TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

TOM VÄHÄ-SALO

KOLMIULOTTEISTEN PARAMETROITUJEN KÄYRIEN SUUNNITTELUTYÖKALUN TOTEUTTAMINEN

Diplomityö

Tarkastaja: professori Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston kokouksessa
6. helmikuuta 2013

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
VÄHÄ-SALO, TOM: Kolmiulotteisten parametroitujen käyrien suunnittelutyökalun toteuttaminen
Diplomityö, 46 sivua
Huhtikuu 2013
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen
Avainsanat: Parametroidut käyrät, 3D-grafiikka, OpenGL, Qt

Parametroidut käyrät ovat nykyään oleellinen osa graafista suunnittelua ja grafiikan tuottamista, vaikka yhteyttä ei aina ole helppo havaita. Tällaiset käyrät mahdollistavat pehmeästi kaartuvien muotojen tuottamisen käyrän hallittavuudesta tinkimättä. Parametroitujen käyrien avulla on myös mahdollista muodostaa pintoja, mikä mahdollistaa vapaasti muokattavien kolmiulotteisten kappaleiden tuottamisen. Käyrien historia juontaa juurensa 1950- ja 1960-lukujen taitteeseen, jolloin Bézier-käyrä kehitettiin. Bézier-käyrä on eräs tunnetuimmista parametroiduista käyristä.

Käyrien hyödyntäminen ei ole yksinkertaista. Ilman asianmukaista työkalua on vaikea hahmottaa, millaiseksi lopullinen käyrä muodostuu. Lisäksi käyrän laskeminen käsin on hidasta, koska käyrän laskeminen vaatii monivaiheisia matemaattisia operaatioita.

Tässä diplomityössä tarkastellaan kolmiulotteisia parametroituja käyriä, ja niiden suunnitteluun tarkoitetun sovelluksen toteuttamista. Sovellus helpottaa käyrien suunnittelua ja mahdollistaa käyrien tallentamisen myöhempää käyttöä varten. Sovellukselle asetettiin kolme päätavoitetta; sovellus mahdollistaa uusien käyrien suunnittelun, sovelluksella on mahdollista muokata olemassa olevia käyriä ja käyrä on mahdollista tallentaa jatkokäsittelyn mahdollistavaan muotoon.

Sovellus toteutettiin käyttäen Qt-sovelluskehystä, joka mahdollistaa alustariippumattoman sovelluskehityksen C++-ohjelmointikielellä. 3D-grafiikan tuottamisessa hyödynnettiin OpenGL:ää. OpenGL on avoin grafiikkarajapinta 2D- ja 3D-grafiikan piirtämiseen. Sovelluksen toteuttamiseen liittyvistä ratkaisuista merkittävin on liitännäisien käyttäminen. Liitännäiset mahdollistavat sovelluksen joustavan laajentamisen. Kaikki sovelluksessa käytetyt käyrätyypit toteutetaan liitännäisinä, jolloin uuden käyrätyypin lisääminen ei vaadi sovelluskoodin muokkaamista.

Toteutettu sovellus täyttää kaikki sille asetetut tavoitteet. Kaikkia ratkaisuja ei kuitenkaan voitu toteuttaa täydellisesti tämän työn rajoissa. Erityisesti sovelluksen käytettävyyden osalta jatkokehitys on suotavaa. Käyrän laskeminen tietokoneen suorittimella ei aiheuta huomattavaa viivettä normaalitilanteessa, mutta laskenta saattaisi nopeutua tietokoneen grafiikkaohjainta hyödyntämällä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
VÄHÄ-SALO, TOM: Implementation of a tool for designing three-dimensional parametric curves
Master of Science Thesis, 46 pages
April 2013
Major: Software Engineering
Examiner: Professor Tommi Mikkonen
Keywords: Parametric curves, 3D graphics, OpenGL, Qt

Parametric curves are an essential part of graphics and graphical design even if the relation is not always easy to see. The main benefit of these curves is that they provide a way to create smooth shapes without sacrificing the controllability. It is also possible to form surfaces with the help of parametric curves, which creates an opportunity to produce freely adjustable three-dimensional objects. The history of parametric curves dates back to the time between 1950s and 1960s when the Bézier curve was developed. The Bézier curve is probably one of the best known parametric curves.

Utilization of parametric curves is not straightforward. It is quite difficult to foresee a result of a curve calculation process without an appropriate tool that visualizes the result. Also, calculating a curve by hand is slow because several mathematical operations are required.

In this thesis we examine an implementation of a tool, which enables the design of three-dimensional parametric curves. The aim was to create a tool, which simplifies the design process and allows to save the curve for later use. Three goals were set; new curves can be designed, existing curves can be opened for editing and the curve can be stored in a way that it can be used also for other purposes.

The tool was implemented using the Qt application framework, which provides a way to develop cross-platform applications using C++ programming language. 3D graphics were implemented using OpenGL, an open graphics library usually utilized for rendering 3D graphics. The most significant architectural decision in the tool's development process was the decision to use a plugin framework. Plugins provide a flexible way to add new functionality. All the curve types in the application were implemented as plugins. The plugin architecture provides a way to add new curves without touching the tool's source code.

The tool meets all the goals that were set. However, it was not possible to implement a perfect solution for every problem in the thesis' time frame. Further development is desirable especially when considering the current usability aspects. Also, it might be beneficial to use a graphics processing unit (GPU) for the curve calculation.

ALKUSANAT

Ajatus tämän työn aiheeseen syntyi jo kesällä 2012, kun etsin parametroitujen käyrien suunnitteluun tarkoitettua sovellusta, joka mahdollistaisi käyrän kontrollipisteiden tallentamisen. Jokseenkin tuloksettoman etsinnän jälkeen päädyin leikittelemään ajatuksella oman sovelluksen toteuttamisesta. Kesti kuitenkin syksyyn 2012 ennen kuin ryhdyin miettimään aiheen soveltuvuutta diplomityöksi. Erinäisten kokeilujen ja onnistuneen prototyypin perusteella totesin sovelluksen olevan toteutettavissa. Virallinen diplomityöaihe muodostui, kun keskustelin asiasta professori Tommi Mikkosen kanssa.

Työn toteuttaminen tapahtui suhteellisen nopealla aikataululla, koska aihe oli motivoiva ja työn aikana ei esiintynyt ylitsepääsemättömiä ongelmia. Diplomityöprosessin suurimmat haasteet ajoittuivat työn alkupuolelle. Tällöin oli muun muassa löydettävä sopiva rytmi ohjelmointityön ja kirjoitustyön välillä.

Kiitän työn tarkastajana toiminutta professori Tommi Mikkosta asiantuntevista neuvoista, opastuksesta sekä työn kieliasun parantamisesta. Kiitän myös Antti Kolehmaista englanninkielisen tiivistelmän oikoluvusta. Lopuksi esitän kiitokseni perheelleni ja ystävilleni koko opiskeluajan kestäneestä tuesta ja kannustuksesta.

Tampereella 21.2.2013

Tom Vähä-Salo

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

3D	Three dimensional. Kolmiulotteinen. Mielivaltainen kappale kuvataan kolmen ulottuvuuden, leveyden, korkeuden ja syvyyden, avulla.
3D-grafiikka	Kokoelma tekniikoita ja apuvälineitä, joilla kolmiulotteinen maailma voidaan esittää kaksiulotteisella pinnalla.
Bézier-käyrä	Pierre Bézierin mukaan nimetty käyrä, joka on eräs tunnetuimmista parametroiduista käyristä. Kehitettiin 1950- ja 1960-lukujen taitteessa autoteollisuuden tarpeisiin.
C++11	ISO:n vuonna 2011 hyväksymä standardi C++-ohjelmointikielelle.
Catmull-Rom-spline	Vastaavan tyyppinen käyrä kuin uniformi B-spline. Catmull-Rom-splinen selkein etu on, että se kulkee kontrollipisteidensä kautta.
Chaikinin käyrä	Geometrisella algoritmilla muodostettu käyrä. Käyrä muodostetaan jakamalla kontrollimonikulmiota pienempiin osiin, kunnes haluttu tarkkuus on saavutettu.
Compute Shader	Laskentaan tarkoitettu sävytin, joka suoritetaan tietokoneen grafiikkaohjaimella. Ei ole osa 3D-piirron liukuhihnaa, vaan suoritetaan täysin erillisenä vaiheena.
CSS	Cascading Style Sheets. Erityisesti web-ohjelmointiin tarkoitettu määrittelykieli, joka mahdollistaa web-sivujen tyylimäärittelyiden toteuttamisen.
CUDA	Compute Unified Device Architecture. NVIDIA:n hallinnoima tekniikka, joka mahdollistaa rinnakkaisen laskennan NVIDIA:n valmistamilla grafiikkaohjaimilla.
Direct3D	Microsoftin 3D-grafiikan tuottamiseen tarkoitettu rajapinta Windows-käyttöjärjestelmälle.
Geometriasävytin	Grafiikkaohjaimella suoritettava ohjelma, joka mahdollistaa muun muassa uuden geometrian tuottamisen.
GLSL	OpenGL Shading Language. OpenGL:n käyttämä korkean tason kieli sävytinohjelmointiin.
Grafiikkaohjain	Tietokoneeseen kuuluva komponentti, joka tuottaa kuvan tietokoneen näytölle. Modernit grafiikkaohjaimet sisältävät erityisesti 3D-grafiikan tuottamiseen tarvittavaa toiminnallisuutta.
JavaScript	Korkean tason ohjelmointikieli, jota käytetään erityisesti web-pohjaisessa ohjelmoinnissa.
JSON	JavaScript Object Notation. Dokumenttiformaatti, joka mahdollistaa tiedon tallentamisen avain−arvo-parien avulla.

Kirjasto	Kokoelma aliohjelmia, luokkia tai määrittelyitä, joita voidaan hyödyntää ohjelmoinnissa. Voidaan ottaa käyttöön dynaamisesti tai staattisesti.
Kontrollimonikulmio	Control polygon. Viivoilla yhdistettyä kontrollipisteiden joukkoa kutsutaan kontrollimonikulmioksi.
Kontrollipiste	Koordinaatistoavaruudessa sijaitseva piste, jota käytetään parametroidun käyrän muodostamiseen.
Kulmapiste	Vertex. 3D-avaruuden kappaleet muodostetaan kulmapisteistä, joita käytetään kappaleen pinnan muodostamiseen. Kulmapisteet voivat sisältää myös lisäinformaatiota koordinaattipisteen lisäksi.
Käyräpiste	Koordinaatistoavaruuden piste, jonka kautta käyrä kulkee. Käyräpisteet yhdistetään yleensä viivoilla käyrää piirrettäessä.
Käyttöliittymä	Rajapinta, jonka avulla käyttäjän on mahdollista hallita sovelluksen toimintaa.
Käyttöliittymäkomponentti	Osa käyttöliittymää, esimerkiksi painonappi. Tyypillisesti käyttöliittymä koostuu useista eri komponenteista, jotka voivat olla käyttäjän tai pelkästään sovelluksen hallittavissa.
Lambda	Anonyymifunktio. Lambda-funktiolle ei määritellä nimeä, joten sitä ei voida kutsua, kuten normaalia funktiota.
Liitännäinen	Plugin. Lisäosa, joka laajentaa sovelluksen tai kirjaston toiminnallisuutta. Voidaan ottaa käyttöön dynaamisesti tai staattisesti.
Liukuhihna	Pipeline. Piirrettävälle 3D-kappaleelle suoritetaan erilaisia muunnoksia vaihe kerrallaan ikään kuin kappale liikkuisi liukuhihnalla. Erityisesti grafiikkaohjaimet hyödyntävät tätä ajattelutapaa.
Mainostaulu	Billboard. 3D-avaruuden kappale, joka kääntyy katsojan suuntaan, kun katsoja liikkuu.
NURBS	Nonuniform Rational B-spline. Käyrätyyppi, joka tuo lisää ilmaisuvoimaa ei-rationaalisiin käyriin verrattuna. NURBS-käyrän käsittely tapahtuu neljännessä ulottuvuudessa.
OpenCL	Open Computing Language. Khronos Groupin hallinnoima tekniikka. Mahdollistaa rinnakkaisen laskennan erilaisilla alustoilla, kuten grafiikkaohjaimella.
OpenGL	Open Graphics Library. 2D- ja 3D-piirtämiseen tarkoitettu avoin ohjelmointirajapinta.
Parametroitu käyrä	Määritelty polynomifunktio, jonka avulla on mahdollista laskea kaikki käyrän pisteet parametrin *t* suhteen.

Perspektiivi	Mahdollistaa kolmiulotteisen avaruuden esittämisen luonnollisen näköisenä kaksiulotteisella pinnalla. Perspektiivi saa kauimmaiset kappaleet näyttämään pienemmiltä.
Pikselisävytin	Grafiikkaohjaimella suoritettava ohjelma, joka käsittelee kerrallaan yhtä tietokoneen näytölle piirrettävää pikseliä.
Poikkialustainen	Cross-platform. Kokoelma tekniikoita, joiden avulla on mahdollista tuottaa usealla eri alustalla, esimerkiksi käyttöjärjestelmällä, toimivia sovelluksia.
Poiminta	Picking. Tekniikka, jonka avulla 3D-avaruuden kappaleita on mahdollista valita esimerkiksi hiirellä.
Projektio	Tekniikka, jolla kolmiulotteinen avaruus esitetään kaksiulotteisella pinnalla. Yleisin käytetty projektio on perspektiiviprojektio.
QML	Qt Meta Language. Korkean tason käyttöliittymien määrittely- ja ohjelmointikieli, joka pohjautuu JavaScriptiin ja JSON-tyyppiseen syntaksiin.
QSS	Qt Style Sheet. CSS:ään perustuva tyylimäärittelykieli Qt-sovelluksille.
Qt	Digian omistama sovelluskehys C++-ohjelmointikielelle. Mahdollistaa alustariippumattoman sovelluskehityksen sekä käyttöliittymien tuottamisen.
Qt Quick	Qt:n sisältämä moduuli, joka tarjoaa peruselementit QML-pohjaisten käyttöliittymien toteuttamiseen. Qt Quick mahdollistaa myös QML-toteutuksen ja C++-toteutuksen välisen kommunikoinnin.
Qt3D	Qt:n lisäkomponentti, joka helpottaa 3D-grafiikkaa käyttävän sovelluksen toteuttamista.
Rasterointi	3D-piirron liukuhihnan loppupuolella suoritettava vaihe, joka muuntaa piirrettävät kappaleet pikseleiksi.
Rinnakkaisuus	Rinnakkaisuutta hyödyntämällä sovellus voi suorittaa useita operaatioita samanaikaisesti. Rinnakkaisuus voidaan saavuttaa esimerkiksi toisen suorittimen, suoritinytimen tai toisen laitteen avulla.
Signaali–lokero-tekniikka	Signals and slots. Qt:n C++-laajennos, joka mahdollistaa olioiden välisen vuorovaikutuksen ilman, että oliot tietävät toisistaan.
Sovelluskehys	Framework. Kokoelma sovelluskehityksessä käytettyjä työkaluja, kirjastoja ja tekniikoita.
Spline	Joukko *n*-asteisia osakäyriä, jotka on sulavasti liitetty toisiinsa. Osakäyrät muodostavat kokonaiskäyrän.

SSE	Streaming SIMD Extensions. X86-pohjaisten suorittimien käskykantalaajennus, joka mahdollistaa usean tietoalkion käsittelyn yhdellä käskyllä.
Sävytin	Shader. Tietokoneen grafiikkaohjaimella suoritettava ohjelma. Sävytinohjelmointi mahdollistaa muun muassa grafiikkaohjaimen suorittamien liukuhihnavaiheiden mukauttamisen.
Tesselaatio	Modernin grafiikkaohjaimen liukuhihnan vaihe, jossa geometriaa voidaan jakaa pienempiin osiin.
Tesselaatiosävytin	Grafiikkaohjaimen tesselaatiovaiheessa suoritettava ohjelma. Tesselaatiosävyttimiä on kahdenlaisia; kontrollisävyttimiä ja evaluointisävyttimiä.
Uniformi B-spline	B-spline määritellään painottamalla kanta-spline-funktioita kontrollipisteillä. Uniformi tarkoittaa, että käytetyn parametrin *t* välit ovat vakioita. Yleensä uniformi B-spline muodostetaan kuutiollisista osakäyristä, jolloin kyseessä on uniformi kuutiollinen B-spline.
Verteksisävytin	Grafiikkaohjaimella suoritettava ohjelma, joka käsittelee yhtä verteksiä eli kulmapistettä kerrallaan.
WebKit	Selainmoottori, joka mahdollistaa web-sivujen esittämisen käyttäjäystävällisessä muodossa. Monet WWW-selaimet perustuvat WebKit-moottoriin.
XML	Extensible Markup Language. Sovellusriippumaton merkintäkieli hyvin jäsennettyjen dokumenttien tuottamiseen ja tiedon tallentamiseen.

1 JOHDANTO

Pyörä on yksi ihmiskunnan historian merkittävimmistä ja käytetyimmistä keksinnöistä. Pyörän merkittävin ominaisuus on sen pyöreys, johon myös esineen nimi viittaa. Jokapäiväisissä käyttöesineissä pyöreät muodot ovat miellyttäviä katsella ja käsitellä. Pienten lasten leikkikaluissa suositaan pyöreitä muotoja. Teollisuudessa pyöreät muodot ovat mahdollistaneet muun muassa entistä energiatehokkaampien autojen tuottamisen, koska pyöreät muodot auttavat vähentämään ilmanvastusta. Pyöreitä muotoja voidaan tuottaa tarkasti tai epätarkasti. Epätarkasti tuottamalla lopputulos on todennäköisesti jokin approksimaatio halutusta lopputuloksesta. Teollisuudessa pyöreitä muotoja on tuotettava matemaattisen tarkasti. Tarkkojen muotojen tuottaminen vaatii matematiikan, suunnittelun ja tehtävään soveltuvien työkalujen yhteensovittamista.

Parametroitujen käyrien avulla on mahdollista tuottaa tarkkoja ja hallittavia pyöreitä muotoja. Ongelmana on, että käyrän suunnittelu ilman tarkoituksenmukaista työkalua on työlästä ja lopullisen käyrän muotoa voi olla vaikea hahmottaa. Lisäksi olemassa olevaa käyrää saattaa olla tarve muokata. Jo pelkästään näistä syistä on perusteltua toteuttaa työkalu työn helpottamiseksi.

Parametroitujen käyrien käsite on laaja. Yksinkertaisimmillaan parametroidulla käyrällä voidaan tarkoittaa esimerkiksi lukiomatematiikasta tuttua paraabelia. Reaalimaailman sovelluksissa ei yleensä voida käyttää näin yksinkertaista lähestymistapaa, koska kappaleen muoto ei välttämättä noudata mitään tunnettua funktiota. Tällöin käyrän muodostamiseen on käytettävä toisenlaista lähestymistapaa. Parametroitujen käyrien historia juontaa juurensa 1950- ja 1960-lukujen taitteeseen. Tällöin huomattiin, että parametroituja käyriä voidaan tuottaa erityisten kontrollipisteiden avulla. Kontrollipisteet mahdollistavat muun muassa käyrän tehokkaamman käsittelyn, ja yksityiskohtaisemman hallinnan. Vuosien saatossa on suunniteltu useita erilaisia käyrätyyppejä, joilla on erilaisia ominaisuuksia. Tyypillisesti tietyt käyrätyypit soveltuvat tiettyyn tarkoitukseen paremmin kuin toiset käyrätyypit. Nykyään kontrollipisteiden avulla muodostettavia käyriä hyödynnetään monilla eri osa-alueilla aina teollisuudesta 3D-grafiikkaan. 3D-grafiikassa eräs tyypillinen käyttökohde on liikeratojen määrittäminen.

Tämän diplomityön tarkoituksena on perehtyä kolmiulotteisten parametroitujen käyrien suunnitteluun tarkoitetun sovelluksen toteuttamiseen. Käyrän suunnittelu esimerkiksi paperilla on työlästä, koska käyrän laskennassa tarvitaan monivaiheisia laskutoimituksia ja käyrä saattaa koostua tuhansista pisteistä. Sovelluksen ansiosta käyttäjä voi keskittyä käyrän suunnitteluun kontrollipisteitä käsittelemällä, ja jättää matemaattisen laskennan tietokoneen suoritettavaksi. Käyttäjällä on mahdollisuus luoda uusia käyriä sekä muokata aiemmin toteutettuja käyriä. Sovelluksen ominaisuuksia voidaan laajentaa

joustavasti, ja esimerkiksi uuden käyrätyypin lisääminen on mahdollista ilman sovelluskoodin muokkaamista.

Työn rakenne on seuraavanlainen. Luvussa 2 tarkastellaan parametroituja käyriä, niiden muodostumista, ja ominaisuuksia. Lisäksi luvussa 2 esitellään muutamia työssä käytettyä käyrätyyppejä. Luvussa 3 perehdytään 3D-grafiikkaan, ja sen perusperiaatteisiin. Tässä luvussa tarkastellaan myös OpenGL:ää (Open Graphics Library) 3D-grafiikan toteutusvälineenä. OpenGL mahdollistaa 3D-grafiikan tuottamisen tietokoneen grafiikkaohjainta hyödyntäen. Luvussa 4 esitellään Qt-sovelluskehys, johon koko sovelluksen tekninen toteutus perustuu. Sovelluksen toteuttamiseen liittyviin seikkoihin perehdytään luvussa 5. Luvussa 6 tarkastellaan toteutuneita ominaisuuksia ja arvioidaan ratkaisujen onnistuneisuutta. Lopuksi luvussa 7 suoritetaan yhteenveto koko työn osalta.

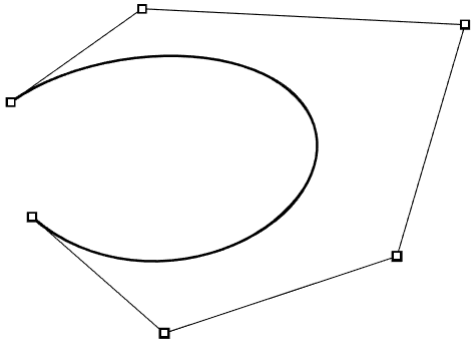
2 PARAMETROIDUT KÄYRÄT

Tässä luvussa tarkastellaan parametroituja käyriä, käyrän muodostumisen yleisiä periaatteita sekä joitain tyypillisiä käyrien käyttökohteita. Luvussa käsitellään myös käyrän jatkuvuuden käsitettä, kun käyrä muodostetaan useasta pienemmästä palasta. Lopuksi esitellään tässä työssä käytettyjä käyrätyyppejä.

