CHALMERS

Högprestandaapplikationer

En portering av spelet The Journey Down

ALEXI RIVERA

JOHAN EKLUND

Examensarbete

Högskoleingenjörsprogrammet för datateknik

CHALMERS TEKNISKA HÖGSKOLA
Institutionen för data- och informationsteknik

Göteborg 2012

Abstract

The market for mobile units is constantly growing and new hardware and operative systems are being renewed in a steady pace. To maximize profit it is important that companies create software that without modification, or with as small changes as possible, can be run on several platforms. The goal of this project has been to examine what problems might rise when porting existing code for high performance applications with heavy graphics, where processor and work memory are being fully utilized in a mobile environment. To test this an existing PC-game is being ported to Android. Since Android uses other systems regarding sound there were complications with implementation. Lack of time was responsible for not being able to complete a graphical implementation, which in turn made a complete testing in this area impossible.

Sammanfattning

Marknaden för mobila enheter växer ständigt och ny hårdvara och operativsystem förnyas i stadig takt. För att maximera vinsterna är det därför viktigt för företag att skriva mjukvara som utan modifikation, eller med så få förändringar som möjligt, kan köras på flera plattformar. Målet för projektet har varit att undersöka vilka problem som kan uppkomma vid portering av existerande kod för högprestandaapplikationer med mycket grafik, där processor och arbetsminne utnyttjas fullt ut i en mobil miljö. För att testa detta porteras ett existerande PC-spel till Android. På grund av att Android använder andra system för ljud uppkom det implementeringskomplikationer. Tidsbrist bidrog till att grafikimplementering aldrig slutfördes, detta medförde att fullständning testning inte kunde genomföras inom detta område.

Keywords: Android, Java, JNI, mobile platforms, portability

Innehållsförteckning

TERMINOLOGI

JD – The Journey Down, ett existerande PC spel från SkyGoblin.
CPU - Central Processing Unit
GPU - Graphics Processing Unit
JVM – Java Virtual Machine
JRM – Java Runtime Enviroment
JNI – Java Native Interface
API - Application Programming Interface
SDK – System Development Kit
NDK – Native Development Kit
STL – Standard Template Library
JIT – Just In Time
FPS - Frames Per Second
OpenGL - Open Graphics Library
OpenAL - Open Audio Library
OpenSL - Open Sound Library
ES - Embedded Systems

1. INLEDNING

1.1 Bakgrund

Allt eftersom marknaden för mobila enheter växer tillkommer ständigt ny hårdvara och operativsystem för mjukvara. För att maximera vinsterna är det därför viktigt för företag att skriva mjukvara som utan modifikation, eller med så få förändringar som möjligt, kan köras på flera plattformar - oavsett om det handlar om spelutveckling, nyttoprogram eller vetenskapliga beräkningsverktyg.

Hårdvaran i mobila enheter, framförallt smarta telefoner, har utvecklats i mycket hög takt de senaste åren. Många av de kraftfullare enheterna idag har processorer med dubbla kärnor och är kapabla till att både spela in och spela upp högupplösta filmer, något som för bara några år sedan var en utmaning även för stationära arbetsdatorer.

Vid diskussioner om processorprestanda nämns ofta Moore's lag, som säger att antalet transistorer per processorkärna kommer öka med det dubbla en gång vart annat år ("Moore's-Law" 2012). Processorer avsedda för stationära datorer har utvecklats i den här takten sedan 70-talet, och nu utvecklas även smartphones i samma takt, om inte snabbare (Shimpi 2012). En modern iPhone- eller Android-enhet kan många gånger vara mer kraftfull än en äldre PC. Enligt grafikkrets-tillverkaren nVidia kan mobila enheter, och framförallt då surfplattor, vara lika kraftfulla som dagens tvspels-konsoller så snart som år 2014 (Ployhar 2010)

Trots detta är många av de vanligaste applikationerna till smartphones relativt enkla. Några av de populäraste spelen till Android är till exempel porteringar av gamla Super Nintendo-spel, samt spelet Angry Birds(Ahlund 2010). Inget av dessa spel hade varit särskilt utmanande ur ett prestandaperspektiv för en modern stationär PC.

SkyGoblin är en Göteborgsbaserad spelstudio, grundad och ägd av bland annat två Chalmersalumner. Sedan 2005 utvecklar de spel till webb, mobil och PC. SkyGoblins nästkommande titel är första kapitlet av fyra i äventyrsspelsserien "The Journey Down". En första version av spelet utvecklades utan finansiellt stöd från utomstående källor (ett så kallat indie-spel) och fick då goda omdömen av kritiker. Företaget beslöt sig därför att driva vidare projektet och lägga sina resurser på att göra det större och mer avancerat. Spelet släpptes till PC och Mac under våren. Samt för att bredda sin publik så beslutade företaget sig för att dessutom portera spelet till Android och iOS (Rose 2012).

1.2 Problem

Mobila plattformar har begränsad hårdvaruprestanda jämfört med både bärbara och stationära datorer. Dessutom saknar de mobila plattformarna bredden av underliggande system för tjänster så som grafik och ljud som den stationära sidan har.

1.3 Syfte

Målet för projektet har varit att undersöka vilka problem som kan uppkomma vid portering av existerande kod för högprestandaapplikationer med mycket grafik, där processor och arbetsminne utnyttjas fullt ut i en mobil miljö.

1.4 Avgränsning

Porteringen gjordes endast till Android version 2.3 och senare.

2 METOD

För att utforska problem som uppkommer vid portering av existerande mjukvara porterades ett existerande PC-spel till Android miljön. Utöver detta gjordes en teoretisk diskussion av de problem som uppstår för att ta fram rekommendationer för framtida projekt.

Kunskap om Android inhämtas från den officiella dokumentationen. Kunskap om språken Java och C++ används litteratur såväl som officiell referensdokumentation. Kunskap om spelet *The Journey Down* står företaget för.

3 TEKNISK BAKGRUND

Här ges en övergripande beskrivning av begreppet prestanda, en kort beskrivning av plattformen Android, en beskrivning av programspråken Java och C++ utifrån ett plattforms- och prestanda-perspektiv samt The Journey Downs struktur.

3.1 Prestanda

Det är viktigt att spelet under körning kan renderas med tillräckligt många bilder per sekund (frames per second, fps). SkyGoblin vill uppnå 30 fps för bästa spelupplevelse, men ett minimikrav för att kunna köra spelet är 15 fps. Alldeles för låg fps i ett spel gör att upplevelsen av verklighetstrogen och mjuk rörelse i spelet störs, istället får man en ojämn uppspelning av spelkaraktärers rörelse och uppspelning av filmer och ljud. Det blir också svårare för spelaren att interagera med spelet i sig (Frame rate, 2012). Ett annat krav är att uppspelning av tal och ljudeffekter görs synkroniserat med renderingen. Därför måste avkodning och uppspelning av ljudresurser ske mycket snart efter att spelmotorn efterfrågat detta. Ett minimumkrav från SkyGoblin är 0,5 sekunders fördröjning. För uppspelning av musik är detta inte nödvändigt.

För att uppnå dessa krav måste systemets hårdvara hålla hög prestanda. Rendering av grafik i tre dimensioner kräver en mängd matematiska beräkningar. Ett flertal komponenter existerar för att åstadkomma detta.

En processor (CPU - Computer Processing Unit) är designad för att hantera relativt få anrop i taget för uträkning från mjukvara. Antalet exekveringar som kan hanteras samtidigt relaterar direkt till antalet kärnor i processorn.

En grafikprocessor (GPU - Graphical Processing Unit) är den komponent i en dator som i regel hanterar uträkningar gällande grafik. Den är designad för att hantera ett stort antal upprepade anrop från mjukvara åt gången. Nu för tiden används dock denna komponent i samspel med CPU för andra uträkningar också.

