Fachseminar
Graphische Datenverarbeitung SS98
Prof. M.Gross

Designing Real-Time Graphics for Entertainment (Nintendo64)

Internet Research

Seminararbeit von
Alexander Beck

Paper: Internetsourcen & Designing Real-Time Graphics for Entertainment
SIGGRAPH ’97 Course

Inhalt

1. Einleitung

Meine Aufgabe war es so viel wie möglich über die Grafikpipeline der Spielkonsole Nintendo64 herauszufinden, was sich nicht als einfache erwies.

Der Nintendo64 enthält momentan die ausgefeilteste Technik von den Geräten auf dem Spielkonsolenmarkt. Dennoch ist zur Zeit nur sehr wenig über die interne Verarbeitung der Grafik bekannt. Darum gibt es auf dem Internet sehr viele Spekulationen darüber, wie diese Konsole funktioniert. Ich habe versucht, alle wichtigen Daten zusammenzutragen..

2. Hardware

2.1 Übersicht über das System

Der Nintendo64 ist ein Produkt aus der Zusammenarbeit von Nintendo Inc. und Silicon Graphics Inc. (SGI). SGI entwickelte dabei zwei Prozessoren für Nintendo, den MIPS R4300i RISC-Prozessor und einen MIPS 64-bit RISC Custom Chip, der als Grafik-CoProzessor arbeitet. Der Nintendo64 basiert auf Cartridges (Module) als Medium für die Spiele und nicht wie die Sony Playstation auf CD-ROM. Damit ist zwar der verfügbare Speicherplatz nicht so gross (max. 32 MB), dafür ist aber der Zugriff auf das Cartridge-ROM viel schneller.

Die wichtigsten Teile des Nintendo64 sind:

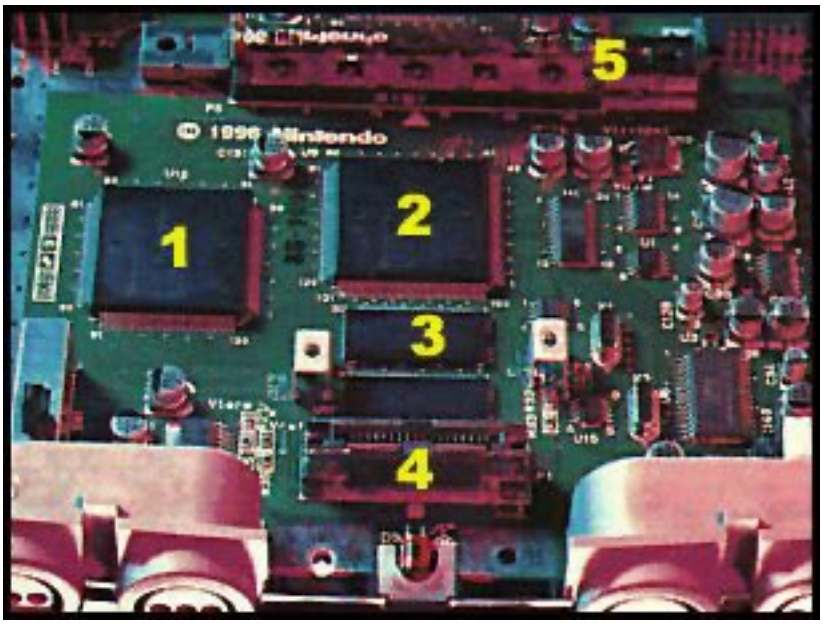

Abbildung 1: Bausteine des Nintendo64

1. **MIPS R4300i RISC-Prozessor als CPU:** Es ist ein echter 64-bit Prozessor (d.h. Busse und Register sind 64-bit breit) und ist mit 93.75 MHz getaktet.
2. **MIPS 64-bit RISC „Reality Immersion" Grafik-CoProzessor (RCP):** Dieser ist mit 62.5 MHz getaktet und ist das eigentliche Herzstück des Systems. Genauere Informationen können in Kapitel 2.2 nachgeschlagen werden.
3. **4 MB Rambus 9-bit DRAM:** Dieses RAM hat eine max. Transferrate von über 500 MB/s. Der Datenbus zum RCP ist 128-bit breit und kann bis max. 500 MHz getaktet werden.

