Universität Zürich[UZH]

Praktikum

Gravitational Lensing Fermat, Laplace

Author:
Jonas Odermatt

Supervisor:
Prasenjit Saha
Rafael Kueng

July 26, 2013

Contents

1 Vorwort

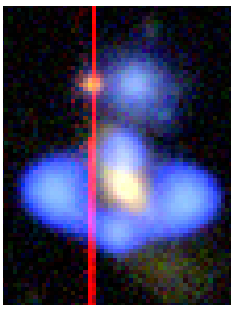
In meinem Praktikum geht es darum, Rafael Kueng bei seiner Masterarbeit bestmoeglichst zu unterstuetzen. Die Masterarbeit handelt von der Entdeckung und Untersuchung von Gravitationslinsen.

2 Gravitationslinsen

Gravitationslinsen bezeichnen Gebilde, welche vornehmlichst aus einer Galaxie und einem dahinter liegenden Quasar bestehen. Das Licht des Quasaren, wird durch die Raumkruemmung der davorliegenden Galaxie zu uns gelenkt. Die Raumkruemmung ist Teil der allgemeinen Relativitaetstheorie. Daraus folgt, dass die vordere Galaxie wie eine Linse funktioniert, welche aus Sicht der Erde, kleine blaue Flecken erzeugt. Um zu verstehen, warum nur Flecken und keine Flaechen (Ausnahme ist der Einstein Ring.) entstehen, haben wir das Fermatprinzip betrachtet. Nach diesem Prinzip hat Licht, welches wir wahrnehmen, immer den Weg des Maximas, Minimas und des Sattelpunktes genommen. Aus diesem Grund sind unter normalen Umstaenden, Flaechen kaum moeglich, da sie diesen Anforderungen nicht genuegen. Maximas, Minimas und Sattelpunkte sind im Idealfall Punkte. Aufgrund der raeumlichen Ausdehnung eines Quasaren und aufgrund weiterer, mir unbekannter Gruende, nehmen wir Maximas usw. nicht als exakte Punkte, sondern als runde Flecken wahr.

Bild einer Linse Mit diesen Informationen habe ich versucht, Gravitationslinsen auf knapp 1000 Bildern (unter spacewarps.org) zu identifizieren.

Figure 1: Beispiel einer Linse:



Nachdem ich mit dem Betrachten dieser Bildmasse fertig war, habe ich angefangen einige Simulationen mit SpaghettiLens unter die Lupe zu nehmen. So kann man versuchen, die blauen Punkte der Gravitationslinsen, Minimas und Sattelpunkten zuzuordnen. (Der maximale Weg geht immer gerade durch die Galaxie und ist daher nicht wahrnehmbar.) Mit SpaghettiLens kann man dann ueberpruefen, ob die Zuordnung sinnvoll ist oder ob eine andere Konstellation der Punkte sinnvoller waere. Sobald man eine sinnvolle Zusammenstellung entdeckt hat, kann man die Masseverteilung der vorderen Galaxie, wie auch einige weitere Informationen herauslesen. Hier eine von mir bearbeitete Linse:

Beispiel (ASW0001a8c): http://mite.physik.uzh.ch/data/002980

Figure 2: In das Programm eingespeiste Information:
(rot = Maxima, blau = Minima, gruen = Sattelpunkt)

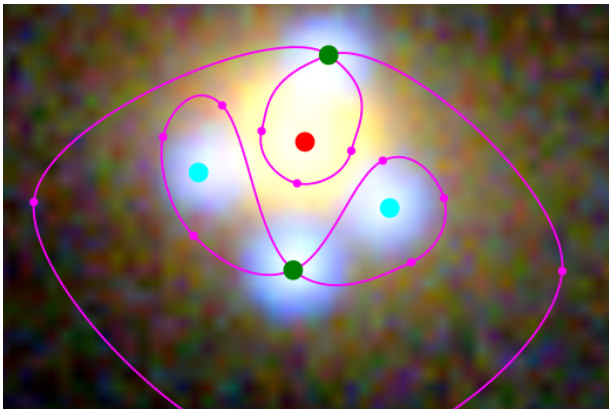


Figure 3: errechnete Masseverteilung der Galaxie:

