Effektiv 3D-visualisering med stråleføIging

Grafikkprosessorer brukes til å tegne stadig mer realistiske 3D-bilder. Men for å oppnå høyere kvalitet kan det hende at man snart må se til mere virkelighetsnære algoritmer.

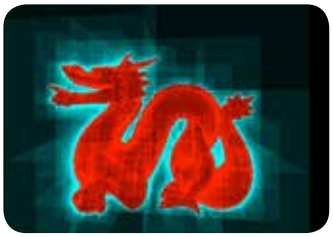
Av Audun Wilhelmsen, vinner av Mikroelektronikkprisen 2012

Dagens grafikkprosessorer (GPU) bruker en teknikk som kalles rastrering for å tegne 3D-bilder med sanntidsytelse. Denne teknikken går grovt sett ut på å transformere polygoner slik at de plasseres på skjermen i et 3D-perspektiv, og deretter fylle inn polygonene. For å oppnå effekter som skygger og refleksjoner må man bruke diverse triks som gir tilnærmet realistiske resultater.

StråleføIging er en alternativ teknikk som tegner bilder ved å følge lysstråler i revers, fra kamera til lyskilde. Skygger og refleksjoner faller naturlig på plass fordi algoritmen nærmere følger lysets fysiske oppførsel. Men det har vært utfordringer med å få teknikken rask nok til bruk i sanntidsapplikasjoner. Hovedutfordringen ligger i å raskt finne kollisjoner mellom lysstrålene og objekter i en 3D-verden.

Oktrær

En visualiseringsalgoritme trenger en datamodell av objektet som skal visualiseres. Istedenfor å bruke stråleføIging med polygonmodeller, som er idéelt for rastrering, kan man bruke datastrukturer som er mer egnet for stråleføIging. Oktrær er en trestruktur som rekursivt deler et rom i åtte like store underrom. Ved å fortsette å dele slik, får vi til slutt rom som er så små at vi kan se på dem som partikler, og en partikkel kan enten være solid eller tom. Oktrær kan effektivt representere mange scener, fordi vanlige scener består av mye tomrom (luft). Hvis et rom er tomt trenger vi ikke å dele rommet i mindre biter.



Figur 1: Visualisering av et oktre fra programvaremodellen

Det å traversere en stråle for å finne en kollisjon, blir oversatt til et søk i oktreet. StråleføIgingen blir dermed effektiv fordi man enkelt kan hoppe over store tomrom i scenen og raskt finne partikkelen som strålen kolliderer med.

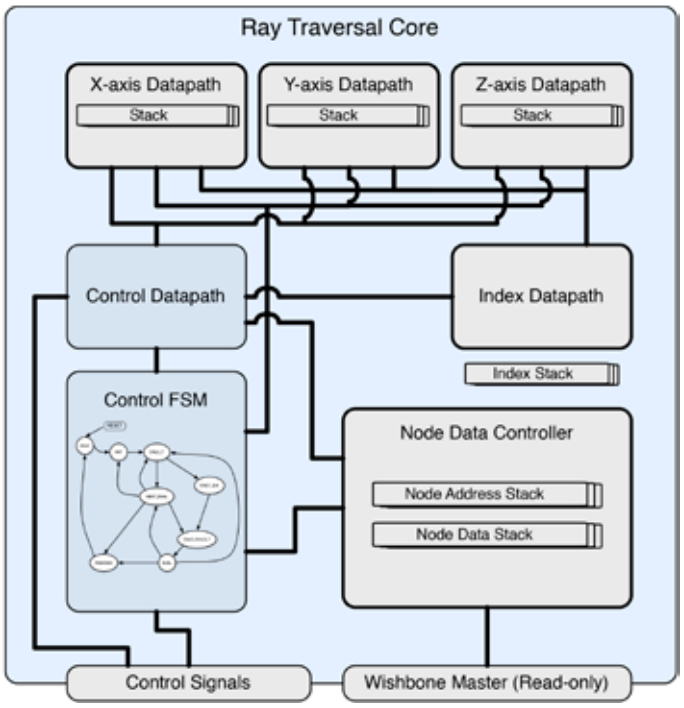
Problemstilling

Det finnes flere implementasjoner av stråleføIging i oktrær i programvare og på programmerbare GPUer som oppnår tilnærmet sanntidsytelse. Men da har man lite ressurser til overs til andre ting som skal utregnes, og man står igjen med en teknologidemo som er imponerende visuelt sett, men har lite interaksjon.

