Riemannin kuvauksen numeerinen approksimoiminen Zipper-algoritmilla

Tatu Leinonen

Perustieteiden korkeakoulu

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 27.5.2019.

Työn valvoja

Prof. Nuutti Hyvönen

Työn ohjaaja

TkT Harri Hakula

Aalto-yliopisto
Perustieteiden korkeakoulu

Aalto-yliopisto
Perustieteiden korkeakoulu

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Tatu Leinonen

Työn nimi Riemannin kuvauksen numeerinen approksimoiminen Zipper-algoritmilla

Koulutusohjelma Teknillinen fysiikka ja matematiikka

Pääaine Applied Mathematics **Pääaineen koodi** SCI3053

Työn valvoja Prof. Nuutti Hyvönen

Työn ohjaaja TkT Harri Hakula

Päivämäärä 27.5.2019 **Sivumäärä** 41 **Kieli** Suomi

Tiivistelmä

Konformikuvaukset ovat kulmien suunnat ja suuruudet säilyttäviä kuvauksia. Fysikaalisista suureista esimerkiksi sähkömagneettinen kenttä ja virtaukset ovat konformisia, joten konformikuvausten teoriaa tutkimalla voidaan kehittää ratkaisumenetelmiä käytännön ongelmiin. Eräs perustavanlaatuisimmista konformikuvauksia koskevista väittämistä on Riemannin kuvauslause, jonka mukaan yhdesti yhtenäisiltä (”reiättömiltä”) tasoalueilta on olemassa konformikuvaus yksikkökiekolle.

Riemannin kuvauslauseen toteuttavaa kuvausta kutsutaan Riemannin kuvaukseksi. Sillä on suljettu muoto vain harvoissa poikkeustapauksissa, joten ainoa luotettava tapa keino on ratkaista se numeerisesti. Tämä työ käsittelee kahta Riemannin kuvauksen ratkaisevaa algoritmia, Circlepackia ja Zipperiä. Työhön kuuluu myös Zipperin implementaatio MATLAB-kielellä, jonka toiminta selostetaan yksityiskohtaisesti.

Avainsanat Konformikuvaus, Riemannin kuvaus, MATLAB, Zipper, ympyräpakkaus, numeerinen menetelmä

Aalto-yliopisto
Perustieteiden
korkeakoulu

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Tatu Leinonen

Title Numeric approximation of Riemann mappings with the Zipper algorithm

Degree programme Engineering Physics and Mathematics

Major Applied mathematics **Code of major** SCI3053

Supervisor Prof. Nuutti Hyvönen

Advisor DSc (Tech.) Harri Hakula

Date 27.5.2019 **Number of pages** 41 **Language** Finnish

Abstract

Conformal mappings are those that preserve the orientation and magnitude of angles. Such physical quantities as electromagnetic field and flows are conformal, so the theory of conformal mappings can be applied to solving practical problems. One of the most fundamental claims regarding conformal mappings is the Riemann mapping theorem. It states that all simply connected regions (informally, regions with no holes) can be conformally mapped onto the unit disk.

A mapping that satisfies the Riemann mapping theorem is called a Riemann mapping. They can be expressed in a closed form only in rare exceptions, so the only reliable way to solve them is by numerics. This thesis describes two algorithms capable of solving Riemann mappings, Circlepack and Zipper. An implementation of Zipper, written for MATLAB, is also presented and its functionality explained in detail.

Keywords Conformal mapping, Riemann mapping, MATLAB, Zipper, circle packing, numeric method

Esipuhe

Tieteestä sanotaan, että kauemmaksi näkee nousemalla seisomaan jättiläisten hartioille. Lausahdus ei kuitenkaan kerro, miten sinne kiivetään. Auttajia on ollut enemmän kuin olisi mahdollista muistaa mainita, ja toivon joskus auttavani jonkun kiipeämisessä omalle jättiläiselleen.

Suuret kiitokset Harri Hakulalle ohjauksesta ja loputtomasta kärsivällisyydestä. Kiitos myös Nuutti Hyvöselle työn valvomisesta ja muusta tuesta sekä Antti Rasilalle aiheeseen perehdyttämisestä.

Vain muutamia mainitakseni, kiitos myös

Ninalle,

Lauralle,

Marja-Tertulle ja Aarolle,

Räihille,

Windioille,

Karustoille,

Harri V:lle ja matematiikan laitoksen henkilökunnalle,

Johannalle, Juholle, Hannulle ja muille fyysikoille,

Karlille, Makelle ja muille Olarilaisille,

Eerolle ja muille Espoon Taidon aktiiveille,

Miialle ja muille Koaloille, sekä

Topille, Idalle ja muille koirille.

Pikku-Huopalahti, 27.5.2019

Tatu Leinonen

Sisältö

1 Johdanto

Konformikuvaukset ovat kulmien suunnat ja suuruudet säilyttäviä kuvauksia. Kun kaksi toisensa leikkaavaa polkua kuvataan konformisesti, leikkauskulmat lähtö- ja kuvajoukossa ovat samansuuntaiset ja -suuruiset.

Konformikuvausten historia alkaa varhaisesta antiikista. Yhdisteet tasokuvion siirroista, pyörityksistä ja tasaisista skaalauksista olivat ilmiselvästi konformisia; [Car98] kutsuu niitä euklidisiksi suurennoksiksi. Ensimmäinen epätriviaalina pidettävä konformikuvaus on stereografinen projektio, jota tiedettiin käytetyn taivaankappaleiden liikkeiden seuraamiseen. Ptolemaios (n. 100-170) kirjoitti aiheesta parhaiten säilyneen teoksen, mutta viitteitä projektion ymmärryksestä on jo häntä edeltävältä ajalta [SB07]. Ei ole varmaa tietoa, tunnettiinko stereografisen projektion konformisuus tuona aikana, mutta käyttötarkoituksensa se toteutti ilmankin.

Seuraava konformikuvaus sen sijaan oli työkalu, jonka toimivuus perustui nimenomaan konformisuuteen. Flaamilainen kartantekijä Gerardus Mercator julkaisi vuonna 1569 teoksensa, joka esitti maailman uudenlaisessa karttaprojektiossa. Navigaatio merellä perustui tuolloin kulmien tarkkaan mittaukseen, jolloin Mercatorin konformisesta kartasta saattoi määrittää suuntansa ja sijaintinsa entistä helpommin. Mercator ei itse tiennyt projektioonsa liittyviä matemaattisia kaavoja [VR14], vaan tuotti karttansa käyttäen taulukoita ja kolmiulotteisia maailman malleja – oman aikansa numeerisia menetelmiä.

Käytännön sanelemista sovelluksista päästiin abstraktimpaan tarkasteluun, kun kompleksianalyysi alkoi kehittyä. Yksi perustavimmista konformikuvauksia koskevista tuloksista on Riemannin kuvauslause, jonka mukaan kaikki yhdesti yhtenäiset (”reiättömät”) tasoalueet voidaan kuvata konformisesti avoimelle yksikkökiekolle. Lause ottaa kantaa vain olemassaoloon, ei konstruktioon, ja kuvauksella on suljettu muoto vain harvoissa poikkeustapauksissa. Luotettavimmaksi keinoksi jää siis Riemannin kuvauksen ratkaiseminen numeerisesti.

Yhdesti yhtenäinen tapaus on kuitenkin vasta alkua. Riemannin kuvauslauseen alkuasetelma voidaan yleistää monenlaisiin tason osajoukkoihin tai jopa topologisten pintojen kuvaamiseen toisilleen. Nämä kuitenkin vaativat Riemannin kuvauslauseesta huomattavasti poikkeavaa lähestymistapaa, ja ratkaisuja pulmaan kehitetään yhä. Paras menetelmä riippuu usein sovelluksen sanelemasta geometriasta, tarkkuudesta ja nopeudesta.

Yleisimpiä sovelluksia ovat tietokonegrafiikka, useat fysikaaliset mallit ja tietenkin myös aiemmin mainittu kartografia. Konformikuvaukset ovat myös tärkeä osa lääketieteellistä aivokuvantamista. Fysiikassa moni kompleksiarvoinen suure noudattaa konformikuvausten ominaisuuksia, jolloin tehtävän voi muuntaa helpompaan geometriaan tai sen ratkaisuavaruutta voidaan rajata. Tällaisia suureita ovat esimerkiksi sähkömagneettinen kenttä ja virtaukset [Gan08], [Olv18].

Yleistyksiinsä verrattuna Riemannin kuvauslause on paljon perusteellisemmin tutkittu, ehkä jopa jossakin mielessä ”ratkaistu”. Siitä huolimatta tämä työ käsittelee kahta Riemannin kuvauslauseen ratkaisevaa algoritmia ja esittää näiden mahdolliset seuraukset yleistyksiä koskien.

Luvussa kolme käsitellään näistä ensimmäistä, Circlepackia. Sen syntytarinaan

liittyy geometrinen intuitio, joka kasvaa ensin konjektuuriksi ja sen jälkeen suureksi joukoksi matemaattisia faktoja. Tämä vuonna 1985 esitelty konjektuuri koski ympyräpakkausten suppenemista Riemannin kuvausta kohti, ja sen jälkeen algoritmin on todistettu toimivan mitä erilaisimmille topologisille pinnoille. Pakkaukset toimivat myös analogiana diskreetteihin analyyttisiin kompleksifunktioihin. Menetelmää on tutkittu aktiivisesti tähän päivään saakka, eikä syyttä.

Luvussa neljä esitellään Zipperin toimintaa ja implementointia. Zipper on kuin Circlepackin vastakohta: Ei geometrisia konjektuureja, vain pitkä konformikuvausten yhdiste. Siinä missä Circlepack sovittaa ympyröitä yhteen paikallisesti odottaen globaalia suppenemista, Zipper siirtää kerralla koko puolitasoa ja kuvaa monikulmion reunaa sivu kerrallaan reaaliakselille. Vastakohta se on myös käyttötarkoitusten laajuudessa, sillä Riemannin kuvauksen lisäksi sillä ei voi ratkaista mitään muuta – näin käy, kun algoritmi iteroi monikulmion reunaa ja käytännössä kaikissa yleistyksissä reunoja on enemmän kuin yksi.

Ehkä juuri huonosta yleistyvyydestään johtuen Zipper on jäänyt unohduksiin, eikä alkuperäistä Fortran-versiota [Mar] uudempia ohjelmia ole helppo löytää. Juliakielellä kirjoitettu paketti [Watb], jonka kehitys on aloitettu vasta vuonna 2014, on yksi harvoista. Sen ohelle toivoisi myös numeeriseen laskentaan erikoistuneen ohjelmointikielen implementaation, ja tämä puute inspiroi työn kirjoittajan ohjelmoimaan oman Zipper-toteutuksensa MATLAB:lle [Lei19b]. Kirjoittaja ohjelmoi työtä varten myös Circlepack-toteutuksen [Lei19a], mutta pääpaino on Zipperissä sekä työhön että ohjelmointiin käytetyn panoksen osalta. Implementaatiota käsittelevässä neljännen luvun osuudessa erityishuomiota annetaan yksityiskohdille, joita lähdemateriaalissa ei ole täysin selitetty ja jotka ilmenivät ohjelmaa kirjoitettaessa ongelmina.

Viidennessä luvussa annetaan esimerkkejä konformikuvausten testitapauksista. Luku ei käsittele kokeiden tuloksia, vaan tavoitteena on koota testaamisessa huomioitavia asioita. Kirjoittaja on ohjelmoidessaan ajanut luvussa kuvatun kaltaisia testejä, mutta kuvausmenetelmien välinen vertailu ei vaikuttanut mielekkäältä, joten viisaammaksi katsottiin rajata numeerista tarkkuutta ja suoritusnopeutta koskevat vertailut pois. Konformikuvauksiin liittyvää tutkimusta tekevä lukija toivottavasti onnistuu luvun ansiosta paremmin itse rakentamissaan testeissä.

2 Esitiedot

Konformisuus on geometrinen ominaisuus, mutta sen tarkastelu pelkän klassisen geometrian keinoin osoittautuisi nopeasti kömpelöksi. Monet geometriset ominaisuudet kuitenkin saadaan ilmaistua tehokkaammin kompleksilukuja ja -funktioita käyttämällä. Nykyaikaiset konformikuvauksiin liittyvät käsitteet ja metodit käytännössä vaativatkin kompleksianalyysin hyödyntämistä.

Oletamme lukijan tuntevan kompleksianalyysiin liittyvät perusteet. Luvussa esitellään kahdelle seuraavalle luvulle yhteisiä esitietoja sekä työssä käytettäviä merkintöjä ja määritelmiä. Lähteenä on käytetty [Rud87]:a.

2.1 Kompleksiluvut

Luku z on *kompleksiluku*, jos se on esitettävissä muodossa

$$z = x + iy, \qquad x, y \in \mathbb{R}. \tag{1}$$

Kompleksilukujen joukkoa merkitään symbolilla $\mathbb{C}$ ja viittaamme siihen myös *kompleksitasona*. Yhtälön 1 luvut x ja y ovat z:n *reaaliosa* ja *imaginaariosa*, ja *imaginaariyksikölle* i pätee $i^2 = -1$.

Kompleksiluvun reaali- ja imaginaariosaan viitataan funktioilla

$$\begin{aligned} \mathfrak{Re} &: \mathbb{C} \mapsto \mathbb{R} \qquad \mathfrak{Re}\,(x+iy) = x \\ \mathfrak{Im} &: \mathbb{C} \mapsto \mathbb{R} \qquad \mathfrak{Im}\,(x+iy) = y. \end{aligned} \tag{2}$$

Kompleksiluvun *konjugaatti* $\overline{z}$ määritellään sen reaali- ja imaginaariosan kautta

$$\mathfrak{Re}\,\overline{z} = \mathfrak{Re}\,z, \qquad \mathfrak{Im}\,\overline{z} = -\mathfrak{Im}\,z. \tag{3}$$

Kompleksiluvun *itseisarvo* on

$$|z| = \sqrt{(\mathfrak{Re}\,z)^2 + (\mathfrak{Im}\,z)^2}. \tag{4}$$

Itseisarvolle pätee myös $|z|^2 = z\overline{z}$.

Kompleksifunktio on funktio, jonka maalijoukko on $\mathbb{C}$. Funktio $f : \mathbb{C} \mapsto \mathbb{C}$ on *holomorfinen pisteessä* z_0, jos sillä on sarjamuotoinen esitys pisteessä z_0; f on *holomorfinen*, jos se on holomorfinen kaikissa lähtöjoukkonsa pisteissä.

Esimerkki 2.1 (Eksponenttifunktio)**.** Kompleksinen *eksponenttifunktio* $\exp(z)$, tai e^z, määritellään

$$\exp(z) = \sum_{n \geq 0} \frac{z^n}{n!}. \tag{5}$$

Eksponenttifunktion toimintaa havainnollistaa Eulerin kaavaksi kutsuttu yhtälö

$$e^{x+iy} = e^x \left(\cos y + i \sin y\right). \tag{6}$$

Funktio on koko kompleksitasossa holomorfinen ja nollasta poikkeava.

