Ville Jokinen

Fraktaalien luominen tietokoneella

Tietotekniikan kandidaatintutkielma

4. toukokuuta 2016

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Ville Jokinen

Yhteystiedot: ville.o.jokinen@student.jyu.fi

Työn nimi: Fraktaalien luominen tietokoneella

Title in English: Computer generation of fractals

Työ: Kandidaatintutkielma

Sivumäärä: 30+0

Tiivistelmä: Tässä tutkielmassa esitellään muutamia tunnettuja menetelmiä Mandelbrotin ja Julian joukkojen ja iteroitujen funktiojärjestelmien luontiin.

Avainsanat: Fraktaalit, Mandelbrotin joukko, Julian joukko, iteroidut funktiojärjestelmät

Abstract: This paper presents some known methods for generating Mandelbrot and Julia sets and iterated function systems.

Keywords: Fractals, Mandelbrot set, Julia set, iterated function systems

Kuviot

Sisältö

1 Johdanto

Fraktaalit ovat monimutkaisia ja visuaalisesti mielenkiintoisia, mutta monesti yksinkertaisesti rakentuvia matemaattisia objekteja. Fraktaaleja voidaan hyödyntää tietokonegrafiikassa esimerkiksi monimutkaisten geometrioiden tai tekstuurien proseduraaliseen generoimiseen.

Tässä tutkielmassa käsitellään joitakin fraktaalien luontiin tietokoneella käytettyjä menetelmiä. Erityisesti tutkielmaan on valittu käsiteltäväksi Mandelbrotin ja Julian joukot ja iteroidut funktiojärjestelmät. Lisäksi kaikille valituille fraktaalityypeille esitellään tarkemmin kaksi eri menetelmää.

Kaikki tutkielmassa esiintyvät kuvat ovat tekijän itse luomia. Mandelbrotin ja Julian joukkojen kuvat 2, 3, 4, 5, 6, 7, 8 ja 9 on toteutettu GLSL ES -varjostimien avulla. Iteroitujen funktiojärjestelmien kuvat 1, 10 ja 11 on toteutettu JavaScript-kirjastoa Three.js käyttäen. Samalla menetelmistä tehtiin myös Common Lisp -implementaatiot.

Tutkielman luvussa 2 kerrotaan yleisesti fraktaaleista ja pohjustetaan lyhyesti kompleksilukuja. Luvussa 3 käsitellään muutamia menetelmiä Mandelbrotin ja Julian joukkojen piirtoon liittyen. Luvussa 4 esitellään joitakin iteroitujen funktiojärjestelmien piirtoon käytettyjä menetelmiä. Yhteenveto on luvussa 5.

2 Keskeiset käsitteet

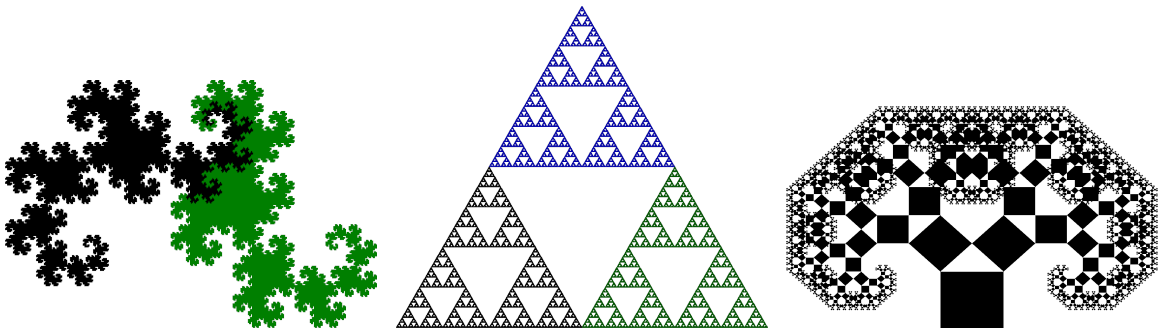
Tässä luvussa pohjustetaan jatkon kannalta oleelliset käsitteet eli määritellään lyhyesti fraktaalit ja kompleksiluvut.

2.1 Mikä on fraktaali

Alun perin sanan fraktaali keksijä Mandelbrot (1982) kutsui fraktaaleiksi joukkoja, joiden Hausdorffin dimensio on aidosti suurempi kuin joukon topologinen dimensio. Tämä määritelmä jätti ulkopuolelle monia joukkoja, jotka kuitenkin olisi voitu laskea fraktaaleiksi. Falconer (2014) määrittääkin fraktaalit seuraavasti:

- Fraktaalilla on yksityiskohtia kaikilla mielivaltaisilla väleillä.
- Fraktaali on liian epäsäännöllinen määriteltäväksi tavanomaisilla geometrisilla määritelmillä.
- Usein fraktaalissa esiintyy jonkintyyppistä itsesimiliaarisuutta.
- Yleensä fraktaalijoukon fraktaalidimensio on suurempi kuin sen topologinen dimensio.
- Fraktaali on usein määritelty yksinkertaisella tavalla, esimerkiksi rekursiivisesti.

Falconer (2014) huomauttaa lisäksi, että fraktaaleille ei varsinaisesti ole olemassa yksiselitteistä määritelmää. Eri tyyppisiä fraktaaleja löytyykin monia, ja tässä tutkielmassa mainitaan niistä vain muutama.

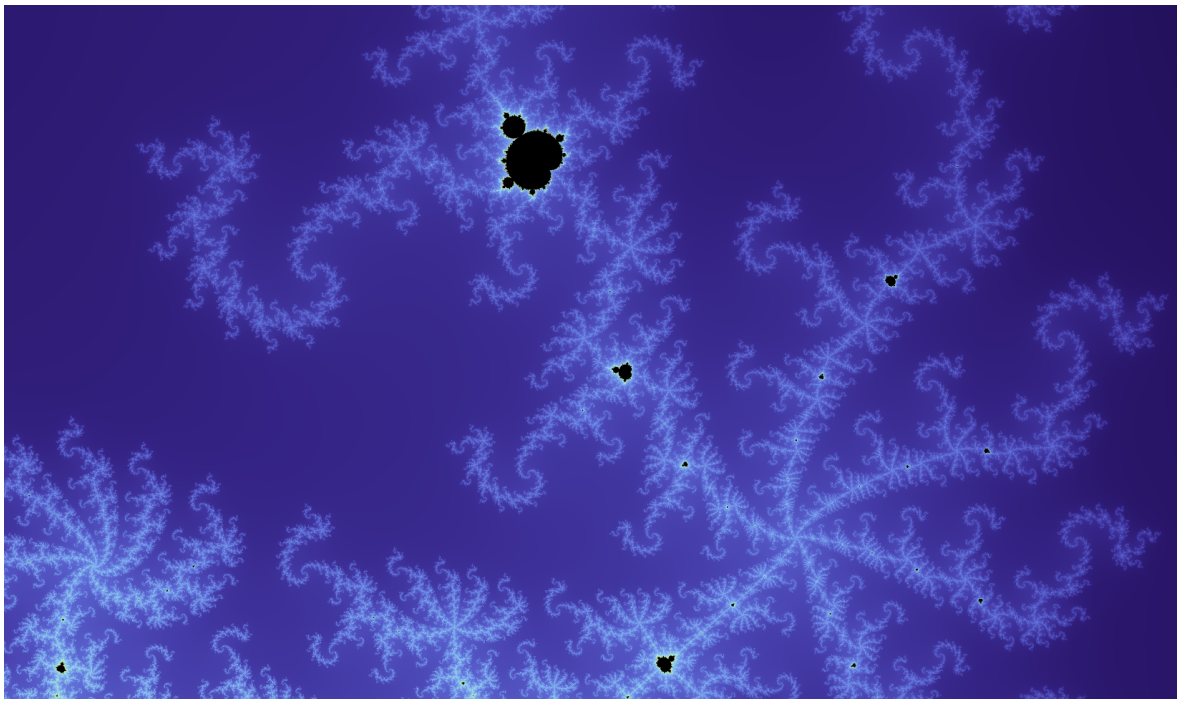


(a) (b) (c)

Kuvio 1: Iteroitujen funktiojärjestelmien menetelmillä piirrettyjä fraktaaleja.

Tutkielman kannalta edellä mainittujen eri dimensioiden määritelmiä ei tarvitse tuntea, mutta itsesimiliaarisuus on oleellinen ominaisuus iteroiduille funktiojärjestelmille. Jokin muoto on itsesimiliaarinen, jos sen kokonaismuoto on samanlainen, joko tarkasti tai suurinpiirtein, kuin kokonaismuodon osien muoto (Falconer 2014).