2.1 Käyrän muodostuminen ja käyrien käyttötarkoitus

Parametroiduilla käyrillä on suhteellisen pitkä historia. Eräs tunnetuimmista parametroiduista käyristä, Bézier-käyrä, kehitettiin Ranskassa jo 1950- ja 1960-lukujen taitteessa autoteollisuuden tarpeisiin (Puhakka 2008, s. 61). Parametroidulla käyrällä tarkoitetaan määriteltyä polynomifunktiota, jonka avulla on mahdollista laskea kaikki käyrän pisteet parametrin t suhteen (Puhakka 2008, s. 61). Kaksiulotteisessa avaruudessa parametroidun käyrän piste $\mathbf{p}(t)$ noudattaa muotoa $\mathbf{p}(t) = (x(t), y(t))$, jossa funktiot x ja y tuottavat käyrän pisteen x- ja y-koordinaatit parametrin t suhteen. Parametri t kulkee ennalta määritellyn välin $[a, b]$, joka on yleensä [0, 1], jolloin $\mathbf{p}(0)$ on käyrän alkupiste ja $\mathbf{p}(1)$ on käyrän loppupiste. Käytännön sovelluksissa joudutaan tyypillisesti käyttämään monimutkaisia käyriä mallintamaan esimerkiksi pinnan kaarevuutta, ja tällöin käyrän funktiota ei yleensä tiedetä. (Salomon 2006, s. 11.) Usein parametroitujen käyrien tapauksessa käytetään käyrälle ennalta määriteltyjä kantapolynomeja, joiden avulla käyrän pisteen koordinaatit saadaan laskettua, kun kantapolynomeja painotetaan määriteltyjen kontrollipisteiden koordinaateilla (Puhakka 2008, s. 62).

Käyrän kontrollipisteet ovat joukko pisteitä käytetyssä koordinaatistossa. Usein käyrä ei kulje kontrollipisteidensä kautta, vaan käyrä on lähinnä kontrollipisteidensä approksimaatio (Salomon 2006, s. 11). Kuvassa 2.1 on esitetty eräs parametroitu käyrä kontrollipisteineen kaksiulotteisessa avaruudessa. Jos kontrollipisteet yhdistetään viivalla, kuten kuvassa 2.1, tätä viivoilla yhdistettyjen pisteiden joukkoa kutsutaan kontrollimonikulmioksi (engl. control polygon) (Puhakka 2008, s. 63).



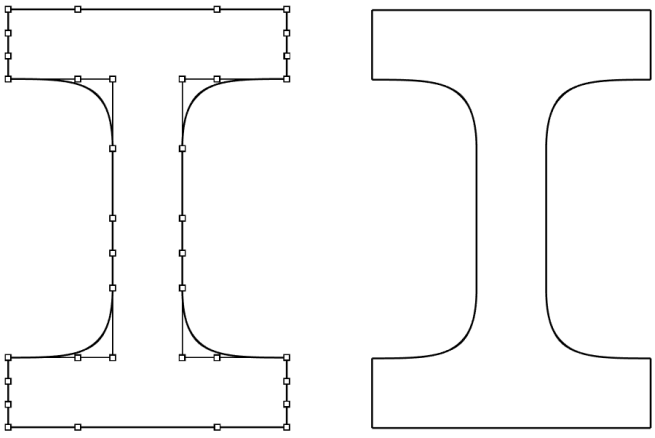
Kuva 2.1. *Parametroitu käyrä ja käyrän kontrollipisteet.*

Kontrollipisteiden käytöllä on useita etuja. Ensinnäkin kaikki käyrän pisteet saadaan laskettua käyttämällä tiettyjä kantapolynomeja ja ainoastaan painottamalla näitä kontrollipisteillä, kuten jo aiemmin mainittiin. Eräs merkittävä etu on myös, että kaikki käyrään kohdistettavat muunnokset (siirto, kierto tai skaalaus) voidaan suorittaa ainoastaan kontrollipisteille, ja käyrä mukautuu näiden muunnosten mukaiseksi, kun käyrä lopulta lasketaan (Puhakka 2008, s. 62-63, 72). Lisäksi kontrollipisteitä on yleensä huomattavasti vähemmän kuin varsinaisia käyrän pisteitä, joten käyrän muokkaaminen ja esimerkiksi tallentaminen on yleensä tehokkaampaa pelkkiä kontrollipisteitä käsittelemällä. Kontrollipisteiden avulla voidaan muodostaa käyriä myös ilman polynomeja, jolloin käyrä muodostetaan kontrollipisteistä tekemällä esimerkiksi rekursiivista alijakoa (Puhakka 2008, s. 73).

Valmiin käyrän muoto ja ominaisuudet vaihtelevat käytetyn käyrätyypin mukaan, mutta joillain käyrätyypeillä on myös yhteneväisiä ominaisuuksia, joita voi havaita esimerkiksi kohdassa 2.3 esiteltävistä käyristä. Erilaisten ominaisuuksiensa ansiosta, tietyn ominaisuuden omaavat käyrät soveltuvat tyypillisesti parhaiten yhteen käyttötarkoitukseen ja jonkin toisen ominaisuuden omaavat käyrät soveltuvat paremmin toiseen. (Puhakka 2008, s. 61-73.) Toisaalta sopivan käyrätyypin löytyminen voi olla hyvin tilannekohtaista ja lisäksi tehtävään soveltuvia käyrätyyppejä voi olla useita.

Parametroiduille käyrille on useita käyttökohteita. Teollisuudessa, erityisesti autoteollisuudessa, havaittiin ensimmäisten joukossa tarve muodostaa vapaasti muokattavia kaarevia viivoja (Puhakka 2008, s. 61). Tämä johti siihen, että pystyttiin muodostamaan vapaasti muokattavia kaarevia pintoja, jotka muodostuvat äärettömästä joukosta parametroituja käyriä (Puhakka 2008, s. 74). Parametroituja pintoja ei käsitellä tarkemmin tämän työn rajoissa, mutta ne voidaan siitä huolimatta nähdä yhtenä parametroitujen käyrien käyttökohteena.

Nykyään parametroituja käyriä käytetään paljon grafiikan ja graafisen suunnittelun sovelluksissa. Yhtenä käyttökohteena voidaan pitää kirjasimien ja kirjasintyyppien suunnittelua, kuten kuvassa 2.2 on havainnollistettu (Farin 2002, s. 71). Kuvassa vasemmanpuoleisessa kirjasimessa on havainnollistettu myös käytetyt kontrollipisteet ja kontrollimonikulmio. Kontrollipisteistä laskettu käyrä muodostaa kirjasimen ääriviivan.



Kuva 2.2. *Kirjasin esitettynä sarjana parametroituja käyriä.*

Käyriä hyödyntäviin sovelluskohteisiin kuuluu myös SVG-tiedostoformaatti (Puhakka 2008, s. 64). 3D-grafiikassa parametroituja käyriä voidaan hyödyntää muun muassa kameran liikeratojen määrittelyyn (Puhakka 2008, s. 73). Viimekädessä on kuitenkin täysin mielikuvituksesta kiinni, mihin kaikkeen käyriä voidaan hyödyntää. Esimerkiksi visuaalisissa efekteissä voidaan saada mielenkiintoisia tuloksia parametroituja käyriä hyödyntämällä.

2.2 Käyrän jatkuvuus

Parametroitujen käyrien yksi oleellinen seikka on käyrän aste. Tällä tarkoitetaan sitä, minkä asteisia polynomeja käyrän muodostamisessa käytetään (Puhakka 2008, s. 61-62). Näin ollen siis esimerkiksi kolmannen asteen, eli kuutiollisen, käyrän tapauksessa käytetään kolmannen asteen polynomeja. Käyrän astetta kuvataan kirjaimella n ja n-asteisen käyrän laskemiseen tarvitaan n+1 kappaletta kontrollipisteitä, eli kuutiollisen käyrän tapauksessa tarvitaan neljä kontrollipistettä (Puhakka 2008, s. 62-64). Käytännössä yli kolmannen asteen käyriä ei kuitenkaan yleensä käytetä, koska näitä on usein vaikea hallita korkean asteen polynomeista johtuen (Salomon 2006, s. 18). Toisaalta esimerkiksi n-asteinen Bézier-käyrä voidaan muodostaa ilman kantapolynomien evaluoimista (Puhakka 2008, s. 64).

Korkeamman asteen polynomien käyttämisen sijaan monimutkaisia käyriä voidaan muodostaa liittämällä yhteen useita matalamman asteen käyriä (Puhakka 2008, s. 65). Tällaista palasista muodostettua käyrää kutsutaan usein spline-käyräksi tai vain yksinkertaisesti splineksi (Salomon 2006, s. 19). Spline on määritelmänsä mukaisesti joukko n-asteisia polynomeja, jotka on sulavasti liitetty toisiinsa tietyistä pisteistä (Salomon 2006, s. 141). Määritelmässä mainittu sulava liitos asettaa vaatimuksia sille, miten kokonaiskäyrän tulee käyttäytyä osakäyrien liitoskohdissa.

Liitoskohdissa tapahtuvaa käyttäytymistä kutsutaan käyrän jatkuvuudeksi, ja tälle on olemassa eri asteita. Jatkuvuudella on merkitystä, kun kokonaiskäyrän halutaan käyttäytyvän tietyllä tavalla osakäyrien liitoskohdissa ja nämä vaatimukset voivat vaikuttaa käyrätyypin valintaan. Muodon jatkuvuutta kuvaa G^i-jatkuvuus, jonka perustapaukset koostuvat G^0- ja G^1-jatkuvuuksista. Näistä G^0-jatkuvuus tarkoittaa, että osakäyrät kohtaavat yhdistymispisteessä, ja G^1-jatkuvuus sitä, että käyrän tangentit ovat yhteneväiset yhdistymiskohdan molemmin puolin. Yleensä G^0-jatkuvuus toteutuu aina, koska muutoin kokonaiskäyrä ei ole yhtenäinen. (Puhakka 2008, s. 65.)

G^i-jatkuvuuden lisäksi on myös C^i-jatkuvuus, joka asettaa tiukempia vaatimuksia jatkuvuudelle. C^i-jatkuvuus määritellään käyrän funktion $\mathbf{p}(t)$ jatkuvuutena käytetyn parametrin t suhteen ja koostuu jatkuvuuksista C^0, C^1 ja C^2. Näistä C^0-jatkuvuus on vastaava kuin G^0-jatkuvuus, mutta sen sijaan C^1-jatkuvuus asettaa merkittävämpiä vaatimuksia kuin G^1-jatkuvuus. C^1-jatkuvuus lisää G^1-jatkuvuuden vaatimukseen sen, että käyrän tangentin pitää olla liitoksen molemmin puolin myös yhtä suuri. C^2-jatkuvuus asettaa vielä vaatimuksen, että käyrän toisen parametrisen derivaatan pitää olla jatkuva. C^1-jatkuvuudessa siis vastaava vaatimus kohdistuu käyrän ensimmäiseen parametriseen

derivaattaan. C^1- ja C^2-jatkuvuuksilla saadaan käyrään tuotua yhä enemmän pyöreyttä. (Puhakka 2008, s. 65.)

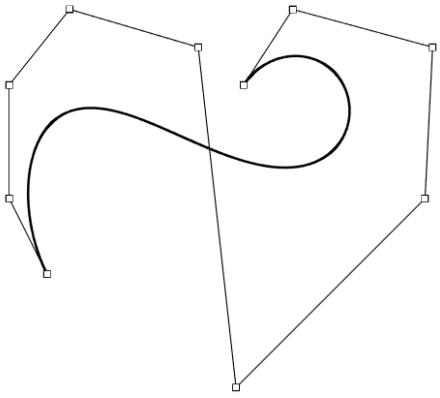
2.3 Työssä käytettyjä käyrätyyppejä

Erilaisia käyriä ja käyrätyyppejä on olemassa useita ja erilaisilla käyrillä on tyypillisesti myös erilaisia ominaisuuksia. Näin ollen yksi käyrätyyppi saattaa soveltua paremmin yhteen käyttötarkoitukseen ja toinen käyrätyppi toisenlaiseen tarkoitukseen, mutta toisilla käyrätyypeillä on havaittavissa myös yhteneväisiä ominaisuuksia. Seuraavissa alakohdissa tarkastellaan muutamaa tässä työssä hyödynnettyä käyrää, jotka on valittu lähinnä yleisyyden ja yksinkertaisen muodostumisen perusteella.

2.3.1 Bézier-käyrä

Bézier-käyrää voidaan pitää yhtenä tunnetuimmista käyrätyypeistä. Se on saanut nimensä ranskalaisen matemaatikon Pierre Bézierin mukaan, joka kehitti käyrän työskennellessään Renaultilla 1960-luvulla. Merkittävää kuitenkin on, että eräs toinen ranskalainen matemaatikko, Paul de Casteljau, onnistui kehittämään oman menetelmänsä jo vuonna 1959 työskennellessään Citroënille, mutta tuloksia ei koskaan tuotu julkisesti esiin vaitiolovelvollisuuksista johtuen. Vuonna 1975 löydetyt sisäiset muistiot kuitenkin paljastavat de Casteljaun työn saavutukset, mutta menetelmä oli jo ehditty nimeämään Bézierin mukaan. (Salomon 2006, s. 175.) Joka tapauksessa autoteollisuuden roolia ei voida vähätellä parametroitujen käyrien historiassa.

Koska Bézier-käyrä ei yleisesti ottaen kulje kontrollipisteidensä kautta muuten kuin käyrän alku- ja loppupisteessä, voidaan käyrää pitää approksimoivana käyränä. Kontrollipisteet kuitenkin vaikuttavat käyrän suuntaan vetämällä käyrää puoleensa. Mitä lähempää käyrä kulkee kontrollipistettä, sitä voimakkaammin kontrollipiste vaikuttaa käyrään. (Salomon 2006, s. 176.) Näin ollen kontrollipisteiden lisääminen, poistaminen tai muokkaaminen vaikuttaa suoraan käyrän muotoon. Kuvassa 2.3 on esitetty eräs Bézier-käyrä.



Kuva 2.3. *Bézier-käyrä.*

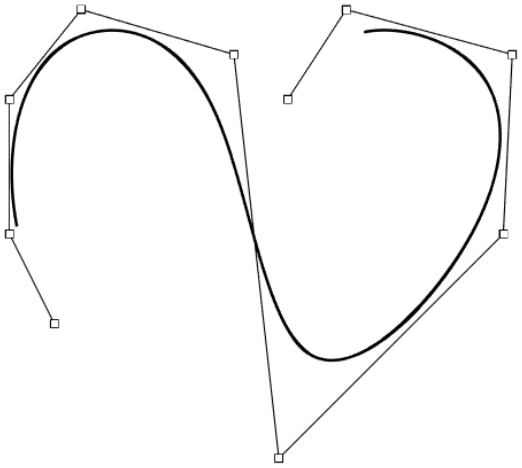
Bézier-käyrän aste määräytyy käytettyjen kontrollipisteiden mukaan, kuten aiemmin kuvattiin (Salomon 2006, s. 176). Yksi tapa *n*-ulotteisen Bézier-käyrän laskentaan on

käyttää kantapolynomeina niin sanottuja Bernsteinin polynomeja (Puhakka 2008, s. 63). Toisaalta samaan tulokseen päästään myös käyttämällä de Castejaun algoritmia, joka on vaihtoehtoinen tapa *n*-asteisen Bézier-käyrän määrittämiseen. Tämän menetelmän etu on lisäksi se, että käyrä voidaan muodostaa ilman kantapolynomeja. (Puhakka 2008, s. 64.) Bernsteinin polynomeja käytettäessä ongelmaksi voi muodostua se, että laskennassa käytetään kertomaa (Puhakka 2008, s. 63). Tämä voi olla rajoittava tekijä käytännön sovelluksissa, koska kertoman tulos ei välttämättä mahdu ohjelmointikielen tarjoamiin tietotyyppeihin. Kertomaa on mahdollista approksimoida reaalilukuina käyttäen Stirlingin kaavaa, mutta tällöin lopputuloksessa voi olla pieniä tarkkuuseroja (Weisstein 2013). Kertoman laskeminen on myös suhteellisen raskasta, joten hyvä tapa voi olla tallettaa useimmin tarvitut arvot esimerkiksi taulukkoon (Salomon 2006, s. 185).

Aiemmin mainitulla de Casteljaun algoritmilla voidaan muodostaa *n*-asteinen Bézier-käyrä ilman kantapolynomien hyödyntämistä. Ajatuksena tässä algoritmissa on, että kontrollipisteiden väliltä lineaarisesti interpoloidaan uudet pisteet käyttäen parametria *t*, joka kuuluu välille [0, 1]. Tämän jälkeen sama operaatio toistetaan näille uusille pisteille, kunnes jäljellä on enää yksi piste, joka on haluttu piste käyrällä. Algoritmi on helppo toteuttaa rekursion avulla ja se ei sisällä yhtä raskasta laskentaa kuin Bernsteinin polynomeja käyttävä ratkaisu. (Puhakka 2008, s. 64.)

2.3.2 Uniformi kuutiollinen B-spline-käyrä

Uniformi kuutiollinen B-spline-käyrä on kokonaiskäyrä, joka koostuu vastaavista kuutiollisista osakäyristä. Nämä osakäyrät muodostetaan neljän kontrollipisteen avulla, kuten kohdassa 2.2 mainittiin. Uniformia kuutiollista B-spline-käyrää on havainnollistettu kuvassa 2.4.



Kuva 2.4. *Uniformi kuutiollinen B-spline-käyrä.*

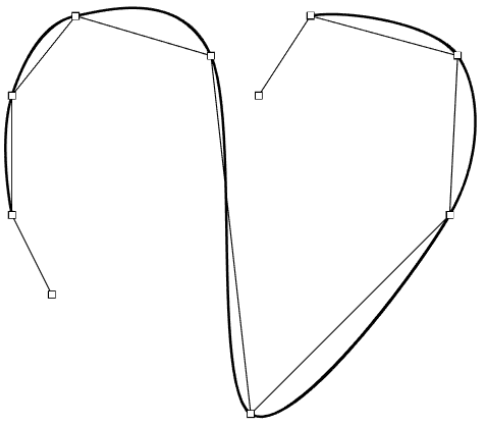
Yleisesti B-spline-käyräksi kutsutaan käyrää, joka määritellään painottamalla kanta-spline-funktioita kontrollipisteillä (Puhakka 2008, s. 66). Jos B-spline-käyrä on uniformi, tällä tarkoitetaan, että kaikki käytetyn parametrin *t* välit ovat vakioita ja kyseistä käyrää voidaan pitää B-spline-käyrän erikoistapauksena (Puhakka 2008, s. 67). Uniformin käyrän muodostamisessa on myös se etu, että käyrä saadaan muodostettua

kertomalla ennalta määriteltyjä vakiopolynomeja kontrollipisteillä, ja näin ollen käyrän muodostamista voidaan pitää helpompana kuin ei-uniformin käyrän (Puhakka 2008, s. 67-68; Spielberger 2010, s. 1). Kääntöpuolena kuitenkin on, että uniformi käyrä ei sovellu yhtä hyvin kaikkiin tarkoituksiin ja lopputulos voi esimerkiksi näyttää luonnottomalta, jos käyrää yritetään sovittaa pistejoukkoon, jossa on paljon hajontaa (Salomon 2006, s. 16).

B-spline-tyyppiset käyrät ovat C^{n-1}-jatkuvia, jossa n tarkoittaa käyrän astetta (Sederberg 2005, s. 1). Näin ollen siis esimerkiksi kuutiollisen käyrän tapauksessa saavutetaan C^2-jatkuvuus, jolloin saadaan muodostettua pyöreästi kaartuvia käyriä. Tätä voidaan pitää B-spline-käyrien etuna Bézier-käyriin nähden (Sederberg 2005, s. 1). B-spline-käyrät eivät yleensä kulje minkään kontrollipisteen kautta, mikä voi muodostua ongelmaksi, jos käyrän halutaan alkavan ensimmäisestä kontrollipisteestä ja päättyvän viimeiseen (Puhakka 2008, s. 71). Toisaalta käyrän eräs etu on siinä, että se pysyy kontrollipisteidensä konveksin peitteen sisällä (Puhakka 2008, s. 69). Konveksilla peitteellä tarkoitetaan pienintä mahdollista pistejoukkoa, joiden muodostama kappale kattaa kaikki tarkasteltavat pisteet ja jonka kaikki sisäkulmat ovat alle 180° (Puhakka 2008, s. 47, 131).

2.3.3 Catmull-Rom-spline-käyrä

Catmull-Rom-spline on vastaavan tyyppinen käyrä kuin uniformi B-spline, mutta näissä käytetään eri polynomeja (Puhakka 2008, s. 72). Catmull-Rom on C^1-jatkuva, ja käyrän muodostamisessa käytetään yleensä kuutiollisia osakäyriä (Twigg 2003, s. 1). Käyrän selkein etu on siinä, että se kulkee kontrollipisteidensä kautta, lukuun ottamatta alku- ja loppupistettä (Puhakka 2008, s. 72). Käyrä saadaan kuitenkin kulkemaan alku- ja loppupisteensä kautta toistamalla nämä pisteet. Koska käyrä kulkee kontrollipisteidensä kautta, on Catmull-Rom suosittu valinta esimerkiksi liikeratojen määrittelyyn 3D-avaruudessa (Yuksel et al. 2011, s. 754). Catmull-Rom:n suosiosta grafiikkaohjelmoinnissa kertoo myös se, että Microsoftin DirectXMath-kirjastossa on valmis toteutus Catmull-Rom-splinen laskemiseen (Geometric Vector Functions 2012). Catmull-Rom-spline-käyrää on havainnollistettu kuvassa 2.5.

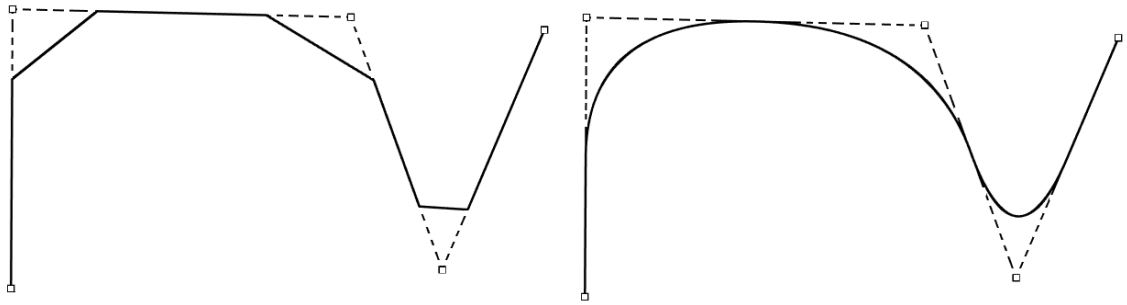


Kuva 2.5. *Kuutiollinen Catmull-Rom-spline-käyrä.*

Yksi Catmull-Rom-splinen merkittävä ominaisuus on, että käyrä on hallittavissa paikallisesti. Tämä tarkoitta sitä, että yhden kontrollipisteen muuttaminen vaikuttaa ainoastaan pieneen osaan käyrästä, jolloin käyrään on helpompi tehdä yksityiskohtaisempia muutoksia. (Yuksel et al. 2011, s. 747.) Eräs toinen ominaisuus on, että käyrän tangentin laskeminen on suhteellisen yksinkertaista, koska tangentti pisteessä $\mathbf{p}_i$ on samansuuntainen kuin pisteiden $\mathbf{p}_{i+1}$ ja $\mathbf{p}_{i-1}$ kautta kulkeva suora (Twigg 2003, s. 1). Selkein kääntöpuoli Catmull-Rom-splinessä on, että käyrä ei pysty kontrollipisteidensä konveksin peitteen sisällä, joten tämä ominaisuus on syytä huomioida esimerkiksi liikeratoja suunniteltaessa (Puhakka 2008, s. 73).

