Documentation

The calculator is used by creating an instance of the class “ColorCalculator” from the namespace Mandelfash as the below example in C#:

```
using Mandelfash;

ColorCalculator calculator = new ColorCalculator();
```

Math Methods:

void startCalculator(String ^x, String ^y, String ^z):
This is where the magic happens. This method expects the coordinates to be given in string format to keep it easier for conversion between different types of floating point that might be necessary inside and outside the calculator. This is where most of the optional method’s properties will take effect. If multi-threading is set to false, you will only exit the ColorCalculator when the calculation ends.

If the multi-threading is set to true this method will create a thread for the calculator with the given values, the returning data from the thread can then be called back with the method getResults() explained below. The thread created will be stored in order to make sure that doesn’t matter how many are created and which ends first. The data’s order will never be lost.

For performance concerns consider that each call of this method with multi-threading set to active, creates a new thread.

List<List<int>^>^ getResults():
This method will go over all the created threads in order, await foreach to return its value and afterwards return an ordered list of lists of three “int”s representing the three colors from RGB or BGR or if with.

For performance concerns consider that each time this method is called the thread list will be cleared so this method can be used to limit the multi-threading inside a big project.

Optional Methods:

void setLoopLimit(int loopLimit) [Default: 100.000]:
This method is used to change the loopLimit. The minimum value is 20 and the maximum is 100.000. A bigger loop limit means bigger precision but slower execution while the opposite is also true.

void setBlackBypass(bool active) [Default: true]:
With black bypass active the calculator will assume that some pixels are black based on some predefined patterns already found when Z = 0. This method might decrease the execution time with no clear change to precision.

void setMultithreading(bool active) [Default: true]:
If multi-threading is set to false the calculator won't create any extra threads for the math calculations. For more details see the method "startCalculator".

void setInvertedRGB(bool active) [Default: false]:
If set the returning values will have the first value of the RGB lists as blue and the third as red.

void setBlackLoopStatus(bool active) [Default: false]:
If set the calculator will verify if there is an adjacent pixel in any direction (x+, x-, y+, y-, z+, z-) that is black; before checking the current pixel. If it finds a black pixel on the already calculated adjacent pixels the current calculator will have its loopLimit set to ⅕ of the set standard value, with a minimum of 20. As there is a clear tendency for the black pixels to be the ones that take more time to be calculated most of time, so this might give a great increase of performance.

Be advised that the black loop works expecting a logical sequence in the values given to the calculator. If let's say you send the x value as 2, 5 and 1: it will assume that 2 is adjacent to 5 which is adjacent to 1, so to make sure the method works properly x needs to be sent as 1 2 any in-betweens and 5, unless of course you want 2 to be adjacent to 5.

void saveBenchmark():
When this method is called, the dll will save a file named "mandelfash.log" or edit an existing one on the current folder it is being executed. In each line added (if already existing file) it will be written the start time when this calculator object was instantiated,

the time this method was called and all relevant configurations for performance measuring.

void setBackgroundColorCode(int newCode) [Default: 0]:
This method changes the value of the first possible color, with ‘white’ as the default. The first color will be the background. This method accepts an integer from 0 to 44. Anything below 0 or above 44 will result in an exception.

void setBackgroundColorMode(int newCode) [Default: 2]:
This method changes the value that will be considered the limit of the background’s color on the breakout. If set to 0, the breakout base value will be 6, if set to 1, the breakout value will be 27, and if 2 it will be 40. The biggest change is that many colors of the mandelfash will disappear on the background with its value set to 27 or 40. Any value different from 0, 1 or 2 set to this method will result in an exception. Note: When set to 0, ‘blue skin’ will appear around the edges of babies and will be integrated into the fractal color.

Example using the DLL with Visual Basic:

```
Imports Mandelfash

Module ColorCalculator

        Sun Main()

                Dim  calc As New ColorCalculator()

                calc.setBlackBypass(False)

                calc.setMultithreading(True)

                calc.setInvertedRGB(False)

                calc.setBlackLoopStatus(False)

                calc.saveBenchmark()

                calc.setBackgroundColorMode(0) ‘ blue skin

                ‘ ######################################################

                calc.startCalculator(“-1.814”, “-0.0000000101”, “0.0000410”)

                ‘ ######################################################

                Dim results As List(Of List(Of Integer))

                Dim a, b, c As String
```

```
Results = calc.getResults() ‘ Retrieving three RGB color codes for that one pixel

a = results.Item(0).Item(0).ToString()
Console.Write(a)

b = results.Item(0) ).Item(1).ToString()
Console.Write(b)

c = results.Item(0) ).Item(2).ToString()
Console.Write(c)
```