4. **RAM expansion slot:** Hier kann der Nintendo64 auf 8 MB RAM erweitert werden.
5. **Cartridge slot:** Hier ist die Verbindung mit dem Cartridge-ROM vorgesehen.

Die weiteren Leistungsdaten des Nintendo64 sind:
- Schnittstellen:
    - **21-bit color Output** (32-bit RGBA pixelcolor framebuffer) auf R/F, Stereo A/V, S-Video oder HDTV
    - **CD-Qualität 16-bit Stereo Sound** mit 44.1 kHz maximal

- Auflösung:
    - Low-resolution: 256 x 224
    - High-resolution: 640 x 480

2.2 Realitiy Co-Prozessor (RCP)

Der RCP soll intern mit 128-bit breiten Datenbussen arbeiten, darauf weißt schon der 128-bit breite Bus zum RAM hin. Damit wäre es möglich zwei Operationen auf einmal zu bearbeiten und sehr schnell auf das RAM zuzugreifen. Extern ist sein Bus auf jeden Fall nur 64-bit breit.

Der RCP ist intern in zwei Teile aufgeteilt, den Reality Signal Prozessor (RSP) und den Reality Display Prozessor (RDP).

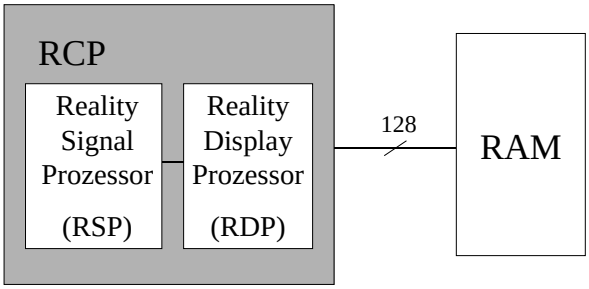

Abbildung 2: Aufbau des Reality CoProzessors

2.2.1 RSP

Der RSP übernimmt die Generierung des Audio mit einer „wavetable synthesis sound“ Einheit. Von ihm werden auch alle Signale des Benutzers (Joypad) entgegengenommen und verarbeitet. Als drittes berechnet er auch die grundsätzliche Positionierung der Polygone auf dem Bildschirm, d.h. er berechnet alle Rotationen, Skalierungen, und Translationen der Objekte.

Er ist ein Vektor Prozessor, d.h. wenn eine Operation wie:

```
ADD V1, V2, V3
```

ausgeführt wird, wird nicht nur ein Register V1 mit V2 addiert und in V3 gespeichert, sondern ein ganzer Vektor von Registern. Im Falle des RSP werden eine Gruppe von 8 Registern V1 komponentenweise mit den 8 Registern V2 addiert und die Resultate in 8 weitere Register V3 geschrieben. Und das passiert alles mit einer einzigen Operation zur selben Zeit.

Von SGI wurde veröffentlicht, dass der RCP über 500 Mio. Operationen / Sekunde beherrscht. Da nicht bei jeder Operation 64-bit Genauigkeit benötigt wird, wird angenommen, dass der RCP auch auf 32- oder 16-bit konfiguriert werden kann, da man sonst viele Ressourcen verschwendet. Dadurch würden auch zwei 32-bit oder vier 16-bit Operationen anstelle einer 64-bit Operation unterstützt. Das hiesse, es wären viel mehr Operationen pro Sekunde bei tieferer Genauigkeit möglich als diese 500 Mio.

Design Typ	64-bit	32-bit	16-bit
Vektor (konfigurierbar)	>500M	>1000M	>2000M

Dies trifft auch mit der PR-Aussage überein, dass SGI nicht eine genaue Zahl preisgab, sondern nur eine untere Grenze angab.

Dieses Vekordesign ist natürlich besonders effektiv, um Polygontransformationen im 3D Raum zu berechnen.

2.2.2 RDP

Der RDP fügt die interessanten Hardwareeffekte wie Texturmapping, bilineare Interpolation, Mip-Mapping, usw. zu den transformierten Polygonen hinzu und übergibt die berechneten Pixel als Strom zur Darstellung auf dem Bildschirm weiter. Auf die Hardwareeffekte wird in Kapitel 4 noch näher eingegangen.