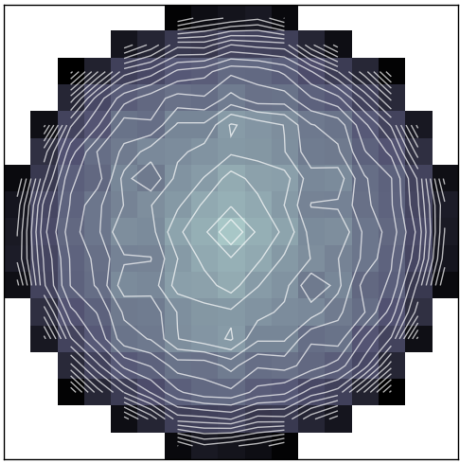


Figure 4: Empfangenes Licht mit der Verteilung der Simulation:
(Desto dunkler, desto hoehere Photonendichte)

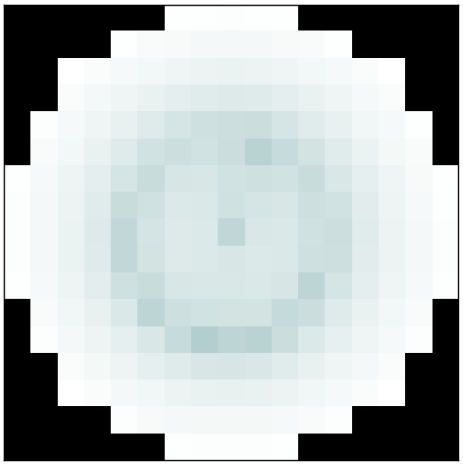
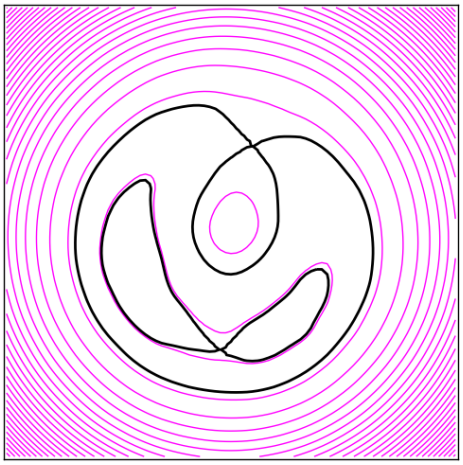


Figure 5: Minimas, Maxima, Sattelpunkte:



3 Laplace

Danach habe ich mit dem Programmieren des Laplace Problemes begonnen. Dazu habe ich zuerst eine vereinfachte Version programmiert.

3.1 Uebung zum Programmieren

Im einfacheren Problem geht es darum, die genauen Punkte einer moeglichst kleinen Flaeche zu berechnen. Dies machen wir mit der an der Kantonsschule behandelten Programmiersprache Python. Stellen sie sich ein Rechteck aus Draht vor. Nun verbiegen sie es etwas, so dass es von oben betrachtet wie ein Rechteck, von der Seite jedoch uneben ist. Geben sie diesen Draht in etwas Seifenblasenwasser. Die Flaeche der innerhalb des Rechtecks entstandenen Seifenblase versucht sich moeglichst klein zu halten. Genau fuer diese Flaeche habe ich ein Programm geschrieben, welches jeden beliebigen Punkt der Oberflaeche mit Hilfe der Topografie der Drahtraender errechnen kann. Die Berechnung fuer diese Flaeche erfolgt durch das wiederholte Ersetzen der Punkte durch den Durchschnitt der umliegenden Punkte bis die Aenderung des Punktes durch die wiederholte Rechnung gegen Null sinkt.

