Ray-packet-tracing med ARM:s NEON-arkitektur

POPULÄRVETENSKAPLIG SAMMANFATTNING
GUSTAF WALDERMARSON

Handledare: Johan Grönqvist (ARM)
Examinator: Michael Doggett (LTH)

Introduktion

På senare år har datorgrafiken utvecklats enormt, framförallt på mobila enheter som smartphones och tablets. Förut kunde man knappt surfa på dem, men numera klarar dessa enheter mycket grafiskt krävande spel.

Processorn som finns i dessa enheter, ARM-processorn, kan göra betydligt mer. Den har funktioner som till exempel vektorprocessering vilket gör det möjligt att accelerera mer generella applikationer såsom ray tracing.

Just ray tracing används flitigt för att skapa realistiska bilder, men denna metod tar ofta mycket lång tid. Detta gör att varje förbättring som påskyndar metoden är mycket värdeful. I detta examensarbetet har jag utvecklat en enkel men väloptimerad ray tracing-applikation för några ARM-processorer och analyserat dess prestanda med hänsyn till både renderingstid och strömförbrukning.

Ray tracing

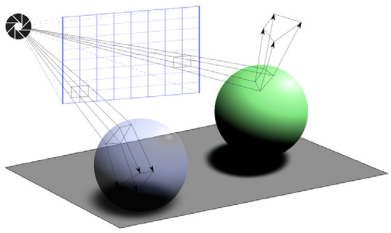
Ray tracing har länge varit den metod man använt för att skapa näst intill fotorealistiska bilder. Dessa bilder skapas genom att man följer fiktiva strålar från kamerans pixlar ut i scenen. Strålarna registrerar vilket objekt de träffat och väljer sedan antingen att använda objektets färg eller skapa en ny stråle. Den nya strålen kan till exempel ta den reflekterande riktningen för att också använda färgen från objektet därifrån. Denna metod är dessvärre oftast mycket långsam och lämpar sig ännu inte för realtids-applikationer. Trots det vill man självklart att applikationen ska vara så snabb som möjligt.

SIMD och NEON

Beräkningstunga applikationer kan ofta gå mycket snabbare om man utför samma operation på all data samtidigt. Av den anledningen utvecklades tidigt så kallade Single Instruction Multiple Data-instruktioner. Med dessa kan man samla ihop data i så kallade SIMD-vektorer, på vilka man sedan kan utföra samma operation på alla element samtidigt. I dagsläget är det vanligast att arbeta med vektorer som har fyra vektorelement. Processortillverkare kan tillhandahålla olika SIMD-teknologier med olika vektorlängder, operationer och datatyper, men addition och multiplikation på flyttals-vektorer finns i princip alltid tillgängliga. I ARM-processorer kallas dessa för Neon-instruktioner och de har tillgång till de flesta vanliga datatyper och operationer.

Ray-packet-tracing

Strålar från fyra närliggande pixlar går i många fall åt ungefär samma håll. Ofta går de samma väg genom scenen och kanske till och med träffar samma objekt. En effektiv optimering är då att paketera dessa strålar tillsammans och sedan använda SIMD-instruktioner för att göra beräkningar på alla strålar samtidigt. Just detta är tanken med så kallade ray-packet-tracers. I denna typ av ray tracer skapar man paket av strålar som är lika stora som vektorlängden för SIMD-teknologin, och följer således alla strålar i paketet samtidigt. Detta kräver i vissa fall att enstaka strålar i ett paket måste specialbehandlas, men generellt kommer de flesta strålar i paketet att träffa samma objekt vilket gör att prestandan ofta flerdubblas.

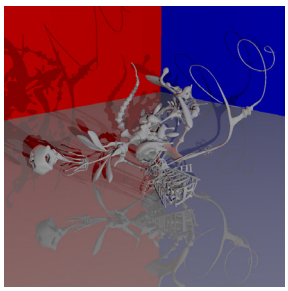


Exempel på hur ray-packet-tracing går till

Utvärdering och resultat

För att bedöma hur väl vår packet-tracer presterar har vi testat den på ett par olika ARM-plattformar och på ett urval av scener. En av scenerna vi testade vår optimerade packet-tracer på kan ses i figur 1. Renderingstiden och strömförbrukningen för denna scen på en Samsung Chromebook (XE303C12) kan ses i tabellen nedan.

	Referens	Packet-tracer
Renderingstid	32.6 s	12.6 s
Strömförbrukning	225 J	100 J



Figur 1 YeahRight-skulpturen, skapad av Keenan Crane

Slutsats

Resultat på denna scen visar att ray-packet-tracern ger en förbättring på ungefär 250% för renderingstiden jämfört med vår referens-ray-tracer, vilket i många fall mer än väl gör skäl för den större komplexiteten i en packet-tracer.

Den procentuella strömförbrukningen är något sämre än renderingtiderna men den är fortfarande ganska bra, med en förbättring på mellan 150-200% i genomsnitt i de scener vi testade.