Till skillnad från en CPU som innehåller en handfull kärnor så kan en GPU innehålla närmare 100 kärnor. Skillnaden är att en CPU är designad för att hantera större mängd av kod per anrop där en GPU är designad för att hantera upprepad mindre mängd kod (Krewell, 2009).

Det går inte att göra någon direkt jämförelse mellan hårdvaruprestanda på PC med mobila enheter då allt för många saker skiljer sig mellan system.. Men för att få en fingervisning om hur kraven för spelet i fråga relaterar till en modern Android enheter tittar vi på de minimumkrav som SkyGoblin ställt för PC versionen av The Journey Down:

- CPU: Intel Pentium 4 1.8 Ghz
- RAM: 1 GB
- Hårdisk: 1 GB ledigt diskutrymme

PC-versionen av spelet använder DirectX för grafik, det är dock inget krav att använda detta bibliotek för de mobila enheterna då spelet använder OpenGL på de plattformarna. Som jämförelse använder vi här hårdvaruspecifikationen för Sony Xperia P (Sony, 2012):

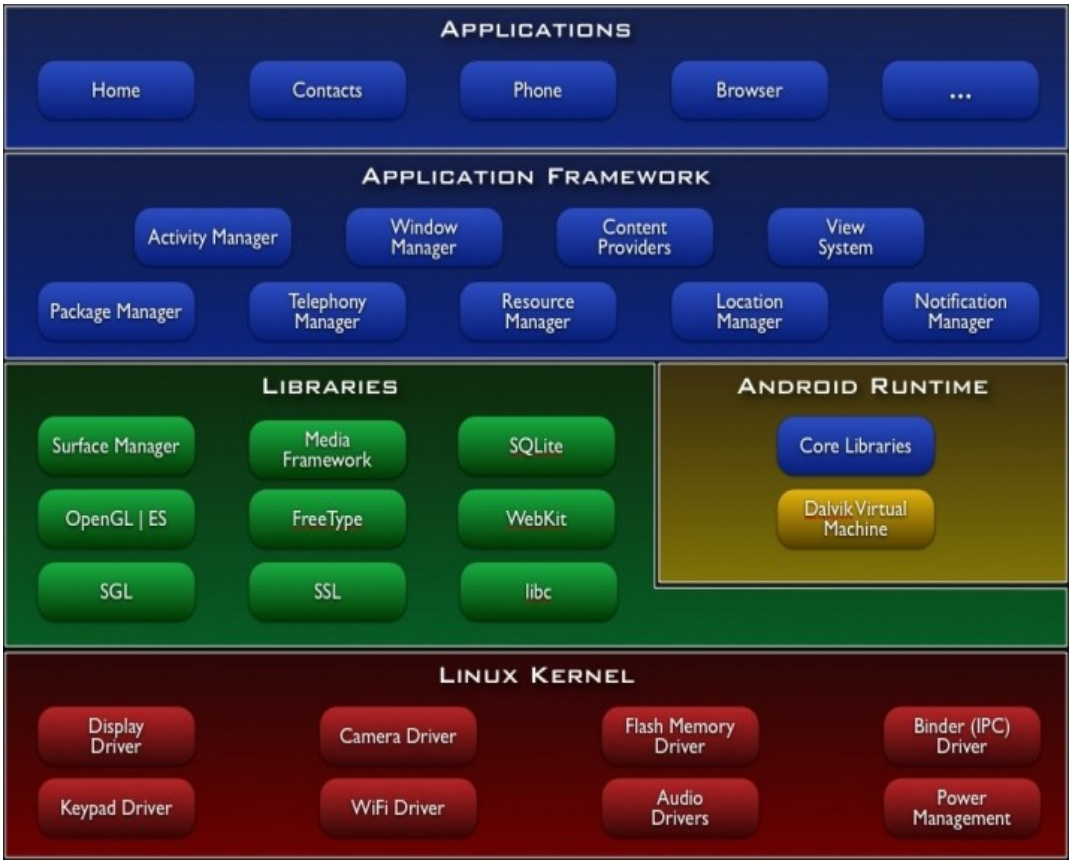
- CPU: 1 GHz U8500-processor med två kärnor
- RAM: 1 GB
- Internt telefonminne: 16 GB (upp till 13 GB ledigt för användaren)

Android-enheten har inte lika hög prestanda som en modern PC. Processorns hastighet når inte heller upp till de minimikrav som ställs på PC-versionen. Dock har enheten två kärnor, vilket bör hjälpa något. Dessutom kommer spelet köras i lägre upplösning på en mobil enhet än på PC, vilket bör sänka kravet på processorhastighet ytterligare. Enheten uppfyller övriga krav på minne och lagringsutrymme. Det är alltså inte orimligt att hårdvaran i en modern Android enhet skall klara av en portering av spelet.

3.2 Android

3.2.1 Androids olika lager

Android är uppbyggt av flera olika lager, vilket kan visas grafiskt genom följande diagram:



F 1.De olika lagren av mjukvara på Android (System architecture, 2012)

Grunden till Androidsystemet körs på Linuxkärnan 2.6 (2012) och tillhandhahåller också alla drivrutiner för hårdvaran i mobilen.

På lagret ovanför kärnan finner vi C/C++-bibliotek för bland annat tillgång till 2D/3D-grafik med OpenGL ES, databashantering med SQLite och möjlighet till avkodning av ett antal populära mediaformat. Dessa bibliotek går att komma åt genom de högre lagren, antingen direkt genom Androids API eller genom användandet av intern kod och Java Native Interface, JNI. På samma lager finner vi också Android Runtime som består av delar av Javas standard-API och den virtuella maskinen Dalvik.

Ytterligare ett steg upp finner vi Applikations-ramverket vilket är Androids egna API. Genom detta kan man komma åt lagret nedanför och härmed kan man alltså använda både metoder ifrån Androids API och det mesta av Javas API.

De plattformsberoende metoderna hanterar:

- *Delad data mellan applikationer*
- *Resurser*
- *Ljud, bilder, layout*
- *Notifikationer*
- *Möjligheten till att skapa notifikationer till operativsystemet från applikationer*
- *Aktivitetshanterare*
- *Hantering av en aktivitets olika stadier*
- *Vyer*
- *Omformbara komponenter som knappar, textboxar, listor och liknande*

Det översta lagret består av de applikationer som följer med Android som grund. Där inräknat telefon, telefonbok, browser och sms-hanterare.

3.2.2 Aktiviteter

En applikation i Android består av en eller flera aktiviteter, där varje aktivitet har sin egen livscykel. Android kan ha en aktivitet aktiv åt gången, det är den aktivitet som visas på skärmen. När en användare startar en ny aktivitet samtidigt som en tidigare aktivitet körs läggs den gamla aktiviteten på aktivitetsstacken. Detta möjliggör användade av bakåt-knappen för att återgå till den tidigare aktiviteten som kördes.

