# Numerical estimation of the area of the Mandelbrot set using quad tree tessellation and distance estimation

T. Förstemann*

(Dated: January 23, 2017)

An analytical expression of the area of the Mandelbrot set is unknown. There are different numerical approaches to estimate the area. A drawback of some approaches is that the numerical error of the area estimation is based on statistics and has limited theoretical significance. This approach is based on quad tree tessellations combined with distance estimation and provides precise lower and upper bounds of the area of Mandelbrot set. So far the most accurate numerical estimation of the precise lower and upper bounds by by Y. Fisher and J. Hill (2001) on 100 standard single core PCs is based on a quad tree tessellation of depth 19. In this approach a corresponding tessellation of depth 26 is reached using Mathematica and OpenCL, i.e. about $(2^{26-19})^2 = 16\,000$ times larger tessellation. The difference between the upper and lower bound can be halved.

## CONTENTS



* thorsten@foerstemann.name

## I. INTRODUCTION

### A. Problem Statement

An analytical expression of the area of the Mandelbrot[1] is unknown. There are many numerical estimations of the area, e.g. [2], based on different methods. At present the most precise estimations are based on Monte Carlo integration[3][4]. A main drawback of Monte Carlo integrations is that they do not provide precise upper and lower bounds of the Mandelbrot set area. Their numerical error is just based on statistics and has limited theoretical significance.

At the moment the most accurate estimation of precise upper and lower bounds by Y. Fisher and J. Hill in about 2001 [5] is based on 100 standard single core PCs. The aim of this approach is to narrow the precise upper and lower bound of this numerical area estimation.

### B. Solution Statement

This approach is based on quad tree tessellations combined with distance estimation and provides precise lower and upper bounds of the area of Mandelbrot set. The initial suggestion for this approach came from Prof. L. Bartholdi (Univerity of Göttingen) in 2015. This approach is based on the same method applied by Y. Fisher and J. Hill[5].

By now the approach by Y. Fisher and J. Hill is the most accurate numerical estimation of the precise lower and upper bounds and is based on a quad tree tessellation of depth 19. In this approach a tessellation of depth 26 is reached using Mathematica and OpenCL, i.e. about $(2^{26-19})^2 = 16\,000$ times larger tesselation.

In addition Y. Fisher and J. Hill supposes a factor 2 in the interior distance estimation method based on their numerical findings (refer to [5] section 5, there). Both cases, i.e. $\alpha = 1$ and $\alpha = 2$, are investigated and compared, here.

The GPU code runs in a Mathematica 10 environment on a standard PC with two dual core graphic cards.

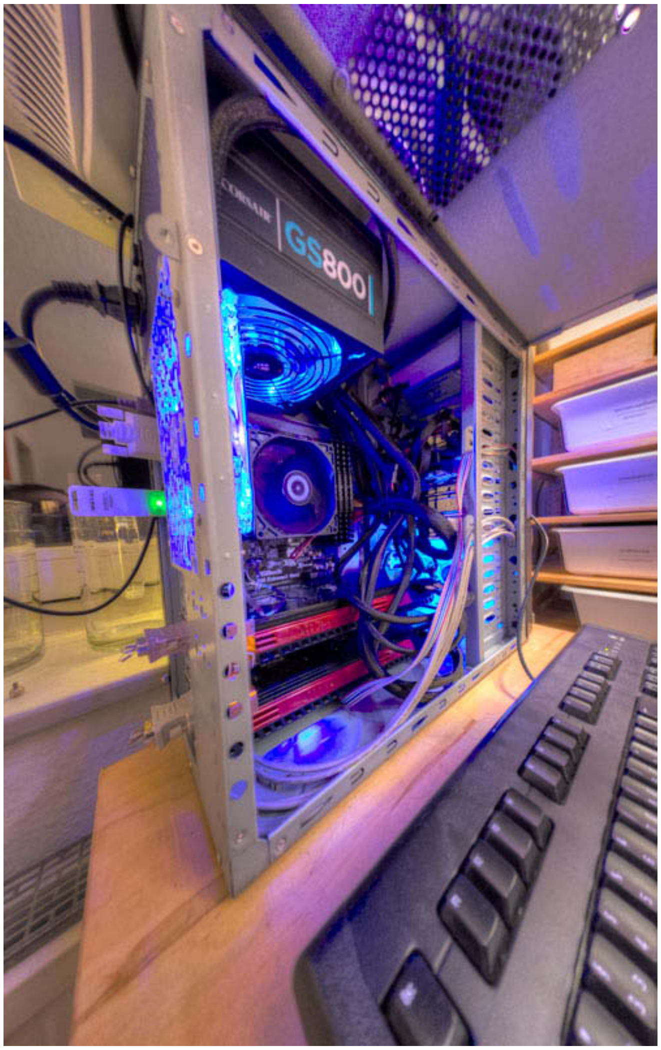


FIG. 1. Standard PC with two graphic cards.

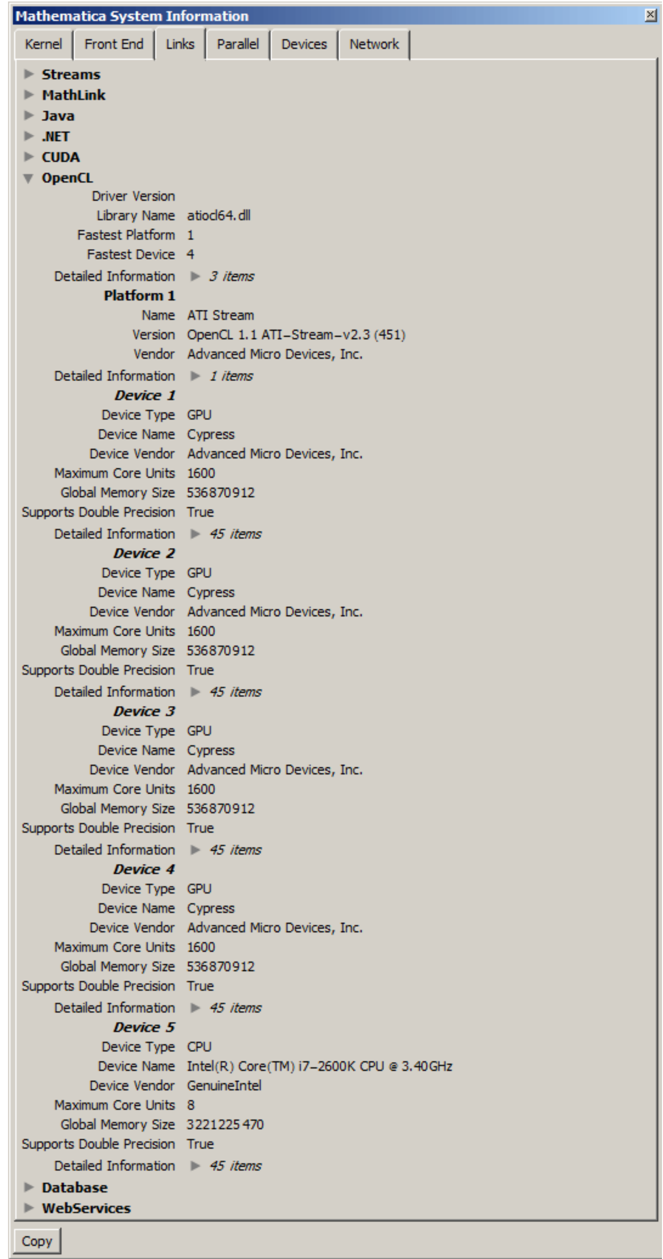


FIG. 2. Mathematica's OpenCL system information. Most important information: support for double precision.

## II. MATERIALS AND METHODS

### A. Hardware and Software

A typical PC (refer to figure 1) is used for the numerical calculations. Specific details of the hardware and software configuration are:

- CPU: Intel Core i7 2600K (about $500 full system)
- GPU: 2x Radeon HD 5970. Thus, there are 4 GPUs with 1600 stream processors (Cypress) each. Ebay prices: $100 each.
- Power consumption under load: approx. 350 watts. Thus, about 300 kWh per month, i.e. about $90 energy costs (Germany).
- Mathematica 10, Windows 7 (refer to figure 2)
- ATI driver Catalyst 11.2 with AMD Stream SDK 2.3 [7]

### B. Methods

In this section the applied method based on quad tree tessellation and distance estimation is illustrated. The implementation with OpenCL and Mathematica is detailed in the following sections.

Quad tree tessellations and distance estimations are quite established. The challenges of this approach are in particular performance considerations detailed in section II B 3.

### 1. Exterior and Interior Distance Estimation

The distance to the edge of the Mandelbrot set is needed in the quad tree tessellation, which is outlined in section II B 2.

There are two different algorithms to estimate the distance of a given point to the edge of the Mandelbrot set, e.g. [8]: The exterior distance estimator is suitable for points outside the Mandelbrot set and the interior distance estimator is suitable for points inside the Mandelbrot set.

The estimation of the exterior distance is based on iteration sequences generated by the standard escape time algorithm of the Mandelbrot set, e.g. [11]. For the interior distance estimation additional orbit detection is needed. Parameters of the implemented escape time algorithm with orbit detection are:

- The escape radius is set to 100 instead of 4 in order to gain a few additional iterations for a better convergence of the exterior distance estimation (refer to line 24 in section A 1). For a numerical evaluation of this inaccuracy refer to [5] appendix A, there.

- The maximum iteration is larger than $1024+20\,000 \cdot 2^9 = 10\,241\,024$. At tessellation depths 25 and 26 the limit is $1024 + 20\,000 \cdot 2^0 = 21\,024$. Specialities of depths 25 and 26 are discussed in section III D.

- Obits of maximum cycle length $20\,000 \cdot 2^8 = 5\,120\,000$ can be detected if the cycle starts at the latest after $1024 + 20\,000 \cdot 2^8 = 5\,121\,024$ iterations. At tessellation depths 25 and 26 obits of maximum cycle length $20\,000 \cdot 2^0 = 20\,000$ can be detected if the cycle starts at the latest after 1024 iterations.

- Cardioid and period-2 bulb testing are implemented (refer to e.g. [12]).

