Aufgabe 1 - Geburtstagskuchen

Dominic Meiser

34. BwInf Runde 2

Inhaltsverzeichnis

1 Lösungsidee

1.1 Ein formales Maß für die Gleichmäßigkeit

Meine Grundidee für dieses Maß ist einfach: Ich betrachte die Abstände zwischen den Kerzen zueinander und zwischen den Kerzen und dem Rand des Kuchens. Da es keinen Sinn macht, den Abstand zwischen zwei Kerzen zu betrachten, wenn eine Kerze dazwischen liegt, muss man diese Abstände ignorieren. Dies kann man durch einen abgewandelten Floyd-Warshall-Algorithmus machen, der alle Werte in der Adjazenzmatrix (mit den Abständen der Kerzen zueinander) auf ∞ setzt, wenn der Abstand des Intermediary zu beiden Kerzen kleiner ist als der Abstand zwischen den beiden Kerzen. Anschließend schaue ich mir Maximum, Minimum und Durchschnitt der nicht auf ∞ gesetzten Werte in der Adjazenzmatrix an und nehme den größeren Wert aus Durchschnitt minus Minimum und Maximum minus Durchschnitt. Da hierbei beliebig große Werte entstehen, mit denen kein Mensch etwas anfangen kann, teile ich diesen Wert letztendlich noch durch die Größte Abweichung, die Entstehen kann, nämlich $\sqrt{(xmax - xmin)^2 + (ymax - ymin)^2}$, wobei $xmax$ der größte x-Wert des Randes, $xmin$ der kleinste, und für y das Gleiche. Dadurch ergibt sich, dass **0 der beste** und **1 der schlechteste Wert** ist.

1.2 Der Algorithmus zur Bewertung

Der Algorithmus muss nun zunächst die beteiligten Punkte bestimmen. Bei den Kerzen ist dies bekannt, aber da ich nicht ohne Weiteres den Abstand einer Kerze zum Rand bestimmen kann, wenn dieser Kurven enthält, muss ich den Rand als eine Liste von nebeneinanderliegenden Punkten auffassen. Dadurch kann ich den Wert nur annähern. Nachdem ich also die Abstände zwischen den Kerzen zueinander und zum Rand bestimmt habe, kann ich den oben genannten, abgewandelten Floyd-Warshall-Algorithmus ausführen und anschließend Minimum, Maximum und Durchschnitt bestimmen. Mit diesen Werten und den zusätzlichen $[xy](min|max)$-Werten kann ich dann einfach die Gleichmäßigkeit bestimmen.

1.3 Der Algorithmus zum Verteilen

Die erste Idee, die ich hatte, um die Kerzen zu verteilen, war, diese als Magneten anzusehen, da sich diese (bei gleicher Flussdichte) gleich stark abstoßen und so denselben Abstand bekommen sollten. Dann ist mir aufgefallen, dass es keine magnetischen Monopole gibt, die ich hierfür gebraucht hätte. Deshalb habe ich mich dazu entschloßen, die Punkte des Randes und die Kerzen als Elektronen aufzufassen. Die Formel für die Kraft, die ein Coloumbfeld (das Feld eines einzelnen Elektrons im Vakuum) beschreibt, lautet $F = \frac{1}{4 \cdot \pi \cdot \varepsilon_0} \cdot \frac{q_1 \cdot q_2}{r^2}$. Dabei fällt auf, dass der erste Teil der Formel nur Konstanten enthält, die für diese Aufgabe irrelevant sind. Die Ladung q eines Elektrons ist auch konstant. Somit bleibt am Ende die Formel $F = \frac{1}{r^2}$ stehen. r ist hier der Abstand zwischen den beiden Elektronen, nach dem Satz von Pythagoras also $r = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Eingesetzt in die Formel ergibt dies $F = \frac{1}{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Um nun die Kerzen nach diesem Prinzip zu positionieren, fasse ich den Kuchen als Grid auf. Jedes Feld in diesem Grid hat einen Wert, der entweder größer als 1 ist, wenn das Feld außerhalb des Kuchens liegt, oder die Summe aus den F jedes Elektrons (am Anfang sind dies nur die Punkte auf dem Rand). Zunächst suche ich mir also das Feld mit dem kleinsten Wert aus und positioniere die erste Kerze dort. Nun update ich die Werte aller anderen Felder, indem ich F dieser Kerze zu jedem anderen Feld dazuaddiere.