3. Aufgabenverteilung

Bei Nintendo64 sind Audio und Video in zwei separate völlig unabhängige Aufgaben aufgeteilt. Auf die Verarbeitung des Audio wird nicht genauer eingegangen.

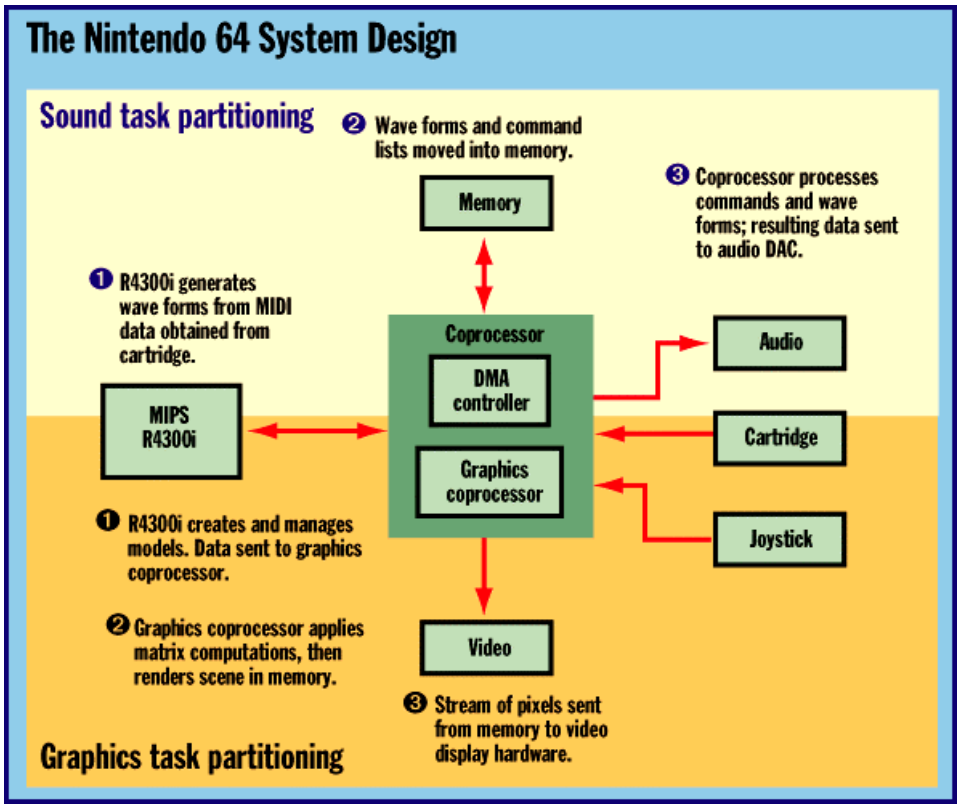

Abbildung 3: Aufgabenverteilung im Nintendo64

3.1 R4300i CPU

Die CPU arbeitet als zentraler Controller und übernimmt generelle Systemfunktionen und grundlegende Audio Operationen. Um die Grafik zu generieren, erstellt und manipuliert der R4300i
Modelle (3D-Objekte, die als Polygonnetz beschrieben sind), um sie in der virtuellen Spielwelt zu benutzen. Wenn der Programmcode des Spiels es verlangt, dass die Modellposition und –attribute aktualisiert werden, kann dies die CPU in Echtzeit erledigen. Jetzt werden die Modelle an den RCP übergeben, damit dieser die Matrixmanipulationen und das Bildrendering vornehmen kann. Zusätzlich übernimmt er noch die Aufgabe, Daten aus dem Cartridge zu dekomprimieren, was zum Beispiel bei den Texturen notwendig ist, da diese im JPEG-Format im Cartridge gespeichert sind.
Die 64-bit des R4300i geben den Entwicklern zusätzliche Präzision für die Modelle und andere Berechnungen, ohne Hochpräzisionsalgorithmen schreiben und dadurch Geschwindigkeit einbüssen zu müssen.