The interior and exterior distance estimator is part of the *decide* procedure, which is described in section II B 4. The OpenCL implementation of the *decide* procedure is outlined in section A 1. In order to implement these algorithms in OpenCL pseudo code resources are helpful, e.g. [9][10] for the exterior method and [10] for the interior method.

Due to Koebe quarter theorem[13] the factor 2 at line 229 in section A 1 should amount to 4 instead. In fact, [10] uses factor 4. On the other side Y. Fisher and J. Hill[5] suggests that factor 2 works quite well for the area estimation applied here. The factor 2, here, corresponds to the case $\alpha = 2$ in [5]. Accordingly, factor 4 corresponds to $\alpha = 1$ in [5]. In addition, using the actual implementation of the *MandelbrotDistance* function in Mathematica 10 yields the same results as this implementation with factor 2. More details are discussed in section III E.

Thus, we have the following two different interior distance estimators:

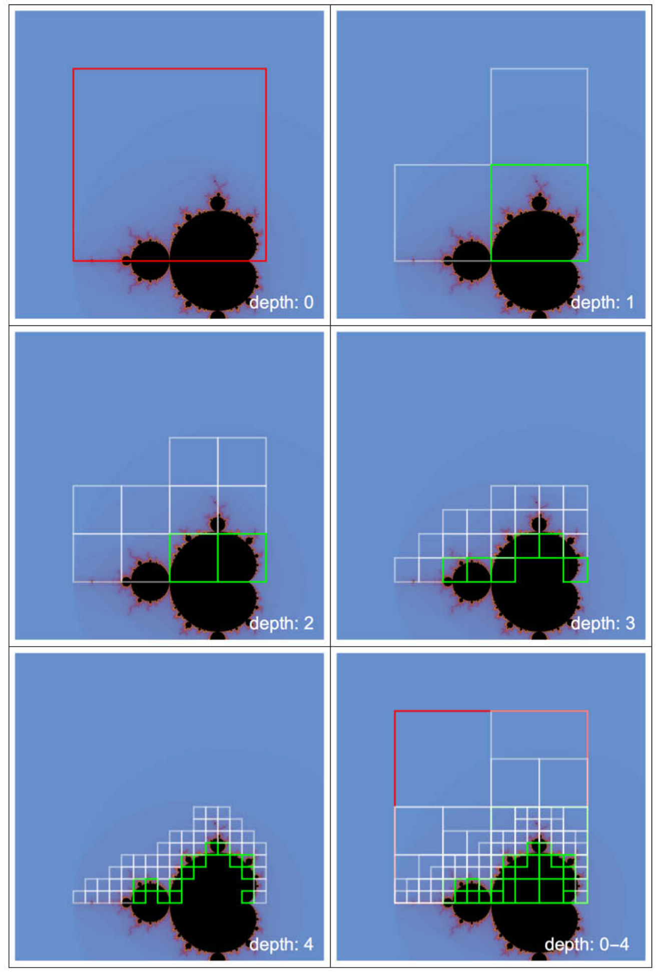


FIG. 3. The lowest five depths of the applied tessellations and a synopsis (bottom right), i.e. the tessellation tree. The initialization tile is marked red. Tiles that are fully outside or inside the Mandelbrot set are omitted (type 1 or 2, refer to section II B 2). Tiles with center outside the Mandelbrot set but covering parts of the Mandelbrot set are marked white (type 4). Tiles with center inside the Mandelbrot set but partly outside the Mandelbrot set are marked green (type 2). Observe that the synopsis of the different depths covers the hole area of the red tile at depth = 0. Interior distance estimation, here, is based on case $\alpha = 1$ (refer to section II B 1).

**Case $\alpha = 1$:** This is a conservative, i.e. smaller, under estimate of the interior distance based on Koebe quarter theorem. In this case the factor 2 at line 229 in section A 1 should amount to 4.

**Case $\alpha = 2$:** This is a more progressive, i.e. larger, under estimate of the interior distance. This case corresponds with factor 2 at line 229 in section A 1. Y. Fisher and J. Hill[5] suppose this method based on their numerical findings. In addition, the actual implementation of the *MandelbrotDistance* function in Mathematica 10 uses this approach.

### 2. Quad Tree Tessellation

The lowest five depths of the applied tessellations are depicted in figure 3. Selected parts of the tessellation tree at larger depths are shown in figures 4 and 7.

Each tile of the tessellation tree can fully be specified by its center $(x, y)$ and its side length $s$, since only quadratic tiles are allowed. The red initialization tile in figure 3 upper left has center $(-0.75, 1.25)$ and a side length of 2.5.

When depth is increased by one each tile is divided into four sub tiles ('quad'); or, in other words, the side length of each new tile is divided by two. This repeated segmentation is similar to repeated branching ('tree').

Together with the distance estimator (refer to II B 1) five different types of tiles can be distinguished:

**Type 1:** Full member tiles – The center of the tile is inside the Mandelbrot set and the interior distance of the center is larger than $s\sqrt{2}$, where $s$ is the side length of the tile. These tiles are omitted in figures 3 and 4 because they need no further refinement in the following depths of the tessellation.

**Type 2:** Full outside tiles – The center of the tile is outside the Mandelbrot set and the exterior distance of the center is larger than $s\sqrt{2}$, where $s$ is again the side length of the tile. These tiles are omitted in figures 3 and 4, too. These tiles also need no further refinement in the following depths of the tessellation.

**Type 3:** Partly member tiles – The center of the tile is inside the Mandelbrot set and the interior distance of the center is equal or smaller than $s\sqrt{2}$, where $s$ is again the side length of the tile. These tiles are marked green in figures 3 and 4. These tiles will be segmented in the next depth of the tessellation.

**Type 4:** Partly outside tiles – The center of the tile is outside the Mandelbrot set and the exterior distance of the center is equal or smaller than $s\sqrt{2}$, where $s$ is again the side length of the tile. These tiles are marked white in figures 3 and 4. These tiles will also be segmented in the next depth of the tessellation.

**Type 5:** Chaotic tiles – With regard to spesific parameters of the distance calculation, i.e. the maximum iteration, it can not be decided wether the center of the tile is outside or inside the Mandelbrot set. Thus, the distance of the center can not be estimated. These tiles are marked red in figure 4 and will be segmented in the next depth of the tessellation. Observe that the initialization tile in figure 3 is marked red, too.

Thus, for a given tile, i.e. the center $(x, y)$ and the side length $s$ the *decide* procedure returns the respective type (1 – 5) of the tile. That's why the procedure is called 'decide', here. The procedure is part of the quad tree algorithm, which is described in section II B 4. The OpenCL implementation of the procedure is outlined in section A 1.

Another important procedure of the quad tree algorithm is the *expand* procedure. This procedure runs on the CPU and is implemented as an compiled function within Mathematica (refer to A 2). Essentially, the *expand* procedure calculates the new tile parameters, i.e. the new centers $(x, y)$ and the halved side lengths $s$ of the new tiles in the next depth of the tessellation.

Basically both, the *decide* and the *expand* procedures, are applied alternating to the tiles. Refer to figure 5 for a flow chart of the quad tree algorithm.

The complete tessellation tree of the red initialization tile in figure 3 upper left consists of quadratic tiles of different sizes, i.e. tiles at different depths. With respect to figure 3 bottom right observe on the one hand that the tiles cover the hole initialization tile – but, on the other hand, the total area of the tiles is larger than the area of the initialization tile, because the sub tiles do partly overlap. Nevertheless, is possible to calculate the exact parts of the area of the different tile types. The calculation of the different areas is outlined in section II B 5.

To sum this up the following nomenclature is used in the following sections:

**Tile:** Tiles are always quadratic and can be fully specified by their center $(x, y)$ and by their side length $s$. There are five different types of tiles (see above).

**Tessellation:** Here, a tessellation is a set of tiles of the same size. Observe that a given tessellation typically does not gapless cover the area. Tessellations can be enumerated by an index called depth, here.

**Tessellation tree:** A tessellation tree is a set of successive tessellations – starting with the tessellation of depth = 0, that contains only one (initialization) tile – up to tessellation of depth = $n$. Observe that a given tessellation tree covers the complete area of the initialization tile, but the area of all tiles in the tessellation tree does not sum up to the area of the initialization tile, because the tiles from different tessellations do partly overlap.

### 3. Performance Considerations

With increasing depth the number of tiles is growing fast, i.e. factor 4. Actually, because the tiles of type 1 and 2 are removed from the quad tree the number of tiles grows just by a factor of 3.5 approximately. This rapidly growing number leads to a rapidly growing CPU time and a rapidly growing file size of the quad tree, which both have to be reasonably addressed.

