Eetu Oinasmaa

3D-MOBIILIPELIMOOTTORIN KEHITTÄMINEN

Opinnäytetyö
Kajaanin ammattikorkeakoulu
Tradenomi
Tietojenkäsittely
Syksy 2014

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Luonnontieteet	Koulutusohjelma Tietojenkäsittely
Tekijä(t) Eetu Oinasmaa	
Työn nimi 3D-mobiilipelimoottorin kehittäminen	
Vaihtoehtoiset ammattiopinnot	Toimeksiantaja
Aika Syksy 2014	Sivumäärä ja liitteet 49

Opinnäytetyön tavoitteena oli 3D-mobiilipelimoottorin suunnittelu ja toteutus. Työssä esitellään pelimoottorit yleisesti ja niiden historiaa. Lisäksi työssä kuvataan mobiilialustojen ominaispiirteet ja mobiilipelimoottorin kehityksessä huomioitavia seikkoja. Tyypillisen 3D-pelimoottorin rakenne käydään läpi ennen suunnittelusta ja toteutuksesta kertovaa osiota.

Pelimoottorin suunnittelussa tuli hyödyntää kaupallisissa pelimoottoreissa käytettyä arkkitehtuuria sekä suurissa ohjelmistojärjestelmissä käytettyjä rakenteita ja malleja. Pelimoottorin oli tarkoitus tukea vain yhtä alustaa, mutta mahdollisesta uusien alustojen tukemisesta tuli tehdä vaivatonta. Pelimoottoriin tuli toteuttaa keskeisimmät alijärjestelmät ja komponentit, kuten renderöinti- ja äänijärjestelmä. Lopuksi pelimoottorilla tuli toteuttaa testisovellus, jolla sen toimintaa voitiin testata.

Pelimoottori suunniteltiin alijärjestelmittäin ennen sen toteuttamista, ja se toteutettiin alijärjestelmä kerrallaan. Toteutettuihin alijärjestelmiin kuuluivat alustariippumattomuus-, ydin-, resurssienhallinta-, renderöinti-, käyttäjäsyöte- ja äänijärjestelmä. Pelimoottorissa käytettiin myös kolmannen osapuolen ohjelmistokehityspaketteja ja ohjelmointikirjastoja. Pelimoottoria testattiin yksikkötestien ja testisovelluksen avulla.

Pelimoottori onnistuttiin kehittämään suunnitelman mukaisesti. Sitä jatkokehitetään opinnäytetyövaiheen jälkeen tehokkaammaksi pelinkehitystyökaluksi lisäjärjestelmien ja -komponenttien avulla.

Kieli	Suomi
Asiasanat	Pelimoottori, 3D, mobiili
Säilytyspaikka	☒ Verkkokirjasto Theseus ☐ Kajaanin ammattikorkeakoulun kirjasto

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

THESIS ABSTRACT

School	Degree Programme
Business	Business Information Technology

Author(s)
Eetu Oinasmaa

Title
Developing a 3D Mobile Game Engine

Optional Professional Studies	Commissioned by

Date	Total Number of Pages and Appendices
Autumn 2014	49

The objective of the thesis was to design and implement a 3D mobile game engine. The thesis introduces game engines in general and provides some history. Also, it describes the properties of mobile platforms and matters to consider in mobile game engine development. The structure of a typical 3D game engine is presented before the design and implementation sections.

The architecture used in commercial game engines and the structures and patterns used in large software systems were to be utilised in the design of the game engine. The game engine had to support a single platform, but supporting new platforms had to be effortlessly possible. Crucial subsystems and components, such as rendering and audio systems, had to be implemented in the game engine. Finally, a test application had to be developed to test the functionalities of the game engine.

Before implementation, the game engine was designed from the perspective of subsystems. It was implemented a subsystem at a time, and the implemented subsystems were platform independence, core, resource management, rendering, human interface device and audio. The game engine also utilised third party software development kits and programming libraries. Unit tests and the test application were used to test the game engine.

The game engine was successfully implemented as planned, and the development will continue after the thesis. With additional subsystems and components, the game engine will be made into a more powerful game development tool.

Language of Thesis	Finnish
Keywords	Game engine, 3D, mobile
Deposited at	☒ Electronic library Theseus ☐ Library of Kajaani University of Applied Sciences

ALKUSANAT

Kiitos Murulle aiheesta ja tuesta sekä Vimpulalle jalkojen lämmittämisestä

SISÄLLYS

1 JOHDANTO 1

2 PELIMOOTTORI 2

2.1 Historiaa 3

2.2 Mobiilialustat 4

3 RAKENNE 6

3.1 Aktiiviosa 6

3.2 Passiiviosa 18

4 TAVOITE JA SUUNNITTELU 21

5 TOTEUTUS 23

5.1 Alusta 23

5.2 Kehitystyökalut 24

5.3 Projektin rakenne 25

5.4 Ohjelmistokehityspaketit 26

5.5 Alustariippumattomuus 27

5.6 Ydin 28

5.7 Resurssienhallinta 35

5.8 Renderöinti 36

5.9 Käyttäjäsyötteet 42

5.10 Äänet 43

6 TESTAUS 46

7 POHDINTA 47

LÄHTEET 48

SYMBOLILUETTELO

Alusta	Englanniksi platform Laitteiston, ajurien ja käyttöjärjestelmän muodostama kokonaisuus
Assembly	Prosessoriarkkitehtuurikohtainen matalan tason ohjelmointikieli
Mobiili	Kannettavuus; tietotekninen mobiililaite on tietokoneeseen rinnastettava pienikokoinen kannettava laite, kuten matkapuhelin, taulutietokone tai sulautettu järjestelmä.
Kehitysympäristö	Ohjelma tai ohjelmisto, jolla suunnitellaan ja toteutetaan ohjelmia.
Kontekstinvaihto	Englanniksi context switch Prosessi, jossa ohjelmaprosessin tai säikeen tila tallennetaan ja palautetaan, jotta useampi ohjelmaprosessi voisi käyttää yksittäistä prosessoria. Kontekstinvaihto on yleensä hidas operaatio.
Mesh	Verteksien muodostama kolmiulotteinen monitahokas, joka määrittelee kolmiulotteisen mallin muodon.
Natiivi kieli	Ohjelmointikieli, joka käännetään suoritettavaksi tietyllä prosessoriarkkitehtuurilla.
Ohjelmistokehys	Englanniksi software framework Ohjelmistokokoelma, joka toimii runkona kehitettäville ohjelmille ja tarjoaa valmiita toimintoja niiden käytettäväksi.
Ohjelmistonrakennus	Englanniksi software build Prosessi, jossa ohjelman lähdekoodi käännetään ja linkitetään suoritettavaksi ohjelmatiedostoksi tai -kirjastoksi.

Ohjelmointirajapinta	Englanniksi (application) programming interface Määrittelee, kuinka objektit toimivat itsenäisesti ja toistensa kanssa sekä tarjoaa funktioita ja luokkia toimintojen käyttämiseksi.
Renderöinti	Prosessi, jossa digitaalinen kaksiulotteinen kuva luodaan kolmiulotteisesta datasta tietokoneohjelman avulla.
Scene-graafi	Englanniksi scene graph Tietorakenne objektien ominaisuuksien ja suhteiden kuvaamiseen; käytetään erityisesti kolmiulotteisten objektien kanssa.
Skripti	Englanniksi script Tulkittava koodi, jolla voidaan muuttaa ohjelman toimintaa jopa sen ajon aikana.
Sprite	Kaksiulotteinen renderöitävä kuva, jota käytetään erityisesti peleissä.
Varjostin	Englanniksi shader Näytönohjaimessa suoritettava ohjelma, joka määrittelee, kuinka kuva renderöidään.
Verteksi	Englanniksi vertex Geometrisen kuvion kulmapiste ja kolmiulotteisen mallin pienin yksittäinen komponentti
Virtuaalikone	Englanniksi virtual machine Ohjelmistopohjainen tietokone, joka emuloi oikeaa tai hypoteettista tietokonetta.
Wrapper	Rajapinta, joka piilottaa toisen rajapinnan toiminnot taakseen. Wrapperin avulla voidaan esimerkiksi yhtenäistää toistensa kanssa yhteensopimattomia rajapintoja.

1 JOHDANTO

Pelejä kehitetään nykyään harvoin alusta asti itse. Pelinkehitystä helpotetaan ja nopeutetaan pelimoottorilla tai pelinkehitykseen sopivalla ohjelmistokehyksellä. Modernit pelimoottorit ovat tehokkaita ja helposti lähestyttäviä helppokäyttöisine työkaluineen. Useimmat tukevat yleisimpiä alustoja työpöytäkäyttöjärjestelmistä pelikonsoleihin ja mobiililaitteisiin. Jopa laadukkaat ilmaiseksi käytettävät pelimoottorit saattavat kyseenalaistaa tarpeellisuuden kehittää oma pelimoottori.

Monet peliyritykset kehittävät omia pelimoottoreita saadakseen kohdealustoista kaiken suorituskyvyn irti pelejään varten. Pelimoottorin kehittäminen on hyvä tapa kehittää ohjelmointitaitoja ja tietämystä ohjelmistoarkkitehtuureista, vaikka pelimoottoria ei kehitettäisikään tuotantotasoiseksi. Tehokkaimmat tuotantotasoiset pelimoottorit kehitetään edelleen suurimmaksi osaksi C++:lla tai vastaavalla natiivilla ohjelmointikielellä.

Mobiililaitteet ovat kehittyneet valtavasti viime vuosien aikana. Laadukkaat näytöt ja kaiuttimet mahdollistavat näyttävien ja viihdyttävien pelien kehittämisen. Mobiililaitteet muistuttavat yhä enemmän työpöytätietokoneita laitteisto- ja ohjelmistoarkkitehtuureiltaan sekä suorituskyvyltään. Siksi esimerkiksi pelisovellukset on helpompi toteuttaa työpöytätietokoneella ja mobiililaitteella toimivaksi lähestulkoon samalla lähdekoodilla.

Opinnäytetyön lukijan on hyvä tietää, kuinka tietokoneohjelmaa suoritetaan. Työssä oletetaan, että lukijalla on kokemusta C++:sta ja pelinkehityksestä. OpenGL:n tai vastaavan grafiikkarajapinnan tunteminen on myös eduksi. Opinnäytetyössä käytetään termejä pelimoottorin käyttäjä ja pelaaja. Pelimoottorin käyttäjällä tarkoitetaan pelimoottorin kehittäjää tai pelinkehittäjää. Pelaaja on pelimoottorin avulla luodun pelin käyttäjä eli pelimoottorin loppukäyttäjä.

2 PELIMOOTTORI

Digitaalipelin lähdekoodi sisältää pelikohtaisen logiikan lisäksi tietorakenteita, algoritmeja ja alustakohtaisia toimintoja. Samoja tietorakenteita ja algoritmeja voidaan käyttää erilaisissa peleissä yhä uudestaan. Alustakohtaiset toiminnot, kuten tiedostojen ja käyttäjäsyötteiden käsittely, renderöinti sekä äänentoisto, ovat nimensä mukaisesti kohdealustariippuvaisia. Yleisesti käytettyjen tietorakenteiden, algoritmien ja alustakohtaisten toimintojen pohjalta voidaan kehittää pelimoottori, jonka päälle varsinainen peli kehitetään. Pelimoottori on uudelleenkäytettävissä pelinkehityksessä, eikä sen tarjoamia toimintoja tarvitse toteuttaa jokaista peliä varten uudestaan. Pelimoottorin tarjoamat toiminnot eivät kuitenkaan rajoitu vain edellä mainittuihin, vaan niihin kuuluu lisäksi animointia, fysiikan mallinnusta ja tekoälyä. (Bhattacharya, Goon & Paul 2012; Gregory 2014, 3; Gregory 2014, 11.)

Pelimoottori on ohjelmistokehys, koska se tarjoaa valmiita toimintoja pelinkehitykseen eikä toimi sovelluksena sellaisenaan. Jotkin pelinkehitykseen erikoistuneet ohjelmistokehykset eivät välttämättä ole pelimoottoreita, vaikka peliohjelmistokehyksen ja pelimoottorin eroja ei ole virallisesti määritelty. Voidaan kuitenkin ajatella, että pelimoottori sisältää kehittyneempiä toimintoja ja pelisovelluskehys vain minimaalisen pelin kehittämiseen tarvittavat toiminnot. (Gregory 2014, 343–344; Ward 2008.)

Yleensä pelimoottoreissa käytetään datakeskeistä ohjelmistoarkkitehtuuria, jossa ohjelman suoritus riippuu enimmäkseen sille syötetystä datasta kovakoodatun logiikan sijaan. Tämä mahdollistaa pelimoottorin uudelleenkäytettävyyden ja erottaa pelimoottorin puhtaasta pelisovelluksesta. Kuitenkin raja, johon pelimoottori loppuu ja josta peli alkaa, voi olla häilyvä. Pelimoottoreita on erilaisia, koska kaikenlaisten pelien kehitykseen soveltuvaa pelimoottoria on lähes mahdoton toteuttaa. Esimerkiksi ensimmäisen persoonan ammuntapelien kehitykseen erikoistunut pelimoottori ei välttämättä sovellu massiivisen verkkomoninpelin kehitykseen, koska näiden lajityyppien ominaispiirteet ovat hyvinkin erilaisia. Ensimmäisen persoonan ammuntapeleissä suositaan realistista ja korkeatasoista renderöintiä, kun taas massiivisissa verkkomoninpeleissä panostetaan pelimekaniikkaan ja verkkotoimintoihin. (Gregory 2014, 11–24.)

Nykyään monet pelimoottorit tukevat useaa alustaa, kuten eri PC-käyttöjärjestelmiä, pelikonsoleita ja mobiilikäyttöjärjestelmiä. Tällöin pelimoottorissa toteutetaan tuettujen alustojen

alustakohtaiset toiminnot ja piilotetaan niiden yksityiskohdat käyttäjältä. Alustakohtaisiin toimintoihin pääsee käsiksi yhtenäisen rajapinnan kautta, jolloin käyttäjän ei tarvitse välttämättä tietää toteutuksen yksityiskohdista. Usean alustan tuki saattaa kuitenkin hidastaa pelimoottorin toimintaa. Mitä enemmän alustoja pelimoottori tukee, sitä todennäköisemmin joudutaan tekemään kompromisseja abstraktion ja optimoinnin välillä. (Bhattacharya, Goon & Paul 2012; Gregory 2014, 12–13.)

2.1 Historiaa

Viimeisen kahden vuosikymmenen ajan pelimoottori on ollut olennainen osa pelinkehitystä. Vielä 80-luvulla pelien kehitys aloitettiin tyhjästä, jotta rajoittuneista laitteistoista saatiin kaikki mahdollinen hyöty irti. Koska laitteistoissa oli yleensä hyvin vähän muistia, oli pelit kirjoitettava assembly-kielellä korkeatasoisten ohjelmointikielten sijaan. Tällöin muistia ja laitteiston muita resursseja voitiin hyödyntää mahdollisimman tehokkaasti, mutta ohjelmakoodi ei ollut uudelleenkäytettävissä. Nykyään tietokoneissa, pelikonsoleissa ja jopa mobiililaitteissa on kymmeniätuhansia kertoja enemmän muistia kuin 80-luvun pelikonsoleissa. Lisäksi nykyiset korkeatasoisten kielten kääntäjät ovat niin kehittyneitä, että assemblya tarvitaan harvoin. (Gregory 2014, 3; Lilly 2009.)

Ensimmäiset alkeelliset pelimoottorit julkaistiin 80-luvun lopulla, mutta ne olivat lähinnä kehittämiensä yritysten sisäisessä käytössä. Näihin kuuluvat esimerkiksi LucasArtsin SCUMM (Script Creation Utility for Maniac Mansion) ja Sierra Entertainmentin SCI (Sierra's Creative Interpreter). Nykyisessä merkityksessään pelimoottorit tulivat tunnetuksi 90-luvun alussa id Softwaren kehittämän Doomin yhteydessä. Doomin sisältö on helposti muokattavissa, koska sen käyttämä pelimoottori Doom engine erottaa selkeästi toisistaan pelimoottorin toiminnot, kuten renderöinti-, törmäyksentunnistus- ja äänijärjestelmän, ja pelikohtaisen datan, kuten pelimaailman, tekstuurit ja äänet. Tämä teki Doom enginestä merkittävän, ja monet kaupalliset pelit kehitettiin sen avulla. (Gregory 2014, 11; Lilly 2009; Ward 2008.)

Monet 90-luvun alun pelimoottorit, kuten Doom engine, käyttivät spritejä luodakseen kolmiulotteisen vaikutelman pelimaailmasta. Ensimmäiset todelliset 3D-pelimoottorit, kuten Bethesda Softworksin XnGine, kehitettiin 90-luvun puolivälissä. Nämä pelimoottorit käyttivät kuitenkin ohjelmistorenderöintiä. Ensimmäisiä laitteistorenderöintiä tukevia pelimoottoreita oli id Softwaren Quake engine, johon laitteistorenderöintituki lisättiin vuonna 1997.

