Numerical approximation of conformal mappings

Bjørnar Steinnes Luteberget

Master of Science in Physics and Mathematics
Submission date: July 2010
Supervisor: Kari Hag, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Problem Description

Examine and present different numerical methods for approximating conformal mappings.

Assignment given: 08. February 2010
Supervisor: Kari Hag, MATH

Abstract

A general introduction to conformal maps and the Riemann mapping theorem is given. Three methods for numerically approximating conformal maps from arbitrary domains to the unit disc are presented: the Schwarz-Christoffel method, the geodesic algorithm and the circle packing method. Basic implementations of the geodesic algorithm and the circle packing method were made, and program code is presented. Applications of these numerical methods to problems in physics and mathematical research are briefly discussed.

Preface

This thesis will explore the world of conformal mappings and the techniques available to numerically approximate the mappings between arbitrary simply connected subsets of the complex plane, as promised by Riemann's mapping theorem. I chose this subject for my thesis after exploring complex analysis and doing a project work on circle packing. While doing this project work, I made a computer program for computing conformal mappings from a simply connected polygon in the plane to the unit disc to demonstrate some properties of the circle packing theory. This led me to look into the other methods available for computing such mappings. The section on the circle packing method in this text was originally written for a previous project work done in 2009.

I wish to thank my advisor Kari Hag for good help. With her wide knowledge about complex analysis and people's areas of expertise both at NTNU and around the world, she could always help me with my questions or know where to send me. This led me to contact Donald E. Marshall about his Zipper program and the geodesic algorithm, and I was so lucky to get the opportunity to visit him at the University of Washington. This visit gave me good insight into these algorithms, and Donald's general knowledge about the field of conformal mappings greatly influenced this thesis. This is also why this text is written in English instead of Norwegian, like my project work.

The completion of this master thesis concludes my studies at the program for industrial mathematics at NTNU. This study program has very much taken care of my interests and given me a good background in mathematics, numerics, computer programming and general enginnering related problem solving. Especially the courses in complex analysis, numerics and data visualization have defined my scientific field of interest and influenced this text.

Bjørnar Steinnes Luteberget
Trondheim, July 2010.

Contents

Introduction 1

1.1 Conformal maps

A conformal map is a transformation of the complex plane that preserves local angles. The maps are functions $f : A \to B$ with $A, B \in \overline{\mathbb{C}}$ that are holomorphic and have non-zero derivative. $\overline{\mathbb{C}}$ is the extended complex plane, $\mathbb{C} \cup \{\infty\}$ The conformal properties of a function are important in complex analysis and many problems in physics and engineering [10].

Let γ_1 and γ_2 be curves so that $\gamma_1(t_1) = \gamma_2(t_2)$ and γ_1,γ_2 are regular (differentiable with $\gamma(t) \neq 0$) in these points. If the angle between $(f \circ \gamma_1)(t_1)$ and $(f \circ \gamma_2)(t_2)$ is equal to the angle between $\gamma_1(t_1)$ and $\gamma_2(t_2)$, then f is said to be conformal in $z_0 = \gamma_1(t_1) = \gamma_2(t_2)$, see figure 1.1.

This is equivalent to the fact that f is holomorphic in z_0 and $f'(z_0) \neq 0$.

1.2 Plotting complex functions

The graph of a real function g is a set with a member of the form $(x, g(x))$ for each element x of the domain. By plotting these tuples as points in the plane, we get the familiar plot of a real function. This method is not directly applicable to complex functions, of course, because for complex functions, each member of the domain and codomain are themselves composed of two real points and need a plane to be visualized. So, to make the equivalent of such a graph plot for a complex function, one would need four dimensions to represent the two dimensions of the domain and the two dimensions of the codomain in one point.

We do not have any good general purpose methods for plotting in four dimensions. The obvious way to do it is of course to make a three dimensional plot and let the last dimension be realized as a change of this three dimensional

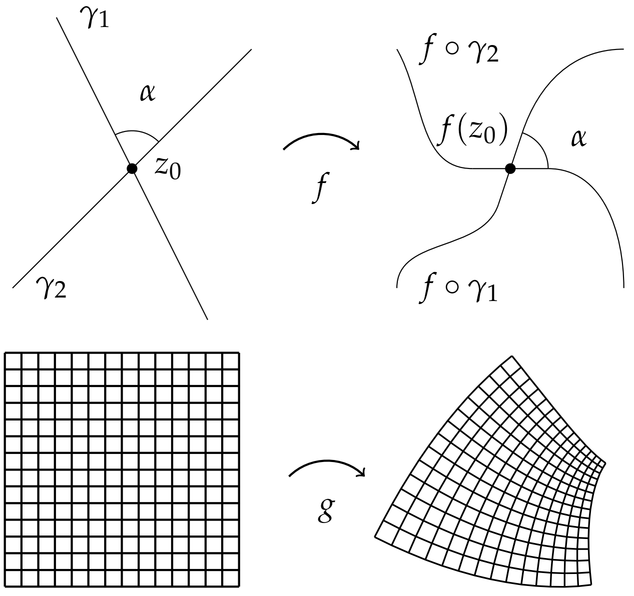


Figure 1.1: The map f preserves the angle α, so it is conformal in z_0. The map g preserves the angles in the grid (the lines are still meeting in right angles), so it is conformal.

plot over time. This approach can be used with success with other types of four-dimensional data, mostly when there are three spatial dimensions and temporal changes to them, but for visualizing complex functions it is unsuitable. Besides the difficulties already present in plotting in three dimensions, the separation into 3+1 dimensions does not fit well with the 2+2 dimensions in the graph.

The first bit of insight that can help one to find a better method is to represent complex numbers in polar form, $z = re^{i\theta}$. Plotting r and θ instead of x and y in $z = x + iy$ can open up some possibilities. The coloring of a plot can be thought of as adding another dimension, and by using a color space model with the dimension of **hue**, the phase dimension of either the domain or the codomain can be translated into hue, which is also an angle.

By looking into color models, we soon find out that a single color can identify a point in three dimensional space, at least inside a cylinder or cone. By making use of the color **value** as well as the hue, we can go back to plotting in two dimensions because we can use these two color dimensions to show a value for each point. This gives us the method of **domain coloring**, as described in [11]. This method avoids the problems that come with three dimensional drawings, but loses detail at either high or low values of $|f(z)|$ when the color value

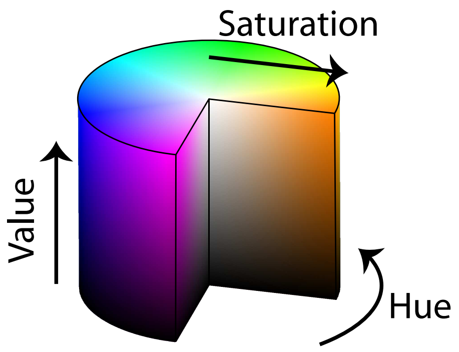


Figure 1.2: The HSV color model can assign a color value to each angle. Figure from [19].

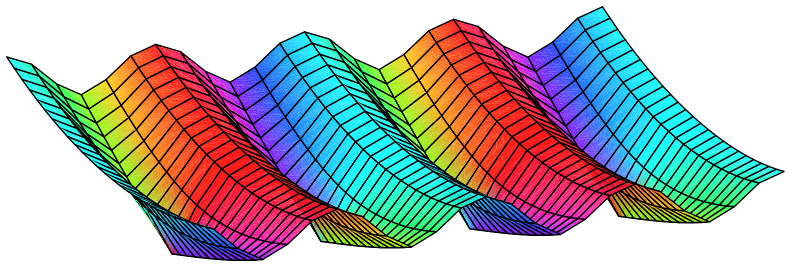


Figure 1.3: Three dimensional plot of the function $f(z) = \cos(z)$ on the domain $\mathfrak{Re}\, z \in [-2\pi, 2\pi]$ and $\mathfrak{Im}\, z \in [-0.75, 0.75]$. The height of the surface represents the modulus of f and the color represents the phase angle. Figure made with *Maple 13*, mathematics and modelling software by Maplesoft.

is low. Also, these color dimensions do not give the most intuitively understandable plots, but for some uses they are well suited.

1.3 Plotting conformal complex functions

In general, plotting complex functions is hard. The methods mentioned above can solve the problem in many cases. Fortunately, the defining property of conformal maps opens up possibilities that make this problem much more solveable. The fact that angles are preserved tells us that shapes must be the same at infinitesimal scale and we can therefore see conformal maps geometrically

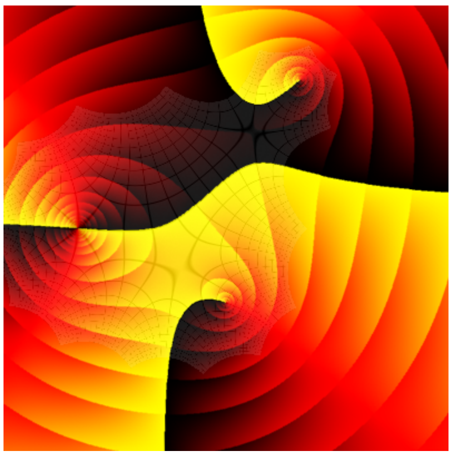


Figure 1.4: $f(z) = (z+2)^2(z-1-2i)(z+i)$ on the domain $\mathfrak{Re}\, z \in [-3,3]$ and $\mathfrak{Im}\, z \in [-3,3]$. Figure from [11].

as some distortion of shapes on the global scale. We can imagine a unit disc made out of some elastic material that you can stretch so that the boundary has the shape of the region you wish to map to or from. This makes is possible to visualize a conformal map just by drawing a picture of the effect it has on some image that you put over the domain in question.

The pictures that are generally used are grids.

- **Rectangular grid**

  If you put a rectangual grid over the domain Ω and map this grid to the unit disc $\mathbb{D}$, you can recognize the grid in the unit disc and because of its regularity, you can easily see how it has been stretched to fit.

- **Polar grid**

  When mapping to the unit disc, it can be illustrative to plot the map as the inverse over a polar grid in the unit disc.

- **Carleson grid**

  The Carleson grid is a grid that you put over the unit disc which gets finer and finer as you get closer to the boundary. This grid is then mapped to the domain Ω in question. This method emphasizes the behavior of the map close to the boundary, which is often the interesting place to examine and the place that needs the most accuracy. Bounded analytic functions have bounded differences on the boxes made with this grid.

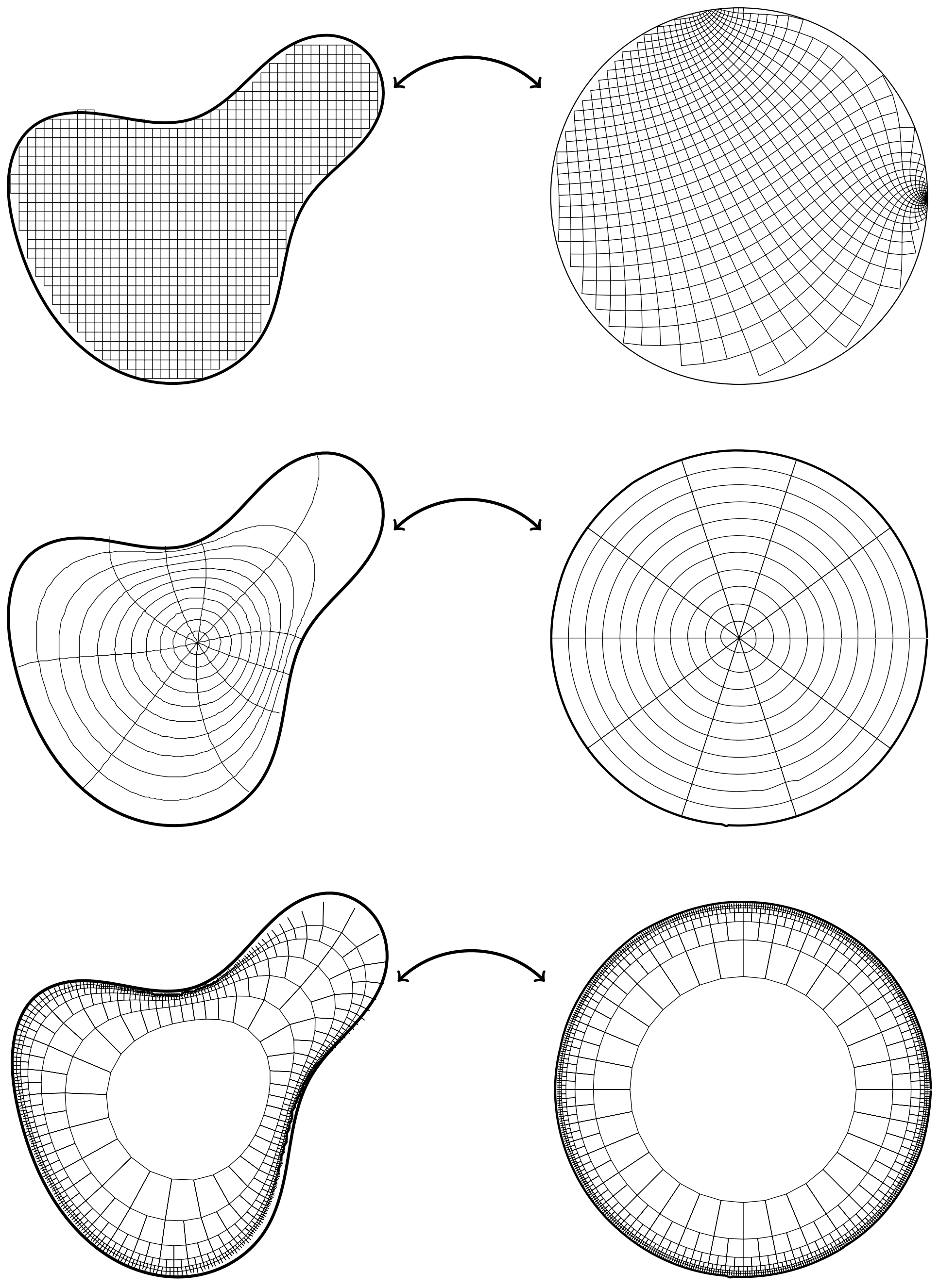


Figure 1.5: Examples of different grid types used to graphically present conformal maps. The first one is a rectangular grid mapped to the disc. The second one is a polar grid mapped from the disc to the desired domain. The third one is a Carleson grid mapped from the disc to the desired domain.

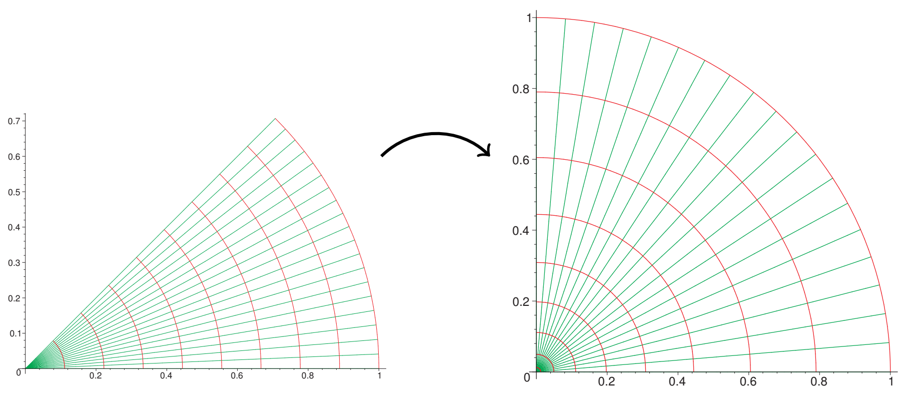


Figure 1.6: The map $f(z) = z^2$ applied to a grid of concentric circles and lines trough the origin.

See figure 1.5 for examples of what these grids can look like.

1.4 Examples of conformal mappings

Let us look at some example functions that are conformal. Many of the complex variants of the familiar elementary functions are conformal in most of the plane.

We can begin by considering $f(z) = z^2$. This function satisfies the requirements to be conformal except at the origin, where $f'(z = 0) = 0$. The geometric properties of this mapping become clearer if we write it in polar form: $f(z) = (re^{i\theta})^2 = r^2e^{i2\theta}$. We see that the distance from the origin is squared and the argument angle is doubled. If we plot this map as it is applied to a set of concentric circles around the origin and lines through the origin, we see this doubling of angles, see figure 1.6.

One can see that locally, for example for each grid cell, the shapes are mostly the same although they are moved, rotated and stretched. If we go all the way to the infinitesimal level, this represents the fact that angles are conserved. You can see that the right angles in the grid are still right angles after the mapping has been applied. To further illustrate this local preservation of shapes, we can consider a more detailed figure under the map $f(z) = z^2$. Figure 1.7 shows a photo of a clock under the map. Here we can see that the overall shape of the clock is distorted, but we can still recognize the photo because of the preservation of angles at infinitesimal scale.

Of course, other familiar analytical functions are also conformal where their

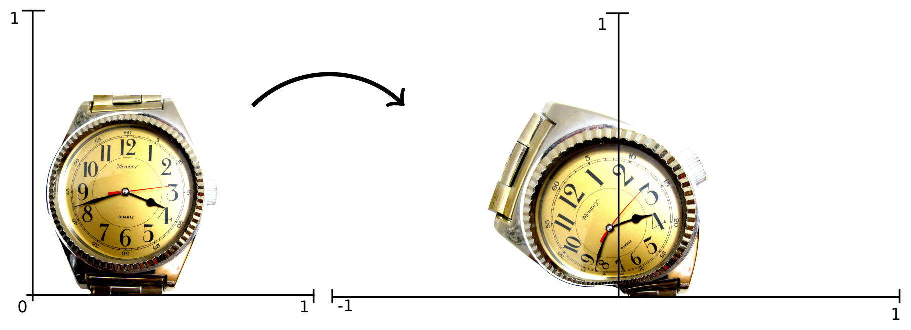


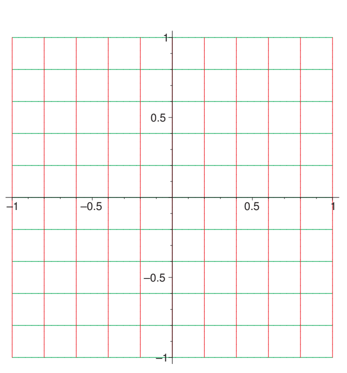
Figure 1.7: The map $f(z) = z^2$ applied to a photo of a clock. See appendix A.1 for program code.

derivatives are non-zero, like z^k, $\sin(z)$, $\cos(z)$, e^z. See figure 1.8 for examples.