Kompleksifunktion *derivaatan* määritelmä ja merkintätapa on hyvin samanlainen kuin reaalisen:

$$f'(z) = \frac{df}{dz} = \lim_{z \to z_0} \frac{f(z) - f(z_0)}{z - z_0}. \tag{7}$$

Nollasta poikkeavilla kompleksiluvuilla on *suunta*

$$A(z) = \frac{z}{|z|}. \tag{8}$$

Suunnallisilla kompleksiluvuilla on myös *argumentti* eli *suuntakulma*. Argumentti $\arg z$ toteuttaa yhtälön

$$e^{i \arg z} = A(z), \tag{9}$$

ja kontekstista riippuen saatamme vaatia siltä yksikäsitteisyyttä. Tämä onnistuu esimerkiksi määrittämällä $\arg z$:n kuuluvan välille $[0, 2\pi)$ tai $(-\pi, \pi]$.

Sanomme kompleksifunktion f säilyttävän kulmat z_0:ssa jos

$$\lim_{r \to 0} e^{-i\theta} A \left[f(z_0 + re^{i\theta}) - f(z_0) \right], \quad r > 0. \tag{10}$$

Kulmien säilyminen tarkoittaa sekä niiden suuntien että suuruuksien säilymistä. Mikäli f on holomorfinen z_0:ssa ja $f'(z_0) \neq 0$, se säilyttää kulmat z_0:ssa:

Todistus. Yleisyyttä menettämättä asetetaan $z_0 = 0$ ja $f(z_0) = 0$. Jos $f'(0) = a \neq 0$, niin

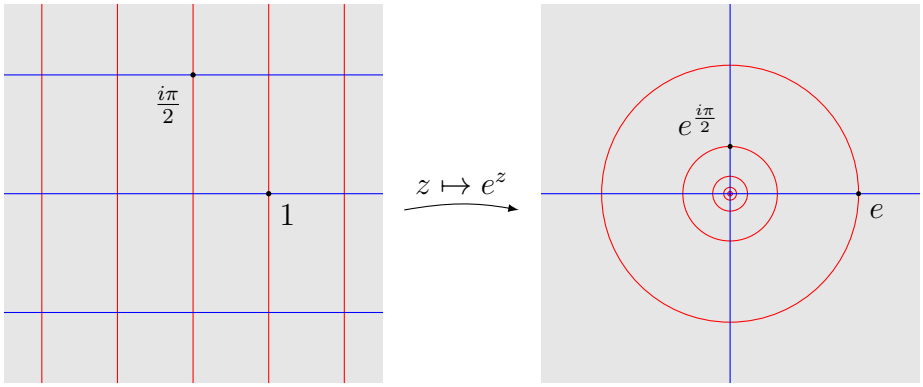
$$e^{-i\theta} A \left[f\left(re^{i\theta}\right) \right] = \frac{e^{i\theta} f\left(re^{-i\theta}\right)}{|f\left(re^{i\theta}\right)|} \to \frac{a}{|a|}. \tag{11}$$

□

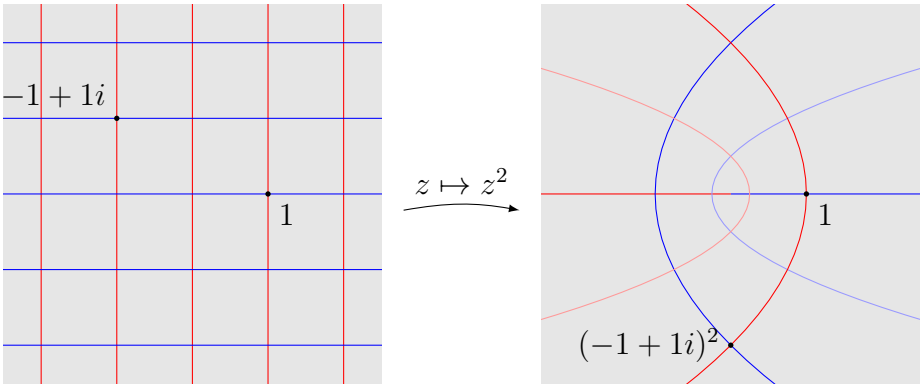
Näin ollen holomorfinen f, jonka derivaatta on kaikkialla nollasta poikkeava, säilyttää kulmat kaikkialla. Kutsumme tällaista funktiota *konformikuvaukseksi*. Yhdistetyn funktion derivaatan avulla on helppoa todistaa konformikuvausten yhdisteen olevan konforminen, mahdollisesti lähtö- ja maalijoukkojen rajaamista edellyttäen.

Esimerkki 2.2. Reaalisen eksponenttifunktion derivaatta on eksponenttifunktio itse. Kompleksisessa tapauksessa voidaan derivoida eksponenttifunktion sarjaesitystä ja havaita saman pätevän. Eksponenttifunktio on siis holomorfinen, ja koska $e^z \neq 0$ koko kompleksitasossa, se on myös konforminen.

Esimerkki 2.3. Kompleksifunktio $z \mapsto z^2$ on holomorfinen, mutta derivaatta 0:ssa on 0. Funktio on siis konforminen $\mathbb{C} \backslash 0$:ssa. Epäkonformisuuden voi havaita esimerkiksi tarkastelemalla reaali- ja imaginaariakselin leikkauskulmaa lähtö- ja kuvajoukossa. Tätä on havainnollistettu kuvassa 2.



Kuva 1: Eksponenttifunktio. Siniset ja punaiset viivat risteävät kohtisuoraan sekä lähtö- että kuvajoukossa. Huomaa, että 0:lla ei ole e^z:ssä alkukuvaa, joten siniset viivat eivät todellisuudessa risteä kuvajoukon origossa.



Kuva 2: Funktio z^2. Siniset ja punaiset viivat risteävät kohtisuoraan sekä lähtö- että kuvajoukossa, paitsi kuvajoukon origossa, jossa kulmat ovat kaksinkertaistuneet. Koko reaaliakseli on kuvautunut positiiviseksi reaaliakseliksi, ja koko imaginaariakseli negatiiviseksi.

2.2 Topologia

Määritelmä 2.1 (Topologia)**.** Kokoelma τ on *X:n topologia*, jos sen jäsenet ovat X:n osajoukkoja ja τ:lle pätevät ehdot

- $\emptyset \in \tau$ ja $X \in \tau$,
- Jos $V_i \in \tau$ kaikilla $i = 1, \ldots, n$, niin $V_1 \cap V_2 \cap \cdots \cap V_n \in \tau$,
- Jos $\{V_\alpha\}$ on mielivaltainen kokoelma τ:n jäseniä (äärellinen, numeroituva, tai ylinumeroituva), niin $\cup_\alpha V_\alpha \in \tau$.

Jos τ on X:n topologia, X:ää kutsutaan *topologiseksi avaruudeksi*, ja τ:n jäseniä kutsutaan *X:n avoimiksi joukoiksi*.

Suljetut joukot määritellään avointen joukkojen komplementteina. Joukon X *sulkeuma* $\overline{X}$ on pienin suljettu joukko, joka sisältää X:n. Avoimen joukon X *reuna* ∂X on $\overline{X} \setminus X$.

Työssä $\mathbb{C}$ on *metrinen avaruus*, jonka metriikka on $\rho(x, y) = |x - y|$. Avoin pallo, jonka keskus on x ja säde r, on joukko $\{z \in \mathbb{C} : \rho(x, z) < r\}$. Määrittelemme $\mathbb{C}$:n topologiaksi kaikkien avointen pallojen mielivaltaiset yhdisteet.

Määritelmä 2.2 (Homeomorfismi)**.** Kuvaus $\varphi : \Omega_1 \mapsto \Omega_2$ on *homeomorfismi*, jos

- φ on bijektio,
- φ on jatkuva,
- $\varphi^{-1} : \Omega_2 \mapsto \Omega_1$ on jatkuva.

Topologinen avaruus Ω_1 on Ω_2:een *homeomorfinen*, jos niiden välillä on olemassa homeomorfismi. Topologisten avaruuksien homeomorfisuus on ekvivalenssirelaatio.

Homeomorfismin jatkuvuusvaatimus tarkoittaa, että avaruudet on pystyttävä muotoilemaan toisikseen ”repimättä”. Esimerkiksi kiekkoa ei voi kuvata renkaalle ilman jatkuvuuden rikkoutumista. Keskenään homeomorfiset avaruudet ovat topologisesti ”samat”.

Esimerkki 2.4 (Laajennettu kompleksitaso)**.** Lisätään kompleksilukujen joukkoon ns. ideaalipiste ∞, jota käsitellään 0:n käänteislukuna. Merkitään näin saatua joukkoa symbolilla $\mathbb{C}_\infty$ ja kutsutaan sitä *laajennetuksi kompleksitasoksi*.

Stereografinen projektio on homeomorfismi pallopinnalta $\{(x, y, z) : x^2 + y^2 + z^2 = 1\}$ laajennettuun kompleksitasoon. Laajennettu kompleksitaso ja pallopinta ovat siis homeomorfiset. Lisäksi kuvaus on konforminen, mutta tässä sillä ei ole topologian kannalta merkitystä.

Automorfismi on homeomorfismi topologisesta avaruudesta itselleen.

Esimerkki 2.5 (Möbius-kuvaus)**.** Möbius-kuvaus on $\mathbb{C}_\infty$:n konforminen automorfismi:

$$M(z) = \frac{az + b}{cz + d}, \quad ad - bc \neq 0. \tag{12}$$

Kutsutaan laajennetun kompleksitason euklidisia ympyröitä ja suoria *yleistetyiksi ympyröiksi*. Möbius-kuvaus kuvaa yleistetyt ympyrät yleistetyiksi ympyröiksi, kun suorat tulkitaan pisteen ∞ läpäiseviksi ääretönsäteisiksi ympyröiksi. Möbius-kuvauksen voi konstruoida yksikäsitteisesti valitsemalla kolme eri pistettä ja määräämällä niille maalijoukosta toisistaan eroavat vastinparit. Tämä kuvaa lähtöjoukon kolme pistettä läpäisevän ympyrän maalijoukon kolme pistettä läpäisevälle ympyrälle.

Esimerkki 2.6 (Ylemmän puolitason automorfismi)**.** Merkitään ylempää puolitasoa $\mathbb{H} = \{z \in \mathbb{C}_\infty : \mathfrak{Im}\, z > 0\}$. Tässä työssä tarvitsemme Möbius-kuvauksena määriteltyä yhden parametrin $\mathbb{H}$:n automorfismia. Möbius-kuvaus muotoa

$$z \mapsto \frac{z}{1 - z/b} \tag{13}$$

kuvaa reaaliakselin itselleen säilyttäen sen suunnan, joten $\mathbb{H}$ kuvautuu $\mathbb{H}$:lle ja sen komplementti komplementille. Lähtö- ja maalijoukossa toisiaan vastaavat pisteet ovat esimerkiksi

$$0 \mapsto 0, \quad b \mapsto \infty, \quad \infty \mapsto -b. \tag{14}$$

2.3 Riemannin kuvauslause

Yksikkökiekko $\mathbb{D}$ on kompleksitason joukko

$$\mathbb{D} = \{z \in \mathbb{C} : |z| < 1\}. \tag{15}$$

Yksikköympyrä on yksikkökiekon reuna, eli $\{z \in \mathbb{C} : |z| = 1\}$. Sanomme, että yksikkökiekkoon homeomorfinen alue on *yhdesti yhtenäinen*.

Lause 2.1 (Riemannin kuvauslause)**.** *Olkoon $\Omega \subset \mathbb{C}$ avoin ja yhdesti yhtenäinen, joka ei ole tyhjä eikä koko $\mathbb{C}$. Tällöin on olemassa biholomorfinen kuvaus $f : \Omega \mapsto \mathbb{D}$. Kutsumme tällaista f:ää Riemannin kuvaukseksi.*

Riemannin kuvaus on yksikkökiekon konformisia automorfismeja vaille yksikäsitteinen. Käytännössä tämä tarkoittaa, että Riemannin kuvaukseen voi yhdistää minkä tahansa yksikkökiekon konformisen automorfismin, jolloin tuloksena on yhä Riemannin kuvaus. Jokaista mielivaltaista pistettä $z_0 \in \Omega$ ja kulmaa φ vastaa siis yksikäsitteinen Riemannin kuvaus f, jolla $f(z_0) = 0$ ja $\arg f'(z_0) = \varphi$. Tämä on seurausta Schwarzin lemmasta.

Riemannin kuvausta yleisemmät väittämät eivät ole yhtä suoraviivaisia. Yhdesti yhtenäiset alueet ovat konformikuvattavissa toisilleen, koska niitä yhdistävät biholomorfiset Riemannin kuvaukset yksikkökiekon kautta. Näin ei ole esimerkiksi kaikkien rengasmaisten avointen joukkojen välillä. Todistus löytyy luvun lähteestä [Rud87], mutta koska tämä työ käsittelee vain yhdesti yhtenäistä tapausta, se ja muut yleisemmät tapaukset jätetään tässä käsittelemättä.

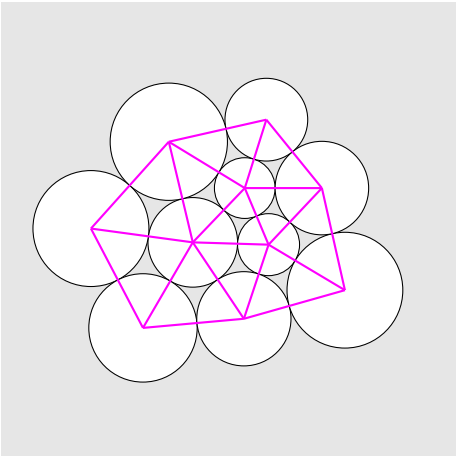
3 Circlepack

Circlepack on monikäyttöinen menetelmä Riemannin pintojen välisten konformikuvausten löytämiseksi. Toimiakseen se vaatii sileän, suuntautuvan pinnan diskretoituna kolmioinniksi. Kolmioinnin jokaiseen kärkeen sijoitetaan ympyrä, joiden säteet toimivat menetelmässä muuttujina. Tavoitteena on saada säteet sopimaan toisiinsa niin, että kolmioinnin sanelemat ympyrät sivuavat toisiaan.

Menetelmä on monesta syystä erikoinen. Ensinnäkin Circlepack-ongelman ratkaisun ja Riemannin kuvauksen yhteys on peräisin geometrisesta intuitiosta, joka ei ole aivan ilmeinen mutta on kylläkin todistettu. Kiinnostavaa on myös, miten ratkaisu löydetään. Menetelmässä tarkkaillaan yhteensopivuutta ympyröiden asettelussa, mutta ehtoja asetetaan vain hyvin paikallisella tasolla. Yhteensopivuuden hakeminen puolestaan onnistuu pelkkien säteiden perusteella, eikä ympyröiden paikkaa tasossa tarvitse määrätä ennen kuin pakkaus on valmis.

Ensimmäisen selvästi Circlepackiksi tunnistettavan idean esitteli William Thurston. Vuonna 1985 hän toi julki konjektuurinsa, jonka käsittelemän metodin hän väitti suppenevan Riemannin kuvaukseen. Thurstonin konjektuurin todistivat myöhemmin Rodin ja Sullivan [RS87], ja ideaa jatkokehitti Kenneth Stephenson nimeämällä kirjoittamansa Java-ohjelmiston Circlepackiksi [Ste]. Näiden ensimmäisten vaiheiden jälkeen Circlepackille on todistettu useita ominaisuuksia, jotka kuvastavat sen tulkintaa analyyttisten funktioiden diskreettinä analogiana. Nykyisin Circlepack, tarkoitettiin sillä sitten ohjelmistoa tai menetelmää, henkilöityy vahvasti Stephensoniin, ja Thurstonin menetelmä tunnetaan eräänä Circlepackin erityistapauksena.

