Jämförelse av 3D spelmotorer

(Arbetsrubrik)

Mattias Berg, 33167

Institutionen för informationsteknologi

Handledare: Jan Westerholm

Referat

Innehållsförteckning

1 Inledning

En spelmotor är ett system för utveckling av primärt spel men används också till specialeffekter i filmer, skapandet av hela filmer och modellering av till exempel hus i arktitektur. Målsättningen med en spelmotor är att ge utvecklare en samling återanvändbara komponenter som gör utvecklingen effektivare.

Ordet spelmotor kan vara aningen missvisande för utvecklare av dessa eftersom en spelmotor inte bara är en spelmotor utan en samling av olika typer av motorer. Däremot en användare av en spelmotor dvs en spelutvecklare har stor nytta av ordet spelmotor eftersom det är lättare att arbeta med än flera skillda motorer. [1]

1.1 Vad är en spelmotor?

Det finns ingen exakt definition på vad en spelmotor är, i stället bör man fokusera på en definition som bygger på vad en spelmotor kan göra. En spelmotor är en samling motorer vars uppgift är att ge en bra grund att bygga ett spel på. En spelmotor måste med andra ord hantera alla data som den blivit given, göra beräkningar på den givna datan och skicka den vidare till nästa instans i processen.

Undermotorer till en spelmotor kan vara 3D-motorer, ljudmotorer, fysikmotorer, artificiell intelligens-motorer(AI-motorer) och så vidare. Den viktigaste av dessa är 3D-motorn som i stora drag sköter behandlingen av geometrisk data och överföringen av den till en 2D-bild som vidare skickas till datorns skärm. 3D-motorn anses vara den mest komplexa delen av en spelmotor. När kraven på spels grafiska återgivningen ökar så blir spelmotorernas utvecklare tvungna att utnyttja objekt- och scenhantering för att få flera effekter och mera detaljer på skärmen samtidigt. Detta görs oftast genom att inte rendera det som

inte är synligt för tillfället för spelaren. En spelmotor behöver inte innehålla någon speciell undermotor för att klassas som en spelmotor.

Ett spel det vill säga interaktiv media så kräver egentligen bara en form av indata. Indatan kan vara i form av text, indata från en styrenhet eller någon annan form av indata som datorn kan använda för att gå vidare i exekveringen. Utdatan från spelet behöver inte nödvändigtvis ändra beroende på indatan till exempel 'tryck på valfri tangent för att fortsätta'. Från den beskrivningen av ett spel så kan man ge enkla krav på indata till en spelmotor för att skapa ett spel. Det som krävs är ett sätt att läsa indata, göra någon form av beräkning och visa något annat på skärmen efter läst indata, till exempel ny text visas.

En spelmotor bör vara oberoende av projekt den används till och bör fungera för andra spel och andra multimedia mjukvaruprojekt utan att behöva ändra på källkoden. En spelmotor är alltså inte ett spel även om gränserna mellan spel och motor ibland kan vara i en gråzon. Enligt Zerbst och Duevel skall en spelmotor inte innehålla spelrelaterad kod men de flesta spelmotorer som finns tillgängliga för allmänheten idag innehåller standardkod för rörelse i spel. [1]

1.2 Historia

En av de första digitala spelen som är känt är Spacewar. Spacewar är ett tvåpersoners spel där spelarna kontrollerar varsitt rymdskepp och har som uppdrag att förstöra den andra spelarens rymdskepp. Spelet utvecklades för DEC:s PDP-1 (Digital Equipment Corporations Programmed Data Processor-1) som använde sig av 18-bit ordstorlek och med minnesutrymme för 4096 ord. Spelet kodades av Steve Russell och den första versionen blev klar i februari 1962, utvecklingen hade tagit uppemot 200 timmar. [2] [3]

Från Spacewar i 1962 till slutet av 1980-talet så programmerades spel från grunden upp för specifik hårdvara utan att använda sig av eller skapa

återanvändbara komponenter. Projektgrupperna för dessa spel var oftast små, en till några få personer. Men vartefter industrin växte så gjorde även projektgrupperna och spelens komplexitet ökade. Programmeringstiden av spel ökade kraftigt i takt med att datorer blev kraftfullare och flera funktioner skapades och behovet av återanvändbara komponenter ledde till de första spelmotorerna.