Laitteistorenderöinti mahdollisti tehokkaamman ja nopeamman renderöinnin, koska renderöinti suoritettiin siihen erikoistuneessa laitteistossa. Nykyään laitteistorenderöintiä tuetaan jopa pienissä mobiililaitteissa. (Gregory 2014, 494–495; Lilly 2009.)

2.2 Mobiilialustat

Mobiililaitteissa on pienen kokonsa takia yleensä vähemmän resursseja työpöytätietokoneisiin ja pelikonsoleihin verrattuna. Tietokoneen komponentit, kuten prosessori, grafiikkaprosessori ja muistimoduulit, on koottu mobiililaitteissa tiiviisti pieneen tilaan, minkä takia niiden on oltava pienempiä kuin niiden työpöytävastineet. Yleensä pieni koko rajoittaa komponentin tehokkuutta, mutta toisaalta se vähentää komponentin virrantarvetta ja lämmöntuottoa. Mobiililaitteita suunniteltaessa on tärkeää huomioida komponenttien lämmöntuotto, koska mobiililaitteissa ei ole tilaa työpöytätietokoneissa käytettäville tehokkaille jäähdytysjärjestelmille. Yleensä mobiililaitteissa luotetaan passiivijäähdytykseen, jossa ylimääräinen lämpö johdetaan lämpöä johtavia kanavia pitkin laitteen ulkopuolelle. (Aarnio, Miettinen, Pulli, Roimela & Vaarala 2008, 6–9.)

Muistin määrä nykyisissä mobiililaitteissa alkaa olla varsin suuri. Uusimmissa laitteissa on yhtä paljon RAM-muistia kuin keskivertoisissa työpöytätietokoneissa. Mobiililaitteiden muistintarve ei kuitenkaan ole yhtä suuri kuin työpöytäkoneilla, koska mobiilisovellukset eivät ole erityisen raskaita. Mobiilisovelluksia ei myöskään ole tarkoitus ajaa samanaikaisesti samalla tavalla kuin työpöytäkoneissa. Tallennuskapasiteettiakin mobiililaitteista alkaa löytyä riittävästi. Mobiililaitteissa yleisesti käytettävä flash-muisti vie fyysisesti vähemmän tilaa kuin työpöytäkoneissa käytettävät kovalevyt ja SSD-asemat. Lisäksi monet mobiililaitteet tukevat lisätallennustilaa flash-muistikorttien muodossa. (Aarnio, Miettinen, Pulli, Roimela & Vaarala 2008, 5–7.)

Mobiilialustoille suunnatut pelimoottorit eivät välttämättä eroa rakenteeltaan työpöytätietokoneille ja pelikonsoleille suunnatuista pelimoottoreista. Mobiilipelimoottorin suunnittelussa, toteutuksessa ja sitä käytettäessä pelinkehityksessä on kuitenkin otettava huomioon kohdelaitteiden mahdolliset rajoitukset. Niitä ovat muun muassa prosessorin suorituskyky, näytönohjaimen suorituskyky sekä käytettävissä oleva muistin ja tallennuskapasiteetin määrä. Mobiililaitteiden käyttäjäsyötelaitteet eroavat työpöytäkoneiden ja pelikonsolien käyttäjäsyötelaitteista. Moderneja mobiililaitteita ohjataan yleensä laitekohtaisilla fyysisillä painik-

keilla ja kosketusnäytöllä. Muita mahdollisia peleissä hyödynnettäviä käyttäjäsyötelaitteita ovat muun muassa kiihtyvyysanturi ja GPS-antenni. (Aarnio, Miettinen, Pulli, Roimela & Vaarala 2008, 4–7.)

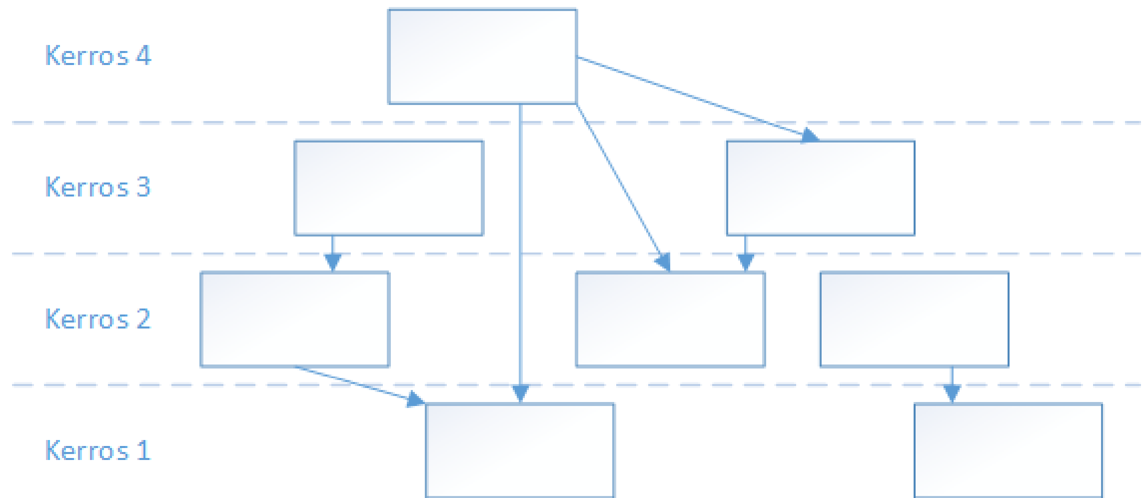
3 RAKENNE

Tässä luvussa esitellään tyypillisen 3D-pelimoottorin rakenne. Tätä rakennetta voidaan hyödyntää kaikenlaisissa niin työpöytätietokoneille, pelikonsoleille kuin mobiililaitteillekin suunnatuissa pelimoottoreissa. Pelimoottorit ovat yleensä suuria ohjelmistojärjestelmiä ja koostuvat useasta pienemmästä osasta. Pelimoottorin rakenne voidaan jakaa kahteen kokonaisuuteen, joita tässä työssä kutsutaan aktiivi- ja passiiviosaksi. (Gregory 2014, 32.)

Pelimoottorin rakenne on suunniteltu C++-kieltä silmällä pitäen sen tehokkuuden ja laajan tuen vuoksi. Rakenteeseen vaikuttaa myös vahvasti oliopohjainen lähestymistapa. Tehokkaat 3D-pelimoottorit on kirjoitettu yleensä C++:lla, koska abstraktimmilla kielillä kirjoitetut ohjelmat eivät välttämättä pyöri yhtä nopeasti kuin C++-ohjelmat. Pelimoottorin rakennetta voi kuitenkin hyödyntää muita ohjelmointikieliä käytettäessä. (Gregory 2014, xxii.)

3.1 Aktiiviosa

Aktiiviosa (englanniksi runtime) sisältää kaikki pelimoottorin ajon aikana tarvittavat toiminnot ja muodostaa suurimman osan pelimoottorin rakenteesta. Se koostuu alijärjestelmistä, jotka sisältävät keskenään samankaltaisia toimintoja eli komponentteja. Alijärjestelmä voi olla riippuvainen toisista alijärjestelmistä, ja alijärjestelmien välisiä riippuvuussuhteita voidaan kuvata kerrosrakenteen avulla. Kerrosrakenteessa toisista riippuvaisimmat alijärjestelmät sijoitetaan ylempiin kerroksiin. Alijärjestelmän tulisi olla riippuvainen vain alemmissa kerroksissa olevista alijärjestelmistä, jolloin pelimoottorin kehitystä haittaavia kaksisuuntaisia riippuvuussuhteita ei syntyisi. Kuviossa 1 kuvataan ohjelmistojärjestelmän kerrosrakenne ilman kaksisuuntaisia riippuvuussuhteita. (Gregory 2014, 32–34.)

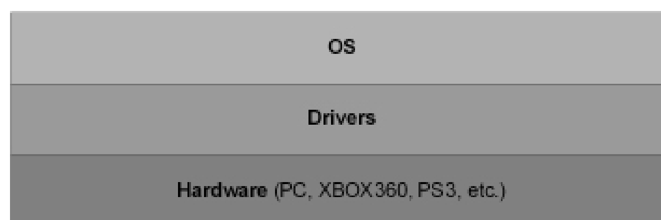


Kuvio 1. Ohjelmistojärjestelmän kerrosrakenne

Suurissa ohjelmistojärjestelmissä on pystyttävä kehittämään useita alijärjestelmiä samanaikaisesti. Alijärjestelmien väliset kaksisuuntaiset riippuvuussuhteet estävät alijärjestelmien itsenäisen kehittämisen ja testaamisen, mikä tekee ohjelmistojärjestelmän kehityksestä tehotonta. Esimerkiksi kaksi toisistaan riippuvaista alijärjestelmää muodostavat turhan monimutkaisen järjestelmän itsenäisiin alijärjestelmiin verrattuna. (Smacchia 2008.)

Alusta

Alustan muodostavat kolme alinta kerrosta, jotka ovat laitteisto, ajurit ja käyttöjärjestelmä (kuvio 2). Laitteisto määrittelee laitteistotason toiminnot, joita pelimoottori pystyy hyödyntämään. Laitteistotason toimintoja ovat esimerkiksi laitteistokiihdytetty renderöinti, verkkoyhteydet ja äänentoisto. Pelikäytössä yleisiä laitteistoja ovat muun muassa työpöytätietokoneet sekä pelikonsolit, kuten PlayStation 4, Xbox One ja Wii U. (Gregory 2014, 34.)



Kuvio 2. Alustakerrokset (Gregory 2014, 34–35)

Käyttöjärjestelmä määrittelee, miten ohjelmia suoritetaan. Joissain käyttöjärjestelmissä käyttäjä voi vuorovaikuttaa usean sovelluksen kanssa samanaikaisesti, kun taas toiset käyttöjärjes-

telmät voivat keskeyttää sovelluksen suorituksen toisen sovelluksen vaatiessa käyttäjän huomiota. Pelikonsoli- ja mobiilikäyttöjärjestelmät ovat yleensä toiminnoiltaan rajoittuneempia kuin työpöytäkäyttöjärjestelmät. Laitteiston ja käyttöjärjestelmän välissä ovat ajurit eli matalan tason ohjelmat, joiden avulla käyttöjärjestelmä kommunikoi laitteiston kanssa. Periaatteessa kaikki alustan yläpuolella olevat alijärjestelmät riippuvat alustasta. (Gregory 2014, 34–35.)

Ohjelmistokehityspaketit

Ohjelmistokehityspaketin tarkoituksena on tarjota valmiita toimintoja kehitettävään ohjelmistoon. Siihen kuuluu ohjelmointirajapinta, jonka kautta ohjelmistokehityspaketin toimintoja käytetään. Kaikkia pelimoottorin toimintoja ei välttämättä kannata toteuttaa itse, vaan ne voidaan lisätä pelimoottoriin ohjelmistokehityspaketteina (kuvio 3). Joskus ohjelmistokehityspakettien käyttö on kuitenkin välttämätöntä. Esimerkiksi laitteistotason ja käyttöjärjestelmän toimintoja ei yleensä voi käyttää ilman niiden virallisia ohjelmistokehityspaketteja. Ohjelmistokehityspaketit muodostavat kerroksen alustan yläpuolelle, ja useat ylempänä sijaitsevat komponentit ovat ohjelmistokehityspaketeista riippuvaisia. (Gregory 2014, 35.)

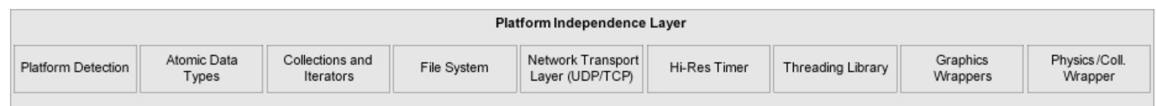
Kuvio 3. Esimerkkejä pelimoottorissa käytettävistä ohjelmistokehityspaketeista (Gregory 2014, 35)

Suosittuja pelimoottoreissa käytettäviä kolmannen osapuolen ohjelmistokehityspaketteja ovat grafiikkakirjastot, kuten Direct3D ja OpenGL, fysiikkamoottorit, kuten Havok ja PhysX, sekä animaatiokirjastot, kuten Granny. Peleissä käytetään paljon erilaisia tietorakenteita ja algoritmeja datan muokkaamiseen, minkä takia esimerkiksi Boost-kirjasto voi olla hyvä lisä pelimoottorin tietorakenne- ja algoritmiarsenaaliin. (Gregory 2014, 35–38.)

Alustariippumattomuus

Useaa alustaa tukevassa pelimoottorissa on hyvä olla alustariippumattomuusjärjestelmä (kuvio 4), joka sisältää toteutukset tuettujen alustojen alustakohtaisille toiminnoille. Toimintojen toteutukset abstrahoidaan usein wrappereiden avulla, jolloin alustariippumattomuusjärjes-

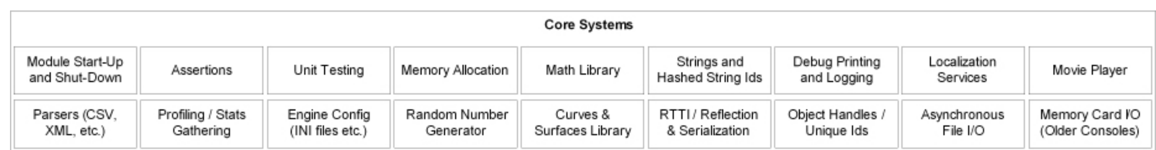
telmä piilottaa toteutusten yksityiskohdat muilta alijärjestelmiltä. Wrappaamisella varmistetaan yhtenäinen toiminnollisuus eri alustojen välillä. Myös ohjelmistokehityspakettien ja standardikirjastojen rajapinnat voidaan wrapata alustariippumattomuusjärjestelmässä. On tärkeää tarjota yhtenäinen rajapinta alustakohtaisten toimintojen käyttämiseen, jotta pelimoottorin käyttökokemus olisi mahdollisimman samanlainen kaikille tuetuille alustoille kehitettäessä. Alustariippumattomuuden voi toteuttaa usealla tavalla, ja joissain pelimoottoreissa alustariippumattomuusjärjestelmää ei toteuteta ollenkaan. (Gregory 2014, 38; Gregory 2014, 297.)



Kuvio 4. Alustariippumattomuusjärjestelmä (Gregory 2014, 38)

Ydin

Pelimoottorin ydinjärjestelmä sisältää yleishyödyllisiä tietorakenteita, algoritmeja ja toimintoja (kuvio 5). Ydinjärjestelmän osia käytetään lähes joka puolella pelimoottoria, minkä takia se sijoitetaan aktiiviosan alapäähän. Ydinjärjestelmään kuuluvat muun muassa muistinhallinta, viesti- ja lokijärjestelmä, tiedostonhallinta sekä matematiikkakirjasto. Muistinhallinta mahdollistaa yleensä muistin varaamisen nopeammin kuin ohjelmointikielten sisäiset muistinhallintatoiminnot. Viesti- ja lokijärjestelmän avulla pelinkehittäjä voi kirjoittaa monipuolisia viestejä kehityskonsoliin ja -lokiin. Tiedostonhallinnan avulla pelimoottorin ulkopuolisia tiedostoja voidaan lukea ja käsitellä tehokkaasti. Matematiikkakirjasto on tärkeä osa pelimoottoria, koska pelit perustuvat vahvasti matematiikkaan. (Gregory 2014, 39.)



Kuvio 5. Ydinjärjestelmä (Gregory 2014, 39)

Ydinjärjestelmään voi myös sisältyä mukautetut merkkijono- ja säiliötietorakenteet. Kohdealustasta riippuen C++-standardikirjaston merkkijono- ja säiliötietorakenteet eivät välttämättä toimi pelikäytössä yhtä tehokkaasti kuin mukautetut tietorakenteet. Lisäksi mukautetut tietorakenteet on mahdollista räätälöidä toimimaan pelimoottorin kanssa saumattomasti.

Ydinjärjestelmä muodostaa kerroksen ohjelmistokehityspakettien yläpuolelle. (Gregory 2014, 39.)

Resurssit

Peleissä käytetään resursseja eli assetteja, kuten kolmiulotteisia malleja, tekstuureja ja ääniä. Suoritettavasta peliohjelmasta erillään olevat resurssit ovat olennainen osa datakeskeistä ohjelmistoarkkitehtuuria. Pelimoottorissa resursseihin liittyvät tietorakenteet ja toiminnot sijaitsevat resurssijärjestelmässä (kuvio 6). Järjestelmän ydin on resurssienhallintakomponentti, jonka avulla resurssit ladataan massamuistista keskusmuistiin pelimoottorin ja pelin käytettäviksi. Resurssienhallinta huolehtii resurssien lataamisesta, säilyttämisestä ja tuhoamisesta. Se varmistaa, että resurssit ladataan oikein, ja huolehtii, ettei jo ladattuja resursseja ladata uudestaan. Resurssienhallinta tuhoaa resurssit, kun niitä ei enää tarvita. (Gregory 2014, 40; Gregory 2014, 297.)