Stephenson on kirjoittanut aiheesta runsaasti, ja painottaa aina – hyvästä syystä – että ”sphere packing” ja nyt käsiteltävä ”circle packing” ovat eri käsitteitä. Ensin mainittu käsittelee n-ulotteisten pallojen pakkaamista n-ulotteiseen tilaan, kun taas jälkimmäinen on 2-ulotteisten ympyröiden upottamista $\mathbb{C}$:een tai $\mathbb{D}$:een. Molempia on tutkittu matematiikassa laajalti, mutta niillä ei ole kovin paljoa yhteistä.



Kuva 3: Pakkausehdon toteuttava joukko ympyröitä ja niiden välinen kolmiointi.

3.1 Määritelmiä

Collinsin ja Stephensonin julkaisu [CS02] on alkuperäinen Circlepackin esittely modernissa muodossaan. Siihen pohjautuu myös algoritmin esittely tässä työssä. Keskitymme kuitenkin suurimmaksi osaksi vain käytännön seikkoihin, ja luotamme että vastaukset syvällisempiin algoritmia koskeviin kysymyksiin joko löytyvät muista lähteistä tai ilmenevät empiirisesti. Etenkin oletamme tunnetuksi, että luvussa kuvailtuun tehtävään on yksikäsitteinen ratkaisu. Pyrkimyksenä on myös esitellä algoritmi muodossa, jossa epäoleelliset selitykset on saatu karsittua. Esimerkiksi sivuutamme muutaman topologisen lisäehdon, joiden läpikäyminen olisi pitkää ja yksityiskohtaista eikä niiden puute erityisemmin haittaa algoritmin suoritusta. Lukija voi loppujen lopuksi tutustua yksityiskohtiin lähdemateriaalia lukemalla.

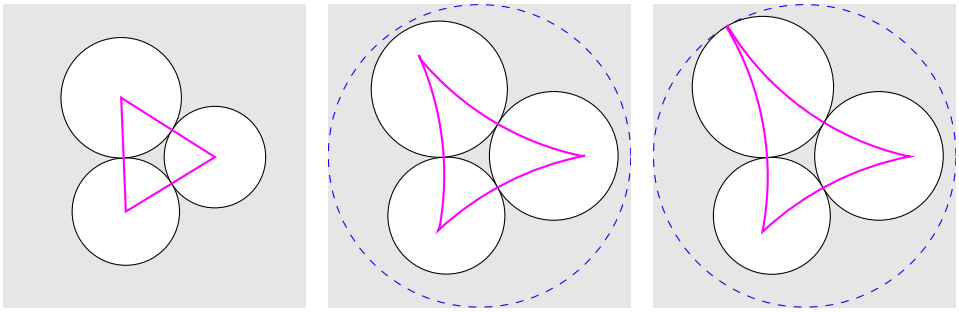
Määritelmä 3.1 (Geometria)**.** Circlepack-pakkauksen *geometria* on joko *euklidinen* tai *hyperbolinen*. Käytettävä geometria vaikuttaa mm. sallittuihin ympyröiden säteisiin ja kulmien laskemiseen. Euklidisessa geometriassa ympyrät ovat tavanomaisia $\mathbb{C}$:n ympyröitä, joiden säteet voivat vaihdella välillä $(0, \infty)$. Hyperbolista geometriaa esitämme *Poincarén kiekkomallilla*, jossa hyperbolinen taso on tiivistetty yksikkökiekon sisään. Kiekossa käytetään pituusmittaa $\frac{2|dz|}{1-|z|^2}$, jolloin hyperbolisten ympyröiden (hyperboliset) säteet kuuluvat välille $(0, \infty]$.

Poincarén kiekon reunapisteet ovat edellämainittua pituusmittaa käyttäen äärettömän kaukana muista pisteistä. Muulta osin etäisyydet Poincarén kiekossa ovat *geodeesien* eli lyhimpien reittien pituuksia. Nämä geodeesit ovat suorien osia; Suorat puolestaan ovat kiekon reunan kohtisuorasti leikkaavia ympyränkaaria. Kolmiot ovat kolmen pisteen välisten geodeesien rajaamia alueita. Euklidisesta geometriasta poiketen niiden kulmat summautuvat alle 180°:een.

Hyperboliset ympyrät ovat pistejoukkoina samalla euklidisia ympyröitä, mutta niiden keskuksina ja säteinä käsitellään kiekon pituusmittaa noudattavia hyperbolisia säteitä ja keskuksia. Useimpien hyperbolisten ympyröiden säteet ovat väliltä $(0, \infty)$. Tärkeänä poikkeuksena kuitenkin määritellään kiekon reunaa sisäpuolelta sivuavat ympyrät (*horocycle*), joiden hyperbolisena keskuksena voidaan johdonmukaisesti pitää sen tangenttipistettä ja ääretöntä sen hyperbolisena säteenä. Kuva 4 havainnollistaa ympyröiden keskipisteiden välisten kolmioiden muodostusta eri geometrioissa.

Määritelmä 3.2 (Kolmiointi)**.** Circlepack suoritetaan kolmioinnille (*triangulation*), joka kuvastaa sileää, suuntautuvaa topologista pintaa. Matemaattisesti kolmiointi on suuntautuva 2-ulotteinen *simplinen kompleksi* (*simplicial complex*, lyh. *kompleksi*, *complex*). d-ulotteinen simplinen kompleksi koostuu *simplekseistä* (*simplex*), jotka puolestaan ovat $1, \ldots, d+1$-jäsenisiä joukkoja. Kolmioinnin kärjet, särmät ja tahkot esitetään kompleksissa 0-, 1- ja 2-ulotteisina simplekseinä.

Käytännössä kolmioinnin määrittely on helpointa indeksoimalla kompleksin kärjet ja listaamalla tahkot näiden indeksien kolmikkoina eli 2-simplekseinä. Suorituksen aikana laskentaan vaikuttavat vain tahkojen osina esiintyvät särmät ja kärjet, joten algoritmin kannalta tällainen listaus on yksikäsitteinen esitystapa. Suuntautuvan



Kuva 4: Vasemmalla: Toisiaan sivuavat ympyrät euklidisessa geometriassa. Keskellä: Toisiaan sivuavat ympyrät hyperbolisessa geometriassa. Oikealla: Toisiaan sivuavat ympyrät hyperbolisessa, kun yksi ympyröistä sivuaa Poincarén kiekkoa.

kolmioinnin tahkot voidaan ilmoittaa joko kaikki vastapäivään tai kaikki myötäpäivään, ja ongelmien välttämiseksi näin kuuluukin tehdä. Tästä eteenpäin oletamme käsittelemämme simpliset kompleksit 2-ulotteisiksi, suuntautuviksi ja topologisen pinnan kolmioiviksi.

Circlepackin kolmioinnin kärjet jakautuvat *sisä-* ja *reunakärkiin*. Tunnistamme nämä laskemalla, kuinka moneen särmään ja tahkoon jokin tietty kärki kuuluu. Merkitään K:n sisältämien d-simpleksien joukkoa $K^{(d)}$:llä, jolloin kärjen v naapurien lukumäärä eli *aste* on

$$\deg(v) := |\{\Delta \in K^{(1)} : v \in \Delta\}|. \tag{16}$$

Jos v:tä ympäröi yhtä monta tahkoa kuin sillä on naapureita, eli $|\{\Delta \in K^{(2)} : v \in \Delta\}| = \deg(v)$, se on sisäkärki. Jos taas naapureita on enemmän kuin tahkoja, se on reunakärki. Muissa tapauksissa K ei täytä aiemmin annettuja vaatimuksia. Merkitsemme K:n reunakärkien joukkoa ∂K:lla.

Määritelmä 3.3 (Leima)**.** Leima (*label*) liittää kolmioinnin jokaiseen kärkeen ympyrän säteen. Euklidisessa geometriassa leimat ovat funktioita

$$R : K^{(0)} \mapsto (0, \infty). \tag{17}$$

Hyperbolisessa geometriassa määritelmä on muuten sama, paitsi määritelmän 3.1 nojalla reunakärjille sallitaan lisäksi ääretön säde.

Leimat sisältävät kaiken tarpeellisen tiedon Circlepack-iteraattien seuraamiseen ilman, että kiinnitettäisiin liiaksi huomiota ympyröiden geometriseen asetteluun. Emme vaadikaan, että leiman kuvailemat ympyrät pakkautuisivat siististi, vaan sallimme minkä tahansa vasttavuuden kärkien ja sallittujen säteiden välillä olevan leima. Tähän Circlepackin toiminta perustuukin: Kun ympyröiden välisiä yhteensopivuuksia parannetaan riittävästi paikallisella tasolla, myös kokonaisuus lähestyy täydellistä pakkausta. Jos tai kun leiman R määrittelemät ympyrät voidaan latoa omassa geometriassaan niin, että K:ssa tahkolla yhdistetyt ympyrät sivuavat toisiaan, sanomme R:n toteuttavan *pakkausehdon*. Numeerista suoritusta varten pakkausehdon toteutumiselle voidaan myös antaa virhetoleranssi.

Määritelmä 3.4 (Reunafunktio)**.** Reunafunktio g määrää kolmioinnin ohella ympyräpakkauksen muotoa asettamalla reunakärkiä vastaavien ympyröiden säteet. Määritelmä on euklidisessa geometriassa

$$g : \partial K \mapsto (0, \infty). \tag{18}$$

Jälleen määritelmään 3.1 perustuen hyperbolisessa geometriassa g:n maalijoukko on $(0, \infty]$, koska lähtöjoukko sisältää vain reunakärkiä. Leima R ja reunafunktio g ovat yhteensopivat, jos $R(v) = g(v)$ aina kun $v \in \partial K$.

3.2 Paikallinen yhteensopivuus

Koska leimat ovat mielivaltaisia luetteloita ympyröiden säteistä kombinatoriikassa K, on helppoa kuvitella esimerkki, jossa leiman kuvailemat ympyrät on mahdotonta asetella pakkausehdon toteuttavaksi kokonaisuudeksi. Siksi tarvitsemme heuristiikan, jolla leimoja muotoillaan paremmin pakkautuviksi. Idea on tarkkailla yhtä sisäkärkeä kerrallaan ja mitata kulmia ympyröiden välille viritetyissä kolmioissa. Kulmien perusteella arvioidaan, tarvitseeko sädettä suurentaa vai pienentää, ja leimaa päivitetään tuloksen perusteella.

Tästä eteenpäin kiinnitämme vähemmän huomiota kolmiointiin ja oletamme sen olevan symbolilla K merkittävä vakio. Notaatiota lyhentääksemme saatamme merkitä sädettä $R(v)$:n vastaavalla kärjellä v, kun leima on kontekstista selvä.

Määritelmä 3.5 (Kulmasumma)**.** Kulmasumma $\theta(v; R)$ mittaa paikallisten yhteensopivuusehtojen toteutumista kärjessä v leimalla R. Tarkastellaan kaikkia v:n sisältäviä tahkoja $\{v, u, w\} \in K^{(2)}$. Leimassa R jokaista tahkoa vastaa kolme sädettä: $R(v)$, $R(u)$ ja $R(w)$. Piirretään nämä kolme ympyrää sivuamaan toisiaan ja muodostetaan näiden keskipisteiden välinen kolmio. Tämän kolmion sivujen pituudet ovat $R(v) + R(u)$, $R(v) + R(w)$, ja $R(u) + R(w)$. Mitataan kolmiossa se kulma, joka sijoittuu v:n ympyrän sisään, ja summataan tämä luku kaikkien tahkojen yli. Kulmasumma voidaan siis lausua muodossa

$$\theta(v; R) := \sum_{\{v,u,w\} \in K^{(2)}} \alpha(R(v), R(u), R(w)), \tag{19}$$

missä α mittaa v:tä vastaavaa kulmaa $R(v)$-, $R(u)$- ja $R(w)$-säteisen ympyrän keskipisteiden välisessä kolmiossa.

Kun pakkausehto on voimassa ja v:n naapuriympyrät asetellaan järjestyksessä v:n ympärille, viimeinen ympyrä sivuaa ensimmäistä. Tällöin kolmioiden kulmat v:ssä summautuvat 2π:iin. Jos pakkausehto ei päde, kulmasumma voidaan silti ratkaista ja v:n sädettä voidaan muuttaa tavoitteen saavuttamiseksi.

Määritelmä 3.6 (Uniform neighbor model)**.** Ratkaistaan sisäkärjelle $v \in (K^{(0)} \setminus \partial K)$ yhteensopivuusehdon paremmin toteuttava säde käyttäen uniform neighbor -päivitystä.

Olkoon R vakio. On olemassa yksikäsitteinen säde $\hat{v}$, jolla kaikki v:n naapurit voitaisiin korvata niin, että v:n kulmasumma ei muutu. Tämä säde on se, joka toteuttaa yhtälön

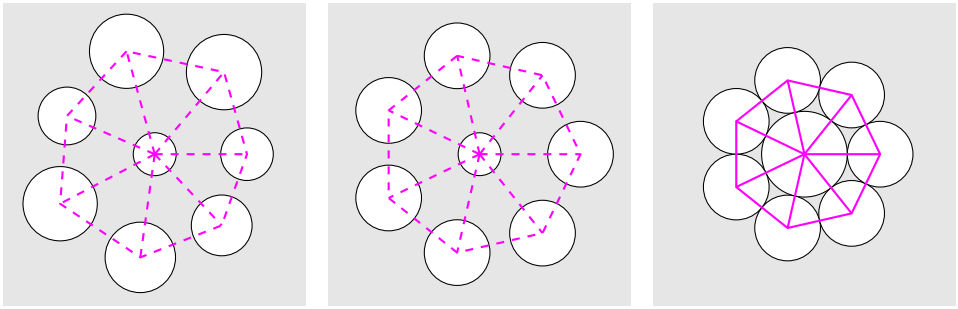
$$\theta(v; R) = \deg(v)\ \alpha(v, \hat{v}, \hat{v}). \tag{20}$$

Oletetaan, että $\hat{v}$ voidaan ratkaista ja on nyt tunnettu. Ratkaistaan se yksikäsitteinen u, joka toteuttaa yhtälön

$$\hat{\theta}(u; \hat{v}) = 2\pi. \tag{21}$$

Tällöin v:n perusteella uniform neighbor -päivitetty leima R' on

$$R'(v) = u, \tag{22}$$
$$R'(w \neq v) = w. \tag{23}$$



Kuva 5: Vasemmalla: Sisäkärki naapureidensa ympäröimänä. Sisäkärjen ympyrä on naapurustoon verrattuna liian pieni. Keskellä: Naapurien koko on tasattu muuttamatta kulmasummaa sisäkärjessä. Oikealla: Sisäkärjen kokoa on muutettu, ja se on nyt paikallisesti yhteensopiva naapurustonsa kanssa. Paikallinen yhteensopivuus on kuitenkin samalla saattanut rikkoutua viereisissä kärjissä.