I denne masteroppgaven ønsket jeg å undersøke om det ville være gunstig å implementere teknikken i spesialisert maskinvare, og om man ville kunne oppnå sanntidsytelse med et lite maskinvaredesign. Jeg tok utgangspunkt i en eksisterende programvarealgoritme som så egnet ut til å tilpasse til maskinvare.

Programvaremodell

For å tilpasse algoritmen til maskinvare, ble først en modell utviklet i programvare. Hensikten var å raskt utforske ulike optimaliseringer og verifisere at den optimaliserte algoritmen fortsatt var korrekt. Samtidig ble det undersøkt om algoritmen ville fungere i parallel med rastrering utført av en tradisjonell GPU. Dette skulle vise om teknikkene kunne brukes ved siden av hverandre, slik at man kan bruke hver av dem til det de er mest egnet til. Sjeldent kan man erstatte gamle løsninger ved å kaste ut alt det gamle. Ofte er praktiske løsninger evolusjoner heller enn revolusjoner.



Figur 2: Kjernen i stråleføIgingsmodulen

Optimaliseringer

Den første vurderingen som ble gjort var om algoritmen skulle implementeres med flyttall eller tallrepresentasjon med fast komma. Flyttall krever mer ressurser, og kan være tregere, men kan gi mer nøyaktighet. Det viste seg at kjernen i algoritmen fint kunne utføres med fast komma, og dette krevde veldig lite ressurser siden kjernen kun utfører addisjoner og subtraksjoner. Algoritmen ble nøye analysert for å finne konsekvensene av valg av bitbredde og plassering av komma.

Ettersom algoritmen traverserer et tre, har den nytte av en stack. Den andre store optimaliseringen gikk ut på å redusere mengden data som måtte lagres på stacken. Den originale algoritmen lagret tre tall på stacken. Det ble oppdaget at det var kun tre bits med informasjon som var nødvendig for å raskt rekonstruere disse tallene fra nåtilstanden. Dette reduserte dramatisk størrelsen på stacken.

Til slutt ble algoritmen formulert som en tilstandsmaskin, og kalkulasjonene som ble utført i hver tilstand ble nøye valgt, slik at maskinvareimplementasjonen skulle kreve minimalt med funksjonelle enheter.

Maskinvareimplementasjon

Etter at en algoritmen var utviklet, ble en maskinvareimplementasjon designet i Verilog. Designet tok utgangspunkt i et SoC design, og algoritmen ble laget som en akselerator. Oktreet blir plassert i et minne, og modulen får adressen til dette minneområdet. Modulen går igjennom alle pikslene i bildet,

og setter igang en kjerne som utfører stråleføgingen. Når kjernen finner en kollisjon mellom strålen og en solid partikkel, blir pikselen for denne partikkelen tegnet til en rammebuffer i minnet, som siden kan lastes ut av minnet eller sendes til en skjerm over HDMI.

Modulen kan instansiere opp til fire kjerner, for å kunne følge flere stråler i parallell. En kontrollenhet bestemmer hvilken kjerne som skal brukes når en ny stråle skal følges. To teknikker ble prøvd ut; "round-robin", hvor modulen alltid velger den neste kjernen i rekken for hver stråle, og "first-available", hvor modulen velger den kjernen som først blir tilgjengelig. Sistnevnte viste seg å være mest effektivt i dette designet, men her er det fortsatt mye rom for forskning. Eksempelvis, hvis to kjerner startes samtidig med to stråler som ligger rett ved hverandre, vil kjernene stort sett aksessere samme områder i oktreet samtidig. Dette gir mulighet for optimaliseringer, for eksempel ved å samkjøre minneaksess og dele resultater.