Kuvio 6. Resurssijärjestelmä (Gregory 2014, 40)

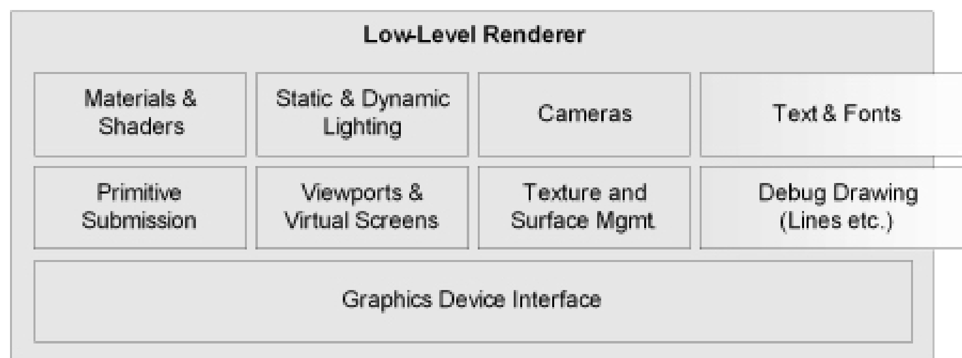
Resurssijärjestelmän toteuttamiseen on useita tapoja. Esimerkiksi jokaiselle resurssityypille voi tehdä oman tietorakenteensa, joka sisältää resurssikohtaisen datan pelimoottorille sopivassa muodossa. Näitä tietorakenteita voidaan käyttää resursseista riippuvien objektien luomiseen. Esimerkiksi tekstuuriobjektin luomiseen tarvitaan tekstuuriresurssia. Resurssijärjestelmä muodostaa kerroksen ydinjärjestelmän yläpuolelle. (Gregory 2014, 40.)

Renderöinti

Renderöintijärjestelmä on suurin ja monimutkaisin osa pelimoottoria. Sitä kutsutaan myös renderöintimoottoriksi, ja se vastaa pelin visuaalisen sisällön piirtämisestä. Renderöintijärjestelmän voi toteuttaa usealla tavalla näytönohjaimen ominaisuuksista ja pelimoottorin renderöintitarpeista riippuen. Yleensä renderöintijärjestelmät suunnitellaan yleisten periaatteiden mukaan, jotka myötäilevät näytönohjainten arkkitehtuureja ja grafiikkaohjelmointirajapintoja.

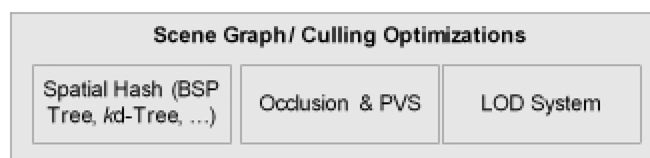
Renderöintijärjestelmä sijaitsee resurssijärjestelmän yläpuolella, ja se voidaan jakaa alijärjestelmiin. (Gregory 2014, 40.)

Pohjimmaisena alijärjestelmänä renderöintijärjestelmässä on matalan tason renderöijä (kuvio 7), jonka tehtävänä on renderöidä geometrisia primitiivejä mahdollisimman nopeasti. Matalan tason renderöinnissä ei käytetä korkean tason optimointeja, kuten näkyvyystarkistuksia. Matalan tason renderöijän pohjana on grafiikkalaite, joka kommunikoi näytönohjaimen kanssa grafiikkarajapinnan avulla. Grafiikkalaitteen kautta määritellään renderöintiasetukset ja lähetetään geometriset primitiivit näytönohjaimelle. Matalan tason renderöijässä määritellään tietorakenteet muun muassa näkymille ja kameroille, tekstuureille, valoille sekä materiaaleille ja varjostimille. (Gregory 2014, 40–41.)



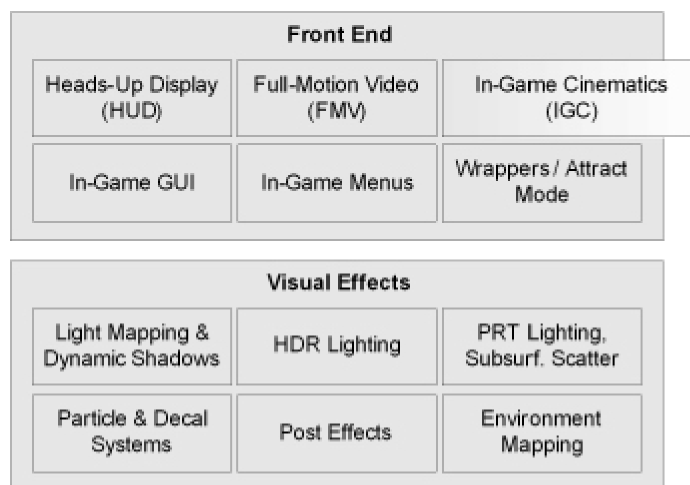
Kuvio 7. Matalan tason renderöijä (Gregory 2014, 41)

Matalan tason renderöijän yläpuolella on scene-graafeihin ja näkyvyysoptimointiin erikoistunut järjestelmä (kuvio 8). Scene-graafeja käytetään kolmiulotteisten objektirakenteiden kuvaamisen lisäksi näkyvyysoptimoinnissa. Suurissa pelimaailmoissa piilossa olevien ja etäisten objektien piirtäminen liian tarkasti voi alentaa ruudunpäivitysnopeutta huomattavasti. Näkyvyysoptimointitoiminnot auttavat määrittämään osittain tai kokonaan piilossa olevat objektit ja tarvittaessa muuttamaan objektien tarkkuutta niiden tarkasteluetäisyyden mukaan. Tällöin voidaan renderöidä vain näkyvissä olevat objektit, jolloin laskenta-aikaa säästyy muihin tehtäviin. (Gregory 2014, 42.)



Kuvio 8. Scene-graafi- ja näkyvyysoptimointijärjestelmä (Gregory 2014, 42)

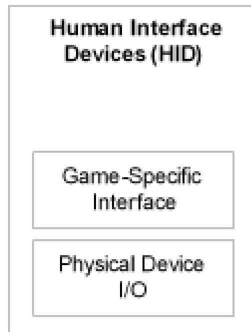
Ylimpinä alijärjestelminä renderöintijärjestelmässä ovat visuaalisista efekteistä ja kaksiulotteisista grafiikoista vastaavat järjestelmät (kuvio 9). Visuaaliset efektit lisäävät peleihin näyttävyyttä, ja niihin sisältyvät muun muassa dynaamiset varjot, edistynyt valaistus, kuvan jälkikäsittely ja partikkeliefektit. 2D-grafiikkajärjestelmän tehtävänä on renderöidä kaksiulotteista grafiikkaa, kuten spritejä, kolmiulotteisen grafiikan päälle. Se auttaa esimerkiksi käyttöliittymän, valikoiden ja HUD-näytön renderöinnissä, ja se voi myös mahdollistaa videon renderöinnin. (Gregory 2014, 42–44.)



Kuvio 9. Visuaaliset efektit ja 2D-grafiikkajärjestelmä (Gregory 2014, 43–44)

Käyttäjäsyötteet

Käyttäjäsyötejärjestelmä (kuvio 10) käsittelee erilaisista käyttäjäsyötelaitteista lähetetyt syötteet. Pelikäytössä yleisiä käyttäjäsyötelaitteita ovat näppäimistö ja hiiri sekä erilaiset peliohjaimet, kuten pad-, sauva- ja rattiohjain. Käyttäjäsyötteisiin kuuluvat tulevien syötteiden lisäksi myös lähtevät syötteet, kuten värinä- ja äänipalautteet. Käyttäjäsyötejärjestelmä määrittelee tuetut käyttäjäsyötelaitteet ja rajapinnat niiden käyttämiseksi. (Gregory 2014, 48.)

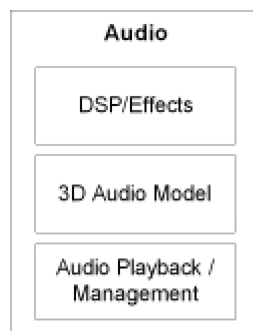


Kuvio 10. Käyttäjäsyötejärjestelmä (Gregory 2014, 48)

Käyttäjäsyötteiden käsittely ei välttämättä ole käyttäjäsyötejärjestelmän ainoa tehtävä. Se voi myös tukea kehittyneempiä toimintoja, kuten näppäinkartoitusta sekä syöteyhdistelmiä ja -eleitä. Näppäinkartoituksella tarkoitetaan fyysisten näppäinten sitomista loogisiin toimintoihin, jolloin pelaaja voi määritellä näppäimet, joilla peliä pelataan. Yhdistelmiin kuuluvat useiden syötteiden yhdistelmät ja sarjat. Eleisiin kuuluvat esimerkiksi kosketusnäytöllä tehtävät kosketuskuviot, kuten pyyhkäisyt ja nipistykset. Käyttäjäsyötejärjestelmä sijaitsee resurssijärjestelmän yläpuolella. (Gregory 2014, 48.)

Äänet

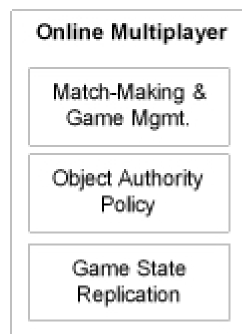
Äänet ovat peleissä yhtä tärkeää kuin grafiikka. Äänijärjestelmän (kuvio 11) ensisijaisena tehtävänä on äänenhallinta ja -toisto. Lisäksi se voi tarjota efektejä äänien manipuloimiseksi ja tukea äänen nauhoittamista. Kolmiulotteisissa peleissä käytetään yleensä kolmiulotteista äänimallia, jossa äänillä on sijainti ja nopeus kolmiulotteisessa avaruudessa. Se saa äänet kuulostamaan realistisilta. Äänijärjestelmä sijaitsee resurssijärjestelmän yläpuolella. (Gregory 2014, 49.)



Kuvio 11. Äänijärjestelmä (Gregory 2014, 49)

Verkkopeli

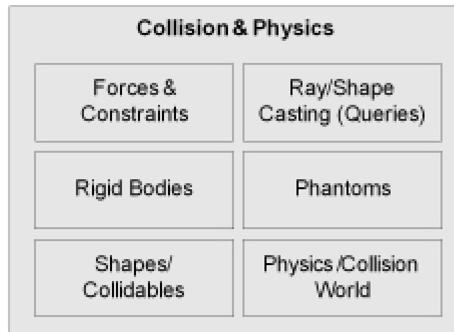
Reaaliaikaisissa verkkopeleissä pelaajien pelitilojen yhtenäistäminen on tärkeää. Pelin on edettävä kaikilla pelaajilla samalla tavalla ja samaan aikaan, sekä sen on myös kyettävä reagoimaan väliaikaisiin yhteyshäiriöihin. Verkkopelijärjestelmä (kuvio 12) tarjoaa toimintoja edellä mainittujen ongelmien ratkaisemiseksi. Sen käyttö ei kuitenkaan rajoitu vain reaaliaikaisiin peleihin, vaan sitä voi käyttää myös tiedon luotettavaan siirtämiseen kaikenlaisissa verkkopeleissä. Lisäksi verkkopelijärjestelmä voi tarjota toimintoja verkkopeli-istunnon luomiseen ja pelaajien yhdistämiseen. Verkkopelijärjestelmä sijaitsee resurssijärjestelmän yläpuolella. (Gregory, 2009, 49–50.)



Kuvio 12. Verkkopelijärjestelmä (Gregory 2014, 50)

Fysiikka

Realistinen fysiikan mallinnus saa pelin tuntumaan todenmukaiselta. Fysiikkajärjestelmä (kuvio 13) tunnetaan myös nimellä fysiikkamoottori, ja sen tehtävänä on simuloida pelimaailman fysiikkaa osittain tai mahdollisimman realistisesti. Fysiikkajärjestelmä liikuttaa kappaleita niihin vaikuttavien voimien perusteella, tunnistaa kappeleiden välisiä törmäyksiä ja reagoi törmäyksiin. Törmäyksentunnistusta käytetään joskus erillään fysiikasta, mutta usein fysiikka ja törmäykset liittyvät vahvasti toisiinsa. Kuten oikeassa maailmassa, myös realistisissa peleissä reagoidaan törmäyksiin fysiikan avulla. (Gregory 2014, 46.)

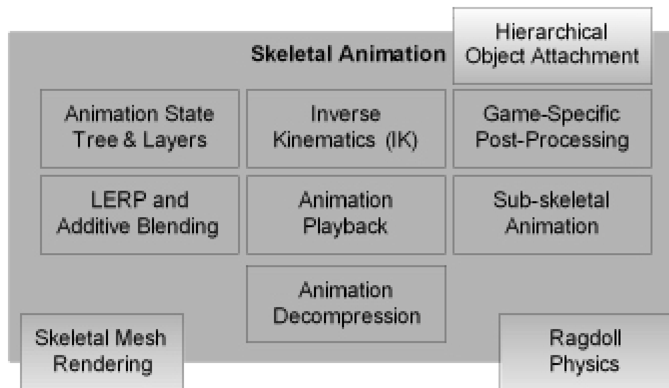


Kuvio 13. Fysiikkajärjestelmä (Gregory 2014, 46)

Fysiikkajärjestelmiä kehitetään harvoin itse, koska ne vaativat paljon tietämystä matematiikasta, fysiikasta ja koodin optimoimisesta. Monet pelimoottorit käyttävät valmista fysiikkamoottoria, joka liitetään pelimoottoriin ohjelmistokehityspakettina. Suositut fysiikkamoottorit, kuten Havok, Nvidia PhysX ja Open Dynamics Engine, ovat monipuolisia ja tehokkaita. Fysiikkajärjestelmä sijaitsee resurssijärjestelmän yläpuolella. (Gregory 2014, 46–47.)

Animaatiot

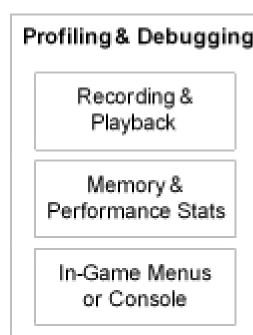
Animaatiojärjestelmän tehtävänä on peliobjektien animoiminen ja animaatioiden hallinta. Yleisin kolmiulotteisten mallien animointimenetelmä on luurankoanimointi, ja kuviossa 14 kuvataan luurankoanimointiin erikoistunut järjestelmä. Luurankoanimoinnissa kolmiulotteisen hahmon verteksit on sidottu niin kutsuttuihin luihin, joita liikutetaan erilaisten asentojen aikaansaamiseksi. Asennosta toiseen siirtyminen edellyttää animaatiojärjestelmältä väliasentojen laskemista, jotta liike näyttäisi sulavalta. Yleensä animaatiojärjestelmä on riippuvainen renderöintijärjestelmästä, jolle se välittää tiedon kolmiulotteisen mallin luurangon asennosta. Tehokkaat pelimoottorit suorittavat luurankoanimoinnin varjostimien avulla. Animaatiojärjestelmä voi olla riippuvainen myös fysiikkajärjestelmästä, jos kolmiulotteisille malleille lasketaan asentoja räsynukkefysiikkaa hyödyntämällä. Animaatiojärjestelmä sijaitsee tavallisesti renderöinti- ja fysiikkajärjestelmien yläpuolella. (Gregory 2014, 47–48.)



Kuvio 14. Luurankoanimaatiojärjestelmä (Gregory 2014, 47)

Profilointi

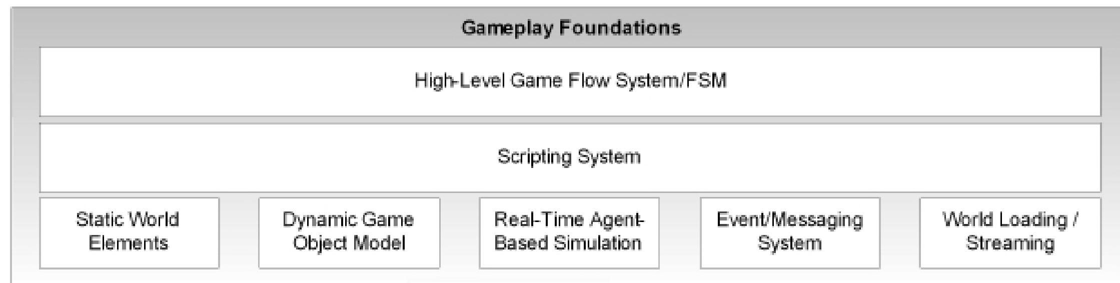
Erityisesti vaativien pelien suhteen on tärkeää, että ne pyörivät mahdollisimman tehokkaasti. Profiloinnilla voidaan hankkia tietoa peliohjelman suorittamisesta ja selvittää sen pullonkaulat. Profilointijärjestelmä (kuvio 15) auttaa pelin profiloimisessa sen kehityksen aikana. Kaikissa pelimoottoreissa ei välttämättä ole sisäistä profilointijärjestelmää, vaan pelinkehittäjät saattavat käyttää erillisiä profilointityökaluja. Profilointijärjestelmä auttaa pelin optimoimisessa tarjoamalla muun muassa muisti- ja suorituskykytilastointia, kehityksenaikaisia piirto-ominaisuuksia sekä ajonaikaisen kehityskonsolin. Kehittynyt profilointijärjestelmä mahdollistaa myös pelin nauhoittamisen ja nauhoitusten toiston. Profilointijärjestelmä sijaitsee tyypillisesti renderöintijärjestelmän yläpuolella. (Gregory 2014, 44–45.)



