Lappeenrannan teknillinen yliopisto
Tuotantotalouden tiedekunta
Tietotekniikan koulutusohjelma

Kandidaatintyö

Valtteri Mehtonen

NATIIVIN ANDROID-PELIN KÄYTTÖLIITTYMÄN SKAALAUTUMINEN JA SUORITUSKYVYN OPTIMOINTI

Työn tarkastaja: Nuorempi tutkija Erno Vanhala

Työn ohjaaja: Nuorempi tutkija Erno Vanhala

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Tuotantotalouden tiedekunta
Tietotekniikan koulutusohjelma

Valtteri Mehtonen

Natiivin Android-pelin käyttöliittymän skaalautuminen ja suorituskyvyn optimointi

Kandidaatintyö

2014

29 sivua, 1 kuva, 1 tauluko, 3 liitettä

Työn tarkastaja: Nuorempi tutkija Erno Vanhala

Hakusanat: android, mobiili, skaalautuvuus, optimointi, peli, käyttöliittymä
Keywords: android, mobile, scalability, optimization, game, user interface

Tässä työssä esiteltiin Android laitteisto- ja sovellusalustana sekä kuvattiin, kuinka Android-pelisovelluksen käyttöliittymä voidaan pitää yhtenäisenä eri näyttölaitteilla skaalauskertoimien ja ankkuroinnin avulla. Toisena osiona työtä käsiteltiin yksinkertaisia tapoja, joilla pelisovelluksien suorituskykyä voidaan parantaa. Näistä tarkempiin mittauksiin valittiin matalatarkkuuksinen piirtopuskuri ja näkymättömissä olevien kappaleiden piilotus. Mittauksissa valitut menetelmät vaikuttivat demosovelluksen suorituskykyyn huomattavasti. Tässä työssä rajauduttiin Android-ohjelmointiin Java-kielellä ilman ulkoisia kirjastoja, jolloin työn tuloksia voi helposti hyödyntää mahdollisimman monessa eri käyttökohteessa.

ABSTRACT

Lappeenranta University of Technology
School of Industrial Engineering and Management
Degree Program in Computer Science

Valtteri Mehtonen

User interface scalability and performance optimization of a native Android-game

Bachelor's Thesis
29 pages, 1 figure, 1 table, 3 appendices

Examiner: Junior researcher Erno Vanhala

Keywords: android, mobile, scalability, optimization, game, user interface

This thesis described Android as a hardware- and software platform and presented how the user interface of a demo application can be scaled to remain usable on different display devices by using scaling factors and anchoring. In the second part of this thesis several simple methods for increasing the performance of the demo application were described. Lower resolution rendering buffer and hidden object culling were selected for closer measurements. The selected methods demonstrated significant performance benefits. This thesis is limited to Android programming using its native Java programming language so that the results can be benefited from as broadly as possible.

ALKUSANAT

Tämän kandidaatintyön empiirisen osan työ aloitettiin jo alkuvuodesta 2014 relevantin laitteistopäivityksen myötä, mutta vielä siinä vaiheessa empiirinen osa oli vasta pelinraakile, jolla ei ollut mitään yhteyttä tähän työhön. Kun keväällä pohdin mahdollisia aiheita kandityölle, palasivat ajatukset siihen. Koin tarpeelliseksi laatia opastusta modernien mobiilisovellusten kirjoittamiseen, ja kandidaatintyö aiheesta tuntui melko luonnolliselta tavalta tehdä vaadittavaa taustatutkimusta. Aiheelle myös löytyi nopeasti innostunut ohjaaja, ja niinpä tämä kandidaatintyö on nyt kirjoitettu osana Lappeenrannan teknillisen yliopiston alempaa tietotekniikan korkeakoulututkintoa.

Tekstin ensimmäisen version kirjoitin vasta alkukesästä. Kesän kiireiden takia en kyennyt palaamaan työn pariin kuin syksystä. Loppuvuoden aikana hioin tekstiä ohjaajalta ja tekniikan kirjoitusviestintä -kurssilta saadun palautteen perusteella.

Tahdon kiittää etenkin perhettäni, joka on tukenut opiskeluani kaikissa sen vaiheissa. Lisäksi kiitän etenkin työn ohjaajaa Ernoa työn kehittämiseen liittyvästä palautteesta.

SISÄLLYSLUETTELO

SYMBOLI- JA LYHENNELUETTELO

ADK	Android Development Kit
ADT	Android Development Tools
API	Application Programming Interface
FPS	Frames per Second

1 JOHDANTO

1.1 Tausta

Jatkuvasti käytössä olevien mobiililaitteiden määrä, saatavuus ja suorituskyky ovat viimeisen vuosikymmenen aikana lisääntyneet niin valtavasti, että mobiilialustat ovat muuttuneet suurimmaksi yksittäiseksi sovellusalustaryhmäksi. Alan nopean kehityksen ja käyttäjien suuresti vaihtelevien mieltymysten ja tarpeiden takia mobiilialustojen laitteisto on kuitenkin huomattavan sirpaloitunut etenkin tässä työssä tarkasteltavien Android-laitteiden osalta sekä näyttöjen koossa että suorituskyvyssä.

Tämä työ rakentuu natiivin Android-pelisovelluksen ympärille; työssä kerrotaan teoriaa, jonka kuvaamia tapoja demosovelluksessa käytetään. Työn tulosten halutaan olevan mahdollisimman laajasti sovellettavissa, joten demosovelluksessa käytetä mitään ADK:n (Android Development Kit) eli virallisen Androidin kehitystyökalupaketin ulkopuolisia kirjastoja toiminnallisuuden toteuttamiseen ja ohjelmointikielenä käytetään ainoastaan Androidille ominaista Java-kieltä.

1.2 Tavoitteet ja rajaukset

Tämän kandidaatintyön ensisijainen tavoite on selvittää, mitä kaikkea on otettava huomioon, kun natiivin Android-pelin halutaan toimivan käyttäjälle sulavasti eritasoisella mobiililaitteistolla pienistä puhelimista suuriin taulutietokoneisiin. Tarkemmin työssä siis perehdytään käyttöliittymäkomponenttien asemointiin ja pikselitarkkuuden skaalaamiseen sekä tutkitaan, millaisia kompromisseja pelin näyttävyydessä on suoritettava, jotta saavutetaan riittävä suorituskyky myös heikommalla laitteistolla. Lopulta testataan käytännön vaikutus pelin suorituskykyyn eri optimointiasetuksilla.

Täten tutkimuskysymyksiksi muodostuivat seuraavat:

- Mitä vähimmäistoimia tarvitaan, että käyttöliittymäkokemus pysyy yhtenäisenä eri laitteilla?
- Millä yksinkertaisilla tavoilla demosovelluksen suorituskykyä voidaan optimoida?