Esimerkiksi kuviossa 1 olevat Heighwayn lohikäärme (kuvio 1a), Sierpinskin kolmio (kuvio 1b) ja Pythagoraan puu (kuvio 1c) ovat yksinkertaisia fraktaaleja, jotka muodostuvat samaa alkumuotoa säännönmukaisesti toistettaessa. Niiden itsesimiliarisuus on siten selvästi havaittavissa. Kuviossa 2 on suurennos Mandelbrotin fraktaalista, jossa itsesimiliaarisuus ei ole yhtä selvästi havaittavaa kuin kuvion 1 fraktaaleissa. Kuitenkin kuvissa 1 ja 2 esiintyvät fraktaalit toteuttavat kaikki Falconerin käyttämistä viidestä ehdosta.



Kuvio 2: Eräs Mandelbrotin fraktaalin suurennos.

2.2 Kompleksiluvut

Tämän tutkielman luvussa 3 käsitellään erityisesti erästä yksinkertaisen kompleksifunktion muodostamaa fraktaalia (esim. kuviossa 2). Kompleksifunktioita käsiteltäessä tarvitaan kompleksilukuja, joilla on omat laskusääntönsä. Jatkossa kompleksilukujen joukkoa merkitään

kirjaimella $\mathbb{C}$, ja reaalilukujen joukkoa kirjaimella $\mathbb{R}$. Tutkielman tarpeisiin riittää ymmärtää kompleksilukujen koostuvan reaali- ja imaginaariosasta ja tuntea näille muutama laskusääntö.

Kompleksiluku z voidaan esittää muodossa $z = x + iy$, missä $x, y \in \mathbb{R}$ ja i on imaginaariyksikkö. Lisäksi kompleksiluvut voidaan myös esittää vektoreina

$$z = x + iy = (x, y). \tag{2.1}$$

Luku x on kompleksiluvun z reaaliosa ja y imaginaariosa. Tutkielmassa käytettävät kompleksilukujen laskusäännöt ovat tulo- ja summasääntö. Merkitään $z = a + ib$ ja $w = c + id$. Tällöin kompleksilukujen z ja w tulo on

$$zw = (a + ib)(c + id) = (ac - bd) + i(bc + ad), \tag{2.2}$$

ja summa on

$$z + w = (a + ib) + (c + id) = (a + c) + i(b + d). \tag{2.3}$$

Lisäksi tarvitaan kompleksiluvun, myös *moduuliksi* kutsuttu, itseisarvo

$$|z| = \sqrt{a^2 + b^2}. \tag{2.4}$$

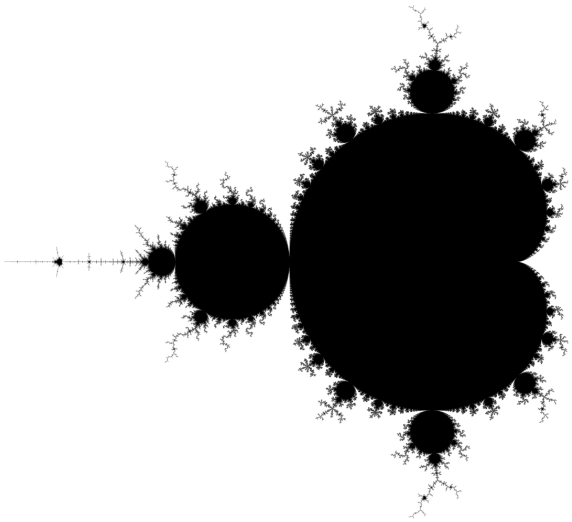
3 Mandelbrotin ja Julian joukot

Tässä luvussa käsitellään Mandelbrotin ja Julian joukkojen iterointia ja piirtämistä, erityisesti funktiolla

$$f_c : \mathbb{C} \to \mathbb{C},\ f_c(z) = z^2 + c,\ c \in \mathbb{C}. \tag{3.1}$$

Mahdollista on myös valita jokin muu funktio f_c, esimerkiksi $f_c(z) = ce^z$.

3.1 Mandelbrotin joukko



Kuvio 3: Funktion $z^2 + c$ Mandelbrotin joukko etäisyyden estimaattorimenetelmän avulla piirrettynä.

Mandelbrotin joukko voidaan määritellä joukkona

$$M = \{c \in \mathbb{C} : f_c^k(0) \not\to \infty \text{ kun } k \to \infty\}, \tag{3.2}$$

missä merkintä $f_c^k(0)$ tarkoittaa funktion f iteraatiota k kertaa. Toisin sanoen kaikki pisteet c joiden iteraatiot $\{f_c^k(0)\}_{k=1}^{\infty}$ ovat rajoitettuja, kuuluvat Mandelbrotin joukkoon. (Falconer 2014; Peitgen, Jürgens ja Saupe 2004; Peitgen 1988)

Esimerkiksi jos valitaan $c = 0 + i0$, niin

$$f_c^3(0) = f(f(f(f(0)))) = (((0^2 + c)^2 + c)^2 + c)^2 + c = 0 \tag{3.3}$$

nähdään siis selvästi, että $f_c^k(0) \to 0$ kun $k \to \infty$ eli $c \in M$. Yleisesti funktion f_c äärettömyyteen karkaaminen eli *hajaantuminen* voidaan havaita, jos jollekin $n \in \mathbb{N}$ pätee $|z_n| \geq 2$ (Peitgen, Jürgens ja Saupe 2004).Mandelbrotin joukon kokonaiskuva näkyy kuviossa 3.

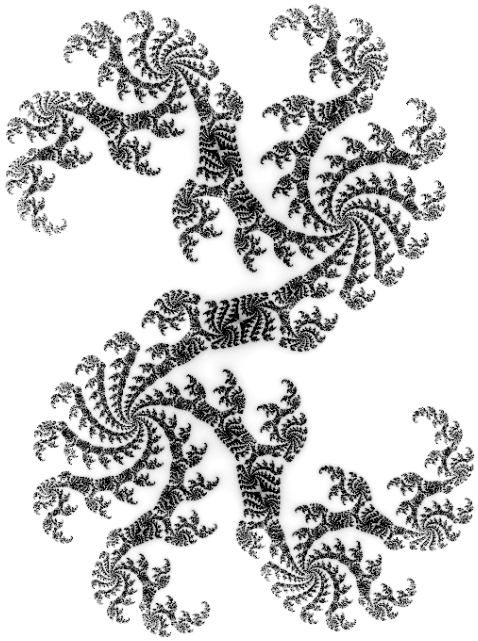
3.2 Julian joukko

Julian joukko voidaan määritellä monella tavalla, seuraavaksi esitellään yksi niistä. Olkoon $f : \mathbb{C} \to \mathbb{C}$ polynomi. Funktion f Julian joukko $J(f)$ on täytetyn Julian joukon $K(f)$ reuna, missä

$$K(f) = \{z \in \mathbb{C} : f^k(z) \not\to \infty\} \tag{3.4}$$

(Falconer 2014). Lyhyesti sanottuna jos $f^k(z)$ iteraatiot hajaantuvat äärettömyyteen kompleksiluku z ei kuulu Julian joukkoon. Kuten aikaisemminkin, k merkistee iteraatioita.

Julian joukolle kompleksiluvut z ja c alustetaan toisin kuin Mandelbrotin tapauksessa, missä c vastasi koordinaatiston pisteitä ja z:n alkuarvo oli 0. Julian joukolle c on jokin vakio ja z vastaa koordinaatiston pisteitä. (Falconer 2014; Peitgen, Jürgens ja Saupe 2004). Esimerkki funktion f_c Julian joukkosta eräällä c:n arvolla näkyy kuviossa 4.



Kuvio 4: Funktion $z^2 + c$ Julian joukko etäisyyden estimaattorimenetelmän avulla, kun $c = 0.36 + i0.09$.

