_1)$$

$$P_2(x_2, y_2)$$

$$\vec{a} = P_2 - P_1 = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$$

Da wir nun unsere Gerade bestimmt haben können wir sie zeichnen. Dazu bestimmen wir zu jedem Pixel<sub>A</sub> einen weiteren Pixel<sub>B</sub> auf der Linie der mit dem Pixel<sub>A</sub> einen Vektor bildet, und prüfen, welchen Wert unser Parameter r bei diesem Punkt hat. Ist der Wert zwischen null und eins ( $0 \leq r \leq 1$ ), so wird der Abstand zur Gerade berechnet und mit dem Wert für den Kern (Kern-Parameter) verglichen. Wenn die Entfernung kleiner oder gleich dem Kern-Wert ist, so wird der Pixel in voller Farbe gezeichnet.

Ist er größer als der Kern-Wert und kleiner als der Korona-Wert so wird er je nach Abstand zur Linie mit einem Alpha-Wert (Transparenzwert) gezeichnet (je weiter weg desto durchsichtiger wird der Pixel). Der dritte Fall ist wenn der Abstand zur Linie größer ist als der Korona-Wert, in diesem Fall bekommt der Pixel den Farbwert der Hintergrundfarbe.

Ist der Parameterwert r kleiner null oder größer eins, so wird er wenn er kleiner ist auf null gesetzt und wenn er größer ist auf eins. Dies hat zur Folge, dass die Pixel, die nahe dem Liniende liegen, um das Ende einen Halbkreis bilden.



Bild 1: Vergrößertes Bild der Pixeldaten.

## 2.2. Supersampling

Die Technik des Supersampling basiert im wesentlichen darauf, dass die zu zeichnende Menge an Linien in ein Speicherarray mit einer größeren, im Normalfall vielfachen Auflösung geschrieben wird. Durch den erhöhten Speicherbedarf kommen Vielfache größer als vier nur bei Spezialanwendungen, beispielsweise dem Raytracing, zum Einsatz. Dort müssen beispielsweise nicht alle Pixeldaten gleichzeitig vorliegen.

Das größere Speicherarray wird, nach Abschluss aller Grafikoperationen auf die Dimensionen des Framebuffers herunter skaliert und in diesen geladen. Diese Technik

funktioniert analog zum DoubleBuffering. Moderne Grafikkarten verfügen über Mechanismen, die diese Technik nativ unterstützen. Zum Vergleich haben wir einen Samplealgorithmus ausgewählt und umgesetzt.

Hierzu implementierten wir einen normalen Liniensalgorithmus der als weiteren Parameter nur die Liniendicke als Eingabeparameter benötigt. Mit diesem Algorithmus schreiben wir nun eine oder mehrere Linien mit der doppelten Stärke in ein Speicherarray.

Nach dem Ende der Zeichenoperationen bestimmen wir jeweils den Durchschnittswert von vier Pixeln und schreiben diesen in den Framebuffer.

Außer der normalen Durchschnittswertberechnung gibt es eine Vielzahl an anderen möglichen Filteralgorithmen. Hierzu sei auf [2] verwiesen. Für die Grundlagen von gerasterten Linien (und anderen grafischen Primitiven) ist das Standardwerk von Bresenham [3] unverzichtbar.



Bild 2 Supersampling

## 2.3. Optimiertes vektorbasierendes Antialiasing

Um den in 2.1 vorgestellten Algorithmus praxistauglich zu machen, muss die Geschwindigkeit erhöht werden. Die Qualität soll dabei möglichst gleich bleiben. Dazu wurden mehrere Optimierungsschritte getätigt. Der wichtigste ist, nur den Bereich der nach abgeschlossener Grafikoperationen auch verändert sein wird, zu berechnen. Dadurch ist sichergestellt, dass nahezu nur die Pixel in die Berechnung einbezogen werden, die sich auch potentiell verändern werden. Dazu wurde der für 2.2 entwickelte Algorithmus zum Linienzeichnen benutzt.

Dieser wurde von Fließkommazahlen auf Festkommazahlen umgewandelt, was marginale Geschwindigkeitsvorteile brachte.

Die zweite Optimierung ergibt sich aus einer Änderung des Subpixel-Farbmodells. Hierzu wurde angenommen, dass jeder Pixel nicht rechteckig ist, sondern einem Kreis entspricht. Das kommt dem Beleuchtungsmodell von CRT-Monitoren sehr nahe und bedeutet auch auf TFT-Monitoren keinen starken Qualitätsverlust.

Die Informationen zu Pixel und deren Darstellung auf CRT-Monitoren entnehmen wir [4].

