2D-ANIMAATIOT OSANA ANDROID-SOVELLUSTA

Henri Honkonen

Opinnäytetyö
Toukokuu 2013

Mediatekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

Tekijä(t) HONKONEN, Henri	Julkaisun laji Opinnäytetyö	Päivämäärä 3.5.2013
	Sivumäärä 59	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty (X)
Työn nimi 2D-ANIMAATIOT OSANA ANDROID-SOVELLUSTA		
Koulutusohjelma Mediatekniikan koulutusohjelma		
Työn ohjaaja(t) NIEMI, Kari		
Toimeksiantaja(t) Silmu Software		

Tiivistelmä

Opinnäytetyön tavoitteena oli selvittää erilaisia mahdollisuuksia toteuttaa 2D-animaatioita Android-sovellukseen. Tutkittaviin animointitekniikoihin liittyi lisäksi niiden mahdollisuuksien ja rajoituksien selvittäminen sekä pohtiminen kyseisten tekniikoiden soveltuvuudesta eri tyyppisiin sovelluksiin. Opinnäytetyön toimeksiantajana toimi Jyväskyläläinen ohjelmistoyritys Silmu Software Oy.

Opinnäytetyön tietoperustassa käsitellään Androidia sovelluskehyksenä sekä animaation lähtökohtia Android-ympäristössä. Tähän sisältyy selvitys vektorigrafiikan käyttömahdollisuuksista Android-natiivisovelluksessa sekä 2D-grafiikan laitteistokiihdytyksen käytöstä sovelluksissa. Android-ympäristössä käytetettäviä animointitekniikoita esitellään työssä havainnollistavien esimerkkien myötä. Opinnäytetyössä toteutetaan lisäksi yksikertainen pelihahmoanimaatio käyttäen kahta eri animointimenetelmää, joita lisäksi vertaillaan ja tutkitaan tarkemmin. Työssä myös suoritetaan testausta, jonka tarkoituksena oli tutkia yleisimpien Android-puhelimien suorituskykyä animaatioiden ja grafiikan piirtämisen osalta.

Opinnäytetyön tuloksena syntyi kattava selvitys mahdollisuuksista toteuttaa animaatioita erityyppisiin Android-sovelluksiin, josta käy myös ilmi menetelmien mahdollisuudet ja rajoitukset.

Avainsanat (asiasanat) Android, animaatio	
Muut tiedot	

Author(s)
HONKONEN, Henri

Type of publication
Bachelor´s Thesis

Date
3.5.2013

Pages
59

Language
Finnish

Permission for web publication
(X)

Title
2D-ANIMATION AS A PART OF ANDROID APPLICATION

Degree Programme
Media Engineering

Tutor(s)
NIEMI, Kari

Assigned by
Silmu Software

Abstract
The assignment of the bachelor's thesis was to research various techniques to implement animations in Android applications. The research also included studying the pros and cons of these animation techniques and whether they would fit in a certain type of application. The assigner of the Bachelor's Thesis was Jyväskylä-based software company Silmu Software Oy.

The theory section of this bacherlor's thesis deals with Android as an application framework and also with the basis of animation in Android-platform. This section also includes research about use of vector graphics on a native Android application in addition to use of hardware acceleration in rendering 2D-graphics. The animation techniques discussed in this bachelor's thesis are demonstrated with examples. As a demonstration, a creation of a simple character animation done by using two different animation techniques is illustrated. These two methods are examined in more detail. As a part of bachelor's thesis some testing was also done to find out the 2D-graphics rendering capabilities of the most common Android phones.

As the result of the bachelor's thesis an inclusive report was written about the possibilities of implementing animations in Android applications, which also explains the pros and cons of these methods.

Keywords
Android, animation

Miscellaneous

SISÄLTÖ

KUVIOT

TAULUKOT

KÄSITTEET

Apache-lisenssi = vapaan ohjelmiston lisenssi, joka vaatii tekijänoikeuden huomautuksen säilyttämisen ja erottamislausuman, ja sallii lähdekoodin vapaan käytön.

Framework = sovelluskehys, joka muodostaa rungon sen päälle rakennettavalle tietokoneohjelmalle. Framework on myös ohjelmoinnin apuväline, jonka tarkoituksena on nopeuttaa uusien ohjelmistotuotteiden valmistusta ja joka tarjoaa valmiiksi rakennettuja ohjelman osia.

Illustrator = Adobe Systemsin kehittämä vektorigrafiikan piirto-ohjelma.

iOS = iPhone Operating System on Applen kehittämä käyttöjärjestelmä, joka on käytössä etenkin Applen iPhone- ja iPad-mobiililaitteissa.

Retina-näyttö = Applen nimitys kehittämälleen näytölle, jossa sanotaan olevan niin suuri pikselitiheys, että ihmissilmä ei havaitse pikselöitymistä normaalilla katseluetäisyydellä.

Symbian = Ensisijaisesti älypuhelimille suunniteltu mobiilikäyttöjärjestelmä, joka oli pitkään käytössä etenkin Nokian valmistamissa puhelimissa.

XML = Extensible Markup Language on rakenteellinen kuvauskieli, joka auttaa jäsentämään laajoja tietomääriä selkeämmäksi

1 TYÖN LÄHTÖKOHDAT

1.1 Toimeksiantaja

Opinnäytetyön idea lähti jyväskyläläisen Silmu Softwaren tarpeesta kartoittaa erilaisia mahdollisuuksia animointien toteuttamiselle Android-mobiilisovellukseen. Silmu Software on nykyisellään kaksitoista henkinen yritys, joka on erikoistunut mobiili- sekä websovelluskehitykseen. Heidän käyttämiinsä mobiilialustoihin lukeutuvat Android, iOS, Windows Phone, Symbian sekä MeeGo.

Silmu Software toteuttaa kaikki projektit alusta loppuun talon sisällä ja jokaista toteuttamassa on oma tiiminsä, johon kuuluu ohjelmistokehittäjät, käytettävyyssuunnittelija sekä graafikko. Yritys toimii säännöllisessä yhteistyössä asiakkaidensa kanssa taatakseen, että sovellukset vastaavat asiakkaan tarpeita.

Silmu Softwarella ei ollut varsinaisesti kokemusta animointien toteuttamisesta Android-sovelluksiin, joten opinnäytetyö tulisi hyödyttämään heidän tulevia projektejaan tältä osin.

1.2 Opinnäytetyön tehtävät ja tavoitteet

Alunperin opinnäytetyön tavoitteena oli toteuttaa Android-mobiilipelin hahmoanimoinnit, mutta projektin peruuntuessa opinnäytetyö kohdennettiin enemmän tutkivalle kannalle. Opinnäytetyön tärkeimmäksi tavoitteeksi tuli selvittää mahdollisimman kattavasti eri menetelmiä toteuttaa animaatioita Android-sovellukseen sekä vertailla näiden sopivuutta erityyppisiin sovellusprojekteihin.

Opinnäytetyö sisältäisi sekä Androidin omien animointiluokkien, että kolmannen osapuolen animointikirjastojen tutkimista, mikäli nämä vaikuttaisivat potentiaalisilta vaihtoehdoilta. Animaatioiden mahdollisuuksien ja rajoitusten tutkiminen tulisi myös kyseeseen. Opinnäytetyössä käsiteltäisiin lisäksi hyväksi todettuja tapoja animaatioiden suunnitteluun ja toteutukseen sekä animaatioiden soveltamista erilaisiin käyttötarkoituksiin peleissä ja hyötysovelluksissa.

1.3 Aiheen rajaus

Opinnäytetyössä päätettiin rajata tutkimuksen kohteeksi Android-laitteista älypuhelimet. Lisäksi Androidin versioista tutkimuksen kohteeksi päätettiin rajata versiot 2.3 – 2.3.2 (Gingerbread), 4.0.3 – 4.0.4 (Ice Cream Sandwich) sekä 4.1 – 4.2 (Jelly Bean), sillä näiden markkinaosuudet ovat tällä hetkellä suurimmat. Animaatioiden osalta rajaus tehtiin 2D-animaatioihin, ja opinnäytetyössä puhuttaessa animaatioista viitataan juuri 2D-elementtien animointiin.

1.4 Animaatio median muotona

Animaation tärkeimpänä tehtävänä on herättää katsojan kiinnostus esillä olevaan aiheeseen sekä kasvattaa asioiden huomioarvoa. Animaation kautta hankalasti ymmärrettävät asiat tuodaan esille katsojalle selkeämmin ymmärrettävässä muodossa. Kuvitellaan esimerkiksi kolmen toisiaan liikuttavan rattaan toimintaa. Jos mekanismin toimintaa yritetään kuvata tekstimuodossa, ei lukija varmasti ymmärrä ensilukemalla tämän toimintaperiaatetta. Kuvamuodossa katsoja saattaa visuaalisesti hahmottaa rattaat ja näiden pyörimissuunnat, mutta tämäkään ei vielä kerro mekanismin dynamiikasta. Kun rattaat animoidaan ja katsoja näkee, kuinka rattaiden hampaat liikkuvat toistensa suhteen, hän ymmärtää näiden toiminnan vaivattomasti.

Animaation avulla on helppo korostaa huomioarvoiset informaation osaset ja vastaavasti piilottaa tiedonvälittymistä häiritsevät tekijät. Animaation käyttäminen esimerkiksi opetusvideoihin on edellä mainittuun viitaten erinomainen sovellus. (Luukkonen 1996, 26). Käyttöliittymissä animaatioiden avulla voidaan antaa käyttäjälle lisäinformaatiota ohjelman toiminnoista, esimerkiksi ohjelman ladatessa voidaan tämä esittää graafisena palkkina. Animaatioiden avulla voidaan myös parantaa sovelluksen käyttökokemusta ja visuaalista ilmettä, lisäämällä siihen hieman liikettä. Tarinan kerronta on myös tärkeä animaation sovellus ja useat menestyneet elokuvat on toteutettu kokonaan tai osittain animaation keinoin.

Mikä sitten on animaation ja videon välinen ero? Molemmat ovat pohjimmiltaan tapoja esittää katsojalle kuvia peräkkäin siten, että katsojalle muodostuu näistä kokemus liikkuvasta kuvasta. Erot löytyvät siitä, miten animaatiota ja videota teh-

dään. Kameralla kuvattu video on sarja kuvia todellisesta elämästä, kun taas animaatio lähtee kuvittajan mielikuvituksesta siirtyen kuviksi paperille tai tietokoneelle, ja lopulta kokonaiseksi animaatioksi. Perinteiset kamera-animaatiot, kuten piirros- ja nukkeanimaatiot, ovat hyvin lähellä videota siinä mielessä, että näissä yksittäiset kuvat kuvataan videokameralla ja koostetaan sitten yhtenäiseksi videoksi.

2 ANDROID- SOVELLUSKEHYKSENÄ

2.1 Yleistä

Android on avoimen lähdekoodin käyttöjärjestelmä, joka pohjautuu Linuxin-käyttöjärjestelmäytimeen. Se on lähtökohtaisesti suunniteltu kosketusnäytöllisille laitteille, kuten älypuhelimille ja tablet-tietokoneille. Alun perin Androidin kehityksestä vastasi Android Inc., mutta Google osti tämän myöhemmin, ja nykyään kehittäjänä toimii Open Handset Alliance (lyh. OHA). OHA on 84 mobiilialan yrityksen yhtymä, joka kehittää avointa standardia mobiililaitteille (What would it take to build a better mobile phone? n.d.). Ensimmäinen Android-käyttöjärjestelmällä varustettu puhelin tuli myyntiin vuonna 2008. Sovelluskehitys Androidille tapahtuu Java-ohjelmointikielellä.

2.2 Markkinaosuus

Android saavutti johtavan mobiilikäyttöjärjestelmän aseman vuonna 2010, kun se ohitti Nokian-Symbian käyttöjärjestelmän käyttäjämäärässä. Vuonna 2012 Androidin markkinaosuus oli 70,1 prosenttia, kun taas toiseksi menestyneimmän mobiilikäyttöjärjestelmän, Applen iOS:n, markkinaosuus jäi vain 22,0 prosenttiin. Puhelinmääränä tämä tarkoitti 152,1 miljoonaa myytyä Android-älypuhelinta. (Androidilla ja Applella hurja markkinaosuus älypuhelimissa n.d.).

2.4 Google Play –ohjelmakauppa

Aiemmin nimellä Android Market kulkenut Google Play –ohjelmakauppa on digitaalinen sisältöpalvelu Androidille. Google Play muodostui Googlen uudistaessaan digitaalisen jakelun strategiaansa maaliskuun 6. päivä 2012, yhdistäen Android Marketin, Google Musicin ja eBookstoren (Velazco 2012.) Uudistuksen myötä palvelu kattaa nyt sovellusten ja pelien lisäksi elokuvien, televisiosarjojen, lehtien, kirjojen sekä musiikin selaamisen ja lataamisen. Toisaalta suuri osa sisällöistä ei toimi vielä kansainvälisesti, ja esimerkiksi Suomessa, Google Play mahdollistaa vain sovelluksien ja pelien lataamisen. Nähtäväksi jää, laajentaako Google uudistusta myös maihin, joissa Google Play vastaa edelleen vanhaa Android Marketia tarjontansa puolesta.

Kuka tahansa voi ryhtyä Android-sovellusten kehittäjäksi rekisteröidyttyään Google Play –kehittäjäkonsolitiliin. Uusilta sovelluskehittäjiltä veloitetaan tilin luonnin yhteydessä 25 dollarin kertaluontoinen rekisteröintimaksu. (Developer registration 2013) mukaan tämän tarkoituksena on torjua ei-toivottua sisältöä ja näin pitää Google Playn tuotteiden laatu korkeana.

Google Playhin lisättävät sovellukset voidaan määritellä maksuttomiksi tai maksullisiksi. Maksullisten sovellusten lisääminen vaatii kehittäjältä myös Google Checkout –tilin luomista. Google Checkout –tili täytyy lisäksi linkittää Google Play –kehittäjäkonsolin tiliin. Sovelluskehittäjät voivat itse määrätä lisäämänsä sovelluksen hinnan ja myös sen, mistä maista sovellusta voi ostaa. Sovelluskehittäjä saa myyntituotoista 70 %, ja 30 % menee tapahtumakuluihin. (Transaction Fees 2012.)

2.3 Androidin nopea kasvu

Yhtenä Androidin nopean kasvun avaimena voisi pitää sitä, että se tarjoaa hyvän ja monipuolisen mobiilialustan joka hintaluokan puhelimille. Avoimen lähdekoodin takia monet älypuhelinvalmistajat ovat havainneet Androidin kaupallisen potentiaalin. Tämä on johtanut siihen, että Android laitteita on markkinoilla valtava määrä, jopa lähes 4000 erilaista laitetta (The many faces of a little green robot 2012).