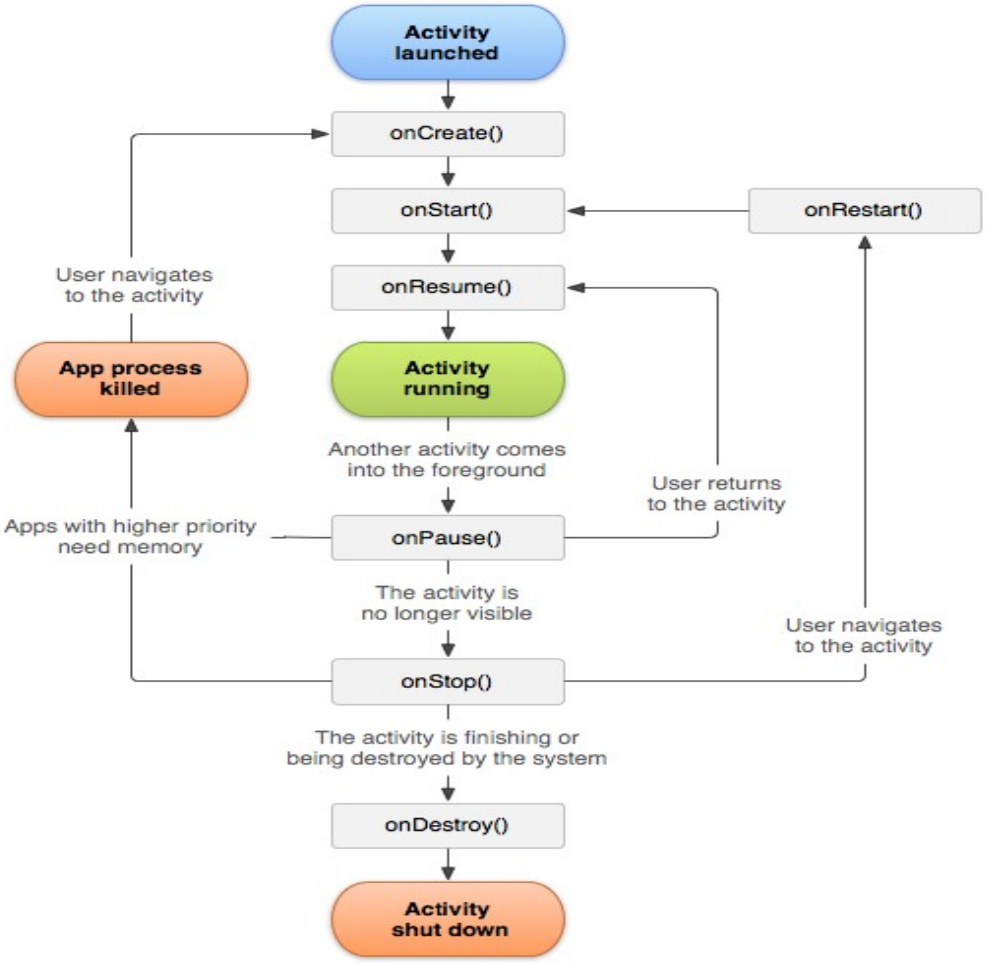
En aktivitet kan befinna sig i tre olika tillstånd:

- *Aktiv*
- *Aktiviteten har fokus och är synlig*
- *Pausad*
- *Aktiviteten har ej fokus men är synlig*
- *Stoppad*
- *Aktiviteten är ej synlig*

Då Android behöver minnesutrymme tar det sig friheten att terminera exekverande aktiviteter utan förvarning. Stoppade aktiviteter har högst prioritet att termineras, därefter stängs synliga aktiviteter ner vid behov.

Utvecklare har möjlighet att själv definiera hur aktiviteten ska reagera vid olika stadier i livscykeln för att undvika komplikationer som kan uppstå när Android själv väljer att terminera en aktivitet eller när den kommer ur fokus på grund av ett applikationsbyte.

Till exempel så är det ofta önskvärt att spara nödvändig data när en aktivitet stängs för att sedan hämtas när aktiviteten startar nästa gång.
Den fullständiga livscykeln för en aktivitet och dess anropade metoder vid olika stadier kan åskådliggöras med följande flödeschema:



F 2. Flödesschema över en aktivitets livscykel på Android (Activity lifecycle, 2012)

En aktivitets livscykel är något som utvecklare måste vara väl insatta i vid portering av mjukvara, då detta är ett av de verkligt plattformsberoende områdena. Ett spel bör tillexempel inte termineras av systemet utan att data först sparas undan så att det kan återupptas vid ett senare tillfälle.

3.2.3 Android Native Development Kit

Native Development Kit, NDK, är en samling verktyg som möjliggör användandet av intern kod i Android-projekt. Det vill säga kod som kompilerats speciellt för systemets processor, och genom JNI, kan exekveras under Java – det dominerande språket i Android-miljöer. Detta gör att delar eller i princip hela applikationen kan skrivas i till exempel C/C++, antingen för att använda existerande bibliotek och källkod eller för att i vissa fall få en applikation med bättre prestanda än ren Java. (Turner 2009).

NDK innhåller bland annat stöd för språket C, samt stöd för stora delar av C++. För att ett system skall klara av applikationer med avancerad grafik och ljud är det viktigt att det finns mjukvarustöd för detta. I NDK ingår stöd för 3D grafik med OpenGL ES 1.1 och 2.0 och stöd för ljudsystem med OpenSL ES 1.0.1. För en fullständig genomgång av de olika delarna se appendix 1(Overview 2012).

3.2.4 Open Graphics Library for Embedded Systems

För att visa upp grafik i 3D används Open Graphics Library for Embedded Systems, OpenGL ES. Android har sedan version 1.0 haft stöd för OpenGl ES 1.1. Sedan Android version 2.2 har plattformen dessutom haft stöd för OpenGL ES version 2.0. Idag har mer än 80% av dagens Android enheter har stöd för version 2.0 (OpenGL ES versions 2012).

Standarden är fri från avgifter och har ett brett plattformsstöd. ES-versionen är dessutom anpassad för inbyggda system med mycket begränsad hårdvara, vilket gör den användbar för mobila enheter då både porteringsmöjligheter och prestanda är av vikt.

OpenGL är ett rent grafikbibliotek som låter programmeraren skapa geometriska figurer i form av punkter, linjer och polygoner, samt en bildyta för dessa. Det finns även avancerat stöd för ljussättning och skuggor, samt bildförbättring, så kallad anti-aliasing. OpenGL har dock inget stöd för inläsning av kontrollenheter, så som tangentbord eller mus, detta måste alltså utvecklaren lösa med något annat system (Segal & Akeley 2004).

3.2.5 Open Sound Library for Embedded System

För att spela upp ljud från intern kod i Android används Open Sound Library for Embedded Systems, OpenSL ES. Det är ett gränssnitt skapat för språket C, som från Android anropas via JNI eller med intern C/C++-kod. De flesta av OpenSL ES 1.0 funktioner är implementerade i Android 2.3 (Android 2.3 platform highlights 2012). På grund av detta valde SkyGoblin att endast rikta in sig mot Android 2.3 och senare.

OpenSL ES fungerar med hjälp av en ljudmotor som körs i bakgrunden och som sedan anropas med funktioner som antingen laddar in ljud av okomprimerad data, det vill säga Pulse-code modulation data (PCM data) för uppspelning eller någon av de filtyper som stöds, så som .mp3 eller .ogg, för dekomprimering och uppspelning. OpenSL ES stödjer förutom ren uppspelning av ljud och musik dessutom allehanda effekter som efterklang eller 3D-positionerat ljud (Khronos 2009).

3.3 Programmeringsspråk på Android

Förutom Java så kan Androidplattformen även köra C/C++ kod genom användandet av NDK. Nedan följer en kort beskrivning av hur dessa språk fungerar och hur de används på plattformen.

3.3.1 Java

För att uppnå plattformsoberoende kompileras inte program skrivna i Java till maskinkod som många andra språk, istället kompileras dessa till Javas egna maskinkod, vilka under körning tolkas av en virtuell maskin (Java Virtual Machine, JVM). Den virtuella maskinen är ett abstrakt system som inte förutsätter någon specifik hårdvara eller operativsystem. Eftersom alla system med en implementation av JVM kan använda samma mjukvara, är användandet av en virtuell maskin värdefullt då portabilitet mellan system är av stor vikt. Värt att notera är att JVM inte förutsätter att mjukvaran är skapad i Java, vilket språk som helst som kan generera en giltig .class-fil med maskinkodsinstruktioner fungerar (Oracle 2012).

Det faktum att en JVM alltid måste köras för att köra Java program gör att dessa generellt har något sämmre prestanda än program kompilerade direkt till maskin kod, så som C++. För förbättrad prestanda kan tolkningen av bytefilerna göras Just in Time (JIT).

