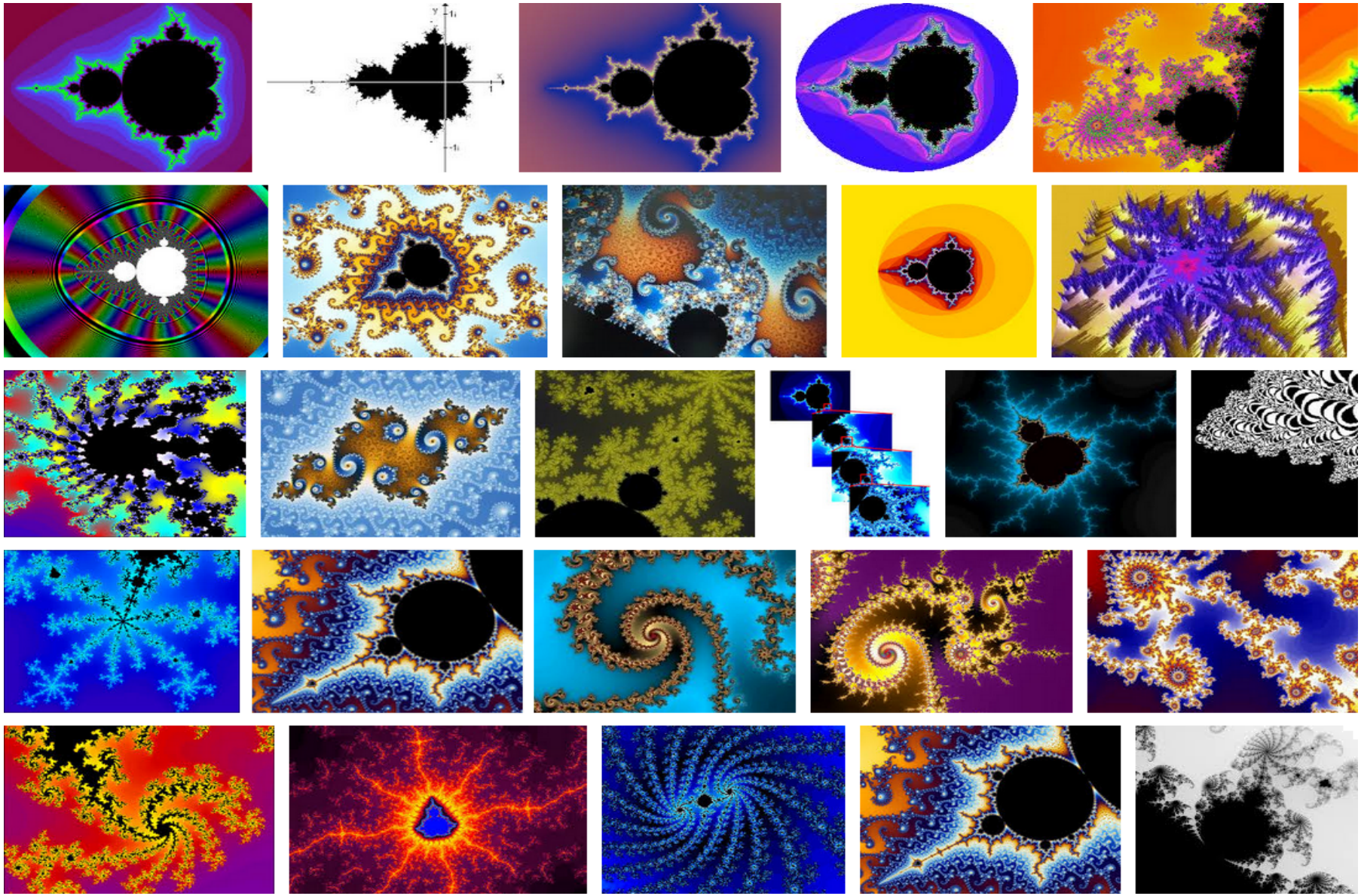
Outline

Mandelbrotmenge

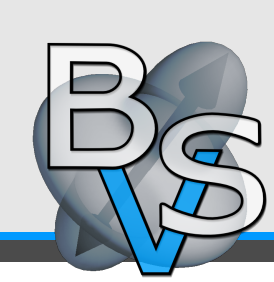


Definition

- Gegeben sei für $c \in \mathbb{C}$ die Folge:

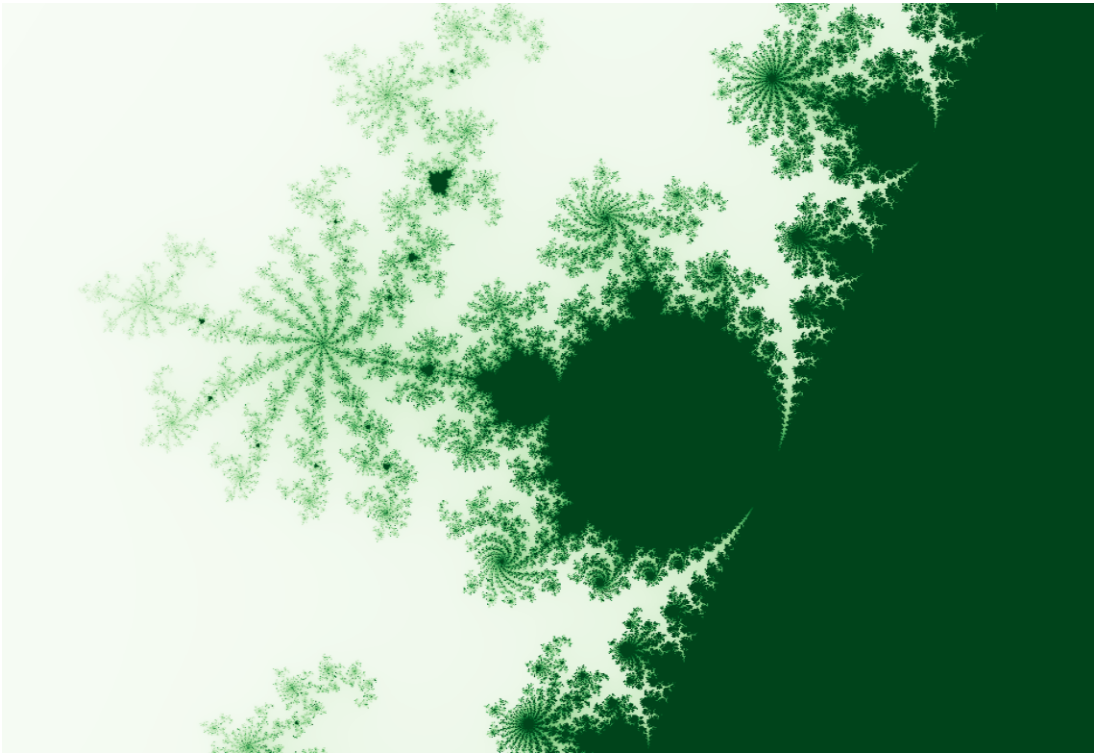
$$z_0 := 0$$
$$z_{n+1} := z_n^2 + c$$

- $\mathbb{M} = \{ c \in \mathbb{C} \mid \forall_{z_i} : |z_i| < \infty \}$ heisst Mandelbrotmenge
- Ferner gilt: $|z_i| > 2 \rightarrow c \notin \mathbb{M}$
- Dennoch ∞ schlecht für endliche Maschinen
- Workaround: Endliche Folge mit maximal n Folgegliedern
- $c \in \mathbb{M} \leftrightarrow \forall_{z_i} : i \leq n, |z_i| \leq r$
- Oftmals $r > 2$ für maximalen Betrag gewählt um Farboszillation zu vermeiden



Von der Menge zum Bild

- Jeder Bildpunkt (Pixel) wird einem $c \in \mathbb{C}$ zugeordnet und die Folge bis zu einem festen n überprüft
- Die Anzahl der Iterationsschritte welche zum Abbruch führt wird auf Farbwert abgebildet



Was hat das mit Python zu tun?

- Heute am Beispiel von Python
- 1. Ziel: Zweidimensionales Feld für Iterationsschritte

```
1  Image: [14, 20]
2    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5    0 0 0 42 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
7    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
8    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
9    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
10   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
11   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
12   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
13   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
14   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Lösung: 1. Ziel

```
1  class Image(object):
2    """ Represents an bitmap image of given width and height. """
3
4    def __init__( self, width, height ):
5      """ Constructs a new bitmap """
6      self.size = [ width, height ]
7      self.data = []
8      for i in xrange(width):
9        self.data.append( [] )
10       for j in xrange(height):
11         self.data[i].append( 0 )
12
13   def __getitem__(self, (i, j) ):
14     """ Returns the pixel i,j. """
15     if(i>=0 and i<self.size[0] and j>=0 and j<self.size[1]):
16       return self.data[i][j]
17
```

Lösung: 1. Ziel