The *decide* and the *expand* procedures can be applied in parallel to a large number of tiles. Thus, a OpenCL

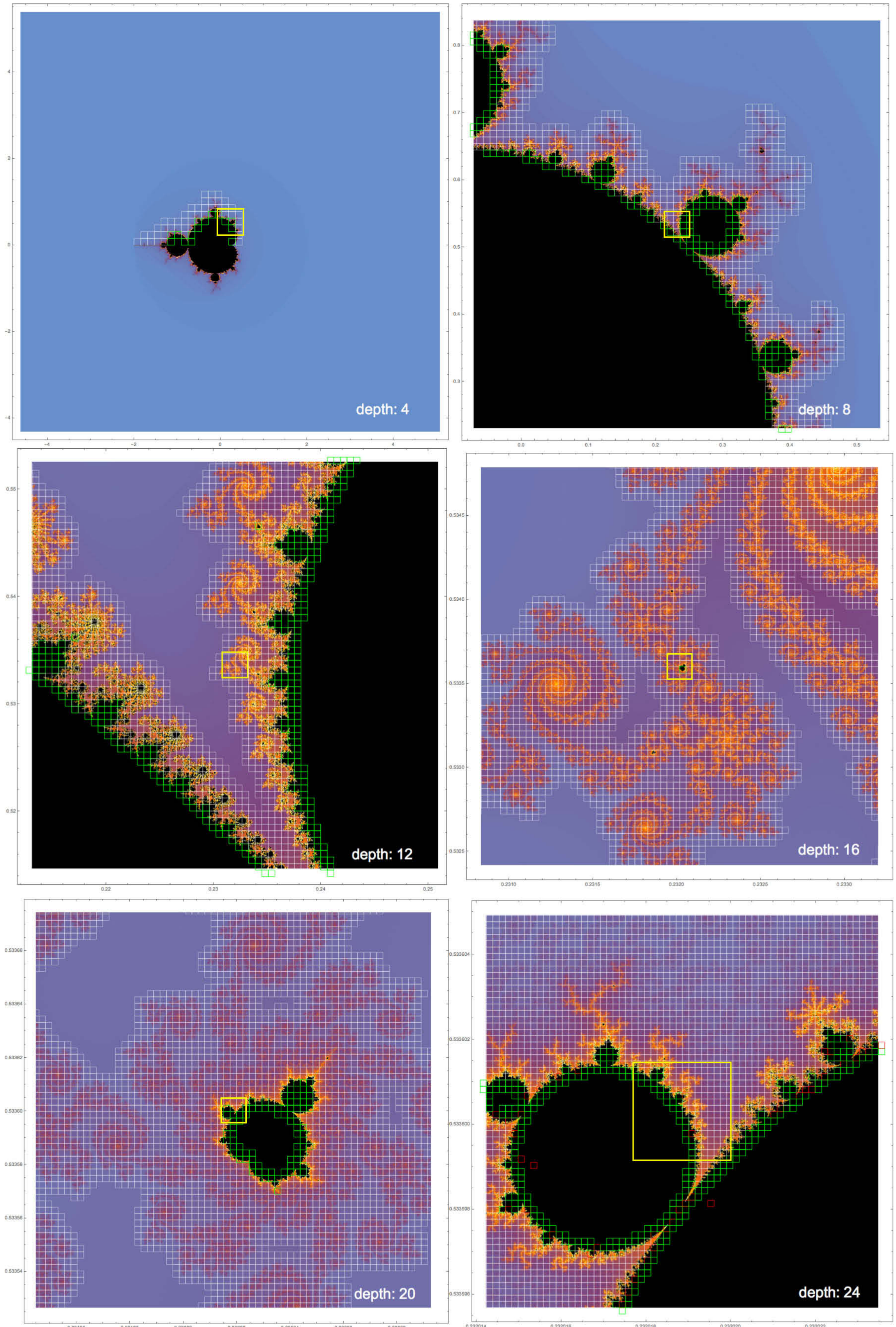


FIG. 4. Selected parts of the tessellation tree at different depths with $\alpha = 1$. Zoomed regions are marked yellow in lower depths. For details refer to caption of figure 3 and section II B 2. The yellow square at depth 24 corresponds to figure 7.

implementation is handy. Here, the *decide* procedure is implemented in OpenCL (refer to A 1) and the *expand* procedure is implemented as a compiled Mathematica function with compilation target C and allowed parallelization (refer to A 2).

In order to handle a large number of tiles at a given depth of tessellation, the list of tiles is partitioned into lists of less than 100 000 tiles. Each list is stored in a separate file. For each tile the center $(x, y)$ and the type is stored. The side length $s$ need not to be stored because it is constant at a given depth. A typical file in text format has a file size of about 7 MB. Compression reduces the file size to about 0.7 MB. The last tessellation that is stored on the hard disc has depth 23. It consists of about 2.3 million files and thus has a total file size of about 1.6 TB of already compressed data. Compression consumes about 30% of CPU time. For more details about timings refer to III C.

Since the tessellation of depth 23 is the last one saved on the hard disc, the tiles of the tessellation at depth 24 are counted but their coordinates are not saved to disc. In oder to count the tiles of the tessellations 25 and 26 the tiles of tessellation 23 have to be expanded twice and thrice before counting (refer to the dashed arrow in figure 5 and following section II B 4).

In order to reduce CPU time the maximum iteration of tessellations 25 and 26 is reduced from 10 241 024 to 21 024 (refer to section II B 1). Since the maximum iteration mainly affects the lower bound of the area estimation the upper bound can still be lowered even with this reduced maximum iteration. Specialities of depths 25 and 26 are discussed in section III D.

#### *4. Outline of the Quad Tree Algorithm*

A flow chart of the quad tree algorithm is depicted in figure 5. The algorithm starts with the initialization tessellation at depth = 0 consisting of one tile with center $(-0.75, 1.25)$ and a side length of 2.5 (refer to figure 3 upper left). The type of the initialization tile is set to 5. Thus, it is marked red in figure 3. Refer to section A 3 for the definition of the initialization tile in the program code.

In the next step the *separate* procedure is called. Tiles of type 1 and 2 of the current tessellation are removed and counted. The results are stored in separate 'result' files. The coordinates of the other tiles are stored in 'quad tree' files. If the number of tiles is larger than 100 000 the tiles stored in two 'quad tree' files. Thus, the file size of the 'quad tree' files is limited to about 0.7 MB. The 'quad tree' files are compressed using Mathematica's build-in *Compress* function. Refer to section A 2 for the program code. The separation and the export/import is combined in a function called 'saveDecide'.

To sum this up, there are two important file types:

**Quad tree files:** These files contain the data needed for representing the tessellation tree. To limit their size to about 0.7 MB the maximum number of tiles in on file is limited to 100 000. The files are compressed using Mathematica's Compress function. Only tiles of type 3, 4 and 5 are stored in these files. So, in order to store the hole tessellation at depth = 23 for example, 2.5 million files and thus a total file size of about 1.6 TB are needed.

**Result files:** For each 'quad tree' file a corresponding 'result' file is written to disc. These files contain only the frequencies of the different tile types. Thus, they have very small file size and do not need further compression. The precise numbers of different tile types at different depths of the tessellation tree can be derived from these files (refer to section II B 5).

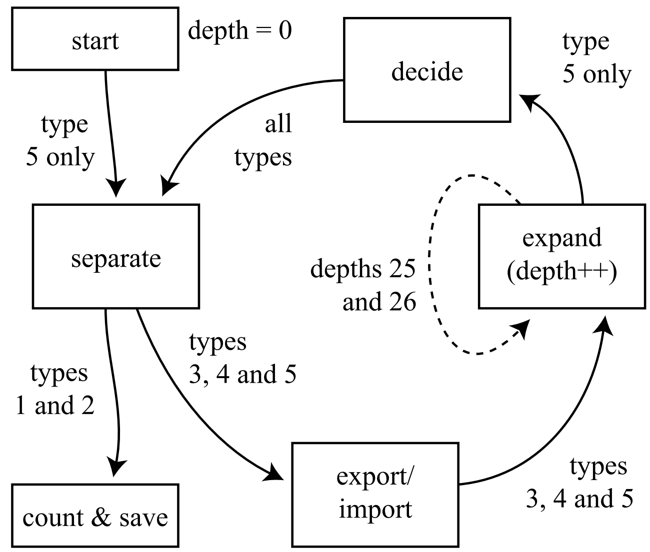


FIG. 5. The algorithm of the quad tree tessellation. The quad tree algorithm is outlined in section II B 4.

When all 'quad tree' and 'result' files of the actual tessellation are written to hard disc, the 'quad tree' files are successively imported again. Then the *expand* procedure is applied to each tile of the actual tessellation. Each tile is divided into four sub tiles with new centers $(x, y)$ and halved side lengths. The type of the new tiles is set to 5. The parameter depth is increased by one. Refer to section A 2 for the program code.

An important feature of the *expand* procedure is that it can be called repeatedly, as indicated in by the dashed arrow in figure 5. This feature is used to reach the tessellations of depth 25 and 26 without need for saving tessellations higher than 23 on hard disc (refer to section II B 3).

In the next step the types of the new tiles have to be determined. The *decide* procedure is applied in parallel to each tile. Thus, the *decide* procedure is implemented in OpenCL (refer to A 1). Finally all tiles are associated to the five types and they are again transferred to the *separate* procedure. A new cycle of the quad tree algorithm starts.

The quad tree algorithm ends at a given depth with one or many 'result' files for each depth containing the counts of the different tile types. These files need further handling which is detailed in section II B 5.

#### 5. *Post Processing of the Tessellation Tree*

'Result' files contain the frequencies of the different tile types (refer to section II B 4). There is at least one 'result' file for each depth of the tessellation tree. If there are more than one 'result' file the counts in these files have to be totalized, resulting in one 'total result' file for each depth of the tessellation tree.

Since every tessellation consists of tiles only of type 3, 4 and 5 of the previous tessellation, the counts for the tiles of type 1 and 2 of the actual tessellation have to be reconstructed from previous tessellations. For example, the single tile of type 2 at depth 1 adds $(2^{26-1})^2 = 33\,554\,432^2 = 1\,125\,899\,906\,842\,624$ tiles of type 2 to the tessellation at depth 26.

The reconstructed counts for the different tile types at different depths are listed in table I. Observe, that the tile counts are given for both cases $\alpha = 1$ (conservative) and $\alpha = 2$ (progressive) of the interior distance estimator (refer to section II B 1).

## III. RESULTS

### A. Tile counts at different depths

The evaluated tile counts for depths $d = 0..26$ are listed in table I. For each depth the tile counts of both cases $\alpha = 1$ (conservative) and $\alpha = 2$ (progressive) of the interior distance estimator are given. Refer to section II B 1 for details of the applied interior distance estimator.

The individual counts are calculated as described in section II B 5. Precise upper and lower bounds of the area of the Mandelbrot set can be derived from the tile counts with $\alpha = 1$ (conservative) (refer to section III B and table II).

Observe, that some tile counts at depths 25 and 26 change discontinuously. The reason is that the parameters of the distance estimation (refer to section II B 1) are changed, due to performance considerations (refer to section II B 3). Specialities of depths 25 and 26 are discussed in section III D.

### B. Area Estimate

Based on the tile counts in table I precise lower and upper bounds of the area of the Mandelbrot set can be calculated. The lower bound corresponds to the tile count of type 2 tiles. The upper bound corresponds to the total tile counts of type 2 and hybrid tiles. For the definition of hybrid tiles refer to table I and section II B 2. The calculation of the precise lower and upper bounds is based on tile counts resulting from the conservative interior distance estimator ($\alpha = 1$).

Calculated lower and upper bounds of the area at different tessellation depths are listed in table II and depicted in figure 6.

The current precise lower and upper bounds can be read of table II: the area of the Mandelbrot set is between 1.506 40 (refer to row 26'(1)) and 1.531 21 (refer to row 26'(1)). Corresponding bounds from Y. Fisher and J. Hill amount to 1.502 97 and 1.570 13 [5]. The difference of 0.063 16 between the bounds from Y. Fisher and J. Hill can approximately be more than halved by this approach.

The bounds stated by Y. Fisher and J. Hill are quite close to the calculated bounds of depth 19, here. Curiously Y. Fisher and J. Hill call this level 17. The small differences of the bounds may result from different parameters of the exterior/interior distance estimation (refer to section II B 1) and form the parameters of the initialization tile (refer to section II B 2).

### C. Time Consumption

In table II the CPU (and GPU) times for calculations with $\alpha = 1$ are listed for each depth, i.e. the listed timings are not accumulated. The values at depths 24, 25 and 26 are estimated because of discontinued computations. With $\alpha = 2$ the times are approximately 2 % smaller.

In the range from 18 to 23 the CPU time is increasing by factor 3 with increasing depth. Thus, at depth 24 a CPU time of about 400 h is expected. Since no quad tree files (refer to section II B 4) are written to disc at depth 24, the resulting CPU time is about 30 % smaller (refer to section II B 3).

Based on extrapolation the timings for depth 25 and 26 add up to 1200 h (50 days) and 3600 h (150 days). In order to keep the CPU time below one month depths 25 and 26 are calculated with altered parameters of the interior distance estimator (refer to section III D) resulting in about 6× smaller timings.

### D. Parameters of depths 25 and 26

In order to significantly reduce CPU time (refer sections II B 3 and III C) the maximum iteration is considerably reduced (refer to section II B 1) at depths 25 and 26. To indicate this, these depths are typically primed, i.e. 25' and 26', in tables and figures.

Figure 7 contrasts the resulting tessellations based on the standard maximum iteration (left column) with the corresponding tessellations based on the reduced maximum iteration (right column) at depth 26 exemplarily. The yellow marked zoom region in figure 4 corresponds

TABLE I. Tile counts at different depths for both cases $\alpha = 1$ (conservative) and $\alpha = 2$ (progressive) of the interior distance estimator (refer to section II B 1). For more details about the different tile types refer to enumeration in section II B 2. Colors are used in figures 3 and 4. Depths 25 and 26 are indicated by primes because they are calculated with different parameters. Thus, some of the tile counts change discontinuously. For more details refer to section III A.

| depth $d(\alpha)$ | side length $s = 2^d$ | total tiles $s^2 = T_1 + T_2 + T$ | full outside tiles type 2 $T_2$ | full inside tiles type 1 $T_1$ | hybrid tiles $T = T_3 + T_4 + T_5$ | partly outside type 3, white $T_3$ | p.inside t.4, green $T_4$ | chaotic tiles type 5, red $T_5$ |
|---|---|---|---|---|---|---|---|---|
| 0(1) | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0(2) | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1(1) | 2 | 4 | 1 | 0 | 3 | 2 | 1 | 0 |
| 1(2) | 2 | 4 | 1 | 0 | 3 | 2 | 1 | 0 |
| 2(1) | 4 | 16 | 6 | 0 | 10 | 8 | 2 | 0 |
| 2(2) | 4 | 16 | 6 | 0 | 10 | 8 | 2 | 0 |
| 3(1) | 8 | 64 | 39 | 2 | 23 | 17 | 6 | 0 |
| 3(2) | 8 | 64 | 39 | 2 | 23 | 17 | 6 | 0 |
| 4(1) | 16 | 256 | 177 | 18 | 61 | 46 | 15 | 0 |
| 4(2) | 16 | 256 | 177 | 18 | 61 | 46 | 15 | 0 |
| 5(1) | 32 | 1 024 | 771 | 92 | 161 | 129 | 32 | 0 |
| 5(2) | 32 | 1 024 | 771 | 92 | 161 | 129 | 32 | 0 |
| 6(1) | 64 | 4 096 | 3 238 | 407 | 451 | 359 | 92 | 0 |
| 6(2) | 64 | 4 096 | 3 238 | 413 | 445 | 359 | 86 | 0 |
| 7(1) | 128 | 16 384 | 13 370 | 1 750 | 1 264 | 1 022 | 241 | 1 |
| 7(2) | 128 | 16 384 | 13 370 | 1 780 | 1 234 | 1 022 | 211 | 1 |
| 8(1) | 256 | 65 536 | 54 564 | 7 316 | 3 656 | 3 083 | 573 | 0 |
| 8(2) | 256 | 65 536 | 54 564 | 7 443 | 3 529 | 3 083 | 446 | 0 |
| 9(1) | 512 | 262 144 | 221 242 | 30 137 | 10 765 | 9 304 | 1 455 | 6 |
| 9(2) | 512 | 262 144 | 221 242 | 30 504 | 10 398 | 9 301 | 1091 | 6 |
| 10(1) | 1 024 | 1 048 576 | 893 222 | 122 698 | 32 656 | 28 982 | 3 656 | 18 |
| 10(2) | 1 024 | 1 048 576 | 893 222 | 123 750 | 31 604 | 28 974 | 2 612 | 18 |
| 11(1) | 2 048 | 4 194 304 | 3 597 092 | 496 280 | 100 932 | 91 681 | 9 188 | 63 |
| 11(2) | 2 048 | 4 194 304 | 3 597 092 | 499 164 | 98 048 | 91 640 | 6 345 | 63 |
| 12(1) | 4 096 | 16 777 216 | 14 460 134 | 1 999 241 | 317 841 | 294 959 | 22 593 | 289 |
| 12(2) | 4 096 | 16 777 216 | 14 460 134 | 2 006 829 | 310 253 | 294 792 | 15 177 | 284 |
| 13(1) | 8 192 | 67 108 864 | 58 059 613 | 8 032 011 | 1 017 240 | 960 764 | 55 339 | 1 137 |
| 13(2) | 8 192 | 67 108 864 | 5 8059 613 | 8 051 978 | 997 273 | 960 037 | 36 108 | 1 128 |
| 14(1) | 16 384 | 268 435 456 | 232 915 056 | 32 214 438 | 3 305 962 | 3 166 121 | 135 418 | 4 423 |
| 14(2) | 16 384 | 268 435 456 | 232 915 056 | 3 2266 676 | 3 253 724 | 3 163 158 | 86 202 | 4 364 |
| 15(1) | 32 768 | 1 073 741 824 | 933 790 101 | 129 073 027 | 10 878 696 | 10 535 474 | 325 653 | 17 569 |
| 15(2) | 32 768 | 1 073 741 824 | 933 789 804 | 129 209 229 | 1 0742 791 | 1 0523 680 | 201 755 | 17 356 |
| 16(1) | 65 536 | 4 294 967 296 | 3 741 951 161 | 516 822 508 | 3 6193 627 | 35 351 140 | 771 867 | 70 620 |
| 16(2) | 65 536 | 4 294 967 296 | 3 741 947 638 | 517 181 676 | 35 837 982 | 35 305 699 | 462 813 | 69 470 |
| 17(1) | 131 072 | 17 179 869 184 | 14 989 686 577 | 2 068 585 656 | 121 596 951 | 119 522 811 | 1 792 260 | 281 880 |
| 17(2) | 131 072 | 17 179 869 184 | 14 989 661 153 | 2 069 534 138 | 120 673 893 | 119 351 062 | 1 055 891 | 266 940 |
| 18(1) | 262 144 | 68 719 476 736 | 60 029 958 552 | 8 277 406 710 | 412 111 474 | 406 879 291 | 4 121 677 | 1 110 506 |
| 18(2) | 262 144 | 68 719 476 736 | 60 029 809 562 | 8 280 019 251 | 409 647 923 | 406 235 644 | 2 390 547 | 1 021 732 |
| 19(1) | 524 288 | 274 877 906 944 | 240 353 790 780 | 33 116 792 335 | 1 407 323 829 | 1 393 553 259 | 9 414 774 | 4 355 796 |
| 19(2) | 524 288 | 274 877 906 944 | 240 353 007 132 | 33 124 439 264 | 1 400 460 548 | 1 391 156 484 | 5 368 652 | 3 935 412 |
| 20(1) | 1 048 576 | 1 099 511 627 776 | 962 189 238 785 | 132 483 851 445 | 4 838 537 546 | 4 800 134 425 | 21 256 443 | 17 146 678 |
| 20(2) | 1 048 576 | 1 099 511 627 776 | 962 185 387 560 | 132 507 822 680 | 4 818 417 536 | 4 791 239 255 | 11 885 269 | 15 293 012 |
| 21(1) | 2 097 152 | 4 398 046 511 104 | 3 851 336 461 567 | 529 973 966 852 | 16 736 082 685 | 16 621 019 575 | 47 327 542 | 67 735 568 |
| 21(2) | 2 097 152 | 4 398 046 511 104 | 3 851 318 353 176 | 530 054 346 629 | 16 673 811 299 | 16 588 078 940 | 25 880 154 | 59 852 205 |
| 22(1) | 4 194 304 | 17 592 186 044 416 | 15 413 996 505 530 | 2 119 984 208 854 | 58 205 330 032 | 57 833 365 121 | 103 565 735 | 268 399 176 |
| 22(2) | 4 194 304 | 17 592 186 044 416 | 15 413 913 972 934 | 2 120 269 727 493 | 58 002 343 989 | 57 711 539 338 | 55 236 852 | 235 567 799 |
| 23(1) | 8 388 608 | 70 368 744 177 664 | 61 685 165 726 455 | 8 480 136 960 072 | 203 441 491 137 | 202 153 609 362 | 221 774 194 | 1 066 107 581 |
| 23(2) | 8 388 608 | 70 368 744 177 664 | 61 684 798 155 308 | 8 481 196 327 648 | 202 749 694 708 | 201 703 361 795 | 115 099 822 | 931 233 091 |
| 24(1) | 16 777 216 | 281 474 976 710 656 | 246 839 623 842 876 | 33 920 994 359 344 | 714 358 508 436 | 709 653 138 630 | 462 828 303 | 4 242 541 503 |
| 24(2) | 16 777 216 | 281 474 982 710 656 | 246 838 016 125 231 | 33 925 044 558 813 | 711 922 026 612 | 707 994 674 519 | 233 811 346 | 3 693 540 747 |
| 25'(1) | 33 554 432 | 1 125 899 906 842 624 | 988 090 988 392 369 | 135 684 048 606 756 | 2 522 499 667 947 | 2 174 847 868 839 | 23 704 025 | 347 628 095 083 |
| 25'(2) | 33 554 432 | 1 125 899 906 842 624 | 987 688 207 138 272 | 135 699 211 786 362 | 2 512 487 917 990 | 2 171 104 588 977 | 23 459 772 | 341 359 869 241 |
| 26'(1) | 67 108 864 | 4 503 599 627 370 496 | 3 956 445 430 108 888 | 542 736 526 509 859 | 8 938 453 210 521 | 7 547 894 457 325 | 47 400 846 | 1 390 511 352 350 |
| 26'(2) | 67 108 864 | 4 503 599 627 370 496 | 3 951 902 496 891 266 | 542 796 894 067 968 | 8 900 236 411 262 | 7 534 750 938 783 | 46 894 494 | 1 365 438 577 985 |

