Engine, er

Viktor Fröberg, Emil Lindström, Robin Skånberg, Alexander Svensson

`https://github.com/scanberg/engine/`

Inledning

Idén var att skapa en simpel grafikmotor från grunden så långt praktiskt möjligt. Följande paket används i projektet: glfw, glew, glm och newton-dynamics.

Plattformsoberoende

Ett av de största kraven som ställdes på grafikmotorn var att den skulle vara plattformsoberoende (Windows, Mac OS X och Linux). Detta krav, som är fint på alla sätt och vis, har medfört att en hel del av arbetet har gått åt till att se till att det verkligen kompilerar som det ska i alla operativsystem. Det som onekligen tog längst tid var när Newtonfysik-paketet syddes in, framförallt för att ett så stort paket lätt blir svåröversiktligt.

All kod finns tillgänglig på `https://github.com/scanberg/engine/`.

Objektinläsning

En del som har underlättat arbetet är möjligheten att läsa in objekt som exporterats från modelleringsprogram. För statiska figurer används formatet ASE och för animationer MD5. ASE-formatet är simpelt och välstrukturerat så programmet för inläsning skrevs från grunden. MD5-inläsningen är litet mer avancerad och vägledning har tagits från: `http://myapp.nl/?p=1053`.

Scenmesh

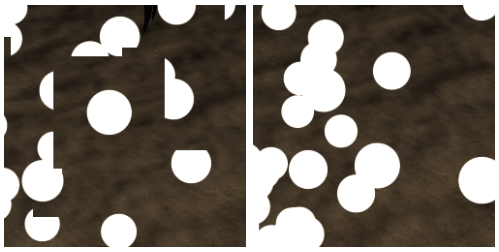
För att bygga upp scenen valde vi att skissa upp modellen i Google Sketchup, för att sedan exportera till 3DS Max och lägga på texturer och dylikt där. Vi kom fram till att Sketchup fungerar lika bra som 3DS Max i vår scen som bara innehåller enkla geometriska figurer.

Det blev problem med felriktade normaler i början och vissa sidor på objekt syntes därav ej. Men när problemet var löst var det lätt att skapa en ny scen som bestod av ett hus med olika rum och en trappa. Slutligen föll valet på en kyrkogård som scen.

Partikelsystem

Vi valde även att implementera ett partikelsystem. Det är väldigt enkelt uppbyggt med 100 partiklar som var och en är en rektangel. Dessa partiklar initieras en gång och får bland annat följande startvärden:

- Startposition: (0.0, 0.0, 0.0)
- Livstid, hur länge partikeln har varit i luften: 0.0
- Färdriktning i x- resp. y-led: slumpmässigt [-1.0, 1.0]



(a) Read + write (b) Read only

Figur 1: Olika läge på z-buffern

Inga nya partiklar skapas efter detta, utan de som har 'brunnit ut' återställs och får en ny slumpmässig färdriktning. För att få varje rektangel att alltid vara riktad mot kameran användes så kallad billboarding, där ytans normal konstant pekar mot kameran.

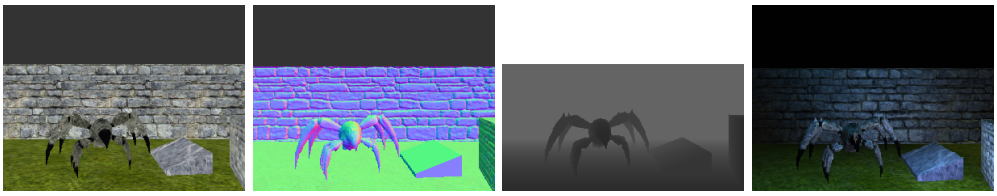
Det som gav mest huvudbry var överlappande partiklar med alfakanaler, se figur 1. För att komma runt detta hindrades skrivning till z-buffern när partiklarna hanterades.

Deferred rendering

Genom att stycka upp renderingen av en bild i flera steg så kan man lyckas göra ljusberäkningar som är oberoende av scenens komplexitet, d.v.s. 100 polygoner eller 1 000 000 polygoner spelar ingen roll för själva ljusberäkningarna. Till skillnad från den klassiska forwardrenderingen som renderar till en färg och djup-buffer så använder deferred rendering ytterligare en buffer där normalerna sparas. Detta görs i det första renderingspasset, även kallat 'geometrypass'.

Tabell 1: Pixelformat

Textur	Format	Storlek (bit)	Kanaler
Färg	RGBA	32	Röd Grön Blå Ambient
Normal	RGBA	32	X Y Z Specular
Djup	Depth24	24	Djup



(a) Färg (b) Normal (c) Djup (d) Färdig bild

Figur 2: Deferred shading

I det andra passet så beräknas pixelns position från djupdata, sedan beräknas ljus och detta kombineras till den slutliga bilden. Deferred rendering är en teknik som blir allt vanligare och används idag av spel som t.ex. Crysis 2, Killzone 2, GTA 4, Battlefield 3 och Starcraft 2. Lite saker som vållat problem:

- Normalmap i view-space, vanligtvis så gör man alla ljusberäkningar i tangent-space ('triangelns koordinatsystem'). Men nu krävdes det att alla beräkningar skulle göras i view-space.
- Depthmap, först användes en vanlig olinjär depthmap och denna 'linjäriserades' sedan i shadern när pixel-positionen skulle beräknas. Denna linjärisering ledde till att positionen blev förskjuten och ljuset belyste fel pixlar. Genom att istället skriva en linjär depthmap från början så kunde vi kringgå problemet.