```
18   def __setitem__(self, (i, j), value ):
19     """ Sets the pixel i,j to the given value. """
20     if(i>=0 and i<self.size[0] and j>=0 and j<self.size[1]):
21       self.data[i][j] = value
22
23   def __str__(self):
24     """ Transforming this image into a string."""
25     result = "Image: " + str(self.size) + "\n"
26     for i in xrange(0, self.size[0] - 1):
27       for j in xrange( 0, self.size[1] - 1 ):
28         result += str(self[i,j]).rjust(3) + " "
29       result += "\n"
30     return result
31
32 img = Image( 14, 20 )
33 img[3,3] = 42
34 print img
```

- 2. Ziel Iterationsschritte eintragen

```
Image: [14, 14]
255 255 255 255 255 255 255 255 255  33  16  17  18
255 255 255 255 255 255 255 255 255  25  10   9   8
255 255 255 255 255 255 255 255 255  37   9   7   7
255 255 255 255 255 255 255 255  16  10   8   7   6
 17 255 255 255 255 255 255 255 255  10   8   6   6
 10  29 255 255 255  29  57  14  19 255   8   6   5
  8   8   9  24  10  19   9   8   8   7   6   5   5
  6   7   7   7   7   7   7   6   6   6   5   5   5
  6   6   6   6   6   6   6   6   5   5   5   5   5
  5   5   5   5   5   5   5   5   5   5   5   4   4
  5   5   5   5   5   5   5   5   5   4   4   4   4
  5   5   5   5   5   5   4   4   4   4   4   4   4
  4   4   4   4   4   4   4   4   4   4   4   4   4
```

Lösung: 2. Ziel

```
1  from mandel0 import Image # import Image class
2
3  def mandelfractal( c, max_val=10, max_iter=255 ):
4      """Computes the mandelbrot sequence for given c and check if
           max_iter or max_val is reached. Return the iteration."""
5      z = 0 + 0j
6      it = 0
7      while( abs(z) <= max_val and it < max_iter ):
8          z = z**2 + c
9          it += 1
10     return it
11
12 # some tests
13 img = Image(14, 14)
14 for i in xrange(img.size[0]): # for every pixel:
15     for j in xrange(img.size[1]):
16         c = complex(float(i)/img.size[0], float(j)/img.size[1])
17         img[i,j] = mandelfractal(c)
18 print img
```

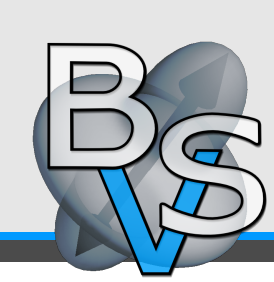
Wann seh ich endlich Bilder?

- 3. Ziel: Berechne einen gegeben Auschnitt der komplexen Ebene