Uniform neighbor modelin määritelmä jättää avoimeksi sen, miten $\hat{v}$ ja u ratkaistaan. Tämä käsitellään yksityiskohtaisesti [CS02]:ssa, eikä saman toistaminen tässä esittelyssä olisi tarkoituksenmukaista. Erityisesti α:n ja samalla θ:n ratkaisemiseen pelkkien säteiden avulla on annettu tehokkaat kaavat, ja vaikeampi kysymys, miten $\hat{v}$ ja u johdetaan näistä, on myös käsitelty seikkaperäisesti kummassakin geometriassa. Implementaation kannalta nostamisen arvoista on, että kulmasummaa ei kannata kirjaimellisesti toteuttaa $\sum_{\{v,u,w\}\in K^{(2)}}$:na, hakuna kaikkien tahkojen yli, vaan jokaisen sisäkärjen naapurusto olisi tallennettu helposti saavutettavaan muotoon. Tätä vastaava käsite, *kukka* (*flower*), onkin määritelty [CS02]:ssa jo teoriaosuudessa yksityiskohtaisempaa kolmioinnin topologian määrittelyä varten.

3.3 Suoritus

Circlepackissa kahden iteraatin, R_n:n ja R_{n+1}:n, välillä jokaista sisäkärkeä käydään päivittämässä tasan kerran. Näin saadaan aikaan melko tasaista ja ennakoitavaa

suppenemista. Empiirisesti on havaittu, että useimmissa kolmioinneissa kärkien päivitysjärjestyksellä ei ole juurikaan väliä. Järjestyksen permutointi iteraatioaskelten välillä voi siis olla implementoijasta houkutteleva idea, mutta sillä ei voiteta paljoa. On tosin olemassa kolmiointien luokkia, joille on helppoa keksiä huonosti suppenevia permutaatioita.

Hyvin määritellyllä ympyräpakkaustehtävällä on aina yksikäsitteinen leima $\bar{R}$, jossa kulmasummat toteutuvat virheettä. Yhden säteen muuttaminen uniform neighborilla on *konservatiivista*, eli jos leimaa päivitetään v:ssä, uudelle säteelle u pätee

$$v \leq u \leq \bar{R}(v) \quad \text{tai} \quad v \geq u \geq \bar{R}(v). \tag{24}$$

Koska iteraatti muodostetaan usealla uniform neighbor -päivityksellä, myös R_{n+1}:n voidaan sanoa olevan konservatiivinen R_n:ään nähden.

Algoritmin suppenemista seurataan tavanomaisella neliövirheellä, jonka alituttua iterointi lopetetaan. Virhetermeinä toimivat luonnollisesti kulmasummien erotukset 2π:stä. Neliövirhettä ei tarvitse laskea aivan tarkalleen, vaan virhetermejä voi summata uniform neighbor -päivitysten aikana, vaikka muuttuvat ympyröiden säteet vaikuttavatkin toistensa kulmasummiin.

3.3.1 Riemannin kuvauksen ratkaiseminen

Nyt olemme koonneet riittävästi määritelmiä asettaaksemme tehtävän, joka ratkaisee Riemannin kuvauksen. Muodostetaan aluksi tehtävässä käytettävä kolmiointi K. Täytetään kuvattava alue euklidisessa geometriassa säännöllisellä, maksimaalisella ympyräpakkauksella. Kolmioinnin tahkot ovat pakkauksen toisiaan sivuavien ympyröiden kolmikkoja. Tässä toimii erityisen hyvin kuusikulmainen ”penny pack”, mutta alueen geometriasta riippuen muutkin pakkaukset ovat mahdollisia.

Siirrytään hyperboliseen geometriaan, jossa käytämme samaa K:ta. Valitsemme reunafunktioksi $g \equiv \infty$ pakottaaksemme reunakärkien ympyrät sivuamaan yksikkökiekon reunaa. Tämän jälkeen on enää suoritettava Circlepack-algoritmi riittävän tarkkaan pakkausehdon toteuttamiseen asti.

3.3.2 Kiihdytys

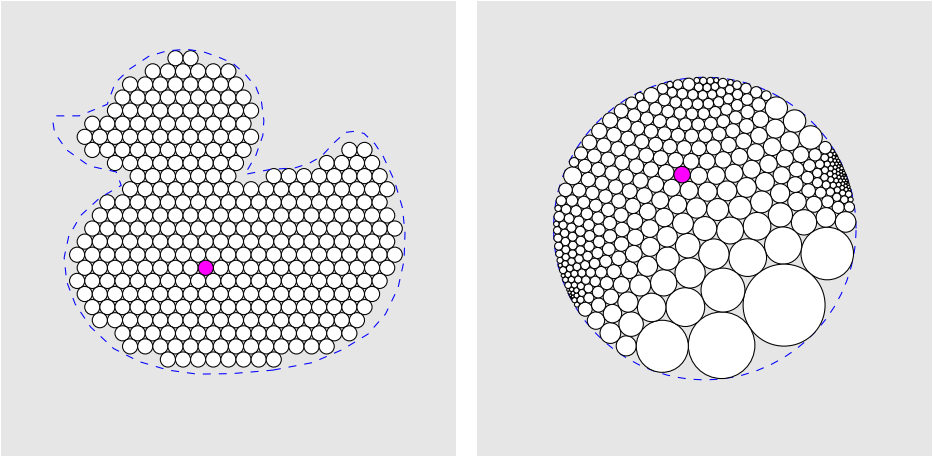
Circlepack-iteraattien konservatiivisuus varmistaa, että iteraatit lähestyvät aina oikeaa arvoa. Toisaalta se tarkoittaa myös, ettei algoritmi ota perusteltujakaan riskejä supetakseen nopeammin. Siksi käsittelemmekin nyt keinoja, joilla Circlepack-iteraatio saadaan tekemään arvaus iteraatista yhden tai useamman askeleen päässä.

Kokeellisista tuloksista tiedetään, että suppeneminen on paikallisesti lineaarista, eli

$$R_{l+1} - \bar{R} \approx \lambda \left(R_l - \bar{R} \right) \tag{25}$$

pätee jokaisessa sisäkärjessä jollakin $\lambda < 1$. Oletetaan 25 tarkaksi, jolloin sen voi tulkita rekursioyhtälönä. Tällöin voidaan ratkaista R_{l+2} ja $\bar{R}$ riippuvina R_l:sta, R_{l+1}:sta ja λ:sta:

$$R_{l+2} = R_{l+1} + \lambda \left(R_{l+1} - R_l \right) \tag{26}$$



Kuva 6: Riemannin kuvauksen ratkaiseminen Circlepackilla. Vasemmalla: Kolmiointi muodostetaan euklidisessa geometriassa. Oikealla: Algoritmi on supennut hyperbolisessa geometriassa reunafunktiolla $g \equiv \infty$.

ja

$$\bar{R} = R_{l+1} + \frac{\lambda}{1-\lambda}\left(R_{l+1} - R_l\right). \tag{27}$$

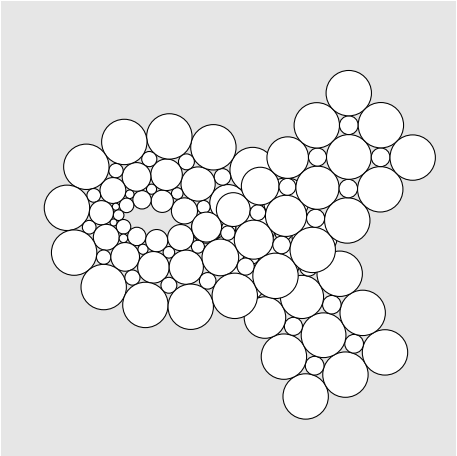
Riippuu Circlepackin implementoinnista, milloin ja miten näitä kiihdytysmenetelmiä käytetään. Molempien toteuttamisessa tarvitaan peräkkäisten iteraattien tallennus (R_l ja R_{l+1}) ja arvio virheen kokoluokasta (λ). Koska algoritmissa voi käyttää lähinnä säteistä johdettavia suureita, λ:n valintaan ei ole kulmasummien neliövirheen lisäksi juuri vaihtoehtoja. Onneksi empiiriset tulokset vahvistavat tämän olevan toimiva valinta. Implementaatiosta riippuu myös, millä λ:n arvoilla suppeneminen arvioidaan riittävän lineaariseksi, jotta kiihdytys olisi turvallista.

3.4 Huomioita ja yhteenveto

Ohitimme luvussa 3.1 kaiken, mikä ei ole kriittisen tärkeää numeriikan kannalta. Topologian osalta sivuutimme sisäkärkien joukon yhtenäisyyden, eli kaikista sisäkärjistä tulisi olla polku sisäkärkiä pitkin kaikkiin muihin. Jos näin ei ole, sisäkärjet jakautuvat vähintään kahteen yhtenäiseen komponenttiin. Circlepackin suorittaminen tällaisessa kolmioinnissa on lähes sama kuin sen suorittaminen jokaiselle komponentille erikseen. Kolmiointiin voi myös jäädä ylimääräisiä reunakärkiä, joiden olemassaolo ei millään tavalla vaikuta suoritukseen. Näitä ”orpoja” reunakärkiä voi ennen iterointia poistaa kolmioinnista ilman seuraamuksia.

Yllättäen Circlepackissa sallitaan ympyröiden latominen toistensa päälle (*non-univalence*). Tällaisen tilanteen saa aikaan jopa yksinkertaisille kolmioinneille euklidisessa geometriassa, tarvitaan vain sopiva reunafunktio (kuva 7). Tämän voisi luulla olevan puute, joka johtuisi Circlepackin tavasta käsitellä geometriaa ainoastaan ympyröiden säteinä kolmioinneissa. Kuitenkin Circlepackista on olemassa myös laajennettu versio, joka suorastaan vaatii ominaisuuden käyttöä. Tässä laajennoksessa

sisäkärjille sallitaan kulmasummina 2π:n monikerrat, ja jolloin ympyrät välttämättä peittävät toisiaan. Näitä ympyräpakkauksia on tutkinut mm. Dubejko, joka on esimerkiksi ratkaissut välttämättömät ja riittävät ehdot kulmasummille [Dub93].



Kuva 7: Circlepack-”käärme” peittää ladottuna itsensä.

Ympyröiden latomisesta puheenollen, valmiin ympyräpakkauksen ohjelmallisesta asettelusta voi olla vaikea löytää lähdettä. Erityisesti hyperbolisessa geometriassa latominen ei välttämättä ole aivan helppoa, jos leimojen säteitä ei ole tottunut käsittelemään. Ympyröiden asetteluun liittyvää aritmetiikkaa on johdettu [Lut10]:ssa.

Vaikka Riemannin kuvauslauseen ratkaiseminen onkin Circlepackin alkuperäinen käyttötarkoitus, nykyään algoritmi on siihen liian hidas. Kuvauslauseen ratkaisee suoremmin ja nopeammin moni muu algoritmi, kuten Zipper tai Schwarz-Cristoffel. Circlepackin vahvuus onkin sen kyky ratkaista monenlaisia ongelmia: Sopivalla geometrian ja reunafunktion valinnalla algoritmi suppenee konformikuvaukseen myös muilla kuin yhdesti yhtenäisillä pinnoilla, ja se toimii jopa tason osajoukkoja yleisemmillä Riemannin pinnoilla. Stephenson sanookin ”If you can triangulate it, circlepack it!” täysin liioittelematta.

Klassisen Circlepackin epäilemättä suurin ongelma on sen voimakas epälineaarisuus. Jokaista iteraattia varten jokaisessa sisäkärjessä on laskettava $\deg(v)$:n verran (penny packissa 6) trigonometrisia käänteisfunktioita, joiden tuloksia ei edes voi järkevästi käyttää uudelleen. Tarkkuuden lisäämiseksi kolmioinnin resoluutiota on suurennettava, ja sisäkärkien lukumäärän voi odottaa kasvavan neliöllisesti ympyröiden kokoa pienennettäessä. Tosin ongelman voi aluksi ratkaista karkeammassa geometriassa, ja tästä saadun tuloksen perusteella voidaan luoda hienompaa kolmiointia varten hyvä alkuarvaus. Aiheesta kiinnostuneen lukijan kannattaa tutustua Dubejkon ja Stephensonin julkaisuun [DS95], joka auttaa alkuun koeasetelmien kanssa.

Jos kuitenkin mennään kehitysaskelissa eteenpäin, tuoreimmista Circlepackin muunnoksista löytyy Collinsin, Orickin ja Stephensonin julkaisema algoritmia linearisoiva versio [COS17b], [COS17a]. Jos kolmiointi sopii `GOPack`:lle, algoritmin luvataan olevan paljon nopeampi ja tarkempi kuin klassinen, erityisesti suurissa kolmioinneissa. Tähän algoritmiin liittyvä jatkotutkimus saattaa myös kiinnostaa lukijaa.

Tätä työtä varten kirjoittaja on ohjelmoinut yksinkertaisen Circlepackin toteutuksen [Lei19a], joskaan ei sen pääaiheena (sen kunnian saa vastaava Zipper-toteutus [Lei19b]. MATLAB-paketti `packit` ratkaisee työssä kuvaillun tyyppisen Circlepack-tehtävän käyttäen kiihdytysmenetelmiä. Implementaatiosta puuttuu mm. kolmioinnin resoluution tarkentaminen, joten optimoitavaa kyllä riittää. Lukija voi kokeilla pakettia tai halutessaan myös johtaa oman versionsa.

4 Zipper

Kutsutaan *Zipper-algoritmin* ja sen kahden kantamuodon joukkoa *Zipper-perheeksi*. Zipper-perheen algoritmit käsittelevät kärkipistein määriteltyä monikulmiota kompleksitasossa ja kuvaavat sen sisäpisteet konformisesti ylemmälle puolitasolle. Koska ylempi puolitaso on helppo kuvata yksikkökiekolle, Zipper-perhe soveltuu hyvin Riemannin kuvauksen numeeriseen ratkaisuun.

Nimi ”Zipper” kuvailee algoritmin toimintaa osuvasti: Origo toimii monikulmion reunaa pitkin vedettävänä ”vetoketjuna”, joka avaa monikulmion sisäpisteiden joukon koko puolitasoksi. Tämä tapahtuu yhdistämällä peräkkäin yksinkertaisia konformikuvauksia, jonka ominaisuudet tunnetaan hyvin ennalta.

Tarkastelemme tässä luvussa kaikkia kolmea Zipper-perheen algoritmia. Esittely pohjautuu Marshallin ja Rohden julkaisuun [MR06], jossa algoritmit kuvaillaan yksi kerrallaan toistensa ominaisuuksiin nojaten. Samassa julkaisussa käsitellään algoritmien suppenemisen todistavat matemaattiset faktat. Emme nyt kanna niistä huolta, vaan näkökulma on enimmäkseen käytännönläheinen. Tässä vaiheessa lienee myös hyvä mainita Marshallin Fortran-ohjelmasta [Mar], joka on Zipperin alkuperäinen ohjelmoitu toteutus.

