Lehrstuhl für Nachrichtentechnik, Lehrstuhl für Graphische Datenverarbeitung
Friedrich-Alexander Universität Erlangen-Nürnberg

Seminar
Spezielle Kapitel der Computergraphik

Wintersemester 1994/95

Prof. Dr. H.-P. Seidel
Erwin Keeve, Phillipp Slusallek

Reality Engine Graphic

Referent: Herbert Thoma
2. Februar 1995

Inhaltsverzeichnis

1 Graphikhardware

Im Gegensatz zu frühen Graphikhardware-Designs, die nur einen Framebuffer und die Videosignalerzeugung enthielten, denen mit der Zeit einfache 2D und 3D Funktionen hinzugefügt wurden, unterstützen moderne Designs komplexe Funktionen wie Gouraud-Shading, Texture Mapping oder Antialiasing.

Eine solche moderne Hardware ist die RealityEngine von Silicon Graphics, die im folgenden beschrieben werden soll. Sie führt sämtliche geometrischen Transformationen, die zur Projektion von 3D-Objekten auf Windowkoordinaten nötig sind, aus, berechnet die Beleuchtungsverhältnisse und stellt Funktionen für Texture Mapping sowie Antialiasing bereit. Ein Anwendungsprogramm muß sich also „nur“ noch um die Bereitstellung einer geeigneten Datenbasis kümmern.

2 Architektur

Die RealityEngine ist auf 3, 4 oder 6 Leiterplatten aufgebaut, Abbildung 1 zeigt ein Blockdiagramm. Auf dem Geometry Board befinden sich der Command Processor und 6, 8 oder 12 Geometry Engines. Die Geometry Engines sind über den Triangle Bus mit den Fragment Generators auf den Raster Memory Boards verbunden. Das Gesamtsystem kann mit 1, 2 oder 4 Raster Memory Boards bestückt sein. Ein Raster Memory Board enthält 5 Fragment Generators. Jeder Fragment Generator ist mit 16 Image Engines verbunden. Das Display Generator Board führt die D/A-Wandlung der RGB Signale durch, erzeugt die Sync Signale und stellt einige zusätzliche Fahigkeiten, wie z. B. Genlock, bereit.

2.1 Command Processor

Der Command Processor ist im wesentlichen deshalb nötig, weil als Programmierschnittstelle für die RealityEngine die Graphikbeschreibungssprache OpenGL [5] gewählt wurde. OpenGL ist eine Sprache, die Elemente zur Beschreibung von 3D-Graphiken enthält. Einerseits sind das Befehle zur Beschreibung der Geometrie von 3D-Objekten, wie Punkte, Linien und Polygone, andererseits Befehle zur Steuerung der Darstellung, wie z. B. Art und Position der Lichtquellen. Befehle der zweiten Art werden sofort an alle Geometry Engines gesandt, während die der ersten Art nur an jeweils eine Geometry Engine weitergeleitet werden. Der Command Processor zerlegt längere Sequenzen von Geometrie-Befehlen in kürzere Folgen für verschiedene Geometry Engines, um die Auslastung der GEs möglichst gleich zu halten. Da