Ratkaisevasti eri käyttöjärjestelmien ja laitteiston tukeminen jätetään kuitenkin työn ulkopuolelle. Työssä keskitytään puhtaasti välttämättömyyksiin ja laitteistonäkökulmaan; varsinainen monialustainen kehitys eri ohjelmistoversioineen ei siten kuulu työn varsinaiseen fokukseen. Lisäksi vaikka työssä keskitytäänkin kannettaviin laitteisiin, jätetään akkukeston vaikutukset ja internetyhteyden saatavuus tarkastelun ulkopuolelle.

1.3 Työn rakenne

Luvussa 2 esitellään Android sovellusalustana ja kerrotaan esimerkkejä laitteistoista ja sen vaikutuksista sovelluksiin. Luvussa 3 käydään tarkemmin läpi skaalautuvuuden teoriaa sekä käyttöliittymissä että suorituskyvyn osalta. Luvussa 4 teoria laitetaan käytäntöön ja toteutetaan skaalautuva käyttöliittymä demosovellukseen sekä testataan suorituskykyä muutamalla eri laitteella ja sovelluksen asetuksella. Työn päätetään luvun 5 yhteenvetoon.

2 ANDROID

2.1 Esittely

Android on ylivoimaisesti maailman yleisin sekä käytetyin mobiilikäyttöjärjestelmä. International Data Corporationin julkaiseman lehdistötiedotteen mukaan Androidin lukumäärällinen markkinaosuus viime vuoden kolmannella neljänneksellä oli 81 %, tehden siitä selkeästi suurimman yksittäisen mobiilialustan [1]. Androidin suuren suosion taustalla on sen avoimuus ja ilmaisuus, jolloin mikä vain siitä kiinnostunut instanssi voi ottaa sen vapaasti käyttöönsä. Ja vapaa käyttö on todellakin mahdollistanut Androidin käytön mitä erilaisimmilla laitteilla matkapuhelimista taulutietokoneisiin ja kodinkoneisiin. Tämä ei kuitenkaan ole pelkästään hyvä asia, sillä pitkälle sirpaloituneen laitteistojoukon tukeminen tuo omat haasteensa vaikka nimellisesti kyseessä onkin koko ajan sama alusta.

Yleisyyden lisäksi toinen kiinnostava piirre Androidissa on se, että jo sen ensimmäisessä Androidia käyttävässä laitteessa, HTC Dreamissa, oli kosketusnäyttö. Kosketusnäyttökeskeinen suunnittelu näkyykin Androidin selkeässä käytettävyydessä kosketusnäytöllisillä laitteilla. Esimerkiksi kilpaileva Symbian-käyttöjärjestelmä on alun perin luotu fyysisellä näppäimistöllä varustettuja laitteita varten ja kosketusnäyttöjen tuki lisättiin siihen vasta jälkikäteen.

Ohjelmistokehittäjän elämää suuresti helpottava Androidin ominaisuus on sen ohjelmistorajapintojen (Application Programming Interface, API) vakaus. Vanhoille rajapinnoille versioille laadittu sovellus toimii yleensä suoraan myös kaikkein uusimmilla laitteilla ja käyttöjärjestelmäversioilla. Vastaavasti uudet sovellukset on helppo koodata tukemaan myös vanhempia laitteita, sillä Androidin kehitysympäristö Android Development Tools (ADT) antaa sovellusprojektia luodessa valita kohteena olevan API-tason ja huomauttaa, mikäli yritetään käyttää API-kutsuja, joita kohteena oleva taso ei tue.

2.2 Laitteistot

Kuten jo aiemmin mainittu, Android on vapaa alusta ja siten sitä tukevia laitteita on ilmestynyt ajan saatossa lukuisilta eri laitevalmistajilta. Eri laitteissa on käytössä eri versioita sekä itse käyttöjärjestelmästä, mutta myös lukemattomia eri laitteistokomponentteja. Tämän lisäksi monet suuret laitevalmistajat ovat kehittäneet omia

muunnelmia sekä itse käyttöjärjestelmästä että sen päällä toimivasta käyttöliittymästä. Esimerkiksi suurin [1] Android-laitteiden valmistaja Samsung on kehittänyt omissa tuotteissaan käytettävän TouchWiz-käyttöliittymän ja sovellusperheen, jolla se pyrkii erottumaan kilpailijoistaan.

Kaikkein näkyvin osa laitteita on näyttö, jonka kautta käyttäjä kokee ja vaikuttaa sovelluksiin. Sovelluskehittäjän kannalta näytön tärkeimmät ominaisuudet ovat sen pikselitiheys ja fyysinen koko kahdestakin syystä. Tarkemmalla näytöllä on mahdollista esittää enemmän tietoa, mutta samalla on huolehdittava siitä, että näyttöobjektit pysyvät riittävän suurina saman tarkkuuden, mutta pienemmän pikselikoon näytöillä. Sormella käytettävien kosketusnäyttöjen tapauksessa on myös pidettävä kosketuspinnat riittävän suurina, sillä sormi ei ole tarkka osoitinlaite hiireen tai kynään verrattuna.

Näytön jälkeen yksi tärkeimmistä osista on suoritin, jonka laskentakapasiteetti on yhdessä näytönohjaimen kanssa tärkein pelisovellusten monimutkaisuutta rajoittava tekijä. Vaihteleva suorituskyky kannattaa pitää mielessä etenkin ohjelmiston kehitysvaiheessa. Mahdollisimman lyhyttä iteraatioaikaa hakiessa syntyy helposti kiusaus kehittää ohjelmistoa uusimmalla ja nopeimmalla laitteella ja naiiveilla algoritmeilla. Kannattaa kuitenkin aina kirjoittaa laadukasta koodia, sillä laajaan käyttäjäkuntaan tähtäävän sovelluksen on toimittava sulavasti myös heikommilla laitteilla.

Siinä missä suoritin on enemmän sovelluksen mekaniikan mahdollistaja, näyttävyys on puolestaan pitkälti grafiikkapiiriin kytköksissä oleva ominaisuus. Raa'alla suorituskyvyllä on oma roolinsa, mutta tärkeää on myös piirtokirjaston rajapinnan versio. Androidin dokumentaation mukaan suurin osa [2][3] aktiivisesti käytetyistä Android-laitteista tukee OpenGL ES -kirjaston versiota 2.0. Aivan uusimmat laitteet ja Androidin versiot tukevat myös versiota 3.0, ja kirjoitushetkellä näiden laitteiden arvioitu osuus aktiivisista laitteista on noin neljännes. Uudemman rajapinnan käyttöönottoa kannattaa kuitenkin aina tarkkaan miettiä, sillä pelkästään nopeuden tavoitteluun se saattaa joskus olla väärä ratkaisu. Vaikka uudempi rajapinta auttaakin käyttämään laitteistoa tehokkaammin esimerkiksi tessaloinnin, geometrian instansoinnin ja joustavankokoisten tekstuureiden [4] avulla, sitä ei vielä tueta kuin uusimmilla ja nopeimmilla laitteilla. Näillä laitteilla on kuitenkin yleensä riittävästi suorituskykyä tehdä asiat vanhalla ja alaspäin yhteensopivalla tavalla.