2.3.1 Androidin pirstaloituminen

Androidin ensimmäisen versio (1.0) julkaistiin vuonna 2008, jonka jälkeen uusia versioita on julkaistu hyvin nopealla aikavälillä, kuten taulukosta 1 käy ilmi. Tämä on ollut yhtä aikaa sekä hyvä että huono asia. Käyttöjärjestelmän päivitykset sinällään ovat hyvä asia, sillä ne korjaavat bugeja, parantavat tietoturvaa, lisäävät ja laajentavat sen toiminnallisuutta sekä parantavat yleisesti suorituskykyä. Toisaalta Androidin osalta asian suhteen on käynyt siten, että johtuen laajasta laitekannasta sekä laitevalmistajien tekemistä omista käyttöjärjestelmäversioista on Android niin sanotusti pirstaloitunut.

TAULUKKO 1. Androidin versiohistoria ja näiden esittelemät suurimmat uudistukset (Syed 2012)

Versio	Koodinimi	Julkaistu	Isoimmat uudistukset
1.0		23.9.2008	Ensimmäinen julkinen versio, Android Market, Web-selain
1.1		9.2.2009	Bugien korjauksia
1.5	*Cupcake*	30.4.2009	Widgetit, animoidut ruutusiirtymät, Bluetooth-tuki
1.6	*Donut*	15.9.2009	Parannuksia kosketusnäytön käyttöön, parannettu Android Market –ohjelmakauppa (nyk. Google Play), haku- ja kamerasovelluksien käytön nopeutus
2.0 - 2.1	*Enclair*	26.9.2009	Optimoitu laitteistokiihdytys, selaimen HTML5 –tuki, Bluetooth 2.1 –tuki, tuki usealle näytön koolle ja resoluutiolle
2.2	*Froyo*	20.5.2010	Adobe Flash 10.1 -tuki, USB-tethering, Wi-Fi-hotspot -tuki
2.3	*Gingerbread*	6.12.2010	Parannettu käyttöliittymä, parannettu virtuaalinäppäimistö, laajennettu kodekki-tuki video -ja ääniformaateille, parannuksia virranhallintaan
3.0	*Honeycomb*	22.2.2011	Ensimmäinen ensisijaisesti tablet-laitteille suunniteltu versio, moniydin-tuki, tehokkaampi laitteistokiihdytys, suuremmille näyttöresoluutioille optimoitu virtuaalinäppäimistö
4.0	*Ice Cream Sandwich*	19.10.2011	Uudistettu käyttöliittymä tukee sekä puhelimia että tabletteja, kasvontunnistus-tuki lukitusnäyttöön, parannettu tekstinsyöttö ja oikoluku
4.1	*Jelly Bean*	27.6.2012	Sulavampi käyttöliittymä, monikanavaäänet, parannettu ja nopeampi selain, parannettu äänentunnistukseen perustuva hakutoiminto Googleen
4.2	*Jelly Bean*	29.10.2012	Useampi käyttäjätili samassa laitteessa, widget-tuki myös lukitusnäyttöön, Photo Spere –ominaisuus mahdollistaa 360-asteen panoramakuvia, elekirjoitus

Osa laitevalmistajista muokkaa Android-käyttöjärjestelmää omiin tarkoituksiinsa sopiviksi (Cunningham 2012), mikä sinällään on hyvä, sillä näin käyttöjärjestelmä voidaan optimoida tietylle laitteistolle tai valmistajan oman brändin mukaiseksi. Toisaalta tämä synnyttää ongelman käyttöjärjestelmän päivittämisen suhteen: kun uusi päivitys julkaistaan, tulee laitevalmistajien toteuttaa päivitykset myös omiin ver-

sioihinsa ja jakaa nämä eteenpäin myös käyttäjille. Tämä prosessi voi olla erittäin hidasta, ja päivitysten toteuttaminen saatetaan pahimmassa tapauksessa jopa laiminlyödä uusien laitemallien saapuessa markkinoille.

Ihannetilanteessa uusimman Android-version ilmestyessä mahdollisimman moni laitteenomistajista siirtyisi käyttämään tätä, mutta todellisuus on kaukana tästä. Eri Android-laitevalmistajat jakavat päivityksiään eri aikatauluilla, mikä riippuu suuresti myös laitemallista. Päivityksen saaminen vanhalle Android-laitteelle voi kestää kauan, ja ajattelutavaksi tuntuu muodostuneen, että helpompaa onkin hankkia suoraan uudempi laite uudemman käyttöjärjestelmän varjolla. Päivityksen esteeksi voivat myös muodostua laitteen fyysiset ominaisuudet, etenkin vanhemman laitekannan yhteydessä.

Kuten taulukosta 2 käy ilmi, lähes puolet Android-käyttäjistä käyttää edelleen jo pari vuotta vanhaa Androidin versiota (2.3, Gingerbread). Vaikka 4.x versioiden osuus on lähtenyt hitaaseen kasvuun, tämä selittyy lähes yksinomaan uusien laitteiden yleistymisellä eikä niinkään päivitysten jakelulla.

TAULUKKO 2. Androidin versioiden käyttöosuudet 6.3.2013 (Platform Versions 2013, muokattu)

Versio	Koodinimi	API	Osuus
1.6	*Donut*	4	0,2%
2.0 - 2.1	*Enclair*	7	1,9%
2.2	*Froyo*	8	7,5%
2.3 – 2.3.2	*Gingerbread*	9	0,2%
2.3.3. – 2.3.7		10	43,9%
3.0 - 3.1	*Honeycomb*	12	0,3%
3.2		13	0,9%
4.0.3 – 4.0.4	*Ice Cream Sand-wich*	15	28,6%
4.1	*Jelly Bean*	16	14,9%
4.2		17	1,6%

Androidin pirstaloituminen asettaa haasteita myös sovelluskehittäjille. Itse sovelluksen kehittämisen lisäksi täytyy resursseja käyttää sen toiminnan takaamiseksi myös vähemmän tunnetuilla Android-laitteilla. Koska sama versio Androidista voi olla käyttöjärjestelmänä lukuisassa, fyysisiltä ominaisuuksiltaan hyvin eritasoisessa älypuhelimessa, tämä asettaa haasteen sovelluksen toiminnalle kaikissa näistä laitteista.

2.3.2 Androidin tulevaisuus

Androidin pirstaloitumista vastaan taistelemaan Google on julkaissut aloitteen nimeltä Android Update Alliance, jonka ideana on parantaa yhteistyötä laitevalmistajien kanssa ja taata Android-laitteille päivityksien jakelu 18 kuukaudeksi niiden myyntiin tulosta (Cunningham 2012). Nähtäväksi jää, kuinka tämä yhteistyö toteutuu tulevaisuudessa, ja esimerkiksi jo huhutun Androidin version 5.0 (Key Lime Pie) suhteen.

Yhtenä askeleena yhteistyön parantamiseksi on ehdotettu Androidin versioiden lähdekoodin jakelua laitevalmistajille jo sen kehitysprosessin aikaisemmassa vaiheessa sen sijaan, että nämä joutuvat odottelemaan tämän julkaisua (Cunningham 2012). Tämä ainakin nopeuttaisi tahtia, jolla laitevalmistajat voisivat toteuttaa ja jaella heidän omat päivityksensä käyttäjille. Myös laitekannan yksinkertaistaminen ja siten myös päivitysten yksinkertaistaminen olisi yksi askel parempaan päin.

3 ANIMAATION LÄHTOKOHDAT ANDROID-YMPÄRISTÖSSÄ

3.1 2D-grafiikan eri piirtomahdollisuudet Androidissa

Androidin framework tarjoaa eri tapoja piirtää ja esittää graafisia elementtejä sovelluksissa. Nämä menetelmät sisältävät sekä omien graafisten objektien luomisen sovelluksen ajon aikana että valmiiden kuvien lataamisen ulkoisesta resurssista.

Piirtotavat voidaan jakaa karkeasti kahtia sen mukaan, tarvitseeko grafiikan muuttua dynaamisesti sovelluksen ajon aikana. Mikäli sovelluksessa käytetään vain staattisia kuvia tai ennalta määritettyjä animaatioita, graafiset elementit piirretään käyttäen View-objektia (Canvas and Drawables n.d.). Jos taas sovelluksessa tarvitaan graafisten elementtien toistuvaa uudelleenpiirtämistä tai täyttä kontrollia animaatioihin, Canvas-elementin käyttö on suositeltavaa (Mt).

3.1.1 Canvas

Mikäli sovelluksessa tarvitaan omia tai kustomoituja piirto-operaatioita sekä mahdollisuutta kontrolloida animaatioita, piirtäminen on järkevää tehdä käyttäen Canvas-elementtiä. Canvas toimii ikään kuin raameina pinnalle johon grafiikka piirretään (Canvas and Drawables n.d.), ja se huolehtii piirto-operaatioista. Tämä pinta on Canvas-elementin alla oleva Bitmap, joka vaaditaan aina Canvasille piirrettäessä. Tyhjän Bitmapin voi luoda Canvasin luonnin yhteydessä, mutta tämän voi myös korvata jo olemassa olevalla kuvalla.

Piirtäminen käyttäen Canvasia edellyttää aina neljää asiaa (Canvas 2013):

- Bitmap – sisältää pikselit joille piirtäminen tapahtuu
- Canvas – huolehtii piirtokutsuista
- Drawing primitive – määrittää piirrettävän elementin muodon (neliö, ympyrä, viiva pisteestä x pisteeseen y...)
- Paint – määrittää piirrettävän elementin värin ja mahdollisen tyylin

Uuden Canvas-elementin luonti ja linkitys määritettyyn Bitmapiin:

```
Bitmap b = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
Canvas c = new Canvas(b);
```

Canvas-luokalla on omia piirtometodeja joita voi käyttää, kuten drawBitmap(), drawRect(), drawText() sekä muita vastaavia (Canvas 2013).

Tässä esimerkissä piirretään vihreä neliö olemassa olevan kuvan päälle (taustakuva.jpg) käyttäen drawRect() piirtometodia:

```
protected void onDraw(Canvas c){

// ladataan taustakuva projektin resursseista
Bitmap myBitmap = BitmapFactory.decodeResource(getResources(),
R.drawable.taustakuva);

// lisätään taustakuva Canvasille
c.drawBitmap(myBitmap, 0, 0, null);

// luodaan uusi tyyli neliölle, joka sisältää värin ja reunaviivan
Paint myPaint = new Paint();
myPaint.setColor(Color.GREEN);
myPaint.setStyle(Paint.Style.STROKE);
myPaint.setStrokeWidth(3);

//piirretään neliö Canvasille
c.drawRect(10, 10, 100, 100, myPaint);
}
```

3.1.2 Drawable ja Shape Drawable

Drawable-luokka tarjoaa menetelmiä erilaisten 2d-muotojen ja -kuvien piirtämiseen ja esittämiseen sovelluksessa (Canvas and Drawables n.d.). Yksinkertaisin tapa lisätä gafiikkaa sovellukseen käyttäen Drawable-luokkaa on kuvan lataaminen sovelluksen resursseista:

```
// Luodaan uusi ImageView-elementti i
ImageView i = new ImageView(this);
// Lisätään my_image drawable-resursseista
i.setImageResource(R.drawable.my_image);
// Määritetään imageviewin koko my_imagen kokoon
i.setAdjustViewBounds(true);
// Asetetaan ImageView näkymäksi
setContentView(i);
```

Sama esimerkki XML layoutissa:

```
<ImageView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/my_image" />
```

Valmiiden kuvien käyttämisen lisäksi Android tukee XML:n kautta luotavia Shape Drawable -kuvia. Shape Drawable mahdollistaa geometristen objektien määrittelyn, ja näiden ominaisuuksien kuten värin, reunaviivan (stroke) ja liukuvärin (gradient) kuvaamisen XML-muodossa (Canvas and Drawables n.d.). XML-muodossa olevien shape drawable –grafiikkaelementtien etuna on, että ne skaalautuvat aina oikean kokoisiksi (mt.). Shape Drawable –grafiikkaelementtejä käytetään etenkin tyylittelemään käyttöliittymäkomponentteja. Tässä esimerkissä luodaan Shape Drawable, jolle annetaan yksinkertaisia tyylimääreitä:

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <stroke
        android:width="5dp"
        android:color="#FF00FF00" />
    <gradient
        android:startColor="#DD666666"
        android:endColor="#FFFF0000"
        android:angle="45" />
    <corners
        android:radius="15dp" />
</shape>
```

3.2 Tuetut kuvaformaatit

3.2.1 Bittikarttagrafiikkamuodot

Bittikarttagrafiikkassa tai toiselta nimitykseltään rasterigrafiikkassa kuva muodostuu pikseleistä, joista jokaisella on tietty väriarvo. Androidissa tuettuja bittikarttagrafiikan formaatteja ovat JPEG, GIF, PNG, BMP sekä WebP (Android 4.0+) (Android Supported Media Formats n.d.)

Bittikarttakuvat voivat muodostua tiedostokooltaan hyvinkin suuriksi, joten tiedostoa tallennettaessa sen kokoa voidaan pyrkiä pienentämään pakkauksella eli kompressoinnilla. Pakkaus voi olla joko häviöllistä tai häviötöntä.

Häviöllisellä pakkauksella tarkoitetaan sitä, että alkuperäisestä kuvasta luodaan lähes identtinen kuva, jonka yksityiskohtien ja värien määrää pudotetaan erilaisilla pakkausalgoritmeilla. Näin kuvan pikseleiden väriarvot voidaan ilmaista vähemmillä biteillä. Tämän ansiosta kuvan tiedostokoko pienenee, eikä ihmissilmä pysty juurikaan huomaamaan näitä pieniä eroja. Kun häviöllistä pakkausta on sovellettu kuvalle, ei alkuperäistä kuvaa voida enää palauttaa. (Keränen, Lamberg & Penttinen, 2005, 92.)

Androidissa tuettu häviöllisen pakkauksen kuvaformaatti on **JPEG** (Joint Photographic Experts Group). Se on yksi käytetyimmistä bittigrafiikan tallennusformaateista. Formaatti on tarkoitettu erityisesti valokuvien pakkaamiseen, mutta sitä käytetään laajalti myös verkkosivuilla olevissa kuvissa. JPEG-pakkauksen määrää voidaan säätää tallennusvaiheessa: mitä enemmän kuvaa pakataan, sitä pienemmäksi sen tiedostokoko tulee, kuvan laatu myös heikkenee samassa suhteessa (ks. taulukko 3). JPEG-kuvat pystyvät tallentamaan 24 bittiä väri-informaatiota jokaista pikseliä kohden. JPEG –kuvien käyttö Android-sovelluksissa on Googlen suosituksien mukaan hyväksyttävää, mutta ei suositeltavin kuvaformaatti (Canvas and Drawables n.d.).

TAULUKKO 3. JPEG-pakkauksen vaikutus kuvakokoon ja kuvanlaatuun

Pakkaus	Kuvakoko	Lopputulos
▪ JPEG-High -esiasetus ▪ 60%-kuvanlaatu ▪ Kohtalainen kuvanlaatu, suhteellisen pieni tilantarve	5,45 kilotavua	
▪ JPEG-Medium -esiasetus ▪ 30%-kuvanlaatu ▪ Välttävä kuvanlaatu, pieni tilantarve	3,51 kilotavua	
▪ JPEG-Low -esiasetus ▪ 10%-kuvanlaatu ▪ Huono kuvanlaatu, erittäin pieni tilantarve	2,92 kilotavua	

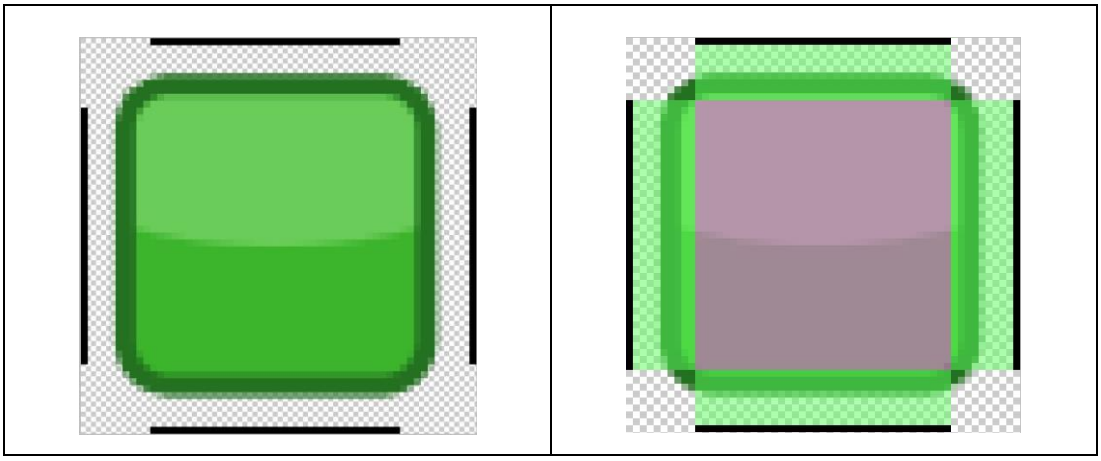
Häviöttömässä pakkauksessa kuvan sisältö säilyy muuttumattomana alkuperäiseen kuvaan nähden. Tämä perustuu siihen, että kuvasta etsitään väriarvoiltaan yhteneväisiä alueita, joiden ilmoittamiseen voidaan käyttää vähemmän bittejä. Koska

häviöttömässä pakkauksessa kuvasta ei hävitetä informaatiota, myöskään kuvan laatu ei heikkene lainkaan. (Keränen ym. 2005, 92.)

Androidissa tuettuja häviöttömään pakkaukseen perustuvia kuvaformaatteja ovat seuraavat:

PNG (Portable Network Graphics) mahdollistaa hyvälaatuisten kuvien tekemisen web-ympäristöön, jotka myös latautuvat nopeasti. PNG–kuvat voivat olla joko 8-, 24- tai 32-bittisiä, joista kaksi viimeksi mainittua tukevat läpinäkyvyyden mahdollistavaa alpha-kanavaa. PNG–kuvat ovat yleensä suurempia tiedostokooltaan kuin esimerkiksi JPEG–kuvat, sillä ne sisältävät enemmän informaatiota. PNG:n ominaisuuksiin luketuu läpinäkyvyyden lisäksi gamma-korjaus, jonka avulla kuvien kirkkautta voidaan kontrolloida laitteistosta riippumatta. (George 2011). PNG on Googlen suositus Androidissa käytettäväksi bittikarttagrafiikan formaatiksi (Canvas and Drawables. n.d.).

Androidissa PNG-kuvatiedostojen yhteydessä voidaan soveltaa myös .9.png-päätettä, jolloin järjestelmä tulkitsee kyseiset kuvatiedostot **Nine-patch**-grafiikkaobjekteina. Nine-patch -kuvan nimitys tulee siitä, että kuva jaetaan yhdeksään osaan: neljään kulmaan, neljään sivuun ja keskiosaan (Ninepatch n.d.) Kuvan osia kulmia lukuun ottamatta skaalataan täyttämään koko objekti, jonka taustaksi se on asetettu. Nine-patch -kuvia käytetään esimerkiksi Androidin käyttöliittymän painikkeissa (ks. kuvio 2.) Kuvan skaalattava osa määritetään piirtämällä sen viereen yhden pikselin kokoinen musta viiva, kuten kuviossa 1. Draw 9-patch on kätevä työkalu Nine-patch –kuvien tekemiseen, ja se tulee Android -kehitystyökalujen mukana /tools-kansiossa.



KUVIO 1. Nine-patch –kuvan skaalattavan alueen määrittäminen Draw 9-patch –työkalulla

KUVIO 2. Nine-patch -kuva eri mittaisten painikkeiden taustana

GIF-kuvan (Graphics Interchange Format) värien määrä on rajoitettu 8-bittiseen värisyvyyteen, mikä mahdollistaa korkeintaan 256 eri värisävyä. GIF tukee PNG:n tavoin läpinäkyvyyttä, ja häviöttömän pakkauksen ansiosta kuvatiedostot ovat tiedostokooltaan melko pienikokoisia laadun kärsimättä. GIF–kuvien kokoa voidaan pienentää myös vähentämällä värien määrää. GIF tukee myös kuvan lomitusta (eng. interlacing), mikä mahdollistaa kuvan lataamisen vaiheittain siten, että kuva latautuu ensin

heikompilaatuisena ja tarkentuu vähitellen. GIF-kuvaformaatti soveltuu parhaiten vähäväristen grafiikkaelementtien, kuten logojen ja ikonien, toteuttamiseen web-sivuille. GIF mahdollistaa myös yksinkertaisten animaatioiden tekoa, ja tätä ominaisuutta sovelletaan yleisesti web-sivujen mainosbannereissa. (George 2011.) Vaikka Androidissa on tuki GIF-kuville, niiden käyttö sovelluksissa ei Googlen mukaan ole suositeltavaa (Canvas and Drawables n.d.).

BMP (Bitmap) on Androidissa tuettu, tosin ei suositeltu, pakkaamaton kuvaformaatti, jossa jokaisen pikselin väritiedot tallennetaan erikseen. BMP-kuva voidaan tallentaa joko 24-, 8-, 4-, tai 1-bittisenä. Alun perin BMP kehitettiin Windows-käyttöjärjestelmän pääasialliseksi kuvaformaatiksi, mutta suuri osa ohjelmista ja muista käyttöjärjestelmistä tukee sitä (Keränen ym. 2005, 93.) BMP-kuvaformaatti on nykyään vähän käytetty sen suuren tiedostokoon johdosta.

WebP on Googlen vuonna 2010 julkaisema bittikarttagrafiikan formaatti, jossa voidaan käyttää sekä häviöllistä että häviötöntä pakkausta. Se tukee PNG- ja GIF-formaattien tavoin alpha-kanavaa, mikä mahdollistaa läpinäkyvien kuvien tekemisen. Google lupaa häviöttömästi pakattujen WebP-kuvien olevan 26 % pienempiä tiedostokooltaan kuin PNG-kuvat ja häviöllisesti pakattujen kuvien 25-34 % pienempiä kuin JPEG-kuvien (Webp – A new image format for the Web n.d.). Tällä hetkellä Webp-formaatti on Androidissa tuettuna versiosta 4.0 (Ice Cream Sandwich) eteenpäin.

3.2.2 Vektorigrafiikkamuodot

Vektorigrafiikassa kuva esitetään koordinaattien sekä matemaattisten funktioiden avulla pikseleiden sijaan. Tästä johtuen kuvanlaatu ei heikkene, ja niin sanotusti ”pikselöidy”, kuvaa skaalatessa, kuten bittikarttakuvalle käy (ks. kuvio 3). Vektorigrafiikan etu bittikarttagrafiikkaan nähden on juuri sen resoluutioriippumattomuus. Siinä missä vektorikuva skaalautuu heikentämättä laatuaan joka näyttöresoluutiolle, täytyy bittikarttakuvasta tehdä useita eri resoluutioisia kuvia tukemaan eri näyttöresoluutioita. Yleisimpiä vektorigrafiikan muotoja ovat PDF (Portable Document Format), Postscript (PS) sekä Scalable Vector Graphics (SVG). Näistä kaksi ensimmäistä ovat vakiintunut tulostettavan materiaalin muodoiksi ja SVG on laajalti käytetty webissä.

KUVIO 3. Vektorikuvan ja bittikarttakuvan ero suurennettaessa kuvaa

SVG-kuvien määrittelyt tehdään XML-tekstitiedostossa, joten niitä voidaan luoda ja muokata periaatteessa millä tahansa tekstieditorilla. SVG-kuvien etuna on skaalautuvuuden lisäksi niiden pieni tiedostokoko, mikä mahdollistaa nopean kuvien vaihtamisen sovelluksen ajon aikana. (Doss, Lo & Pai 2012.)

Androidissa ei ole oletusarvoisesti suoraa tukea SVG-elementin käyttämiselle natiiveissa sovelluksissa. Jos sovelluksessa on kuitenkin tarvetta vektorigrafiikan käytölle, on tähän tarjolla kolmannen osapuolen kirjastoja, joista tässä esitellään **svg-android**.

Svg-android mahdollistaa SVG-kuvien parseroinnin ja renderöinnin Androidissa. Kirjasto on lisensoitu Apache 2.0 lisenssillä, mikä mahdollistaa sen käytön myös kaupallisissa sovelluksissa. Kirjaston käyttöönotto vaatii ainoastaan yhden jar-tiedoston lisäämistä Android-sovelluksen projektikansioon. SVG-kuvan voi piirtää tarpeesta riippuen joko suoraan Canvasiin tai ImageViewin kaltaiseen elementtiin. (Vector Graphics Support for Android! n.d.)

Svg-android noudattaa SVG Basic 1.1 määrityksiä, ja suurinta osaa vektorigrafiikan ominaisuuksista on mahdollista käyttää. Käytännössä tämä tarkoittaa sitä, että SVG-kuvan voi tallentaa esimerkiksi Adoben Illustratorista käyttäen SVG Basic 1.1 asetusta ja kuva näkyy oikein Androidissa (ks. kuvio 4). Jotkin SVG:n ominaisuudet, kuten skriptaus, animointi, rasterikuvat ja tekstiominaisuudet, eivät ole tuettuja SVG Basic 1.1 –versiossa. Tästä syystä kaikki Illustratorin efektit, kuten drop shadow (varjo),

eivät välttämättä toistu oikein niiden, niiden tarvitessa kuvan rasteriominaisuutta. (Mt.)

Yksinkertainen esimerkki SVG-kuvan parseroimiseen ja piirtämiseen (Mt.):

```
public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            // Luodaan uusi ImageView-elementti
            ImageView imageView = new ImageView(this);
            // Asetetaan ImageViewin tausta valkoiseksi
            imageView.setBackgroundColor(Color.WHITE);
            // Parseroidaan SVG-kuva (android.svg)
            SVG svg = SVGParser.getSVGFromResource(getResources(),
            R.raw.android);
            // Tehdään parseroidusta SVG-kuvasta drawable-elementti
            imageView.setImageDrawable(svg.createPictureDrawable());
            // Asetetaan ImageView näkymäksi
            setContentView(imageView);
            }
}
```

KUVIO 4. SVG-kuva renderöitynä ImageView-elementtiin käyttäen svg-android -kirjastoa

3.3 Usean näyttökoon tukeminen

Hyvän Android-sovelluksen lähtökohtana on, että se pyrkii tukemaan mahdollisimman montaa eri ruutukokoa. Android-käyttöjärjestelmä pyrkii skaalaamaan sovelluksen eri näytöille sopiviksi, mutta parhaan käyttökokemuksen takaamiseksi tämä täytyy huomioida jo suunnittelussa. Näin käyttäjä saa vaikutelman siitä, että sovellus näyttää siltä kuin se oli luultavasti suunniteltukin, eikä sitä ole vain venytetty täyttämään ruutua.

Android-sovelluksia suunniteltaessa ei yleensä käytetä kiinteitä pikselimääriä, vaan ennemmin huomioidaan näytön koko ja pikselitiheys (engl. density) (Supporting Multiple Screens. n.d.) Näytön koosta puhuttaessa tarkoitetaan sen fyysistä kokoa lävistäjänä mitattuna. Vastaavasti näytön pikselitiheydestä puhuttaessa tarkoitetaan pikseleiden määrää tietyllä näytön alalla, esimerkiksi pikseliä tuumalle (engl. pixels per inch) tai pistettä tuumalle (eng. dots per inch).

Android-laitteiden moninaisuuden vuoksi, näyttöjen koot eivät ole vakioituneet mihinkään tiettyihin lukuihin, ja sovellukset suunnitellaan tästä johtuen jollekkin kokoluokalle. Näitä näytön kokoluokkia on neljä: small, normal, large ja extra large. Näytön pikselitiheyden mukaan laitteet jaetaan puolestaan seuraavasti: ldpi (120 dpi, low), mdpi (160 dpi, medium), hdpi (240 dpi, high), xhdpi (320 dpi, extra high). (Mt.)

Aiemmat määritelmät on suunniteltu ensimmäisen Android-laitteen T-mobile G1:n ympärille. Tämän pohjalta niin sanotuksi lähtötasoksi (eng. baseline) muodostui normaalikoon näyttö, jossa on medium-tason pikselitiheys. Toisin sanottuna, vertailukohtana on pidetty 320 x 480 pikselin HVGA (half-size VGA) –näyttöä. (Mt.)

Kuten taulukosta 4 käy ilmi, joka toisessa Android-laitteessa nykyään on normalkokoluokan näyttö, jossa on myös korkea pikselitiheys. Google Developers verkkosivuston (Mt.) mukaan näissä laitteissa on 480 x 800 pikselin WVGA (wide VGA) –näyttö. Tämän perusteella voisi sanoa, että normaalikokoisen Android-puhelimen määritelmä on muuttumassa yhä suuremmaksi. Erittäin suuren pikselitiheyden omaavat näytöt ovat myös taulukon 4 mukaan selvästi yleistymässä ja Applen iPho-

ne-mobiililaitteissa käytetty Retina-näytöntarkkuus (326 ppi) on saatavilla yhä useammassa Android-laiteessa.

TAULUKKO 4. Android-laitteiden näyttöjen koot ja pikselitiheydet (Platform versions n.d., muokattu)

	Low density (120 dpi), ldpi	Medium density (160 dpi), mdpi	High density (240 dpi), hdpi	Extra high density (320 dpi), xhdpi
small screen	1.7 %		1.0 %	
normal screen	0.4 %	11%	50.1 %	25.1 %
large screen	0.1 %	2.4 %		3.6 %
extra large screen		4.6 %		

Erikokoisten näyttöjen tukeminen Android-sovelluksessa edellyttää tuettujen näyttöjen ilmoittamista sovelluksen Android-manifestissa (ks. kuvio 5). Kyseinen tiedosto ilmoittaa Android-käyttöjärjestelmälle sovelluksen toiminnasta, kuten käyttöoikeuksista ja vähimmäisvaatimuksesta Androidin version suhteen.

