DIFFERENTIAL EQUATIONS
AND
CONTROL PROCESSES
N. 3, 2024
Electronic Journal,
reg. N ΦC77-39410 at 15.04.2010
ISSN 1817-2172

http://diffjournal.spbu.ru/
e-mail: jodiff@mail.ru

Computer software for the investigation of
differential equations, dynamical systems, and control processes

Light and fast renderings of limit sets for Fuchsian and Kleinian groups

Alessandro Rosa
Software developer
alessandro.a.rosa@gmail.com

Abstract. After reviewing some basic theory on groups of Möbius transformations in $\mathbb{C}^1$, we will discuss features and drawbacks of the classic approach for the digital rendering of their limit sets. We will then introduce the technical details of a new easier and faster algorithm, simply based upon numerical base conversion. Finally, we will draw the guidelines for the implementation and examine the benefits. Readers are just assumed to know complex arithmetics and how to program graphic plots.

Keywords: Limit set, dynamical systems, Kleinian group, linear fractional transformation, circle inversion.

1 The framework

Let $\mathcal{M}$ be the collection of automorphisms g_n of the complex plane $\mathbb{C}$ in the form of the linear fractional transformation

$$z_1 = g_n(z) = \frac{a_n z_0 + b_n}{c_n z_0 + d_n}, \quad a_n, b_n, c_n, d_n, z_0 \in \mathbb{C}, \quad ad - bc \neq 0, \quad n \in \mathbb{Z} \qquad (1)$$

also termed *Möbius transformation* (or *map*) after the in-depth studies carried out by August Ferdinand Möbius (1790–1868) during the beginnings of the XIX century. This map is *conformal* because angles and orientation are preserved. Simple computations show that both inversion, $g_n^{-1}(z) = \dfrac{d_n z - b_n}{-c_n z + a_n}$, and composition, $g_1 \circ g_2 = g_1(g_2) = g_3$, are closed in $\mathcal{M}$. The composition of Möbius maps enjoys the same properties as of 2×2 matrix, namely

$$\begin{pmatrix} a_1 & b_1 \\ c_1 & d_1 \end{pmatrix} \circ \begin{pmatrix} a_2 & b_2 \\ c_2 & d_2 \end{pmatrix} = \begin{pmatrix} a_1a_2 + b_1c_2 & a_1b_2 + b_1d_2 \\ a_2c_1 + c_2d_1 & b_2c_1 + d_1d_2 \end{pmatrix},$$

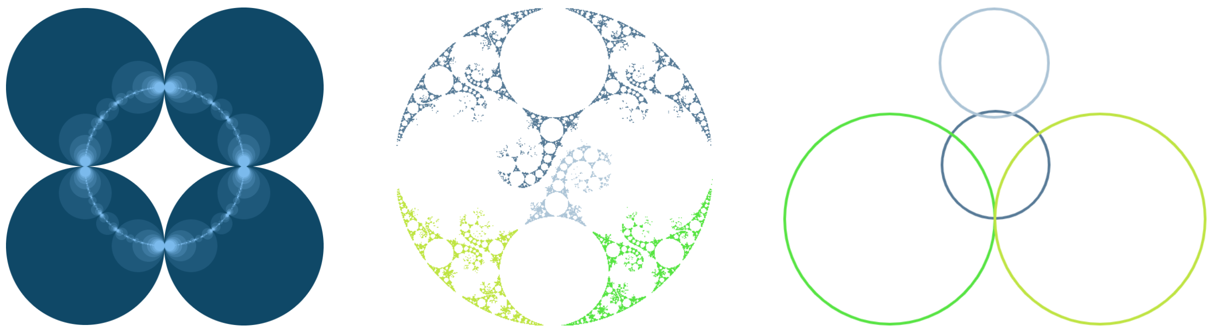
and the associative property: $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$. This relation can be extended to arbitrarily many elements in the chain

$$g_1 \circ g_1 \circ g_2 \circ g_1^{-1} \circ g_1^{-2} \circ g_2 \circ \dots \ . \tag{2}$$

Conversely, every element in $\mathcal{M}$ cannot be regarded as primitive, being the composition of arbitrarily many transformations g_n. The identity $I(z) = z = g \circ g^{-1}$ is a special Möbius map with coefficients $a = d = 1$, $b = c = 0$, $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, and it is the neutral element in $\mathcal{M}$: $g_n \circ I = I \circ g_n = g_n$. The inverse g^{-1} commutes with g, i.e., $g \circ g^{-1} = g^{-1} \circ g = I$; Möbius maps composition is not in general commutative anyway. We have collected sufficient information to classify $\mathcal{M}$ as an *algebraic group*, known as *function* or *Möbius group* and including subgroups $\mathcal{G} \subset \mathcal{M}$ that enjoy special properties. Subgroups are characterized by their *generators*, Möbius maps assumed to give rise to all other elements in $\mathcal{G}$. The term *generator* may sound fictitious because, according to (2), this map cannot be prime and indecomposable; it just refers to the role played in the subgroup and to special properties enjoyed by its coefficients.

From the combination of (2) with (1), we obtain a ordered sequence of values, $z_1, z_2, \dots, z_n$, defined *orbit* in the theory of dynamical systems. (Iteration is a special case of chains (2) including one only transformation: $g_1 \circ g_1 \dots g_1 \circ \dots$.) We say that the action of $\mathcal{G}$ is *freely discontinuous* when $g(U) \cap U = \emptyset$ holds, for $z \in U \subset \mathbb{C}$ [13, p. 16]. It is proven that orbits of Möbius maps are *asymptotically stable* [13, p. 17], thus (1°) it makes sense to question about their final destination, the *limit value*, and the collection of such values, the *limit set* Λ; we also understand that computing Λ is not algorithmically feasible, because of the infinitely many combinations (2) involved by asymptoticity. Orbits shall be necessarily halted at some finite step $d < \infty$: we can only deal with approximations Λ_d of $\Lambda \equiv \Lambda_\infty$, where $\lim\limits_{d \to \infty} \Lambda_d = \Lambda_\infty \equiv \Lambda$.

The studies on subgroups $\mathcal{G} \subset \mathcal{M}$ arose between the late 1870s and 1890s and represented a branch of geometric groups theory, aiming at inspecting the



(a) Fuchsian (b) Quasi-Fuchsian

Figure 1: **Simple examples of limit sets.** (a) Generators are the four and bigger inversion circles, tangent to each other: the limit set is a circle and the subgroup is defined *Fuchsian*; (b) the isometric circles of the generators are shown in the little picture on the far right.

fate of (2) and at classifying the limit sets including more than two points. Along the development of this new theory, the classification rules of the subgroups $\mathcal{G}$ involved:

(a) the *shape* and *topology* of limit sets: it could be a circle, a general curve and a dust of points for the categories of *Fuchsian*, *quasi-Fuchsian* and *Schottky* groups respectively (fig. 1);

(b) the *numerical nature* of the four coefficients in (1); for example, they are real values in *modular* groups and Gaussian integers in the *Picard*, after Émile Picard (1856-1941) groups;

(c) special *algebraic relations* between coefficients: for example, the *trace operator* $tr^2(g) = (a+d)^2$ gathers Möbius maps into the elliptic, parabolic, hyperbolic or loxodromic category, whether $0 \le tr^2(g) < 4$, $tr^2(g) = 4$, $tr^2(g) > 4$ and $tr^2 \in \mathbb{C}\backslash[0,4]$ respectively.

All these are the basic notions we need about groups of Möbius transformations in this work. For in-depth information, see [2, 13, 15].