2.3.4 Chaikinin käyrä

Chaikinin käyrä on uniformi neliöllinen B-spline (Puhakka 2008, s. 73). Mielenkiintoisen tästä käyrästä tekee se, miten käyrä muodostuu verrattuna muihin käyriin. Toisin kuin yleensä, Chaikinin algoritmissa on jätetty pois analyyttinen lähestymistapa ja käyrä muodostetaan ainoastaan kontrollipisteitä käsittelemällä. Chaikinin algoritmia voidaan näin ollen kutsua geometriseksi algoritmiksi. Käytetyn algoritmin perusajatus on, että jokainen kontrollimonikulmion kulma ohitetaan uudella viivasegmentillä ja tällaista alijakoa jatketaan saadulle käyrälle rekursiivisesti, kunnes haluttu tarkkuus on saavutettu. (Joy 1999, s. 1-2.) Kuvassa 2.6 vasemmalla on esitetty eräs käyrä ensimmäisen alijaon jälkeen. Oikealla on esitetty sama käyrä kuudennen alijaon jälkeen. Katkoviivalla piirretty kuvio havainnollistaa alkuperäisen kontrollimonikulmion muotoa.



Kuva 2.6. *Chaikinin käyrä ensimmäisen ja kuudennen alijaon jälkeen.*

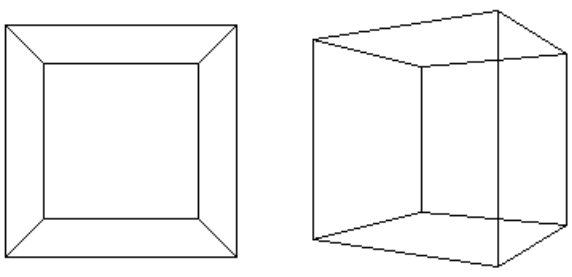
Alijako toteutetaan laskemalla uudet pisteet käsiteltävästä viivasegmentistä käyttämällä suhteita $\frac{1}{4}$ ja $\frac{3}{4}$ (Joy 1999, s. 2). Tämän jälkeen uudet pisteet yhdistetään viivasegmentillä, kuten myös kuvasta 3 voi havaita (Puhakka 2008, s. 73). Teoriassa tällaista jakoa voidaan jatkaa loputtomiin, mutta yleensä jakaminen lopetetaan, kun käyrä näyttää halutunlaiselta (Joy 1999, s. 6). Alijako muuttuu sitä raskaammaksi, mitä enemmän jakamista tehdään, koska pisteiden määrä lisääntyy joka kierroksella. Siksi jakamista on tarkoituksenmukaista rajoittaa myös tästä näkökulmasta.

3 3D-GRAFIIKKA

Nykyään useimmilla on jonkinlainen käsitys 3D-grafiikasta tietokonepelien, elokuvien ja esimerkiksi erilaisten suunnitteluohjelmistojen ansiosta. 3D-grafiikka, kaikessa hienoudessaan, ei kuitenkaan ole yksinkertainen asia, vaan se vaatii useita eri prosessointivaiheita, lukuisia algoritmeja ja näyttävyysvaatimusten kasvaessa myös tehokasta laitteistoa. Tässä luvussa tarkastellaan perustasolla, mitä 3D-grafiikka on ja miten kolmiulotteinen maailma saadaan esitettyä kaksiulotteisella tietokoneenruudulla. Lisäksi perehdytään parametroitujen käyrien esittämiseen 3D-avaruudessa. Lopuksi tehdään katsaus suosittuun OpenGL-grafiikkarajapintaan, jota myös tässä työssä hyödynnetään.

3.1 3D-grafiikan perusajatus

Kolmiulotteisuus tarkoittaa, että jokin mielivaltainen kappale kuvataan kolmen ulottuvuuden, leveyden, korkeuden ja syvyyden avulla (Wright et al. 2010, s. 11). Hyvä esimerkki tästä on kuutio, jota voidaan pitää yhtenä 3D-grafiikan peruskappaleista. Oleellisin kysymys on, että miten kolmiulotteinen maailma saadaan esitettyä kaksiulotteisella tasolla, joka tietokoneen tapauksessa on yleensä näyttöruutu. Ongelma ei kuitenkaan ole millään lailla tietokoneiden mukanaan tuoma, vaan on koskettanut muun muassa taidemaalareita jo satojen vuosien ajan (Wright et al. 2010, s. 11). Avaintekijä ongelman ratkaisemisessa on perspektiivi, joka saa kauempana katsojasta sijaitsevat kappaleet näyttämään pienemmiltä, jolloin lopputulos näyttää luonnollisemmalta ihmissilmälle (Wright et al. 2010, s. 13). Kuvassa 3.1 on havainnollistettu perspektiiviä yksinkertaisen rautalankakuution avulla.

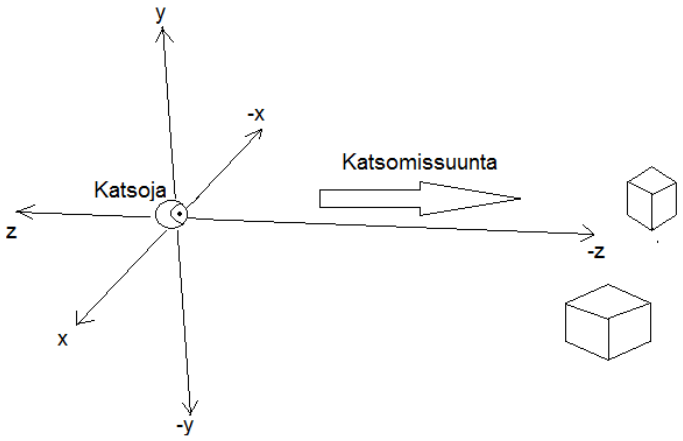


Kuva 3.1. *Perspektiiviprojisoitu rautalankakuutio edestä ja sivulta kuvattuna.*

Ehdottomasti yleisin tapa tuottaa 3D-grafiikkaa tietokoneen näytölle on käyttää projisointia ja erityisesti perspektiiviprojektiota (Puhakka 2008, s. 163, 178). Projisointiin perustuvassa piirtämisessä kappaleet muodostuvat kulmapisteistä, joita käytetään perusprimitiivien, kuten viivojen, kolmioiden ja neliöiden, muodostamiseen. Näitä perusprimitiivien

mitiivejä käytetään kokonaisten mallien tai esineiden muodostamiseen (Wright et al. 2010, s. 27). Kulmapiste ei ole muuta kuin piste käytetyssä avaruudessa, mutta saattaa sisältää erilaista piirtämiseen vaikuttavaa oheistietoa, kuten normaalivektoreita ja väritietoa (Puhakka 2008, s. 177, 197; Wright et al. 2010, s. 27).

Pelkkä projisointi ei yleensä ole riittävä muunnos, vaan usein 3D-avaruuden kappaleita halutaan muokata, siirtää tai muuntaa toiseen koordinaatistoon, jotta lopullisesta kuvasta saadaan halutunlainen. Yleisimmät muunnokset ovat maailmankoordinaatistoon muuntaminen, katsojan koordinaatistoon muuntaminen ja projisointi (Puhakka 2008, s. 173-177). Maailman koordinaatistoon muuntamisella tarkoitetaan kappaleen muutamista käsiteltävään kolmiulotteiseen maailmaan (Puhakka 2008, s. 166-167). Tämä muunnos voi sisältää esimerkiksi siirtämistä toisiin kappaleisiin nähden. Katsojan koordinaatistoon muuntamisella tarkoitetaan muunnosta sellaiseen koordinaatistoon, jossa katsoja on origossa, ja katsoo negatiivisen tai positiivisen *z*-akselin suuntaan (Wright et al. 2010, s. 132-133). Yksinkertaistettuna tämä muunnos voidaan ajatella ikään kuin kameran sijoittamisena kolmiulotteiseen maailmaan (Wright et al. 2010, s. 133). Muunnoksen lopputulosta on havainnollistettu kuvassa 3.2.

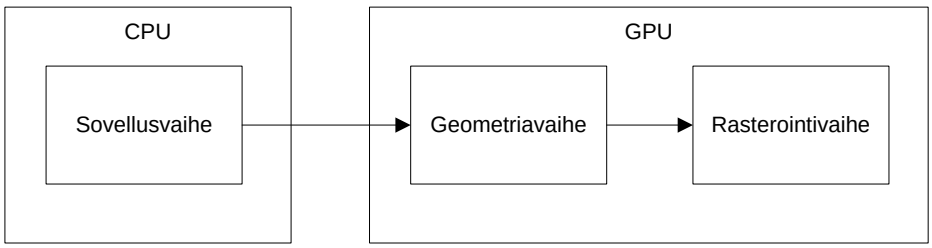


Kuva 3.2. *Katsojan koordinaatisto. Mukaillen lähteestä (McReynolds & Blythe 2005, s. 21).*

Projektiomuunnoksen tarkoituksena on saada kolmiulotteisen maailman kappaleet muunnettua sellaiseen muotoon, että ne voidaan piirtää kaksiulotteiselle pinnalle, joka yleensä on taso (Puhakka 2008, s. 177; Wright et al. 2010, s. 136). Kolmiulotteisen maailman muunnokset, kuten edellä kuvatut muunnokset, esitetään yleensä 4×4-matriisien avulla (Puhakka 2008, s. 103, 173-199). Selkein etu matriisien käytössä on se, että erilaisia muunnosmatriiseja voidaan yhdistää kertomalla niitä keskenään ja halutut kulmapisteet voidaan tällöin muuntaa käyttämällä yhdistämisen tuloksena saatua matriisia (Puhakka 2008, s. 102). Tällöin kulmapisteiden muuntaminen voidaan suorittaa parhaimmillaan yhdellä matriisikertolaskulla, mikä voi olla merkityksellistä suorituskyvyn kannalta, jos muunnettavia kulmapisteitä on paljon.

Edellä esitellyt muunnokset kuvataan usein osana liukuhihnaa, jonka läpi piirrettävä kappale kulkee ja jonka eri vaiheissa kappaleeseen kohdistetaan erilaisia muunnoksia (Puhakka 2008, s. 163). Liukuhihna-ajattelu ei sinänsä ota mitään kantaa siihen, mikä

tietokoneen osa muunnokset suorittaa, mutta nykyään käytettävät grafiikkaohjaimet on erityisesti suunniteltu tällaisen liukuhihna-ajattelun mukaisiksi (Puhakka 2008, s. 164-165). 3D-piirron liukuhihnan päävaiheet voidaan jakaa kolmeen osaan; sovellusvaiheeseen, geometriavaiheeseen ja rasterointivaiheeseen (Puhakka 2008, s. 164). Jokainen päävaihe koostuu vielä useasta osavaiheesta, jotka sovellusvaiheessa vaihtelevat sovelluksesta riippuen, mutta geometria- ja rasterointivaiheissa osavaiheet on tarkkaan määritelty, koska ne suoritetaan grafiikkaohjaimella (Graphics Pipeline 2012; Rendering Pipeline 2012). Liukuhihnan päävaiheet on esitetty myös kuvassa 3.3.



Kuva 3.3. *3D-piirron liukuhihnan päävaiheet.*

Sovellusvaiheen päätehtävä on muokata piirrettävät kappaleet sellaiseen muotoon, että ne voidaan lähettää grafiikkaohjaimen prosessoitaviksi. Nämä tehtävät suoritetaan tyypillisesti tietokoneen keskusyksiköllä. Tehtäviin kuuluu muun muassa animointia, fysiikan laskentaa ja näkyvyyskarsintaa. Myös aiemmin kuvattu maailmankoordinaatistoon muuntaminen suoritetaan tyypillisesti sovellusvaiheessa. (Puhakka 2008, s. 164-167.)

Seuraava vaihe, geometriavaihe, suoritetaan tyypillisesti grafiikkaohjaimella. Tässä vaiheessa toteutetaan muun muassa muunnos katsojan koordinaatistoon, projektiomuunnos ja piirrettävän kappaleen leikkaus näkyvyystestien perusteella. Lisäksi on mahdollista laskea esimerkiksi valaistusta kulmapistekohtaisesti. Nykyään grafiikkaohjaimen suorittaman geometriavaiheen toimintoja on mahdollista muokata käyttämällä ohjelmoitavia sävyttimiä (engl. shader). (Graphics Pipeline 2012; Puhakka 2008, s. 167-169; Rendering Pipeline 2012.)

Geometriavaiheen jälkeen vuorossa on rasterointivaihe. Tämän vaiheen päätehtävä on tuottaa lopullinen kuva kohdepinnalle. Kuten edeltävät vaiheet, myös rasterointivaihe koostuu monesta osavaiheesta. Osavaiheita ovat muun muassa takapintojen poisto, teksturointi, Z-testi ja lopulta pikselin kirjoittaminen videopuskuriin. Takapintojen poistolla tarkoitetaan sellaisten pintojen poistoa, jotka osoittavat katsojasta poispäin. Teksturointi tarkoittaa väritiedon hakemista esimerkiksi kuvasta, jos sellainen halutaan piirtää kappaleen päälle. Z-testi on eräs tärkeimmistä vaiheista, ja sen perusperiaate on varmistaa, että kappaleet piirretään oikein niiden näkyvyyteen ja toisiin kappaleisiin nähden. Toteutuksen kannalta tämä tarkoittaa jokaisen kappaleen pikselin syvyysarvon tutkimista aiemmin prosessoitujen kappaleiden pikseleiden syvyysarvoihin nähden. Tällöin tiedetään jääkö pikseli osin tai kokonaan jonkin toisen kappaleen taakse, jolloin tutkittava

pikseli voidaan hylätä. Kuten geometriavaiheessa, myös rasterointivaiheen toimintoja on mahdollista muokata ohjelmoitavia sävyttimiä käyttämällä. (Graphics Pipeline 2012; Rendering Pipeline 2012; Puhakka 2008, s. 169-171.)

3.2 Parametroidut käyrät 3D-avaruudessa

Parametroidut käyrät yleistyvät kolmanteen ulottuvuuteen varsin suoraviivaisesti esimerkiksi kaksiulotteisesta tilanteesta. Merkittävin muutos on, että 3D-avaruudessa käsiteltävissä pisteissä on mukana *z*-komponentti *x*- ja *y*-komponenttien lisäksi. (Puhakka 2008, s. 64.) Luonnollisesti 3D-avaruuden mukanaan tuomat vaatimukset pätevät käyriin vastaavalla tavalla kuin muihin kappaleisiin. Lisäksi jokaisen käyrän on kuljettava aiemmin kuvatun liukuhihnan läpi, jotta käyrä voidaan piirtää. Etuna tässä kuitenkin on, että käyrän piirtämisessä, ja mahdollisesti myös muussa prosessoinnissa, voidaan hyödyntää modernia grafiikkaohjainta. Tosin grafiikkaohjaimen hyödyntäminen on mahdollista myös kaksiulotteisessa tilanteessa. Eräs käyttökohde parametroiduille käyrille 3D-sovelluksissa on esimerkiksi liikeradan määrittäminen avaruutta kuvaavalle virtuaaliselle kameralle (Yuksel et al. 2011, s. 754).

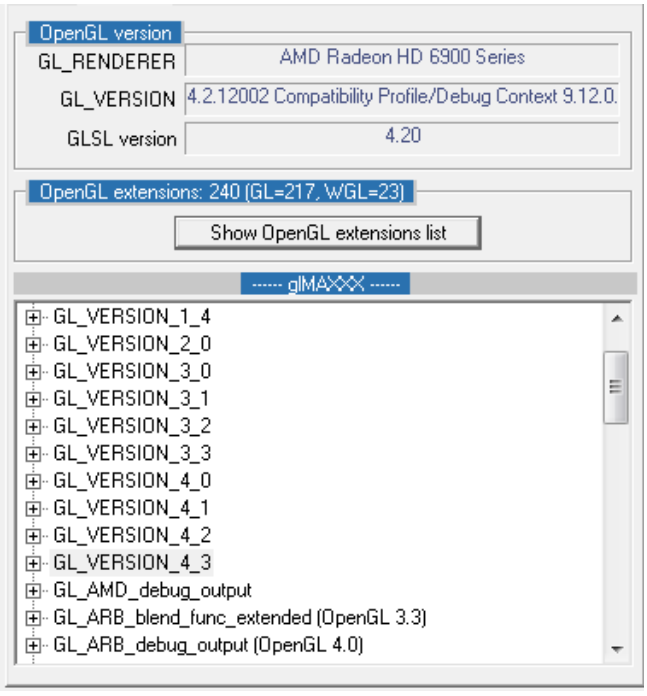
Kuten aiemmin mainittiin, 3D-avaruuden kappaleeseen on kohdistettava erinäisiä muunnoksia ennen kuin kappale voidaan piirtää kaksiulotteiselle pinnalle. Tämä koskee myös parametroituja käyriä. Toisaalta jos käyrää ei tarvitse piirtää, tarvitsee todennäköisesti suorittaa ainoastaan sovellusvaiheeseen kuuluvia muunnoksia. Muunnosten kohdistaminen suoraan käyrään voi kuitenkin olla erittäin raskasta, koska käyrä saattaa sisältää tuhansia pisteitä. Tässä on yksi kontrollipisteiden etu, koska perusmuunnokset (siirto, kierto ja skaalaus) voidaan suorittaa ainoastaan kontrollipisteille, ja käyrä mukautuu automaattisesti näiden muunnosten mukaiseksi (Puhakka 2008, s. 62-63, 72). Käyrää piirrettäessä joudutaan kuitenkin suorittamaan erityisesti perspektiivimuunnos suoraan käyrän pisteille, koska useimmat käyrät eivät säily, jos muunnos tehdään kontrollipisteille (Puhakka 2008, s. 72). On olemassa yksi käyrätyyppi, jolla on tämä hyödyllinen ominaisuus. Tällöin tarkoitetaan NURBS-käyriä (Nonuniform Rational B-spline), jotka tuovat lisää ilmaisuvoimaa ei-rationaalisiin käyriin verrattuna (Puhakka 2008, s. 71-72). NURBS-käyrien pisteissä otetaan käyttöön neljäs komponentti *w*, jolloin käyrää käsitellään neljännessä ulottuvuudessa. Käyrän kontrollipisteitä painotetaan kertoimella w_i, joka määrää käyrän muotoa sitä enemmän, mitä suurempi painokerroin on kyseessä. Tavallisilla B-spline-käyrillä tämä painokerroin on ykkönen. (Puhakka 2008, s. 72.) Tässä työssä ei kuitenkaan tarkastella NURBS-käyriä tämän enempää.

3.3 OpenGL 3D-toteutusvälineenä

OpenGL on 2D- ja 3D-piirtämineen tarkoitettu rajapinta, joka mahdollistaa grafiikkaohjaimen hyödyntämisen grafiikkaan liittyvässä laskennassa ja prosessoinnissa. Rajapinta perustuu Silicon Graphicsin kehittämään IRIS GL 2D- ja 3D-grafiikkakirjastoon, joka oli alunperin tarkoitettu käytettäväksi yrityksen myymissä huippuluokan työasemissa

(Wright et al. 2010, s. 34). Sittemmin teknologia kehittyi avoimeksi standardiksi ja Silicon Graphicsin konkurssin myötä OpenGL:n hallinnointi siirtyi Khronos Groupille, jonka jäsenistö koostuu useista tietotekniikka-alan yrityksistä (Wright et al. 2010, s. 35). Tällä hetkellä OpenGL:n uusin versio on 4.3, jonka määrittely julkaistiin 6. elokuuta 2012 (Khronos 2012). Nykyään OpenGL on erittäin suosittu vaihtoehto 3D-grafiikan toteuttamiseen Microsoftin Direct3D-kirjaston rinnalla. Erityisesti suosio on kasvanut mobiililaitteissa OpenGL ES:n (Embedded Systems) myötä.

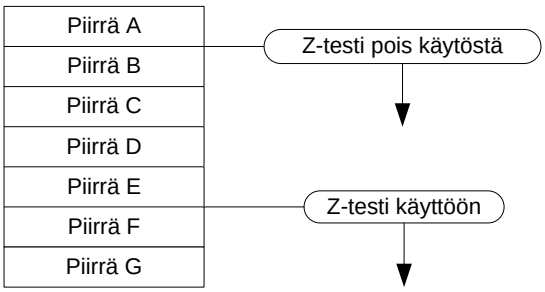
OpenGL:n merkittävin etu on avoimuus. Tämä tarkoittaa, että kaikki teknologiaan liittyvä materiaali, kuten määrittelydokumentaatio, on avointa ja näin ollen teknologia on mahdollista toteuttaa useille eri alustoille. Laitteistovalmistajat joutuvat kuitenkin hankkimaan lisenssin Khronos Groupilta, jos haluavat tarjota OpenGL-teknologiaa omissa tuotteissaan. (Wright et al. 2010, s. 38-39.) Tilanne on siis jokseenkin erilainen kuin Microsoftin Direct3D:n tapauksessa, jossa yksittäinen yritys hallitsee teknologiaa, ja sen toteuttamista erilaisille kohdealustoille (Wright et al. 2010, s. 39). Toisaalta OpenGL:n avoimuudessa voidaan nähdä myös kääntöpuolia. Laitteistovalmistajat ovat vastuussa OpenGL:ää tukevien laitteistoajurien tuottamisesta, jolloin tuettu versio, ominaisuudet ja standardinmukaisuus saattavat vaihdella valmistajasta ja ajuriversiosta riippuen (McReynolds & Blythe 2005, s. 129; Wright et al. 2010, s. 35). Vastaavasti valmistajakohtaisten laajennuksien käyttö saattaa aiheuttaa ongelmia, koska näitä laajennuksia ei välttämättä ole saatavilla muiden laitteistovalmistajien ajureissa (McReynolds & Blythe 2005, s. 131; Wright et al. 2010, s. 36-37). On olemassa työkaluja, kuten CPU Caps Viewer (GPU Caps Viewer 2010), joilla mahdollista tarkastella käytettävissä olevia OpenGL-ominaisuuksia, kuten kuvassa 3.4.



Kuva 3.4. *OpenGL-ominaisuudet tarkastelussa.*

Jos toteutettavan ohjelmiston halutaan toimivan usealla eri käyttöjärjestelmällä, OpenGL on yleensä ainoa vaihtoehto tämän vaatimuksen saavuttamiseksi. Tällä hetkellä OpenGL:ää voidaan hyödyntää kaikilla yleisimmillä käyttöjärjestelmillä, kuten Windowsilla, Mac OS X:llä, Linuxilla ja käytetyimmillä UNIX-varianteilla (OpenGL Platform 2013). Lisäksi ohjelmointiin voidaan käyttää useita eri ohjelmointikieliä (OpenGL Platform 2013; Wright et al. 2010, s. 38).