De första spelmotorerna utvecklades internt för första partens spel. Det fanns några få tredje parts spelmotorer under 1980-talet en av de första av dessa var War Games Construction Kit [4]. Tredje partens spelmotorer blev vanliga först på 1990-talet när 3D-spelmotorerna började utvecklas. Termen spelmotor kom till i mitten av 1990-talet i samband med utgivningen av de första stora 3D-spelen Doom och Quake av Id Software. Dessa spel ledde senare till designvalet att hålla spelmotorn och spelinnehållet skillt i dagens spelmotorer.

1.3 Spelmotortyper, ändamål och industri

Många spelmotorer tillåter utveckling till många olika plattformer till exempel Windows, Xbox, Mac och Playstation utan att någon ändring i källkoden krävs, med undantag av hårdvaruspecifika saker såsom inmatning från styrenheter och dylikt. Spelmotorer kan även innehålla specialiserade komponenter t.ex. fysikmotorn Havok som ger tillgång till komplexa algoritmer för att simulera många fysiska egenskaper snabbt [5]. RenderWare är en spelmotor som till skillnad från många andra låter användaren bestämma vilka undermotorer och komponenter som motorn skall använda sig av och på så sätt ge utvecklare en specialanpassad spelmotor som passar deras projekt.

Spelmotorer licensieras oftas ut med visuella utvecklingsverktyg nerpackat i en integrerad utvecklingsmiljö för att ge en så snabb spelutveckling som möjligt åt utvecklare. Dessa utvecklingsverktyg och –miljöer innehåller många element som tillåter snabb utveckling av ett komplett spel utan att spelutvecklarna

behöver skapa egna anordningar för att utföra diverse operationer såsom effekter, ljud och fysik. Denna typ av spelmotor som ger en utvecklare alla funktioner som kan tänkas behövas så kallas för mellanvara (eng. *middleware*) . Med hjälp av dessa spelmotorer så kan utvecklare sänka kostnader, komplexitet och tiden att få ut produkten till marknaden, och således sänka risken med spelutveckling. [6]

Marknaden för spel och spelrelaterade produkter är enorm. I 2006 uppskattades den globala marknaden ha ett värde på över 40 miljarder USD och är en av de marknader som växer snabbast. Men när endast en av tio spel klarar av att bli framgångsrikt eller ens betala tillbaka utvecklingskostnaderna så är spelutveckling väldigt riskfyllt. Detta har utvecklare försökt lösa med hjälp av spelmotorer licensierade av ett annat företag eftersom den dyraste delen av spelutveckling hör till ickespelrelaterade funktioner såsom renderering och scenhantering. Utvecklare av spelmotorer kan då fokusera på att få en så effektiv motor som möjligt och sälja licenser åt företag som utvecklar spel.

2 Unreal Engine

TODO: Metatext?

2.1 Historia

Unreal Engine 1 och spelet Unreal utvecklades som en seriös konkurrent till Id Softwares spel Quake2. Unreal som utgavs år 1998 ansågs vara sin tids grafiskt sett bästa spel med stora detaljerade miljöer både inomhus och utomhus. Unreal Engine 1 var utrustat med finesser som dynamiskt ljus och volymetrisk dimma. [7] [8]

Unreal Engine 2, utgivet år 2002, var ett försök av Epic att få en spelmotor för flera plattformer. Dess utgivningstitel var en militaristisk simulator America’s