```
<supports-screens android:resizeable=["true"| "false"]
                  android:smallScreens=["true" | "false"]
                  android:normalScreens=["true" | "false"]
                  android:largeScreens=["true" | "false"]
                  android:xlargeScreens=["true" | "false"]
                  android:anyDensity=["true" | "false"]
                  android:requiresSmallestWidthDp="integer"
                  android:compatibleWidthLimitDp="integer"
                  android:largestWidthLimitDp="integer"/>
```

KUVIO 5. Tuettujen näyttökokojen ilmoittaminen Android-manifestissa

Oletusarvoisesti Android-järjestelmä skaalaa sovelluksen laitteen näyttöruudun mukaan, ja useimmissa tapauksissa tämä toimii hyvin. Joissain tapauksissa lopputulos ei kuitenkaan ole halutunlainen ja sovelluksen ulkoasuun halutaan tehdä muutoksia,

jotta se näyttää hyvältä skaalauksen jälkeenkin. Isoimmille näytöille suunnitellun sovelluksen käyttöliittymäelementit eivät välttämättä mahdu pienenpään näyttöön, ja vastaavasti pienelle näyttökoolle suunnitellun sovelluksen elementtien yleensä halutaan täyttävän paremmin tyhjää tilaa isommalla näyttökoolla. Näissä tapauksissa sovelluskehittäjän tulee tarjota vaihtoehtoisia näkymiä (engl. layout), kuten kuviossa 6 (mt.)

```
res/layout/my_layout.xml                // normal-näytön näkymä (oletus)
res/layout-small/my_layout.xml          // small-näytön näkymä
res/layout-large/my_layout.xml          // large-näytön näkymä
res/layout-xlarge/my_layout.xml         // xlarge-näytön näkymä

// res/layout-xlarge-land/my_layout.xml // xlarge-näytön vaakatason-näkymä
```

KUVIO 6. Listaus sovelluksen tarjoamista eri näkymistä

Myös kaikesta sovelluksen käyttämästä grafiikasta tulee tarjota vaihtoehtoiset versiot vastaamaan eri näyttöjen pikselitiheyksiä. Vaikka Android osaa oletuksena skaalata siinä tuettuja bittikarttagrafiikoita, tämä voi johtaa kuvien ”pikselöitymiseen”. Tämä voidaan välttää tekemällä kuvatiedostoista eri resoluutioiset versiot eri piskelitiheyksille. Nämä kuvatiedostot tulee sijoittaa oikeaa pikselitiheyttä vastaavaan kansioon sovelluksen kansiorakenteessa. Esimerkiksi kaikki korkean pikselitiheyden omaaville näytöille tarkoitetut kuvatiedostot tulee sijoittaa drawable-hdpi/ -kansioon. Kun kaikki kuvatiedostot on sijoitettu edellisen esimerkin tavoin niitä vastaaviin kansioihin, osaa Android valita näytön koosta ja pikselitiheydestä riippuen näistä parhaiten soveltuvan version kuvasta. Jos sopivaa kuvaa ei löydy, järjestelmä käyttää normaalitasoista kuvaa ja skaalaa sitä tarpeen mukaan. (Mt.)

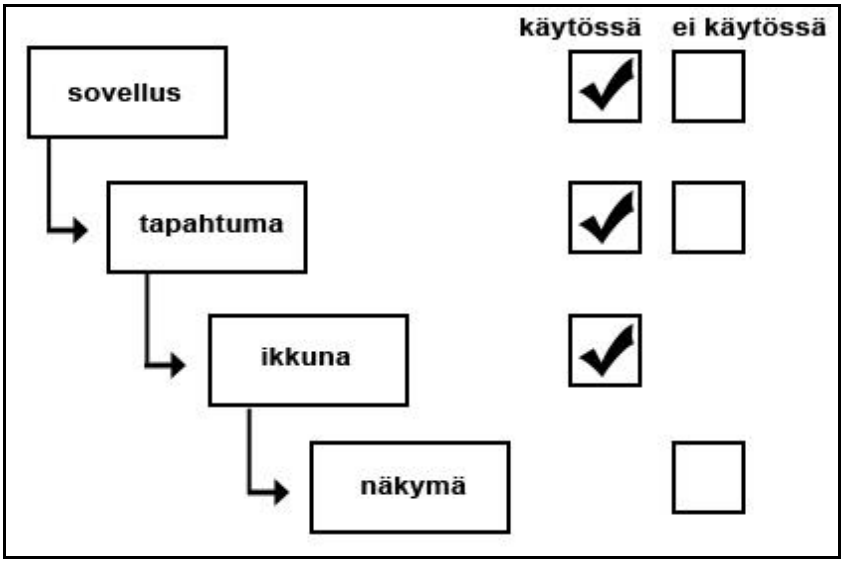
3.4 2D-grafiikan laitteistokiihdytys

Androidin versiosta 3.0 (Honeycomb) alkaen sovellukset ovat pystyneet käyttämään tehokkaasti hyödykseen laitteistokiihdytystä 2D-grafiikan piirtämiseen ruudulle. Laitteistokiihdytyksen ansiosta kaikki piirto-operaatiot Canvas-elementille suoritetaan prosessorin avulla. Myös Androidissa suoritettavat animaatiot hyötyvät tästä, tehden niistä nopeampia ja sulavampia. (Hardware Acceleration n.d.).

Laitteistokiihtydyksen voi Androidissa asettaa koko sovelluksen laajuiseksi, tai sitä voi kohdentaa esimerkiksi vain tietylle tapahtumalle. Jos sovelluksessa käytetään vain yleisimpiä piirto-operaatioita, laitteistokiihdytyksen asettaminen koko sovelluksen laajuiseksi ei aiheuta mitään ei-toivottuja vaikutuksia. Laitteistokiihdytyksen asettaminen sovelluksen laajuiseksi tapahtuu lisäämällä manifest.xml –tiedostoon seuraava rivi (mt.):

```
<application android:hardwareAccelerated="true" ...>
```

Kaikki 2D-piirto-operaatiot eivät kuitenkaan tue laitteistokiihdytystä, joten tämän menetelmän käyttö ei ole kaikissa tapauksissa suositeltavaa. Jos sovelluksessa käytetään esimerkiksi omia tai muokattuja piirto-operaatioita, laitteistokiihdytystä tulisi käyttää tai olla käyttämättä tarkemmalla tasolla (mt.). Nämä tasot ovat kuvattuina kuviossa 7. Kuten kuviosta käy ilmi, laitteistokiihdytyksen voi asettaa sovellustason lisäksi tietyille tapahtumille ja ikkunoille. Sen käyttöä voi vastaavasti rajoittaa sovellustason lisäksi tapahtumatasolla, ja mikäli laitteistokiihdytys on asetettu sovellustasolle, myös näkymätasolla.



KUVIO 7. Laitteistokiihdytyksen määrittäminen sovelluksen eri tasoilla

Laitteistokiihdytyksen määrittäminen eri sovelluksen tasoille mahdollistaa sen, että esimerkiksi kaikki, lukuunottamatta yhtä ei-tuettua, sovelluksen piirto-operaatioista

suoritetaan käyttäen laitteistokiihdytystä, ja tähän ei-tuettuun piirto-operaatioon puolestaan käytetään softapuolen renderöintiä.

Tapahtumatasolla laitteistokiihdytys otetaan käyttöön tai pois käytöstä seuraavasti (mt.):

```
<!--   Asetetaan laitteistokiihdytys koko sovelluksen laa-
juiseksi lukuun ottamatta  yhtä tapahtumaa -->

<application android:hardwareAccelerated="true">
    <activity ... />
    <activity android:hardwareAccelerated="false" />
</application>
```

Vastaavasti tietyn ikkunan voi määrittää käyttämään laitteistokiihdytystä seuraavasti (mt.):

```
getWindow().setFlags(
    WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED,
    WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED);
```

Yksittäiseltä näkymältä voi poistaa laitteistokiihdytyksen seuraavasti (mt.):

```
imageView.setLayerType(imageView.LAYER_TYPE_SOFTWARE,
null);
```

Laitteistokiihdytyksen tilan tarkistaminen voi olla hyödyllistä etenkin jos sovelluksessa käytetään omia piirto-operaatioita tai joitakin ei-tuetuista operaatioista. Tämän tarkistuksen voi tehdä sekä View- että Canvas-elementille, joista jälkimmäistä suositellaan käytettäväksi aina jos mahdollista, sillä joissakin tapauksissa View-elementti voidaan renderöidä softapuolella vaikka se olisi liitetty laitteistokiihdytettyyn ikkunaan. (Mt.)

- View.isHardwareAccelerated(), palauttaa arvon *true* jos kyseinen näkymä on liitetty laitteistokiihdytettyyn ikkunaan.
- Canvas.isHardwareAccelerated(), palauttaa arvon *true* jos Canvas-elementti käyttää laitteistokiihdytystä.

Laitteistokiihdytystä voidaan soveltaa myös sovelluksessa käytettäviin animaatioihin, mikäli ne eivät toistu jo ennestään sulavasti. Animaatiot voivat nopeasti käydä raskaiksi, kun piirto-operaatioita vaaditaan useita ja tämä aiheuttaa animaation hidastumista ja ”tökkimistä”.

Tässä tapauksessa animoitavan näkymän voi renderöidä käyttäen laitteistokiihdytettyä hardware-layeria. Tämä menetelmä poistaa animoitavalta näkymältä tarpeen piirtää itsensä uudestaan ja uudestaan animaation ajan, mikä on hyvin resursseja vievä toimenpide. Koska hardware-layer varaa paljon muistia, suositellaan sen käyttöä ainoastaan animaation keston ajaksi. (Mt.)

Tässä esimerkissä (mt.) näkymä animoidaan käyttäen hardware-layeriä, ja animaation päätyttyä näkymän renderöinti palautetaan normaaliksi:

```
View.setLayerType(View.LAYER_TYPE_HARDWARE, null);
ObjectAnimator animator = ObjectAnimator.ofFloat(view, "rotationY",
180);
animator.addListener(new AnimatorListenerAdapter() {