Python Das Herzstueck meines ersten Programmtypes, welcher die neuen Punkte berechnet:

```
# Parameter

h = 0
i = 20 # gewuenschte Anzahl X-Werte
j = 20 # gewuenschte Anzahl Y-Werte
z = 0.01 #gewuenschte Maximalabweichung (absolut)

# Erstellen einer Liste k

m = range(0, j)
l = []
k = []
for i in range(i):
  l.append(m)
k.append(l)

# Start der Berechnungen

while z*((i - 2)*(j - 2)) < r :

    h += 1

    # Fuer jede Berechnungstiefe wird ein neues
    # Listenelement erzeugt und hinzugefuegt.

    for a in range(0, i):
        for b in range(0, j):
            e = k[h - 1][a][b]
            u.append(e)
        v.append(u)
    k.append(v)

    # Hier werden die neuen Punkte errechnet

    for a in range(1, i - 1):
        for b in range(1, j - 1):
            k[h][a][b] = (k[h-1][a+1][b] \
+ k[h-1][a-1][b] + k[h-1][a][b+1] \
```

```
39 + k[h-1][a][b-1]) / 4
40
41     # In diesem Bereich wird geprueft, ob die
42     # gewuenschte Genauigkeit bereits erreicht wurde.
43
44     r = 0
45     for a in range(1, i - 1):
46         for b in range(1, j - 1):
47             e = (k[h][a+1][b] + k[h][a-1][b] \
48 + k[h][a][b+1] + k[h][a][b-1]) / 4
49             f = (e-k[h][a][b])*(e-k[h][a][b])
50             #print("Tiefe:", h, "X-Achse:", a, \
51 "Y-Achse:", b, "Hoehe:" , k[h][a][b], \
52 "Abweichung:", f)
53             if f <= z:
54                 r += f
55             else:
56                 r = p*i*j
```

3.2 Laplace - Poisson

Fuer das echte Laplace Problem habe ich den Rand auf Null gesetzt und in der Mitte die Berechnung einiger Punkte etwas komplizierter gestaltet. Die meisten Punkte werden immer noch durch den Durchschnitt der umliegenden Punkte errechnet. Dort wo sich im Model eine Masse befindet, wird die Gleichung etwas angepasst und durch eine Konstante und die Abstaende der Punkte zueinander ergaenzt. Komplexere Berechnung (dargestellt ohne Tiefenverzeichnis 'h'):

```
1 z[i][j] = (z[i+1][j] + z[i-1][j] + z[i][j+1] \
2 + z[i][j-1] + k*delta**2)/4
```

Wobei delta dem Abstand zwischen den Punkten entspricht und k der Masse des anzuschauenden Punktes mal eine Konstante. Stellt man diese Punkte dreidimensional dar, so entsteht das typische Bild von diesem Algorithmus, von welchem es zahlreiche Bilder im Internet gibt. Fertig ist die Berechnung der Raumkruemmung. Da meine Programmierkentnisse nicht ausserordentlich gross sind, habe ich viele Programmierschritte durch einfache Befehle geschrieben. Dies hatte unweigerlich die Verlangsamung und die extreme Groesse des Programmes zur Folge. Nachdem ich in weitere Pythonbefehle instruiert wurde, konnte ich das Programm bis auf etwa 50 Zeilen kuerzen und

extrem viel schneller machen. (Die komplexeste Version hatte >250 Zeilen.) Dafuer habe ich viele Parameter entfernt und nur noch das Grundgeruest gelassen. Mein Programm waere eigentlich noch kuerzer. Jedoch musste ich eine Maske definieren. Diese hat die Eigenschaft, dass sie alle Punkte in einem gewissen Abstand zum Zentrum auf 0 setzt, um das verzerren der Berechnungen durch allfaellige Ecken zu verhindern. Nachdem die Programmierung fertig war, habe ich sie in das abschliessende Programm eingefuegt und meine Arbeit beendet. Unter www.github.com/Eraster/Laplace ist es moeglich auf meine Programme zuzugreifen, wobei die komplexeste Version Laplace original all in one.py und die kuerzeste Laplace funktion.py heisst.