Army som utvecklades som ett rekryteringsverktyg för den amerikanska armén. Men Unreal Engine 2s styrka visades när Mary Flanagan släppte sitt spel Domestic som med samma motor och samma verktyg som ett rekryteringsspel för armén lät spelaren styra en karaktär genom ett brinnande hus med en fokusering på poesi och nostalgi. Unreal Engine 2 introducerade även trasdocksfysik. I ett senare skede så släpptes Unreal Engine 2.5 som gav stöd åt fordonsfysik samt 64bit stöd i spelet Unreal Tournament 2004. [7]

Figur 1. Unreal Engine jämförelse [9]

Unreal Engine 3 med dess utgivningstitel Gears of War var den kraftigaste motorn hittills i Unreal Engine serien med finesser som blev tillagda hela vägen fram till dess utgivningsdag. Detta visade sig vara ett stort problem för många utvecklare som fick svårt att nå sina deadlines i tid. Problemet ansågs vara så stort att företaget Silicon Knights lämnade in en stämningsansökan mot Epic. Detta resulterade i att många spelutvecklare fortsatte att använda sig av en modifierad version av Unreal Engine 2.5. Men tack vare Unreal Engine 3s kraftiga verktyg som UnrealKismet och Matinee vilka tillät spelupplägg och spellogik att ändras utan att behöva ändra något i spelets källkod så började fler och fler företag flytta över till den nya versionen av Unreal Engine. [7]

Unreal Engine 3s kraftiga 3D-motor gentemot tidigare versioner (se Figur 1) gav nya möjligheter. Tv serien LazyTown använde sig av Unreal Engine 3 för att generera en scen och integrera den med film av skådespelare framför en grön skärm (eng *green screen*). [10]

Unreal Engine 4 som är nästa versionen av Unreal Engine serien kommer att fokusera på åttonde generationens PC-hårdvara och konsoler. En av dess centrala inslag är global realtidsbelysning som skulle få bort behovet av förberäknad belysning.

