Rendering voor PDA of GSM

Rafael KLIMAS

promotor :
Prof. dr. Wim LAMOTTE

Eindverhandeling voorgedragen tot het bekomen
van de graad Licentiaat in de Informatica,
afstudeervariant Multimedia

Rendering voor PDA of GSM

Thesis voorgedragen tot het behalen van de graad van master in de informatica/ICT/kennistechnologie

Rafael Klimas

Promotor: Prof. dr. Wim Lamotte
Begeleider: Tom Jehaes

Academiejaar 2005-2006

Abstract

In vele gevallen kan data door middel van 2D of 3D beelden verduidelijkt of aantrekkelijker gemaakt worden. Ook games worden al snel interessanter met mooie, gedetailleerde 3D beelden. Door de snelle vooruitgang in technologie lenen mobiele toestellen zich meer en meer tot het renderen van zulke beelden. Toch blijven mobiele toestellen zoals GSM's en PDA's vandaag de dag nog beperkt in rekenkracht, opslagcapaciteit, schermgrootte en batterijduur. Dit maakt complexe beelden renderen op deze toestellen interessant en uitdagend.

Deze thesis bestaat uit twee delen: een literatuurstudie en enkele implementaties. De literatuurstudie onderzoekt welke ontwikkelingen en technologieën er momenteel bestaan voor PDA's en GSM's. Er wordt onder andere gekeken naar de besturingssystemen en de "state of the art" processor architecturen. Deze worden vervolgens onderling vergeleken. Verder kijken we naar mogelijke renderingstechnieken voor deze toestellen, ontwikkelingssoftware zoals API's en SDK's en reeds gerealiseerde 3D toepassingen.

Het implementatie-gedeelte bestaat ook uit twee delen: een case study waarin we een efficiënt algoritme ontwikkelen om in first-person view door de stad Hasselt te navigeren enerzijds en het uittesten van enkele besproken SDK's anderzijds. Met de case study tonen we aan dat een efficiënt algoritme altijd nodig is omwille van geheugenbeperkingen en om bevredigende framerates te behalen. Preprocessing en het dynamisch inladen van textures vormen de kern van deze case study. Bij het uittesten van de SDK's tonen we aan dat het niet evident is om te programmeren voor mobiele toestellen en dat er vele verschillen zijn met de ontwikkeling van applicaties voor gewone desktop PC's.

Woord Vooraf

Ik zou graag mijn promotor Wim Lamotte willen bedanken voor de kans om dit onderwerp aan te vatten. Verder wil ik Tom Jehaes bedanken voor de nauwe opvolging en feedback gedurende mijn werk. Ook wil ik hen bedanken voor de interessante case study horende bij deze thesis en voor stage die ik heb mogen lopen ter voorbereiding van deze thesis.

Verder wil ik mijn ouders en vriendin bedanken voor de steun gedurende de periode van dit werk, mijn zus voor een grammatica- en spellingscontrole en mijn nonkel Billy Klimas voor het uitlenen van zijn PDA. Ook wil ik mijn vrienden en vriendin bedanken voor hun geduld. Als laatste wil ik nogmaals mijn ouders bedanken voor de kans en vrijheid om deze studies aan te vatten.

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

Hoofdstuk 1

Inleiding

De laatste jaren hebben we een enorme groei gekend van complexe virtuele 3D beelden op de alom aanwezige computers die we nu gebruiken voor ons werk, in het dagelijkse leven en voor ontspanning. Door de stijgende vraag naar gedetailleerde, realistische beelden van hoge kwaliteit, bestaan zulke beelden al gemakkelijk uit duizenden tot miljoenen primitieven. Gelijklopend met deze groei worden zogenaamde "mobile display devices" zoals PDA's en GSM's veelzijdiger en krachtiger, wat hun bruikbaarheid voor het renderen van deze complexe virtuele beelden aanzienlijk verhoogt.

Complexe of gedetailleerde beelden renderen op mobiele toestellen is uitdagend om verschillende redenen: we hebben te maken met trage processoren, weinig geheugen en kleine displays. Ook werken deze toestellen op batterijen. Een goede balans tussen beeldkwaliteit, performantie, batterijduur en het gebruik van resources is noodzakelijk en niet vanzelfsprekend.

Waarom zouden we eigenlijk zulke complexe beelden willen renderen op mobiele toestellen? Welke technieken kunnen hiervoor gebruikt worden? Welke architecturen bestaan er ondertussen en welke specificaties hebben ze?
In deze thesis zal een antwoord gegeven worden op deze vragen. Verder zal deze thesis handelen over het huidige gebruik van PDA's en GSM's, de verschillende API's, frameworks en development kits die er bestaan om rendering mogelijk te maken, de bestaande besturingssystemen voor deze mobiele toestellen en hun eigenschappen. Ook bevat deze thesis een implementatiegedeelte dat enerzijds enkele besproken SDK's en API's uitprobeert en anderzijds een stad lokaal op een PDA rendert.

Hoofdstuk 2

GSM's, PDA's en Smartphones

2.1 Introductie

Meer dan 1,5 miljard mensen bezitten een GSM en deze zijn ondertussen zo bekend en verspreid dat ze niet veel toelichting vergen. De laatste generatie GSM-modellen bezitten hoge resolutie kleurenschermen, multimediatoepassingen, internettoegang en 3D capaciteiten. Door deze ontwikkelingen kunnen deze GSM's – net als PDA's – gebruikt worden voor het renderen van complexe scènes en vallen ze dus binnen het bestek van deze thesis.

We gaan verder met een korte introductie van PDA's en Smartphones.
Personal Digital Assistants (PDA's of palmtops) zijn kleine mobiele toestellen, die oorspronkelijk waren ontworpen als "personal organizers" maar al snel veel veelzijdiger werden door de jaren.
Een standaard PDA bestaat meestal uit: een klok, een adressen- en databoek, een takenlijst, een memo pad en een rekenmachine. Een groot voordeel van het gebruik van PDA's is de mogelijkheid om data uit te wisselen en te synchroniseren met desktops, laptops, desknotes, sattelieten en andere digitale apparaten.

Smartphones zijn GSM's met (eventueel enkel de basis-) functionaliteiten van PDA's. Vermits er tussen de laatste generatie GSM's en Smartphones niet veel verschil meer is in de context van deze thesis, zullen deze beide termen door elkaar gebruikt worden. Wanneer we spreken over GSM's bedoelen we dus de laatste generatie GSM's: met kleurenscherm en 3D renderingscapaciteiten.

2.2 Motivatie

Waarom of wanneer zouden we beelden willen renderen op PDA's of GSM's? Communicatie in eender welke vorm kan meestal verbeterd worden door het gebruik van foto's en beelden [57] . Beelden kunnen bijvoorbeeld tekst complementeren, berichten verduidelijken of zelf het bericht zijn en hierdoor de boodschap beter overbrengen. Een mobiel toestel zoals een PDA of GSM is niets meer dan een communicatiemiddel, soms tussen de gebruiker en anderen enerzijds, soms tussen de gebruiker en het toestel anderzijds. Deze communicatie uit zich in de vorm van de functionaliteit van het toestel. Om een overzicht te geven wordt er een categorisatie gegeven van de mogelijke functionaliteiten die mobiele toestellen kunnen bieden, zowel op cognitief niveau [29] als op applicatieniveau [37].

Categorisatie van functionaliteiten

- Op cognitief niveau:

  - Korte- en lange-termijn geheugen ontlasten
    Het "lange-termijn geheugen" wordt geholpen door programma's die representaties van gedachten aanbieden die normaal in het intern menselijk geheugen worden opgeslagen of op een externe agent (bv papier). Enkele voorbeelden van zulke programma's zijn: verjaardagslijsten, adresboeken, databoeken,...
    Het "korte-termijn geheugen" wordt geholpen door programma's die representaties zijn van kleine gedachten die normaal tijdelijk onthouden moeten worden voor gebruik in de nabije toekomst. Enkele voorbeelden: Reminders, memopads, alarm clocks,...

  - Gebruik maken van grote hoeveelheden data
    Programma's kunnen gebruikers de mogelijkheid geven om lokaal toegang te krijgen tot grote hoeveelheden informatie, zowel taak-specifieke encyclopedieën en woordenboeken als niet taak-specifieke referenties zoals hypertekst-referenties, boeken, kranten en medische referenties.

  - Gebruik maken van interactieve referentie tools
    Programma's zoals calculators, unit convertors, astronomische referenties,...

- Op applicatieniveau:

  - Locatie gebaseerde diensten
    Door locatie gebaseerde informatie en applicaties hebben gebruikers

een keuze uit een brede waaier van diensten, zoals weg- en weersinformatie, interactieve kaarten, reservaties voor theaters, restauranten en de bioscoop, enz. Deze diensten zullen naar de toekomst toe enkel uitbreiden en verbeteren.

- Mapping en positionering
  GPS ontvangst op mobiele toestellen is zeer bruikbaar en wordt steeds meer en meer gebruikt.

- Versturen van foto- en multimediaberichten
  Berichten versturen is reeds een zeer populaire trend bij GSM gebruikers. Zo kunnen tekstberichten, foto's, ringtones, muziek, operator logo's, voice clips, video clips en geanimeerde, interactieve graphics verstuurd worden.

- Entertainment
  Interactieve applicaties, zoals games.

- Industriele applicaties
  Ingenieurs of professionals in het veld kunnen eender wanneer interactieve kaarten en voorstellingen bekijken. Interactieve constructieplannen kunnen bijvoorbeeld geraadpleegd worden om de vordering ter plaatse op te volgen en om snel problemen aan te kunnen kaarten.

- eCommerce
  Niet alleen kunnen bestellingen en aankopen "on the go" worden gedaan, ook de beurs en zijn transacties kunnen rechtstreeks en overal aangesproken worden.

Zoals reeds gezegd is, bestaat al deze functionaliteit uit communicatie tussen de gebruiker met het toestel of anderen en deze communicatie kan verbeterd worden door het gebruik van beelden. Deze beelden kunnen 2D of 3D beelden zijn, elk op hun beurt opgebouwd door een bepaalde renderingstechniek. Als we kijken naar de categorisatie hierboven wordt het al snel duidelijk dat bijna alle functionaliteiten hierboven opgesomd 2D of 3D rendering van beelden gebruiken of kunnen gebruiken om zo hun boodschap duidelijker, eenvoudiger of leuker te maken.

Een mooi voorbeeld waarbij beelden een bericht kunnen verduidelijken is een GPS systeem. Oudere GPS systemen zijn nog tekstgebaseerd, ze laten via tekst zien in welke straat je je bevindt en over hoeveel meter je in welke richting moet afdraaien. Een voorbeeld is zichtbaar in figuur 2.1. Nieuwere GPS systemen gebruiken een 2D gerenderde wegenkaart die een bovenaanzicht van

Figuur 2.1: Tekstgebaseerd GPS systeem

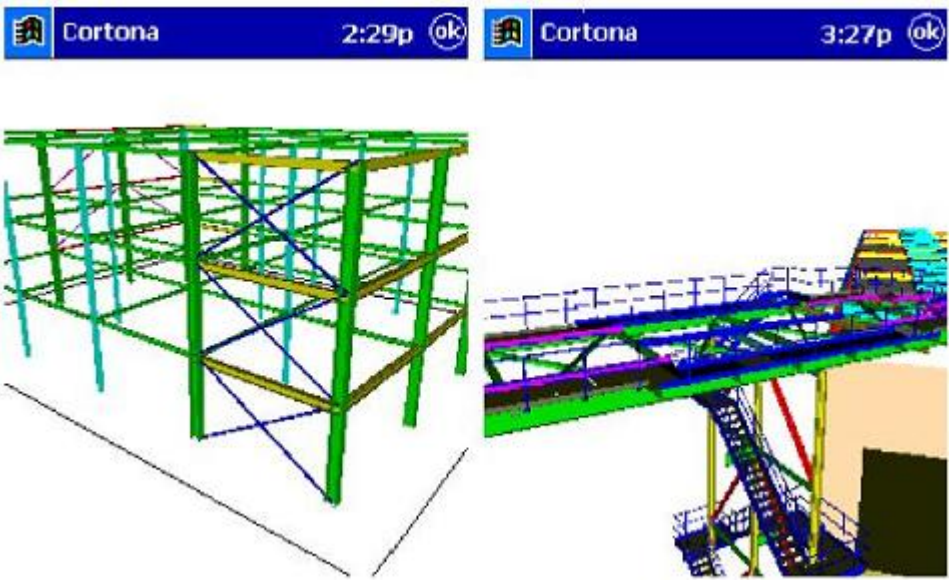
de omgeving weergeeft. Een voorbeeld hiervan is zichtbaar in figuur 2.2 [2]. Zulke voorstellingen zijn gemakkelijker en intuïtiever dan een tekstgebaseerde

Figuur 2.2: 2D GPS systeem [2].

voorstelling. Ook wordt de informatie minder dubbelzinnig waardoor minder vergissingen gebeuren en minder fouten gemaakt worden.
Een nog betere representatie van gegevens voor een GPS systeem is een 3D voorstelling van de omgeving, zie figuur 2.3 [35]. Deze is nog intuïtiever en door het gevoel van immersie kan de gebruiker zijn positie in de omgeving onmiddellijk herkennen en zich oriënteren.

Een ander voorbeeld is een 3D rendering van een gebouw tijdens de constructie. De ingenieur kan via de PDA gemakkelijk de vorderingen opvolgen en fouten vroegtijdig opsporen. Zie figuur 2.4 [54]. Doordat deze beelden interactief bekeken kunnen worden en gedetailleerde informatie over elk deel opgevraagd kan worden is deze methode krachtig en bruikbaar. Een ingenieur moet niet meer verschillende pagina's plannen de hoogte in sleuren om de vorderingen op te volgen. Ook kan nu alle data in 3D weergegeven worden in plaats van 2D plannen te gebruiken. Verdere voorbeelden kunnen

Figuur 2.3: 3D GPS [35].



Figuur 2.4: 3D weergave van de stellingen van een gebouw [54].

gevonden worden in hoofdstuk 5.

2.3 Ontwikkelingen en Technologieën

Nu besproken is waarom en wanneer rendering bruikbaar kan zijn voor mobiele toestellen, is het tijd om te gaan kijken naar de technologieën en ontwikkelingen die deze toestellen ons bieden. Eerst beschouwen we enkele populaire besturingssystemen. Vervolgens beschouwen we enkele "state of the art" processor architecturen. Tenslotte bespreken we kort het geheugen aanwezig in PDA's. Nadat we alle processor architecturen één voor één beschouwd hebben, maken we een vergelijking.

2.3.1 Besturingssystemen

We beginnen met een kleine voorstelling van de meest voorkomende besturingssystemen en hun fabrikanten [27]. Deze zijn:

- Palm OS van PalmSource, Inc.
- Windows Mobile (gebaseerd op de Windows CE kernel) van Microsoft
- Symbian OS (vroeger EPOC) van Ericsson, Panasonic, Nokia, Samsung, Siemens en Sony Ericsson
- BlackBerry van Research In Motion
- Verschillende besturingssystemen gebaseerd op de Linux kernel (Open Source)

Enkele screenshots van deze besturingssystemen kunnen gezien worden in figuur 2.5 [27]. Van deze besturingssystemen bezet Windows Mobile samen met Blackberry meer als 90% van de markt en de populariteit van Windows Mobile blijft maar stijgen.
Wat is nu precies het verschil tussen Windows CE, Windows Mobile en Pocket PC? Deze termen worden vaak door elkaar gebruikt terwijl er toch een verschil aanwezig is. Windows CE is een modulair besturingssysteem dat als basis gebruikt wordt voor verschillende klassen van mobiele toestellen. Het is een variatie van het Windows besturingssysteem met een volledig nieuwe kernel. Windows CE wordt ondersteund door Intel x86 (en varianten), MIPS, ARM en Hitachi SuperH processoren. Windows Mobile kan dan best beschreven worden als een verzameling van platformen gebaseerd op Windows CE.

Figuur 2.5: Enkele beelden van verschillende besturingssystemen: (a) Palm OS, (b) Windows Mobile, (c) Symbian OS, (d) Blackberry en (e) Een linux gebaseerd besturingssysteem [27].

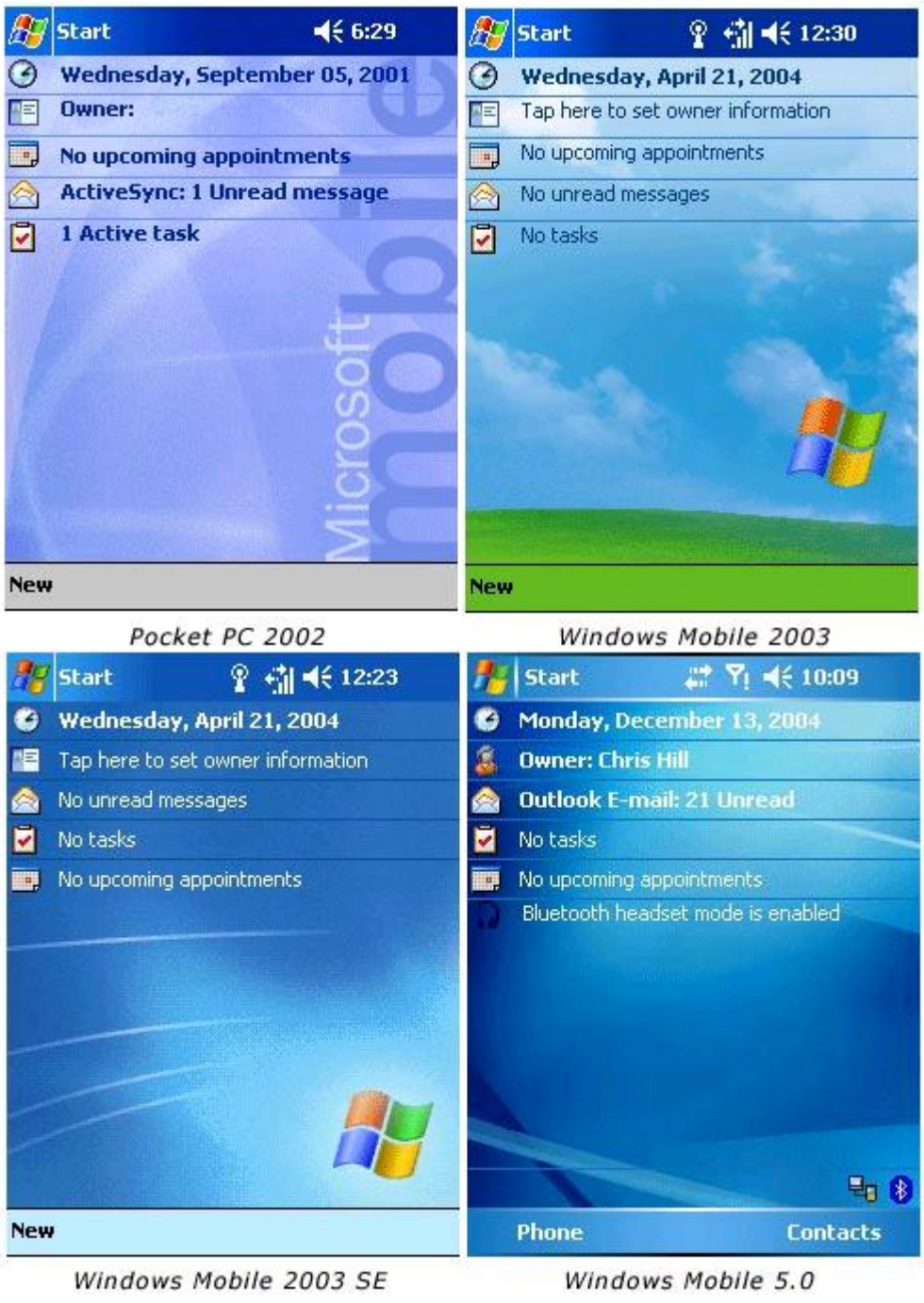
Momenteel vallen Pocket PC's (die vanaf 2003 Windows Mobile worden genoemd), Smartphones en Portable Media Centers binnen de benaming Windows Mobile. De volledige lijst van platformen bestaat uit: AutoPC, Handheld PC, Pocket PC, Pocket PC 2002, Pocket PC 2003, Pocket PC 2003 SE, Smartphone 2002, Smartphone 2003 en Windows Mobile 5.0. Elk van deze platformen gebruiken verschillende componenten van Windows CE afhankelijk van de applicaties en gebruikte toestellen. In figuur 2.6 zien we enkele screenshots van de evolutie van Pocket PC tot Windows Mobile 5.0 [30].
Windows CE is geoptimaliseerd voor toestellen met weinig opslagcapaciteit. De Windows CE kernel kan al werken met minder dan een megabyte geheugen. Verder is het een real-time besturingssyteem met een deterministische interrupt latency. Het ondersteunt 256 priority levels en voorziet priority inversion. In tegenstelling tot Unix gebaseerde besturingssystemen is de fundamentele uitvoeringscomponent de thread. Window CE 5.0 is de meest recente stabiele versie.

2.3.2 Processor architecturen

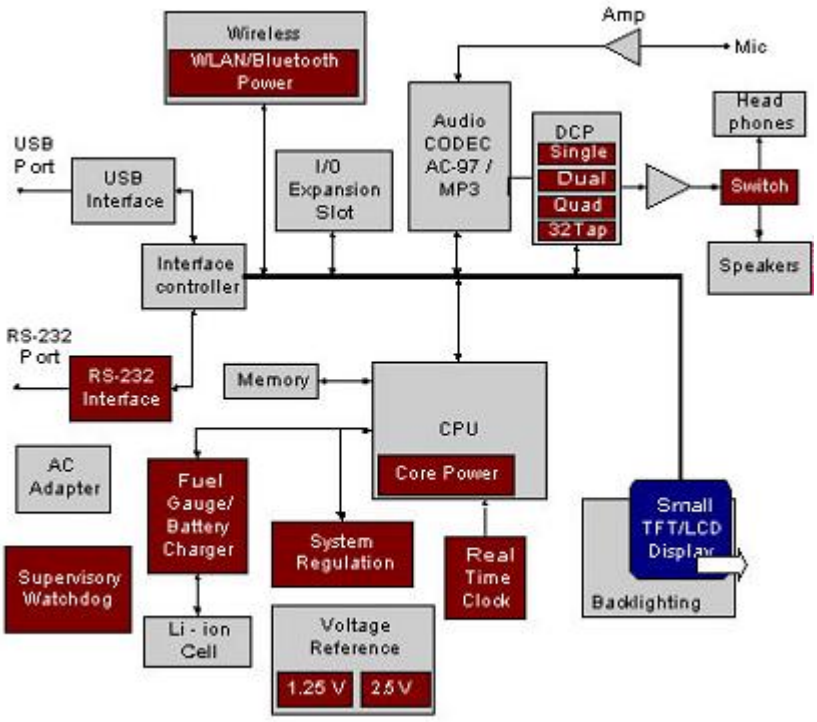
Vooraleer we verder gaan is het nuttig om de algemene opbouw van een PDA te beschouwen om een idee te krijgen van de aanwezige elementen. Dit kan best gezien worden in figuur 2.7 [28].

Voor rendering zijn de processor, het geheugen, de display, een eventuele grafische accelerator en de batterijduur de belangrijkste componenten. Deze zijn het belangrijkste omdat net al deze componenten beperkt zijn in een mobiel toestel. Het is daarom ook het doel van de processorontwikkelaars om zo goed mogelijke beelden te kunnen laten genereren terwijl zo efficiënt mogelijk omgegaan wordt met de beperkingen van deze verschillende hardware componenten. Ze doelen dus allemaal op hoge performantie en laag stroomverbruik.

Enorm veel mobiele toestellen werken met één of meerdere processoren die een variatie zijn op de ARM architectuur (Acorn RISC Machine)[4]. Dit is een 32-bit RISC (Reduced Instruction Set Computing) processor architectuur die in vele mobiele toestellen is verwerkt vanwege zijn lage productiekost en speciale stroombesparend ontwerp. Bedrijven zoals Freescale, IBM, Texas Instruments, Nintendo, Philips, VLSI, Atmel, Sharp, Samsung en STMicroelectronics hebben reeds licenties genomen op deze architectuur. De ARM chip is een van de meest gebruikte CPU's in de wereld, gevonden in harde schijven, telefoons, routers, rekenmachines, speelgoed,.... De ARM chip telt mee voor 75% van alle 32-bit mobiele toestellen. Enkele andere vaak gebruikte chips zijn de MIPS CPU en de SH3 CPU. De MIPS CPU is ook een



Figuur 2.6: De evolutie van Pocket PC 2002 tot Windows Mobile 5.0 [30].



Figuur 2.7: Een algemeen overzicht van de aanwezige hardware in PDA's [28].

RISC processor gebruikt in onder andere Nintendo 64, Playstation, Playstation 2, Playstation Pocket .

We gaan wat dieper in op een bekende implementatie van deze architectuur, de Intel XScale. Ook beschouwen we een 3D graphics processor ontwikkeld door ARM zelf, de MBX mobile core. Vervolgens worden nog enkele andere state of the art architecturen aangehaald, elk gebruikmakend van zijn eigen processor core.

Intel XScale

De meest bekende implementatie van de ARM processor is de opvolger van de StrongARM, de Intel XScale [17] [16]. De Intel XScale microarchitectuur is speciaal ontworpen voor laag verbruik en hoge performantie op een breed gamma van toestellen met dezelfde processorkern.

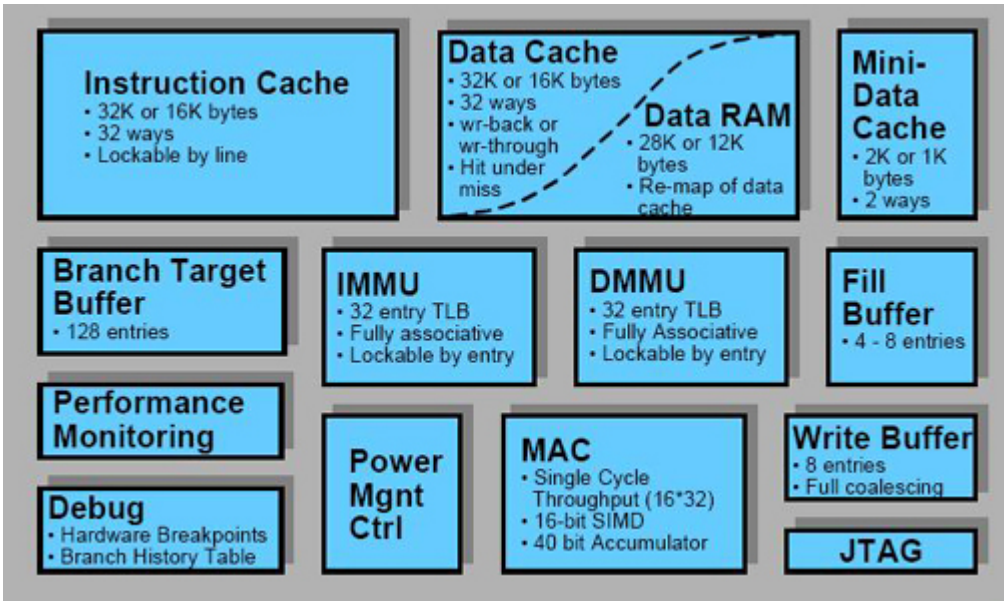
Intel combineert de lage energieconsumptie van de Intel XScale microarchitectuur met Intel Dynamic Voltage Management en Intel Media Processing Technology. Intel Dynamic Voltage Management scaleert de klokfrequentie en het voltage naargelang de behoeften van applicaties. Zo wordt energie efficiënt uitgespaard. De Intel Media Processing Technology is een co-processor engine die allerhande multimediatoepassingen mogelijk maakt.
De Intel XScale kern bevat ook uitbreidingen van de ARM architectuur, zoals de ARM Thumb instructies om de hoeveelheid code te verminderen en de ARM media extentions om Digital Signal Processor (DSP) functionaliteit toe te laten. De performantie van deze Intel XScale architectuur wordt verbeterd door de Intel Superpipelined technologie, dat resulteert in een hogere kloksnelheid (reikend tot 1 GHz!).