Funktion: Laplace (Pythonxy) Hier noch das Abschlussprogramm Laplace funktion.py

```
1  from numpy import zeros, amax, amin
2  import numpy as np
3
4  def Mask(c):
5
6      a = 0*c
7      len = np.alen(a)
8      l = float(len)
9
10     dk = (l-1)/2-0.01
11
12     for i in range(0, len):
13         for j in range(0, len):
14             dx = ((l-1)/2-i)**2
15             dy = ((l-1)/2-j)**2
16             dis = sqrt(dx+dy)
17
18             if dis < dk:
19                 a[i][j] = 1
20             else:
21                 a[i][j] = 0
22
23 def Laplace(m):
24     # m == array mit Massenverteilung
25
26     # Einstellen der Parameter
27
```

```
28      Genauigkeit = 0.001
29      z = np.abs(Genauigkeit)
30      delta = 2
31      repeat = 10000
32
33      mas = Mask(m)
34
35      konstant = ((1*m)*delta**2)/4
36      kd = 0*m
37      k = []
38      k.append(kd)
39
40      h = 0
41      r = z + 1
42
43      while z < r and repeat > h :
44
45    # Berechnung der neuen Tiefe +
46    # neues Listenelement + Pruefung
47
48          h += 1
49          kd = k[h - 1]
50          average = 1*kd
51          average[1:-1, 1:-1] = (kd[0: -2, 1: -1] \
52  + kd[2:, 1: -1] \
53  + kd[1: -1, 0: -2] + kd[1: -1, 2:])/4
54          kd = (average + konstant)*mas
55          k.append(kd)
56
57          if amax(k[h] - k[h-1]) <= z and amin(k[h] \
58  - k[h-1]) >= -1*z:
59              r = 0
60          else:
61              r = z + 1
62
63      return kd
```

4 Lensing

(Es tut mir leid, diesen Text nicht auch in Deutsch verfasst zu haben. Ich bitte vielmals um Entschuldigung. Dieser Text wurde in Englisch verfasst, da wir in in Englisch verwenden werden (zooniverse).)

In this part of my little article, I will give you a closer look on my calculation (fermat and laplace). First of all, the calculation itself isn‘t very difficult but the theorie behind is. So what I will do is just describing the laplace- and farmacalculation. In the end, you will recognize the sattlepoints, the minimas and the maximum and know their meaning. But you won‘t understand the rason why it is this way!

4.1 Laplace

The first part of the calculation is to solve the Laplace-problem for the mass you have. (It is simple to work with a big mass in the very center of your map.) The goal is that every point on your mass profile fulfills its definition. For the perfect calculation, every Point of the surface comes true with this formula:

$$\text{average} \cdot \frac{\text{mass} \cdot \Delta}{4} - \text{value} = 0$$

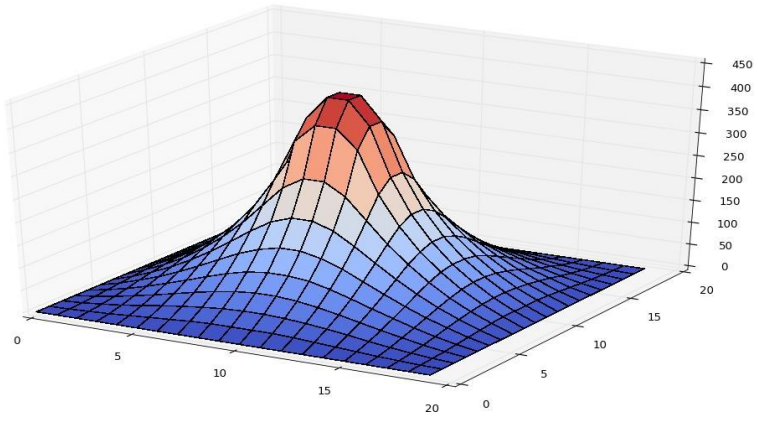
average: Average of the points around the point you‘re looking at.
delta: distance between your measurements
mass: Thats the most important part! For nice programming this should be an array with 2 Dimensions. Every Point has his specific mass.
value: Value of the point you‘re looking at.

After this calculation you don‘t have a mass profile anymore, but the map of a kind of space curvature. Because we calculate with numeric systems, in most cases it‘s impossible to reach the absolute correct solution. (That means that the computer will stay in a loop if you program it to give you the perfect solution.) An approximation is required. I programmed this calculation with a loop. The loop replaces the value of every point by this calculation:

$$\text{Laplace(point)} := \text{average} \cdot \frac{\text{mass} \cdot \Delta}{4}$$

The loop goes as long, as the approximation isn‘t good enough. (You can have a look at this on the funciton: Laplace (Pythonxy) from this article.