Kuvio 15. Profilointijärjestelmä (Gregory 2014, 44)

Pelilogiikan apukomponentit

Pelilogiikka on suurimmaksi osaksi pelikohtaista, mutta sen toteuttamisessa voi käyttää yleisiä apukomponentteja. Näihin komponentteihin kuuluvat muun muassa peliobjektimalli, skriptauskomponentti ja tekoälykomponentti. Järjestelmä pelilogiikan apukomponenteille (kuvio 16) sijaitsee aktiiviosan yläpäässä lähimpänä varsinaista peliä. (Gregory 2014, 50–53.)



Kuvio 16. Pelilogiikan apukomponentteja (Gregory 2014, 51)

Peleissä esiintyviä objekteja kutsutaan yleensä peliobjekteiksi, ja pelilogiikan apukomponentteihin kuuluu usein peliobjekteja kuvaava malli. Peliobjektiarkkitehtuuri toteutetaan yleensä oliopohjaisesti. Tällöin peliobjekti sisältää sitä kuvaavia attribuutteja ja sen käyttäytymistä määritteleviä toimintoja. Oliopohjaisuus on luonnollista monissa ohjelmointikielissä, kuten C++:ssa. Oliopohjainen arkkitehtuuri voi kuitenkin aiheuttaa ongelmia, kun peliobjektihierarkiat kasvavat liian suuriksi. Vaihtoehto oliopohjaiselle arkkitehtuurille on suosiotaan kasvattava komponenttipohjainen arkkitehtuuri. Siinä peliobjektit ovat vain tyhjiä olioita, jotka sellaisenaan eivät tee mitään. Peliobjektia ja sen käyttäytymistä kuvaavat erilaiset komponentit, jotka on sidottu peliobjektiin. (Gregory 2014, 50–52; Gregory 2014, 853–854; Gregory 2014, 877–878; Gregory 2014, 881–887.)

Jotta pelilogiikan kehittäminen olisi tehokasta ja nopeaa, monet pelimoottorit sisältävät skriptauskomponentin. Skriptaamalla muutkin kuin ohjelmoijat voivat muuttaa helposti pelin logiikkaa ilman, että pelin lähdekoodia täytyisi kääntää uudelleen. Joissain pelimoottoreissa peliä ei tarvitse edes sulkea skriptejä muutettaessa. Suosittuja skriptikieliä ovat Lua sekä Python, ja jotkin pelimoottorit käyttävät omaa skriptikieltä. (Gregory 2014, 52; Gregory 2014, 954–960.)

Monissa peleissä tarvitaan tekoälyä, kuten reitinhakua ja keinotekoista päättelykykyä. Esimerkiksi itsenäisesti toimivia peliobjekteja ja tietokonevastustajaa ohjaa tekoälykomponentti. Aiemmin tekoäly kehitettiin pelikohtaisesti lähes alusta asti, mutta nykyään pelimoottorin

pelilogiikan apukomponenteista alkaa löytyä yleiskäyttöinen tekoälykomponentti. Pelimoottori voi myös hyödyntää kolmannen osapuolen tekoälyjärjestelmää, kuten Autodesk Kynapsea. (Gregory 2014, 53.)

3.2 Passiiviosa

Passiiviosa koostuu lähinnä työkaluista, joita käytetään pelinkehityksen apuna. Nämä työkalut liittyvät suurimmaksi osaksi pelin resurssien käsittelyyn ja pelimaailman rakentamiseen. Ilman resursseja resurssidata pitäisi kovakoodata peliin, ja peli pitäisi kääntää uudestaan aina resurssidatan muuttuessa. Tämä vaikeuttaisi ja hidastaisi pelin kehitystä. Pelimoottorin passiiviosaa ei tarvita lopullista peliä ajettaessa. (Gregory 2014, 54.)

Sisällönluontityökalut

Pelit ovat multimediasovelluksia, koska ne sisältävät grafiikkaa, animaatioita, ääniä ja mahdollisesti videoita. Yleensä pelin sisältö luodaan kolmannen osapuolen sisällönluontityökaluilla, kuten Adobe Photoshopilla, Autodesk 3ds Maxilla ja Sony Sound Forgella. Sisällönluontityökalujen tuottama sisältö ei yleensä sovi sellaisenaan pelimoottorin käytettäväksi. Esimerkiksi 3ds Maxin .max-tiedostot sisältävät paljon pelikäytössä tarpeetonta tietoa, mikä hidastaa tiedoston lukemista ja käsittelyä pelimoottorissa. Lisäksi monet tiedostomuodot, kuten .max, ovat suljettuja tiedostomuotoja, minkä takia niitä voi olla vaikeaa tai jopa mahdotonta käsitellä järkevällä tavalla. (Gregory 2014, 54–56.)

Resurssinkäsittelyketju

Resurssien kulkua niiden luonnista pelimoottoriin viemiseen kuvataan resurssinkäsittelyketjulla. Ketju alkaa sisällönluontityökalusta, josta resurssi tuodaan väliaikaisessa tai pelikäyttöön sopivassa muodossa. Moniin sisällönluontityökaluihin voidaan kehittää lisäosia, joilla resurssit saadaan tuotua halutussa muodossa. Vaihtoehtoisesti väliaikaisessa muodossa oleva resurssi voidaan muuntaa pelimoottorin käyttämään muotoon passiiviosan työkalun avulla. (Gregory 2014, 55–57.)

Tuonnin jälkeen resursseja voidaan joutua käsittelemään lisää, ennen kuin ne ovat valmiita pelimoottoria varten. Useaa alustaa tuettaessa resursseja on mahdollisesti käsiteltävä eri taval-

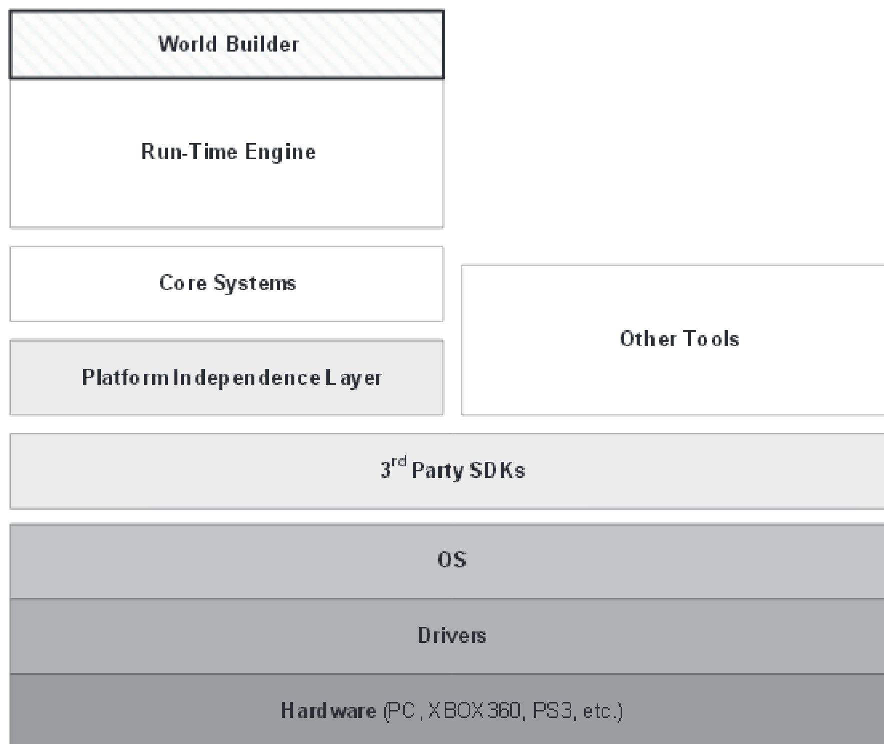
la jokaista alustaa varten. Samaa materiaalia käyttävät mesh-resurssit voidaan ehkä yhdistää ja suuria mesh-resursseja voidaan joutua pilkkomaan pienempiin osiin. Resursseja voi myös joutua linkittämään toisiinsa, jotta pelimoottori osaa ladata kaikki resurssin vaatimat komponentit. Linkittäminen voi käsittää useamman resurssin yhdistämisen yhdeksi kokonaisuudeksi. Esimerkiksi mesh-resurssi käyttää tyypillisesti yhtä tai useampaa materiaalia, ja materiaali yhtä tai useampaa tekstuuria. Tällöin mesh-resurssi linkitetään käyttämiinsä materiaaleihin ja materiaali käyttämiinsä tekstuureihin. (Gregory 2014, 56; Gregory 2014; 318–319.)

Lopuksi, ennen resurssien viemistä pelimoottoriin, resurssit voidaan pakata yhteen tai useampaan tiedostoon tilan säästämiseksi. Yhden suuren tiedoston käyttäminen saattaa myös nopeuttaa resurssien lukemista monen irrallisen tiedoston käsittelyyn verrattuna. Tiedoston avaaminen, lukeminen ja tiedosto-osoittimen siirtäminen voivat olla hitaita operaatioita. Hitaus voi näkyä erityisesti monta resurssia peräkkäin ladattaessa. (Gregory 2014, 320–322.)

Työkaluarkkitehtuuri

Pelimoottorin työkalujen kehityksessä kannattaa hyödyntää aktiiviosan komponentteja. Esimerkiksi ydinjärjestelmän tiedostonhallintaa hyödyntämällä työkaluja varten ei tarvitse kehittää tiedostonhallintaa uudestaan. Kuviossa 17 kuvataan työkaluarkkitehtuuri, jossa työkalut hyödyntävät pelimoottorin aktiiviosan alempia järjestelmiä. Työkalut voivat käyttää myös ylempiä järjestelmiä, kuten renderöintijärjestelmää ja pelilogiikan apukomponentteja. Esimerkiksi pelimaailman rakentamiseen tarkoitettu työkalu voidaan rakentaa kokonaan aktiiviosan päälle, minkä ansiosta peliä on mahdollista pelata työkalusta käsin. Tällöin pelimaailma on nähtävissä työkalussa juuri sellaisena, kuin se pelissä esiintyy. Koko aktiiviosaa hyödyntävä työkaluarkkitehtuuri kuvataan kuviossa 18. (Gregory 2014, 60–61.)

Kuvio 17. Aktiiviosan järjestelmiä hyödyntävä työkaluarkkitehtuuri (Gregory 2014, 61)



Kuvio 18. Aktiiviosaa hyödyntävä työkaluarkkitehtuuri (Gregory 2014, 61)

4 TAVOITE JA SUUNNITTELU

Opinnäytetyön tavoitteena oli toteuttaa mobiilialustalla toimiva pelimoottori aikaisemmissa luvuissa kuvatun teorian pohjalta. Kunnollisen pelimoottorin kehittäminen on suuri ja vaativa työ, minkä takia pelimoottoriin tuli toteuttaa vain tärkeimmät kolmiulotteisissa peleissä hyödynnettävät järjestelmät. Pelimoottorin oli tuettava lokiviestejä, tiedostonhallintaa, moniajoa, korkeatarkkuuksisia ajastimia, resurssienhallintaa, kolmiulotteisten objektien renderöintiä ja animointia, käyttäjäsyötteitä sekä ääntä. Lisäksi pelimoottorin alustakohtaiset toiminnot tuli abstrahoida niin, että tulevaisuudessa tuen lisääminen muille alustoille olisi vaivatonta. Kehityskieleksi valittiin C++, koska se on tehokas, monipuolinen ja ennestään tuttu ohjelmointikieli.

Pelimoottorin resurssienhallinnan oli oltava helposti laajennettavissa, jotta pelimoottorin käyttäjä voisi tarvittaessa lisätä tuen mukautettujen resurssien lataamiseen. Renderöintijärjestelmän tuli tukea renderöintiasetusten muuttamista, grafiikkapuskureita, teksturointia, käyttäjän luomia varjostimia, tekstin piirtoa ja valaistusta. Korkeamman tason renderöinnissä tuli tukea kolmiulotteisten objektien piirtoa ja sprite-piirtoa. Kolmiulotteiseen objektiin sisältyvät materiaalit ja meshit. Pelimoottorin kehityksen jälkeen sen avulla tuli kehittää testisovellus, jonka tarkoituksena on testata pelimoottorin toimintaa. Sovelluksen oli tarkoitus olla pieni peli, joka käyttää kaikkia pelimoottorin toimintoja. Sen tuli toimia myös esimerkkisovelluksena pelimoottorin käyttäjille.

Pelimoottorin suunnittelu aloitettiin sen rakenteesta. Se tuli jakaa sopiviin alijärjestelmiin, ja alijärjestelmät tuli sijoittaa kerroksiin niiden riippuvuussuhteiden mukaisesti. Kerrosten välisiä kaksisuuntaisia riippuvuussuhteita tuli välttää, jotta pelimoottoria voitiin kehittää alijärjestelmä kerrallaan alimmasta kerroksesta alkaen. Kuviossa 19 kuvataan pelimoottorille suunniteltu kerrosrakenne, jossa alijärjestelmät sijaitsevat kerroksittain.

Renderöinti | Käyttäjäsyötteet | Äänet

Resurssienhallinta

Ydin

Alustariippumattomuus

Ohjelmistokehityspaketit

Alusta

Kuvio 19. Pelimoottorin kerrosrakenteen suunnitelma

Kerrosrakenteen suunnittelun jälkeen listattiin kaikki pelimoottoriin tulevat komponentit. Jokaisesta alijärjestelmästä piirrettiin komponenttikaavio, josta ilmeni komponenttien väliset suhteet ja riippuvuudet. Jokaiselle komponentille suunniteltiin alustava rajapinta varsinaisen ohjelmointityön helpottamiseksi. Rajapintojen nimet ja rajapintojen metodien nimet kirjattiin suunnitelmaan.

5 TOTEUTUS

Tässä luvussa kerrotaan, miten pelimoottori toteutettiin. Luku on jaettu suurimmaksi osaksi alijärjestelmien mukaan luvun 3.1 tavoin. Jokaisen alijärjestelmän kohdalla kuvataan sen sisältämiä komponentteja ja annetaan tietoa komponenttien toteuttamisesta.

5.1 Alusta

Pelimoottorin alustaksi valittiin Googlen ylläpitämä Android-käyttöjärjestelmä. Android on alun perin mobiililaitteille suunniteltu avoimen lähdekoodin käyttöjärjestelmä, joka on kehitetty avoimen Linux-käyttöjärjestelmän ytimen päälle. Android-sovellukset kehitetään Java-ohjelmointikielellä, ja ne suoritetaan mukautetussa Java-virtuaalikoneessa. (Android Developers 2014 a; Android Developers 2014 b.)

Android on käytetyimpiä mobiilikäyttöjärjestelmiä Apple iOS:n ja Microsoft Windows Phonen ohella. Android valittiin kehitysalustaksi, koska Android-sovelluksia voi kehittää ilmaiseksi. IOS:lle tai Windows Phonelle kehittäminen vaatii maksullisen kehittäjäjäsenyyden. Lisäksi käytössä oli Android-puhelin, jolla pelimoottoria voitiin testata. Kirjoittamalla pelimoottori C++:lla tulevaisuudessa se voidaan saada vähällä työmäärällä tukemaan esimerkiksi iOS:ia. Javalla kehitettyä pelimoottoria ei saa sellaisenaan toimimaan iOS:lla, koska iOS ei tue Javaa. (Hassell 2010; QuartSoft 2014.)

Graafiset Android-sovellukset, kuten hyötysovellukset ja pelit, käyttävät yhtä tai useampaa activity-komponenttia. Sovellus piirtää käyttöliittymänsä ja peligrafiikkansa activityn ikkunaan ja vastaanottaa käyttäjäsyötteet activityn kautta. Activity voi toimia myös sovelluksen aloituspisteenä. Sovellukselle luodaan Activity tekemällä luokka ja perimällä se Activity-luokasta. Activity-objekti vastaanottaa tietoa tärkeistä tapahtumista, kuten activityn luomisesta ja tuhoamisesta, sen metodien kautta. (Android Developers 2014 c.)

Android-sovellusten kehittäminen natiiveilla ohjelmointikielillä, kuten C:llä ja C++:lla, vaatii Android NDK -työkalukokoelman. Android NDK ei kuitenkaan poista Java-koodin tarvetta, vaan sovelluksen aloituspiste on kirjoitettava Javalla. Sovellus lataa natiivikielisen koodin, minkä jälkeen se voi kutsua natiivikielisiä funktioita Java Native Interfa-

ce -ohjelmointirajapintaa hyödyntämällä. Natiivikielisessä koodissa voidaan käyttää esimerkiksi POSIX-standardin rajapintoja, koska Android on rakennettu Linux-ytimen päälle. (Android Developers 2014 d.)

