# CS/SE 4F03 Final Project: Mandelbox Video

Ned Nedialkov

11 March — 6 April 2015

## 1 Introduction

The *Mandelbox* appeared in 2010 in *fractalforums.com*:
http://www.fractalforums.com/ifs-iterated-function-systems/amazing-fractal/
For an introduction to Mandelboxes, see e.g.
http://en.wikipedia.org/wiki/Mandelbox
http://blog.hvidtfeldts.net/index.php/2010/04/folding-space-the-mandelbox-fractal/
http://www.fountainware.com/Funware/Mandelbrot3D/Mandelbrot3d.htm

The goal of this project is to use parallel computing to produce efficiently high-resolution images of Mandelboxes and "stitch" them together into a video. You can use MPI, OpenMP, and/or GPU programming.

You are given prototype code (developed by Thomas Gwosdz and improved by Ned Nedialkov) which you can improve further and parallelize. You can download it from http://www.cas.mcmaster.ca/~nedialk/COURSES/4f03/private/MandelBoxCode.zip. (Typing `make` creates the executable `mandelbox`.)

This code is based on *ray marching* (also called *ray casting*).

## 2 Background

### 2.1 Ray marching

This is a technique used in computer graphics when the shape of an object to be visualized is not known by explicit formulas. From a camera point, we cast a ray and find the point where it hits the surface of the object. At this point, we compute a gradient. The point and its gradient are the input to a coloring scheme.

Given a point $p \in \mathbb{R}^3$ on a ray, a *distance estimator* (DE) is a function that returns the distance to the nearest point of the object to $p$. Denote this distance by $d_p$. Then we can move along this ray $d_p$ distance. We keep marching along the ray (so the term ray marching) until we

- reach a point $q$ such that the distance $d_q$ to the nearest point of the object is below some given tolerance $\epsilon > 0$, or
- the total distance traveled is $> d_{\max}$, which is a given constant, or
- a maximum number of iterations is reached.

**Algorithm.** Ray-Marching$(p, r, d_{\text{max}}, m, \epsilon)$
$d_{\text{total}} = 0$ % total distance
for $j = 1 : m$
    % move along ray $r$
    $v = p + d_{\text{total}}\, r$
    $d_{\text{est}} =$ Distance-Estimator$(v)$
    if $d_{\text{est}} < \epsilon$ or $d_{\text{total}} > d_{\text{max}}$ then break
    $d_{\text{total}} = d_{\text{total}} + d_{\text{est}}$

Figure 1: $p$ is a starting point, $r$ is ray direction, and $m$ is the maximum number of ray-marching steps.

A basic ray marching algorithm is given in Figure 1. See also the file `raymarching.cc`.

A good explanation of ray marching is
http://www.iquilezles.org/www/articles/terrainmarching/terrainmarching.htm

A key for implementing ray marching is the distance estimator; see §2.3.

## 2.2 Mandelbox map

Like the Mandelbrot set, a Mandelbox is calculated by applying an iterative formula to every point in space. A point is part of a Mandelbox if it does not escape to infinity.

Mapping a vector $z \in \mathbb{R}^3$ is done through the algorithm Mandelbox-Map in Figure 2, which uses the algorithms in Figures 3 and 4.

**Algorithm.** Mandelbox-Map$(z, r_{\text{min}}, r_{\text{fixed}}, s, c)$
$z \leftarrow$ Box-Fold$(z)$
$z \leftarrow$ Sphere-Fold$(z, r_{\text{min}}, r_{\text{fixed}})$
$z \leftarrow sz + c$

Figure 2: $s$ is a scale factor, e.g. 2, but can also be negative; $c \in \mathbb{R}^3$.

**Algorithm.** Sphere-Fold$(z, r_{\text{min}}, r_{\text{fixed}})$
if $\|z\|_2 < r_{\text{min}}$ then
    $z \leftarrow (r_{\text{fixed}}/r_{\text{min}})^2 z$
else if $\|z\|_2 < r_{\text{fixed}}$ then
    $z \leftarrow (r_{\text{fixed}}/\|z\|_2)^2 z$

Figure 3: Typically $r_{\text{min}} = 0.5$ and $r_{\text{fixed}} = 1$; $\|z\|_2 = \sqrt{z_1^2 + z_2^2 + z_3^2}$.