Deze microarchitectuur is in overeenstemming met de ARM Version 5TE ISA instructie set. Dit maakt deze architectuur compatibel met de nodige besturingssystemen (Microsoft Windows CE, VXWorks en IxWorks van WindRiver, EPOC van Symbian en Linux), applicaties en tools. De ARM kern wordt door deze microarchitectuur omvat door vele componenten:

- Instructie en data management units
- Instructie, data en mini-data caches
- Write, Fill, Pend en Branch target buffers
- Energiebeheer

- Performantieopvolging
- Debug en JTAG units
- Coprocessor interface
- MAC coprocessor
- Core memory bus

Figuur 2.8 illustreert deze microachitectuur [17].



Figuur 2.8: De Intel XScale architectuur [17].

Deze microprocessor kern wordt niet alleen geïntegreerd in PDA's en GSM's (Dell Axim X50v, Palm Zire, Sharp Zaurus) maar ook in allerlei verschillende producten zoals LCD controllers, IO-netwerkprocessoren, Portable Video Players (PVP's), Portable Media Centers (PMC's) (de Zen), enzovoort.

De hierboven besproken Intel XScale is een CPU en is niet speciaal ontwikkeld voor de rendering van beelden (dus is enkel software rendering mogelijk). De volgende architecturen die we bespreken zijn dit wel. We zullen deze kort bespreken en vervolgens onderling vergelijken.

ARM mobile cores

ARM biedt "Multimedia & Graphics Solutions" aan in samenwerking met Imagination Technologies [21]. Dit gebeurt in de vorm van 2D en 3D accelerator cores die 3D graphics en multimedia toepassingen toelaten op mobiele ARM-gebaseerde producten, zoals PDA's en GSM's. Deze zogenaamde MBX accelerators bieden hoge kwaliteit en hoge performantie graphicsacceleraties aan voor mobiele producten. De MBX Cores zijn gebaseerd op een PowerVR 3D architectuur [32] (van Imagination Technologies).

Er zijn 2 verschillende MBX accelerators:

- **MBX R-S**: QVGA-resoluties, vooral voor draadloze toestellen.
- **MBX HR-S**: SVGA(800x600), ideaal voor high-end, draadloze gebruikerstoestellen (vb. gaming)

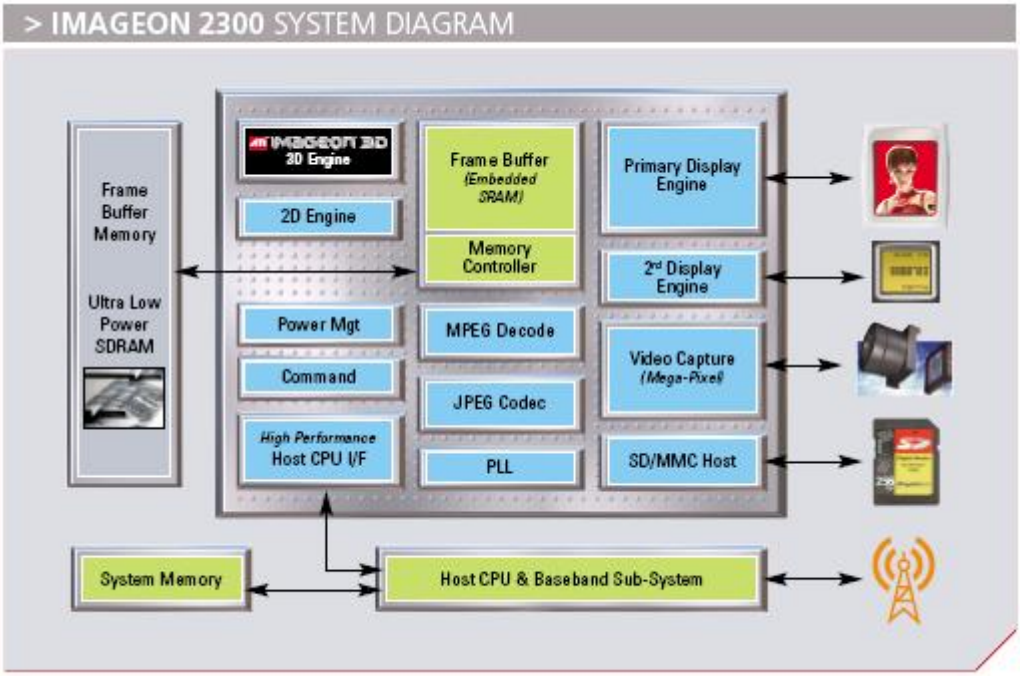
Deze cores maken gebruik van een unieke "tile" gebaseerde architectuur voor betere performantie en beeldkwaliteit tegen lagere stroomcomsumptie (in vergelijking met andere producenten).

Rendering gebeurt via tile gebaseerde rendering [51]. Deze techniek deelt de scène op in kleinere gebieden (tiles) en rendert deze tiles dan afzonderlijk. Een on-chip tile buffer houdt alle bandbreedte-intensieve pixelprocessing op de graphics chip zelf. Zo kan het verkeer voor Z-buffering tussen de processor en het geheugen worden weggelaten door ervoor te zorgen dat enkel de zichtbare pixels naar de framebuffer geschreven worden. Deze tile gebaseerde rendering biedt 24-bit precisie aan en deferred texturing. Deferred texturing [33] is een techniek waarbij in de eerste fase van de rendering pipeline hidden surface removal wordt uitgevoerd voor de texturing en shading gebeurt. Zo krijgen enkel de zichtbare pixels een texture toegewezen en worden redundante texturing operaties vermeden.

ATI IMAGEON

ATI heeft recent een geoptimaliseerde IMAGEON architectuur ontwikkeld [5]. Deze is de eerste hardware 3D engine voor mobiele toestellen. De IMAGEON integreert de IMAGEON 3D engine: Een geavanceerde, mobiele 3D graphics accelerator, een 2D graphics engine, een video decoder en een camera sub-systeem engine. Deze technologie, gebaseerd op hardware uit spelconsoles en PC's, wil performante, hoge kwaliteit 3D beelden aanbieden, video playback mogelijk maken en digitale camera functionaliteit toevoegen. Deze architectuur is zo ontwikkeld en geoptimaliseerd dat stroomverbruik minimaal is en performantie behouden blijft.

De IMAGEON architectuur is zichtbaar in figuur 2.9. De exacte specificaties komen verderop bij de vergelijking aan bod. Merk op dat deze microprocessor geen CPU is, maar een 3D multimedia engine die via een interface met een CPU communiceert. De IMAGEON 3D engine heeft bijvoorbeeld een interface naar de Intel XScale van hierboven.



Figuur 2.9: De IMAGEON architectuur [5].

NVIDIA GoForce 3D

Met de GoForce 3D core, voegt NVIDIA een multimedia- en 3D engine toe aan haar gamma [23]. De GeForce 3D is een nieuwe architectuur voor mobiele toestellen die, net zoals de IMAGEON van ATI, toelaat om de performantie te verhogen en het stroomverbruik te verminderen. De GoForce handelt alle intensieve multimedia taken af. Rr is geen processorinterventie of extern geheugen nodig. Door gebruik te maken van een unieke shader instuctie set wil NVIDIA twee fundamentele voordelen uitbuiten voor mobiele toestellen: lager stroomverbruik en geavanceerde beelden. In tegenstelling tot andere architecturen zoals de IMAGEON, is de GoForce 3D niet gebaseerd op hardware van PC's en game consoles. Het is een volledig nieuwe architectuur die tot 250 miljoen pixels per seconde kan genereren tegen 1/10de van het normale verbruik. Deze videokaart heeft in totaal 1280KB geheugen op een

128-bit geheugeninterface.
We zullen enkele belangrijke punten van deze architectuur aanhalen in functie van performantie, beeldkwaliteit en energiebeheer. Exacte specificaties kunnen onderaan bij de vergelijking gevonden worden.

- Performantie: Deze architectuur integreert een speciaal ontwikkelde geometrische processor om de complexe transformaties en initialisaties snel en energie-efficiënt af te handelen. Verder, vermits er in 3D applicaties enorm veel 2D operaties voorkomen, is het belangrijk om een goede 2D performantie te hebben. NVIDIA heeft daarom een krachtige 64 bit 2D graphics engine ontwikkeld.

- Beeldkwaliteit: NVIDIA heeft met de GoForce 3D de eerste core uitgebracht met een programmeerbare shader technologie. Door middel van shaders kunnen mooiere en realistischere beelden worden gerenderd. Verder is multitexturing mogelijk met tot 6 verschillende textures per pixel. Hiermee worden reflecterende oppervlakten, shaduw- en belichtingseffecten en meer mogelijk. Het bandbreedte-verbruik bij multitexturing is tot een minimum gehouden om zo energie uit te sparen. Een nadeel is wel dat deze processor geen anti-aliasing ondersteunt.

- Energiebeheer: De GoForce 3D maakt gebruik van de nPower technologie. Deze technologie bestaat uit een hele reeks optimalisaties gaande van on chip processing om de CPU te ontlasten tot meerdere hardware processors en dynamic clocking. Zo worden er pipelines en andere onderdelen van de chip uitgeschakeld om energie te besparen. Verder maakt de Goforce gebruik van "Early Z". Dit is een optimalisatie die kijkt welke delen van de scène niet zichtbaar zijn, en negeert deze vervolgens. Hierdoor wordt processing tijd uitgespaard en dus batterijduur.

Deze GoForce 3D architectuur wordt in NVIDIA's nieuwe mobiele toestellen verwerkt en biedt multimedia ervaringen, hogere performantie ten opzichte van oudere architecturen en een langere batterijduur. We spreken hier van hoge resolutie fotografie, full-motion video opname en weergave, interactieve 3D games, enz...

BitBoys G12, G40 en G34

De BitBoys G-reeks biedt drie verschillende architecturen aan [6]. De G12, een snelle vector graphics processor, de G34, een 2D/3D graphics processor en tenslotte de G40, een processor die 3D- en vector graphics mogelijkheden

combineert.
We zullen deze drie afzonderlijk bespreken.

De G12 is een 2D vector graphics processor die snel en energie-efficiënt vector graphics kan genereren op mobiele toestellen. Deze processor kan meer dan 60 beelden per seconde genereren en 100 maal sneller werken dan software rendering. De voordelen van vector graphics komen later aan bod in hoofdstuk 4 en sectie 9.2: bijlage B biedt ook meer informatie.

De belangrijkste voordelen van de G12 zijn: de core is zeer klein, waardoor zeer weinig stroom- en CPU vermogen verbruikt wordt en waardoor deze chip zelfs geschikt is voor low-end CPU's. Deze core ondersteunt de twee belangrijkste standaarden voor vector graphics: SVG Tiny 1.2 en OpenVG 1.0. Tenslotte hebben we een volledige ondersteuning van de vector graphics rendering pipeline in de hardware.

Door het gebruik van de vector graphics wordt de bestandsgrootte van beelden kleiner zonder kwaliteitsverlies (zie figuur 2.10). Ook kunnen beelden gescaleerd worden zonder kwaliteit te verliezen. Sectie 9.2: bijlage A biedt meer informatie in verband met vector graphics.

Figuur 2.10: Voorbeeld van vector graphics [57].

Vermits deze core geen 3D processor is kunnen enkel de 2D aspecten vergeleken worden met de vorige twee processoren van ATI en NVIDIA. Daarom geven we enkel een kleine samenvatting van enkele belangrijke technische specificaties zichtbaar in tabel 2.1.

De G12 core richt zich niet alleen op GSM's en PDA's maar ook op digitale horloges, mobiele game consoles, TV's, Set Top boxes, DVD spelers, autoradio's, digitale camera's, MP3 spelers, GPS toestellen, ...

Product	2D en vector graphics processor IP core
Compatibiliteit	2D bitmaps, OpenVG, SVG Tiny 1.2
Intern Geheugen	Niet noodzakelijk, enkel frame buffer geheugen wordt gebruikt
Silicon processen	Alle moderne logische processen zijn mogelijk
klokfrequentie	200 Mhz
Resoluties	Tot 2048x1024 met 32 bit kleuren

Tabel 2.1: Enkele technische specificaties van de BitBoys G12.

De G34 en G40 zijn 2D/3D graphics processoren. De exacte specificaties en verschillen tussen beide zullen we stap voor stap bekijken. We beginnen met de fysische en algemene eigenschappen. Beschouw tabel 2.2. Het grootste verschil tussen beide is dat de G40 ook een vector graphics processor is en de 3D processor OpenGL ES 2.0 ondersteunt, waar de G34 enkel OpenGL ES 1.1 ondersteunt. Verder heeft de G34 geen ondersteuning voor pixel shaders. Merk op hoe hoog de resoluties zijn die deze processoren aankunnen. Een gewone PDA heeft meestal maar een scherm voor een resolutie van 320x240 of 640x480 en GSM's nog kleiner. Deze chips kunnen dus gebruikt worden voor apparaten met grote schermen.
We gaan verder met de texture- en shadingseigenschappen van deze processoren. Beschouw tabel 2.3.

Zoals we kunnen zien in deze tabel ondersteunen deze processoren reeds vele van de eigenschappen die grafische kaarten voor desktops kunnen realiseren. Tussen beide processoren is er verder niet veel verschil meer, behalve dat de G40 wat preciezer werkt en meer textures per pixel aankan. De G40 is dan ook een nieuwere processor.

Falanx Mali 200, 110, 55, GP

Net zoals de vorige drie producenten, ontwikkelt Falanx 3D graphics accelerator IP cores met hetzelfde doel voor ogen: beelden van hoge kwaliteit en een laag stroomverbruik mogelijk maken[9]. Falanx wil met de unieke en scaleerbare Mali processoren een breed gamma van applicaties aanspreken. Ze willen hun processoren, net zoals ARM, aan System on Chip (SoC) handelaars verkopen voor mobiele toestellen zoals PDA's, Tablet PC's, game consoles en in-car info en navigatie.

	G34	**G40**
Fysische eigenschappen		
Maximale klokfrequentie	200 Mhz	200 Mhz
Interfaces	AMBA, OCP, klantspecifiek	AMBA, OCP, klantspecifiek
Frame buffer grootte	tot 2048x1024	tot 2048x1024
Frame buffer modes	16/32-bit	16/32/64-bit floating-point
Z-buffer modes	16/32-bit	16/32-bit
Stencil buffer modes	1/4/8-bit	1/4/8-bit
Algemene eigenschappen		
Architectuur	Programmeerbare geometrische processor met een fixed function pixel pipeline	Programmeerbare geometrische- en pixelprocessor
2D graphics acceleratie	Ja	Ja
3D graphics acceleratie	Ja	Ja
Vector graphics acceleratie	Nee	OpenVG, SVG
API Support	OpenGL ES 1.1, Direct3D Mobile, M3G	OpenGL ES 2.0, Direct3D Mobile, M3G
Transformation en lighting	Hardware	Hardware
OpenGL ES pixel pipeline	Ja	Ja
Vertex shaders	Ja	Ja
Pixel shaders	Nee	Ja
Anti aliasing	FLIPQUAD	FLIPQUAD
Early Z-check	Ja	Ja
Point rendering	Ja	Ja
Line rendering	Ja	Ja
Triangle and quad rendering	Ja	Ja

Tabel 2.2: Algemene en fysische eigenschappen van de BitBoys G34 en G40

	G34	**G40**
Texture- en Shadingseigenschappen		
Aantal textures/pixel	2	4
Maximale texture grootte	1024x1024	1024x1024
Perspectief correcte texturing	Ja	Ja
Texture LoD	Per-Pixel	Per-pixel
Geprojecteerde textures	Ja	Ja
Texture formaten	4/5/16/32-bit	4/5/16/32 en 64-bit floating point
Texture compressie formaat	DXT1, PACKMAN	DXT1, PACKMAN
YUV texture formaat	UYVY, YUY2, YVYU	UYVY, YUY2, YVYU
Geprojecteerde textures	Ja	Ja
MIP-mapping	Ja	Ja
Bilinear filter	Ja	Ja
Trilinear filter	Ja	Ja
Shading eigenschappen		
Perspectief correcte shading	Ja	Ja
Ambient en Gouraud shading	Ja	Ja
Specular shading	Ja	Ja
DOT3 blend mode	Ja	Ja
Alpha blending	Ja	Ja
Dithering	Ja	Ja
Fog	Ja	Ja
Internal color precision	10-10-10-10	16-16-16-16 floating point

Tabel 2.3: Texture- en shadingseigenschappen van de BitBoys G34 en G40

Er zijn momenteel vier versies van de Mali graphics IP cores die kunnen geïntegreerd worden in apparaten, afhankelijk van de nodige performantie en/of energieverbruik. Deze vier zijn:

- **Mali GP**: GP staat voor geometrische processor. Deze core wordt in combinatie gebruikt met de 2D/3D processing cores van Falanx. De Mali GP kan gezien worden als een hoge performantie floating point DSP, een programmeerbare vertex shader of een fixed function vertex pipeline. Deze chip kan ongeveer vijf miljoen driehoeken per seconde aan en is compatibel met OpenGL ES v1.1 en Direct3D. Let wel, het maximaal aantal driehoeken per seconde wordt gemeten voor enkel transformaties, en is nog afhankelijk van de belichting in de scène. Hoe meer lichtbronnen de omgeving bevat, hoe lager de performantie. Dit geldt voor alle architecturen. De maximale klokfrequentie is 150 Mhz.

- **Mali 55 en 110**: Dit zijn twee 2D/3D graphics cores. De Mali 55 is ontwikkeld om de ARM 11 te evenaren op het gebied van 3D graphics. De Mali 110 is ontwikkeld in combinatie met de Mali GP voor hogere performantie en kwaliteit bij het renderen van beelden (door oa. de CPU te ontlasten). De Mali 55 is een lichtere versie van de Mali 110, en is speciaal ontwikkeld voor toestellen met een lagere performantie of kost. Beide cores ondersteunen 4x en 16x Full Scene Anti-Aliasing (FSAA) en werken op 200 Mhz. Door zowel tile-based als immediate mode rendering te gebruiken, kan de performantie verhoogd worden (zie ook lager: Falanx Mali 200).
  Enkele verschillen kunnen in tabel 2.4 beschouwd worden.

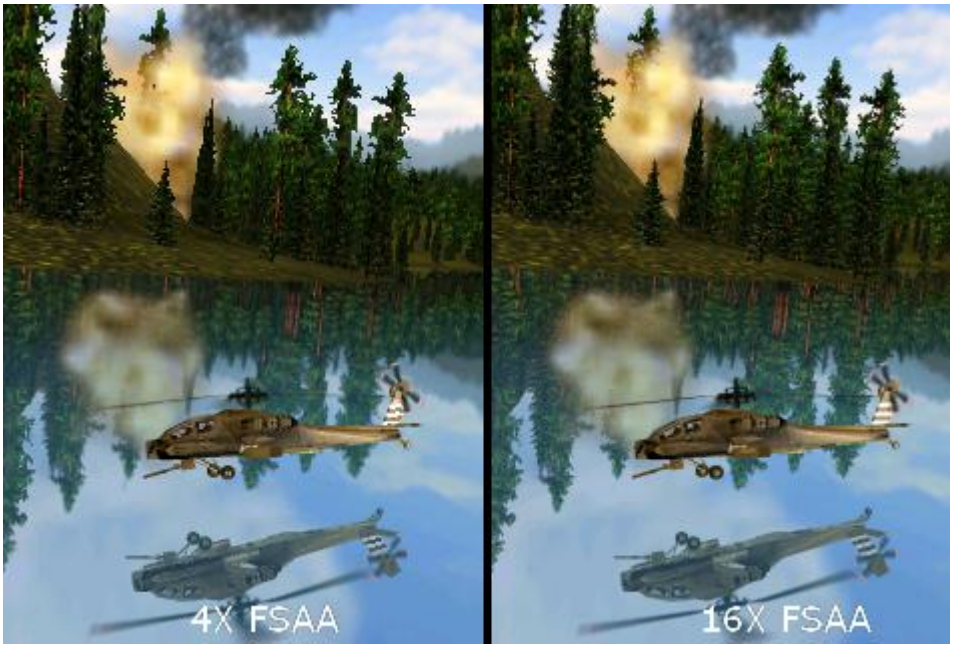
	Mali 55	**Mali 110**
Driehoeken per seconde	1 miljoen	5 miljoen
Pixels per sec met 4x FSAA	100 miljoen	300 miljoen
Pixels per sec met 16x FSAA	25 miljoen	75 miljoen
Verdere eigenschappen	Geoptimaliseerd voor "gate count"	"Early Z occlusion"

Tabel 2.4: Enkele verschillen tussen de Mali 55 en Mali 110

  De "gate count" optimalisatie geeft ongeveer één derde van de performantie van de Mali 110, om rekening te houden met de low-end apparaten. De Mali 110 gebruikt verder "Early Z occlusion" om een 70% snelheidswinst te krijgen.

- **Mali 200**: Net zoals de Mali 110 integreert de Mali 200 de Mali GP core voor complexe transformaties, initialisatie berekeningen, video acceler-

atie, enzovoorts,... om de CPU te ontlasten. Dit paar van rendering core en geometrische processor laat complexe en gedetailleerde scènes toe (figuur 2.11).



Figuur 2.11: Beelden gerendered met de Falanx Mali 200 [9].

Verder gebruikt de Mali 200 een combinatie van tile-based rendering en immediate mode rendering, net zoals de Mali 110. Dit geeft Falanx een competitief voordeel ten opzichte van andere producenten die slechts één techniek toepassen. Bijvoorbeeld: 80% van de per-pixel bandbreedte die nodig is voor immedate mode kan weggelaten worden door de tile-based methode.
Door gebruik te maken van hun propriëtaire FLXTC texture compressiemethode wordt nog minder bandbreedte en stroom verbruikt. Enkele van de belangrijkste technische eigenschappen van de Mali 200 zijn:

- Maximale klokfrequentie van 200 Mhz
- Maximaal 10 miljoen driehoeken per seconde
- 4x en 16x FSAA
- Programmeerbare vertex en pixel shaders

Verdere specificaties komen bij de vergelijking op het einde van deze paragraaf aan bod.

Ook [13] en [32] bieden graphics processoren aan. Vermits deze vergelijkbaar zijn met de andere cores en niets nieuws toevoegen, worden deze niet verder besproken.

De vergelijking

We hebben nu enkele bekende producenten en hun architecturen besproken. We gaan deze nu onderling kort vergelijken. We vergelijken de laatste nieuwe architectuur van elke besproken producent: De IMAGEON 2300 van ATI, de Goforce 3D van NVIDIA, de G40 van Bitboys en de Mali 200 van Falanx. We beginnen met enkele belangrijke specificaties van de 3D engine. Beschouw tabel 2.5.

3D Engine	**IMAGEON2300**	**GoForce3D**	**G40**	**Mali200**
Geometrische processor	Ja	Ja	Ja	Ja
Pixel rendering pipeline	Ja	Ja	Ja	Ja
Vertex shader	Nee	Nee	Ja	Ja
Pixel shader	Nee	Ja	Ja	Ja
Multi texturing	Ja	Ja	Ja	Ja
Mipmapping	Ja	Ja	Ja	Ja
Perspective correction	Ja	Ja	Ja	Ja
FSAA	Nee	Nee	Nee	Ja
Dithering	Ja	Ja	Ja	Nee
Alpha blending	Ja	Ja	Ja	Ja

Tabel 2.5: Een vergelijking van de verschillende architecturen: De IMAGEON2300, de GoForce3D, de Falanx Mali G40 en de Falanx Mali 200

Wat het meeste opvalt bij deze vergelijking is het ontbreken van zowel vertex als pixel shaders bij de IMAGEON 2300 en de unieke aanwezigheid van FSAA bij de Mali 200. Verdere verschillen en opvallende kenmerken zijn:

- De Goforce 3D gebruikt krachtige 64-bit 2D acceleratie om zowel de 2D als de 3D performantie te verhogen (doordat 3D operaties 2D operaties gebruiken).
- De mali 200 en de G40 hebben beide een vector graphics processor.

- De IMAGEON 2300 ondersteunt OpenGL ES 1.0, de Goforce 3D ondersteunt OpenGL ES 1.0, OpenGL ES 1.1, Direct 3D Mobile en M3G. De G40 ondersteunt OpenGL ES 2.0, Direct 3D Mobile en M3G en de Mali 200 tenslotte ondersteunt OpenGL ES 1.1, OpenGL ES 2.0 en Direct 3D Mobile.

Verder zien we bijna uitsluitend gelijkenissen, ook als er wat gedetailleerder naar deze architecturen gekeken wordt. Dit is niet verwonderlijk vermits het al jarenlang een nek aan nek race is voor de beste grafische kaart (voor desktops) tussen NVIDIA en ATI. Kijkend naar de verschillende architecturen en hun vergelijking concluderen we dat er niet zoveel verschil is tussen deze eerste generatie hardware processoren voor mobiele toestellen. De nieuwere architecturen (Mali 200) ondersteunen enkel wat meer functionaliteit.

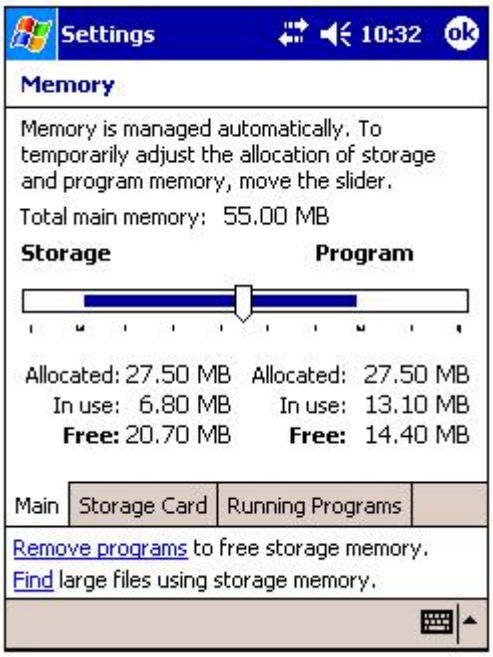
2.3.3 Geheugen

Het geheugen van PDA's en andere mobiele toestellen wordt anders gebruikt dan bij desktop computers [34]. PDA's bezitten, net als desktops, twee soorten intern geheugen: RAM en ROM geheugen.
ROM geheugen kan enkel gelezen worden en wordt gebruikt om het besturingssysteem op te slaan (eventueel nog andere programma's).
RAM geheugen wordt op 2 manieren gebruikt (in desktops maar op 1 manier):

- als opslaggeheugen:
  Wanneer een document of programma wordt opgeslagen, wordt dit in de RAM opgeslagen. Hier kan het programma aangesproken en gebruikt worden tot het gewist wordt. RAM is "volatile memory", dit wil zeggen dat het continu stroom nodig heeft om gegevens op te kunnen slaan. Dit heeft als gevolg dat wanneer de batterij het begeeft, al de opgeslagen data verloren gaat. Om gegevens permanent weg te schrijven kunnen bijvoorbeeld Flash geheugen kaartjes gebruikt worden.

- als programmageheugen:
  Ook wordt de RAM – net zoals bij desktops – gebruikt om programma's uit te voeren en documenten te openen.

De hoeveelheid opslaggeheugen ten opzichte van programmageheugen kan vaak door de gebruiker zelf ingesteld worden (zie figuur 2.12), maar altijd moet een gedeelte voor elk gereserveerd blijven. Hierdoor kan het zijn dat wanneer een toestel 64MB RAM heeft, eigenlijk maar 27MB voor de gebruiker overblijft omdat de rest door het besturingssysteem en werkende programma's ingenomen wordt.



Figuur 2.12: Screenshot van Windows Mobile. Hier kan gebruiker de hoeveelheid programma- en opslaggeheugen bepalen.

Waarom is rekening houden met geheugen eigenlijk zo belangrijk voor rendering?
Zoals vermeld in hoofdstuk 1 hebben programma's in een mobiel toestel niet veel geheugen ter beschikking. Daar komt nog bij dat toegang krijgen tot het geheugen een van de meest energieverbruikende operaties is. Men moet dus voorzichtig omspringen met het geheugen dat men gebruikt om zo efficiënt mogelijk de bandbreedte tussen het geheugen en de processor te benutten. Een goed voorbeeld is texture mapping. Hoe meer textures we gebruiken of hoe groter ze zijn, hoe zwaarder de geheugenoperaties worden. Goede texture compressie en efficiënt omgaan met het inladen en uitladen van textures is dus de boodschap.