Circle inversions. Circles are special shapes in the theory of Möbius transformations. The simplest subgroup of $\mathcal{M}$ that enjoy special properties features formulas in the form

$$T(z) = a_C + \frac{r_C^2}{\overline{z - a_C}}. \tag{3}$$

They are called *circle inversions* (or *reflections*) and map particular circles C to themselves, while their interior and exterior are swapped (fig. 2/a). C is centered at $a_C = x_C + iy_C$ and with radius r_C. Conversely, we can determine a_C and r_C in (3) from C. T maps every object outside (resp. inside) C inside (resp. outside) it (fig. 2); then, $T \circ T = I$, or $T^2(z) = I$.

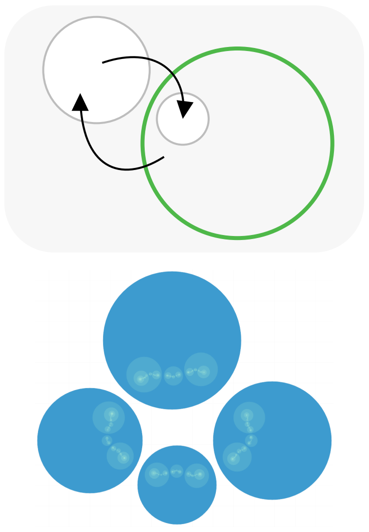


Figure 2: **Circle inversions.** The top diagram illustrates how circle inversion works. At the bottom, the disc images under the action of a subgroup whose generators have no self-intersecting inversion circles and known as of *Schottky* type.

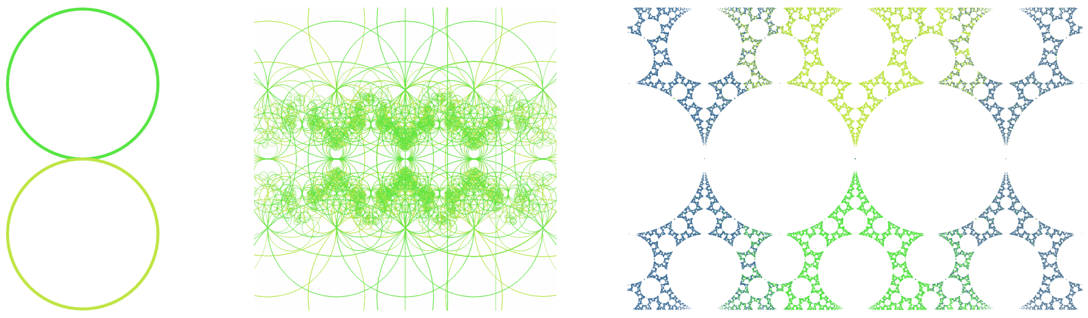
C is the inverse of itself and thus invariant under T; it is defined *inversion* circle. From now on, let it be C_{INV}. The inversion kind belongs to the general class of invariant circles for (1) – given $c \neq 0$, said *isometric* (coming from the union of the two Greek terms *iso* and *metric* = *same size*). Let them be C_{ISO}; they satisfy the equality relation $g(C_{\text{ISO}}) = C_{\text{ISO}} = g^{-1}(C_{\text{ISO}})$.

Isometric circles belong to the toolkit for the exploration of Möbius maps and groups: *invariant* objects prevent ambiguities, so they are essential for grounding mathematical theories. See [7, pp. 23 et ff.]. These circles, or specifically those of inversion for maps in the form (3), are also of help for getting a graphical representation of subgroup generators (fig. 1/a and 2). We will mainly deal with subgroups of *self-inverse* and of *non-self-inverse* Möbius maps, whether the form is (3) or not respectively.

1.1 No fuzzy pictures

The renderings inside these pages are joined by the same goal: displaying the limit sets for subgroups of $\mathcal{M}$. We notice that they show up in two forms. The choice is by no means accidental however: if discs are painted, colors are picked out from palettes that are sorted through a gradient, in order to obtain a chromatic analogy of the shrinking discs sequence; otherwise, if discs are not plot, we only draw the end points (by any color of choice) of the orbits, in order to get the approximation Λ_d. In some sense, these two approaches highlight the *behavior* and *fate* of the orbits respectively.

The choice between *disc images* and the *pointwise representation* is mainly subjected to styling reasons. The subgroup action could induce the disc images to overlap each other; therefore displaying isometric circles is no longer useful because renderings look quite messy and confusing (fig. 3/b). Circles are dropped in favor of points/pixels (the composite neologism from picture, or *pix*, and *el*ement) and orbits are not rendered step by step: only their last



(a) Generators
isometric circles

(b) Disc images
isometric circles

(c) Limit set
points

Figure 3: **Renderings of a same subgroup.** Let a 2-generators subgroup in the *Riley slice* (roughly, subgroups of $\mathcal{M}$ where the coefficients of generators satisfy particular relations). (a) Generators are parabolic. The subgroup action has been rendered through (b) circles and (c) points/pixels. Colors in (c) are associated to the starting generator of each orbit.

element will be processed (fig. 3/c).

Disc images boils down to the representation via two-dimensional objects, here the circle, and therefore the visual quality is, in turn, affected by the covering action of such objects. The shapes of the limit set come to eye during the disc images, as a consequence from the decreasing size of the images, under the subgroup action, of the isometric circles associated to the subgroup generators.

The shapes of limits set $\mathcal{K}$ for subgroups of $\mathcal{M}$ are generally ruled by fractal patterns, like it happens to *Julia sets* $\mathcal{J}$, the limits for the iteration

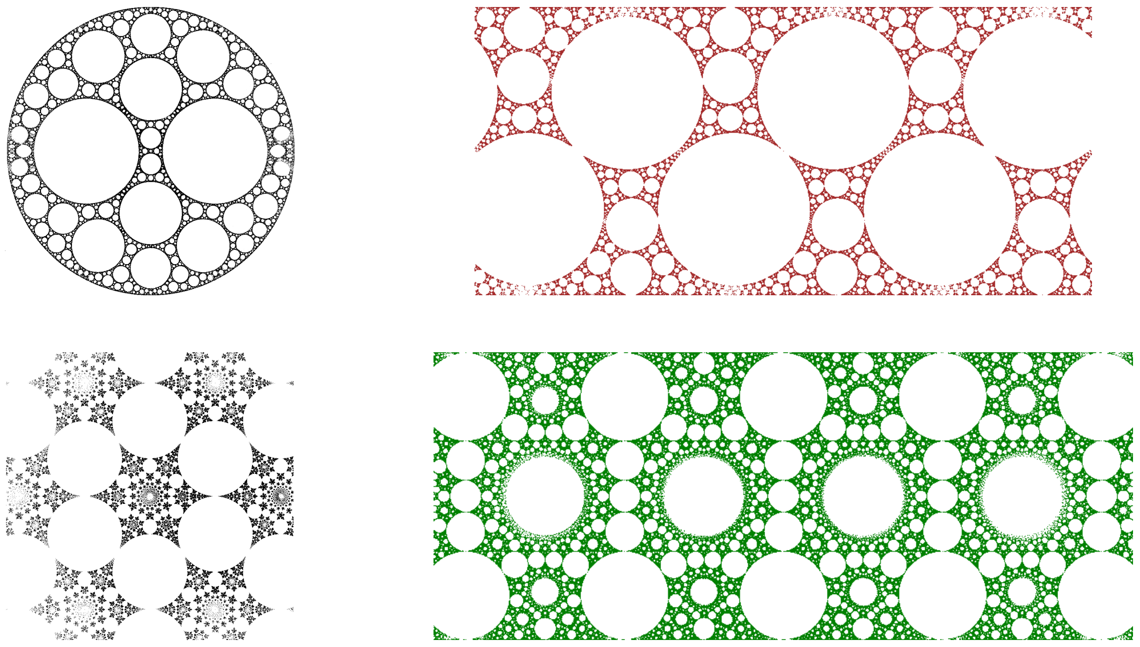


Figure 4: **Circle packing.** Shapes of limit sets showing ways of packing surfaces.