3.3 Mandelbrotin ja Julian joukkojen luonti tietokoneella

Käytännössä tietokoneella toteutettavassa algoritmissä (ks. esim. algoritmin 2 funktio MANDELBROT) jokaiselle kuvan pisteelle iteroidaan funktion f_c arvoja valittuun rajaan N asti. Iterointi voidaan myös lopettaa, jos todetaan että f_c hajaantuu. (Drakopoulos, Mimikou ja Theoharis 2003; Peitgen, Jürgens ja Saupe 2004). Koska Mandelbrotin ja Julian joukkoihin kuuluvien pisteiden kohdalla funktio f_c ei hajaannu, täytyy niiden pisteiden kohdalla iteroida valittuun rajaan N asti. Toisaalta joukkojen ulkopuolella olevien pisteiden kohdalla ei funktiota f_c välttämättä tarvitse iteroida läheskään yhtä pitkään. Kuitenkin huonoimmassa tapauksessa joudutaan iteroimaan kuvan pisteiden määrä kertaa N. Tästä johtuen arvojen laskeminen Mandelbrotin ja Julian joukoille on erityisesti hidasta kuvattaessa alueita, jotka sisältävät suuria määriä joukkoihin kuuluvia pisteitä. Tosin menetelmästä riippuen voi myös joskus olla toivottavaa havaita mahdollisimman myöhään arvojen hajaantuminen tai olla havaitsematta sitä lainkaan. Tällöin myös joukkojen ulkopuolella olevien pisteiden laskeminen hidastuu.

Mandelbrotin tai Julian joukkoa funktiolle f_c piirrettäessä kannattaa valita yleensä kuvan koordinaattien väleiksi jokin väli I lähellä väliä $[-2,2]$, sillä funktiolla f_c Mandelbrotin ja Julian joukot saavat arvoja vain välillä $[-2,2]$ (Peitgen, Jürgens ja Saupe 2004). Toisaalta joskus voidaan myös haluta visualisoida joukkojen ympäristöä, jolloin voi olla hyödyllistä käyttää jotakin suurempaa väliä I.

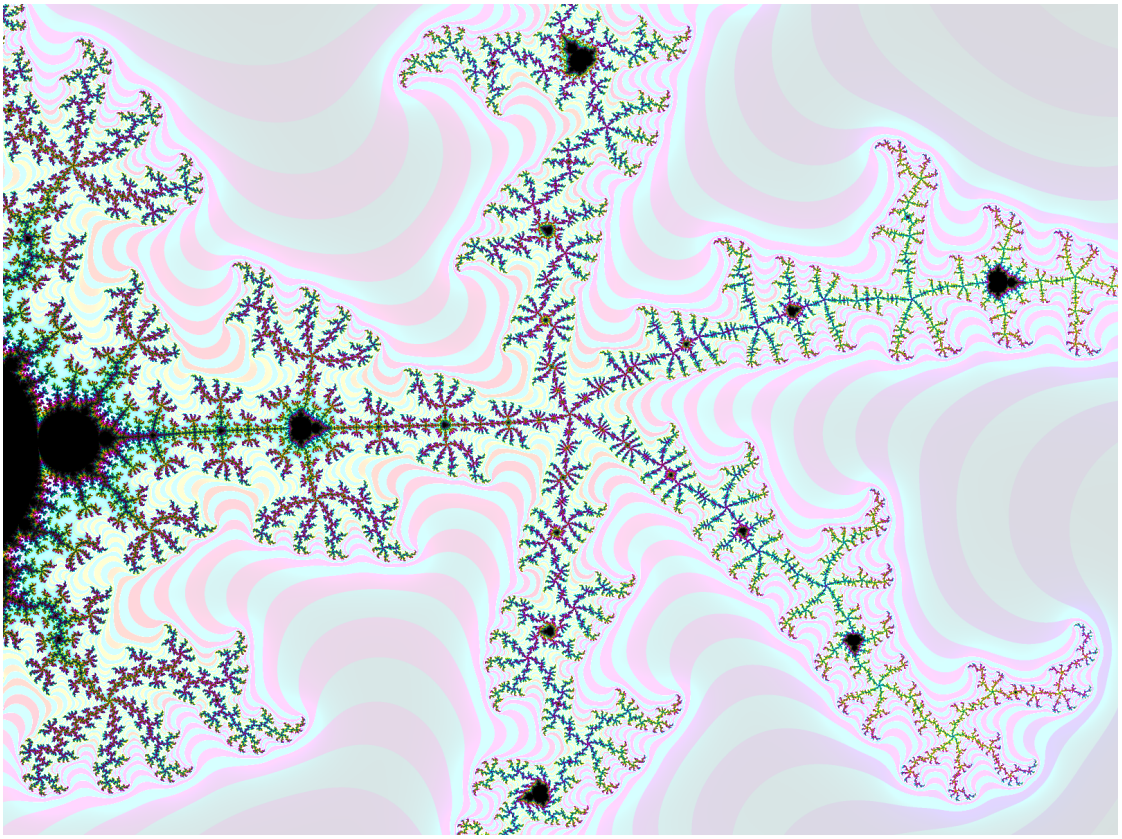
Algoritmiä Mandelbrotin tai Julian joukon piirtoon toteutettaessa halutaan pisteiden kuvautuvan joko johonkin kuvatiedostoon tai suoraan kuvaruudulle. Tällöin käytössä olevan koordinaatiston välit riippuvat kuvan korkeudesta ja leveydestä. Käytännössä pitää siis usein konvertoida koordinaatit kuvan koordinaatiston väliltä J välille I. (Peitgen 1988). Normalisointifunktio on esitettynä pseudokoodina algoritmissä 1. Lisäksi myös kuvasuhteella on vaikutusta, tällöin voidaan korjata kuvasuhde esimerkiksi kertomalla kaikki leveysakselilla olevat arvot jaolla $\frac{leveys}{korkeus}$.

Mandelbrotin tai Julian joukkoa iteroitaessa ongelmaksi voi tulla merkitsevien lukujen määrän kasvaminen, kuvatessa arvoja pienillä väleillä eli suurentaessa. Tällöin luvut menevät nopeasti niin pieniksi, että tavalliset liukuluvut eivät pysty esittämään niitä joko riittävän

tarkasti tai lainkaan. Tällöin voidaan käyttää menetelmiä suurempien liukulukujen simuloimiseen, esimerkiksi jakamalla yksittäiset luvut useaan muuttujaan (ks. Da Graça ja Defour 2006; Thall 2006). Tämä tosin hidastaa laskentanopeutta huomattavasti ja auttaa vain viivyttämään ongelman esiintymistä, sillä myös simuloituja liukulukuja käyttäessä raja tulee vastaan jossakin vaiheessa.

function NORMALISOIVÄLI(x,$minJ$, $maxJ$, $minI$, $maxI$)
    $return\ minI + (x - minJ) \cdot \frac{maxI - minI}{maxJ - minJ}$
end function

Algoritmi 1: Välillä J olevan pisteen x normalisointi välille I.



Kuvio 5: Mandelbrotin fraktaalin suurennos, etäisyyden estimaattori ja tasa-arvojoukkojen menetelmiä yhdistämällä.

3.4 Menetelmät Mandelbrotin joukon luontiin

Tässä tutkielmassa käsiteltävissä Mandelbrotin ja Julian joukkojen luontiin käytetyissä menetelmissä jokaisen kuvapisteen kohdalla ajetaan iterointifunktio, jonka tuloksen perusteellä päätetään kuvapisteen väri. Käsiteltävät menetelmät ovat Tasa-arvojoukkojen menetelmä (engl. *level set method*) ja etäisyyden estimaattorimenetelmän (engl. *distance estimator method*). Menetelmien eroja voidaan havaita kuvioissa 6 ja 7, jotka molemmat kuvaavat samaa kohtaa Mandelbrotin joukkosta. Lisäksi menetelmiä voidaan yhdistää eli käyttää samanaikaisesti, esimerkiksi kuvio 5 on luotu näin. Menetelmät ovat myös pseudokoodilla esitettyinä algoritmeissä 2 ja 4.

Tasa-arvojoukkojen menetelmä, jota voidaan kutsua myös pakoaika-algoritmiksi, on muunnelma reunan skannausmenetelmästä (engl. *boundary scanning method*) (Peitgen 1988; Drakopoulos, Mimikou ja Theoharis 2003). Algoritmissä 2 esitetään tämä menetelmä pseudokoodina. Aluksi menetelmässä asetetaan jokin lukuarvo N suurimmalle mahdolliselle määrälle iteraatioita, ja jokin sopiva lukuarvo B mahdolliseen aikaisempaan lopetukseen.

Iterointifunktiossa toimitaan kuten normaalisti Mandelbrotin joukkoa iteroitaessa, iteroinnin jälkeen palautetaan iterointien määrä i. Aluksi alustetaan z ja c joukon iteroinnin oletusarvoihin, lisäksi i:n arvoksi tulee nolla. Tämän jälkeen z:n arvo korvataan toistuvasti kompleksifunktion $z^2 + c$ arvolla ja i:n arvoa korotetaan yhdellä. Tätä jatketaan kunnes i saavuttaa arvon N tai z:n moduuli (kaava (2.4)) saavuttaa arvon B. Iteroinnin loputtua iterointifunktio palauttaa iterointien määrän ja tätä käytetään kuvapisteen väritykseen. (Peitgen 1988, sv. 188).