2.2 Scriptspråk

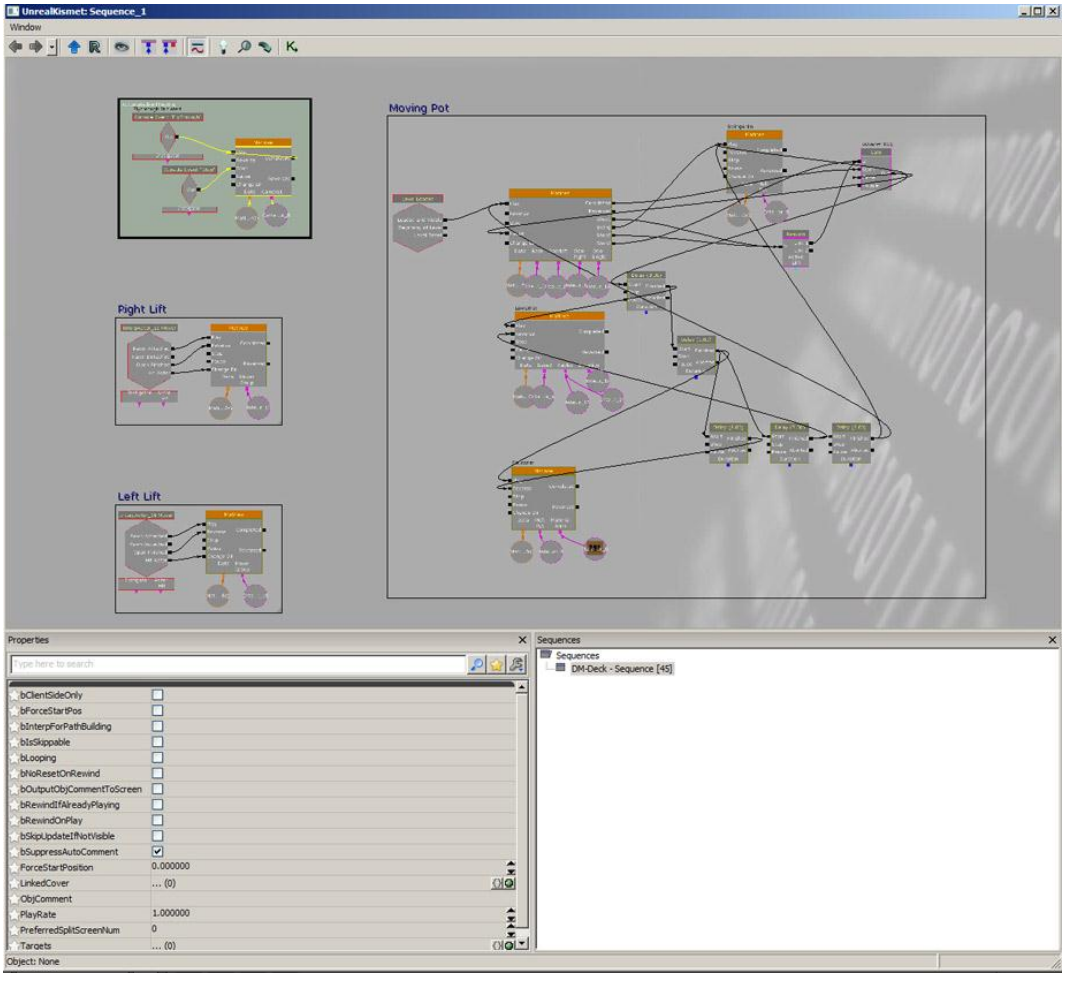
UnrealScript skapades för att underlätta och förenkla spelprogrammering. De största designmålen med språket är att ge stöd för spellogik var händelser tar en specifik speltid att utföras och är beroende på olika objekts tillstånd. Att ge en så simpel programmeringsstil som möjligt det vill säga felkontroll under kompileringstid, objektorientering och inget behov av pekare. Att ge utvecklare en hög nivå programmeringsstil där man använder sig av spelobjekt och interaktioner istället för bitar och pixlar. Denna simplicitet och kraft blev på bekostnad av exekveringstid, men licenstagare av Unreal Engine får tillgång till lågnivåsystem som till exempel rendereringsmotorn och kan med hjälp av det programmera på en lägre nivå och utnyttja snabbare exekveringstid av koden på bekostnad av simplicitet. [11] [12]

UnrealScript sköter tidshantering med hjälp av så kallade ”Ticks”. Ett tick är tiden det tar att exekvera uppdateringsfunktionen i alla script som körs i spelet, vanligtvis mellan 0,01 och 0,1 sekunder. Eftersom alla script körs på processorn (CPU:n) så fås att tiden för en tick är endast begränsad av hastigheten på processorn. UnrealScript kan innehålla kommandon som tar noll tickningar att utföra men också kommandon som tar flera tickningar. Funktioner som kräver

speltid för att utföras kallas latenta funktioner ett exempel på en sådan är en övergångsfunktion från en animation till en annan. [12]

Även fast en latent funktion körs så kan funktioner anropas i samma script av den virtuella maskinen eller andra script. Från en traditionell programmeringssynpunkt ser detta ut som varje objekt körs i en egen exekveringstråd. På grund av effektivitetskraven så används inte Windows trådar istället så simulerar UnrealScript användning av trådar. [12]

2.3 Kismet



Figur 2. Unreal Kismet [13]

Kismet är ett verktyg för att på ett visuellt sätt göra ändringar i spellogiken och objekt utan att behöva skriva eller ändra kod i script. Detta tillåter nivå

konstruktörer (eng. *level designers*) att göra ändringar i spelet utan att behöva gå via en programmerare.