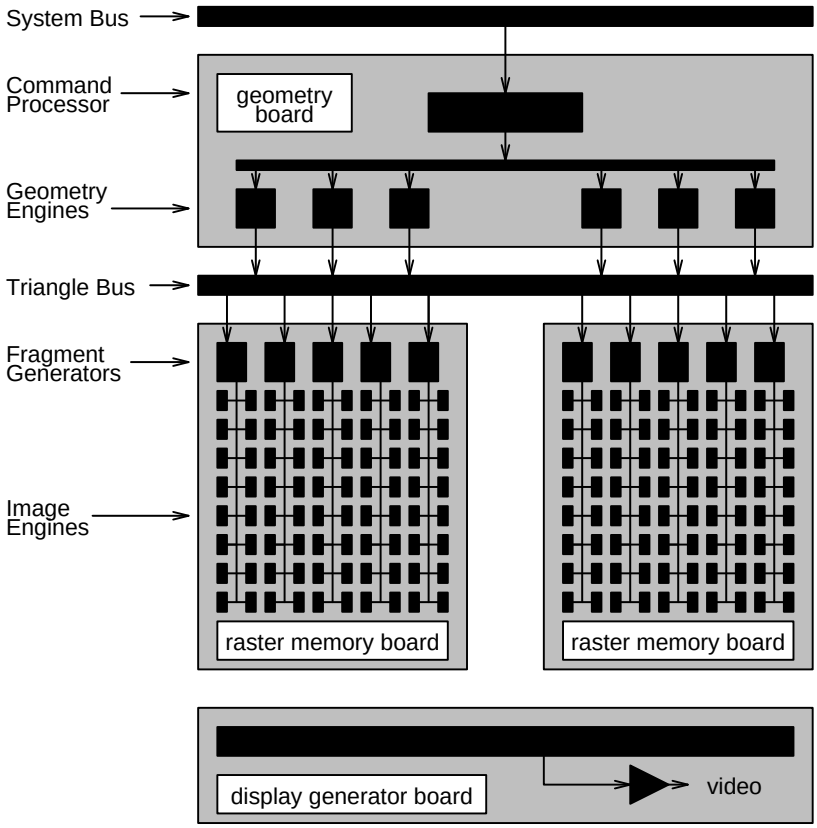

Abbildung 1: Blockdiagramm

der Command Processor jeden Befehl interpretiert, kann er auch ungültige Befehle erkennen und eine Fehlerbehandlung durchführen.

2.2 Geometry Engine

Die Geometry Engines führen alle Berechnungen aus, für die Fließkommaarithmetik benötigt wird. Das sind sämtliche geometrischen Transformationen, die Berechnung der Beleuchtung, das Clipping und die Projektion auf Window-Koordinaten.

Um die hohe benötigte Fließkommarechenleistung zu erzielen, ist jede Geometry Engine mit einem i860XP Prozessor von Intel ausgestattet, der mit 50 MHz Taktfrequenz betrieben wird und 2 MBytes Speicher zur Verfügung hat. Der i860 ist ein 64 Bit RISC Prozessor mit separatem Fließkomma-Addierer und -Multiplizierer. Durch die spezielle Architektur des i860 ist es möglich, daß Addierer und Multiplizierer je ein Ergebnis pro Takt liefern [4]. Daher

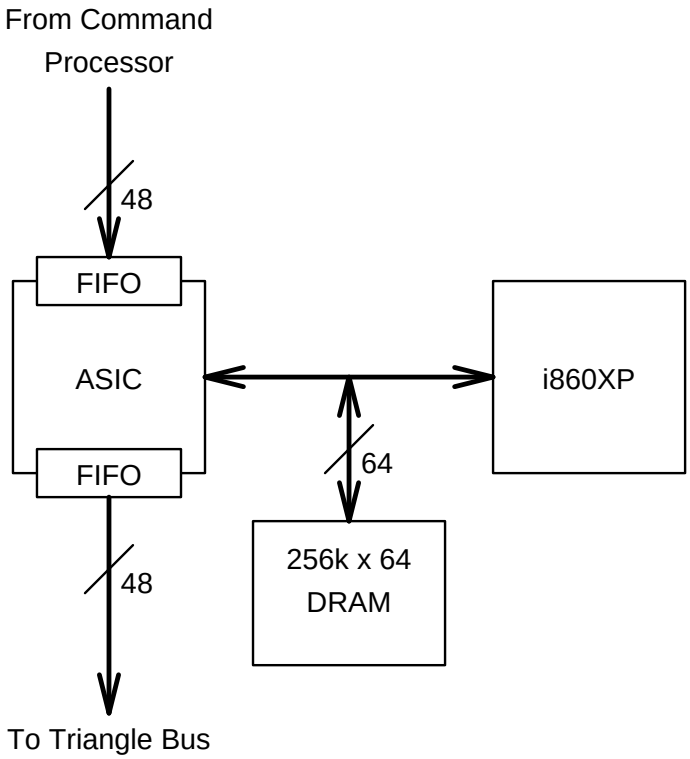

Abbildung 2: Geometry Engine

wird eine Spitzenleistung von 100 MFLOPS pro GE erreicht.

Da die Farbkomponenten und z-Werte zur Darstellung interpoliert werden müssen, berechnet die GE Steigungen in x- und y-Richtung, die die Fragment Generators zur Interpolation benutzen. Polygone mit mehr als 3 Ecken werden in Dreiecke zerlegt, da mehr als 3 Eckpunkte nicht unbedingt in einer Ebene liegen.

Um zu zeigen, wie groß die benötigte Fließkommarechenleistung ist, werden hier alle Schritte zur Darstellung und die Anzahl der Fließkommabefehle aufgeführt [2]:

1. *Transformation der Eckpunktkoordinaten in Augenkoordinaten:* Multiplikation eines Vektors mit einer 4×4 Matrix (homogene Koordinaten) benötigt 16 Multiplikationen und 12 Additionen, also *28* Befehle.