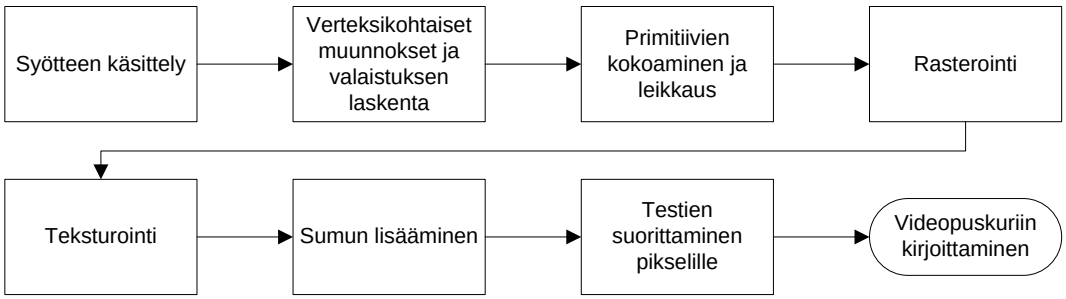
OpenGL on erittäin matalan tason rajapinta. Tämä tarkoittaa, että OpenGL keskittyy ainoastaan piirtämään perusprimitiivejä, jotka kulkevat OpenGL:n liukuhihnan läpi. Näin ollen OpenGL ei tarjoa täydellistä ratkaisua grafiikkaa hyödyntävän ohjelman tarpeisiin. Aiemmin kuvatuista liukuhihnan vaiheista OpenGL toteuttaa geometria- ja rasterointivaiheet. Eräs tärkeä OpenGL:n ominaisuus on sen tilakonemalli. Tämä tarkoittaa, että sovelluksessa asetettu tila, esimerkiksi Z-puskurin käyttäminen, on voimassa niin kauan, kunnes tila muutetaan (Puhakka 2008, s. 357). Tilakonemallin vaikutusta on havainnollistettu kuvassa 3.5. Kuvaan merkityt kohteet B-E piirretään ilman Z-testin suorittamista, koska Z-testi kytketään pois käytöstä kohteen A piirtämisen jälkeen.



Kuva 3.5. *Esimerkki OpenGL:n tilakonemallin vaikutuksesta.*

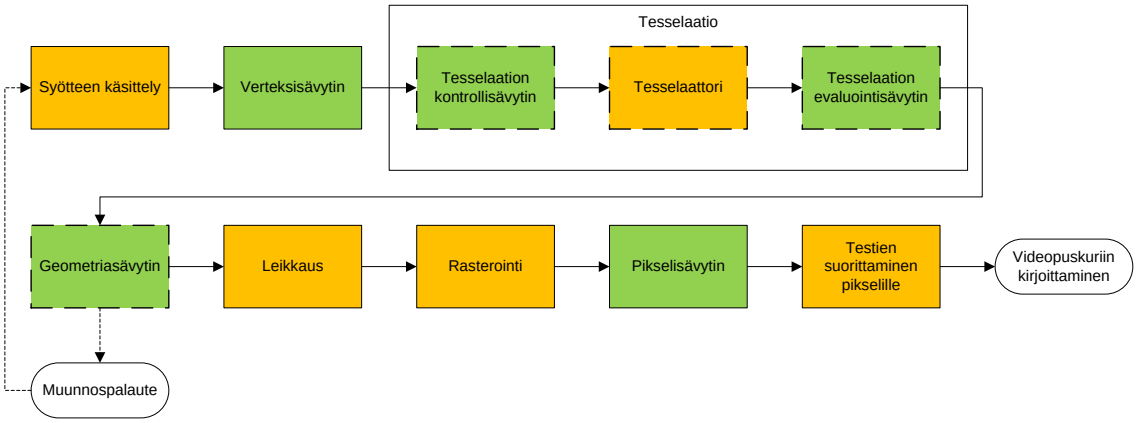
Sovelluksen toteuttamiseen liittyviin seikkoihin OpenGL ei ota mitään kantaa. Tällaisia seikkoja ovat muun muassa ikkunan luominen, tiedostojen lataaminen ja käyttäjän syötteiden vastaanottaminen. (FAQ 2012; Wright et al. 2010, s. 42.) Muista matalan tason ominaisuuksista mainittakoon, että OpenGL:n määrittely ei ota tarkalleen kantaa siihen, minne lopullinen piirretty kuva päätyy. Kohde voi olla käyttöjärjestelmän luoma ikkuna, mutta toisaalta kuva voidaan piirtää myös esimerkiksi grafiikkaohjaimen tai tietokoneen keskusmuistiin myöhempää käsittelyä varten. (McReynolds & Blythe 2005, s. 119-120.) Edellä mainituista syistä johtuen OpenGL:n käyttöön on toteutettu erilaisia apukirjastoja, jotka tekevät OpenGL:n käytöstä yksinkertaisempaa ja samalla toteuttavat ominaisuuksia, joita OpenGL ei tarjoa (Wright et al. 2010, s. 42-44).

Monia vuosia OpenGL:n hyödyntäminen tapahtui niin sanottua kiinteää liukuhihnaa käyttäen. Tämä tarkoitti sitä, että OpenGL:n suorittamat geometria- ja rasterointivaiheet olivat kiinteitä ja ohjelmoijalla oli hyvin vähän valtaa kiinteän toiminnallisuuden muokkaamiseen. Toisaalta tämä tapa koettiin usein helpoksi käyttää, varsinkin aloittelijoiden keskuudessa. Kuvassa 3.6 on nähtävissä kiinteän liukuhihnan perusperiaate.



Kuva 3.6. *OpenGL:n kiinteä liukuhihna. Mukaillen lähteistä (Puhakka 2008, s. 168, 170; Segal & Akeley 1997, s. 11).*

OpenGL-version 3.1 julkaisun yhteydessä tapahtui uudistus. Vanha ohjelmointimalli nimettiin yhteensopivuustoiminnallisuudeksi, jota ei tulisi enää käyttää uusissa sovelluksissa, vaan olisi ainoastaan olemassa vanhan ohjelmakoodin toimivuuden takaamiseksi. Uusissa sovelluksissa tulisi käyttää uutta ydintoiminnallisuutta, jonka perusperiaate on, että valtaosa vanhasta kiinteästä toiminnallisuudesta on poistettu, ja näiden toteutus on sovellusohjelmoijan vastuulla. Tämä johtaa siihen, että sovellusohjelmoijan työmäärä kasvaa, mutta toisaalta mahdollisuudet ovat myös parantuneet. Uusi ydintoiminnallisuus on siistimpi ja nopeampi kuin vanha kiinteä toiminnallisuus ja mahdollisuudet geometria- ja rasterointivaiheiden muokkaamiseen ovat parantuneet ohjelmoitavien sävyttimien ansiosta. (Rendering Pipeline 2012; Wright et al. 2010, s. 40-42.) Kuvassa 3.7 on havainnollistettu OpenGL:n liukuhihnaa, joka noudattaa OpenGL:n versiota 4.3 (Rendering Pipeline 2012).



Kuva 3.7. *Modernin OpenGL:n liukuhihna. Mukaillen lähteistä (Rendering Pipeline 2012; Wright et al. 2010, s. 509).*

Kuvassa 3.7 esiintyvät vihreät laatikot kuvaavat vaiheita, joita on mahdollista ohjelmoida sävyttimillä. Sävytin on ohjelma, joka suoritetaan tietokoneen grafiikkaohjaimella, ja toteuttaa piirrettävälle kappaleelle erilaisia operaatioita liukuhihnavaiheesta riippuen. Tyypillisimpiä sävyttimiä käsitellään seuraavassa kappaleessa. Kuvassa 3.7

oranssilla värjätyt laatikot kuvaavat liukuhihnan kiinteitä vaiheita. Katkoviivalla rajatut laatikot havainnollistavat valinnaisia vaiheita, jotka on mahdollista jättää suorittamatta. (Rendering Pipeline 2012.) Kuvaan merkitty muunnospalaute mahdollistaa verteksi- tai geometriasävyttimeltä saadun datan tallentamisen puskuriin, jonka sisältö voidaan lähettää takaisin liukuhihnan alkuun (Wright et al. 2010, s. 509).

Ohjelmoitavien sävyttimien käyttäminen on pakollista, jos halutaan hyödyntää modernia OpenGL:ää ja grafiikkaohjaimen uusimpia ominaisuuksia. Sävytinohjelmointiin on korkean tason ohjelmointikieli, GLSL (OpenGL Shading Language), joka muistuttaa syntaksiltaan C-ohjelmointikieltä (Wright et al. 2010, s. 230). Tyypillisimmät sävyttimet ovat verteksi- ja pikselisävyttimet (Puhakka 2008, s. 386). Verteksisävytin käsittelee yhtä verteksiä eli kulmapistettä kerrallaan ja suorittaa tyypillisesti erilaisia muunnoksia saamalleen pisteelle, jonka jälkeen sävytin lähettää sen liukuhihnan seuraavalle vaiheelle (Vertex Shader 2012). Pikselisävytin puolestaan käsittelee yhtä kohdepinnalle piirrettävää pikseliä ja määrittelee pikselille lopullisen värin (Fragment Shader 2012). Modernin grafiikkaohjaimen tapauksessa ei kuitenkaan tarvitse tyytyä ainoastaan verteksi- ja pikselisävyttimiin, koska on mahdollisuus käyttää myös tesselaatio- ja geometriasävyttimiä (Rendering Pipeline 2012). Vaiheet suoritetaan verteksisävyttimen jälkeen, ja näissä voidaan jakaa geometriaa pienempiin osiin tai jopa luoda uutta geometriaa (Rendering Pipeline 2012). OpenGL version 4.3 myötä on myös mahdollista käyttää puhtaaseen laskentaan soveltuvia sävyttimiä (engl. compute shader) (Compute Shader 2013). Laskentaan tarkoitetut sävyttimet eivät kuulu samaan liukuhihnaan, kuten aiemmin esitellyt sävyttimet, vaan toimivat yleensä täysin erillisenä vaiheena. Yksi tärkeä ero muihin sävyttimiin nähden on myös, että laskennallisille sävyttimille ei erikseen määritellä parametri- tai paluuarvoja. Näin ollen on täysin sävyttimen toteutuksesta kiinni, mistä prosessoitava materiaali ladataan ja minne laskettu tulos tallennetaan. (Compute Shader 2013.)

4 QT 5

Qt on alunperin norjalaisen Trolltechin kehittämä ohjelmistokehys graafisten käyttöliittymien toteuttamiseen C++-ohjelmointikielellä. Ensimmäinen julkinen versio, versio 0.90, julkaistiin toukokuussa 1995, minkä jälkeen Qt on kehittynyt yhdeksi parhaimmista ohjelmistokehitykseen tarkoitetuista työkaluista. Qt on saatavissa yleisimmille käyttöjärjestelmille, ja sitä on mahdollista käyttää sekä kaupallisella että avoimen lähdekoodin lisenssillä. (Blanchette & Summerfield 2006, s. xv-xvii.) Vuonna 2008 Qt:n omistus siirtyi Nokialle, kun Nokia päätyi ostamaan Trolltechin ja siirtämään Qt:n osaksi strategiaansa (Nokia 2008). Strategiamuutoksesta johtuen Nokia luopui Qt:n omistuksesta vuonna 2012, ja elokuussa 2012 Digia ilmoitti ostavansa koko Qt-kehitysympäristön (Digia 2012).

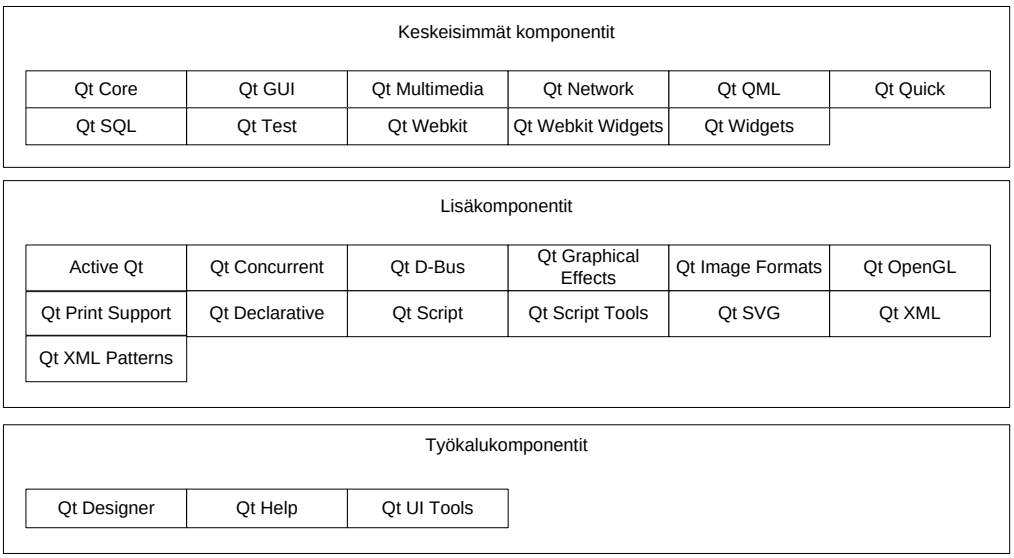
Tässä luvussa tarkastellaan Qt:ta erityisesti uusimman version näkökulmasta, ja millaisia uudistuksia tässä on vanhempaan Qt 4 -sarjaan nähden. Lisäksi tutustutaan Qt3D-komponenttiin, ja mitä ominaisuuksia se tuo 3D-grafiikan toteuttamiseen Qt-pohjaisissa sovelluksissa. Lopuksi tarkastellaan Qt:n tarjoamia työkaluja, joilla on mahdollista laajentaa toteutettavan sovelluksen ominaisuuksia joustavasti.

4.1 Qt:n yleisesittely

Qt on poikkialustainen (engl. cross-platform) ohjelmistokehys C++-ohjelmointikielelle. Qt:n perusajatuksena on, että ohjelmakoodi kirjoitetaan kerran, ja samaa lähdekoodia voidaan käyttää sellaisenaan tehtäessä sovelluskehitystä esimerkiksi usealle eri PC-käyttöjärjestelmälle (Qt Doc 5.0: Qt 5.0 2013). Erona esimerkiksi Java-ohjelmointikielellä toteutettuihin sovelluksiin on, että Qt-pohjaiset sovellukset käännetään erikseen jokaiselle kohdealustalle C++:n luonteesta johtuen (Blanchette & Summerfield 2006, s. 456). Uusin Qt-versio on saatavissa käytetyimmille työpöytäjärjestelmille, kuten Windowsille, Linuxille ja Max OS X:lle. Qt on saatavissa myös tietyille sulautetuille järjestelmille, kuten Embedded Linuxille ja Windows Embedded:lle. Lisäksi kehitystyötä tehdään Qt:n tuomiseksi käytetyimmille mobiilijärjestelmille, kuten Androidille, iOS:lle ja Windowsille. (QtDoc 5.0: Supported Platforms 2013.)

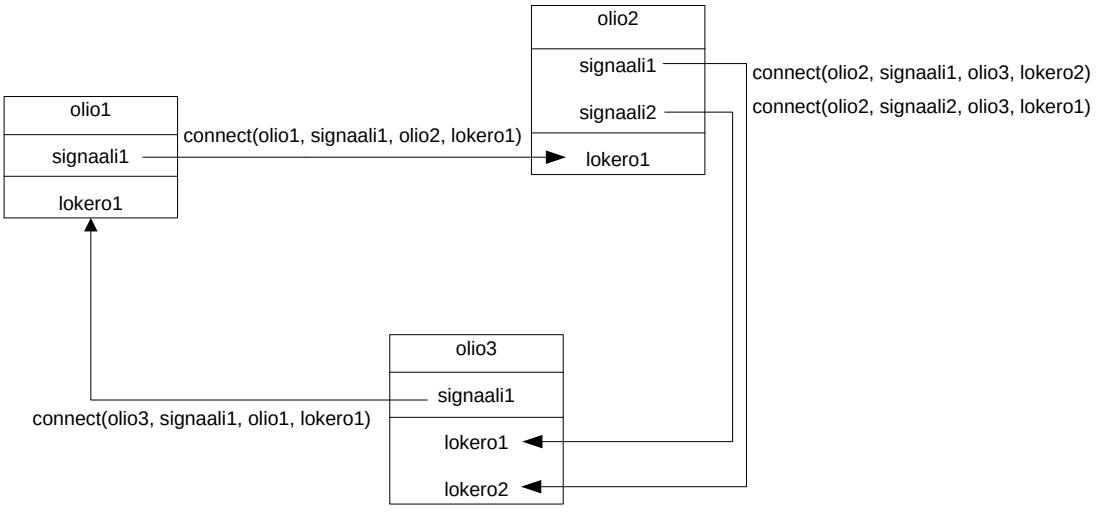
Vaikka Qt on alunperin suunniteltu graafisten käyttöliittymien toteuttamiseen, nykyään Qt tarjoaa laajan valikoiman sovelluskehityksessä tarvittuja komponentteja (QtDoc 5.0: All Classes 2013). Komponenttivalikoima sisältää muun muassa tyypillisimpiä tietorakenteita ja algoritmeja sekä rinnakkaisuuden, lokalisoinnin, tietoliikenneyhteyksien ja grafiikan tuottamisen mahdollistavia komponentteja. Qt:n komponentit on jaettu keskeisimpiin komponentteihin, lisäkomponentteihin ja työkalukomponentteihin. Tämä mahdollistaa, että ohjelmistokehittäjä pystyy entistä paremmin valitsemaan,

mitä ominaisuuksia Qt:sta käytetään (QtDoc 5.0: All Modules 2013). Qt:n sisältämät komponentit on esitetty kuvassa 4.1.



Kuva 4.1. *Qt:n sisältämät komponentit (QtDoc 5.0: All Modules 2013).*

Yksi merkittävimmistä C++-ohjelmointikieleen toteutetuista laajennuksista on Qt:n tarjoama signaali–lokero-tekniikka (engl. signals and slots), joka mahdollistaa kahden olion välisen vuorovaikutuksen ilman, että kyseiset oliot tietävät toisistaan (Blanchette & Summerfield 2006, s 20; QtCore 5.0: Signals & Slots 2013). Perusajatus on yhdistää signaaleja ja lokeroita niin, että olion A lähettämä signaali tuottaa funktiokutsun olion B lokeroksi määriteltyyn funktioon (Blanchette & Summerfield 2006, s. 20). Kuvassa 4.2 on myös havainnollistettu tätä menetelmää. Yhtenä esimerkkinä menetelmästä on painonappi, jonka painaminen lähettää signaalin ja painallustapahtuma voidaan käsitellä signaaliin yhdistetyssä lokerossa.



Kuva 4.2. *Qt:n signaali–lokero-tekniikka. Mukaillen lähteestä (QtCore 5.0: Signals & Slots 2013).*

Jotta signaali–lokero-tekniikkaa on mahdollista hyödyntää, on käsiteltyjen luokkien periydyttävä QObject-luokasta joko suorasti tai epäsuorasti (QtCore 5.0: Signals & Slots 2013). QObject on kantaluokka kaikille Qt:n metaoliojärjestelmää (engl. meta-object system) hyödyntäville luokille ja valtaosa Qt:n tarjoamista luokista periytyy QObject-luokasta (Blanchette & Summerfield 2006, s. 22). Metaoliojärjestelmän perusperiaate on tallentaa luokkaan metatietoa, joka mahdollistaa muun muassa signaali–lokero-tekniikan toteuttamisen (Blanchette & Summerfield 2006, s. 22). Metatiedon määrittelemiseksi on luokan esittelyssä käytettävä Q_OBJECT-makroa. Tämän jälkeen metatiedon tuottamiseen käytetään Qt:n omaa metaoliokääntäjää (engl. meta-object compiler), joka suoritetaan ohjelmakoodille ennen C++-kääntäjän suorittamista (QtCore 5.0: Signals & Slots 2013).

4.2 Qt 5:n tuomat uudistukset

Qt 5 julkaistiin joulukuussa 2012 (Knoll 2012). Tätä ennen käytössä ollut Qt 4 -sarja ehti palvella ohjelmistokehittäjiä yli seitsemän vuotta siitä lähtien, kun Qt 4.0 julkaistiin vuoden 2005 kesällä (Blanchette & Summerfield 2006, s. xvi). Qt 5 on pyritty toteuttamaan mahdollisimman yhteensopivaksi Qt 4 -versioihin nähden, jotta vanhat sovellukset olisi helpompi siirtää käyttämään Qt 5 -sarjaa (Knoll 2012).

Uusi versiosarja tuo luonnollisesti mukanaan paljon uudistuksia. Qt 5 painottaa erityisesti JavaScriptin ja Qt Quick:n merkitystä käyttöliittymäkehityksessä. Tätä varten koko piirtoarkkitehtuuri on suunniteltu uudelleen, ja uusi arkkitehtuuri perustuu OpenGL-pohjaiseen hierarkiseen malliin (engl. scene graph), mikä mahdollistaa tehokkaan piirtämisen, sulavien käyttöliittymien toteuttamisen sekä näyttävien efektien tuottamisen grafiikkaohjainta hyödyntämällä. Toisaalta myös laitteistovaatimukset nousevat vastaavassa suhteessa. Qt 5 tuo myös paremmat työkalut HTML5-kehittäjille parannetun JavaScript-tuen sekä uuden WebKit version myötä. (QtDoc 5.0: What's New in Qt 5 2013.)

Perinteisiä C++-pohjaisia käyttöliittymäkomponentteja (engl. widgets) ei kuitenkaan ole unohdettu, vaan ne on siirretty omaan komponenttiinsa. Komponentit ovat käytettävissä, kuten aiemmissa Qt-versioissa, mutta vanhan ohjelmakoodin päivittäminen Qt 5 -tyyliseksi vaatii, että projektitiedostoon lisätään widgets-komponenttimääritys. On myös mahdollista, että ohjelmakoodi vaatii päivitystä joiltain osin. Tämä muutos on osa modularisointia, joka toteutettiin Qt 5:n kehittämisen yhteydessä. (QtDoc 5.0: What's New in Qt 5 2013.)

Eräs C++-ohjelmointiin vaikuttava uudistus on uusi signaali–lokero-syntaksi. Uusi syntaksi käyttää funktio-osoittimia ja toimii myös muille kuin lokeroiksi merkityille funktioille. Syntaksi mahdollistaa myös staattisten funktioiden ja C++11-standardissa esiteltyjen lambdojen, eli anonyymifunktioiden, käytön perinteisten lokeroiden sijaan. (New Signal Slot Syntax Coming in Qt 5 2012.) Selkein etu uudessa toimintatavassa on, että syntaksi voidaan tarkistaa jo käännösaikakana, jolloin virheellisen toiminnan mahdollisuus pienenee. Vanha syntaksi aiheutti myös ongelmia nimiavaruuksista johtuen.

Nämä ongelmat pystytään välttämään käyttämällä uutta syntaksia. Kuvassa 4.3 on havainnollistettu sekä uutta että vanhaa syntaksia.