5.2 Kehitystyökalut

Tärkein työkalu Android-sovelluskehityksessä on Android SDK -ohjelmistokehityspaketti. Se sisältää muun muassa Android-ohjelmointirajapinnan ja hyödyllisiä työkaluja, kuten emulaattorin. Android SDK tarjoaa Activity-luokan lisäksi muun muassa NativeActivity-luokan, joka on peritty Activitysta. Se helpottaa Android-sovelluksen kehittämistä natiiveilla ohjelmointikielillä, sillä Java-koodia ei tarvitse kirjoittaa ollenkaan. Sovelluksen käynnistyessä NativeActivity kutsuu sen natiivikielistä vastinetta, jonka toteutus löytyy Android NDK:sta. Pelimoottorissa käytettiin NativeActivitya, koska se nopeutti projektin aloitusta. (Android Developers 2014 e; Android Developers 2014 f.)

Android SDK:n lisäksi Android-sovelluskehitykseen tarvitaan Java-ohjelmistokehityspaketti, kuten Java Development Kit, ja ohjelmistonrakennustyökalu, kuten Apache Ant. Android SDK vaatii Java-ajoympäristön toimiakseen ja Java-työkaluja Android-sovellusten asennuspakettien digitaaliseen allekirjoittamiseen. Nämä tulevat Java-ohjelmistokehityspaketin mukana. Antia käytetään Android-sovellusten kääntämiseen ja niiden asennuspakettien luomiseen. (Android Developers 2014 e.)

Android NDK oli tärkeä työkalu, koska pelimoottori kehitettiin kokonaan C++:lla. Android NDK sisältää C ja C++ -kääntäjän, linkkerin, ohjelmointikirjastoja sekä tehokkaan ohjelmistorakennusjärjestelmän. Android NDK:n kääntäjä pystyy kääntämään C- ja C++-koodia yleisimmille Androidin tukemille prosessoriarkkitehtuureille. Ohjelmointikirjastoihin kuuluu muun muassa NativeActivity-luokan natiivikielinen rajapinta, jonka avulla Android-sovellusta voidaan hallita natiivikielisestä koodista. (Android Developers 2014 d.)

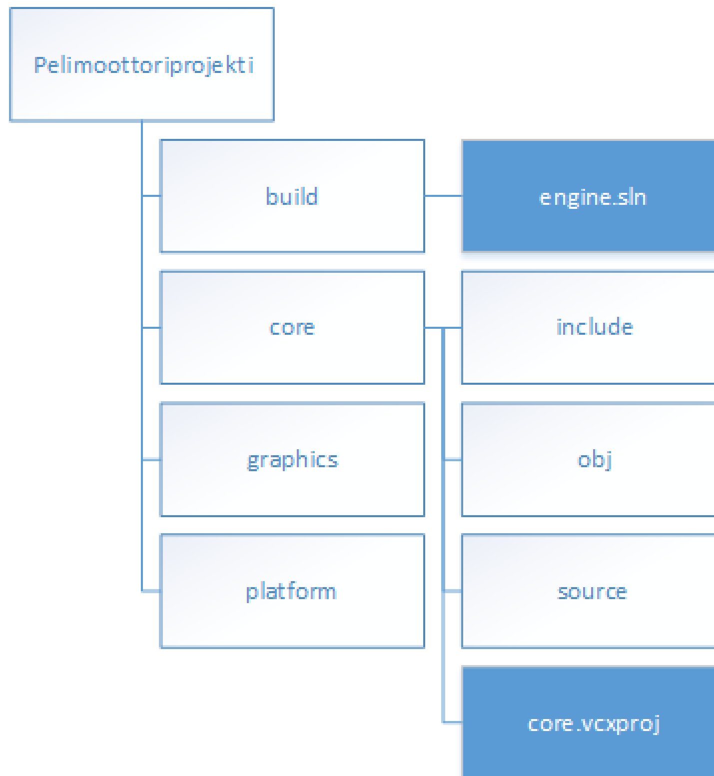
Pelimoottorin kehitysympäristöksi valittiin Microsoft Visual Studio 2013, koska Android SDK ei sisällä minkäänlaista kehitysympäristöä. Visual Studio saatiin toimimaan Android-kehitysympäristönä vs-android-lisäosan avulla, joka mahdollistaa Android-sovellusten ohjelmoimisen, kääntämisen ja laitteelle asentamisen suoraan Visual Studiossa. Valitettavasti Android-sovelluksia ei voi debugata Visual Studiossa vs-androidin avulla, mutta hyvät puolet

Visual Studion käyttämisessä koettiin menevän puuttuvan debug-tuen edelle. (Android Developers 2014 e; Vs-android 2014.)

5.3 Projektin rakenne

Visual Studion toimiessa kehitysympäristönä pelimoottoriprojekti rakennettiin Visual Studion solutioneja ja projekteja käyttäen. Solution toimii säiliönä useille toisiinsa liittyville Visual Studio -projekteille. Jokaiselle alijärjestelmälle, itse käännettävälle ulkoiselle kirjastolle ja pelimoottorin testisovellukselle luotiin oma Visual Studio -projekti. Pelimoottoriprojektille luotiin yksi solution, johon nämä projektit lisättiin. (Microsoft Developer Network 2014 a.)

Pelimoottoriprojektin juurihakemistoon luotiin hakemisto jokaista Visual Studio -projektia varten. Kaikki projektihakemistot sisältävät Visual Studio -projektitiedoston (tiedostopääte .vcxproj), projektin otsikko- ja lähdekooditiedostot omissa hakemistoissaan sekä väliaikaiset käännöstiedostot omassa hakemistossaan. Juurihakemistoon luotiin myös build-hakemisto, joka sisältää pelimoottoriprojektin solution-tiedoston (tiedostopääte .sln) sekä käännetyt kirjastot ja suoritettavat ohjelmatiedostot. Kuviossa 20 kuvataan projektin hakemistorakenne. Include-hakemisto sisältää otsikkotiedostoja, source-hakemisto lähdekooditiedostoja ja obj-hakemisto väliaikaisia käännöstiedostoja. Kuvioon ei ole merkitty kaikkia projektihakemistoja.



Kuvio 20. Projektin hakemistorakenne

5.4 Ohjelmistokehityspaketit

Pelimoottorissa käytetään seuraavia Android NDK:n tarjoamia kirjastoja: libstdc++, Android-natiivikirjasto, Android-lokikirjasto, EGL, OpenGL ES 2.0, OpenSL ES ja zlib. Libstdc++ on GNU-projektin toteutus C++-standardikirjastosta. Android-natiivikirjasto sisältää rajapintoja Android-sovelluksen hallintaan, ja Android-lokikirjasto sisältää rajapinnan Android-lokiviestien kirjoittamiseen. EGL:ää käytetään grafiikkakontekstin luomiseen, ja OpenGL ES:ää käytetään kolmiulotteisen grafiikan laitteistokiihdytettyyn renderöintiin. OpenSL ES:n avulla voidaan toteuttaa laitteistokiihdytystä tukeva äänentoistojärjestelmä, ja zlib-kirjastoa hyödynnetään PNG-kuvadatan purkamisessa. (Android Developers 2014 d.)

Android NDK:n tarjoamien kirjastojen lisäksi pelimoottorissa käytetään libpng-kirjastoa, joka on virallinen kirjasto PNG-kuvien lukemiseen, kirjoittamiseen ja käsittelyyn. Se vaatii zlib-kirjaston toimiakseen. Koska libpng on avoimen lähdekoodin kirjasto, sen lähdekoodi sisällytettiin pelimoottoriprojektiin ja käännettiin itse. (Libpng 2014.)

5.5 Alustariippumattomuus

Alustariippumattomuuden toteuttamiseen on useita tapoja. Todennäköisesti helpoin tapa on olla käyttämättä erillistä alustariippumattomuusjärjestelmää, jolloin kaikki alustakohtainen koodi sijoitetaan eri puolille pelimoottoria. Todennäköisesti tämä tekee koodin muokkaamisesta ja uuden alustan tukemisesta hankalaa. Ongelma voidaan ratkaista sijoittamalla kaikki pelimoottorissa käytettävä alustakohtainen koodi alustariippumattomuusjärjestelmään. Alustakohtaisia toimintoja käyttävien komponenttien rajapinnat sijaitsevat järjestelmissä, joihin komponentit kuuluvat. Näiden komponenttien toteutukset kuitenkin sijaitsevat alustariippumattomuusjärjestelmässä. Esimerkiksi Tiedostonhallintakomponentin rajapinta sijaitsee ydinjärjestelmässä. Sen toteutus puolestaan sijaitsee alustariippumattomuusjärjestelmässä, koska se on riippuvainen alustakohtaisista toiminnoista.

Alustakohtaisen toiminnon toteutuksessa kaikkien tuettujen alustojen toteutukset voidaan sijoittaa samaan paikkaan. Kohdealusta voidaan määritellä esikääntäjän makrojen avulla, jolloin muiden alustojen toteutukset voidaan jättää kokonaan kääntämättä. Tämä voi kuitenkin tehdä koodista hankalasti luettavaa ja funktioista turhan pitkiä:

```
#if defined(ANDROID)
	// Android implementation
#elif defined(IOS)
	// iOS implementation
#elif defined(WINDOWS_PHONE)
	// Windows Phone implementation
#endif
```

Vaihtoehtoisesti alustakohtaisesta toiminnosta voidaan tehdä erilliset toteutukset jokaiselle tuetulle alustalle. Toteutukset sijaitsevat omissa käännösyksiköissään, ja pelimoottoria käännettäessä kääntämiseen sisällytetään vain kohdealustan toteutukset. Esimerkiksi tiedostonhallintakomponentin Android-toteutus voisi sijaita AndroidFileStream.cpp-tiedostossa, joka sisällytetään käännösprosessiin kohdealustan ollessa Android.

Pelimoottorissa käytetään alustariippumattomuusjärjestelmää, jossa kaikki alustakohtainen koodi sijaitsee. Alustakohtainen koodi lajiteltiin alihakemistoihin alustan mukaan. Android-alihakemiston lisäksi luotiin POSIX-alihakemisto, johon POSIX-kohtaiset toteutukset sijoi-

tettiin. Tällöin POSIX-kohtaisia toimintoja on mahdollista hyödyntää erillään Androidista esimerkiksi toista Linux-pohjaista alustaa tuettaessa.

5.6 Ydin

Ydinjärjestelmä on pelimoottorin tärkeimpiä järjestelmiä, koska sen komponentteja käytetään lähes kaikkialla pelimoottorissa. Monet pelimoottorin toiminnalle tärkeät komponentit sijaitsevat ydinjärjestelmässä. Alla kuvataan ydinjärjestelmän tärkeimpiä komponentteja ja niiden toteutuksia.

Perustietotyypit

C++:n perustietotyyppien koko eli tila, jonka ne varaavat muistista, riippuu alustasta ja kääntäjästä. C++-standardi määrittelee perustietotyypeille minimikoot, mutta ne voivat olla suurempiakin. Perustietotyypeille määriteltiin ydinjärjestelmän Types.h-otsikkotiedostossa tietyn kokoiset tyyppialiakset, koska pelimoottoria kehitettiin alustariippumattomuutta silmällä pitäen. Tällöin perustietotyypit ovat samankokoisia kaikilla tuetuilla alustoilla. Kokonaislukutyyppien aliakset alkavat sanalla Int tai Uint ja loppuvat numerolla, joka ilmaisee tyypin koon bitteinä. Esimerkiksi 16-bittiselle etumerkilliselle kokonaisluvulle (yleensä short 32-bittisissä järjestelmissä) määriteltiin alias Int16 ja 32-bittiselle etumerkittömälle kokonaisluvulle (yleensä unsigned int 32-bittisissä järjestelmissä) alias Uint32. Merkkityypeille määriteltiin aliakset Char8 ja Char16 sekä liukulukutyypeille aliakset Float32 ja Float64. (Gregory 2014, 118–119.)

Asetukset

Pelimoottori sisältää paljon sen toimintoihin vaikuttavia asetuksia. Asetukset liittyvät muun muassa grafiikkaan, ääniin ja peliohjaimiin. Asetuksia on pystyttävä muuttamaan ilman pelimoottorin uudelleenkääntämistä, minkä takia niitä ei voida kovakoodata pelimoottorin lähdekoodiin. Asetukset voidaan tallentaa esimerkiksi tiedostoon käyttäen sopivaa tiedostomuotoa, kuten XML:ää. Binäärimuotoa käyttämällä voidaan säästää tallennustilaa, jos sitä on käytettävissä vain vähän. (Gregory 2014, 290–291.)

Pelimoottorissa asetusten arvot on järkevää sijoittaa globaaleihin muuttujiin tai komponenttiin, joka lukee ne pelimoottorin ulkopuolelta. Usein asetukset on pystyttävä tallentamaan pelimoottorista tiedostoon tai muuhun tallennusmediaan. Pelimoottorille voi välittää asetuksia myös komentorivin tai mahdollisen ajonaikaisen konsolin kautta. Ydinjärjestelmään toteutettiin asetuskomponentti, joka lukee ja kirjoittaa asetuksia tiedostonhallintakomponentin avulla. Peliohjelmalle voidaan syöttää myös komentoriviparametreja. Pelimoottori ei kuitenkaan käytä niitä, vaan ne välitetään pelikoodiin. (Gregory 2014, 290–291.)

Virheidenhallinta

Ohjelmistokehityksessä tulee vastaan kahdenlaisia perusvirheitä: käyttäjävirheitä ja ohjelmoijavirheitä. Käyttäjävirheen sattuessa ohjelman käyttäjä on tehnyt virheen, esimerkiksi yrittäessään avata tiedoston, jota ei ole olemassa. Ohjelmoijavirhe tunnetaan myös bugina eli ohjelmointivirheenä. Sen on estettävissä, vaikka se aiheutuisi käyttäjän toimesta. Virheiden jako käyttäjä- ja ohjelmoijavirheisiin ei ole aina selvää. Esimerkiksi ohjelmoija A, joka käyttää ohjelmoija B:n kirjoittamaa koodia, voidaan ajatella käyttäjänä ohjelmoija B:n näkökulmasta. (Gregory 2014, 145.)

Käyttäjävirheiden hallitsemiseksi pelimoottorin komponenteissa käytetään tarvittaessa tarkastusfunktioita. Tarkastusfunktiolla voidaan esimerkiksi varmistaa, että tiedosto on olemassa, ennen kuin se yritetään avata. Olemattoman tiedoston avaaminen tulkitaan ohjelmoijavirheeksi, koska virheellinen tiedostopolku voi johtua ohjelmoijan virheestä. Ohjelmoijavirheen tapahtuessa virheidenhallintakomponentti kirjoittaa virheestä mahdollisimman paljon tietoa virhelokiin ja kaataa ohjelman. Ohjelman kaataminen pakottaa kehittäjän korjaamaan virheen ennen kehitystyön jatkamista. (Gregory 2014, 146.)

Pelimoottorissa päätettiin olla käyttämättä poikkeuksia, koska ne monimutkaistavat koodia. Lisäksi poikkeusturvallisen koodin suunnittelu ja kirjoittaminen vaatii aikaa. Pelimoottori ei käsittele ulkoisten kirjastojen, kuten C++-standardikirjaston, heittämiä poikkeuksia. Muistinhallintaan liittyviin poikkeuksiin ei oikein voida reagoida sulkematta peliä, ja muunlaisten poikkeusten heittäminen voidaan estää tai niihin voidaan varautua edellä mainittujen virheidenhallintamenetelmien avulla. (Gregory 2014, 148–149.)

Muistinhallinta

Tietokoneohjelman on käytettävä muistia hyödykseen järkevästi, jotta ohjelmaa voitaisi suorittaa tehokkaasti. Moderneilla prosessoriarkkitehtuureilla ohjelman suorituskyky määritellään suurimmaksi osaksi sen muistinkäytön perusteella. Muistinkäytön optimointi on erityisen tärkeää peleissä ja muissa reaaliaikasovelluksissa. Optimointi koskee lähinnä dynaamisen muistin eli kekomuistin (englanniksi heap) varaamista ja vapauttamista. (Gregory 2014, 239–240.)

Dynaamisen muistin varaaminen ja vapauttaminen C-standardikirjaston malloc- ja free-funktiolla tai C++:n new- ja delete-operaattorilla voi olla hyvin hidasta. Yleiskäyttöisen kekomuistinvaraajan pitää pystyä varaamaan kaikenkokoisia muistialueita, mikä aiheuttaa muistinhallinnallisia suorituskykykustannuksia. Lisäksi useimmilla alustoilla kekomuistinvaraaja joutuu tekemään kontekstinvaihdon käyttäjätilasta kernel-tilaan, jossa muistinvarauspyyntö käsitellään. Sen jälkeen joudutaan vielä tekemään kontekstinvaihto takaisin käyttäjätilaan. (Gregory 2014, 240.)

Pelimoottoreissa ei voida välttää dynaamisen muistin käyttämistä, minkä takia muistinvaraus- ja muistinvapautusmenetelmiä on optimoitava. Yleinen optimointimenetelmä on mukautettujen muistinvaraajien toteuttaminen. Mukautettu muistinvaraaja saattaa esivarata suuren muistialueen malloc-funktiolla tai new-operaattorilla, ja palauttaa pelimoottorin käyttäjälle pyydetyn kokoisen lohkon esivaratusta muistista nopeasti. Valitettavasti mukautettujen muistinvaraajien ja kehittyneen muistinhallinnan kehittämiseen ei riittänyt aikaa. (Gregory 2014, 240–241.)