Figure 6: space curvature (matplot)



4.2 Fermatprinzip

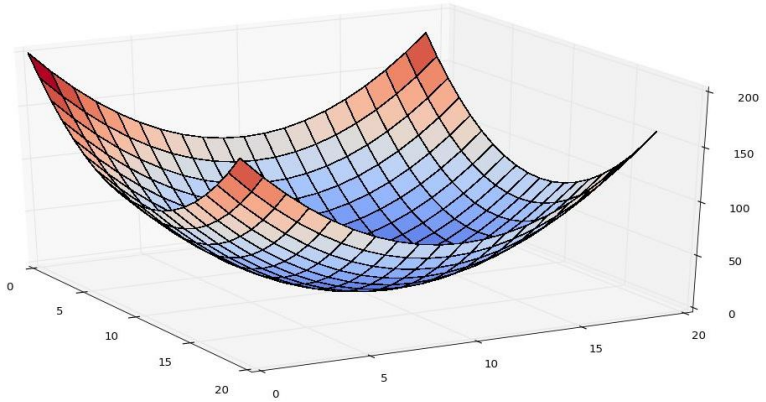
As a second step, we have to add a value to the solution from Laplace to get the solution for the fermat law. The value which we add is definied as the distance to the centre in square. The center is a definied point in the middle of the calculation.

$$Ferma(point) := \mathrm{k} \cdot \left(x^2 \cdot \ y^2\right) + \mathrm{Laplace(point)}$$

k: difficult konstant (I havent worked with it yet but for qualitative right results just set it to 1.)
$x^2 \cdot y^2$: This is just the distance to the middle point in square.

If you have calculatet that its easy to animate it. I just did it with matplot and it looked like this:

Figure 7: Distance in square



Here you see the calculation(distance in square + Laplace):
h: This is the calculated array.

```
import numpy as np
from matplotlib import cm
from mpl_toolkits.mplot3d.axes3d import get_test_data
import matplotlib.pyplot as plt

# Twice as wide as it is tall.
fig = plt.figure()

#---- First subplot
ax = fig.add_subplot(1, 1, 1, projection='3d')
#ax.set_zlim(1500,3000)

X = np.arange(0, 100, 1)
Y = np.arange(0, 100, 1)
X, Y = np.meshgrid(X, Y)

plt.axis('equal')

surf = plt.contour(X, Y, h, 75,rstride=1, \
 cstride=1, cmap=cm.coolwarm, \
 linewidth=0.1, antialiased=False)

#---- Second subplot

plt.show()
```

In the appendix you find the whole program and an example for my fermat calculation. (It‘s the shortest version I have from this sort of calculation.)

4.3 Usage

As soon as the calculation is complete. You can see the sattlepoints who look like an ”8”. Maximas are just the highest spots and minimas the pits. It should look like at the Picture 8.
In Spaghettilens the imput is just a map with these 3 kinds of points. The goal is then to calculate this vice versa. So what Spaghettilens does is just the calculation I explained the other way round. Because this is difficult and you can‘t find the absolute true solution. In the project you give some Citizen Scientists pictures from spacewarps like on Picture 9.

Figure 8: 2D Fermat (simple)