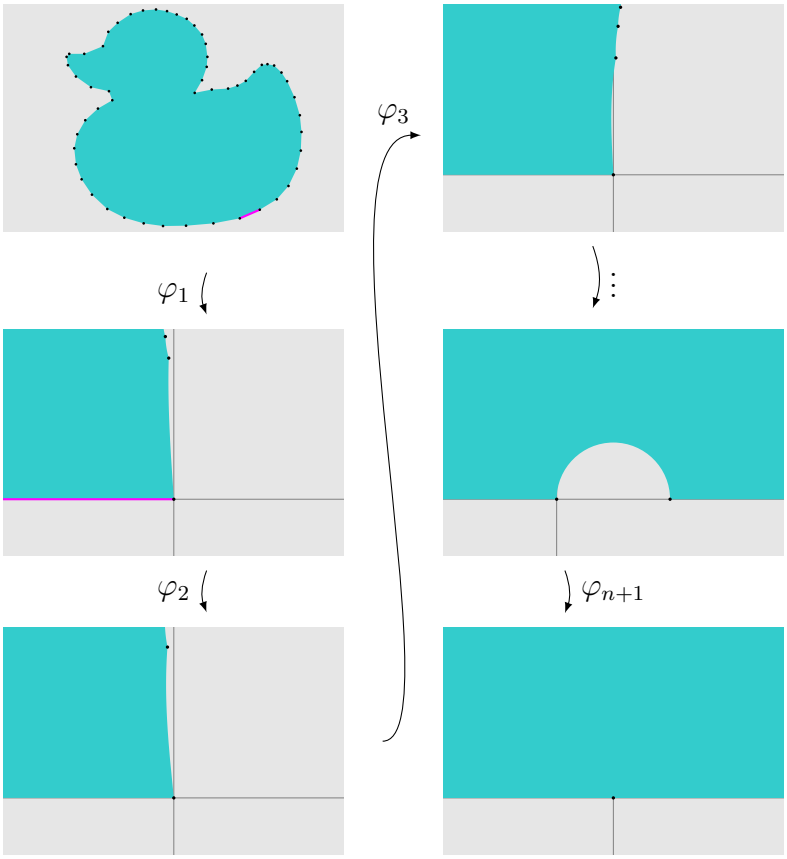
4.1 Zipper-algoritmiperhe

Zipper-perheen algoritmit suoritetaan tietyllä yhteisellä kaavalla, joka on karkeasti esitetty kuvassa 8. Algoritmit approksimoivat monikulmion reunaa ja kuvaavat sen reaaliakselille. Jokaisella yksittäisellä askeleella approksimoidaan yhtä monikulmion sivua; Tämä approksimaatio on algoritmien välinen pääasiallinen ero.

Voimme olettaa, että käsiteltävä monikulmio on annettu kompleksilukujen jonona (z_i). Jono määrittelee äärellisen, itseään leikkaamattoman monikulmion kärkipisteet kiertäen vastapäivään. Merkitään lisäksi monikulmion sisäpisteiden joukkoa Ω:lla. Nämä z_i:n ja Ω:n merkitykset säilyvät luvun 4.2 alkuun saakka, jossa samoja symboleja käytetään eri merkityksissä (myös lähdemateriaalissa tehdään sama muutos).

Zipper-perheeseen kuuluva algoritmi tuottaa $(n+1)$ kappaletta konformisia *osakuvauksia* φ_i, joiden yhdiste $(\varphi_{n+1} \circ \ldots \circ \varphi_1)$ on approksimaatio konformikuvauksesta $\Omega \mapsto \mathbb{H}$. Osakuvauksien muodostustapa vaihtelee algoritmien välillä, mutta algoritmin sisäisesti $\varphi_2, \ldots, \varphi_n$ ovat keskenään hyvin samankaltaisia. Tämän vuoksi on perusteltua jaotella osakuvaukset kolmeen *osakuvaustyyppiin*: ”φ_1”, ”φ_i”, ja ”φ_{n+1}”. Osakuvaukset on myös helppo kääntää, joten koko konformikuvauksen käänteisfunktio saadaan yhdistämällä osakuvausten käänteisfunktiot vastakkaisessa järjestyksessä. Algoritmia suoritettaessa muodostetaan myös parametrijono (ζ_i), joka koostuu z_i:n kuvapisteistä ”ajankohtaisessa” vaiheessa algoritmin suoritusta.

Tulevissa kappaleissa käsitellään Zipper-perheen teoreettista pohjaa, mutta algoritmit on lopulta tarkoitus suorittaa numeerisesti. Kaikki algoritmin tuottama on siis tavalla tai toisella approksimaatiota, mutta tämän painottaminen erikseen kaikissa tapauksissa kävisi kömpelöksi. Esimerkiksi väite ”Zipper-algoritmi tuottaa konformikuvauksen Ω:lta $\mathbb{H}$:lle” ei eksaktisti pidä paikkaansa, vaan kyseessä on numeerisia epätarkkuuksia sisältävä kuvaus approksimoidusta lähtöjoukosta approksimoituun



Kuva 8: Geodeesisen ja viiltoalgoritmin kulku.

maalijoukkoon. Väite silti kertoo mitä algoritmin on *tarkoitus* tehdä, ja luotamme lukijan osaavan tulkita eron kontekstista.

Puhumme myös paljon pistejoukoista jonkin kuvauksen alaisuudessa. Useimmiten tilanne liittyy osakuvauksen – olkoon se φ_{j+1} – konstruointiin, jolloin olemme ratkaisseet ennalta osakuvaukset φ_1:stä φ_j:een. Tällöin olemme kiinnostuneita siitä, mihin Ω tai z_i:t ovat ”tähänastisilla” osilla kuvattu. Tässä esimerkissä kuvaus olisi muotoa $(\varphi_j \circ \ldots \circ \varphi_1)$, mutta jatkossa oletamme kuvauksen ja indeksien olevan kontekstista selviä.

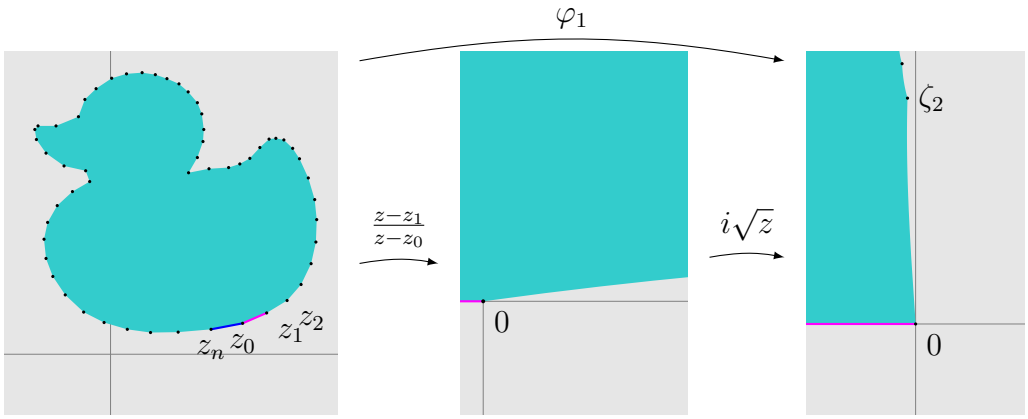
Zipper-perheen algoritmeissa käytetään paljon kompleksista neliöjuurta. Neliöjuuren haaran valinnassa vallitsee kaksi hyvin perusteltua konventiota: Kompleksiluvun vaihekulma kuuluu joko välille $[0, 2\pi)$ tai $(-\pi, \pi]$ ja se puolittuu juurta otettaessa. Emme tässä työssä valitse yhtä konventiota ja hylkää toista, vaan valitsemme haaran jokaisessa osakuvauksessa sen maalijoukon mukaisesti (joka useimmiten on $\mathbb{H}$). Tämä näyttää olevan myös [MR06]:ssa tehty päätös, vaikka julkaisussa suositellaankin välttämään konventioeroista johtuvat ongelmat implementoimalla algoritmi oikeaan puolitasoon.

4.1.1 Geodeesinen algoritmi

Zipper-perheessä φ_1-osakuvausten tehtävä on alustaa ja jossakin mielessä normalisoida monikulmio kuvaamalla ensimmäinen jana negatiiviselle reaaliakselille. Geodeesisen algoritmin osakuvaus φ_1 on

$$\varphi_1 : \Omega \mapsto \mathbb{H}, \quad \varphi_1(z) = i\sqrt{\frac{z - z_1}{z - z_0}},$$

ja sen toiminta on esitetty kuvassa 9. Funktio kuvaa z_0:n äärettömään, z_1:n origoon ja näiden välisen janan negatiiviselle reaaliakselille. Sisäkulma z_1:ssä puolittuu, ja Ω:n sisäpisteet kuvautuvat ylempään puolitasoon.



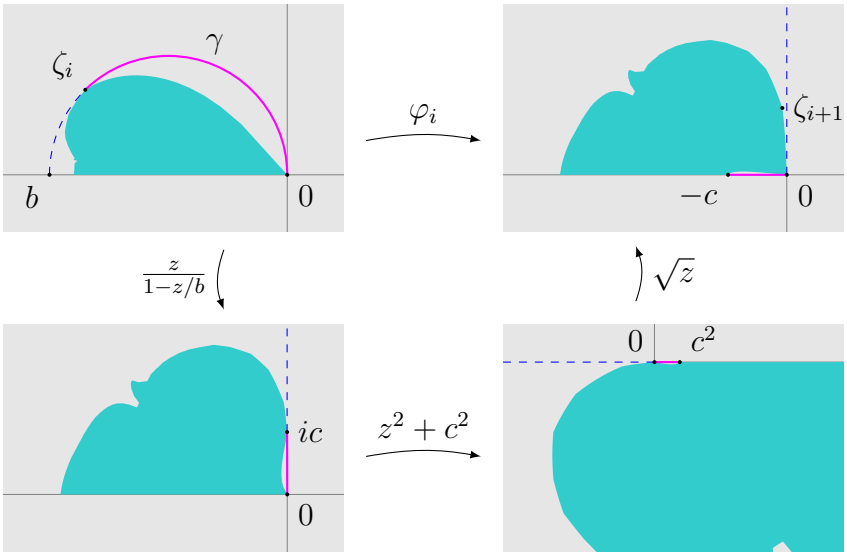
Kuva 9: Geodeesisen ja viiltoalgoritmin φ_1.

Osakuvaukset φ_i approksimoivat reunaa origon ja seuraavan kärkipisteen välillä. Geodeesinen algoritmi käyttää reunan approksimaatioinaan ympyränkaaria, jotka risteävät reaaliakselin kanssa origossa kohtisuorasti. Kaari määrittyy yksikäsitteisesti, kun päätepisteiksi valitaan origo ja $\zeta_i \in \mathbb{H}$. Merkitään tätä kaarta γ:lla, ja määritellään lisäksi ζ_i:stä riippuvat apuvakiot

$$b = |\zeta_i|^2 / \mathfrak{Re}\, \zeta_i, \tag{28}$$
$$c = |\zeta_i|^2 / \mathfrak{Im}\, \zeta_i. \tag{29}$$

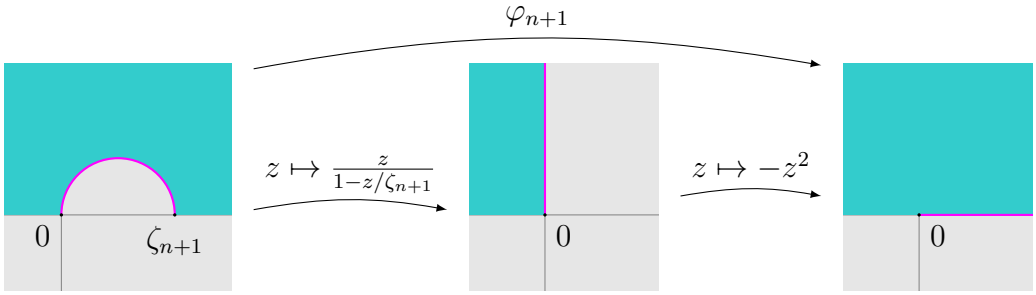
Nyt voimme ryhtyä pala palalta rakentamaan φ_i:tä, jonka kulku on esitetty kuvassa 10. Käytetään aluksi ylemmän puolitason automorfismia $\frac{z}{1-z/b}$. Tämä kuvaa b:n äärettömään ja ζ_i:n ic:een, joten automorfismi tosiasiassa kuvaa γ:n positiiviselle imaginaariakselille. Kuvataan $z^2 + c^2$, niin ζ_i päätyy origoon. Lopuksi vielä otetaan kompleksinen neliöjuuri, jonka haara valitaan ehtoon $\varphi_i : \mathbb{H} \mapsto \mathbb{H}$ sopivana. Tätä kaavaa toistetaan tuleville ζ_i:lle $(n-1)$ kertaa, eli siihen asti, kunnes jäljellä on enää yksi parametri, ζ_{n+1}.

Ennen viimeistä osakuvausta φ_{n+1} (kuva 11) oletamme Ω:n kuvautuneen puoliympyrän rajaamalle alueelle. Tämän puoliympyrän halkaisija on jana origosta reaaliseen ζ_{n+1}:een, ja parametrin merkki kertoo, onko kyse puoliympyrän sisäpuolesta vai sen



Kuva 10: Geodeesisen algoritmin φ_i.

komplementista ylemmässä puolitasossa. Suorituksen kannalta tällä ei kuitenkaan ole väliä: Ensimmäinen vaihe on joka tapauksessa ylemmän puolitason automorfismi $\frac{z}{1-z/\zeta_{n+1}}$, jolla pistejoukko kuvautuu ylävasemmaksi neljännestasoksi. Toinen vaihe on $-z^2$, jonka voidaan lukea olevan yhdiste myötäpäivään kiertämisestä (kertominen $(-i)$:llä) ja neliöinnistä. Kierto kohdistaa alueen oikean reunan positiiviselle reaaliakselille, joka pysyy paikallaan neliöitäessä. Vasen reuna taas kiertyy positiiviselle imaginaariakselille ja neliöitäessä negatiiviselle reaaliakselille, jolloin koko alue kuvautuu puolitasoksi, kuten halusimmekin.



Kuva 11: Geodeesisen ja viiltoalgoritmin φ_{n+1}.

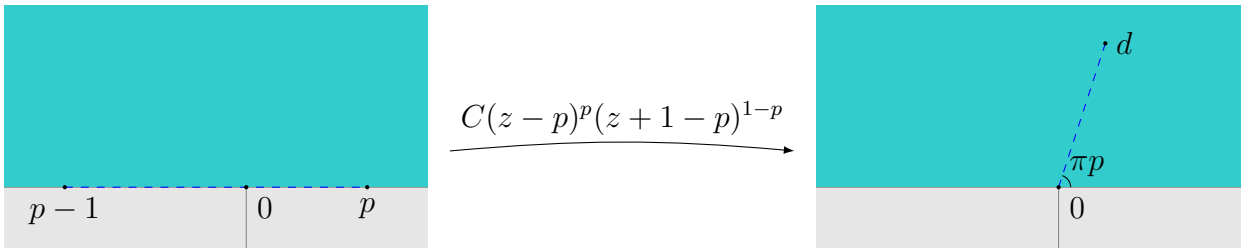
4.1.2 Viiltoalgoritmi

Viiltoalgoritmi johdetaan geodeesisesta vaihtamalla reunan approksimaatio ympyränkaaresta janaan. Muita muutoksia ei päällisin puolin ole, ja erityisesti φ_1 ja φ_{n+1} pysyvät samoina. Uudella φ_i-osakuvauksen tyypillä ei ole esitystä suljetussa muodossa, mutta sen käänteisfunktiolla on. Funktio ratkaistaankin numeerisesti käänteisfunktionsa avulla.

