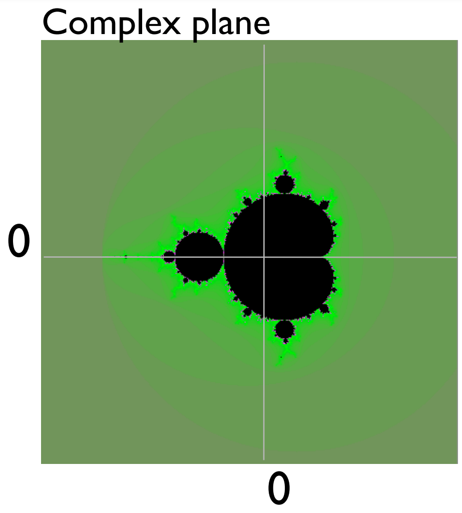
Mandelbrot quest

DIRAC Project

Mandelbrot set



- The Mandelbrot set definition
  - http://en.wikipedia.org/Mandelbrot
- The vicinities of the Mandelbrot set area provide an astonishingly rich fractal images
  - The algorithm consists in assigning a color to each point in the complex plane as a function of a speed of divergence of the Mandelbrot sequence
  - You certainly have seen some of them but even more are even not discovered yet
- In the tutorial we will explore those images while exercising the use of DIRAC tools and grid resources

mandelbrot application

- In the quest we will be using the *mandelbrot* application
  - http://dirac.france-grilles.fr/demo/mandelbrot
- The *mandelbrot* application is a simple python script to construct fractal images:
  - Builds a fractal image around a chosen C point
  - One can vary the size of the image, its precision (zoom level), color scheme
  - The output is an image file in BMP format
    - Can be easily visualized in a Web browser
- The *mandelbrot* application is available also from a grid DIRAC-USER Storage Element:
  - LFN:/dirac/user/a/atsareg/mandelbrot

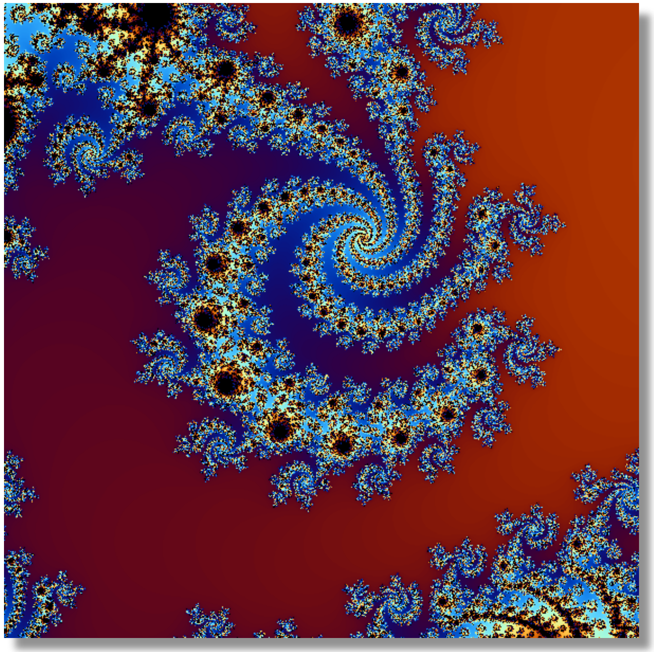
mandelbrot application usage

- Usage:
  - `mandelbrot [options] [<output_file>]`
- Options:
  - `-X, --cx - the real part of the C parameter in the center of the image, default = -0.5`
  - `-Y, --cy - the imaginary part of the C parameter in the center of the image, default = 0.0`
  - `-P, --precision - the step size of the C parameter increment per pixel of the image, default = 0.01`
  - `-M, --max_iterations - the maximum number of the mandelbrot algorithm iterations, default = 100`
  - `-W, --width - image width in pixels, default = 300`
  - `-H, --height - image height in pixels, default = 300`
  - `-B, --bw - force black and white image, default is a color image`
  - `-F, --color_factor - color palette parameter defining how quickly the colors are changing, the value should be in the range 0.<x<1.0, default = 0.02`
  - `-S, --color_phase - a magic color palette parameter, default = 1.0`
  - `-D, --color_delta - yet another magic color palette parameter, default = 1.0`
  - `-h, --help - print this usage info`

Mandelbrot Quest

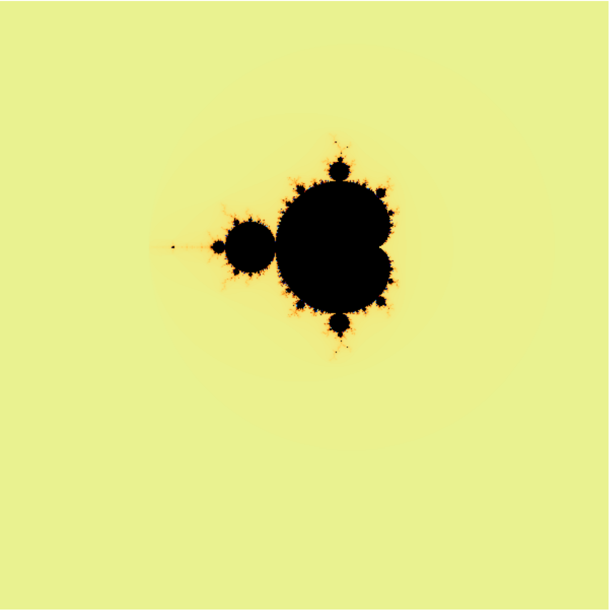
- Goal: find a new interesting and beautiful area in the Mandelbrot set vicinity and let everybody admire it !
  - Of course, by doing the whole work with the *mandelbrot* grid jobs

  -
    ```
    mandelbrot –W 600 –H 600 –X -0.46490 –Y -.56480 –P .000002 –M 500
    ```



Mandelbrot quest steps

- Task steps
  1. Find an interesting seed C point
  2. Build a series of images with an increasing zoom level centered around the seed C point from 1.
  3. Build a movie using the images from 2. as frames



Mandelbrot quest task: step 1

- Run several mandelbrot jobs with varying C and precision
  - Use the Job Launchpad Web interface
  - Submit a number of mandelbrot jobs with varying contents of the Arguments JDL parameter
    - Hint: put the *mandelbrot* application into the Input Sandbox
  - Store the output file in the Output Sandbox
  - Get the output file from the Web portal Job Monitor and inspect it in the browser
  - Choose the most appealing C point
- Remark
  - Running mandelbrot locally on your computer is allowed but this is cheating !

Mandelbrot quest task: step 2

- Run a series of 300-500 mandelbrot jobs with a fixed C parameter and increaing precision (zoom level)
  - Use Parametric Jobs with the Job Launchpad
  - Store output files in a grid Storage Element
    - DIRAC-USER

Mandelbrot quest task: step 3

- Collect the output image files from Step 2. and build a « Mandelbrot journey » movie
  - Use DIRAC API to write a script to launch a grid job creating the movie
    - Use *convert* Unix program as an Executable to do the work. For example:

      ```
      convert –loop 0 *.bmp movie.gif
      ```
    - Use image files from Step 2. as Input Data
    - Store the resulting animated gif image as Output Data
  - Alternatively, use DIRAC API to write a script to collect the image files on your local computer to create the movie
    - Download and store image files in one directory
    - Invoke the *convert* program locally to create the mouvie
    - Upload the resulting animated gif file to a grid Storage Element