```
Image: [14, 14]
  3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   4   4   4   4   4   3   3
  3   3   3   3   3   4   4   5 255   5   4   4   3
  3   3   3   3   4   4   5   6 255   6   5   4   4
  3   3   3   4   4   5   6   8 255   8   6   5   4
  3   3   4   4   4   5   7  32 255  32   7   5   4
  3   3   4   4   5   6   8  13 255  13   8   6   5
  3   3   4   5   5   8 255 255 255 255 255   8   5
  3   3   4   5   8 255 255 255 255 255 255 255   8
  3   3   4   5   6   9 255 255 255 255 255   9   6
  3   3   4   4   5   6  14 255  10 255  14   6   5
  3   3   3   4   4   5   5   5   5   5   5   5   4
  3   3   3   3   4   4   4   4   4   4   4   4   4
```

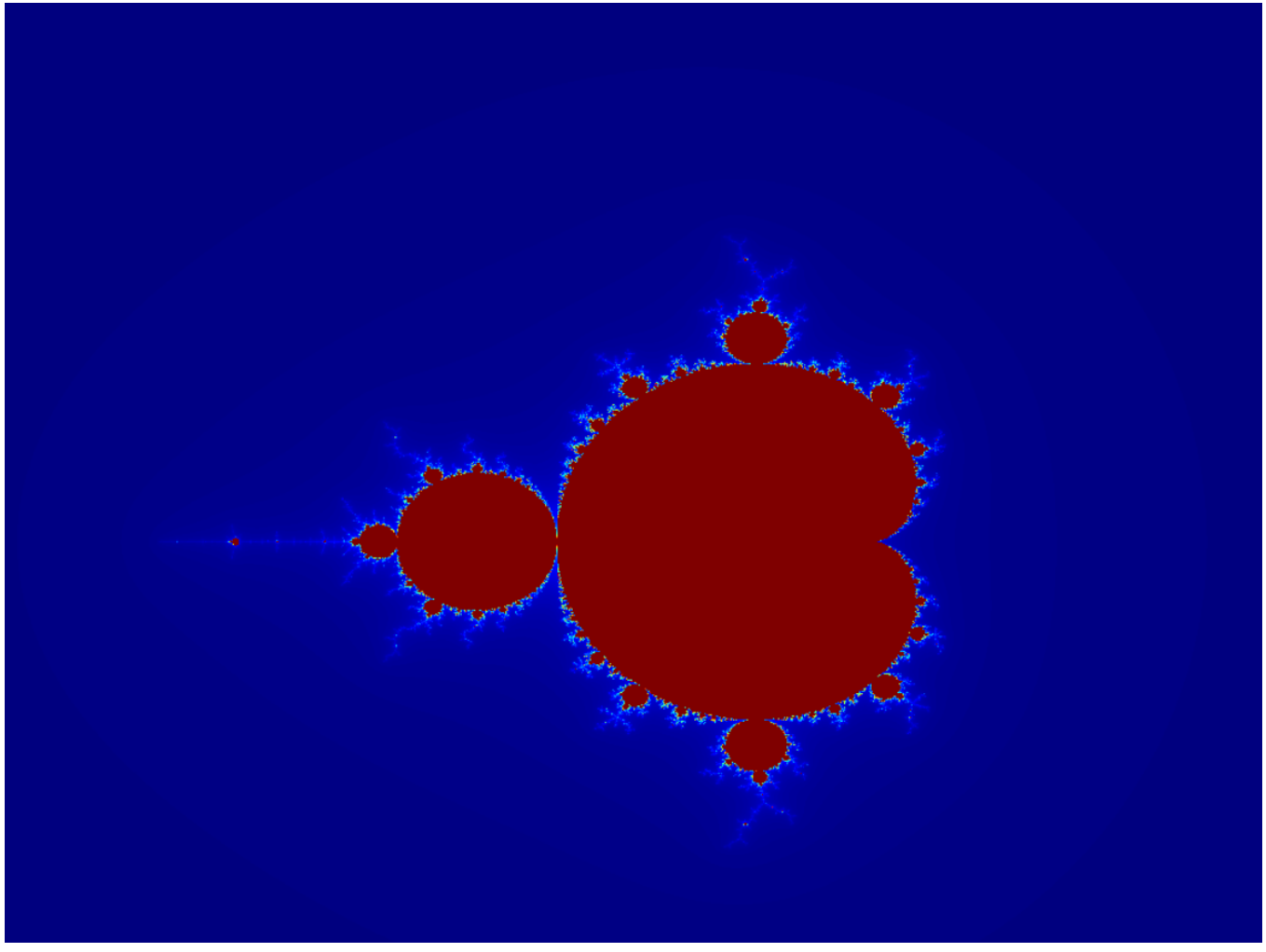
Lösung: 3. Ziel

```
1  from mandel1 import mandelfractal,Image
2
3  def createImage(minX=-2.5, minY=-2.0, maxX=1.5, maxY=1.5, width=14,
       height=14):
4      """ Generates an Image for the given clip """
5      img = Image(width,height)
6      for i in xrange(width):
7          for j in xrange(height):
8              x = float(i)/(width)
9              y = float(j)/(height)
10             c = complex(minX + x*(maxX-minX), minY + y*(maxY-minY))
11             img[i,j] = mandelfractal(c)
12     return img
13
14 # some tests
15 print createImage()
```



Laaangweilig, immer noch keine Bilder

- 4. Ziel: Mache endlich Bilder!



Lösung: 4. Ziel

```
1  from mandel1 import mandelfractal
2  import matplotlib.pyplot as plt # for image rendering
3  import numpy as np # for 2d-array which is understood by matplotlib
4
5  def createImage(minX=-2.5, minY=-2.0, maxX=1.5, maxY=1.5, width
       =1024, height=768):
6      """ Generates an Image for the given clip """
7      img = np.zeros( (height, width), dtype=np.uint8 )
8      for j in xrange(height):
9          for i in xrange(width):
10             x = float(i)/(width)
11             y = float(j)/(height)
12             c = complex(minX + x*(maxX-minX), minY + y*(maxY-minY))
13             img[j,i] = mandelfractal(c)
14     return img
15
16 # some tests
17 plt.imshow( createImage(-0.132, -0.992, -0.124, -0.983 ) )
18 plt.show()
```

Hmm, können wir das irgendwie schneller?

- 5. (optionales Ziel) "Take cores!"
- Jedes Pixel kann unabhängig von den anderen berechnet werden
- Parallelisierung: Pixel, Zeile, Spalte, Block,. . .
- Technische Umsetzung:
  - Multiprocessing anstelle von Multithreading
  - Warum? → Selbststudium: Gobal Interpreter Lock (GIL)
  - Ferner: Funktionelle Programmierung kann es stark vereinfachen

Einschub für Lösung: map-funktion