Tämän lisäksi laitteissa on eroja muistien määrässä. Suurempi muisti mahdollistaa tarkemmat tekstuurit, monimutkaisemmat objektit ja nopeammat tietorakenteet. Kuitenkin esimerkiksi tämän työn demosovelluksessa rajoitutaan ennemmin datan käsittelynopeuteen kuin sen määrään. Toisaalta kun laitteessa on suuri määrä muistia, on käyttöjärjestelmän mahdollista pitää käynnissä monta sovellusta samaan aikaan. Käytännössä tämän huomaa helpoimmin vähän muistia sisältävissä laitteissa vaihtaessa eri sovellusten välillä. Aivan äskettäin käytetty sovellus saattaakin käynnistyä nollasta, ja käyttäjä joutuu odottelemaan mahdollisesti pitkänkin ajanjakson sovelluksen ladatessa kaikki resurssinsa ennen kuin sitä voi taas käyttää. Erityisen ongelmallista on, jos sovellusta ei ole asianmukaisesti toteutettu tallentamaan tilaansa käyttökertojen välillä ja käyttäjä menettää sen takia keskeneräisiä tietoja.

3 SKAALAUTUVUUS JA OPTIMOINTI TEORIASSA

Tässä työssä skaalautuvuudella tarkoitetaan sovelluksen kykyä mukauttaa toimintaansa mahdollisimman huomaamattomasti käytössä olevan laitteiston mukaan. Tällöin esimerkiksi graafisten efektien laatua voidaan karsia heikompitasoisella näytönohjaimella varustetussa laitteessa niin, että ruudunpäivitysnopeus pysyy sulavana ja sovelluksen pääasiallinen käyttötarkoitus ei häiriinny.

3.1 Android-pelien haasteet

Android on alustana varsin nuori, mutta grafiikkasovelluksissa käytetään kuitenkin pitkälti samoja taustatekniikoita kuin vakiintuneemmilla alustoilla, kuten PC:llä. Kosketusvetoiset mobiilialustat vaativat kuitenkin erilaisen ajattelutavan käyttöliittymiin kuin mihin on perinteisesti totuttu. Fyysisiä näppäimiä on harvoin tarjolla, joten kaiken vuorovaikutuksen on tapahduttava yleensä verrattain pienen näytön ja epätarkan osoitinlaitteen avulla.

Käyttöliittymällä on suuri vaikutus loppukäyttäjän käsitykseen sovelluksesta, joten käyttöliittymän ulkoasun tulisi vastata muun sovelluksen tematiikkaa. Androidin kehitystyökaluihin toki kuuluu hyvä ja monipuolinen käyttöliittymien suunnitteluun tarkoitettu työkalu, mutta sillä luotuja käyttöliittymiä ei voi upottaa osaksi sovelluksen OpenGL-kontekstia. Niinpä lisähankaluuksia aiheuttaa kattavien ja vakiintuneiden kosketusohjausta tukevien OpenGL-pohjaisten käyttöliittymäkirjastojen puute. Useita avoimen lähdekoodin käyttöliittymäkirjastoja on toki saatavilla, mutta ne ovat joko rajoittuneita ominaisuuksiltaan, eivät sisällä riittävää dokumentaatiota tai ovat rajoittuneita ulkoasun muokattavuudessa. Toisaalta kutakin tapausta varten alusta asti luotu käyttöliittymä on yleensä ylivertainen muokattavuudellaan, mutta sellaisen laadukas toteutus voi pahimmillaan viedä enemmän aikaa kuin sovelluksen muun toiminnallisuuden toteutus yhteensä.

Uuden tekniikan nopea kehitys sekä PC:llä että pelikonsoleilla on saanut käyttäjän odottamaan suuria myös mobiilipelien grafiikoilta, mutta mobiilialustojen isoja haasteita on rajoittunut suorituskyky ja sen tuomat käytännön ongelmat. Laitteet itsessään ovat kuitenkin jo osaratkaisu ongelmaan pienten näyttöjensä ansiosta. Ihmissilmä ei kykene kovin helposti erottamaan hyvin pieniä kohteita, joten niitä on turha edes yrittää näyttää

tarkasti. Käytännössä voidaan siis käyttää matalamman tarkkuuden tekstuureita, malleja ja varjostimia sekä approksimaatioita tarkemmista algoritmeista.

3.2 Käyttöliittymän skaalautuvuus

Hyvä ja skaalautuva käyttöliittymä on näyttötarkkuudesta riippumaton ja kykenee tarkoituksenmukaisesti skaalaamaan elementtiensä kokoa ja mahdollisesti jopa rakennetta kulloinkin käytössä olevan näytön mukaan. Hyvä käytettävyys on saavutettavissa yhdistelemällä useita eri skaalaustapoja.

Kaikkein yksinkertaisin tapa rakentaa käyttöliittymä on staattinen asettelu. Siinä jokaiselle käyttöliittymäelementille määritetään muuttumaton sijainti manuaalisesti. Asettelu on helppo toteuttaa, mutta sen käytännöllisyys hajoaa heti kun käyttöliittymälle tarjolla olevan tilan koko muuttuu. Käyttöliittymä ei esimerkiksi enää peitä koko kuva-alaa tai osa siitä jää sen ulkopuolelle. Yksi vaihtoehto olisi tietysti laatia staattinen asettelu jokaiselle eri näytölle, mutta laitteiden paljous tekee tehtävän melko mahdottomaksi. Naiivi jatkumo staattiseen asetteluun onkin laatia käyttöliittymä referenssikoolle ja skaalata sitä näyttökoon suhteessa, jolloin kuva-alaan liittyvät ongelmat poistuvat. Uusia ongelmia kuitenkin seuraa elementtien vääristyessä näytön kuvasuhteen muuttuessa tai skaalausalgoritmissa, sillä kaikki resoluutiot eivät ole toistensa kerrannaisia ja tällöin voi syntyä lopputuloksen laatua heikentäviä artefakteja.

Järkevämpää onkin ankkuroida eri käyttöliittymäelementit määrättyihin referenssipisteisiin, kuten näytön reunoihin, keskikohtiin ja muihin elementteihin. Tällöin kaikki elementit ovat omilla luonnollisilla paikoillaan, mutta kuitenkin säilyttävät terävyytensä. Ankkurointi ei kuitenkaan huomioi elementtien käytettävyyden vaatimaa tilaa. Jos asettelu on laadittu pienen tarkkuuden näytölle, voivat näyttöelementit kutistua suuren pikselitiheyden näyttölaitteella niin pieniksi, että niiden käyttäminen on hankalaa.

