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 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 \((2^{26-19})^2 = 16000\) times larger tessellation. The difference between the upper and lower bound can be halved.

CONTENTS

I. Introduction 1
  A. Problem Statement 1
  B. Solution Statement 1

II. Materials and Methods 2
  A. Hardware and Software 2
  B. Methods 2
    1. Exterior and Interior Distance Estimation 3
    2. Quad Tree Tessellation 4
    3. Performance Considerations 4
    4. Outline of the Quad Tree Algorithm 6
    5. Post Processing of the Tessellation Tree 7

III. Results 7
  A. Tile counts at different depths 7
  B. Area Estimate 7
  C. Time Consumption 7
  D. Parameters of depths 25 and 26 7
  E. Comparison between \(\alpha = 1\) and \(\alpha = 2\) 9

IV. Discussion 9
  A. Source Code 12
    1. GPU Code for the \texttt{decide} Procedure 12
    2. Definition of Mathematica Functions 16
    3. Initialisation of Working Files 18
    4. Mathematica Code for the Main Loop 18
  B. Outline of the Mathematica Code 19

References 19

*thorsten@foerstemann.name

I. INTRODUCTION

A. Problem Statement

An analytical expression of the area of the Mandelbrot set 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 (University 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 \((2^{26-19})^2 = 16000\) times larger tessellation.

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.
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

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 $1024 + 20000 \cdot 2^9 = 10241024$. At tessellation depths 25 and 26 the limit is $1024 + 20000 \cdot 2^9 = 21024$. Specialities of depths 25 and 26 are discussed in section III D.

- Orbits of maximum cycle length $20000 \cdot 2^8 = 5120000$ can be detected if the cycle starts at the latest after $1024 + 20000 \cdot 2^9 = 5121024$ iterations. At tessellation depths 25 and 26 orbits of maximum cycle length $20000 \cdot 2^9 = 20000$ 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 D.

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

- 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 A 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 specific parameters of the distance calculation, i.e. the maximum iteration, it cannot be decided whether the center of the tile is outside or inside the Mandelbrot set. Thus, the distance of the center cannot 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 left 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, it 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 HB2. The yellow square at depth 24 corresponds to figure 9.
implementation is handy. Here, the decide procedure is implemented in OpenCL (refer to A1) and the expand procedure is implemented as a compiled Mathematica function with compilation target C and allowed parallelization (refer to A2).

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 order 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. Specialties 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 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 up this, 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 one file is limited to 100,000. The files are compressed using Mathematica's Compress function. Only files of type 3, 4 and 5 are stored in these files. So, in order to store the whole 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).

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 A1). 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 3.

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 \((2^{26-1})^2 = 33554432^2 = 1125899906842624\) 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 3. 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 II B 1 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 7) are changed, due to performance considerations (refer to section II B 3). Specialties 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 II 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 \(1.50640\) (refer to row 26(1)) and \(1.53121\) (refer to row 26(1)). Corresponding bounds from Y. Fisher and J. Hill amount to \(1.50297\) and \(1.57013\). The difference of 0.06316 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 II D) resulting in about 6\times 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 11B). For more details about the different tile types refer to enumeration in section 11B.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 11A.

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 cardoid of the Mandelbrot set. This lowest row shows the effect of the implementation of the chaotic tiles (refer to section 11B).
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.