Kismet använder sig av sekvensobjekt och linjer, se figur 2. Det finns fyra typer av sekvensobjekt. Det första är en händelse (eng. *event*) som ger en start till en sekvens till exempel när en nivå har laddats in. Den andra typen är en handling (eng. *action*) som utför en handling till exempel släcker en lampa. Den tredje typen är ett villkor som styr vilken väg sekvensen tar. Den fjärde typen är en variabel som sparar information av en speciell typ och använder färgkoder för att utvecklaren lätt skall se vilken typ av variabel det är frågan om. [14]

2.4 Centrala inslag

Unreal Engine 3 använder sig av en 64-bitars färgrendereringsledning (eng. *64-bit color rendering pipeline*) med stort dynamiskt omfång (eng. *high dynamic range*). Och har stöd för populära efterbehandlingseffekter som rörelseoskärpa, skärpedjup och sken (eng. *bloom*). [15]

En stor fördel för Unreal Engine 3 på mindre kraftfulla system är dess möjlighet att förberäkna synlighet. Med förberäknad synlighet så reduceras rendereringstiden speciellt vid tillfällen som deladskärmspel på konsoler med en bekostnad på högre minnesanvändning vid exekveringstid och byggtid för ljus. Denna metod fungerar dock endast för mellan eller små nivåer eftersom minnes- och beräkningskraven ökar med nivåns storlek. Under testning av en Gears of War 3 flerspelarnivå uppmättes en insparning av 2,7 millisekunders rendereringstid jämfört med utan förberäknad synlighet. [16]

Figur 3. Bildprojicering med ljusfunktion i Unreal Engine 3 [17]

Ljusfunktioner i Unreal Engine 3 representerar matematiskt hur ljus skall upplysa en scen. Ljusfunktioner använder sig av Unreal Engine 3s material system vilket tillåter det att projicera en bild och samtidigt lysa upp objektet på samma gång se figur 3. [17]

3 Unity 3d

TODO: metatext?

3.1 Historia

Utvecklingen av Unity började år 2001 och den första versionen blev klar och lanserades år 2005 på Apples Worldwide Developers Conference. Unity var designat för att användas och bygga spel till Mac datorer. Unitys popularitet

växte och redan år 2007 så lanserades version 2.0 och 2008 så hade Unity Technologies tredubblats i storlek. [18]

I september 2010 släpptes Unity 3 som gav tillgång till många nya funktioner som till exempel ett kraftfull partikelsystem, bättre pathfinding och stort dynamiskt omfång (eng. *high dynamic range*) renderering. I slutet av 2012 lanserades Unity 4 som introducerade DirectX 11 stöd för PC och realtidsskuggor för mobila enheter. [18]

3.2 Scriptspråk

JavaScript

```
function Machine(x) {
   this.kind = ["bulldozer", "lathe", "car"][x];
}

var c = new Machine(2);
print(typeof c.announce); // "undefined"

Machine.prototype.announce = function() {
   print("I am a "+this.kind+".");
};

print(typeof c.announce); // "function"
c.announce(); // prints "I am a car."
```

UnityScript

```
class Machine {
   var kind : String; // fields are public by default
   function Machine(x : int) {
      this.kind = ["bulldozer", "lathe", "car"][x];
   }
   function announce() {
      print("I am a "+this.kind+".");
   }
}

print(typeof Machine.prototype); // causes a compile-time error

var c = new Machine(2);

c.announce(); // prints  "I am a car."
```

Figur 4. Jämförelse mellan JavaScript och UnityScript [19]

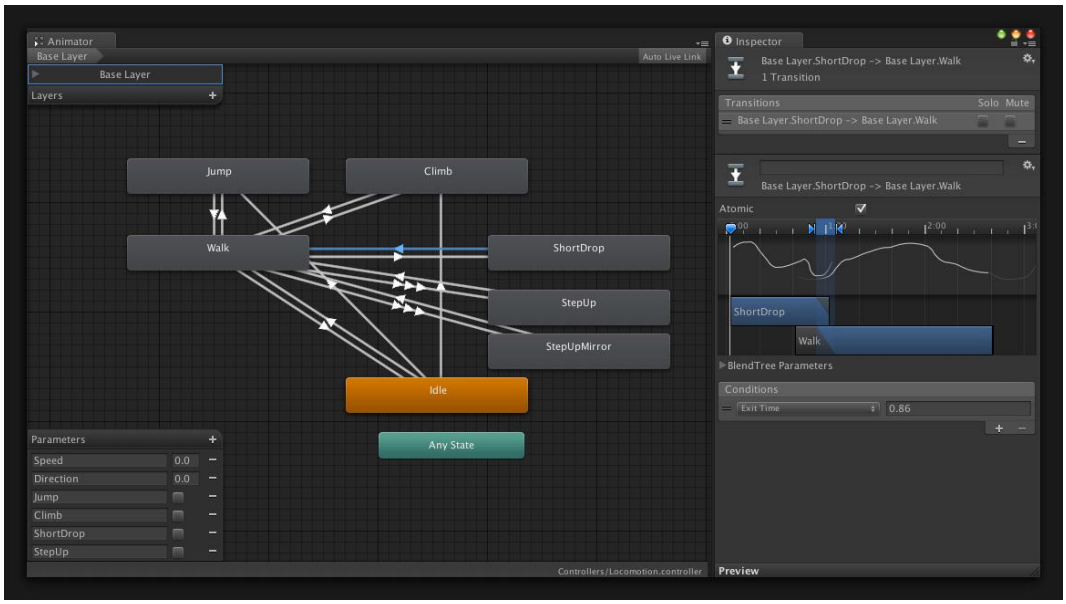
Unity använder sig av programmeringsspråket UnityScript som kan ses som en typ av JavaScript med avseende på dess filändelse ”.js” men JavaScript är ett

prototypspråk (eng. *prototypal*) medans UnityScript är klassiskt (eng. *classical*) se figur 4. [20]

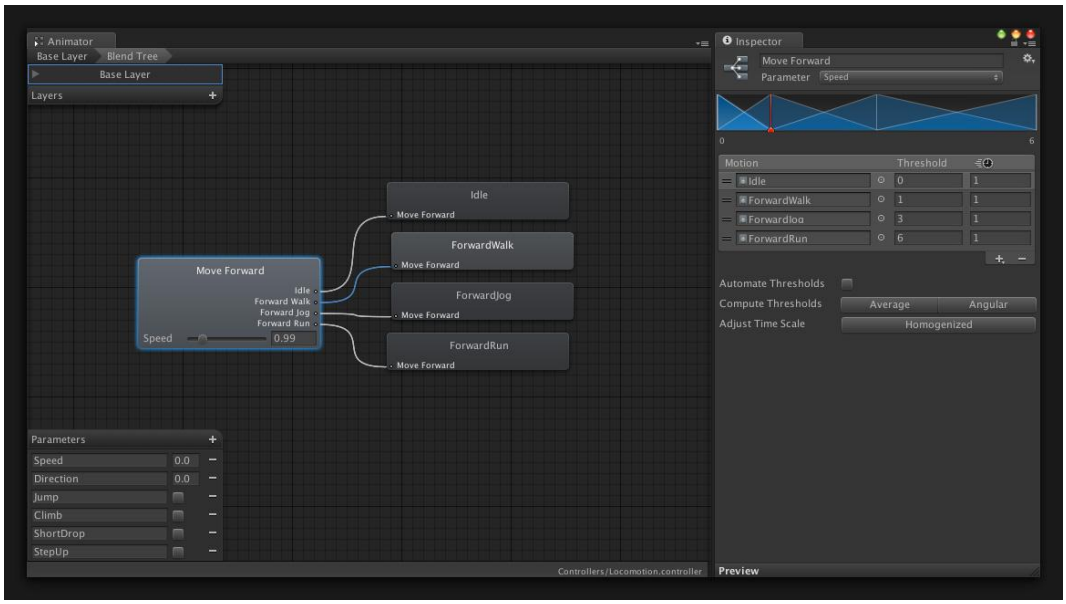
UnityScript har en funktion Update som körs på alla scripts som innehåller funktionen i scenen före rendereringen av en bild. Det är i denna funktion som det mesta av spelkoden finns. Det finns också en funktion som heter FixedUpdate som körst för varje fysiksteg som motorn gör. I denna funktion bör de flesta fysikoperationer läggas för att fungera korrekt. Kod utanför någon funktion körs när objektet laddas in, detta kan användas för att initialisera variabler som skall användas senare under exekveringen. [21]

3.3 Centrala inslag

Unity kan köpas i många olika versioner börjandes från en gratis Unity bas som tillåter utveckling till Windows, Linux och Mac men saknar egenskaper som realtidsskuggor och efterbehandlingseffekter. Nästa version kallas Unity Pro och stöder tidigarenämnda egenskaper. Utöver dessa två så kan man välja tilläggsdelar som ger stöd för iOS, Android och Flash utveckling även dessa kommer i en standard version och en Pro version. [22]



Figur 5. Tillståndsmaskin för animationer i Unity [23]



Figur 6. Blandningsträd för animationer i Unity [23]

Unity använder sig av Mecanim som är ett kraftfullt animeringssystem som är inbyggt och optimerat för Unity motorn. För att få animationerna så verkliga som möjligt används blandningsträd (eng. *blend tree*) och tillståndsträd (eng. *state machine*). Tillståndsträdet bestämmer i vilket tillstånd det som skall animeras är så att rätt animation körs till exempel om en karaktär är inaktiv eller går se figur 5. Blandningsträd bestämmer vilken animation som skall köras till exempel om karaktären skall gå eller springa se figur 6. Blandningsträdet sköter också om övergången mellan en animation och en annan för att ge en så jämn övergång som möjligt. [23]

4 Cryengine

TODO Metatext?

4.1 Historia

Cryengine 1 utvecklades av Crytek börjandes från år 2001 och blev klart år 2004. Samtidigt som spelmotorn utvecklades så arbetade Crytek på sitt första stora spel Far Cry som använder sig av den motorn. Cryengine 1 var den första motorn med per pixel skuggning (eng. *per pixel shading*) och stort dynamiskt omfång (eng. *high dynamic range*). Cryengine 1 hade stöd för grafiska processeringsenheter så långt bak som Nvidias GeForce 2 genom att både använda sig av per pixel skuggning som är den nya standarden för rendereringsmotorer och fast funktionsomvandling (eng. *fixed function transfrom*) med en register kombinerare som blandade texturer. [24] [25]

I Cryengine 2 så togs stöd för OpenGL och fast funktionsomvandling bort för att ge en lättare och klarare utvecklingsmiljö för skuggningsutvecklare. TODO!

4.2 Scriptspråk

TODO: Samma som för unrealscript

4.3 Centrala inslag

TODO: key features

5 Avslutande discussion

TODO: Avslutning

6 Referenser

[1] D. O. Zerbst Stefan, 3D Game Engine Programming, Boston, MA, USA: Course Technology / Cengage Learning, 2004.

[2] J. Markoff, ”Alan Kotok, 64, a Pioneer In Computer Video Games,” 3 Juni 2006. [Online]. Available: http://query.nytimes.com/gst/fullpage.html?res=9E0CE0DB1731F930A35755C0A9609C8B63. [Använd 27 Mars 2013].

[3] E. F. Anderson, S. Engel, L. McLoughlin och P. Comninos, ”The Case for Research in Game Engine Architecture,” *Future Play '08 Proceedings of the 2008 Conference on Future Play: Research, Play, Share,* pp. 228-231, 2008.

[4] RetroPC, ”War Games Construction Kit,” 21 Juni 2007. [Online]. Available: http://retropc.net/fm-7/museum/softhouse/ascii/000701300.html , http://translate.google.co.uk/translate?hl=en&sl=ja&tl=en&u=http%3A%2F%2Fretropc.net%2Ffm-7%2Fmuseum%2Fsofthouse%2Fascii%2F000701300.html. [Använd 30 Mars 2013].