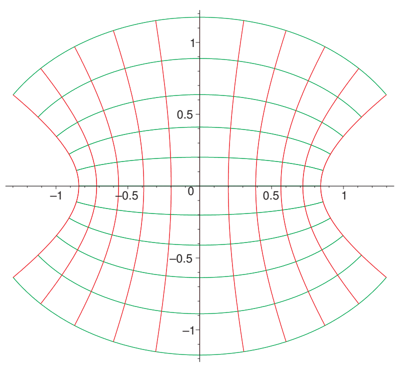
1.5 The goal of this text

The well-known theorem of complex analysis called the Riemann mapping theorem says that any two simply connected domains in the complex plane are conformally equivalent. This means that there always exists a conformal function that maps a simply connected domain to any other simply connected domain. The map is sometimes called the Riemann mapping, although in the complex analysis literature, it is so well-known that you can just take it for granted and call it a conformal map. Riemann's theorem is very central to complex analysis and can be quite surprising if you are new to the field. From basic complex analysis, we have learned that holomorphicity is something that is a very strict demand to make for a function, that is to say that holomorphic functions are very well behaved. It can therefore be suprising to learn that *any* simply connected domain can be mapped to any other. For example a simply connected open set bounded by a fractal curve of infinite length can be mapped by a conformal (and holomorphic) function to the unit disc.

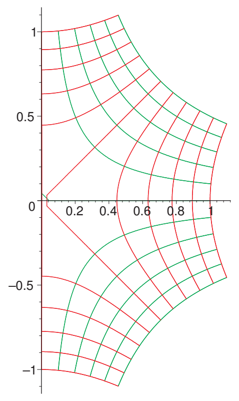
Although these unexpected maps are shown to exist by the Riemann mapping theorem, only a very few of them can actually be written explicitly using elementary functions. One can find a lot of the maps from half-planes, discs, triangles, strips and other special domains, but in general there is little chance that a domain you are interested in will be possible to map explicitly to where you want. The goal of this text is to show some of the different methods for numerically approximating such maps that makes it possible to release the full



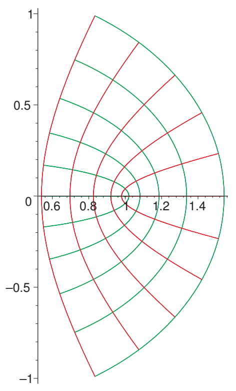
(a)



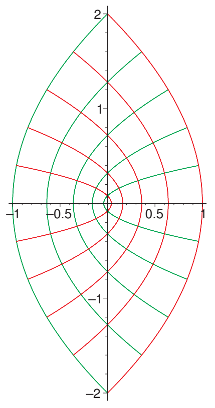
(b)



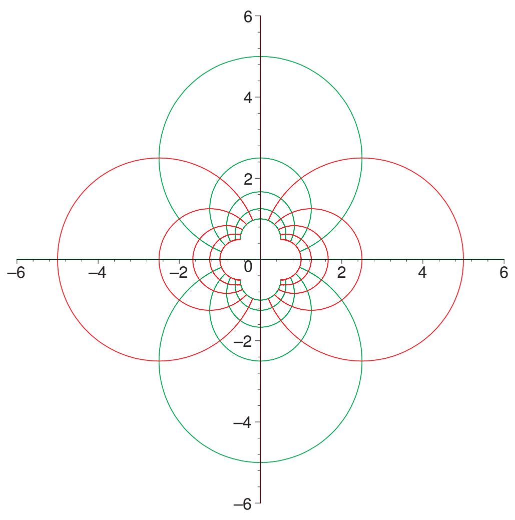
(c)



(d)



(e)



(f)

Figure 1.8: Examples of conformal maps of a rectangular grid. (a) The grid to be mapped. (b) The grid under the map $f(z) = \sin(z)$. (c) $f(z) = \sqrt{z}$. (d) $f(z) = \cos(z)$. (e) $f(z) = z^2$. (f) $f(z) = 1/z$.

potential of Riemann's promise.

The oldest method, the Schwarz-Christoffel transformation, was discovered and established as a possible method for calculating maps from arbitrary polygons long before any computers were invented and capable of finding these maps for complicated polygons. It was nonetheless important for the development of complex analysis as a significant branch of mathematics. The two other methods, the geodesic algorithm and the circle packing method, that are presented in this text are more recent discoveries that have also shed new light on mathematical research.

For numerically calculating Riemann maps, the Schwarz-Christoffel method is the most well-known and used method, but the Zipper program that uses a variant of the geodesic algorithm can also give fast and good results. Basic implementations of the geodesic algorithm and the circle packing method were made in the work with this thesis. See appendices B and C for program code.

Preliminaries 2

We will start out by giving an overview of some well-known facts of complex analysis that are essential to the algorithms that will be explained later.

2.1 Simply connected domains

A **domain** is a non-empty, connected, open subset of the extended complex plane $\overline{\mathbb{C}}$. A **connected domain** is a domain where every point can be connected to every other point by a curve in the domain. Such a domain can also be **simply connected** if such curves can always be continuously deformed to a point. This property of the curve is called null-homotopy [21]. See figure 2.1 for examples.

The domain in figure 2.2 is connected, but not simply connected, as can be seen by considering the closed curve γ that cannot be continuously deformed into a point.

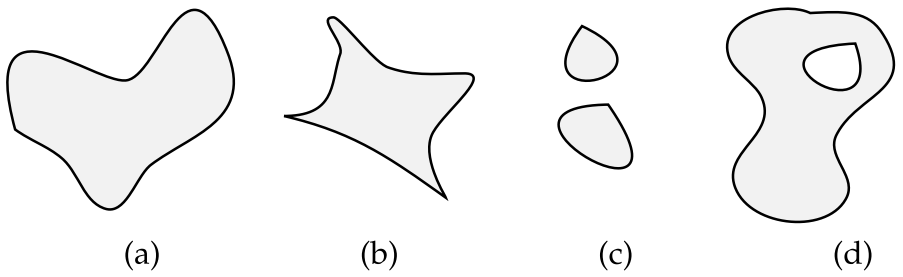


Figure 2.1: Examples of planar domains that are: (a) connected, (b) connected, (c) not connected, (d) connected, but not simply connected.

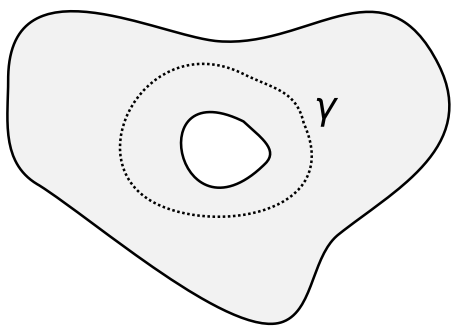


Figure 2.2: This domain is not simply connected because the dashed curve γ cannot be continuously deformed into a point.

Another definition of a simply connected domain that is convenient when working in the extended complex plane $\overline{\mathbb{C}}$ is that it is a domain Ω for which the complement in the extended complex plane is connected.

Yet another characterization of simply connected domains is needed for the proof of the Riemann mapping theorem given below. If f is an analytic, non-vanishing function on Ω and there exists a branch of $\log(f)$ on Ω, then Ω is simply connected.

2.2 Möbius transformations

A class of functions of special interest in the realm of conformal mappings are the Möbius transformations, or linear-fractional transformations. They are non-constant functions ϕ of the form

$$\phi(z) = \frac{az+b}{cz+d}, \tag{2.1}$$

where a, b, c, d are complex constants ($\in \overline{\mathbb{C}}$), and $ad - bc \neq 0$. We can see that the function must be holomorphic and we can also find its derivative:

$$\phi'(z) = \frac{ad-bc}{(cz+d)^2}. \tag{2.2}$$

Since this is always non-zero, the Möbius tranformations are conformal.

We can generalize the definition of a circle in $\overline{\mathbb{C}}$ to include straight lines. The lines can be regarded as circles passing through ∞ and therefore having infinite radius. This notion of a circle is sometimes called a clircle, for example in [21]. The Möbius tranformations map circles to circles, and this property will we useful later in this text.

Example 2.2.0.1
Let us say that we want a function mapping the upper half-plane to the unit disc. The real axis can be considered a circle in the generalized definition given above, so if we let the upper half-plane be the interior of this disc, we can find a Möbius tranformation that gives the wanted map. The function

$$f(z) = \frac{z-i}{z+i}$$

is a Möbius transformation. If we calculate the points

$$f(0) = \frac{-i}{i} = 1, \quad f(1) = \frac{1-i}{1+i} = -i, \quad f(-1) = \frac{-1-i}{-1+i} = i,$$

and use the fact that Möbius tranformations map circles to circles, we see that this function maps the unit disc to the real axis. Since $f(i) = \frac{i-i}{i+i} = 0$, we see that it maps the upper half-plane to the interior of the unit disc.

Theorem 1 Fixed points of Möbius tranformations [21]
A Möbius tranformation that is not the identity map has one or two fixed points.

Proof. We see that ∞ is a fixed point if and only if $c = 0$. The finite fixed points are the solutions in z of the equation $cz^2 + (d-a)z - b = 0$. If $c \neq 0$, this equation has either two distinct roots or one repeated root. If $c = 0$ then there is one root for $d \neq a$ and zero roots for $d = a$. In both cases, the transformation has one or two fixed points. □

Theorem 2 Three-fold transitivity of Möbius tranformations [21]
If z_1, z_2, z_3 are three distinct points of $\overline{\mathbb{C}}$ and w_1, w_2, w_3 are three distinct points of $\overline{\mathbb{C}}$, then there is a unique Möbius tranformation ϕ such that $\phi(z_i) = w_i$ for $i = 1, 2, 3$.

Proof. If two maps ϕ and ψ both have the required property, then $\psi^{-1} \circ \phi$ is a Möbius tranformation with three fixed points, so it must be the identity by the previous theorem. So $\phi = \psi$ and the map is unique.

We can write down a Möbius tranformation mapping z_1, z_2, z_3 to $\infty, 0, 1$,

$$\phi(z) = \begin{cases} \frac{(z-z_2)(z_1-z_3)}{(z-z_1)(z_2-z_3)} & \text{if } z_1, z_2, z_3 \text{ are all finite,} \\ \frac{z-z_2}{z_3-z_2} & \text{if } z_1 = \infty, \\ \frac{z_3-z_1}{z-z_1} & \text{if } z_2 = \infty, \\ \frac{z-z_2}{z-z_1} & \text{if } z_3 = \infty, \end{cases}$$

and since an inverse of a Möbius tranformation is a Möbius tranformation, we can find another map ψ from w_1, w_2, w_3 to the same points and compose $\psi^{-1} \circ \phi$. □

Theorem 3 Preservation of circles [21]
A Möbius tranformation maps generalized circles onto generalized circles

2.3 Riemann mapping theorem

One of the most important theorems that makes conformal mappings interesting is the mapping theorem from G. F. B. Riemann's PhD thesis written in 1851. It can be stated as

Theorem 4 Riemann mapping theorem
All simply connected domains $\Omega \subset \mathbb{C}$ that are not the whole of $\mathbb{C}$ are conformally equivalent.

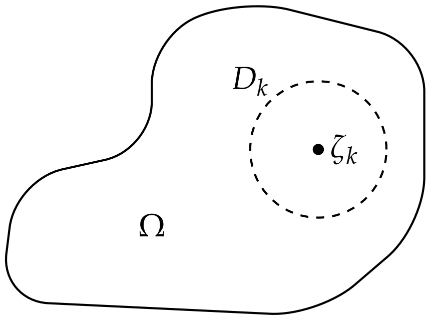
This means that there exists a holomorphic, bijective (in other words, conformal) mapping between any two simply connected subsets of $\mathbb{C}$. The theorem is often stated by promising the existence of a conformal function $f : \Omega \to \mathbb{D}$ that maps a simply connected domain Ω onto the unit disc $\mathbb{D}$. This is of couse equivalent to saying that all simply connected domains can be mapped to eachother. If you have two such domains Ω_1 and Ω_2, find f_1 and f_2 that map each of the domains to the unit disc. Conformality is preserved under inversion and composition, so $g = f_1 \circ f_2^{-1}$ must be a conformal map between the two domains.

2.4 A proof of the Riemann mapping theorem

The Riemann mapping theorem is the basis for all the results discussed in this text, so a proof due to Ahlfors will be presented [1]. It begins with Montel's theorem.

Theorem 5 Montel' theorem
Let $\mathfrak{F}$ be a family of holomorphic functions $f : \Omega \to \mathbb{D}$ where Ω is a domain in $\mathbb{C}$ and $\mathbb{D}$ is the unit disc. Then every sequence $(f_n)_{n=1}^{\infty}$ of functions in $\overline{\mathfrak{F}}$ has a locally uniformly convergent subsequence.

Proof. We start by showing that it suffices to prove that (f_n) has a locally uniformly convergent subsequence in the disc for any open disc $D_k \subset \mathbb{C}$ whose closure is in Ω.

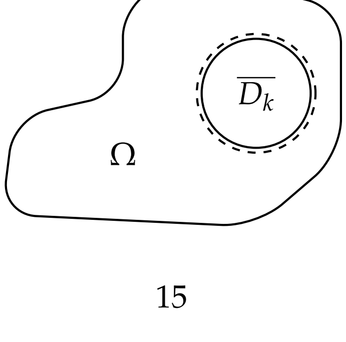


We can cover Ω with a countable set of discs so that $\Omega = \bigcup_{k=1}^{\infty} D_k$. For example, for each rational point ζ_k in Ω let D_k be the open disc with center on the point and some radius $r_k < \operatorname{dist}(\zeta_k, \partial\Omega)$. Then, for each disc, we assume that the sequence (f_n) has a subsequence $f_{n_1}, f_{n_2}, \dots$ on D_k that converges uniformly on D_k. Let the sequence for the first disc D_1 be denoted $(f_{1,n})_{n=1}^{\infty}$, and for the second disc D_2, let $(f_{2,n})_{n=1}^{\infty}$ be a convergent subsequence of $(f_{1,n})_{n=1}^{\infty}$. Since $(f_{2,n})_{n=1}^{\infty}$ converges on D_2, and since it is a subsequence of $(f_{1,n})_{n=1}^{\infty}$ it also converges on D_1.

$$\begin{array}{llllll}
f_{1,1}, & f_{1,2}, & f_{1,3}, & f_{1,4}, & \dots, & \text{converges uniformly on } D_1. \\
f_{2,1}, & f_{2,2}, & f_{2,3}, & f_{2,4}, & \dots, & \text{converges uniformly on } D_2 \text{ and } D_1. \\
f_{3,1}, & f_{3,2}, & f_{3,3}, & f_{3,4}, & \dots, & \text{converges uniformly on } D_3 \text{ and } D_1, D_2. \\
\vdots, & \vdots, & \vdots, & \vdots, & \ddots & \qquad\qquad \vdots
\end{array}$$

If we continue in this fashion, we see that $(f_{k,n})_{n=1}^{\infty}$ converges for the k-th disc, and so the functions on the diagonal $g_n = f_{n,n}$ converge locally uniformly on all of Ω.

So now we can prove the theorem for a disc D_k. Since the closure $\overline{D_k}$ is in Ω, there exists an open disc in Ω containing $\overline{D_k}$. We can map this larger disc to the unit disc $\mathbb{D}$.



We now need to prove that $f_n : \mathbb{D} \to \mathbb{D}$ has a subsequence that converges locally uniformly. Let $\sum a_{n,k}z^k$ be the power series of f_n. Then,

$$|a_{n,k}| = \left| \frac{1}{2\pi i} \int_{C_r} \frac{f_n(z)}{z^{k+1}} \mathrm{d}z \right| \leq \frac{1}{2\pi} \frac{2\pi r}{r^{k+1}} = \frac{1}{r^k},$$

where $0 < r < 1$. We let $r \to 1$ and get that $|a_{n,k}| \leq 1$. Each sequence $(a_{n,k})_{n=1}^{\infty}$ for $k = 0, 1, 2, \ldots$ is bounded and has a convergent subsequence. By using the same diagonal argument as above, we find a sequence $(a_{n_j,k})_{n=1}^{\infty}$ that converges for each k. Let $a_k = \lim_{j\to\infty} a_{n_j,k}$ for $k = 0, 1, 2, \ldots$. We have $|a_k| \leq 1|$ for all k, so the power series $\sum_{k=0}^{\infty} a_k z^k$ converges for radius $R = \frac{1}{\limsup |a_k|^{\frac{1}{k}}} \leq 1$, which is in $\mathbb{D}$.

Finally, we need to show that $f_{n_j} \to f$ locally uniformly in the disc. Fix $r \in (0, 1)$. We will show that $f_{n_j}(z) \to f(z)$ uniformly for $|z| \leq r$.

$$\begin{aligned} |f(z) - f_{n_j}(z)| &= \left| \sum_{k=0}^{\infty} (a_k - a_{n_j,k}) z^k \right| \\ &\leq \sum_{k=0}^{m} |a_k - a_{n_j,k}| + 2 \sum_{k=m+1}^{\infty} r^k \\ &= \sum_{k=0}^{m} |a_k - a_{n_j,k}| + \frac{2r^{m+1}}{1-r}. \end{aligned}$$

So given $\epsilon > 0$, we choose m so large that $\frac{2r^{m+1}}{1-r} < \frac{\epsilon}{2}$ and then $j = j_0$ so large that $|a_k - a_{n_j,k}| < \frac{\epsilon}{2(m+1)}$ for $k = 0, \ldots, m$ whenever $j \geq j_0$. Then we get that

$$|f(z) - f_{n_j}(z)| < \epsilon \quad \text{for } |z| \leq r,$$

and the theorem is proved.