2 Umsetzung

Ich habe die Lösungsidee in C++11 mit dem Framework Qt (>=5.5) implementiert. Dabei ist es sowohl möglich, den Code mit CMake zu bauen, als auch ihn mit qmake zu bauen. Mit qmake kann man jedoch nur die GUI, nicht aber das CLI, benutzen.

2.1 Datei `algo.cpp`

2.1.1 Der Algorithmus zum Bewerten

Die `rate`-Methode nimmt den Kuchen als `QPainterPath` und die Kerzen als `QList<QPointF>` entgegen und berechnet die Gleichmäßigkeit, so wie es die Lösungsidee beschreibt.

```
double rate(const QPainterPath &path, const QList<QPointF> &candles)
{
```

In diesen Variablen werden Minimum, Maximum und Durchschnitt gespeichert

```
    double min=dinf, max=-1, avg=0;
```

In diesen Variablen werden die entsprechenden Werte des Kuchens gespeichert

```
    double xmin=dinf, xmax=0, ymin=dinf, ymax=0;
```

Diese Variable zählt mit, wieviele Abstände ich bereits angeschaut habe. Dies ist wichtig, um den Durchschnitt zu bestimmen.

```
    int count = 0;
```

Zuerst die Adjazenzmatrix füllen

```
    vector<vector<double>> adj(candles.size()+1, vector<double>(candles.size()+1, dinf));
    for (int i = 0; i < candles.size(); i++)
    {
```

Der Abstand einer Kerze zu sich selbst ist ∞

```
        adj[i+1][i+1] = dinf;
```

Den Abstand zum Rand bestimmen. Dabei werden die Punkte, die den Rand approximieren, durch die Methode `QPainterPath::pointAtPercent` bestimmt.

```
        double d = dinf;
        for (double r = 0; r < 1; r += stepRate)
        {
            auto p = path.pointAtPercent(r);
            double xd = p.x() - candles[i].x();
            double yd = p.y() - candles[i].y();
            if (i == 0) // only do this once
            {
                xmin = std::min(xmin, p.x());
                xmax = std::max(xmax, p.x());
                ymin = std::min(ymin, p.y());
                ymax = std::max(ymax, p.y());
            }
            d = std::min(d, xd*xd + yd*yd);
        }
        adj[i+1][0] = sqrt(d);
        adj[0][i+1] = adj[i+1][0];
```

Den Abstand zu den anderen Kerzen bestimmen. Diesen kann ich einfach über den Satz des Pythagoras bestimmen.

```
        for (int j = i+1; j < candles.size(); j++)
        {
            int xd = candles[i].x() - candles[j].x();
            int yd = candles[i].y() - candles[j].y();
            adj[i+1][j+1] = sqrt(xd*xd + yd*yd);
            adj[j+1][i+1] = adj[i+1][j+1];
        }
    }
```

Der in der Lösungsidee beschriebene, abgewandelte Floyd-Warshall-Algorithmus

```
    for (size_t k = 1; k < adj.size(); k++)
        for (size_t i = 0; i < adj.size(); i++)
            for (size_t j = 0; j < adj.size(); j++)
                if (adj[i][k] < adj[i][j] && adj[k][j] < adj[i][j])
                    adj[i][j] = dinf;
```