Hoofdstuk 3

Rendering voor PDA of GSM

3.1 Inleiding

Zoals reeds vermeld zijn mobiele toestellen zoals een PDA of GSM vandaag de dag nog steeds zeer beperkt in snelheid en opslagcapaciteit. Toch zijn er opkomende streefdoelen. Men wil al de voordelen van een vast toestel zoals bijvoorbeeld een gewone desktop PC, kunnen combineren met de mobiliteit van PDA's en GSM's. Zo wil men bijvoorbeeld dat het mogelijk is om op mobiele toestellen:

- grote hoeveelheden data efficiënt en snel weer te geven (in 3D)
- te kunnen navigeren in een 3D omgeving
- 3D games te spelen
- augmented reality te realiseren
- ...

Deze doelen zijn onstaan uit een bepaalde nood. Men streeft naar een intuïtievere en meer bruikbare voorstelling van gegevens, eender waar men zich bevindt (zie hoofdstuk 1). Het is aangetoond in [52] dat het veel eenvoudiger is om door een stad te wandelen als we een volledige, realistische 3D weergave van de stad hebben om ons te leiden, in plaats van een kaart of een simpel 2D bovenaanzicht. Waarom nog thuis achter een TV scherm 3D games spelen als je hetzelfde spel eender waar (in de trein, in de tuin, op vakantie, ...) draadloos kan spelen tegen je vrienden? Er is al jaren vraag naar zulke technologieën en mogelijkheden, en we komen steeds dichter en dichter.

Zeer belangrijk bij deze doelen is:

- streven naar realisme
- gevoel van immersie opwekken
- een specifieke snelheid garanderen voor de weergave van initiële beelden
- de vlotte weergave van de beelden, voldoende beelden per seconden
- de afwezigheid van artefacten in de wereld

Vooraleer deze doelstellingen kunnen gerealiseerd worden, kijken we eerst op welke manier beelden op een PDA of GSM gerenderd kunnen worden.

3.2 Wat is rendering

Om even op te frissen geven we een definitie van rendering: rendering is een methode om de pixels (picture elements) van een digitaal beeld te genereren aan de hand van een high-level beschrijving van objecten. Het genereren van pixels bestaat uit het toekennen van een kleur aan een pixel.
Een beeld bestaat uit een grid van pixels, dus een grid van kleuren. Het doel van rendering in computer graphics is om op die manier complexe en realistische digitale beelden te genereren.

In de volgende secties van dit hoofdstuk (secties 3.3 en 3.4) zullen we toelichten welke technieken het meest gebruikt worden voor rendering en waar deze rendering precies plaats kan vinden. Het lijkt op het eerste zicht misschien raar dat er verschillende locaties voor rendering zijn maar dit zal snel duidelijk worden in sectie 3.3. Sectie 3.4 handelt over de renderingstechnieken zelf.

3.3 Locaties voor rendering

Deze sectie zal handelen over de mogelijke locaties waar rendering voor mobiele toestellen kan gebeuren. Zoals we reeds vermeld hebben in hoofdstuk 1 en 2 is de rekenkracht en capaciteit van zulke toestellen beperkt. Het is daarom vaak niet mogelijk voor het mobiele toestel om zelf alle berekeningen te maken van de rendering. Een of meerdere servers kunnen hierbij helpen. Met “server” bedoelen we een krachtig systeem zoals een desktop PC of high end computer. Met “client” (of ook wel “thin client” genoemd) bedoelen we het mobiele toestel zelf.

Algemeen gezien zijn er drie locaties waar (3D) beelden gerenderd kunnen worden:

- Server-gebaseerde rendering
- Client-gebaseerde rendering
- Hybride rendering (zowel client als server)

Deze lichten we nu toe.

3.3.1 Client-gebaseerde rendering

Bij deze techniek bevat de client lokaal de te renderen data van de 3D scène (zoals bijvoorbeeld [41] en [54]). Deze data kan zowel reeds lokaal opgeslagen zijn of dynamisch geladen worden via een draadoos netwerk. De beelden worden door de client zelf gerenderd en rechtstreeks weergeven. De efficiëntie is volledig afhankelijk van de rekenkracht van de client. Indien de data nog gedownload moet worden, is de efficiëntie ook nog afhankelijk van het netwerk. Vermits mobiele toestellen beperkt zijn in rekenkracht, opslagruimte en beeldgrootte, is het enkel mogelijk kleine hoeveelheden objectdata te verwerken. Realisme en bevredigende framerates zijn de grootse problemen bij deze techniek.
Andere technieken zoals server-gebaseerde rendering (zie sectie 3.3.2) kunnen deze problemen oplossen, maar brengen weer nieuwe problemen met zich mee.
De implementatie bij deze thesis gebruikt client-gebaseerde rendering en leent zich tot het incrementeel en dynamisch downloaden van data.
Voorbeelden van client-gebaseerde rendering zijn draagbare spelcomputers zoals de Nokia NGage [22] en de Sony PSP [36].

3.3.2 Server-gebaseerde rendering

Bij deze methode wordt er gebruik gemaakt van één of meerdere servers (zoals bijvoorbeld [46] en [49]). De server beschikt over alle objectdata, in tegenstelling tot de client die niets van de scène afweet. Alle beelden worden op de server zelf gerenderd en naar de client gestuurd. Deze moet dan enkel de beelden op het scherm weergeven.

Gebruik maken van deze techniek brengt veel voordelen met zich mee:

- De zwakke computatiekracht van de client wordt volledig omzeild.
- Doordat de server alle zware berekeningen doet is real-time interactie op de client mogelijk. Clients kunnen dan de beelden manipuleren en de server moet dan gepast de nieuwe beelden renderen en doorsturen.
- Een ander voordeel van het feit dat enkel de server het zware werk levert is dat realisme mogelijk wordt. Hoe sterker de server(s), hoe realistischer en mooier de beelden gemaakt kunnen worden. Dit kan allemaal onafhankelijk gebeuren van de client, die deze beelden slechts moet weergeven.
- Vermits enkel beelden verstuurd worden is deze techniek platform onafhankelijk.

De performantie is dan enkel afhankelijk van de computatiekracht van de server(s) en de beschikbare bandbreedte van het gebruikte netwerk.

De bandbreedte kan een probleem vormen bij een server-gebaseerde methode. Huidige mobiele toestellen kunnen al gemakkelijk 16-bit kleuren (65536 verschillende kleuren) weergeven en QVGA resoluties ondersteunen. Ook zullen ze al snel veel hogere resoluties en meer kleuren aankunnen. Als het volledige weer te geven beeld naar de PDA doorgestuurd moet worden en vloeiende beelden al gemakkelijk 25 afbeeldingen per seconde vereisen, hebben we enorm veel bandbreedte nodig. Dit kan gezien worden in figuur 3.1 [19]. Hoe kan dit bandbreedte-probleem nu aangepakt worden? Twee strategieën zijn mogelijk: compressie van de beelden of gebruik maken van een protocol gebaseerd op grafische primitieven [19]. Deze worden nu toegelicht.

Beeldcompressie

Bij deze strategie wordt gebruik gemaakt van de compressie van beelden. Het volledige te tonen beeld wordt door een server naar een mobiel toestel gestuurd. Om deze strategie efficiënt toe te passen moet de server enorm snel en goed kunnen comprimeren en de client snel kunnen decomprimeren en weergeven. Hoewel deze techniek effectief is en de decompressie snel genoeg gebeurt, blijkt dat de overhead van de input en output meestal nog te traag is. Dit houdt in dat het te lang duurt om het beeld te maken (of te tonen), naar de compressor (of decompressor) te sturen en op de transport software te duwen (of eraf te halen). Hierdoor kunnen in vele gevallen geen echt goede resultaten bekomen worden. Door nieuwe en betere netwerktechnologieën

Display format (16 bits/pixel)	Frame update rate (fps)	Raw data (kbits/s)	JPEG compressed data (kbits/s)
QCIF: 176 x 144	5	2,028	168
	10	4,055	338
CIF: 352 x 240	5	6,758	563
	10	13,517	1,126
QVGA: 320 x 240	5	6,144	512
	10	12,288	1,024
VGA: 640 x 480	5	24,576	2,048
	10	49,152	4,096
1,024 x 768	5	62,915	5,242
	10	125,829	10,486
1,280 x 1,024	5	104,858	8,738
	10	209,715	17,476

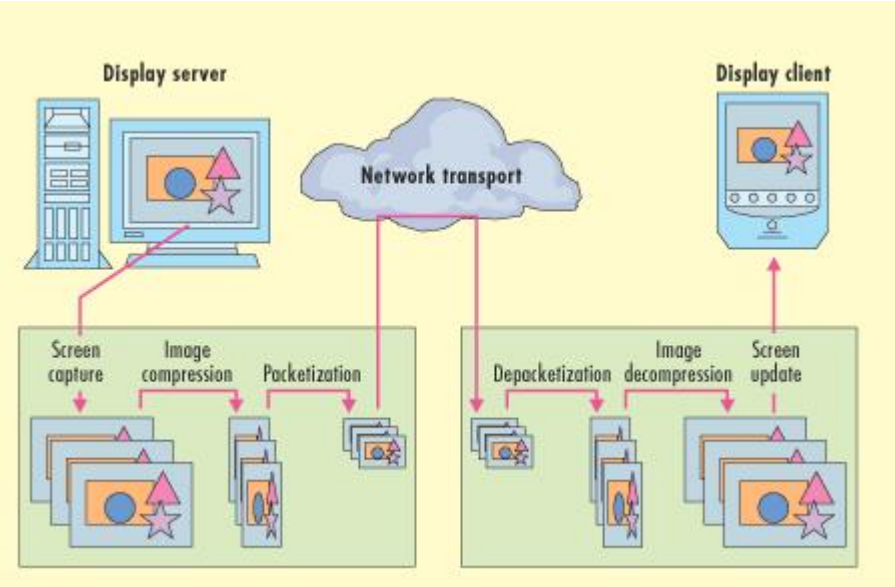
Figuur 3.1: Beelden kunnen al snel veel bandbreedte vereisen [19].

zullen deze problemen waarschijnlijk verdwijnen. Een grafische weergave van deze techniek is te zien in figuur 3.2 [19].

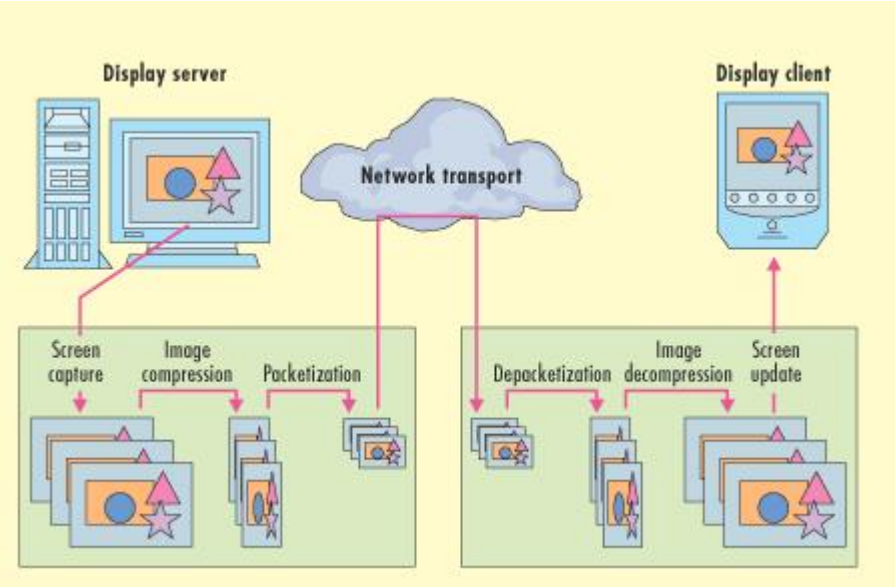
Protocollen gebaseerd op primitieven

Om het probleem van deze overhead te voorkomen kan een andere techniek toegepast worden, namelijk één die primitieven gebruikt. Deze techniek maakt gebruik van de grafische capaciteiten van de thin client. De server transformeert het weer te geven beeld in een reeks primitieve grafische requests die als pakketten over het netwerk gezonden worden. Zo kan een user interface bijvoorbeeld opgebouwd worden door requests voor rechthoeken, driehoeken, fonts, etc.... Merk op dat een 3D scène op dezelfde manier kan worden uitgetekend. De thin client ontvangt deze pakketten, interpreteert de requests, voert ze uit – gebruikmakend van de grafische mogelijkheden van het desbetreffende toestel – en stuurt een bevestiging naar de server. Deze techniek is grafisch zichtbaar in figuur 3.3 [19].

Doordat de getransporteerde data enkel bestaat uit (eventueel gecodeerde) commando's en argumenten voor de primitieve requests, neemt deze techniek veel minder bandbreedte in beslag dan de beeld-gebaseerde aanpak. Merk op dat dit niet altijd het geval is. Als de te renderen scène uit enorm veel objecten bestaat, dan zou het wel eens kunnen dat een afbeelding verzenden efficiënter is dan de beschrijving van elk zichtbaar object te verzenden.



Figuur 3.2: Techniek gebruikmakend van beeldcompressie [19].



Figuur 3.3: Techniek gebruikmakend van primitieven [19].

Architecturaal zijn deze twee technieken compleet verschillend, toch moeten beide kiezen uit enkele gemeenschappelijke algoritmische eigenschappen:

- **Server-push vs. client-pull**: Ofwel stuurt de server (server-push) continu scherminformatie naar de client, ofwel vraagt de client continu scherminformatie aan de server (client-pull)
- **Lazy vs. eager**: De scherminformatie wordt ofwel geupdate aan de framerate van de client (lazy), ofwel wordt de informatie geupdate aan de snelheid van de server zijn framerate (eager).
- **Compressietechnieken**: Beeld-gebaseerde aanpakken kunnen kiezen tussen verschillende compressietechnieken afhankelijk van de beschikbare bandbreedte. De aanpak met primitieven kan gebruik maken van compressie op bitmap data.

Een voorbeeld van het gebruik van server-gebaseerde rendering is een walkthrough door een gebouw renderen. De client kan dan in real-time door het gebouw lopen op het mobiele toestel en zo zijn weg vinden. De server bevindt zich dan ergens in het gebouw en stuurt continu informatie naar de client, afhankelijk van deze zijn handelingen. Indien door beperkte bandbreedte de weg niet in real-time kan weergegeven worden, dan kan bijvoorbeeld een manipuleerbare video verstuurd worden die een walkthrough biedt van een beginpunt tot een eindpunt in het gebouw. Hoofdstuk 5 biedt meer informatie over toepassingen van rendering voor mobiele toestellen.

3.3.3 Hybride rendering

Hierbij wordt een combinatie gebruikt van server-gebaseerde en client-gebaseerde rendering (bijvoorbeeld [42]). Een server kan bijvoorbeeld al het zware werk verrichten en de client zelf kan de gemakkelijke taken uitvoeren. Een concreet voorbeeld van het gebruik van server-client rendering kan gevonden worden in sectie 3.4.2.

Nu we de locaties waar rendering plaats kan vinden hebben besproken, gaan we verder met de meest gebruikte technieken voor rendering.

3.4 Renderingstechnieken

Voor 3D rendering op mobiele toestellen worden er drie renderingstechnieken op grote schaal toegepast, namelijk model-gebaseerde rendering, beeld-gebaseerde

rendering en hybride rendering dat gebruik maakt van een combinatie van de eerste twee.

3.4.1 Model-gebaseerde rendering

Dit is de meest voorkomende techniek. Model-gebaseerde rendering bestaat op zich nog uit 2 verschillende technieken, namelijk "pixel by pixel rendering" en "primitive by primitive rendering".

- **Pixel by pixel rendering**: De pixels genereren van een digitaal beeld komt neer op een kleur bepalen voor elke pixel. Deze techniek neemt dit letterlijk en gaat voor elke pixel van het beeld kijken met welk deel van de high-level beschrijving van de objecten deze overeenkomt en vervolgens wordt de pixel gepast ingekleurd. Deze techniek wordt ook wel (basic) raytracing genoemd.

- **Primitive by primitive rendering**: Meestal is pixel per pixel rendering te traag of niet praktisch genoeg. Primitive by primitive rendering kan dit probleem oplossen. De high-level beschrijvingen van de te renderen objecten bevatten informatie uit een volledig ander domein dan pixels. Deze objecten bestaan uit primitieven (lijnen, driehoeken, polygonen, ...) en worden voorgesteld door geometische informatie. Bij deze techniek wordt gekeken naar alle (relevante) primitieven en vervolgens wordt gekeken welke pixels deze beïnvloeden. Aan de hand van al de primitieven kan dan de kleur van elke pixel bepaald worden. Deze techniek wordt ook rasterisatie genoemd en is de renderingstechniek van alle huidige grafische kaarten.

3.4.2 Beeld-gebaseerde rendering

Zoals reeds vermeld hebben mobiele toestellen relatief zwakke processoren, weinig geheugen en een kleine display. Een kleine display speelt echter in het voordeel van beeld-gebaseerde rendering doordat de renderingstijd bij deze techniek voornamelijk bepaald wordt door de grootte van het scherm (dit in tegenstelling tot de normale 3D-graphics pipeline die kijkt naar het aantal polygonen in de scène).

Mobiele toestellen kunnen reeds gemakkelijk audio en video afspelen, ze hebben kleurrijke GUI's, ze kunnen draadloos communiceren,... 3D rendering op deze toestellen daarentegen is nog steeds een zeer moeilijke opdracht omwille van de grote computatietijd. De performantie van deze methoden is dus nog beperkt. Een andere aanpak dan model-gebaseerde rendering is

gebruik maken van beeld-gebaseerde rendering. Hierbij wordt in plaats van de scène te renderen aan de hand van primitieven een foto getoond van al de objecten in de scène. De renderingstijd is hier enkel afhankelijk van de grootte van het scherm en onafhankelijk van het aantal primitieven in de scène. We krijgen dus een maximaal voordeel bij zeer complexe scènes.
Een techniek om beeld-gebaseerde rendering te realiseren is een warping algoritme (McMillan [55]). Zo een algoritme neemt als input een "depth image" (een dieptebeeld bestaande uit 2D beelden met voor elke pixel een kleur- en diepte-informatie). Deze depth image moet wel eerst gegenereerd worden en is computationeel nog te zwaar voor een thin client. De client moet deze dieptebeelden reeds opgeslagen hebben om volledig zelfstandig te kunnen renderen. Indien de client de nodige dieptebeelden niet allemaal kan opslaan, moet een ander systeem, zoals een server, deze offscreen genereren (hybride client-server structuur). Een warping techniek vergt aanzienlijk minder computatietijd dan gewone rendering en kan dus door de client uitgevoerd worden [42]. Indien de dieptebeelden niet aanwezig zijn en er van de server slechts één (of weinig) dieptebeeld verzonden wordt naar de client, zullen er gaten ontstaan tijdens de warping (door onder andere occlusie). Om dit te vermijden kunnen we gebruik maken van "Layered Depth Images" (splatting)[58]. Door gebruik te maken van een client-server structuur kan het mobiele toestel zelf zorgen voor de warping en de sterke server voor de de constructie van het dynamische model en de opvulling van nieuwe pixels. De voordelen van zo een client-server structuur zijn:

- De server is niet nodig voor warping. Dus de client kan het beeld zelf zonder server aanpassen.
- De server kan gebruik maken van geavanceerde 3D graphics hardware.
- Het wordt mogelijk om resultaten van bestaande 3D graphics programma's te tonen op mobiele toestellen en ermee te interageren.

De server kan natuurlijk ook zelf al het werk verrichten maar dan is het gebruik van beeld-gebaseerde rendering niet meer nodig.
Beeld-gebaseerde rendering is wel veel sneller dan model-gebaseerde rendering, maar de resultaten bij model-gebaseerde rendering zijn altijd beter. Bij beeld-gebaseerde rendering zijn de objecten bovendien statische figuren. Hierdoor is animatie met beeld-gebaseerde rendering niet mogelijk. Voorbeelden van het gebruik van beeld-gebaseerde rendering kunnen gevonden worden in sectie 3.4.3.

3.4.3 Hybride rendering

Dit is een combinatie van model-gebaseerde rendering en beeld-gebaseerde rendering. Hieronder volgen enkele voorbeelden van het gebruik van een combinatie van beide technieken [59].

Vermits een mobiel toestel beperkt is in computatietijd moet er rekening gehouden worden met een eventuele overbelasting. Overbelasting zal leiden tot een te lage framerate, schokkerige beelden, verlies van gevoel van immersie, ...
Als we een scène willen renderen op een mobiel toestel, is het vaak zo dat slechts enkele objecten van de scène relevant zijn op dat moment, bijvoorbeeld dichtbij gelegen of grote objecten. De minder belangrijke objecten kunnen dan snel gerenderd worden via beeld-gebaseerde rendering en de belangrijke met model-gebaseerde rendering. Let wel, de beeld-gebaseerde gerenderde objecten kunnen niet geanimeerd worden tenzij enkel transformaties nodig zijn.

Een andere toepassing wordt gebruikt wanneer objecten in een scène enige tijd vergen om gerenderd te worden. Het is ongewenst dat objecten plotseling één voor één verschijnen op het scherm. We willen van het begin af al zien wat er zich in de scène bevindt. Om dit op te lossen kan er eerst een beeld-gebaseerde representatie weergegeven worden totdat de model-gebaseerde representatie berekend is en weergegeven kan worden.

Nog een andere toepassing van deze techniek is wanneer we een bepaalde framerate willen garanderen. Wanneer model-gebaseerde rendering voor alle objecten teveel computatietijd vergt om de gewenste framerate te behalen, kan overgegaan worden op beeld-gebaseerde rendering. De objecten, in volgorde van minst belangrijk naar belangrijk, worden dan beeld-gebaseerd gerenderd totdat de gewenste framerate behaald wordt.

Deze drie besproken technieken zijn algemene technieken die ook op desktops gebruikt worden. Technieken zoals beeldgebaseerde rendering en hybride rendering komen nog regelmatig voor bij desktops, maar steeds minder en minder omdat desktops almaar krachtiger worden. Deze technieken zijn dan eerder geschikt voor de zwakkere mobiele toestellen.

Hoofdstuk 4

Ontwikkeling

Nu besproken is wanneer en waarom rendering gebruikt kan worden, welke technologiën en ontwikkelingen er bestaan en welke renderingstechnieken het meest voorkomen, gaan we kijken naar de software die ons in staat stelt om applicaties te schrijven voor mobiele toestellen.

4.1 API's

In deze sectie worden enkele dominerende API's (Application Programming Interfaces) voor mobiele toestellen toegelicht. Een API biedt een verzameling abstracte functies aan aan de programmeur. De programmeur kan deze functionaliteit dan gebruiken om niet alles zelf van nul te moeten schrijven maar op een ander, veel hoger, niveau te werken. Een API biedt bijvoorbeeld functionaliteit om driehoeken, polygonen, enz... uit te tekenen. Vaak is een API een onderdeel van een Software Development Kit (SDK)(zie sectie 4.2). Een SDK bestaat uit een verzameling ontwikkelingstools – waar een API deel van kan uitmaken – die een programmeur toelaat om applicaties te schrijven voor een bepaald softwarepakket, framework, hardware platform, computer systeem, spelconsole, besturingsysteem, enzoverder. Alle beschouwde toestellen en producten in deze tekst waarvoor 3D rendering mogelijk is, gebruiken een van de besproken API's. In de implementatie horende bij deze tekst zal OpenGL ES gebruikt worden en deze zal hier dan ook uitgebreid toegelicht worden.

4.1.1 OpenGL ES

OpenGL ES (OpenGL for Embedded Systems)[24] is een cross-platform API voor 2D en 3D graphics op embedded of mobiele toestellen. Het is een goed

gedefiniëerde deelverzameling van de standaard desktop OpenGL en biedt een low-level interface tussen de software en de grafische accelerator. OpenGL ES ondersteunt profielen voor zowel floating-point als voor fixed-point systemen. Ook wordt de EGL specificatie voor portabiliteit naar verschillende besturingsystemen ondersteund. OpenGL ES heeft 3 versies. OpenGL ES 1.x is ontworpen voor fixed function hardware en dient samen met OpenGL ES 2.0 voor zo goed mogelijke acceleratie, beedkwaliteit en performantie. OpenGL ES 2.x ondersteunt programmeerbare 3D graphics. Tenslotte is OpenGL ES-SC, in tegenstelling tot OpenGL ES 1.x en 2.x, specifiek ontworpen voor applicaties waar betrouwbaarheid primair is. We lichten deze drie nu kort toe.

- **OpenGL ES 1.x**: Deze versie is voor fixed function hardware. Onder fixed function verstaan we het paradigma van de laatste 10 jaar: om tot een gerenderd beeld te komen zijn er verschillende operaties en algoritmes ter beschikking. Bij fixed function hardware zijn we beperkt tot enkel deze algoritmes en operaties. We kunnen geen zelfgemaakte toevoegen. OpenGL ES 1.1 is gedefinieerd relatief aan de OpenGL 1.5 specificatie en benadrukt het gebruik van hardware acceleratie. OpenGL ES 1.0 is gedefinieerd aan de OpenGL 1.3 specificatie en benadrukt software rendering. OpenGL ES 1.1 is volledig compatibel met 1.0. Deze versie is ontwikkeld voor mobiele toestellen zoals GSM's en PDA's.

  Zowel OpenGL ES 1.1 als OpenGL ES 1.0 hebben een zogenaamd common en common-lite profiel. Het verschil tussen beide profielen is dat het common-lite profiel enkel fixed-point operaties toelaat en het common profiel floating point. Het common profiel is een wrapper rond het common-lite profiel door floating point getallen om te zetten naar hun equivalente fixed point vorm. Verder voegt OpenGL ES 1.1 aan OpenGL ES 1.0 functionaliteit toe zoals multitextures, automatische mipmapping, vertex buffer objecten, user clip planes en meer controle over point rendering.

- **OpenGL ES 2.x**: OpenGL ES 2.0 is gedefinieerd relatief aan de OpenGL 2.0 specificatie en benadrukt een programmeerbare 3D graphics pipeline met vertex- en fragment shader mogelijkheden door middel van de OpenGL ES Shading Language. Hier kunnen we, in tegenstelling tot OpenGL ES 1.x, wel nieuwe algoritmes en operaties ontwerpen. Deze versie is ontwikkeld voor spelconsoles en andere krachtige systemen.

- **OpenGL ES-SC 1.0**: Bij deze versie speelt betrouwbaarheid een primaire rol. Veiligheid en betrouwbaarheid is bij medische applicaties, militaire en industriele toepassingen een must. Dit "safety critical" profiel garandeert real-time performanties en vergemakkelijkt vertrouwenscertificaten.

Deze drie versies zijn allemaal deelverzamelingen van een desktop OpenGL specificatie en bieden verschillende voordelen aan ontwikkelaars.

Voordelen

- **Industriële standaard**: Iedereen kan OpenGL ES downloaden en gebruiken. Het is een volledige open, multi-platform graphics standaard.
- **Lage eisen en laag verbruik**: OpenGL ES voldoet aan een waaier van specificaties voor mobiele toestellen (van 400Mhz - 64 MB RAM tot 50Mhz - 1 MB RAM) en minimaliseert instructie en data verkeer.
- **Overgangen tussen software en hardware rendering**: Ondanks de graphics pipeline kunnen programma's geschreven worden voor speciale hardware, als software routines of als combinatie van de twee.
- **Uitbreidbaar en evoluerend**: door het gebruik van de zogenaamde "Extensions mechanisms" kan OpenGL ES gemakkelijk met hardware mee evolueren.
- **Gemakkelijk in gebruik**: OpenGL ES is gebaseerd op OpenGL, met hetzelfde intuïtief design en logische commando's.
- **Goed gedocumenteerd**: Er zijn vele boeken, tutorials en voorbeeldcode te vinden.

Doordat een lichtgewicht versie gemaakt moest worden, werd veel van de oorspronkelijke functionaliteit van OpenGL weggelaten en slechts een beetje toegevoegd. Het grootste verschil tussen OpenGL ES en OpenGL is het weglaten van de glBegin en glEnd semantiek voor het renderen van primitieven. Enkel nog vertex arrays worden toegelaten. Fixed point data types werden toegevoegd om de computationele kracht van mobiele toestellen beter te ondersteunen. Enkele van de weggelaten functionaliteiten zijn: quad en polygon rendering, texgen, line en polygon stipple, polygon mode, ARB_Image klasse pixel operaties, bitmaps, 3D textures, tekenen op de frontbuffer, pixels kopiëren, display lists, push en pop state attributen, enz...