□

Theorem 6 Riemann Mapping Theorem
If Ω is a simply connected domain that is not the whole of $\mathbb{C}$, there exists a univalent holomorphic map of Ω onto the open unit disc $\mathbb{D}$.

Proof. Fix $z_0 \in \Omega$ and let $\mathfrak{F} = \{f : \Omega \to \mathbb{D} \mid f(z_0) = 0\}$ be the set of univalent holomorphic functions f.

First, we show that $\mathfrak{F} \neq \emptyset$. If there exists a bounded univalent holomorphic function in Ω, we can compose it with a Möbius transformation to the unit disc to get a function in $\mathfrak{F}$. Let $c \in \mathbb{C} \setminus \Omega$ be a point outside of Ω and consider a branch l of $\log(z-c)$. This branch exists since Ω is simply connnected. The function l is holomorphic and also univalent, since $l(z_1) = l(z_2) \Rightarrow z_1 - c = z_2 - c$. Also, $l(\Omega)$ is an open set and $l(\Omega) \cap \{l(\Omega) + 2\pi i\} = \emptyset$. We have

$$|l(z) - (l(z_0) + 2\pi i)| \geq \operatorname{dist}(l(z_0), \partial l(\Omega)) = \epsilon > 0,$$

so that

$$\left|\frac{1}{l(z) - l(z_0) - 2\pi i}\right| \leq \frac{1}{\epsilon},$$

and $(l(z) - l(z_0) - 2\pi i)^{-1}$ is bounded.

Let $\zeta = \sup\{|f'(z_0)| : f \in \mathfrak{F}\} > 0$, and let $(f_n)_{n=1}^{\infty}$ be a sequence from $\mathfrak{F}$ such that $|f_n'(z_0)| \to \zeta$. By Montel's theorem, this sequence has a locally uniformly convergent subsequence. Let us assume that (f_n) converges locally uniformly to a function f, because if it does not, then we can replace it with a subsequence that does. By Weierstrass' convergence theorem [21], f is holomorphic and $f_n' \to f'$ locally uniformly.

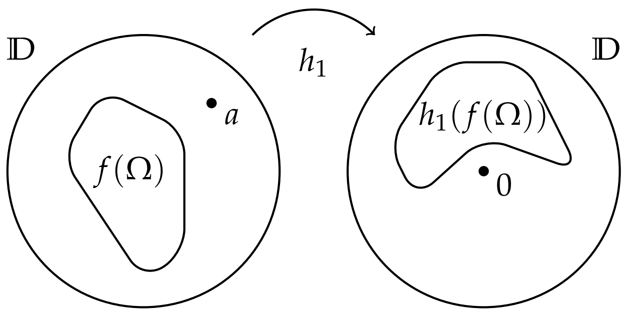
$$\lim f_n'(z_0) = f'(z_0) \Rightarrow \zeta = \lim |f_n'(z_0)| = |f'(z_0)|.$$

We see that f must be non-constant. It is also univalent, since each f_n is. We see that $|f(z)| \leq 1$ for $z \in \Omega$ and $f(z_0) = 0$. Since $f(\mathbb{D})$ is an open set, $|f(z)| < 1$ for $z \in \Omega$, and $f \in \overline{\mathfrak{F}}$.

Lastly, we need to show that $f(\Omega) = \mathbb{D}$. Suppose that this is not the case, $f(\Omega) \subsetneqq \mathbb{D}$. Let $a \in \mathbb{D} \setminus f(\Omega)$ and define h_1 by

$$h_1 = \frac{f - a}{1 - \overline{a} f}.$$

So $h_1 = A \circ f$ where $A : \mathbb{D} \to \mathbb{D}$ is a Möbius transformation mapping a to 0.



Since $a \notin f(\Omega)$, h_1 does not vanish on Ω and since Ω is simply connected, there exists a branch h_2 of $\sqrt{h_1}$ defined in Ω. h_2 must be a univalent holomorphic map of Ω to $\mathbb{D}$. Finally, define

$$g = \frac{h_2 - b}{1 - \bar{b}h_2},$$

where $b = h_2(z_0)$ so that $g(z_0) = 0$ and $g \in \mathfrak{F}$.

We can now do some calculations and find

$$f = \left(\frac{g + \alpha}{1 + \bar{\alpha}g}\right) g,$$

where $\alpha = 2b/(1 + |b|^2)$, which gives us

$$f'(z_0) = \alpha g'(z_0) \quad \Rightarrow \quad |f'(z_0)| < |g'(z_0)|,$$

since $\alpha < 1$.

We have reached a contradiction, and can conclude that $f(\Omega) = \mathbb{D}$. This shows the exitence of the socalled Riemann mapping, and shows that simply connected subsets of $\mathbb{C}$ are conformally equivalent. $\square$

2.5 Quasiconformality

Quasiconformal mappings are a generalization of conformal mappings where the preservation of angles is not exact, but has some bound on the distortion of angles. Such mappings can be defined by analogy to conformal maps by saying that conformal functions map infinitesimal circles to infinitesimal circles while quasiconformal functions map infinitesimal circles to infinitesimal ellipses with bounded eccentricity [7].

Let $f : D \to D'$ be a homeomorfism (continuous bijection with f^{-1} continuous), and $D, D' \subset \mathbb{C}$. For $z \in D \backslash \{\infty, f^{-1}(\infty)\}$ and $0 < r < \operatorname{dist}(z, \partial D)$, let

$$l_f(z,r) = \min_{|z-w|=r} |f(z) - f(w)|,$$
$$L_f(z,r) = \max_{|z-w|=r} |f(z) - f(w)|.$$

See figure 2.3. We then call

$$H_f(z) = \limsup_{r \to 0} \frac{L_f(z,r)}{l_f(z,r)}$$

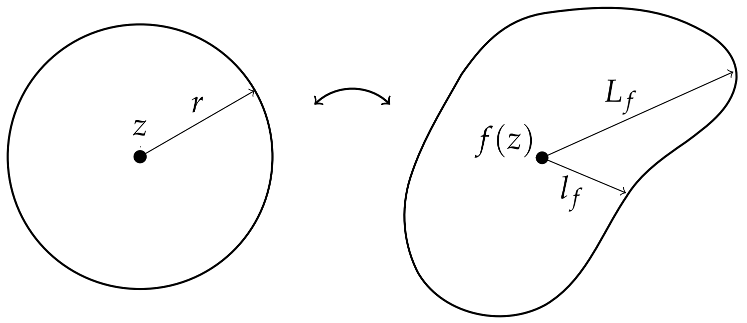


Figure 2.3: Greatest and smallest dilatation, l_f and L_f, of f.

the linear dilation of f at the point z.

A homeomorfism $f : D \to D'$ is K-quasiconformal for $1 \leq K < \infty$ if H_f is finite in $D \backslash \{\infty, f^{-1}(\infty)\}$ and

$$H_f(z) \leq K$$

almost everywhere in D [5].

f is 1-quasiconformal if and only if f or its complex conjugate, $\overline{f}$, is a conformal mapping [5]. We see that when L_f and l_f become equal, H_f and thereby K approach 1 and we must have a circle on the right hand side of figure 2.3 and so f becomes a conformal map by the first definition.

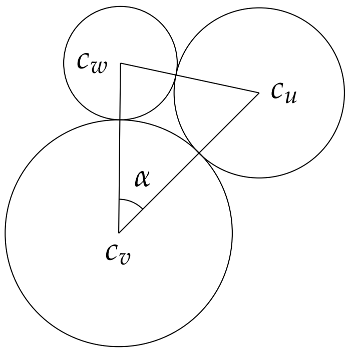
Quasiconformal mappings can be defined in other ways, for example by the moduli of families of curves, but this geometrical definition is best suited for the understanding of the circle packing method given in this thesis.

2.6 Circle packing

The theory of circle packing describes existence, uniqueness, calculation, manipulation, drawing and applications of configurations of circles where it is specified which of the circles that are tangent to eachother.

This specification can, in the simplest way, be done with an intersection graph where each node represents a circle and each edge represents the fact that two circles are tangent. In two dimensional geometry, such a graph is planar, which means that it can be drawn in the plane without any edges intersecting.

A circle packing is an assignment of radius and center to each of the circles in an intersection graph so that the tangency specifications in the graph are



(a)

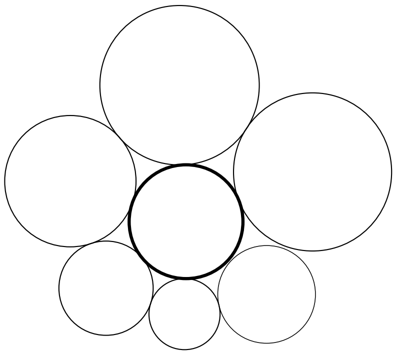

(b)

Figure 2.4: (a) Triangle between the centers of three mutually tangent circles. c_v is here the center circle and α is the angle inside it. (b) A **flower** made up from all the circles around a center circle. The circle packing requirement that the angles of the triangles to all the tangent circles from the center circle sums to 2π is fulfilled. This means that the circles are as close together as possible.

satisfied. This can be done in Euclidean, spherical or hyperbolic geometry. The basis for the theory of circle packing is the following theorem:

Theorem 7 Circle packing theorem (Koebe-Andreev-Thurston)
For any planar graph G there exists a circle packing in the plane whose intersection graph is isomorphic to G.

In the circle packing literature in general, G is said to be a simplical complex constructed from a triangulation [24], but for the purposes of this text we can let G be just an intersection graph representing a tangency pattern. The theorem was shown in spherical geometry as early as 1936 by Koebe, but was not paid much attention before William Thurston generalized the theorem and showed a connection to the Riemann mapping theorem in 1985 [23].

The calculations needed to find such a circle packing for a graph G are mostly about adjusting the list of radii $R(c_i)$ which assigns a radius to each circle c_i, which is a node in G. For three circles that are mutually tangent, one can find the angles in the triangle with vertices at the circle centers as a function of the radii by using the cosine law. For a central circle c_i, we can find the sum of the angles of all the tangent circles around it. We would then like the circles not to overlap, but also to cover the area around the center circle completely, so the sum should be 2π. This is called an univalent circle packing, as oppposed to letting this sum be $2\pi n$ with $n \neq 1$ in branch points. See figure 2.4(b).

A graph G is said to be a triangulation of a geometric object if the edges of the graph can be drawn to make triangles that cover the object. If G is a triangulation of a simply connected compact domain in $\mathbb{C}$, we say that G has the topology of a closed disc.

Theorem 8 Maximal packing for a closed disc
Let G be a triangulation of a closed disc. Then there exists an essentially unique circle packing P_K for G in $\mathbb{D}$ which is univalent, and whose boundary circles are tangent to the unit circle.

The fact that the boundary circles are tangent to the unit disc, means that there is no room for more circles and means that the circle packing covers all of $\mathbb{D}$. The fact that it is esssentially unique, means that the circle packing is preserved under (conformal) automorphisms of the codomain. The degree of freedom in choosing these automorphisms is the same as the degree of freedom in specifying the Riemann map. See section 2.8 for an explanation of the automorphisms in the hyperbolic geometry that will be used in this text.

This theorem gives us a one-to-one correspondance between an arbitrary circle packing, via its intersection graph G, to a circle packing that fills all of $\mathbb{D}$ and has the same intersection graph.

2.7 Discrete analytic function theory

Isomorphisms between circle packings are functions between two sets of circles that have isomorphic intersection graphs. These functions can be said to have their own **discrete** analytic function theory. The idea came from the many similarities one can find in the theory if circle packing and complex analytic function theory. For example, one can formulate a discrete version of Schwarz' lemma in the following way:

Theorem 9 Schwarz' lemma
Let R be a hyperbolic circle packing representing a closed disc K, and let R_K be the maximal packing. Then we have that $R(c_i) \leq R_K(c_i)$ for each circle c_i in K. If $R(c_i) = R_K(c_i)$ for at least one circle, then $R = R_K$ [23].

The theorem for the maximal packing of a closed disc can also be generalized to simply connected surfaces and to spherical and Euclidean geometry. This is called the discrete Riemann mapping theorem [23]. This theorem is what gives circle packing many of the other applications that makes the theory interesting.

It also turns out that the connection between the circle packing's discrete analytic function theory and the complex function theory is not just an analogy, but that the discrete circle packings can be used for approximating continuous objects in complex analysis. To compare with conformal mappings, who can be said to map infinitesimal circles to infinitesimal circles, circle packings map actual circles to actual circles [23]. One can then let the radii of these circles become very small and they will look more and more like continuous, conformal functions, and Rodin and Sullivan showed that this is acutally the case [20].

The isomorphisms between circle packings will only give you a map from each circle in the domain to a circle in the codomain. To extend this to a continuous function between the subsets of $\mathbb{C}$ that the circle packings represent, one can define the function for each triangle in the intersection graph as an affine transformation (linear transformation plus a translation). This gives us functions $f_r : \Omega \to \mathbb{D}$ for circle radius r.

Theorem 10 Rodin-Sullivan [24]
Let Ω be a compact simply connected subset of $\mathbb{C}$. Let f_r be defined as in the previous paragraph. Then we have

$$\lim_{r\to 0} f_r = F,$$

where $F : \Omega \to \mathbb{D}$ is the Riemann map. f_r converges uniformly to F on compact subsets of $\mathbb{D}$.

2.8 Hyperbolic geometry and the Poincaré model

Hyperbolic geometry can be modelled in the unit disc $\mathbb{D}$ by the Poincaré model which has arc length and surface element

$$\mathrm{d}s = \frac{2|\mathrm{d}z|}{1-|z|^2}, \quad \mathrm{d}s^2 = \frac{4\mathrm{d}x\mathrm{d}y}{(1-|z|^2)^2}, \quad |z| < 1. \tag{2.3}$$

The arc length element approaches infinty when $|z|$ approaches 2, so small Euclidean distances near the edge of $\mathbb{D}$ represent large hyperbolic distances. We write $d(x, y)$ for the hyperbolic distance between two points (complex numbers) x and y in the unit disc, and we can easily find distances on the positive x-axis:

Theorem 11 Hyperbolic distance on the real axis
Hyperbolic distance from the origin to the point x on the positive real axis in the unit disc is given by

$$d(0, x) = \log \frac{1+x}{1-x}. \tag{2.4}$$

Proof. We easily see that the shortest curve from 0 to x must be a straight line segment γ on the real axis. We can then use the arc length element to find

$$d(0,x) = \int_\gamma \mathrm{d}s = \int_\gamma \frac{2|\mathrm{d}z|}{1-|z|^2} = \int_0^x \frac{2\mathrm{d}x}{1-x^2} = \log\frac{1+x}{1-x}. \tag{2.5}$$

□

Even more useful is finding distances in the unit disc given hyperbolic distance. We find x given $d(0,x)$:

Corollary 1 Euclidean distance given hyperbolic distance on the real axis The disctane in the unit disc under the Poincaré model, given a hyperbolic distance r_h from the origin, is

$$l(r_h) = \frac{e^{r_h}-1}{e^{r_h}+1}. \tag{2.6}$$

Proof. We solve equation 2.4 for x. We let x be called l and $d(x,0)$ be called r_h. □

The automorphisms on $\mathbb{D}$ (bijective conformal maps from $\mathbb{D}$ to $\mathbb{D}$) are functions of the form

$$\phi(z) = \lambda\frac{z-z_0}{1-\overline{z_0}z}, \quad |\lambda| = 1, \quad z_0 \in \mathbb{D}. \tag{2.7}$$

These are Möbius tranformations that translate z_0 to 0 and rotate by the angle $\arg\lambda$.

A hyperbolic circle in $\mathbb{D}$ is also a Euclidean circle. We can imagine drawing a circle C centered at the origin. The distance to the periphery on the real axis can be found from the hyperbolic radius by (2.6), where l is Euclidean radius and r_h is hyperbolic radius. Since the model has angular symmetry around the origin, this distance must be the same in all directions, giving us a circle in both hyperbolic and Euclidean geometry. Let now ϕ map the origin to the point z_0. $\phi(C)$ is still a circle, so hyperbolic circles are also Euclidean circles. We notice that the center of the hyperbolic circle is not the same as the center in the Euclidean circle when it is not the origin. The center in the hyperbolic cicle lies farther out towards the boundary of the unit circle, since the distances become longer when you move away from the origin. This also lets ut have circles of infinite hyperbolic radius drawn as finite Euclidean circles that are tangent to the boundary of the unit disc.

In the triangle contructed by drawing lines between the circle centers of three mutually tangent circles of radius x,y and z, the angle of the circle of radius x is given in hyperbolic geometry as

$$\alpha(x;y,z) = \begin{cases} \arccos\left(\frac{\cosh(x+y)\cosh(x+z)-\cosh(y+z)}{\sinh(x+y)\sinh(x+z)}\right), & x,y,z \in (0,\infty), \\ \arccos\left(\frac{\cosh(x+y)-e^{y-x}}{\sinh(x+y)}\right), & x,y \in (0,\infty), z=\infty, \\ \arccos\left(1-2e^{-2x}\right), & x \in (0,\infty), y=z=\infty, \\ 0 & x=\infty. \end{cases} \tag{2.8}$$

See figure 2.5 for an example of how a drawing in the Poincaré model of the unit disc can look like.

2.9 Polygons

The simple connectedness property that we have used so far to characterize the domains for the calculations are fine for mathematical purposes, but there is no guarantee that an explicit formula for the border exists. To get down to the numerical calculations we need a class of domains that are finitely representable by numbers. The practical way to do this is to use domains that are **polygons** with specified vertex coordinates and angles.