to the upper row of figure 7. This row shows that the reduced maximum iteration leads to red tiles inside the Mandelbrot set. The middle row of figure 7 is a zoom view from the 'spike' region in the far left of the Mandelbrot set. This middle row shows that the reduced maximum iteration does not affect white tiles outside the Mandelbrot set. The lowest row of figure 7 is a zoom view from the boundary of the main cardioid of the Mandelbrot set. This lowest row shows the effect of the implemented cardioid testing (refer to section II B 1).

Thus, the reduction of the maximum iteration mainly affects the lower bound of the area estimation, because the new red tiles appear mainly inside the Mandelbrot set. Since at higher depths the main reduction of the area bounds is gained from the upper bound, the reduction of the maximum iteration still helps to reduce the area bounds.

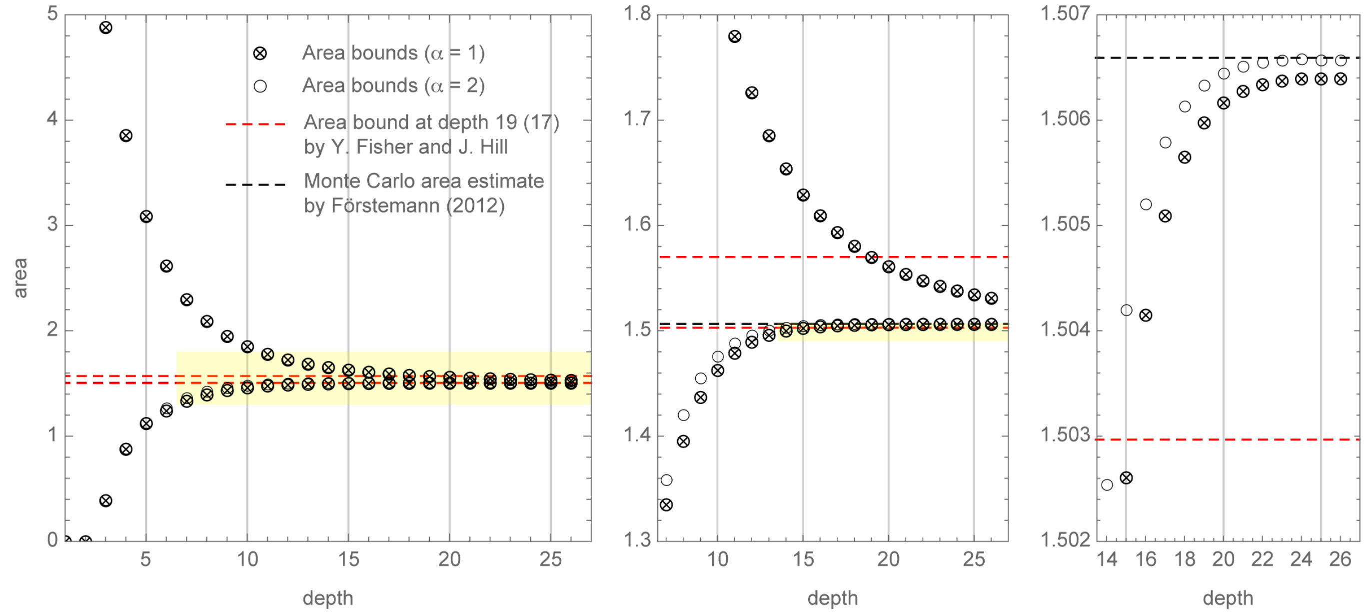


FIG. 6. The lower and upper bounds of the Mandelbrot area as a function of depth. Bounds are given for $\alpha = 1$ and $\alpha = 2$. The final area bound by Y. Fisher and J. Hill is marked by a dashed red line (refer to III B). The Monte Carlo area estimate by Förstemann[4] is marked by a dashed black line. The middle and the right figures are blow-ups – zoomed regions are marked yellow. This figure is inspired by the similar figure 2 in Y. Fisher and J. Hill [5].

TABLE II. Precise lower and upper bounds of the area of the Mandelbrot set at different tessellation depths $d$ with $\alpha = 1$. In addition the difference between the upper and the lower bound and the CPU/GPU time needed are listed.

| $d(\alpha)$ | lower bound | upper bound | difference | CPU time |
|---|---|---|---|---|
| 0(1) | 0.000 00 | 12.500 00 | 12.500 00 | $<$ 0.01 h |
| 1(1) | 0.000 00 | 9.375 00 | 9.375 00 | $<$ 0.01 h |
| 2(1) | 0.000 00 | 7.812 50 | 7.812 50 | $<$ 0.01 h |
| 3(1) | 0.390 63 | 4.882 81 | 4.492 19 | $<$ 0.01 h |
| 4(1) | 0.878 91 | 3.857 42 | 2.978 52 | $<$ 0.01 h |
| 5(1) | 1.123 05 | 3.088 38 | 1.965 33 | $<$ 0.01 h |
| 6(1) | 1.242 07 | 2.618 41 | 1.376 34 | $<$ 0.01 h |
| 7(1) | 1.335 14 | 2.299 50 | 0.964 35 | $<$ 0.01 h |
| 8(1) | 1.395 42 | 2.092 74 | 0.697 32 | $<$ 0.01 h |
| 9(1) | 1.437 04 | 1.950 36 | 0.513 31 | $<$ 0.01 h |
| 10(1) | 1.462 67 | 1.851 96 | 0.389 29 | $<$ 0.01 h |
| 11(1) | 1.479 03 | 1.779 83 | 0.300 80 | 0.01 h |
| 12(1) | 1.489 55 | 1.726 36 | 0.236 81 | 0.02 h |
| 13(1) | 1.496 08 | 1.685 55 | 0.189 48 | 0.03 h |
| 14(1) | 1.500 10 | 1.654 05 | 0.153 95 | 0.04 h |
| 15(1) | 1.502 61 | 1.629 25 | 0.126 64 | 0.05 h |
| 16(1) | 1.504 15 | 1.609 49 | 0.105 34 | 0.09 h |
| 17(1) | 1.505 09 | 1.593 57 | 0.088 47 | 0.21 h |
| 18(1) | 1.505 65 | 1.580 61 | 0.074 96 | 0.55 h |
| 19(1) | 1.505 98 | 1.569 98 | 0.064 00 | 1.53 h |
| 20(1) | 1.506 17 | 1.561 17 | 0.055 01 | 4.47 h |
| 21(1) | 1.506 28 | 1.553 84 | 0.047 57 | 13.38 h |
| 22(1) | 1.506 34 | 1.547 70 | 0.041 36 | 41.83 h |
| 23(1) | 1.506 37 | 1.542 51 | 0.036 14 | 135.65 h |
| 24(1) | 1.506 39 | 1.538 12 | 0.031 72 | $\sim$ 275 h |
| 25'(1) | 1.506 40 | 1.534 40 | 0.028 00 | $\sim$ 170 h |
| 26'(1) | 1.506 40 | 1.531 21 | 0.024 81 | $\sim$ 550 h |

### E. Comparison between $\alpha = 1$ and $\alpha = 2$

The parameter $\alpha$ and the two different cases $\alpha = 1$ and $\alpha = 2$ are introduced in section II B 1. In figure 8 the resulting tessellations based case $\alpha = 1$ (left column) and case $\alpha = 2$ (right column) are exemplarily shown. The zoomed regions are the same as in figure 7 (refer to section III D). Both cases are calculated with standard maximum iteration (refer to section III D).

Figure 8 suggests that the progressive interior distance estimator with $\alpha = 2$ yields reasonable results. The CPU time benefit of the progressive estimator of about 2 % (refer to section III C) is rather small in the calculations applied here.

## IV. DISCUSSION

The area bounds of the Mandelbrot set are quite slowly converging, as already mentioned by Y. Fisher and J. Hill [5]. In this approach a 16 000× larger tessellation just leads to halved area bounds.

The tile counts listed in table I are predestined for further fractal dimension analysis [15] based on box counting dimension or Minkowski-Bouligand dimension [16]. In addition, since the interior distance estimator needs orbit detection, lower area bounds of hyperbolic components [17] can be derived from numerical data gathered here. These and further investigations shall be subject to future projects.