Luonnollinen ratkaisu onkin yhdistää ankkurointi ja skaalautuvuus siten, että käyttöliittymä on pohjaltaan ankkuroitu, mutta elementtien kokoa skaalataan diskreetein askelin näyttötiheyden funktiona. Ja tämä onkin juuri se tapa, jota virallinen Androidin kehitysopas suosittelee [5]. Kunkin laitteen näytön tiheys ja koko luokitellaan kukin yhteen

ennalta määrättyyn luokkaan ja käyttöliittymä suunnitellaan keskitason luokitukselle laitteistoriippumattomien pikseleiden avulla. Käyttäessä natiivia käyttöliittymäkirjastoa tapahtuu skaalaaminen automaattisesti ajon aikana käytössä olevan laitteen luokituksen mukaan taulukon 1 mukaisilla diskreeteillä skaalauskertoimilla [6]. Muussa tapauksessa käytössä olevaa laitetta vastaava skaalauskerroin on mahdollista noutaa DisplayMetrics luokan density-kentän avulla ja skaalata elementtien koko manuaalisesti sen avulla.

Taulukko 1. Skaalauskertoimia eri näyttötiheyksille.

Nimike	**Skaalauskerroin**	**Huomioita**
ldpi	0.75	
mdpi	1.00	Oletus, Testilaite 2
hdpi	1.50	
xhdpi	2.00	
xxhdpi	3.00	Testilaite 1
xxxhdpi	4.00	

Kuvatuilla toimilla saavutetaan vähintään kohtuullinen toimivuus kaikilla eri näyttölaitteilla ilman, että jokainen tapaus on erikseen manuaalisesti testattava. Etenkin pelisovelluksissa, joissa näytettävät elementit yleensä vievät suhteellisen pienen näyttöalan tämä tapa toimii hyvin. Suurilla näytöillä elementit reunustavat pelikenttää olematta liian isoja ja ovat pienemmillä näytöillä suhteessa isompia jättäen silti tilaa pelikentälle. On kuitenkin tilanteita, joissa skaalaaminen ei tuota riittävän hyviä tuloksia. Jos esimerkiksi iso määrä vierekkäisiä elementtejä kutistuisi pienellä näytöllä liikaa, voi olla tarpeen laatia erillinen asettelu kyseistä kokoluokkaa varten.

3.3 Suorituskyvyn optimointi

Nimensä mukaisesti suorituskyky kuvaa sovelluksen kykyä suorittaa sille annettu tehtävä. Pelin tapauksessa tehtävänä on simuloida pelimaailman kulkua uskottavasti ja piirtää kuvia pelitilanteesta mahdollisimman nopeasti, jotta saavutetaan sulava käyttökokemus. Tavoitteena on luoda uskottava virtuaalimaailma, joten oleellista on, että etenkin interaktiiviset objektit ja kameran näkymä päivitetään nopeasti ja tarkasti. Sen sijaan

maailma joka jää näkymän ulkopuolelle voi aivan hyvin lakata olemasta, sillä kukaan ei näe sen puuttumista. Näkymättömien kohteiden piirtämättä jättäminen eli *culling* onkin monissa sovelluksissa yksi tärkeimpiä suorituskykyoptimointeja [7]. Kuitenkin myös näkymän sisäpuolelle jäänyttä pelimaailmaa on mahdollista optimoida. Pelaajan katse pyrkii keskittymään huomiota kaipaavien objektien ympärille ja tarkan näön ulkopuolisen alueen piirron laatua on mahdollista laskea pelikokemuksen suuresti kärsimättä [8].

Peleissä laskennallinen rasittavuus jakautuu karkeasti kahteen osaan. Toisen puolikkaan muodostaa pelilogiikka, joka pyörii pääsääntöisesti suorittimella ja toinen puolikas, eli grafiikan piirto, pääosin grafiikkapiirillä. Sovelluksen suorituskykyyn vaikuttaa oleellisesti suoritusympäristö sekä käytetty kääntäjä tai virtuaalikone ja sen suorittamat optimoinnit, mutta sen lisäksi tarvitaan kokonaisvaltaisempaa katsausta järjestelmään ja sen käyttämiin tietorakenteisiin ja algoritmeihin. Esimerkiksi tilojen osittelun avulla on mahdollista optimoida ruudulla näkyviin objekteihin pääsyä ja ohittaa näkymän ulkopuolella olevien objektien käsittely. Osittelun kannattavuus on kuitenkin tapauskohtaista, sillä joskus piilossa olevien alueiden halutaan kuitenkin edelleen päivittyvän tai niiden piirtokäskyjen muodostus ja suorittaminen on niin nopeaa, että ohittamisen vaatima lajittelu hidastaa kokonaisuutta. Kaikessa toiminnassa kannattaa kuitenkin muistaa vanha sananlasku: ennen kuin jotain voi optimoida, pitää se pystyä mittaamaan [9].

Uusia työkaluja suoritinkuorman vähentämiseen on tullut ohjelmoitavien näytönohjainten varjostinten myötä. Heikkokin näytönohjain sisältää yleensä moninkertaisesti laskentakapasiteettia saman laitteen suorittimeen verrattuna. Rakenteensa vuoksi varjostinyksiköitä ei kuitenkaan voi käyttää kuin tietynlaisiin toistuviin tehtäviin. Esimerkiksi partikkelitehosteessa kunkin partikkelin paikka voidaan toki laskea joka ruutu uudestaan suorittimen avulla, mutta sijainti on mahdollista laskea myös suoraan verteksivarjostimella. Jokaisella partikkelilla on vain aloituspaikkansa ja syntyhetken aika, ja varjostimessa kunkin hetken paikka lasketaan nopeuden ja aikaleiman perusteella samalla kaavalla jokaiselle partikkelille.

Grafiikassa suurin kuorma on piirtokäskyjen määrä ja monimutkaisuus [10], sillä jokainen käsky vaatii erillisprosessointia sekä suorittimelta että grafiikkapiiriltä itse käskyn suorittamisen lisäksi. Yhdistämällä useita pieniä grafiikkaobjekteja isommiksi

kokonaisuuksiksi tämän erillisprosessoinnin määrää saadaan karsittua huomattavasti. Toinen huomattava graafinen kuorma on grafiikkaliukuhihnan loppupuolella oleva rasterointioperaatio, jonka kuormittavuus on helppo havaita tilanteissa, joissa on paljon ylipiirtoa [9]. Tällaisia tilanteita ovat esimerkiksi paljon partikkeliefektejä sisältävät näkymät. Suorituskykyongelman voi ratkaista ei-toivotusti vähentämällä efektien kokoa ja määrää, ja siten näyttävyyttä. Toinen tapa on siirtää paljon ylipiirtoa sisältävien kohtien piirto matalamman tarkkuuden näyttöpuskuriin, joka sitten skaalataan takaisin täyteen tarkkuuteen ja piirretään muun näkymän päälle [11].

4 SKAALAUTUMISEN TOTEUTUS JA SUORITUSKYVYN MITTAUS DEMOSOVELLUKSELLA

Tässä työssä kuvattujen tekniikoiden testaamiseen käytettiin alkeellista pelisovellusta, joka on luotu käyttäen Androidin kehitystyökalupakettia. Demosovelluksessa on tarkoituksena rikkoa maailmasta löytyvät tiilet niitä koskettamalla. Ensimmäisestä kosketuksesta tiilet virittyvät ja toisesta räjähtävät. Räjähdykset leviävät viritetyistä tiilistä toisiin, joten paikoin pelissä voi olla suuri määrä yhtäaikaisia vuorovaikuttavia tiiliä ja räjähdystehosteita.