Je nach Anteil der von der Linie geschnittenen Fläche wird der Pixel mit Farbe gefüllt. Da es sehr langsam wäre, für jeden zu zeichnenden Pixel jeweils die Fläche zu berechnen, erfolgt dies beim Programmstart und wird in einem Array gespeichert.

Als weitere Optimierung wurde der Koronabereich auf einen Pixel eingeschränkt, was aber für die meisten Anwendungen ausreicht.



Bild 3 Schnelles Vektorbasierendes Antialiasing

### 3. QUALITÄTSVERGLEICH

Der Qualitätsvergleich erfolgte rein optisch und ist mit bloßem Auge zu erkennen. Genauere Analysen können durch die vergrößerten Bilder erfolgen.

Die unserer Meinung nach beste Qualität erreicht das schnelle vektorbasierte Antialiasing was auf das verbesserte Pixelmodell zurück zuführen ist. Der Supersampling Algorithmus stellt qualitativ die schlechteste Lösung dar. Dies liegt aber am verwendeten Samplingalgorithmus. Normales lineares Sampling stellte sich also als unzureichend für unseren Verwendungszweck heraus.

Alle die von uns entwickelten Algorithmen zeigen bei verschiedenen Gamma-Einstellung noch makroskopische Aliasingeffekte. Für eine weitere qualitative Verbesserung ist also ein Ausgleich der Gammawerte erforderlich.

### 4. GESCHWINDIGKEITSVERGLEICH

Um einen praxisnahen Vergleich aufzustellen ermittelten wir die Rastergeschwindigkeit empirisch.

Dafür konstruierten wir folgendes Mess-Szenario: Ziel aller Schreiboperationen war ein Speicherfeld. Zum Einsatz kam ein Standard PC mit einer 2 GHz getakteten AMD Athlon CPU.

Der jeweilige zu testende Algorithmus schrieb 300 mal in das Speicherfeld und die gemessenen Zeiten wurden gemittelt.

Linienanzahl	Vektor	Supersample	optimiert
1	147	77	0,32
30	4296	81	10
1000	-	99	346

Alle Angaben sind in Millisekunden.

Die ermittelten Werte zeigen, dass der normale Vektorbasierender Algorithmus für die Praxis untauglich ist.

Supersampling zeigte im Versuch die kleinste Steigerungsrate, was daran liegt, dass erst nachdem alle Linien gezeichnet wurden, das eigentliche Supersampling erfolgt. Für grössere Linienmengen oder komplexe Strukturen ist dieser Algorithmus geeigneter.

Der optimierte Algorithmus liegt allerdings bei niedrigen Linienanzahlen vorn und zeigt als Ergebnis durchaus praxistaugliche Werte.

## 5. ZUSAMMENFASSUNG

### 5.1. Ergebnis

Wir sind mit dem derzeitigen Entwicklungsstand sehr zufrieden und halten das optimierte Vektorbasierende Antialiasing für einen Algorithmus der sich mit Lösungen der Computerindustrie durchaus messen kann.

### 5.2. Ausblick

Die Technik des Antialiasing kann durchaus als gut erforschetes Feld betrachtet werden. Trotzdem gibt es auch hier weiterhin potential, da Computer erst in den letzten 5 Jahren über ausreichend Leistung verfügen, um diese Algorithmen einer breiten Masse zur Verfügung zu stellen. Da die nötige Grundlagenforschung bereits erfolgt ist, wird es in den folgenden Jahren um direkte Anwendungen gehen. Auch die Unterstützung durch Hardware kann als sich entwickelndes Forschungsfeld betrachtet werden.

Dabei darf natürlich auch trotz wachsender Rechnerkapazität die Performance nicht außer Acht gelassen werden. Beispielsweise bietet der Algorithmus in 2.3 noch weiteres Optimierungspotential sowohl durch Anpassung an bestehende Hardware als auch den Algorithmus selbst betreffend.

## 6. QUELLEN

- [1] Franlin C. Crow *The aliasing problem in computer-generated shaded Images.* *Communications ACM*, 20(11): Seite 799-805, November 1977
- [2] Michael E. Gross und Kevin Wu *Study of Supersampling Methods for Computer Graphics Hardware Antialiasing* Hewlett-Packard Laboratories Palo Alto, California, December 5, 2000
- [3] R. A. Bresenham *Theoretical Foundations of Computer Graphics and CAD* Springer Verlag Heidelberg 1988
- [4] Blair McIntyre and William B. Cowan *A practical Approach to Calculating Luminance Contrast on a CRT.* March 6 1994