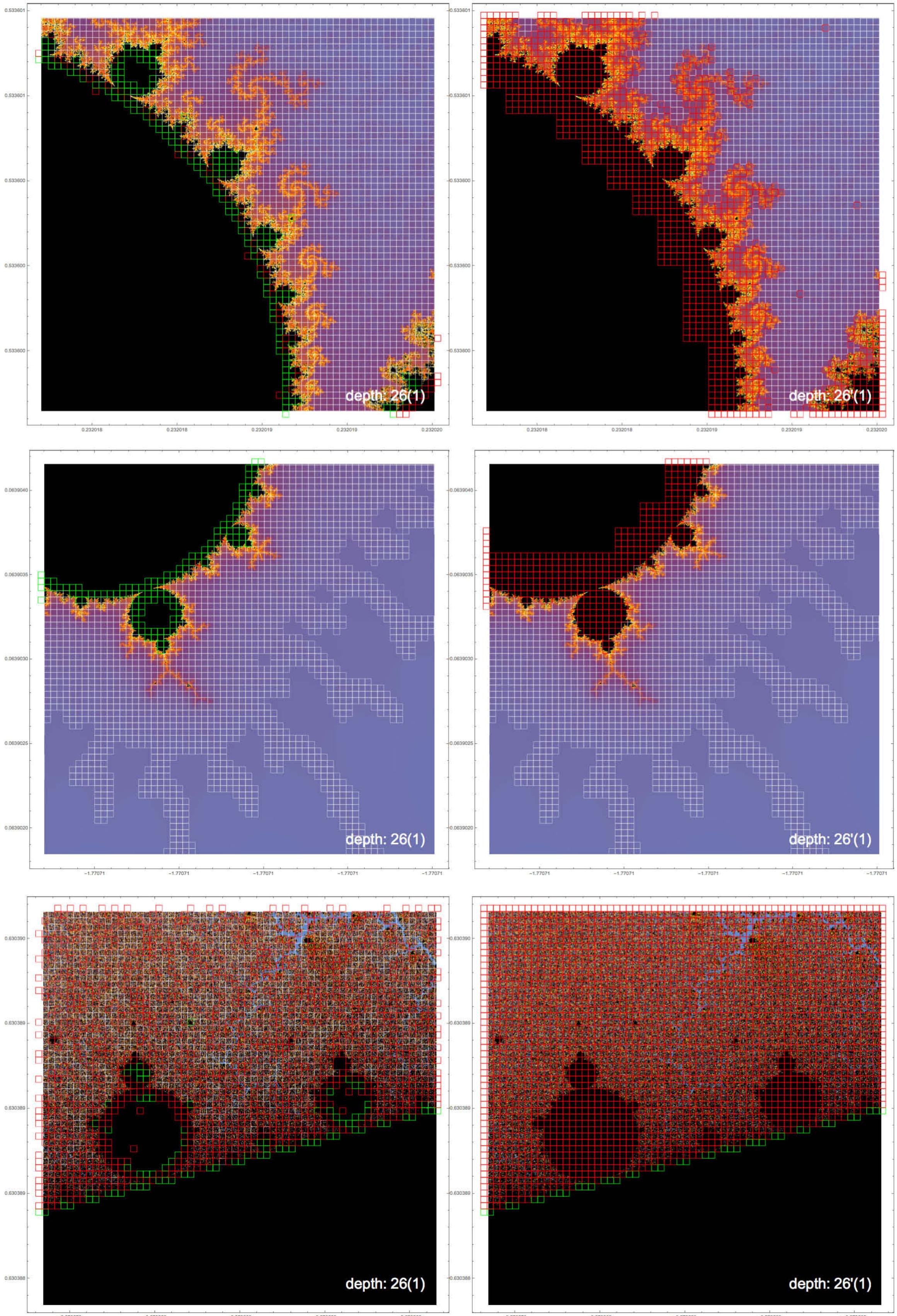


FIG. 7. Comparison of tessellations based on the standard maximum iteration (left column) with the corresponding tessellations based on the reduced maximum iteration (right column) for three different regions at depth 26 with $\alpha = 1$. For discussion refer to section III D.

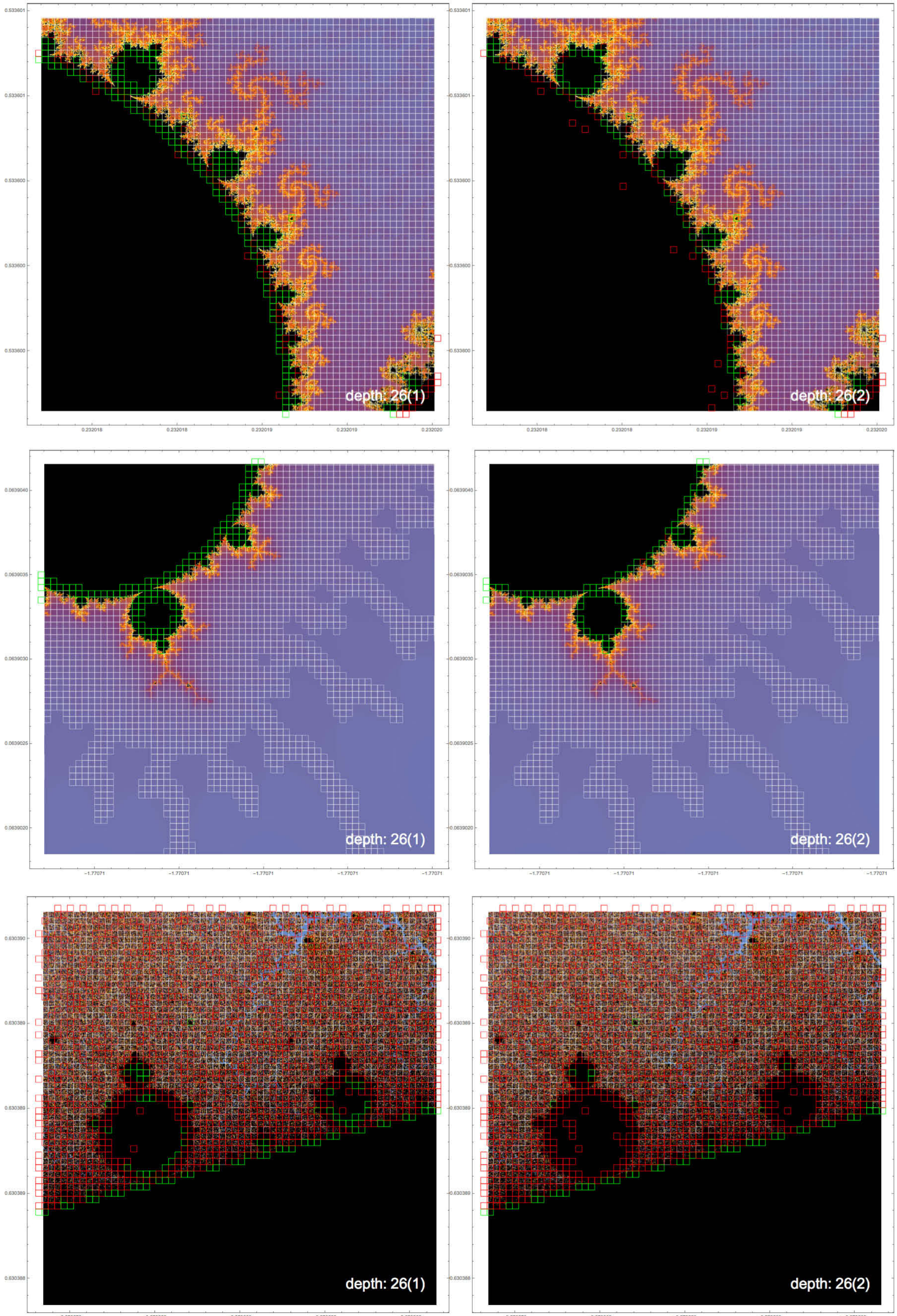


FIG. 8. Comparison of resulting tessellations based on the conservative interior distance estimator with $\alpha = 1$ (left column) and the corresponding tessellations based on the progressive interior distance estimator with $\alpha = 2$ (right column). For discussion refer to section III E.

