A fast algorithm for limit sets of Kleinian groups with the Maskit parametrisation.

Jos Leys

January 2017

1. Introduction

We consider Kleinian group limit sets with the Maskit parametrisation. The limit set is defined by two Moebius transformations: $a:\ z \to \frac{tz-i}{-iz}$ and $b: z \to z+2$, where $t = u + iv$ is a complex number. The inverse transformations are $A:\ z \to \frac{i}{iz+t}$ and $B: z \to z-2$.

This produces limit sets like the one in figure 1 for $u = 2, v = 0$:

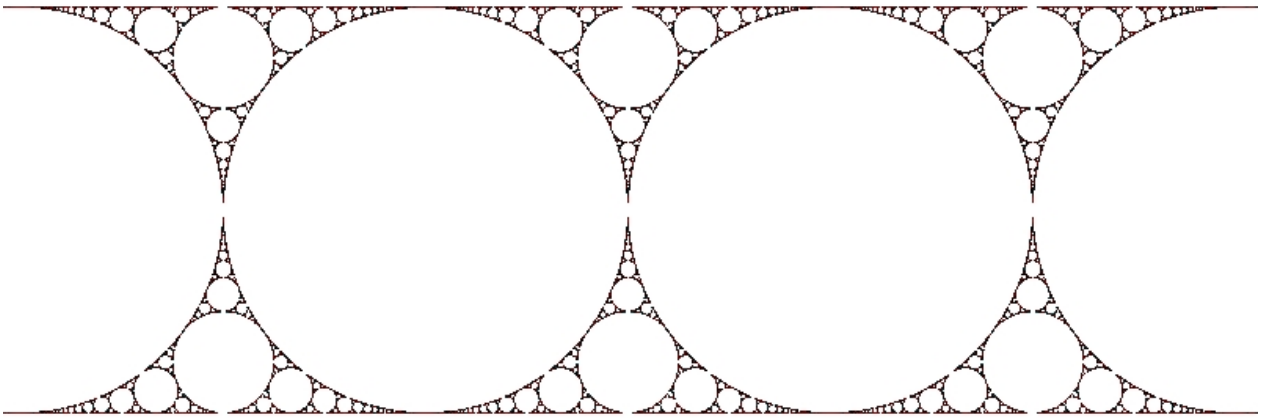

Figure 1

For $u < 2$ and $v \neq 0$, for instance $u = 1.95,\ v = 0.07$, the limit set is shown in figure 2 :

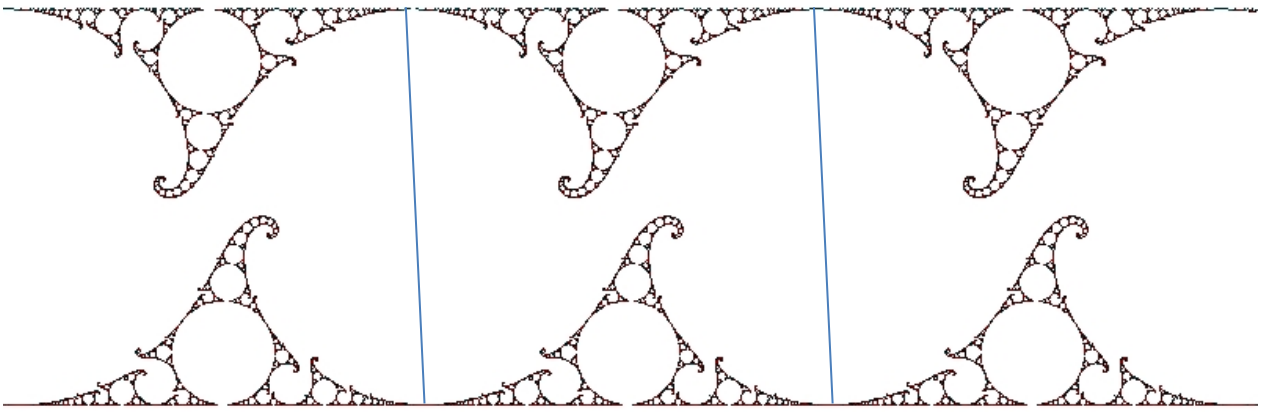

Figure 2

Inside the parallelogram drawn in figure 2, the shapes are symmetrical to the point $z = -\frac{v}{2} + iu/2$. The action of b and B produces endless identical copies to the left and to the right.

The largest circles, tangent to the lines $y = 0$ and $y = u$ are the images of said lines by a circle inversion of radius 1, centered at the origin, and at the point $-v + iu$ respectively. When any circle that belongs to the limit set is transformed by a Moebius transformation formed by any 'word' in the letters a, A, b and B , the resulting circle will also be part of the limit set. This allows generating images like the ones in figures 3 and 4.