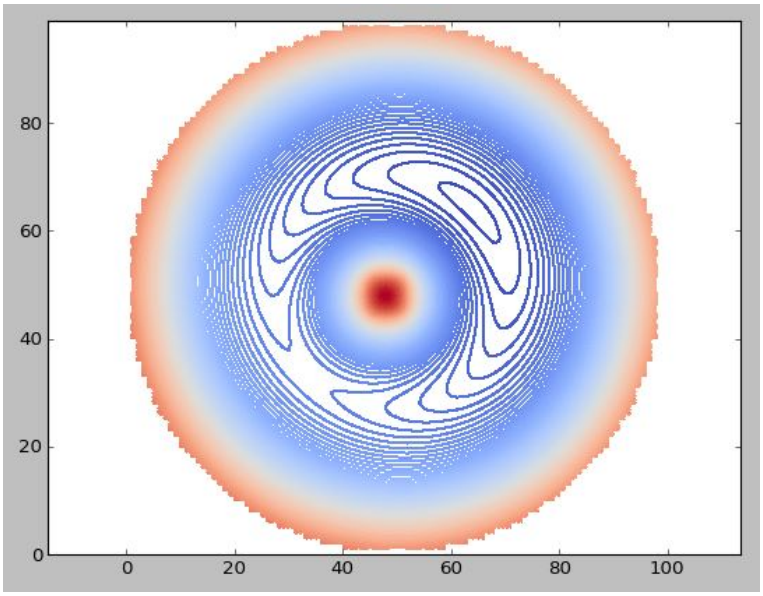
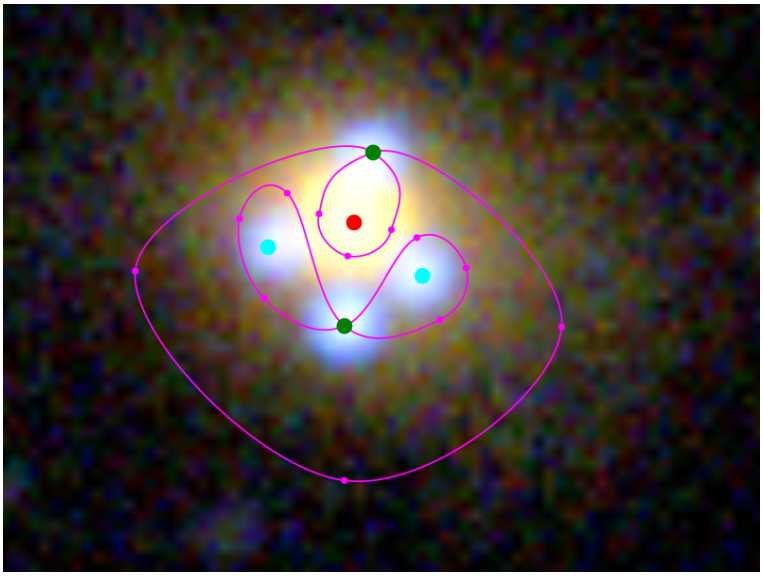
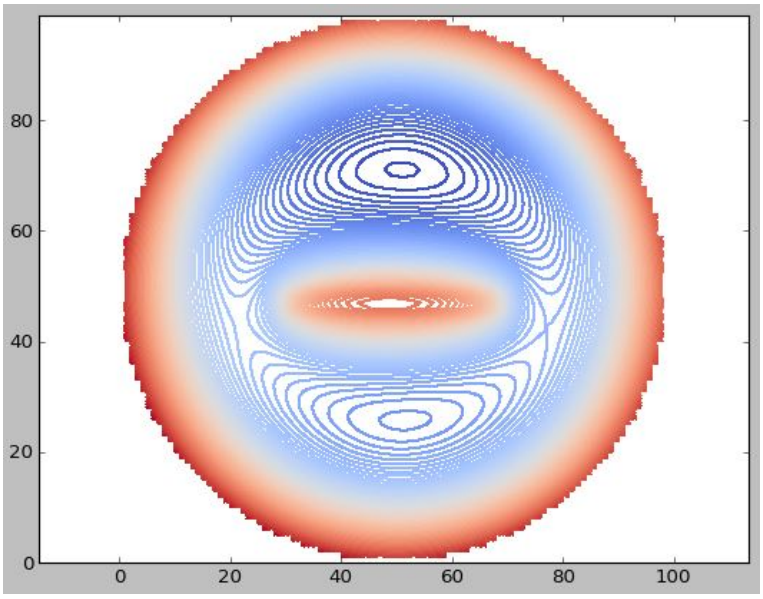


Figure 9: Input



http://mite.physik.uzh.ch/data/002996
They then mark these 3 kinds of points (red = maximum, blue = minimum, green = sattlepoint) and the calculation can run. First we tried this with simulated lenses and looked how close the Citizen Scientists come to the solution. We found out, that if the mass is centered, they manage to mark everything right. Becuase then you only see 1 maximum, minimum and sattlepoint. But if the mass gets more complicated there aren‘t just 3 spots anymore. In most of these cases, there are 2 sattlepoints, 2 minimas and 1 maxima. In this case, there often are some difficulties to put them on the right place. This work really is important because there is no way to write an algorithm yet who manages to do this work. The findings are then looked at and may pull us a bit further in science. I wish all of these scientists good look because my time here at UZH is over. Good Luck and good bye.