2. *Normalenvektoren Transformieren:* Da Normalen nur eine Richtung angeben, von Verschiebungen also nicht verändert werden, braucht man keine homogenen Koordinaten. Die Normalenvektoren müssen allerdings die Länge 1 haben, daher muß jede Komponente mit $\frac{1}{\sqrt{n_x^2+n_y^2+n_z^2}}$

multipliziert werden. Für die Berechnung der inversen Wurzel hat der i860 einen eigenen Befehl, der solange wie 4 pipelined Add oder Mul dauert. Insgesamt ergeben sich *27* Befehle.

3. *Beleuchtungsberechnung:* Für ein Beleuchtungsmodell mit einer Lichtquelle und Specular Highlights muß für jede Farbkomponente (RGB) die Gleichung

$$C_{object} = C_{ambient} + C_{diffuse} C_{light} (N \cdot L) + C_{specular} C_{light} (N \cdot H)^n$$

ausgewertet werden. Die Skalarprodukte werden nur einmal berechnet, für die n-te Potenz wird eine Lookup-Table erstellt. Außerdem wird noch auf Überlauf geprüft, eine Farbkomponente kann maximal 1.0 werden. Für die Beleuchtungsberechnung werden *32* Operationen benötigt.

4. *Transformation zu Clip-Koordinaten: 28* Operationen.

5. *Clipping:* Wenn der Punkt innerhalb der Clippinggrenzen liegt, genügen *6* Vergleiche. Liegt der Punkt außerhalb der Grenzen, so sind weitere Berechnungen nötig.

6. *Projektion:* x-, y- und z-Koordinaten durch w teilen: *7* Operationen.

7. *Viewport:* Skalieren und Verschieben der Koordinaten in den Viewport und Umwandlung in Integer-Werte: *9* Operationen.

8. *Berechnen der Texture Koordinaten: 15* Operationen.

Das ergibt 152 Fließkommabefehle pro Eckpunkt. Zusätzlich müssen für jedes Dreieck Steigungen zur Interpolation der Farb- und z-Werte, sowie für die Edge Funktions berechnet werden. Dies benötigt ca. 100 Operationen pro Dreieck. Sollen 1 Million Dreiecke pro Sekunde dargestellt werden, müssen dafür 556 Millionen Fließkommabefehle pro Sekunde ausgeführt werden. 12 Geometry Engines stellen eine theoretische Spitzenleistung von 1200 MFLOPS bereit. Die Rechenleistung muß also ca. 46% der theoretischen Maximalleistung erreichen, das ist schon als sehr gut für einen Vektorrechner zu bezeichnen.

Im ASIC sind Ein- und Ausgabe FIFO, ein Registersatz zur Kommunikation vom Command Processor mit der Geometry Engine und Schaltungen, die die berechneten Daten in ein spezielles Format für die Fragment Generators umkodieren, integriert.

2.3 Fragment Generators

Die Fragment Generators führen die Scan-Konvertierung durch, berechnen für jedes Pixel einen z-, Farb- und Texture-Wert und bestimmen für das Antialiasing, welche Subpixel von einem Dreieck überdeckt sind.