4.1.2 Direct3D Mobile

Microsoft Direct3D Mobile is een API dat ondersteuning geeft aan 3D applicaties op Windows CE-gebaseerde platformen [8]. Deze API is afgeleid van de originele Direct3D API in standaard desktop systemen en is vervolgens geoptimaliseerd voor gebruik in mobiele toestellen. Enkele belangrijke eigenschappen zijn:

- Veel minder zwaar dan de gewone Direct3D. Veel functionaliteit is weggelaten om beter om te gaan met de beperkingen van de hardware aanwezig in mobiele toestellen.
- De architectuur laat toe drivers te implementeren die gebaseerd zijn op enkel software, enkel hardware of een combinatie van beide.
- Direct3D Mobile heeft een multi-type architectuur: floating point waarden, fixed point waarden, ...

Net zoals OpenGL ES een lichte versie is van OpenGL, is Direct3D Mobile een lichte versie van de desktop Direct3D. Direct3D Mobile lijkt op Direct3D 8 met enkele elementen van Direct3D 9. De belangrijkste functionaliteiten zijn:

- Laden, transformeren en manipuleren van primitieven
- Ondersteuning voor 3D z-buffers
- Verwisselbare diepte buffers (z- en w-buffers)
- Flat en Gouraud shading
- Meerdere lichtbronnen
- Material en texture ondersteuning (inclusief mipmapping)
- Transformaties en clipping
- Hardware onafhankelijk

Veel functionaliteit is weggelaten om de resources van mobiele toestellen beter te gebruiken. Zware operaties zijn bijvoorbeeld weggelaten om de batterijduur te verlengen. We geven een kleine opsomming van de weggelaten functionaliteit: Bump mapping, cube maps, doorschijnende materialen, gamma correctie, hoge orde primitieven, line stippling, phong shading, pixel shaders, point sprites, spot lights, user clip planes, vertex shaders, volume textures, ...

4.1.3 M3G

M3G staat voor Mobile 3D Graphics [20]. Dit is een API voor J2ME (Java 2 Micro Edition) en wordt ook wel JSR-184 genoemd. J2ME is een ontwikkelingsplatform voor mobiele toestellen. M3G breidt J2ME uit door 3D graphics mogelijk te maken. M3G is een object georiënteerde interface die uit meer dan 30 klassen bestaat die helpen om complexe, geanimeerde 3D scènes te renderen. M3G is ontwikkeld rekening houdend met de noden van mobiele toestellen. De architectuur van deze API maakt zowel software als hardware rendering mogelijk.

Vaak wordt M3G verward met Java 3D. Java 3D breidt de mogelijkheden van het gewone Java platform uit en is ontwikkeld voor desktop PC's. Java 3D en M3G zijn twee aparte en incompatibele API's, ontwikkeld voor verschillende doeleinden. M3G is speciaal ontwikkeld voor mobiele toestellen rekening houdend met de beperkingen ervan. Java3D is een scene-graph gebaseerde 3D API voor het Java platform.

Met M3G kunnen 3D graphics op twee manieren getekend worden: in immediate mode of in retained mode.

- **Immediate mode**: Hier kunnen graphics commando's rechtstreeks aan de graphics pipeline gegeven worden. Zo kunnen deze onmiddellijk door de rendering engine uitgetekend worden. Bij deze methode moet de ontwikkelaar code schrijven die specifiek zegt wat de rendering engine elke frame moet tekenen, zoals bij OpenGL ES en Direct3D.

- **Retained mode**: In retained mode wordt gebruik gemaakt van een scene-graph die alle objecten in de scène aan elkaar linkt in een boom structuur. Deze methode werkt op een veel hoger niveau. Hierbij moeten de objecten en hun eigenschappen in de scene-graph voor elke frame bepaald worden.

Verder heeft M3G een modelloader voor .obj bestanden waarbij ook keyframed animatie mogelijk is. Hierdoor kunnen modellen eenvoudig ingeladen worden op mobiele toestellen.

4.2 SDK's

Een Software Development Kit (SDK) is een verzameling van tools waarmee programmeurs applicaties kunnen schrijven voor bepaalde specifieke systemen (zie ook 4.1). In dit geval, het renderen van 2D of 3D beelden op een

mobiel toestel. Merk op dat een API een onderdeel kan zijn van een SDK. Verder kan een SDK nog bestaan uit andere tools zoals hulpmiddelen voor debugging, voorbeeldcode, technical notes, ... In dit hoofdstuk bespreken we enkele SDK's.

4.2.1 BREW SDK

Qualcomm biedt met de BREW SDK [7] (Binary Runtime Environment for Wireless) een volledig software pakket aan om applicaties te ontwikkelen voor mobiele toestellen. BREW bestaat uit een grote verzameling API's om software te kunnen ontwikkelen in C/C++ en Java. Er worden debugging tools, voorbeeld applicaties met source code, referenties, user guides en een emulator aangeboden aan de ontwikkelaar. De footprint van deze SDK is ongeveer 150K. Om applicaties voor verschillende platformen en toestellen met verschillende capaciteiten te kunnen ontwikkelen, zijn er vijf versies van deze SDK uitgebracht.

- **BREW SDK 1.0**: Dit is de eerste SDK en deze bevat alle basisfunctionaliteit. Er bestaan geen commercieel verkrijgbare toestellen meer die deze versie ondersteunen. Vroegere modellen die 1.0 ondersteunden waren de Motorola V731 en de Sharp Z800

- **BREW SDK 1.1**: Deze versie is reeds ruim verspreid en ondersteunt toestellen zoals de Motorola T-720 en de LGE VX4400. Deze versie voegt onder andere encryptie, webprotocollen en licentiemogelijkheden toe.

- **BREW SDK 2.0**: Toestellen die deze versie ondersteunen zijn: De LGE VX6000, de Kyocera SE47 (Slider), en de Samsung SCH-A670. Toevoegingen zijn in-memory bitmaps, mogelijkheid tot DNS queries, verschillende tekstoperaties, bluetooth interfaces,...

- **BREW SDK 2.1**: Gebruikt door de Toshiba CDM-9900. Bij deze versie worden definities van een 3D engine gegeven om driehoeken en groepen van driehoeken (modellen) uit te tekenen. Deze functionaliteit wordt in versie 3.0 weer weggelaten. Andere toegevoegde functionaliteit is toegang tot digitale camera's en encoderingen van media bestanden.

- **BREW SDK 3.0**: Dit is de laatste versie en laat bijvoorbeeld toe om externe hardware aan te sluiten en externe beeldschermen te gebruiken.

Welke versie ook wordt gebruikt, BREW heeft enkele interessante eigenschappen en mogelijkheden.

- BREW is "thin": BREW is geen verkleinde versie van bestaande software voor PC's of PDA's. BREW is "van de chip uit" speciaal ontworpen om meerdere keren kleiner te zijn dan andere applicatie-platformen of besturingssytemen.

- BREW is snel: het BREW platform bevindt zich net boven de systeemchip software. Hierdoor kunnen snelle applicaties geschreven worden (C/C++) en is de integratie van browsers, virtual machines (Java) en andere extensies eenvoudig.

- BREW is "open": Naast C/C++ en Java ondersteunt BREW ook nog andere talen en "execution environments" zoals XML en Flash.

- BREW is uitbreidbaar: Doordat de User Interfaces bovenop BREW ontwikkeld zijn, kunnen de ontwikkelaars of derden zelf nieuwe functionaliteit toevoegen met de BREW extensions.

- BREW is kost efficiënt: Doordat er minder tijd nodig is voor het ontwikkel en integreren van applicaties, komen deze sneller op de markt.

- BREW is veilig: Alle draadloze operaties zijn volledig beveiligd. Verder wordt er gebruik gemaakt van verschillende authenticatie services. Daarbij gebruikt BREW zelf ook nog digitale handtekeningen om alle naar het toestel gedownloade bestanden te verifiëren.

Verder heeft BREW verschillende voordelen. Door middel van de BREW Porting Kit kan BREW snel en gemakkelijk op mobiele toestellen gezet worden. BREW is onafhankelijk van een "air-interface", wat wil zeggen dat deze GPRS, UMTS en CDMA ondersteunt. BREW applicaties kunnen gemakkelijk tussen alle ASIC's van Qualcomm geport worden. Verder bevindt BREW zich tussen de applicaties en besturing van de wireless chip, waardoor de programmeur op een hoog niveau wireless applicaties kan schrijven.

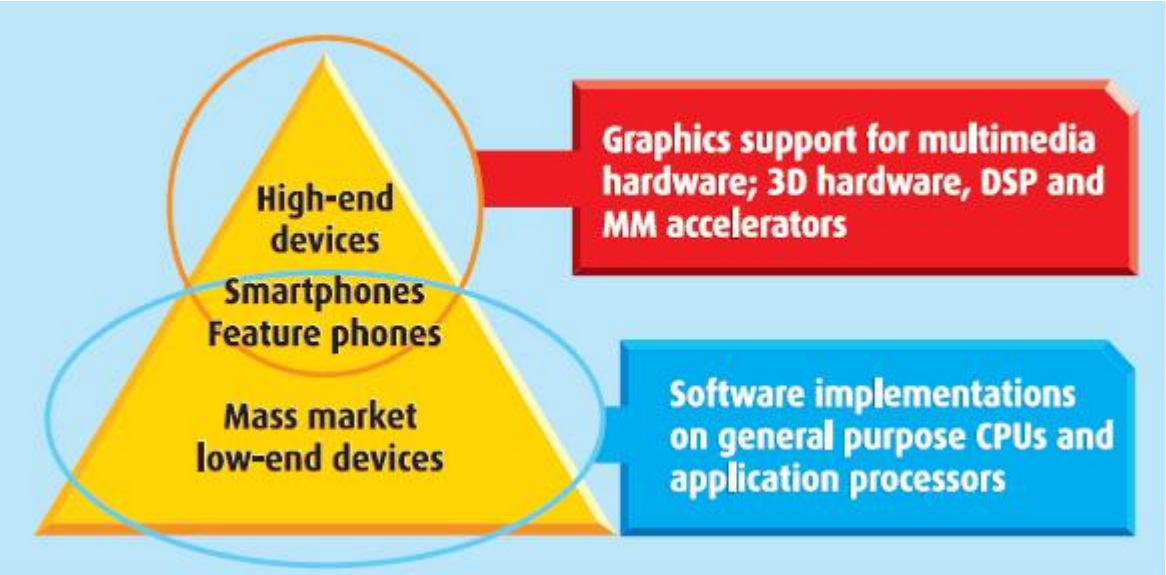
Zoals we reeds vermeld hebben kan software voor BREW-toestellen ontwikkeld worden in C, C++ of Java. OpenGL ES wordt ondersteund. In het implementatie-gedeelte van deze thesis wordt de BREW SDK uitgetest en besproken.

4.2.2 Vincent 3-D Library

Vincent [38] is een 3D renderingsbibliotheek voor mobiele toestellen die werkt met een software implementatie van de OpenGL ES 1.1 API voor Pocket PC's en Smartphones. Deze bibliotheek richt zich op Pocket PC's die gebruik maken van de Intel XScale (zie 2.3.2). De huidige versie van Vincent kan reeds succesvol de officiële OpenGL ES 1.0 conformance test uitvoeren en, op een paar details na, ook de OpenGL ES 1.1 test. Vincent kan gebruikt worden voor applicatie-ontwikkeling op mobiele toestellen en om OpenGL applicaties te porten naar mobiele toestellen.
Een belangrijk punt van deze bibliotheek is om een runtime compilatie infrastructuur te bieden die geoptimaliseerde rasterisatie code creëert voor de graphics context die gebruik maakt van een JIT compiler. Wat wil dit nu precies zeggen? Elke oproep naar "glDrawArrays" of "glDrawElements" zorgt ervoor dat de rasterizer de interne instellingen opnieuw valideert en herinitialiseert vooraleer de geometrische primitieven naar de rendering pipeline gestuurd worden. Bij deze herinitialisatie gebruikt de rasteriser een code generator (JIT compiler) om een geoptimaliseerde functie voor de scanline conversie te maken. Door gebruik te maken van een function cache, wordt recompilatie tot een minimum gehouden. De JIT compiler is aanwezig in de bibliotheek en gebruikt een variant van Johnson's triVM intermediate language [18].
Het doel van de Vincent 3D library is om te werken naar een implementatie van het "common" profiel van OpenGL ES (zie 4.1.1) voor toestellen werkend met Windows Mobile. Deze SDK werkt, in tegenstelling tot de BREW SDK, met floating point getallen.

4.2.3 Hybrid Graphics, Ltd.

Hybrid [14] ontwikkelt online en mobiele grafische technologieën en houdt zich bezig met implementaties van verschillende mobiele grafische standaarden voor mobiele toestellen. Hybrid biedt een mobiel framework aan. Dit framework bevat implementaties van de bekendste standaarden voor mobiele toestellen, namelijk OpenGL ES, OpenVG en M3G. Met dit framework doelen ze op drie afzetmarkten. Ten eerste doelen ze op producten zonder specifieke rendering hardware; ze noemen dit "mass market" toestellen. Ten tweede richten ze zich op toestellen met multimedia hardware, zogenaamde "multipurpose" toestellen. Tenslotte richten ze zich op high-end toestellen. Dit zijn toestellen met specifieke 3D hardware (zie 2.3.2). Figuur 4.1 illustreert deze drie categorieën.

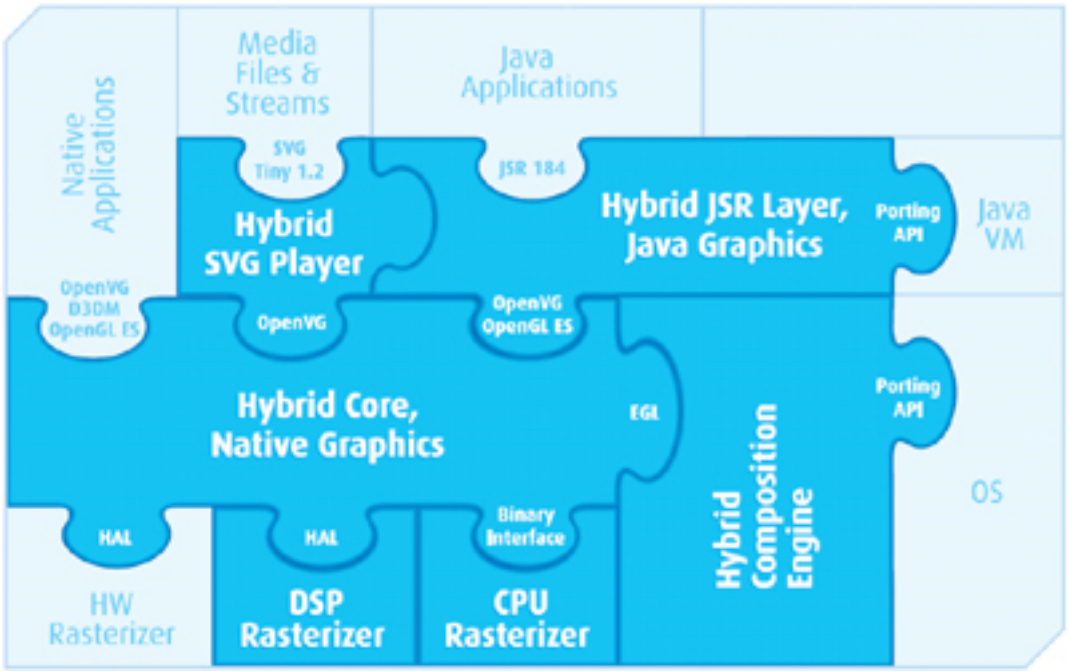


Figuur 4.1: De drie afzetmarkten van Hybrid en de aangeboden oplossingen [14].

Voor de drie categorieën worden oplossingen aangeboden. Voor de mass market toestellen wordt een volledige en op voorhand geïntegreerde software implementatie aangeboden. Voor de toestellen met multimedia hardware - de zogenaamde multipurpose toestellen - zijn hardware geaccelereerde en geoptimaliseerde implementaties voorzien. Voor de high-end toestellen tenslotte, is een volledige hardware driver infrastructuur voorzien.

Er bestaan momenteel twee versies van dit framework.
De eerste heet "Mobile Framework v5" en is een standalone 3D graphics framework. Deze is ontwikkeld voor toestellen werkend met een ARM of SH3 processor. Het framework implementeert OpenGL ES en M3G en richt zich op toestellen zonder graphics acceleratie hardware.

Het tweede framework heet "Mobile Framework v6" en bestaat uit dezelfde basiscomponenten als Framework v5. Verder zijn er nog vele extra componenten toegevoegd, zoals bijvoorbeeld 2D vector graphics (zie sectie 9.1: bijlage A) en ondersteuning voor Direct3D Mobile. Een grafische voorstelling van dit framework kan gezien worden in figuur 4.2. In de Hybrid Core bevinden zich de 2D en 3D graphics functies en deze bevatten de implementaties van de OpenGL ES en OpenVG standaarden. De Hybrid JSR Layer geeft Java-



Figuur 4.2: Hybrid's Framework v6 [14].

gebruikers toegang tot de 2D en 3D functionaliteit van de Hybrid Core. Deze interface implementeert M3G. Verder zorgt de Hybrid Composition Engine voor low-level layering en surface management. De Hybrid SVG Player dient voor het weergeven van vector graphics.

Deze twee versies bevatten uiteraard een SDK voor ontwikkelaars om te programmeren in OpenGL ES, M3G of Direct3D Mobile. Mobile Framework v7 zal OpenGL ES 2.0 implementeren. Dit framework is nog in ontwikkeling en wordt nog in 2006 verwacht.

4.2.4 Klimt 3D Library

Klimt [1] is een open-source 3D bibliotheek gericht op PDA's en GSM's. De bijhorende API heeft vele gelijkenissen met OpenGL ES en OpenGL (maar voldoet niet aan de conformance tests). Klimt ondersteunt daarbij ook nog Coin3D, een grote scene-graph bibliotheek die op OpenGL gebouwd is. Deze bibliotheek is gebaseerd op OpenGL ES, maar is geen gecertificeerde implementatie ervan.

Klimt richt zich op hardware onafhankelijkheid en goede performantie. De enigste eis die gesteld wordt van het mobiele toestel is een 16 bits (RGB565) lineaire framebuffer. In de minimale versie zijn geen dependencies of verdere bibliotheken nodig. De volgende platformen zijn ondersteund: Windows CE (maar ook Windows XP en Windows 2000), Pocket PC 2002 & 2003, MS Smartphone, Linux op de IPAQ, Shart Zaurus en X11.
We gaan eens kijken naar de huidige functionaliteit van Klimt.

- Volledige ondersteuning van projectie en modelview matrix
- Volledige ondersteuning van texture matrices
- Alle primitieven van OpenGL
- Vertex lighting
- Perspectief en affiene texturing
- Video background (glDrawPixels)
- Display lists & Attribute stacks

Weggelaten functionaliteit is bijvoorbeeld polygon stippling, antialiasing, alpha buffer, ...
Klimt gelijkt op zowel OpenGL als op OpenGL ES. Maar wat verschilt er nu precies? Wat kan Klimt meer of minder? We maken een vergelijking.

We zien dus vele gelijkenissen met zowel OpenGL als OpenGL ES [1]. Bepaalde functionaliteit, zoals bijvoorbeeld de glBegin - glEnd semantiek, kan nu gebruikt worden voor mobiele toestellen. Andere functionaliteit is dan weer weggelaten om een snelle en eenvoudige implementatie toe te laten (zoals alpha tests, stencil test, gecompresseerde textures, multitextures,...). Applicaties kunnen ontwikkeld worden met Embedded VisualC++ 3.0 en 4.0, VisualC++ 6.0 en Visual Studio .NET 2003. Verder kan er ook gekozen worden of met fixed point (16.16 of Intel GPP) of floating point operaties gewerkt wordt.

4.2.5 PowerVR

De PowerVR "MBX Family" SDK [31] biedt allerlei SDK's aan voor toestellen werkend op PowerVR MBX [32] platformen. Deze SDK's bevatten voorbeeldcode, demo's, documentatie en allerlei utilities om de ontwikkelaar te helpen applicaties te schrijven, gebruikmakend van de OpenGL ES graphics library. PowerVR biedt een hele reeks SDK's aan, afhankelijk van de aanwezige software platformen:

	OpenGL	**OpenGL ES**	**Klimt**
glBegin - glEnd	Ja	Nee	Ja
Primitieve types	Allemaal	Geen quads, quad strips of polygonen	Allemaal
Vertex Arrays	Ja	Ja	Ja
Data types	Float,double, int,enz...	Float en fixed	Float, double, int, fixed
Polygon stipple	Ja	Nee	Nee
glDraw/Read Pixels	Ja	Nee	Nee
Textures	Alle types	2D	2D
Texture wrap en repeat	Ja	Wrap, clamp_to_edge	Wrap
Gecompresseerde textures	Ja	Ja	Nee
Multitexture	Ja	optioneel	Nee
Fog	Ja	Ja	Ja
Alpha test	Ja	Ja	Nee
Stencil test	Ja	optioneel	Nee
Depth test & blending	Ja	Ja	Ja
Clipplanes	Ja	Nee	Nee
WGL functies	Ja	Nee	Nee
EGL functies	Nee	Ja	Ja

Tabel 4.1: Vergelijking tussen OpenGL, OpenGL ES en Klimt

- **Windows PC Emulation**: Deze OpenGL ES 1.0 en 1.1 SDK richt zich op gewone desktop PC's. Ontwikkelaars kunnen de SDK zo instellen dat deze hun gewenste MBX platform nabootst. Zo kunnen applicaties gemakkelijk ontwikkeld en getest worden zonder de rechtstreekse nood aan een mobiel toestel werkend op een MBX platform. Het ondersteunde platform voor deze SDK is een Windows XP PC met OpenGL hardware ondersteuning.

- **Windows Mobile 5 Pocket PC, OpenGL ES Common**: Een OpenGL ES 1.1 SDK voor ARMV4I instruction set CPU's. Het ondersteunde platform voor deze SDK is: Texas Instruments' OMAP2420 Software Development Platform.

- **Windows Mobile 5 Smartphone**: Een OpenGL ES 1.1 SDK voor ARMV4I instruction set CPU's. Het ondersteunde platform is hetzelfde als voor Windows Mobile 5 Pocket PC: Texas Instruments' OMAP2420 Software Development Platform.

- **Windows Mobile 5 Pocket PC, OpenGL ES CommonLite**: OpenGL ES 1.0 SDK voor een ARMV4I instruction set CPU. Ondersteund platform: Dell Axim X51v.
- **Pocket PC ARM V4 (Windows CE 4.2)**: Een OpenGL ES 1.0 SDK voor een ARMV4 instruction set CPU. Ondersteunde platformen: De Dell Axim X50v/Axim X51v en de Intrinsyc Carbonado.
- **Linux MX31**: Een OpenGL ES 1.1 SDK voor een ARMV6 instruction set CPU. Ondersteund platform: Freescale Semiconductor's i.MX31 multimedia applications processor.
- **Linux ARMV4**: Een OpenGL ES 1.1 Linux SDK voor een ARMV4 instruction set CPU (met besturingssysteem: ARM Linux 2.6 en windowing system: X11 or NULL). Ondersteunde platformen: ARM Versatile Platform Baseboard for ARM926EJ-S en de ARM Versatile Flash AB926EJ-S.
- **Linux MontaVista ARMV6**: Een OpenGL ES 1.1 Linux SDK voor een ARMV6 instruction set CPU (met besturingssysteem: MontaVista 2.4 en windowing system: X11 or NULL). Ondersteund platform: Texas Instruments' OMAP2420 Software Development Platform.
- **Symbian ARMV4 en Symbian ARMV6**: Een OpenGL ES 1.1 Symbian SDK voor een ARMV4, respectievelijk ARMV6 instruction set CPU (met besturingssysteem: Symbian en windowing system: TextShell of TechView). Ondersteunde platformen zijn: De ARM Versatile Platform Baseboard for ARM926EJ-S en de ARM Versatile Flash AB926EJ-S voor de ARMV4 chip en de Texas Instruments' OMAP2420 Software Development Platform voor de ARMV6 chip.

Elk van deze SDK's is vervolgens opgesplitst in volgende directories:

- **Builds**: Hierin zitten alle platform specifieke bestanden/libraries/includes (zie bovenstaande lijst).
- **Demo's en Extra's**: Deze directories bevatten een reeks voorbeeldprogramma's die allerlei OpenGL ES mogelijkheden illustreren zoals belichting, texturing, particles, cell shading, skyboxes...
- **Documentation**: Hier vinden we allerhande nuttige informatie zoals tips over hoe het best programmeren voor PowerVR, een overzicht van de PowerVR technologie uit een hardware perspectief, hoe de texture compressie van PowerVR werkt,... Ook wordt documentatie voor

enkele OpenGL ES extenties [24] gegeven: EGL_IMG_power_management, GL_IMG_texture_compression_pvrtc,
GL_IMG_texture_env_enhanced_fixed_function, ...

- **Shell**: OGLESShell is een framework om snel OpenGL ES applicaties te kunnen schrijven. Deze shell helpt met vele taken zoals bijvoorbeeld opstarten, afsluiten, windowing, inititalisatie, input en output. Alle meegeleverde demo's maken gebruik van deze Shell.

- **Tools**: OGLESTools is een verzameling van bestanden die vele helperfuncties bevat om taken in 3D graphics te vergemakkelijken. Enkele voorbeelden:

  - Fixed-point ondersteuning
  - Matrix math
  - MD2 file loader
  - POD file library
  - Print3D voor rendering van 2D text on-screen

- **Utilities**: Deze directory bevat utilities voor de PC waarop de applicatie ontwikkeld wordt. Zo kunnen 3DSMAX bestanden geconverteerd worden naar data leesbaar door PowerVR, een texture compressie tool om zelf textures te kunnen maken ondersteund door PowerVR, ...

We zien dus dat we met een SDK rechtstreeks van start kunnen gaan om OpenGL ES applicaties te schrijven. Het is aangeraden om steeds eerst de Desktop PC Emulator te gebruiken voor snellere ontwikkeling en efficiënte debugging. Naderhand moeten enkel de shell en libraries vervangen worden voor het specifieke toestel waarop gericht wordt.

Deze SDK heeft enkele interessante voordelen. Snel en gemakkelijk van start kunnen gaan met de bijgeleverde shell. Vele taken zoals opstarten, afsluiten en initialisatie, worden van de gebruiker afgeschermd. Goede texture compressie (met bijgeleverde tools) om geheugenverbruik te minimaliseren. Complexe vertexdata (3dmax) is eenvoudig te gebruiken.

Hoofdstuk 5

Toepassingen

In hoofdstuk 2 hebben we besproken waarom en wanneer rendering van 2D of 3D beelden nuttig kan zijn. Beelden kunnen communicatie verbeteren, vergemakkelijken of simpelweg leuker maken. In hoofdstuk 3 hebben we gezien welke renderingstechnieken er bestaan. Dit hoofdstuk zal enkele praktische toepassingen van rendering beschouwen. Merk op dat steeds één van de technieken besproken in hoofdstuk 3 gebruikt wordt.

5.1 Navigatie

Navigatiesystemen spelen een belangrijke factor bij mobiele toestellen. GPS systemen zijn bijvoorbeeld ruim aanwezig. Om de bruikbaarheid van zulke navigatiesystemen te verhogen, is het nodig dat de informatie in een vorm aangeboden wordt die het best bruikbaar is voor de gebruiker.