Loki

Lokikomponentin tehtävänä on tulostaa viestejä erilaisiin viestilaitteisiin, kuten viestikonsoleihin ja tiedostoihin. Kaikki viestit tulee tulostaa lokikomponentin kautta, jotta ne päätyvät oikeisiin viestilaitteisiin. Näihin viesteihin kuuluvat muun muassa virhe- ja varoitusviestit, kehityksenaikaiset apuviestit sekä erilaiset informatiiviset viestit. Tulostettavia viestejä voidaan suodattaa viesteille määritettävien tasojen avulla. (Gregory 2014, 412–414.)

Kohdealustan konsoli-ikkunaan tai lokityökaluun tulostettaessa lokikomponenttiin wrapataan alustan tulostustoiminnallisuus. Yleisimmillä työpöytäkäyttöjärjestelmillä tähän sopii C-standardikirjaston printf-funktio tai C++-standardikirjaston std::cout-objekti, mutta Androi-

dissa ei ole sisäänrakennettuna erillistä konsoli-ikkunaa. Androidilla viestit voidaan tulostaa logcat-lokityökaluun käyttämällä Android NDK:n tarjoamaa lokirajapintaa. (Android Developers 2014 d; Gregory 2014, 412.)

Säiliöt ja merkkijono

Säiliöt ovat tärkeitä tietorakenteita datan säilömiseen. Niitä käytetään joka puolella pelimoottoria, ja niiden pitäisi toimia sulavasti pelimoottorin kanssa. Pelimoottoreissa käytettäviä säiliöitä ovat muun muassa taulukko, vektorilista, linkitetty lista, pino, joukko, kartta ja puut. Jokaisella säiliötyypillä on hyvät sekä huonot puolensa, ja oikean säiliön valinta tiettyä tehtävää varten riippuu säiliölle asetetuista vaatimuksista. (Gregory 2014, 254–256.)

Joissain pelimoottoreissa käytetään suoraan C++-standardikirjaston säiliöitä. Yleensä C++-standardikirjasto on toteutettu eri tavalla eri kääntäjillä, jolloin säiliöt eivät välttämättä toimi kaikilla alustoilla samalla tavalla ja yhtä tehokkaasti. Vaihtoehtona on käyttää kaikilla tuetuilla alustoilla toimivaa mallikirjastoa tai toteuttaa säiliöt itse. Ydinjärjestelmään toteutettiin seuraavat säiliökomponentit itse: taulukko, vektorilista, linkitetty lista ja kartta. Säiliöt toteutettiin oppimismielessä, ja niitä tuskin on toteutettu pelikäyttöön optimaalisimmalla tavalla. (Gregory 2014, 260–261.)

Merkkijonokin voidaan ajatella säiliöksi, joka säilöö merkkejä yleisen datan sijaan. C-tyyliseen merkkijonoon eli merkkitaulukkoon verrattuna merkkijonoja on tehokkaampi käsitellä oliopohjaisen merkkijonokomponentin avulla. Toisaalta merkkijono-objekteista voi aiheutua suorituskykykustannuksia harkitsemattomasti käytettyinä. Pelimoottoriin toteutettiin merkkijonokomponentti itse. Siitä on kaksi variaatiota, joista toinen tukee ASCII-merkistöä (Char8) ja toinen UTF-16-merkistöä (Char16). UTF-16-merkistön avulla pelissä voidaan esittää tekstiä lähes kaikilla maailman kielillä, jolloin se soveltuu hyvin monikielisiin peleihin. (Gregory 2014, 274; Gregory 2014, 279–282.)

Tiedostonhallinta

Pelimoottorin on pystyttävä lukemaan erilaisia tiedostoja tehokkaasti, minkä takia ydinjärjestelmään toteutettiin tiedostonhallintakomponentti. Tiedostonhallintakomponentti on matalan tason komponentti tiedostojen avaamiseen, lukemiseen, kirjoittamiseen ja sulkemiseen. Se lukee ja kirjoittaa tiedostoja käyttäjän komentojen mukaisesti, mutta ei osaa tulkita niiden

sisältöä. Eri tiedostot tunnistetaan toisistaan tiedostopolun avulla, joka kertoo tiedoston sijainnin ja nimen tiedostojärjestelmässä. Tiedostopolun rakenne on käyttöjärjestelmäkohtainen, minkä takia sen käsittely voi olla hankalaa. Tiedostonhallintakomponenttia käytetään pääasiassa resurssienhallinnassa, mutta sitä hyödynnetään myös asetustiedostojen lukemisessa ja kirjoittamisessa. (Gregory 2014, 298–302.)

Työpöytäalustoilla voidaan yleensä käyttää C-standardikirjaston tiedostonhallintaa, kuten fopen-, fread- ja fwrite-funktiota, tai C++-standardikirjaston std::fstream-luokkaa, mutta ne eivät välttämättä tue käyttöjärjestelmäkohtaisia tiedostonhallintaominaisuuksia. Androidilla sovelluksen resurssit sisällytetään sen asennuspakettiin ja tallennetaan laitteelle sovellusta asennettaessa. Sopiva paikka pelisovellusten resurssien säilömiseen on Android-projektin assets-hakemisto. Assets-hakemistossa sijaitsevat tiedostot tallennetaan raakana käsittelemättömänä datana, ja niitä voidaan lukea Android NDK:n AAssetManager-rajapinnan avulla. (Android Developers 2014 d; Android Developers 2014 g; Gregory 2014, 302–303.)

Moniajo

Tehokas pelimoottori hyödyntää moniajoa (englanniksi multithreading) ja tarjoaa moniajoon liittyviä komponentteja, kuten säikeen (englanniksi thread), lukon (myös muteksi, englanniksi mutex) ja semaforin (englanniksi semaphore). Säie on ohjelmaprosessin komponentti, jossa ohjelmakoodia suoritetaan. Useamman säikeen avulla koodia voidaan ajaa useasta kohdasta samaan aikaan. Moniytimiset prosessorit mahdollistavat ohjelman nopean suorituksen usean säikeen avulla. Lukkoja käytetään estämään saman datan muokkaaminen eri säikeissä samanaikaisesti, mikä voi aiheuttaa datan korruptoitumista. Semaforeja käytetään lukkojen kanssa, ja yleensä niillä pysäytetään säikeen suoritus, kunnes tietty ehto täyttyy toisen säikeen toimesta. (Juujärvi 1999.)

Androidilla activitya ajetaan sovellusprosessin pääsäikeessä, jossa activity vastaanottaa tietoa tapahtumistaan. Siksi pääsäiettä ei saa tukkia, eikä siinä voi ajaa pelisilmukkaa. Pelimoottorissa pelisilmukkaa varten luodaan pelisäie, jossa pelilogiikka ensisijaisesti suoritetaan. Sovelluksen pääsäie ilmoittaa pelisäikeelle tärkeistä sovellukseen liittyvistä tapahtumista, ja pelisäie reagoi niihin. Esimerkiksi sovelluksen sammuessa pelisilmukka lopetetaan ja pelisäie tuhoaa itsensä. Säikeet ovat yleensä käyttöjärjestelmäkohtaisia. Ydinjärjestelmän säie-, lukko- ja semaforikomponenttien Android-toteutukset käyttävät POSIX-standardin pthread-rajapintaa. (Android Developers 2014 h.)

Korkeatarkkuuksinen ajastin

Aika on tärkeä elementti peleissä, koska pelit ovat reaaliaikaisia sovelluksia. Peliobjekteja on päivitettävä ja piirrettävä tasaisin väliajoin sulavan pelikokemuksen aikaansaamiseksi. Myös monia alijärjestelmiä ja komponentteja on päivitettävä tasaisin väliajoin, mutta aikavälien pituudet saattavat vaihdella järjestelmästä tai komponentista riippuen. Ajastinta käytetään kahden pisteen välissä kuluneen ajan laskemiseen. Korkeatarkkuuksisen ajastimen (englanniksi high resolution timer) avulla kulunut aika voidaan laskea jopa nanosekunnin tarkkuudella. Peleissä yleensä riittää millisekunnin tarkkuus, mutta esimerkiksi sekunnin tarkkuus on liian epätarkka. (Gregory 2014, 339–341; Gregory 2014, 353.)

Korkeatarkkuuksisen ajastimen toteutus on alustakohtainen, koska ajat lasketaan prosessoritasolla. Ydinjärjestelmän korkeatarkkuuksisen ajastinkomponentin Android-toteutus käyttää POSIX-standardin clock_gettime-funktiota. Clock_gettime palauttaa kutsumishetken ajan jopa nanosekunnin tarkkuudella alustasta riippuen. Pistettä, josta aikaa mitataan, ei ole välttämättä määritelty, mutta korkeatarkkuuksista ajastinta ei ole tarkoitettukaan absoluuttisen ajan ilmoittamiseen. Kahden aikapisteen välissä kulunut aika lasketaan aikojen erotuksena. (The Open Group 2014.)

Ikkuna

Ikkuna on suorakulmion muotoinen alue, jolle pelin visuaaliset elementit piirretään. Yleensä työpöytäkäyttöjärjestelmissä ikkunalla on viestijono, jonka kautta vastaanotetaan tärkeitä ikkunaan liittyviä viestejä. Viestit kertovat esimerkiksi ikkunan sulkeutumisesta ja tulevista käyttäjäsyötteistä. NativeActivityn natiivikielisessä rajapinnassa käytetään samankaltaista viestijonoa, mutta sitä ja käyttäjäsyötteitä ei ole sidottu ikkunaan. (Android Developers 2014 f; Gregory 2014, 41.)

Ikkuna on vahvasti käyttöjärjestelmäkohtainen, koska käyttöjärjestelmissä käytetään erilaisia ikkunajärjestelmiä. Androidilla jokaiselle activitylle annetaan valmis ikkuna, minkä takia ikkunaa ei voida luoda eksplisiittisesti. Pelimoottorin on odotettava ikkunan luomisesta kertovaa viestiä, ennen kuin ikkunaan voidaan piirtää. Vastaavasti ikkunan tuhoamisesta kertova viesti tarkoittaa, ettei ikkunaan voida enää piirtää. Androidilla activityn ikkuna tuhotaan aina activityn pysähtyessä, esimerkiksi toisen sovelluksen vaatiessa käyttäjän huomion. Ydinjärjestelmään toteutettiin ikkunakomponentti, joka sisältää ikkunakahvan. Ikkunaobjektilta voi-

daan muun muassa hakea ikkunan leveys ja korkeus. (Android Developers 2014 c; Android Developers 2014 f.)

Matematiikka

Pelit ovat hyvin matemaattispainotteisia, minkä takia tehokas pelimoottori tarvitsee tehokkaan matematiikkajärjestelmän tai -kirjaston. Pelit voivat hyödyntää lähes kaikenlaista matematiikkaa, kuten algebraa, trigonometriaa ja tilastotiedettä. Kolmiulotteisissa peleissä hyödynnetään yleisimmin lineaarialgebraa, johon kuuluu vektorit, matriisit ja kvaterniot. Monipuolinen matematiikkajärjestelmä tai -kirjasto tarjoaa edellä mainittujen komponenttien lisäksi viiva-, säde- ja tasokomponentit sekä aputoimintoja, kuten lineaarisen interpoloinnin, projektiomatriisien luomisen ja satunnaislukugeneraattorin. (Gregory 2014, 39; Gregory 2014, 165.)

Pelimoottorissa käytetään aikaisemmin kehitettyä matematiikkakirjastoa, josta löytyy tarpeelliset komponentit kolmiulotteisen pelin kehittämiseen. Kirjasto soveltuu myös kaksiulotteisten pelien kehittämiseen ja tarjoaa hyödyllisiä matemaattisia apufunktioita. Matematiikkakirjasto sulautettiin kiinteäksi osaksi ydinjärjestelmää ja muokattiin toimimaan pelimoottorin kanssa saumattomasti.

Alijärjestelmien käynnistys ja sammutus

Pelimoottori sisältää tärkeitä alijärjestelmiksi luokiteltavia kokonaisuuksia ja komponentteja, jotka pitää alustaa ja käynnistää ennen varsinaisen pelin suorittamista. Näihin alijärjestelmiin kuuluvat esimerkiksi muistinhallinta ja lokijärjestelmä, ja ne on yleensä käynnistettävä oikeassa järjestyksessä. Järjestys riippuu pääasiassa alijärjestelmien välisistä riippuvuuksista. Jos esimerkiksi alijärjestelmä B riippuu alijärjestelmästä A, on A käynnistettävä ennen B:tä. Alijärjestelmät sammutetaan käänteisessä järjestyksessä. (Gregory 2014, 231.)

Alijärjestelmäobjektit säilötään staattisiin muuttujiin. Koska C++-standardi ei määrittele staattisten objektien alustusjärjestystä, objektien ei voida olettaa alustuvan oikeassa järjestyksessä. Ratkaisuksi alijärjestelmien konstruktorit eivät hoida varsinaista alustusta, vaan alijärjestelmille on määritelty erilliset alustus- ja sammutusmetodit. Näitä metodeja kutsutaan oikeassa järjestyksessä peliohjelman käynnistyessä ja sammuessa. Lisäksi alijärjestelmät hyödyn-

tävät singleton-mallia. Sillä varmistetaan, että jokaisesta alijärjestelmästä on olemassa korkeintaan yksi instanssi koko ohjelman suorituksen ajan. (Gregory 2014, 232–236.)

5.7 Resurssienhallinta

Resurssienhallintaa voidaan pitää viimeisenä osana resurssinkäsittelyketjua. Resurssit valmistellaan pelimoottoria varten passiiviosan työkalujen avulla, ja aktiiviosan resurssienhallinta lataa ja käsittelee niitä pelin ajon aikana. Tässä luvussa kuvataan resurssienhallinnan kaksi oleellisinta komponenttia. (Gregory 2014, 308–309.)

Resurssienhallinnan ydin

Resurssienhallinnan ytimenä toimii resurssienhallintakomponentti, jonka kautta kaikki pelimoottorin tukemat resurssit ladataan ja luodaan. Se pitää listaa ladatuista resursseista ja varmistaa, ettei samaa resurssia ladata muistiin useaan kertaan. Muuten resurssien kopiot varaisivat muistia turhaan. Resurssienhallintakomponentti huolehtii myös resurssien tuhoamisesta, kun niitä ei enää tarvita. Tällöin muistia vapautuu muuhun käyttöön. (Gregory 2014, 319–320.)

Resurssienhallintajärjestelmään haluttiin kehittää resurssienhallintakomponentti, jota on helppo laajentaa niin pelimoottorin kehittäjien kuin käyttäjienkin toimesta. Tähän käytettiin resurssienhallintakomponentin mallimetodia (englanniksi template), jonka avulla voidaan ladata kaikkia resurssienhallinnan tukemia resurssityyppejä:

```
T* ContentManager::load<T>(const Core::String& filename);
```

Metodi palauttaa osoittimen ladattuun resurssiin, jos resurssi on ladattu aiemmin. Muuten metodi hakee tyypin T resurssilukijan, joka lataa resurssin. Resurssilukija ja resurssienhallinnan laajennettavuutta esitellään alempana. Jotkin resurssit vaativat itsensä lisäksi muiden resurssien lataamista. Esimerkiksi mesh-resurssi tarvitsee materiaalissaan määriteltyjä tekstuureja, minkä takia kyseiset tekstuurit on ladattava mesh-resurssin lataamisen yhteydessä. Kun resurssi on ladattu, resurssienhallintakomponentti lisää resurssin karttasäiliöön. Karttaelementin avaimena toimii resurssin tiedostopolun tiiviste (englanniksi hash) ja arvona osoitin

resurssiin. Avaimeksi kävisi myös tiedostopolku merkkijonona, mutta merkkijonojen vertailu on paljon hitaampaa kuin tiivisteiden vertailu. (Gregory 2014, 323–324; Gregory 2014, 332.)

Resurssilukija

Resurssilukijakomponentti on malliluokka, jonka tehtävänä on tietyn tyyppisten resurssien lataaminen. Kaikille tuetuille resurssityypeille luotiin siis oma erikoistoteutuksensa (englanniksi specialisation) resurssilukijakomponentista. Pelimoottorin käyttäjä voi helposti lisätä tuen omille resurssityypeilleen luomalla niille resurssilukijakomponentin erikoistoteutukset. Resurssia ladattaessa resurssienhallintakomponentti hakee oikeantyyppisen resurssilukijan. Se myös välittää resurssilukijalle tiedostonhallintakomponentin instanssin, jossa resurssitiedosto on valmiiksi avattuna. Resurssilukija lukee resurssitiedoston, luo resurssin muistiin ja palauttaa resurssienhallintakomponentille osoittimen resurssiin.