If the vertices are finite, then the angle at vertex n, α_n is given by the vertices w_{n-1}, w_n and w_{n+1} by the cosine formula.

$$\alpha_n = \arccos\left(\frac{w_n}{|w_{n-1}-w_n||w_{n+1}-w_n|}\right). \tag{2.9}$$

For the example programs made for this thesis, specifying the vertices of polygons in counter-clockwise order is enough, because we will require that the polygons are finite. For other applications one can let vertices be positioned at infinity and specify the angles at vertices that are adjacent to these infinite vertices.

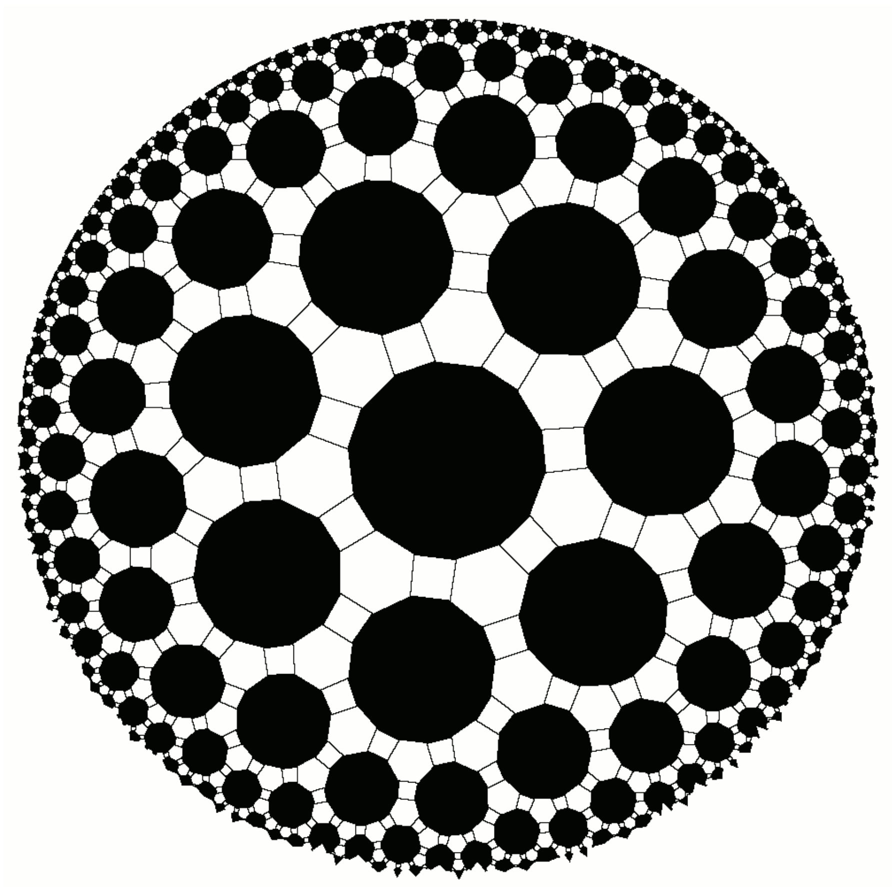


Figure 2.5: Hyperbolic covering of the plane in the Poincaré unit disc. The black regular fourteen-edged polygons are of the same hyperbolic size.

3 Physical applications of conformal mappings

Laplace's equation is an important partial differential equation in a range of different physical problems called **potential theory**. It is generally written

$$\nabla^2 \Phi = 0,$$

where ∇^2 is the Laplacian operator. Function satisfying this condition is said to be **harmonic**. In two dimensional cartesian coordinates this can be written

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = 0.$$

If we divide up a complex analytic function f in its real and imaginary components,

$$f(z) = u(x, y) + iv(x, y),$$

where $z = x + iy$, we know that the Cauchy-Riemann equations that must apply to any analytical function,

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \quad \text{and} \quad \frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y},$$

and so it follows that

$$\frac{\partial^2 u}{\partial y^2} = \frac{\partial}{\partial y}\left(-\frac{\partial v}{\partial x}\right) = -\frac{\partial}{\partial x}\left(\frac{\partial v}{\partial y}\right) = -\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right),$$

so u satisfies the Laplace equation, $\nabla^2 u = 0$. The same can be shown for v.

Definition 1 Harmonic conjugate
The harmonic conjugate ψ of a function $\phi(x,y)$ defined on a domain $\Omega \in \mathbb{R}^2$ exists if the two functions are the real and imaginary part of an analytic function $f(z)$ where $z = x + iy$. That is, if

$$f(z) = \phi(x,y) + i\psi(x,y)$$

is analytic in Ω.

Theorem 12 Harmonic functions under conformal mapping [10]
Let Φ^* be harmonic in a domain Ω^* in the w-plane. Suppose that f given by $f(z) = w$ is analytic in a domain Ω in the z-plane and maps Ω conformally onto Ω^*. Then the function

$$\Phi(z) = \Phi^*(f(z))$$

is harmonic in Ω.

Proof. The fact that the composition of analytic functions is analytic follows from the chain rule. If we can take the harmonic conjugate Ψ^* of Φ^*, then we can form the analytic funtion $F^*(z) = \Phi^*(x,y) + i\Psi^*(x,y)$ and get $F(z) = F^*(f(z))$ which is analytic in Ω. The real part of $F(z)$, $\mathfrak{Re}\, F(z) = \Phi(x,y)$ is then harmonic in Ω. $\square$

It can be shown that if Ω^* is simply connected, then a harmonic conjugate of Φ^* exists [10].

3.1 Example in fluid flow

If we have an irrotational, incompressible, steady, nonviscous fluid flow that can be expressed as a complex potential, then the flow can be expressed as stream lines in a canonical domain and then conformally mapped to a domain of interest. If we imagine a cross section of a flow above a flat ocean floor surface with horizontal stream lines, we can find the flow above an ocean floor that is not flat by conformally mapping the upper half-plane to some other unbounded domain. If we conformally map the upper half-plane to the region above a horizontal line with a vertical jump, we get the stream lines shown in figure 3.1.

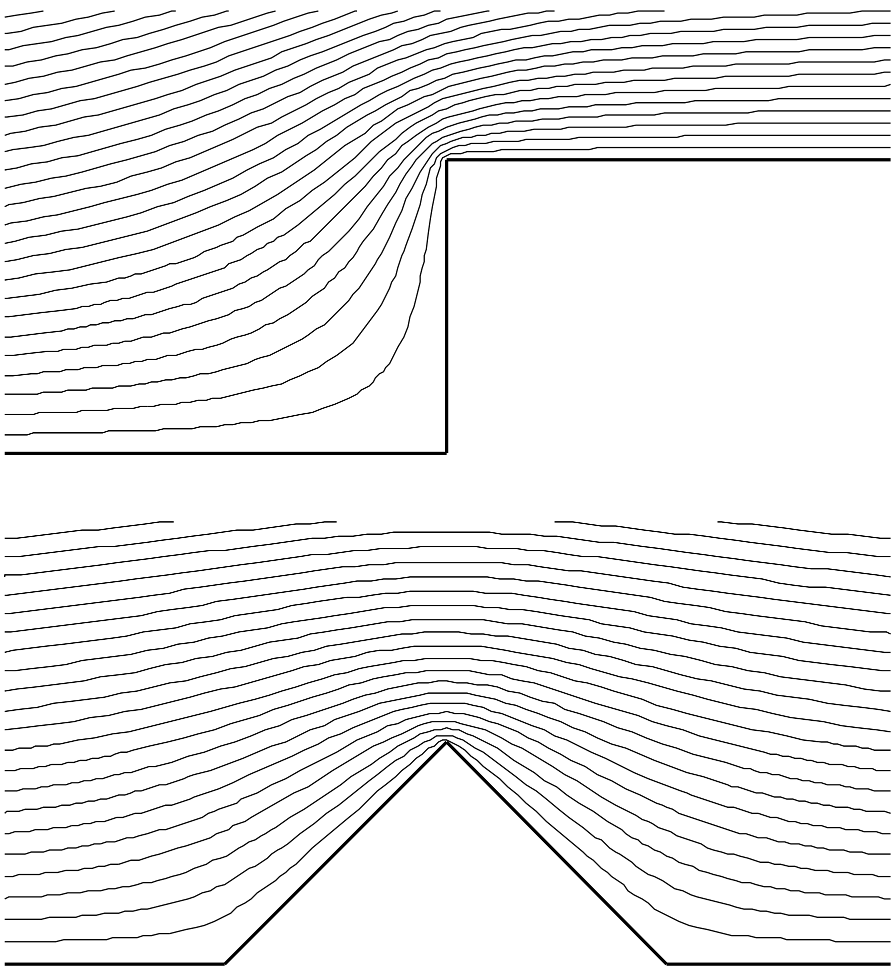


Figure 3.1: Stream lines for fluid flow around an irregularity in the border. See appendix A.2 for the program used to make the plots.

4 The Schwarz-Christoffel mapping method

After Riemann put forth his mapping theorem in 1851, less that 20 years went by before the Schwarz-Christoffel formula was discovered. Elwin Bruno Christoffel and Hermann Amandus Schwarz independently discovered a formula for conformal maps to arbitrary polygons that can be used in numerical computations to give explicit Riemann maps.

4.1 Basic idea

The basic idea that eventually resulted in the general formula is that the derivative of a conformal map $f : \mathbb{H} \to \Omega$ from the upper half-plane $\mathbb{H}$ to an arbitrary domain Ω can often be written as a product of some elementary functions

$$f' = \prod f_k$$

because then we get

$$\arg f' = \sum \arg f_k.$$

If we let each $\arg f_k$ be a step function, then f maps the real axis to a polygon. So, we let $\Omega = P$ be a polygon with vertices $w_1, w_2, \ldots, w_n$ and interior angles $\alpha_1\pi, \alpha_2\pi, \ldots, \alpha_n\pi$ given in counter-clockwise order. Let us for now assume that the vertices are finite and that the polygon is simple, so that $\alpha_i \in (0, 2)$. We call $z_i = f^{-1}(w_i)$ the **prevertices** of f. We can assume that $z_n = \infty$, because if it is not, we can add another vertex some place on the polygon with interior angle π. Because the real axis is mapped to the boundary of the polygon, ∞ must be mapped to some place on that boundary.

By the Schwarz reflection principle, f can be analytically continued across the segment (z_k, z_{k+1}) of the real axis, so that f' exists there. We also see that $\arg f'$ must be constant on such segments, and that $\arg f'$ must make a jump at each z_k,

$$\lim_{z \to z_k^+} \arg f'(z) - \lim_{z \to z_k^-} \arg f'(z) = (1 - \alpha_k)\pi = \beta_k \pi,$$

and we call this jump $\beta_k \pi$ the **turning angle** at vertex k. If we now consider the function

$$f_k = (z - z_k)^{-\beta_k},$$

we see that we have a function that is analytic on $\mathbb{H}$, has the specific jump on $z = z_k$ and has $\arg f_k$ constant on the real axis.

This whole chain of reasoning leads to the idea that a conformal map to a polygon can now be written

$$f'(z) = C \prod_{k=1}^{n-1} f_k(z) \tag{4.1}$$

and this is the essential idea of the Schwarz-Christoffel formula.

4.2 Schwarz-Christoffel formula

By summing up the previous section and specifically equation 4.1, we get the Schwarz-Christoffel formula for the half-plane,

Theorem 13 Schwarz-Christoffel formula for a half-plane
Let P be the interior of a polygon Γ having vertices $w_1, \ldots, w_n$ and interior angles $\alpha_1\pi, \ldots, \alpha_n\pi$ in counterclockwise order. Let f be any conformal map from the upper half-plane H^+ to P with $f(\infty) = w_n$. Then

$$f(z) = A + C \int^z \prod_{k=1}^{n-1} (\zeta - z_k)^{\alpha_k - 1} \mathrm{d}\zeta \tag{4.2}$$

for some complex constants A and C, where $w_k = f(z_k)$ for $k = 1, \ldots, n-1$.

The lower limit of integration only changes the constant A. Of course, the prevertices z_k are not known in advance, so one cannot simply perform this integration and get a Riemann map. But before exploring how to find these, let us consider the formula itself.

We can also find a formula for the map from the unit disc $\mathbb{D}$:

Theorem 14 Schwarz-Christoffel formula for the unit disc
Let P be the interior of a polygon Γ having vertices $w_1, \ldots, w_n$ and interior angles $\alpha_1\pi, \ldots, \alpha_n\pi$ in counterclockwise order. Let f be any conformal map from the unit disc $\mathbb{D}$ to P. Then

$$f(z) = A + C \int^z \prod_{k=1}^{n} \left(1 - \frac{\zeta}{z_k}\right)^{\alpha_k - 1} \mathrm{d}\zeta \tag{4.3}$$

for some complex constants A and C, where $w_k = f(z_k)$ for $k = 1, \ldots, n$.

4.3 Explicit Schwarz-Christoffel maps

Although we do not know the z_k for a chosen polygon, we know from the general theory of conformal maps that we can use Möbius tranformations to map any three points on the real axis to any three points on the boundary of the polygon as long as the order is preserved. These three degrees of freedom lets us use the Schwarz-Christoffel formula explicity for $n \leq 3$.

Consider for example the special kind of polygons with one vertex $w_1 = \infty$ and $\alpha_1 = -1$. This is of course, only a straight line. The formula for the half-plane gives us

$$f = A + Cz,$$

which is a scaling, rotation and translation of the half-plane, which is what you would expect for just mapping the real axis to any line. By using the map from the unit disc, we get

$$f(z) = A + C \int^z (\zeta - z_1)^{-2} \mathrm{d}\zeta = A + \frac{C}{z - z_1}.$$

If we allow two vertices, $\alpha_1 + \alpha_2 = 0$, so either $\alpha_1 = \alpha_2 = 0$, or $\alpha_1 = -\alpha_2 \neq 0$. The first case has both vertices at ∞ and is a strip. The half-plane map gives

$$f(z) = A + C \int^z (\zeta - z_1)^{-1} \mathrm{d}\zeta = A + C \log(z - z_1).$$

and the unit disc map gives

$$\begin{aligned} f(z) &= A + C \int^z (\zeta - z_1)^{-1} (\zeta - z_2)^{-1} \mathrm{d}\zeta \\ &= A + C \int^z \left(\frac{1}{\zeta - z_1} - \frac{1}{\zeta - z_2} \right) \mathrm{d}\zeta \\ &= A + C \log \left(\frac{z - z_1}{z - z_2} \right). \end{aligned}$$

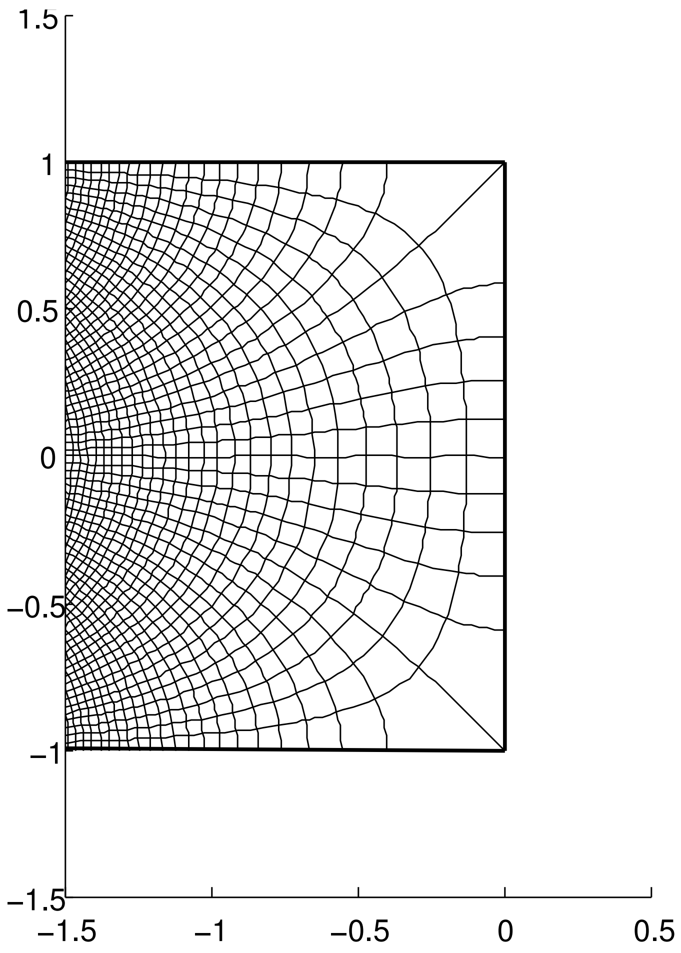


Figure 4.1: A conformal map from the half-plane to a polygon with one vertex at infinity. The polygon is specified as two vertices at i and $-i$ with internal angles $\pi/2$ and one vertex at infinity.

4.4 Vertices at infinity

One advantage of the Schwarz-Christoffel method is that it can be used with one or more vertices of the polygon positioned at infinity. If one specifies the interior angles of the vertices of the polygon that are connected to the vertices at infinity, the polygon is fully determined and the Schwarz-Christoffel transformation can be used as on any other polygon. See figure 4.1 for an example.

4.5 Calculating the prevertices

The main challenge in computing a conformal map to an arbitrary polygon with the Schwarz-Christoffel method is to solve the parameter problem of finding the prevertices z_k. The basic way to do this is to find some equations involving the side lengths and numerically solve a system of equations, as described by Trefethen [25].