Vanha tapa:

```
connect(olio1, SIGNAL(signaali()), olio2, SLOT(lokero()));
```

Uusi tapa:

```
connect(olio1, &Luokka1::signaali, olio2, &Luokka2::lokero);
```

Uusi tapa - lambda:

```
connect(olio1, &Luokka1::signaali, []() {
    qDebug() << ”test”;
});
```

Kuva 4.3. *Vanha ja uusi signaali–lokero-syntaksi.*

Uudessa syntaksissa on myös kääntöpuolensa. Joidenkin mielestä uusi syntaksi saattaa olla monimutkaisempi vanhaan verrattuna, mutta eräs merkittävä seikka on, että lokeroissa ei enää voida käyttää vakioparametreja. Lisäksi signaali–lokero-yhteyden katkaisemiseen ei voida käyttää perinteistä disconnect-funktiota, jos yhteyden muodostamiseen on käytetty lambdoja tai staattisia funktioita. Tällöin on yhteyden katkaisemiseen käytettävä esimerkiksi connect-funktiolta saatavaa paluuarvoa. (New Signal Slot Syntax Coming in Qt 5 2012; QtDoc 5.0: What's New in Qt 5 2013.) Uutta syntaksia voidaan kuitenkin pitää silti parempana vaihtoehtona vanhaan nähden, koska vanhan syntaksin ongelmia on korjattu ja uudet ominaisuudet tekevät käytöstä joustavampaa.

Vielä mainitsemattomista uudistuksista mainittakoon muun muassa tuki C++11-standardille ja täysin uudelleenkirjoitettu alusta-arkkitehtuuri, joka mahdollistaa muun muassa ikkunointijärjestelmän dynaamisen latauksen käytetyn käyttöjärjestelmän perusteella. Kosketusnäytöllä suoritettavaan ohjaukseen on myös tehty muutoksia parantamalla usean kosketuspisteen tukea. Uutena tietorakenteena on tuotu tuki JSON-dokumenteille (JavaScript Object Notation). Tämä mahdollistaa tiedon varastoimisen JSON-dokumentteihin, ja vastaavasti näistä saatavan tiedon käsittelyn toteutetussa sovelluksessa. (QtDoc 5.0: What's New in Qt 5 2013.)

4.3 Qt:n käyttöliittymäkomponentit

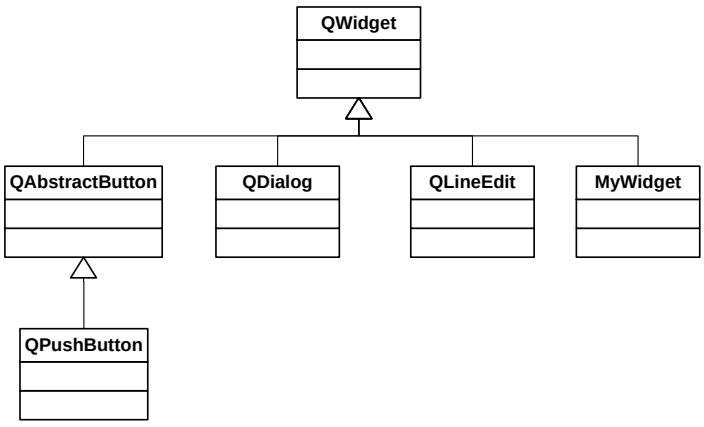
Qt 5 painottaa käyttöliittymien toteutuksessa Qt Quick -pohjaista toteutustapaa (QtDoc 5.0: What's New in Qt 5 2013). Qt Quick on moduuli, joka tarjoaa peruselementit QML-pohjaisten (Qt Meta Language) sovellusten toteuttamiseen ja mahdollistaa C++-toteutuksen ja QML-toteutuksen välisen kommunikoinnin (QtQuick 5.0: Qt Quick 2013). QML on korkean tason käyttöliittymien määrittely- ja ohjelmointikieli, joka poh-

jautuu JavaScriptiin ja JSON-tyyppiseen syntaksiin (QtQuick 5.0: QML 2013). JavaScript-pohjaisuus mahdollistaa sen, että QML-ohjelmakoodin seassa on mahdollista käyttää JavaScript-ohjelmakoodia esimerkiksi monimutkaisempien toiminnallisuuksien toteuttamiseen (QtQuick 5.0: QML 2013). QML:n suurin etu on sen dynaamisuus. QML-ohjelmakoodi ei tarvitse erillistä käännösprosessia, vaan ohjelmakoodi tulkataan suorituksen aikana. Tämä mahdollistaa esimerkiksi sen, että käyttöliittymää voidaan muokata koskematta C++-toteutukseen, jonka muokkaaminen vaatii aina sovelluksen uudelleenkääntämisen.

Qt Quick:n suurin heikkous tällä hetkellä on, että se ei tarjoa työpöytäympäristöön valmiita käyttöliittymäkomponentteja. Tämä tarkoittaa, että käyttöliittymäkomponenttien toteuttaminen on sovellusohjelmoijan vastuulla, mikä voi aiheuttaa merkittävää lisätyötä. Toisaalta kerran toteutettuja komponentteja voi hyödyntää useassa eri projektissa. Todennäköisesti tilanne paranee tulevaisuudessa, kun Qt Desktop Components -projektista julkaistaan ensimmäinen virallinen versio (Qt Desktop Components 2013).

Perinteiset käyttöliittymäkomponentit on toteutettu C++-ohjelmointikielellä. Nämä komponentit ovat tehokkaita, mutta käyttöliittymän muokkaaminen vaatii aina sovelluksen uudelleenkääntämisen. Yksi perinteisien käyttöliittymäkomponenttien etu on, että niiden tyyli mukautuu käytetyn käyttöjärjestelmän mukaiseksi (QtDoc 5.0: User Interfaces 2013). Näin ollen Qt ei pakota Qt-pohjaisia sovelluksia erottumaan käyttöjärjestelmän käyttöliittymätyylistä.

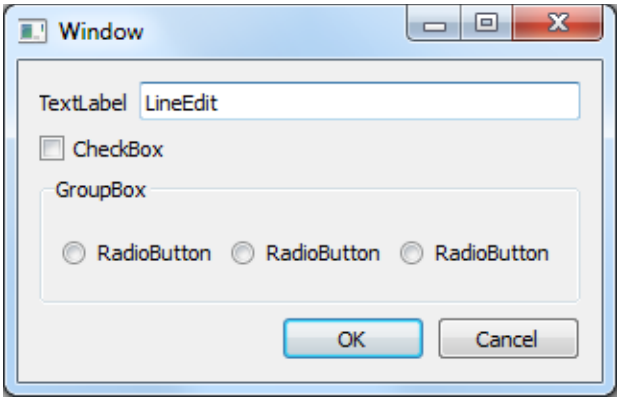
Qt:n C++-pohjaiset käyttöliittymäkomponentit periytyvät QWidget-luokasta joko suorasti tai epäsuorasti (QtWidgets 5.0: Qt Widgets 2013). Menetelmä mahdollistaa sovellusohjelmoijalle myös omien käyttöliittymäkomponenttien tuottamisen. Kuvassa 4.4 on esitetty muutaman käyttöliittymäkomponentin periytymishierarkia. Kuvan MyWidget-luokka on sovellusohjelmoijan toteuttama komponentti.



Kuva 4.4. *Esimerkki käyttöliittymäkomponenttien periytymishierarkiasta.*

QWidget-luokkaa on mahdollista hyödyntää myös sellaisenaan, ja yksi yleinen tapa on käyttää sitä säiliönä muille käyttöliittymäkomponenteille (QtWidgets 5.0: Widgets Tutorial 2013). Tällöin komponentit muodostavat vanhempi–lapsi-suhteen, jossa säiliökomponentti on vanhempi ja säiliöön lisätyt komponentit ovat lapsia. Jos komponentilla

ei ole määriteltyä vanhempaa, komponentti on ikkuna (QtWidgets 5.0: Qt Widgets 2013). Tyypillinen esimerkki tästä on sovelluksen pääikkuna. Kuvassa 4.5 on esitetty muutamia ikkunaan sijoitettuja käyttöliittymäkomponentteja. Ikkunassa esiintyvät komponentit ovat ikkunakomponentin lapsikomponentteja.



Kuva 4.5. *Ikkunaan sijoitettuja käyttöliittymäkomponentteja.*

C++-pohjaisia käyttöliittymäkomponentteja on mahdollista muokata käyttämällä Qt:n omaa tyylimäärittelyä (QtWidgets 5.0: Qt Style Sheets 2013). Tyylimäärittelyssä käytetään QSS-syntaksia (Qt Style Sheet), joka perustuu web-suunnittelussa paljon käytettyyn CSS-syntaksiin (Cascading Style Sheets) (QtWidgets 5.0: Qt Style Sheets 2013). Tyyli määritellään käyttäen käyttöliittymäkomponentin luokan nimeä:

```
QLabel {
     border: none;
     background: transparent;
}
```

Tyyli voidaan määritellä joko yksittäiselle käyttöliittymäkomponentille tai koko sovelluksen laajuiseksi, jolloin tyylimäärittely vaikuttaa kaikkiin määritellyn tyyppisiin komponentteihin (QtWidgets 5.0: Qt Style Sheets 2013). Tämä mahdollistaa sen, että käyttöliittymän tyyliä voidaan muokata yksityiskohtaisesti. Tyylimäärittely ei kuitenkaan mahdollista yhtä vapaata käyttöliittymän muokkausta kuin Qt Quick -pohjainen käyttöliittymä mahdollistaa. Näin ollen Qt Quick on parempi vaihtoehto täysin räätälöityjen käyttöliittymien tuottamiseen.

4.4 Qt3D

Qt3D on uusi moduuli, joka helpottaa 3D-grafiikan käsittelyä Qt-pohjaisissa sovelluksissa. Qt 5 ei vielä sisällä valmista Qt3D-moduulia, koska moduulin Qt 5 -toteutus on keskeneräinen, mutta sisällyttämistä on suunniteltu Qt-versioon 5.1 (Knoll 2012). Tällä hetkellä ainoa keino moduulin käyttämiseksi on noutaa lähdekoodi projektin versiohallinnasta ja kääntää moduuli omatoimisesti. Riskinä kuitenkin on, että lähdekoodi ei vält-

tämättä ole loppuun asti testattua, joten ohjelmistoa ei todennäköisesti kannata luovuttaa loppukäyttäjille ennen Qt3D-moduulin seuraavaa virallista julkaisua.

3D-grafiikan käsittely Qt3D:n avulla on melko vaivatonta. Qt3D:n oma rajapinta on toteutettu OpenGL-rajapinnan päälle ja mahdollistaa sen, että ohjelmoijalla ei välttämättä tarvitse olla aiempaa OpenGL-osaamista. Erityistoiminnallisuutta tai uusimpia OpenGL:n ominaisuuksia tarvittaessa kokemuksesta on luonnollisesti hyötyä. Geometrian käsittely tapahtuu pääsääntöisesti hierarkisen mallin avulla ja Qt3D tarjoaa useita peruskappaleita geometrian luomisen apuna (Qt3D Reference Documentation 2013). Qt3D:n hierarkisen mallin suurin ongelma on, että se mahdollistaa ainoastaan kolmioiden piirtämisen (Qt3D Reference Documentation 2013). Muiden primitiivien, esimerkiksi viivojen, tapauksessa on tehtävä oma piirtototeutus.

Qt3D tarjoaa myös perustoteutukset muun muassa kameran, tekstuurien, valaistuksen, materiaalien, ulkoisten 3D-malliformaattien ja sävyttimien käsittelyyn. Lisäksi tarjolla on aputoteutuksia matemaattisten operaatioiden ja muunnosten suorittamiseen. Yksi Qt3D:n tarjoama hyödyllinen ominaisuus on poiminta (engl. picking). Yksinkertaistettuna tämä tarkoittaa, että 3D-avaruuden kappaleita on mahdollista hallita hiirellä. Jotta poiminta olisi mahdollista, on kappale rekisteröitävä näkymäelementille käyttäen yksilöllistä tunnistetta. Tämän jälkeen kappaleen C++-toteutuksessa on mahdollista vastaanottaa hiiritapahtumia. (Qt3D Reference Documentation 2013.)

C++ ei kuitenkaan ole ainoa vaihtoehto Qt3D:n hyödyntämiseen. Qt3D sisältää myös Qt Quick -toteutuksen, jonka avulla on mahdollista tuottaa 3D-grafiikkaa JavaScriptiä ja QML-tekniikkaa käyttäen (Qt3D Reference Documentation 2013). Tämä mahdollistaa esimerkiksi sulavien kolmiulotteisten käyttöliittymien toteuttamisen.

4.5 Qt:n tarjoamat työkalut sovelluksen laajentamiseen

Qt tarjoaa tehokkaat työkalut oman sovelluksen toiminnallisuuden laajentamiseen. Yleisin tapa on toteuttaa uusi liitännäinen, eli plugin, joka voidaan ottaa käyttöön tai poistaa käytöstä dynaamisesti. Dynaaminen liitännäinen mahdollistaa sen, että laajennuksen toteuttamiseksi ei tarvitse muokata sovelluksen lähdekoodia. Qt mahdollistaa myös staattisten liitännäisten tuottamisen, mutta tässä työssä keskitytään ainoastaan dynaamisten liitännäisten käsittelyyn. Näiden tuottamiseen Qt tarjoaa sekä korkean että matalan tason rajapinnan. Korkean tason rajapinnan avulla on mahdollista laajentaa Qt:n omaa toiminnallisuutta, jolloin voidaan tuoda käyttöön esimerkiksi uusia kuvaformaatteja tai käyttöliittymätyylejä. Matalan tason rajapinnan avulla on mahdollista laajentaa ohjelmoijan omaa sovellusta. Tällöin ohjelmoijalla on täysin vapaat kädet liitännäisten tuottamisen suhteen. (QtCore 5.0: How to Create Qt Plugins 2013.) Tässä työssä keskitytään pelkästään jälkimmäiseen vaihtoehtoon.

Ensimmäinen vaihe laajennusmahdollisuuden toteuttamisessa on rajapinnan suunnittelu. Jokainen liitännäinen toteuttaa tämän rajapinnan ja vastaavasti sovellus käyttää rajapintaa liitännäisen hyödyntämiseen. Näin ollen sovelluksen ei tarvitse tietää liitännäisen sisäisestä toteutuksesta mitään, vaan se voi käyttää mitä tahansa liitännäistä yh-

den rajapinnan kautta. Kun rajapinta on määritelty, täytyy se vielä määritellä Qt:n ymmärtämäksi rajapinnaksi Q_DECLARE_INTERFACE-makrolla, jonka ensimmäisenä parametrina annetaan rajapintaluokan nimi ja toisena parametrina rajapinnan yksilöllinen tunniste. (QtCore 5.0: How to Create Qt Plugins 2013.) Ohjelmassa 4.1 on havainnollistettu rajapintaluokan toteuttamista.

```
#include <QtPlugin>

class IPlugin
{
public:
    virtual ~IPlugin() {}
    virtual void doSomething() = 0;
};

#define IPlugin_iid "MyApplication.IPlugin"
Q_DECLARE_INTERFACE(IPlugin, IPlugin_iid)
```

Ohjelma 4.1. *Rajapintaluokan määritteleminen.*

Kun rajapinta on määritelty, voidaan toteuttaa liitännäinen. Toteutuksessa on syytä huomioida kolme seikkaa, jotta liitännäinen toimisi oikein. Ensimmäinen seikka on, että pääluokan on periydyttävä rajapinnan lisäksi myös QObject-luokasta. Toisen huomioitava asia on, että luokan esittelyssä on käytettävä Q_INTERFACES-makroa, jonka avulla Qt:ta informoidaan toteutetusta rajapinnasta. Tämä makro ottaa ensimmäisenä parametrinaan toteutettavan rajapinnan nimen. Kolmantena asiana luokan määrittelyyn on lisättävä Q_PLUGIN_METADATA-makro, joka ottaa ensimmäisenä parametrinaan käytetyn rajapinnan yksilöllisen tunnisteen ja vapaavalintaisena toisena parametrina JSON-muotoisen dokumentin nimen, minkä avulla liitännäiselle voidaan määrittää metatietoa. Tämä jälkimmäinen makro tuottaa liitännäiselle käyttöönottoon tarvittavan toiminnallisuuden. (QtCore 5.0: How to Create Qt Plugins 2013.) Liitännäisen pääluokan määrittelemistä on havainnollistettu ohjelmassa 4.2.

```
class MyPlugin : public QObject, public IPlugin
{
    Q_OBJECT
    Q_PLUGIN_METADATA(IID IPlugin_iid)
    Q_INTERFACES(IPlugin)

public:
    MyPlugin();
    ~MyPlugin();
    void doSomething();
};
```

Ohjelma 4.2. *Liitännäisen pääluokan määritteleminen.*

Sovelluksessa liitännäisien käyttöönottaminen on mahdollista QPluginLoader-luokan avulla (QtCore 5.0: How to Create Qt Plugins 2013). Käyttöönotossa on kuitenkin syytä huomioida muutamia seikkoja. Ensinnäkin sekä sovelluksen että liitännäisen on syytä käyttää samaa Qt-versiota, koska muutoin käyttöönotto ei välttämättä ole mahdollista. Toiseksi sekä sovelluksen että liitännäisen kääntämiseen on syytä käyttää samaa kääntäjää, koska eri kääntäjät saattavat käyttää erilaista nimeämiskäytäntöä, jolloin liitännäisen käyttöönotto ei ole mahdollista. Kehitystyössä on myös syytä huomioida, että debug- ja release-käännösten yhdistäminen saattaa myös johtaa käyttöönoton epäonnistumiseen. (QtDoc 5.0: Deploying Plugins 2013.)

5 TOTEUTETUN SOVELLUKSEN KUVAUS

Tässä luvussa käsitellään parametroitujen käyrien suunnitteluun tarkoitettua sovellusta, jonka toteuttamista on käytetty tämän työn perustana. Aiemmin mainittiin, että kaikki käyrätyypit eivät noudata kontrollimonikulmion muotoa, joten lopullisen käyrän muodostumista voi olla vaikea hahmottaa etukäteen. Lisäksi jos kontrollipisteitä on paljon tai jos pisteitä halutaan muokata, hahmottaminen muuttuu entistä vaikeammaksi. Jo pelkästään näiden syiden pohjalta on perusteltua toteuttaa työkalu, joka visualisoi käyrän 3D-avaruudessa ja helpottaa käyrien suunnittelua sekä muokkausta.

Sovellukselle asetettiin kolme päätavoitetta, joista ensimmäiseen tavoitteeseen kuuluu, että sovelluksella tulee pystyä luomaan uusia käyriä 3D-avaruuteen. Toiseksi tavoitteeksi asetettiin, että olemassa olevaa käyrää tulee pystyä muokkaamaan. Kolmas tavoite liittyy käyrien jatkokäsittelyyn, mikä tarkoittaa, että käyrän kontrollipisteet tulee pystyä tallentamaan jatkokäsittelyn mahdollistavaan muotoon.

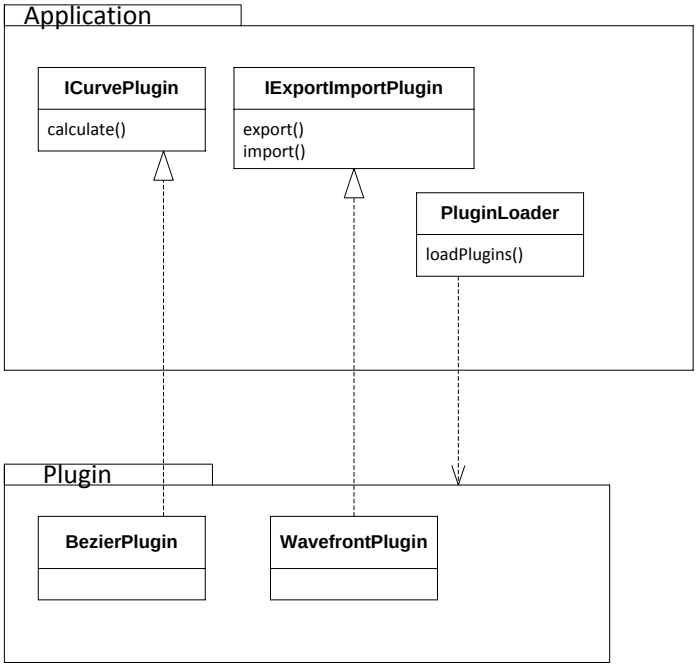
5.1 Toteutuksen kannalta merkittävät ratkaisut

Sovelluksen toteutuksessa käytettiin C++-ohjelmointikieltä ja Qt 5 -sovelluskehystä. 3D-grafiikan tuottamiseen valittiin Qt3D-moduuli, koska se yksinkertaistaa sovelluksen toteutusta ja tarjoaa valmiita komponentteja 3D-grafiikan tuottamiseen. Qt3D on toteutettu OpenGL:n päälle, joten OpenGL oli luonnollinen valinta piirtototeutukseen. Sovelluskoodin toteutuksessa pyrittiin siihen, että koodi olisi siirrettävissä sellaisenaan myös muille käyttöjärjestelmille.

Jotta käyrän hallitseminen olisi mahdollista, on kontrollipisteitä pystyttävä käsittelemään. Tämä tarkoittaa, että pelkkä kontrollipisteiden piirtäminen ei riitä, vaan niitä on pystyttävä valitsemaan 3D-avaruudesta käsittelemistä varten. Qt3D helpottaa kontrollipisteiden valitsemista poiminnan avulla. Jokainen kontrollipiste rekisteröidään 3D-näkymään yksilöllisellä tunnuksella, minkä jälkeen kontrollipiste voi vastaanottaa siihen kohdistuvia hiiritapahtumia. Kontrollipisteiden visualisoinnissa päädyttiin käyttämään neliöitä, koska neliö on helposti muokattavissa ja vaatii grafiikkaohjaimelta ainoastaan kahden kolmion piirtämistä, mikä on pieni määrä esimerkiksi kuutioon verrattuna. Neliön tai tason ominaisuus 3D-avaruudessa on, että sillä ei ole syvyyttä (Puhakka 2008, s. 41). Näin ollen sivulta katsottuna neliöllä ei ole pinta-alaa. Tämä aiheuttaa ongelman poiminnan kannalta, koska kontrollipistettä ei ole mahdollista valita mielivaltaisesta suunnasta tarkasteltuna. Ongelma ratkaistiin muuttamalla neliöt mainostauluiksi (engl. billboard). Mainostauluilla tarkoitetaan monikulmiota, joka kääntyy aina katsojan suuntaan (Puhakka 2008, s. 216-217). Neliön tapauksessa neliön etupuoli on aina kääntyneenä katsojaan päin, jolloin kontrollipiste on mahdollista valita katsojan sijainnista riip-

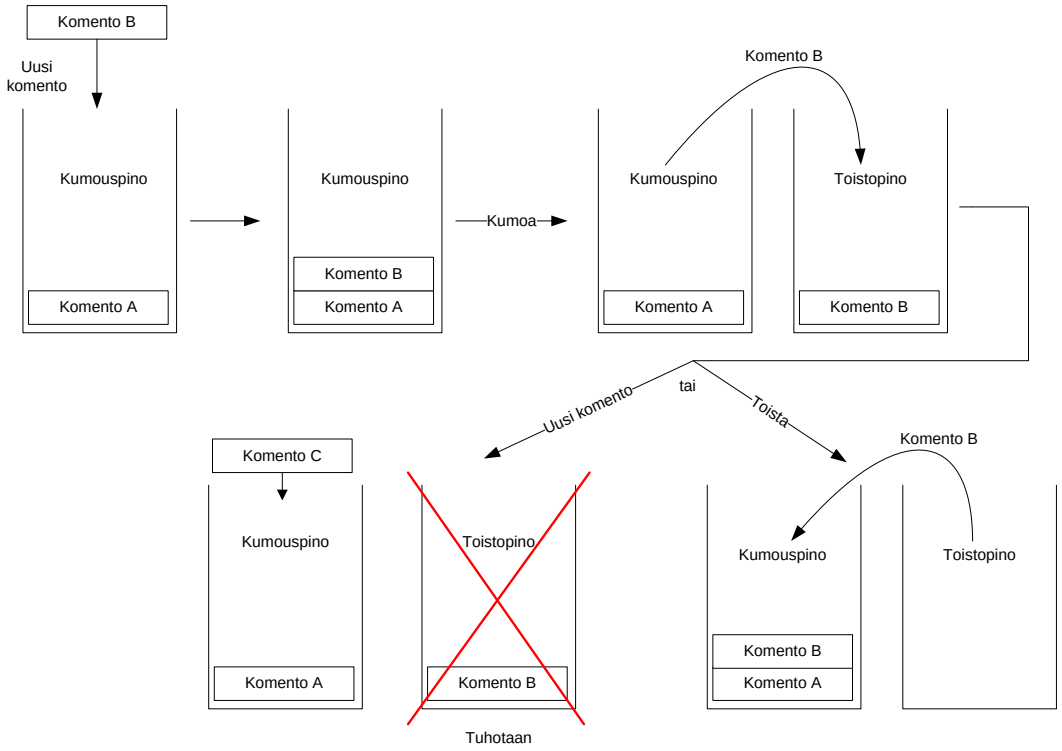
pumatta. Qt3D tarjoaa QGraphicsBillboardTransform-luokan mainostaulujen toteuttamiseen (Qt3D Reference Documentation 2013).