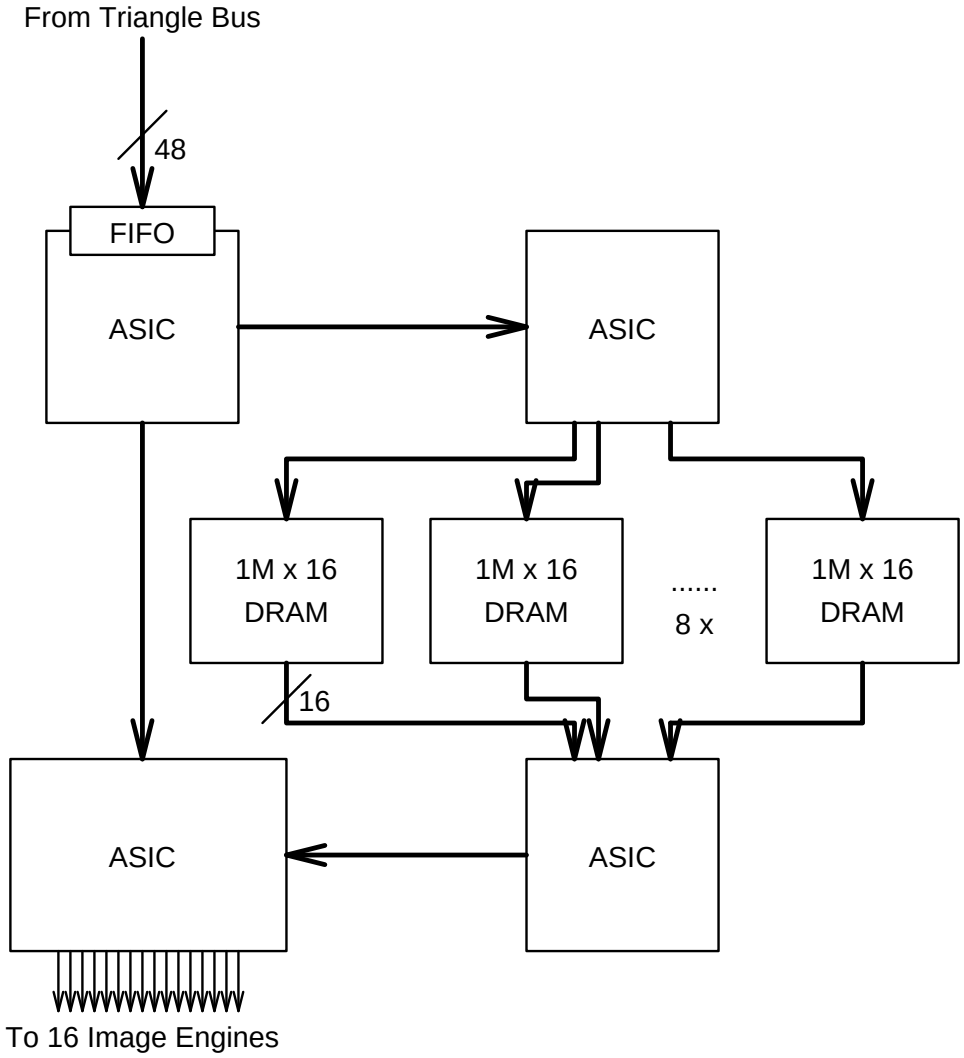

Abbildung 3: Fragment Generator

Abhängig von der Anzahl der installierten Raster Memory Boards ist ein Fragment Generator für $\frac{1}{5}$, $\frac{1}{10}$ oder $\frac{1}{20}$ der Pixel im Framebuffer zuständig. Die Fragment Generators sind den Pixeln so zugeordnet, daß benachbarte Pixel stets von verschiedenen Fragment Generators bearbeitet werden, um zu erreichen, daß auch kleine Dreiecke von allen Fragment Generators teil-

weise behandelt werden. Die Fragment Generators erzeugen noch keine Pixel für den Framebuffer, sondern sog. Fragments, die von den Image Engines weiterverarbeitet werden.

Die Scan-Konvertierung wird nach einem Algorithmus, der von Pineda [6] beschrieben wurde, ausgeführt. Dieser Algorithmus bedient sich zur Beschreibung der Dreiecke sog. Edge Funktions:

$$E(x,y) = (x - X)\mathrm{d}Y - (y - Y)\mathrm{d}X$$

Diese Edge Functions sind Geradengleichnugen und haben folgende Eigenschaften:

$$\begin{aligned} E(x,y) &> 0, \text{ wenn } (x,y) \text{ rechts von der Geraden ist} \\ E(x,y) &= 0, \text{ wenn } (x,y) \text{ auf der Geraden ist} \\ E(x,y) &< 0, \text{ wenn } (x,y) \text{ links von der Geraden ist} \end{aligned}$$

Der Wert $E(x,y)$ ist dabei ein Maß für den Abstand des Punktes (x,y) von der Geraden. $E(x,y)$ läßt sich inkremental berechnen:

$$\begin{aligned} E(x+1,y) &= E(x,y) + \mathrm{d}Y \\ E(x-1,y) &= E(x,y) - \mathrm{d}Y \\ E(x,y+1) &= E(x,y) - \mathrm{d}X \\ E(x,y-1) &= E(x,y) + \mathrm{d}X \end{aligned}$$

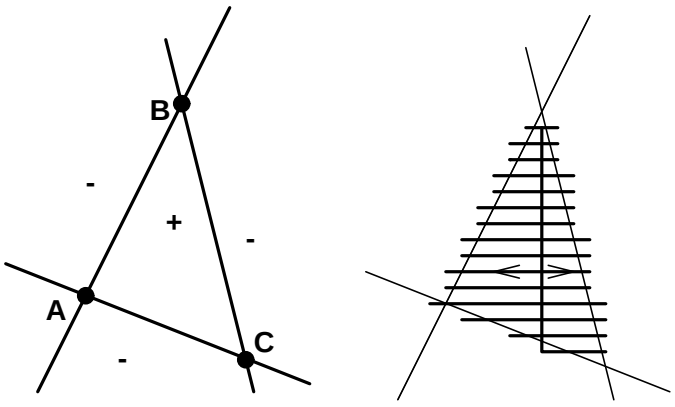

Abbildung 4: Scan-Konvertierung

Durch 3 solcher Edge Funktions wird ein Dreieck beschrieben. Der Scan-Konvertierungs-Algorithmus muß von einem Startpunkt ausgehend alle Pixel des Dreiecks besuchen und den Wert der drei Edge Funktions berechnen. Sind

alle drei Werte positiv, gehört das entsprechende Pixel zum Dreieck (siehe Abblidung 4).