5.8 Renderöinti

Renderöintijärjestelmän kehityksessä on hallittava muun muassa lineaarialgebraa, koordinaattimuunnokset ja 3D-grafiikkaketjun (englanniksi 3D graphics pipeline) tuntemista. Käytettävä grafiikkarajapinta on hyvä olla tuttu ennen renderöintijärjestelmän kaltaisen järjestelmän toteuttamista. Alla kuvataan renderöintijärjestelmään kuuluvat komponentit. (Gregory 2014, 443.)

Grafiikkarajapinta ja grafiikkakonteksti

Renderöintijärjestelmässä käytetään sulautetuille laitteille suunnattua laitteistokiihdytettyä OpenGL ES 2.0 -grafiikkarajapintaa. Se perustuu OpenGL 2.0 -grafiikkarajapintaan, mutta ei tue OpenGL ES 1.1:n kiinteää funktioketjua (englanniksi fixed function pipeline). Android tukee OpenGL ES 2.0:aa API 5 -versiosta eteenpäin. Pääsyynä OpenGL ES 2.0:n valinnalle oli ohjelmoitavien varjostimien joustavuus. OpenGL ES 1.1 ei tue ohjelmoitavia varjostimia, eikä uutta OpenGL ES 3.0:aa tuettu käytettävässä testipuhelimessa. (Aarnio, Miettinen, Pulli, Roimela & Vaarala 2008, 18; Android Developers 2014 d.)

Grafiikkakontekstin luomisessa ja hallitsemisessa pelimoottori käyttää EGL-rajapintaa, jonka kanssa OpenGL ES toimii yhteistyössä. EGL on OpenGL ES:n ja natiivin ikkunajärjestel-

män, tässä tapauksessa Android-ikkunajärjestelmän, välissä toimiva rajapinta. Tärkeimpiä tehtäviä grafiikkakontekstin luonnissa on oletuskehyspuskurin (englanniksi framebuffer) määritteleminen. Oletuskehyspuskurin ominaisuudet määrittelevät renderöidyn kuvan ulkoasun ja oletuskehyspuskurin tukemat kuvapuskurit, kuten syvyyspuskurin. (Aarnio, Miettinen, Pulli, Roimela & Vaarala 2008, 18.)

Androidilla grafiikkakonteksti voidaan luoda vasta activityn ikkunan luomisen jälkeen, koska grafiikkakonteksti sidotaan ikkunaan. Vastaavasti grafiikkakonteksti on tuhottava ennen kuin ikkuna tuhotaan, esimerkiksi activityn pysähtyessä. Grafiikkakontekstin tuhoutuessa kaikki OpenGL ES -objektit tuhotaan. Activityn jatkaessa suoritustaan on grafiikkakonteksti ja OpenGL ES -objektit luotava uudestaan. Grafiikkakontekstin luominen on wrapattu grafiikkakontekstikomponentin toteutukseen. (Aarnio, Miettinen, Pulli, Roimela & Vaarala 2008, 259; Android Developers 2014 f.)

Grafiikkalaite

Grafiikkalaite on renderöintijärjestelmän keskus. Se wrappaa grafiikkarajapinnan toiminnot toteutuksessaan, ja esimerkiksi grafiikkaketjun toimintaan vaikuttavat asetukset asetetaan grafiikkalaitteen kautta. Myös piirtokomennot suoritetaan grafiikkalaitteen kautta. Ennen piirtämistä grafiikkalaitteelle asetetaan piirrossa käytettävät varjostimet sekä verteksi- ja indeksipuskuri. Piirtokomentoa suorittaessaan grafiikkalaite hyödyntää asetettuja asetuksia, varjostimia ja puskureita. Grafiikkalaite vaatii onnistuneesti luodun grafiikkakontekstin toimiakseen. (Gregory 2014, 40–41.)

Grafiikkapuskuri

Grafiikan piirtäminen on nopeampaa, jos piirrettävä data sijaitsee näytönohjaimen muistissa keskusmuistin sijaan. Grafiikkapuskureita käytetään datan säilömiseen näytönohjaimen muistiin. OpenGL ES -puskuriobjekteja voidaan käyttää sekä verteksi- että indeksidatan säilömiseen. Näytönohjaimen muistissa olevaa dataa voidaan myös päivittää puskuriobjektien kautta. (Aarnio, Miettinen, Pulli, Roimela & Vaarala 2008, 180–182.)

Renderöintijärjestelmään toteutettiin grafiikkapuskurikomponentti, joka wrappaa OpenGL ES -puskuriobjektin toteutuksessaan. Lisäksi järjestelmään toteutettiin grafiikkapuskurikomponenttia käyttävä verteksipuskuri- ja indeksipuskurikomponentti. Verteksipuskuri on eri-

koistunut lomittaisen verteksidatan säilömiseen ja indeksipuskuri indeksidatan säilömiseen. Näiden komponenttien rajapinnat suunniteltiin helposti käytettäviksi. Verteksipuskurin luomisessa sille asetetaan verteksiformaatti, joka määrittelee verteksidatan muodostavat komponentit ja komponenttien järjestyksen. Tämän ansiosta verteksiformaattia ei tarvitse ilmoittaa erikseen piirron yhteydessä, vaan grafiikkalaite saa sen verteksipuskurilta.

Tekstuuri

Renderöintijärjestelmässä tekstuureja käytetään kolmiulotteisten mallien pinnan kuviointiin ja muodon muuttamiseen sekä spritejen esittämiseen. Kolmiulotteisen mallin pinnan kuvioinnin ja värin määrittelevää tekstuuria kutsutaan diffuusikartaksi, ja mallin muotoa muuttavaa tekstuuria kutsutaan normaalikartaksi. Spritet piirretään diffuusikarttojen avulla. (Gregory 2014, 462.)

Tekstuurikomponentin toteutuksessa on wrapattu OpenGL ES -tekstuuriobjekti, jonka kautta tekstuuridata ja tekstuurin ominaisuudet asetetaan. Tekstuuridataa on myös mahdollista muokata jälkeenpäin. Ominaisuuksiin kuuluvat muun muassa osoitustila (englanniksi addressing mode) ja suodatus (englanniksi filtering). Tekstuuriobjekteja voidaan luoda manuaalisesti, mutta pääasiassa ne luodaan resurssienhallinnan kautta kuvatiedostoja lataamalla. Tekstuureja varten resurssilukijaan kehitettiin erikoistoteutus, joka lataa tekstuurin kuvatiedostosta ja luo tekstuuriobjektin. Tekstuurilukija käyttää libpng-kirjastoa, minkä takia pelimoottorissa tuetaan vain PNG-kuvaformaattia. Tuki muille formaateille voidaan kuitenkin lisätä vaivattomasti tekstuurilukijaa muokkaamalla. (Gregory 2014, 463–464; Gregory 2014, 467.)

Varjostimet

Renderöintijärjestelmä tukee vain verteksi- ja fragmenttivarjostimia, koska OpenGL ES 2.0 ei tue uudempia OpenGL-varjostintyyppejä. Renderöintijärjestelmän varjostinkomponentti wrappaa toteutuksessaan OpenGL ES:n ohjelmaobjektin sekä verteksi- että fragmenttivarjostinohjelman. Komponentti nimettiin efektikomponentiksi, jotta sitä ei sekoitettaisi yksittäiseen varjostinohjelmaan. Efektiobjektia luotaessa sille välitetään varjostimien lähdekoodit, jotka on kirjoitettu OpenGL Shading Language -kiellellä. Se kääntää lähdekoodit OpenGL ES:n verteksi- ja fragmenttiohjelmaksi, ja linkittää ohjelmat OpenGL ES -ohjelmaobjektiin. (Aarnio, Miettinen, Pulli, Roimela & Vaarala 2008, 18.)

Efektiobjekteja voidaan lisäksi luoda lataamalla varjostimien lähdekoodit tiedostoista. Tähän käytettiin mukautettua tiedostomuotoa, josta löytyy sekä verteksi- että fragmenttiohjelman lähdekoodi. Resurssilukijaan kehitettiin erikoistoteutus efektien lataamista varten. Se lukee lähdekoodit tiedostosta ja luo niiden avulla efektiobjektin.

Varjostimien uniform-muuttujien hallintaan kehitettiin efektiparametrikomponentti, jonka instanssi edustaa OpenGL ES -ohjelmaobjektin uniform-muuttujaa. Efektiparametri sisältää muun muassa uniform-muuttujan sijainnin, minkä ansiosta muuttujan arvoa voidaan muuttaa tehokkaasti. Efektiobjekti sisältää efektiparametriobjektit kaikille uniform-muuttujilleen, ja ne voidaan hakea helposti efektiobjektista.

Teksti

Fontit ovat keskeisessä asemassa tekstin esittämisessä. Usein fontit toteutetaan tekstuureina, jotka sisältävät pelissä tarvittavat merkit kuvadatana. Fonttitekstuurin lisäksi fontista tarvitaan lisätietoja, jotta tekstiä voidaan piirtää oikein. Fontin lisätiedot määrittelevät esimerkiksi merkkien sijainnit tekstuurissa sekä niiden perusviivan ja välistyksen. Fonttitekstuurin luominen on todennäköisesti järkevintä pelinkehityksen aikana, eikä vasta pelin ajon aikana. (Gregory 2014, 539.)

Renderöintijärjestelmään kehitettiin fonttikomponentti, joka sisältää fonttitekstuurin, tiedon merkkien sijainneista tekstuurissa ja fontin ulkoasutietoja. Tekstin piirtäminen tapahtuu merkki kerrallaan fonttitekstuuria ja fontin ulkoasutietoja hyödyntäen. Resurssilukijan erikoistoteutusta fonttien lataamista varten ei ehditty toteuttamaan, minkä takia fonttiobjektit joudutaan luomaan manuaalisesti.

Valaistus

Valaistus on tärkeä elementti realistista maailmaa piirrettäessä. Renderöintijärjestelmään toteutettiin yksinkertainen ympäristö-, suunta- ja pistevalaistus sekä tuki normaalikartoille. Normaalikartoilla määritellään, kuinka valo käyttäytyy kolmiulotteisen mallin pinnalla. Niiden avulla mallin muotoon voidaan lisätä yksityiskohtia ja parantaa sen tarkkuutta. Kehittyneempiä ominaisuuksia, kuten heijastumista ja varjoja, ei ehditty toteuttamaan. Valaistus toteutettiin pääasiassa varjostimia käyttämällä, koska varjostimet on suunniteltu valaistuksen tehokkaaseen laskemiseen. (Gregory 2014; 468–470; Gregory 2014, 520.)

Valaistusvarjostimien kirjoittaminen on pääasiassa pelinkehittäjän vastuulla, koska yleisesti kaikenlaisissa peleissä toimivia valaistusvarjostimia on vaikea kirjoittaa. Renderöintijärjestelmään kirjoitettiin kuitenkin kaksi yleistä Blinn-Phong-valaistusmallia hyödyntävää efektiä, joista toinen tukee normaalikarttoja. Nämä efektit ovat helposti käytettävissä missä tahansa pelimoottoria hyödyntävässä projektissa.

Materiaali

Materiaalilla määritellään kolmiulotteisen objektin visuaaliset ominaisuudet, kuten käytettävät tekstuurit ja varjostimet, ja mahdolliset varjostinparametrit, kuten värit, läpinäkyvyys ja heijastavuus. Lisäksi materiaali voi määritellä grafiikkalaitteen asetuksia, jos mallin piirtäminen vaatii erityisiä toimenpiteitä grafiikkarajapinnalta. Renderöintijärjestelmään toteutettiin yksinkertainen materiaalikomponentti, joka määrittelee kolmiulotteisen mallin piirtämisessä tarvittavat tekstuurit ja efektin. Materiaali tukee diffuusi- ja normaalikarttoja. (Gregory 2014, 468.)

Mesh

Meshillä määritellään kolmiulotteisen mallin muoto yleensä kolmioita käyttäen. Kolmio on yksinkertaisin monikulmio, ja sen verteksit sijaitsevat aina samalla tasolla. Kolmiulotteisille malleille ei ole omaa komponenttia, vaan mesh-komponenttia käytetään mallien piirtämiseen. Siksi se sisältää verteksi- ja indeksipuskurin sekä materiaalin. Verteksipuskuri sisältää kaikki meshin kolmioiden verteksit ja indeksipuskuri kolmioiden vertekseihin viittaavat indeksit piirtojärjestyksessä. Mesh-objekti piirretään välittämällä grafiikkalaitteelle sen verteksi- ja indeksipuskuri sekä materiaalista saatava efekti ja tekstuurit. (Gregory 2014, 447.)

Solmu

Renderöintijärjestelmään ei ehditty kehittää näkyvyysoptimointiin soveltuvaa puurakennetta. Sen sijaan kolmiulotteisten objektien säilömiseen ja piirtämiseen toteutettiin yksinkertainen solmuista koostuva puurakenne. Solmulla voi olla sisar- ja lapsisolmuja, ja jokaisella solmulla on viittaus vanhempaansa. Juurisolmulla eli solmulla, joka aloittaa puurakenteen, ei ole vanhempaa. Mesh-objekti voi toimia solmuna, jolloin se edustaa piirrettävää kolmiulotteista mallia. Solmu, joka ei ole mesh-objekti, toimii vanhempana muille solmuille. Esimerkiksi juurisolmu on tällainen solmu.

Renderöintijärjestelmän solmukomponentti sisältää sijainnin ja orientaation, joiden avulla solmuobjektia voidaan liikuttaa ja orientoida kolmiulotteisessa avaruudessa. Vanhempana toimivan solmun liikuttaminen ja orientoiminen vaikuttavat sen lapsisolmujen sijaintiin ja orientaatioon. Mesh-komponentti on peritty solmukomponentista, jolloin se toimii solmun tavoin.

Sprite

Spritejä käytetään kaksiulotteisen grafiikan piirtämiseen. Yleensä niiden avulla toteutetaan pelin käyttöliittymä ja HUD-elementit, jotka piirretään mahdollisen kolmiulotteisen grafiikan päälle. Spriteillä voidaan myös toteuttaa ainoastaan kaksiulotteista grafiikkaa hyödyntäviä pelejä. Spriten piirrossa käytetään kahdesta kolmiosta koostuvaa suorakulmiota, johon spritekuva kartoitetaan tekstuurina. Yleensä spritet piirretään ortografista projektiota käyttäen perspektiivisen projektion sijaan, koska perspektiivinen hahmotus ei ole toivottavaa sprite-grafiikkaa piirrettäessä. (Gregory 2014, 538.)

Renderöintijärjestelmään toteutettiin sprite-hallintakomponentti, jotta spritejen piirtäminen olisi mahdollisimman tehokasta. Komponenttiin listataan piirrettävät spritet, mikä mahdollistaa kerralla useamman spriten piirtämisen. Piirrettäville spriteille voidaan myös asettaa syvyysarvot, jotka määrittelevät spritejen piirtojärjestyksen. Tällöin spritejä ei tarvitse syöttää hallintakomponentin listaan piirtojärjestyksessä.

Kolmiulotteisten mallien lataaminen

On kannattavaa ladata kolmiulotteiset mallit tiedostosta, jossa data on optimaalisessa muodossa. Tällöin tiedostosta luettu data on pelimoottorin käytettävissä mahdollisimman vähällä käsittelyllä. Tiedosto on myös nopeampi lukea, jos se ei sisällä pelimoottorille tarpeetonta dataa. Binäärimuodot vievät yleensä vähemmän tilaa kuin tekstimuodot, mutta eivät ole helposti luettavissa ilman erillistä työkalua. (Gregory 2014, 493–494.)

Pelimoottorissa käytetään omaa binääritiedostomuotoa kolmiulotteisten mallien tallentamiseen. Tiedostossa määritellään kokonaisen puurakenteen solmut sijainteineen ja orientaatioineen. Mesh-solmulle määritellään verteksi-, indeksi- ja animaatiodata sekä materiaali. Animaatioista kerrotaan tarkemmin seuraavassa kappaleessa. Tiedostot luodaan pelimoottorin työkalun avulla Autodeskin .fbx-tiedostoista.

Animaatio

Pelimoottorissa tuetaan yksinkertaista meshin luurankoanimointia. Kokonaisen animaatiojärjestelmän sijaan toteutettiin animaatiokomponentti, joka sijoitettiin renderöintijärjestelmään. Animaatiokomponentti sisältää meshin animaatiodatan eli tiedon meshin luiden asennoista kehyksittäin sekä tiedon luiden ja meshin verteksien suhteista. Kehys vastaa tiettyä ajanhetkeä, ja pelimoottorin käyttäjä määrittelee kehysten välissä kuluvan ajan eli kehystaajuuden. Animaatiota voi nopeuttaa pienentämällä kehystaajuutta ja hidastaa suurentamalla kehystaajuutta. Animaatiokomponentissa tuetaan neljää verteksiä luuta kohden, eli yksi luu voi vaikuttaa korkeintaan neljän verteksin sijaintiin. (Gregory 2014, 551–552.)

Animaatio-objektia päivitettäessä se laskee kuluneen ajan avulla luiden asennot. Jos kulunut aika osuu kehyksen kohdalle, luiden asennot määritellään suoraan kehysdatassa. Kuluneen ajan osuessa kahden kehyksen väliin pitää kehysdataa interpoloida oikeiden asentojen määrittelemiseksi. Animaatiokomponentti tukee lineaarista interpolointia sijaintidatalle ja kvaterniointerpolointia orientaatiodatalle. Jokaisella päivityskerralla luille luodaan transformaatiomatriisit, joiden avulla meshin verteksien sijainnit lasketaan. (Gregory 2014, 558; Gregory 2014, 570.)