För plattformen Android finns en speciell JVM kallad Dalvik som tillhandahålls av Google. Dalvik är optimerad för små lagringsutrymmen, då både Dalvik självt och de bytekodfiler som kompilerats för denna tar mindre plats än motsvarande filer för PC. Alla program på Android körs i en egen instans av Dalvik (what-is-Android? 2012).
I ett projekt för Android placeras alla källkodsfiler skrivna i Java i katalogen src. Alla resurser såsom bilder och ljudfiler som applikationen skall ha tillgång till placeras i någon av katalogerna assets eller res (förkortning av resources), och kan nås genom standardmetoder som getAssets().

3.3.2 Java Native Interface

För att visa hur Java Native Interface, JNI, fungerar följer här ett enkelt exempel:

C++ kod:

```
JNIEXPORT void JNICALL
Java_HelloWorld_print(JNIEnv *env, jobject obj)
{
printf("Hello World!\n");
return;
}
```

Java:

```
class HelloWorld {

static {
System.loadLibrary("HelloWorld!");
}
private native void print();
      public static void main(String[] args) {
print();
}
    }
```

Vid kompilering konstrueras ett bibliotek av C++-koden som sedan anropas med System.loadLibrary() i Java-koden. Metoden Print() deklareras i Java med nyckelordet native för att markera att koden för denna funktion finns i det inlästa biblioteket. Print() i Java anropar i sin tur Java_HelloWorld_print() som ligger i biblioteket libHelloWorld och skriver ut texten “Hello World!” (Liang and Safari Tech Books Online. 1999).

3.3.3 C++

På grund av snabbheten hos kompilerad kod, kombinerat med de kraftfulla möjligheterna hos de objektorienterade delarna i C++, är programspråket mycket populärt för att skapa mjukvara där prestanda är av största vikt, så som stora spel och program för vetenskapliga beräkningar och simuleringar. Detta gör att många program redan är skrivna i språket, och det är därför önskvärt att kunna köra samma kod på olika plattformar.

För Android projekt placeras källkodsfiler skrivna i C++ i katalogen jni. Att hantera resursfiler är något mer komplicerat i C++ än i Java, då inga standardiserade funktioner existerar för detta. En lösning som dock fungerar är att använda Java för att läsa in pekare till de resurser som finns, och sedan via JNI skicka vidare dessa till C++ koden. Värt att notera är att Android komprimerar alla typer av resurser för att spara på utrymme, vilket skapar problematik när man har flera resurser i en stor binärfil. För att undvika detta kan man ge binärfilen en filändelse som indikerar att den redan är komprimerad, exempelvis mp3 eller png; Android lämnar den då i oförändrat skick.

3.4 Make-filer

En så kallad make-fil är en textfil, vanligen med filändelsen .mk, som beskriver vilka källkodsfiler som ingår i ett projekt och hur de hänger samman. Här kan även målsystemets processor arkitektur definieras.

För att bygga ett projekt med native-kod till Android krävs två stycken makefiler, Android.mk och Application.mk. Android.mk beskriver vilka källkodsfiler som ingår samt hur dessa skall kompileras. Application.mk beskriver samtliga native moduler som ingår i projektet ("What-is-the-NDK?" 2012).

För ett fullständigt exempel på hur Android.mk och Application.mk kan se ut, se appendix 2 och 3.

3.5 Struktur av The Journey Down

Strukturen av The Journey Down är från början anpassad för att kunna porteras. Basen bygger på en motor (GobbyCore) som exekveras likadant på alla system. Motorn i sig kallar på initiering av abstrakta moduler som i sin tur refererar till systemspecifika implementationer av dessa moduler beroende på vilken miljö den exekveras i.

För att starta JD används en plattformsberoende initieringsklass. Under exekvering i Android startar en aktivitet som initierar nödvändiga Android-komponenter som grafik och touchfunktion. Aktiviteten anropar sedan GobbyCore genom JNI och därefter exekveras endast intern kod.

Varje abstrakt modul implementerar gränssnittet IModule med de abstrakta funktionerna init(), update() och stop(). Init() initierar de nödvändiga variablerna för modulen, update() anropas konstant på med korta intervall och stop() terminerar modulen.

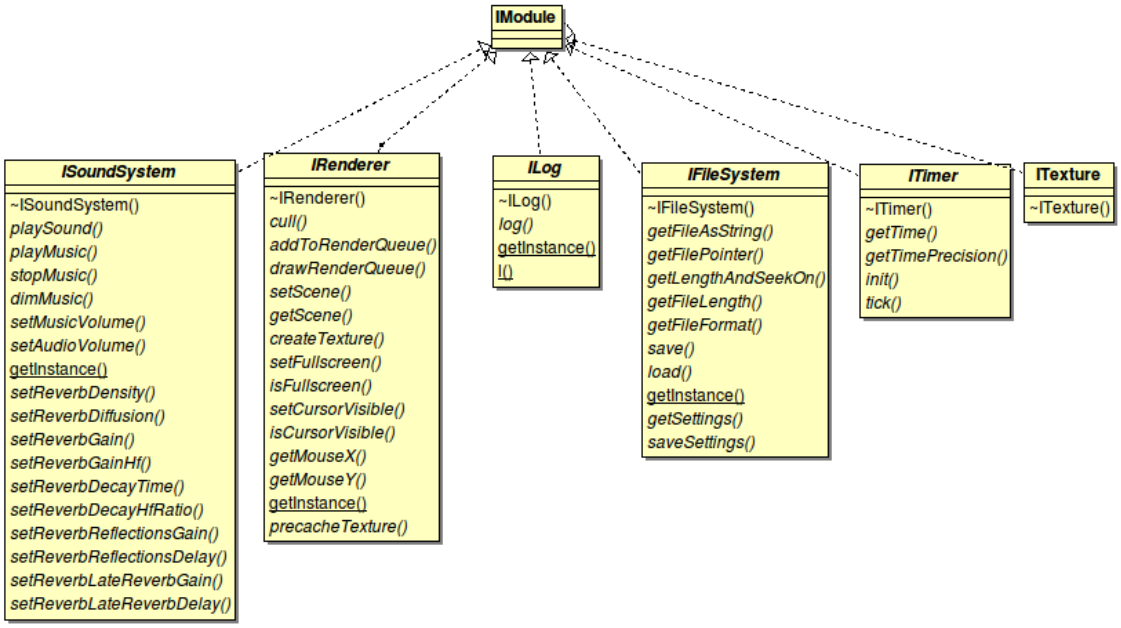
The Journey Down använder script-spåket AngelScript, ett språk vars syntax bygger på C, och från början är anpassat för plattformsoberoende. Varje rum i spelet har ett eget script tilldelat sig. Scriptet kallar på de abstrakta modulerna och deras funktioner så att de systemspecifika modulerna kan utföra önskad handling.

De olika modulerna är:

- **IRenderer** (Modul som renderar grafik.)
- **ISoundSystem** (Modul för dekodning och uppspelning av musik, ljud och tal. Allt ljud är i formatet Ogg Vorbis.)
- **IFileSystem** (Modul för att läsa filsystemet i The Journey Down.)
- **ITimer** (Modul dedikerad till att synkronisera de olika delarna av mjukvaran.)
- **ILog** (Modul som används för att skriva ut log-meddelanden under exekvering.)

Dessutom tillkommer en plattformspecifik modul vid namn **<plattform>TheoraPlayer** för uppspelning av video.

Hur de olika abstrakta modulerna hänger ihop kan åskådliggöras med följande UML diagram:



F 3. UML diagram över de olika abstrakta modulerna i The Journey Down

Alla resurser i spelet - ljud, musik, tal, grafik, video och script - är sammanfogade i en enda stor binär fil. Filen startar med ett index över alla resurser som finns i den och är sedan avgränsad med sökvägar som efterföljs av själva datan till motsvarande sökväg. Filen läses av med AndroidFileSystem som hämtar ut den binära datan och skickar den till modulen som anropat den.

Filen är packeterad med hjälp av ett fristående program vid namn GobEd som är konstruerat av SkyGoblin (SG) själva. Detta möjliggör en enkel förändring i datastrukturen genom att lägga till eller ta bort resurser och sedan packetera om filen med hjälp av programmet.