Tasa-arvo joukkoja voidaan värittää siten, että väritetään kuvattavan joukon ulkopuoliset pisteet, joille siis $i \neq N$, valitulla värillä jos i on parillinen luku. Esimerkki tästä näkyy Julian joukon kuviossa 8. Toisaalta kuviossa 6 väritykseen käytettiin suoraan iteraatioiden määrää.

Myös tasa-arvojoukkojen menetelmään sovellettavia väritysmenetelmiä löytyy, kuten binäärihajotusmenetelmän (engl. *Binary decomposition method*). Menetelmässä väritykseen ei suoranaisesti käytetä iterointien määrää, vaan sitä onko z:n imaginaariosa positiivinen vaiko negatiivinen. Julian joukon kuvioista 8 ja 9 voidaan havaita pintapuolisesti menetelmien eroja. (Peitgen 1988; Peitgen, Jürgens ja Saupe 2004)

```
N ← jokin riittävän suuri luku
B ← jokin luku jolle B > 0                                   ▷ Yleensä B ≥ 2
function MANDELBROT(x, y)
    z ← 0 + i0
    c ← x + iy
    i ← 0
    while i < N and |z| < B do                              ▷ Säännön (2.4) mukaan
        z ← z^2 + c                                         ▷ Lasketaan sääntöjä (2.2) ja (2.3) käyttäen
        i ← i + 1
    end while
    return i                                                ▷ Jos i = N, piste (x, y) kuuluu Mandelbrotin joukoon
end function
function MANDELBROTLSM(korkeus, leveys, ymin, ymax, xmin, xmax)
    for x ← 0, leveys do
        nx ← NormalisoiVäli(x, 0, leveys, xmin, xmax)
        for y ← 0, korkeus do
            ny ← NormalisoiVäli(y, 0, leveys, ymin, ymax)
            väri ← Mandelbrot(nx, ny)                       ▷ Voitaisiin jakaa N:llä jolloin 0 ≤ väri ≤ 1
            piirrä(x, y, väri)                              ▷ Tai kerää arvot ja palauta ne lopussa
        end for
    end for
end function
```

Algoritmi 2: Tasa-arvojoukkojen menetelmä Mandelbrotin joukolle funktiolla $z^2 + c$. (Peitgen 1988, sv. 188)

Etäisyyden estimaattorimenetelmä (ks. Peitgen 1988, sv. 196) tuottaa tarkempia kuvia kuin monet muut menetelmät (Drakopoulos, Mimikou ja Theoharis 2003). Menetelmässä toimitaan osittain samoin kuin tasa-arvojoukkojen menetelmässä. Algoritmeissä 3 ja 4 esitetään menetelmä pseudokoodina, kuvioissa 3 ja 7 näkyy Mandelbrotin joukko menetelmän avulla kuvattuna. Aluksi asetetaan alkuarvot N ja B samoin kun tasa-arvojoukkojen menetelmässä. Lisäksi asetetaan *kynnysluku*-muuttujan arvoksi jokin pieni luku, jonka avulla lasketaan arvo muuttujalle δ. Iterointifunktiossa ETÄISYYS muuttujat z ja c alustetaan samoin kuin

tasa-arvojoukkojen menetelmässä, mutta lisäksi alustetaan muuttuja dz. Mandelbrotin joukon tapauksessa dz alustetaan kompleksiluvulla $0+i0$. Tämän jälkeen iterointi tapahtuu samoin kuin tasa-arvojoukkojen menetelmässä, mutta aina aluksi korvataan muuttujan dz:n arvo kompleksifunktion $2z \cdot dz + 1$ tuloksella. Jos iteroinnin jälkeen z:n moduuli (kaava (2.4)) on suurempi kuin B, palautetaan funktiosta etäisyysfunktion tulos

$$log\left(|z|^2\right) \cdot \frac{|z|}{|dz|}, \tag{3.5}$$

muutoin palautetaan nolla. Iterointifunktion palautusarvo asetetaan *dist*-muuttujan arvoksi. Jos *dist* on pienempi kuin δ, niin kuvapiste väritetään. (Peitgen 1988, sv. 196)

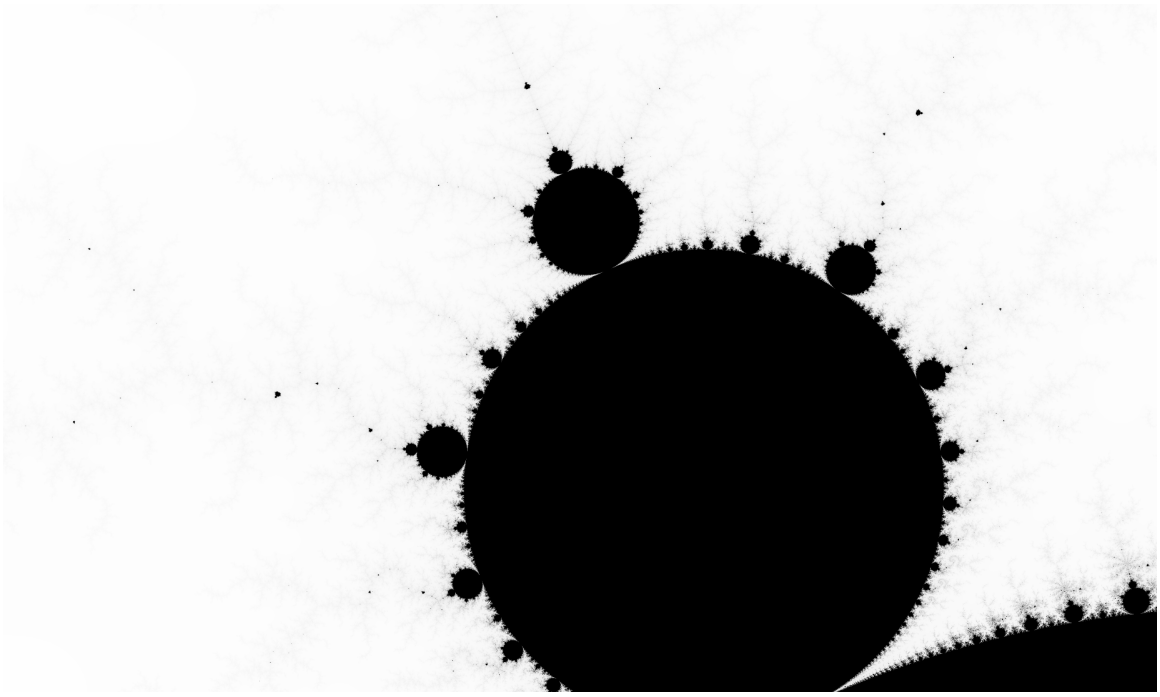
Myös paljon muita menetelmiä on olemassa, kuten esimerkiksi jatkuvan potentiaalin menetelmä (engl. *continuous potential method*) ja reunan skannausmenetelmä (ks. esim. Peitgen 1988).

```
N ← jokin riittävän suuri luku
B ← jokin luku jolle B > 0                          ▷ Yleensä B ≥ 2
function ETÄISYYS(x, y)
    z ← 0 + i0
    dz ← 0 + i0
    c ← x + iy
    i ← 0
    while i < N and |z| < B do                      ▷ Säännön (2.4) mukaan
        dz ← 2 · z · dz + 1                         ▷ Lasketaan sääntöä (2.2) käyttäen
        z ← z^2 + c                                 ▷ Lasketaan sääntöjä (2.2) ja (2.3) käyttäen
        i ← i + 1
    end while
    if |z| > B then
        return log(|z|^2) · |z|/|dz|                ▷ Lasketaan säännön (2.4) mukaan
    end if
    return 0
end function
```