Die Werte der Edge Funktions werden mit einer Schaltung nach Abbildung 5 berechnet. Die gleiche Schaltung eignet sich auch zur Interpolation der Farb- und z-Werte. Es sind also 8 solcher Schaltungen in einem Fragment Generator enthalten (3 Edge Funktions, RGBA, z). Die Anfangswerte und Steigungen werden von den Geometry Engines berechnet.

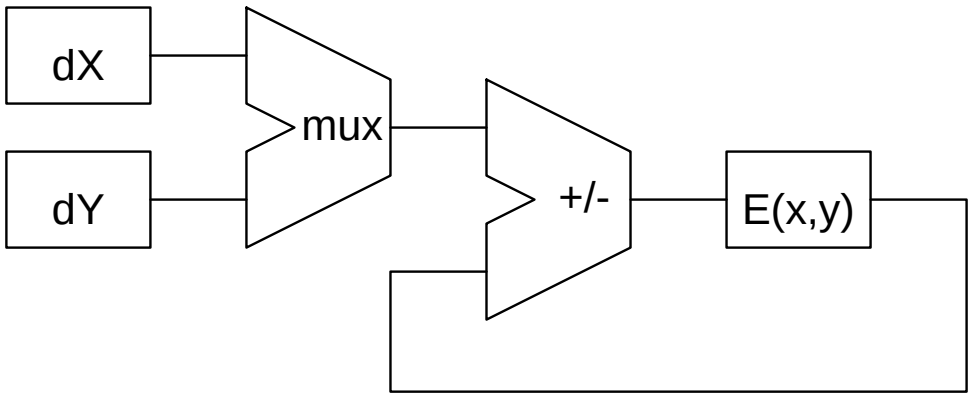

Abbildung 5: Berechnung der Edge Funktions

Die Eigenschaft der Edge Funktion, daß ihr Wert ein Maß für den Abstand zur Geraden ist, wird zusammen mit der Steigung der Geraden ausgenutzt, um für das Antialiasing zu bestimmen, welche Subpixel zum Dreieck gehören (siehe Abschnitt 3.1). Der überdeckte Bereich wird durch eine Maske (Bitmuster) repräsentiert [7].

Bei 2D MIP Mapped Textures (siehe 3.2) werden 8 Texels benutzt, um die Texture für ein Pixel zu bestimmen. Der Fragment Generator berechnet die entsprechenden Adressen im Texture Memory, aus dem alle 8 Werte gleichzeitig ausgelesen werden. In den 8 Speicherbänken ist also jeweils das gleiche gespeichert. Alle Fragment Generators haben ihr eigenes Texture Memory. Aus den erhaltenen Werten wird der entgültige Texture-Wert errechnet.

Die Texture- und z-Werte werden immer im Mittelpunkt eines Fragments berechnet, die Farbwerte nur dann im Mittelpunkt, wenn dieser auch überdeckt ist, andernfalls an einer Position nahe dem Zentrum des überdeckten Bereichs.

Die letzte Aufgabe die der Fragment Generator erledigt, ist das Einblenden von Nebel. Abhängig vom z-Wert wird der Nebelanteil aus einer Tabelle ausgelesen und mit der Farbe verrechnet.

2.4 Image Engines

Die so erzeugten Fragments, die aus Pixelkoordinate, Farbwert, z-Wert und Überdeckungsmaske bestehen, werden von den Image Engines in darstellbare Pixel umgewandelt. Jede Image Engine verwaltet einen Teil des Accumulation Buffers, in dem neben bis zu 4 eigentlichen Framebuffern die Farb- und z-Werte der Subpixels und zusätzliche Status- und Window-ID-Bits gespeichert werden.