If we specify the three degrees of freedom by letting $z_{n-2} = -1$, $z_{n-1} = -i$ and $z_n = 1$, we need to determine the rest of the points z_k on the unit circle, that is $n - 3$ angles, which are real quantities. If we assume the polygon to be bounded, we get

$$\frac{\left|\int_{z_j}^{z_{j+1}} f'(\zeta)\mathrm{d}\zeta\right|}{\left|\int_{z_1}^{z_2} f'(\zeta)\mathrm{d}\zeta\right|} = \frac{|w_{j+1} - w_j|}{|w_2 - w_1|}, \quad j = 2, 3, \ldots, n-2,$$

This gives us $n - 3$ conditions to satisfy, and this system of equations can be solved by some general iterative method for non-linear equations. One should constrain the angles to lie in order on the unit circle. To avoid solving a constrained system of equations, which can be harder, one can transform the equations, for example by

$$\phi_k = \log\left(\frac{\theta_k - \theta_{k-1}}{\theta_{k+1} - \theta_k}\right),$$

to get an unconstrained system. This transformation has turned out to work well in practice [25].

4.6 Calculating the integral

When you have computed the prevertices of the mapping, you still need to evaluate the integral (4.2) or (4.3) to map specific points to the polygon. General numerical integration methods can have problems with integrals of this form for some points, but the form of the integral lets us use a special type of numerical integration called Gauss-Jacobi quadrature. This is a method for approximating integrals of the form

$$\int_{-1}^{1} f(x)(1-x)^{\alpha}(1+x)^{\beta}\mathrm{d}x.$$

4.7 Finding the inverse map

We have seen that when the prevertices are found, it is only a matter of evaulating the integral (4.2) or (4.3) to find specific image points $w = f(z)$. But finding arbitrary inverse points $z = f^{-1}(w)$ has no direct approach. Trefethen [25] proposed two methods for solving this problem

1. Solve $f(z) - w = 0$ by Newton's method.
2. Solve the initial-value problem

$$\frac{\mathrm{d}z}{\mathrm{d}w} = \frac{1}{f'(z)} \quad \text{and} \quad z(w_0) = z_0$$

   with a numerical solver for initial-value problems.

The first method is fast and simple, because $f'(z)$ is explicitly known, but needs a good starting point to guarantee convergence. The differential equation method is much slower, but more reliable.

4.8 Computer programs

The Schwarz-Christoffel method has been widely used and so there exists several mature computer programs for calculating these maps for arbitrary polygons. Toby Driscoll's *The Schwarz-Christoffel Toolbox for MATLAB* [4] was used to make some of the graphics in this text and was found to be fast and easily usable.

The geodesic algorithm and the Zipper program 5

In the early 1980s an algorithm called "Zipper" for computing conformal maps was discovered independently by Reiner Künhnau and Donald E. Marhsall [17].

The idea of the algorithm is to use simple conformal functions to map curve segments approximately on the boundary curve of the specified domain Ω to the real axis in such a way that the composition of these functions maps the whole boundary of Ω to the real axis. This gives a conformal map from a domain resembling Ω to the upper half-plane, which of course can be mapped to the unit disc by a simple Möbius transformation, giving the wanted map. The workings of this algorithm will be explained in detail in the next section.

This method gave fast and accurate results. Also, the one can find the inverse of the computed map with little extra computation, and this is a big advantage in many applications. The convergence properties of the algorithm was not known, but some recent work has shown convergence with some restrictions on the input [17].

Different types of functions for approximating the boundary segments can be used as the building blocks for this algorithm. When using straight line segments, the algorithm is called Zipper. Substituting straight line segments for circle arcs gives another way of calculating the map that is easier to implement.

5.1 The geodesic algorithm

We assume that we are given input points $z_0, \dots, z_n \in \mathbb{C}$ that lie in order on the boundary $\gamma = \partial\Omega$.

The basic building blocks of the geodesic algorithm are conformal functions $f_a : \mathbb{H} \setminus \gamma \to \mathbb{H}$. The geodesic algorithm, as opposed to other variants of this

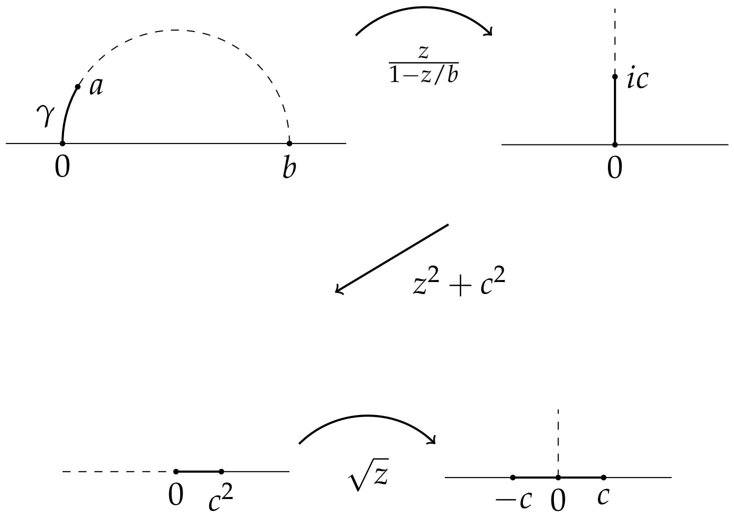
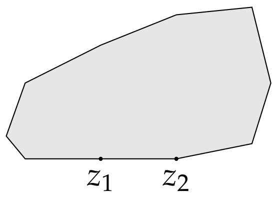


Figure 5.1: The map f_a which is the basic component out of which the geodesic algorithm's conformal map ϕ is made.

method, uses circle arcs as the γ curve. The functions f_a can then be described by Möbius tranformations composed with squaring and square roots and can therefore be explicitly inverted. The maps f_a that conformally remove the circle arcs from 0 to a from $\mathbb{H}$ can be visualized as in figure 5.1

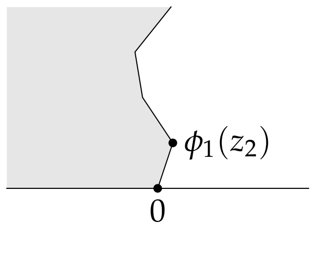
The geodesic algorithm works as follows:

- Start out with some polygon representing the domain Ω to find the Riemann map from.



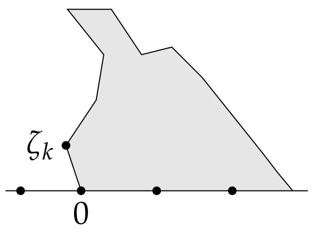
- Map the straight line segment between the two first vertices to the real line so that the rest of the vertices are in the upper half-plane. This can be done with the function

$$\phi_1 = i\sqrt{\frac{z-z_1}{z-z_0}}.$$



- For the rest of the vertices, do the following. Let the vertex z_k under the $k-1$ previous maps be called $\zeta_k = \phi_{k-1} \circ \phi_{k-2} \circ \cdots \circ \phi_1(z_k)$. Apply the k-th map

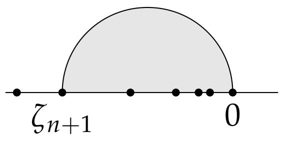
$$\phi_k = f_{\zeta_k}.$$



- After applying n maps, the interior of the polygon has been mapped to a half-disc in $\mathbb{H}$. We can map it to $\mathbb{H}$ by

$$\phi_{n+1} = -\left(\frac{z}{1 - z/\zeta_{n+1}}\right)^2.$$

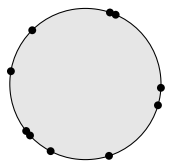
The minus sign can be replaced with $+$ if the data points are in clockwise order, but for the purposes of this text, we will assume that the points are given in counterclockwise order.



- To get a map to the disc, one can apply the Möbius transformation

$$\phi_d = \frac{z-a}{z-\bar{a}}$$

where a is an interior point. This point can be specified in the original polygon.



The function $\phi = \phi_d \circ \phi_{n+1} \circ \cdots \circ \phi_1$ is then a conformal map from a domain approximately equal to Ω to the unit disc $\mathbb{D}$. Since all the functions used have explicit inverses, the inverse map ϕ^{-1} can be found in the same way.

5.2 Implementation details

The functions used here can easily and concisely be implemented in a programming language with basic facilities for complex calculations. See appendix B for an example implementation. One important tip is to use the right half-plane instead of the upper half-plane, because the common way to implement the complex square root in programming languages has the branch cut along the negative real axis. By just rotating all the figures presented in this chapter, one gets rid of most problems. For points that are close to the imaginary axis (if we are now in the right half-plane), numerical inaccuracies in squaring can put points on the wrong side of the branch cut, so that when the square root is applied, the point ends up as its complex conjugate. This can be remedied in the following way: when applying f_a, check

$$w = \sqrt{z^2 - 1},$$

and if $(\mathfrak{Im}\, w)(\mathfrak{Im}\, z) < 0$, then replace w with $-w$. See source code in appendix B. The example program made for this text works well for the forward map ϕ, but has precision problems for the inverse ϕ^{-1}. The most important loss of precision comes from the squaring and square root functions. To illustrate, consider the number $1 + \epsilon$, where ϵ is some small number larger than the machine precision epsilon. This is commonly about 10^{-16}. The square root $\sqrt{1+\epsilon}$ will then have half as many significant digits, so when squaring again to $\sqrt{1+\epsilon}^2$, we know that this should be equal to $1 + \epsilon$, but half the precision has been lost.

Another problem to watch out for is mapping points that should be inside of the domain, but are very close to the border. Because the geodesic algorithm computes a conformal map from a domain approximating Ω, points that are close to the border could be outside of the domain of ϕ. Since ϕ maps the whole plane to the whole plane, points outside the domain of ϕ will still be mapped, but could end up far from where they should be. See figure 5.2 for an illustration of the phenomenon.

5.3 The slit algorithm

When using the geodesic algorithm, one can say that one is approximating the curve sections of $\partial\Omega$ by circular arcs. For polygons, this should intuitively be replaced by some g_a that replaces f_a and uses straight line segments to approximate the border. This can be done with the functions

$$g_a^{-1}(z) = C(z-p)^p(z+1-p)^{1-p},$$

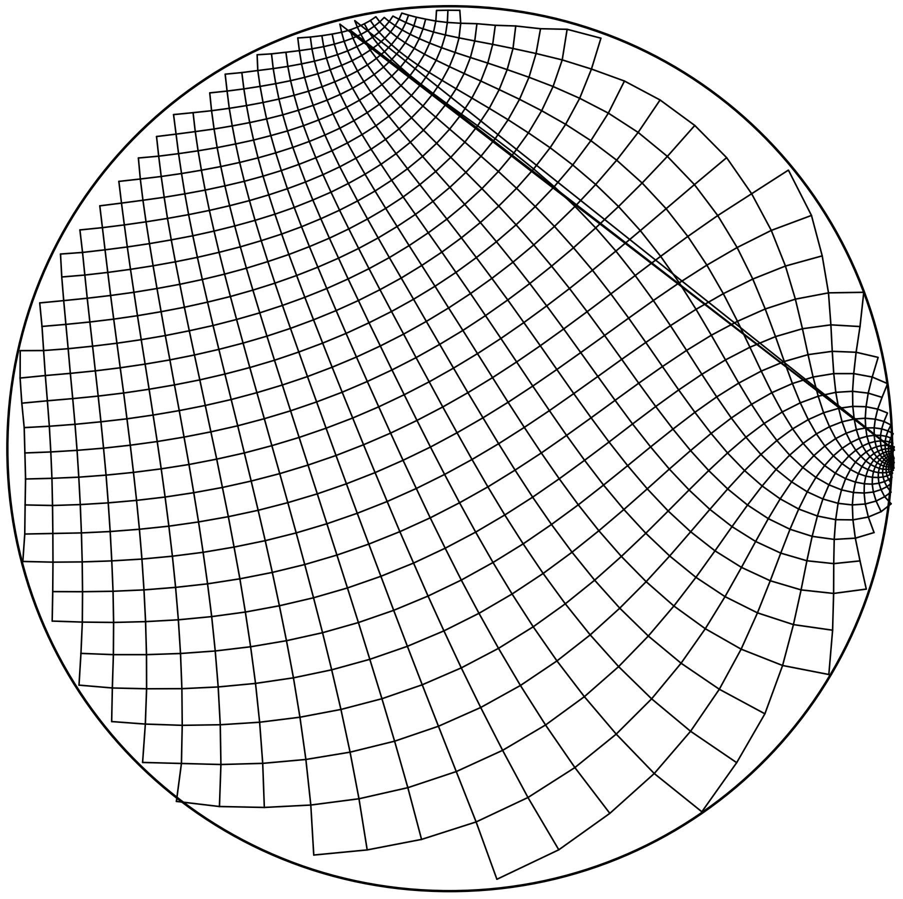


Figure 5.2: The geodesic algorithm maps an approximation of the domain Ω to the unit disc. Points near the boundary could be outside of the approximate Ω and end up in unexpected places.

where $p = \arg a/\pi$ and $C = |a|/p^p(1-p)^{1-p}$. These map $\mathbb{H}$ to $\mathbb{H} \setminus L$, where L is the straight line segment from 0 to a, and can be shown to be conformal. The g_a^{-1} cannot be inverted explicitly to give the map from Ω to $\mathbb{H}$ (or $\mathbb{D}$), but using a numerical inverse calculation one can get more accurate results than with the geodesic algorithm.

5.4 The Zipper program

Another improvement on the approximation of $\partial\Omega$ can be made by using circular arcs that are not orthogonal to the real axis. At each stage in the algorithm,

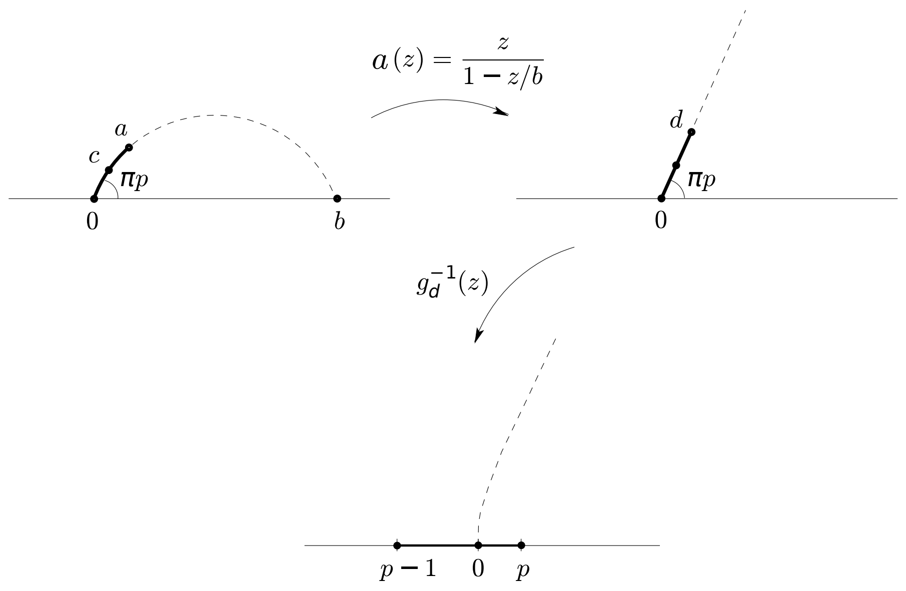


Figure 5.3: The basic map f_a for the zipper algorithm. Figure from [17].

instead of just mapping ζ_k to 0 by some map, we can use the images of the two next points, ζ_{2k-1} and ζ_{2k}, assuming an even number of data points. If we map the circular arc from 0 through these two points to a straight line by a Möbius transformation, we can use the g^{-1} from the slit algorithm to map the points to the real axis. See figure 5.3.

This approach is more complicated than the simple geodesic algorithm, but it can be thought of as a quadratic approximation of each curve segment, and it turns out that it gives better approximations. This version of the algorithm is the one that is used in Marshall's Zipper program [13].

5.5 Basic proof of convergence for the geodesic algorithm

If we find the conformal map ϕ with the geodesic algorithm for some desired domain Ω, we can find a bound on how far off the boundary of the computed domain $\partial\Omega_C$ can be from $\partial\Omega$ [14].

Let us assume that the domain Ω is a polygon with vertices $z_0, z_1, \ldots, z_n$ given in order on the boundary. Define a **closed disc-chain** $D_0, D_1, \ldots, D_n$ as a sequence of pairwise disjoint open discs such that ∂D_j is tangent to ∂D_{j+1} for

$j = 0, \ldots, n-1$, and ∂D_n is tangent to ∂D_0.

There are several ways to make a closed disc chain covering a simple closed polygon P. A simple way to do it is as follows: Let $\epsilon > 0$, and find pairwise disjoint discs $\{B_j\}$ with centers at each vertex z_j and radius $r < \epsilon$. If we remove these discs from the boundary of the polygon,

$$\partial P \setminus \bigcup_j B_j = \bigcup_k L_k,$$

a set L_k of pairwise disjoint line segments remains. We now cover each L_k with a disc-chain where the centers of the discs are on L_k and the circles at the ends are tangent to the corresponding B_j and the radii are less than half the distance to any other L_i and less than ϵ. All these discs, taken in correct order will then be a closed disc-chain covering ∂P. There are other ways to do this that are better suited for direct computation [14].

Define a **geodesic** in Ω as a curve that can be conformally mapped to the unit disc $\mathbb{D}$ so that it is a geodesic in the hyperbolic geometry.

Theorem 15 Jørgensen's lemma
Let D be an open disc contained in a simply connected domain Ω and let J be a geodesic in Ω. Then $J \cap D$ is connected, and if not empty, then J is not tangent to ∂D in Ω.

This lemma can be interpreted as saying that discs are convex in the metric on a simply connected domain given by the geodesics.

We now let the vertices of the polygon be the tangent points $z_0, z_1, \ldots, z_n$ of the discs in the closed disc-chain constructed above. This gives the same polygon P with some added vertices.