Luuanimoinnissa voidaan hyödyntää verteksivarjostinta, jossa animoitujen verteksien sijainnit voidaan laskea tehokkaasti. Varjostimelle syötetään luiden transformaatiomatriisit, tiedot jokaiseen verteksiin sidotuista luista ja näiden luiden painoarvot. Luun painoarvolla määritellään, kuinka paljon luun liike vaikuttaa verteksin sijaintiin. Verteksin sijainnin laskemisessa käytetään siihen vaikuttavien luiden transformaatiomatriiseja ja painoarvoja. (Gregory 2014, 570.)

5.9 Käyttäjäsyötteet

Käyttäjäsyötejärjestelmässä tuetaan yleisimpiä modernien mobiililaitteiden tarjoamia käyttäjäsyötelaitteita. Niihin kuuluvat fyysiset näppäimet, kosketusnäyttö ja kiihtyvyysanturi. Tässä luvussa kuvataan käyttäjäsyötteisiin kuuluvat komponentit.

Käyttäjäsyötteiden ydin

Kaikki käyttäjäsyötteet kulkevat käyttäjäsyötekomponentin kautta, joka toimii käyttäjäsyötejärjestelmän ytimenä. NativeActivity hyödyntää käyttäjäsyötteiden vastaanottamisessa viestijonoaan. Viestijonon käyttäjäsyöteviestit lähetetään käyttäjäsyötekomponentille, joka tarkastaa käyttäjäsyötteen tyypin. Käyttäjäsyötekomponentti käsittelee näppäinsyötteet, mutta kosketus- ja sensorisyötteet lähetetään vastaavien komponenttien käsiteltäviksi. Käyttäjäsyötekomponentti pitää listaa laitteen näppäinten tiloista, ja pelisovellus voi tiedustella näppäimen tilaa milloin tahansa komponentin rajapinnan kautta. (Android Developers 2014 f.)

Kosketuspaneeli

Kosketuspaneelikomponentti käsittelee kosketussyötteet. Jokaisesta kosketussyötteestä luodaan kosketusobjekti, joka sisältää kosketuksen sijainnin kosketusnäytöllä sekä kosketussyötteen tapahtumisajan ja tyypin. Kosketussyötteen tapahtumisaikaa voidaan käyttää muun muassa kosketuseleiden tunnistamisessa. Esimerkiksi liikkuva kosketus pisteestä A pisteeseen B voidaan tulkita pyyhkäisyeleeksi, jos pyyhkäisy tapahtuu tietyn aikavälin sisällä. Kosketussyötteen tyyppi ilmaisee, onko kosketus alkava, liikkuva vai loppuva. Kosketusobjektit lisätään listaan, johon pääsee käsiksi kosketuspaneelikomponentin rajapinnasta.

Kiihtyvyysanturi

Kiihtyvyysanturi on vain yksi monista modernien mobiililaitteiden tukemista antureista. Käyttäjäsyötejärjestelmä tukee vain kiihtyvyysanturia, mutta järjestelmän anturirajapinta suunniteltiin laajennettavaksi. Kiihtyvyyssyötteet käsitellään kiihtyvyysanturikomponentissa, jonka kautta kiihtyvyysdata haetaan. Kiihtyvyysanturi ei ole oletuksena päällä, ja se aktivoidaan käyttäjäsyötekomponentin kautta. Laitteen kiihtyvyys ilmaistaan karteesista koordinaatistoa käyttämällä kolmen akselin suhteen.

5.10 Äänet

Äänijärjestelmä on yhtä tärkeä osa pelimoottoria kuin renderöintijärjestelmä. Molemmilla on samoja ominaisuuksia ja iso rooli pelin immersion luomisessa. Sekä renderöintijärjestelmä että äänijärjestelmä voidaan rakentaa omaksi moottorikseen. Molemmille on määritelty pro-

sessiketju, joka kuvaa resurssin kulun alkupisteestä määränpäähän. Äänijärjestelmässä resurssina toimii äänisignaali, joka aloittaa matkansa digitaalisena datana ja päätyy lopulta analogisena ääniaaltona ilmaan. Alla kuvataan äänijärjestelmän komponentit. (Gregory 2014, 743–744; Gregory 2014, 808.)

Äänirajapinta

Äänijärjestelmässä käytetään laitteistokiihdytettyä OpenSL ES -äänirajapintaa, joka on suunnattu erityisesti sulautetuille laitteille. Android tukee OpenSL ES 1.0.1 -versioon perustuvaa rajapintaa Android API 9 -versiosta eteenpäin. OpenSL ES mahdollistaa äänentoiston, äänen nauhoittamisen ja äänen käsittelyn suoraan natiivikoodista, minkä ansiosta Java-kutsut voidaan välttää. OpenSL ES tukee myös kolmiulotteisia ääniä, joiden avulla ääntä voidaan hyödyntää kolmiulotteisissa peleissä immersiivisesti. Kolmiulotteisilla äänillä on sijainti ja nopeus, joiden avulla äänen ominaisuudet voidaan määrittää realistisesti suhteessa pelaajahahmoon. (Android Developers 2014 d; The Khronos Group 2009, 2-4.)

OpenSL ES on määritelty C-kielellä. Toisin kuin OpenGL ja OpenGL ES, se hyödyntää objektipohjaista lähestymistapaa. OpenSL ES:n objektipohjaisuus perustuu kahteen konseptiin: objektiin ja rajapintaan. Objekti sisältää resursseja ja voidaan rinnastaa C++-luokkaan. Rajapinta sisältää metodeja, joiden kautta objektin toimintaa hallitaan. Joitain objekteja voidaan hallita useamman rajapinnan kautta. (The Khronos Group 2009, 13.)

Äänenhallinta

Äänenhallintakomponentti toimii äänitoimintojen ytimenä. OpenSL ES:n aloituspisteenä toimii moottoriobjekti (englanniksi engine), jonka rajapinnan avulla luodaan kaikki muut OpenSL ES -objektit. Moottoriobjekti luodaan ja alustetaan äänenhallintakomponentissa. Seuraavaksi luodaan ulostulon miksausobjekti (englanniksi output mix), joka edustaa yhtä tai useampaa äänilaitetta. Ulostulon miksausobjektia käytetään äänen ohjaamiseen oikeisiin ulostulolaitteisiin. Äänenhallintakomponentin avulla luodaan toistettavat ääniobjektit, joista kerrotaan seuraavassa kappaleessa. (The Khronos Group 2009, 17; The Khronos Group 2009, 85.)

Ääni

Äänikomponentti sisältää toiminnot yksittäisen äänen toistamiseen, tauottamiseen ja pysäyttämiseen. Sen avulla on myös mahdollista kelata ääntä ja muuttaa äänen ominaisuuksia, kuten äänenvoimakkuutta. Äänikomponentissa luodaan äänisoitinobjekti (englanniksi audio player), jonka avulla yksittäistä ääntä toistetaan. Äänisoitin luodaan moottoriobjektin avulla, ja sille määritellään datalähde (englanniksi data source) ja data-allas (englanniksi data sink). Datalähde määrittelee toistettavan äänen datan sijainnin ja formaatin. Androidilla datalähteen sijaintina voidaan käyttää AAssetManager-rajapinnan tarjoamia tiedostokuvaajia (englanniksi file descriptor), jolloin äänitiedostoja ei tarvitse ladata erikseen. Data-allas määrittelee äänen ulostulon, ja siinä käytetään ulostulon miksausobjektia. (Android Developers 2014 d; The Khronos Group 2009, 67.)

Äänentoistoa ohjataan äänisoitinobjektin toistorajapinnalla (SLPlayItf), joka mahdollistaa äänen toistamisen, tauottamisen ja pysäyttämisen. Äänisoitinobjektin kelausrajapinnan (SLSeekItf) avulla voidaan siirtyä tiettyyn kohtaan ääntä ja määritellä äänen uudelleentoistoasetukset. Äänenvoimakkuutta ohjataan äänisoitinobjektin voimakkuusrajapinnalla (SLVolumeItf), jonka avulla onnistuu myös äänen mykistäminen. (The Khronos Group 2009, 67–70.)

6 TESTAUS

Pelimoottoria testattiin yksikkötestien ja testisovelluksen avulla. Yksikkötestit ovat koodinpätkiä, joilla testataan ohjelman osien toimivuutta. Ne antavat ilmoituksen, jos testattava koodi ei täytä sille asetettuja ehtoja. Yksikkötesteillä testattiin yksittäisten komponenttien toimintaa ja varmistettiin niiden toimivan tarkoituksenmukaisesti. Komponenttien muuttuessa niiden toimivuus voitiin tarkistaa nopeasti yksikkötestit ajamalla. Yksikkötestien kirjoittaminen vei aikaa, mutta käytetyn ajan uskottiin korvautuvan ajan myötä. Yksikkötesteillä testattiin lähinnä ydinjärjestelmän komponentteja, kuten säiliöitä ja merkkijonoa. (Microsoft Developer Network 2014 b.)

Testisovelluksesta ei ehditty kehittää peliä. Siitä tehtiin grafiikkademo, jossa kuitenkin testataan kaikki pelimoottorin toiminnot. Testisovelluksessa pyörii korkeusakselinsa ympäri animoitu kolmiulotteinen malli, joka hyödyntää diffuusi- ja normaalikarttaa. Malli on valaistu Blinn-Phong-valaistusmallia käyttäen, ja siihen lisätään yksityiskohtia normaalikartan avulla. Sovelluksessa piirretään myös pieni käyttöliittymä spritejen avulla. Käyttöliittymän kautta voi näyttää ja piilottaa tekstipohjaista tietoa sovelluksen suorituksesta, kuten ruudunpäivitysnopeuden ja piirrettyjen kolmioiden määrän kehystä kohden. Mallin pyörimissuuntaa voi muuttaa kosketusnäytön kosketuseleillä, ja kuvakulmaa voi muuttaa kiihtyvyysanturin avulla laitetta kallistamalla. Sovelluksessa soitetaan taustamusiikkia, ja käyttöliittymän painikkeet reagoivat ääniefekteillä.

Yksikkötestit ja testisovellus ajettiin Samsung Galaxy S III -puhelimella, joka toimi projektin testilaitteena. Puhelimessa oli Android 4.2.2:een perustuva CyanogenMod 10.1 -käyttöjärjestelmä. Testisovellus toimi erinomaisesti, mutta sillä ei voitu testata pelimoottorin suorituskykyä. Ennen kuin pelimoottoria voidaan hyödyntää todellisessa pelinkehityksessä, on sitä testattava monilla erilaisilla Android-laitteilla ja -käyttöjärjestelmäversioilla mahdollisten yhteensopivuusongelmien löytämiseksi.

7 POHDINTA

Pelimoottorin, kuten muidenkin suurten ja monimutkaisten ohjelmistojärjestelmien, suunnittelu on tärkeää ennen toteuttamista. Pelimoottori koostuu useasta alijärjestelmästä, jotka koostuvat monista komponenteista. Huolellinen ohjelmistosuunnittelu takaa pelimoottorin sulavan kehityksen. Järjestelmiä on pystyttävä kehittämään ja testaamaan samanaikaisesti muista järjestelmistä riippumatta. Täsmällinen suunnittelu ja ohjelmointitaidot eivät tietenkään takaa pelimoottoriprojektin onnistumista, vaan lisäksi tarvitaan tuntemusta muun muassa kohdealustojen ja grafiikkalaitteiston arkkitehtuureista.

Pelimoottorin kehittäminen oli mielenkiintoista ja hyvin opettavaista. Opinnäytetyön aikana opittiin paljon uutta muun muassa suurten ohjelmistojärjestelmien ja pelimoottorien arkkitehtuureista sekä erinäisistä pelimoottorin alijärjestelmistä ja komponenteista. Myös C++-ohjelmointitaidot kehittyivät valtavasti. Uutta tietoa saatiin sekä kirjallisuudesta että käytännöstä ja itse oivaltamisesta. Työstä saatu kokemus on varmasti hyödyllistä tulevaisuudessa.

Opinnäytetyön aikana toteutettu pelimoottori ei ole vielä valmis. Jopa nykyisessä laajuudessaan se oli liian iso projekti opinnäytetyöksi. Pelimoottoriin toteutettiin kaikki suunnitellut alijärjestelmät ja komponentit onnistuneesti. Alustariippumattomuus onnistuttiin toteuttaa niin, että uusia alustoja voidaan tukea vähällä vaivalla. Pelimoottorista jäi kuitenkin puuttumaan joitain oleellisia pelinkehitystä helpottavia komponentteja. Se ei tarjoa esimerkiksi pääsilmukkaa, pääsilmukan ajanhallintaa, fysiikkaa tai profilointia. Nämä toiminnot joudutaan lisäämään peliin itse kehittämällä tai ulkoisia kirjastoja hyödyntämällä. Pelimoottorin kehitystä jatketaan opinnäytetyön jälkeen, ja siihen aiotaan lisätä edellä mainittujen toimintojen lisäksi muun muassa renderöinnin optimointijärjestelmä, verkkopelijärjestelmä ja pelilogiikan apujärjestelmä. Pelimoottori todennäköisesti julkaistaan avoimen lähdekoodin projektina kaikkien tutkittavaksi ja hyödynnettäväksi.

LÄHTEET

Aarnio, T., Miettinen, V., Pulli, K., Roimela, K. & Vaarala, J. 2008. Mobile 3D Graphics with OpenGL ES and M3G. Burlington, Massachusetts: Morgan Kaufmann Publishers.

Android Developers 2014 a. Android Open Source Project. http://source.android.com/ (Luettu 3.11.2014).

Android Developers 2014 b. Android Security Overview. http://source.android.com/devices/tech/security/index.html (Luettu 3.11.2014).

Android Developers 2014 c. Activities. http://developer.android.com/guide/components/activities.html (Luettu 3.11.2014).

Android Developers 2014 d. Android NDK. http://developer.android.com/tools/sdk/ndk/index.html (Luettu 3.11.2014).

Android Developers 2014 e. Get the Android SDK. http://developer.android.com/sdk/index.html (Luettu 3.11.2014).

Android Developers 2014 f. NativeActivity. http://developer.android.com/reference/android/app/NativeActivity.html (Luettu 7.11.2014).

Android Developers 2014 g. Managing Projects. https://developer.android.com/tools/projects/index.html (Luettu 6.11.2014).

Android Developers 2014 h. Processes and Threads. http://developer.android.com/guide/components/processes-and-threads.html (Luettu 7.11.2014).

Bhattacharya, A., Goon, S. & Paul, P. 2012. History and Comparative Study of Modern Game Engines. http://bipublication.com/files/IJCMS-V3I2-2012-07.pdf (Luettu 2.8.2014).

Gregory, J. 2014. Game Engine Architecture, Second Edition. Boca Raton, Florida: CRC Press.

Hassell, J. 2010. Developing for the iPhone and Android: The pros and cons. http://www.computerworld.com/article/2518482/enterprise-applications/developing-for-the-iphone-and-android--the-pros-and-cons.html (Luettu 3.11.2014).

Juujärvi, J. 1999. Rinnakkaislaskenta: Jaetunmuistin koneet, säikeet. http://www.lce.hut.fi/teaching/S-114.240/k99/esitykset/Jouni_Juujarvi/ (Luettu 7.11.2014).

Libpng 2014. http://www.libpng.org/pub/png/libpng.html (Luettu 3.11.2014).

Lilly, P. 2009. Doom to Dunia: A Visual History of 3D Game Engines. http://www.maximumpc.com/?q=article/features/3d_game_engines (Luettu 31.7.2014).

Microsoft Developer Network 2014 a. Solutions and Projects. http://msdn.microsoft.com/en-us/library/b142f8e7.aspx (Luettu 8.11.2014).

Microsoft Developer Network 2014 b. Unit Testing. http://msdn.microsoft.com/en-us/library/aa292197%28v=vs.71%29.aspx (Luettu 10.11.2014).

QuartSoft 2013. Top 5 Mobile Platforms in 2013. http://quartsoft.com/blog/201308/top-5-mobile-platforms-2013 (Luettu 3.11.2014).

Smacchia, P. 2008. Layering, the Level metric and the Discourse of Method. http://codebetter.com/patricksmacchia/2008/02/10/layering-the-level-metric-and-the-discourse-of-method/ (Luettu 9.8.2014).

The Khronos Group 2009. OpenSL ES Specification. https://www.khronos.org/registry/sles/specs/OpenSL_ES_Specification_1.0.1.pdf (Luettu 10.11.2014).

The Open Group 2004. clock_getres. http://pubs.opengroup.org/onlinepubs/009695399/functions/clock_gettime.html (Luettu 7.11.2014).

Vs-android 2014. https://code.google.com/p/vs-android/ (Luettu 3.11.2014).

Ward, J. 2008. What is a Game Engine? http://www.gamecareerguide.com/features/529/?page=2 (Luettu 2.8.2014).