3.2 RCP

Der RCP übernimmt jetzt die Aufgaben von der CPU, bei denen Softwarealgorithmen die Rahmenbedingungen der Geschwindigkeit nicht mehr alleine erfüllen können. Er übernimmt die Modelle von der CPU und der RSP führt zuerst alle Matrixoperationen auf das Modell aus. Das transformierte Objekt wird nun an den RDP übergeben, der die gewünschten Hardwareeffekte auf die Modelle ausführt und das darzustellende Bild rendert. Dieses wird dann als Pixelstrom den Displaykomponenten übergeben, die das Bild dann anschliessend auf dem Bildschirm darstellen.

3.3 Multiprocessing

Ein weiterer Geschwindigkeitsvorteil ergibt sich dadurch, dass der Nintendo64 separate Prozesse auf der CPU, dem Reality Signal Prozessor und dem Reality Display Prozessor gleichzeitig ausführen kann. Dies erlaubt es zum Beispiel im gleichen Moment Pixel zu zeichnen, Audio zu generieren und auf Spielereingaben zu reagieren.

4. Implementierte Grafikfunktionen

Alle Grafikeffekte werden allesamt im Reality Display Prozessor berechnet. In diesem Kapitel wird ein bisschen genauer auf die interessantesten eingegangen, damit klar wird, was für ein Potential dahinter steht. Ein wichtiger Punkt ist, dass alle diese vorgestellten Effekte in Hardware implementiert sind, und dadurch fast keine Rechenleistung verloren geht!

4.1 Spekulare Reflexion

Diese Funktion erzeugt einen Glanzpunkt auf der glänzenden Oberfläche eine Objektes, wodurch vor allem stark spiegelnde Materialien wie zum Beispiel Metall erzeugt werden können. Ein Beispiel ist im Spiel "Super Mario 64" zu finden.

Abbildung 4: Beispiel zu spekularer Reflexion

4.2 Gouraud-Shading

Dies ist eine Funktion, die den Realismus eines Objektes gewaltig steigern kann, denn um mit Flat-Shading den gleichen Effekt zu erreichen, wären über 100 Mal mehr Polygone nötig, wobei man immer noch die Facetten des Objekts sehen würde. Beim Gouraud-Shading werden die Normalen an den Eckpunkten eines Objektdreieckes entweder durch das mathematische Modell des Objektes analytisch ausgerechnet, oder über die anliegenden Flächennormalen gemittelt.

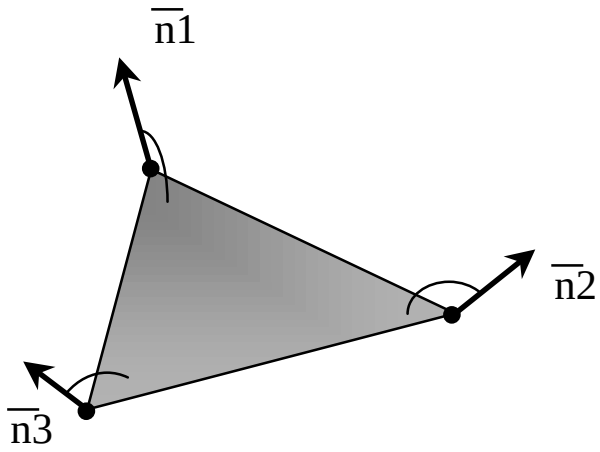

Abbildung 5: Normalen eines Gouraud-Dreiecks

Diese Normalen werden dann genommen, um mit einem Beleuchtungsmodell (z.B. Phong) die Farbe der Eckpunkte zu berechnen. Die drei Farben der Eckpunkte werden dann über das Dreieck linear interpoliert. Da die Normalen, den Echten bzw. den Gemittelten entsprechen ist ein "weicher" Farbübergang bei den Ecken und Kanten zu sehen.

Das einfachere Flat-Shading, wo ein Modelldreieck nur eine Farbe besitzt, wird durch das Gouraud-Shading simuliert, da wegen der Hardwareimplementierung ja kein Geschwindigkeitsverlust vorhanden ist. Es wird nur eine Farbe berechnet, die dann allen Punkten gegeben wird. Durch Verzicht auf eine Flat-Shading Funktion wird Platz auf dem Chip gespart.