## Appendix A: Soucre Code

### 1. GPU Code for the *decide* Procedure

This is the GPU code of the *decide* procedure. The code is stored as a simple string. Essentially an exterior and interior distance estimator is implemented in OpenCL.

```
(* openCL source code *)
clSourceDecide="
#ifdef USING_DOUBLE_PRECISIONQ
#pragma OPENCL EXTENSION cl_khr_fp64 : enable
#pragma OPENCL EXTENSION cl_amd_fp64 : enable
#endif /* USING_DOUBLE_PRECISIONQ */
__kernel void rawDecide(
__global double * a,  // real part of tile center
__global double * b,  // real part of tile center
__global double * c,  // result flag
__global double * d,  // result cycle count
mint startIterations, // maximum start iterations, parameter
mint maxDepth,        // maximum depth, parameter
double boxSize,       // side length of box, parameter
mint n) {
  int iGID = get_global_id(0);
  if (iGID < n) {

    // prolog: init variables
    double cx = a[iGID]; // real part of tile center
    double cy = b[iGID]; // real part of tile center
    double zx = a[iGID];
    double zy = b[iGID];
    double escapeRadius2 = 10000.; // square of escape radius 100, empiric paramter
    long maxCycleCount = 20000;    // maximum length of detected cycles, empiric paramter
    double q = 0.0; // interim value
    bool refuge = false; // cx ,cy is refugee
    bool member = false;  // cx ,cy is refugee
    bool card = false;    // cx ,cy is member of cardioid
    bool circ = false;    // cx ,cy is member of circle
    long iterationCount = 0; // iteration counter
    long iterationDepthLimit = 0; // iteration limit post iteration
    long cycleCount = 0;
    long depth = 0;
    double tx = 0.0; // interim value
    double ty = 0.0; // interim value
    bool card1 = false; bool card2 = false;
    bool card3 = false; bool card4 = false;
    bool circ1 = false; bool circ2 = false;
    bool circ3 = false; bool circ4 = false;
    double z2 = 0.;  double absz = 0.;
    double z2x = 0.; double z2y = 0.;
    double dzx = 1.; double dzy = 0.;
    double zCyclex = 0.; double zCycley = 0.;
    double D1r = 1.; double D1i = 0.;
    double D2r = 0.; double D2i = 0.;
    double D3r = 0.; double D3i = 0.;
    double D4r = 0.; double D4i = 0.;
    double D1rT = 0.; double D1iT = 0.;
    double D2rT = 0.; double D2iT = 0.;
    double D3rT = 0.; double D3iT = 0.;
    double D4rT = 0.; double D4iT = 0.;
    double valA = 0.; double valB = 0.;
    double distance = 5.; // default value
    long i = 0;
    double result = 0.;

    // cardioid and circle test

    if (!refuge) {
      q = pown((cx - 0.25), 2) + cy * cy;
      card = q * (q + (cx - 0.25)) < 0.25 * cy * cy;
```

```
      circ = pown((cx + 1.0), 2) + cy * cy < 0.0625;
      member = (card || circ);
    }

    // pre iteration

    for ( iterationCount = 0;
      (!member) && (!refuge) && (iterationCount < startIterations);
      iterationCount++) {
        z2 = pown(zx, 2) - pown(zy, 2) + cx;
        z2y = 2.0 * zx * zy + cy;
        z2x = z2;
        z2 = 2.0 * (zx * dzx - zy * dzy) + 1;
        dzy = 2.0 * (zx * dzy + zy * dzx);
        dzx = z2;
        zx = z2x;
        zy = z2y;
        refuge = (pown(zx, 2) + pown(zy, 2)) > escapeRadius2;
    }

    // post iteration and cycle detection

    for (depth = 1;
      (!refuge) && (!member) && (depth < maxDepth + 1);
      depth++) {
        iterationDepthLimit = startIterations * pown(2., depth);
        zCyclex = zx;
        zCycley = zy;
        for (iterationCount = iterationCount;
          (!refuge) && (!member) && (iterationCount < iterationDepthLimit);
          iterationCount++) {
            z2 = pown(zx, 2) - pown(zy, 2) + cx;
            z2y = 2.0 * zx * zy + cy;
            z2x = z2;
            z2 = 2.0 * (zx * dzx - zy * dzy) + 1;
            dzy = 2.0 *(zx * dzy + zy * dzx);
            dzx = z2;
            zx = z2x;
            zy = z2y;
            refuge = (pown(zx, 2) + pown(zy, 2)) > escapeRadius2;
            member = (zx == zCyclex) && (zy == zCycley);
        }
    }

    depth = depth - 1;

    // case 1: chaotic tile center
    if (!(iterationCount < startIterations * pown( 2., depth) ) ) {
      // distance calculation needless
      distance = 0.0; } // dist = 0 => chaotic point

    // case 2: tile center is refuge (exterior distance estimator)
    if (refuge) {
      // distance calculation
      absz = sqrt(zx * zx + zy * zy);
      distance = 0.25 * 2. * log(absz) * absz / sqrt(dzx * dzx + dzy * dzy);
      cycleCount = iterationCount; // return iteration count instead of cycle count
      }

    // case 3: tile center is member
    if (member) {

      // case 3a: member of cardioid
      if (card) {
        tx = cx + boxSize * 0.5;
        ty = cy + boxSize * 0.5;
        q = pown((tx - 0.25), 2) + ty * ty;
        card1 = q * (q + (tx - 0.25)) < 0.25 * ty * ty;
        tx = cx - boxSize * 0.5;
        ty = cy + boxSize * 0.5;
```

```
        q = pown((tx - 0.25), 2) + ty * ty;
        card2 = q * (q + (tx - 0.25)) < 0.25 * ty * ty;
        tx = cx - boxSize * 0.5;
        ty = cy - boxSize * 0.5;
        q = pown((tx - 0.25), 2) + ty * ty;
        card3 = q * (q + (tx - 0.25)) < 0.25 * ty * ty;
        tx = cx + boxSize * 0.5;
        ty = cy - boxSize * 0.5;
        q = pown((tx - 0.25), 2) + ty * ty;
        card4 = q * (q + (tx - 0.25)) < 0.25 * ty * ty;
        if (card1 && card2 && card3 && card4) {
          distance = -boxSize;         // bigger than box
          cycleCount = 1;              // cycle count of cardioid
        }
        else {
          distance = -boxSize * 0.5; // smaller than box
          cycleCount = 1;            // cycle count of cardioid
        }
      }

      // case 3b: member of circle
      if (circ) {
        tx = cx + boxSize * 0.5;
        ty = cy + boxSize * 0.5;
        circ1 = pown((tx + 1.0), 2) + ty * ty < 0.0625;
        tx = cx - boxSize * 0.5;
        ty = cy + boxSize * 0.5;
        circ2 = pown((tx + 1.0), 2) + ty * ty < 0.0625;
        tx = cx - boxSize * 0.5;
        ty = cy - boxSize * 0.5;
        circ3 = pown((tx + 1.0), 2) + ty * ty < 0.0625;
        tx = cx + boxSize * 0.5;
        ty = cy - boxSize * 0.5;
        circ4 = pown((tx + 1.0), 2) + ty * ty < 0.0625;
        if (circ1 && circ2 && circ3 && circ4) {
          distance = -boxSize;         // bigger than box
          cycleCount = 2;              // cycle count of circle
        }
        else {
          distance = -boxSize * 0.5;   // smaller than box
          cycleCount = 2;              // cycle count of circle
        }
      }

      // case 3c: not member of circle nor member of cardioid
      if (!circ && !card) {
        // determine cycle length
        member = false;
        for (iterationCount = iterationCount, cycleCount = 0;
          (!member) && (cycleCount < maxCycleCount + 2);
          iterationCount++, cycleCount++) {
              z2 = zx * zx - zy * zy + cx;
              z2y = 2.0 * zx * zy + cy;
              z2x = z2;
              z2 = 2.0 * (zx * dzx - zy * dzy) + 1;
              dzy = 2.0 *(zx * dzy + zy * dzx);
              dzx = z2;
              zx = z2x;
              zy = z2y;
              refuge = (pown(zx, 2) + pown(zy, 2)) > escapeRadius2;
              member = (zx == zCyclex) && (zy == zCycley);
        }

        // too long cycle (chaotic) case: cycle count greater than maxCycleCount
        if (cycleCount > maxCycleCount) { distance = 0.; } // dist = 0 => chaotic point

        // go on case 3c: distance calculation (interior distance estimator)
        else {
          zx = zCyclex;
          zy = zCycley;
```

```
        for ( i = 1;
          i <= cycleCount;
          i++) {
            D1rT = 2. *(zx * D1r - zy * D1i);
            D1iT = 2. *(zy * D1r + zx * D1i);
            D2rT = 2. *(zx * D2r - zy * D2i) + 1.;
            D2iT = 2. *(zy * D2r + zx * D2i);
            D3rT = 2. *((zx * D3r - zy * D3i) + (D1r * D1r - D1i * D1i));
            D3iT = 2. *((zx * D3i + zy * D3r) + (2. * D1r * D1i));
            D4rT = 2. *((zx * D4r - zy * D4i) + (D1r * D2r - D1i * D2i));
            D4iT = 2. *((zx * D4i + zy * D4r) + (D1r * D2i + D1i * D2r));
            D1r = D1rT; D1i = D1iT;
            D2r = D2rT; D2i = D2iT;
            D3r = D3rT; D3i = D3iT;
            D4r = D4rT; D4i = D4iT;
            z2 = zx * zx - zy * zy + cx;
            zy = 2.0 * zx * zy + cy;
            zx = z2;
        }
        valA = 1. - (D1r*D1r + D1i*D1i);
        valB = (1 - D1r) * (1 - D1r) + D1i * D1i;
        D1rT = (D2r * (1 - D1r) - D2i * D1i) / valB;
        D1iT = (D2i * (1 - D1r) + D2r * D1i) / valB;
        D2rT = D4r + (D3r * D1rT - D3i * D1iT);
        D2iT = D4i +(D3i * D1rT + D3r * D1iT);
        valB = sqrt(D2rT * D2rT + D2iT * D2iT);
        distance = -valA / (2. * valB);

      } // inner dist calc

    } // not card nor circ case

  } // member case

  // epilog: calculate result flag:

  // case 1: point refugee && box refugee
  if (distance > 0. && distance > sqrt(0.5) * boxSize) {result = 1.; }

  // case 2: point refugee && box mixed
  if (distance > 0. && distance <= sqrt(0.5) * boxSize) {result = 2.; }

  // case 3: point member && box member
  if (distance < 0. && -distance > sqrt(0.5) * boxSize) {result = 3.; }

  // case 4: point member && box mixed
  if (distance < 0. && -distance <= sqrt(0.5) * boxSize) {result = 4.; }

  // case 5: chaotic point
  if (!(distance > 0. && distance > sqrt(0.5) * boxSize) &&
    !(distance > 0. && distance <= sqrt(0.5) * boxSize) &&
    !(distance < 0. && -distance > sqrt(0.5) * boxSize) &&
    !(distance < 0. && -distance <= sqrt(0.5) * boxSize) ) {
      result = 5.; } // chaotic point

  c[iGID] = result;
  d[iGID] = (double) cycleCount;
 }
}";
```