of non-linear functions in one complex variable (fig. 5/b). (These are two well-known kindred theories, based on orbits built according to the criteria of the algebraic structures which functions belong to: groups or singletons.) There exist exceptional groups, called *degenerate*, cracking such circle-like and fractal patterns (fig. 14 at p. 87). There exist limit sets $\mathcal{K}$ spreading in ways that their complement (the set of discontinuity Ω) consist of circles that are said to *tessellate* a given portion of $\mathbb{C}$ (i.e., they *cover*, or *fill*, the space by a well-ordered geometric distribution). Tessellation belongs to a much more intriguing topic, known as *circle packing* and focusing on optimal (i.e., aiming at reducing the gap between the original area and the packing to 0) patterns for filling in areas by means of circles (fig. 4, 5/a); then packing can be assumed as an approximation algorithm.

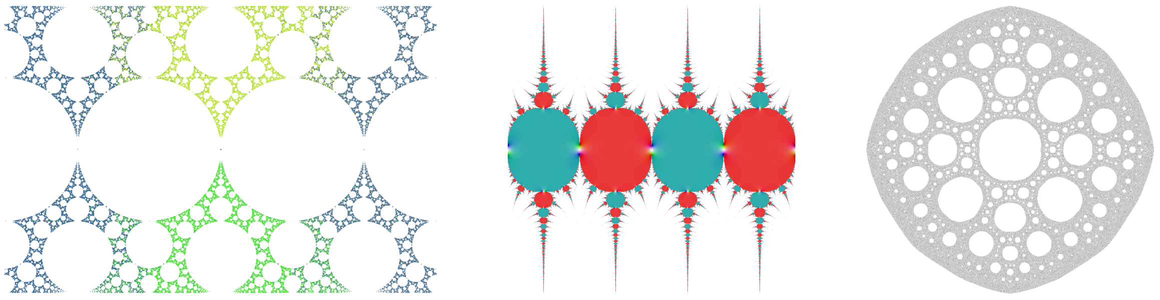


Figure 5: **Resemblances.** (a) A limit set for a subgroup of $\mathcal{M}$ and belonging to the *Riley slice* (see fig. 3); (b) the Julia set for the iteration of $\sin(z)$. On the left, each color binds to the initial generator; on the right, to the final value taken on by the iterated orbit. (c) Apollonian-like Julia set similar to the limit set in fig. 4 [5].

2 Introduction to the lexicographic approach

A clever strategy for optimizing the computation of the chains (2) is required because they could include contiguous pairs $g_k \circ g_k^{-1}$ of inverse generators: such pair resolves into the identity map $I(z)$ which can be safely skipped because redundant.

The earliest renderings, via disc images or limit sets, were already available on paper at the end of the XIX century, in the masterpiece by Fricke and Klein [8]. In the era of digital computing, this problem was tackled through a lexicographic approach based upon a *finite state automata* (see [6].) that generates all the *permutations* of chains (2) whose length is $l = d < \infty$ for the limit set or $l \leq d < \infty$ for disc images representation. This approach was not originally devised for computational goals as it dates back to a time when there

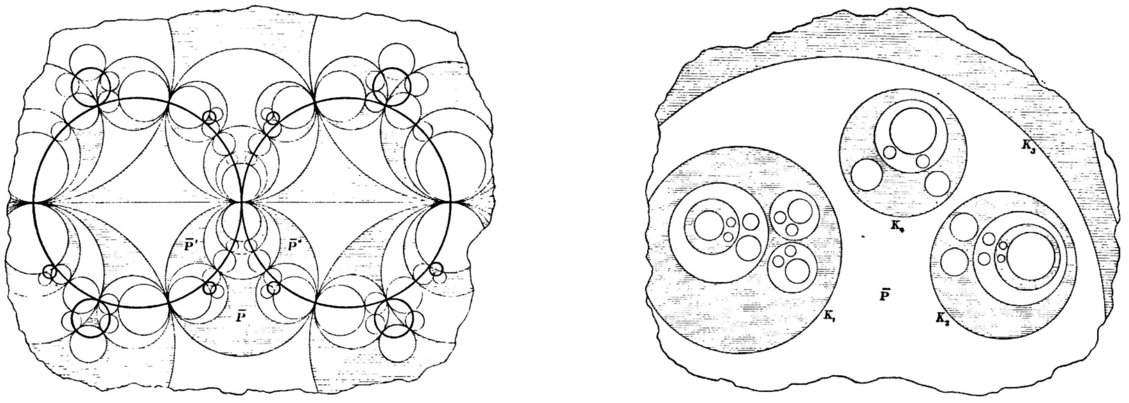


Figure 6: **Disc images renderings in 1897.** See [8, vol. I, p. 429, 437].

was no enough familiarity and confidence with computers, so that *its features were not originally geared to optimizing speed, efficiency, and to saving memory resources.*

2.1 Trees of words

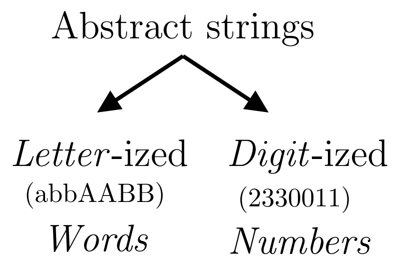


Figure 7: **Materializations.**

There exists a primitive and abstract concept behind the lexicographic approach and essentially relying upon symbolic *concatenation*, which produces objects known as *strings* (fig. 7). In general, the lexicographic approach works on a bounded set $\mathcal{W}_n$ of n symbols, each associated to one and only one generator in the given subgroup; here symbols were chosen to be letters l_n of the Western alphabet (a, b, c, d, …). The set $\mathcal{W}_n$ is defined *alphabet*, because of being the base for encoding the abstract chains (2) into *strings* of letters, analogously defined *words*. These words are open to be read from left to right (LR) or from right to left (RL). In this article, we will read them all in the RL order. Stepping once back, we see that the abstract symbolization, intended as a one-to-one relation between a grapheme and a generator, has materialized into the *letter*ized form (fig. 7). For what follows, it is essential to disengage this binding and consider that *letters just represent a viable choice.*

On this train of ideas, at a larger scale, the lexicographic approach builds the *dictionary*, the collection of finite length words w. Dictionaries are filled by every new word coming by appending single nodes up to some bounded length l. Along the next sections, we will consider the generic 2-generators subgroup

$$g_1 = a, \quad g_2 = b, \quad g_1^{-1} = A, \quad g_2^{-1} = B, \tag{4}$$

where each generator and its inverse binds to small and capital form of the same alphabetic letter. (Unless otherwise specified, the expression 'n-generators subgroup' does not count the inverse maps g_n^{-1}.)

In this case, letters concatenation into words can be schemed through a $(4-1=3)$-branched tree (fig. 8). The writing (4) suggests the composition of identity words aA, Aa, bB, Bb, which *cancel* the action of the last two generators (for example, in $aaA \rightarrow a$) and then prevent the insertion of new nodes because of getting the formation back to existing words. Identities show that formality and action are disjoint, they involve redundant actions and are then negligible. These remarks open to the problem of checking when and how identities occur.