            @Override
            public void onAnimationEnd(Animator animation) {
              view.setLayerType(View.LAYER_TYPE_NONE, null);
            }
});
animator.start();
```

4 ANIMAATIOIDEN SOVELLUTUKSET ANDROID-SOVELLUKSISSA

4.1 Yleistä

Opinnäytetyössä on aiemmin puhuttu animaatioiden merkityksestä huomionherättäjinä sekä niiden tarinankerronnallisista mahdollisuuksista. Näitä ominaisuuksia voidaan soveltaa Android-sovelluksissa hyvin moninaisesti pelihahmojen liikkeistä käyttöliittymän animointeihin. Sovelluksesta riippumatta kaikkia animaatioita yhdistää käyttäjän ohjaaminen sekä lisäinformaation antaminen. Esimerkiksi tasohyppelypelissä pelaaja voidaan ohjata oikealle reitille esittämällä hänelle vilkkuva

suuntanuoli. Vastaavasti käyttöliittymässä käyttäjää voidaan kehottaa painamaan jotakin tiettyä elementtiä nostamalla sen huomioarvoa ”vilkuttamalla” sitä. Animaatioilla on Android-sovelluksissa myös tärkeä merkitys viihteellisen lisäarvon tuojana. Tämä perustuu siihen, että animaatio koetaan usein staattisia esitysvaihtoehtoja miellyttävämmäksi esitystavaksi (Kuuranta 2011). Hyvin toteutetut animaatiot jättävät käyttäjälle positiivisen kokomuksen sovelluksen käytöstä, mikä puolestaan lisää todennäköisyyttä palata sen pariin uudestaan. On myös havaittu, että hauskoiksi koetut asiat ovat yleisesti ottaen helppokäyttöisempiä (Kuuranta 2011).

4.2 Pelit

Peleissä animaatiot ovat yksi merkittävistä tekijöistä, joiden avulla pelaaja saadaan uppoamaan pelimaailmaan. Animaation keinoin pelimaailma sekä pelihahmot herätetään eloon, mikä puolestaa lisää pelaajan kokemusta siitä, että hän on mukana pelin tapahtumissa. Animaation keinoin pelaajalle annetaan myös lisäinformaatiota pelin tapahtumista ja lisätään pelin tunnelmaa.

4.2.1 Käyttöliittymä

Toimivan ja helppokäyttöisen pelin luominen vaatii myös hyvin toimivan käyttöliittymän toteutusta. Pelimaailmaan uppoutumiseen käyttöliittymällä on suuri vaikutus, sillä kaikki pelin keinotekoisuudesta muistuttavat tekijät koetaan haittaaviksi (Kuuranta 2011). Parhaassa tapauksessa näiden kahden näkyvä raja saadaan häivytettyä, esimerkiksi animoimalla siirtymä näiden välillä. Joitakin käyttöliittymän elementtejä voidaan myös tehdä osaksi pelimaailmaa.

4.2.2 Ensivaikutelma

Android-peleissä ensivaikutelmalla on suuri merkitys: pelit ovat suhteellisen edullisia tai jopa ilmaisia, joten kynnys pelin lataamiseen on matala. Jos peli ei ensivaikutelmansa puolesta ole kiinnostava, on helppo etsiä tilalle toinen peli. Pelin alkuvalikko on useasti se näkymä mistä kiinnostus peliin alkaa ja jonka käyttäjä näkee aina ensimmäisenä, joten siihen kannattaa myös panostaa. Yksi tapa herättää kiinnostus peliin jo alkuvalikossa on lisätä siihen animaatioita, esimerkiksi esitellä pelissä esiintyviä hahmoja. Näin pelaajaa johdatellaan pelimaailmaan jo ennen kuin peli on

varsinaisesti alkanut. Toinen esimerkki on niin sanotun alkuintro-animaation tekeminen, joka myös johdattelee pelaajaa pelimaailmaan ja kertoo jopa alustavasti pelin tarinasta.

4.2.3 Tiedon välittäminen pelaajalle

Peleissä on tavallista ilmoittaa pelaajalla asioita puolihuomaamattomasti, jolloin käyttäjän huomiota ei viedä pois itse pelitapahtumasta (Kuuranta 2011). Otetaan esimerkiksi autopeli, jossa pelaajan kaikki keskittyminen on ajoneuvon hallinnassa ja tiellä olevien esteiden väistelyssä. Oletetaan että autosta on loppumassa polttoaine, joka vaatii pelaajan toimia tämän estämiseksi. Tämä tieto voitaisiin esittää lyhenevänä palkkina ruudun laidalla, mutta tämän huonona puolena on se, että pelaaja joutuu jatkuvasti tarkkailemaan mittaria sivusilmällä. Sama tieto voitaisiin esittää pelaajalle ruudulle ilmestyvällä vilkkuvalla polttoaineen loppumisesta ilmoittavalla symbolilla. Liikkeellä on tapana vetää katsojan huomio helpommin puoleensa (Luukkonen 1996, 26), joten pelaaja huomioi polttoaineen loppumisen helpommin animaationa, jopa alitajuisesti.

4.2.4 Pelihahmot

Pelihahmot ovat pelien näyttelijöitä ja näiden liikehdinnällä on suora yhteys siihen, kuinka uskottavana ja samaistuttavana pelihahmo koetaan. Animaation avulla pelihahmoille annetaan luonne, ja animoinnin myötä hahmot osaavat myös ilmaista tilastaan ja toiminnastaan. Kuviossa 8 on esimerkki tästä pelihahmon toimintaa ilmaisevasta animaatiosta: Android-pelin (Replica Island) päähahmon jaloista suihkuavat rakettimoottorin liekit, mikä ilmaisee sen aikeista nousta ylöspäin.

Pelihahmon luonne muodostuu ulkonäön lisäksi sen liikehdinnän ja eleiden kautta. Tämä myös vaikuttaa siihen, kuinka pelaaja arvelee pelihahmon soveltuvan eri käyttöön (Kuuranta 2011). Aggressiivesti liikehtivä hahmo vihjaa pelaajalle sen oleva hyvä hyökkäämään ja pelaaja osaa käyttää tätä sen mukaan. Sama periaate pätee myös muihinkin pelin hahmoihin kuin pelaajan ohjastamaan, ja esimerkiksi pelin viholliset voivat kertoa olevansa hyvin vaarallisia liikkumalla terävästi ja esittelemällä hampaitaan.

Animaatiolla voidaan tehokkaasti ilmaista pelihahmoon vaikuttavia voimia, kuten painovoimaa: kun pelihahmo on putoavassa tilassa se voi esimerkiksi näyttää hätääntyneeltä ja heiluttaa käsiään. Pelihahmoihin vaikuttavien voimien ilmentämistä animaation avulla käytetään myös useassa pelissä, jossa pelihahmo voi väliaikaisesti saada jonkin erikoiskyvyn. Pelihahmon muuttuessa esimerkiksi hetkellisesti kuolemattomaksi, hahmo alkaa välkkyä tai pelimaailman värit muttuvat esimerkiksi kirkkaammiksi. Animaatiolla voidaan myös ilmaista nopeutta ja sen muutosta (Kuuranta 2011), kuten pelihahmon vaatteiden lepattaessa sen juostessa lujaa.

KUVIO 8. Pelihahmon tilan ilmaiseminen animaation avulla (Replica Island Media n.d.)

Pelihahmon terveyden tilan ilmaiseminen on yleinen sovellutus hahmoanimaatiolle: hahmon ottaessa vahinkoa sen liikkuminen muuttuu vaivalloisemmaksi tai sen väri muuttuu punasävyiseksi. Pelihahmon ollessa huomion keskipisteenä pelaaja ei voi olla huomioimatta vahinkoanimaatiota, niin kuin voisi käydä tarkkaillessa pelihahmon vahinkoja ruudulla olevista käyttöliittymäelementeistä. Toisaalta animaatio ei korvaa numeerisen tiedon esittämistä (Kuuranta 2011), joten yleensä molempien yhdistäminen antaa pelaajalle eniten tietoa.

Pelihahmoa animoidessa sille luodaan animaatiosyklejä, jotka toistavat hahmon liikkeitä ja toimintoja jollain tietyllä aikavälillä. Pelihahmojen uskottavuutta rikkovaa

konemaista liikehdintää voidaan pyrkiä vähentämään lisäämällä animaatiosykleihin poikkeavuuksia (Kuuranta 2011). Näin hahmon tekemät liikkeet ja eleet eivät aina noudataakaan säännöllistä kaavaa ja hahmo koetaan luonnollisempana.

4.2.5 Pelimaailma

Pelimaailmalla tarkoitetaan ympäristöä, jonka kanssa pelaaja on vuorovaikutuksessa pelin aikana. Pelimaailmaan kuuluu hyvin olennaisesti se, että se reagoi kaikkiin pelaajan/pelihahmon tekemisiin. Mitä enemmän pelaajan teot vaikuttavat ympäristöön, sitä realistisempana pelimaailma voidaan kokea. Animaation keinoin voidaan esimerkiksi hajottaa seinä pelihahmon juostessa lujaa sen läpi.

Peliympäristön elävöittäminen animaation keinoin luo peliin tunnelmaa ja antaa lisäinformaatiota pelin tapahtumista. Jos pelaaja astuu pelimaailmassa esimerkiksi vihamieliselle ja uhkaavalle alueelle voidaan tämä ilmaista esimerkiksi animoimalla taustalle salamoiva taivas ja huojuvia puita. Pelaajan päästessä vihamielisen alueen ohi, taivas selkeytyy ja taustalla lentelevät perhoset kertovat pelaajan olevan turvassa.

4.2.6 Pelin efektit

Pelin efekteillä tarkoitetaan tässä siinä esiintyvien objektien välisen vuorovaikutuksen ilmentämistä. Esimerkiksi pelihahmon jarruttaessaan juoksuaan voidaan tämä esittää animoimalla hiekkapilven pöllähdys. Useasti riippuen pelin hakemasta tyylistä, efektien määrä ja näyttävyys vaihtelee. Efektit voivat olla tarkoituksella liioittelevia jos se tukee haettua pelityyliä, esimerkiksi humoristista tai hieman jopa lapsellista. Opetustyylisissä peleissä efektit ovat useasti hyvin hillittyjä, jotta itse pelin tarkoitus pysyy selkeänä.

Efektien käyttö ei oletusarvoisesti tee pelistä viihdyttävää, mutta niiden oikealla käytöllä on positiivinen vaikutus tähän. Viihdyttävyyden lisäksi efektit vahvistavat pelin illuusiota siitä, että siinä tapahtuu enemmän kuin itse asiassa oikeasti tapahtuu. Koska ruudulla on enemmän kohteita mihin silmä voi hakeutua, on vaikeampi keskittyä tiettyyn pisteeseen ja näin syntyy vaikutelma vauhdikkuudesta (Kuvio 9).

KUVIO 9. Efektianimaatiot lisäävät pelin viihdearvoa ja lisäävät yksityiskohtia, joihin silmä voi hakeutua (Angry Birds 2009).

4.2.7 Pelaajan ohjaaminen

Joskus peleissä tulee tilanteita, joissa pelaaja ei tiedä miten hänen tulisi edetä kyseisen kohdan yli. Pelaaja kohtaa esimerkiksi esteen, jonka ohi hänen tulisi päästä edetäkseen pelissä. Lukuisista yrityksistä huolimatta pelaaja ei keksi ratkaisua esteen ohittamiseen ja hän turhautuu. Näissä tilanteissa pelaajalle voidaan näyttää pieniä vihjeitä, kuinka kohdassa tulisi edetä. Ruudulle voi esimerkiksi animoida katkoviivana reitin, jota seuraamalla esteen voi ohittaa. Jos ratkaisua ei heti haluta paljastaa, voidaan ensin animoida ruudulle tekstivihje.

Animaatioilla voidaan helposti korostaa huomionarvoiset kohteet ruudulla ja peleissä tätä käytetään yleisesti osoittamaan pelaajan tarvitsemia esineitä tai etenemiseen vaadittavia interaktiivisia elementtejä, kuten vipuja ja painikkeita. Animaation kestolla on merkitystä siihen, miten tärkeän viestin se välittää (Kuuranta 2011). Jos pelaaja poimii pelissä vähämerkityksisen esineen ei animaation tarvitse olla mahtipontinen. Jos hän puolestaan löytää esimerkiksi pelin parhaan varusteen voi tämän merkityksen korostaa näyttävällä animaatiolla.

4.3 Interaktiiviset animaatiot

Interaktiivisissa animaatioissa voidaan lisätä vaihtelevuutta animaatioon sovelluksen ja sen käyttäjän välisen vuorovaikutuksen kautta. Sovellus voi lukea esimerkiksi käyttäjän sormen liikkeitä kosketusnäytöllä tai puhelimen liikesensoireita, ja animaatio voi muuttua näiden seurauksesta. Animaation tapahtumia voidaan sitoa toki myös käyttöliittymäkomponentteihin kuten painikkeisiin.

Interaktiivisia animaatioita käytetään yleisesti Android-peleissä ja interaktiivisissa taustakuvissa (live wallpapers). Talking Tom Cat on esimerkki menestyneestä Android-pelisovelluksesta, jossa hyödynnetään interaktiivisia animaatiota. Kyseisessä pelissä pelihahmona toimiva kissa reagoi käyttäjän tekemisiin, kuten sormella tökkimiseen, silittämiseen ja jopa käyttäjän puheeseen. Pelihahmon animaatiot mukautuvat näiden käyttäjän eleiden mukaan. Android Universe Free on puolestaan esimerkki interaktiivista animaatiota hyödyntävästä taustakuvasta. Siinä Androidin logosta tutut robottihahmot leijailevat avaruudessa, ja käyttäjä pystyy antamaan niille lisää vauhtia sormellaan. Animaation kontrollointi toimii myös kun se on asetettuna puhelimen taustakuvaksi. Nämä live wallpaperit vaativat toimiakseen vähintään Androidin version 2.1 (What are live wallpapers? n.d).

4.4 Käyttöliittymä ja näkymät

Käyttöliittymän animointi antaa käyttäjälle visuaalista viestiä siitä, mitä sovelluksessa on tapahtumassa, jonka lisäksi se myös antaa sovellukselle viimeistellyn ilmeen (Adding Animations n.d.). Androidissa käyttöliittymän animointi on erityisesti hyödyllistä kun siirrytään näkymästä toiseen. Ilman animoitua ruutusiirtymää käyttäjän voisi olla hankala hahmottaa visuaalisesti tapahtuvaa siirtymää näkymästä toiseen. Käyttöliittymäanimaatioiden turhanpäiväinen käyttö voi johtaa sovelluksen hidastumiseen, joten kannattaa miettiä tarvitseeko jokainen siirtymä animointia.

Yksi yleisimmistä käyttöliittymissä ja näkymissä käytetyistä animaatioista on niin sanottu häivytys (crossfade), jossa näkymästä toiseen siirrytään vaiheittain. Häivytysanimaatiota käytetään yleisesti ladatessa uutta sisältöä saman näkymän si-

sälle tai siirryttäessä sovelluksessa toiseen näkymään. Häivytysanimaation tekeminen kahden näkymän välille edellyttää ensinnäkin näkymien määrittelyä, tässä esimerkissä häivytys tehdään skrollattavan tekstikentän ja latauspalkin (ProgressBar) välille. Esimerkki on tehty mukaillen (Adding Animations n.d.) Crossfading Two Views esimerkkiä.

Näkymien määrittelyt layout-xml tiedostossa:

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ScrollView
        android:id="@+id/content"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TextView
            style="?android:textAppearanceMedium"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:lineSpacingMultiplier="1.2"
            android:padding="16dp"
            android:text="@string/lorem_ipsum" />

    </ScrollView>

    <ProgressBar android:id="@+id/loading_spinner"
        style="?android:progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />
</FrameLayout>
```

Näkymälle joka halutaan piilottaa, asetetaan näkyvyyden arvoksi *GONE.* Lisäksi määritetään animaation kesto - oletuksena tämä on *short,* mutta myös *medium* ja *long* arvoja voi käyttää mikäli haluaa häivytyksen kestävän kauemmin.

```
public class MainActivity extends Activity {

private boolean mContentLoaded;
private View mContentView;
private View mLoadingView;
private int mShortAnimationDuration;


@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mContentView = findViewById(R.id.content);
        mLoadingView = findViewById(R.id.loading_spinner);
```

```
        // piilotetaan tekstikenttä
        mContentView.setVisibility(View.GONE);

        // määritetään lyhyen animaation kestoksi järjestelmän   ole-
tuksena oleva lyhyen animaation arvo
           mShortAnimationDuration = getResources().getInteger(
           android.R.integer.config_shortAnimTime);
}
```

Lisätään sovelluksen ylälaidassa olevaan palkkiin menu-item, jota painamalla voi vaihtaa näkymiä:

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        getMenuInflater().inflate(R.menu.activity_crossfade, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {

            case R.id.action_toggle:
                // Tutkitaan kumpi näkymistä ladataan
                mContentLoaded = !mContentLoaded;
                showContentOrLoadingIndicator(mContentLoaded);
                return true;
        }

        return super.onOptionsItemSelected(item);
    }
```

Seuraavaksi määritellään häivitysanimaation aiemmin luotujen näkymien välille:

```
final View showView = contentLoaded ? mContentView : mLoadingView;
final View hideView = contentLoaded ? mLoadingView : mContentView;
// asetetaan tekstikentän alpha-arvoksi 0%  - näkymä on olemassa
// mutta se on täysin läpinäkyvä
showView.setAlpha(0f);
showView.setVisibility(View.VISIBLE);

// animoidaan tekstikenttä täysin näkyväksi ja poistetaan mahdolliset
// kuuntelijat näkymältä
showView.animate()
           .alpha(1f)
           .setDuration(mShortAnimationDuration)
           .setListener(null);

// animoidaan latauspalkin alpha-arvo 0% asti
// lopuksi asetetaan sen näkyvyyden arvoksi GONE (optimointina)
hideView.animate()
           .alpha(0f)
           .setDuration(mShortAnimationDuration)
           .setListener(new AnimatorListenerAdapter() {
                     @Override
                     public void onAnimationEnd(Animator animation){
                               hideView.setVisibility(View.GONE);
                     }
           });
}
```

Muita yleisiä käyttöliittymissä käytettyjä animaatioita ovat näkymän liukuanimaatio (screen slide), näkymän pyöräytys (card flip) sekä näkymän suurennus (zooming) (Adding Animations n.d.). Lisäksi Android tarjoaa oletuksena mahdollisuuden animoida layoutissa tapahtuvat muutokset, kuten listaelementin lisäämisen ja poistamisen, Androidin versiosta 3.0 alkaen. Tämä vaatii *android:animateLayoutChanges="true"* -atribuutin lisäämistä animoitavalle layoutille, jonka jälkeen kaikki siihen kohdistuvat muutokset tapahtuvat animoituna (Adding Animations n.d.). Tämä on esitellyistä menetelmistä yksinkertaisin ja nopein tapa lisätä sovellukseen edes vähän elävyyttä.

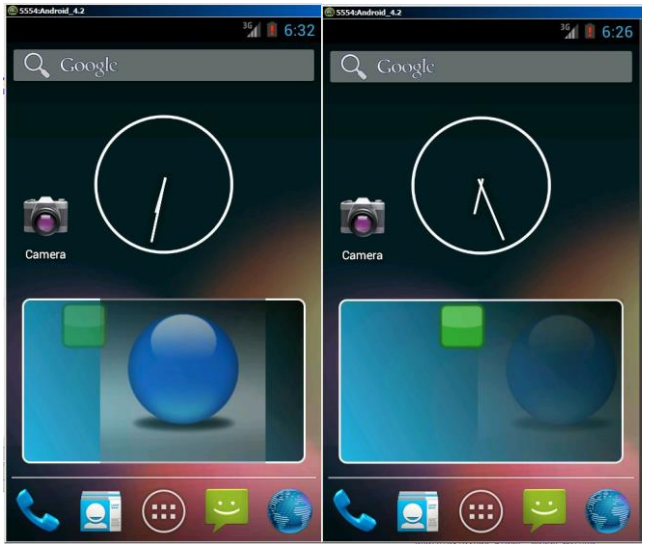
4.5 Widgetit

Widgeteistä puhuttaessa Androidin yhteydessä tarkoitetaan pienoisohjelmia, joita käyttäjä voi lisätä puhelimen ”kotinäkymään” tai lukitusruutuun. Widgettien idea on yleensä lisätä jokin oikopolku nopeuttamaan puhelimen käyttöä, kuten langattoman verkon kytkeminen päälle yhdellä painalluksella. Widgettien avulla voi myös saada tärkeän tiedon tai ilmoituksen yhdellä vilkaisulla, kuten saapuneet sähköpostit tai säätiedot.

Yksinkertaisten animaatioiden kuten fade-efektien lisääminen widgetteihin on mahdollista, ja tätä monimutkaisempien animaatioiden lisääminen ei suositeltavaa ellei sitten luo widgettiä alusta loppuun itse. Widgettien päivittäminen vaatii aina tiedonsiirtoa HomeScreen -aplikaation ja widgetin ulkoasun luovan AppWidgetProviderin välillä, mikä itsessään on jo resursseja vievä prosessi (Vogel 2013). Widgetit on lisäksi suunniteltu päivittymään oletuksena vain 30 minutiin välein (mt). Raskaan animaation pyörittäminen ”kotinäkymässä” voi hidastaa sen toimintaa ja saada sen vastaamaan huonosti käyttäjän sormieleisiin.

Yksi tapa lisätä yksinkertainen animaatio widgettiin on käyttää ViewFlipper-luokkaa, joka on ViewAnimator-luokan alaluokka (ViewFlipper n.d.). ViewFlipperillä voi tehdä animaatiosyklin siinä määritetyille elementeille. Tässä esimerkissä tehdään animaatiosykli kolmelle kuvalle, kuva vaihtuu viiden sekunnin välein ja kuvat liukuvat vasemmalta oikealle (kuvio 10):