Määritellään *viiltokuvaus (slit map)* seuraavasti:

$$g_d(z) = C(z-p)^p(z+1-p)^{1-p}, \tag{30}$$

missä $p = \frac{\arg d}{\pi} \in (0,1)$ ja $C = \frac{|d|}{p^p(1-p)^{1-p}}$. Ylemmän puolitason kuvajoukko on ylempi puolitaso ilman janaa 0:sta d:een. Joukkojen reunalla välit $[0,p]$ ja $[p-1,0]$ kuvautuvat kumpikin tälle janalle. Kuvauksen hahmottamisessa auttaa, kun ajattelee 0:aa vedettävän kohti d:tä samalla kun reaaliakselilla olevat janat ”nipistetään” yhteen. Tätä voi seurata kuvasta 12.



Kuva 12: Viiltokuvaus g_d.

Käänteisfunktio vastaavasti kuvaa janan 0:sta d:een reaaliakselille. Voimme käyttää janaa monikulmion reunan approksimaationa Zipper-perheen algoritmissa: Kun ζ_i on seuraava kärkipiste, viiltoalgoritmin osakuvaus φ_i on $g_{\zeta_i}^{-1}$. Koska viiltokuvauksen numeerinen ratkaisu on omana aiheenaan suhteellisen laaja ja koskee myös seuraavaa algoritmia, käsittelemme sen myöhemmin. Toistaiseksi kuitenkin riittää tieto, että funktiolla on suljettu muoto yhteen suuntaan, jonka avulla se voidaan ratkaista numeerisesti myös toiseen.

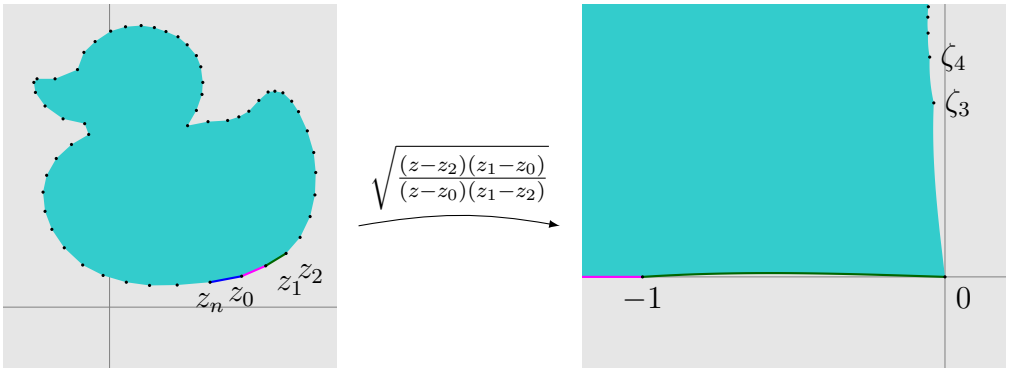
4.1.3 Zipper-algoritmi

Siinä missä geodeesinen algoritmi ja viiltoalgoritmi arvioivat monikulmion reunaa vain seuraavan pisteen perusteella, Zipper sovittaa *kaksi* seuraavaa pistettä samalle ympyränkaarelle. Geodeesisesta algoritmista poiketen ympyränkaaren ei tarvitse olla reaaliakseliin nähden kohtisuora, sillä origo ja kaksi pistettä asettavat kaarelle tiukemmat ehdot. Erityisen hyvin sovitus toimii silloin, kun monikulmio ”kaartuu” parittomissa pisteissä eli suurimmat muutokset reunan suunnassa osuvat muualle kuin sovitettavien kaarien päätepisteisiin.

Algoritmiin tuodun ”parillisuuden” vuoksi vaadimme monikulmion kärkien lukumäärän olevan parillinen. Kun kärkien lukumäärää merkitään $(2n+2)$:na, Zipper-algoritmi suoritetaan $(n+1)$:ssä askeleessa. Parametreja ζ_i tulee tällöin $(2n+3)$ kappaletta, joista viimeinen on jälleen z_0:n kuvapiste ennen viimeistä vaihetta.

Zipperin φ_1 (kuva 13) ottaa kahden parametrin sijaan kolme. Kuvaus on

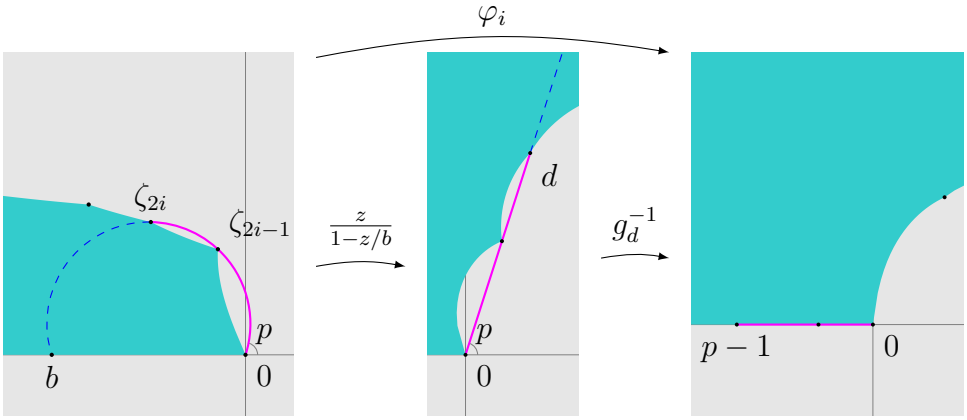
$$\varphi_1 : \Omega \mapsto \mathbb{H}, \quad \varphi_1(z) = \sqrt{\frac{(z-z_2)(z_1-z_0)}{(z-z_0)(z_1-z_2)}},$$



Kuva 13: Zipperin φ_1.

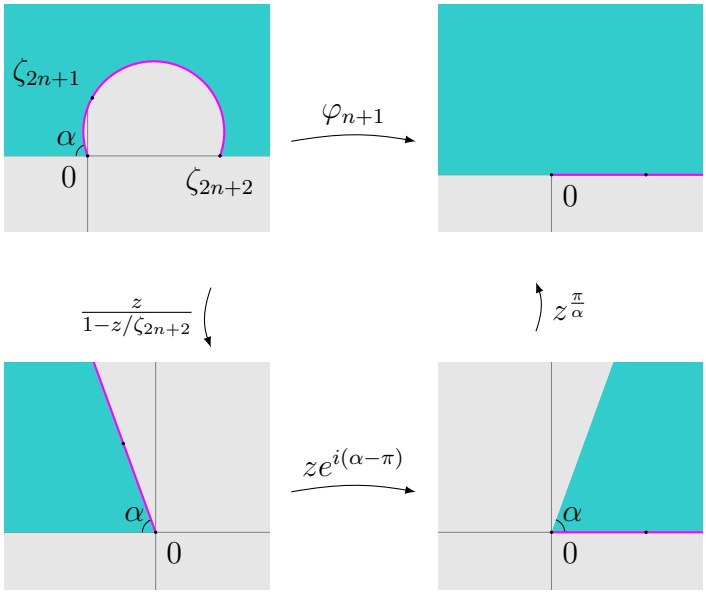
joka vie z_0:n äärettömään, z_1:n (-1):een ja z_2:n origoon.

Zipperin φ_i:ssä (kuva 14) etsitään aluksi ympyränkaari, joka kulkee 0:n, ζ_{2i-1}:n ja ζ_{2i}:n läpi. Merkitään tämän kaaren toista leikkauspistettä reaaliakselilla b:nä. Osakuvauksen φ_i ensimmäinen vaihe on ylemmän puolitason automorfismi $\frac{z}{1-z/b}$, joka kuvaa käsiteltävän ympyränkaaren säteeksi origosta. Kaaren osa 0:sta ζ_{2i+2}:een kuvautuu janaksi, jonka päätepistettä merkitsemme d:llä. Voimme nyt kuvata puolitason ilman origon ja d:n välistä janaa viiltokuvauksen käänteiskuvauksella g_d^{-1}, jolloin saamme molemmat sovitettavista pisteistä negatiiviselle reaaliakselille.



Kuva 14: Zipperin φ_i.

Geodeesista ja viiltoalgoritmia mukaillen myös Zipperin φ_{n+1} (kuva 15) kuvaa Ω:n tähänastiset kuvapisteet ympyränkaaren rajaamalta alueelta puolitasoon. Kuten on Zipperille ominaista, kaaren ei tarvitse olla reaaliakseliin nähden kohtisuora, vaan sen määräävät 0, ylempään puolitasoon kuuluva ζ_{2n+1} ja reaaliakselille kuuluva ζ_{2n+2}. Ω:n kuvapisteet voivat jälleen olla joko kaaren sisä- tai ulkopuolella. Ottamalla nyt ylemmän puolitason automorfismin $\frac{z}{1-z/\zeta_{2n+2}}$ tämä alue kuvautuu origokeskeiseksi sektoriksi, jonka kulmaa merkitään α:lla. Kierretään sektoria niin, että sen oikea reuna asettuu positiiviselle reaaliakselille ja vasen $e^{i\alpha}$:n suuntaiseksi. Korottamalla lopulta potenssiin $\frac{\pi}{\alpha}$ alue avautuu ylemmäksi puolitasoksi.



Kuva 15: Zipperin φ_{n+1}.

4.2 Viiltokuvauksen käänteiskuvauksen numeerinen ratkaiseminen

Geodeesinen algoritmi voitiin suorittaa kokonaisuudessaan suljetussa muodossa. Viiltoalgoritmin ja Zipperin φ_i-tyypin osakuvaukset jäivät vielä kesken, koska funktio 30 pitää vielä ratkaista käänteisfunktionsa kautta. Käytämme tähän Newtonin menetelmää, kuten [MR06]:ssa ehdotetaan. Samassa lähteessä on todistettu menetelmän suppeneminen $\mathbb{C}$:n osajoukossa ja ohjeistettu lukija todistamaan loput.

Newtonin menetelmällä approksimoidaan yhtälön $f(z) = 0$ ratkaisevaa z:aa, kun f ja f' tunnetaan. Iterointi aloitetaan jostakin alkuarvauksesta z_0 ja lopetetaan, kun z_n on riittävällä toleranssilla f:n juuri, eli $|f(z_n)| < \varepsilon_{tol}$. Yksi Newtonin menetelmän iteraatioaskel on

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}. \tag{31}$$

Yhtälö $g_d(z) = w$ saadaan ratkaistua asettamalla $f(z) = g_d(z) - w$, jonka derivaatta on g_d'. Yksinkertaisin menettely viiltokuvauksen kääntämiseksi Newtonin menetelmällä on siis käyttää funktiota ja sen derivaattaa

$$f(z) = C(z-p)^p(z+1-p)^{1-p} \; - \; w, \tag{32}$$
$$f'(z) = Cz(z-p)^{(p-1)}(z+1-p)^{-p}. \tag{33}$$

Tätä menettelyä käytetään [MR06]:ssa, mutta vain itseisarvoltaan riittävän suurilla w:n arvoilla. Muualla menetelmän suppeneminen paranee käyttämällä erinäisiä apufunktioita k. Käytetään k:ta yhtälöön $g_d(z) = w$, jolloin Newtonin menetelmällä ratkaistavaksi yhtälöksi tulee $k(g_d(z)) = k(w)$. Iteroitavat funktiot johdetaan kuten

yllä g_d:n, k:n ja w:n avulla lausuttavaan muotoon

$$f(z) = k(g_d(z)) - k(w) \tag{34}$$
$$f'(z) = g_d'(z)k'(g_d(z)). \tag{35}$$

Käytettävä apufunktio valitaan riippuen siitä, mihin w sijoittuu suhteessa d:een. Lähtöjoukko $\mathbb{H}$ jaetaan neljään osaan d:n perusteella.

Ensimmäinen joukko on

$$\Omega_{\text{Kaukana}} = \left\{ z \in \mathbb{H} : |z| > \frac{9}{8}|d| \right\}. \tag{36}$$

Joukko on ”parhaiten käyttäytyvä”, eikä apufunktiota tarvita. Newtonin menetelmään saadaan tarkempi alkuarvaus, kun otetaan muutama ensimmäinen termi f:n sarjakehitelmän äärettömässä:

$$z_0 = w + 2p - 1 + \frac{p(1-p)}{2w} + \frac{(1-2p)p(1-p)}{3w^2}. \tag{37}$$

Newtonin menetelmä käytännössä suppenee jo parin iteraation jälkeen.

Toinen joukko on

$$\Omega_{\text{Kärki}} = \left\{ z \in \mathbb{H} : |z-d| < \frac{1}{4}\mathfrak{Im}\, d \right\} \setminus \Omega_{\text{Kaukana}}. \tag{38}$$

Joukko on d-keskeinen kiekko, josta puuttuu 0:aa kohti osoittava säde. Tämän säteen kohdalla funktio käyttäytyy epäjatkuvasti. Käyttämällä apufunktiota $k(z) = \sqrt{(z-d)e^{-i\pi p}}$ alue aukeaa puoliympyräksi, joka kestää enemmän häiriöitä epäjatkuvuusalueellaan. Newtonin menetelmän alkuarvauksena käytetään w:tä.

Kolmas joukko on

$$\Omega_{\text{Oikea sektori}} = \{ z \in \mathbb{H} : \arg z < \arg d \} \setminus (\Omega_{\text{Kaukana}} \cup \Omega_{\text{Kärki}}), \tag{39}$$

Joukko on πp-kulmainen sektori, ja Newton-iteroinnissa se avataan puoliympyräksi apufunktiolla $k(z) = z^{\frac{1}{p}}$. Alkuarvauksena käytetään w:tä.

Neljäs joukko on

$$\Omega_{\text{Vasen sektori}} = \{ z \in \mathbb{H} : \arg z \geq \arg d \} \setminus (\Omega_{\text{Kaukana}} \cup \Omega_{\text{Kärki}}). \tag{40}$$

Joukko on edellisen vastinpari, $\pi(1-p)$-kulmainen sektori. Tämäkin joukko avataan puoliympyräksi käyttäen apufunktiota $k(z) = z^{\frac{1}{1-p}}$. Jos maalijoukoksi välttämättä halutaan ylemmän puolitason osajoukko, voidaan kuvata $(ze^{-i\pi p})^{\frac{1}{1-p}}$, mutta tällä ei ole Newtonin menetelmän suorituksen kannalta vaikutusta. Newtonin menetelmän alkuarvauksena käytetään w:tä.

Kaikista näistä esivalmisteluista huolimatta Newtonin menetelmässä voi ilmetä ongelmia. Empiirisesti havaittiin, että iteraatio voi joskus päätyä alempaan puolitasoon ja jopa supeta siellä toiseen g_d:n ratkaisuun, joka $\mathbb{H}$:n ulkopuolisena pisteenä pitää luonnollisesti hylätä. Käytämme ongelmaan [Watb]:n yksinkertaista, joskaan ei

täysin varmaa ratkaisua: Alempaan puolitasoon päätyvät iteraatit konjugoidaan heti. Toinen, mahdollisesti ensimmäiseen liittyvä ongelma on g_d:n hylkivät kiintopisteet p:ssä ja $p-1$:ssä. Marshallin alkuperäinen Zipper-ohjelma sisältää funktion `rnewt`, jonka tarkoitus on ilmeisesti iteroida reaaliluvuilla juuri näiden pisteiden läheisyydessä. Tätä työtä varten kirjoitetussa implementaatiossa ei käytetä mitään erityistä menetelmää näiden pisteiden ympäristössä toimimiseen.