Till en början använde SG OpenGL som renderare. Mestadels eftersom det då teoretiskt sett skulle fungera mot alla målsystem gällande både desktop- och porterade versioner. Enligt SkyGoblin var det inte så i praktiken - många grafikkortsdrivrutiner för OpenGL är alldeles för buggiga för att det ska fungera utan problem, särskilt när det kommer till laptops med gamla drivrutiner i bundles som inte kan uppgraderas. Dessutom är även nya drivrutiner - särskilt mot grafikhårdvara från Intel - buggiga och i många fall helt oanvändbara gällande OpenGL. Det var helt enkelt inte en hållbar lösning. Därför används nu DirectX 9 som renderare när det kommer till PC-system.

The Journey Down (JD) backas upp av ett stort antal hjälpbibliotek för bland annat hantering av ljud- och bilddekodning och komprimering.

- **Angelscript** - scriptspråket
- **Freetype** - visning av text
- **Libpng** - dekodning av .png-filer
- **Ogg** - ljudcodec
- **Vorbis** - container för ogg
- **Vorbisfile** - för uppläsning av vorbisfiler
- **Theora** - filmcodec
- **TheoraDec** - dekodning av theorafiler
- **TheoraPlayer** - filmspelare för theorafiler, ljud sköts av ogg/vorbis
- **LibWebP** - dekodning av WebP-filer

Se appendix 2 för make-fil för projektet och dess tillhörande hjälpbibliotek.

4 GENOMFÖRANDE

Under projektet påbörjades en portering av spelet The Journey Down. Nedan följer en sammanfattning av de verktyg som använts samt en redogörelse för de problem som uppstått under utvecklingen.

4.1 Utvecklingsverktyg

Vi valde att använda Eclipse IDE tillsammans med de insticksmoduler nedan för att utveckla projektet. Eclipse IDE är en utvecklingsmiljö som har stöd för flera språk, det är också den officiella utvecklingsmiljön för Android-applikationer.

- **Android Software Development Kit**

Software Development Kit, SDK, är ett paket som möjliggör utveckling för Android. Det innehåller källkod och API för de existerande Android-versionerna på marknaden.

- **Android-NDK**

NDK är det plugin som möjliggör användandet av native kod i Android applikationer. Se kaptitel 3.2.3 för mer teknisk information.

- **Sequoyah**

Sequoyah är ett plugin till Eclipse som skapar en brygga mellan NDK och Eclipse och gör det därmed enklare att kompilera intern kod direkt i Eclipse. Speciellt om arbetet görs i en Windows-miljö, då man annars måste använda en Linux-lik terminal, till exempel Cygwin, för att kompilera med NDK.

- **CDT**

Användandet av CDT till Eclipse möjliggör utvecklandet av C/C++ applikationer i Eclipse miljön genom att skapa en speciell vy för detta.

- **DS-5**

Ett verktyg för att kunna felsöka C/C++-kod. Absolut nödvändigt vid skrivandet av applikationer under Android som använder sig av intern kod då det annars bara går att felsöka Java-kod.

4.2 Portering av The Journey Down

Under Android har filstrukturen fått namnet data.mp3. Detta beror på att Android komprimerar alla resurser så länge de inte redan är komprimerade - som .mp3 eller .png. Vi lurar alltså helt enkelt Android att tro att filen redan är komprimerad. På så sätt förstörs inte de sökvägar som anger vart i binärfilen de individuella filerna finns. Utan denna omväg hade Android komprimerat den binära filen och AndroidFileSystem hade inte kunnat komma åt resurserna då alla pekare hade varit fel.

För att portera filsystemet till Android behövdes endast mindre förändringar göras. I PC-versionen letar C++-koden själv upp datafilen. Vid utvecklandet av Android-versionen fick vi använda Java för att hämta ut filpekare och längd på filen för att sedan instansiera filsystemet med hjälp av dessa. Utöver detta kunde existerande kod användas utan förändringar.

Ljudsystemet på PC-versionen är skriven för OpenAL, något som gjorde företagets egna portering till iOS mycket enkel då det är något som stöds även av det operativsystemet. På Android finns däremot inget officiellt stöd för OpenAL, och tanken var därför att vi skulle utveckla ett eget system med hjälp av OpenSL ES 1.0.1 som finns implementerat i Android sedan version 2.3. Problem uppstod dock på grund av dålig dokumentation av OpenSL. Den enda officiella dokumentationen som fanns var en PDF-fil med diagram

över hur de olika klasserna sammarbetar. Dessutom är Androids implementation av version 1.0.1 inte fullständig, trots att version 1.1 funnits sedan länge.

För att lösa dessa problem använde vi en existerande open-source portering av OpenAL (OpenAL-soft) till Android, som i sin tur använder OpenSL. Efter kompilering fungerade då den existerande PC koden utan några förändringar. Nackdelen med detta är att vi måste förlita oss på ett utomstående projekt, vilket kan läggas ned utan förvarning. Vi måste dessutom följa deras licens, vilket i det här fallet råkade vara L-GPL version 3. Vi skapade ett delat bibliotek för OpenAL-soft och kunde på så sätt följa licensen (OpenAL Soft, 2012).

Android-versionen av JD är till skillnad mot PC-versionen fortfarande baserad på OpenGL, då rendering i OpenGL mot Android-system fungerar som tänkt. Detta kunde påvisas genom att köra exempelkod för OpenGL 2.0 som då renderade 3D-grafik på våran testenhet. På grund av oidentifierade fel lyckades vi aldrig visa upp någon grafik eller video genom spelets grafikmotor även om kompilering av tillhörande hjälpbibliotek lyckades.

Mycket av den felsökning vi gjort har varit direkt kopplad till kompilering av hjälpbibliotek och våran make-fil. Med en full kompileringstid på 3 minuter för hela projektet så approximerar vi att vi lagt 1 timme om dagen på att endast sitta och vänta på att kod ska kompileras.

5 RESULTAT

Utvecklingen av mobila enheter går fort framåt men de är fortfarande inte på samma nivå som dagens datorer, se sidan 4. Där man på PC har flera valmöjligheter gällande bibliotek för rendering av grafik finns det endast OpenGL ES att tillgå för Android, se sidan 6. Utbudet för tjänster gällande ljuduppspelning har heller inte samma stöd till Android, OpenSL ES är det system som finns tillgängligt, se sidan 7. Det är tydligt att utbud av underliggande system inte existerar i samma bredd till Android som till PC.

6 SLUTSATS

Det existerar ett flertal mobila plattformar på marknaden i dagsläget och dessa utvecklas i snabb takt. Detta gör att det blir allt viktigare för företag att utveckla till flera plattformar samtidigt. SkyGoblin är ett Göteborgsbaserat företag som utvecklat ett äventyrsspel med namnet “The Journey Down” med inriktning mot PC, Android och iOS. För att undersöka de problem som kan uppstå vid en portering av en högprestandaapplikation från dator till mobil plattform så påbörjades en portering av spelet mot Android. Ett flertal komponenter lyckades porteras med smärre problem men större oväntade problem uppkom också under tiden. Det visade sig att Android saknar bredden av stöd för underliggande system som PC innehar. Detta skapade problem under porteringsprocessen.

6.2 Kritisk diskussion

De system som erbjuds av Android innehåller all funktionalitet som ett relativt krävande spel som The Journey Down kräver. Dock existerade många problem som gjorde processen mer komplicerad än väntat. Då Android inte stödjer samma ljudsystem som många andra plattformar blir portering av just detta mer problematiskt än det borde vara.

På grund av att ljudsystemet på Android använde ett helt annat gränssnitt än det system som användes på PC-versionen uppkom ytterligare komplikationer. Det faktum att en tredjepartsportering av PC-versionens OpenAL-system existerade löste dock problemet genom att agera som en adapter mellan Androids OpenSL ES och källkodens efterfrågade OpenAL. Den här typen av lösningar kräver att man har kännedom om licenser och följer dessa. I vårat fall var detta inget problem då ett skapande av ett delat bibliotek löste detta.