Theorem 16 Bound for Ω_C [14]
Let $D_0, D_1, \ldots, D_n$ be a closed disc-chain. Then the geodesic algorithm applied to the centers of these discs, $z_0, z_1, \ldots, z_n$ gives a conformal map ϕ^{-1} from the unit disc $\mathbb{D}$ to a region Ω_C bounded by

$$\partial\Omega_C \in \bigcup_{j=0}^{n} (D_j \cup z_j).$$

Proof. Let γ_j be the segment of the computed boundary $\partial\Omega_C$ between vertices z_j and z_{j+1}. Geodesics in the hyperbolic metric on the unit disc are preserved under conformal maps, so γ_j is a geodesic in

$$\overline{\mathbb{C}} \setminus \bigcup_{k=0}^{j-1} \gamma_k.$$

The inverses of the basic maps f_a are analytic across $\mathbb{R} \setminus \{\pm c\}$. $f_a(\pm c) = 0$ and f_a^{-1} can be approximated by a square root near $\pm c$. Let f_b be another one of these basic maps. Then f_b^{-1} is analytic and asymptotic to z^2 near 0, so $f_b^{-1} \circ f_a^{-1}$ preserves angles at $\pm c$. The geodesic γ_j is then a smooth arc which joins with γ_{j-1} at z_j with angle π. So, the computed boundary $\partial\Omega_C$ is smooth. The segment from the first vertex, γ_0, is a chord of D_0 and cannot be tangent to ∂D_0. Since the angle between γ_0 and γ_1 meeting in z_1 is π, γ_1 must continue into D_1. Jørgensen's lemma then gives us that $\gamma_1 \in D_1$ and γ_1 cannot be tangent to ∂D_1. We can continue in this fashion and conclude that

$$\gamma_j \in D_j$$

for $j = 0, 1, \ldots, n$. The curve $\partial\Omega_C$ lies within the disc-chain, and the theorem is proved. $\square$

This proof of convergence can be improved to use lenses instead of discs [15]. There are also proofs of convergence for the Zipper algorithm [17] that are out of the scope of this text.

6 Circle packing method

As shown earlier, circle packing can be used to find a Riemann map. This chapter recapitulates my project work on this method [12].

6.1 Constructing the circle complex

We will use the domain Ω in figure 6.1 and cover it with small circles to find a graph G that can be used for the circle packing.

Gauss showed that a hexagonal grid structure between the circles gives the best possible covering of the plane [26], so we can begin by imagining the whole plane covered by circles of radius r in a hexagonal grid. We then remove all the circles that are not contained in Ω.

If Ω is a polygon, we can decide if a circle center is contained in Ω by a winding number test as described in [6]. The algorithm uses the boundary points of the polygon and finds the winding number around the specified point, and this tells you if the point is inside or outside the polygon. See program code in C.3. If Ω is not a polygon, we find a polygonal approximation and use that.

We now have a set of circles that are contained in Ω. We require that each of the circles are tangent to at least three other circles. This requirement can be avoided, but requires a more robust method for calculating the circle packing (see [3]). Since Ω is simply connected, this can be satisfied by making r so small that the circles that lie on the boundary of the graph do not represent more that one contiguous segment of the border of Ω. We can then be sure that each circle on the border is tangent to at least one other circle on the boundary and two circles in the interior. Circles that are not on the border are always tangent to six other circles.

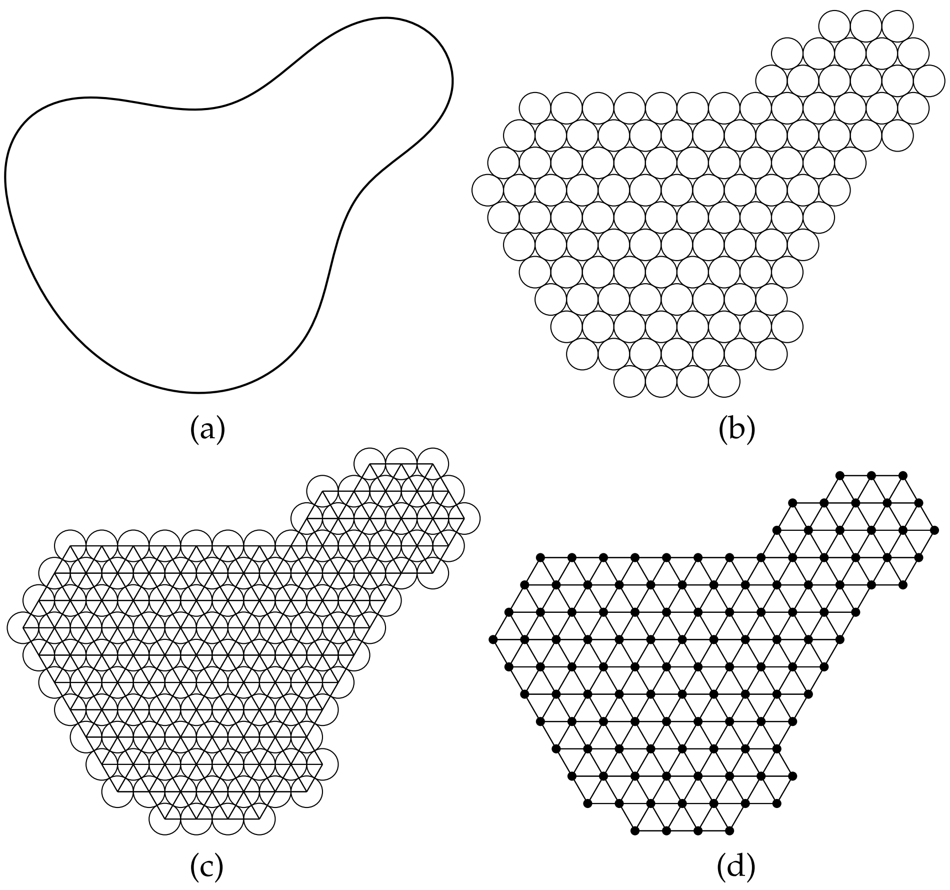


Figure 6.1: (a) The domain Ω. (b) Small circles that cover Ω as well as possible. (c) Graph edges between tangent circles. (d) Approximate representation of Ω as the intersection graph between the circles.

By letting each circle center be a node and each two tangent circles be an edge, we get an intersection graph representing the hexagonal circle covering of Ω.

Because we laid down the hexagonal grid without taking into account the choice of Ω, it is possible that translating the original grid could have given a marginally better covering of the domain, but both the polygonal approximation of Ω and the choice of r can easily be improved to make this effect negligible.

6.2 Radius list for circle packing in hyperbolic geometry

We have a graph G representing Ω and know that it is topologically equivalent to a closed disc. The theorem of maximal packing of a closed disc tells us that there is a unique maximal circle packing P_K for G in $\mathbb{D}$, and we will find an approximation to this.

Let the radii of the internal circles be initialized to some arbitrary value, and let the boundary circles have infinite radius since they will be tangent to the unit circle. We can approximate the infinite radius by some large value, for exampel 1000 times the radius of the internal circles, to simplify the calculations. The boundary circles can easily be identified as the only circles that do not have exactly six neighbors (see 2.8).

This gives us a set of circle radii and tangency requirements that do not necessarily fit together. The requirement for these radii to give a univalent circle packing is that the sum of the angles around an internal circle is 2π. See figure 2.4.

The radii $R(c_i)$ can be found as the solution to a Dirichlet problem with the boundary circles as boundary conditions [3], but it can be shown by a monotony argument that one can iterate to a solution just by adjusting the radii locally so that the angle sum of a flower gets closer to 2π [3]. In *Introduction to Circle Packing*, Stephenson gives a simplified algorithm in pseudocode:

```
1. Initialize $R(c_i)$ by giving the boundary circles their prescribed radii and the internal circles arbitrary radii.
2. For every internal circle $c_i$:
    a) Find the angle sum $\alpha(i)$ of the flower around $c_i$ by the cosine law (2.8).
    b) If the error $|2\pi - \alpha(i)|$ is larger than $\epsilon$, adjust $R(c_i)$:
        i. If $\alpha(i) < 2\pi$, decrease $R(c_i)$.
        ii. If $\alpha(i) > 2\pi$, increase $R(c_i)$.
3. If $|2\pi - \alpha(i)| < \epsilon$ for all internal circles $c_i$ then the radii in the circle packing have been found. If not, go to step 2.
```

This algorithm is a very simple way to find the radii in the circle packing. The infinite radii on the boundary circles ensures that we get the maximal packing, and by letting α be given by the cosine law in the Poincaré geometry (2.8) we get the packing in $\mathbb{D}$. The difficulty in the implementation of this algorithm is finding good ways to decrease and increase $R(c_i)$.

By trial and error, it turned out that multiplying the radius of a circle c_i by a value depending on the error $e = |2\pi - \alpha(i)|$ made the program converge to the circle packing. The method used in the program made for this text was as follows:

Let $d = \min\{e, 1\}$. Insert the following instructions in step 2b of the algorithm.

- If $\alpha(i) < 2\pi$, set the radius to $(1 - \frac{1}{10}d)R(c_i)$.
- If $\alpha(i) > 2\pi$, set the radius to $(1 + \frac{1}{10}d)R(c_i)$.

This procedure worked well enough to quickly find the circle packings shown in this text, see figure 6.3 for examples of packings and appendix C.4 for program code. Stephenson has made a program called CirclePack [22] that has a much more efficient algorithm and many other uses than finding the Riemann map, but for the purpose of this text a simpler prorgam was made to illustrate how to calculate circle packings. This was also recommended by Stephenson in *Introduction to Circle Packing* to better understand the underlying mathematics.

6.3 Placing the circles in hyperbolic geometry

Theorem 17 Placing circle centers in a hyperbolic circle packing
Given the hyperbolic radii $R(c_i)$ in a circle packing of a triangulation G of a simply connected domain and the position of one circle and one of its neighbords, the centers of the rest of the circles in the circle packing are uniquely determined.

Proof. We have found the graph G and the radii R for the circles that the nodes of G represent. If we know the position of two circles, the position of a third circle that is tangent to the first two is uniquely determined since we know the radii of all the circles and can construct the triangle with vertices in the circle centers and side lengths given by their radii. See figure 2.4. Since the graph is simply connected and we have required that each internal circle cannot be tangent to more that one contiguous segment of the boundary, we see that as long as we have circles with unknown position, at least one of them must be tangent to two other circles that have already been positioned. Applying this argument several times shows that all circles can be positioned as soon as the two first are in place.

This process is not described by Stephenson in *Introduction to Circle Packing*, [24], but we can proceed as follows:

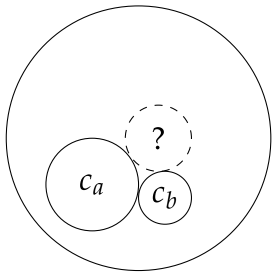
To place the two first circles, we choose an internal circle c_0 of G and place it with its center at the origin and radius $l(r_{c_0})$ determined by (2.6). Find a circle tangent to c_0 and place it on the positive real axis in this way:

- Let he center be at the distance $\frac{l(r_{c_0})+l(r_{c_0}+r_{c_1})}{2}$ from the origin on the positive real axis.
- Let the radius be $\frac{l(r_{c_0}+r_{c_1})-l(r_{c_0})}{2}$.

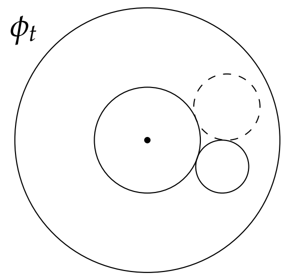
When this is done, the position of the rest of the circles can be determined. The choice of the two circles is arbitrary and is equivalent to the choice of $f(0) = z_0$ and $\arg f'(0) = \theta_0$ that makes the Riemann map uniquely determined.

The procedure after the two first circles have been positioned is then as follows:

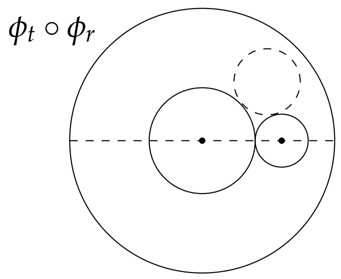
- Find a circle c that has not been placed and two circles c_a, c_b that have already been postioned so that all three are mutually tangent and their counter-clockwise order is c_a, c_b, c.



- Let z_0 be the hyperbolic center of c_a so that the isomorphism $\phi_t(z) = \frac{z-z_0}{1-\bar{z}_0 z}$ translates the center of c_a to the origin.

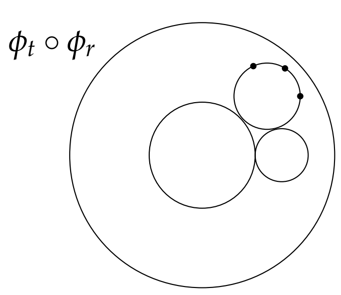


- Let λ be the angle between the center of $\phi_t(c_b)$ and the positive real axis, so that the isomorphism $\phi_r(z) = \lambda z$ rotates the unit disc so that $\phi(z) = (\phi_t \circ \phi_r)(z)$ places the center of c_b on the positive real axis.

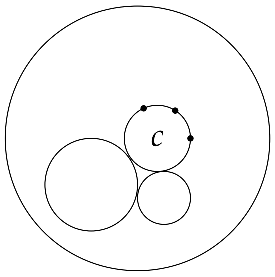


- Find the angle α to the line from the origin that passes through the center of c by using (2.8), see figure 2.4(a).

- We know that the circle c has Euclidean center in $\frac{l(r_{c_a})+l(r_{c_a}+l(r_c)}{2}e^{i\alpha}$ and Euclidean radius $\frac{l(r_{c_a}+r_c)-l(r_{c_a})}{2}$ under ϕ. Find three points p_i on the periphery of this circle, since these three points uniquely determine the circle.



- The center of the Euclidean circle is not the same as the center of the hyperbolic circle, but the periphery is the same, so $\phi^{-1}(p_i)$ for the three points gives us the circle translated and rotated back to where it should be.



- Repeat this process until all circles have been positioned.

See appendix C.5 for program code. □

For the domain Ω, a circle packing in $\mathbb{D}$ looks like in figure 6.2.

6.4 Refining the circle grid

By letting the radii of the circles in the original hexagonal grid covering Ω tend to zero, the circle packing will approximate the Riemann map, by theorem 10. See figure 6.3 showing how the circle packings look for decreasing radius r.

A simple way to give a map looking more like the complex plots described in the introduction to this text is to put a color map over the circles in the hexagonal grid and color the corresponding circles in the unit disc with the same color. The map shown in figure 6.4 uses colors that depend on the distance from a point in the interior of Ω.

6.5 Continuous functions

The graphics in figure 6.3 show a function between two circle packings. If we map each triangle between three mutually tangent circles to the corresponding triangle in the unit disc by an affine transformation, we get a piecewise continuous map from Ω to $\mathbb{D}$. This map is K-quasiconformal, where K depends on the maximal angle distortion one can find in each triangle. This K tends to 1 as the circle grid radius tends to 0, so that we get a conformal map in the limit [24].

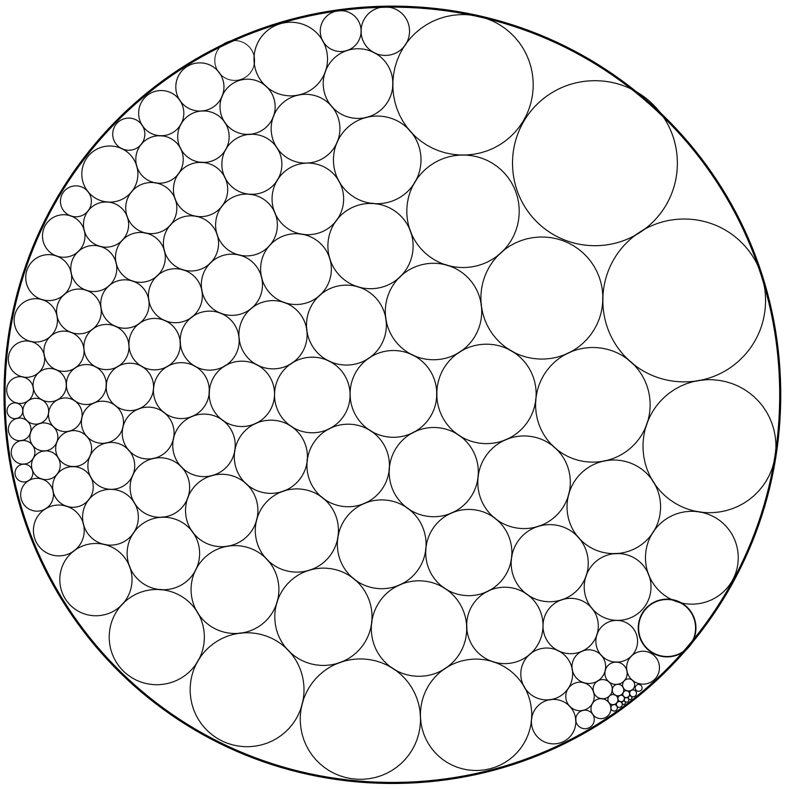


Figure 6.2: Circle packing in $\mathbb{D}$ of the domain Ω, shown in figure 6.1

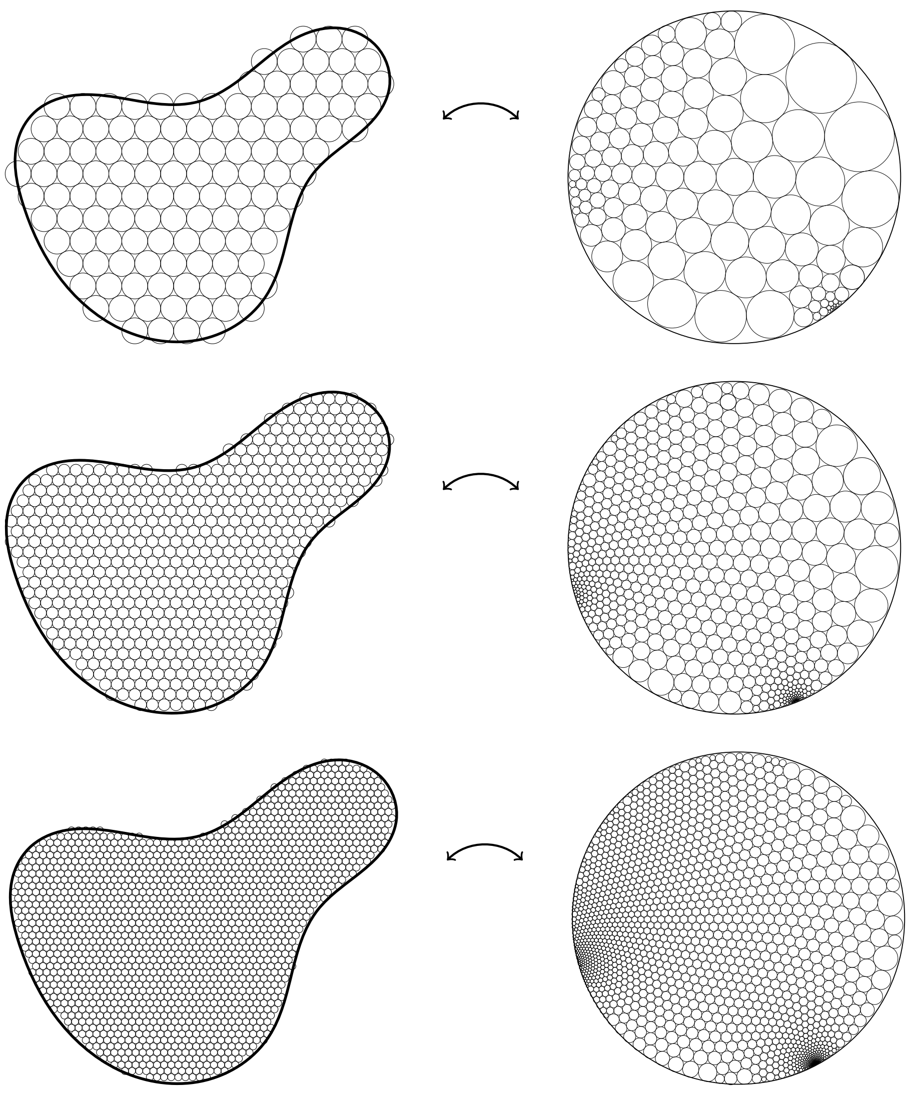


Figure 6.3: Circle packings for the domain Ω with decreasing radius in the hexagonal grid