4.3 Zipper-perheen MATLAB-implementaatio `zippit`

MATLAB [mat18] on MathWorksin kehittämä ohjelmointikieli ja -ympäristö. Se on erityisesti optimoitu laskemaan taulukkomaisesti tallennetulla datalla, mistä ohjelman nimikin tulee: MATLAB on lyhenne sanoista MATrix LABoratory. Eräs tällainen optimointi on *vektorointi*, jossa monta kertaa toistettavaa operaatiota tehostetaan prosessoritasolla. Vektoroitava kohta kirjoitetaan erityisellä syntaksilla, joka useimmissa tapauksissa korvaa ohjelmasta silmukkarakenteen. Koska tarkoituksena on suorittaa lukumäärältään suuren pistejoukon kaikille jäsenille sama operaatio, vektorointiominaisuutensa ansiosta MATLAB voi olla hyvä valinta Zipper-perheen implementointiin.

Työtä varten kirjoitettu implementaatio `zippit` on saatavilla GitHubista [Lei19b]. Paketti määrittelee MATLAB-luokan `Zippit`, joka on eräs tehokas tapa tallentaa Zipper-perheen mukaisia konformikuvauksia. `Zippit`-instanssin saa suoraviivaisimmin luotua kutsumalla funktiota `zippit`, joka ottaa syötteenään monikulmion ja käytettävän algoritmin. Tätä kutsuttaessa ohjelma laskee heti parametrijonon ζ_i ja tallentaa sen kompleksisena vektorina, jottei jonoa tarvitsisi ratkaista aina, kun kuvausta kutsutaan eri kontekstissa. Koska osakuvauksia on aina kolmea tyyppiä, tietorakenteeseen tallennetaan myös viittaukset parametria vaille valmiisiin osakuvauksiin. Lopulta sekä konformikuvaus että sen käänteiskuvaus tallennetaan funktioina, jotka suorittavat luvun 4.1 mukaisen osakuvausten yhdisteen.

Esimerkki 4.1 (Yksinkertainen Zippit-paketin käyttö)**.** Geodeesisen algoritmin mukainen konformikuvaus luodaan kirjoittamalla

```
geod = zippit([0, 1, 1+1i, 1i], 'Geodesic');
```

Kun tämä rivi on suoritettu, konformikuvaus on valmis käytettäväksi ja sen parametrit ovat tarvittaessa saatavilla vektorina `geod.zn`. Itse konformikuvaus on funktio `geod.forward`, joka ottaa yhden vektori- tai matriisiarvoisen parametrin. Kuvaus suoritetaan kaikille vektorin tai matriisin jäsenille. Käänteiskuvaus löytyy vastaavasti funktiona `geod.inverse`.

Algoritmien osakuvaukset ovat paketin päähakemistossa kukin erillisenä tiedostona. Osakuvausten tiedostonimissä on neljä alaviivalla (`_`) yhdistettyä osaa – esimerkiksi `phi_geod_fwd_1`. Etuliitteellä `phi` ilmaistaan, että kyseessä on osakuvaus. Algoritmiosa on lyhenne jostakin Zipper-perheen algoritmista: `geod`, `slit` tai `zip`. Osakuvauksen suunta tarkoittaa joko kuvausta tai käänteiskuvausta, *forward* tai *inverse*, jotka tiedostonimessä ovat lyhennetty muotoon `fwd` ja `inv`. Lopuksi merkitään indeksillä, onko kuvaus tyyppiä φ_1, φ_i, vai φ_{n+1}. Indeksi voi olla `1`, `i` tai `np1` (luetaan

”n plus 1”). Koska geodeesisella ja viiltoalgoritmilla on yhteisiä osakuvauksia, `slit`:n `1`- ja `np1`-kuvauksia ei ole implementoitu, jolloin osakuvaustiedostoja on yhteensä 14.

Osakuvausten lisäksi päähakemistosta löytyy ylemmän puolitason automorfismi `uhpaut`, jota moni osakuvaus käyttää. Riemannin kuvauksen viimeistelyä varten on myös funktiot `uhp2udisk` ja `udisk2uhp`, jotka kuvaavat ylemmän puolitason ja yksikkökiekon toisilleen. Molemmille voi antaa lisäparametrin, joka ilmoittaa $\mathbb{H}$:sta origoon kuvautuvan pisteen tai origon kuvapisteen $\mathbb{H}$:ssa.

Erityisesti osakuvauksia varten on ratkaistava kappaleessa 4.1 mainittu ongelma neliöjuuren konventioeroista. Tässä implementaatiossa useimmat neliöjuuret on tämän vuoksi kirjoitettu koodiin `sqrt(x)`:n sijaan `1i*sqrt(-x)`:na. Lukija voi tarkistaa, että $(-\pi/2, \pi/2]$-konventiolla laskettuna jälkimmäinen neliöjuuri tuottaa ylemmän puolitason pisteitä.

4.3.1 Viiltokuvauksen käänteiskuvaus MATLAB:ssa

Epäilemättä tärkein osakuvaus on `phi_slit_fwd_i`, joka on osakuvauksista ainoa numeerisesti ratkaistava ja johon pohjautuu kaksi algoritmia. Käytännössä koko luku 4.2 on koottu tähän funktioon, jota myös `phi_zip_fwd_i` kutsuu.

Tiedämme tarvitsevamme Newton-iteraatiota myöhemmin, joten määrittelemme sitä suorittavan aliohjelman heti tiedoston alkuun.

```
function zn = errcorr(zn)
    % Yksinkertainen virheenkorjaajafunktio
    err = imag(zn) < 0;
    zn(err) = conj(zn(err));
end

function z = newton(ff, dff, z, w, corr)

    % Korjausfunktion valinta
    if corr
        corrfn = @errcorr;
    else
        corrfn = @(x) x;
    end

    % Operoitavat pisteet
    op = ((imag(z) < 0) | abs(ff(z, w)) > tol);

    for ii=1:100
        % Newton-iteraatti
        z(op) = corrfn( z(op)-ff(z(op), w(op))./dff(z(op), w(op)) );

        % Poistetaan supenneet pisteet operoitavista
        op(op) = abs(ff(z(op), w(op))) > tol;

        % Jos kaikki on supennut, poistutaan funktiosta
        if isempty(find(op, 1))
            return;
        end
```

```
    end

    warning(['Some points did not converge in Newton''s method']);
end
```

Ensimmäinen funktio on luvun 4.3 lopussa mainittu virheenkorjausfunktio vektoroituna. Toinen funktio on itse Newton-iteraattori, jonka parametrit `ff`, `dff` ja `z` ovat iteraatioon liittyvä funktio, derivaatta ja alkuarvaus. Tarkoituksena on ratkaista pisteiden `w` alkukuvat. Lisäksi viimeinen parametri `corr` kertoo tosi/epätosi -arvollaan, sovelletaanko virheenkorjausfunktiota vai ei. Myöhemmin kutsuttava `corrfn` on pelkkä identiteettikuvaus, jos korjaus ei ole käytössä.

Luomme Newton-funktion sisäisesti loogisen matriisin, jolla tarkkaillaan yksittäisten pisteiden suppenemista. Kutsumme tätä `op`:ksi (”operoitavat”) ja merkitsemme siihen heti 0:lla pisteet, joita ei tarvitse iteroida lainkaan. Näin voi tapahtua alueen Ω_{Kaukana} pisteille, joten ehtolause ei yllättävää kyllä ole turha.

Newton-iteraatteja lasketaan jonkin ennalta määrätyn maksimimäärän verran. Joka askeleella lasketaan uusi Newton-iteraatti ja merkitään supenneet pisteet `op`-matriisiin, kumpikin vektoroidusti. Jos kaikken pisteiden havaitaan supenneen, poistutaan silmukasta. Suppenemattomista pisteistä nostetaan varoitus MATLAB:n `warning`-funktiolla.

Newtonin menetelmän lisäksi on hyvä määritellä ennalta viiltokuvaus ja sen derivaatta. Merkitään viiltokuvauksen parametria $d \in \mathbb{H}$ ja olkoon w MATLAB:n kompleksinen matriisi, joka sisältää käänteiskuvattavan pistejoukon. Tällöin viiltokuvauksen parametrit lasketaan

```
p = angle(d) / pi;
C = abs(d) / (p^p * (1-p)^(1-p));
```

ja funktiot ovat vektoroidussa muodossa

```
f  = @(z) (z-p).^p .* (z+1-p).^(1-p);
df = @(z) (z-p).^(p-1) .* (z+1-p).^-p .* z;
```

Funktioista puuttuu kerroin C, koska ratkaisemme kuvaukset muodossa $(z-p)^p(z+1-p)^{1-p} = w/C$. Tämän muutoksen tehtyämme myös parametri d ja pisteet w on skaalattava:

```
d = d/C;
w = w/C;
```

Siirrytään seuraavaksi valmistelemaan $\mathbb{H}$:n osittelua. Luodaan neljä w:n kokoista loogista matriisia `lrCond`, `ntCond`, `rsCond` ja `lsCond`. Nämä vastaavat yhtälöissä 36, 38, 39 ja 40 määriteltyjä $\mathbb{H}$:n osia ja niillä määrätään, minkä säännön mukaan kutakin pistettä käsitellään. Yksinkertainen tapa kirjoittaa ehdot MATLAB:lla on:

```
lrCond = abs(w/d) > 9/8;
ntCond = ~lrCond & (abs(w - d) < 0.25*imag(d));
rsCond = ~lrCond & ~ntCond & (angle(w) < p*pi);
lsCond = ~lrCond & ~ntCond & ~rsCond;
```

Ensimmäinen ehto (`lr` = ”large radius”), on puhtaasti geometrinen. Toinen ja kolmas ehto (`nt` = ”near tip” ja `rs` = ”right sector”) sisältävät viittauksia ylemmän puolitason geometriaan sekä sulkevat pois aiempiin joukkoihin kuuluvia pisteitä. Viimeinen ehto

(`ls` = ”left sector”) ottaa yksinkertaisesti jäljelle jäävät pisteet, eikä mitään tarvitse enää tarkistaa geometrisesti.

Loogisten matriisien muodostamisen jälkeen voimme ryhtyä iteroimaan. Suoritamme iteroinnin ensin kaikille `lrCond`-pisteille, sitten kaikille `ntCond`-, ja niin edelleen. Newton-iteroidut pisteet tallennetaan muuttujaan `phi`.

Pisteiden Ω_{Kaukana} ratkaisuun saadaan lisätarkkuutta iteroimalla z/w:n yli z:n sijaan. Tämä tapahtuu kirjoittamalla funktio ja derivaatta muotoon

```
ff = @(z, w) f(z .* w) ./ w - 1;
dff = @(z, w) df(z .* w);
```

Käytämme Newton-iteraatiossa tarkempaa alkuarvausta (37), ja skaalaamalla siitä saadut tulokset w:llä saamme oikeat lukuarvot. Alueessa ei tarvita virheenkorjausfunktiota.

Pisteiden $\Omega_{\text{Kärki}}$ apufunktio ja sen derivaatta kirjoitetaan MATLAB:lla

```
rota = exp(-1i*p*pi);
k  = @(u) sqrt( rota*(u - d) );
dk = @(u) rota / 2 ./ sqrt( rota*(u - d) );
```

Newton-iteraattorille annetaan funktioksi ja derivaataksi

```
@(z, w) (k(f(z)) - k(w))
@(z, ~) (df(z) .* dk(f(z)))
```

Alkuarvauksena toimii w, ja käytämme virheenkorjausfunktiota. Paketissa esiintyvä koodi näyttää hieman erilaiselta, mutta siihen on sisällytetty pieni optimointi ja toiminnallisuuden pitäisi olla sama.

$\Omega_{\text{Oikea sektori}}$ ja $\Omega_{\text{Vasen sektori}}$ eivät tuo kovin paljoa uutta. Alkuarvaus on jälleen w ja virheenkorjausfunktio on käytössä. Sisällytämme apufunktiot $k(z) = z^{\frac{1}{p}}$ ja $k(z) = z^{\frac{1}{1-p}}$ itseensä Newton-iteroitaviin funktoihin, koska näin toimiminen yksinkertaistaa laskutoimituksia. Oikeassa sektorissa iteroitavaksi funktioksi tulee

```
pwr = 1/p;
fp  = @(z) (z-p) .* (z+1-p).^(pwr-1);
dfp = @(z) (z+1-p).^(pwr-1) + (z-p)*(pwr-1).*(z+1-p).^(pwr-2);
```

ja vasemmassa

```
pwr  = 1/(1-p);
fpp  = @(z) (z-p).^(p*pwr) .* (z+1-p);
dfpp = @(z) p*pwr*(z-p).^(p*pwr-1) .* (z+1-p) + (z-p).^(p*pwr);
```

4.4 Huomioita ja yhteenveto

Zipper on nopea, mutta ei ratkaise muuta kuin Riemannin kuvauksen. Algoritmia ei juuri voi yleistää, koska iteraatio suoritetaan tasan yhtä reunaa pitkin, ja yleisemmissä tapauksissa reunoja on tyypillisesti useampia. Yhdesti yhtenäisiä vaikeampia alueita voisi kyllä iteroida niiden kaikkia reunoja pitkin, mutta tulos tuskin merkitsee mitään konformikuvausten kannalta. Pohjatyönä Zipper kuitenkin on erinomainen, ja siksi on vaikea uskoa, että kaikki sen johdannaiset olisivat jo löytyneet.

Kirjoittajan implementaatio `zippit` [Lei19b] on vapaasti muokattavissa. Siitä puuttuu vielä esimerkiksi erityiskäsittely hylkivien kiintopisteiden p ja $p-1$ kohdalla, ja sen toteuttaminen olisi seuraava suuri askel algoritmin tarkentamiseen. Myös lähteenä käytetystä paketista `ConformalMaps` [Watb] tällainen käsittely puuttuu. Algoritmit kohtaavatkin samanlaisia ongelmia (5.2) joko tästä tai muusta syystä, kun taas alkuperäinen Zipper ei.

5 Esimerkkejä

Lukijaa saattaa kiinnostaa Circlepackin, Zipperin tai jonkin muun konformikuvauksia tuottavan algoritmin testaus tai vertailu. Sen vuoksi tähän lukuun on koottu muutamia testitapauksiin liittyviä huomioita. Kirjoittajan testit pohjautuvat itse kirjoittamiinsa toteutuksiin `packit` [Lei19a] ja `zippit` [Lei19b], mutta lukija voi hyvin käyttää algoritmien vanhojakin toteutuksia, ohjelmoida omansa, tai kehittää aivan uudenlaisia menetelmiä.

On myös mainittava, mitä luku ei ole. Luvusta ei löydy numeerista vertailua tai nopeusvertailua. Numeerinen vertailu on jätetty pois, koska Circlepackin ja Zipperin lähtö- ja maalijoukkojen vertailu on hankalaa. Erityisesti pitäisi selvittää, voisiko Circlepackissa kiekkojen kuvauksen kiekoille toteuttaa johdonmukaisesti. Zipper ottaa kompleksiluvun ja antaa kompleksiluvun, mutta jos vastaavaa haluaisi tehdä Circlepackille, kiekon oikean paikan löydyttyä pitäisi valita kiekkoon kuuluvalle pisteellekin oikea, konformisuusehdon toteuttava paikka. Circlepackia käsittelevät julkaisut eivät näytä tuovan asiaa käsittelyyn, joten luultavasti Circlepackin ratkaisu tarkkuusongelmaan on yksinkertaisesti kasvattaa resoluutiota. Algoritmien nopeusvertailu on myös jätetty pois Circlepackin takia. Circlepack ei pärjää Zipperille, jos korkeamman resoluution ratkaisuja ei johdeta matalampiresoluutioisista (kuten `packit` ei johda). Tulevat kappaleet keskittyvätkin esittelemään, millaisia esimerkkejä kannattaa testata ja miksi.