<table>
<thead>
<tr>
<th>d(\alpha)</th>
<th>lower bound</th>
<th>upper bound</th>
<th>difference</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0(1)</td>
<td>0.000 00</td>
<td>12.500 00</td>
<td>&lt; 0.01 h</td>
<td></td>
</tr>
<tr>
<td>1(1)</td>
<td>0.000 00</td>
<td>9.375 00</td>
<td>9.375 00</td>
<td>&lt; 0.01 h</td>
</tr>
<tr>
<td>2(1)</td>
<td>0.000 00</td>
<td>7.812 50</td>
<td>7.812 50</td>
<td>&lt; 0.01 h</td>
</tr>
<tr>
<td>3(1)</td>
<td>0.390 63</td>
<td>4.882 81</td>
<td>4.492 19</td>
<td>&lt; 0.01 h</td>
</tr>
<tr>
<td>4(1)</td>
<td>0.878 91</td>
<td>3.857 42</td>
<td>2.978 52</td>
<td>&lt; 0.01 h</td>
</tr>
<tr>
<td>5(1)</td>
<td>1.123 05</td>
<td>3.088 38</td>
<td>1.965 33</td>
<td>&lt; 0.01 h</td>
</tr>
<tr>
<td>6(1)</td>
<td>1.242 07</td>
<td>2.618 41</td>
<td>1.376 34</td>
<td>&lt; 0.01 h</td>
</tr>
<tr>
<td>7(1)</td>
<td>1.335 14</td>
<td>2.299 50</td>
<td>0.964 35</td>
<td>&lt; 0.01 h</td>
</tr>
<tr>
<td>8(1)</td>
<td>1.395 42</td>
<td>2.092 74</td>
<td>0.697 32</td>
<td>&lt; 0.01 h</td>
</tr>
<tr>
<td>9(1)</td>
<td>1.437 04</td>
<td>1.950 36</td>
<td>0.513 31</td>
<td>&lt; 0.01 h</td>
</tr>
<tr>
<td>10(1)</td>
<td>1.462 67</td>
<td>1.851 96</td>
<td>0.389 29</td>
<td>&lt; 0.01 h</td>
</tr>
<tr>
<td>11(1)</td>
<td>1.479 03</td>
<td>1.779 83</td>
<td>0.300 80</td>
<td>0.01 h</td>
</tr>
<tr>
<td>12(1)</td>
<td>1.489 55</td>
<td>1.726 36</td>
<td>0.236 81</td>
<td>0.02 h</td>
</tr>
<tr>
<td>13(1)</td>
<td>1.496 08</td>
<td>1.685 55</td>
<td>0.189 48</td>
<td>0.03 h</td>
</tr>
<tr>
<td>14(1)</td>
<td>1.500 10</td>
<td>1.654 05</td>
<td>0.153 95</td>
<td>0.04 h</td>
</tr>
<tr>
<td>15(1)</td>
<td>1.502 61</td>
<td>1.629 25</td>
<td>1.26 64</td>
<td>0.05 h</td>
</tr>
<tr>
<td>16(1)</td>
<td>1.504 15</td>
<td>1.609 49</td>
<td>0.105 34</td>
<td>0.09 h</td>
</tr>
<tr>
<td>17(1)</td>
<td>1.505 09</td>
<td>1.593 57</td>
<td>0.088 47</td>
<td>0.21 h</td>
</tr>
<tr>
<td>18(1)</td>
<td>1.505 65</td>
<td>1.580 61</td>
<td>0.074 96</td>
<td>0.55 h</td>
</tr>
<tr>
<td>19(1)</td>
<td>1.505 98</td>
<td>1.569 98</td>
<td>0.064 00</td>
<td>1.53 h</td>
</tr>
<tr>
<td>20(1)</td>
<td>1.506 17</td>
<td>1.561 17</td>
<td>0.055 01</td>
<td>4.47 h</td>
</tr>
<tr>
<td>21(1)</td>
<td>1.506 28</td>
<td>1.553 84</td>
<td>0.047 57</td>
<td>13.38 h</td>
</tr>
<tr>
<td>22(1)</td>
<td>1.506 34</td>
<td>1.547 70</td>
<td>0.041 36</td>
<td>41.83 h</td>
</tr>
<tr>
<td>23(1)</td>
<td>1.506 37</td>
<td>1.542 51</td>
<td>0.036 14</td>
<td>135.65 h</td>
</tr>
<tr>
<td>24(1)</td>
<td>1.506 39</td>
<td>1.538 12</td>
<td>0.031 72</td>
<td>415.25 h</td>
</tr>
<tr>
<td>25'1</td>
<td>1.506 40</td>
<td>1.534 40</td>
<td>0.028 00</td>
<td>∼ 170 h</td>
</tr>
<tr>
<td>26'1</td>
<td>1.506 40</td>
<td>1.531 21</td>
<td>0.024 81</td>
<td>∼ 550 h</td>
</tr>
</tbody>
</table>

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 III B. 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 $16000 \times$ larger tessellation just leads to halved area bounds.

The tile counts listed in table I are predestined for further fractal dimension analysis [6] based on box counting dimension or Minkowski-Bouligand dimension [10]. 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 [H13].
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].
Appendix A: Source 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.

```c
(* 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 parameter
        long maxCycleCount = 20000; // maximum length of detected cycles, empiric parameter
        double q = 0.0; // interim value
        bool refuge = false; // cx ,cy is refugee
        bool member = false; // cx ,cy is member of cardioid
        bool card = false; // cx ,cy is member of circle
        bool card1 = false; bool card2 = false;
        // cardioid and circle test
        if (!refuge) {
            q = pown(((cx - 0.25) + cy * cy);
            card = q * (q + (cx - 0.25)) < 0.25 * cy * cy;
        }
    }
}
```

\[ \text{circ} = \text{pown}((\text{cx} + 1.0), 2) + \text{cy} \times \text{cy} < 0.0625; \]
\[
\text{member} = (\text{card} \mid\mid \text{circ});
\]