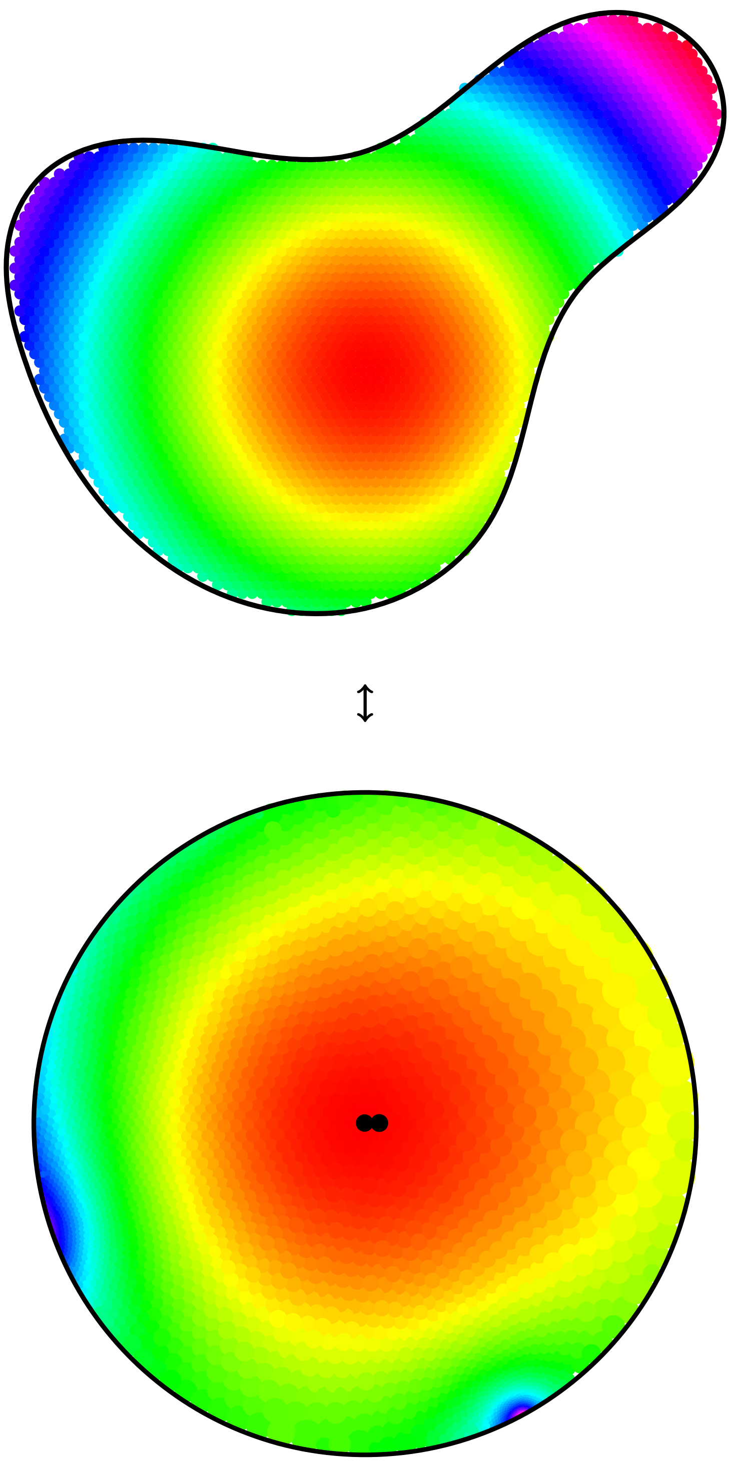


Figure 6.4: The approximate Riemann map made with the circle packing method. The domain Ω is colored by choosing colors depending on the distance from a point in the interior, and the colors follow the map to the unit disc.

7 Evaulation of the different methods

The three different methods for approximating conformal mappings presented in this text make maps that are quite different in the way that they are approximate to the Riemann map.

Making maps with the Schwarz-Christoffel method depends on quite general numerical methods. One needs to solve a system of non-linear equations and numerically integrate a complex integral with singularities. The speed and precision of the conformal mapping algorithm depends on these numerical routines.

For the geodesic algorithm or the Zipper program, the map is always conformal because it is composed of elementary functions that are conformal. This is convenient if the conformal property is more important than the exact boundary of the domain. It can also cause problems, for example if you try to map a point near the border of the domain. If this point is outside the approximate domain that the mapping gives, then you will get a function value outside the unit disc, which could be far off from the wanted value. This is illustrated in figure 5.2. Recent research has made progress in proving the convergece properties of these algorithms.

The circle packing method makes a map that is quasi-conformal. The beautiful theory of circle packing shows that this map converges to the Riemann map, but for planar simply connected domains such as the ones studied in this text, the circle packing method is slow compared to the two others. It requires manipulation of a quadratically increasing number of circles independently of the domain chosen.

7.1 Computer software

There are many computer software packages available for calculating conformal maps. Besides the two example programs made for this text (code in appendices), there are widely used packages available for download from the Internet. The best general purpose program seems to be the Schwarz-Christoffel toolbox [4], which has good documentation and is easy to use. The Zipper program [13] can in many cases be faster, but it requires being a little more careful with the input and checking of the output. Stephenson's CirclePack program [22] is not very fast compared to the two others, but Stephenson has suggested that circle packing could be used to make a coarse approximation that could be used as an initial guess for the prevertices problem in the Schwarz-Christoffel method [24].

7.2 Usage

The numerical calculation of conformal mappings was much used in computational physics [9], in many two-dimensional problems involving Laplace's equation. In recent years, however, numerically solving partial differential equations, especially by the finite element method, has taken over as the preferred method for solving such numerical physics problems [16, 18]. The increase in available computing power has made calculations in three dimensions feasible and the added flexibility of being able to solve other types of equations in the same way has made this the dominant method in computational physics.

Conformal mappings of the plane are closely related to the theory of analytic functions of a complex variable, and mathematical research on the two are closely related and benefit from eachother [9].

The Schwarz-Christoffel formula has had a large impact on mathematical research. The Schwarz reflection principle was first presented by Schwarz in his papers about the Schwarz-Christoffel formula. It was also used to prove the Riemann mapping theorem [25].

The geodesic and Zipper algorithms have been used in mathematical research as an approximate method for conformal welding and as a discretization of the Loewner differential equation [17]. The Zipper program has been used to explore Schramm-Loewner evolution, which is a solution to Loewner's differential equation with Brownian motion as input, generating a family of planar curves. This is a conformally invariant stochastic process which can be studied under conformal maps made by the Zipper program [17].

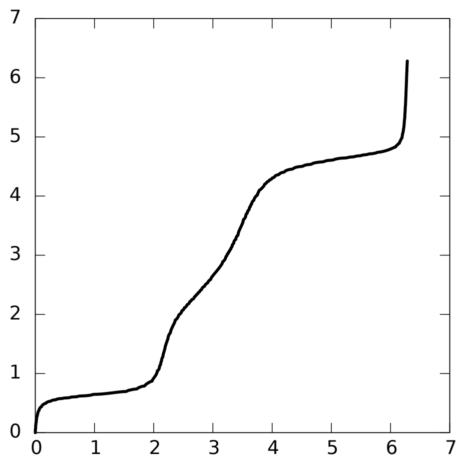


Figure 7.1: Welding map for the domain Ω used in earlier chapters, see for example figure 1.5.

Let $f : \mathbb{D} \to \Omega$ be the inverse of the Riemann map we have considered earlier in this text, and let $g : \mathbb{D}' \to \Omega'$ be the inverse of the map from the complement of Ω. Then $h = g^{-1} \circ f : \mathbb{T} \to \mathbb{T}$, where $\mathbb{T}$ is the unit circle, is called the conformal welding of $\partial\Omega$. For a smooth $\partial\Omega$, h is smooth, univalent and injective. See figure 7.1 for a welding map for the domain Ω from figure 1.5, made with the Zipper program [13]. Welding maps can be used for pattern recognition [2].

Circle packing methods have been used in recent research to represent and visualize surfaces in three dimensional space. Discrete conformal geometry can be used on surfaces in space in medical imaging, computer graphics, nano science and image analysis [23]. The best examples of practical uses of circle packing comes from Hurdal's research [8] on flat maps of the human cerebrum. With new technology available to make three dimensional images of the human brain (MRA, fMRI, PET), comes a need for representing these images in a usable way for the medical researchers. Most of the connections in the human brain exist on the surface, but this surface has creases and folds that hide large parts of it and makes a regular three dimensional image hard to use. Computer programs using circle packing to discretely represent conformal geometry can transform the three dimensional image into a flat map [8]. See figure 7.2 for example images.

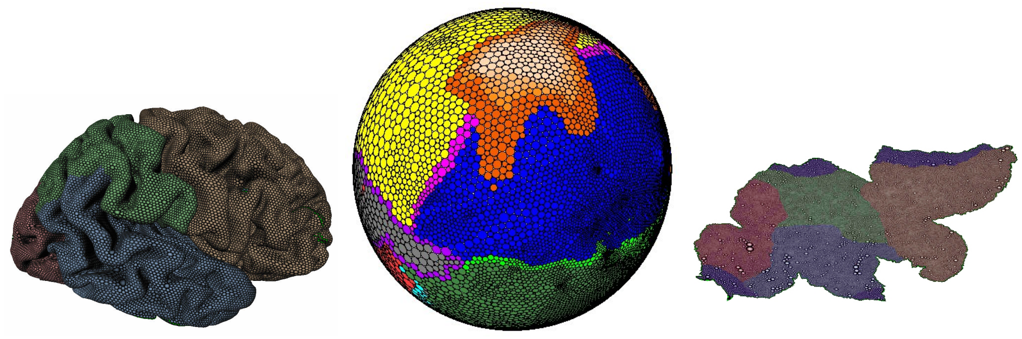


Figure 7.2: Examples from Hurdal's research on flat maps of the brain. Figures from [8].

Miscellaneous computer code A

A.1 Transforming a bitmap with the $f(z) = z^2$ function

Figure 1.7 was made with the following program. It is written in Python and uses PyGame to read a picture, assumed to be located in the first quadrant of the complex plane. It then calculates the preimage of each point in the upper half-plane and colors the point with the color of the nearest pixel in the original image.

```
import pygame,math,cmath

orig = pygame.image.load("clock.jpg");
oversq2 = 1/math.sqrt(2)

size = orig.get_height()
white = 255,255,255

newimage = pygame.Surface((size*2,size))

for x in xrange(newimage.get_width()):
        for y in xrange(newimage.get_height()):
                coords = (float(x)-size)/size + 1j*(1-float(y)/size)
                transformed = cmath.sqrt(coords)
                origcoords = int(transformed.real*size), (size-1)-int(transformed.imag*
                    size)
                if origcoords[0] >= size or origcoords[1] >= size or origcoords[0] < 0
                    or origcoords[1] < 0:
                        newimage.set_at((x,y),white)
                else:
                        newimage.set_at((x,y),orig.get_at(origcoords))

pygame.image.save(newimage,"clock_z2.jpg");
```

A.2 Flow along a wall

Figure 3.1 was made with the MATLAB Schwarz-Christoffel toolbox with the following code.

```
p = polygon([-i,i,Inf],[1/2,3/2,-1]);
f = hplmap(p);
axis([-3 3 -1.5 2]), hold on
plot(f,0,0.15*(1:40))

p = polygon([-1,i,1,Inf],[1-1/4,3/2,1-1/4,-1]);
f = hplmap(p);
axis([-2 2 -0.25 2]), hold on
plot(f,0,0.05*(1:40))
```

Computer program code for the geodesic algorithm B

Example implementation of the geodesic algorithm written in Python. Reads a file with the polygon and files with points to be mapped by the forward map ϕ and the inverse map ϕ^{-1}. The forward map gives good results, but the inverse map has precision problems.

```
import sys,cmath

# the first map phi_1 that maps a straight line segment
# to the real axis and the polygon to the upper half plane
def phi1(z0,z1,z):
        if z == z0: return float("Infinity")
        return cmath.sqrt((z-z1)/(z-z0))

# the inverse of phi_1
def phi1inv(z0,z1,z):
        if z == float("Infinity"):
                return z0
        sq = z**2
        return (z1-sq*z0)/(1-sq)

# the i-th map made from the basic map f_a
def phii(a,z):
        if z == a:
                return 0
        if z == float("Infinity"):
                if abs(a.imag) <= 1e-14:
                        return float("Infinity")
                else:
                        L = a.real/(1j*a.imag)
        else:
                L = a.real*z / (abs(a)**2 + 1j*a.imag*z)

        S = cmath.sqrt(L**2-1)
        if S.imag*L.imag < 0:
                S = -S
        return S
```

```
# the inverse of the i-th map
def phiiinv(a,z):
        sq = cmath.sqrt(z**2+1)
        if sq.imag*z.imag < 0:
                sq = -sq
        if abs(a.real - sq*1j*a.imag) <= 1e-11:
                return float("Infinity")
        if z == float("Infinity"):
                return abs(a)**2/(1j*a.imag)
        return sq*abs(a)**2/(a.real - sq*1j*a.imag)

# the (n+1)-th map mapping a halfdisc in the
# upper half plane to the whole of the upper
# half plane
def phinp1(a,z):
        if z == float("Infinity"):
                return -(1j*a)**2
        if abs(1-z/a) < 1e-14:
                return float("Infinity")
        return -(1j*z/(1-z/a))**2

# the inverse of the (n+1)-th map
def phinp1inv(a,z):
        if abs(z) <= 1e-11:
                return 0
        if z == float("Infinity"):
                return a
        if abs(z+(1j*a)**2) <= 1e-11:
                return float("Infinity")
        sq = cmath.sqrt(-z)
        if sq.imag*z.imag < 0:
                sq = -sq
        return sq*a/(1j*a+sq)

# the map from the upper half plane to
# the unit disc
def phidisc(a,z):
        if z == float("Infinity"):
                return complex(1)
        aconj = a.real-1j*a.imag
        if abs(z-aconj) < 1e-14:
                return float("Infinity")
        return (z-a)/(z-aconj)

# the inverse of phidisc
# maps the unit disc to the upper half plane
def phidiscinv(a,z):
        if z == complex(1):
                return float("Infinity")
        aconj = a.real-1j*a.imag
        if abs(z-1) < 1e-11:
                return float("Infinity")
        #return (1/(aconj-a))*(aconj*z -a)/(z-1)
        return (aconj*z-a)/(z-1)


# writes out the vertices z_i, letting infinity
# be written out as the origin
def writez(filename):
        with open(filename,"w") as f:
                for i in xrange(len(z_i)):
                        p = z_i[i]
                        if p != float("Infinity"):
                                f.write(str(p.real) + "␣" + str(p.imag)+"\n")
                        else:
                                f.write("0␣0␣#␣Infinity\n")

# writes out the grid with every pair
# of points grouped to allow for grid plotting
def writegrid(v,filename):
        with open(filename,"w") as f:
                for i in xrange(len(v)):
                        p = v[i]
                        if p != float("Infinity"):
                                f.write(str(p.real) + "␣" + str(p.imag)+"\n")
                        else:
                                f.write("0␣0␣#␣Infinity\n")
                        if i%2 == 1:
                                f.write("\n")


# read in the vertices of the polygon
z_i = []
for line in open("polygon.txt").xreadlines():
        a = line.split()
        z_i.append(float(a[0]) + float(a[1])*1j)

# read in the grid (or other points to be mapped)
grid = []
for line in open("grid.txt").xreadlines():
        if len(line) > 1:
                a = line.split()
                grid.append(float(a[0]) + float(a[1])*1j)

# set the interior point (conformal center) to the origin
z_i.append(complex(0))

# apply the first map
z0,z1 = z_i[0],z_i[1]
z_i = map(lambda x: phi1(z0,z1,x),z_i)
grid = map(lambda x: phi1(z0,z1,x),grid)

# apply the i-th maps, saving the zeta_k for inverse mapping
zeta = {}
for i in xrange(2,len(z_i)-1):
        print i
        a = z_i[i]
        zeta[i] = a
        z_i = map(lambda x: phii(a,x),z_i)
        grid = map(lambda x: phii(a,x),grid)

# apply the (n+1)-th map
chinp1 = z_i[0]
z_i = map(lambda x: phinp1(chinp1,x), z_i)
grid = map(lambda x: phinp1(chinp1,x), grid)

# finally, map to the disc and write out the polygon
# map and the grid
```

```
zint = z_i[-1]
z_i = map(lambda x: phidisc(zint,x), z_i)
grid = map(lambda x: phidisc(zint,x), grid)

writez("map.txt")
writegrid(grid,"gridout30.txt")

# read in the points to be inversely mapped
# (for example Carleson grid)
cgrid = []
for line in open("carlesongrid.txt").xreadlines():
        if len(line) > 1:
                a = line.split()
                cgrid.append(float(a[0]) + float(a[1])*1j)


# inverse map to the half plane
z_i = map(lambda x: phidiscinv(zint,x),z_i)
cgrid = map(lambda x: phidiscinv(zint,x),cgrid)

# inverse of the (n+1)-th map
z_i = map(lambda x: phinp1inv(chinp1,x),z_i)
cgrid = map(lambda x: phinp1inv(chinp1,x),cgrid)

# inverse of the i-th map
for i in reversed(xrange(2,len(z_i)-1)):
        a = zeta[i]
        z_i = map(lambda x: phiiinv(a,x),z_i)
        cgrid = map(lambda x: phiiinv(a,x),cgrid)

# inverse of phi_1
z_i = map(lambda x: phi1inv(z0,z1,x),z_i)
cgrid = map(lambda x: phi1inv(z0,z1,x),cgrid)

# write out the inverse points and the identity map
# for error checking. This could give bad results
# because of precision problems.
writegrid(cgrid,"gridoutcarleson.txt")
writez("mapidentity.txt")
```

Computer program code for the circle packing method C

The code for this example implementation of the circle packing program was written in Python using PyGame and TiKZ for drawing the results. The code is shortened to be more concise and readable, so much of the drawing code is not included.