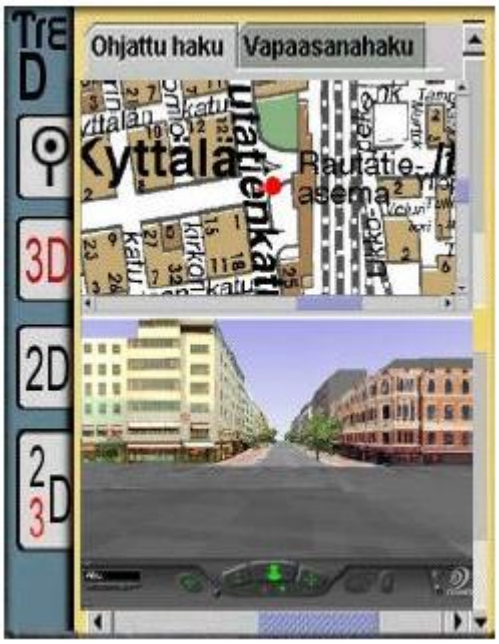
Zoals we in sectie 2.2 reeds hebben vermeld, kan een 3D voorstelling gegevens duidelijker of interessanter maken. Dit wordt aangetoond door een studie in [52]. Men toont aan dat zoeken naar bepaalde locaties in een stad door het gebruik van 3D rendering, intuïtiever en gemakkelijker is dan door het gebruik van een 2D representatie. Alle gegevens van de stad worden eerst via een server naar het mobiele toestel gestuurd. Vervolgens kunnen er, zonder verder gebruik te maken van bandbreedte van server naar client, onbeperkt veel interactieve beelden gerenderd worden van de stad. Er wordt gebruik gemaakt van VRML. Windows CE ondersteunt immers een VRML browser. Zeer realistische beelden konden gerealiseerd worden door textures van hoge kwaliteit te gebruiken (zie figuur 5.1 [52]). Op deze figuur zie je bovenaan het echte beeld en onderaan de VRML rendering. Merk op hoe realistisch het beeld is door gedetailleerde textures tegen eenvoudige modellen

Figuur 5.1: Screenshot van de stad (VRML) [52].

te plakken.

De modellen van de gehele stad zijn zo eenvoudig mogelijk gehouden en de textures zo gedetailleerd mogelijk. De gemodelleerde stad bevat slechts 5100 polygonen, de modellen zijn 120 KB groot en de textures zijn 5 MB groot. Een beeld van de uitgewerkte toepassing in [52] kan gezien worden in figuur 5.2. Merk ook op dat een duidelijke en intuïtieve interface nodig is. Het is niet gemakkelijk om op een klein scherm voldoende informatie weer te geven. Het is storend en ongewenst wanneer er gescrolled of continu tussen schermen gewisseld moet worden.

In [41] en [45] wordt aangetoond dat virtuele 3D omgevingen de echte omgevingen van de gebruikers kunnen simuleren en zo kunnen helpen intuïtief en gemakkelijk een locatie te vinden. De gebruiker kan interactief op het mobiele toestel in de omgeving navigeren. Door een begin- en eindpunt te specifiëren kan een pad berekend worden en kan de volledige route voor de gebruiker in 3D gerenderd worden. Er wordt een server gebruikt om de beelden te renderen en te compresseren. Deze worden vervolgens uit de framebuffer gehaald en naar een PDA gestuurd. Beide papers gebruiken hun eigen compressiemethode om de beelden te verkleinen en zo de bandbreedte



Figuur 5.2: Een afbeelding van de applicatie uit [52]

te beperken.

5.2 Datavisualisatie

Vele hedendaagse bedrijven en producenten maken gebruik van 2D/3D visualisatie van data. In vele gevallen is het onoverzichtelijk om grote hoeveelheden gegevens te beschouwen of er een verband tussen te vinden. Een goede grafiek of grafische voorstelling kan vaak onmiddellijk een beter beeld geven. Als deze beelden dan nog eens interactief én op locatie kunnen bekeken worden op een mobiele client (PDA of Tablet), dan wordt datavisualisatie pas echt krachtig. We geven enkele praktische toepassingen van de visualisatie van data.

In [43] wordt een methode en software-implementatie aangeboden die ingenieurs en anderen toelaat om gedetailleerde 3D CAD (Computer Aided Design) modellen interactief te bekijken op een PDA. De software implementatie laat volledige duplex communicatie toe wat real-time performantie

Figuur 5.3: Een afbeelding van de applicatie uit [45]

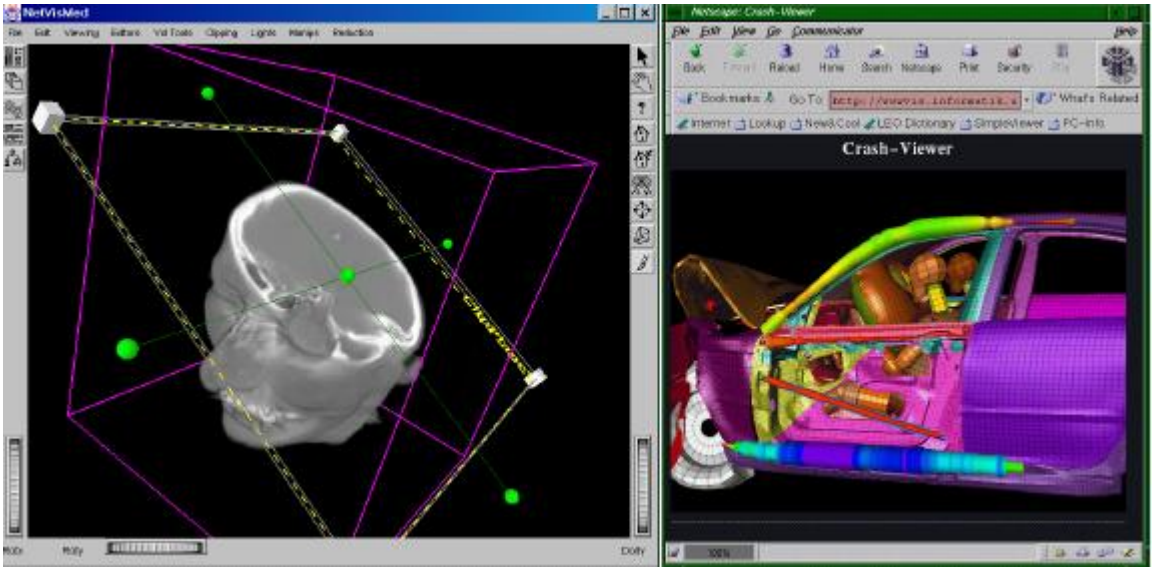
biedt. Het principe van deze methode is eenvoudig: een centrale server rendert het model en stuurt het gerenderde beeld gecompresseerd naar de PDA. Zo kan een ingenieur op locatie de constructie of reparatie van een toestel of gebouw opvolgen. Zie figuur 2.4 voor een voorbeeld.

Ook [53] maakt gebruik van een server om de beelden te renderen. Deze visualisatie-server werkt met Open Inventor en Cosmo 3D gebaseerde applicaties. Er wordt een framework aangeboden dat interactie van 3D beelden toelaat op mobiele toestellen. De visualisatie-server stuurt de gecompresseerde beelden vanuit zijn frame buffer naar Java gebaseerde clients. Verdere visualisatieparameters en GUI-events zijn mogelijk via CORBA-aanvragen naar de server.

In tegenstelling tot de twee vorige toepassigen gebruiken [48] en [39] niet één, maar een cluster van visualisatie-servers. De Chromium architectuur wordt gebruikt: elke server van de cluster rendert een deel van het scherm [49]. Vervolgens worden de aparte delen samengevoegd, gecompresseerd en naar de client gestuurd. Ook hier wordt de zwakke computatiekracht van de client omzeild. Merk op dat door gebruik te maken van meerdere krachtige visualisatieservers, zeer gedetailleerde en realistische beelden gegenereerd kunnen worden.

Al deze methoden bieden transparante toegang tot één of meerdere visualisatie-servers op afstand, waardoor de belasting gedeeld wordt door server en

client. Hierdoor wordt de zwakke computatiekracht van de client omzeild en kunnen er interactief beelden van hoge kwaliteit bekeken worden. In figuur 5.4 [53] zien we twee beelden van deze toepassingen. Links zien we een 3D visualisatie van een aangezicht op een Java gebaseerde client. Dit medisch beeld kan via manipulatoren interactief bekeken worden door dokters en medisch personeel. Rechts zien we een 3D visualisatie van een virtuele crash test. Dit beeld is te zien in een HTML browser op een PDA. Ook dit beeld kan interactief bekeken worden.



Figuur 5.4: Links: medisch beeld van een aangezicht [53]. Rechts: een virtuele crash test [53]

Merk op dat interactieve beelden van zeer hoge kwaliteit zonder de servers, niet of niet snel gerenderd kunnen worden. De performantie is dus vooral afhankelijk van de rekenkracht van de server(s) en van de vertraging die wordt opgelopen door het versturen van de beelden van de server naar de client.

Ook in het algemeen zien we dat bij visualisatie van grote hoeveelheden data op mobiele toestellen, steeds gebruik gemaakt wordt van servers voor de rendering. Kan een PDA dit niet zelf? Het antwoord hierop is in vele gevallen negatief. Door de beperkingen in opslagcapaciteit is het vaak niet mogelijk al deze data lokaal op te slaan. Ook de verwerking en het renderen van de data zou te veel tijd vergen en als gevolg ongewenste resultaten leveren.

Een voorbeeld van lokale rendering op een mobiel toestel zonder tussenkomst van een server is te vinden in [54] en in de case study horende bij deze thesis (zie sectie 6.1). In [54] wordt gebruik gemaakt van VRML modellen om het skelet van een gebouw te visualiseren. Balken en pilaren van hetzelfde materiaal of met gelijke functie worden in dezelfde kleur uitgetekend. De gebruiker van het mobiele toestel kan interactief in de scène navigeren. Deze toepassing is zichtbaar op figuur 2.4 [54].

5.3 Augmented reality

Het principe van augmented reality is eenvoudig: we hebben een foto of beeld van een omgeving en we plaatsen er een digitaal gegenereerd object bij. Stel dat we een nieuwe zetel willen kopen, zou de kleur en het model in de woonkamer passen? Door een digitale voorstelling van de zetel in de woonkamer kan men dit snel en gemakkelijk te weten komen. Zie figuur 5.5 [47].

Figuur 5.5: Illustratie van augmented reality: Een virtuele zetel in een woonkamer [47].

Augmented reality wordt reeds enkele jaren gebruikt door architecten. Een gepland gebouw kan op zijn locatie getoond worden voordat dit eigenlijk gebouwd is.
[47] maakt augmented reality een real-time proces. Een PDA wordt met een

camera uitgerust en filmt continu beelden. Deze beelden worden naar een laptop gestuurd die de positie en richting van de camera aan de hand van herkenningpunten in de omgeving analyseert. De laptop plaatst vervolgens het virtuele model in het opgenomen beeld en stuurt het terug naar de PDA. De PDA moet dit beeld dan enkel nog weergeven. We verduidelijken dit even met figuur 5.6 [47]. Bij de twee wagens staan herkenningpunten. Zo kan de

Figuur 5.6: illustratie van real-time augmented reality [47].

laptop te weten komen hoe de camera van de PDA op de scène gericht is en waar het stadion geplaatst moet worden.

Een ander voorbeeld van augmented reality wordt gegeven door [60]. Hier wordt aan de hand van het herkennen van symbolen op kaarten een spel gespeeld.

5.4 Games

Doordat mobiele toestellen steeds krachtiger worden, wordt ook de vraag naar mooiere en complexere spelletjes groter. Spelletjes op mobiele toestellen is een zeer grote markt en er heerst reeds een enorme concurrentiestrijd. Kijk maar naar de Sony PSP, Nintendo DS, Nokia NGage, en talloze GSM toestellen. Spel ontwikkelaars willen dan ook het meeste halen uit deze toestellen en de beste en mooiste games ontwikkelen.
Twee bekende spel ontwikkelaars zijn Ideaworks3D [15] en Fathammer Ltd

[10]. Beide ontwikkelen de laatste generatie 3D games voor mobiele toestellen en benutten de hardware zo optimaal mogelijk voor de beste resultaten. In figuur 5.7 ziet u een foto van een racing spel ontwikkeld door Fathammer.

Figuur 5.7: Een screenshot van een racing game ontwikkeld door Fathammer Ltd [10].

Evenaren de huidige games voor mobiele toestellen die van onze huidige desktop PC's of spelconsoles? Neen. De grafische mogelijkheden van de nieuwste games voor mobiele toestellen kunnen vergeleken worden met die van spelconsoles van ongeveer 6 jaar geleden (Sony Playstation).

Hoofdstuk 6

Implementatie

Dit hoofdstuk bestaat uit twee delen. Enerzijds een case study waarin de stad Hasselt in 3D uitgetekend wordt en anderzijds kleinere testimplementaties om enkele SDK's uit sectie 4.2 te testen. Deze implementaties gebeuren op een PDA (of emulator) en de gebruikte API is OpenGL ES.

6.1 Case Study: renderen van een stad

6.1.1 Doelstelling

Het doel van deze implementatie is om lokaal op het toestel een real-time walkthrough door een stad te renderen in first-person view. Een stad bestaat gemakkelijk uit duizenden muren. Het is daarom niet efficiënt om de gehele stad en bijhorende textures naar de rendering pipeline te sturen. Er moet een techniek geïmplementeerd worden die enkel de zichtbare muren - en eventueel ook enkele onzichtbare muren - voor elke frame bepaalt. Hierdoor wordt de rendering pipeline minder belast en worden geheugenoperaties geminimaliseerd. Geheugenoperaties zijn immers batterijverslindend en doordat de rendering pipeline ontlast is, zal het programma sneller renderen.

6.1.2 Algoritme

In deze sectie wordt het geïmplementeerde algoritme toegelicht. Het algoritme zal voor elke frame van de walkthrough altijd alle op dat moment zichtbare muren correct uittekenen. Het grootste gedeelte van de onzichtbare muren wordt niet naar de rendering pipeline gestuurd. Hierdoor worden al meteen duizenden muren weggelaten. Het belangrijkste aspect van dit algoritme is preprocessing. Omdat we in real-time renderen, willen we alle berekeningen in verband met de zichtbaarheid voor elke frame reeds gedaan

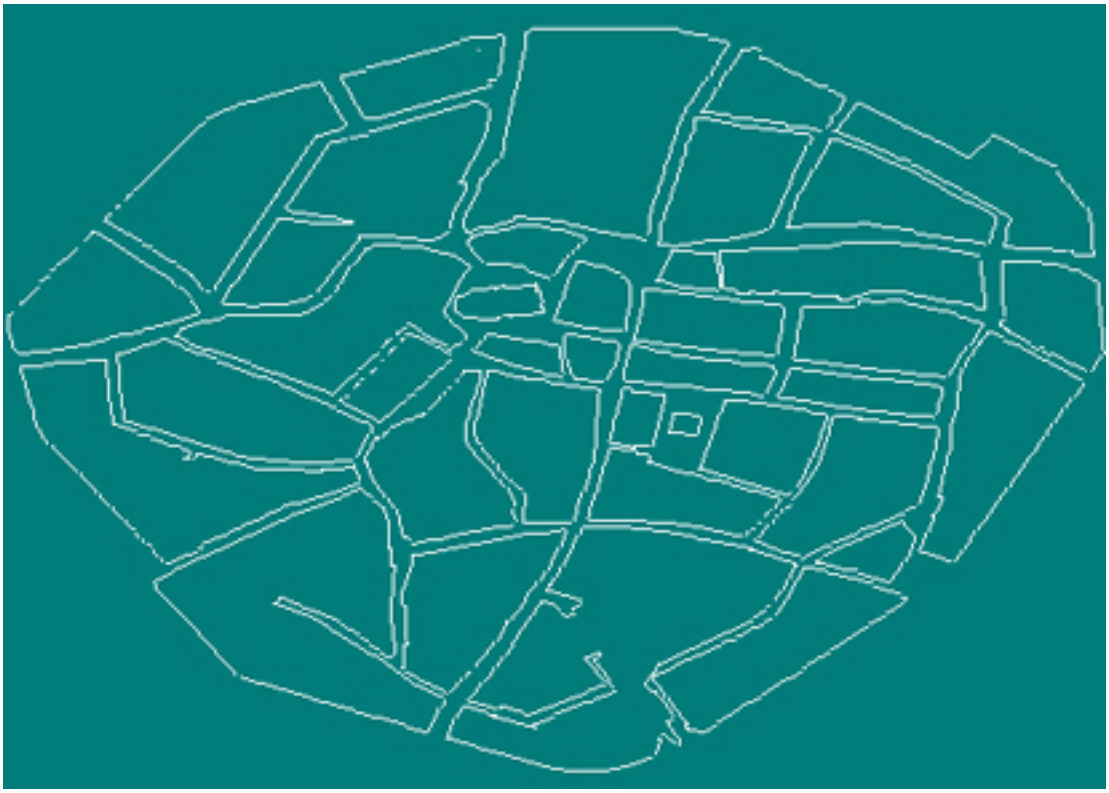
hebben. Hierdoor blijft enkel nog het renderen van de zichtbare muren over. Eens het programma start zijn geen verdere berekeningen voor zichtbaarheid meer nodig, deze zouden enkel de rendering vertragen. Merk op dat OpenGL zelf nog wel clipping en hidden surface removal toepast.

Data van de stad

De gegevens die we voor deze stad hebben gekregen zijn vrij beperkt. We hebben enkel informatie waar een muur begint en eindigt. Verdere informatie, zoals de hoogte van muren, samenhorende muren of bijbehorende textures, is niet aanwezig.

Aanpak en algoritme

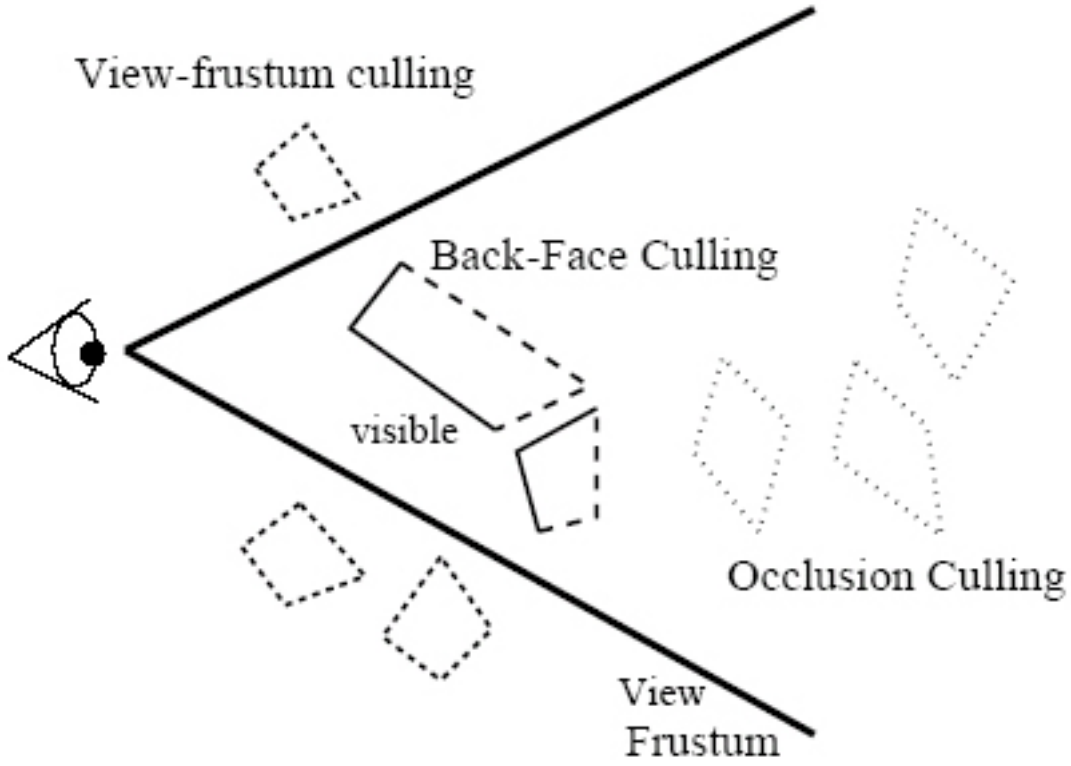
De gegevens van de stad worden ingelezen en het grondplan is zichtbaar in figuur 6.1. Als we ons in de stad bevinden, hoe kunnen we dan bepalen wat



Figuur 6.1: Een bovenaanzicht van de ingelezen stad (90 graden geroteerd)

zichtbaar is voor ons en wat niet [40][44][50][56]? Hiervoor hebben we een

visibility culling algoritme nodig. Er bestaan drie types visibility culling: view frustum culling, back-face culling en occlusion culling [44]. Deze zijn zichtbaar in figuur 6.2 [44].

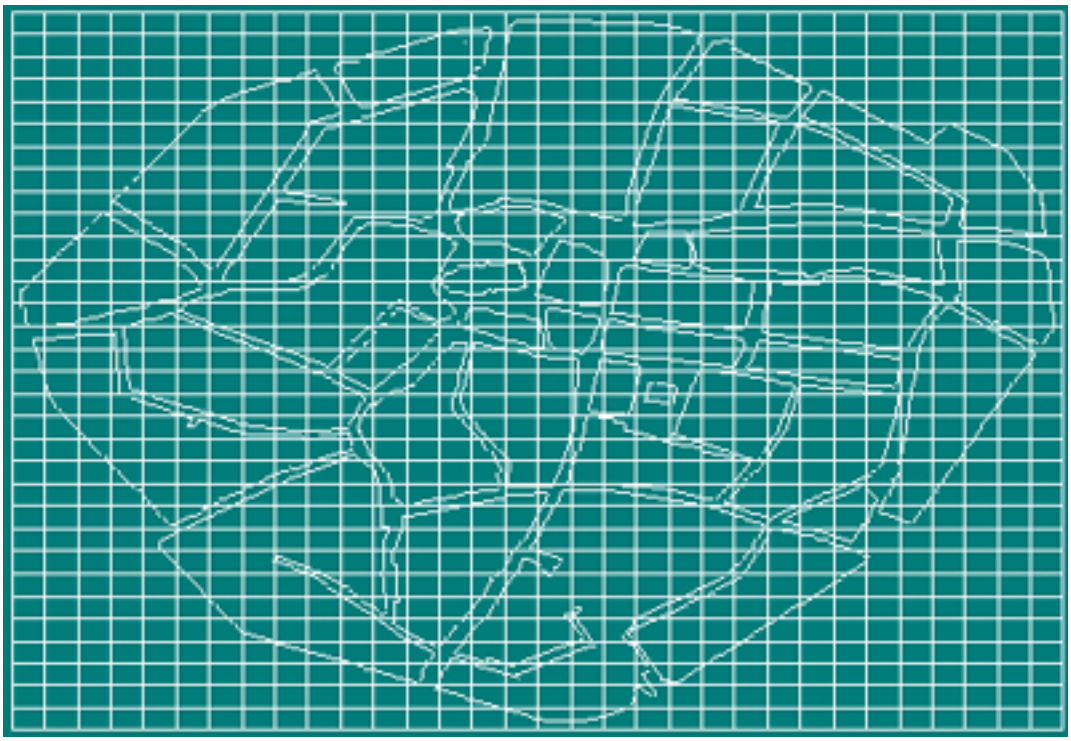


Figuur 6.2: De drie types visibility culling [44].

De ideale techniek is een techniek die enkel de zichtbare muren naar de rendering pipeline stuurt die voor de huidige viewing cone zichtbaar zijn. Onder "zichtbare muren" verstaan we enkel muren die vanuit het huidige camerastandpunt zichtbaar zijn, dus na toepassing van view-frustum culling, back-face culling en occlusion culling. Een voorbeeld is figuur 6.2: enkel de volle lijnen worden naar de rendering pipeline gestuurd. Omdat voor elke positie en oriëntatie van de viewing cone bepaald wordt wat zichtbaar is, wordt deze techniek from-point visibility genoemd. Doordat enkel en alleen de zichtbare muren bepaald worden is dit een zogenaamd "exact algoritme". Een eerste naïve methode om visibility culling te implementeren is real-time raycasting. Hierbij schieten we van links naar rechts in onze viewing cone rays uit voor elke mogelijk zichtbare pixel en we berekenen welke muren deze snijden. Deze muren worden dan uitgetekend. Deze methode vergt enorm

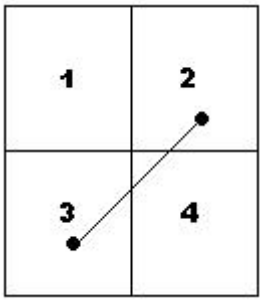
veel berekeningen in real-time en dit willen we ten aller tijde vermijden. Een betere techniek is misschien een ray uiterst links casten, berekenen welke muur deze raakt en vervolgens juist rechts van het einde van deze muur een nieuwe ray casten, enzoverder. Hierbij zijn al veel minder berekeningen nodig, maar toch blijft deze techniek achter omdat er berekeningen gemaakt moeten worden tijdens het renderen. Ook kunnen door deze techniek muren gemist worden. Het is dus zeer moeilijk om snel voor elke mogelijke positie en richting van de viewing cone te bepalen welke muren zichtbaar zijn. Deze ideale (exact algoritme, from-point visibility) oplossing is dus te streng.

Een oplossing voor dit probleem is de stad opdelen in cellen. De strenge eis van enkel de zichtbare muren naar de pipeline te sturen laten we vallen. Dit wordt, in tegenstelling tot een exact algoritme, een conservatief algoritme genoemd: alle zichtbare muren worden zeker naar de pipeline gestuurd en misschien ook nog enkele onzichtbare. Elke cell bevat een deel van de gegevens van de stad, namelijk alle muren die binnen deze cell vallen. Figuur 6.3 toont ons de stad met een grid erover. Ook alle muren die met slechts



Figuur 6.3: Een bovenaanzicht van de ingelezen stad (90 graden geroteerd) met het grid erover

één uiteinde in een cell vallen, horen bij de cell. We moeten wel opletten dat muren waarvan het begin en einde niet binnen dezelfde cell ligt, in alle bijhorende cellen aanwezig zijn. Dit probleem is zichtbaar in figuur 6.4. We zien



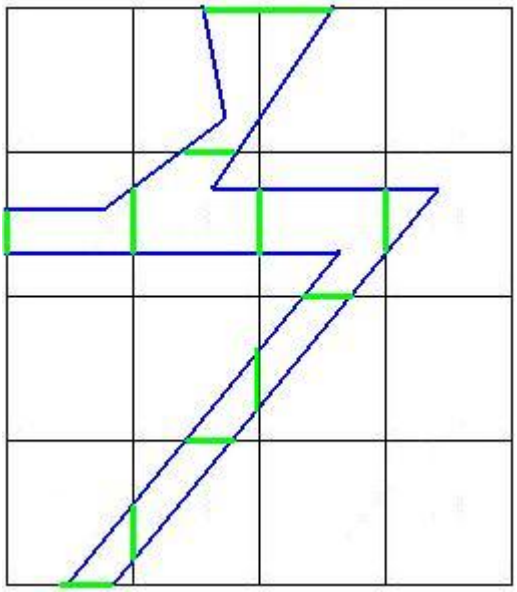
Figuur 6.4: Een mogelijk probleem: muren missen

op deze figuur een muur die begint in cell 2, door cell 4 gaat en vervolgens eindigt in cell 3. We moeten ervoor zorgen dat deze muur zeker aanwezig is in zowel cell 2, 3 als 4.

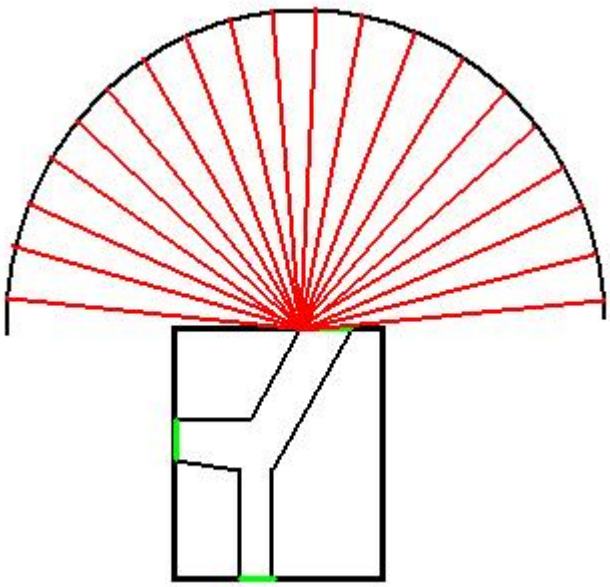
Waar we nu ook staan binnen de stad, we staan altijd binnen een cell. De muren van de cell waarin we staan, worden als zichtbaar beschouwd en zullen naar de rendering pipeline gestuurd worden. Als we de cellen klein genoeg nemen, worden er niet veel onzichtbare muren meegestuurd. Verder gaan we niet bepalen wat zichtbaar is in onze viewing cone, maar wel wat zichtbaar is vanuit onze cell. Deze techniek heet from-cell visibility. Merk op dat dit allemaal mogelijk is in preprocessing: de stad in cellen opdelen en voor elke cell bepalen wat er uit deze cell zichtbaar is.