5.1 Neliö

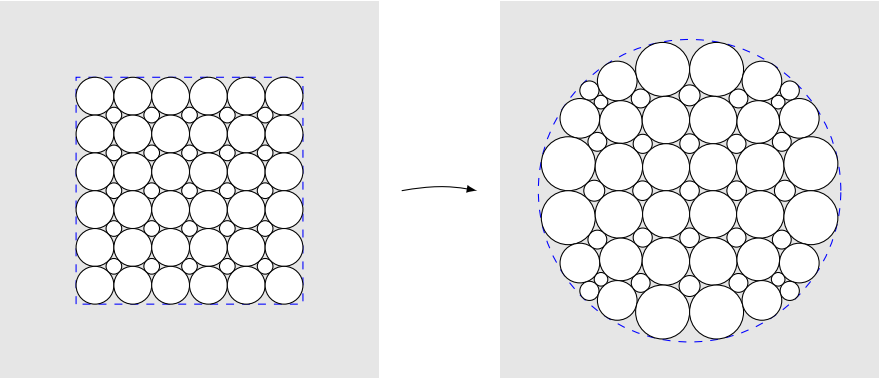
Algoritmien testaus tai vertailu on hyvä aloittaa yksinkertaisesta muodosta, josta räikeimmät virheet näkyvät heti epäsäännöllisyyksinä. Tällainen muoto on esimerkiksi neliö, jonka konformikuvauksen ympyrälle pitäisi olla kauniin symmetrinen, kun keskipiste kuvataan origoon.

Käsittelemme ensin Circlepackin. Standardimenettely olisi täyttää muoto kuusikulmaisella penny packilla. Neliön sisällä kuusikulmainen pakkaus kuitenkin hukkaisi paljon tilaa ja kuviointien säännöllisyydet eivät vastaisi toisiaan. Siksi käytämmekin ruudukkomaista pakkausta, jossa neliön sisään pakataan $(n-1)^2$ pientä ja n^2 isoa ympyrää. Jos neliön sivun pituus on 1, ison ympyrän halkaisija on $\frac{1}{n}$, ja pieni ympyrä sovitetaan jokaisen neljän ison ympyrän väliin jäävään tilaan.

Jäljellä on enää 3.3.1:n ohjeiden seuraamista. Pakkauksen kombinatoriikka siirretään hyperboliseen geometriaan ja reunimmaisille ympyröille asetetaan ääretön säde. Alku- ja lopputilanne tapauksessa $n = 6$ ovat esitetty kuvassa 16.

Asetelman kannalta kiinnostavia alueita ovat neliön kulmat, joiden ympäristöt – kolmiomaiset, ympyröiden peittämättömät alueet – katoavat kiekkoon siirryttäessä. Pisteiden kuvautumista näiltä alueilta ei voi järkevästi ratkaista muuten kuin resoluutiota kasvattamalla. Iteraattien laskemisessa voisi myös hyödyntää neliön symmetriaa, mutta säteiden päivittämisessä muuten kuin paikallisesti saattaa olla vaikutuksia lineaariseen suppenemiseen. On siis ehkä parempi varmistua lineaarisuudesta, jos kiihdytysmenetelmät ovat käytössä.

Zipperiä varten valmistellaan aluksi neliön kärkipisteet kompleksisena vektorina:



Kuva 16: Neliön Riemannin kuvaus Circlepackilla ratkaistuna.

`[0; 1; 1+1i; 1i];`. Peräkkäisten kärkien väliin lisätään pariton määrä n pisteitä tasaisin välimatkoin. Määrä riippuu halutusta tarkkuudesta, etenkin kärkien lähistöllä, ja parittomuus auttaa Zipperiä kääntämään reunat algoritmille edullisimmissa kohdissa. Samasta syystä viimeinen valmisteluaskel on muuttaa alkioiden järjestystä syklisesti: Järkevintä on hypätä pariton määrä askelia eteenpäin ja aloittaa siitä. Yksinkertainen esimerkki tällaisesta Zipperille annettavasta syötteestä olisi

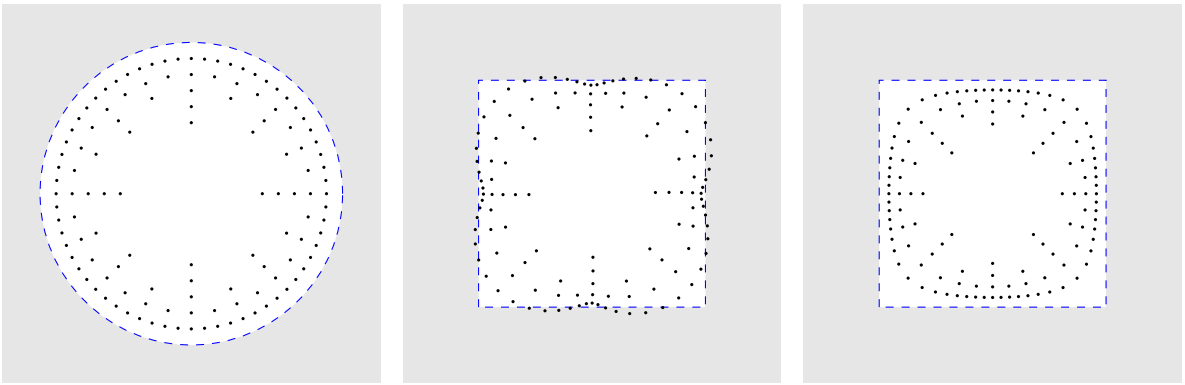
```
[0.5; 1; 1+0.5i; 1+1i; 0.5+1i; 1i; 0.5i; 0];
```

Vaikka Zipperkin suoritetaan mielivaltaiselta alueelta ympyrälle, sitä koskevissa julkaisuissa konformikuvaukset esitetään tyypillisesti käänteiskuvauksen kautta. Carlesonin ruudukko on yksikkökiekon säteittäinen ruudukko, joka reunaa lähestyessä muuttuu hienojakoisemmaksi. Kuvassa 17 on esitetty neliön Riemannin kuvaukseen liittyviä Carlesonin ruudukon solmupisteitä kuvausta ennen ja jälkeen. Keskimmäisessä kuvassa näkyy selvästi, kuinka Zipper välivaiheenaan suoristaa kulmiin muodostuvat kaaret. Lisäämällä janoille pisteitä tämän välivaiheen aiheuttamat virheet paikallistuvat lähemmäksi kulmia, jolloin myös kokonaisuus vaikuttaa onnistuneemmalta.

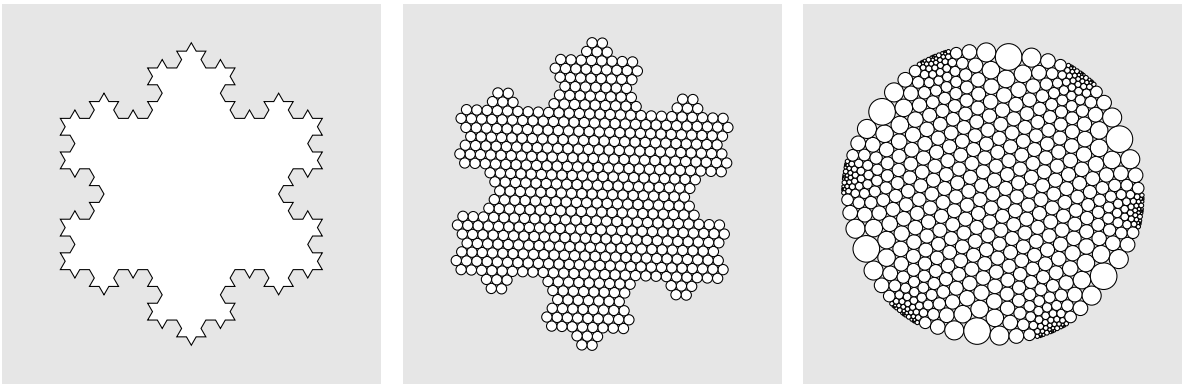
5.2 Kochin lumihiutale

Kun yksinkertainen esimerkki toimii, voi olla paikallaan kokeilla monimutkaisempaa, mutta edelleen säännöllistä aluetta. Fraktaalimainen Kochin lumihiutale sopii näihin vaatimuksiin. Kochin lumihiutale luodaan iteratiivisesti aloittamalla kolmiosta ja lisäämällä jokaisen janan keskimmäiseen kolmannekseen 60°:n ”kiila”. Fraktaalina Kochin lumihiutale muodostuisi toistamalla iteraatiota loputtomasti, mutta meille tarkoituksenmukaisempaa on katkaista iteraatio n:n toiston jälkeen.

Circlepackilla pääsemme nyt käyttämään penny packia, mutta joustamme hieman tavanomaisista säännöistä. Asettamalla euklidisessa geometriassa ympyrät monikulmion kärkiin ja jatkamalla kuviointia sisäpisteisiin penny pack asettuu lumihiutaleelle todella säännöllisesti, vaikkakin tällöin tingimme vaatimuksesta asetella ympyrät kokonaisuudessaan monikulmion sisälle. Lähtöjoukon säännöllisyys voisi olla jälleen hyödynnettävissä säteitä iteroidessa, jos lineaarisesta suppenemisesta huolehditaan.



Kuva 17: Neliön Riemannin kuvaus Zipperillä ratkaistuna. Vasemmalla Carlseonin ruudukon solmupisteet yksikkökiekossa ja keskellä annetun esimerkin tuottama ratkaisu. Oikealla ennen suorittamista jokaiselle janalle on lisätty tasaisesti 99 pistettä.

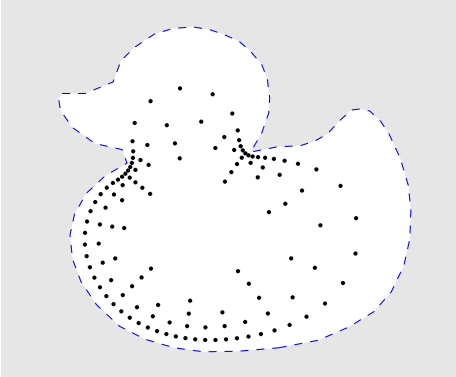


Kuva 18: Vasemmalla 3. sukupolven Kochin lumihiutale. Keskellä ja oikealla Kochin lumihiutaleen Riemannin kuvaus Circlepackilla ratkaistuna.

Hieman yllättäen uudemmat Zipper-implementaatiot joutuvat ongelmiin Kochin lumihiutaleen kanssa. Tietyillä asetuksilla ne eivät saa ratkaistua lumihiutaleen keskustan paikkaa kiekossa, ja ongelman lähde saattaa olla reunan suuri määrä 60°:n ja 120°:n käännöksiä. Ongelma ilmenee sekä [Lei19b] että [Watb]:lla, ja molempien ohjelmoijat ovat tietoisia tästä – ensimmäinen implementaatio on tämän työn kirjoittajan, ja jälkimmäisestä on tehty merkintä paketin issue trackeriin [wata]. Viimeksimainitussa lähteessä myös kerrotaan Marshallin alkuperäisen implementaation pystyvän ratkaisemaan tehtävän samoilla asetuksilla, joten kummaltakin uudelta toteutukselta saattaa puuttua osa vanhaa ohjelmaa tai Fortran on kestävämpi epätarkkuuksia vastaan.

5.3 Mitä seuraavaksi?

Näiden esimerkkien jälkeen riippuu algoritmien ominaisuuksista, millaisilla muodoilla niitä testaa. Circlepackille kannattanee antaa erilaisia monikulmioita penny packin kautta prosessoituna. Kuvion sisään mahtuvien ympyröiden määrä vaikuttaa toisaalta algoritmin nopeuteen, toisaalta muuttujien vapausasteeseen. Zipperissä taas kaikki riippuu reunan muodosta, jota voi vaihdella testitapauksien välillä äkkinäisesti kääntyilevistä tasaisempiin.



Viitteet

[Car98] Constantin Caratheodory: *Conformal Representation*. Dover Books on Mathematics. Dover Publications, 1998, ISBN 9780486400280. https://books.google.fi/books?id=dDG8ffQ-kNkC.

[COS17a] Charles Collins, Gerald L. Orick ja Kenneth Stephenson: *GOPack*, helmikuu 2017. https://se.mathworks.com/matlabcentral/fileexchange/61472-kensmath-gopack.

[COS17b] Charles Collins, Gerald L. Orick ja Kenneth Stephenson: *A linearized circle packing algorithm*. Computational Geometry, 64, maaliskuu 2017.

[CS02] Charles R. Collins ja Kenneth Stephenson: *A Circle Packing Algorithm*. Computational Geometry: Theory and Applications, 25:233–256, 2002.

[DS95] Tomasz Dubejko ja Kenneth Stephenson: *Circle packing: experiments in discrete analytic function theory*. Experiment. Math., 4(4):307–348, 1995. https://projecteuclid.org:443/euclid.em/1047674391.

[Dub93] Tomasz Dubejko: *Branched Circle Packings, Discrete Complex Polynomials, and the Approximation of Analytic Functions*. väitöskirja, 1993. AAI9404575.

[Gan08] Suman Ganguli: *Conformal Mapping and its Applications*. Department of physics, University of tennessee, knoxville, TN, 37996, 2008.

[Lei19a] Tatu Leinonen: *packit*, 2019. https://github.com/copyrite/packit.

[Lei19b] Tatu Leinonen: *zippit*, 2019. https://github.com/copyrite/zippit.

[Lut10] Bjørnar Steinnes Luteberget: *Numerical approximation of conformal mappings*. Pro Gradu -työ, Institutt for matematiske fag, 2010.

[Mar] Donald E. Marshall: *Numerical conformal mapping software: zipper*. https://sites.math.washington.edu/~marshall/zipper.html.

[mat18] *MATLAB R2018a*, 1984-2018. https://mathworks.com/products/matlab.html.

[MR06] Donald E. Marshall ja Steffen Rohde: *Convergence of the Zipper algorithm for conformal mapping*. ArXiv Mathematics e-prints, toukokuu 2006.

[Olv18] Peter J. Olver: *Complex analysis and conformal mapping*. 2018.

[RS87] Burt Rodin ja Dennis Sullivan: *The convergence of circle packings to the Riemann mapping*. J. Differential Geom., 26(2):349–360, 1987. https://doi.org/10.4310/jdg/1214441375.

[Rud87] Walter Rudin: *Real and Complex Analysis, 3rd Ed.* McGraw-Hill, Inc., New York, NY, USA, 1987, ISBN 0070542341.

[SB07] Nathan Sidoli ja J. Berggren: *The Arabic version of Ptolemy's Planisphere or flattening the surface of the sphere: text, translation, commentary.* SCIAMVS, tammikuu 2007.

[Ste] Kenneth Stephenson: *CirclePack web page.* http://www.math.utk.edu/~kens/CirclePack/.

[VR14] Martin Vermeer ja Antti Rasila: *Maailman kartta.* Tähtitieteellinen Yhdistys URSA Ry, 2014.

[wata] *Unexpected and idiosyncratic failures to converge.* https://github.com/sswatson/ConformalMaps.jl/issues/1.

[Watb] Samuel S. Watson: *ConformalMaps.* https://github.com/sswatson/ConformalMaps.jl.