4.3 Anti-Aliasing

Was sind Aliasingeffekte? Sie erzeugen das stufige Aussehen, wenn nicht-horizontale oder nicht-vertikale Linien auf einem Rasterbildschirm gezeichnet werden. Bei einer Animation kann es sogar sein, dass es aussieht, als ob diese Linien zu laufen anfangen. Nachdem eine Linie mit Anti-Aliasing bearbeitet wurde, scheint sie weicher, aber auch nicht mehr so scharf.

Es muss aber immer vom Spieleprogrammierer beachtet werden, dass Anti-Aliasing eingeschaltet ist.

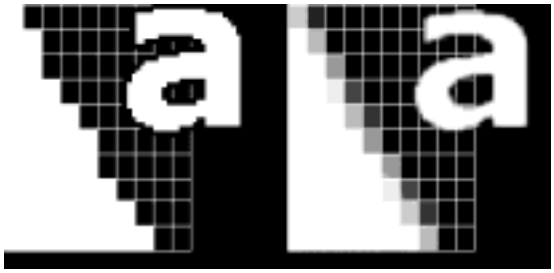

Abbildung 6: Beispiel Anti-Aliasing

Es gibt nun mehrere Arten dieses Aliasing zu eliminieren. Eine übliche Methode ist, die Unterschiede der Pixelhelligkeiten zu Nachbarn basierend auf den berechneten Linien zu untersuchen, und die Farbe über die Nachbarpixel mitteln. Eine zweite Methode ist das Bild an den Linien mit einer höheren Auflösung zu rendern, und dann basierend auf diesen Subpixeln, den Farbwert für den Bildschirmpixel zu mitteln. Eine komplexere Methode ist, das Bild mathematisch zu vibrieren und dann das Mittel zu nehmen. Da die letzten zwei Methoden sehr komplex sind und viel Speicher benötigen, wird angenommen, dass die erste Methode in der Hardware des Chips integriert ist, da der Nintendo64 nur begrenzte Speicherressourcen hat.

4.4 Texturmapping

Texturmapping vergrössert den Realismus eines Bildes enorm! Alle Texturen sind beim Nintendo64 im Spielemodul als JPEG komprimiert gespeichert und werden vom Hauptprozessor in Echtzeit geladen und dekomprimiert. Die nächsten vier Grafikeffekte beziehen sich alle auf Texturen.

4.5 Bi- und trilineare Interpolation

Aliasing eliminiert aber nicht den Effekt der "grossen" Pixel innerhalb einer Textur, wenn man zu nahe an diese herankommt, und sie nicht detailliert genug ist. So eine Textur kann durch bilineare Interpolation (Filterung) bearbeitet werden und dieser Effekt wird behoben, wie man in der Abbildung 7 sieht.

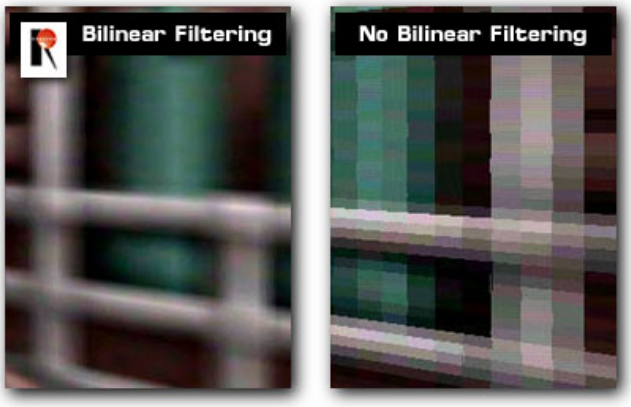

Abbildung 7: Beispiel bilineare Filterung

Hierbei werden einfach die Farben zwischen zwei Farbquadraten linear interpoliert. Hat die Textur zu wenig Details, kann es sein, dass sie nachher etwas verschwommen aussieht.

4.6 Trilineares Mip-Mapping

MIP steht für "multi in partem", was soviel heisst wie "mehrere Teile". In der Tat wird die Originaltextur in verschiedenen Grössen gespeichert. Wenn eine Textur zum Beispiel 16x16 Pixel gross ist, wird sie auch noch Mip-Maps der Grössen 8x8, 4x4, 2x2 und 1x1 haben. Jetzt wird je nach Distanz des Objektes zum Betrachter eine geeignete Mip-Map für das Texturmapping genommen. Mip-Mapping wird angewandt, um Texturen nicht immer mit tiefer Qualität skalieren zu müssen, denn diese Mip-Maps können schon im voraus mit guten Bildbearbeitungsalgorithmen verkleinert werden. Auch wird das Schimmern der Texturen, wenn sie zu klein werden behoben. In Zusammenhang mit level-of-detail Management (LOD), was ich noch später beschreiben werde, wird entschieden, welche Textur verwendet werden soll.