Doordat onze stad bestaat uit blokken van gebouwen en elk gebouw in feite een aaneensluitende veelhoek vormt, kunnen we de zichtbare delen van de ene cell naar een andere cell toe bepalen. Beschouw figuur 6.5.

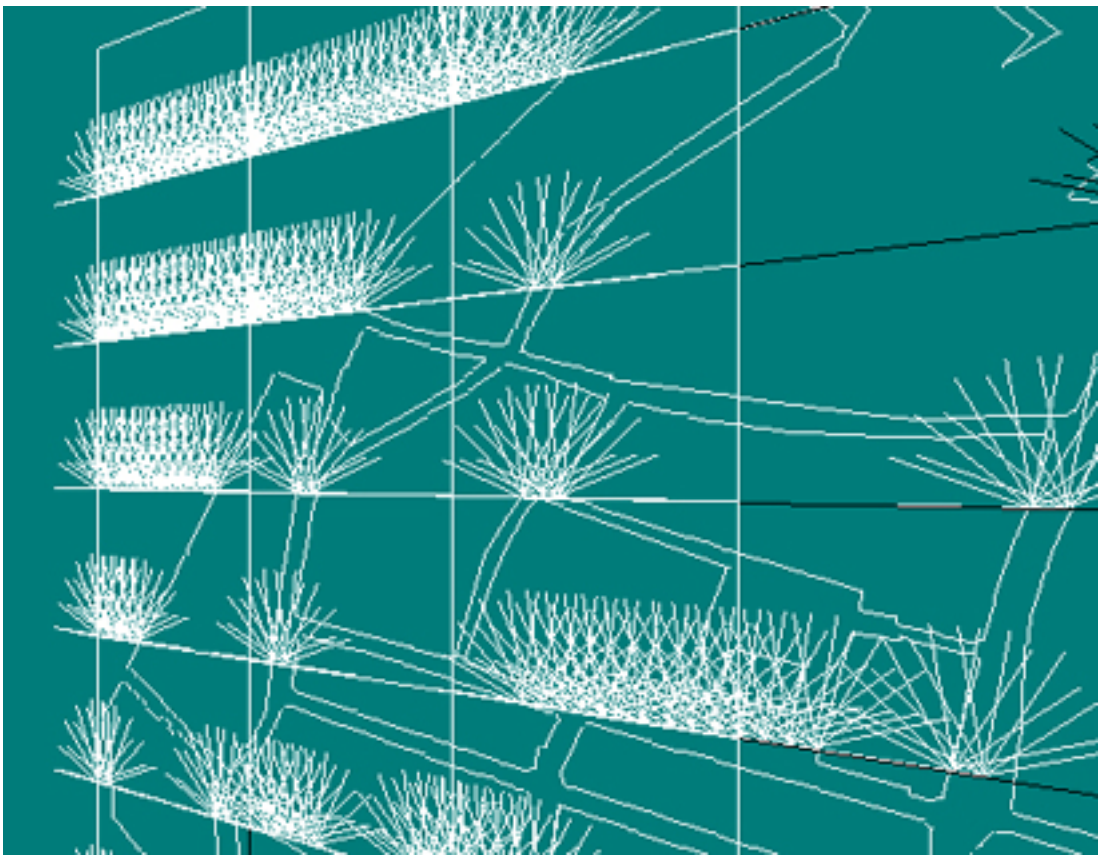
Enkel de groene delen van de cellranden zijn nuttig om te beschouwen. De overige cellranden bevinden zich immers binnen een gebouw of blok.
Hoe wordt nu deze from-cell visibility bepaald? Beschouw figuur 6.6. We weten welke delen van de cell zichtbaar zijn naar buiten toe, namelijk de groene randen. We gaan nu rays schieten in een hoek van 180 graden vanuit punten op de groene lijnstukken en dit voor verschillende punten op gelijke afstand van elkaar. Dit wordt geïllustreerd in figuur 6.7. Hierop zien we de rondgeschoten rays naar de rechterkant. Deze zijn ter illustratie verkort en er zijn voor elk punt slechts enkele rays uitgeschoten. Als we de afstand tussen de punten waaruit de rays geschoten worden klein nemen en het increment



Figuur 6.5: Zichtbaarheid van cell tot cell



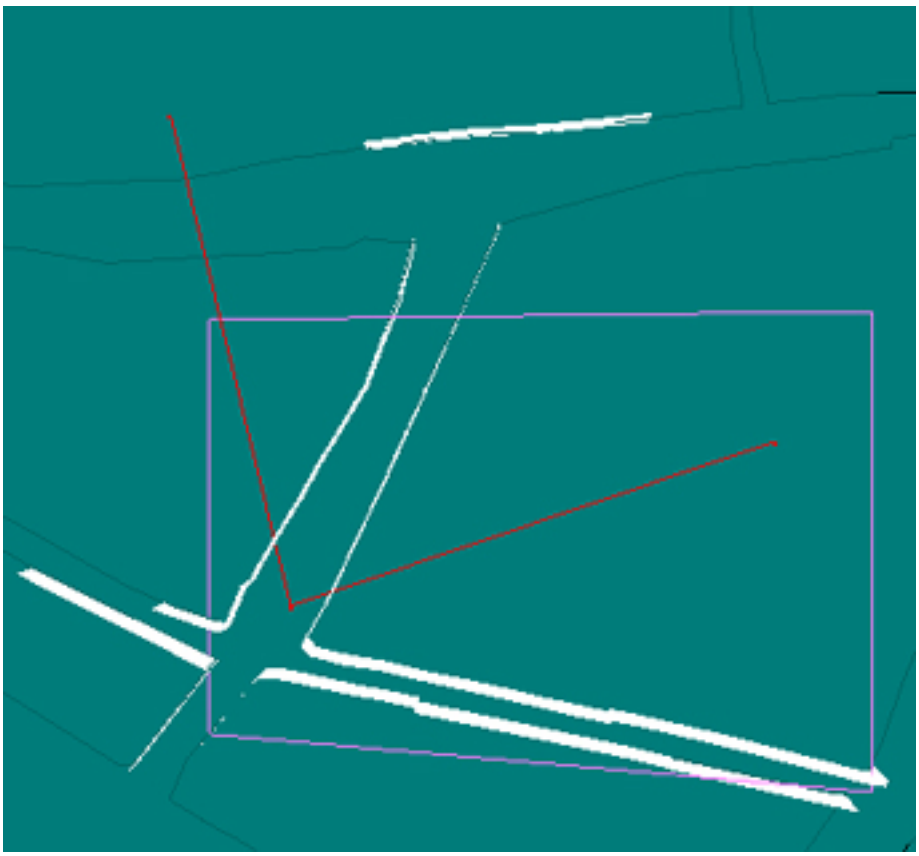
Figuur 6.6: Rays worden gecast om te zien wat er zichtbaar is vanuit deze cell



Figuur 6.7: Screenshot van het programma met verkorte en minder rays. (figuur is 90 graden geroteerd)

voor de hoek van de rays ook, dan worden er geen muren gemist.

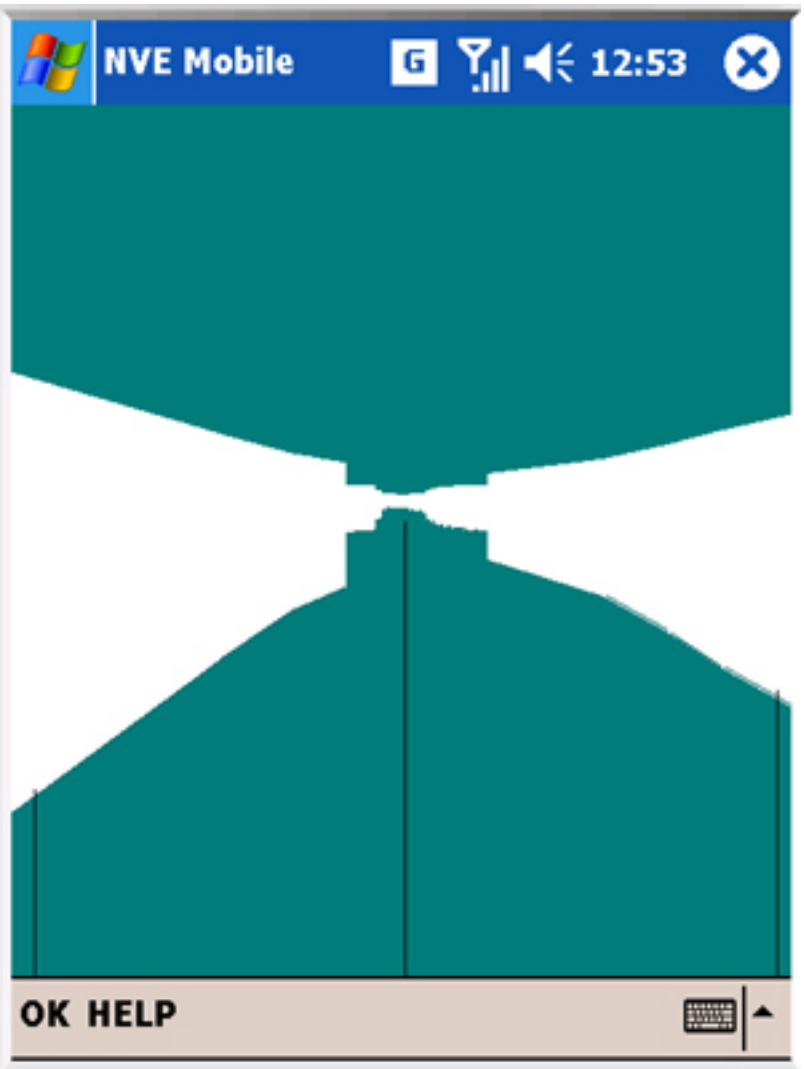
Elke cell bevat dus alle muren die zich binnen de cell bevinden en alle muren die mogelijk zichtbaar zijn vanuit deze cell. Stel nu dat we in een cell staan en onze viewing cone snijdt enkel de bovenste rand van de cell, dan moeten we de muren van de cell zelf tekenen en alle muren zichtbaar vanuit de bovenste cellwand ook. Zie figuur 6.8.



Figuur 6.8: Voorbeeld van het algoritme in een grote cell

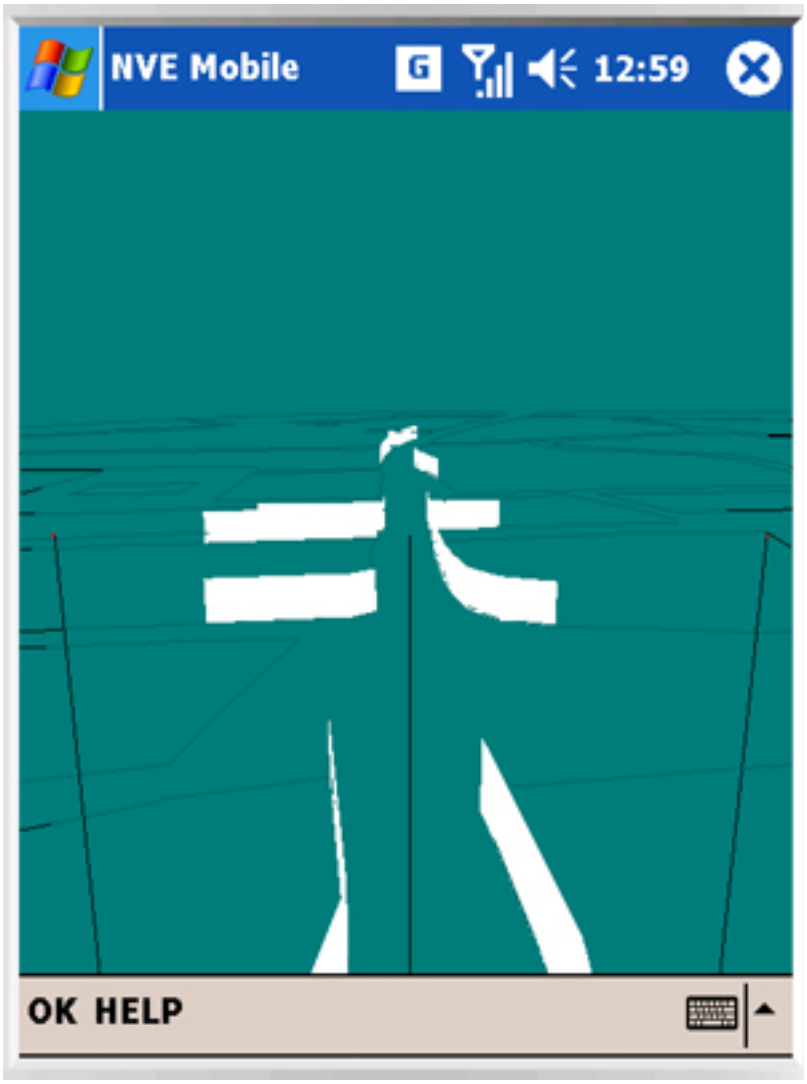
Zoals zichtbaar in figuur 6.8 zullen ook niet-zichtbare muren naar de rendering pipeline gestuurd worden. Het is nu aan OpenGL ES om clipping en hidden surface removal uit te voeren. Hoe kleiner de cellen, hoe nauwkeuriger het algoritme zal zijn. Een voorbeeld van de werking van het algoritme is zichtbaar op figuur 6.9 en 6.10. In figuur 6.9 zien we de scène in first-person view en in figuur 6.10 in third-person view. Hierop zien we duidelijk dat enkel de muren die zichtbaar zijn voor ons, uitgetekend worden. De rest van

de stad is voor dit voorbeeld in grijze lijnen uitgetekend.

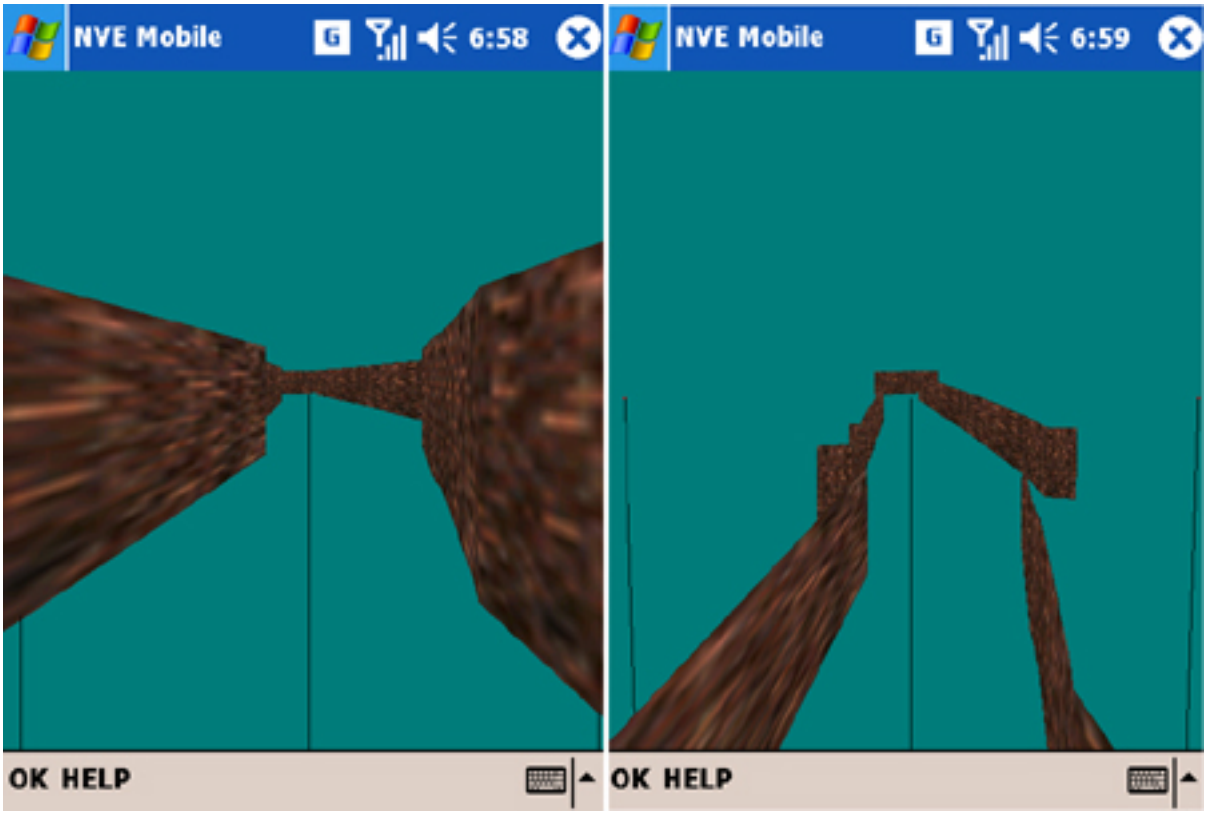


Figuur 6.9: Werking van het algoritme: first-person view

Het algoritme is verder uitgebreid met real-time dynamische texture mapping. Er wordt tijdens de run van de applicatie voor elke nieuwe frame bepaald hoeveel nieuwe textures ingeladen moeten worden ten opzichte van de vorige frame. Doordat bij onze testdata textures ontbreken en de referentie tussen texture en muur ook, laden we continu dezelfde texture in om een echte applicatie te simuleren. Zo kunnen we realistische performantie tests uitvoeren. Een screenshot van de applicatie die gebruik maakt van textures kan gezien worden in figuur 6.11.

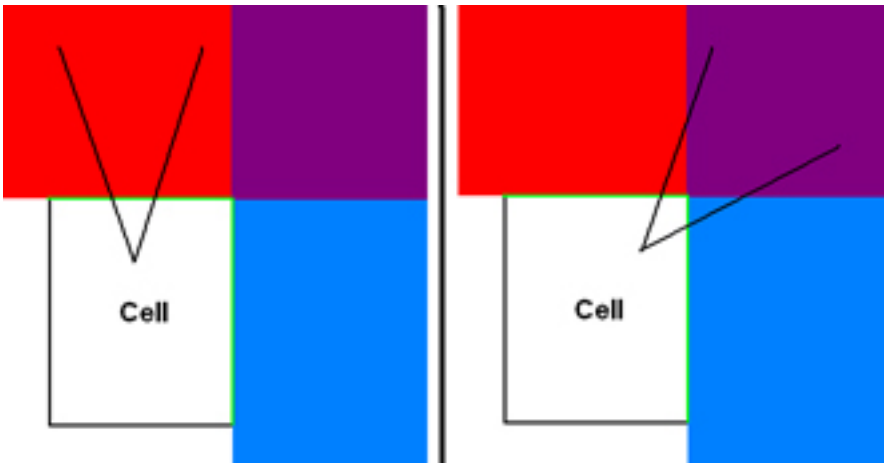


Figuur 6.10: Werking van het algoritme: third-person view



Figuur 6.11: Werking van het algoritme met textures.

We zullen deze techniek even toelichten. Beschouw figuur 6.12. We zien



Figuur 6.12: Toelichting van dynamische texturing

in deze figuur een cell. Stel dat van deze cell de volledige boven- en rechterrand zichtbaar zijn naar buiten toe (dit zijn de groene randen). De bovenste rand van de cell "ziet" dan het rode en paarse gebied. De rechter rand "ziet" het blauwe en het paarse gebied. Deze twee randen hebben het paarse gebied gemeen en dus ook de textures en muren die binnen dit gebied liggen.
Als we kijken naar de linkerkant van figuur 6.12. We kijken naar boven en zien op dat moment volgens het algoritme enkel wat er binnen de cell zelf ligt en alles wat binnen het rode en paarse gebied ligt. De textures van de muren in de cell en binnen het rode en paarse gebied worden ingeladen. Als de gebruiker zich nu naar rechts draait, zien we ook het blauwe gebied. Het paarse gebied zien we nu vanuit de bovenste cellrand en vanuit de rechter cellwand. Enkel nieuwe textures moeten nu ingeladen worden. Dit zijn de textures gelegen in het blauwe gebied.

Merk op dat deze techniek zich goed leent tot draadloze (navigatie)systemen: wanneer we een stad binnenrijden wordt bepaald in welke cell we ons bevinden. Alle informatie van deze cell kan dan naar de PDA gezonden worden. We weten ook welke muren zichtbaar zijn vanuit deze cell en dus ook welke cellen zichtbaar zijn vanuit de huidige cell. Deze cellen kunnen vervolgens al naar de PDA gestuurd worden omdat de gebruiker enkel naar een van deze cellen kan gaan. Zo kunnen de cellen efficiënt naar de PDA gestuurd worden en kan de stad altijd correct uitgetekend worden.

6.1.3 Resultaten

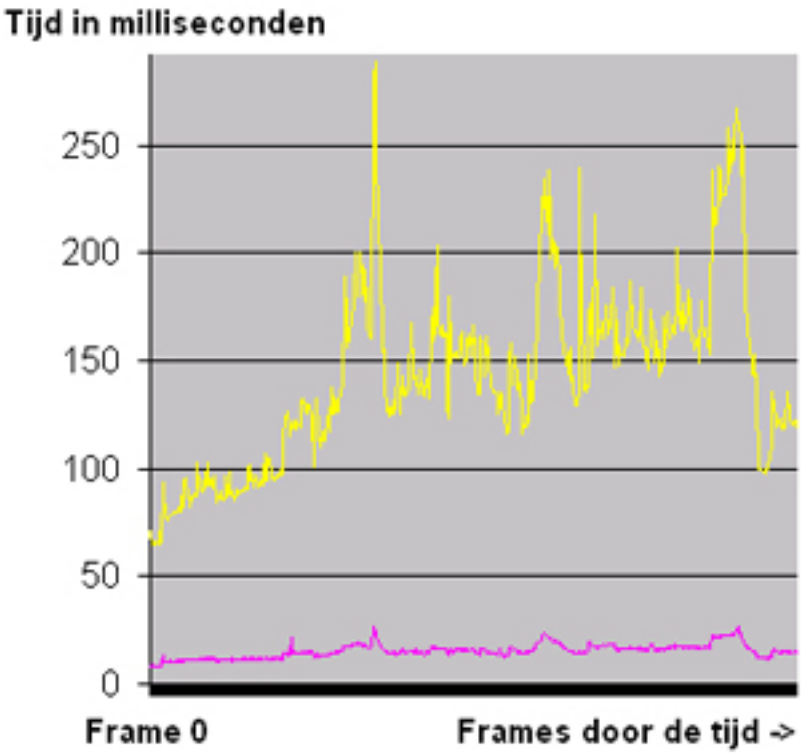
We zullen nu eens kijken hoe goed het algoritme werkt. Als test is er een pad door de stad bepaald dat automatisch doorlopen zal worden op verschillende toestellen. De resultaten van deze tests worden vervolgens in grafieken weergegeven. We hebben drie verschillende testplatformen:

- **Emulator**: Dit is een Pocket PC platform geëmuleerd door visual studio 2005 zelf. Doordat het slechts een emulatie is zijn exacte gegevens over de snelheid niet beschikbaar. De emulator bevat geen hardware acceleratie en het scherm heeft een resolutie van 640x480.
- **Typhoon Myguide**:
  - Platform: Pocket PC 2003
  - Processor: Intel XScale PXA255 processor - 300MHz
  - Resolutie: 320x240
  - Grafische acceleratie: neen
- **Dell Axim x51v**:
  - Platform: Windows Mobile 5.0
  - Processor: Intel XScale PXA270 processor - 624MHz
  - Resolutie: 640x480
  - Grafische acceleratie: Intel 2700G multimedia accelerator met 16MB video geheugen

Voor de tests zal de emulator van visual studio niet gebruikt worden. Deze is veel te traag en deze zou de resultaten van de overige toestellen onderdrukken op een grafiek. Dit is zichtbaar in figuur 6.13. De gele lijn is die van de emulator, de paarse die van de Typhoon. Doordat de gele lijn zo sterk varieert wordt een veel belangrijker resultaat – namelijk dat van een echt toestel – onderdrukt. Om deze reden worden in de volgende grafieken de gegevens van de emulator weggelaten.

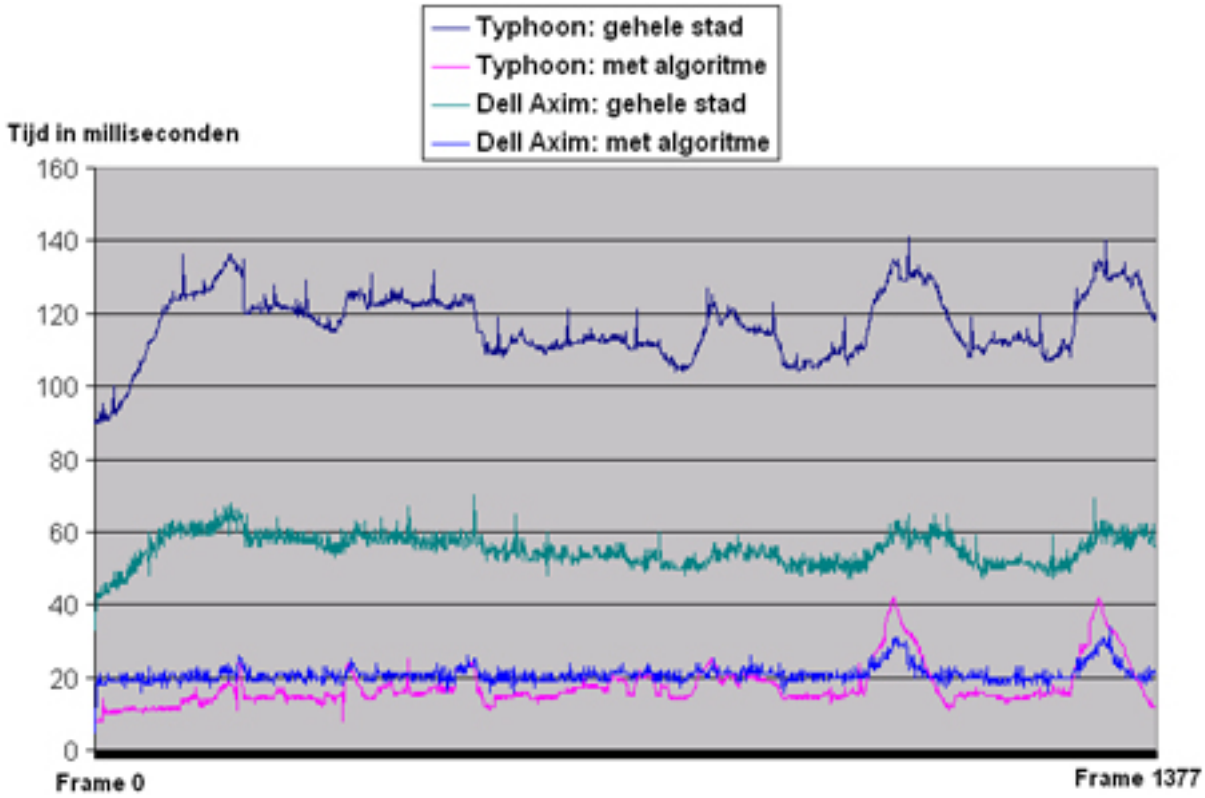
Test 1: Performantie zonder texturing

Onze eerste test voert de automatische walkthrough uit zonder real-time dynamische texturing. Voor elk van de twee toestellen wordt de walkthough uitgevoerd met en zonder het geïmplementeerde algoritme. Wanneer het algoritme niet gebruikt wordt, worden alle muren van de stad naar de rendering



Figuur 6.13: Een stukje uit de grafiek van de automatische walkthrough. Resultaten van de Emulator (geel) en de Typhoon (paars).

pipeline gestuurd. De grafiek met de resultaten van deze tests is te zien in figuur 6.14. Horizontaal op de grafiek zien we de opeenvolgende frames en verticaal de tijd in milliseconden nodig om elke frame uit te tekenen. Beschouw figuur 6.14.