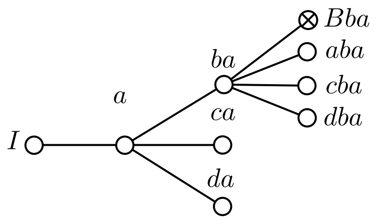
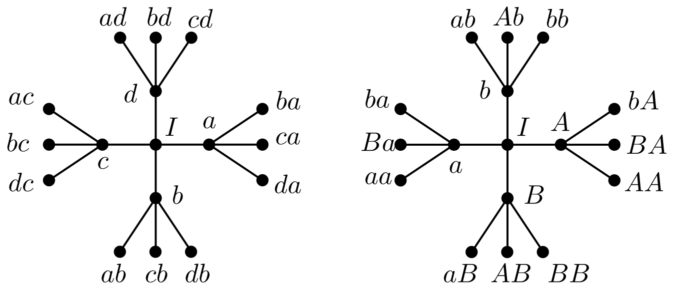


Figure 8: **Tree growth.** Identities aA, Aa, bB, Bb stop the tree growth.



(a) self-inverse (b) non-self-inverse

Figure 9: **Multi-branched trees up to depth 2.**

2.2 Presentations and multiplication tables

The *phyllotaxis*, i.e., how the tree grows, is not wild, because the corresponding subgroup action is defined by a concise list of identity recognition rules. The following example presents a largely applied form of compact writing that just reports generators on the left and identity recognition rules on the right:

$$\langle a, b \mid aa = bb = I \rangle \tag{5}$$

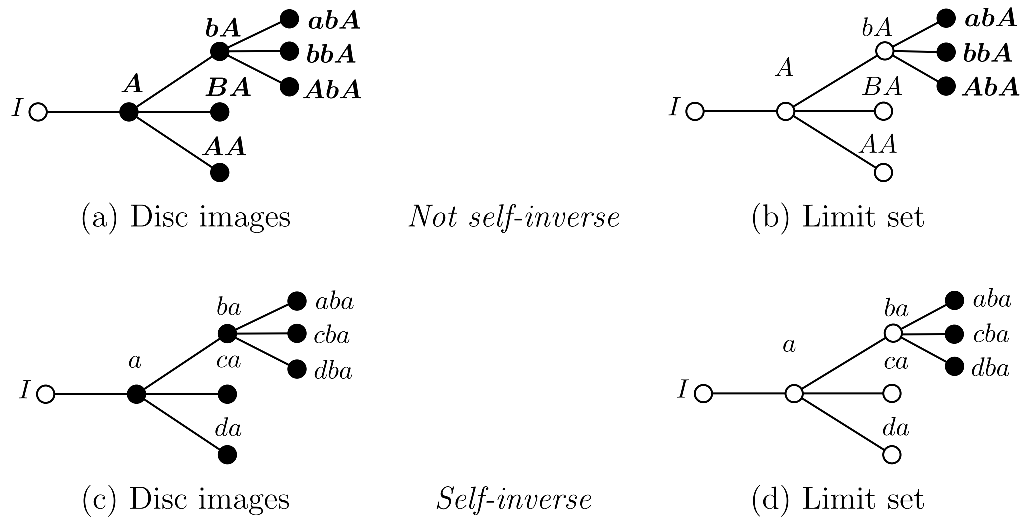
We notice that this is a subgroup of *self-inverse* maps, such as circle inversions for instance (3). The letter I, meaning to the *i*dentity, can be equivalently replaced by the unit value 1. In the example below, we picked subgroup (4), with identities coming from pairs of mutually inverse generators

$$\langle a, b, A, B \mid aA = Aa = bB = Bb = 1 \rangle, \tag{6}$$

ensuring that it is in no way a 2-generators subgroup of *non-self-inverse* maps. The writings (5) and (6) are called group *presentations* (or *defining relations*). Each identity recognition rule on the right is defined *cancellation*, as symbols will be deleted when identities occur. Since the goal of presentations just amounts to obtaining synthetic writings, they turn obsolete as will not fit the action of more complicated groups, being endowed with several composition rules and identity recognitions.

According to the remark at p. 68, there exists no unique and irreducible presentation. Cancellations often appear along the chain composition, bringing different words but same action. For example, the word $aaBb$ resolves into the equivalent aa due to the table (6). Again, let $ababBAA$: after cancelling bB, we get $abaAA$; aA can be dropped too and we finally have abA.

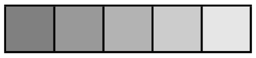
Figure 10 shows four partial trees that have been built up according to presentations (5) and (6) respectively. The vast casuistry of groups teaches that phyllotaxis could be more complicated than the sole cancellation rules. Every cancellation, intended as formal composition, could represent one option for groups of Möbius transformations, because the latter are closed under composition: for example, it may happen that we move from one generator to the other: $a^3 = aaa = b$; or fall into a periodic behavior: $a^3 = aaa = a$. Moreover, all the previous examples shall not induce to believe that the one-to-one relation from



(a) Disc images *Not self-inverse* (b) Limit set

(c) Disc images *Self-inverse* (d) Limit set

Figure 10: **Partial trees up to depth 3.** The entries in bold are those to be rendered.



	a	**b**	**A**	**B**
a	a	b	†	B
b	a	b	A	†
A	†	b	A	B
B	a	†	A	B

(a) *abABB*
Not self-inverse subgroup

	a	**b**	**c**	**d**
a	†	b	c	d
b	a	†	c	d
c	a	b	†	d
d	a	b	c	†

(b) *cabdc*
Self-inverse subgroup

Table 1: **Trasversing the multiplication table.** The tree growth in figs. 10 and 9 is driven by Cayley tables. The cancellations '†' appear in the presentations (6) and (5). We can follow the zig-zag path through the upper bar with the gray shades gradient.

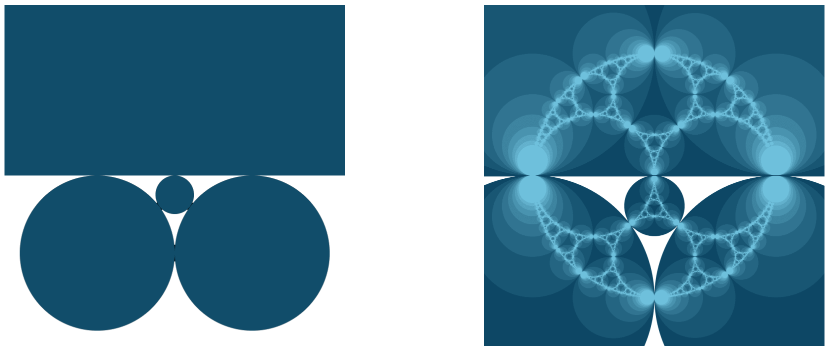
letters to generators is the sole way to detect identities. Suppose we have the rule $ab = A$, then aab turns into $aA = I$.

In this formally variegated scenario, we require the identity recognition rules would be structured in a way more versatile than in the presentation: this task is fulfilled by the so-called *multiplication table*, also known as *Cayley table*, named after Arthur Cayley (1821–1895) and ruling every single step along the formation of each orbit. Every table features the same number of columns as of the generators (including the inverse ones), whereas rows list *all unique combinations* allowed in a given group, including the cancellation rules. Every row is announced by one combination of generators on the far left column, and it is accessed by means of the combination with the cells in the other columns, following a sort of zig-zag path that ends at a cancellation rule (table 1). Presentations are *synthetic* versions of the *analytic* multiplication tables [10, p. 88], which provide specific composition rules besides cancellations.