## 2. Definition of Mathematica Functions

The Mathematica functions are defined here, e.g. *decide* and *expand*. In addition, this Mathematica code initializes parameters.

```
(* apply opencl code *)
decide[{list_,startIter_,maxDepth_,boxSize_,level_,levelStep_}]:=Block[{tmp},
  tmp=Join[Transpose[list],{startIter,maxDepth,boxSize,Length[list]}];
  tmp=Apply[rawDecide,tmp];
  tmp=Transpose[tmp];
  {tmp,startIter,maxDepth,boxSize,level,levelStep}
];

(* quadtree generator *)
rawExpand=Compile[ {{list,_Real,1},{boxSize,_Real}},
  Module[{delta=0.25*boxSize},
    {{list[[1]]-delta,list[[2]]-delta,list[[3]],-1},
    {list[[1]]-delta,list[[2]]+delta,list[[3]],-1},
    {list[[1]]+delta,list[[2]]+delta,list[[3]],-1},
    {list[[1]]+delta,list[[2]]-delta,list[[3]],-1}}
  ],
  CompilationTarget->"C",
  RuntimeAttributes->{Listable},
  Parallelization->True,
  RuntimeOptions->"Speed"
];

(* apply quadtree generator *)
expand[{list_,startIter_,maxDepth_,boxSize_,level_,levelStep_}]:={
  Flatten[rawExpand[list,boxSize],1],
  startIter,
  maxDepth,
  boxSize*0.5,
  level+1,
  levelStep
};

(*  different compiled select functions for function saveDecide *)
rawSelector3=Compile[{{list,_Real,2}},
  Select[list,#[[3]]==3.&],
  CompilationTarget->"C",RuntimeOptions->"Speed"
];

rawSelector245=Compile[{{list,_Real,2}},
  Select[list,#[[3]]==2.||#[[3]]==4.||#[[3]]==5.&],
  CompilationTarget->"C",RuntimeOptions->"Speed"
];

rawSelector2=Compile[{{lst,_Real,2}},Select[lst,#[[3]]==2.&],CompilationTarget->"C",RuntimeOptions->"Speed"];
rawSelector4=Compile[{{lst,_Real,2}},Select[lst,#[[3]]==4.&],CompilationTarget->"C",RuntimeOptions->"Speed"];
rawSelector5=Compile[{{lst,_Real,2}},Select[lst,#[[3]]==5.&],CompilationTarget->"C",RuntimeOptions->"Speed"];

rawSorter245[lst_]:=Join[rawSelector5[lst],rawSelector4[lst],rawSelector2[lst]];

(* create individual file name and directory *)
rawFilename[list_,level_,status_,workingDirectory_]:=Module[{hash,file},
  hash=ToString[Hash[First[list],"MD2"]];
  file=FileNameJoin[{workingDirectory,status<>"_"<>ToString[level],hash<>".mx"}];
  If[!(DirectoryQ[DirectoryName[file]]),CreateDirectory[DirectoryName[file]]];
  file
];

(* save individual todo and result "buzz" files *)
saveDecide[{list_,startIter_,maxDepth_,boxSize_,level_,levelStep_}]:=Module[{split,tally,hyper,data,files,file,cpu,hdd},
  (* timing *) split=AbsoluteTime[];
  (* result "buzz" file handling *)
```

```
  tally=list[[All,3]];(* extract box type column from decided list *)
  tally=Join[tally,{1.,2.,3.,4.,5.}];(* housekeeping: get one with zero lengths  *)
  tally=Sort[Tally[tally]][[All,2]]-1;(* count member and refugee boxes *)
  hyper=rawSelector3[list];(* select members / hyperbolic components *)
  hyper=Tally[hyper[[All,4]]]; (* extract and count cycle count column from decided list *)
  data={{startIter,maxDepth,boxSize,level,levelStep},tally,hyper,AbsoluteTime[]};(* collect data *)
  file=rawFilename[list,level,"buzz",globalWD];(* create unique filename *)
  Export[file,data,"MX"];(* save result "buzz" file *)
  (* todo file handling *)
  data=rawSelector245[list];(* select !refugees, !members and unknowns *)
  data=Partition[data,globalFileSize,globalFileSize,{1,1},{}];(* divide large lists *)
  data=Map[rawSorter245,data];(* empiric: sort chaotic, member, refugee *)
  files=Map[rawFilename[#,level+levelStep,"todo",globalWD]&,data];(* create unique filename *)
  data=Map[{#,startIter,maxDepth,boxSize,level,levelStep}&,data];(* collect data *)
  (* timing *) cpu=AbsoluteTime[]-split;split=AbsoluteTime[];
  Map[Export[#[[1]],Compress[#[[2]]],"MX"]&,Transpose[{files,data}]];(* save compressed todo files *)
  If[level>-1,file=rawFilename[list,level,"done",globalWD]];(* housekeeping: init case *)
  (* timing *) hdd=AbsoluteTime[]-split;
  {cpu,hdd,Length[files]}
];

(* get due todo file count *)
getTodoFiles[]:=Block[{tmp},
  tmp=FileNames["*",FileNameJoin[{globalWD,"todo"<>"_"<>"*"}],1];(* get all files *)
  tmp=Map[{DirectoryName[#],#}&,tmp];(* extract dirs *)
  tmp=Split[Sort[tmp],#1[[1]]==#2[[1]]&][[1,All,2]]; (* bin dirs and get first dir *)
  tmp
];

(* number formatter *)
floatForm[float_]:=StringTake[ToString[PaddedForm[Round[float,0.001],{4,3},NumberPadding->{"","0"}]],5];
integerForm[integer_]:=StringTake[ToString[PaddedForm[Round[integer,1],4,NumberPadding->{"0","0"}]],5];

(* procedure for single todo file *)
processTodoFile[file_]:=Module[{split,todo,done,levelStep,timeImport,timeExpand,timeDecide,timeTally,timeExport,numberFiles,logline},
  (* timing *) split=AbsoluteTime[];
  todo=Uncompress[Import[file]];(* read todo file *)
  (* timing *) timeImport=AbsoluteTime[]-split;split=AbsoluteTime[];
  levelStep=todo[[6]];(* extract level step from todo file *)
  todo=Nest[expand,todo,levelStep];(* expand todo file via nested quadtree *)
  (* timing *) timeExpand=AbsoluteTime[]-split;split=AbsoluteTime[];
  done=decide[todo];(* TADA: decide todo file *)
  (* timing *) timeDecide=AbsoluteTime[]-split;
  {timeTally,timeExport,numberFiles}=saveDecide[done];(* post process todo files *)
  (* logging *) logline=DateString[{"Day","-","Hour",":","Minute",":","Second"}]<>
    " l:"<>integerForm[done[[5]]]<>
    " k:"<>integerForm[$KernelID]<>
    " i:"<>floatForm[timeImport]<>
    " m:"<>floatForm[timeExpand]<>
    " g:"<>floatForm[timeDecide]<>
    " c:"<>floatForm[timeTally]<>
    " e:"<>floatForm[timeExport]<>
    " n:"<>integerForm[numberFiles];
  (* logging *)PutAppend[TextString[logline],FileNameJoin[{globalWD,"logfile.txt"}]];
  RenameFile[file,StringReplace[file,"todo"->"done"]];(* move done todo file  *)
];

(* init main kernel *)
Needs["OpenCLLink`"];(* main kernel: load opencl library *)
$HistoryLength = 0;(* main kernel: reduce memory consuption *)

(* global parameters *)
globalWD = "D:\\Mandel3_Data_V1";(* working directory *)
globalFileSize = 500000;(* max file size, approx. size * 2^levelstep < 500 000 *)
gobalTodoSetMax = 1000; (* max cycle count of kernels before restart *)
```

```
preIterations = 1024;
maxDepth = 9;
boxSize = 2.5;
levelStep = 1;

(* House keeping: start and init worker kernels *)

startKernels[] := Block[{},
  LaunchKernels[];
  DistributeDefinitions[rawDecide, clSourceDecide, globalWD];
  ParallelEvaluate[Needs["OpenCLLink`"]];
  ParallelEvaluate[$HistoryLength = 0];
  (* define opencl function from source *) ParallelEvaluate[
  rawDecide = OpenCLFunctionLoad[
    clSourceDecide,
    "rawDecide",
    {{_Real}, {_Real}, {_Real}, {_Real}, _Integer, _Integer, _Real, _Integer},
    16,
    "ShellOutputFunction" -> None,
    "Platform" -> 1,
    "Device" -> Mod[$KernelID, 4] + 1]
  ];
];
```

## 3. Initialisation of Working Files

This Mathematica code initializes the working files. This code is only called once.

```
(* save first todo file *)
todo = {{{-0.75, 1.25, 5., -1}}, preIterations, maxDepth, boxSize, 0, levelStep};
saveDecide[todo];
```

## 4. Mathematica Code for the Main Loop

This Mathematica code represents the main loop. This code can be restarted.

```
(* main loop *)
Do[
  files = getTodoFiles[];
  todoSets = Partition[files, gobalTodoSetMax, gobalTodoSetMax, {1, 1}, {}];
  Print[DateString[], " ***** level ", i, " ***** Total files: ", Length[files]];
  (* logging *)
  logline =
  DateString[{"Day", "=", "Hour", ":", "Minute", ":", "Second"}] <>
    " ************* start new round, " <>
    " counter:" <>
    integerForm[i];
  (* logging *)
  PutAppend[TextString[logline],
  FileNameJoin[{globalWD, "logfile.txt"}]];
  Do[
    startKernels[];
    Print[DateString[], " * Kernel restart * todoset: ", todoSet, "/", Length[todoSets]];
    (* logging *)
    logline = DateString[{"Day", "=", "Hour", ":", "Minute", ":", "Second"}] <> " *************
        restart kernels";
    (* logging *)
    PutAppend[TextString[logline], FileNameJoin[{globalWD, "logfile.txt"}]];
    ParallelMap[processTodoFile[#] &, todoSets[[todoSet]], Method -> "FinestGrained"];
    CloseKernels[];
    Pause[5];
  , {todoSet, Length[todoSets]}]
, {i, 23}];
```

[1] http://en.wikipedia.org/wiki/Mandelbrot_set
[2] http://oeis.org/A098403
[3] http://www.mrob.com/pub/muency/pixelcounting.html
[4] ...
[5] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.2337
[6] http://en.wikipedia.org/wiki/OpenCL
[7] http://forums.wolfram.com/mathgroup/archive/2011/Jun/msg00615.html
[8] https://en.wikipedia.org/wiki/Mandelbrot_set#Distance_estimates
[9] http://mrob.com/pub/muency/distanceestimator.html
[10] http://www.moleculardensity.net/buddhabrot/appendix/2
[11] https://en.wikipedia.org/wiki/Mandelbrot_set#Escape_time_algorithm
[12] https://en.wikipedia.org/wiki/Mandelbrot_set#Cardioid_.2F_bulb_checking
[13] https://en.wikipedia.org/wiki/Koebe_quarter_theorem
[14] http://www.mrob.com/pub/muency/pixelcounting.html
[15] https://en.wikipedia.org/wiki/Fractal_dimension
[16] https://en.wikipedia.org/wiki/Minkowski?Bouligand_dimension
[17] https://en.wikipedia.org/wiki/Mandelbrot_set#Hyperbolic_components