```
<ViewFlipper android:layout_width="match_parent"
android:layout_height="wrap_content"
android:measureAllChildren="true"
android:flipInterval="5000"
android:autoStart="true"
android:inAnimation="@android:anim/slide_in_left"
android:outAnimation="@android:anim/slide_out_right"
android:animateFirstView="true">
<ImageView
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:src="@drawable/kuva1" />
<ImageView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:src="@drawable/kuva2" />
<ImageView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:src="@drawable/kuva3" />
</ViewFlipper>
```



KUVIO 10. Yksinkertainen kuvan liukuanimaatio vasemmalta oikealle lisättynä widgettiin ViewFlipperin avulla

5 ANIMAATIOIDEN TOTEUTUS ANDROID-SOVELLUKSEEN

5.1 Sopivimman lähestymistavan valinta

Tässä opinnäytetyössä on jo esitelty esimerkkien muodossa joitakin tapoja tehdä animaatioita Android-sovelluksiin, kuten yksinkertaisen animaation lisääminen widgettiin käyttäen ViewFlipper-luokkaa. Tässä kappaleessa esitellään yleisimmin käytetyt animointimenetelmät ja pohditaan niiden sopivuutta erilaisiin sovelluksiin. Kappaleessa myös esitellään yksinkertaisen pelihahmoanimoinnin toteutus (kuvio 12), jonka tarkoituksena on demonstroida eri lähestymistapoja animoinnin toteuttamiseen.

5.1.1 Frame-by-Frame -animaatio

Frame-by-Frame -animaatiossa animaatio luodaan esittämällä kuvia peräkkäin tietyllä syklillä. Tämä on animointimetodeista lähimpänä perinteistä animaatiota, joka koostuu eri kuvista toistettuna järjestyksessä peräkkäin. Androidin yhteydessä tästä animaatiotyypistä puhutaan myös nimellä Drawable Animation, sillä siinä ladataan kuvia Drawable –resursseista jotka toistetaan peräkkäin muodostaen animaation.

Frame-by-Frame –animaatiossa animoitavat kuvat on helpoin määritellä XML-tiedostossa, jossa juurielementtinä (root node) toimii <animation-list> (Drawable Animation n.d.). Jokainen lapsielementti (item) vastaa yhtä piirrettävää kuvaa drawable-resursseissa. Android:duration –parametri määrää kuinka monta millisekuntia kyseinen kuva näkyy ruudulla ennen seuraavan kuvan vaihtumista. Animation-list on oletuksena asetettuna toistamaan animaatiosykliä loputtomasti. Mikäli animaation haluaa toistuvan vain kerran voi animation-listille antaa atribuutiksi android:oneshot=”true”, jolloin kuvasarja pyörähtää kerran läpi ja jää viimeiseen frameen.

Animaation uudelleen käynnistäminen on toteutettu frameworkissa hieman hämmentävästi, sillä se on sidottu setVisible-funktioon. Pelkkä .start()-komennon

toistaminen ei aloita animaatiota uudelleen, vaan se siirtyy pelkästään viimeiselle framelle. Sen sijaan, käskyllä setVisible(false, true); animaatio palautetaan ensimmäiselle framelle, jonka jälkeen se voidaan toistaa uudelleen .start()-komennolla. Huomioitavaa on myös, että .start()-komennon kutsuminen Activityn onCreate()-metodissa ei toista animaatiota, sillä AnimationDrawable ei ole vielä tässä vaiheessa liitettynä ikkunaan (Drawable Animation n.d.). Kun animaatio on määritelty XML-tiedostossa voi tämän lisätä esimerkiksi ImageView-elementin taustaksi ja lisätä halutut toiminnallisuudet.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- myframeanimation.xml  -->
<animation-list
xmlns:android="http://schemas.android.com/apk/res/android" and-
roid:oneshot="false">
<item android:drawable="@drawable/hahmo_1" android:duration="100" />
<item android:drawable="@drawable/hahmo_2" android:duration="100" />
<item android:drawable="@drawable/hahmo_3" android:duration="100" />
...
<item android:drawable="@drawable/hahmo_17" android:duration="100" />
</animation-list>
```

Tässä esimerkissä aiemmin luotu myframeanimation.xml lisätään ImageViewin i taustalle, ja animaatio lähtee toistumaan kun ikkuna saa fokuksen:

```
AnimationDrawable hahmoAnimaatio;

            public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.activity_frame_animation);

              ImageView i = (ImageView) findViewById(R.id.tausta);
              i.setBackgroundResource(R.drawable.myframeanimation);
              hahmoAnimaatio = (AnimationDrawable) i.getBackground();

            }
            public void onWindowFocusChanged(boolean hasFocus){

                        super.onWindowFocusChanged(hasFocus);

                        if(hasFocus){
                        hahmoAnimaatio.setVisible(false, true);
                        hahmoAnimaatio.start();
                        }
            }
}
```

Frame-by-Frame –animaation heikkous tulee nopeasti vastaan, kun kasvattaa animoitavien kuvien määrää. Kipurajaksi havaittiin noin 50 framea riippuen vähän kuvien koosta. Myös useamman kuin yhden animaation yhtäaikainen toistaminen

johti usein sovelluksen kaatumiseen (OutOfMemory exception). Aiemmin esitellyssä esimerkissä animaatiossa käytettiin 17-jpeg-kuvaa, joiden koko oli 300x300 pikseliä ja näillä luvuilla animaatio toistui sujuvasti. Animoitavat kuvat olivat valmisteltu kuvankäsittelyohjelmassa siten, että hahmoa liikutettiin hieman jokaisessa kuvassa ja toistettaessa ne peräkkäin syntyy vaikutelma hahmon liikkeestä.

Ilman minkäänlaista kuvien etukäteen lataamista (preloading) animaation käynnistämisessä havaittiin olevan yli sekunnin viive, ja Animation Drawable –luokka ei itsessään tarjoa mitään keinoja tähän. Yhtenä ratkaisuna pohdittiin mahdollisuutta ladata animaatio ensin piilotettuun ImageView-elementtiin, josta animaatio ladattaisiin tarvittaessa näkyville.

Frame-by-Frame –animointia voisi aiempiin seikkoihin vedoten käyttää lähinnä hyvin yksinkertaisten animaatioiden toteuttamiseen, kuvien määrän ja koon ollessa pieniä. Muutoksien tekeminen tällä menetelmällä toteutettuun animaation on työlästä, sillä se edellyttää aina kuvatiedostojen muokkaamista. Hyvinä puolina Frame-animaatiossa on sen helppo toteutus ja toimintavarmuus Androidin versiosta riippumatta. Frame-by-Frame –animaatiota voisi harkita käytettäväksi esimerkiksi elävöittämään sovelluksen taustaa tai käyttöliittymäkomponentteja, joissa voidaan käyttää samaa animaatiolooppia.

5.1.2 Tween-animaatio

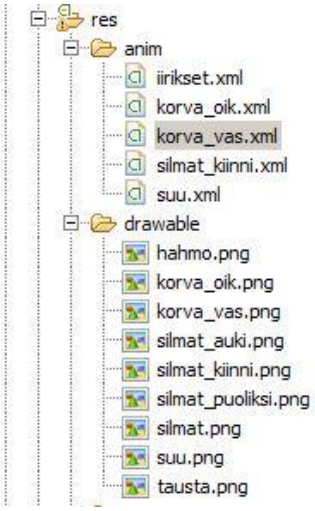
Tween-animaatio mahdollistaa graafisten elementtien eri ominaisuuksien, kuten läpinäkyvyyden (alpha), käännöksen (rotation), skaalauksen (scale) ja paikan (translate) animoimisen. Tween-animaatiolla tarkoitetaan yleisesti puhuttaessa animointitekniikkaa, jossa animoitavat arvot muuttuvat kahden pisteen, key framen, välillä tietyn aikajakson aikana. Tietokone huolehtii näiden key framejen väliin jäävien framejen piirtämisestä, sekä animoitavan objektin muodomuutoksista. Tween-animaatiosta puhutaan Androidin yhteydessä myös nimellä Property Animation, sillä siinä animoidaan juurikin objektin tiettyjä ominaisuuksia. Tween-animaatio antaa huomattavasti enemmän vapauksia animaation tekoon kuin aiemmin mainittu Frame-animaatio.

Tween-animaatiossa animaation määrittely on Frame-animaation tapaan helpoin tehdä XML-tiedostossa, erona tässä on että juurielementtinä (root node) voi olla <alpha>, <scale>, <translate>, <rotate>, <objectAnimatior> tai <set> (Animation Resources. n.d.). Koska animaatiossa yleensä halutaan muuttaa yhtäaikaisesti useampaa eri arvoa, käyttämällä <set>-elementtiä tämän sisään voi määrittää kaikki muutettavat arvot. Set-elementtejä voi myös käyttää sisäkkäin ja jaotella animoitavia ominaisuuksia omiksi kokonaisuuksiksi tarkemmin tällä tavalla.

Android:interpolator-atribuutti määrittää muutoksen nopeuden animaatiossa, kuten esimerkiksi kiihtyvyyden tai hidastuvuuden animaation loppua kohden (mt.). Tämän voi asettaa koko animaatiosetille määrittämällä android:shareInterpolator=”true”, tai sitten määrittää jokaiselle animaation osalle erikseen. Android:ordering puolestaan määrittää, toistetaanko <set>-elementissä määritetyt animoitavat asiat yhtä aikaa vai XML-tiedoston rakenteen mukaisessa järjestyksessä.

Yksi Androidin version 3.0 (Honeycomb) esittelemistä uusista animaatio-ominaisuuksista oli AnimatorSet-luokka, joka mahdollistaa animaatioiden ketjuttamisen sekä niiden toistojärjestyksen määrittämisen (Property Animation n.d.). AnimatorSet-luokkaa käyttämällä animaatioita voi käskeä toistumaan yhtäaikaisesti, peräkkäin tai tietyn viiveen jälkeen. AnimatorSet-objekteja voi itsessään myös ketjuttaa haluamallaan tavalla, mikä mahdollistaa monipuolisten animaatoiden toteuttamisen.

Tween-animaationa toteutettu hahmoanimaatio muodostui useasta eri osasta, jotka käyvät ilmi kuviosta 11. Hahmon eri osien, kuten silmien, korvien ja suun liikkeet animoitiin omina kokonaisuuksina ja lopulta yhdistettiin yhdeksi animaatioksi käyttäen AnimatorSet-luokkaa. Hahmon eri osien animoimisessa omina kokonaisuuksina on suuri etu, sillä jonkin osan animoinnin muuttaminen ei vaadi koko animaation muuttamista. Näin tehtynä animaatiota on helppo hienosäätää ja muuttaa hahmon liikkeitä.



KUVIO 11. Tween-animaationa toteutetun hahmoanimaation resurssit Eclipse-kehitysympäristössä

Animaation osien lataaminen niitä vastaavista XML-tiedostoista:

```
// ladataan animaatiot niiden XML-resursseista
final AnimatorSet iirikset_anim = (AnimatorSet) AnimatorInfla-
ter.loadAnimator(this, R.anim.iirikset);

final AnimatorSet silmat_kiinni_anim = (AnimatorSet) AnimatorInfla-
ter.loadAnimator(this, R.anim.silmat_kiinni);

final AnimatorSet suu_anim = (AnimatorSet) AnimatorInfla-
ter.loadAnimator(this, R.anim.suu);

final AnimatorSet korva_vas_anim = (AnimatorSet) AnimatorInfla-
ter.loadAnimator(this, R.anim.korva_vas);

final AnimatorSet korva_oik_anim = (AnimatorSet) AnimatorInfla-
ter.loadAnimator(this, R.anim.korva_oik);
```

Animaatioiden liittäminen niitä vastaaviin ImageView-elementteihin:

```
// kohdistetaan animaatiot niitä vastaaviin grafiikoihin
iirikset_anim.setTarget(iirikset_imgV);
silmat_kiinni_anim.setTarget(silmat_kiinni_imgV);
suu_anim.setTarget(suu_imgV);
korva_vas_anim.setTarget(korva_vas_imgV);
korva_oik_anim.setTarget(korva_oik_imgV);
```

Määritetään järjestys, jossa animoidut osat toistetaan hahmoanimaatiossa:

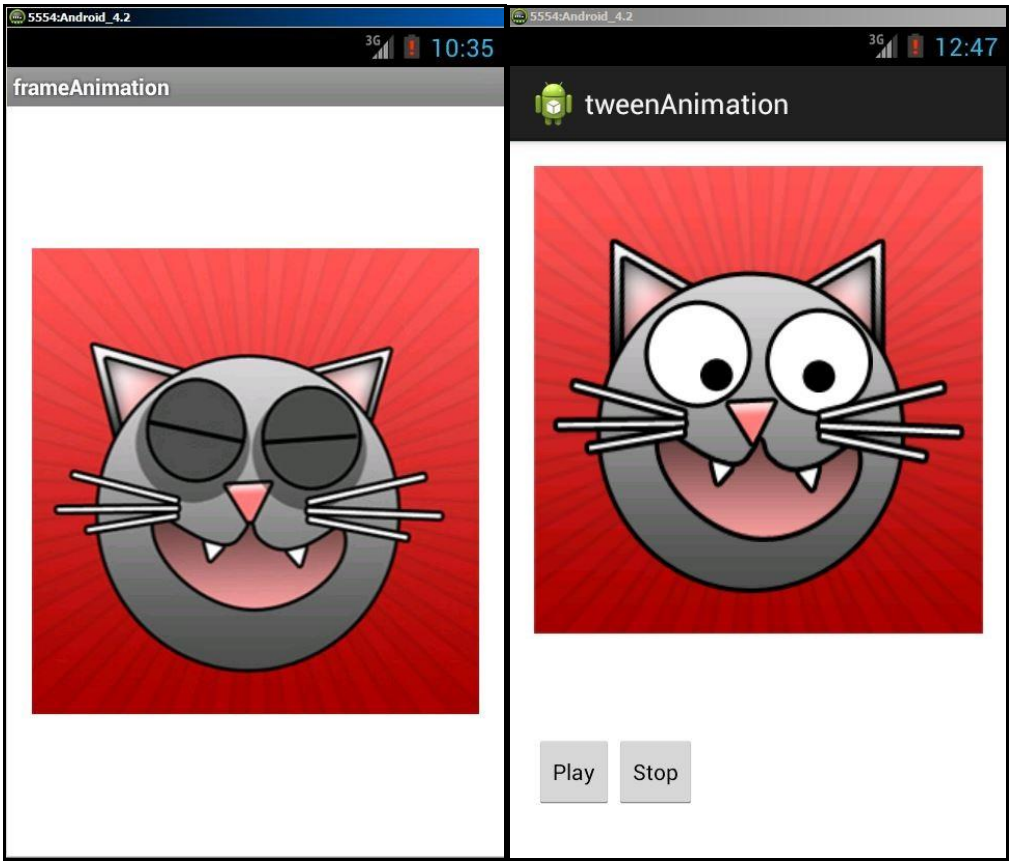
```
// silmän iirikset liikkuvat samalla kun suu aukeaa
hahmoanimaatio.play(iirikset_anim).with(suu_anim);
// silmät sulkeutuvat kun ne ovat liikkuneet aikansa
hahmoanimaatio.play(silmat_kiinni_anim).after(iirikset_anim);
// määritetään molemmat korvat liikkumaan yhtä aikaa
hahmoanimaatio.play(korva_vas_anim).with(korva_oik_anim);
```

Animaation käynnistäminen sidottiin painikkeeseen:

```
play_btn.setOnClickListener(new OnClickListener(){
public void onClick(View v){
// käynnistetään animaatio
hahmoanimaatio.start();
}
});
```

Hahmoanimaatiolle liitettiin kuuntelija, joka käynnistää animaation uudelleen sen päättyessä:

```
// lisätään hahmoanimaatiolle kuuntelija
hahmoanimaatio.addListener(new AnimatorListenerAdapter(){
public void onAnimationEnd(Animator animation){
// kun animaatiosykli menee loppuun, käynnistetään se uudestaan
super.onAnimationEnd(animation);
hahmoanimaatio.start();
}
});
```



KUVIO 12. Lopullisessa hahmoanimaatiossa hahmon korvat, silmät ja suu liikkuvat. Molemmilla toteutustavoilla päästiin lähes samannäköiseen lopputulokseen.

Tween-animaatio soveltuu erinomaisesti animaatioihin, joissa tarvitsee muuttaa useaa eri ominaisuutta yhtäaikaa tai tietyssä järjestyksessä. AnimatorSet –luokka

mahdollistaa animaatioiden yhdistelyn ja näiden toistojärjestyksen määrittämisen, mistä johtuen tällä menetelmällä on mahdollista tehdä monipuolisia animaatioita. Tween-animaation menetelmiä voi hyödyntää sekä peleissä että käyttöliittymäelementten animoinnissa, ja yksinkertaisuudessaan se mahdollistaa objektien liikuttelun ja ominaisuuksien muuttamisen määrittämällä vain alku- ja lopputilanteen.

5.1.3 Siirtymäanimaatiot

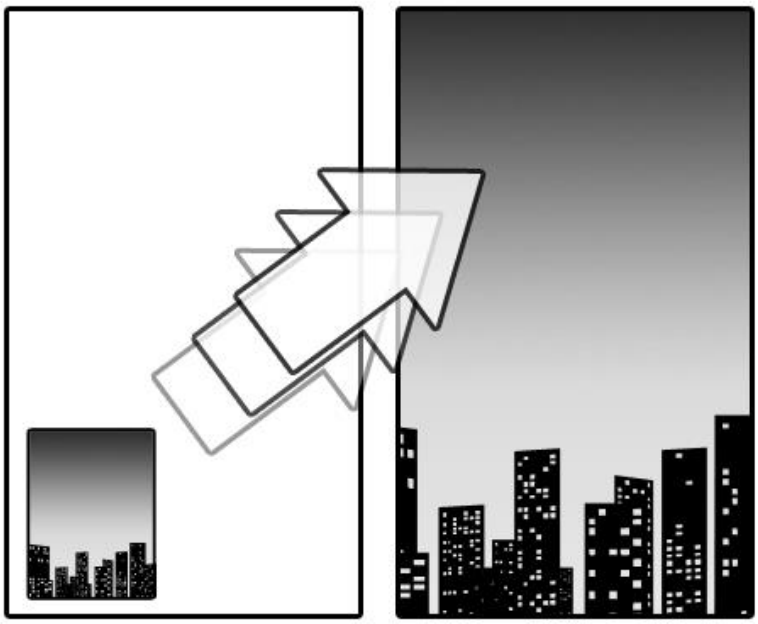
Tässä opinnäytetyössä on jo aiemmin esitelty hieman animaation käyttämistä käyttöliittymässä, kuten häivytysanimaation tekemistä kahden näkymän välille saman ikkunan (window) sisällä. Tässä kappaleessa esitellään kustomoidun siirtymäanimaation tekemistä sovelluksen eri ikkunoiden/näkymien välille.

Sovellus käyttää Android-käyttöjärjestelmän oletusanimaatiota siirtymäefekteihin ellei sovellukseen määritä omia kustomoituja siirtymäanimaatioita. Androidin kehittäjän Chet Haasen mukaan käyttäjille on tärkeää tarjota yhdenmukainen kokemus myös siirtymäanimaatioiden osalta heidän käyttäessään Android-sovelluksia. Tästä syystä siirtymäanimaatio tulisi pitää oletuksena siirryttäessä sovellukseen. Sovelluksen sisällä omien kustomoitujen siirtymäanimaatioiden käyttäminen on suotavaa sillä näin sovellukseen saadaan yksilöllinen ilme. (DevBytes: Window Animations 2013).

Kustomoiduilla siirtymäanimaatioilla on mahdollista luoda halutunlaiset siirtymäefektit sovelluksen eri näkymien välille. Sovellustyypistä riippumatta siinä on yleensä aina useita näkymiä, kuten päänäkymä sekä alanäkymä/alanäkymiä. Pelisovelluksessa näkymiä ovat usein päävalikko, päävalikon alavalikot, itse pelinäkymä sekä mahdollinen pistenäkymä (score screen). Kustomoiduilla ruutusiirtymillä sovellukselle voidaan luoda yhtenevä ja samalla tyylikäs ilme.

Androidin versiossa 4.1 (Jellybean) esiteltiin uudentyyppisiä siirtymäanimaatioita, joista Scale Animation mahdollistaa näkymän skaalaamisen esille jostakin pisteestä, esimerkiksi painikkeesta. Rajoituksena tässä on, että näkymän voi skaalata vain pienemmästä suurempaan (DevBytes: Window Animations 2013).

Scale-animaation hienona puolena on, että näkymän voi skaalata näkyville myös bittikarttakuvasta, joka voi olla esimerkiksi esikatselukuva kyseisestä näkymästä (Kuvio 13).



KUVIO 13. Näkymän skaalaaminen esille esikatselukuvasta Scale-animaationa

Tässä esimerkissä on esitettynä kuinka kyseinen Scale-animaatiosiirtymä toteutetaan klikkaamalla esikatselukuvaa näkymästä (DevBytes: Window Animations 2013):

```
// näkymän skaalaaminen esille sitä vastaavasta esikatselukuvasta
thumbnail.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
BitmapDrawable drawable = (BitmapDrawable) thumbnail.getDrawable();
Bitmap bm = drawable.getBitmap();
Intent subActivity = new Intent(WindowAnimations.this, AnimatedSubAc-
tivity.class);
Bundle scaleBundle = ActivityOptions.makeThumbnailScaleUpAnimation(
thumbnail, bm, 0, 0).toBundle();
startActivity(subActivity, scaleBundle);
}
});
```

5.2 Valmiit animointikirjastot

Opinnäytetyön yhteydessä Tween-animaationa toteutetun hahmoanimaation testauksen yhteydessä huomattiin, että jotkin siinä käytetyt animointiluokat eivät olleet tuettuja eräässä testipuhelimessa. Tämä johtui siitä, hahmoanimaatiossa käytetty AnimatorSet-luokka esiteltiin vasta Androidin versiossa 3.0 (Honeycomb) ja testipu-

helimessa versiona oli 2.3.5. Ratkaisuna ongelmaan löydettiin avoimen lähdekoodin (Apache 2.0 lisenssi) animointikirjasto NineOldAndroids, joka mahdollistaa Android 3.0 –versiossa lisättyjen animointiluokkien käyttämisen Androidin versioon 1.0 saakka (NineOldAndroids 2012).

NineOldAndroids –animointikirjaston käyttö vaatii sen linkittämistä projektin resursseihin, jonka jälkeen sitä voi käyttää samoin kuin Honeycombin animointiluokkia. Esimerkki animointiluokkien korvaamisesta:

```
import com.nineoldandroids.animation.AnimatorSet;
// ennen: import android.animation.AnimatorSet;
```

5.3 Testaus

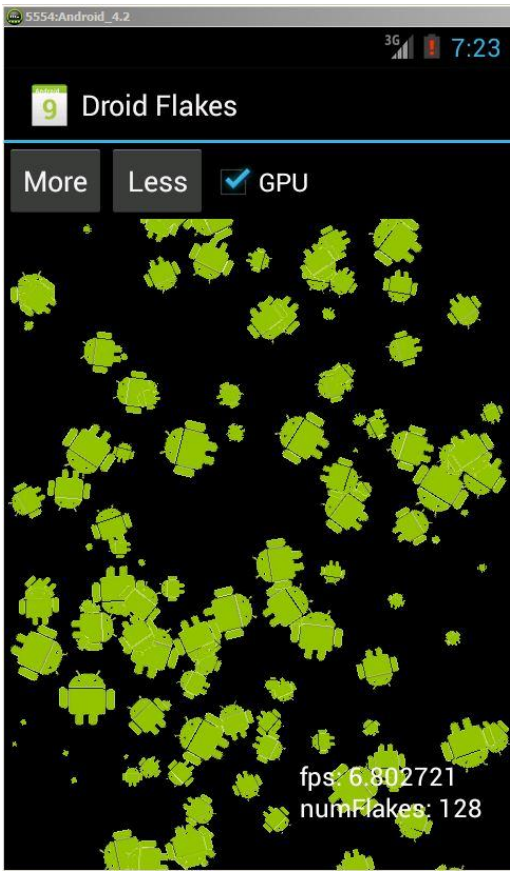
Opinnäytetyön yhtenä osana oli tutkia eri animointitekniikoiden eroavaisuuksia myös suorituskyvyn osalta. Opinnäytetyön yhteydessä tehty testaus koostui kahdesta eri osiosta: ensimmäisessä testissä tutkittiin yleisimpien Android-puhelimien suorituskykyä animaation ja grafiikan piirtämisen osalta rasitustestissä, toisessa testissä tutkittiin aiemmin esiteltyjen hahmoanimaatiototeutusten (frame-animaatio ja tween-animaatio) eroavaisuuksia. Jälkimmäistä testiä ei saatu erinäisistä syistä suoritettua kaikilla testipuhelimista.

Testattuja puhelimia olivat HTC Desire (2.3.3), Samsung Galaxy Gio (2.3.5), Samsung Galaxy Mini 2 (2.3.6), Samsung Galaxy S2 (4.1.2) sekä Samsung Galaxy S3 (4.2.2, Cyanogen Mod 10.1). Näistä testipuhelimista kolme ensin mainittua ovat ominaisuuksiltaan melko keskinkertaisia älypuhelinten luokassa, ja kaksi viimeksi mainittua ovat ominaisuuksiltaan huippupäästä.

5.3.1 Droid Flakes –testi

Testauksen ensimmäisen osan tarkoituksena oli tutkia yleisimpien Android-puhelimien suorituskykyä animoidun grafiikan piirtämisessä ruudulle. Kyseinen testi suoritettiin yhdellä NineOldAndroids –animointikirjaston mukana tulevista esimerkeistä nimeltään Droid Flakes. Droid Flakes –testisovelluksessa Canvas-elementille piirretään Android-robottihahmoja, jotka vierivät alaspäin hiutaleiden tapaan (kuvio

14). Käyttäjä voi kasvattaa tai vähentää piirrettävien hahmojen määrää, joka tuplaantuu tai puolittuu aina painettaessa vastaavaa painikketta. Ruudulla näkyy lisäksi kuvataajuuslaskuri (frames per second), joka kertoo animaation sulavuudesta. Mikäli testipuhelimessa oli vähintään Androidin versio 3.0, testissä kytkettiin myös laitteistokiihdytys päälle ja vertailtiin suorituskyvyn muutosta tämän kanssa.



KUVIO 14**.** Droid Flakes –testisovelluksessa mitattiin rasitusta nostamalla graafisten elementtien määrää.

Droid Flakes –testissä kaikki testipuhelimet suoriutuivat vähintään 256 animoidun graafisen elementin piirtämisessä ruudulle siten, että kuvataajuus pysyi lähellä tai yli 24 kuvaa sekunnissa, jota pidetään rajana sulavana koettavalle animaatiolle tai videolle. Vastaavasti kipurajaksi, eli pisteeksi jossa animaatio kävi selvästi hitaaksi ja tökkiväksi, havaittiin olevan 256 ja 512 ”hiutaleen” välissä. Laitteistokiihdytyksen kytkemisellä päälle havaittiin olevan todella suuri merkitys, ja esimerkiksi Galaxy S3 –testipuhelimella kuvataajuutta ei saatu laskettua alle 60 kuvaa sekunnissa kun laitteistokiihdytys oli kytkettynä päälle. Mielenkiintoisena havaintona Samsung Galaxy

Gio –testipuhelin ylsi jopa 90 fps –lukemaan. Syynä tähän epäiltiin, että joissakin uudemmissa laitteissa/Androidin versioissa voi olla rajoituksia kuvataajuuden suhteen akkukeston parantamiseksi tai näytön ominaisuuksien takia. Muissa testipuhelimissa ylärajaksi havattiin noin 60 kuvaa sekunnissa, joka yleensä on täysin riittävä erittäin sulavan animaation toteuttamiseen. (Liite 1).

5.3.1 Frame- ja tween-animaatio –testi

Testauksen toisessa osiossa tutkittiin opinnäytetyön yhteydessä tehdyn hahmoanimaation kahden eri toteutuksen eroavaisuuksia. Testissä oltiin erityisesti kiinnostuneita näiden toteutustapojen eroavaisuuksista muistinvarauksessa sekä prosessorin kuormituksessa testipuhelimilla. Tween-animaation osalta testattiin myös, kuinka paljon animoitavien osien määrä vaikuttaa aiemmin mainittuihin tutkimuskohteisiin.