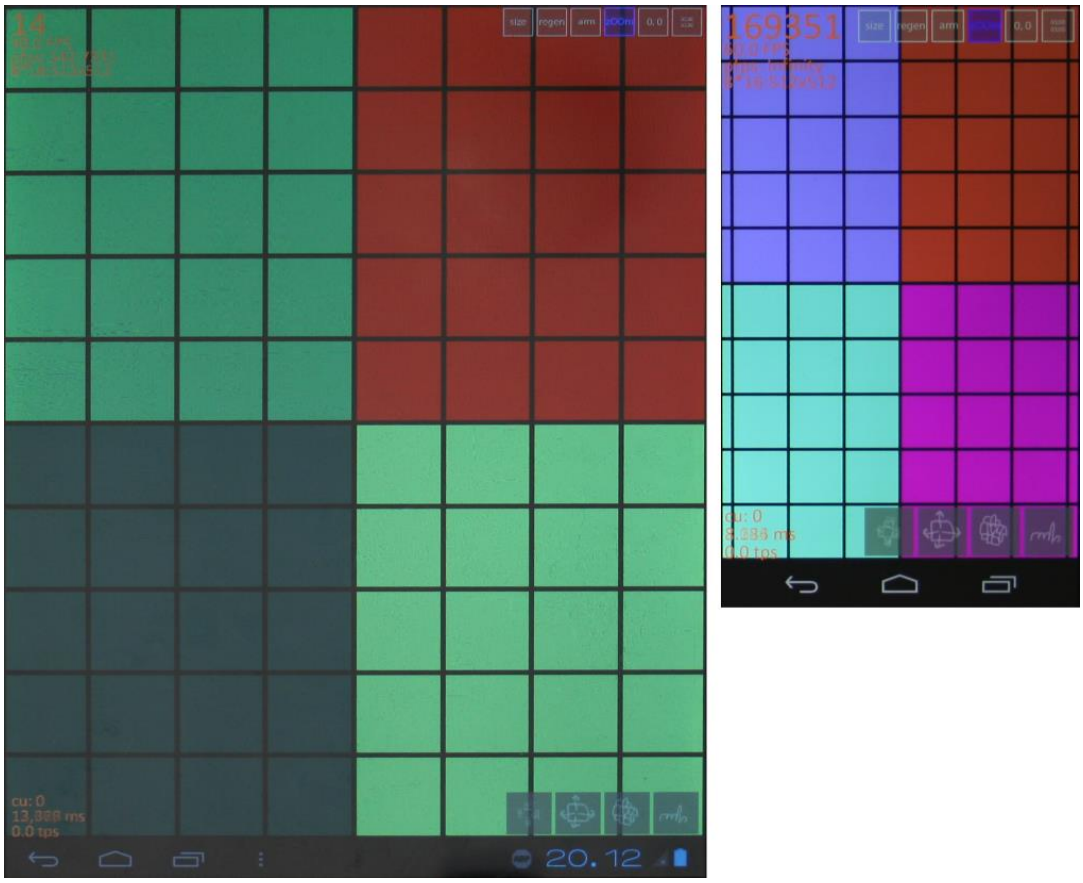
Sovellukseen kuuluu myös useita pelin hallintaan vaikuttavia painikkeita ja tietoa antavia tekstikenttiä. Näiden elementtien on tarkoitus pysyä käytettävissä vaikka näytön ominaisuudet vaihtelevatkin.

4.1 Testilaitteet

Pääasiallisena kehityslaitteena käytössä oli varsin uusi ja edistynyttä tekniikkaa edustava LGE Nexus 5 -älypuhelin ja toissijaisena laitteena hieman vanhempi, entry-tason tekniikkaa sisältävä Blaupunkt Endeavour 800 -tabletti. Testilaitteiden tärkeimmät ominaisuudet on kuvattu liitteessä 1.

4.2 Käyttöliittymän skaalautuvuus käytännössä

Demosovelluksen käyttöliittymä on hyvinkin pelkistetty, mutta siitä käy selkeästi ilmi erinäisten luvussa 3 kuvailtujen käyttöliittymän skaalautumiseen vaikuttavien toimintojen vaikutus. Kuvassa 1 on esillä sama käyttöliittymä, mutta laitteessa olevan näytön koko ja tarkkuus vaihtelee.



Kuva 1. Demosovellus ja sen käyttöliittymä testilaitteilla (muokattu valokuva).

Huomattavissa on etenkin ruudun oikealta puolelta löytyvien painikkeiden absoluuttisen koon pysyminen lähes vakiona. Tämä saavutetaan luvussa 3.2 kuvatulla näytön pikselitiheydestä johdetulla käyttöliittymäelementtien skaalauskertoimella. Pieni varianssi syntyy teoriaosuudessa selitetystä näyttötiheyden luokittelusta ja sen vaikutuksena skaalauskertoimen diskreettiin askellukseen. Toinen merkittävä huomio on käyttöliittymäelementtien ankkuroinnissa toisiinsa. Vaikka absoluuttiset pikselikoordinaatit muuttuvatkin, eivät elementit siltikään mene toistensa päälle, jätä suuria rakoja eivätkä muillakaan tavoin poikkea referenssilaitteen asettelusta.

Mahdollinen jatkokehityksen suunta olisi skaalata käyttöliittymää tässä kuvatun normalisaation jälkeen vielä entistäkin suuremmaksi isoilla näytöillä, esimerkiksi suoraan näytön koon mukaan. Yleensä isonäyttöisten laitteiden käyttöasento ei salli niin tarkkaa valintaa kuin asento pieniä laitteita käytettäessä. Myös käyttäjien odotukset voivat poiketa huomattavasti eri laitetyypeillä, tai ainakin itse odottaisin sovelluksen käytettävyyden mukautuvan käytössä olevan laitteen mukaan.

4.3 Suorituskyvyn optimointi käytännössä

Demosovellukseen tapauksessa huomio kiinnitettiin kehityksen aikana havaittuihin suurimpiin ongelmakohtiin. Näitä olivat suurikokoisten partikkelitehosteiden raskauteen liittyvät ongelmat ja pelikentän laajentamisen aiheuttama piirtokäskyjen määrään kasvaminen.

Demosovelluksen kehitysvaiheen testeissä havaittiin, että partikkelitehosteet muuttuivat sitä raskaammiksi mitä lähemmäs pelin kamera oli asetettu. Kyseessä oli siis selkeästi partikkelitehosteiden rasterointiin liittyvä ongelma. Suoritin ja näytönohjain kykenivät kyllä laskemaan jokaiselle partikkelille sijainnin riittävän nopeasti, mutta partikkelien piirtämisessä näytönohjaimella rasterointivaihe vei poikkeavissa määrin suoritusaikaa ja aiheutti siten piirron hidastumisen. Kun käyttöön otettiin oikeaa kuva-alaa huomattavasti matalampitarkkuuksinen piirtopuskuri partikkeleita varten, oli odotettavissa huomattava parannus ruudunpäivitysnopeuteen. Hypoteesi pitääkin paikkansa tarkasteltaessa mitattuja tuloksia.