The two examples in table 1 are the simplest ever, since the only directive to follow for trasversing this table amounts to replacing the current symbol by the next in line, and so, again and again up to some maximal depth or when we would stumble into a cancellation. The (RL) reading of the word abA is equivalent to the following path $A \underset{b}{\rightarrow} b \underset{a}{\rightarrow} a$, running through three rows, one per each symbol. There exist groups whose multiplication tables include rows that are announced by words being longer than one symbol (see the example at [15, p. 359]), such as the path $a \underset{b}{\rightarrow} ba \underset{A}{\rightarrow} Aba$ that running over the table rows announced by a, ba, and Aba. The formal word $Bbba$ will eventually meet the cancellation rule in the row announced by the letter B. This is an excerpt of a multiplication table including longer entries than one letter [15, p. 359]:

	a	b	A	B
bAB	I	I	BA	B
Bab	ba	b	I	I

In any version, either as presentations or as tables, the tests for identity recognition are essential to ensure the correct processing of the subgroup action and they cannot be exempted from implementation of visual rendering algorithms due to the reasons stressed before; otherwise said, missing to perform the cancellation tests brings inaccurate results. *The lexicographic approach* preprocesses *words: it builds them in progression and checks if the newly appended symbol has met an identity rule and then triggered a cancellation.*



(a) Generators (b) Final rendering

Figure 11: **Disc images by circle inversion.** (a) A different disposition of four mutually tangent circles than fig. 1/a was adopted here to work inversion maps (3). The tangency condition is preserved along the construction and it prevents images circles from overlapping each other. Coloring visually enhances the convergent dynamics.

During the early 1990s, digital pictures of limit sets were produced through the lexicographic approach in a few works, such as Manna and Vicsek's [12], Bullets and Mantica's [4], McShane, Parker and Redfern's [14], and Parker's [16]. None of them hit the technical details of the rendering. This gap in the literature was filled in the beginnings of the new millennium, inside the book *Indra's pearls* by Mumford, Series and Wright [15], which provided a very extensive and plain discussion of the lexicographic approach.

3 Drawbacks of the lexicographic approach

Given n generators and chains (2) of maximal depth d, disc images and limit sets require $N_T = \sum_{i=1}^{d} n^i$ (intermediate nodes and leaves) and $N_L = n^d$ words (leaves only, i.e. the number of permutations) respectively. The lexicographic approach features the following additional computation costs:

(1) a first *table* for binding letters to the indexes of the generators stored in an array;

(2) a second *table* for registering the association between the letters of generators and of their inverses;

(3) the *implementation* of bread-first (equivalently, depth-first) *algorithm for running through the words* tree (i.e., trasversing the tree model) and generating the words dictionary up to a finite length;

(4) the *memory space* required to store the dictionary;

(5) the *translation of symbols into indexes* to pick up the generator.

The tables below report the memory sizes of dictionaries including words up to length 17, a sufficient value for accomplishing average quality renderings. Costs will dramatically grow, because even longer words are needed for higher quality, especially to render close-ups or for those groups whose the orbits convergence rate rapidly decreases to 0 near the limit set, as it happens for groups including parabolic generators (see the remarks on the *trace* operator at p. 69).

Compiling the dictionary turns into a very expensive process, which demands long computation times and huge memory resources.

words length	0	1	3	5	7	9	11	13	15	17

Disc images

process steps	1	5	53	485	4373	39365	354293	3188645	28697813	258280325
dictionary size	1B	5B	53B	485B	4.2KB	38KB	346KB	3.04MB	27MB	246.3MB

Limit set

process steps	1	4	9	81	729	6561	59049	531441	4782969	43046721
dictionary size	1B	4B	9B	81B	729B	6.4KB	57.7KB	519KB	4.56MB	41.06MB

Table 2: **Memory size for disc images**. This upper and the lower table have been compiled for groups of four generators of self-inverse maps and of non-self-inverse maps respectively.

The need of huge memory loads has been already pointed out at [15, p. 141], where three approaches were given for working this problem around: one is based upon recursion, the others on the tree model. They all require considerable resources in terms of function calls stack. The first approach is based upon recursion and looks as the most onerous, as it triggers as many calls as the dictionary size, i.e. N_T or N_L; whereas the other two approaches look rather complicate and expensive, because dropping the dictionary would need to constantly track down the paths across the tree for trasversing them back and forth. In whatever version, the lexicographic approach does not care of saving resources and, in long terms, the running speed eventually slows down. A candidate different approach would aim at being lighter and running faster, and then at dropping storages in any form and devising alternative ways to the tree model and trasversion.

4 Index generation: the numerical alternative

In order to get away from the lexicographic environment, we have to step back to the primary level of composition, relying upon the abstraction of symbols, as discussed in §2 (fig. 7). We recall that letters are just meant as a choice for opening to rendering operations.

A different materialization of abstract symbols consists in developing an approach that involves digits and numbers. We start from reviewing the insertion of tree nodes in fig. 9 under this new perspective. According to the *Basis Representation Theorem* [1, pp. 8–9], any numerical quantity Q can be written as a unique string q_b in base $b \geq 2$. Q acts like an abstract concept that allows travelling through arbitrary representations. Q is invariant under base conversion, only the formal appearance change; thus the increasing (or decreasing) trend of sequences of numbers in base 10, say 1_{10}, 2_{10}, 3_{10}, ..., will be kept up the same trend in another base. Given q_2 and q_{10}, then $q_{10} \rightarrow Q \rightarrow q_2$: for every integer in base 10, there exists one and only one conversion into a different base. Base conversion stands as *unambiguous and reliable* approach to the formalization of orbits. As we already remarked at p. 70, invariance represents a nice start for developing mathematical ideas.

0_{10}	1_{10}	2_{10}	3_{10}	4_{10}	5_{10}	6_{10}	7_{10}	8_{10}	9_{10}	10_{10}	11_{10}	12_{10}	13_{10}	14_{10}	15_{10}
0_4	1_4	2_4	3_4	10_4	11_4	12_4	13_4	20_4	21_4	22_4	23_4	30_4	31_4	32_4	33_4

Table 3: **Conversion from base 10 to 4.**

We notice that every base n is equipped with a set of exactly n distinct digits, which we could, even if improperly, call as *alphabet* again, because of performing homologous tasks.

We follow the transition between the lexicographic and the indexed approach in table 4, by comparing the formal compositions. At this initial stage, the difference just regards the adopted symbols. Every new number in base

Generator	g_1	g_2	g_1^{-1}	g_2^{-1}				
Symbol	a	b	A	B				
Array index	0	1	2	3				
Cancellations in letters	aA	bB	Aa	Bb	aa	bb	AA	BB
Cancellations in digits	02	13	20	31	00	11	22	33
	non-self-inverse generators				*self-inverse* generators			

Table 4: **Environmental arrays for 4-generators groups.**

4, for example, shows up as the concatenation of digits, taken from the set [0, 1, 2, 3] (table 3). In every 4-generators subgroup, the letters a, b, c, d are respectively associated to the digits of the 4-base number system: 0, 1, 2, 3. The formal expressions of numerical quantities under some given value are permutations of digits. Let the value 10000, then all smaller numbers from 0000 to 9999 are permutations of all digits in the 10-base system. Given an alphabet of cardinality $n = 4$, the concatenation of letters up to depth d is equivalent to writing all numbers in base n up to the value n^d. *The generation of all words in the lexicographic approach can be completely replaced by the increasing sequence of positive integers in some arbitrary base system.* We no longer need to build up dictionaries and store massive bulks of data: the n-base representation can return the same information as from the combinations built on purpose through the step-by-step concatenation of letters. Numbers devolve into sequences of digits, i.e. of strings being managed through the symbols concatenation; they however keep all we need to render the subgroup action.

Zero management. This new approach runs faster, but it shall be handled with care as we are *switching from quantities represented by numbers to*

Level 0	I				Level 0	I			
Level 1	A	B	a	b	Level 1	a	b	c	d
	0_4	1_4	2_4	3_4		$0_4 = 0_{10}$	$1_4 = 1_{10}$	$2_4 = 2_{10}$	$3_4 = 3_{10}$
Level 2	AA	AB	Ab		Level 2	ab	ac	ad	
	00_4	01_4	03_4			$01_4 = 1_{10}$	$02_4 = 2_{10}$	$03_4 = 3_{10}$	
	BB	BA	Ba			ba	bc	bd	
	11_4	10_4	12_4			$10_4 = 4_{10}$	$12_4 = 6_{10}$	$13_4 = 7_{10}$	
	aa	aB	ab			ca	cb	cd	
	22_4	21_4	23_4			$20_4 = 8_{10}$	$21_4 = 9_{10}$	$23_4 = 11_{10}$	
	bb	bA	ba			da	db	dc	
	33_4	30_4	32_4			$30_4 = 12_{10}$	$31_4 = 13_{10}$	$32_4 = 14_{10}$	
	self-inverse generators					non-self-inverse generators			

Table 5: **Applications to subgroups**. Some entries have been skipped because of cancellation rules. Index generation covers all combinations yielded by the lexicographic approach (fig. 9).

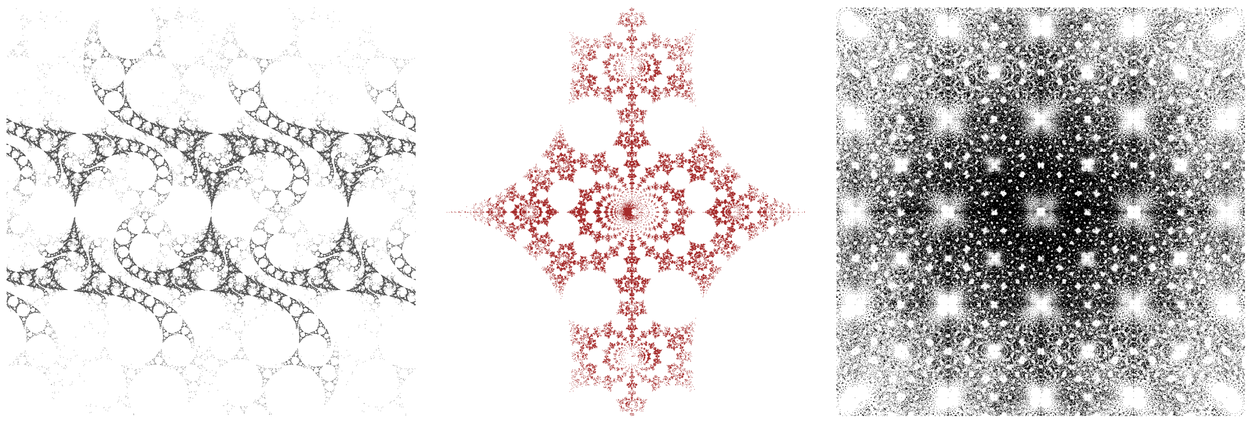


Figure 12: These limit sets have been rendered via the *index generation* algorithm. The dots at the outer sides of the picture at the far left prove that accurate renderings would be very long time consuming.

the quality (i.e., the visual appearance) *of digits, i.e. of symbols deprived from their native quantitative meaning.* This transition goes from positional numerical systems to strings of concatenated symbols and it requires to settle the ambiguous role played by the digit 0 (table 4), which is applied to encode the action of the Möbius map stored in the *array* at the index 0 (arrays are data structured endowed with zero-based indexing). The digit 0, unlike all others from 1 to 9, could relate to quantification or not, depending on the position within the string of digits: it plays the multiplicative role if *ap*pended to the far right (ex: 10000), or none if *pre*pended to the far left (ex: 00001); we mean to *trailing* and of *leading zeros* respectively. For instance, the unit value 1 is encoded into the word '1'; assuming 5 as maximal length, we get the words '1', '01', '001', '0001', '00001'. In general, there exist infinitely many formal words '$0n$', '$00n$', ..., '$\overline{0}n$', being padded by leading zeros and which encode the same integer quantity n; they feature a same sub-orbit of iterates of the Möbius map associated to the index 0. In the formal context of symbols concatenation, the digit 0 drops the role played in the positional representation of numbers; being no longer a numerical value but just as a symbol, it is an index that refers to a generator for the given subgroup; this is another reason why *leading zeros are as important as trailing ones here.* We see that the zeros left padding does not occur for groups of self-inversions for instance, where the composition of words does not allow contiguous symbols repetition.

The transformation from numerical values to strings of symbols is one-to-many (= multi-valued); conversely, strings with leading zeros would be encoded back to the same numerical value in the many-to-one fashion (= single-valued). Because of the aforementioned reasons, base conversion cannot cover strings with leading zeros; hence we have to implement a separate procedure for managing these special strings. Resuming, the implementation of the index generation algorithm consists in

(1°) *taking on a number* in base 10, say 93_{10};

(2°) *encoding it* into the new base, say 1131_4;

(3°) *left padding every string* yielded in the step 2 through a sequence of leading zeros up to a given finite maximal length. Suppose the latter is 8, the 4-base number obtained above would increasingly left padded in order to obtain the four strings $(8 - 4 = 4)$: 01131_4, 001131_4, 0001131_4, 00001131_4;

(4) *feeding* the resulting string in the new base to *the rendering engine*.

All process boils down to converting numbers into the new base and processing the obtained strings. We no longer need to store them. *The index generation algorithm* post-processes *words: the base conversion yield a new words which is checked whether there are subsets triggering cancellations.*

5 The pseudo-code implementation

We will illustrate the code implementation guidelines for the index generation algorithm, either for rendering disc images or limit sets. Code has been split into blocks which isolate the services into distinct working compartments; this way could be also of help to easily follow how each service is implemented and what its task is. Code is presented in the form of pseudo object-oriented language in order to ease the customization into readers' favorite environment. Generators have been embedded into class objects, endowed with *members*:

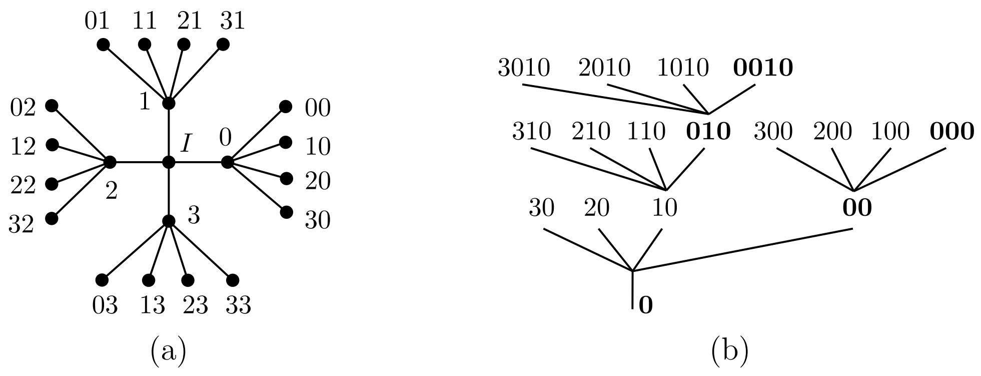


Figure 13: (a) The tree in fig. 9/b was rebuilt under the index generation principles. Words are not filtered by prior multiplication table tests; (b) A branch was picked up from the tree in fig. 13 and walked through two nodes deeper.

namely, they consist of *methods* and of *data containers*, referred by the membership notation to the object: `obj.⟨method_id⟩(parameters)` for methods and for `obj.⟨variable_id⟩` for containers (i.e., for variables) respectively.