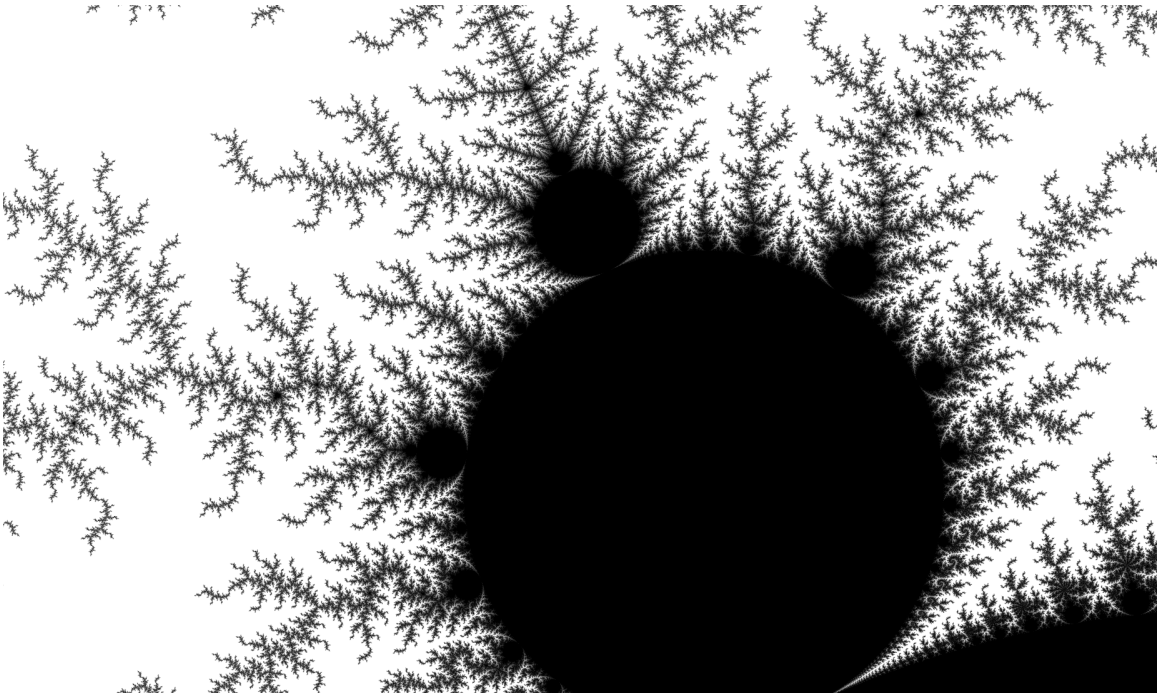
Algoritmi 3: Etäisyyden estimaattorimenetelmän (algoritmi 4) etäisyysfunktio. (Peitgen 1988, sv. 196)

```
kynnysluku ←jokin pieni luku suhteutettuna pistekokoon
function MANDELBROTDEM(korkeus, leveys, ymin, ymax, xmin, xmax)
    δ ← kynnysluku · (xmax−xmin)/leveys
    for x ← 0, leveys do
        nx ← NormalisoiVäli(x, 0, leveys, xmin, xmax)
        for y ← 0, korkeus do
            ny ← NormalisoiVäli(y, 0, leveys, ymin, ymax)
            dist ← Etäisyys(nx, ny)
            väri ← 0
            if dist < δ then
                väri ← 1
            end if
            piirrä(x, y, väri)
        end for
    end for
end function
```

Algoritmi 4: Etäisyyden estimaattorimenetelmä. ETÄISYYS-funktio näkyy algoritmissä 3. (Peitgen 1988, sv. 196)



Kuvio 6: Eräs Mandelbrotin fraktaalin suurennos tasa-arvojoukkojen menetelmää soveltamalla (vrt. kuvio 7). Väritykseen käytetty suoraan iteraatioiden määrää.



Kuvio 7: Eräs Mandelbrotin fraktaalin suurennos etäisyyden estimaattorimenetelmällä (vrt. kuvio 6).

3.5 Menetelmät Julian joukon luontiin

Tutkielmassa käsitellään erityisesti tasa-arvojoukkojen menetelmää ja etäisyyden estimaattorimenetelmää, jotka soveltuvat myös Mandelbrotin joukon luontiin. Löytyy myös muitakin menetelmiä, kuten esimerkiksi käänteinen iterointimenetelmä (engl. *inverse iteration method*) (ks. Devaney 1988, sv. 154). Lisäksi myös muut Mandelbrotin joukon luontii sopivat menetelmät käyvät Julian joukon luontii pienin muutoksin (ks. Peitgen 1988).

Tasa-arvojoukkojen menetelmää Julian joukolle soveltaessa riittää muuttaa vain muuttujien z ja c alkuarvoja Julian joukolle sopiviksi. Tämä on helposti nähtävissä vertailemalla algoritmejä 2 ja 5. Esimerkki menetelmän tuottamasta kuvasta näkyy kuviossa 8.

```
N ← jokin riittävän suuri luku
B ← jokin luku jolle B > 0                          ▷ Yleensä B ≥ 2
c ← jokin vakio
function JULIA(x, y, N, B, c)
    z ← x + iy
    i ← 0
    while i < N and |z| < B do                      ▷ Säännön (2.4) mukaan
        z ← z^2 + c                                 ▷ Lasketaan sääntöjä (2.2) ja (2.3) käyttäen
        i ← i + 1
    end while
    return i                                        ▷ Jos i = N, niin piste kuuluu Julian joukkoon
end function
```

Algoritmi 5: Funktion $z^2 + c$ Julian joukon iterointi yhdelle pisteelle (x, y). Menetelmässä käytetään algoritmin 2 MANDELBROTLSM-funktiota, korvaamalla MANDELBROT-funktio tämän algoritmin JULIA-funktiolla. (Peitgen 1988, sv. 187)

Etäisyyden estimaattorimenetelmä Julian joukolle toimii lähes samoin kuin Mandelbrotin joukolle. Koska Julian joukolle z ja c alustetaan eri tavalla, tehdään näin myös muuttujalle dz. Muuttuja dz alustetaan nyt kompleksiluvulla $1 + i \cdot 1$. Lisäksi Julian joukolle dz:n arvo iteroidaan kompleksifunktiolla $2z \cdot dz$ (Peitgen 1988). Eron näkee helposti myös vertailemalla algoritmejä 6 ja 4. Kuviossa 4 on esimerkki menetelmän tuottamasta kuvasta.

$N \leftarrow$ jokin riittävän suuri luku

$B \leftarrow$ jokin luku jolle $B > 0$ ▷ Yleensä $B \geq 2$

kynnysluku $\leftarrow$ kriittinen etäisyys joukosta pikseliyksikköinä

procedure ETÄISYYS(x, y)

    $z \leftarrow x + iy$

    $dz \leftarrow 1 + i \cdot 1$

    $c \leftarrow$ jokin vakio

    $i \leftarrow 0$

    while $i < N$ **and** $|z| < B$ **do** ▷ Säännön (2.4) mukaan

        $dz \leftarrow 2 \cdot z \cdot dz$ ▷ Lasketaan sääntöä (2.2) käyttäen

        $z \leftarrow z^2 + c$ ▷ Lasketaan sääntöjä (2.2) ja (2.3) käyttäen

        $i \leftarrow i + 1$

    end while

    if $|z| > B$ **then**

        return $\log\left(|z|^2\right) \cdot \frac{|z|}{|dz|}$ ▷ Lasketaan säännön (2.4) mukaan

    end if

    return 0

end procedure

Algoritmi 6: Funktion $z^2 + c$ Julian joukon etäisyyden estimaattorimenetelmän etäisyysfunktio. Vastaa algoritmin 4 ETÄISYYS-funktiota. (Peitgen 1988, sv. 199)

3.6 Menetelmien tehokkuus

Algoritmeistä 2, 4 ja 5 näkee helposti algoritmien aikavaativuuden riippuvan kuvan koosta ja valituista iterointiparametreista. Algoritmejä 4 ja 2 vertailemalla voidaan huomata, että etäisyyden estimaattorimenetelmän ja tasa-arvojoukkojen menetelmän erot ovat suhteellisen pieniä. Kuitenkin näin vertailemalla voidaan nähdä, että etäisyyden estimaattorimenetelmän täytyy olla hitaampi. Koska menetelmässä joudutaan iteroimaan kahta eri muuttujaa yhden sijasta.