Merkittävin tulos testissä oli, että frame-animaation havaittiin varaavan keskumuistia huomattavasti enemmän kuin tween-animaatiototeutuksen (liite 2). Tämä oli toisaalta ennalta arvattava tulos, sillä frame-animaatiossa käytettävät kuvat ladataan muistiin sovelluksen ajon aikana. Testipuhelimien välillä erot muistinvarauksessa olivat puolestaan erittäin suuria: vähiten (8,28 MB) vei Galaxy Mini 2 ja eniten (45 MB) vei Galaxy S3 (liite 2). Yhtenä selityksenä suurille eroille voi olla myös suuret erot testipuhelimien RAM-muistin määrässä, jota Mini 2:ssa on 512 MB ja Galaxy S3:ssa 2 GB (Samsung phones n.d.). Suhteutettuna siis käytettävissä olevaan kokonaismuistin määrään, erot eivät ole niin suuria kuin testitulokset antavat odottaa.

Tween-animaation huomattiin testissä kuormittavan enemmän prosessoria kuin frame-animaatio (liite 2). Tämä selittyy sillä, että tween-animaatiossa suoritettavat graafisten elementtien liikuttelut ja muodonmuutokset vaativat jatkuvaa laskemista prosessorilta, toisin kuin frame-animaatiossa, jossa uuden kuvan piirtäminen ruudulle ei rasita suoritinta lähellekkään samalla tavalla. Liikuteltavien osien määrän nostamisen tween-animaation huomattiin kasvattavan kokonaisrasitusta melko lineaarisesti, mikä vastasi ennakoitua lopputulosta (liite 2).

6 POHDINTA

Opinnäytetyön tärkeimpänä tavoitteena oli kartoittaa erilaisia mahdollisuuksia toteuttaa animaatioita Android-sovellukseen. Tähän liittyen tavoitteena oli lisäksi arvioida tutkimuksen kohteena olevien animointimenetelmien mahdollisuuksia ja rajoituksia sekä niiden soveltuvuutta eri tyyppisiin sovelluksiin.

Tuloksina saatiin kattava selvitys Androidin animointimahdollisuuksista, jossa on esitelty yleisimmin käytetyt animointimenetelmät, niihin liittyviä mahdollisia rajoituksia sekä havainnollistavia esimerkkejä. Lisäksi opinnäytetyöstä käy ilmi, että vektorigrafiikan käyttäminen natiivissa Android-sovelluksessa on mahdollista käyttämällä svg-android –kirjastoa, ja että 2d-grafiikan laitteistokiihdytyksen kytkemisellä päälle on huomattavia etuja grafiikan renderöinnissä. Property-animation eli tween–animointitekniikan havaittiin soveltuvan tekniikoista parhaiten kaikentyyppisiin sovelluksiin ja frame-animaatiotekniikan vain lähinnä yksinkertaisten animaatioiden toteutukseen. NineOldAndroids –animointikirjastoa käyttämällä löydettiin ratkaisu, jolla Androidin versiossa 3.0 esitellyt animointiominaisuudet sai toimimaan myös tätä vanhemmissa Androidin versioissa.

Opinnäytetyössä onnistuttiin täyttämään sille asetetut tavoitteet melko tarkasti. Eri animointitekniikoita pyrittiin havainnollistamaan mahdollisimman yksinkertaisilla esimerkeillä, joista myös kävisi ilmi kyseisen tekniikan mahdollisuudet ja käyttökohteet. Eri animointitekniikoiden soveltuvuutta erilaisiin sovelluksiin olisi voinut työssä tutkia tarkemminkin. Haasteena tähän liittyen oli, että hyviä esimerkkejä animointitekniikoiden sovellutuksista olemassa olevissa Android-sovelluksissa oli vaikea löytää. Opinnäytetyön yhteydessä suoritettu testaus keskitettiin animaatioiden suorituskyvyn, eri tekniikoilla tehtyjen toteutuksien ja grafiikan piirto-ominaisuuksien tutkimiseen testilaitteilla. Käytettäessä animaatioita sovelluksissa tulisi lisäksi tutkia miten nämä vaikuttavat koko sovelluksen, jonka osana ne ovat, suorituskykyyn.

Animointitekniikoita tutkiessa kävi nopeasti ilmi, että mitään oikotietä onneen näiden osalta ei ole, vaan animointi on Android-ympäristössä melko työlästä ja pitkälle

Androidin omien animointiluokkien käyttämistä. Mitään yksittäistä all-in-one –animointikirjastoa ei siis ole, joka soveltuisi kaikkii Android-sovelluksiin. Androidissa itsessään tutkimuskohteena haasteelliseksi teki se, että erot versioiden välillä olivat lähes poikkeuksetta suuria. Tästä syystä Androidin omia animointiluokkia käyttämälläkään ei animaatio aina toiminut versioiden välillä.

Android-ympäristö oli minulle täysin tuntematon aloittaessani opinnäytetyön, joten lähes kaikki tähän liittyvä tuli uutena tietona. Opinnäyteyössä haasteellista oli vaadittu ohjelmointiosaaminen, jota animointi Android-ympäristössä pitkälti edellyttää. Esimerkkejä tutkimalla ja omien tekemisellä logiikasta sai kuitenkin melko nopeasti kiinni. Ohjelmointiosaamisen lisäksi sain paremman käsityksen sovelluksen kehittämisestä mobiilialustalle, erityisesti Androidille. Opin tähän liittyen myös miten haasteellista on suunnitella ja toteuttaa sovellus, jonka tulisi parhaassa tapauksessa toimia mahdollisimman monella laitteella.

Opinnäytetyössä esitellyt tulokset ovat toivon mukaan eduksi Silmu Softwaren tulevissa Android-projekteissa, joissa tarvitaan myös animaatioita jossain muodossa.

LÄHTEET

Adding Animations. n.d. Android Developersin verkkosivusto. Viitattu 12.4.2013.
http://developer.android.com/training/animation/index.html

Android Supported Media Formats. n.d. Android Developersin verkkosivusto. Viitattu 8.3.2013.
http://developer.android.com/guide/appendix/media-formats.html

Androidilla ja Applella hurja markkinaosuus älypuhelimissa. 2013. Artikkeli Taloussanomien verkkosivustolla 28.1.2013. Viitattu 8.3.2013.
http://www.taloussanomat.fi/porssi/2013/01/28/androidilla-ja-applella-hurja-markkinaosuus-alypuhelimissa/20131514/170

Angry Birds. 2009. Rovio Entertainment. Viitattu 31.3.2013.

Animation Resources. n.d. Android Developersin verkkosivusto. Viitattu 22.4.2013.
http://developer.android.com/guide/topics/resources/animation-resource.html

Cunningham, A. 2012. What happened to the Android Update Alliance? Artikkeli Ars Technican verkkosivustolla 28.6.2012. Viitattu 11.3.2013.
http://arstechnica.com/gadgets/2012/06/what-happened-to-the-android-update-alliance/

Canvas. 2013. Android Developersin verkkosivusto. Viitattu 16.4.2013.
http://developer.android.com/reference/android/graphics/Canvas.html

Canvas and Drawables. n.d. Android Developersin verkkosivusto. Viitattu 24.3.2013.
http://developer.android.com/guide/topics/graphics/2d-graphics.html

DevBytes: Window Animations. 2013. Video Android Developersin YouTube-kanavalla. Viitattu 29.4.2013.
http://www.youtube.com/watch?v=Ho8vk61lVIU

Developer registration. 2013. Google Playn Android -kehittäjäsivut 6.2.2013. Viitattu 8.3.2013.
http://support.google.com/googleplay/android-developer/answer/113468?hl=en&ref_topic=2365624

Doss, S., Lo, S., Pai, C. 2012. Using the Scalable Vector Graphics Library to Render Graphics on Android for Intel IA. Artikkeli Intel Softwaren verkkosivuilla 17.5.2012. Viitattu 25.3.2013.
http://software.intel.com/en-us/articles/using-the-scalable-vector-graphics-library-to-render-graphics-on-android-for-intel-ia

Drawable Animation. n.d. Android Developersin verkkosivusto. Viitattu 19.4.2013.
http://developer.android.com/guide/topics/graphics/drawable-animation.html

George, J. 2011. GIF, JPG and PNG – What's the Difference? Artikkeli Sitepointin verkkosivuilla 1.5.2011. Viitattu 23.3.2013.
http://www.sitepoint.com/gif-jpg-png-whats-difference/

Hardware Acceleration. n.d. Android Developersin verkkosivusto. Viitattu 27.3.2013.
http://developer.android.com/guide/topics/graphics/hardware-accel.html

Keränen, V., Lamberg, N., Penttinen, J. 2005. Digitaalinen media. 1. p. Porvoo: WS Bookwell.

Kuuranta, M. 2011. Animaatioiden vaikutus pelien käytettävyyteen. Pro gradu – tutkielma. Oulun yliopisto, Tietojenkäsittelytieteiden laitos. Viitattu 31.3.2013.
http://www.heiolenmarkus.com/pages/research/effect-of-animation-on-game-usability-markus-kuuranta-progradu.pdf

Luukkonen, J. 1996. Viestinnäntekijän multimediaopas. Porvoo: WSOY.

NineOldAndroids. 2012. NineOldAndroids –animointikirjaston web-sivut. Viitattu 24.4.2013.
http://nineoldandroids.com/

NinePatch. n.d. Android Developersin verkkosivusto. Viitattu 23.3.2013.
http://developer.android.com/reference/android/graphics/NinePatch.html

Platform Versions. 2013. Androidin kehityssivusto. Viitattu 6.3.2013.
http://developer.android.com/about/dashboards/index.html

Property Animation. n.d. Android Developersin verkkosivusto. Viitattu 23.4.2013.
http://developer.android.com/guide/topics/graphics/prop-animation.html

Replica Island Media. n.d. Replica Island –Android-pelin verkkosivut. Viitattu 17.4.2013.
http://replicaisland.net/index.php?view=en/screenshots.php

Samsung phones. n.d.Laitetiedot kaikista Samsungin puhelimista GSMArenan verkkosivuilla. Viitattu 1.5.2013.
http://www.gsmarena.com/samsung-phones-9.php

Supporting Multiple Screens. n.d. Android Developersin verkkosivusto. Viitattu 23.3.2013.
http://developer.android.com/guide/practices/screens_support.html

Syed, H. 2012. The History of Android: From Cupcakes to Jelly Beans. Droid lessons -verkkosivusto 29.9.2012. Viitattu 22.3.2013.
http://droidlessons.com/the-history-of-android-from-cupcakes-to-jelly-beans/

The many faces of a little green robot. 2012. Open Signalin verkkosivut. Viitattu 8.3.2013.
http://opensignal.com/reports/fragmentation.php

Transaction Fees. 2012. Google Playn Android –kehittäjäsivut 13.12.2012. Viitattu 6.2.2013.
http://support.google.com/googleplay/android-developer/answer/112622?hl=en&ref_topic=2897388

Vector Graphics Support for Android! n.d. Svg-android –kirjaston kehityssivusto. Viitattu 25.3.2013.
http://code.google.com/p/svg-android/, Tutorial

Velazco, C. 2012. Goodbye Android Market, Hello Google Play. Artikkeli Techcrunchin verkkosivustolla 6.3.2012. Viitattu 8.3.2013.
http://techcrunch.com/2012/03/06/goodbye-android-market-hello-google-play/

ViewFlipper. n.d. Android Developersin verkkosivusto. Viitattu 25.4.2013.
http://developer.android.com/reference/android/widget/ViewFlipper.html

Vogel, L. 2013. Android (Homescreen) Widgets – Tutorial. Viitattu 14.4.2013.
http://www.vogella.com/articles/AndroidWidgets/article.html

Webp – A new image format for the Web. 2012. Google Developersin verkkosivut 6.11.2012. Viitattu 13.3.2013.
https://developers.google.com/speed/webp/

What are live wallpapers? n.d. Live wallpapers.org –verkkosivut. Viitattu 13.4.2013.
http://www.livewallpapers.org/faq/

What would it take to build a better mobile phone? n.d. Open Handset Alliancen verkkosivut. Viitattu 8.3.2013.
http://www.openhandsetalliance.com/index.html

LIITTEET

Liite 1: Droid Flakes –testitulokset

Testattu laite	Androidin versio
Samsung Galaxy Gio	2.3.5

Animoituja hiutaleita	Frames per second	Kipuraja
8	90	
16	90	
32	80	
64	58	
128	34	
256	19	X
512	10	

Testattu laite	Androidin versio
Samsung Galaxy S3	4.2.2 Cyanogen Mod 10.1

Animoituja hiutaleita	Frames per second	Laitteistokiihdytys päällä / frames per second	Kipuraja
8	60	60	
16	60	60	
32	60	60	
64	60	60	
128	60	60	
256	30	60	
512	20	60	X

Testattu laite	Androidin versio	
HTC Desire	2.3.3	
Animoituja hiutaleita	**Frames per second**	**Kipuraja**
8	60	
16	60	
32	60	
64	60	
128	50	
256	34	
512	20	X

Testattu laite	Androidin versio		
Samsung Galaxy S2	4.1.2		
Animoituja hiutaleita	**Frames per second**	**Laitteistokiihdytys päällä / frames per second**	**Kipuraja**
8	60	60	
16	60	60	
32	60	60	
64	60	60	
128	60	60	
256	30	30	
512	20	30	X

Testattu laite	Androidin versio	
Samsung Mini 2	2.3.6	
Animoituja hiutaleita	**Frames per second**	**Kipuraja**
8	64	
16	64	
32	64	
64	64	
128	53	
256	30	
512	16	X

Liite 2: Frame- ja tween-animaatio -testitulokset

Frame-animaatio

Testattu laite	Androidin versio
Samsung Galaxy Gio	2.3.5
RAM / MB	**CPU-käyttö / %**
8,44	0,25

Testattu laite	Androidin versio
Samsung Galaxy S2	4.1.2
RAM / MB	**CPU-käyttö / %**
17.61	1,38

Testattu laite	Androidin versio
Samsung Mini 2	2.3.6
RAM / MB	**CPU-käyttö / %**
8,28	0,66

Testattu laite	Androidin versio
Samsung Galaxy S3	4.2.2 Cyanogen Mod 10.1
RAM / MB	**CPU-käyttö / %**
45.0	1,00

Huom. Testituloksia ei saatu ylös HTC Desire –testipuhelimesta.

Tween-animaatio

Testattu laite	**Androidin versio**	
Samsung Galaxy Gio	2.3.5	
Animoituja osia	**RAM / MB**	**CPU-käyttö / %**
2	4,84	4,17
3	5,25	7,39
5	5,46	~12,0

Testattu laite	**Androidin versio**	
Samsung Galaxy S3	4.2.2	
Animoituja osia	**RAM / MB**	**CPU-käyttö / %**
2	11,17	1,09
3	11,17	2,98
5	11,30	3,80

Testattu laite	**Androidin versio**	
Samsung Galaxy S2	4.1.2	
Animoituja osia	**RAM / MB**	**CPU-käyttö / %**
2	8,30	3,43
3	8,87	5,5
5	8,59	6,3

Huom. Testituloksia ei mitattu HTC Desire –ja Galaxy Mini 2 –testipuhelimista.