InfiniteReality: Ein Real-Time-Grafiksystem

Vortrag am GDV-Seminar
ETH Zürich

Andreas Deller
18. Mai 1998

Outline

I. Einführung
- A. Grundinformationen über InfiniteReality
- B. Eckdaten der InfiniteReality
- C. Zum Vortrag

II. Architektur
- A. Übersicht
- B. Detailbeschreibungen
  - 1. Geometry Board
    - a. Host Interface
    - b. Geometry Distributor
    - c. Geometry Engines (2 oder 4 pro Pipeline)
    - d. Geometry Raster FIFO
  - 2. Raster Memory Board (1, 2, oder 4 pro Pipeline)
    - a. Fragment Generators
    - b. Image Engines
  - 3. Display Generator Board

III. Features
- 1. Virtual Texture
- 2. Scene Load Management

IV. Host-Systeme
- A. Systembeschreibung
  - 1. Node Cards
  - 2. Beispielkonfigurationen

V. Performance

VI. Endnoten/Literaturangaben

VII. Index

Einführung

Grundinformationen über InfiniteReality

Das InfiniteReality(IR)-Grafiksystem von SGI (Silicon Graphics, Inc.) ist das erste Workstation-System, das für gleichmässige, garantierte Hertz-Raten entwickelt worden ist. Als Nachfolger der RealityEngine wird es in den Hochleistungs-Grafikcomputern Onyx und Onyx2 von SGI eingesetzt. Das Design musste derart angepasst werden, dass die IR-Engine sowohl von den Vorzügen der verbesserten Systemarchitektur der Onyx2 profitieren als auch in der Onyx optimal eingesetzt werden kann. Zudem sollte sie skalierbar sein, um den verschiedenen Kundenwünschen möglichst gut zu entsprechen und einen benötigten Ausbau ohne grossen Aufwand zu erledigen.

Das IR-Grafiksystem wird als «*third-generation graphics system*» bezeichnet. Dies bedeuetet, dass das System beleuchtete, schattierte, tiefengepufferte, texturierte, antialiased Dreiecke hardwareunterstützt rendern kann, und zwar mit möglichst wenig Performance-Einbusse.

Weiter sollten Texturen eine praktisch unbegrenzte Grösse haben dürfen, ohne die Performance erheblich zu beeinflussen. Der Programmierer sollte mit dem Handling der Textur so wenig Zeit wie möglich aufbringen müssen.

Eckdaten der InfiniteReality

- 48-bit RBGA-Farbe
- 23-bit oder 15bit/16bit compressed floating point Z-buffering
- 4 Geometry Engines (GE)
- 1-4 Raster Managers (RM)
- 1-8 IR-Pipelines parallel einsetzbar
- «Garantierte» Frameraten (z.B. 50 Hz, 60 Hz, 96 Hz, 120 Hz)
- Hardwareunterstütztes OpenGL
- Geometry rate: >6.1 mio beleuchtete, texturierte, antialiased Pixel/s (4 RM, 50 Pixel/Dreieck)
- Fill rate: 710 mio texturierte, antialiased, Z-buffered Pixel/s (4 RM)
- 16-64 MB texture memory pro Pipeline
- 80-320 MB frame buffer size pro Pipeline
- 2-8 Videokanäle pro Pipeline
- Auflösungen bis 1600*1200@60Hz bzw. 1920*1080@72 Hz (HDTV) oder 1920*1200@66 Hz pro Videokanal

Onyx2 (Host-System):

- 2-64 MIPS R10000-Prozessoren mit je 1-4 MB secondary cache
- 64 MB - 128 GB RAM
- 4.5 GB - 400 GB Festplattenkapazität (UW SCSI oder Fibre Channel)

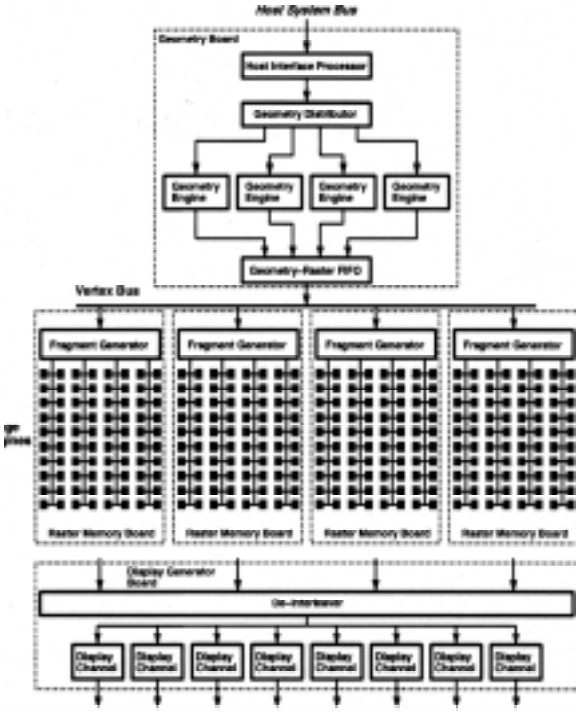
Zum Vortrag

Dieser Vortrag wird vor allem die Systemarchitektur behandeln und die Fähigkeiten der IR aufzeigen. Nicht behandelt werden aber Bildverarbeitungsmöglichkeiten wie Farbraumkonvertierungen, Konvolutionen, Warping etc. Der Grund liegt primär an den nicht vorhandenen Unterlagen zu diesem Thema. Zudem würden diese Erläuterungen den Rahmen dieser Arbeit sprengen.

Bei der Bearbeitung der Hardware ist eine gewisse Vorsicht geboten: Da die Unterlagen des Herstellers (SGI) an unterschiedlichen Daten geschrieben wurden, beherbergen sie auch andere technische Daten. Dies macht die Erstellung einer konsistenten Beschreibung nicht einfacher. Ich habe mich bemüht, überall die jeweils neuesten verfügbaren Daten zu verwenden.

Architektur

Übersicht



Um den Ablauf zu verstehen, verfolgen wir ein Dreieck, wie es gerendert wird:

Die Positions-, Farben-, Normalen- und Texturbefehle werden vom *Host Interface Prozessor* per DMA über den *Host System Bus* geladen und dem *Geometry Distributor* weitergeschickt. Dieser teilt den Datenstrom auf und beliefert die verschiedenen *Geometry Engines.* Dort werden die Koordinaten transformiert zu Kamerakoordinaten, beleuchtet, rotiert/translahiert/skaliert, geclippt und in Fensterkoordinaten transformiert. Die Daten der verschiedenen GE werden vom *Geometry Raster FIFO* in der richtigen Reihenfolge wieder gesammelt und an den *Vertex Bus* weitergeleitet.

Von dort liest jeder *Fragment Generator* seine Vertex-Daten, setzt sie zu einem Dreieck zusammen, holt die Textur aus der erhaltenen Adresse, führt unter anderem Farbinterpolation, Z-Buffer-Tests, Alpha-Blending, Anti-Aliasing und Scan-Konvertierung durch. Für das Anti-Aliasing wird jeder Pixel in 64 (kleinere Systeme: 16) Subpixel eingeteilt. Bei der Farbberechnung wird die Farbe und der Z-Wert an einem, vier oder acht Subpixel-Stellen ausgerechnet und erst am Schluss aller Berechnungen zu einem Pixelwert verschmolzen. Die resultierenden Daten werden darauf gleichmässig auf die *Image Engines* verteilt, die jeweils 1 MB des Framebuffers darstellen.

Die Image Engines leiten ihre Daten über dedizerte Leitungen an das *Display Generator Board* weiter. Dort werden die Daten vom *De-Interleaver* in der richtigen Reihenfolge zusammengesetzt, die Farben für den Output konvertiert und an einen *Display Channel* weitergeleitet, der die Daten in analoge Signale wandelt und zum Darstellungsgerät (Bildschirm, Kamera...) schickt.

Detailbeschreibungen

Geometry Board

Host Interface

Dieses muss sowohl mit einer Onyx als auch einer Onyx2 funktionieren. Der grösste Unterschied der beiden Systeme besteht darin, dass das neuere Modell die Daten statt mit 220 MB/s mit 320 MB/s liefert. Zudem ist das Shared-Memory-Modell unterschiedlich (Onyx2: *ccNUMA = cache coherent non uniform memory access*). Als «Host» wird in diesem Zusammenhang das System bezeichnet, in welches die IR eingebettet wird, also Onyx oder Onyx2.

Die Lösung dieses Kompatibiltätsproblems wird mit den *Display Lists* erreicht:

Der Host speichert die Display Lists in seinem Speicher., auf welchen der Host Interface Processor via DMA (direct memory access) zugreifen kann. Die bei der Onyx möglichen 220 MB/s reichen aber nicht aus, um die Grafikpipeline zu füllen. Deshalb sind an den Host Interface Processor 16 MB SDRAM angeschlossen, worin gewisse Display Lists zwischengespeichert werden. Die Entscheidung für ein Zwischenspeichern wird aufgrund der Grösse und einer in OpenGL spezifizerbaren Priorität gefällt. Das Geometrie-Subsystem liest aus diesem Cache mit etwa 300 MB/s.