C.1 Reading the polygon from a file

```
from Numeric import arange
import sys,math,cmath

gridres = 45

# Reads a file with a polygon where each line is one point with x and y coordinates
# separated by a space
points = map(lambda x: map(float, x.rstrip().split("␣")), list(sys.stdin))

# Finds a suitable drawing area for the given polygon
xmin,xmax,ymin,ymax = float('Infinity'),float('-Infinity'),float('Infinity'),float('-
    Infinity')
for k in points:
        if k[0] < xmin: xmin = k[0]
        if k[0] > xmax: xmax = k[0]
        if k[1] < ymin: ymin = k[1]
        if k[1] > ymax: ymax = k[1]

if xmax-xmin > ymax-ymin:
        xmin,xmax = xmin - (xmax-xmin)*0.20, xmax + (xmax-xmin)*0.20
        ymin,ymax = ymin + (ymax-ymin)/2 - (xmax-xmin)/2, ymax - (ymax-ymin)/2 + (xmax-
            xmin)/2
else:
        ymin,ymax = ymin - (ymax-ymin)*0.20, ymax + (ymax-ymin)*0.20
        xmin,xmax = xmin + (xmax-xmin)/2 - (ymax-ymin)/2,xmax - (xmax-xmin)/2 + (ymax-
            ymin)/2
```

C.2 Miscellaneous functions

```
# Tests if a point is inside the polygon
def inpoly((x,y)):
        c = False
        for p in xrange(len(points)):
                a = points[p]
                b = points[(p+1)%len(points)]
                if((a[1] > y) != (b[1]>y)):
                        if x < (b[0]-a[0])*(y-a[1])/(b[1]-a[1])+a[0]:
                                c = not c
        return c


# Polygonal coordinates to screen coordinates
def scoord(p):
        return ((p[0]-xmin)/(xmax-xmin)*screenx, (-p[1]-ymin)/(ymax-ymin)*screeny)


# Grid coordinates (every point is a circle center in the grid)
# is converted to screen coordinates
def pcoord(p):
        if p[1]%2 == 1:
                return ((float(p[0])/gridres)*(xmax-xmin)+xmin+(xmax-xmin)/gridres/2,
                         (float(p[1])/gridres/2*math.sqrt(3))*(ymax-ymin)+ymin)
        else:
                return ((float(p[0])/gridres)*(xmax-xmin)+xmin,
                         (float(p[1])/gridres/2*math.sqrt(3))*(ymax-ymin)+ymin)


# Assignment of color given coordinates in $\Omega$.
# (HSV to RGB with H as a function of the distance from the center)
def colormap(x,y):
        h = math.log(1+((x)**2+(y)**2))*360
        s = 1
        v = 1

        hi = int(h/60)%6
        f = h/60-int(h/60)
        p = v*(1-s)
        q = v*(1-f*s)
        t = v*(1-(1-f)*s)
        r,g,b = 0,0,0
        if hi == 0:
                r,g,b = v,t,p
        if hi == 1:
                r,g,b = q,v,p
        if hi == 2:
                r,g,b = p,v,t
        if hi == 3:
                r,g,b = p,q,v
        if hi == 4:
                r,g,b = t,p,v
        if hi == 5:
                r,g,b = v,p,q
        return (255*r,255*g,255*b)
```

C.3 Construction of the circle complex

```
# The intersection graph that is to be
# constructed from the grid
graph = {}

origo = None            # The circle that becomes f(c)=0
origo2 = None           # The circle that is tangent to the circle at
                        # the origin and lies on the positive real axis
originalplacement = {}  # Positioning of the circles in $\Omega$
color = {}              # Colors for the colored plot
colorno = 1
for y in xrange(int(gridres*2/math.sqrt(3))):
        for x in xrange(gridres):
                p = pcoord((x,y))
                if inpoly(p):                   # Inside the polygon?
                        graph[(x,y)] = set()
                        if (x-1,y) in graph.keys():
                                graph[(x,y)].add((x-1,y))
                                graph[(x-1,y)].add((x,y))
                        if y%2 == 0:
                                if (x-1,y-1) in graph.keys():
                                        graph[(x,y)].add((x-1,y-1))
                                        graph[(x-1,y-1)].add((x,y))
                        else:
                                if (x+1,y-1) in graph.keys():
                                        graph[(x,y)].add((x+1,y-1))
                                        graph[(x+1,y-1)].add((x,y))
                        if (x,y-1) in graph.keys():
                                graph[(x,y)].add((x,y-1))
                                graph[(x,y-1)].add((x,y))
                        w = 2
                        if origo and not origo2:
                                origo2 = (x,y)
                        if not origo and pcoord((x,y))[0] > 0 and pcoord((x,y))[1] > 0:
                                origo = (x,y)

                        color[(x,y)] = colormap(p[0],p[1])
                        originalplacement[(x,y)] = (scoord(p))[0]+1j*(scoord(p))[1]


# Finds the order of three tangent circles.
# True if they lie in counter-clockwise order
# False otherwise
def orientation(a,b,c):
        v1 = b-a
        v2 = c-a
        cpz = v1.real*v2.imag - v1.imag*v2.real
        if cpz > 0: return True
        else: return False


# Sort the adjacency list on counter-clockwise order
def sortverts(c,l):
        l = list(l)
        s = [l.pop()]
        i = 0
        while len(l) > 0:
                if l[i] in graph[s[-1]] and orientation(originalplacement[c],
                    originalplacement[l[i]],originalplacement[s[-1]]):
                        s += [l.pop(i)]
                        i = 0
                elif l[i] in graph[s[0]] and orientation(originalplacement[c],
                    originalplacement[s[0]],originalplacement[l[i]]):
                        s = [l.pop(i)] + s
                        i = 0
                else:
                        i = (i+1)%len(l)
        return s


for p1 in graph.keys():
        graph[p1] = sortverts(p1,graph[p1])
```

C.4 Calculating the radius list

```
err = float("Infinity")
radii = {}
for p in graph.keys():
        radii[p] = 150

p = origo


# Finds hyperbolic angle in a triangle with vertices
# in the centers of three tangent circles
def angle(a,b,c):
        x,y,z = radii[a],radii[b],radii[c]
        if y == 150:
                y,z = z,y
        if x == 150:
                return 0
        if y == 150 and z == 150:
                return math.acos(1-2*math.exp(-2*x))
        if z == 150:
                return math.acos( (math.cosh(x+y)-math.exp(y-x)) / (math.sinh(x+y)))
        return math.acos( (math.cosh(x+y)*math.cosh(x+z)-math.cosh(y+z)) / (math.sinh(x
            +y)*math.sinh(x+z)) )


# Finds the circle packing.
minerr = float("Infinity")
while err > 1e-6:
        err = 0
        for p in graph.keys():
                if len(graph[p]) == 6:
                        r = 0
                        for p1 in range(len(graph[p])):
                                r += angle(p,graph[p][p1],graph[p][(p1+1)%len(graph[p])
                                    ])
                        e = r - 2*math.pi
                        if abs(e) > err:
                                err = abs(e)
                        if e > 0:
                                radii[p] *= 1+min(err/10,0.1)
                        else:
                                radii[p] *= 1-min(err/10,0.1)
```

C.5 Placing circles in the Poincaré model of the unit disc

```
# Radius from the origin in the unit disc for a circle,
# given hyperbolic radius (as found in the circle packing)
def hyp2eucradius(r):
        return (math.exp(r)-1)/(math.exp(r)+1)

# Transforms coordinates from the unit disc to the screen
def disc2screen(p):
        return ((float(p[0])+1)/2*screenx, (-float(p[1])+1)/2*screeny)

# Rotates a vector p with the angle a around the origin
def rot(p,a):
        return (math.cos(a)*p[0]-math.sin(a)*p[1], math.sin(a)*p[0]+math.cos(a)*p[1])

# Euclidean distance between two vectors
def dist(a,b):
        return math.sqrt((b[0]-a[0])**2+(b[1]-a[1])**2)

# The general conformal automorphism on the unit disc
def phi(z,znull,lmbda):
        return lmbda*(z-znull)/(1-znull.conjugate()*z)

# The inverse of the general conformal automorphism on the unit disc
def phiinverse(z,znull,lmbda):
        return (z+lmbda*znull)/(lmbda+znull.conjugate()*z)

# Initialization of variables
p1 = origo
placed = {}
placedhyp = {}
placed[p1] = (0,0)
placedhyp[p1] = (0,0)
p2 = origo2
placed[p2] = ((hyp2eucradius(radii[p1])+hyp2eucradius(radii[p1]+2*radii[p2]))/2, 0)
placedhyp[p2] = (hyp2eucradius(radii[p1]+radii[p2]),0)
plist = list(graph.keys())
plist.remove(p1)
plist.remove(p2)
last = p2
lastlast = p1
a_b = []
circles = {}
circles[p1] = (placed[p1],hyp2eucradius(radii[p1]))
circles[p2] = (placed[p2], (hyp2eucradius(radii[p1]+2*radii[p2])-hyp2eucradius(radii[p1
    ]))/2)


# Laying out of circles in the hyperbolic unit disc
while len(plist) > 0:
        found = False
        for p in plist:
```

```
for i in xrange(len(graph[p])):
        if graph[p][i] in placed.keys() and graph[p][(i+1)%len(graph[p
            ])] in placed.keys():
                # Let graph[p][(i+1)%len(graph[p])] be circle A
                    centered on (0,0)
                # \phi with z_0 = c_A, \lambda = e^(i \theta), \theta =
                     -atan2(c_B-c-A)
                A = graph[p][(i+1)%len(graph[p])]
                B = graph[p][i]
                p3 = p
                znull = placedhyp[A][0]+1j*placedhyp[A][1]
                phb2 = phi(placedhyp[B][0]+1j*placedhyp[B][1],znull,1)
                theta = -cmath.phase(phb2)
                lmbda = cmath.exp(1j*theta)
                a = angle(A,B,p)
                centre = ((hyp2eucradius(radii[A]) + hyp2eucradius(
                    radii[A]+2*radii[p]))/2, 0)
                centre = rot(centre,a)
                centrehypeuc = (hyp2eucradius(radii[A] + radii[p]), 0)
                centrehypeuc = rot(centrehypeuc,a)
                radiuspoint = ((hyp2eucradius(radii[A])),0)
                radiuspoint = rot(radiuspoint,a)
                radius = dist(centre,radiuspoint)
                radiuspoints = map(lambda x: (centre[0]+1j*centre[1]) +
                     radius*cmath.exp(1j*2*cmath.pi*x/3),range(3))
                if radius > 0:
                        centrehypeuc = phiinverse(centrehypeuc[0]+1j*
                            centrehypeuc[1],znull,lmbda)
                        centre = phiinverse(centre[0]+1j*centre[1],
                            znull,lmbda)
                        radiuspoint = phiinverse(radiuspoint[0]+1j*
                            radiuspoint[1],znull,lmbda)
                        radiuspoints = map(lambda x: phiinverse(x,znull
                            ,lmbda), radiuspoints)
                        z1,z2,z3 = radiuspoints
                        cc = (abs(z1)**2*(z2-z3)+abs(z2)**2*(z3-z1)+abs
                            (z3)**2*(z1-z2))/(z1*(z3.conjugate()-z2.
                            conjugate())+z2*(z1.conjugate()-z3.
                            conjugate())+z3*(z2.conjugate()-z1.
                            conjugate()))
                        rr = abs(z1-cc)
                        drawcircle((cc.real,cc.imag),rr,color[p])
                        circles[p] = ((cc.real,cc.imag),rr)
                        placedhyp[p] = (centrehypeuc.real,centrehypeuc.
                            imag)
                        placed[p] = (centre.real,centre.imag)
                else:
                        print a,radii[A],radii[B],radii[p],centrehypeuc
                            ,centre,znull,lmbda
                        circles[p] = ((0,0),0)
                plist.remove(p)
                found = True
                break
if found: break
```

Bibliography

[1] Lars Ahlfors. *Complex Analysis*. McGraw-Hill Science/Engineering/Math, 1979.

[2] Christopher J. Bishop. Conformal welding and koebe's theorem. Text available from `http://www.math.sunysb.edu/~bishop/lectures`, 2007.

[3] Charles R. Collins and Kenneth Stephenson. A circle packing algorithm. *Computational Geometry: Theory and Applications*, 25:233–256, 2003.

[4] Toby Driscoll. The schwarz-christoffel toolbox for matlab. Software available from `http://www.math.udel.edu/~driscoll/software/`.

[5] Frederick W. Gehring and Kari Hag. *The Ubiquitous Quasidisk*. Unpublished draft, 2009.

[6] Donald Hearn and M. Pauline Baker. *Computer Graphics with OpenGL*. Pearson Prentice Hall, 2004.

[7] Juha Heinonen. What is a quasiconformal mapping? *Notices of the AMS*, 53(11):1334–1335, 2006.

[8] Monica K. Hurdal and Ken Stephenson. Cortical cartography using the discrete conformal approach of circle packing. *NeuroImage*, 23:119–128, 2004.

[9] V. I. Ivanov and M. K. Trubetskov. *Handbook of Conformal Mapping with Computer-Aided Visualization*. CRC Press, 1994.

[10] Erwin Kreyszig. *Advanced Engineering Mathematics*. John Wiley & Sons, Inc., 2006.

[11] Hans Lundmark. Visualizing complex analytic functions using domain coloring. Text available online at `http://www.mai.liu.se/~halun/complex/domain_coloring-unicode.html`, 2004.

[12] Bjørnar Steinnes Luteberget. Approksimering av riemann-avbildninger ved hjelp av sirkelpakking. Project work at NTNU, 2010.

[13] Donald E. Marhsall. Numerical conformal mapping software: zipper. Software available from `http://www.math.washington.edu/~marshall/zipper.html`.

[14] Donald E. Marshall. Complex analysis course notes. Coures notes available from `http://www.math.washington.edu/~marshall/math536-10.html`, 2009.

[15] Donald E. Marshall. Lens chains and the geodesic algorithm for conformal mapping. Preprint available from `http://www.math.washington.edu/~marshall/preprints/preprints.html`, 2009.

[16] Donald E. Marshall. Personal communication, 2010.

[17] Donald E. Marshall and Steffen Rohde. Convergence of a variant of the zipper algorithm for conformal mapping. *SIAM J. Numer. Anal.*, 45, 6:2577–2609, 2007.

[18] Robert Nilssen. Personal communication, 2010.

[19] Wikipedia online encyclopedia. Figure from hls-hsv models. Available online at `http://en.wikipedia.org/wiki/File:Hsl-hsv_models.svg`, 2010.

[20] Burt Rodin and Dennis Sullivan. The convergence of circle packings to the riemann mapping. *J. Differential Geometry*, (26):349–360, 1987.

[21] Donald Sarason. *Complex Function Theory*. American Mathematical Society, 2007.

[22] Kenneth Stephenson. Circlepack. Software available from `http://www.math.utk.edu/~kens/CirclePack/`.

[23] Kenneth Stephenson. Circle packing: A mathematical tale. *Notices of the AMS*, 50(11):1376–1388, 2003.

[24] Kenneth Stephenson. *Introduction to Circle Packing*. Cambridge University Press, 2005.

[25] L. N. Trefethen. Numerical computation of the scwarz-christoffel transformation. *SIAM J. Sci. Stat. Comput.*, 1:82–102, 1980.

[26] Eric W. Weisstein. Circle packing. MathWorld – A Wolfram Web Resource. `http://mathworld.wolfram.com/CirclePacking.html`.