Toinen havaittu suorituskykyongelma liittyi suuren pelikentän aiheuttamaan piirtokäskyjen määrän nousemiseen. Suoritin on kyllin tehokas päivittämään suurenkin pelikentän tilan riittävän nopeasti, mutta käytössä ollut naiivi tapa piirtää kaikki näkymättömissäkin olevat kentän lohkot aiheutti nopeasti ikävää hidastumista piirtokäskyissä. Kyseessä oli selkeä ongelma, mutta koska sovelluksen kehitys oli vielä alkuvaiheissa, ei lopullisia päätöksiä mahdollisesta pelikentän koosta haluttu tehdä ja tästä syystä huomattavaa aikaa ei haluttu käyttää tehokkaaseen tilanosittelualgoritmiin. Testimielessä käyttöön otettiin kuitenkin hyvin yksinkertainen, mutta testatuilla lohkomäärillä yllättävän hyväksi osoittautunut algoritmi, joka yksinkertaisesti vertaa piirrettävän lohkon horisontaalisia koordinaatteja kameran näkymään ja ohittaa piirtokäskyn jos lohko ei näy näkymässä. Jatkokehityksessä tehokkaan tilanosittelualgoritmin käyttöönotto olisi toki yksi tärkeimmistä ominaisuuksista. Peliteollisuudesta löytyy onnistuneita tapoja asian toteuttamiseen esimerkiksi dynaamisen lohkojen lataamisen avulla kuten Minecraft-pelissä [12]. Tällöin ei ole edes tarpeen suorittaa muistissa olevien lohkojen lajittelua piirtokäskyjen generoimista

tai generoimatta jättämistä varten. Vain lähellä olevat lohkot pidetään ladattuina muistiin ja tarvittaessa uusia lohkoja ladataan levyltä tai generoidaan lennosta.

4.4 Suorituskyvyn mittaukset testilaitteilla

Ruutua sekunnissa (FPS, Frames per Second) on loppukäyttäjän kannalta tärkein suorituskykyindikaattori. Mitä enemmän ja tasaisemmin sovellus kykenee piirtämään ruutuja, sitä sulavampi pelikokemus. Suurin osa peleistä on miellyttäviä ruudunpäivitysnopeuden ollessa yli 30 kuvaa sekunnissa, mutta liikkeiden sulavuuden takia on suositeltavaa tavoitella 60 ruutua sekunnissa. 60 Hz on yleisimpien käytössä olevien mobiilinäyttöjen korkein tuettu ruudunpäivitysnopeus.

Järjestelmän suorituskyky on yleensä suoraan verrannollinen ruudunpäivitysnopeuteen [13]. Joskus voi kuitenkin olla käytännöllisempää mitata ruudunpäivitysnopeuden käänteisarvoa ruutuaikaa eli aikaa, joka menee yhden ruudun käsittelyyn. Syitä on esimeriksi absoluuttisen vertailun mahdollistaminen tai järjestelmän suorituskyvyn mittaamiseen liittyvät ongelmat. Androidilla ei ole mahdollista helposti kytkeä ruututahdistusta pois päältä, ja tällöin ruudunpäivitysnopeus on rajattu näytön ruudunpäivitysnopeuteen. Tällöin syystä tarpeeksi tarkkojen tulosten saamiseksi yksinkertaisissa näkymissä osassa mittauksissa on pakko käyttää ruudun muodostukseen kulunutta aikaa mittarina. Tällöin ongelmia kuitenkin aiheuttaa grafiikkajärjestelmän rakenne, sillä ruutuajan tarkempi mittaaminen vaatii OpenGL:n piirtoliukuhihnan tyhjentymisen odottamisen.

Grafiikkaohjaimet koostuvat useista peräkkäin toimivista vaiheista siten, että uutta ruutua saatetaan alkaa käsittelemään jo ennen kuin vanhempi ruutu on täysin valmis. Tämä tehostaa resurssien käyttöä ja siten ruudunpäivitysnopeutta. Resurssien jakamisesta ei kuitenkaan päästä hyötymään mikäli piirtoliukuhihna täytyy tyhjentää eri ruutujen. Tällöin ruutuajan tarkka mittaaminen ei välttämättä anna niin suorituskykyisiä tuloksia kuin mihin laite muutoin kykenisi. Tämän lisäksi on huomattava, että liukuhihnan synkronointiin käytettävä glFinish() käsky aiheuttaa monesti ylimääräistä synkronoinnin vaatimaa hidastumista ja lisäksi se ei välttämättä toimi oletetusti kaikilla alustoilla. Jotkin näytönohjaimet tai ajuriversiot saattavat jatkaa kuvan muodostusta käskyn suorittamisen

jälkeenkin ja siten vääristää tuloksia jossain määrin. Tällöin tulokset eivät ole suoraan vertailukelpoisia eri laitteiden välillä.

Myös Javan automaattinen roskienkeräys vaikuttaa tuloksiin tuomalla pientä ylimääräistä varianssia, mutta sen vaikutus on n. 2 - 10 % luokkaa [14][15] riippuen ohjelmaa suorittavan virtuaalikoneen eri asetuksista. Testeissä muun muassa tätä roskienkeräyksen aiheuttamaa varianssia pyritään kompensoimaan ottamalla useamman testin keskiarvo lopullisiin tuloksiin.

Demosovelluksesta testattavaksi valittiin kaksi eri ominaisuutta suuren vaikutuksensa ja yleisen sovellettavuutensa perusteella. Ensimmäinen on pelikentän koon vaikutus ruutuaikaan eri culling-asetuksilla ja toinen partikkeliefektien piirtopuskurin koon vaikutus ruudunpäivitysnopeuteen. Kuten aiemmin luvussa kolme on todettu, lähes kaikki pelit hyötyvät näkymättömien objektien piilottamisesta ja rasterointikuorman vähentämisestä ja testeissä oletetaan nähtävän huomattavia saantoja suorituskyvyssä. Mittaukseen käytetään sovellukseen sisäänrakennettuja metriikoita.

Pelikentän mittauksissa mitattavana suureena on ruutuaika kolmessa eri tapauksessa ja kahdessa eri näkymässä kolmella eri pelikentän koolla(4,8,16 lohkoa per x/y-akseli ja 32*32 ruutua per lohko). Mittaustulos on 500 ruudun keskiarvo. Tapauksina on x-tasoon rajoittuva näkymän ulkopuolella olevien lohkojen piilotus glFinish() käskyn kanssa kun kaikki lohkot piirretään(Nocull) ja tapauksessa jossa aiemmin kuvattua piilotusalgoritmia käytetään(Cull). Molemman tapauksen kohdalla on lisäksi erikseen mitattu tilanteet, joissa kameran näyttämän näkymän laajuutta muutetaan ja täten tuodaan esiin piilotuksen vaikutus muutoin samoilla asetuksilla. Testitulokset on kirjattu liitteeseen kaksi.