Drakopoulos (2002) vertailee artikkelissaan algoritmejä Julian joukon renderöintiin. Vertailussa oli myös tässä tutkielmassa esiintyvät etäisyyden estimaattorimenetelmä ja tasa-

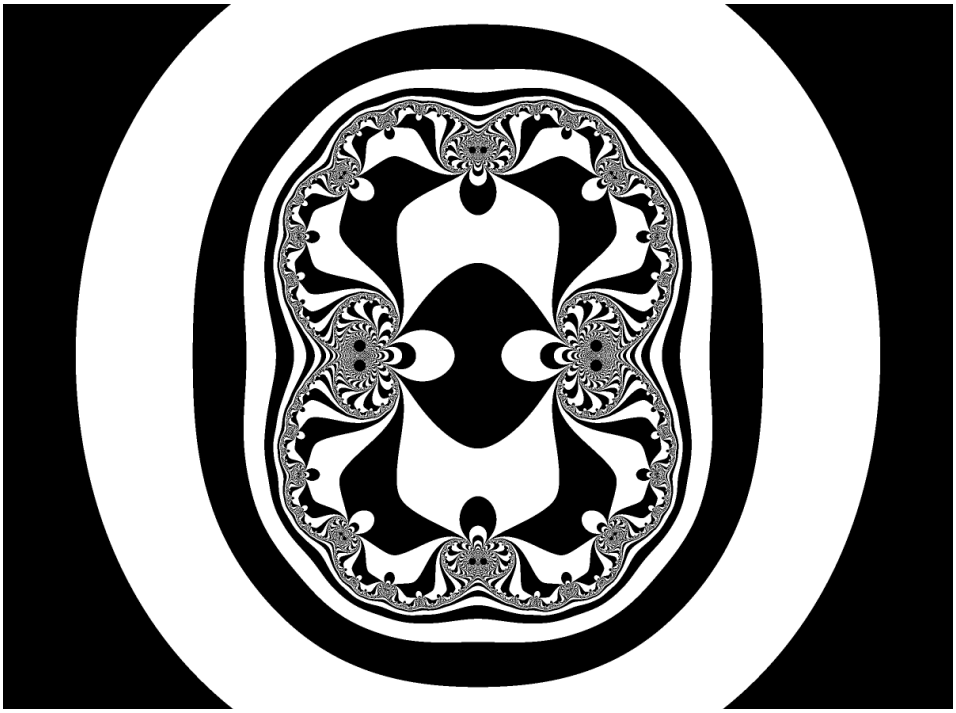
arvojoukkojen menetelmä. Muita artikkelissa vertailtuja algoritmejä ovat käänteinen iterointimenetelmä (engl. *inverse iteration method*), jatkuvan potentiaalin menetelmä (engl. *continuous potential method*) ja reunan skannausmenetelmä (engl. *boundary scanning method*). Lisäksi artikkelin vertailussa on reunan skannausmenetelmän ja käänteisen iterointimenetelmän muunnelmat.

Drakopoulos (2002) päätyi artikkelissaan siihen, että käänteisen iteraatiomenetelmän muunnelma oli kokonaistuloksien mukaan vertailluista menetelmistä paras Julian joukkojen renderöintiin. Toisaalta etäisyyden estimaattorimenetelmä tuotti kaikista tarkimpia tuloksia, mutta oli kaikista hitain menetelmä. Tasa-arvojoukkojen menetelmä sijoittui vertailussa loppupäähän tuotetun kuvan tarkkuuden suhteen, mutta oli kuitenkin kohtuullisen nopea. (Drakopoulos 2002). Tässä tutkielmassa käsiteltävät Julian joukon menetelmät ovat lähes vastaavia Mandelbrotin joukon menetelmien kanssa. Drakopoulosin artikkelin tulokset pätevät siten myös tässä tutkielmassa esiteltyihin Mandelbrotin joukon renderöintimenetelmiin.

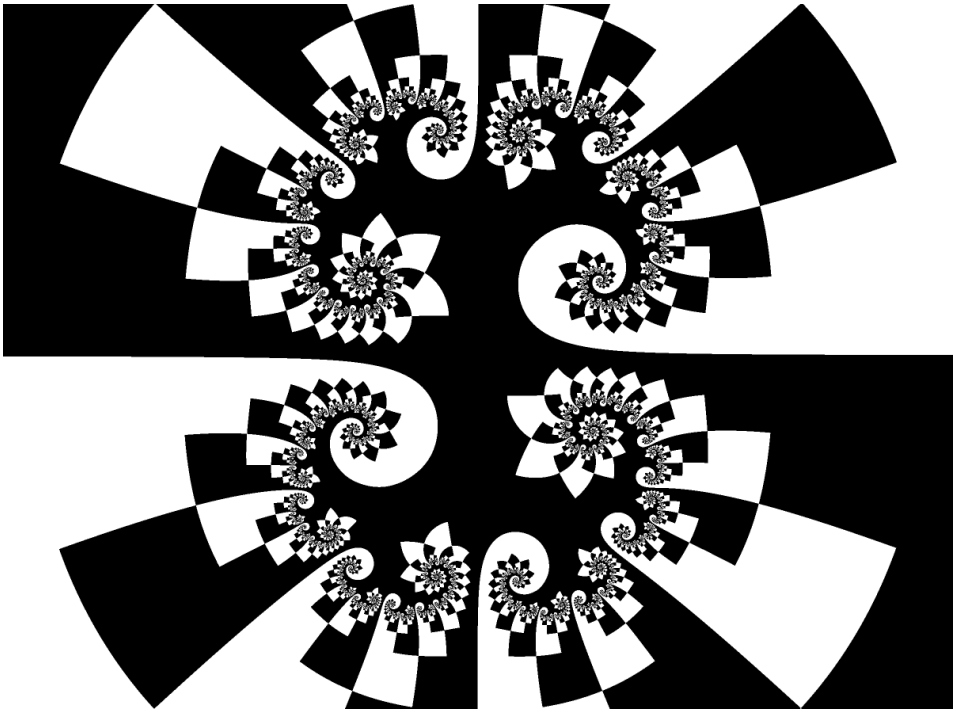
3.7 Rinnakkaistaminen

Mandelbrotin ja Julian joukkoja piirrettäessä voidaan kaikkien kuvattavien pisteiden väriarvot iteroida erikseen. Tämän ominaisuuden vuoksi, ainakin tässä tutkielmassa esiintyvien menetelmien, rinnakaistaminen erittäin helppoa. Kuvapisteiden iterointi voidaan jakaa monelle prosessorille esimerkiksi jakamalla kuva-alue osiin, jotka jaetaan eri prosessoreille iteroitaviksi. Rinnakkaistaminen lyhentää iterointii tarvittavaa aikaa merkittävästi, koska siten voidaan iteroida monta pistettä yhtäaikaisesti. (Drakopoulos, Mimikou ja Theoharis 2003)

Nykyään rinnakaistaminen voitaisiin toteuttaa vaikkapa grafiikkasuorittimien avulla GPGPU-rajapintoja, kuten OpenCL tai NVIDIA:n CUDA, hyödyntäen, sillä grafiikkasuorittimet soveltuvat erinomaisesti pienten toisistaan riippumattomien laskujen suorittamiseen rinnakkaisesti. Myös näytönohjainten OpenGL-rajapinnan fragmenttivarjostimia on mahdollista hyödyntää Mandelbrotin joukon rinnakaistamiseen, esimerkiksi kaikki tämän tutkielman Mandelbrotin ja Julian joukkoja esittävät kuvat ovat luotu näin. Rinnakkaistettuja menetelmiä kokeilemalla voidaan todeta, että nykyaikaisilla tietokoneilla pitäisi olla mahdollista renderöidä Mandelbrotin ja Julian joukkoja interaktiivisesti reaaliajassa.



Kuvio 8: Julian joukko tasa-arvojoukkojen menetelmällä. Funktiona $z^2 + c$, missä $c = 0.25112 + i0.0$.



Kuvio 9: Julian joukko binäärihajotusmenetelmällä. Funktiona $z^2 + c$, missä $c = 0.3535 - i0.01010$.