Figure 10: 2D Fermat



5 Fazit

In meinem Praktikum habe ich im Bereich meiner Moeglichkeiten so gut es geht mitgearbeitet. Schlussendlich konnte ich die Laplace bzw. Fermatfunktion einbauen und bin sehr stolz darauf. Solch ein Praktikum bietet einen besonderen Einblick in das Leben und die Arbeitsweise an der Universitaet Zuerich. Es ist mir daher eine Ehre, hier arbeiten zu duerfen. Das alles waere nicht moeglich gewesen ohne die grosse Unterstuetzung von Rafael Kueng, Doktor Prasenjit Saha und Professor Doktor Thomas Gehrmann, welcher es mir erst ermoeglichte, mein Praktikum hier an der UZH zu starten. Zum Abschluss moechte ich Ihnen ganz herzlich danken.

6 Bibliographie

Figure 1-5, 9: http://mite.physik.uzh.ch/data/002980
(www.spacewarps.org)

Figure 6-8, 10-11: Made with programmes from Jonas Odermatt
free download: (https://github.com/Eraster/Laplace)

LaTeX: http://de.wikibooks.org/wiki/LaTeX/

7 Appendix

Final program Fermat(m)

```
from numpy import zeros, amax, amin
import numpy as np
from matplotlib import cm
from mpl_toolkits.mplot3d.axes3d import get_test_data
import matplotlib.pyplot as plt

def Mask_a(c):

    a = 0*c
    len = np.alen(a)
    l = float(len)

    dk = (l-1)/2-0.01

    for i in range(0, len):
        for j in range(0, len):
            dx = ((l-1)/2-i)**2
            dy = ((l-1)/2-j)**2
            dis = np.sqrt(dx+dy)

            if dis < dk:
                a[i][j] = 1
            else:
                a[i][j] = 0

    return a

def Laplace(m):

    Genauigkeit = 0.001
    z = np.abs(Genauigkeit)
    delta = 2
    repeat = 10000

    mas = Mask_a(m)

    konstant = ((1*m)*delta**2)/4
```

```
38      kd = 0*m
39      k = []
40      k.append(kd)
41
42      h = 0
43      r = z + 1
44
45      while z < r and repeat > h :
46
47          h += 1
48          kd = k[h - 1]
49          average = 1*kd
50          average[1:-1, 1:-1] = (kd[0: -2, 1: -1] \
51          + kd[2:, 1: -1] + kd[1: -1, 0: -2] \
52          + kd[1: -1, 2:])/4
53          kd = (average + konstant)*mas
54          k.append(kd)
55
56          if  amax(k[h] - k[h-1]) <= z  \
57          and amin(k[h] - k[h-1]) >= -1*z:
58              r = 0
59          else:
60              r = z + 1
61
62      return kd
63
64  def Fermat(m):
65
66      place = Laplace(m)
67      xlen = float(len(place))
68      ylen = float(len(place[0]))
69      xy = 0*place
70
71
72      for i in range(len(place)):
73          for j in range(len(place[0])):
74              xy[i][j] = (xlen/2-i)**2+(ylen/2-j)**2
75              # factor k == 1
76
77      ferma = place + xy
78
```

```
79      return ferma
80
81  def Mask_b(c):
82
83      a = c
84      len = np.alen(a)
85      l = float(len)
86
87      dk = (l-1)/2-0.01
88
89      for i in range(0, len):
90          for j in range(0, len):
91              dx = ((l-1)/2-i)**2
92              dy = ((l-1)/2-j)**2
93              dis = sqrt(dx+dy)
94
95              if dis < dk:
96                  pass
97              else:
98                  a[i][j] = None
99
100     return a
101
102
103 ma = zeros((100,100), float)
104
105 ma[45: 50, 30: 70] = 10#ma[45: 50, 30: 70] = 10
106
107 h = Fermat(ma)
108 h = Mask_b(h)
109
110 # Animation
111
112 # Twice as wide as it is tall.
113 fig = plt.figure()
114
115 #---- First subplot
116 ax = fig.add_subplot(1, 1, 1, projection='3d')
117 #ax.set_zlim(1500,3000)
118
119 X = np.arange(0, 100, 1)
```

```
120 Y = np.arange(0, 100, 1)
121 X, Y = np.meshgrid(X, Y)
122
123 plt.axis('equal')
124
125 surf = plt.contour(X, Y, h, 75,rstride=1, \
126  cstride=1, cmap=cm.coolwarm, \
127  linewidth=0.1, antialiased=False)
128
129 #---- Second subplot
130
131 plt.show()
```

Figure 11: 3D plot with the settings from Fermat(m)