```
>>> def foo(i):
...     return range(i)
...
>>> map(foo, range(5))
[[], [0], [0, 1], [0, 1, 2], [0, 1, 2, 3]]
>>> [ foo(i) for i in range(5) ]
[[], [0], [0, 1], [0, 1, 2], [0, 1, 2, 3]]
>>> x = []
>>> for i in range(5):
...     x.append(foo(i))
...
>>> x
[[], [0], [0, 1], [0, 1, 2], [0, 1, 2, 3]]
>>>
```

Einschub für Lösung: partial-funktion

```
1 >>> from functools import partial
2 >>> def foobar( i, j ):
3 ...     return [ [x for x in range(i)] for y in range(j) ]
4 ...
5 >>> foobar(2,2)
6 [[0, 1], [0, 1]]
7 >>> foobar(3,3)
8 [[0, 1, 2], [0, 1, 2], [0, 1, 2]]
9 >>> alteredfoobar = partial( foobar, 3 )
10 >>> alteredfoobar( 3 )
11 [[0, 1, 2], [0, 1, 2], [0, 1, 2]]
12 >>> alteredfoobar( 2 )
13 [[0, 1, 2], [0, 1, 2]]
14 >>>
```

Einschub für Lösung: multiprocessing-package

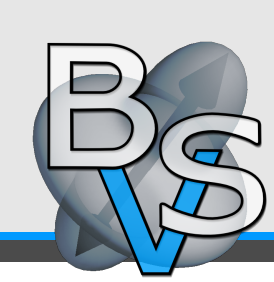
```
1  import multiprocessing # use multiprocessing tool
2
3  # create for each core an own worker-process
4  pool=multiprocessing.Pool()
5
6  # each free worker handle a separate call
7  pool.map(workerfuntion, worker-specific-paramlist)
8
9  # worker is done when map cannot provide tasks anymore
10 pool.close()
11
12 # wait until all workers are done
13 pool.join()
```

Lösung: 5. Ziel