4 Iteroidut funktiojärjestelmät

Tässä luvussa käsitellään menetelmää itsesimiliaaristen fraktaalien (ks. luku 1) luomiseen, jota kutsutaan iteroiduiksi funktiojärjestelmiksi (engl. *iterated function systems*). Lisäksi joskus käytetään myös englanninkielistä nimikettä *multiple reduction copy machine* (Peitgen, Jürgens ja Saupe 2004). Iteroidut funktiojärjestelmät koostuvat joukosta muunnoskuvauksia $\{w_1, w_2, \ldots, w_n\}$. Muunnoskuvaus voidaan määritellä funktiona

$$w_i : \mathbb{R}^2 \to \mathbb{R}^2,\ w_i(\vec{x}) = w_i \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_i & b_i \\ c_i & d_i \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \end{pmatrix}. \tag{4.1}$$

Määritelmään voidaan myös lisätä joukko muunoskuvauksia vastaavia todennäköisyyksiä $\{p_1, p_2, \ldots, p_n\}$. Tarkasti ottaen $\mathbb{R}^2$ voi olla mikä tahansa suljettu n-dimensionaalisen Euklidisen avaruuden $\mathbb{R}^n$:n osajoukko (Falconer 2014). Tästä seuraa, että iteroidut funktiojärjestelmät soveltuvat myös kolmiulotteisten fraktaalien luomiseen. (Barnsley 1988; Peitgen, Jürgens ja Saupe 2004; Falconer 2014)

Kuvioissa 1b, 10 ja 11 käytetään Sierpinskin kolmion muunnoskuvausta $w_i : \mathbb{R}^2 \to \mathbb{R}^2$,

$$\begin{aligned} w_1(\vec{x}) &= \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \vec{x} \\ w_2(\vec{x}) &= \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \vec{x} + \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix} \\ w_3(\vec{x}) &= \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \vec{x} + \begin{pmatrix} \frac{1}{4} \\ \frac{\sqrt{3}}{4} \end{pmatrix}. \end{aligned} \tag{4.2}$$

Nämä muodostavat iteroidun funktiojärjestelmän

$$F = \bigcup_{i=1}^{3} w_i(F) = w_1(F) \cup w_2(F) \cup w_3(F). \tag{4.3}$$

Lisäksi Sierpinskin kolmiolle voidaan keksiä myös monia muita mahdollisia muunnoskuvauksia. (Peitgen, Jürgens ja Saupe 2004; Falconer 2014)

4.1 Menetelmät

Tässä tutkielmassa käsitellään tarkemmin stokastista iterointimenetelmää (engl. *random iteration algorithm*), josta käytetään myös englanninkielistä nimitystä *chaos game*, ja determinististä pisterekursiivista renderöintimenetelmää (engl. *point recursive rendering*). Pseudokoodi menetelmille löytyy algoritmeissä 7 ja 8.

Löytyy myös paljon muita, enemmän tai vähemmän, erilaisia menetelmiä (ks. esim. Nikiel 2007; Hepting, Prusinkiewicz ja Saupe 1990). Esimerkiksi vektorirekursiivinen renderöintimenetelmä (engl. *vector recursive rendering*), joka yleistää pisterekursiivisen renderöintimenetelmän vektoreille. Menetelmässä käytetään yhden pisteen sijaan kahta pistettä, jotka normalisoidaan. (Nikiel 2006, 2007). Algoritmiä tarkasteltaessa voidaan huomata, että yksinkertaistettu versio algoritmistä ilman normalisointia on helppo yleistää mielivaltaiselle määrälle pisteitä. Esimerkiksi kuvion 1c fraktaali on luotu tätä ominaisuutta hyödyntäen. Lisäksi on myös olemassa epälineaarisia iteroituja funktiojärjestelmiä, joista käytetään englanninkielistä nimikkettä *fractal flame*, joiden luonnissa voidaan soveltaa myös stokastista iterointimenetelmää (Lawlor 2012).

4.2 Stokastinen iterointimenetelmä

Stokastinen iterointimenetelmä perustuu muunnoskuvauksen iterointiin eli siihen, että yksittäinen muunnoskuvaus piirtää sitä iteroitaessa tietyn osan fraktaalista. Tällöin muunnoskuvauksia satunnaisesti valittaessa koko fraktaalin kuva saadaan usein hahmotettua nopeasti pienelläkin määräällä iteraatioita. Lisäksi koska käytetty muunnoskuvaus valitaan satunnaisesti seuraa, että saatu kuva on aina hieman erilainen. (Nikiel 2007, luku 5.1.1)

Menetelmässä (ks. myös 7) kiinnitetään aluksi käytetyn muunnoskuvauksen funktioiden määrä muuttujaan N. Seuraavaksi voidaan määritellä iteroinnille jokin minimiraja m. Tämä tarvitaan estämään mahdollisten piirrettävään fraktaaliin kuulumattomien pisteiden piirtäminen, jos aloituspiste on fraktaalin ulkopuolella. Usein riittää valita esimerkiksi $m = 20$ mutta jos mahdollista, on parempi valita aloituspiste siten, että se kuuluu piirrettävään fraktaaliin (Kwaśniewski 2010).

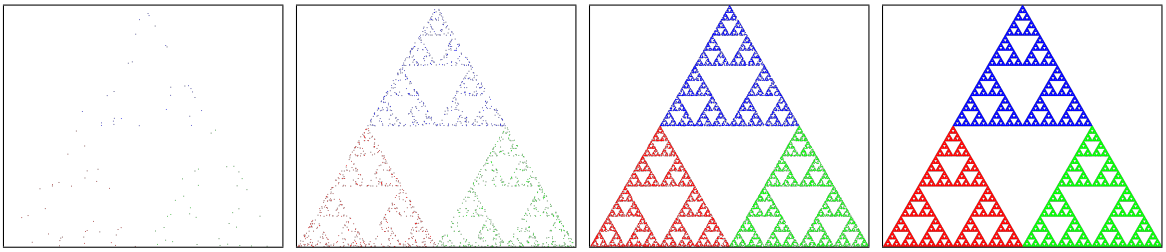
Lisäksi määritellään iteroinnille maksimiraja M, joka määrää kuvan tarkkuuden. Maksimiraja voidaan valita esimerkiksi kaavalla

$$M = korkeus \cdot leveys \cdot 8, \tag{4.4}$$

missä *korkeus* ja *leveys* ovat kuvan mitat pikseliyksikköinä, tosin myös muita tapoja valita maksimiraja löytyy (Nikiel 2007, luku 5.1.1). Ennen iterointia valitaan jokin piste q, joka voi periaattessa olla mikä tahansa, josta iterointi aloitetaan.

Iteroitaessa valitaan aina aluksi satunnainen luku k väliltä $0, 1, \ldots, n$, tämän avulla valitaan mitä muunnoskuvausta käytetään. Tämän jälkeen q:n arvo korvataan valitun muunnoskuvauksen tuloksella arvolla q. Seuraavaksi, jos iterointien määrä on yli minimirajan, piirretään piste q.

Kuviossa 10 näkyy muunnoskuvauksen 4.2 muodostama Sierpinskin kolmio eri iteraatiomäärillä, jonka pisteet on väritetty käytetyn muunnoskuvauksen alaindeksin mukaan. Muunnoskuvaus w_1 on väritetty punaisella, w_2 vihreällä ja w_3 sinisellä.



(i) $L = 200$ (ii) $L = 2000$ (iii) $L = 20000$ (iv) $L = 200000$

Kuvio 10: Sierpinskin kolmio stokastisella iterointimenetelmällä piirrettynä. L merkitsee iteraatioiden määrää.

$q \leftarrow$ jokin piste
$N \leftarrow$ muunnoskuvausten w määrä
$m \leftarrow$ minimimäärä iteraatioita
$M \leftarrow$ maksimimäärä iteraatioita
for $i \leftarrow 0, M$ **do**
    $k \leftarrow$ satunnainen luku väliltä $0, 1, \ldots, n$
    $q \leftarrow w_i(q)$
    if $i > m$ **then**
        $piirrä(q)$
    end if
end for

Algoritmi 7: Stokastinen iterointimenetelmä. (Nikiel 2007; Kwaśniewski 2010)

4.3 Pisterekursiivinen renderöintimenetelmä

Pisterekursiivinen renderöintimenetelmä iteroi alkupisteen systemaattisesti jokaisella muunnoskuvauksella. Tämän vuoksi saatu kuva on aina samanlainen eli menetelmä on deterministinen. Menetelmän pseudokoodi on algoritmissä 8. Kuviossa 11 näkyy menetelmän eteneminen Sierpinskin kolmion muunnoskuvaukselle (4.2) eri rekursiotasoilla. Kuvan pisteet on väritetty käytetyn muunnoskuvauksen (4.2) alaindeksin mukaan. Muunnoskuvaus w_1 on väritetty mustalla, w_2 vihreällä ja w_3 sinisellä.

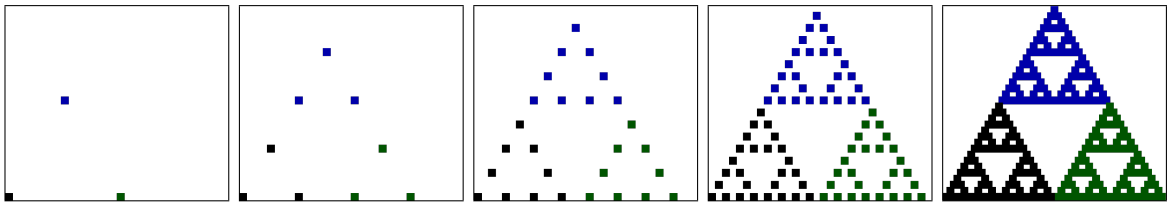
Aluksi menetelmässä kiinnitetään käytetyn muunnoskuvauksen funktioiden määrä muuttujaan N. Lisäksi valitaan rekursiolle jokin maksimisyvyys L. Hyvä arvio maksimisyvyydelle voidaan valita kaavalla

$$L = \frac{\log(korkeus \cdot leveys \cdot suurennus)}{\log(N)}, \tag{4.5}$$

missä *korkeus* ja *leveys* ovat kuvan mitat pikseliyksikköinä ja *suurennus* on kuvan suurennuskerroin (Nikiel 2007, luku 5.8.1). Ennen varsinaista rekursiota valitaan jokin piste q, voi olla mikä tahansa, josta aloitetaan.