For more information, see e.g.
http://en.wikipedia.org/wiki/Mandelbox
https://sites.google.com/site/mandelbox/what-is-a-mandelbox.

**Algorithm.** Box-Fold($z$)
for $i \leftarrow 1:3$
    if $z_i > 1$ then
        $z_i \leftarrow 2 - z_i$
    else if $z_i < -1$ then
        $z_i \leftarrow -2 - z_i$

Figure 4: Box fold algorithm

### 2.3 Distance estimator

The algorithm in Figure 2.3 implements the distance estimator from
https://github.com/rudi-c/mandelbox370/blob/gh-pages/explorer.html
For discussions on distance estimation, see
http://www.fractalforums.com/3d-fractal-generation/a-mandelbox-distance-estimate-formula/msg14893/#msg14893
http://blog.hvidtfeldts.net/index.php/2011/11/distance-estimated-3d-fractals-vi-the-mandelbox/.

**Algorithm.** Distance-Estimator($z, s, r_{\text{min}}, r_{\text{fixed}}, n, e$)
$d_{\text{factor}} \leftarrow 1$
$c_1 = |s - 1|$
$c_2 = |s|^{1-n}$
for $i \leftarrow 1:n$
    $z \leftarrow$ Mandelbox-Map($z, r_{\text{min}}, r_{\text{fixed}}, s, z$)
    if $\|z\|_2 < r_{\text{min}}$ then
        $d_{\text{factor}} \leftarrow (r_{\text{fixed}}/r_{\text{min}})^2 \, d_{\text{factor}}$
    else if $\|z\|_2 < r_{\text{fixed}}$ then
        $d_{\text{factor}} \leftarrow (r_{\text{fixed}}/\|z\|_2)^2 \, d_{\text{factor}}$
    $d_{\text{factor}} \leftarrow |s| d_{\text{factor}} + 1$
    if $\|z\|_2 > e$ break
return $(\|z\|_2 - c_1)/d_{\text{factor}} - c_2$

Figure 5: $n$ is maximum number of iterations, $e$ is escape time. Note that we iterate Mandelbox-Map with $c = z$.

The algorithms in this and in the previous subsection are implemented in the function DE in mandelboxde.cc; see also distance_est.cc.

## 3 Input file

The input to `mandelbox` is given as a text file. These parameters are obtained in `getparams.c`. An example of such a file is given in Figure 6. The resulting image is in Figure 7.

```
# CAMERA
# location  x,y,z   (7,7,7)
14.0 8.0 10.0
# look_at (target)  x,y,z
0 0 0
# up vector x,y,z;  (0, 1, 0)
0 1 0
# field of view     (1)
1.1
# IMAGE
# width height
3840 2160
# detail level, the smaller the more detailed (-3)
-3.5
# MANDELBOX
# scale, rMin, rFixed  (2 0.5 1)
2.0 0.5 1
# max number of iterations, escape time
18 100
# COLORING
# type 0 or 1
1
# brightness
1.2
# super sampling anti aliasing 0 = off, 1 = on
0
# IMAGE FILE NAME
image.bmp
```

Figure 6: Parameters file. A comment starts with `#`. Suitable default values are given in `(...)`.

- The meaning of the first three vectors, *location*, *target*, and *up* can be seen from Figure 8. The field of view is related to the angle in this figure.
- The width and height of the image are given in pixels.
- The detail level relates to the $\epsilon$ in the ray marcher (see `eps` in `raymarching.cc`).
- The `scale`, `rMin`, and `rFixed` values are the $s$, $r_{\text{min}}$, and $r_{\text{fixed}}$ in the Mandelbox-Map.
- `max number of iterations` is the largest number $n$ of iterations allowed in the distance estimator, Distance-Estimator, and `escape time` is the escape time $e$ in it.
- There are two coloring types, 0 and 1. You are free to create your own coloring scheme.
- You can adjust the brightness of the image by changing the `brightness` parameter.