```
1  from mandel1 import mandelfractal
2  import matplotlib.pyplot as plt # for image rendering
3  import numpy as np # for 2d-array understood by matplotlib
4  from functools import partial # alias to altered function
5  import multiprocessing          # use multiprocessing tool
6
7  DEFAULT_WIDTH = 1024;DEFAULT_HEIGHT = 768;DEFAULT_DPI = 72
8
9  def computeRow(minX, minY, maxX, maxY, width, height, j):
10     """ Computes a single row, given by parameter j of the
           image. """
11     tmp = np.zeros( (width), dtype=np.uint8 )
12     for i in xrange(width):
13         x = float(i)/width
14         y = float(j)/height
15         c = complex(minX + x*(maxX-minX), minY + y*(maxY-
               minY))
16         tmp[i] = mandelfractal(c)
17     return tmp
```

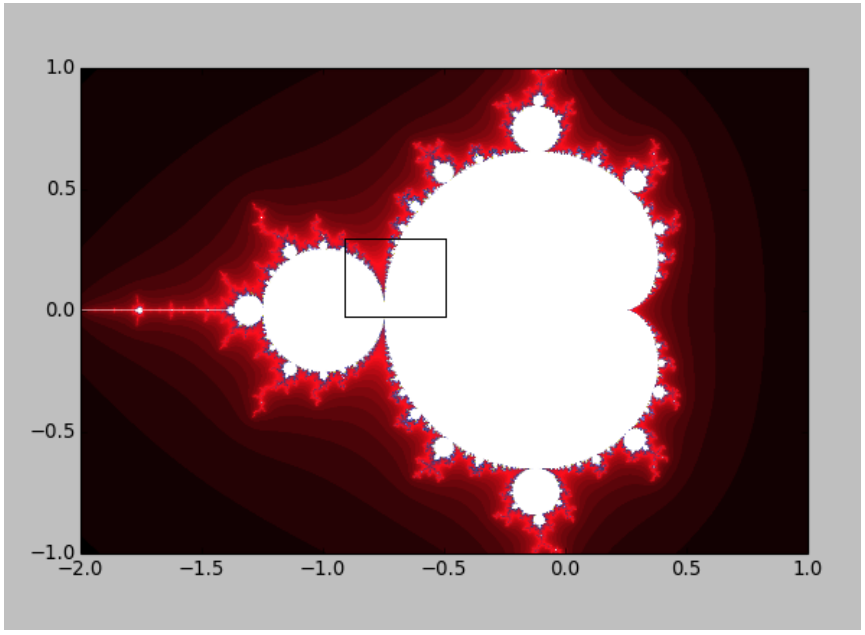
Lösung: 5. Ziel

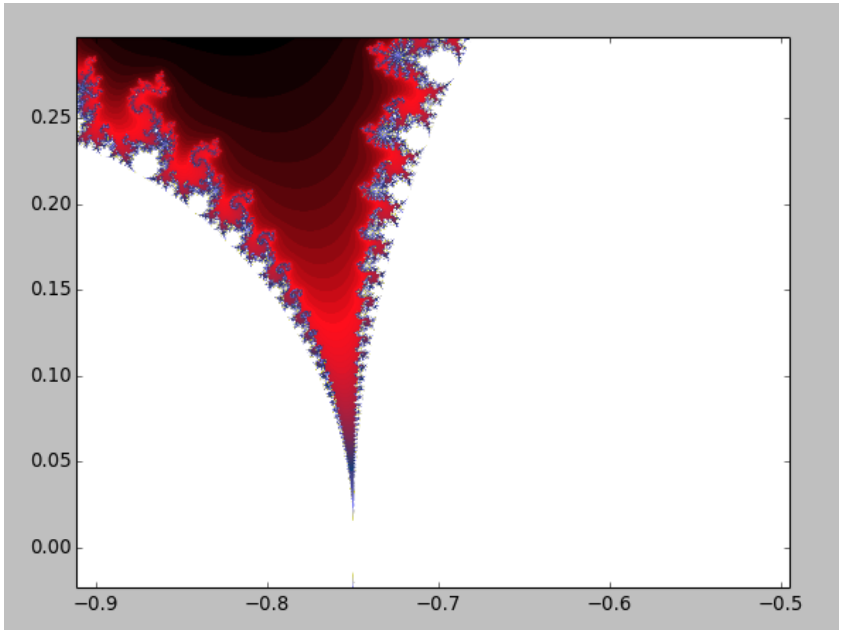
```
19
20 def createImage(minX=-2.5, minY=-2.0, maxX=1.5, maxY=1.5, width=
       DEFAULT_WIDTH, height=DEFAULT_HEIGHT):
21     """ Generates an Image for the given clip """
22     partialCalcRow = partial(computeRow, minX, minY, maxX, maxY,
           width, height)
23     pool=multiprocessing.Pool()
24     img = pool.map(partialCalcRow, xrange(height))
25     pool.close() # do no more tasks, when map fucntion is done
26     pool.join() # wait until all are done
27     return img
```



Haben wir noch Zeit?

- 6. Ziel: Interaktives Zooming und alternative Colormaps





Lösung: 6. Ziel

```
1  from mandel3b import createImage,plt
2  import matplotlib.cm # for list of available colormaps
3  import random # for choosing randomly a colormap
4
5  COLORMAP=random.choice([m for m in plt.cm.datad if not m.
       endswith("_r")])
6  RES = 1000
7
8  def update(ax):
9      """ Creates new image with updated dimensions. """
10     xlim = ax.get_xlim()
11     ylim = ax.get_ylim()
12     img = createImage(xlim[0], ylim[1], xlim[1], ylim[0],
           RES, RES )
13     ax.clear()
14     ax.imshow(img, extent=[xlim[0], xlim[1], ylim[0], ylim
           [1]], zorder=0, cmap=COLORMAP)
15     ax.figure.canvas.draw()
```

Lösung: 6. Ziel

```
16
17 # start image
18 plt.imshow(createImage(-2.0, -1.0, 1.0, 1.0, RES, RES),
       extent=[-2.0, 1.0, -1.0, 1.0], cmap=COLORMAP)
19 ax = plt.gca() # GetCurrentAxis of the plot
20 # connect to event
21 ax.figure.canvas.mpl_connect('button_release_event',
       lambda x: update(ax))
22 plt.show()
```