Trilinear ist noch eine Erweiterung dazu. Damit man nicht sieht, dass die Texturgrössen geändert werden, wenn man näher kommt, wird beim trilinearen Mip-Mapping zur Texturfilterung auch noch zwischen zwei Mip-Maps linear interpoliert. Dies macht den Übergang zwischen zwei Texturgrössen unsichtbar.

Trilineares Mip-Mapping ist eine sehr leistungsfähige Funktion, vor allem, weil sie in Hardware, d.h. Echtzeit, implementiert ist.

Bei dem Spiel SuperMario64 wurde Mip-Mapping nicht ausgenutzt, da das Spielcartridge nur 8MB beinhaltete und so kein Platz für mehrere Grössen der Texturen vorhanden war.

4.7 Environment Mapping

Beim Environment Mapping wird darauf geachtet, dass in einem spiegelnden Objekt oder Oberfläche die Umgebung reflektiert wird, obwohl sie nicht unbedingt modelliert sein muss. Es wird einfach eine Textur, die die Umgebung zeigt, als Sphäre oder Quader um das Objekt gelegt und die Spiegelungen der Innenseite dieser Box auf das Objekt abgebildet. Dies ergibt Objekte die aussehen, als ob sie in einer realen Umgebung plaziert sind.

4.8 Perspektive Korrektur

Die Perspektive Korrektur befasst sich mit Texturen, die auf Objekte gelegt sind. Bewegt oder rotiert man diese aber, so sind sie in Bezug auf die Fluchtpunkte nicht mehr perspektivisch korrekt auf dem Objekt abgebildet, was zu solchen Effekten führt, wie man sie in der Abbildung 8 sieht. Perspektive Korrektur wird von der Hardware des RDP übernommen.

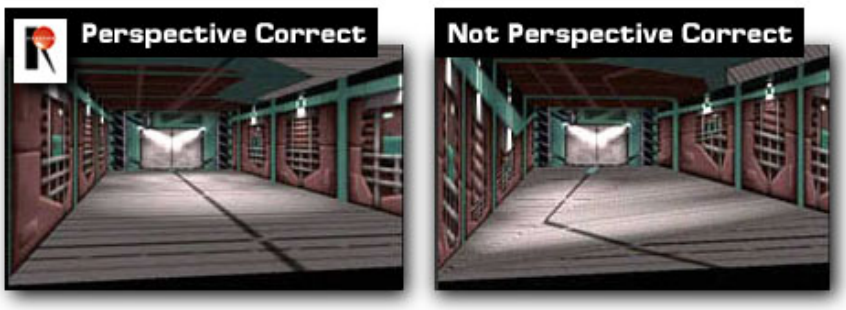

Abbildung 8: Beispiel perspektive Korrektur

4.9 Z-Buffering

Diese Funktion beinhaltet für jeden Pixel des Bildschirms einen Z-Buffer. Dieser hat normalerweise verschiedene Distanzen (Schichten) in 16-bit kodiert, was für die meisten Anwendungen ausreicht (ausser man möchte 1 mm Genauigkeit auf 1 km Distanz haben). Für jede Ebene ist gespeichert, welches Polygon sich darin befindet. Daraus folgt, dass man für jeden Pixel des Bildschirmes die Tiefe der getroffenen Polygone gespeichert hat. Der Z-Buffer wird dann normalerweise dazu benutzt, um im Zusammenhang mit Alpha Transparenzeffekten und Depth die Pixelfarbe zu berechnen. Z-Buffering wird auch von der Hardware unterstützt.