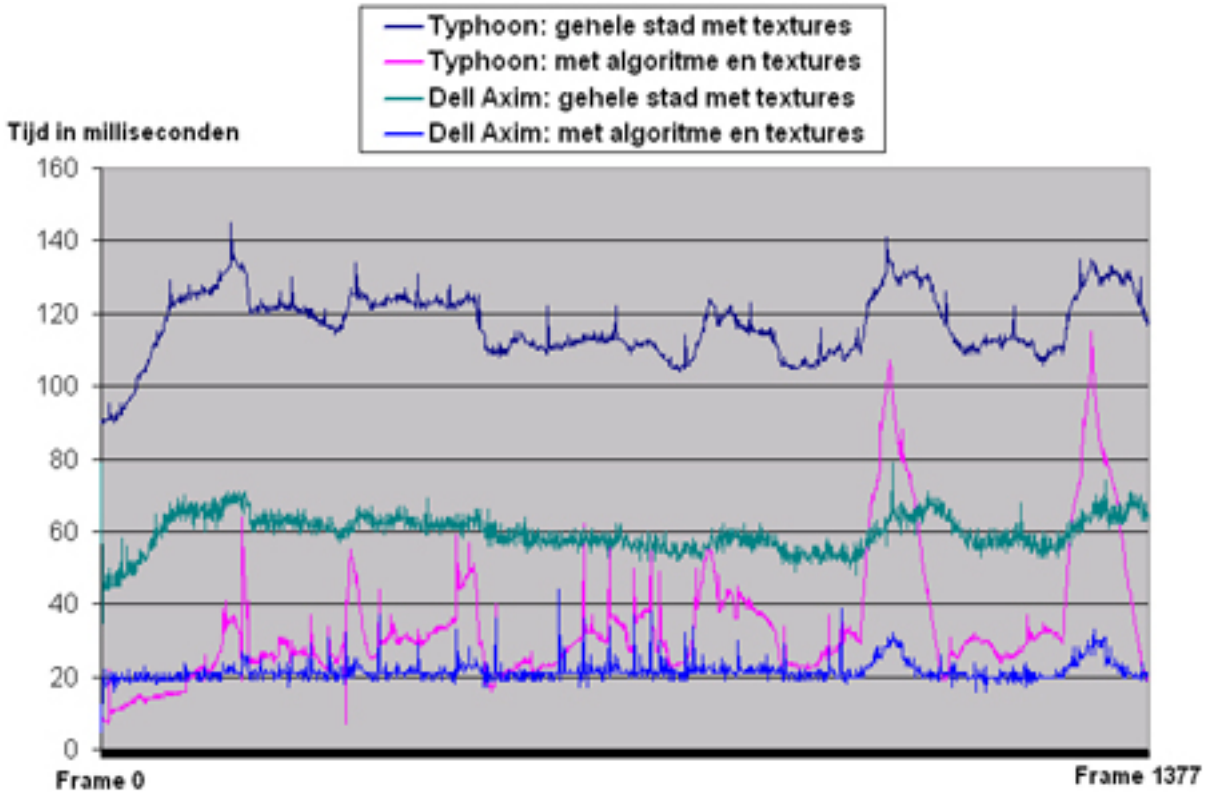
Figuur 6.14: De testresultaten. Test 1: met en zonder algoritme, zonder textures.

De resultaten in deze grafiek zijn niet onverwacht: het algoritme geeft een enorme snelheidswinst. Hoeveel beter is dit algoritme nu precies?

Op het eerste zicht kunnen we zeggen dat op de Typhoon het algoritme elke frame ongeveer 100 milliseconden sneller rendert dan zonder algoritme, en op de Dell Axim ongeveer 40 milliseconden. 0.1 en zelfs 0.04 seconden langer per frame zal een serieuze impact hebben op het aantal beelden per seconde. Als we het gemiddelde nemen van het aantal beelden per seconde dan rendert het algoritme er 58 per seconde op de Typhoon en 51 op de Dell Axim. Zonder algoritme is dit amper 8 beelden per seconde op de Typhoon en 17 op de Dell Axim. We kunnen dus spreken van een gemiddelde zevenmalige versnelling op de Typhoon, en een driemalige versnelling op de Dell Axim. We zien dat de Typhoon gemiddeld meer frames per seconde haalt dan de Dell Axim hoewel deze laatste toch krachtiger is. Dit komt doordat de Dell Axim beelden moet renderen met een resolutie van 640x480. De Typhoon rendert beelden met een resolutie van 320x240. De Dell Axim moet dus veel meer objectpixels bepalen.
Zoals we kunnen zien zijn de grafieken geen mooie horizontale lijnen. Dit is vooral merkbaar bij de Typhoon. Wat zijn deze pieken precies? Deze pieken van langere rendertijd komen voor wanneer we ons dicht bij een muur bevinden of wanneer er vele muren zichtbaar zijn. Hierdoor moeten veel meer objectpixels bepaald en uitgetekend worden. Dit vertraagt de rendering. Objectpixels bepalen is een typische operatie die uitgevoerd wordt door een grafische kaart. We zien dat deze pieken veel minder uitgesproken zijn bij de Dell Axim en dat deze een veel constantere grafiek geeft. De Dell Axim heeft immers een grafische kaart. Merk op dat het gebruik van een visibility algoritme bij toestellen zonder grafische kaart een grotere snelheidswinst geeft dan bij toestellen met een grafische kaart.

Test 2: Performantie met real-time dynamische texturing

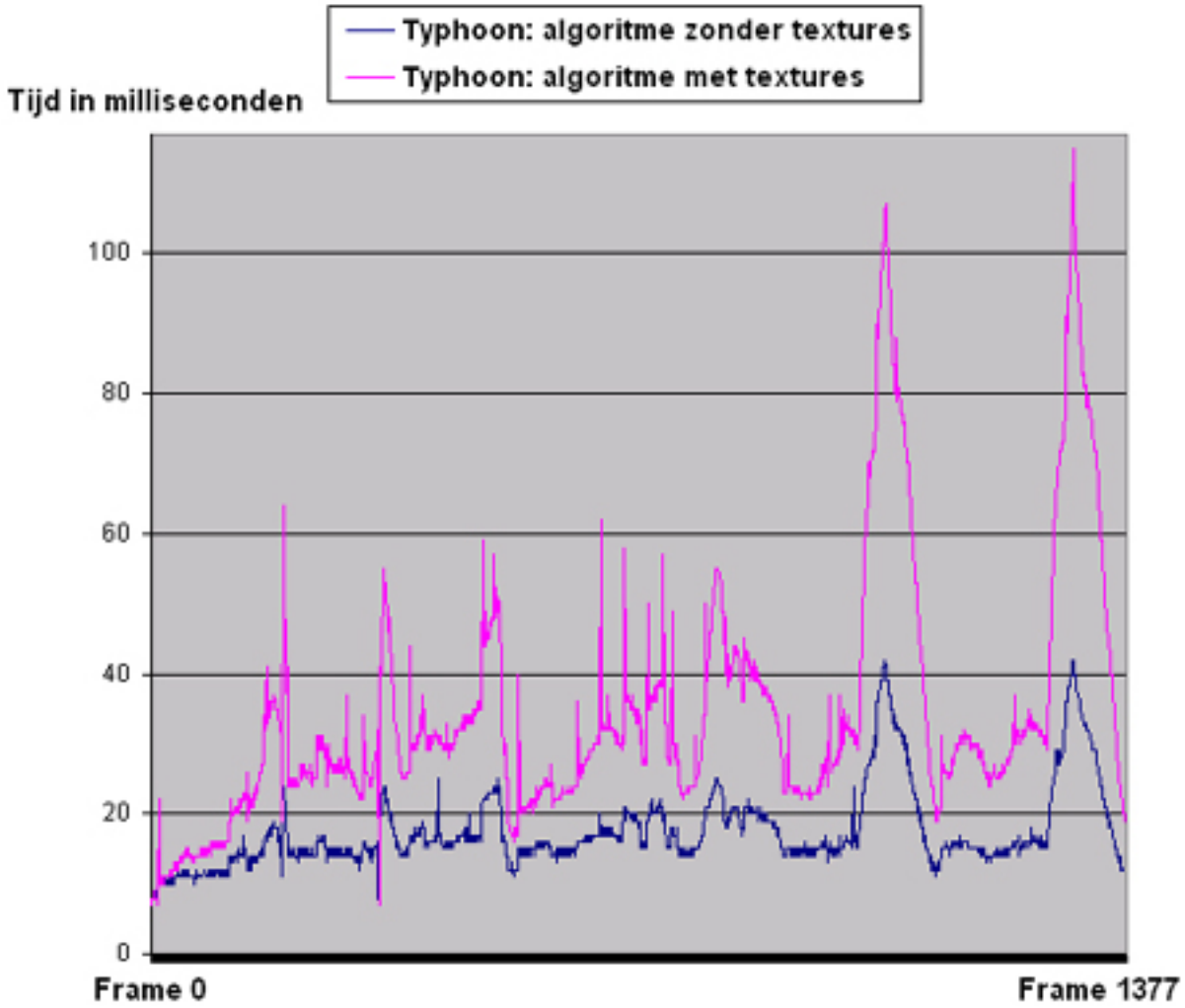
De tweede test maakt gebruik van texturing op de muren. De gebruikte texture heeft een resolutie van 32x32 en is 4kB groot. Net zoals bij de eerste test gaan we op beide toestellen de walkthrough uitvoeren met en zonder algoritme. Merk op dat er geen real-time dynamisch inladen van textures nodig is wanneer we de volledige stad uittekenen. Vermits de PDA niet genoeg geheugen heeft om voor elke muur een verschillende texture in te laden, gebruiken we slechts één texture voor onze tests. Door dit geheugen tekort wordt het al snel duidelijk dat er altijd een visibility algoritme nodig is. Doordat we maar één texture inladen wordt de rendering pipeline niet extra belast en zal de performantie met of zonder textures waarschijnlijk niet veel veranderen. De resultaten van de tests met texturing zijn zichtbaar in figuur 6.15.



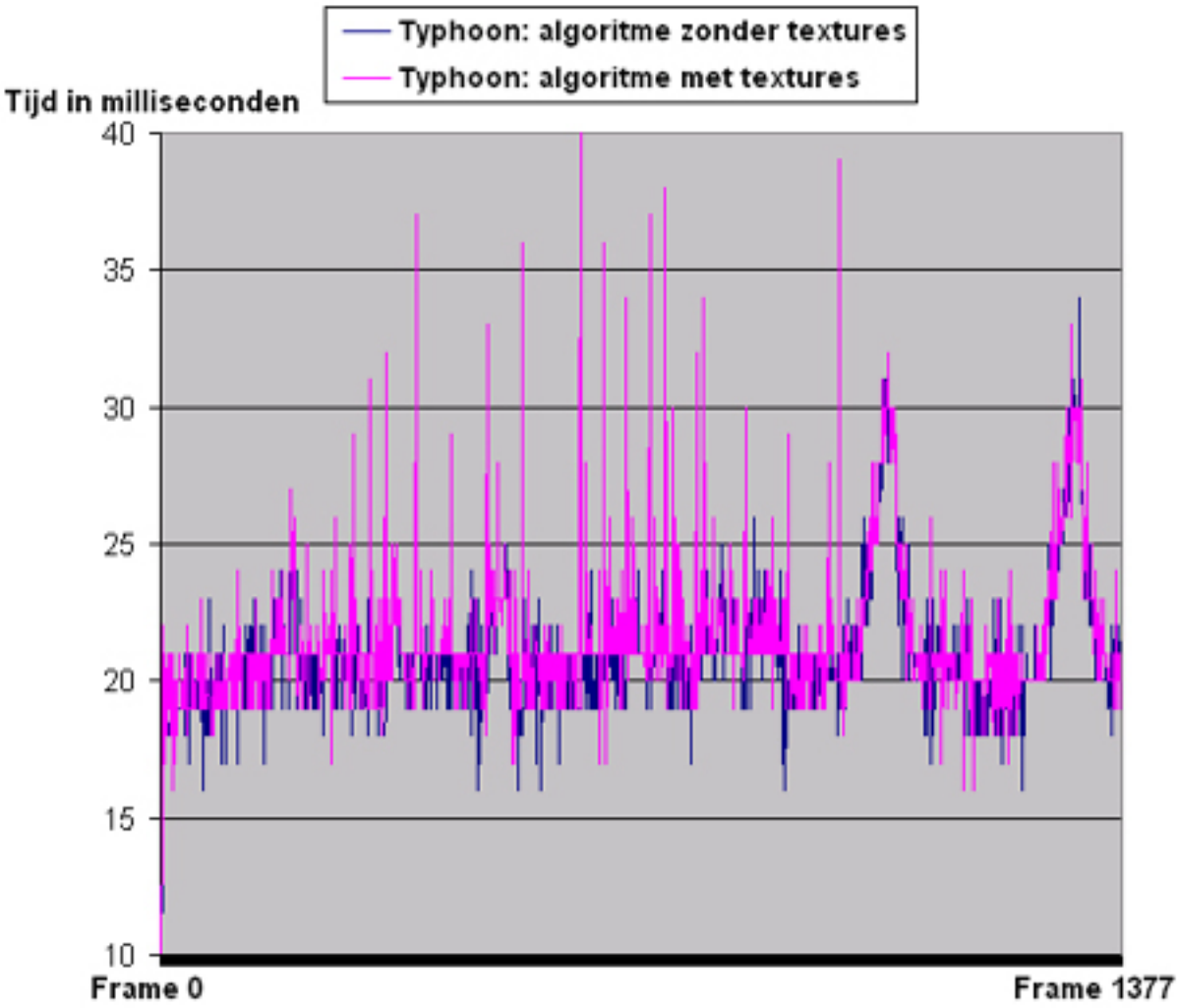
Figuur 6.15: De testresultaten. Test 2: met en zonder algoritme, met real-time dynamische texturing.

Ook hier zijn de resultaten verwacht en gunstig. Wanneer we de stad volledig texturen met één texture en uittekenen, verkrijgen we dezelfde resultaten als in de eerste test. Zoals reeds vermeld is, is dit een normaal resultaat. We beschouwen nu eerst even de resultaten van de Typhoon. Als we kijken naar het algoritme met de real-time dynamische texturing dan zien we een duidelijke vertraging in de rendering ten opzichte van de vorige test. Verder zien we exact dezelfde pieken, maar nu zijn ze meer uitgesproken. Wederom is dit een verwacht resultaat. Behalve de beelden gewoon renderen moet de Typhoon ook bepalen welke textures nog ingeladen moeten worden. Hoe meer muren we zien, hoe meer vergelijkingen we zullen moeten maken voor nieuwe textures te bepalen, hoe trager de frame gerendered kan worden. Uiteraard geldt ook dat hoe meer muren we zien, hoe trager de frame rendert. Nieuwe textures inladen en meesturen naar de rendering pipeline vertraagd de rendering dus duidelijk. Het gemiddeld aantal frames per seconde bedraagt voor het algoritme nu 30 in plaats van 57 op de Typhoon. Als we kijken naar de resultaten van de Dell Axim dan zien we over het algemeen geen sterke vertraging in de rendering. We hebben wel pieken verkregen in de grafiek. Ook deze pieken zijn verklaarbaar door het feit dat nieuwe textures in real-time nog bepaald moeten worden. Het meesturen van de textures lijkt de rendering niet verder te beïnvloeden. Merk ook op hoe klein de pieken bij de Dell Axim zijn ten opzicht van de Typhoon. Dit enorme verschil kan wederom verklaard worden door de grafische kaart aanwezig in de Dell Axim. Object- en texturepixels bepalen is nu eenmaal veel sneller met een grafische kaart. Om het verschil tussen het algoritme met of zonder texturing te verduidelijken zijn de gegevens van beide testen in figuur 6.16 (Typhoon) en 6.17 (Dell Axim) samengevoegd.

We kunnen concluderen dat, door het gebruik van het algoritme, we een enorme snelheidswinst bekomen, zowel met als zonder grafische acceleratie. Het gebruik van een zichtbaarheidsalgoritme is niet alleen sneller, het is noodzakelijk. Het zou onmogelijk zijn om alle textures van de stad gelijktijdig in te laden. Niet alleen is hier te weinig geheugen voor, ook zou de rendering pipeline overbelast zijn, wat tot slechte performantie leidt. Een test is uitgevoerd naar hoeveel textures ingeladen kunnen worden vooraleer het geheugen vol is en het programma crasht. Met textures van slechts 4kB, bestaande uit 32x32 pixels, kunnen op de Typhoon 6046 textures ingeladen worden. Op de Dell Axim zijn dit er 6536. De Dell Axim kan er meer inladen doordat er nog 16MB geheugen beschikbaar is op de grafische kaart. Merk op dat deze textures enorm klein zijn en geen goede kwaliteit leveren.



Figuur 6.16: Vergelijking tussen met of zonder texturing op de Typhoon: een duidelijke vertraging.



Figuur 6.17: Vergelijking tussen met of zonder texturing op de Dell Axim: pieken bij het bepalen van niewe textures.

Een stad bestaat al gemakkelijk uit meer dan 4000 muren. Voor elke muur een goede kwaliteitstexture inladen is dus uit den boze. Een laatste opmerking is dat voor de Dell Axim alle object- en texturegegevens naar fixed-point omgezet moeten worden omdat de grafische kaart enkel fixed-point berekeningen kan uitvoeren. Ook dit draagt bij tot een snelheidswinst.

6.2 Testimplementaties

In deze sectie worden enkele testimplementaties kort toegelicht. Deze implementaties zijn gemaakt om verschillende SDK's uit te proberen en om ervaring op te doen met OpenGL ES. De uitgeprobeerde SDK's zijn: Vincent 3D library, Hybrid Rasteroid en BREW SDK (zie sectie 4.2). Deze SDK's bieden ons de mogelijkheid om applicaties te schrijven voor mobiele toestellen. Ze bieden een interface naar de onderliggende besturingssystemen en zorgen ervoor dat alle nodige componenten correct samenwerken.

6.2.1 Vincent 3D library en Hybrid Rasteroid

We bespreken deze SDK's in dezelfde paragraaf omdat eens alle initialisatie gedaan is, er bijna geen verschil meer is. Ook kunnen we nu gemakkelijk deze twee SDK's rechtstreeks vergelijken [25].
De Vincent library is een Opengl ES implementatie die zowel niet commercieel als commercieel gebruikt mag worden door iedereen. Het grote voordeel aan Vincent is dat de library standaard allerlei initialisatie en management functies aanbiedt (zoals bijvoorbeeld window management) voor een snelle setup. Deze functies gelijken sterk op de functies van GLUT. De Hybrid Rasteroid implementatie is een library die enkel gratis mag gebruikt worden voor niet commerciële doeleinden. Licenties moeten betaald worden als men deze library commercieel wil gaan gebruiken. De voordelen bij Hybrid zijn hogere framerates en meer controle over bepaalde functionaliteit zoals bijvoorbeeld de depth buffer.
Om applicaties te kunnen ontwikkelen voor deze toestellen zijn naast de Vincent of Hybrid SDK de volgende pakketten nodig:

- Embedded Visual C++ 4.0
- Embedded Visual C++ 4.0 Service Pack 4
- Pocket PC 2003 SDK

Of we kunnen de nieuwe Microsoft Visual Studio 2005 gebruiken dat een "all-in-one" pakket vormt.

We zullen nu even illustreren hoe gemakkelijk het is om rechtstreeks van start te gaan met de Vincent library. In onderstaande code zien we de standaard layout van een basisprogramma.

```
void init() {
glClearColor (0.0f, 0.0f, 0.0f, 0.0f);
}
void display(UGWindow uwin) {
   glClear (GL_COLOR_BUFFER_BIT);
   glFlush ();
   ugSwapBuffers(uwin);
}
void keyboard(UGWindow uwin, int key, int x, int y) {
}
void pointer(UGWindow uwin, int button, int state, int x, int y) {
}
int main() {
UGCtx ug = ugInit();
UGWindow uwin = ugCreateWindow(ug, "", "Vincent test", 250, 250, 100, 100);
init();
ugDisplayFunc(uwin, display);
ugKeyboardFunc(uwin, keyboard);
ugPointerFunc(uwin, pointer);
ugMainLoop(ug);
return 0;
}
```

Iemand die ooit met GLUT gewerkt heeft zal dit onmiddellijk herkennen en het gemak ervan inzien. Er is ondertussen ook een GLUT ontwikkeld voor OpenGL ES, genaamd GLUT ES. Deze is bijna volledig hetzelfde als de Vincent library. We zullen dit even illustreren.

```
void init() {
glClearColor (0.0f, 0.0f, 0.0f, 0.0f);
}
void display() {
   glClear (GL_COLOR_BUFFER_BIT);
   glFlush ();
   glutSwapBuffers();
}
void keyboard(int key, int x, int y) {
}
```

```
void menu(int entry) {
}
int main(int argc, char *argv[]) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
glutCreateWindow("GLUT ES test");
init();
glutDisplayFunc(display);
glutSpecialFunc(keyboard);
glutMainLoop();
return 0;
}
```

Of we nu Vincent of GLUT ES gebruiken, de setup is snel en we kunnen onmiddellijk van start gaan. We kunnen vanaf hier rechtstreeks beginnen programmeren in OpenGL ES. Dit lijkt haast hetzelfde als voor desktop PC's behalve dat we rekening moeten houden met de beperkingen in OpenGL ES. Hybrid breidt egl.h uit om alle taken uit te voeren die nodig zijn om beelden te renderen voor een specifiek mobiel toestel. Er zijn geen hulpfuncties voor windowing of keyboard/pointer input aanwezig bij hybrid, maar eens deze initialisatie gedaan is kunnen we verder met het programmeren in OpenGL ES. Er is een testimplementatie gemaakt voor beide SDK's en een screenshot van deze test is zichtbaar in figuur 6.18.

Deze testimplementatie bevat initialisatie, skyboxes, texturing, navigatie en het inladen van 3ds modellen. Deze testimplementaties zijn uitgevoerd op de Pocket PC SDK emulatoren.

6.2.2 BREW

In tegenstelling tot de vorige twee SDK's is BREW veel complexer en moet er op een veel lager niveau gewerkt worden [3]. Bij zowel Vincent als Hybrid werd het gebruik van fixed point getallen en stack of heap space volledig afgeschermd van de programmeur. Bij BREW zien we veel duidelijker dat er een groot verschil is tussen programmeren voor een desktop PC of een mobiel toestel.
Om te beginnen hebben we al geen static of global data beschikbaar. Daarbij is BREW volledig event driven. Dit is in tegenstelling tot het "normale" programmeren waar we gemakkelijk alles in loops (while/for) kunnen steken. BREW kan dus enkel reageren op events zoals toetsaanslagen of timers die afgaan. Zoals reeds gezegd zijn er geen floating point berekeningen mogelijk, enkel 16.16 fixed point formaat (zie bijlage B, sectie 9.2) is mogelijk.

Figuur 6.18: Screenshot van een testimplementie met de Vincent library (zelfde resultaat voor als voor Hybrid).

Storage space

Hoe moeten we nu onze variabelen opslaan zonder global of static data? In BREW wordt één enkele struct voorzien voor de gebruiker. Deze struct moet om te beginnen een AEEAplet structure bevatten. Vervolgens kunnen we alle data die we willen erin plaatsen. Hier wordt bij het opstarten van het programma dan heap space voor voorzien. Een pointer naar deze struct wordt gemaakt en continu doorgegeven waar nodig.

Timers

Alles in BREW is event gebaseerd. Als we bijvoorbeeld in een functie willen blijven door middel van een loop dan zal het BREW toestel gewoonweg opnieuw opstarten. BREW herkent dit als het niet meer reageren van de applicatie en lost dit op door een full reboot. Om real-time rendering mogelijk te maken moet gebruik gemaakt worden van timers om continu nieuwe beelden te verkrijgen. BREW biedt een eenvoudige methode aan om timers te creëren met de ISHELL_SetTimer. We moeten dan het aantal milliseconden meegeven dat bepaalt om de hoeveel tijd een bepaalde functie opgeroepen moet worden.

Fixed point math

Doordat de ARM chips in de BREW toestellen geen floating point berekeningen ondersteunen moet continu gewerkt worden met 16.16 fixed point getallen (zie Bijlage B, sectie 9.2).

Input

Wanneer een toets wordt ingedrukt of gelost, verkrijgen we een event. Het is aan de gebruiker zelf om bij te houden welke toetsen nu ingedrukt zijn en welke niet. Een goede manier om dit bij te houden is een array maken met voor elke toets een boolean die op *true* staat als de toets ingedrukt is en op *false* indien niet.

OpenGL ES

Wanneer al de data, de heap en de stack geïnitialiseerd zijn kan het OpenGL ES gedeelte beginnen. BREW maakt voor de initialisatie en rendering gebruik van IGL en IEGL. IGL is een interface naar GL en IEGL is een platform specifieke interface tussen IGL en de onderliggende architectuur. We moeten voorzichtig omgaan met de initialisatie van de display. Er kunnen snel fouten

gebeuren en een toestel kan gemakkelijk volledig blokkeren. Vanaf dat al de OpenGL ES initialisaties gebeurd zijn kan er op dezelfde manier als voor Vincent en Hybrid OpenGL ES code geschreven worden. We moeten wel steeds rekening houden met het feit dat al onze data zich in die ene struct moet bevinden en dat we altijd manueel floating point getallen naar fixed point notatie moeten omzetten. Ook bij BREW hebben we een testimplementatie gemaakt en we hebben dezelfde resultaten bekomen als bij Vincent en Hybrid: initialisatie, texturing, navigatie en het inladen van 3ds modellen.

Hoofdstuk 7

Toekomstig werk

De geïmplementeerde case study kan op verschillende manieren uitgebreid worden. Doordat texturing informatie ontbreekt en er geen referentie is naar welke texture bij welke muur hoort, is het zeer moeilijk om visueel mooie resultaten te bekomen. De lengtes van de muren zijn zo verschillend dat elke muur apart een op voorhand bepaalde texture nodig heeft. Een uitbreiding van deze case study is bijvoorbeeld een referentielijst maken tussen muren en textures en in plaats van real-time dynamische texturing, op voorhand reeds bepalen hoeveel nieuwe textures ingeladen moeten worden voor elke view. Dit zal de PDA ontlasten en betere resultaten leveren. Ook hebben alle muren momenteel dezelfde hoogte. Dit geeft ons de luxe om alle berekeningen in 2D te maken. Een interessante uitbreiding kan dan muren van variabele hoogte toelaten waardoor 2,5D berekeningen nodig zullen zijn.

Een stad met enkel getexturede muren is maar saai. Het toevoegen van aanwezige elementen in de stad zoals standbeelden, lantaarnpalen, bomen, skyboxen, enzoverder kan het gevoel van immersie en de grafische pracht aanzienlijk verhogen. Deze objecten zijn meestal te klein om als occluderende objecten beschouwd te worden tijdens de preprocessing. Een andere techniek om de eventuele zichtbaarheid van deze objecten te bepalen is dus nodig. Een eenvoudige oplossing hiervoor is gaan kijken of het object zich in de cell waarin we staan bevindt of in een van de cellen die zichtbaar is vanuit de huidige cell.

Nog een interessante uitbreiding is van deze lokale applicatie een draadloze netwerk applicatie maken. De gebruiker bezit dan initieel geen enkele informatie van de stad. Door middel van een GPS ontvanger wordt de positie van de gebruiker bepaald en deze wordt naar een centrale server in de stad verzonden. Deze server kijkt in welke cell de gebruiker zich bevindt en ver-

zendt alle informatie van deze cell naar het mobiele toestel. Vermits de server ook weet welke cellen zichtbaar zijn vanuit de cell, kunnen reeds de volgende cellen efficiënt verstuurd worden. Er stellen zich verschillende problemen bij deze uitbreiding. Een zeer belangrijk probleem zijn visuele artefacten. Zodra de gebruiker zich in de stad bevindt moet er onmiddellijk een beeld getoond worden. Ook als de gebruiker zich door de stad begeeft moet alle informatie tijdig op het toestel aanwezig zijn en op tijd gerenderd worden. Dit kan door beperkingen in bandbreedte soms niet gerealiseerd worden. Hierdoor zullen we gaten in de wereld zien of plots verschijnende objecten. Een mogelijke oplossing hiervoor is om eerst een beeldgebaseerde voorstelling [59] (zie ook sectie 3.4.3) van de zichtbaarheid in een cell te verzenden en vervolgens deze te vervangen door een gerenderd beeld.
Hoe gedetailleerder de stad, hoe meer informatie verzonden moet worden per cell en hoe meer informatie het mobiele toestel moet onthouden. Door de beperking in geheugen van de mobiele toestellen is een goede balans tussen het opslaan, inladen, uitladen en verwijderen van data vereist.