Geometry Distributor

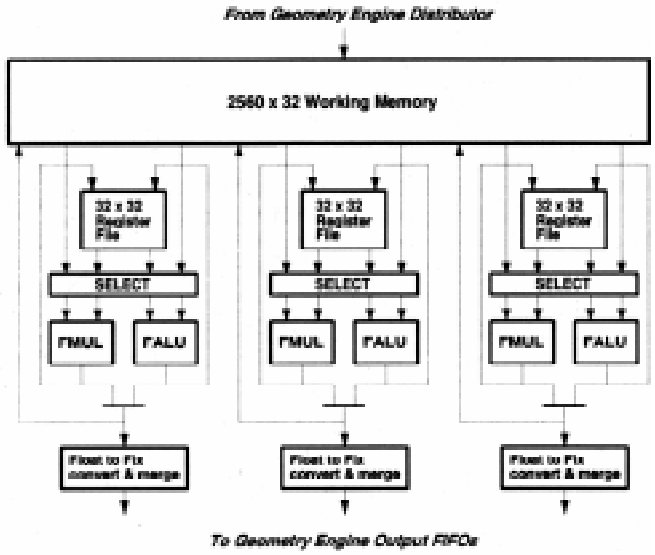
Er interpretiert die hereinkommenden Daten und verteilt sie möglichst gleichmässig auf die 2 oder 4 Geometry Engines. Er beherrscht sowohl das von RealityEngine bekannte Round-Robin- und das Least-Busy-Schema. Die zweite Methode bringt jedoch nicht grosse Geschwindigkeitsverbesserungen.

Damit die Originalsequenz wieder hergestellt werden kann, wird jedem Befehl ein Identifier zugewiesen.

Geometry Engines (4 pro Pipeline)

Die Geometry Engines sind verantwortlich für geometrische Operationen wie Rotation, Zoom, Warp, Konvolution, Skalierung, Real-Time-Morphing und weitere.

Um den gehobenen Anforderungen gerecht zu werden, sind diese mit *ASIC*s (*application specific integrated circuit*) implementiert. Die Anordnung sieht folgendermassen aus:



Wir sehen einen *SIMD*- (*single instruction multiple data*) Datenpfad mit drei Floating-Point-Einheiten, die je mit einer ALU, einem Multiplizierer und einem Registerfile bestückt sind. Das *Working Memory* beinhaltet den OpenGL-Zustand und dient als

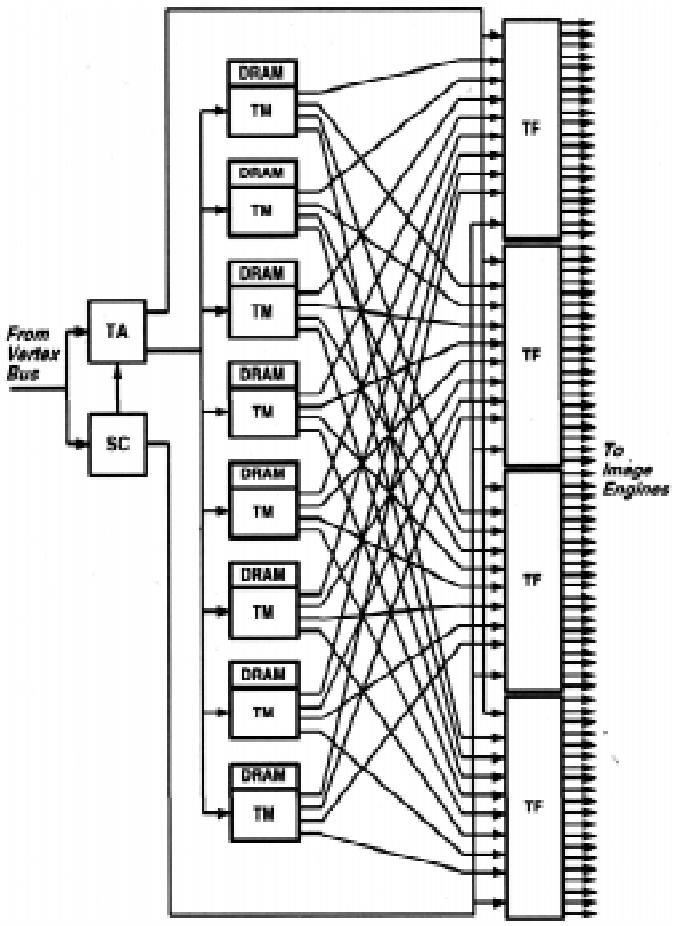
Zwischenspeicher für Kalkulationen und neu hereinkommende Daten. Jede Floating-Point-Einheit kann pro Zyklus zwei Lese- und eine Schreiboperation ausführen. Da der Speicher gemeinsam ist, können die Floating-Point-Einheiten ihre Daten einfach untereinander austauschen. Für häufige OpenGL-Operationen wurden spezielle Microcode-Module geschrieben, um die Verarbeitung zu beschleunigen. In einer Tabelle im Working Memory sind die aktiven Microcode-Module aufgelistet. Die Verarbeitungszeit pro Vertex verändert sich auf diese Weise langsam und voraussehbar.

Geometry Raster FIFO

Aufgrund der Identifier, die vom Geometry Distributor vergeben wurden, kann hier die Originalreihenfolge der Primitiven wiederhergestellt werden. Die in SDRAM implementierte FIFO hat Platz für 2^{16} Vertices, die sie als nächstes auf den Vertex Bus schreibt.

Raster Memory Board (1, 2, oder 4 pro Pipeline)

Fragment Generator



Hier werden die Vertices gesammelt und wieder zu Dreiecken zusammengestückelt. Der *Scan Converter* (SC) und der *Texel Address Calculator* (TA) erledigen die Scan-Konversion, Farb- und Tiefeninterpolation (Z-buffering) sowie perspektifisch korrigierte Textur-Adressberechnung. Diese Adressen werden an die *Texture Memory Controller* (TM) weitergeleitet, wo in den angehängten jeweils 4 SDRAM die Texel-Werte ausgelesen werden. Diese werden weitergeleitet an die *Texture Filtering* ASICs (TF), wo die Textur mit der im Scan Converter interpolierten Farbe kombiniert und wenn nötig mit Hilfe des Z-Wertes noch Nebel dazugemischt wird.

Interessantes Detail: Die Textur ist nicht redundant in den total 32 SDRAM (pro Board) der TM vorhanden, sondern nur genau ein Mal. Deshalb müssen von allen SDRAM alle TF erreichbar sein.

Image Engines

Der Output eines Fragment Generators wird gleichmässig an die 80 angehängten Image Engines verteilt. Jede Image Engine kontrolliert 256K * 32 bit SDRAM, was schlussendlich 80 MB pro Raster Memory Board ergibt. Damit lässt sich in einem RM z.B. ein 1920*1080-Display mit 256 bits/Pixel speichern.

Die Gründe für das gleichmässige Verteilen sind folgende: optimale Ausnützung der möglichen Parallelität, geringere Fluktuationen bei den Image Engines benötigen kleinere FIFOs, Restriktionen bezüglich Chip- und Board-Design.

Um die Last gleichmässig in den *Fragment Generators* (FG) zu verteilen, wird jedem FG eine vertikale, zwei Pixel breite Linie zum berechnen zugewiesen. Da ein Bild auf dem Monitor horizontal aufgebaut wird, werden so alle FG gleichmässig belastet und brauchen kleinere FIFOs.

Display Generator Board

Unabhängig von der Anzahl Raster Memory Boards wird dieses Board mit 1.2 GB/s gefüttert. Die hereinkommenden Pixel werden «de-interleaved», d. h. in die richtige Reihenfolge gebracht, und die Farben werden für den analogen Output auf 8 bit (pro Farbe) heruntergerechnet. Dafür wird Gebrauch von einer *Color Lookup Table* (CLUT) gemacht, die 2^{15} Einträge haben kann. Es existiert aber auch ein digitaler Output mit 12 bit pro Farbe.

Das Minimum-System schickt die Daten zu zwei *Display Channels*, möglich sind aber bis 8, wobei jeder autonom ist, d.h. eigenes Video-Timing und Auflösung (auch selber definierte und programmierte) haben kann. Ein Feedback-Pfad führt zurück zum Raster Memory Board, um Daten über Video-Timing-Anforderungen und Resizing zurückzumelden.

Der *DAC* (*digital to analog converter*) wandelt die digitalen Daten schliesslich in analoge um, die an den Bildschirm (oder ein anderes Gerät) geschickt wird. Die DACs der zwei ersten Video Channels sind mit 220 MHz getaktet - pro Farbkomponente (8 bit). Die DACs der restlichen Video Channels (Kanäle 2-7) sind mit 170 MHz getaktet.

Um die verschiedenen Kanäle von aussen zu synchronisieren, gibt es verschiedene Varianten:

- **Framelock**

  Die Frameraten verschiedener Kanäle werden miteinander synchronisiert mittels «line rate dividers». Dies funktioniert aber nur, wenn die kleineste Hertz-Zahle der ggT aller Frequenzen ist (z.B. 60 Hz und 180 Hz). Es ist nicht garantiert, dass kein Frame verloren geht.

- **Genlock**

  Wie Framelock, aber die Frequenzen müssen dieselben sein. Nach jeder horizontalen Linie werden die verschiedenen Kanäle synchronisiert. So geht garantiert kein Frame verloren.

- **Swap ready wire**

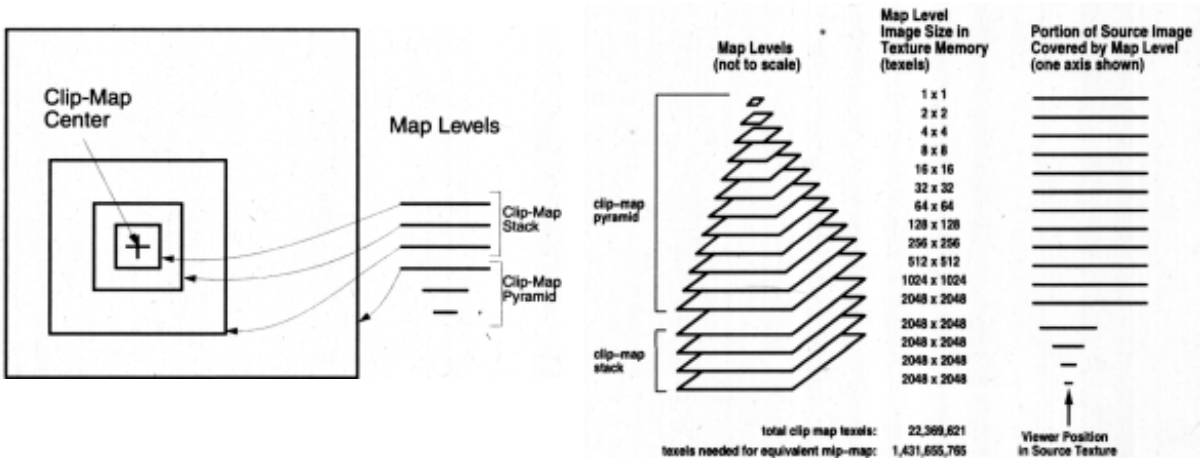
  Hiermit werden verschiedene Grafiksysteme mit einer AND-Konfiguration zusammen verbunden. Alle Grafiksysteme richten sich nach dem langsamsten.

Features

Virtual Texture

Heutige Simulationen verlangen riesige Texturen. Beispiel: Ein Satellitenbild, das die ganze Schweiz in einer 24 Meter Auflösung in 36 bit zeigt, benötigt knapp 308 MB Speicher - auch für eine IR mit maximal 64 MB Texturspeicher zu viel. Da der physische Texturspeicher dazu nicht ausreicht, stellt die IR virtuelle Texturen (*virtual textures*) zur Verfügung. Dies wird mit einer sog. «*clip-map*» erreicht.

Der Gedanke ist, dass bei einer gegebenen Bildschirmauflösung die Texturauflösung nach dem Theorem von Nyquisthöchstens das Doppelte sein muss. (Dies gilt auch für Musik; deshalb wird auf einer Musik-CD mit 44.1 kHz gesampelt.) Für ein 1024*1024-Bild sind also höchstens 2048*2048 Pixel für die Textur nötig. Jetzt kommt die Clip-Map ins Spiel:



Das Clip-Map-Zentrum ist der momentane Fokus. Von dort aus wird die Texturgrösse beibehalten, aber immer ein viermal so grosses Gebiet betrachtet. Dieses Verfahren wird fortgeführt, bis man die ganze Textur in den 2048*2048 Pixeln darstellen kann. Die erhaltenen «Zwischentexturen» bilden den *Clip-Map-Stack*. Jetzt wird das Gebiet beibehalten, aber die Auflösung der Textur wird bei jedem Schritt verkleinert, bis 1*1. Diese «Zwischentexturen» bilden die *Clip-Map-Pyramide*. Sie zeigen immer dasselbe Gebiet, aber in immer gröberer Auflösung. Die Texturen dieser Pyramide sind auch unter dem Namen «*mip-maps*» bekannt und können verschieden interpoliert werden: linear, bilinear, trilinear (für 2D-/3D-Texturen), quadlinear (für 3D-Texturen), bikubisch (für 2D-Texturen).

Wenn das Clip-Map-Zentrum nun wandert, müssen alle diese «Zwischentexturen» ebenfalls laufend angepasst werden. Für die feinen Clip-Map-Level ist diese Anpassung am schnellsten, die gröberen werden langsamer angepasst. Weil spezielle Offset-Register für die Neuberechnung der Adressen gebraucht werden, müssen jeweils nur ein paar Pixel neu geladen werden.

Falls die Update-Rate zu tief ist und die benötigte Textur nicht in der gewünschten Auflösung vorhanden ist, merkt dies das Textur-Subsystem und braucht stattdessen die Textur in einer gröberen Auflösung, bis die benötigte geladen ist. Es wird also im Clip-Map-Stack eine Ebene nach oben gewandert. Die Textur ist zwar so schlechter aufgelöst, aber sie bedeckt dafür das ganze aktuell gebrauchte Gebiet.