We begin from instantiating the generators listed in table 4 at p. 80: for sake of simplicity and with no loss of generalization, we assume to work with 2-generators groups ruled by the simplest presentation (5) or (6). Our algorithm is *scalable* and not affected by the cardinality of the generators set.

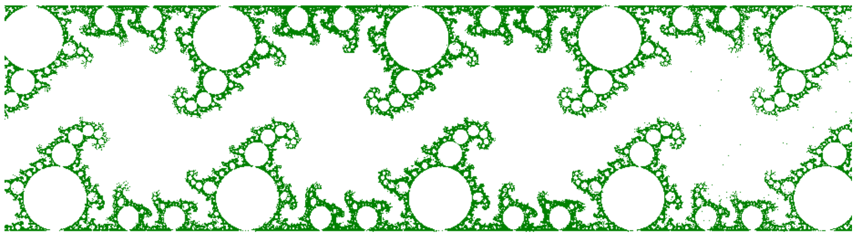
The index–generator association is automatically set up by the instantiation of the logical array (table 4 at p. 80). A number of environmental variables and containers, related to the sub-services run by the algorithm, are first initialized for further needs in the code box below.

```
var _gens_objs = [ g1, g2, g3, g4 ], _gens_num = _gens_objs.length;
var _max_depth = 10, _max_value = power( _gens_num, _max_depth );
var _proc_str = "", _zero_fill_proc_str = "", _index, _circle;
var _str_length = 1, _rec_start = 0, _rec_end = -1;
var _b_length_change = 0, _b_crash_found = 0;
```

We stress again that the numerical nature of this algorithm disengages from the concept of word depth, which is more naturally tied to the transversion of the lexicographic approach. We will deal with the number of steps instead, and so we have to set up an arbitrary maximal value that stops the main loop. We opted to use the `for`-loop syntax because of being conceptually close to the increasing sequences of integers discussed in the previous section; but there are no impediments against the `while`-loop syntax which, as known, generalizes and includes the principles of `for`-loops. For sake of coherence with the initial remarks at §1, we will work with inversion circles or with pixels/points, for rendering disc images or limit sets respectively.

```
for( var _i = 0; _i < _max_value; _i++ ){
    _proc_str = _i.<convert-to-base>( _gens_num );

    //we chose this test to trigger the leading zeros management below
    _b_length_change = _str_length != _proc_str.length;

    //the test below may involve the multiplication table
    //or the subgroup presentation
    if ( <call-to-sub-routine-#1.x:cancellation-rule-test_of_proc_str> ) continue;

    <call-to-sub-routine-#2.x:process-the-numerical-string-in-base-n>
    <call-to-sub-routine-#3:leading-zeros-management>
}
```

We are going to render limit sets first: a string of symbols is returned for each value of the loop counter `_i` and processed by composition of Möbius maps.



Input points. Unlike the rasterized rendering of Julia sets, we do not need to check every point/pixel inside the region of interest (the *escape time* method). According to the theory of *function groups* (which both fuchsian and kleinian kinds belong to), it is sufficient to give an arbitrary input value: *the definition of a limit point depends only on the sequence of elements of the subgroup $\mathcal{G}$, and not on the points belonging to the region U where the action of $\mathcal{G}$ is freely discontinuous* [13, p. 22, D.3]. Moreover, since *the limit set is transformed into itself by any transformation of the group* [7, p. 43], we found worth picking up input values from the fixed points belonging to one of the generators of the subgroup.

The generic label `#2.x` refers to the code block `#2.1` which renders the *limit set*. After processing the input word from right to left, we will display on the screen the last element of every orbit exclusively, as required by the definition of limit sets.

```
//<sub-routine-#2.1:limit set mode>
//right-to-left reading order
_index = <turn-the-symbol-to-integer>( _proc_str[ _proc_str.length-1 ] );
//initialization
_fp = _gens_objs[ _index ].get_one_fixed_point();

//process the rest of the string
for( var _wr = _proc_str.length-2; _wr >= 0; _wr-- ){
   _index = <turn-the-symbol-to-integer>( _proc_str[ _wr ] );
   _fp = _gens_objs[ _index ].map_point( _fp );
}

<call-a-sub-routine-for-drawing-the-pixel-at-the-fixed-point-coordinates>
```

Disc images based renderings need the subroutine `#2.x` to be replaced by the block `#2.2`, where every new inversion circle is plotted on the screen as each new symbol in the word is read, from right to left, and processed:

```
//<sub-routine-#2.2:disc images mode>
//right-to-left reading order
_index = <turn-the-symbol-to-integer>( _proc_str[ _proc_str.length-1 ] );
```

```
//initialization
_circle = _gens_objs[ _index ].get_inversion_circle();

<call-a-sub-routine-for-drawing-the-circle>
//process the rest of the string
for( var _wr = _proc_str.length-2; _wr >= 0; _wr-- ){
    _index = <turn-the-first-symbol-to-integer>( _proc_str[ _wr ] );
    _circle = _gens_objs[ _index ].map_inversion_circle( _circle );
    <call-a-sub-routine-for-drawing-the-circle>
}
```

The label `<sub-routine-#3:leading-zeros-management>` refers to the next pseudo-code implementing both elaboration and rendering of strings with leading zeros, as discussed in the previous section. The strategy boils down to generating such strings from the ones returned inside the main `for`-loop. We chose to run this code when the converted string changes in length, e.g. from 333_4 to 1000_4 in base 4. According to the remarks at the end of §4, this code block mimics the tasks of the main algorithm.

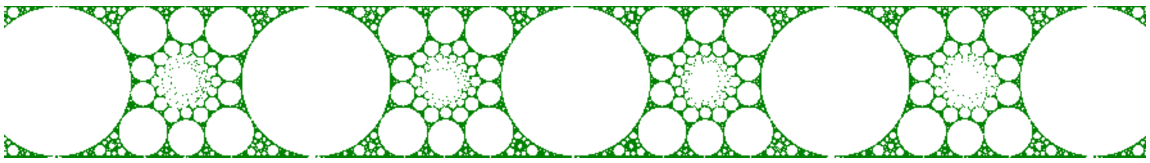
```
//<sub-routine-#3:leading-zeros-management>
if ( _b_length_change )
{
  _rec_end = _n - 1;
  for( var _r = _rec_start; _r <= _rec_end; _r++ )
  {
     _zero_fill_proc_str = _r.toString( _n_gens );
    for( var _filler = _zero_fill_proc_str.length; _filler <= _max_depth; _filler++ )
    {
         _zero_fill_proc_str = "0" + _zero_fill_proc_str;
         if ( <call-to-sub-routine-#1.x:cancellation-rule-test_of_proc_str> )
             continue;
         <call-to-sub-routine-#2.x:process-the-numerical-string-in-base-n>
    }
  }

  _rec_start = _rec_end + 1;
  _rec_end = -1;
  _b_length_change = 0;
}
```

We remark that the pseudo-code

```
if ( <call-to-sub-routine-#1.x:cancellation-rule-test_of_proc_str> ) continue;
```

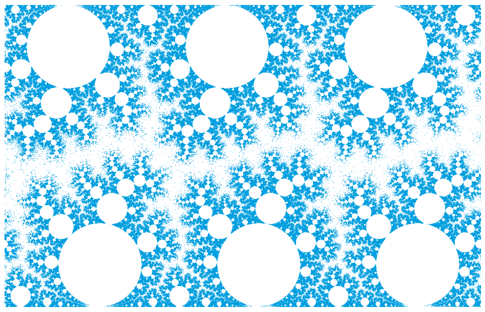
refers to tests to be performed according to the Cayley table or presentation related to the given subgroup. Two examples on how tables 1 at p. 76 are coded follow below. The equivalent cancellation rules are (5) and (6) in terms of presentations. Implementation is easy: instead of trasversing table rows and columns, these cancellation tests check whether every new input string of digits includes at least one of the rules on the right of the presentation.