Hoofdstuk 8

Conclusies

In de literatuurstudie van deze thesis zijn we gaan kijken naar wat er reeds bestaat op het gebied van rendering voor mobiele toestellen zoals PDA's en GSM's. We zien dat mooie en gedetailleerde beelden renderen op mobiele toestellen, voor allerlei verschillende toepassingen, een opkomende trend is. Dit kan gemakkelijk verklaard worden door te kijken naar het enorme succes van rendering op vaste computers, zoals een desktop PC. Als deze rendering op vaste toestellen reeds zo populair is, dan zal dezelfde rendering op een mobiel toestel vaak nog interessanter zijn. Dezelfde informatie kan dan immers eender waar en wanneer bekeken worden op een toestel op zakformaat. We denken hierbij aan games, datavisualisatie bij constructies, medische toepassingen, navigatiesystemen, ...
Deze trend is ook zichtbaar wanneer we kijken naar de vele hardware- en software-ontwikkelaars die zich op deze afzetmarkt gericht hebben. Er zijn reeds vele verschillende besturingssystemen, processorarchitecturen, renderingstechnieken en bruikbare toepassingen ontworpen. Hoewel deze markt al enorm gegroeid is, is deze toch nog jong en lijdt ze onder de verschillende beperkingen van de hardware in mobiele toestellen. Deze toestellen zijn nu eenmaal zeer beperkt in rekenkracht, geheugen, batterijduur en schermgrootte. We staan dus nog ver af van de reeds gerealiseerde toepassingen op vaste computers. Verder onderzoek naar efficiënte rendering en algoritmes moet gedaan worden, evenals naar de ontwikkeling van nieuwe en betere hardware technologieën.

Uit het implementatie-gedeelte van deze thesis kunnen we besluiten dat beelden renderen op mobiele toestellen niet vanzelfsprekend is en vele malen complexer is dan voor vaste computers. Veel fuctionaliteit, libraries, dll-bestanden, headers, ... worden niet ondersteund op mobiele toestellen. Eenvoudige operaties zoals input en output of het gebruiken van een library

kunnen al snel een probleem vormen. In vele gevallen is het gebruik van reeds ontwikkelde code (op vaste computers) een serieus struikelblok door beperkte ondersteuning van de gebruikte functionaliteiten in deze code. Dit leidt vaak tot het opnieuw coderen van reeds bestaande functionaliteit.
Verder moet er continu rekening gehouden worden met de beperkingen van het toestel. Het geheugen en de rekenkracht vormen hierbij het grootste struikelblok. We moeten dus opletten met de hoeveelheid data die we inladen. Textures kunnen bijvoorbeeld al gauw enorm veel geheugen innemen. Efficiënte en snelle algoritmes die het geheugen zorgzaam gebruiken zijn dus een must.

De geïmplementeerde case study toont aan dat een zichtbaarheidsalgoritme noodzakelijk is voor een walkthrough door een stad. We kunnen dit veralgemenen naar vele andere toepassingen. Elke scène die gerenderd moet worden en uit vele objecten en textures bestaat waarbij het doel is een mooi en gedetailleerd eindresultaat te bekomen, zal een efficiënt zichtbaarheidsalgoritme nodig hebben. Het geïmplementeerde zichtbaarheidsalgoritme van de case study ontlast de rendering pipeline zodanig, dat bevredigende framerates gehaald worden en het geheugen optimaal gebruikt wordt.

De vraag is nu: zal rendering op mobiele toestellen ooit de huidige pracht en praal van vaste computers evenaren? Als we kijken naar de snelle vooruitgang in technologie, toepassingen en resultaten van de voorbije jaren dan is het antwoord positief. Maar ook hardware heeft zijn limitaties, denk maar aan de nodige koeling bij sterkere processoren. Zijn nieuwe, sterkere algoritmes en hardwarecomponenten mogelijk? Zullen deze een oplossing bieden? Tijd zal het uitmaken.

Bibliografie

[1] Klimt - the Open Source 3D Graphics Library for Mobile Devices
. http://studierstube.icg.tu-graz.ac.at/klimt/.

[2] 2D GPS System
. http://www.th-ales.cz/gps.htm.

[3] An Introduction to BREW and OpenGL ES
. http://www.gamedev.net/reference/articles/article2135.asp.

[4] ARM: The Architecture for the Digital World
. http://www.arm.com/.

[5] ATI IMAGEON
. http://www.ati.com/products/imageon2300/.

[6] BitBoys: Enabling the visual mode
. http://www.bitboys.com/index_bb.php.

[7] BREW - The Ultimate Mobile Gaming Environment
. http://brew.qualcomm.com/brew/en/developer/resources/dev_resources.html.

[8] Direct3D Mobile for Windows Mobile-based Devices
. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/mobilesdk5/html/mob5oriDirect3DMobile.asp.

[9] Falanx: Image quality without compromise
. http://www.falanx.no/index.html.

[10] Fathammer xforge mobile game development software
. http://www.fathammer.com/.

[11] Fixed-point Arithmetic
. http://www.accu.org/acornsig/public/caugers/volume2/issue6/fixedpoint.html.

[12] Fixed Point Math
. http://members.aol.com/form1/fixed.htm.

[13] Gshark - Takumi
. http://www.gshark.com/en/gshark/two/index.html.

[14] Hybrid Graphics
. http://www.hybrid.fi/.

[15] Ideaworks3d segundo3D mobile 3d graphics technology
. http://www2.ideaworks3d.com/.

[16] Intel targets new "XScale"CPU core at mobile and Internet apps
. http://www.linuxdevices.com/news/NS5976614542.html.

[17] Intel XScale Technology
. http://www.intel.com/design/intelxscale/.

[18] Johnson's triVM intermediate language
. http://www.njohnson.co.uk/.

[19] Leverage the Latest in Mobile Rendering Protocols
. http://www.commsdesign.com/csdmag/sections/featurearticle/showArticle.jhtml?articleID=16502618.

[20] M3G
. http://en.wikipedia.org/wiki/M3G.

[21] MBX Graphics Accelerator Cores
. http://www.arm.com/products/esd/multimediagraphics_mbx.html.

[22] Nokia NGage
. http://www.n-gage.com/.

[23] NVidia GoForce
. http://www.nvidia.com/page/handheld.html.

[24] OpenGL ES
. http://www.khronos.org/opengles/.

[25] OpenGL ES Tutorials
. http://www.zeuscmd.com/tutorials/opengles/index.php.

[26] OpenVG
. http://www.khronos.org/openvg/.

[27] Operating Systems
. http://en.wikipedia.org/wiki/Personal_digital_assistant.

[28] PDA Hardware
. http://www.intersil.com/applications/printdoc/PortableDigitalAssistant(PDA).asp.

[29] PDA Use Study
. http://personal.bgsu.edu/~nberg/chilabs/pda.htm.

[30] Pocket PC
. http://en.wikipedia.org/wiki/Pocket_PC.

[31] PowerVR Developer Relations
. http://www.pvrdev.com/.

[32] PowerVR MBX
. http://www.powervr.com/Products/Graphics/MBX/index.asp?Page=2.

[33] PowerVR/STMicro Kyro preview
. http://tweakers.net/nieuws.dsp?ID=11306.

[34] Processors, Memory, Expansions, and Wireless Explained
. http://www.davespda.com/features/explained.htm.

[35] Sony NV-XYZ777: Navegador GPS 3D
. http://www.fayerwayer.com/archivo/linux/.

[36] Sony PSP
. http://www.us.playstation.com/PSP/.

[37] SVG Mobile Requirements, W3C Working Draft 3 August 2001
. http://www.w3.org/TR/2001/WD-SVGMobileReqs-20010803.html.

[38] Vincent 3-D Library
. http://ogl-es.sourceforge.net/.

[39] F. Lamberti A. Sanna, C. Zunino. A distributed architecture for searching, retrieving and visualizing complex 3d models. *Dip. di Automatica e Informatica - Politecnico di Torino, Italy.*

[40] Jiri Bittner and Peter Wonka. Visibility in computer graphics. *Center for Applied Cybernetics, Czech Technical University in Prague, Institute*

of Computer Graphics and Algorithms, Vienna University of Technology, Graphics, Visualisation and Usability Center, Georgia Institute of Technology.

[41] Slajs J. Slavik P. Brachtl, M. Pda based navigation system for a 3d environment. *Department of Computer Science and Engineering, Czech Technical University, Prague, Czech Republic.*

[42] Chun-Fa Chang and Shyh-Haur Ger. Enhancing 3d graphics on mobile devices by image-based rendering. *Department of Computer ScienceNational Tsing Hua UniversityHsinchu, Taiwan, R.O.C.*

[43] Petri Honkamaa Mika Hakkarainen Charles Woodward, Seppo Valli. Wireless 3d cad viewing on a pda device. *Technical Research Centre of Finland, VTT Information Technology.*

[44] Claudio T. Silva Fredo Durand Daniel Cohen-Or, Yiorgos Chrysanthou. A survey of visibility for walkthrough applications.

[45] Yuval Noimark Daniel Cohen-Or and Tali Zvi. A server-based interactive remote walkthrough. *Enbaya Ltd., Computer Science Department, Tel-Aviv University, IMB Research Lab in Haifa.*

[46] Yuval Noimark Daniel Cohen-Or and Tali Zvi. A server-based interactive remote walkthrough. *Computer Science Department, Tel-Aviv University, IBM Research Lab in Haifa.*

[47] A. Sanna A. Fiume F. Lamberti, C. Zunino and M. Maniezzo. Augmented reality with large 3d models on a pda - implementation, performance and use experiences. *Proceedings of Web3D 2003 Symposium, pages 55-61, Saint Malo, France, March 2003. ACM/SIGGRAPH.*

[48] Andrea Sanna Antonino Fiume Marco Maniezzo Fabrizio Lamberti, Claudio Zunino. An accelerated remote graphics architecture for pdas. *DAUIN - Politecnico di Torino, Italy.*

[49] Ren Ng Randall Frank Sean Ahern Peter D. Kirchner James T. Klosowski Greg Humphreys, Mike Houston. Chromium a stream-processing framework for interactive rendering on clusters. *Stanford University, Lawrence Livermore National Laboratory, IBM T.J. Watson Research Center.*

[50] Sampsa Gummerus. Conservative from-point visibility. *University of Tampere, Department of Computer Science, Master's Thesis.*

[51] Ben Juurlink Iosif Antochi and Stamatis Vassiliadis. Selecting the optimal tile size for low-power tile-based rendering.

[52] Jani Timmerheid Ismo Rakkolainen1 and Teija Vainio. A 3d city info for mobile users. *Digital Media Institute, Tampere University of Technology, P.O.BOX 553, 33101 Tampere, Finland, Hypermedia Laboratory, 33014 University of Tampere, Finland.*

[53] Ove Sommer Klaus Engel and Thomas Ertl. A framework for interactive hardware accelerated remote 3d-visualization. *University of Stuttgart, IfI, Visualization and Interactive Systems Group.*

[54] Robert R. Lipman. Mobile 3d visualisation for construction.

[55] Leonard McMillan. An image-based approach to three-dimensional computer graphics. *Ph.D. Dissertation, University of North Carolina.*

[56] Michael Wimmer Peter Wonka and Francois X. Sillion. Instant visibility. *Vienna University of Technology.*

[57] Computer Graphics Technology Purdue University Sara Kubik, Assistant Professor. Creating graphics for both web pages and pda dispays. *STC's 50th Annual Conference Proceedings.*

[58] Michael F. Cohen Steven J. Gortler, Li-wei He. Rendering layered depth images. *Harvard University, Stanford University, Microsoft research.*

[59] Patrick Monsieurs Wim Lamotte Tom Jehaes, Peter Quax. Hybrid representations to improve both streaming and rendering of dynamic networked virtual environments. *Expertise Centre Digital Media, Limburg Universitair Centrum, Universitaire Campus, Diepenbeek, Belgium.*

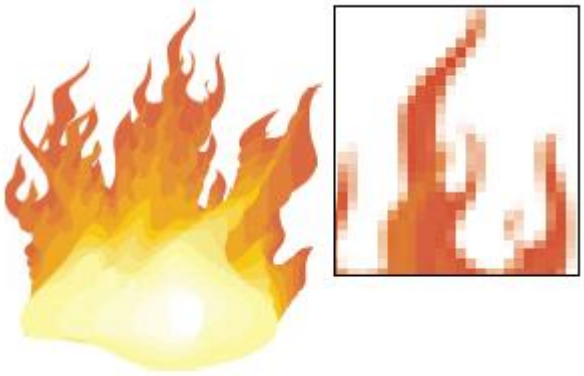
[60] Daniel Wagner and Istvan Barakonyi. Augmented reality kanji learning. *Vienna University of Technology, Vienna, Austria.*

Hoofdstuk 9

Bijlagen

9.1 Bijlage A: Vector graphics

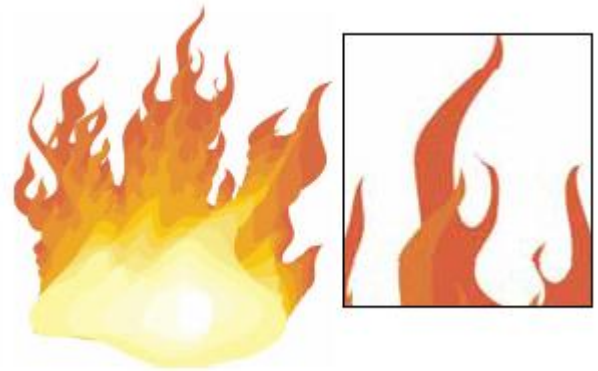
Er bestaan twee methoden om beelden te tonen [57]. Door gebruik te maken van raster graphics of door gebruik te maken van vector graphics. De meest gebruikte methode is raster graphics. Figuur 9.1 illustreert deze methode. Een scherm bestaat uit een vast aantal pixels, net zoals een foto bij raster



Figuur 9.1: Voorbeeld van raster graphics [57]

graphics. Als een foto bestaat uit 100 op 100 pixels, zullen 10000 pixels nodig zijn op het scherm om deze foto in ware grootte te tonen. Als we nu inzoomen op deze foto, zoals naar het topje van de vlam op figuur 9.1, dan zoomen we in op de pixels van die foto. Het topje van de vlam bestaat uit pakweg slechts 500 pixels. Deze 500 pixels moeten dan getoond worden op de 10000 pixels op het scherm. Vele pixels van de foto moeten dan dubbel getoond wor-

den en we verkrijgen zogenaamde schaakbord artefacten. Deze trapsgewijze artefacten zijn duidelijk zichtbaar op figuur 9.1. Dit is ongewenst. Hoewel er verschillende technieken bestaan om dit negatief effect weg te werken blijft het een feit dat bij raster graphics een foto zijn kwaliteit verliest bij het in- of uitzoomen van zijn originele grootte. Vermits 2D/3D omgevingen vaak interactief zijn, gebeuren er vele transformaties en scaleringen, waardoor het regelmatig kan voorkomen dat deze storende artefacten voorkomen.
Een oplossing voor dit probleem is de tweede techniek, namelijk vector graphics.
Vector graphics beschrijven objecten op een hoger niveau. Objecten worden gedefinieerd door hun grootte, positie en geometriek in plaats van door pixels. Doordat deze beelden niet door pixels gedefinieerd worden, is het detail in de beelden invariant voor veranderende resoluties, transformaties en scaleringen. Dit is zichtbaar in figuur 9.2 en 9.3.



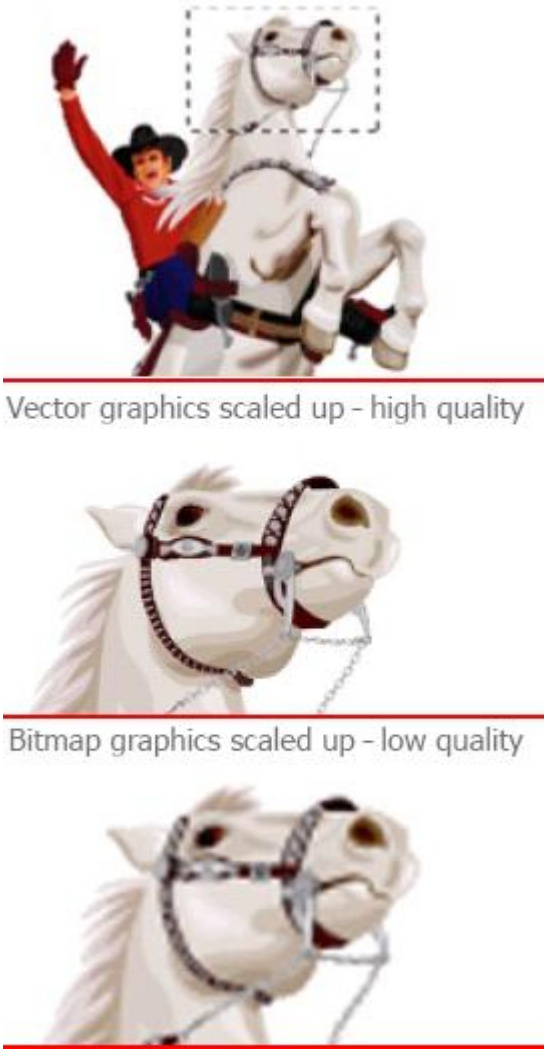
Figuur 9.2: Voorbeeld van vector graphics [57].

Vector graphics worden nog niet zo vaak toegepast. Op het internet zien we deze vector graphics nog maar enkel bij websites ontwikkeld met Macromedia Flash. Andere toepassingen zijn CAD design en navigatiesystemen.

We zullen nu even de voor- en nadelen van beide technieken bespreken.
We beginnen met raster graphics.

Voordelen van Raster graphics

- **Realisme**: Door het gebruik van raster graphics is enorm veel detail mogelijk. Hoe meer pixels in het beeld, hoe gedetailleerder het beeld kan worden. Beelden getrokken door een fototoestel zijn raster graphics.



Figuur 9.3: Een tweede voorbeeld van vector graphics [57].

Nadelen van Raster graphics

- **Gebaseerd op resolutie**: Om artefacten te voorkomen moeten beelden steeds gemaakt worden met de resolutie van het outputscherm in gedachten. Een beeld voor een PDA zal een hele andere resolutie hebben als een beeld voor een 19 inch scherm. Dit maakt de beelden toestelafhankelijk.

- **Onflexibele weergavegrootte**: Zoals we hierboven hebben aangetoond met een voorbeeld kunnen raster graphics niet zomaar vergroot of verkleind worden en hun oorspronkelijke detail behouden. De exacte grootte van het beeld moet dus geweten zijn om het beste resultaat te verkrijgen zonder artefacten.

- **Grote bestandsgrootte**: Raster graphics kunnen zeer gedetailleerd zijn, maar dit komt met een kost. De bestandsgrootte bij raster graphics is al snel groot. Zoals uit hoofdstuk 2 blijkt geheugen niet het sterkste punt van een mobiel toestel.

We gaan verder met de voor- en nadelen van vector graphics.

Voordelen van Vector graphics

- **Scaleerbaar**: Doordat er mathematische beschrijvingen gebruikt worden voor de beelden kunnen deze gemakkelijk vergroot of verkleind worden. Een cirkel vergroten is bijvoorbeeld enkel de parameter voor de straal vergroten. Hierbij hebben we geen verlies aan detail. Een ander voorbeeld is te zien aan de vlam bovenaan. Doordat de vlammen met curves mathematisch beschreven zijn, heeft zoomen geen effect op het detail.

- **Gemakkelijk manipuleerbaar**: Een raster foto die aangepast moet worden moet pixel per pixel aangepast worden. Eventuele artefacten moeten weggewerkt worden. Het beeld moet zo goed als helemaal opnieuw geconstrueerd worden. Bij vector graphics hebben we dit probleem niet. Lijnen blijven lijnen, curves blijven curves, of deze nu groot, klein of gedraaid weergegeven moeten worden. Het beeld dan aanpassen - bijvoorbeeld roteren - kan gemakkelijk gebeuren door de juiste parameter aan te passen.

- **Kleine bestandsgrootte**: Doordat een figuur een mathematische beschrijving op hoog niveau is, is de bestandsgrootte veel kleiner dan bij raster beelden. Een 100 op 100 beeld van een cirkel bij raster graphics is een waarde voor elk van de 10000 pixels bijhouden. Bij vector

graphics is dit slechts een middelpunt, straal, randbreedte en kleur. Een ander voorbeeld kan gezien worden op de figuur in 2.3.2.

Nadelen van Vector graphics

- **Cartoon uitzicht**: Zoals reeds vermeld werken vector graphics met mathematische beschrijvingen. Dit komt overeen met beelden die randen, ronde vormen en sterke kleurwisselingen benadrukken. Hoewel ook met vector graphics realistische beelden gegenereerd kunnen worden, komt het vaak voor dat deze een cartoon-achtige indruk geven.

- **Beperkt in gebruik**: De mathematische beschrijvingen van een vectoriaal beeld moeten geïnterpreteerd worden vooraleer dit uitgetekend kan worden. Dit vereist meestal een speciaal programma of plug-in. De Macromedia Flash Player is hier een goed voorbeeld van. Zulke plug-ins zijn vaak onhandig, ongewenst of niet bruikbaar.

Beide technieken hebben dus hun sterke en zwakke kanten maar bijna overal worden uitsluitend raster graphics gebruikt. Het gebruik van vector graphics op mobiele toestellen komt wel steeds vaker en vaker voor. Bitboys heeft zelfs een hardware chip hiervoor ontwikkeld, de G12. Deze hebben we besproken in 2.3.2. De reden hiervoor is omwille van de voordelen van vector graphics. De bestanden zijn kleiner en er is dus geen (eventuele) compressie nodig en ook zijn de beelden gemakkelijk scaleerbaar en resolutieonafhankelijk, wat zeer belangrijk is bij de kleine schermen. Door speciale hardware zoals de G12 uit 2.3.2 kunnen vector graphics snel op trage toestellen gerenderd worden.

Een API die hardware geaccelereerde 2D vector graphics ondersteunt is OpenVG [26]. Deze is speciaal ontworpen voor mobiele toestellen en moet het mogelijk maken om energie-efficiënt snelle en mooie user interfaces te maken.

9.2 Bijlage B: Floating point vs Fixed point

[11][12]Bij het gebruik van floating point getallen hebben we een zeer groot bereik van getallen, van zeer klein tot zeer groot (bv. 0,0001 tot 10000). Floating point getallen worden opgeslagen als een getal met op een bepaalde positie het decimale punt. Deze getallen heten floating point omdat dit decimale punt van positie kan veranderen. Floating point operaties zijn zware operaties en vergen veel processortijd. Dit is geen goede zaak voor PDA's.

Daarom gebruiken deze toestellen vaak fixed point berekeningen. Hierbij verliezen we veel flexibiliteit en bereik maar we winnen enorm veel snelheid. Fixed point werkt als volgt. We fixeren het decimale punt op een vaste positie in getallen en we gebruiken integer berekeningen hierop. Integer berekeningen zijn een speciaal geval van floating point berekeningen en zijn aanzienlijk sneller. We illustreren dit met een decimaal voorbeeld:
Stel dat we twee getallen na de komma toelaten. We doen dan alle getallen maal 100. 5 wordt 500 en 2.01 wordt 201.
We kijken eerst naar de optelling. Stel dat we 2.02 + 2.02 willen uitwerken met fixed point berekeningen. Beide doen we maal 100. Dit geeft ons 201 + 201 = 402, dit is gelijk aan onze 4.02 die we zouden kijgen bij floating point berekingen als we ons resultaat terug door 100 delen. Analoog geldt dit voor aftrekking.
Hoe zit het nu met vermeningvuldigingen. 201 x 201 geeft ons 40401. Omdat de gebruikte getallen 100 maal groter dan de "echte" waarden zijn zal de oplossing 10000 maal groter zijn dan de "echte" oplossing. Hier vormt zich echter een probleem. Door deze vermenigvuldigingen komen we vaak aan veel te grote getallen die buiten het bereik vallen van bijvoorbeeld een "long". Dit kunnen we oplossen door de deling door 100 niet op het einde te doen, maar tussenin. In plaats van 201 x 201 doen we 201/10 x 201/10. Hier hebben we dus door 100 gedeeld. We verliezen nu wel precisie, want ons resultaat wordt 20 x 20 = 400. Dit kan opgelost worden door het getal 201 te zien als 200 en 1. Onze vergelijking wordt nu

$$\frac{(200+1).(200+1)}{100}$$

of

$$\frac{200.200+200.1+200.1+1.1}{100}$$

Verder geeft dit

$$\frac{200}{10}.\frac{200}{10}+\frac{200.1}{100}+\frac{200.1}{100}+\frac{1.1}{100}$$

We hebben enkel de eerste 3 termen nodig, de laatste term valt automatisch weg. We bekomen zo ons gewenst resultaat. Merk op dat de laatste term altijd de vorm a + b/100 heeft waarbij a en b kleiner zijn dan 10. We hebben zo maximaal een fout van 0.81 (9x9/100).

We hebben nu de werking van fixed point getallen aangetoond met decimale getallen. Analoog kan dit toegepast worden op binaire getallen. Een gewone integer omzetten in een fixed point waarde is dit getal 8 bits voorwaarts schuiven.

```
fixed_var = (long) int_var << 8;
```

Een floating point omzetten is anders omdat deze in IEEE formaat staan. Bit shifting zou deze waarden vernietigen. In de plaats van bitshifting vermenigvuldigen we onze float met 256 (2 tot de macht 8). Optelling en aftrekking zijn gewone integer operaties. Vermenigvuldiging en deling is anders. Net zoals bij ons decimaal voorbeeld komen we uit met teveel bits bij een vermenigvuldiging. Dit lossen we op door 8 bits naar achteren te schuiven.

```
fixed3 = (long) (fixed1 * fixed2) >> 8;
```

Ook bij deling krijgen we ongewenste factoren in onze berekening. Dit kunnen we oplossen door de teller 8 bits naar links te schuiven.

```
fixed3 = (long) (fixed1 << 8) / fixed2;
```

Deze omzettingen in aparte klassen of functies zetten is veel te traag, daarom kunnen deze best als macro's gedefinieerd worden. Enkele macro's die gebruikt worden bij de BREW implementatie kunnen hieronder gevonden worden.

```
typedef long fixed; // Our new fixed point type.

#define itofx(x) ((x) << 8) // Integer to fixed point
#define ftofx(x) ((x) * 256) // Float to fixed point
#define dtofx(x) ((x) * 256) // Double to fixed point
#define fxtoi(x) ((x) >> 8) // Fixed point to integer
#define fxtof(x) ((float) (x) / 256) // Fixed point to float
#define fxtod(x) ((double)(x) / 256)         // Fixed point to double
#define Mulfx(x,y) (((y) * (x)) >> 8) // Multiply a fixed by a fixed
#define Divfx(x,y) ((y << 8) / (x))     // Divide a fixed by a fixed
```

Door het gebruik van fixed point operaties kunnen berekeningen veel sneller gebeuren dan met floating point operaties, soms tot 300% sneller!

Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen en uw akkoord te verlenen.

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Rendering voor PDA of GSM
Richting: **Licentiaat in de informatica** Jaar: **2006**
in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Deze toekenning van het auteursrecht aan de Universiteit Hasselt houdt in dat ik/wij als auteur de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij kan reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

U bevestigt dat de eindverhandeling uw origineel werk is, en dat u het recht heeft om de rechten te verlenen die in deze overeenkomst worden beschreven. U verklaart tevens dat de eindverhandeling, naar uw weten, het auteursrecht van anderen niet overtreedt.

U verklaart tevens dat u voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen hebt verkregen zodat u deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal u als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze licentie

Ik ga akkoord,

Rafael KLIMAS

Datum: