CS/SE 4F03 Final Project
Rendering 3D Fractals on a GPU

Ned Nedialkov

10 March – 5 April 2016

1 Introduction

The *Mandelbulb* and *Mandelbox* are three-dimensional fractals. The former appeared in 2009 and the latter in 2010. For details, see

Mandelbulb

- https://en.wikipedia.org/wiki/Mandelbulb
- http://www.skytopia.com/project/fractal/mandelbulb.html
- http://www.fountainware.com/Funware/Mandelbrot3D/Mandelbrot3d.htm

Mandelbox

- http://en.wikipedia.org/wiki/Mandelbox
- http://www.fractalforums.com/ifs-iterated-function-systems/amazing-fractal/
- http://blog.hvidtfeldts.net/index.php/2010/04/folding-space-the-mandelbox-fractal/

The goal of this project is to use parallel GPU computing to produce efficiently images of 3D fractals and "stitch" them together into a video.

You are given prototype serial code developed by T. Gwosdz and N. 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 produces Mandelbox images. It 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 $> d_{\text{max}}$, which is a given constant, or
- a maximum number of iterations is reached.

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

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 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

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 4 and 3. For more information, see e.g.
http://en.wikipedia.org/wiki/Mandelbox
https://sites.google.com/site/mandelbox/what-is-a-mandelbox.

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. 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 3: Box fold algorithm

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

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,
and
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. The needed 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   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
1080 1080
# detail level, the smaller the more detailed (-3)
-4
# MANDELBOX
# scale, rMin, rFixed  (2 0.5 1)
2  0.5 1
# max number of iterations, escape time
18 100
# COLORING
# type 0 or 1
1
# brightness
1.2
# 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 in pixels.
- The detail level relates to the ϵ in the ray marcher (see `eps` in `raymarching.cc`).
- The `scale`, `rMin`, and `rFixed` values are the s, r_{min}, and r_{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.

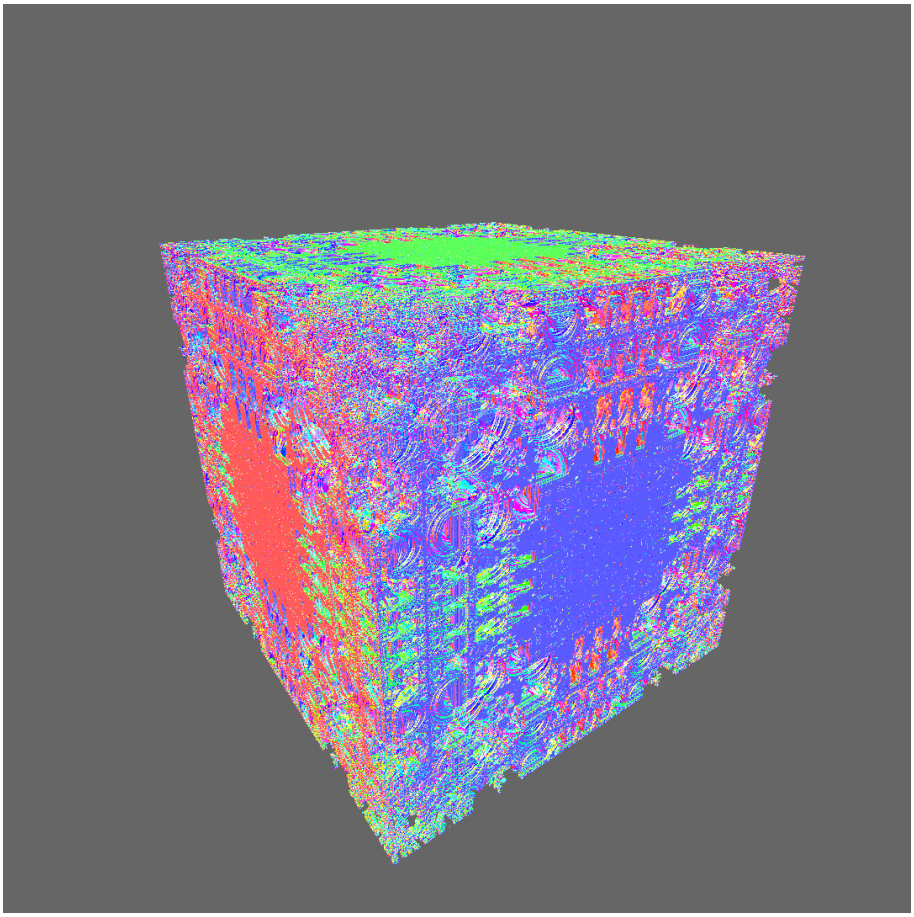

Figure 7: Image produced with the input file in Figure 6.

- You can adjust the brightness of the image by changing the `brightness` parameter.
- The name of the output image file is the last parameter. The image is in BMP (bitmap) format.

4 Project

4.1 Logistics

- You are allowed to work in teams of at most four.
- If you leave this project to the last week or so, chances of finishing it are very small. In addition, if you compute a frame in 3 seconds in average, you will need at least 6 hours for computing 7,200 frames (see below).

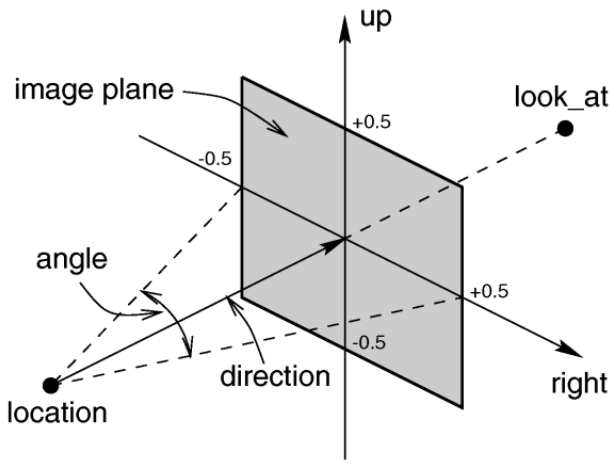

Figure 8: Camera location, target, angle (field of view)

4.2 Accelerating the code on a GPU

You have to modify the given code such that it runs efficiently on a GPU and using OpenACC. Some hints follow.

- It will be easier if you replace the `vec3` class by a structure and the member functions by inline functions or macros. For your convenience, an implementation using macros is http://www.cas.mcmaster.ca/~nedialk/COURSES/4f03/private/vector3d.h.
- Remove all global variables.
- Make the code work correctly using OpenMP. Then converting it to use OpenACC will be easier than converting to OpenACC directly.
- Please ask questions about this code conversion as early as possible so you get it working.

4.3 Computing a Mandelbulb

The given code computes a Mandelbox. To compute a Mandelbulb, you need to add a new distance estimator. For your convenience, we provide such: http://www.cas.mcmaster.ca/~nedialk/COURSES/4f03/private/mandelbulb_dist_est.cc.
It implements the algorithm at
http://blog.hvidtfeldts.net/index.php/2011/09/distance-estimated-3d-fractals-v-the-mandelbulb-different-de-approximations/.

Input file The input file for a Mandelbulb is slightly different; see Figure 9. With the given distance estimator and this file, you should obtain the image in Figure 10.

```
# CAMERA
# location  x,y,z   (7,7,7)
1.2 .26 .2
# look at   x,y,z
0 0 0
# up vector x,y,z;  (0, 1, 0)
0 1 0
# field of view     (1)
2.0
# IMAGE
# width height
1080 1080
# detail level, the smaller the more detailed (-3.5)
-3.0
# MANDELBULB
# ignore the first number, 0.
# the second and third numbers are escape (or bailout) time and power
0  4.0 8.0
# ignore the second number; the first number is the max number of iterations
100 0
# COLORING
# type 0 or 1
1
# brightness
1.2
# IMAGE FILE NAME
imageBulb.bmp
```