// pre iteration

for (iterationCount = 0; (!member) && (!refuge) && (iterationCount < startIterations);)
    iterationCount++ {
        z2 = \text{pown}(\text{zx}, 2) - \text{pown}(\text{zy}, 2) + \text{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 = (\text{pown}(\text{zx}, 2) + \text{pown}(\text{zy}, 2)) > \text{escapeRadius}^2;
    }

// post iteration and cycle detection

for (depth = 1; (!refuge) && (depth < maxDepth + 1);
    depth++) {
    iterationDepthLimit = startIterations * \text{pown}(2.0, \text{depth});
    zCyclenx = zx;
    zCycleyn = zy;
    for (iterationCount = iterationCount;
        (!refuge) && (!member) && (iterationCount < iterationDepthLimit);
        iterationCount++) {
        z2 = \text{pown}(\text{zx}, 2) - \text{pown}(\text{zy}, 2) + \text{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 = (\text{pown}(\text{zx}, 2) + \text{pown}(\text{zy}, 2)) > \text{escapeRadius}^2;
        member = (zx == zCyclenx) && (zy == zCycleyn);
    }
    depth = depth - 1;

    // case 1: chaotic tile center
    if ( ((iterationCount < startIterations * \text{pown}(2.0, \text{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 * 2.0 * log(above) * 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 = \text{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 * dzx + zy * dzy);
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 = 0.0; } // dist = 0 => chaotic point

  // go on case 3c: distance calculation (interior distance estimator)
  else {
    zx = zCyclex;
    zy = zCycley;
  }
}
```c
for ( i = 1; 
     i <= cycleCount; 
     i++) {
    D1rT = 2.0 * (zx * D1r - zy * D1i);
    D1iT = 2.0 * (zy * D1r + zx * D1i);
    D2rT = 2.0 * (zx * D2r - zy * D2i) + 1.0;
    D2iT = 2.0 * (zy * D2r + zx * D2i);
    D3rT = 2.0 * ((zx * D3r - zy * D3i) + (D1r * D1r - D1i * D1i));
    D3iT = 2.0 * ((zx * D3i + zy * D3r) + (2.0 * D1r * D1i));
    D4rT = 2.0 * ((zx * D4r - zy * D4i) + (D1r * D2r - D1i * D2i));
    D4iT = 2.0 * ((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 = 1.0 - (D1r + D1i) * (D1r - D1i) + D1i * D1i;
    valB = (1.0 - D1r) * (1.0 - D1r) + D1i * D1i;
    D1rT = (D2r * (1.0 - D1r) - D2i + D1i) / valB;
    D1iT = (D2i * (1.0 - D1r) + D2r - D1i) / valB;
    D2rT = D4r + (D3r - D1rT - D3i * D1iT);
    D2iT = D4i + (D3i * D1rT + D3r - D1iT);

    valB = sqrt(D2rT * D2rT + D2iT * D2iT);
    distance = -valA / (2.0 * valB);
}

} // inner dist calc

} // not card nor circ case

} // member case

// epilog: calculate result flag:

// case 1: point refugee && box refugee
if (distance > 0.0 && distance > sqrt(0.5) * boxSize) { result = 1.; }

// case 2: point refugee && box mixed
if (distance > 0.0 && distance <= sqrt(0.5) * boxSize) { result = 2.; }

// case 3: point member && box member
if (distance < 0.0 && distance <= sqrt(0.5) * boxSize) { result = 3.; }

// case 4: point member && box mixed
if (distance < 0.0 && distance <= sqrt(0.5) * boxSize) { result = 4.; }

// case 5: chaotic point
if (!(distance > 0.0 && distance > sqrt(0.5) * boxSize) &&
    !(distance > 0.0 && distance <= sqrt(0.5) * boxSize) &&
    !(distance < 0.0 && distance > sqrt(0.5) * boxSize) &&
    !(distance < 0.0 && distance <= sqrt(0.5) * boxSize)) {
    result = 5.; } // chaotic point

} // cycleCount

} // default:
```
2. Definition of Mathematica Functions

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

```mathematica
(* 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},
], 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[{list_, Real, 2}, Select[list, #[[3]]==2. &], CompilationTarget->"C", RuntimeOptions->"Speed"
];

rawSelector4=Compile[{list_, Real, 2}, Select[ list, #[[3]]==4. &], CompilationTarget->"C", RuntimeOptions->"Speed"
];

rawSelector5=Compile[{list_, Real, 2}, Select[list, #[[3]]==5. &], CompilationTarget->"C", RuntimeOptions->"Speed"
];

rawSorter245[list_]:= Join[rawSelector5[list], rawSelector4[list], rawSelector2[list]]

(* create individual file name and directory *)
rawFilename[list_, level_, status_, workingDirectory_, _]:= Module[{hash, file},
hash=ToSymbol[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 *)
```
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.