Kjernen implementerer en tilstandsmaskin som utfører algoritmen slik den ble modellert i programvare. Den har tre databaner for å regne ut nye tilstander for X,Y og Z-aksen, og én databane for å finne indeksen til neste rommet i oktreet som strålen traverserer. Kjernen inneholder også en stack, og dybden på denne stacken er parameteriserbar.

Hurtigbuffer

Den optimaliserte kjernen kjører meget raskt, og det er dermed kort mellom hver gang den må hente data fra oktreet. Dette førte til at jeg fort mettet båndbredden til minnet. En av de viktigste delene av designet ble dermed et hurtigbuffer (cache), som ble plassert mellom kjernene og minnebussen. I dette designet delte alle kjernene et felles hurtigbuffer.

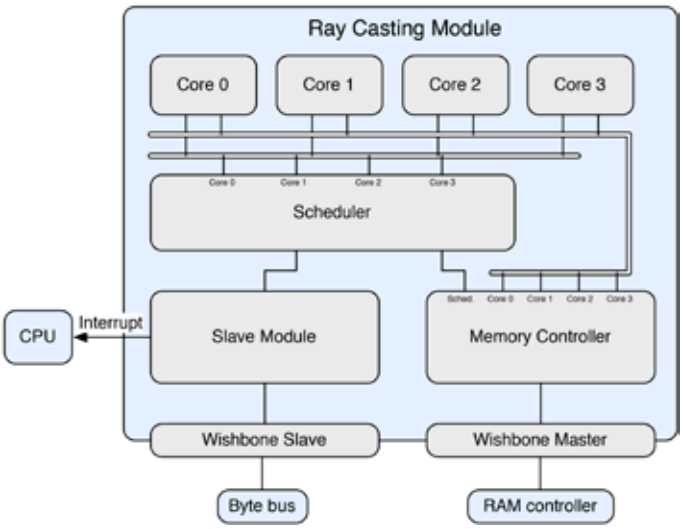
For å finne et optimalt design på dette hurtigbufferet, ble forskjellige hurtigbuffere implementert og sammenliknet. Størrelse på hurtigbufferet, antall ord som ble hentet ved bufferbom, typen til bufferet ("direct mapped" og "two way set-associative") og antall kjerner ble variert. Tiden det tok å tegne et bilde med disse parameterene ble sammenliknet med hvor mye ressurser det resulterende designet krevde. Dette ga god innsikt i hvordan designet av hurtigbufferet påvirket ytelse og bruk av ressurser.

Resultat og videre arbeid

Det endelig designet oppnådde god ytelse på veldig lite area, en relativt beskjeden FPGA og lav klokkehastighet. Algoritmen var hovedsakelig begrenset av minnebåndbredde, og mot slutten av oppgaven ble denne båndbredden hovedsakelig spist opp av andre undermoduler enn kjernen, og denne båndbredden kunne lett vært frigjort med litt videreutvikling. Tross dette, hvis man ekstrapolerer til en moderne GPUs båndbredde, kan man tegne et Full-HD bilde i 60 rammer per sekund med god margin. Med videre optimaliseringer er det tenkelig at man også kan gjøre dette med vesentlig mere kompliserte bilder enn det jeg har demonstrert i denne oppgaven, og dermed oppnå en ytelse som oppfyller kvalitetskravene til et moderne spill. Man kan også se på andre bruksområder av samme algoritmen. Ny forskning fra Nvidia har f.eks. brukt oktrær til å beregne forplantning av diffust lys i en tradisjonel rastreringsmotor. Det bør også forskes på å akselerere konstruksjonen av oktrær, for å kunne muliggjøre animasjon av oktremodellene.

Kildekode og videodemonstrasjon av prosjektet er tilgjengelig på: http://awil.no

Figur 4: Resultater fra FPGA-prototype



Figur 3: Stråleføgingsmodulen