Partikkelitehosteiden suorituskyvyn mittaamiseen puolestaan käytetään puolisynteettistä testiä, jossa erittäin iso määrä pelikenttää ensin valmistellaan räjäytystehostetta varten ja mitataan räjähdystehosteen keskimääräinen FPS tapahtuman alusta viimeisenkin partikkeliefektin loppumiseen. Tulokset mitataan kolmella eri partikkeliefektien piirtopuskurin koolla: 256*256 pikseliä, 512*512 pikseliä ja natiivi eli tilanne, jossa piirron tarkkuutta ei erikseen madalleta. Testi toistetaan kolmesti ja tuloksista lasketaan aritmeettinen keskiarvo. Testitulokset on kirjattu liitteeseen kolme.

4.5 Suorituskykymittausten analyysi

Tarkastellaan ensin piirtokäskyjen karsimisen vaikutuksia. Liitteeseen 2 on koottu edellä kuvatusta testistä mitatut ruutuaikametriikat. Kun suhteellista ruutuaikaa verrataan identtisissä tapauksissa siten, että vain culling-asetusta vaihdetaan, saadaan liitteessä erillisiin taulukkoihin lasketut suorituskyvyn paranemista kuvaavat prosenttiluvut. Luvuista on helppo huomata vahva trendi, jonka mukaan jopa kuvatulla pienellä ja varsin naiivilla culling-optimoinnilla on saavutettavissa huomattava parannus pelin suorituskyvyssä. Saavutettu etu on sitä parempi mitä enemmän piirtokäskyjä saadaan ohitettua: perusnäkymässä ruudulla on vain muutama lohko kerrallaan ja tällöin ohitettujen piirtokäskyjen määrä on suurimmillaan. Sama näkyy myös nopeusedun kasvamisena kentän koon kasvaessa. Suuremman alueen pelikenttää näyttävässä kaukonäkymässä tarvitaan enemmän piirtokäskyjä, mutta piilotettavia lohkoja on silti etenkin suurimman kenttäkoon tapauksessa. Testilaitteen 2 tapauksessa havaitun epäjatkuvuuden mahdollisia syitä ovat aiemmin kuvatut epätarkkuutta aiheuttavat tekijät ja kuvasuhteen vaikutukset näytettävään pelialueeseen. Myös järjestelmän taustaprosessit voivat aiheuttaa ajoittaista epätarkkuutta mittauksiin.

Toisessa testissä puolestaan testattiin madalletun tarkkuuden piirtopuskurin käyttämistä partikkelitehosteiden rasterointiin. Liitteessä 3 kuvatuista tuloksista nähdään taas hyvin helposti trendi, jonka mukaan rasterointikuorman vähenemisellä on suuri vaikututus raskaasti ylipiirtoa sisältävien kohtausten suorituskykyyn. Järjestelmän muista osista johtuen suorituskyvyn paraneminen pienemmillä tarkkuuksilla ei kuitenkaan ole lineaarinen. Esimerkiksi verteksivarjostinohjelman suorittaminen vie aina vakioajan puskurin tarkkuudesta riippumatta.

Huomattavissa on myös, että kehityslaitteena ollut LG Nexus 5 on riittävän tehokas näyttämään pelikentän sulavasti ilman cullingia kaikilla paitsi suurimmalla kentän koolla. Tämä johtaa helposti tilanteeseen, jossa suorituskykyä parantavien tekniikoiden käyttöönottoa ei tule ajatelleeksi kuin vasta myöhemmässä vaiheessa kehitystä. Toisella testilaitteella heikentynyt ruudunpäivitysnopeus on havaittavissa heti.

4.6 Löydösten tiivistelmä

Tässä luvussa testattiin teoriaosuudessa kuvattuja toimia ja arvioitiin niiden toimivuutta käytännössä. Demosovelluksen avulla esiteltiin, kuinka käyttöliittymää voidaan skaalata yhdistelemällä käyttöliittymäelementtien ankkurointia ja koon skaalaamista. Suorituskyvyn osalta keskityttiin demosovelluksen kehityksen aikana havaittuihin ongelmakohtiin: partikkelitehosteiden suureen rasterointikuormaan ja tarpeettomien piirtokäskyjen suureen määrään. Ongelmakohtien syitä pohdittiin ja esitettiin ratkaisutavat aiemmin kuvatun teorian pohjalta. Ratkaisuiksi valittiin matalatarkkuuksinen piirtopuskuri partikkelitehosteille ja piirtokäskyjen määrää puolestaan saatiin vähennettyä ohittamalla ruudun ulkopuolella olevien lohkojen piirtokäskyt.

Esitettyjen optimointitapojen vaikutus suorituskykyyn testattiin demosovelluksen avulla. Suorituskykytestauksen tulosten perusteella havaittiin sekä madalletun tarkkuuden piirtopuskurin että piirtokäskyjen ohittamisen parantavan demosovelluksen suorituskykyä huomattavasti.

5 YHTEENVETO

Android on tämän hetken ylivoimaisesti suosituin käyttöjärjestelmä mobiililaitteilla, ja siten alustan monimuotoisuus voi aiheuttaa haasteita esimerkiksi eri näyttökokojen tai laitteiston vaihtelevan suorituskyvyn muodossa. Tässä työssä esiteltiin tapoja, joilla Android-pelisovellus pystyy skaalaamaan näyttöobjektien kokoa niin, että käyttöliittymän käyttökokemus pysyy mahdollisimman vakiona vaikka käytössä olevan näyttölaitteen ominaisuudet muuttuvat. Tämän lisäksi esiteltiin, kuinka sovelluksen suorituskykyä on mahdollista parantaa tinkimällä graafisten tehosteiden laadussa tai karsimalla vähemmän tärkeää tekemistä. Muutamaa tapaa testattiin demosovelluksen avulla ja suoritettiin testejä, joiden perustella havaittiin todennettava parannus demosovelluksen suorituskyvyssä.

Tulevaisuutta ajatellen työssä esiteltyjä menetelmiä voitaisiin laajentaa esimerkiksi siten, että piirtopuskurin resoluutiota tai näyttöelementtien fyysistä kokoa säädetään automaattisesti käytössä olevan laitteen ominaisuuksien mukaan. Tämän lisäksi esimerkiksi näkymättömien lohkojen piirron ohittamiseen käytetty algoritmi on erittäin yksinkertainen ja edistyksellisempien algoritmien tutkiminen olisi suositeltavaa.

LÄHTEET

[1] ”Android Pushes Past 80% Market Share While Windows Phone Shipments Leap 156.0% Year Over Year in the Third Quarter, According to IDC”, *www.idc.com*. [Verkossa]. Saatavissa: http://www.idc.com/getdoc.jsp?containerId=prUS24442013. [Viitattu: 17-loka-2014].

[2] ”OpenGL ES | Android Developers”. [Verkossa]. Saatavissa: http://developer.android.com/guide/topics/graphics/opengl.html. [Viitattu: 17-loka-2014].