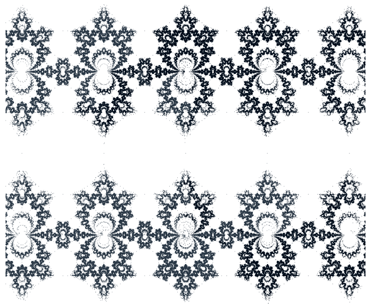
```
//<sub-routine-#1.1:group-presentation>
function __check__group_presentation__( _digitized_word = "" )
{
<let a boolean flag and set it to 0>

<for each entry inside the group presentation>
    <check if the input digitized word includes the current entry>
    <if so, set the above flag to 1 and break this loop>
<end-of-for-loop>

<return the boolean flag>
}
```



And now the pseudo-code for cancellation tests with regard to the multiplication table.

```
//<sub-routine-#1.2:multiplication-table-test>
function __multiplication-table-test__( _digitized_word = "" ) {
<let a boolean flag and set it to 0>

//we assume that the index has been converted into the required base
<split-the-digitized-word-into-an-array-of-single-digits->
<get-the-first-digit-in-the-word>
<get a reference pointer to the related row inside the table>

//we prevent to raise conditional if-statement in the loop
<remove the first digit from the word>

<for each digit in the of rest this array> //sequential read
  <get the next-index in the current row at the index espressed by the digit>
    <if the next-index is invalid, then (1) set the above flag to 1>
    <(2) break the loop and (3) skip this word processing>
  <get a reference pointer to the row inside the-table and related to the next-index>
<end-of-for-loop>

<return the flag>
}
```





6 Comparisons and benefits

The *digit*al nature of the index generation algorithm could take away some of the charm tied to the theory of word processing performed by the lexicographic approach (refer to the algebraic theory of commutators at [15, p. 168]) and emanating from the related literature ([6] über alles). Anyway, for practical purposes, the numerical base conversion enjoys the benefit of generating every chain of generators in just one step and so it runs much faster than the lexicographic approach. *The transformational character of the index generation algorithm relies upon the simpler and quicker generation of words, no longer coming from a constructive progression, like in the lexicographic approach* which sets up a tortuous track disseminated by the technical drawbacks already discussed in §3, i.e., appending symbols, checking words and storing them into (possibly, huge) containers, the dictionaries. The only (and necessary) computational costs of the index generation algorithm are the essential multiplication table and the correlated testings.

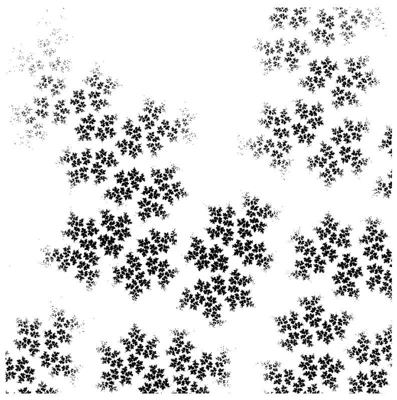


Figure 14: This is a close-up of the limit set for a degenerate subgroup of 2-generators, whose coefficients satisfy special and delicate numerical conditions.

One more drawback of lexicographic approach concerns the sequential building of words of n symbols, which are deduced from those of length $n-1$. At this regard, we observe that the intrinsic tree structure involves nodes dependency, which demands to start from the root and *walk through* the consecutive nodes in order to get to the given depth. On the contrary, the index generation algorithm enjoys the benefits of numerical sequences which is based upon, allowing to *start*, *stop* and *resume* the generation of strings at any arbitrary element of the sequence. If needed, we can *jump* from end to end here, instead of *walking through* the interval. It is known that $d = \left\lfloor \dfrac{\log(i)}{\log(n)} \right\rfloor$ returns the number d of digits required to convert i from base 10 to base n; thus, we can explore limit sets inside some interval of integer values, which match with words of length l, given $d \leq l \leq D$ for instance.

The author has developed a web application that implements both the lexicographic and the index generation algorithm at `http://alessandrorosa.altervista.org/circles/`; a number of demos can be run for introductory purposes, or groups be built either geometrically through inversion circles or algebraically via input of arbitrary coefficients into Möbius maps. Refer to [17] for related examples.

References

[1] Andrews G.E., *Number Theory*, Saunders, 1971.

[2] Beardon A., *The Geometry of Discrete Groups*, Springer, 1983.

[3] Bessis D., Demka S. *Generalized Apollonian packings*, Commun. Math. Phys., 134, 1990, pp. 293–319.

[4] Bullets S., Mantica G., *Group theory of hyperbolic circle packings*, Nonlinearity, 5, 1992, pp. 1085–1109.

[5] Devaney R. L., Marotta S. M., *Mandelpinski necklaces in the parameter plane of rational maps*, Springer Proceedings in Mathematics and Statistics, 2021, pp. 95-119.

[6] Epstein D.B.A. et alia, *Word processing in Groups*, Jones and Bartlett Publishers, Boston, 1992.

[7] Ford L., *Automorphic functions*, McGraw-Hill, New York, 1929.

[8] Fricke R., Klein F., *Vorlesungen über die Theorie der automorphen Functionen*, Teubner, Leipzig, 1897.

[9] Krushkal S.L., Apanasov B.N., Gusevskiĭ N. A., *Kleinian Groups and Uniformization in Examples and Problems*, Translations of Mathematical Monographs, AMS, 1986.

[10] Lyndon R.C., Schupp P.E., *Combinatorial Group Theory*, Springer, 2001.

[11] Magnus W., *Non-Euclidean Tesselations and their Groups*, Elsevier, 1974.

[12] Manna S.S., Vicsek T., *Multifractality of Space-Filling Bearings and Apollonian Packings*, Journal of Statistical Physics, 64, 3/4, 1991.

[13] Maskit B., *Kleinian Groups*, Springer, 1988.

[14] McShane G., Parker J.R., Redfern I., *Drawing limit sets of Kleinian groups using finite state automata*, Experimental Mathematics, vol. 3, 2 (1994), pp. 153–170.

[15] Mumford D., Series C., Wright D., *Indra's pearls: The Vision of Felix Klein*, Cambridge University Press, 2002 (reprinted in 2015).

[16] Parker J.R., *Kleinian circle packings*, Topology, vol. 34, No. 3, 1995, pp. 489–496.

[17] Rosa A., *The pearls of Heavens: A gallery of Kleinian Groups*, 2023, `https://www.academia.edu/95460195/`