Menetelmässä rekursio toimii siten, että jokaiselle muunnoskuvauksille $i = 1, \ldots, N$ lasketaan arvo q saadulla alkupisteellä. Jos rekursiotaso on L, päätetään kyseinen rekursiohaara

piirtämällä piste q. Jos rekursiotaso ei vielä ole L, siirrytään seuraavalle rekursiotasolle pisteelle q. Menetelmä päättyy kun kaikki muunnoskuvaukset on käyty läpi.



(i) $L = 0$ (ii) $L = 1$ (iii) $L = 2$ (iv) $L = 3$ (v) $L = 4$

Kuvio 11: Pisterekursiivisen renderöintimenetelmän avulla piirretty Sierpinskin kolmio rekursiotasoilla $0-4$. Pistekokoa suurennettu selkeyden vuoksi.

$q \leftarrow$ jokin piste
$N \leftarrow$ muunnoskuvausten w määrä
$L \leftarrow$ rekursion maksimisyvyys
procedure PRR(q, L)
    for $i \leftarrow 1, N$ **do**
        $q \leftarrow w_i(q)$
        if $L = 0$ **then**
            $piirrä(q)$
        else
            $prr(q, L-1)$
        end if
    end for
end procedure

Algoritmi 8: Pisterekursiivinen renderöintimenetelmä. (Nikiel 2007; Kwaśniewski 2010)

4.4 Menetelmien tehokkuus

Tähän tutkielmaan valitut iteroitujen funktiojärjestelmien menetelmät perustuvat valitun muunnoskuvauksen iterointiin valitulla alkupisteellä. Menetelmissä ei iteroida funktiota jokaiselle kuvan pisteelle, kuten Mandelbrotin ja Julian joukkoja iteroitaessa. Tästä seuraa että menetelmissä tarvitsee operoida, valitusta alkupisteestä riippuen, vain piirrettävään fraktaaliin

kuuluvilla pisteillä.

Haluttaessa kuvata hyvin tarkka kuva jostakin fraktaalista stokastista iterointimenetelmää käyttäen, tarvittavien iteraatioiden määrä kasvaa helposti hyvin suureksi. Pisterekursiivinen renderöintimenetelmä on tarkan kuvan piirtämiseen parempi kuin stokastinen iterointimenetelmä, sillä tarvittava iteraatioiden määrä on pienempi (Nikiel 2007, luku 5.13.1).

Valituista menetelmistä ainakin pisterekursiivinen renderöintimenetelmä on rinnakkaistettavissa GPGPU-rajapintoja hyödyntäen, mikä nopeuttaa renderöintiä kyseistä menetelmää käytettäessä (Kwaśniewski 2010). Lisäksi epälineaaristen iteroitujen funktiojärjestelmien piirtoon ja reaaliaikaiseen animointiin on olemassa rinnakkaistettu grafiikkasuorittimia hyödyntävä menetelmä, joka perustuu deterministiseen iterointimenetelmään (Lawlor 2012).

5 Yhteenveto

Tässä tutkielmassa käsiteltiin Mandelbrotin ja Julian joukkojen ja iteroitujen funktiojärjestelmien tietokoneella luomiseen käytettyjä menetelmiä. Kirjallisuudesta löytyi paljon menetelmiä, joista monet olivat lähinnä pieniä muunnelmia muista menetelmistä. Löydetyistä menetelmistä valittiin muutamia, jotka esiteltiin tutkielmassa tarkemmin.

Tutkielmassa esiteltiin Mandelbrotin ja Julian joukoille tasa-arvojoukkojen menetelmä ja etäisyyden estimaattorimentelmä, jotka todettiin olevan helppo rinnakkaistaa. Iteroiduille funktiojärjestelmille esiteltiin stokastinen iterointimenetelmä ja pisterekursiivinen renderöintimentelmä. Lisäksi valittujen menetelmien tehokkuutta kommentoitiin lyhyesti.

Voidaan todeta, että fraktaalien luominen on nopeutunut huomattavasti tietokoneiden, lähivuosina etenkin grafiikkasuorittimien, kehittyessä, sillä laskentatehon puute on olennainen ongelma fraktaaleja piirettäessä. Menetelmien rinnakkaistaminen on usein helppo keino tehostaa käytettävissä olevan laskentatehon hyödyntämistä, ja näin nopeuttaa fraktaalien luomiseen käytettyjä menetelmiä. Lisäksi GPGPU-rajapinnat tuovat tähän lisää mahdollisuuksia.

Lähteet

Barnsley, Michael F. 1988. “Fractal modeling of real world images”. Teoksessa *The Science of Fractal Images,* 219–242. New York: Springer-Verlag. doi:10.1007/978-1-4612-3784-6_5.

Da Graça, Guillaume, ja David Defour. 2006. “Implementation of float-float operators on graphics hardware”. *CoRR* abs/cs/0603115. arXiv: cs/0603115.

Devaney, Robert L. 1988. “Fractal patterns arising in chaotic dynamical systems”. Teoksessa *The Science of Fractal Images,* 137–168. New York: Springer-Verlag. doi:10.1007/978-1-4612-3784-6_3.

Drakopoulos, V. 2002. “Comparing rendering methods for Julia sets”. Toimittanut Václav Skala. *Journal of WSCG* 10 (1–2): 155–161. HDL: 11025/5973.

Drakopoulos, Vassileios, N. Mimikou ja Theoharis Theoharis. 2003. “An overview of parallel visualisation methods for Mandelbrot and Julia sets”. *Computers & Graphics* 27 (4): 635–646. doi:10.1016/S0097-8493(03)00106-7.

Falconer, Kenneth. 2014. *Fractal Geometry: Mathematical Foundations and Applications.* Chichester: John Wiley & Sons. ISBN: 978-1-119-94239-9.

Hepting, Daryl, Przemyslaw Prusinkiewicz ja Dietmar Saupe. 1990. “Rendering methods for iterated function systems”. Teoksessa *Proceedings of the 1st IFIP Conference on Fractals in the Fundamental and Applied Sciences,* toimittanut Heinz-Otto Peitgen, José Marques Henriques ja Luís Filipe Penedo, 183–224. North-Holland, Amsterdam, kesäkuu. ISBN: 978-0-444-88757-3.

Kwaśniewski, Bartosz. 2010. “CUDA Based Parallel Version Of Point Recursive Rendering Algorithm Of IFS Attractor”. Teoksessa *Proceedings of the XII International PhD Workshop (OWD 2010),* 51–56. Lokakuu. ISBN: 83-922242-7-2.

Lawlor, Orion Sky. 2012. “GPU-accelerated rendering of unbounded nonlinear iterated function system fixed points”. *ISRN Computer Graphics* 2012. doi:10.5402/2012/825782.

Mandelbrot, Benoit B. 1982. *The fractal geometry of nature.* New York: W. H. Freeman / Co. ISBN: 978-0716711865.

Nikiel, Slawomir. 2006. “Integration of iterated function systems and vector graphics for aesthetics”. *Computers & Graphics* 30 (2): 277–283. doi:10.1016/j.cag.2006.01.002.

Nikiel, Slawomir. 2007. *Iterated function systems for real-time image synthesis.* London: Springer-Verlag. doi:10.1007/1-84628-686-7.

Peitgen, Heinz-Otto. 1988. “Fantastic deterministic fractals”. Teoksessa *The Science of Fractal Images,* 169–218. New York: Springer-Verlag. doi:10.1007/978-1-4612-3784-6_4.

Peitgen, Heinz-Otto, Hartmut Jürgens ja Dietmar Saupe. 2004. *Chaos and fractals: new frontiers of science.* New York: Springer-Verlag. doi:10.1007/b97624.

Thall, Andrew. 2006. “Extended-precision Floating-point Numbers for GPU Computation”. Teoksessa *ACM SIGGRAPH 2006 Research Posters.* SIGGRAPH ’06. Boston: ACM. doi:10.1145/1179622.1179682.