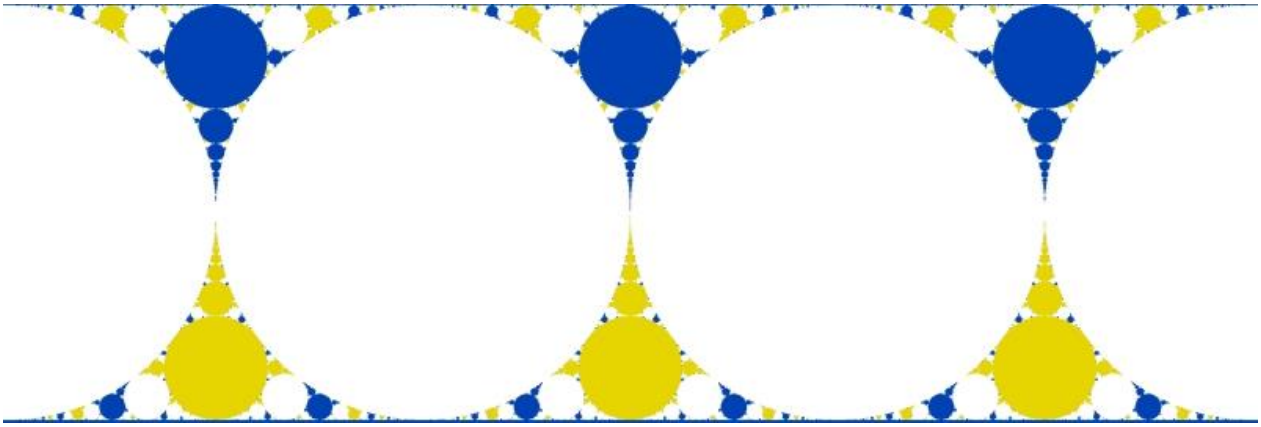

Figure 3

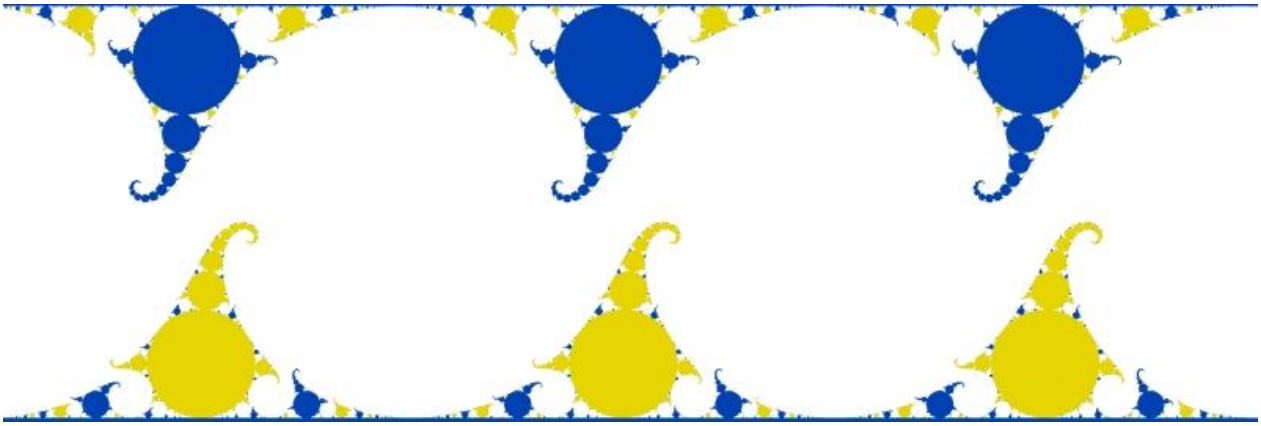

Figure 4

Generating these images can be done with the methods described in [1]. It involves forming a structured set of 'words' with the transformations $a, A, b \; and \; B$. These methods do not allow deciding upfront whether a point in the plane is part of the limit set or not. Detailed images require prolonged calculation times: zooming in on details of an image requires longer 'words', and it is difficult to generate just the part of the limit set that is in the zoom window, resulting in long calculations, in the order of dozens of minutes, sometimes hours.

The method described below does determine if a point is part of the limit set, and is therefore extremely fast, allowing such things as zooming in on the limit set in real time. An image typically takes just a few seconds to complete.

2. The fast algorithm

We draw three lines (figure 5): line 1 between the points $1 + i0$ and $1 - v + iu$, line 2 between the points $-1 + i0$ and $-1 - v + iu$, and finally line 3 separating the upper and lower parts of the limit set . We introduce the following rules:

- if a point is on the right of line 1 we move the point so that it falls between line 1 and line 2 . If the point is to the left of line 2 we do the same. This move on a point $z = x + iy$ can be described by : $x \to \left(x + 2N - 1 + \frac{vy}{u}\right) modulo(2) - 1 - vy/u$, where N is some (large) positive integer.
- if a point is below line 3 we apply transformation a, if not we take transformation A.

Suppose a point is in a one of the yellow circles in figure 5. The above rules will move a point so that it always jumps to a spot that is also in a yellow circle and will eventually make the point end up in the largest circle that is tangent to the line $y = 0$. This circle is actually the line $y = u$, after a circle inversion in a circle of radius 1, centered at the origin. When transform a is again applied to a point in this circle, then the new point will be outside the strip $0 \leq y \leq u$. Figure 5 shows the orbit of such a point.