Det som dock tog mest tid var att sätta sig in i make-filer och felsökning av dessa. För att portera mjukvara mellan system måste den kompileras plattformsspecifikt, och för att få korrekt syntax krävdes mycket jobb.

En hög inlärningskurva och brist på dokumentation skapar en onödigt tidskrävande porteringssprocess för utvecklare utan erfarenhet inom området. Detta kombinerat med de mindre problem som uppkom, så som att det inte existerar någon standardiserad metod för att undvika oönskad komprimering av resursfiler, gör att plattformen idag inte är mogen för portering av den här typen av applikationer.

6.2.1 Projektspecifika problem

Flera faktorer gjorde att våran implementation av JD tog längre tid än väntat. När vi fick koden för JD existerade ingen som helst dokumentation av programmet. Då det är ett mycket stort projekt (se make-filen i appendix 2 för en fullständig lista av alla filer) tog det lång tid innan vi var tillräckligt insatta för att kunna arbeta. En möjlig orsak till att implementering av grafik ej slutfördes kan också ha berott på att vi inte hade tillgång till tillräckligt kraftfull testhårdvara. Detta skulle egentligen SkyGoblin tillhandahållit.

6.2.2 Outforskade problem

Implementationen av grafiksystemet slutfördes aldrig, därför kunde inga mätningar på fps göras. På grund av detta kunde det heller aldrig visas att moderna mobila enheters hårdvara faktiskt stödjer högprestandaapplikationer. Plattformens möjligheter för portering av grafik kunde då heller aldrig utvärderas fullt ut.

Något annat som troligen är mycket plattformsspecifikt som inte undersöktes är användarens kontrollmöjligheter. Om en applikation från början designas enbart för att kontrolleras med mus och tangentbord behöver troligen mycket omarbetas för enheter med pekskärm. Detta var något som SkyGoblin arbetade mycket med vid deras portering av JD till iOS.

Eftersom vi aldrig färdigställde grafiksystemet kunde vi inte undersöka problem med aktiviteters livscykel. Detta är troligen något som kan vara ett problem vid portering då detta skiljer sig markant från PC. Det vore till exempel olyckligt ifall applikationen termineras när användaren får ett sms.

6.2.3 Alternativa undersökningsmöjligheter

Delvis förståelse för uppkommandet av problem kan fås av att endast läsa specifikationer. Man kan då teoretiskt sett se ganska klart att det finns begränsningar i hårdvara gällande mobila enheter jämfört med datorer. Man kan även förstå att begränsningar i valmöjlighet av mjukvarusystem kan skapa problem vid portering, framförallt om mjukvaran inte designats med detta i åtanke.

Dock existerar ytterligare, mindre uppenbara, problem. Hög inlärningskurva för kompilering av projektet i sig, samt de tredjepartsbibliotek som detta behöver, är exempel på detta. Det är därför viktigt med praktiskt erfarenhet av den här typen av projekt.

6.3 Generaliseringar

När applikationen från början designas anser vi att det är viktigt att efterforskningar görs gällande bibliotek som stöds av de tilltänkta plattformarna. Om bibliotek används som inte stöds på alla plattformar uppkommer problem, som till exempel i vårat fall med OpenAL. Om en plattform har för dåligt mjukvarustöd kanske det är mer kostnadseffektivt att strunta i att portera till just denna. Vid design av mjukvaran i fråga bör en grundlig genomgång av planering göras angående hur de målsystem man inriktar sig mot stödjer ljud, grafik och inläsning av filer för att minimera omskrivning av kod. Val av språk är även viktigt. Då Android i första hand använder Java och iOS använder Objective-C, men båda kan hantera C/C++ kod är troligen detta det bästa alternativet.

6.5 Fortsatt forskning

The Journey Down är skapat i C++, vilket skapade vissa problem vid porteringen. Ett program från början skapat i Java ger sannolikt en annan uppsättning problem, men möjligen en enklare porteringsprocess överlag då Android till stor del bygger på Java. Därför vore det intressant att undersöka porteringsmöjligheter av stora Java applikationer från PC till Android.

7 REFERENSER

Activity lifecycle (2012). Retrieved 30 May, 2012, from
http://developer.android.com/images/activity_lifecycle.png

Ahlund, A. (2010, 30 October). "Top 30 Android Apps Of All Time." Retrieved 5 May, 2012, from
http://techcrunch.com/2010/10/30/top-30-android-apps/

Android 2.3 platform highlights (2012). Retrieved 31 May, 2012, from
http://developer.android.com/sdk/android-2.3-highlights.html

Frame rate (2012), Retrieved 30 May, 2012, from
http://en.wikipedia.org/wiki/Frame_rate

Khronos (2009, 24 September). “OpenSL ES Specification 1.0.1.” Retrieved 31 May, 2012, from
http://www.khronos.org/registry/sles/specs/OpenSL_ES_Specification_1.0.1.pdf

Krewell, K. (2009, 16 December). “What's the difference between a CPU and a GPU?” Retrieved 30 May, 2012, from
http://blogs.nvidia.com/2009/12/whats-the-difference-between-a-cpu-and-a-gpu/

Liang, S. and Safari Tech Books Online. (1999). The Java Native interface programmer's guide and specification. Reading, Mass., Addison-Wesley

"Moore's-Law" (2012, 2 May). "Moore's law." Retrieved 5 May, 2012, from
http://en.wikipedia.org/wiki/Moores_law

OpenAL Soft (2012). “LGPLv3 license.” Retrieved 30 May, 2012, from
https://github.com/apportableinc/openal-soft/blob/master/COPYING

OpenGL ES versions (2012). Retrieved 30 May, 2012, from
http://developer.android.com/resources/dashboard/opengl.html

OpenHandsetAlliance (2007, November). "FAQ." Retrieved 5 May, 2012, from
http://www.openhandsetalliance.com/oha_faq.html

Oracle (2012). “Java Virtual Machine Specification.” Retrieved 30 May, 2012, from
http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-1.html

Overview (2012). Retrieved 5 May, 2012, from
http://developer.android.com/sdk/ndk/overview.html

Ployhar, M. (2010, 26 August). "Smart/iPhones – a convergence hub device? I believe so.". Retrieved 5 May, 2012, from
http://software.intel.com/en-us/blogs/2010/08/26/smartiphones-a-convergence-hub-device-i-believe-so/

Rose, M. (2012, 31 May). “*The Journey Down HD* hopes to float in a sea of big adventure game titles”. Retrieved 31 May, 2012, from
http://gamasutra.com/view/news/171091/The_Journey_Down_HD_hopes_to_float_in_a_sea_of_big_adventure_game_titles.php

Segal & Akeley (2004). “The OpenGL Graphics System: A Specification.” Retrieved 30 May, 2012, from
http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf

Shimpi, A. L. (2012, 19 April). "NVIDIA Plots Mobile SoC GPU Performance, Surpassing Xbox 360 by 2014." Retrieved 5 May, 2012, from http://www.anandtech.com/show/5762/nvidia-plots-mobile-soc-gpu-performance-surpassing-xbox-360-by-2014

Sony (2012). “Xperia P specifikationer” Retrieved 30 May, 2012, from http://www.sonymobile.com/se/products/phones/xperia-p/specifications/

System architecture (2012). Retrieved 30 May, 2012, from http://developer.android.com/images/system-architecture.jpg

Turner, D. (2009, 25 June). “Introducing Android 1.5 NDK, Release 1.“ Retrieved 12 June, 2012, from http://android-developers.blogspot.se/2009/06/introducing-android-15-ndk-release-1.html

What-is-Android? (2012). Retrieved 5 May, 2012, from http://developer.android.com/guide/basics/what-is-android.html

"What-is-the-NDK?" (2012). Retrieved 5 May, 2012, from http://developer.android.com/sdk/ndk/overview.html

8 APPENDIX

1. Genomgång av NDK

- libc (C library) headers
Standardbiblioteket för språket C. Det består av makron, typdefinitioner och alla de funktioner som finns med för C - exempelvis matematiska, minnesallokerande, sträng- och in/ut-hanteringsfunktioner.
(http://cplusplus.com/reference/clibrary/)
- libm (math library) headersDavid Turner
I vissa system läggs vissa tyngre och mer avancerade matematiska funktioner separat i libm istället för i libc, och så är också fallet här.
(http://bytes.com/topic/c/answers/811741-what-does-libm-contain)
- JNI interface headers
JNI är ett ramverk som möjliggör för Javakod som exekveras i en JVM att anropa och bli anropas från interna applikationer. Med andra ord kan man både anropa C/C++-kod inifrån Java och tvärtom.
(http://en.wikipedia.org/wiki/Java_Native_Interface)
- libz (Zlib compression) headers
Libz är ett bibliotek som använder sig av zlib. Zlib är ett datakompressions-bibliotek som är portabelt över olika plattformar.
(http://freeware.sgi.com/Installable/libz-1.1.4.html) (http://zlib.net/)
- liblog (Android logging) header
För att effektivt kunna skriva ut log-meddelanden under körning av intern kod används liblog.
(http://sourceforge.net/projects/liblog/)
- OpenGL ES 1.1 och OpenGL ES 2.0 (3D graphics libraries) headers
OpenGL ES är ett låg-level API baserat på Desktop OpenGL ämnat för 2D/3D-grafik. OpenGL ES är riktat mot mobiler, bilar och andra portabla mindre system snarare än mot en kraftfull dator.
(http://www.khronos.org/opengles/)
- libjnigraphics (Pixel buffer access) header (för Android 2.2 och högre).
- OpenSL ES native audio libraries
Ett hårdvaruaccelererat ljud-API i native miljö för mobila system. Det möjliggör en mindre latens för ljuduppspelning än Androids egna ljud-APIn.
(http://www.khronos.org/opensles/)
- Ett minimalt set med headers för C++-support
Stödet för C är komplett och stödet för C++ är växande.
- Android native application APIS
Native APIs för olika versioner av Android. De existerande native APIs som finns vid skrivande stund är mot Android-1.5, 1.6, 2.0, 2.0.1, 2.1, 2.2, 2.3 och 4.0.
(http://mobilepearls.com/labs/native-android-api/STABLE-APIS.html)
(http://developer.android.com/sdk/ndk/overview.html)