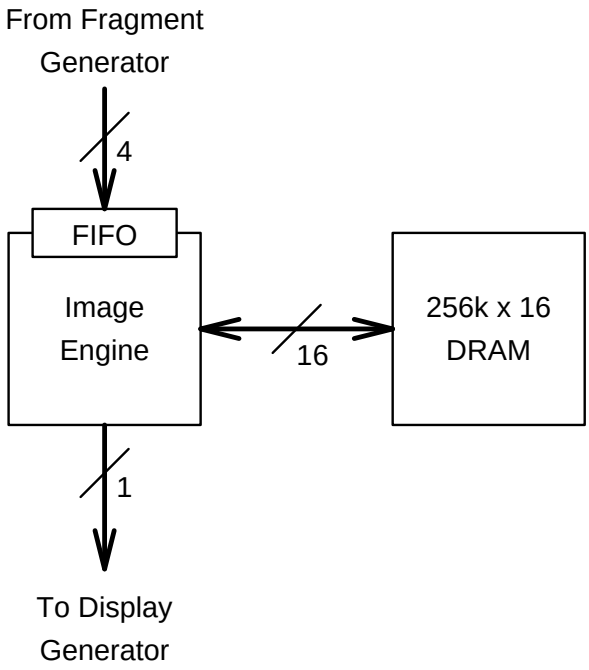

Abbildung 6: Image Engine

Die Image Engine berechnet für jedes Subpixel den genauen z-Wert aus dem z-Wert am Mittelpunkt und den Steigungen in x- und y-Richtung. An den überdeckten Subpixeln wird dann der berechnete z-Wert mit dem gespeicherten Wert verglichen. Ist der neue Wert näher als der alte, wird der gespeicherte Farbwert durch den Farbwert des Fragments ersetzt. Der Farbwert für das Pixel wird dann durch Mittelwertbildung über die Subpixels errechnet.

Der Accumulation Buffer kann mit 256, 512 oder 1024 Bits pro Pixel konfiguriert werden. Eine Image Engine verwaltet dann 16384, 8192 oder 4096 Pixel. Farbwerte werden mit 12 Bits pro Rot-, Grün-, Blau- und Alphakomponente gespeichert. Die Alphakomponete beschreibt die Transparenz des Pixels, sie wird benutzt, wenn das Videosignal der RealityEngine mit einem anderen

Videosignal gemischt wird. Die z-Werte werden mit 32 Bits pro Subpixel gespeichert. Die Farbwerte können auf 8,8,8 Bits und die z-Werte auf 24 Bits reduziert werden, um mehr Subpixel speichern zu können. Es sind 1, 2 oder 4 Displayable Color Buffers möglich, 4 Buffer werden zur Darstellung von bewegten Stereobildern benötigt. Bei 1024 Bits pro Pixel ist es z. B. möglich, 2 Displayable Color Buffers und 8 Subpixel mit 12 Bit Farbauflösung zu speichern.

Weil die Anzahl der Raster Memory Boards unabhängig von der Zahl der Bits pro Pixel im Accumulation Buffer ist, unterstützt die RealityEngine eine große Viefalt von Framebuffergrößen und Farbauflösungen. Ein System mit einem Raster Memory Board erlaubt z. B. Antialiasing mit 16 Subpixeln bei 640×512 Auflösung oder eine Framebuffergröße von 1280×1024 ohne Antialiasing. Sind 4 Raster Memory Boards installiert, ist bei 1280×1024 Pixeln 16 Subpixel Antialiasing möglich.

2.5 Display Generator

Jede Image Engine sendet Daten seriell an das Display Generator Board. Dort werden die Daten wieder nach Parallel gewandelt und der D/A-Wandlung zugeführt. 12 Bit Farbwerte werden nach 8 Bit konvertiert, wobei auch eine Gamma-Korrektur vorgenommen wird. Die RealityEngine unterstützt auch Farb Index Modi mit unterschiedlichen Lookup Tables für bis zu 40 Fenster.

Das Video Timing ist frei programmierbar, so daß eine große Vielfalt an Videoformaten erzeugt werden kann. Der maximale Pixeltakt beträgt 140 MHz, was bei 1600×1200 Pixeln Auflösung eine Bildwiederholfrequenz von 60 Hz ermöglicht.

Das Display Generator Board erzeugt auch ein Composite Video Signal, so daß man einen Videorecorder direkt anschließen kann. Das Videosignal der RealityEngine kann mit einem externen Videosignal synchronisiert werden, diese Eigenschaft bezeichnet man als Genlock.

Für Stereodarstellung stellt der Display Generator ein Ansteuersignal für Shutterbrillen oder Polarisationsfilter bereit.

3 Besondere Fähigkeiten

3.1 Antialiasing

Die RealityEngine unterstützt zwei verschiedene Arten von Antialiasing: Alpha- und Multisample-Antialiasing.