Liitännäisten käyttäminen on yksi merkittävimmistä suunnitteluratkaisuista, koska liitännäiset mahdollistavat sovelluksen joustavan laajentamisen. Sovelluksen jokainen käyrätyyppi on toteutettu liitännäisenä. Lisäksi vienti- ja tuontiformaattien (engl. export ja import) lisääminen sovellukseen on mahdollista toteuttamalla uusi liitännäinen. Liitännäisratkaisun perusperiaatetta on havainnollistettu kuvassa 5.1. Kuvassa alimmaisena oleva Plugin-kokonaisuus tarkoittaa liitännäistä yleisellä tasolla eikä sitä, että kokonaisuuden sisältämät kaksi luokkaa kuuluisivat samaan liitännäiseen.



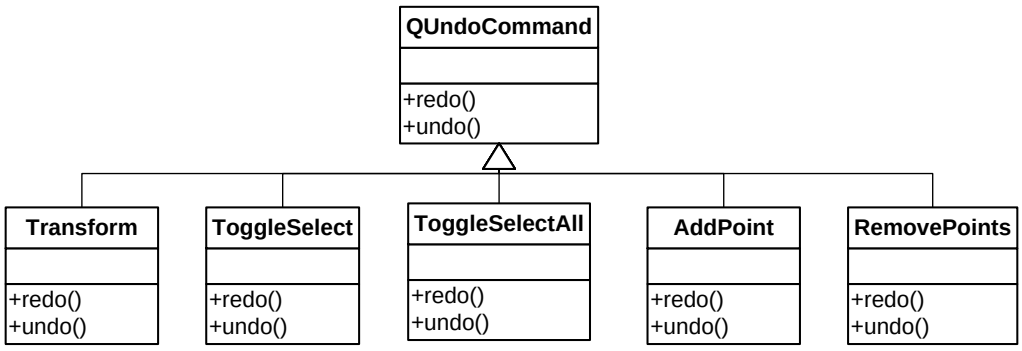
Kuva 5.1. *Periaatekuva sovelluksen liitännäisistä.*

Yksi käytettävyyteen positiivisesti vaikuttava tekijä on kumoa ja toista -toimintojen toteuttaminen. Qt tarjoaa tätä varten QUndoStack-luokan, joka ylläpitää luetteloa toteutuneista komennoista (QtDoc 5.0: Overview of Qt's Undo Framework 2013). Tällaisen tietorakenteen toiminnallisuutta on havainnollistettu kuvassa 5.2. Perusajatus on tallentaa sovelluksessa suoritetut komennot pino-tyyppiseen tietorakenteeseen, jolloin viimeisin suoritettu komento on tietorakenteessa päällimmäisenä. Näin ollen kumoa-käskyt suoritetaan tietorakenteeseen tallennetuille komennoille käänteisessä järjestyksessä. Pelkän kumouspinon toteuttaminen ei välttämättä ole riittävä ratkaisu, koska myös kumotut komennot on syytä tallentaa. Tätä varten voidaan toteuttaa toinen pino-tyyppinen tietorakenne, toistopino, kumotuille käskyille. Toista-toimintoa suoritettaessa toistopinon päällimmäinen komento siirretään toistopinosta kumouspinoon ja komento suoritetaan, kuten uuden komennon tapauksessa. Jos toistopino ei ole tyhjä uutta komentoa suoritettaessa, tällöin toistopinon sisältö tuhotaan, koska uusi komento ylikirjoittaa kumotut komennot. Edellä mainittuja toimintoja on myös havainnollistettu kuvassa 5.2.



Kuva 5.2. *Kumoa ja toista -komentoja ylläpitävät tietorakenteet ja niiden perusperiaate.*

QUndoStack-luokka käsittelee QUndoCommand-tyyppisiä instansseja, joten jokainen sovellusohjelmoijan toteuttama komento on periytettävä tästä luokasta (QtDoc 5.0: Overview of Qt's Undo Framework 2013). Kumoa ja toista -toiminnallisuuden toteuttaminen vaatii, että sovellus suunnitellaan tämän toiminnallisuuden vaatimalla tavalla. Jokainen sovelluksen komento, joka halutaan kumota tai toistaa, on toteutettava siten, että komento on kumottavissa tai toistettavissa käyttäjän niin halutessa. Kuvassa 5.3 on esitetty sovelluksen toiminnot, jotka voidaan kumota ja toistaa.



Kuva 5.3. *Kumottavissa ja toistettavissa olevat komennot.*

Jokainen komento toteuttaa vähintään funktiot redo ja undo. Komennoille on mahdollista antaa yksilölliset tunnukset, mikä mahdollistaa samantyyppisten peräkkäisten komentojen ryhmittelyn. Tällöin kumoaminen tai toistaminen suorittaa kaikki ryhmitel-

lyt komennot samalla kertaa. Ryhmittely vaatii, että komentoihin toteutetaan id- ja mergeWith-funktiot (QtDoc 5.0: Overview of Qt's Undo Framework 2013). Kumoamisen ja toistamisen mahdollistaminen ei ole ainoa QUndoStack-luokan etu. Luokka mahdollistaa myös niin sanotun puhdas-tilan asettamisen, mitä voidaan hyödyntää muun muassa tallennetun tiedon havaitsemiseen. Jos QUndoStack on puhdas-tilassa, tällöin käsiteltävän työn muutokset on tallennettu. Puhdas-tila asetetaan jokaisen tallennuksen jälkeen uudelleen, jotta tiedon hyödyntäminen on mahdollista.

5.2 Parametroidut käyrät sovelluksessa

Seuraavissa alakohdissa käsitellään parametroituja käyriä sovelluksen toteuttamisen kannalta. Näissä kohdissa tarkastellaan uuden käyrän lisäämistä, käyrän piirtämistä sekä käyrän tallentamista jatkokäsittelyn mahdollistavaan muotoon. Lisäksi käsitellään, miten käyrä on tallennettavissa myös muihin kuin sovelluksen käsittelemään tiedostomuotoon.

5.2.1 Uuden käyrän lisääminen sovellukseen

Kuten aiemmin mainittiin, sovelluksessa käsiteltävät käyrätyypit toteutetaan liitännäisinä. Uuden käyrätypin lisääminen ei vaadi sovelluskoodin muokkaamista, vaan käyrätyyppi voidaan toteuttaa täysin erillisenä projektina. Jokaisen käyrätyypin on toteutettava ICurvePlugin-rajapinta, joka on estetty ohjelmassa 5.1.

```
class ICurvePlugin
{
public:
    virtual ~ICurvePlugin() {}
    virtual QString name() const = 0;
    virtual void calculate(const QVector3DArray &controlPoints,
                           QVector3DArray *tgt) = 0;
    virtual void incCount() {}
    virtual void decCount() {}
};
```

Ohjelma 5.1. *Käyräliitännäisien toteuttama rajapinta.*

Jokaisen liitännäisen on toteutettava vähintään name- ja calculate-funktiot. Name-funktio palauttaa merkkijonona kyseisen käyrätyypin nimen, esimerkiksi Bézier. Calculate-funktio suorittaa käyrän laskemisen. Funktio saa ensimmäisenä parametrinaan viitteen taulukkoon, joka sisältää laskemisessa käytettävät kontrollipisteet. Toinen parametri on osoitin taulukkoon, johon käyrän pisteet tallennetaan. Kaksi viimeistä funktiota ovat vapaaehtoisia, ja tarkoitettu käyrän pistemäärän lisäämiseen tai vähentämiseen. Käyrän pistemäärä on nähtävissä sovelluksen tilariviltä, kuten kuvassa 5.4 on havainnollistettu. Kahden viimeisen funktion toteutukselle ei ole määritelty yksityiskohtia, koska käyrän pistemäärän hallitseminen on yleensä käyrätyypistä riippuvaista.

Joka tapauksessa incCount-funktiota kutsuttaessa käyrän pistemäärän tulisi lisääntyä ja decCount-funktiota kutsuttaessa pistemäärän tulisi vähentyä.

Kuva 5.4. *Kontrollipisteiden määrä ja käyräpisteiden määrä esitettynä sovelluksen tilarivillä.*

Käyrän laskennan toteutusyksityiskohdat ovat täysin liitännäisen toteuttajan päätettävissä. Luonnollisesti käyrätyyppi ja käytetty algoritmi asettavat vaatimuksia toteutukselle. Sovellusohjelmoija voi silti käyttää vapaasti esimerkiksi kolmannen osapuolen kirjastoja tai esimerkiksi suorittimen käskykantalaajennuksia, kuten SSE:tä (Streaming SIMD Extensions). Toisaalta esimerkiksi Microsoftin Visual Studio 2012 pyrkii tuottamaan käskykantalaajennuksia hyödyntävää ohjelmakoodia automaattisesti (Auto-Parallelization and Auto-Vectorization 2013). Rinnakkaisuuden hyödyntäminen käyrän laskennassa ei ole suositeltavaa, koska sovellus kutsuu liitännäisen calculate-funktiota synkronisesti.

Käyrätyypin käyttöönottamiseksi toteutettu liitännäinen on siirrettävä curvepluginskansioon, joka sijaitsee sovelluksen ohjelmatiedoston juuressa. Sovellus etsii käyräliitännäisiä ainoastaan tästä kansiosta. Jos sovellus ei löydä yhtään käyräliitännäistä, sovellus antaa asiasta virheilmoituksen ja suoritus lopetetaan, koska sovelluksen oikeaoppinen toiminta vaatii, että käytössä on vähintään yksi käyrätyyppi.

5.2.2 Käyrän piirtäminen

Käyrä koostuu käyräpisteistä, jotka lasketaan kontrollipisteiden ja käyrätyypille määritellyn algoritmin avulla. Laskennassa käytetään käyttäjän valitsemaa käyräliitännäistä, jonka calculate-funktio tuottaa tarvitut käyräpisteet. Sovellus hyödyntää laskennassa rinnakkaisuutta, jotta laskenta ei häiritse käyttöliittymän toimintaa. Qt tarjoaa useita tapoja rinnakkaisuuden toteuttamiseen, mutta tässä työssä käytettiin QtConcurrent-rajapintaa sen yksinkertaisuuden ansiosta. QtConcurrent tarjoaa korkean tason rajapinnan rinnakkaisuuden tuottamiseen, ja asettaa käytettyjen säikeiden määrän käytettävissä olevien suoritinytimien mukaan (QtConcurrent 5.0 2013). Näin ollen sovellus osaa automaattisesti hyödyntää suorittimen koko kapasiteettia.

Yksinkertaisin tapa hyödyntää QtConcurrent-rajapintaa on käyttää QtConcurrent::run-funktiota. Tämä funktio ottaa parametrinaan osoittimen funktioon, joka suoritetaan toisessa säikeessä (QtConcurrent 5.0 2013). QtConcurrent::run-funktio palauttaa futuurin, jonka avulla on mahdollista havaita esimerkiksi säikeen suorituksen loppuminen (QtConcurrent 5.0 2013). Ohjelmassa 5.2 on havainnollistettu käyrän laskentaa QtConcurrent::run-funktiota käyttäen. Ohjelmassa esitetty luokka ControlPointController hallinnoi sovelluksessa käsiteltyjä kontrollipisteitä. Esimerkin lopussa sijaitseva m_calcWatcher on QFutureWatcher-tyyppinen olio, joka mahdollistaa

futuurien tarkkailun signaali–lokero-tekniikkaa hyödyntäen (QtConcurrent 5.0 2013). Ohjelmassa 5.2 QtConcurrent::run-funktion parametrina käytetään yksinkertaista lambda-funktiota.

```
void calculateCurve(const ControlPointController *cpc,
                    ICurvePlugin *curve)
{
   QFuture<QVector3DArray*> res;
   res = QtConcurrent::run([cpc, curve]() -> QVector3DArray* {
         if (cpc->controlPointCount() < 2)
             return nullptr;
         QVector3DArray cPoints;
         cpc->copyCoordinates(&cPoints);
         auto *tgt = new QVector3DArray;
         curve->calculate(cPoints, tgt);
         return tgt;
    });
    m_calcWatcher.setFuture(res);
}
```

Ohjelma 5.2. *Käyräpisteiden laskemisessa käytetty funktio.*

Käyrä voidaan piirtää, kun käyräpisteet on laskettu. Qt3D mahdollistaa 3D-avaruuteen piirtämisen QGLPainter-luokan avulla. QGLPainter-instanssi saadaan QGLView-tyyppisen näkymän paintGL-funktion parametrina (Qt3D Reference Documentation 2013). PaintGL-funktio on tarkoitettu piirtorutiinien toteuttamiseen, joten myös käyrä piirretään tässä funktiossa. QGLPainter mahdollistaa perusprimitiivien piirtämisen ilman, että sovellusohjelmoijan tarvitsee toteuttaa OpenGL-sävyttimiä. Käyrä voidaan piirtää esimerkiksi seuraavalla tavalla:

```
painter->setColor(0xffffff);
painter->setVertexAttribute(QGL::Position, *m_curvePoints);
glLineWidth(2.5f);
painter->draw(QGL::LineStrip, m_curvePoints->size());
```

Ensimmäiseksi käyrälle asetetaan väri, joka tässä tapauksessa on valkoinen. Seuraavaksi määritellään käsiteltävät kulmapisteet, jotka ovat QGL::Position-tyyppisiä. Käyrän tapauksessa kulmapisteillä tarkoitetaan käyrän pisteitä (Qt3D Reference Documentation 2013). Kolmannella rivillä kutsutaan OpenGL-funktiota, joka määrittää käyrän leveyden. Qt3D ei tarjoa vastaavaa toiminnallisuutta, joten leveyden määrittelemiseksi on käytettävä suoraan OpenGL-toteutusta. Viimeisellä rivillä suoritetaan käyrän piirtäminen. Piirtofunktio ottaa ensimmäisenä parametrina piirrettävien primitiivien tyypin, ja toisena parametrina käsiteltävien kulmapisteiden määrän (Qt3D Reference Documentation 2013). Huomionarvoista on, että Qt3D mahdollistaa käyrän piirtämisen yksinkertaisimmillaan kahdella koodirivillä, jos unohdetaan värin ja leveyden määrittäminen. Vastaavan toiminnallisuuden tuottaminen suoraan OpenGL:ää käyttäen vaatisi huomattavasti enemmän sovelluskoodia sekä omien sävyttimien toteuttamista.

5.2.3 Käyrän tallentaminen ja tallennetun käyrän lataaminen

Käyrän jatkokäsittelyn kannalta on oleellista, että käyrä voidaan tallentaa. Jatkokäsittelyllä tarkoitetaan, että tallennettua käyrää on mahdollista hyödyntää tässä tai jossain toisessa sovelluksessa. Tallennusformaatiksi valittiin XML (Extensible Markup Language). XML:n etu on, että sitä voidaan hyödyntää lähes millä tahansa ohjelmointikielellä tai käyttöjärjestelmällä. Qt tarjoaa kattavan työkaluvalikoiman XML:n käsittelyyn C++-ohjelmointikielellä (XML Support in Qt 2013).

Käyrän tallentamisen kannalta on riittävää, että pelkät kontrollipisteet tallennetaan, koska käyrä on aina laskettavissa näiden pisteiden perusteella. Eräs tallennettu käyrä on esitetty kuvassa 5.5. Käyräpisteiden tallettaminen ei ole järkevää, koska tallennetun käyrän muoto riippuu käyrätyypistä. Tosin tätä suuremmaksi ongelmaksi muodostuu se, että käyräpisteiden perusteella ei voida päätellä käyrän kontrollipisteitä.

```
<?xml version="1.0" encoding="UTF-8"?>
<curve>
    <controlpoints>
        <cp x="-1" y="2" z="0"/>
        <cp x="1" y="2" z="0"/>
        <cp x="1" y="4" z="0"/>
        <cp x="-1" y="4" z="0"/>
        <cp x="-1" y="2" z="0"/>
        <cp x="1" y="2" z="0"/>
        <cp x="1" y="4" z="0"/>
    </controlpoints>
</curve>
```

Kuva 5.5. *Tallennettu käyrä.*

XML ei ole ainoa tiedostomuoto, jota sovelluksessa on mahdollista hyödyntää. Kuten aiemmin mainittiin, erilaisten vienti- ja tuontiformaattien toteuttaminen on mahdollista liitännäisien avulla. Tällaisen liitännäisen toteuttama rajapinta on esitetty ohjelmassa 5.3.

```
class IExportImportPlugin
{
public:
    virtual ~IExportImportPlugin() {}
    virtual QString extension() const = 0;
    virtual QString menuName() const = 0;
    virtual bool exportPoints(const QString &fileName,
                              const QVector3DArray &src,
                              QString *error) = 0;
    virtual bool importPoints(const QString &fileName,
                              QVector3DArray *tgt,
                              QString *error) = 0;
};
```

Ohjelma 5.3. *Vienti- ja tuontiliitännäisien toteuttama rajapinta.*

Rajapinnan kaikki funktiot on toteutettava. Ensimmäinen funktio palauttaa liitännäisen tuottaman tiedostoformaatin tiedostopäätteen. Toinen funktio palauttaa tiedostoformaatin nimen, joka näytetään sovelluksen vienti- ja tuontivalikoissa. Kolmas funktio suorittaa tiedon tallentamisen kohdelevylle. Funktio saa parametreinaan kohdetiedoston nimen, tallennettavat pisteet ja osoittimen merkkijonoon, johon voi tallettaa käyttäjälle näytettävän virheviestin operaation epäonnistuessa. Viimeinen funktio suorittaa tiedoston lukemisen ensimmäisenä parametrina annetun tiedostonimen perusteella. Funktio sijoittaa luetut pisteet toisen parametrin osoittamaan taulukkoon. Kolmannen parametrin osoittamaan merkkijonoon on mahdollista tallentaa käyttäjälle näytettävä virheviesti. Kaksi viimeistä funktiota palauttavat boolean-tyyppisen arvon sen perusteella, onnistuiko operaatio.

Liitännäiset mahdollistavat sekä kontrollipisteiden että käyräpisteiden tallentamisen. Käyräpisteiden tallentaminen on sallittu siitä syystä, että liitännäinen saattaa toteuttaa sellaisen tiedostoformaatin, jota on mahdollista hyödyntää esimerkiksi 3D-mallinnusohjelmassa. Tällöin kokonaisen käyrän tallentaminen on perusteltua. Käyrän lataaminen liitännäisien avulla on suunniteltu ainoastaan kontrollipisteiden käsittelyyn, kuten XML-tallennuksien tapauksessa.

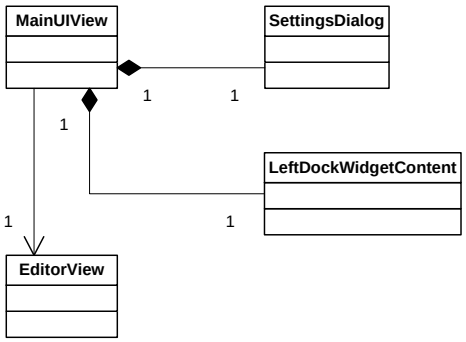
Vienti- ja tuontiliitännäiset tallennetaan sovelluksen juureen exportimportpluginskansioon. Tämä on ainoa paikka, josta sovellus yrittää ladata kyseisiä liitännäisiä. Sovelluksen toiminta ei ole millään lailla vienti- ja tuontiliitännäisistä riippuvaista, joten liitännäisen toteuttaminen on vapaaehtoista. Ne kuitenkin tarjoavat joustavan tavan laajentaa sovelluksella tuotettujen käyrien jatkokäsittelymahdollisuuksia.

5.3 Käyttöliittymä

Alunperin sovelluksen käyttöliittymä oli tarkoitus toteuttaa Qt Quick -tekniikkaa hyödyntäen. Ongelmaksi muodostui Qt3D, jonka luoma 3D-näkymä ei aktivoinut omistamaansa OpenGL-kontekstia, kun 3D-näkymä muuttui aktiiviseksi. Tämä aiheutti sovelluksen kaatumisen, koska Qt Quick käyttää myös OpenGL:ää. Ongelman kiertäminen ei onnistunut sellaisella varmuudella, että käyttöliittymän toteuttaminen Qt Quick:n avulla olisi ollut kannattavaa. Tästä syystä käyttöliittymä toteutettiin C++-pohjaisilla käyttöliittymäkomponenteilla, jotka eivät käytä OpenGL:ää.