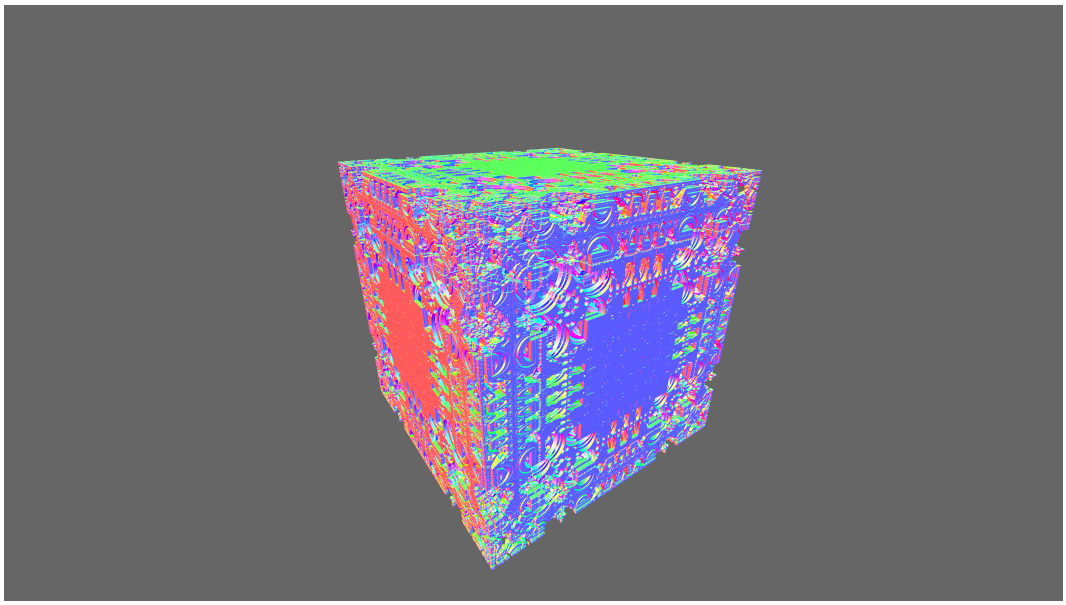

Figure 7: Image produced with the input file in Figure 6.

- If super sampling is set, 1, then the image is smoother, but also more expensive to compute than with super sampling turned off, 0.
- The name of the output image file is the last parameter. The image is in BMP (bitmap) format.

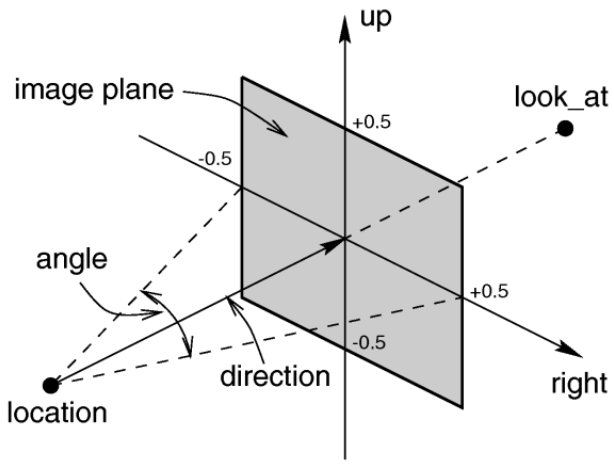


Figure 8: Camera location, target, angle (field of view)

# 4 Project

## 4.1 Rules

You can work in groups of at most four. If you are a graduate student or taking this course at the 600-level, you should work in a group of at most two.

## 4.2 Frames

Each frame must be at least $3840 \times 2160$. (This is also called 4K or Ultra HD, http://en.wikipedia.org/wiki/4K_resolution.) For 1 second of video, you need to have 30 frames.

To create smaller files, you can use `convert` to convert the `bmp` files to `jpg` format. To put the files into a video, you can use `ffmpeg`. See http://www.imagemagick.org/index.php.

Your video should start with a title as shown in Figure 9. Start from outside the object and move towards it.