4.10 Depth Cueing (Nebel, Dunst, Wasser, etc.)

Mit Depth Cueing werden Effekte wie Nebel, Dunst und Wasser generiert. Es benutzt Informationen aus dem Z-Buffer. Eine Farbe wird mit dem aktuellen Farbwert des Pixels aus dem Z-Buffer mit einem bestimmten Gewicht verrechnet. Dieses Gewicht hängt von der Distanz des Objektes ab.

Abbildung 9: Beispiel Depth Cueing

4.11 Level of Detail Management (LOD)

Beim LOD geht es darum, die Objekte die weit entfernt sind, mit einfacheren Polygonen darzustellen, in dem Ausmass, dass es der Beobachter nicht wahrnimmt. Damit wird einiges an Renderingleistung gespart. Es muss einerseits verhindert werden, dass Objekte in der Distanz einfach ins Bild springen (wegen der back clipping plane) und andererseits kann damit das Detailniveau für entfernte Objekte festgelegt werden. Im Zusammenhang mit Depth Cueing kann das "in die Sicht springen" der Objekte auch abgeschwächt werden. Beim Nintendo64 geschieht LOD automatisch.

4.12 Weitere Funktionen

Es gibt noch weitere Funktionen, die in der Hardware des Nintendo64 implementiert sind wie etwa: Alpha Transparenz, Dithering, Clipping, Culling und Rasterisierung. Diese helfen der Nintendo64 Leistung auch ganz gewaltig auf die Sprünge.

4.13 Sprites

Für Sprites (das sind 2D Bitmap-Grafiken) sind auch ein paar spezielle Effekte in der Hardware verankert, und zwar Skalierung, Rotation, Anti-Aliasing und Alpha Transparenz mit 256 Stufen.

5. Grafik-API

Über das Grafik-API ist leider nicht sehr viel bekannt, da diese Angaben sind leider nur für Entwickler offen.

Bei den grafischen Primitiven gibt es drei verschiedene Arten von Punkten: Punkte mit einer Farbe, Punkte mit einer Normalen und Punkte mit beidem. Diese können bei Bedarf benutzt werden. Die Darstellung der Objekte basiert aber hauptsächlich auf Dreiecken, denn hierfür sind extra Datentypen und low-level Operationen für Dreiecke im API gegeben. Diese Dreiecke können dann mit einer Shadingfunktion bearbeitet werden. Es wird nur das Gouraud-Shading-Modell unterstützt. Man kann zwar Flat-Shading anwenden, aber hinter der Kulisse wird einfach das Gouraud-Shading mit der gleichen Farbe für alle Eckpunkte angewandt.

Sprites können sogar auf eine eigene Library zugreifen, wo ihre Struktur und einige Funktionen definiert sind. Viele der Grafikeffekte können auch auf Sprites angewendet werden. So können sogar Texturen auf Sprites gelegt werden. An der Spriteunterstützung erkennt man sehr deutlich, dass der Nintendo64 hauptsächlich für Spiele konzipiert wurde.

Für die Transformationen werden homogene Koordinaten wie bei OpenGL benutzt, was dazu führt, dass mit 4x4 Matrizen gerechnet werden muss. Es gibt auch hier eine Modelviewmatrix und eine Projektionsmatrix, die neu geladen oder multipliziert oder auch in einem Matrixstack abgelegt und zurückgeholt werden können. Ich nehme an, dass sich SGI hier an ihrem OpenGL-Standard orientiert und je einen Stack für die Projektionsmatrizen und Modelviewmatrizen zur Verfügung stellt. Natürlich gibt es auch haufenweise Matrixfunktionen, die aber auch alle in Gleitkomma-Arithmetik ausgeführt werden können, falls der Anwender dies wünscht.

Beim Beleuchtungsmodell wird zur Zeit nur paralleles, gerichtetes Licht unterstützt. Es können bis zu sieben Lichtquellen definiert werden, wobei eine Quelle aus einer Lichtfarbe und einer Richtung zusammengesetzt wird. Zusätzlich kommt noch ein ambienter Anteil hinzu. Wird keine Lichtquelle definiert, so existiert mindestens ein ambientes Licht, das heisst, das mindestens eine Lichtquelle vorhanden ist.

Einzelne Grafikeffekte werden über definierte Funktionen aufgerufen, viele davon sind, wie wir schon gesehen haben, in der Hardware implementiert. Will man also die Modelle mit Effekten versehen, müssen nur die gewünschten Funktionen mit den richtigen Parametern aufgerufen werden. So sind auch Glanzpunkte und Nebel kein Problem.