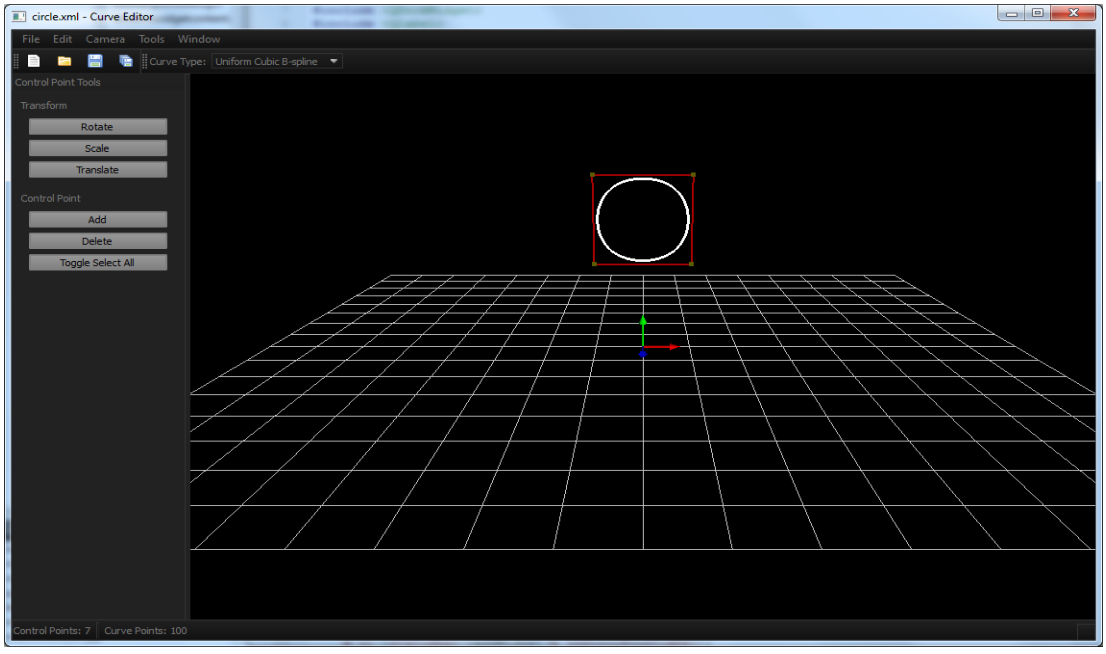
Sovelluksen käyttöliittymäkokonaisuuden toteuttaminen C++-pohjaisilla komponenteilla ei osoittautunut täysin suoraviivaiseksi. Käyttöliittymäkomponentit perityvät QWidget-luokasta, mutta Qt3D:n luoma 3D-näkymä periytyy QWindow-luokasta, jota Qt ei ole sisältänyt ennen Qt 5 -sarjaa (QtGui 5.0: QWindow Class 2013). Nämä luokat eivät ole keskenään yhteensopivia, joten 3D-näkymää ei voi sellaisenaan integroida käyttöliittymään. Qt:n kehittäjät ovat tiedostaneet ongelman ja korjaus on toteutettu Qt-versioon 5.1 ([#QTBUG-25643] 2013; Sletta 2013). Tällä välin ongelman voi kiertää toteuttamalla vastaavan toiminnallisuuden osana sovellusta, kuten tämän sovelluksen tapauksessa päädyttiin tekemään.

Käyttöliittymä toteutettiin mahdollisimman erillisenä sovelluksen ydintoiminnallisuudesta. Näin ollen käyttöliittymän vaihtaminen vaatii mahdollisimman vähän sovelluskoodin muokkaamista. Käyttöliittymä huolehtii käyttöliittymätapahtumista, mutta tapahtumien käsittelyä ei suoriteta käyttöliittymäkoodissa, vaan tapahtuma välitetään käyttöliittymän ulkopuolelle signaali–lokero-tekniikkaa käyttäen. Käyttöliittymän pääluokkia on havainnollistettu kuvassa 5.6.



Kuva 5.6. *Käyttöliittymän pääluokat.*

Kuvan 5.6 EditorView-luokka ei varsinaisesti kuulu käyttöliittymään, koska luokka on Qt3D:n luoma 3D-näkymä, ja se on aina olemassa sellaisenaan käyttöliittymän toteutustavasta riippumatta. 3D-näkymä on kuitenkin liitettävä osaksi käyttöliittymää, jotta 3D-näkymä ei jää käyttöliittymäikkunan ulkopuoliseksi ikkunaksi. Sovelluksen pääkäyttöliittymänäkymän toteuttaa luokka MainUIView. Luokka vastaa koko käyttöliittymän toiminnallisuudesta. Luokka LeftDockWidgetContent toteuttaa käyttöliittymän vasemmassa reunassa sijaitsevan työkaluvalikon vaatiman toiminnallisuuden. Sovelluksen käyttöliittymä on esitetty kuvassa 5.7.



Kuva 5.7. *Sovelluksen käyttöliittymä.*

Käyttöliittymän mukauttamisessa käytettiin Qt:n tyylimäärittelyitä. Jokainen käytetty käyttöliittymäkomponentti määriteltiin tyylitiedostoon, jonka sovellus lukee sovelluksen käynnistymisen yhteydessä. Sovellus ei kuitenkaan pakota tyylitiedoston käyttämiseen, joten tyylitiedoston käyttöönotto voidaan estää poistamalla tyylitiedosto sovelluksen juuresta.

Sovellusta hallitaan usealla eri tavalla. Valtaosa sovelluksen toiminnallisuudesta on sijoitettu käyttöliittymään, mutta tyypillisimmille toiminnoille on asetettu myös pikanäppäimet, jotka nopeuttavat sovelluksen käyttöä. Qt mahdollistaa pikanäppäimien asettamisen käyttöliittymään sijoitetuille toiminnoille. Jos toimintoa ei ole sijoitettu käyttöliittymään, ainoa keino pikanäppäimen asettamiseksi on tarkkailla sovelluksen näppäintapahtumia. Sovelluksen pikanäppäimet on esitetty taulukossa 5.1.

Taulukko 5.1. *Sovelluksen pikanäppäimet selityksineen.*

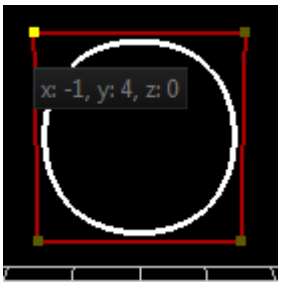
Pikanäppäin	**Selitys**
CTRL+N	Uusi käyrä
CTRL+O	Avaa tallennettu käyrä
CTRL+S	Tallenna käyrä
CTRL+Z	Kumoa
CTRL+Y	Toista
CTRL+Shift+F11	Siirry kokonäyttötilaan tai pois siitä
A	Valitse kaikki kontrollipisteet tai poista kaikki valinnat
Del	Poista valitut kontrollipisteet
R	Aloita kiertomuunnos valituille kontrollipisteille
S	Aloita skaalausmuunnos valituille kontrollipisteille
T	Aloita siirtomuunnos valituille kontrollipisteille
X	Lukitse suoritettava muunnos X-akselille
Y	Lukitse suoritettava muunnos Y-akselille
Z	Lukitse suoritettava muunnos Z-akselille
Enter	Hyväksy muunnos
Esc	Peruuta muunnos
+	Lisää käyräpisteitä
-	Vähennä käyräpisteitä

3D-näkymässä esitetyn kolmiulotteisen maailman hallitseminen suoritetaan hiiren avulla. Kameran liikuttelu tapahtuu painamalla hiiren vasenta painiketta ja liikuttamalla hiirtä. Näkymän tarkentaminen tai loitontaminen suoritetaan hiiren rullan avulla. Hiiren avulla on myös mahdollista lisätä uusi kontrollipiste 3D-avaruuteen tuplaklikkaamalla hiiren vasenta painiketta. Kontrollipisteiden yksilöllinen valitseminen on mahdollista hiiren oikealla näppäimellä. Geometristen muunnosten kohdistaminen kontrollipisteille suoritetaan liikuttamalla hiirtä halutun muunnoksen aikaansaamiseksi, kun muuntami-

nen on aloitettu. Muunnos on mahdollista hyväksyä hiiren vasemman painikkeen painalluksella.

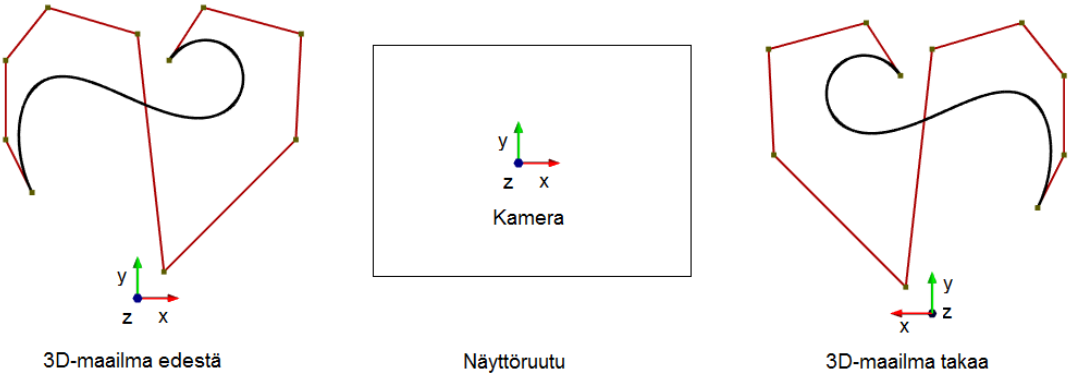
5.4 3D-grafiikan haasteet sovelluksen toteutuksessa

3D-grafiikka asettaa monia haasteita sovelluksen toteuttamiseen lähinnä siitä syystä, että näyttöruutu on kaksiulotteinen. Qt3D:n eräs etu on, että se ratkaisee osan näistä ongelmista sovellusohjelmoijan puolesta, mikä nopeuttaa sovelluksen toteuttamista ja vähentää tarvittavaa ohjelmakoodin määrää. Eräs valmiiksi ratkaistu ongelma on 3D-maailman kameran liikuttaminen. Toinen valmis ratkaisu on jo aiemmin mainittu poiminta, jota kontrollipisteet hyödyntävät. Poimintaa on havainnollistettu kuvassa 5.8, jossa yksi kontrollipisteistä on valittuna. Poiminta mahdollistaa myös hiiren havaitsemisen kontrollipisteen päällä ilman, että piste valitaan. Sovelluksessa tätä hyödynnettiin kontrollipisteen koordinaattien näyttämiseen, kuten myös kuvassa 5.8 on esitetty.



Kuva 5.8. *Poiminnan hyödyntäminen sovelluksessa.*

Eräs ongelma, jota Qt3D ei ratkaissut, liittyi geometristen muunnosten suorittamiseen. Sovelluksessa geometriset muunnokset suoritetaan hiirtä liikuttamalla ja hiiren liike määrittää muunnoksen suuruuden. Tilanteen selkein ongelma on, että hiirtä on mahdollista liikuttaa leveys- ja korkeussuunnassa, mutta ei syvyyssuunnassa. Ongelma ratkaistiin siten, että syvyyssuuntainen muuntaminen suoritetaan hiirellä korkeussuunnan mukaisesti liikuttamalla. Ratkaisu olisi riittävä sellaisenaan, jos 3D-maailman kameraa ei pystyisi liikuttamaan ja kamera katsoisi aina negatiivisen *z*-akselin suuntaan. Tällaista rajoitusta ei voida asettaa, koska se rajoittaisi käyrän suunnittelua merkittävästi. Koska kamera liikkuu, kameran ja 3D-maailman koordinaattiakselien suunnat eivät useimmissa tapauksissa ole samansuuntaiset. Tätä tilannetta on havainnollistettu kuvassa 5.9. Kuvassa on esitetty 3D-maailman kappaleen koordinaattiakselien positiiviset suunnat sekä edestä että takaa kuvattuna. Lisäksi on esitetty näyttöruutu, johon sijoitettu kamera esittää kameran koordinaatistoa näyttöruutuun nähden. Kuvasta voi havaita, että 3D-maailman ja kameran koordinaattiakselit eivät ole samansuuntaiset kaikissa tilanteissa.



Kuva 5.9. *Kameran ja 3D-maailman koordinaattiakselit.*

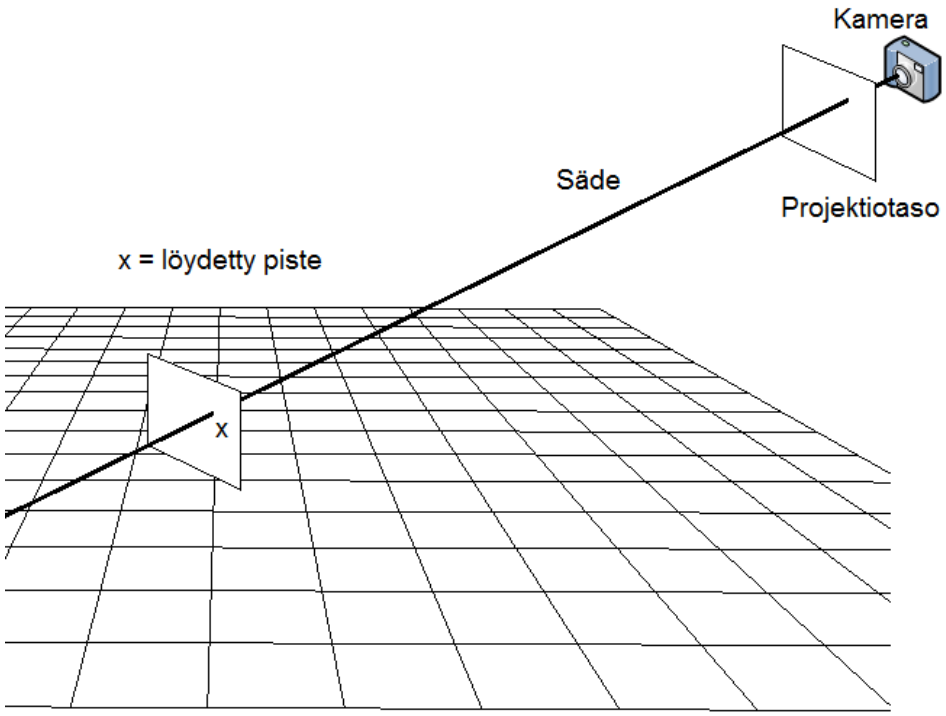
Koska geometristen muunnosten suuruus riippuu hiiren sijainnista näyttökoordinaatistossa, antaa muunnos erilaisen tuloksen kuvan 5.9 tapauksessa esimerkiksi positiivisen *x*-akselin suuntaan liikuttaessa. Näyttökoordinaatisto antaa aina positiivisen tuloksen, mutta takaapäin katsottaessa kappaleen tulisi liikkua negatiivisen *x*-akselin suuntaan. Vielä suurempi ongelma muodostuu, jos 3D-maailman kamera katsoo esimerkiksi positiivisen *x*-akselin suuntaan, jolloin näyttökoordinaatiston positiivinen *x*-akseli osoittaa 3D-maailman positiivisen *z*-akselin suuntaan. Ongelma voitaisiin periaatteessa ratkaista suorittamalla geometriset muunnokset kameran koordinaattiakselien suuntien perusteella. Ongelma ei silti ratkea kokonaan, koska kappaleeseen kohdistetut muunnokset muodostuvat kameran 3D-avaruuden asennon perusteella, jolloin kappale ei liiku 3D-maailman koordinaatiston mukaan.

Ongelma ratkaistiin pistetulon avulla. Pistetulo on vektorioperaatio, jossa kahden eri vektorin komponentit kerrotaan keskenään ja saadut tulot summataan (Puhakka 2008, s. 34). Jos vektorit ovat normalisoituja, eli yksikön mittaisia, pistetulo tuottaa tulokseksi vektorien välisen kulman kosinin (Puhakka 2008, s. 34-35). Jos laskettavat vektorit ovat täysin samansuuntaiset, pistetulon tulos on ykkönen, koska cos(0°) on myös ykkönen. Vastaavasti jos vektorit ovat kohtisuorassa, pistetulon tulos on nolla, koska cos(90°) on nolla. Näiden periaatteiden avulla on mahdollista päätellä koordinaattiakselien suhde toisiinsa.

Jotta pistetulo voidaan laskea, on ensin määritettävä vektorit. Qt3D:n kameraluokka tarjoaa kameran eteenpäin, ylös, ja oikealle osoittavat vektorit. Näiden vektorien ja 3D-maailmankoordinaatiston vastaavien vektorien pistetulon avulla on mahdollista päätellä, minkä koordinaattiakselin suuntaisesti geometriamuunnos suoritetaan. Päätelmä voidaan tehdä, jos lasketun pistetulon itseisarvo on tarpeeksi lähellä ykköstä. Pistetulon avulla saadaan myös muunnoksen suuruudelle oikea etumerkki. Kun pistetulon etumerkillä kerrotaan hiiren liikuttelun tuottama arvo, saadaan lopullinen arvo muunnoksen suorittamiseksi.

Yksi sovelluksen ominaisuuksista on, että uuden kontrollipisteen lisääminen on mahdollista tuplaklikkaamalla hiirtä 3D-näkymässä. Ongelmaksi muodostuu, että hiiritapahtuma suoritetaan näyttökoordinaatistossa, joka on kaksiulotteinen. Näiden koordi-

naattien perusteella ei voida suoraan päätellä, mitä pistettä näyttökoordinaatiston piste vastaa 3D-avaruudessa. Jotta 3D-avaruuden piste olisi mahdollista löytää, on löydettävä se 3D-avaruuden suora (tai säde), jolle käyttäjän valitsema piste kuuluu (Lamminsaari 2012 s. 11). Tekniikan perusajatusta on havainnollistettu kuvassa 5.10. Pisteen löytämisen ensimmäisessä vaiheessa tarvitaan käänteistä projektiomatriisia, joka on mahdollista saada Qt3D:n kameraluokalta (Lamminsaari 2012 s. 11). Sovellusohjelmoijan kannalta helpompi tapa on käyttää Qt3D:n 3D-näkymän mapPoint-funktiota, joka muuntaa näyttökoordinaatiston pisteen katsojankoordinaatistoon.



Kuva 5.10. *Pisteen löytäminen 3D-avaruudesta.*

Jotta 3D-avaruuden suora on mahdollista muodostaa, on katsojan koordinaatistoon muunnetun pisteen avulla laskettava suoran kulkema piste projektiotasolla. Projektiotason piste löydetään, kun kuljetaan katsojan koordinaatiston pisteen *x*-koordinaatin verran kameran oikealle osoittaman vektorin suuntaan, ja *y*-koordinaatin verran kameran ylös osoittavan vektorin suuntaan. Lopuksi kuljetaan vielä katsojan koordinaatistossa määritellyn projektiotason etäisyyden verran kameran katsomissuuntaan. Kun pisteen paikka projektiotasolla on laskettu, voidaan lähettää katsojan ja projektiotason pisteen suuntainen säde 3D-avaruuteen, kuten myös kuvassa 5.10 on esitetty. Pelkkä säteen lähettäminen vielä riitä lopullisen pisteen löytämiseksi, vaan 3D-avaruuteen on muodostettava taso, jonka lähetetty säde leikkaa. Säteen ja tason leikkauskohta on lopullinen 3D-avaruuden piste, johon uusi kontrollipiste luodaan. Jos säde ei leikkaa tasoa, uutta kontrollipistettä ei luoda. Sovelluksessa 3D-avaruuden taso sijoitettiin siten, että säde leikkaa aina tason. Taso sijoitettiin kameran katsomispisteeseen ja tason normaaliksi

valittiin tason keskipisteen ja kameran suuntainen vektori. Käytetty taso on siis vain projektiotason siirretty kopio. Tavallisesti sovellusohjelmoija joutuisi toteuttamaan tarvittavan toiminnallisuuden säteiden ja tasojen luomiseksi. Qt3D tarjoaa säteiden ja tasojen käsittelyyn valmiit QPlane3D- ja QRay3D-apuluokat.

6 TULOSTEN ARVIOINTI

Työn tavoitteena oli luoda sovellus, joka mahdollistaa kolmiulotteisten parametroitujen käyrien käsittelyn. Sovellukselle asetettiin kolme päätavoitetta; sovelluksella on mahdollista suunnitella uusia käyriä, muokata olemassa olevia käyriä, ja tallentaa käsitelty käyrä jatkokäsittelyn mahdollistavaan muotoon. Kaikki mainitut tavoitteet saavutettiin, mutta tavoitteita ei ollut mahdollista toteuttaa täydellisesti tämän työn rajoissa. Tässä luvussa tarkastellaan toteutuneita ominaisuuksia, ja niiden lopputulosta. Lopuksi pohditaan, mitä uusia ominaisuuksia sovellukseen voisi tulevaisuudessa toteuttaa.

6.1 Toteutuneiden ominaisuuksien tarkastelu

Sovellus mahdollistaa erilaisten käyrätyyppien käyttämisen, mikä lisää sovelluksen hyödyllisyyttä. Uuden käyrätyypin lisääminen on joustavaa, koska käyrät toteutetaan liitännäisinä. Tämän ratkaisun heikkoutena on, että sovellus on käyttökelvoton, jos yhtään käyräliitännäistä ei ole saatavilla. Sovelluskehityksen yhteydessä toteutettiin valmiiksi muutamia eri käyrätyyppejä, joten välitöntä ongelmaa liitännäisratkaisusta ei aiheudu. Tällä hetkellä suurin puute käyrätyyppien käsittelyssä on, että sovellus ei tue NURBS-tyyppisiä käyriä. Tuen lisääminen parantaisi sovelluksen hyödyllisyyttä entisestään.

Tällä hetkellä sovellus suorittaa käyrän laskemisen tietokoneen suorittimella. Laskentaprosessi hyödyntää rinnakkaisuutta, joten käyttöjärjestelmä todennäköisesti suorittaa laskennan toisella prosessoriytimellä, jos käytössä on moderni suoritin. Taulukossa 6.1 on esitetty muutamia laskenta-aikoja millisekunteina nopeimmasta hitaimpaan järjestettynä.

Taulukko 6.1. *Käyrien laskenta-aikoja millisekunteina mitattuna.*

Käyrätyyppi	**Käyräpisteitä**		
	700000	**1750000**	**17500000**
Kuutiollinen Bézier	31	70	516
Uniformi kuutiollinen B-spline	47	78	562
Catmull-Rom	47	93	782
Chaikinin käyrä	50	138	1000
N-asteinen Bézier (de Casteljau)	735	1766	17600
N-asteinen Bézier (Bernstein)	870	2100	20200

Chaikinin käyrälle ilmoitetut luvut ovat arvioita, koska tällä käyrätyypillä ei ollut mahdollista saavuttaa täsmälleen samaa käyräpistemäärää kuin muiden käyrätyyppien

tapauksessa. Mittauksessa käytettiin kymmentä kontrollipistettä ja käyrälle laskettiin 700000, 1750000 ja 17500000 pistettä. Pienillä käyräpistemäärillä ei saatu vertailukelpoisia tuloksia, joten nämä jätettiin huomioimatta. Mitatut käyräpistemäärät ovat niin suuria, että todellisessa tilanteessa näin suuria määriä ei yleensä käytetä. Luvut kuitenkin mahdollistavat eri käyrätyyppien nopeusvertailun. Taulukon 6.1 perusteella nopeimmin laskennasta suoriutuu kuutiollinen Bézier-käyrä, joka on myös kaikkein yksinkertaisin laskea. Hitaimmin laskennasta selviytyvät n-asteiset Bézier-käyrät, joiden laskenta on hidasta, koska laskennassa käytetään rekursiota ja raskaita matemaattisia operaatioita. Luvuista on mielenkiintoista havaita, että suurin osa käyristä selviytyy 700000 käyräpisteen laskennasta enintään 50 millisekunnissa. Näin ollen käyrän laskeminen tietokoneen suorittimella ei ole huonoin mahdollinen ratkaisu.