Figure 9: Mandelbulb parameters file

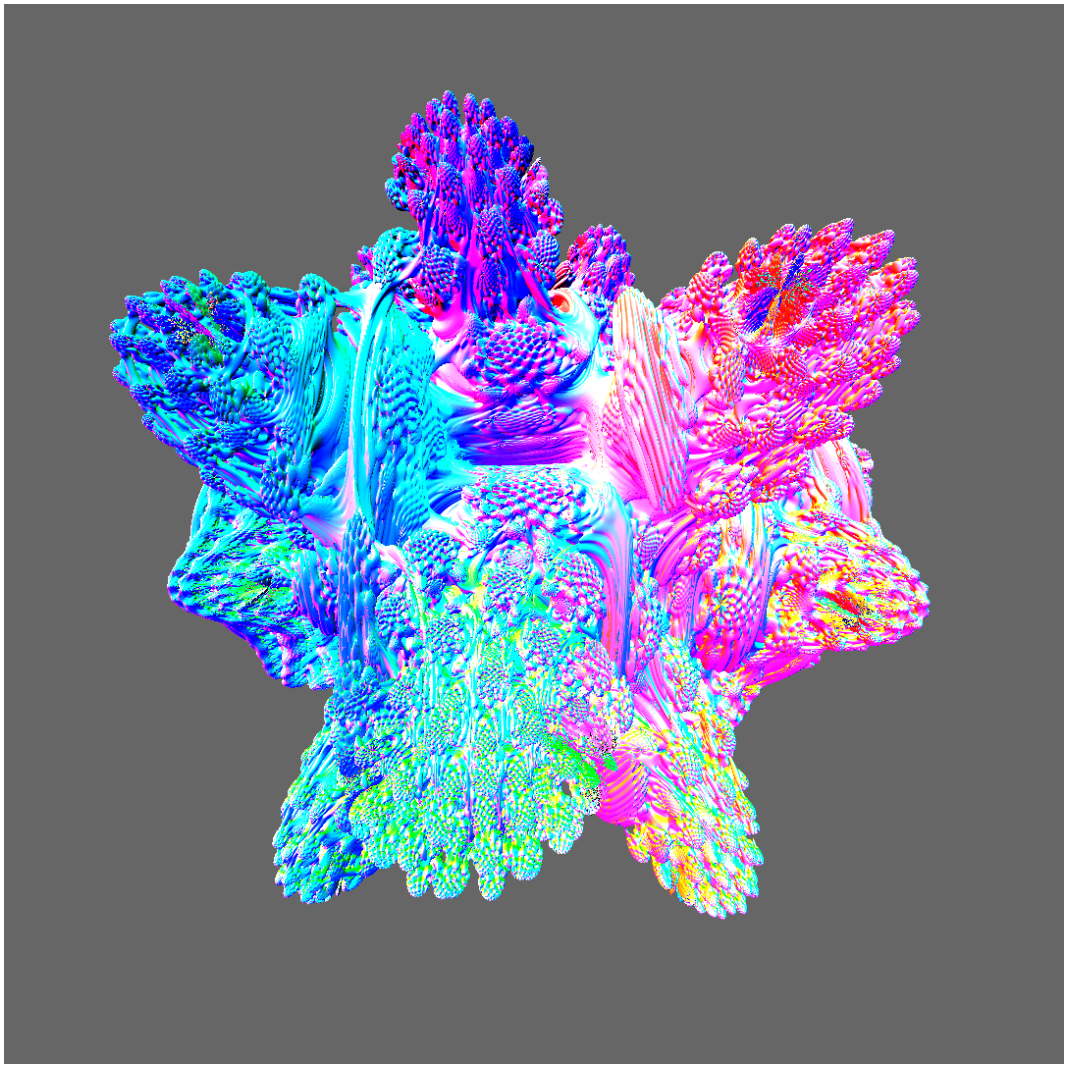

Figure 10: Mandelbulb with power 8; produced with with the input file in Figure 9.