Beim Nintendo64 wird, wie in OpenGL mit Displaylists und Viewports gearbeitet. Bei den Displaylists ist es also auch hier möglich, viele Funktionsaufrufe zu gruppieren, und so zum Beispiel Objekte, die man immer wieder braucht, einfach durch einen Displaylist-Call ausführen zu lassen.

Man hat das Gefühl, dass sich SGI bei diesem API sich sehr stark an ihren anderen Entwicklungen wie OpenGL und RealityEngine abstützt. Es würde mich nicht wundern, wenn noch viel mehr Ähnlichkeiten gefunden werden können.

6. Vorteile – Nachteile - Ausblick

Vorteile

Dieses System beinhaltet auch viele Erfahrungen und schon bekannte Konzepte von SGI, was man vor allem beim API sieht, das viele Ähnlichkeiten mit OpenGL hat. Jedoch ist ihr Know-how zum ersten mal in einem Tiefpreissegment benutzt worden und bietet ein sensationelles Preis/Leistungs-Verhältnis. Damit wird leistungsstarke Grafikhardware einem breiten Publikum zur Verfügung gestellt, was sich auf eine verblüffende Realitätsverbesserung der Spiele auswirkt. Es wurde eindeutig gezeigt, dass die Grafikfunktionen des Nintendo64 das Herz dieser Konsole sind und ihm einen gewaltigen Leistungsschub geben, da diese komplett in der Hardware implementiert sind.

Nachteile

Ein grosser Nachteil dieser Konsole ist, dass der Absatzmarkt für ein Spiel sich stark einschränkt, wenn der Hersteller das Spiel speziell an der Hardware einer Konsole ausrichtet. Die Kosten für eine Portierung auf ein anderes System sind dann sehr hoch. Will er rentabel arbeiten, wird er wahrscheinlich das Spiel nicht auf ein einziges System abstimmen, sondern Kompromisse zu Gunsten der Portabilität machen, wodurch das volle Potential der Grafikhardware wahrscheinlich nicht ausgenutzt wird.

Als weiteres benutzt der Nintendo64 nur Cartridges, was die Grösse eines Spieles stark einschränkt (<32 MB). Dadurch können keine grossen Einleitungsanimationen in ein Spiel einfliessen, was in heutiger Zeit doch fast überall üblich ist.

Ausblick

Detaillierte Informationen über die Grafikpipline des Nintendo64 in Erfahrung zu bringen, ist eine sehr schwierige Aufgabe. Die Firma bewacht dieses Geheimnis wie einen riesigen Schatz, was es ja eigentlich auch ist, da es ihr Wettbewerbsvorteil zu den anderen Spielkonsoleherstellern darstellt. Hätten die anderen Firmen diese Informationen, so wäre Nintendo wahrscheinlich nicht mehr so lange an der Spitze.

Es wird auch von mehreren Programmierern versucht in der Freizeit, Emulatoren des Nintendo64 zu programmieren. Diese Projekte sind meistens in der Anfangsphase. Ein Projekt ist "Project Unreality". Es existiert schon in der Version 0.3 aber bisher wurden noch keine Funktionen des RCP implementiert. Falls sie mal soweit kommen, werden sie glaube ich, sofort merken, dass da die Leistung eines Pentium nicht mehr ausreicht, um diese komplexen Grafikoperationen zu emulieren. Spielen wird man mit diesen Emulatoren daher wahrscheinlich nie in Echtzeit können.

Ein Vergleich mit anderen Spielekonsolen auf Basis der Polygonzeichnungsrate ist hier nicht aussagekräftig, da der Nintendo64 Funktionen hat, die die anderen Konsolen nicht aufweisen. Auch ist überhaupt nichts über die Implementierung der Speicher bekannt, was ein Vergleichen unsinnig macht. Es ist so, als ob man Äpfel mit Birnen vergleicht.

Trotzdem bleibt die Hardware der entscheidende Schlüssel, um realistische Bilder in Echtzeit auf den Bildschirm zu bekommen, und der Nintendo64 scheint dieses Potential zu haben.