Uuden käyrän luominen on mahdollista kontrollipisteitä lisäämällä ja muokkaamalla. Kontrollipisteiden lisääminen on mahdollista sekä hiirellä että sovelluksen vasemmassa työkaluvalikossa sijaitsevalla Add-toiminnolla, joka mahdollistaa kontrollipisteen tarkan sijainnin asettamisen. Kontrollipisteitä on myös mahdollista poistaa valitsemalla yksi tai useampi kontrollipiste ja painamalla Delete-toimintoa. Kontrollipisteiden poistaminen on nopeaa ja selkeää, mutta uuden pisteen lisääminen ei aina ole yhtä tehokasta. Uuden kontrollipisteen lisääminen hiiren avulla on nopeaa, mutta lopputulos ei ole yhtä tarkka kuin Add-toiminnolla lisättäessä. Vastaavasti kontrollipisteen lisääminen Add-toiminnolla on hidasta ja jopa turhauttavaa, jos kaikki kontrollipisteet halutaan asettaa tarkasti.

Käyrän kontrollipisteille on mahdollista suorittaa geometrisia muunnoksia (kierto, skaalaus, siirto) ja käyrä muuttuu näiden muunnosten mukaiseksi. Muunnokset on mahdollista suorittaa joko kaikille kontrollipisteille tai vain osalle niistä. Merkittävin heikkous muunnosten suorittamisessa on, että muunnoksen suuruus määräytyy hiirtä liikuttamalla. Sovellus ei mahdollista muunnosten toteuttamista tarkoilla arvoilla, joten lopputulos on lähinnä silmämääräinen arvio halutusta lopputuloksesta.

Käyrä on mahdollista tallentaa XML-muotoiseen dokumenttiin, joka voidaan myöhemmin avata sovelluksessa, mikä mahdollistaa olemassa olevan käyrän muokkaamisen. XML on yleinen tiedostoformaatti, joten se on erinomainen valinta myös sovelluksen ulkopuolista käyttöä ajatellen. Toisaalta tallennusformaatti ei noudata mitään 3D-grafiikassa yleisesti käytettyä standardia. Tämä ei ole ongelma, koska sovellus mahdollistaa vienti- ja tuontiliitännäisten toteuttamisen, joten sovellukseen on helppo lisätä tuki uudelle tiedostoformaatille.

Yksi käytettävyyttä selvästi parantava tekijä on kumoa- ja toista-toimintojen toteuttaminen. Toimintojen ansiosta loppukäyttäjällä on mahdollisuus käyrän muokkauksessa tapahtuneiden virheiden korjaamiseen. Sovelluskehittäjän kannalta tilanne on monimutkaisempi, koska jokainen sovelluksen komento on toteutettava siten, että komento on mahdollista kumota ja toistaa. Toisaalta Qt tarjoaa tehokkaan työkalukokoelman sovellusohjelmoijan avuksi, kuten tarvittavat tietorakenteet.

Tämän työn perusteella voidaan todeta, että Qt on edelleen yksi parhaista sovelluskehyksistä C++-ohjelmointikielelle. Jo pelkästään Qt:n kattava toiminnallisuusvalikoi-

ma nopeuttaa sovelluskehitystä merkittävästi. Uuden Qt 5 -sarjan myötä on mahdollisuus tuottaa entistä tehokkaampaa ja joustavampaa sovelluskoodia sekä hyödyntää C++-kielen uusia ominaisuuksia. Qt3D on merkittävä lisäys Qt-komponentteihin. Vaikka komponentin kehitystyö on edelleen kesken, Qt3D tarjoaa tehokkaan ja nopean tavan tuottaa 3D-grafiikkaa. Toisaalta Qt3D:n rajapinta ei mahdollista valmiin toiminnallisuuden laajamittaista mukauttamista. Eräs esimerkki tästä on valmis kameraohjaus, johon sovellusohjelmoija ei voi vaikuttaa. Valmiin toteutuksen voi ylikirjoittaa sovellusohjelmoijan toteutuksella, mutta tällöin yhdessä sovellusprosessissa on kaksi lähes identtistä kameratoteutusta, joista toista ei käytetä. Koska komponentin kehitystyö on vielä kesken, edellä kuvattuihin ongelmiin saatetaan saada ratkaisu tulevissa versiojulkaisuissa.

6.2 Jatkokehitysajatuksia

Sovelluksen toteutuksen aikana heräsi muutamia jatkokehitysajatuksia. Näiden toteuttamista ei harkittu tämän työn toteuttamisen aikana, koska työn valmistuminen olisi viivästynyt. Kehitystyön aikana syntyneet ideat on kuitenkin syytä dokumentoida, jotta ne eivät unohdu ennen jatkokehitystyön aloittamista.

Nykyään grafiikkaohjaimia on mahdollista käyttää myös muuhun kuin 3D-grafiikan laskemiseen. Näin ollen jatkokehityksessä voitaisiin harkita laskentaprosessin siirtämistä grafiikkaohjaimelle. Erityisesti *n*-asteiset Bézier-käyrät voisivat hyötyä tästä ratkaisusta. Suurin kysymys grafiikkaohjaimen hyödyntämisessä on teknologian valinta. Grafiikkaohjaimella suoritettavaan laskentaan on saatavilla monia erilaisia tekniikoita, kuten OpenCL (Open Computing Language) ja CUDA (Compute Unified Device Architecture). Toisaalta suuren mittakaavan teknologioiden hyödyntäminen suhteellisen pienessä sovelluksessa vaikuttaa liioittelulta. Todennäköisesti paras ratkaisu on käyttää laskentaan tarkoitettuja sävyttimiä (engl. compute shader), jotka esiteltiin OpenGL-version 4.3 yhteydessä. Toistaiseksi AMD:n grafiikkaohjaimille ei ole saatavilla kyseistä tekniikkaa tukevia ajureita, joten toteutus on siirrettävä myöhempään ajankohtaan.

Käyrän käsittelyä olisi todennäköisesti mahdollista nopeuttaa toteuttamalla kaikki vektori- ja matriisioperaatiot esimerkiksi SSE:n avulla. Tällöin kaikki operaatiot olisi mahdollista toteuttaa kokonaisia vektoreita käsittelemällä. Qt:n tarjoamat matriisi- ja vektoriluokat eivät hyödynnä suorittimen käskykantalaajennuksia, joten käskykantalaajennuksia hyödyntävä toteutus jää sovellusohjelmoijan vastuulle.

Kontrollipisteiden käsittelyä koskevia jatkokehitysajatuksia on useita, joista tosin valtaosa liittyy nykyisten puutteiden korjaamiseen. Ensinnäkin kontrollipisteiden tarkka lisäys olisi syytä saada nopeammaksi, koska nykyinen toiminnallisuus aiheuttaa lähinnä turhautumista. Kontrollipisteiden lisääminen ei ole mahdollista olemassa olevien pisteiden joukkoon. Tämän korjaamiseksi voisi toteuttaa esimerkiksi alijakoon perustuvan ratkaisun, jossa kahden valitun kontrollipisteen väliin olisi mahdollisuus luoda uusia kontrollipisteitä suorittamalla alijakoa valittujen pisteiden välillä. Geometristen muun-

nosten toteuttaminen ei ole mahdollista tarkoilla arvoilla. Ongelma olisi korjattavissa siten, että muunnoksen aikana olisi mahdollisuus syöttää tarkka arvo.

Sovellus ei tue NURBS-käyriä, joten tämän tyyppisten käyrien lisääminen todennäköisesti parantaisi ohjelman hyödyllisyyttä. Ongelma ei tosin ole ratkaistavissa täysin suoraviivaisesti. NURBS-käyriä käsitellään neljässä ulottuvuudessa, mutta Qt3D on suunniteltu piirtämään kolmiulotteisia kappaleita. Ongelman ratkaisemiseksi olisi toteutettava uusi piirtorutiini, joka suorittaisi NURBS-käyrien piirron. Tämä ratkaisu vaatii myös sävyttimien toteuttamista toisin kuin muiden käyrätyyppien tapauksessa.

7 YHTEENVETO

Parametroitujen käyrien hyödyllisyys on havaittu laajalti erilaisessa tietokoneavusteisessa suunnittelussa ja grafiikan tuottamisessa. Parametroitu käyrä tarkoittaa määriteltyä polynomifunktiota, jonka avulla on mahdollista laskea kaikki käyrän pisteet parametrin *t* suhteen. Yleensä parametroitujen käyrien laskennassa käytetään kontrollipisteitä, joilla painotetaan käyrälle ennalta määriteltyjä kantapolynomeja. Kontrollipisteiden käytöllä on monia etuja. Esimerkiksi geometriset muunnokset voidaan suorittaa ainoastaan kontrollipisteille ja käyrä mukautuu näiden muunnosten mukaisesti, kun käyrä lasketaan. Kontrollipisteet mahdollistavat myös käyrän yksityiskohtaisemman muokkaamisen.

Erilaisia parametroituja käyriä on useita, ja tässä työssä esiteltiin niistä muutamaa. Esillä olivat Bézier-käyrä, Uniformi kuutiollinen B-spline-käyrä, Catmull-Rom-spline-käyrä ja Chaikinin käyrä. Jokainen käyrätyyppi sisältää yksilöllisiä ominaisuuksia, mutta joillain käyrätyypeillä on havaittavissa myös yhteneväisiä ominaisuuksia. Tyypillisesti erilaiset käyrätyypit soveltuvat erilaisiin tarkoituksiin paremmin kuin toiset käyrätyypit.

Diplomityön tavoitteena oli toteuttaa sovellus, joka helpottaa kolmiulotteisten parametroitujen käyrien suunnittelua. Sovellukselle asetettiin kolme päätavoitetta. Ensimmäiseksi tavoitteeksi määritettiin, että sovelluksella tulee pystyä luomaan uusia käyriä 3D-avaruuteen. Toisena tavoitteena asetettiin mahdollisuus olemassa olevien käyrien muokkaamiseen. Kolmanneksi tavoitteeksi määritettiin, että käyrä tulee pystyä tallentamaan jatkokäsittelyn mahdollistamaan muotoon.

Sovellus suorittaa 3D-avaruuden visualisoinnin OpenGL:n avulla. OpenGL on avoin 2D- ja 3D-piirtämiseen tarkoitettu rajapinta. Suurin hyöty OpenGL:n käytössä on, että OpenGL hyödyntää tietokoneen grafiikkaohjainta, joka on suunniteltu erityisesti 3D-avaruuden kappaleiden piirtämiseen. 3D-grafiikan liukuhihnan vaiheista OpenGL toteuttaa geometria- ja rasterointivaiheet. Moderni OpenGL-toteutus vaatii ohjelmoitavien sävyttimien käyttöä piirtämisessä, mikä toisaalta lisää sovellusohjelmoijan työmäärää, mutta mahdollistaa grafiikkaohjaimen tehokkaamman ja monimuotoisemman käytön.

Sovelluksen toteuttamisessa käytettiin Qt-sovelluskehystä. Qt on Digian omistama tuote, joka mahdollistaa sovelluskehityksen usealle eri käyttöjärjestelmälle ja laitealustalle ilman lähdekoodin muokkaamista. Tässä työssä hyödynnettiin uutta Qt 5 -sarjaa, joka tarjoaa merkittäviä uudistuksia vanhaan Qt 4 -sarjaan nähden. Yksi uusista komponenteista on Qt3D, joka tosin ei vielä kuulu Qt:n mukana toimitettaviin komponentteihin, mutta sisällyttämistä on suunniteltu versioon 5.1. Qt3D on suunniteltu erityisesti 3D-grafiikan tuottamiseen OpenGL:n avulla, ja helpottaa 3D-grafiikkaa hyödyntävän

sovelluksen toteuttamista merkittävästi. Tästä syystä myös toteutettu sovellus käyttää Qt3D-komponenttia.

Merkittävin sovelluksen toteutukseen vaikuttanut tekijä on liitännäisien käyttäminen. Kaikki sovelluksessa käytetyt käyrätyypit toteutetaan liitännäisinä, mikä mahdollistaa uuden käyrätyypin lisäämisen sovelluskoodiin koskematta. Liitännäisiä käytetään myös uusien tiedostoformaattien lisäämiseen. Merkittävänä käytettävyyttä lisäävänä tekijänä sovellukseen toteutettiin mahdollisuus kumota ja toistaa käyttäjän suorittamia komentoja. Tämän ansiosta käyrän muokkauksessa tapahtuneet virheet on helppo korjata. Sovelluksen käyttöliittymä toteutettiin Qt:n C++-pohjaisilla käyttöliittymäkomponenteilla. Qt 5 korostaa Qt Quick:n käyttämistä käyttöliittymätoteutuksessa, mutta tästä vaihtoehdosta oli luovuttava teknisistä ongelmista johtuen. 3D-grafiikka asetti omat haasteensa sovelluksen toteutukseen. Merkittävimmät haasteet liittyivät hiirellä suoritettaviin operaatioihin 3D-avaruudessa.

Sovellukselle asetetut tavoitteet saavutettiin, mutta tämän työn rajoissa ei ollut mahdollista toteuttaa täydellistä ratkaisua kaikkiin toimintoihin. Erityisesti käytettävyyden osalta jatkokehitys on aiheellista. Jatkokehitykseen siirretyistä ideoista eräs mielenkiintoisimmista on grafiikkaohjaimen hyödyntäminen käyrän laskennassa. Vaikka nykyinen, tietokoneen suoritinta hyödyntävä, laskentatapa ei ole huonoin mahdollinen, voisi grafiikkaohjaimen käyttö nopeuttaa laskentaa joidenkin käyrätyyppien osalta. Jatkokehitys on suotavaa myös käyrien käsittelyn osalta, koska nykyinen ratkaisu ei mahdollista neliulotteisten käyrien suunnittelua.

LÄHTEET

[#QTBUG-25643] Qt Quick 2 cannot co-exist with widgets [WWW]. Digia, Qt Project, Qt Bug Tracker. 3.2.2013, [Viitattu 6.2.2013]. Saatavissa: https://bugreports.qt-project.org/browse/QTBUG-25643.

Auto-Parallelization and Auto-Vectorization [WWW]. Microsoft, MSDN. [Viitattu 4.2.2013]. Saatavissa: http://msdn.microsoft.com/en-us/library/hh872235.aspx.

Blanchette, J. & Summerfield, M. 2006. C++ GUI Programming With Qt 4. 1st edition. USA, Prentice Hall Professional. 537 p.

Compute Shader [WWW]. OpenGL.org. 3.1.2013, [Viitattu 13.1.2013]. Saatavissa: http://www.opengl.org/wiki/Compute_Shader.

Digia ostaa koko Qt-kehitysympäristön Nokialta [WWW]. Digia, Helsinki ja Santa Clara, USA. 9.8.2012, [Viitattu 13.1.2013]. Saatavissa: http://www.digia.com/fi/Digia/Yritys/Uutiset/Digia-ostaa-koko-Qt-kehitysympariston-Nokialta/.

FAQ [WWW]. OpenGL.org. 29.11.2012, [Viitattu 12.1.2013]. Saatavissa: http://www.opengl.org/wiki/FAQ.

Farin, G. 2002. Curves and Surfaces for CAGD: A Practical Guide. 5th Edition. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. 520 p.

Fragment Shader [WWW]. OpenGL.org. 3.10.2012, [Viitattu 13.1.2013]. Saatavissa: http://www.opengl.org/wiki/Fragment_Shader.

Geometric Vector Functions (Windows) [WWW]. Microsoft, Windows Dev Center - Desktop. 14.11.2012, [Viitattu 7.1.2013]. Saatavissa: http://msdn.microsoft.com/en-us/library/windows/desktop/ee415667%28v=vs.85%29.aspx.

GPU Caps Viewer [WWW]. oZone3D.Net. 10.12.2010, [Viitattu 30.1.2013]. Saatavissa: http://www.ozone3d.net/gpu_caps_viewer/.

Graphics Pipeline (Windows) [WWW]. Microsoft, Windows Dev Center - Desktop. 28.11.2012, [Viitattu 9.1.2013]. Saatavissa: http://msdn.microsoft.com/en-us/library/windows/desktop/ff476882%28v=vs.85%29.aspx.

Joy, K.I. Chaikin's Algorithms for Curves [WWW]. Visualization and Graphics Research Group, Computer Science Department, University of California, Davis. 1999, [Viitatu 7.1.2013]. Saatavissa: http://www.idav.ucdavis.edu/education/CAGDNotes/Chaikins-Algorithm.pdf.

Khronos Releases OpenGL 4.3 Specification with Major Enhancements [WWW]. Khronos Group, Khronos Group Press Release. 6.8.2012, [Viitattu 11.1.2013]. Saatavissa: http://www.khronos.org/news/press/khronos-releases-opengl-4.3-specification-with-major-enhancements.

Knoll, L. Introducing Qt 5.0 [WWW]. Digia, Qt Blog. 19.12.2012, [Viitattu 14.1.2013]. Saatavissa: http://blog.qt.digia.com/blog/2012/12/19/qt-5-0/.

Lamminsaari, T. 2012. 3D-maailman kameran ohjaaminen kasvojen paikannuksen avulla. Diplomityö. Tampere. Tampereen teknillinen yliopisto. Tietotekniikan koulutusohjelma. 47 s.

McReynolds, T. & Blythe, D. 2005. Advanced Graphics Programming Using OpenGL. 1st edition. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. 672 p.

New Signal Slot Syntax Coming in Qt 5 [WWW]. Digia, Qt Project, Qt Wiki. 21.5.2012, [Viitattu 14.1.2013]. Saatavissa: http://qt-project.org/wiki/New_Signal_Slot_Syntax.

Nokia to acquire Trolltech to accelerate software strategy [WWW]. Nokia, Press Release. 28.1.2008, [Viitattu 13.1.2013]. Saatavissa: http://press.nokia.com/2008/01/28/nokia-to-acquire-trolltech-to-accelerate-software-strategy/.

OpenGL Platform & OS Implementations [WWW]. OpenGL.org. [Viitattu 12.1.2013]. Saatavissa: http://www.opengl.org/documentation/implementations/.

Puhakka, A. 2008. 3D-grafiikka. Helsinki, Talentum Media Oy. 453 s.

Qt Desktop Components [WWW]. Digia, Qt Project, Qt Wiki. 3.1.2013, [Viitattu 25.1.2013]. Saatavissa: http://qt-project.org/wiki/QtDesktopComponents.

Qt3D Reference Documentation [WWW]. Digia. [Viitattu 15.1.2013]. Saatavissa: http://qt.gitorious.org/qt/qt3d.

QtConcurrent 5.0: Qt Concurrent [WWW]. Digia, Qt Project, Documentation. [Viitattu 5.2.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtconcurrent/qtconcurrent-index.html.

QtCore 5.0: How to Create Qt Plugins [WWW]. Digia, Qt Project, Documentation. [Viitattu 16.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtcore/plugins-howto.html.

QtCore 5.0: Signals & Slots [WWW]. Digia, Qt Project, Documentation. [Viitattu 24.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html.

QtDoc 5.0: All Classes [WWW]. Digia, Qt Project, Documentation. [Viitattu 24.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtdoc/classes.html.

QtDoc 5.0: All Modules [WWW]. Digia, Qt Project, Documentation. [Viitattu 14.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtdoc/modules.html.

QtDoc 5.0: Deploying Plugins [WWW]. Digia, Qt Project, Documentation. [Viitattu 16.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtdoc/deployment-plugins.html.

QtDoc 5.0: Overview of Qt's Undo Framework [WWW]. Digia, Qt Project, Documentation. [Viitattu 1.2.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtdoc/qundo.html.

QtDoc 5.0: Qt 5.0 [WWW]. Digia, Qt Project, Documentation. [Viitattu 24.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtdoc/index.html.

QtDoc 5.0: Supported Platforms [WWW]. Digia, Qt Project, Documentation. [Viitattu 24.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtdoc/supported-platforms.html.

QtDoc 5.0: User Interfaces [WWW]. Digia, Qt Project, Documentation. [Viitattu 26.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtdoc/topics-ui.html.

QtDoc 5.0: What's New in Qt 5 [WWW]. Digia, Qt Project, Documentation. [Viitattu 14.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtdoc/qt5-intro.html.

QtGui 5.0: QWindow Class [WWW]. Digia, Qt Project, Documentation. [Viitattu 6.2.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtgui/qwindow.html.

QtQuick 5.0: QML Application Developer Resources [WWW]. Digia, Qt Project, Documentation. [Viitattu 25.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtquick/qtquick-applicationdevelopers.html.

QtQuick 5.0: Qt Quick [WWW]. Digia, Qt Project, Documentation. [Viitattu 25.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtquick/qtquick-index.html.

QtWidgets 5.0: Qt Style Sheets [WWW]. Digia, Qt Project, Documentation. [Viitattu 26.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtwidgets/stylesheet.html.

QtWidgets 5.0: Qt Widgets [WWW]. Digia, Qt Project, Documentation. [Viitattu 26.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtwidgets/qtwidgets-index.html.

QtWidgets 5.0: Widgets Tutorial [WWW]. Digia, Qt Project, Documentation. [Viitattu 26.1.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtwidgets/widgets-tutorial.html.

Rendering Pipeline Overview [WWW]. OpenGL.org. 2.11.2012, [Viitattu 9.1.2013]. Saatavissa: http://www.opengl.org/wiki/Rendering_Pipeline_Overview.

Salomon, D. 2006. Curves and Surfaces for Computer Graphics. New York, NY, USA, Springer Science+Business Media, Inc. 460 p.

Sederberg, T.W. An Introduction to B-Spline Curves [WWW]. 14.3.2005, [Viitattu 6.1.2013]. Saatavissa: http://tom.cs.byu.edu/~455/bs.pdf.

Segal, M. & Akeley, K. 1997. The OpenGL® Graphics System: A Specification (Version 1.1). Silicon Graphics, Inc. 194 s.

Sletta, G. Introducing QWidget::createWindowContainer() [WWW]. Digia, Qt Blog. 19.2.2013, [Viitattu 19.2.2013]. Saatavissa: http://blog.qt.digia.com/blog/2013/02/19/introducing-qwidgetcreatewindowcontainer/.

Spielberger, C. Uniform Cubic B-Splines [WWW]. 29.11.2010, [Viitattu 5.1.2013]. Saatavissa: http://www.cosy.sbg.ac.at/~cspiel/reports/splines.pdf.

Twigg, C. Catmull-Rom splines [WWW]. 11.3.2003, [Viitattu 7.1.2013]. Saatavissa: http://www.cs.cmu.edu/~fp/courses/graphics/asst5/catmullRom.pdf.

Vertex Shader [WWW]. OpenGL.org. 2.11.2012, [Viitattu 13.1.2013]. Saatavissa: http://www.opengl.org/wiki/Vertex_Shader.

Weisstein, E.W. Stirling's Approximation [WWW]. MathWorld--A Wolfram Web Resource. [Viitattu 5.1.2013]. Saatavissa: http://mathworld.wolfram.com/StirlingsApproximation.html.

Wright, R.S, Jr., Haemel, N., Sellers, G. & Lipchak, B. 2010. OpenGL SuperBible: Comprehensive Tutorial and Reference. 5th edition. USA, Addison-Wesley Professional. 1008 p.

XML Support in Qt [WWW]. Digia, Qt Project, Documentation. [Viitattu 5.2.2013]. Saatavissa: http://qt-project.org/doc/qt-5.0/qtdoc/topics-data-storage.html#xml-support-in-qt.

Yuksel, C., Schaefer, S. & Keyser, J. 2011. Parameterization and applications of Catmull-Rom curves. Computer-Aided Design, 43, pp. 747-755.