Dieses Verfahren stellt zwar hohe Ansprüche an die Hardware als andere, doch die Resultate sind dafür zufriedenstellender als in anderen Systemen, wo grosse Texturen in Regionen aufgeteilt und auf die darunterliegende Geometrie gemappt werden müssen. Bei der IR muss sich der Programmierer nicht um dieses Problem kümmern.

Die Geometry Raster FIFO besitzt einen separaten Pfad, durch welchen sie Texturdaten vom Host Memory mit maximal 176 MB/s zum Vertex Bus durchschleusen kann, wenn die Fragment Generators gerade mal stark beschäftigt sind. Der adressierte TM (Texture Manager) im Fragment Generator unterbricht seine Zeichenarbeit kurz, um aus seiner FIFO die neuen Texturdaten zu lesen. Dies beeinträchtigt die Performance unwesentlich, da der Vertex Bus um eine Grössenordnung langsamer ist als die

FIFO des TM. Durch geeignete Primitive wird sichergestellt, dass auf eine Textur erst zugegriffen wird, wenn sie komplett geladen ist und eine momentan noch gebrauchte Textur noch nicht überschrieben wird.

Scene Load Management

Es kann trotz aller Hardware-Ressourcen passieren, dass entweder die *Geometry Rate* oder die *Fill Rate* am Anschlag ist. Über einen Feedback-Pfad wird eine solche Situation frühzeitig erkannt und der Anwendung zurückgemeldet.

Falls die Pipeline Geometrie-limitiert ist, wird normalerweise von der Anwendung die Komplexität der Scene temporär reduziert. Dabei werden meistens weit entfernte Objekte weggelassen, so dass der Output nicht stark verschlechtert wird.

Falls jedoch die Füllrate das Problem ist, kommt das «*dynamic video resizing*» zum Zug: Bei jedem Frame werden Pipeline-Statistiken gesammelt um herauszufinden, ob die Fill Rate schon knapp erreicht ist. Je nach Resultat wird das nächste in einer geringeren Auflösung in den Framebuffer gerendert; X- und Y-Skalierung können voneinander unabhängig verändert werden. Das Bild benötigt dann weniger Bandbreite, um vom Display Generator Board geladen zu werden. Vor dem DAC wird es dann mit bilinearer Interpolation wieder auf die benötigte Auflösung raufskaliert. Allerdings funktioniert das Resizing nur bei Videomodi mit einem Pixeltakt von 120 Mhz oder weniger.

Bei beiden Methoden gilt: sobald wieder genügend Ressourcen vorhanden sind, werden die Änderungen wieder rückgängig gemacht.

Natürlich kann auch durch diese zwei Tricks keine konstante Framerate garantiert werden. Irgendwann ist selbst der IR eine Szene zu komplex und sie ruckelt. Die vorgestellten Massnahmen heben aber das Limit für konstante Frameraten weiter nach oben. Es kann schliesslich auch nicht das Ziel sein, eine völlig garantierte Framerate zu erzeugen, wenn der Output schlussendlich nur noch aus groben Dreiecken besteht. Hier muss ein Kompromiss eingegangen werden.