4.4 Frames

You need to compute frames with resolution of (at least) 1920×1080. For 1 second of video, you need to have 30 frames. For a 4 minute video, the number of frames is $7,200$.

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`.

4.5 Video

The goal is to explore a Mandelbox or a Mandelbulb (of your choice) by moving the camera around. The main challenge is to develop and implement an algorithm such that the navigation is with minimal or no user effort.

Your video should start with a title as shown in Figure 11. Start from outside the object and move towards it. Then try to move through "holes" or empty spaces and try to avoid going through surfaces.

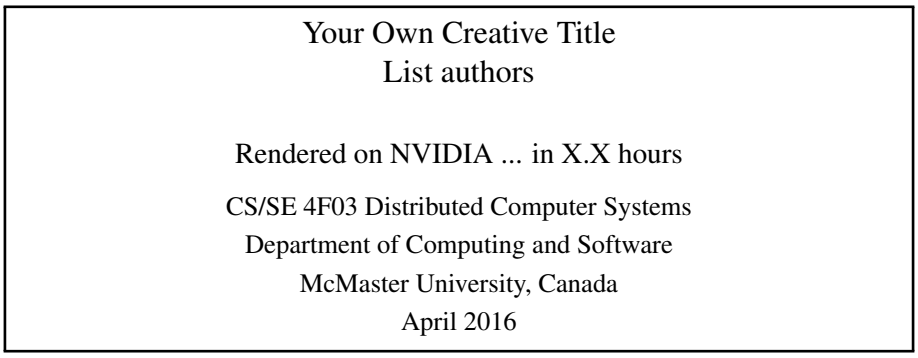

Figure 11: Title for the video. (Do not put your student numbers on it.)

4.6 Submit

- All your source code (including the makefile) to SVN under subdirectory `Project`.
- (10%) When

  `make mandelbulb`

  is typed, the code should be compiled to work on K40 (tesla).

  Then by executing

  `./mandelbulb paramsBulb.dat`,

  where `paramsBulb.dat` is the file from Figure 9, the image in Figure 10 should be produced. You are free to implement better coloring.
- (45%) Your video to SVN or to Youtube and email the link to me and to the TAs.

  Your percentage will be determined from the formula

$$\frac{3}{4}\min\left\{\frac{\text{length in seconds}}{4},\ 60\right\}.$$

- (45%) A report (hard copy) containing
  - (10%) description of your parallelization. Discuss how you have parallelized certain portions of the code and the overall implementation. For example, you may precompute the parameters for each frame and then render them independently on more than one GPU.

- (20%) Describe in sufficient detail your algorithm for generating the frames, starting from the initial time until the end of the frame generation. That is, how you select the parameters for each frame.
- (5%) Computing a Mandelbox is generally slower than a Mandelbulb. Explain why. Include here a summary of your computations: GPU(s) used, number of frames, and seconds (in average) per frame.
- (5%) Well-documented source code. Insert appropriate comments and descriptions of input and output arguments.
- (5%) An explanation of how one can reproduce your final result. The best would be if by typing `make video` the final video file is generated, without any other manual intervention.

4.7 Bonus

- (3%) If your perform all floating-point computations in single precision, which should be faster than the present double precision computation.
- (7%) If you parallelize using MPI and OpenACC together. Then you can render on more then one server at the same time. Submit a description of how you have done it.
- (20%) If your computation navigates automatically through the objects.

5 Final notes

This project requires a *lot of creativity*. In particular, it is challenging to go inside these objects and move through "holes" in them. Also, it is challenging to produce more interesting coloring schemes.

- You can experiment with various values for
  - camera position, target and up
  - field of view
  - detail level
  - scale, and the radiuses r_{min} and r_{fixed} (Mandelbox)
  - power (Mandelbulb)
  - maximum number of iterations n
  - escape time
- 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.
- At the beginning, you may want to compute quickly low resolution frames (say 800×600) to obtain insights about your final result.
- Your program *must scale* well. For example, if one changes the resolution to 4096×2160 (movie projection industry standard) and/or increases the number of frames, your program must compute very efficiently.