2. Android.mk för The Journey Down

```
MY_LOCAL_PATH := $(call my-dir)

#############################################################
## Angelscript
#############################################################

include $(CLEAR_VARS)

LOCAL_CFLAGS    := -g

commonSources:= as_callfunc_arm_gcc.S as_atomic.cpp as_builder.cpp as_bytecode.cpp
as_callfunc.cpp as_callfunc_arm.cpp as_callfunc_mips.cpp as_callfunc_ppc.cpp
as_callfunc_ppc_64.cpp as_callfunc_sh4.cpp as_callfunc_x86.cpp as_callfunc_x64_gcc.cpp
as_compiler.cpp as_context.cpp as_configgroup.cpp as_datatype.cpp as_generic.cpp as_gc.cpp
as_globalproperty.cpp as_memory.cpp as_module.cpp as_objecttype.cpp as_outputbuffer.cpp
as_parser.cpp as_restore.cpp as_scriptcode.cpp as_scriptengine.cpp as_scriptfunction.cpp
as_scriptnode.cpp as_scriptobject.cpp as_string.cpp as_string_util.cpp as_thread.cpp
as_tokenizer.cpp as_typeinfo.cpp as_variablescope.cpp
LOCAL_PATH:= $(MY_LOCAL_PATH)/../../Common/angelscript/angelscript/source

LOCAL_SRC_FILES:= $(commonSources)
LOCAL_MODULE:= libangelscript
include $(BUILD_STATIC_LIBRARY)


#############################################################
## GobbyCore
#############################################################

include $(CLEAR_VARS)

LOCAL_PATH := $(MY_LOCAL_PATH)
LOCAL_MODULE    := GobbyCore

LOCAL_SRC_FILES := ../../GobbyCore/core/AbstractRenderer.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/AnimatedSprite.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/AnimationFrame.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/Core.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/Drawable.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/Input.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/controllers/MoveTo.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/MemoryLog.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/Pathfinder.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/Polygon.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/RCBase.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/Scene.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/SceneElement.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/SceneNode.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/SceneModule.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/ScriptModule.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/scriptmath/scriptmath.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/scriptmath/scriptmathcomplex.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/controllers/Sequential.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/controllers/AlphaNoise.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/controllers/AlphaOscillate.cpp
```

```
LOCAL_SRC_FILES += ../../GobbyCore/core/controllers/FadeTo.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/controllers/ScaleTo.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/StandardAnimation.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/State.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/StaticSprite.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/TextBlock.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/Texts.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/TintPoint.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/IFileSystem.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/ILog.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/IRenderer.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/ISoundSystem.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/scriptarray/scriptarray.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/scriptbuilder/scriptbuilder.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/scriptstdstring/scriptstdstring.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/scriptstdstring/scriptstdstring_utils.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/internalJSONNode.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONAllocator.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSON_Base64.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONChildren.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONDebug.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONIterators.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONMemory.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONNode.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONNode_Mutex.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONPreparse.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONStream.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONValidator.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONWorker.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/JSONWriter.cpp
LOCAL_SRC_FILES += ../../GobbyCore/core/libjson/libjson.cpp

LOCAL_C_INCLUDES := $(LOCAL_PATH)/../../Common/angelscript/angelscript/include
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../../GobbyCore
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../../Common/glm-0.9.3.2
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../../Common/libjson/Source

LOCAL_CFLAGS    := -g -fexceptions -DANDROID

include $(BUILD_STATIC_LIBRARY)

############################################################
## LibPNG
############################################################
include $(CLEAR_VARS)

LOCAL_PATH:= $(MY_LOCAL_PATH)/../../Common/lpng154
LOCAL_MODULE    := png

LOCAL_SRC_FILES := png.c
LOCAL_SRC_FILES += pngerror.c
LOCAL_SRC_FILES += pngget.c
LOCAL_SRC_FILES += pngmem.c
LOCAL_SRC_FILES += pngpread.c
LOCAL_SRC_FILES += pngread.c
LOCAL_SRC_FILES += pngrio.c
LOCAL_SRC_FILES += pngrtran.c
```

```
LOCAL_SRC_FILES += pngrutil.c
LOCAL_SRC_FILES += pngset.c
LOCAL_SRC_FILES += pngtest.c
LOCAL_SRC_FILES += pngtrans.c
LOCAL_SRC_FILES += pngvalid.c
LOCAL_SRC_FILES += pngwio.c
LOCAL_SRC_FILES += pngwrite.c
LOCAL_SRC_FILES += pngwtran.c
LOCAL_SRC_FILES += pngwutil.c

LOCAL_C_INCLUDES := $(MY_LOCAL_PATH)/../../Common/lpng154
LOCAL_CFLAGS    := -g

include $(BUILD_STATIC_LIBRARY)


############################################################
## TheoraPlayer
############################################################
include $(CLEAR_VARS)

LOCAL_PATH:= $(MY_LOCAL_PATH)/../../Common/libtheoraplayer/src
LOCAL_MODULE    := theoraplayer

LOCAL_SRC_FILES := TheoraAsync.cpp
LOCAL_SRC_FILES += TheoraAudioInterface.cpp
LOCAL_SRC_FILES += TheoraDataSource.cpp
LOCAL_SRC_FILES += TheoraException.cpp
LOCAL_SRC_FILES += TheoraFrameQueue.cpp
LOCAL_SRC_FILES += TheoraTimer.cpp
LOCAL_SRC_FILES += TheoraUtil.cpp
LOCAL_SRC_FILES += TheoraVideoClip.cpp
LOCAL_SRC_FILES += TheoraVideoFrame.cpp
LOCAL_SRC_FILES += TheoraVideoManager.cpp
LOCAL_SRC_FILES += TheoraWorkerThread.cpp

LOCAL_C_INCLUDES := $(MY_LOCAL_PATH)/../../Common/libtheoraplayer/src $
(MY_LOCAL_PATH)/../../Common/libtheoraplayer/include/theoraplayer $
(MY_LOCAL_PATH)/../../Common/libtheora-1.1.1/include/ $
(MY_LOCAL_PATH)/../../Common/libogg-1.3.0/include $
(MY_LOCAL_PATH)/../../Common/libvorbis-1.3.2/include/ $
(MY_LOCAL_PATH)/../../Common/libtheoraplayer/include/
LOCAL_CFLAGS    := -g -fexceptions

include $(BUILD_STATIC_LIBRARY)

############################################################
# Sound and video
############################################################

LOCAL_PATH:= $(MY_LOCAL_PATH)/../../Common/apportableinc-openal-soft/libs/armeabi

include $(CLEAR_VARS)

LOCAL_MODULE := openal
LOCAL_SRC_FILES := libopenal.so
LOCAL_C_INCLUDES := $(LOCAL_PATH)/../jni/include
```

```
include $(PREBUILT_SHARED_LIBRARY)

LOCAL_PATH:= $(MY_LOCAL_PATH)/../lib/armeabi

include $(CLEAR_VARS)

LOCAL_MODULE    := ogg
LOCAL_SRC_FILES := libogg.a
LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/../jni/include

include $(PREBUILT_STATIC_LIBRARY)


include $(CLEAR_VARS)

LOCAL_MODULE    := vorbis
LOCAL_SRC_FILES := libvorbis.a
LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/../jni/include

include $(PREBUILT_STATIC_LIBRARY)


include $(CLEAR_VARS)

LOCAL_MODULE    := vorbisfile
LOCAL_SRC_FILES := libvorbisfile.a
LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/../jni/include

include $(PREBUILT_STATIC_LIBRARY)

include $(CLEAR_VARS)

LOCAL_MODULE    := theora
LOCAL_SRC_FILES := libtheora.a
LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/../jni/include

include $(PREBUILT_STATIC_LIBRARY)

include $(CLEAR_VARS)

LOCAL_MODULE    := theoradec
LOCAL_SRC_FILES := libtheoradec.a
LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/../jni/include

include $(PREBUILT_STATIC_LIBRARY)

############################################################
# Freetype
############################################################

include $(CLEAR_VARS)
LOCAL_PATH := $(MY_LOCAL_PATH)/../../Common/freetype2/src

LOCAL_MODULE := freetype

LOCAL_CFLAGS := -DANDROID_NDK -DFT2_BUILD_LIBRARY=1
```

```
LOCAL_C_INCLUDES := $(LOCAL_PATH)/../include

LOCAL_SRC_FILES := autofit/autofit.c
LOCAL_SRC_FILES += base/basepic.c
LOCAL_SRC_FILES += base/ftapi.c
LOCAL_SRC_FILES += base/ftbase.c
LOCAL_SRC_FILES += base/ftbbox.c
LOCAL_SRC_FILES += base/ftbitmap.c
LOCAL_SRC_FILES += base/ftdbgmem.c
LOCAL_SRC_FILES += base/ftdebug.c
LOCAL_SRC_FILES += base/ftglyph.c
LOCAL_SRC_FILES += base/ftinit.c
LOCAL_SRC_FILES += base/ftpic.c
LOCAL_SRC_FILES += base/ftstroke.c
LOCAL_SRC_FILES += base/ftsynth.c
LOCAL_SRC_FILES += base/ftsystem.c
LOCAL_SRC_FILES += cff/cff.c
LOCAL_SRC_FILES += pshinter/pshinter.c
LOCAL_SRC_FILES += psnames/psnames.c
LOCAL_SRC_FILES += raster/raster.c
LOCAL_SRC_FILES += sfnt/sfnt.c
LOCAL_SRC_FILES += smooth/smooth.c
LOCAL_SRC_FILES += truetype/truetype.c

LOCAL_C_FLAGS := -g

include $(BUILD_STATIC_LIBRARY)

############################################################
# LibWebP
############################################################
include $(CLEAR_VARS)

LOCAL_PATH:= $(MY_LOCAL_PATH)/../../Common/libwebp-0.1.3/src
LOCAL_MODULE    := webp

LOCAL_SRC_FILES := dec/alpha.c
LOCAL_SRC_FILES += dec/frame.c
LOCAL_SRC_FILES += dec/idec.c
LOCAL_SRC_FILES += dec/layer.c
LOCAL_SRC_FILES += dec/quant.c
LOCAL_SRC_FILES += dec/tree.c
LOCAL_SRC_FILES += dec/vp8.c
LOCAL_SRC_FILES += dec/webp.c
LOCAL_SRC_FILES += dec/io.c
LOCAL_SRC_FILES += dec/buffer.c
LOCAL_SRC_FILES += dsp/yuv.c
LOCAL_SRC_FILES += dsp/upsampling.c
LOCAL_SRC_FILES += dsp/cpu.c
LOCAL_SRC_FILES += dsp/dec.c
LOCAL_SRC_FILES += dsp/dec_neon.c
LOCAL_SRC_FILES += dsp/enc.c
LOCAL_SRC_FILES += utils/bit_reader.c
LOCAL_SRC_FILES += utils/bit_writer.c
LOCAL_SRC_FILES += utils/thread.c
LOCAL_SRC_FILES += ../swig/libwebp_java_wrap.c
```

```
LOCAL_C_INCLUDES := $(MY_LOCAL_PATH)/../../Common/libwebp-0.1.3/src/webp $
(MY_LOCAL_PATH)/../../Common/libwebp-0.1.3/src/

LOCAL_CFLAGS := -g

include $(BUILD_STATIC_LIBRARY)

############################################################
## GobbyAndroid
############################################################
include $(CLEAR_VARS)

LOCAL_PATH := $(MY_LOCAL_PATH)

LOCAL_MODULE    := GobbyAndroid

LOCAL_SRC_FILES := AndroidFileSystem.cpp
LOCAL_SRC_FILES += AndroidLog.cpp
LOCAL_SRC_FILES += AndroidRenderer.cpp
LOCAL_SRC_FILES += AndroidSoundSystem.cpp
LOCAL_SRC_FILES += AndroidTexture.cpp
LOCAL_SRC_FILES += AndroidTimer.cpp
LOCAL_SRC_FILES += GobbyAndroid.cpp
LOCAL_SRC_FILES += AndroidTextFactory.cpp
LOCAL_SRC_FILES += ScriptCallback.cpp
LOCAL_SRC_FILES += AndroidTheoraPlayer.cpp
LOCAL_SRC_FILES += AndroidText.cpp
LOCAL_SRC_FILES += tinythread.cpp

LOCAL_C_INCLUDES := $(LOCAL_PATH)/../../Common/angelscript/angelscript/include
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../../GobbyCore
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../../Common/glm-0.9.3.2
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../../Common/libjson/Source
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../../Common/lpng154
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../include
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../../Common/libwebp-0.1.3
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../../Common/libwebp-0.1.3/src/webp
LOCAL_C_INCLUDES += $(LOCAL_PATH)/../include/freetype


LOCAL_STATIC_LIBRARIES := GobbyCore angelscript freetype webp png theoraplayer theora
theoradec vorbisfile vorbis ogg
LOCAL_SHARED_LIBRARIES := openal
LOCAL_LDLIBS := -landroid -llog -lEGL -lGLESv1_CM -lz -lOpenSLES -lGLESv2
-DANDROID

LOCAL_CFLAGS    := -g

include $(BUILD_SHARED_LIBRARY)
```

3. Application.mk för The Journey Down

```
APP_ABI := armeabi-v7a
APP_STL := gnustl_static
APP_MODULES := angelscript GobbyCore png GobbyAndroid
```