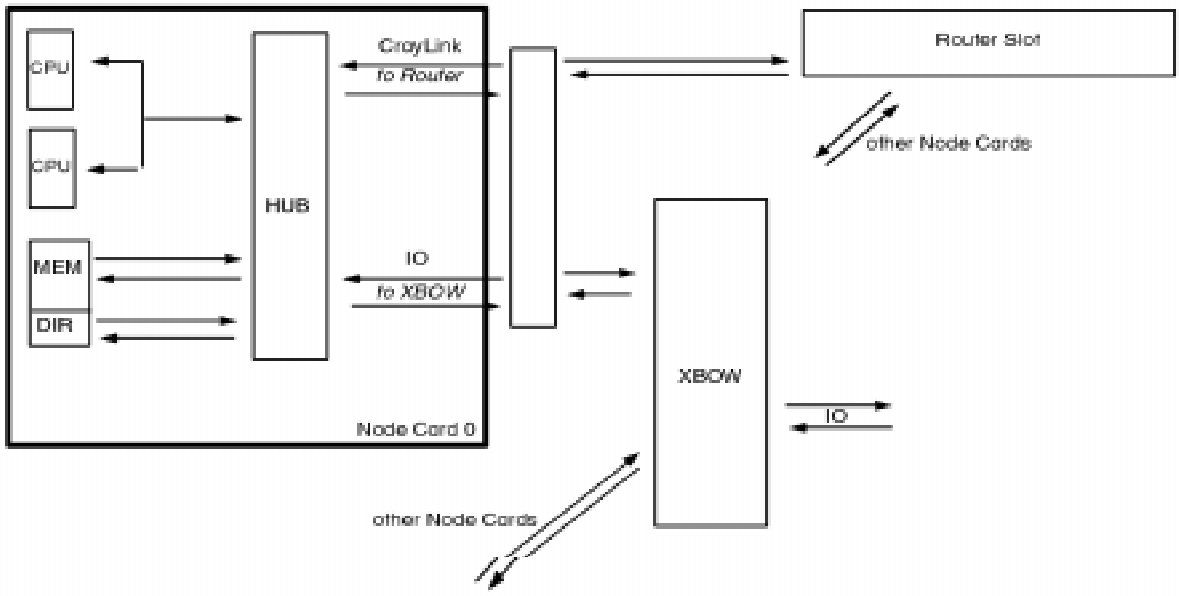
Host-Systeme

Systembeschreibung

Wie in der Einführung schon erwähnt, kann die IR-Engine in den Host-Systemen Onyx und Onyx2 eingesetzt werden. Da die Leistung eines Grafiksubsystems massgeblich durch das Host-System beeinflusst wird, beschreibe ich hier kurz den Aufbau einer Onyx2.

Node Cards

Die *Node Card* ist der Basis-Block einer Onyx2. Auf ihre befinden sich eine oder zwei CPU (MIPS R10000), 1 oder 4 MB L2-Cache, bis zu 1 GB RAM aufgeteilt in Hauptspeicher und «*directory memory*», das für die Cache-Kohärenz mit anderen Node Cards gebraucht wird, ein *Hub-ASIC*, ein IO-Interface und ein *CrayLink* als Verbindung zu anderen Node Cards:



Eine CPU adressiert den gesamten globalen Speicher, wie wenn er bei ihr selber wäre (ein globaler Adressraum). Der Hub ist ein 4-Port-Crossbar-Switch mit 4 bidirektionalen Ports mit je 800 MB/s (Peak) pro Richtung. Er kontrolliert die Node-Card-interne Kommunikation und die externe Kommunikation. Er ist auch verantwortlich für die Cache-Kohärenz.

Die IO-Geräte sind ebenfalls von jeder Node Card aus allozierbar, via den *XBOW*. Dieser ist ein dynamischer ASIC Crossbar-Switch mit 8 16-bit-Ports. 6 Ports werden gebraucht als Verbindung zu IO-Geräten, und zwei als Verbindung zu Node Cards. Je nach Bedarf verbindet der XBOW die gewünschten IO-Ports oder Node Cards, mit bis zu 800 MB/s pro Richung (Peak).

Die Router-Links sind alle Point-to-Point-Verbindungen über die Routers. Werden neue Node Cards eingefügt, steigt die Zahl der Router und CrayLinks linear, so dass die *Bisektionsbandbreite* ebenfalls linear wächst. Jedes Node-Card-Paar ist über mindestens zwei unterschiedliche Routen erreichbar. Auch hier beträgt die Geschwindigkeit zwischen den Routern 800 MB/s in jede Richtung. Die Router selber sind 6-Port-Crossbar-Switches.

Beispielkonfigurationen

Da heute in erster Linie die Onyx2 interessant ist, gebe ich für ein Low-End- und das High-End-Modell ein paar Daten an, basierend auf *. Die angegebenen Werte sind natürlich nur eine kleine Auswahl der erhältlichen, zusätzlich können weitere Optionen eingesetzt werden (PCI- und XIO-Karten).

Um die beeindruckende Leistung der IR-Engine weiter zu steigern, können mehrere solche Pipelines in ein Host-System eingebaut werden. Mehrere in Racks untergebrachte Subsysteme können mit dem CrayLink zusammengehängt und im sogenannten «*MonsterMode*» betrieben werden. Der Overhead ist sehr klein, so dass die Grafikleistung praktisch linear mitwächst. Dieser Modus wird im zweiten unten aufgelisteten Beispiel gebraucht, wo 8 IR-Pipelines parallelgeschaltet sind.

Als weitere Variante können Rack-Konfigurationen auch im *GroupMode* betrieben werden, d.h. ein Rack treibt verschiedene Arbeitsplätze, jeder mit separatem Keyboard und einem oder mehreren Monitoren.

Die Multichannel-Ausgabe kann gebraucht werden für Stereo-Bild-Erzeugung, Grossleinwände oder Simulatoren mit grossem Gesichtsfeld. Jeder Display Channel ist für einen Teil des Gesamtbildes bzw. für den linken oder rechten Stereo-Bildteil zuständig.

Zur Performance habe ich im nächsten Abschnitt (<Performance>) einige weitere Informationen.