Minimum, Durchschnitt und Maximum berechnen

```
    for (size_t i = 0; i < adj.size(); i++)
        for (size_t j = i+1; j < adj.size(); j++)
            if (adj[i][j] != dinf)
            {
                min = std::min(min, adj[i][j]);
                max = std::max(max, adj[i][j]);
                avg = (count * avg + adj[i][j]);
                avg /= ++count;
            }
```

Sollten die Werte noch auf ihren Startwerten stehen, sind keine Kerzen angegeben worden. In diesem Fall gebe ich -1 zurück.

```
    if (min==dinf || max==-1) // no points found
        return -1;
```

Jetzt noch den Wert bestimmen und durch die größtmögliche Abweichung teilen, wie in der Lösungsidee beschrieben.

```
    double xd = xmax - xmin, yd = ymax - ymin;
    Q_ASSERT(xd > 0 && yd > 0);
    return (std::max(avg-min, max-avg) / std::sqrt(xd*xd + yd*yd));
}
```

2.1.2 Der Algorithmus zum Verteilen

Die spread-Methode nimmt den Kuchen als `QPainterPath` und die Anzahl an Kerzen entgegen und gibt eine Liste mit den Positionen der Kerzen zurück. Mir ist bekannt, dass dies nicht zwingend die beste Verteilung sein muss.

```
QList<QPointF> spread(const QPainterPath &path, int numcandles)
{
```

In diesen Variablen werden die entsprechenden Werte des Kuchens gespeichert

```
    double xmin=dinf, xmax=0, ymin=dinf, ymax=0;
```

Die Punkte auf dem Rand werden mithilfe der Methode `QPainterPath::pointAtPercent` bestimmt.

```
    set<QPointF> border;
    for (double r = 0; r < 1; r += stepRate)
    {
        auto p = path.pointAtPercent(r);
        xmin = std::min(xmin, p.x());
        xmax = std::max(xmax, p.x());
        ymin = std::min(ymin, p.y());
        ymax = std::max(ymax, p.y());
        border.insert(p);
    }
    Q_ASSERT(xmin < xmax && ymin < ymax);
```

Das Grid, das den Kuchen repräsentiert

```
    double field[(int)(1 / stepRate)][(int)(1 / stepRate)];
    double xv[(int)(1 / stepRate)], yv[(int)(1 / stepRate)]; // real values for the field index
```

Eine Queue, sortiert nach der Summe aller F-Werte für das Feld.

```
    set<pair<double, QPointF>> q;
```

Zunächst das Grid füllen.

```
    for (int i = 0; i < 1 / stepRate; i++)
    {
        xv[i] = xmin + (xmax - xmin) * stepRate * i;
        for (int j = 0; j < 1 / stepRate; j++)
        {
            if (i == 0)
                yv[j] = ymin + (ymax - ymin) * stepRate * j;
```

Wenn der Punkt auf dem Kuchen liegt, den Wert wie in der Lösungsidee beschrieben berechnen und ihn der Queue hinzufügen.

```
            if (path.contains(QPointF(xv[i], yv[j])))
            {
                field[i][j] = 0;
                for (auto p : border)
                {
                    double xd = xv[i] - p.x();
                    double yd = yv[j] - p.y();
                    field[i][j] += 1 / (xd*xd + yd*yd); // F = 1/(4*PI*E0) * (q1*q2)/r²
                }
                q.insert({field[i][j], QPointF(xv[i], yv[j])});
            }
```

Ansonsten den Wert auf >1, hier 2, setzen.

```
            else
                field[i][j] = 2;
        }
    }
```

Die Kerzen verteilen

```
    QList<QPointF> candles;
    for (int i = 0; i < numcandles; i++)
    {
        if (q.empty())
        {
            QMessageBox::critical(0, "ERROR", "Das Objekt ist schlecht definiert, ich kann "
                                              "keine Punkte darin finden!", QMessageBox::Ok);
            return QList<QPointF>();
        }
```