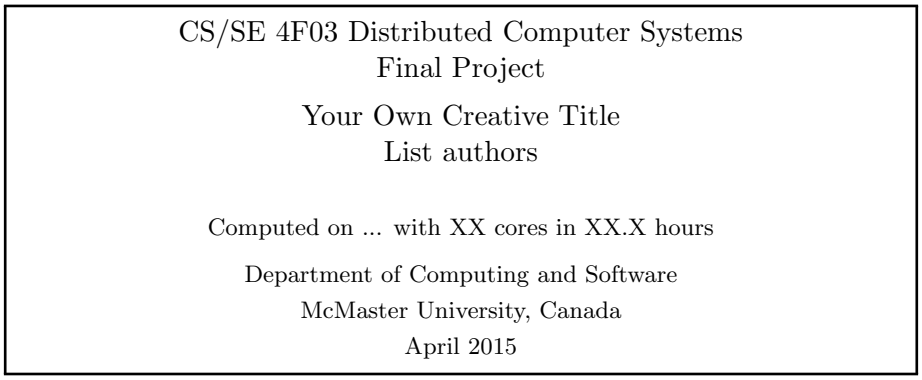


Figure 9: Title for the video

## 4.3 Submit

- (50%) Your video to SVN under subdirectory `Project/video` or to Youtube and email me the link.

  Your percentage will be determined from the formula

$$\frac{5}{6} \min \left\{ \frac{\text{length in seconds}}{4}, 60 \right\}.$$

- (50%) A report (hard copy) containing
  - (10%) description of your parallel approach and implementation. Discuss why and how you have parallelized certain portions of the code and how you have balanced the work. Also, describe how you "navigate" through the object.
  - (5%) summary of the computations
    - machines used

    - total number of cores
    - total computation time
    - number of frames
    - seconds per frame
  - (10%) well-documented source code. If it takes a file with input parameters different from the provided, then describe this file in your report.
  - (25%) This will depend on how well you have parallelized the code. Provide sufficient detail supporting your claims on efficiency. For example, computing 28–30 times faster on 32 cores would give you 30%.

    Also try to compute frames on Sharcnet.

- Your source code under directory `Project/code`. There must be a makefile, such that when `make` is typed, the executable `mandelbox` is created.

  - When
    `make run-parallel`
    is typed, `mandelbox` should produce in parallel a sequence of files containing images. Each line of the output should be `filename cputime`. For example

    ```
    f0001.bmp    90.4
    f0002.bmp    91.2
    ...
    f5400.bmp    50.1
    ```

    where the CPU time is in seconds.
  - When
    `make run-serial`
    is typed, it should produce them sequentially and produce the same output as above.

  Finally, you can obtain up to 15% bonus depending on the quality of the final result. This bonus will be at the discretion of the TA's and the instructor.

# 5 Final notes

This project requires a *lot of creativity.* In particular, it is challenging to go inside the box and move through "holes" in it. Also, it is challenging to produce more interesting coloring schemes. (For ideas about such videos, search on youtube.)

You can experiment with various values for

- camera position, target and up
- field of view
- detail level
- scale, and the radiuses $r_{\text{min}}$ and $r_{\text{fixed}}$

- maximum number of iterations $n$
- escape time

You are encouraged to create your own coloring scheme.

- You may want to profile first the code to discover the most time consuming parts and where to optimize and parallelize.
- You are free to improve any of the algorithms that are implemented. You are also encouraged to try your own.
- Assume that in average you compute serially a frame in 60 seconds. Then to compute a 4 minute video the computing time would be

$$4 \times 60 \times 30 \times 60/3600 = 120 \text{ hours or } 5 \text{ days.}$$

- At the beginning, you may want to compute quickly low resolution frames (say $800 \times 600$) to obtain insights about your final result.
- Your program *must scale* well. For example, if one changes the resolution to $4096 \times 2160$ (movie projection industry standard) and/or increases the number of frames, your program must compute very efficiently.

**Acknowledgments.** Thomas Gwosdz produced the initial version of the code. Without his help and dedication, this project was not going to happen.