[3] ”Dashboards | Android Developers”. [Verkossa]. Saatavissa: https://developer.android.com/about/dashboards/index.html. [Viitattu: 17-loka-2014].

[4] ”Khronos Releases OpenGL ES 3.0 Specification to Bring Mobile 3D Graphics to the Next Level - Khronos Group Press Release”. [Verkossa]. Saatavissa: https://www.khronos.org/news/press/khronos-releases-opengl-es-3.0-specification. [Viitattu: 17-loka-2014].

[5] ”Supporting Multiple Screens | Android Developers”. [Verkossa]. Saatavissa: http://developer.android.com/guide/practices/screens_support.html. [Viitattu: 17-loka-2014].

[6] ”core/java/android/util/DisplayMetrics.java - platform/frameworks/base - Git at Google”. [Verkossa]. Saatavissa: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/util/DisplayMetrics.java. [Viitattu: 18-loka-2014].

[7] K. E. Hoff III, ”Faster 3D Game Graphics by Not Drawing What is Not Seen”, *Crossroads*, vsk. 3, nro 4, ss. 20–23, touko 1997.

[8] A. McNamara, K. Mania, G. Koulieris, ja L. Itti, ”Attention-aware Rendering, Mobile Graphics and Games”, teoksessa *ACM SIGGRAPH 2014 Courses*, New York, NY, USA, 2014, ss. 6:1–6:119.

[9] E. Preisz ja B. Garney, *Video Game Optimization*, 1 edition. Boston, MA ; Singapore: Cengage Learning PTR, 2010.

[10] X. Ma, Z. Deng, M. Dong, ja L. Zhong, ”Characterizing the Performance and Power Consumption of 3D Mobile Games”, *Computer*, vsk. 46, nro 4, ss. 76–82, huhti 2013.

[11] H. Nguyen, *GPU Gems 3*. Upper Saddle River, NJ: Addison-Wesley Professional, 2007.

[12] H. A. Engelbrecht ja G. Schiele, ”Koekepan: Minecraft As a Research Platform”, teoksessa *Proceedings of Annual Workshop on Network and Systems Support for Games*, Piscataway, NJ, USA, 2013, ss. 16:1–16:3.

[13] T. Fischer, A. Böttcher, A. Coday, ja H. Liebelt, ”Defining and Measuring Performance Characteristics of Current Video Games”, teoksessa *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, B. Müller-Clostermann, K. Echtle, ja E. P. Rathgeb, Toim. Springer Berlin Heidelberg, 2010, ss. 120–135.

[14] Y. Shuf ja I. M. Steiner, ”Characterizing a Complex J2EE Workload: A Comprehensive Analysis and Opportunities for Optimizations”, teoksessa *IEEE International Symposium on Performance Analysis of Systems Software, 2007. ISPASS 2007*, 2007, ss. 44–53.

[15] D. Gu, C. Verbrugge, ja E. M. Gagnon, ”Relative Factors in Performance Analysis of Java Virtual Machines”, teoksessa *Proceedings of the 2Nd International Conference on Virtual Execution Environments*, New York, NY, USA, 2006, ss. 111–121.
[16] ”LG Nexus 5 D821 Mobile Phone - LG Electronics UK”. [Verkossa]. Saatavissa: http://www.lg.com/uk/mobile-phones/lg-D821. [Viitattu: 17-loka-2014].
[17] Blaupunkt GmbH, *Manual Tablet PC Endeavour 800 NG*. .
[18] ”Snapdragon 800 Processor Specs and Details | Qualcomm”. [Verkossa]. Saatavissa: https://www.qualcomm.com/products/snapdragon/processors/800. [Viitattu: 17-loka-2014].
[19] ”Qualcomm Adreno 330 (450MHz) vs Qualcomm Adreno 305 - Graphics Cards Specs Comparison”. [Verkossa]. Saatavissa: http://versus.com/en/qualcomm-adreno-330-450mhz-vs-qualcomm-adreno-305. [Viitattu: 17-loka-2014].
[20] ”Amlogic － Leader in video, image and audio processing technology.” [Verkossa]. Saatavissa: http://www.amlogic.com/product02.htm. [Viitattu: 17-loka-2014].

LIITE 1 - TESTILAITTEIDEN TÄRKEIMMÄT TEKNISET TIEDOT

	Laite 1: Nexus 5 [16]	Laite 2: Endeavour 800 [17]
Valmistaja	LG Electronics	Blaupunkt
Suoritin	Qualcomm Krait 400 4*2.26 Ghz	ARM Cortex A9 2*1.5 Ghz
Keskusmuisti	2 GB	1 GB
Näyttö	1920x1080, 4.95”, 445 ppi	1024x768, 8”, 160 ppi
Näytönohjain	Qualcomm Adreno 330 [16][18] 3.6 Gpixel/s, 115.2 Gflops [19]	ARM Holdings Mali-400 [17][20] .5 Gpixel/s, 20 Gflops
Androidin versio	4.4.3	4.0.4

LIITE 2 – MITTAUSTULOKSET: PIILOTTAMINEN

			Ruutuaika (millisekuntia)	
Tapaus	**Laite**	**Kentän koko**	**Perusnäkymä**	**Kaukonäkymä**
Cull	Nexus 5	4*32	14,5	18,6
Cull	Nexus 5	8*32	16,0	21,8
Cull	Nexus 5	16*32	19,2	29,4
Nocull	Nexus 5	4*32	16,9	18,3
Nocull	Nexus 5	8*32	26,2	29,4
Nocull	Nexus 5	16*32	124,8	125,7
Cull	Endeavour 800	4*32	15,9	21,7
Cull	Endeavour 800	8*32	23,3	55,4
Cull	Endeavour 800	16*32	38,3	85,8
Nocull	Endeavour 800	4*32	23,1	26,8
Nocull	Endeavour 800	8*32	56,6	64,0
Nocull	Endeavour 800	16*32	180,5	182,2

Suhteellinen suorituskyky kun culling kytketään käyttöön

Laite 1: Nexus 5

Koko	**Perus**	**Kauko**
4*32	117 %	98 %
8*32	164 %	135 %
16*32	650 %	428 %

Laite 2: Endeavour 800

Koko	**Perus**	**Kauko**
4*32	145 %	124 %
8*32	243 %	116 %
16*32	471 %	212 %

LIITE 3 – MITTAUSTULOKSET: PIIRTOPUSKURI

		Keskimääräinen FPS testijakson ajan			
Laite	**Tarkkuus**	**Mittaus 1**	**Mittaus 2**	**Mittaus 3**	**ka.**
Nexus 5	256*256	48,831	49,426	49,267	49,175
Nexus 5	512*512	29,472	28,952	29,021	29,148
Nexus 5	1080*1704	16,228	16,094	16,174	16,165
Endeavour 800	256*256	28,795	32,707	32,079	31,194
Endeavour 800	512*512	15,667	15,733	15,614	15,671
Endeavour 800	768*976	7,838	7,918	7,848	7,868