[1] *Indra's Pearls, the vision of Felix Klein* by David Mumford, Caroline Series and David Wright. Cambridge University Press

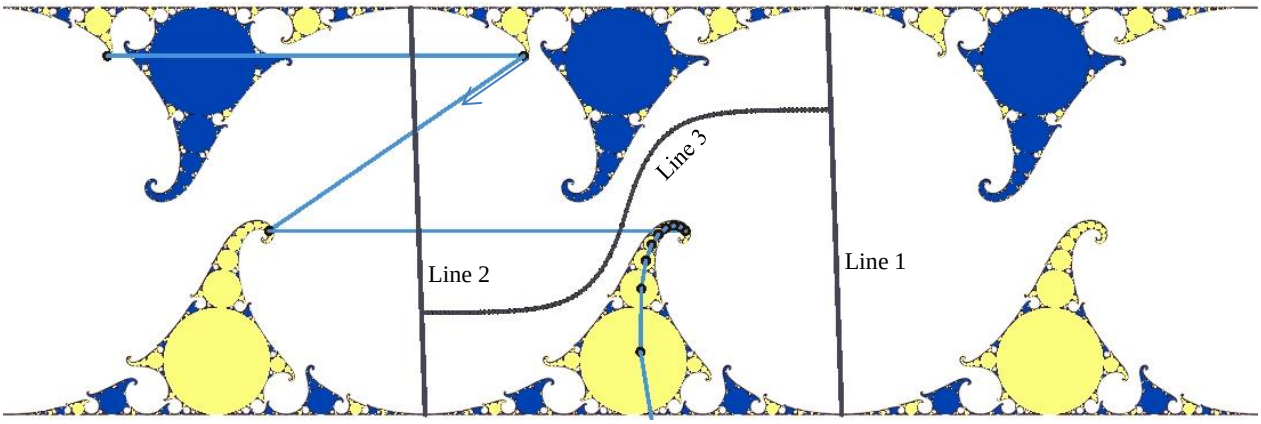

Figure 5

Under the same rules, a point in a blue circle will be carried to the largest upper blue circle that is tangent to the line $y = u$, and this circle is actually the line $y = 0$, transformed by a circle inversion in a circle of radius 1 centered at $-v + iu$. The next application of the transform A will make this point jump above the line $y = u$. See figure 6.

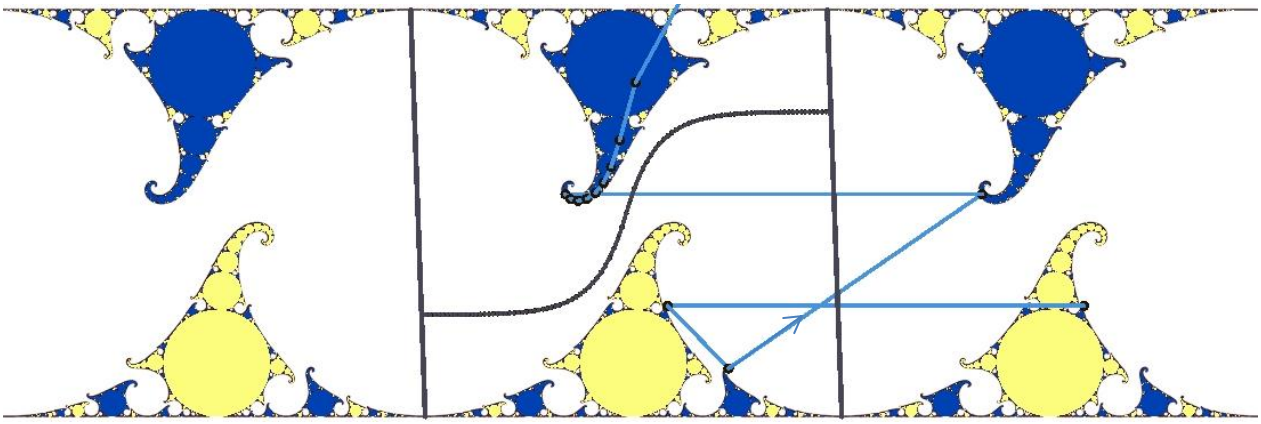

Figure 6

Finally, a point that is not in any yellow or blue circle will not escape from the strip $0 \leq y \leq u$. In fact, after a number of jumps the orbit settles down into a two-cycle : the point bounces back and forth between two positions. See figure 7.

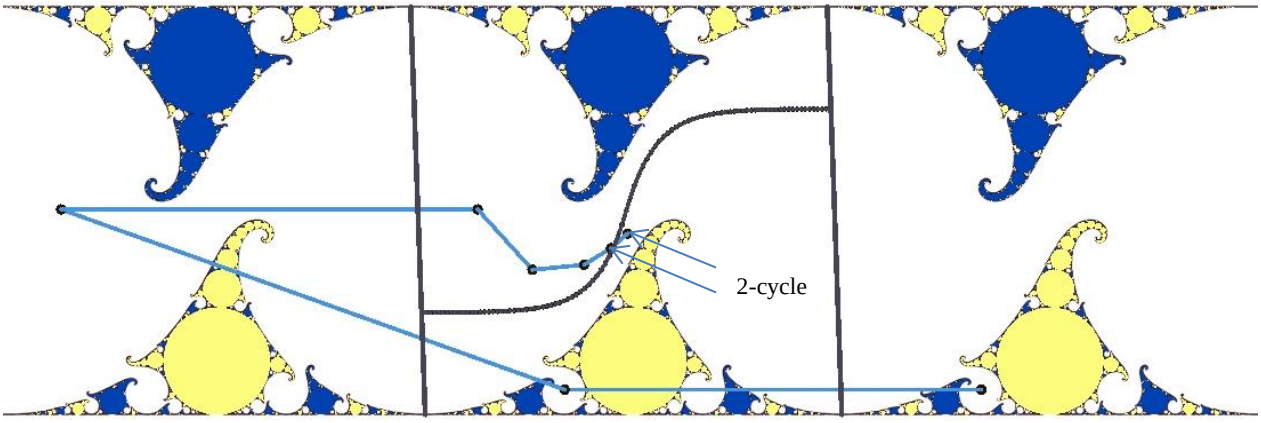

Figure 7

This then produces a simple, but very efficient algorithm :

For every pixel z :

- If $imag(z) < 0$ or $imag(z) > u$, do nothing, go to next pixel.
- Else, 'wrap' the point so it is between line 1 and line 2.
- If it is below line 3 apply a else apply A.
- If $imag(z) < 0$ or $mag(z) > u$, colour the pixel, stop, go to next pixel.
- If a two-cycle is detected, stop, go to next pixel.
- Repeat

This algorithm is extremely fast. The images in figures 5-7 (800*227 pixels) calculate in less than 1 second on a computer with Intel I7-4702MQ processor.

Line 3, the *separation line*, is quite important. As u gets smaller and v is small, the sets of blue circles and yellow circles get closer together, and a straight line does not fit between the two. In figure 8, $t = 1.92 + 0.03i$ and the line is an exponential formula like $y = \frac{u}{2} + fKu(1 - e^{-M.abs(x+\frac{v}{2})})$, where $f = 1$ if $x \geq -v/2$, and $f = -1$ otherwise, and K and M are suitable constants.

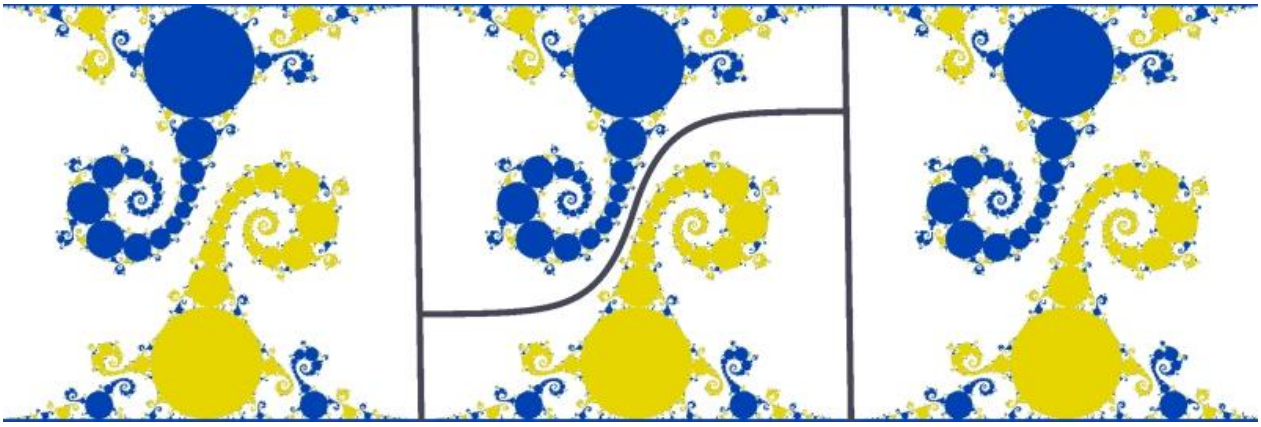

Figure 8

For most values of t, the exponential separation line will do fine. There are cases however where a more complicated separation line is necessary, as explained in the next section.

3. Cusps[2]

For specific values of t, the limit set will be *cusped*. This happens if a matrix of the form $a^n B^m$ is *parabolic*, meaning that its *trace* (the sum of the diagonal elements) is equal to two. Finding the value of t for given n and m is explained extensively in [3], along with the procedure for determining the exact multiplication order of $a^n B^m$. For instance, for $n = 15, m = 2$, this order (where multiplication is done from right to left) is :

aaaaaaaaBaaaaaaaB

This is called the $2/15$ *cusp*. The value of t works out at $t = 1.85738240046052..-i0.0762580599328681...$ The corresponding limits set is in figure 10. The complement of the limit set, the white region in the image, now also consists of tangent circles, and it seems that we need a separation line that passes through the tangency points of a string of these white circles.

[2] I am grateful to Geoffrey R. Smith of San José, California, USA for his help in this section.
[3] *Indra's Pearls, the vision of Felix Klein* by David Mumford, Caroline Series and David Wright . Cambridge University Press. Chapter 9 : "*Accidents will happen*"

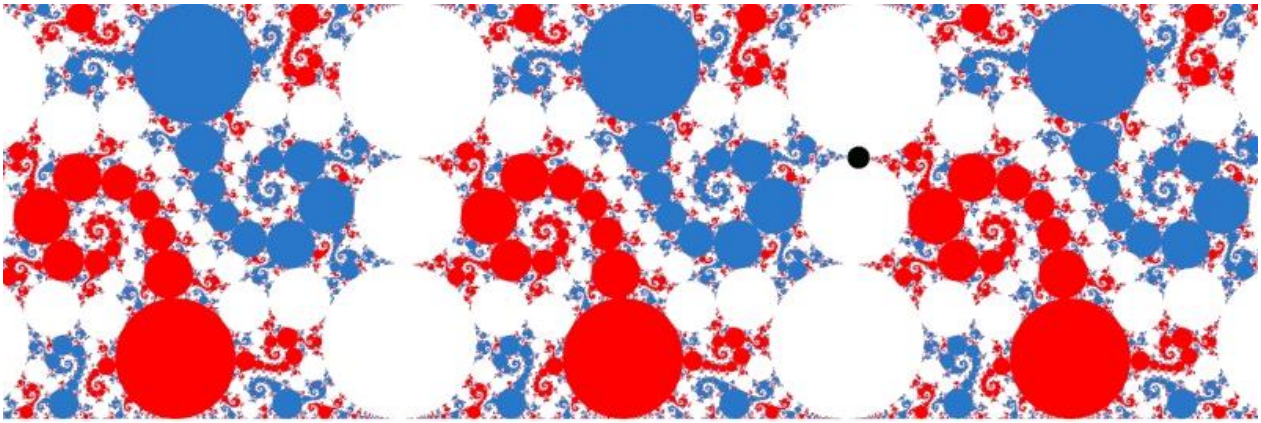

Figure 9

The black point in figure 9 is the fixed point of the matrix $a^{15}B^2$. As the trace equals 2, there is only 1 fixed point for this transformation. This point is on a tangency point of two white circles. We now take the $a^{15}B^2$ word, determine all its cyclic permutations, and calculate the fixed point of each resulting transformation. It turns out that all these points lie on tangency points of white circles as shown in figure 10. We have also added four orange points. These are the points $1 + i0, 1 - v + iu, -1 + i0, -1 - v + iu$.

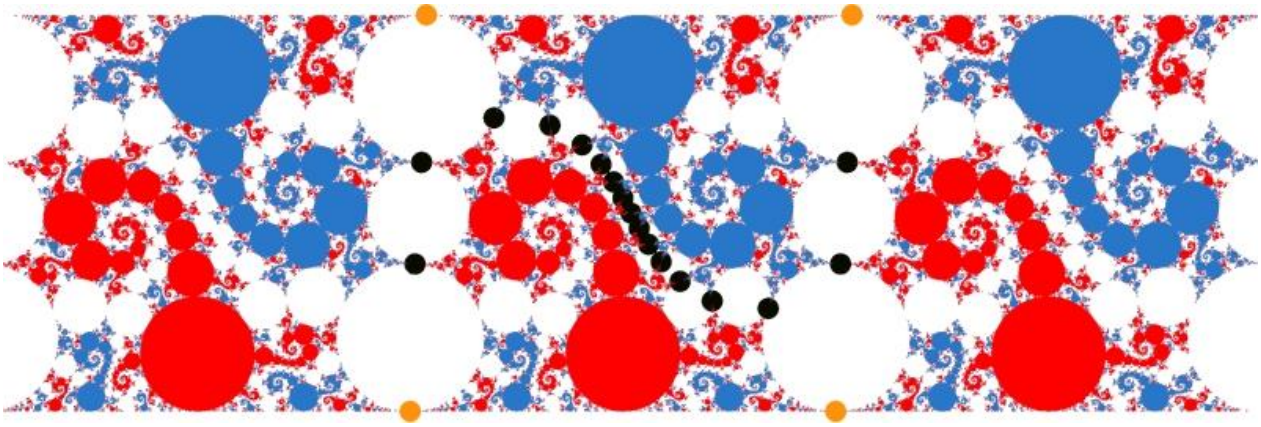

Figure 9

Connecting all these points should yield the correct separation line. However, in order to find adjacent points in the string, we first need to order the words obtained by the cyclic permutations. To determine the order, consider each word as a number, where a and B have each been replaced by some constant integer value, making sure that the value for a is smaller than the value for B. The words can then be ordered in increasing value of the resulting number.This done, we can connect the points, including the four orange ones, with straight line segments to produce the separation line, as shown in figure 11:

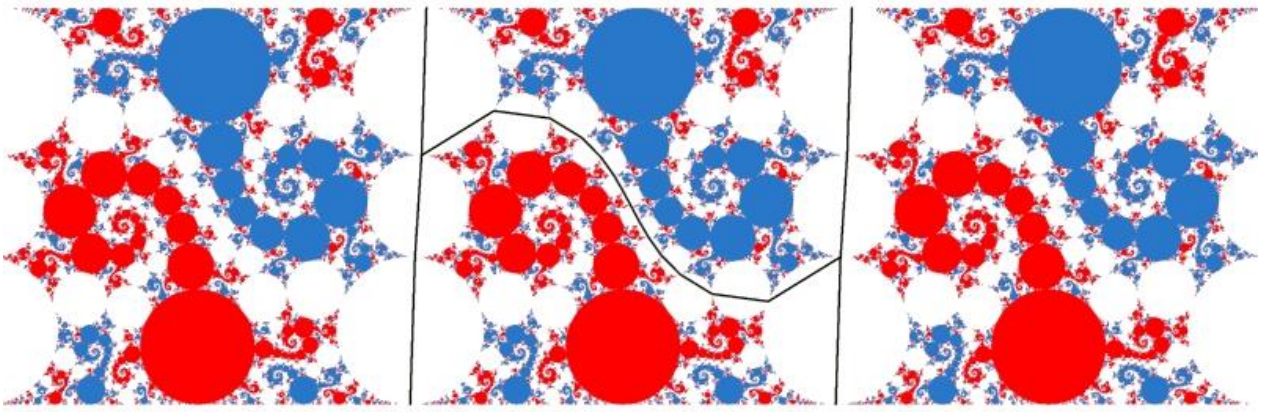

Figure 10

A separation line can become quite complicated. Figure 12 shows the $10/151$ cusp. There are 165 points to consider for the separation line.

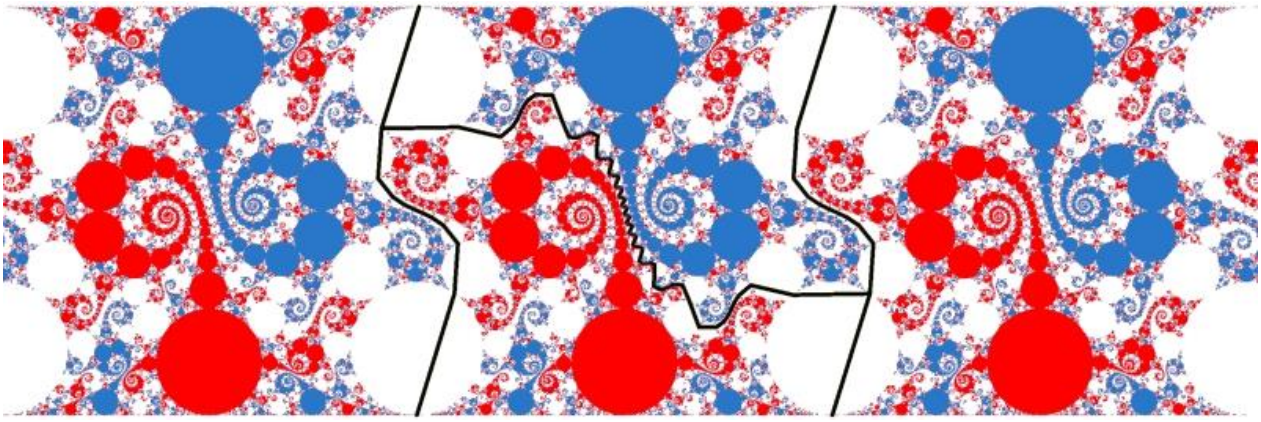

Figure 11

It is clear that the algorithm needs adaptation, as the rule to choose between transformation a or A depending on whether the position of the point being iterated is above or below the separation line, is no longer valid. We now need to check whether the point is inside or outside a polygon, the green polygon as in figure 13. (Efficient algorithms exist to do this. See for instance [4].)

Furthermore, the horizontal 'wrapping' of a point during the iteration now needs to be between the two 'S-shaped' lines in figure 13.

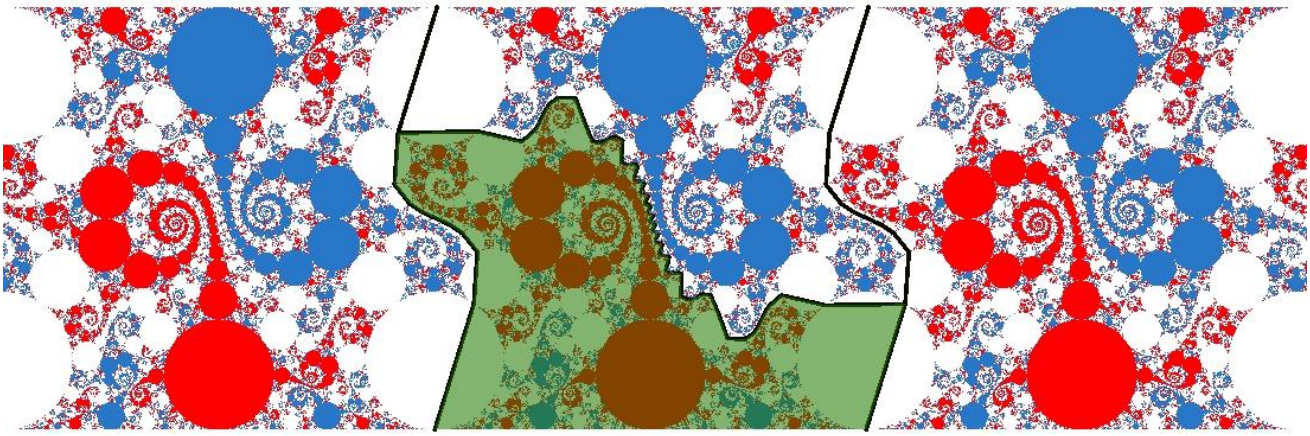

Figure 12

4. Variations.

4.1 The *b* transformation.

This transformation was defined as $b : z \to z + 2$. This can be changed to $b : z \to z + k$, where $k = 2\cos(\pi/n)$ with n an integer number. Figure 14 shows, top to bottom, the results for $n = 6\,, 5\,, 4$.

[4] Inclusion of a point in a polygon : http://geomalgorithms.com/a03-_inclusion.html

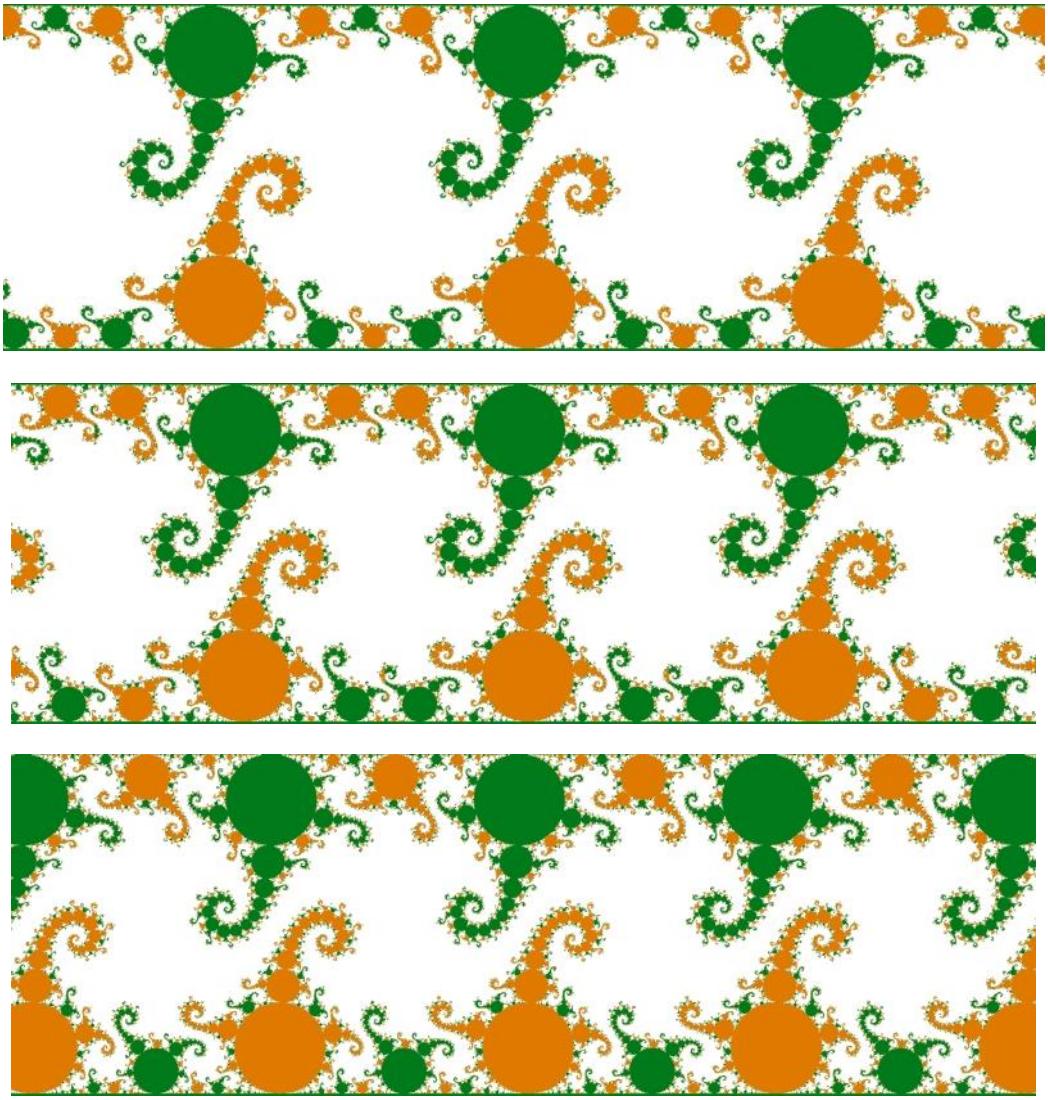

Figure 13

4.2 Circle inversions.

Here we do a pre-transform of a point before starting the algorithm. The pixel coordinate z is first transformed by a circle inversion in a circle of radius R, located at the point P:

Let $q = z - P$, $h = \tan^{-1}(q)$, then $z \to P + R^2 e^{ih}/|q|$.

Some examples:

The 1/15 cusp, $t = 1.95859103011179 - i0.0112785606117658$, with a circle inversion in a circle of radius, centered at $0 - i1$. (figure 15) Rendering time 3 seconds.

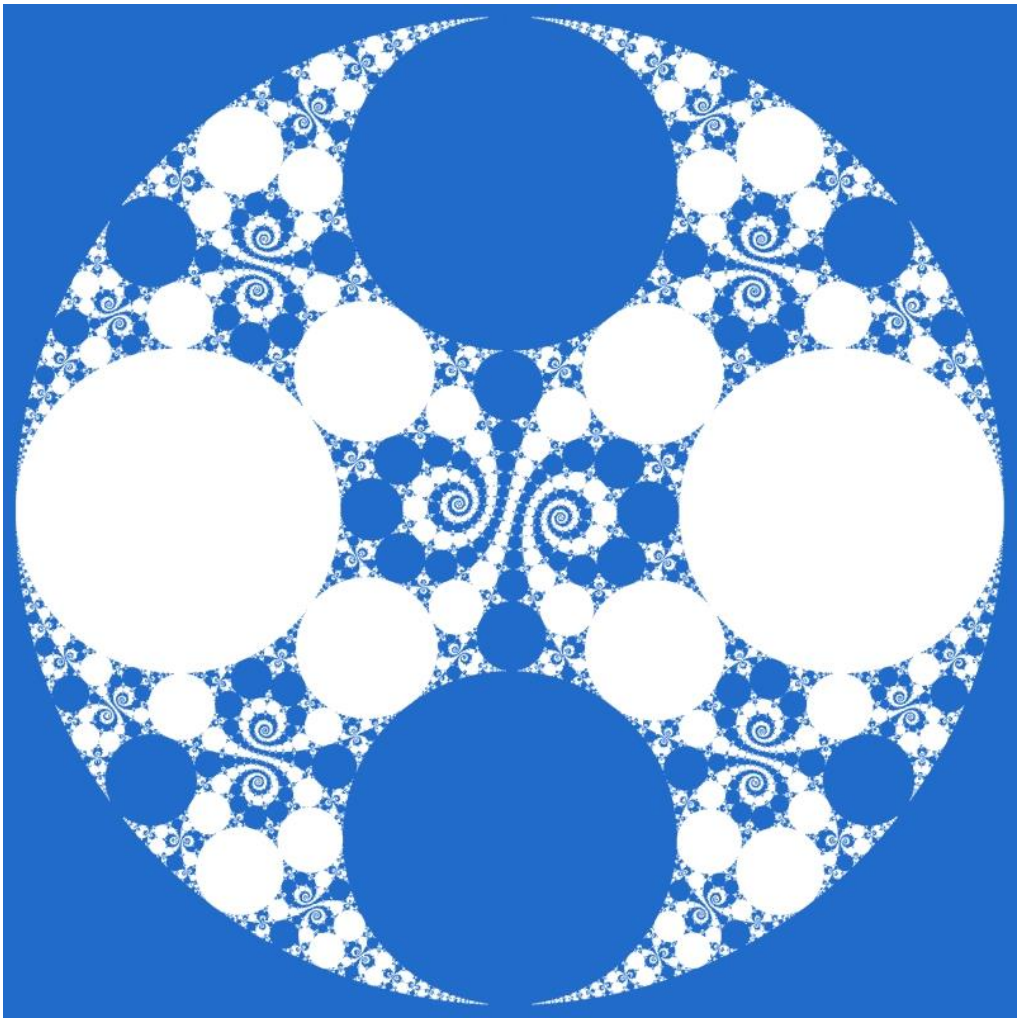

Figure 145

Figure 16 : $t = 1.93 + i0.04$, inversion in a circle of radius 1, located at $0.6 + i0.8$. Rendering time less than 1 second.

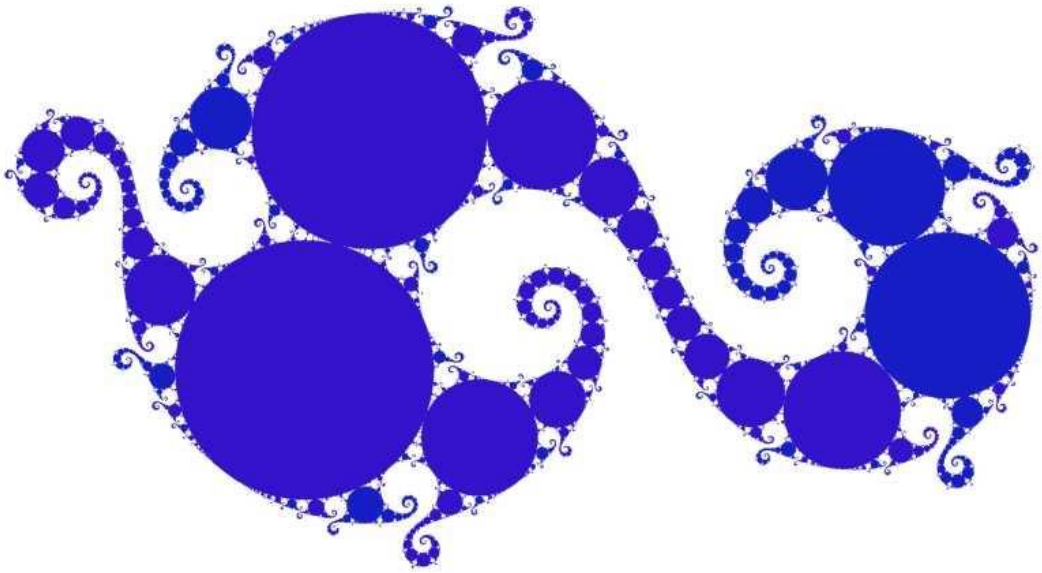

Figure 15

5. Conclusion

The algorithm described in this article provides a novel, very fast way of drawing limit sets of Kleinian groups with the Maskit parametrisation.

Remark : some of the principles used here can be applied to another algorithm that draws three-dimensional limit sets using raymarching techniques. This will be treated in another article.