	Onyx2 InfiniteReality Deskside (1/2 RM)	**Onyx2 IR Multirack with RealityMonster Software**
Polygons/sec (*)	11M	80M
Pixel fill, smooth, Z	224M/448M	5.3G
Pixel fill, smooth, AA	194M/388M	4.7G
AA vectors/sec	3.7M	60G
Voxels/sec	200M/400 M	4.8G
Color	48 bit RGBA	48 bit RGBA
Bits/pixel	256-2048	256-2048
Graphic pipelines	1	1 to 8
GE/pipeline	4	4
RM/pipeline	1 to 2	1 to 2 and 1 to 4
Texture mem/pipeline	16 or 64 MB	up to 512 MB combined
Frame buffer/pipeline	80 MB	80 to 320 MB
Display channels/pipeline	2 to 8	2 to 8
CPU (MIPS R10000)	1 to 4	2 to 64
RAM	256 MB to 8 GB	256 MB to 128 GB
Internal disk storage	1 to 5 *4.5 or 9.1 GB	1 to 11 * 4.5 or 9.1 GB per rack (max. 8 racks)
Power	2500 watts	7000 watts per rack
IO	UW SCSI (40 MB/s)	UW SCSI or Fibre Channel (100 MB/s)
Network	Ethernet, HIPPI, FDDI, ATM	Ethernet, HIPPI, FDDI, ATM

(*) 25 Pixel/Dreieck, unbeleuchtet, untexturiert, RGBA 10/10/10/0, Z-buffered (23 bit), AA (4 Samples)

Legende:		
	smooth	Gouraud-shaded
	Z	Z-buffered = depth buffered
	AA	anti-aliased
	RGBA	red, green, blue, alpha
	GE	Geometry Engine
	RM	Raster Manager
	UW	ultra wide
	HIPPI	HIgh Performance Parallel Interface (800 MB/s)
	FDDI	Fiber Distributed Data Interface (100 Mb/s)
	ATM	Asynchronous Transfer Mode (155.5 Mb/s)

Performance

Benchmarking ist wie in jedem Gebiet der Computerbranche eine unbefriedigende Sache. Zu häufig werden Werte gemittelt, um dann als Testresultat dazustehen, oder es werden spezifische Werte, in denen das eigene Produkt brilliert, besonders hervorgehoben. Die schlechten Werte werden natürlich nie publiziert. Deshalb sind veröffentlichte Werte immer mit Vorsicht zu geniessen.

Trotzdem möchte ich ein paar Werte vorstellen, nämlich Füllraten und Geometrieraten (fill rate und geometry rate), da ich diese speziell im Kapitel «Features» angesprochen habe. Zu diesem Zweck übernehme ich die Werte aus dem betrachteten Paper [2] :

- **Non fill-limited geometry rates:**

Unlit, untextured tstrips	11.3 Mtris/s
Unlit, textured tstrips	9.5 Mtris/s
Lit, textured tstips	7.1 Mtris/s

«tris» sind Dreiecke (triangles). Die Werte sind etwa 7-8 Mal höher als beim Vorgänger RealityEngine. Die Länge der Strips ist wahrscheinlich 10.

- **Non geometry-limited fill rates (4 Raster Memory Boards):**

Non-depth buffered, textured, AA	830 Mpix/s
Depth buffered, textured, AA	710 Mpix/s

Die Werte sind etwa 3 Mal höher als beim Vorgänger RealityEngine.

- **Geometry rates for short triangle strips**

Length 2 triangle strips	4.7 Mtris/s
Length 4 triangle strips	7.7 Mtris/s
Length 6 triangle strips	8.6 Mtris/s
Length 8 triangle strips	9.0 Mtris/s
Length 10 triangle strips	11.3 Mtris/s

Aus diesen Zahlen ist ersichtlich, wie stark der Overhead für die Bearbeitung eines Triangle-Strips ist.

Besonders herauszuheben ist die Tatsache, dass die erwähnten Werte für *eine* Grafik-Pipeline gilt. Falls mehrere verwendet werden (MonsterMode), werden diese Werte entsprechend fast linear multipliziert. Die Beispielkonfigurationen im vorherigen Kapitel zeigen dies deutlich.

Endnoten

[1] Onyx2 Technical Specifications, (c) SGI 1998

[2] InfiniteReality: A Real-Time Graphics System; John S. Montrym, Daniel R. Baum, David L. Dignam, Christopher J. Migdal; SiliconGraphics Computer Systems

Literaturangaben

[3] Homepage von SGI: www.sgi-europe.com bzw. www.sgi.com
[4] Onyx2 MXI Datasheet, (c) SGI 1996
[5] Welcome to Reality, Onyx2 Visualization Supercomputers, Product Guide, (c) SGI 1996
[6] Onyx2 Scalablae Visualization Supercomputers, Product Guide, (c) SGI 1998

Einen besonderen Dank möchte ich Urs Meyer, System Administrator von SGI Schlieren, aussprechen, der mir einige knifflige Fragen beantwortet hat.

Index