[5] Havok.com inc., ”Havok Physics,” [Online]. Available: http://www.havok.com/products/physics. [Använd 30 Mars 2013].

[6] J. Tulip, J. Bekkema och K. Nesbitt, ”Multi-threaded Game Engine Design,” *IE '06 Proceedings of the 3rd Australasian conference on Interactive entertainment,* pp. 9-14, 2006.

[7] M. Thomsen, ”History of the Unreal Engine,” 23 Februari 2010. [Online]. Available: http://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine. [Använd 30 Mars 2013].

[8] Moddb, ”Unreal Engine 1,” 1 Januari 1989. [Online]. Available: http://www.moddb.com/engines/unreal-engine-1. [Använd 30 Mars 2013].

[9] Wikipedia, ”Unreal Engine Comparison,” 14 September 2006. [Online]. Available: http://en.wikipedia.org/wiki/File:Unreal_Engine_Comparison.jpg. [Använd 30 Mars 2013].

[10] J. Gaudiosi, ”Lazy Town,” [Online]. Available: http://www.unrealengine.com/showcase/film_television/lazytown/. [Använd 30 Mars 2013].

[11] EpicGames, ”Getting Started: Programming,” [Online]. Available: http://udn.epicgames.com/Three/GettingStartedProgramming.html. [Använd 1

April 2013].

[12] EpicGames, ”UnrealScript Language Reference,” [Online]. Available: http://udn.epicgames.com/Three/UnrealScriptReference.html. [Använd 1 April 2013].

[13] Epic Games, ”Unreal Kismet,” Epic Games, [Online]. Available: http://www.unrealengine.com/features/kismet/. [Använd 1 April 2013].

[14] EpicGames, ”Unreal Kismet User Guide,” Epic Games, [Online]. Available: http://udn.epicgames.com/Three/KismetUserGuide.html. [Använd 1 April 2013].

[15] Epic Games, ”The Unreal Rendering System,” Epic Games, [Online]. Available: http://www.unrealengine.com/en/features/rendering/. [Använd 1 April 2013].

[16] Epic Games, ”Precomputed Visibility,” Epic Games, [Online]. Available: http://udn.epicgames.com/Three/PrecomputedVisibility.html. [Använd 1 April 2013].

[17] Epic Games, ”Light Functions,” Epic Games, [Online]. Available: http://udn.epicgames.com/Three/LightFunctions.html. [Använd 1 April 2013].

[18] Unity Technologies, ”Fast Facts,” Unity Technologies, [Online]. Available: http://unity3d.com/company/public-relations/. [Använd 1 April 2013].

[19] Unity 3D, ”UnityScript versus JavaScript,” Unity technologies, 31 Mars 2013. [Online]. Available: http://wiki.unity3d.com/index.php/UnityScript_versus_JavaScript. [Använd 1 April 2013].

[20] T. Loewald, ”Head First into Unity with UnityScript,” Unity 3d, 24 Februari 2013. [Online]. Available: http://wiki.unity3d.com/index.php/Head_First_into_Unity_with_UnityScript. [Använd 1 April 2013].

[21] Unity 3D, ”Scripting Overview,” Unity Technologies, [Online]. Available: http://docs.unity3d.com/Documentation/ScriptReference/index.html. [Använd 1 April 2013].

[22] Unity Technologies, ”License Comparisons,” Unity Technologies, [Online]. Available: http://unity3d.com/unity/licenses. [Använd 2 April 2013].

[23] Unity Technologies, ”Simple and powerful animation technology,” Unity Technologies, [Online]. Available: http://unity3d.com/unity/animation/. [Använd 2 April 2013].

[24] Crytek, ”Cryengine 1,” Crytek, [Online]. Available: http://www.crytek.com/cryengine/cryengine1/overview. [Använd 3 April 2013].

[25] M. Mittring, ”Finding next gen: CryEngine2,” *SIGGRAPH '07 ACM SIGGRAPH 2007 courses,* pp. 97-121, 2007.