Beim Alpha-Antialiasing wird der Anteil der Fläche eines Pixels berechnet, der von einem Polygon überdeckt wird, indem das einzelne Pixel in 8×8 Subpixel unterteilt wird und die überdeckten Subpixel abgezählt werden. Die Farbe des Pixels ergibt sich dann durch linerare Interpolation zwischen dem alten Farbwert und der Farbe des Polygons mit dem Flächenanteil als Parameter. Der schwerwiegende Nachteil des Alpha-Antialiasing ist, daß es nicht unabhängig von der Zeichenreihenfolge ist, Polygone müssen vor der Darstellung ihrem z-Wert nach sortiert werden.

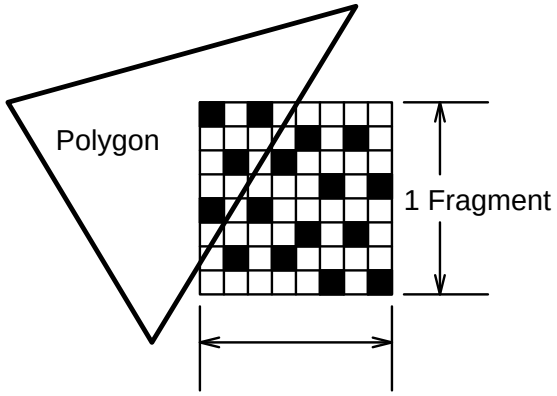

Abbildung 7: Multisample-Antialiasing

Beim Multisample-Antialiasing werden im Accumulation-Buffer die Farb- und z-Werte von 4, 8 oder 16 Subpixeln gespeichert. Die Positionen der Subpixel können in einem 8×8 Gitter frei gewählt werden, es gilt aber für alle Pixel die gleiche Anordnung (siehe Abbildung 7). Die Fragment Generators berechnen, welche Subpixelpositionen von einem Polygon überdeckt sind. Die Image Engines berechnen für jedes Subpixel den z-Wert. Wenn der neu berechnete z-Wert näher als der gespeicherte ist und das Subpixel überdeckt ist, wird der Subpixel-Farbwert durch die Farbe des Polygons ersetzt. Die Farbe des Pixels wird durch Mittelwertbildung über alle Subpixel errechnet und in den Displayable-Color-Buffer geschrieben.

Es gibt zwei Methoden, nach denen bestimmt wird, ob ein Subpixel überdeckt ist: Point Sampled und Area Sampled. Das Point Sampled Verfahren ist geometrisch korrekt, d. h. es werden nur die Subpixel herangezogen, die auch wirklich überdeckt sind. Allerdings werden z. B. lange, schmale Polygone nur schlecht repräsentiert (siehe Abbildung 8), was sich auch in der Bildqualität äußert.

Beim Area Sampled Verfahren werden die Subpixel so ausgewählt, daß die bedeckte Fläche möglichst gut angenähert wird. Dadurch werden auch Subpixel gesetzt, die nicht im Inneren eines Polygons liegen, es können Artifakte

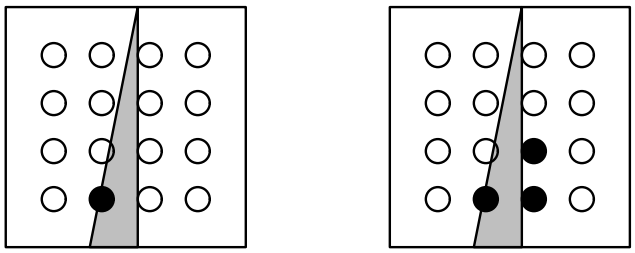

Abbildung 8: Point Sampled — Area Sampled

an den Kanten auftreten.

Wenn die darzustellenden Polygone eine Alpha-Farbkomponente besitzen, also teilweise durchsichtig sind, ist auch Multisample-Antialiasing nicht mehr unabhängig von der Zeichenreihenfolge. Der Alpha-Wert wird dann in eine Pseudo-Zufalls-Maske umgerechnet, die mit der Maske der Subpixelpositionen UND-verknüpft wird. Diese Methode ist zwar geometrisch nicht korrekt, stellt aber eine brauchbare Näherung dar und ist unabhängig von der Zeichenreihenfolge.

3.2 Texture Mapping

Texture Mapping ist ein sehr leistungsfähiges Werkzeug um Computergenerierte Szenen mit mehr Realismus auszustatten. Deshalb stellt die RealityEngine eine Reihe von Texture Mapping Funktionen bereit. Texture Maps können ein-, zwei- und dreidimensional sein, die Textures können im sog. MIP Mapped Format gespeichert werden.

MIP Mapping ist ein Verfahren, bei dem zusätzlich zur Texture Map in voller Auflösung vorgefilterte, verkleinerte Maps gespeichert werden, auf die bei der Darstellung kleiner Polygone zurückgegriffen wird [9]. Bei der Ermitlung des Texture Wertes für ein Pixel interpoliert die RealityEngine zwischen je 4 Texels aus 2 Verkleinerungsstufen. Der Speicherbedarf für 2D MIP Mapped Textures ist um $\frac{1}{3}$ größer als für Non MIP Mapped Textures.

3D Textures sind als quaderförmige Anordnung von Texels definiert. Der Zugriff auf 3D Textures erfolgt durch Schnitt des Quaders mit einer Ebene (Abbildung 9). Auch bei 3D Textures ist MIP Mapping möglich.

Um die Darstellungsqualität von Flächen, die nahe am Beobachter sind, zu verbessern, kann eine zweite Texture aufgebracht werden. Dieses Verfahren wird DetailTexture genannt. Ein Beispiel dafür ist eine Straße, die durch eine erste Texture Pfeile und Begrenzungslinien erhält, auf die bei kurzem Betrachtungsabstand eine zusätzliche Feinstruktur aufgebracht wird.

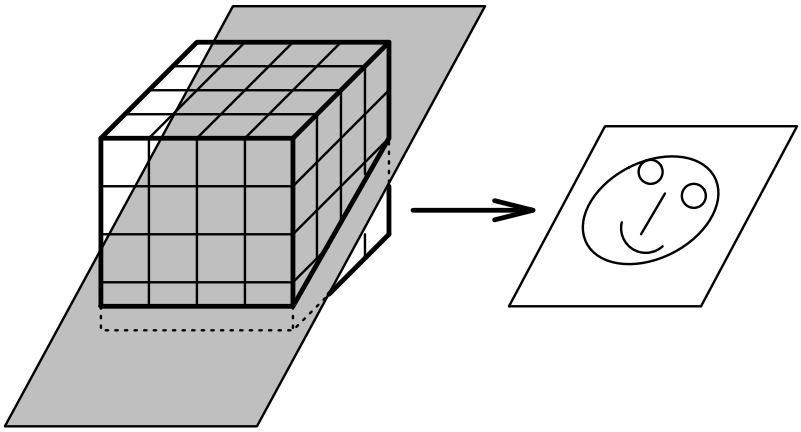

Abbildung 9: 3D Texture

Die maximale Auflösung für 2D MIP Mapped Textures ist 1024 $\times$ 1024 bei 16 Bit pro Texel. 3D Non MIP Mapped Textures können bis zu 256 $\times$ 256 $\times$ 64 groß sein. Die Quantisierung der Farbkomponenten kann auf bis zu 12 Bit erhöht werden, was aber einen Geschwindigkeitsverlust bedingt.

Literatur

[1] Akeley, Kurt. RealityEngine Graphics. In *Proceedings of SIGGRAPH '93* (August 1993), pp. 109-116.

[2] Akeley, Kurt and Jermoluk, Tom. High-Performance Polygon Rendering. In *Proceedings of SIGGRAPH '88* (August 1988), pp. 239-246.

[3] Haeberli, Paul and Akeley, Kurt. The Accumulation Buffer: Hardware Support for High-Quality Rendering. In *Proceedings of SIGGRAPH '90* (August 1990), pp. 309-318.

[4] Klein, Rolf-Dieter and Thiel, Tobias. PC-Karte mit i860. In *mc, Die Mikrocomputer-Zeitschrift* (Februar 1990), pp. 100-109.

[5] Kunze, Michael. Raumbeschreibung: 3D-Grafikstandart OpenGL. In *c't magazin für computer technik* (August 1994), pp. 198-200.

[6] Pineda, Juan. A Parallel Algorithm for Polygon Rasterization. In *Proceedings of SIGGRAPH '88* (August 1988), pp. 17-20.

[7] Schilling, Andreas. A New Simple and Efficient Antialiasing with Subpixel Masks. In *Proceedings of SIGGRAPH '91* (July 1991), pp. 133-141.

[8] Silicon Graphics, Inc. RealityEngine in Visual Simulation Technical Report. Silicon Graphics, Inc., Mountain View, CA, 19??.

[9] Williams, Lance. Pyramidal Parametrics. In *Proceedings of SIGGRAPH '83* (July 1983), pp. 1-11.