Den ersten Punkt aus der Queue nehmen und die Kerze dort positionieren.

```
        auto a = * q.begin();
        candles << a.second;
```

Anschließend die Queue leeren, die Werte für die Felder im Grid updaten und die Queue wieder füllen.

```
        q.clear();
        for (int i = 0; i < 1 / stepRate; i++)
        {
            for (int j = 0; j < 1 / stepRate; j++)
            {
                double xd = xv[i] - a.second.x();
                double yd = yv[j] - a.second.y();
                field[i][j] += 1 / (xd*xd + yd*yd);
                q.insert({field[i][j], QPointF(xv[i], yv[j])});
            }
        }
    }
```

Und zum Schluss die Liste der Positionen der Kerzen zurückgeben.

```
    Q_ASSERT(candles.size() == numcandles);
    return candles;
}
```

2.2 CLI (Command Line Interface)

Die eine Möglichkeit, mein Programm zu bedienen, ist via CLI, also über die Konsole. Dabei werden folgende Optionen unterstützt:

build/Geburtstagskuchen$ ./cake-cli -h

```
Usage: ./cake-cli [options]

Options:
  -h, --help          Displays this help.
  --spread <candles>  Spread the given number of candles
  --omit              Don't print out the spreaded candle positions
  --rate              Rate the candles
  -o <file>           The file to output the cake
  -i <file>           The file to read the cake from (binary format)
  --dump              Will dump the cake as text format (after spread if given)

Elapsed time (in milliseconds): 4.591
```

Wenn keine Datei als Eingabe angegeben ist, wird der Benutzer aufgefordert, die Kuchenform und gegebenenfalls die Kerzen anzugeben. Die Befehle dafür heißen:

build/Geburtstagskuchen$ ./cake-cli

```
Commands:
  move x y               Move to the given position
  line x y               Draw a line to the given position
  quad cx cy x y         Draw a quadratic bezier curve to the given
                         position using the control point cx,cy
  cubic cx cy dx dy x y  Draw a cubic bezier curve to the given position
                         using the control points cx,cy and dx,dy
  candle x y             Put a candle at the given position
  q                      Finish the cake

Elapsed time (in milliseconds): 4.486
```

2.3 GUI (Graphical User Interface)

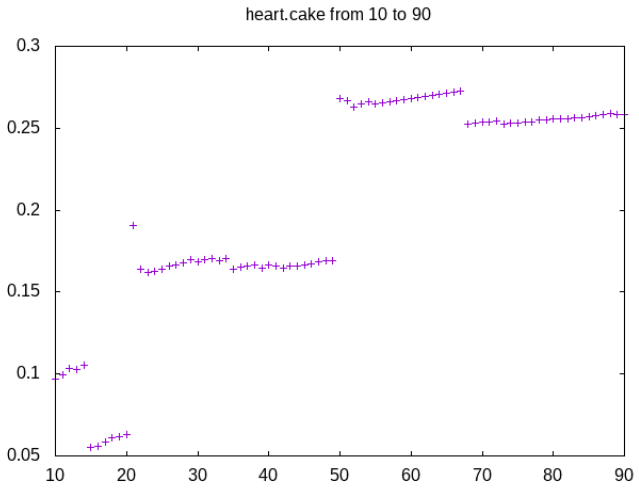
Die andere Möglichkeit, mein Programm zu bedienen, ist via GUI. Dabei ist zu beachten, dass die GUI keine quadratischen oder kubischen Bezier-Kurven unterstützt, dafür aber Kreise, die durch 4 kubische Bezier-Kurven angenähert werden. Die vorhandene Rückgängig- und Wiederherstellen- Funktion ist nicht implementiert. Dafür hat die GUI die Möglichkeit, ein Bild des Kuchens und den Kerzen zu zeichnen, und kann den Kuchen „on the fly“ bewerten. Screenshots der GUI sind bei den Beispielen eingefügt.

3 Beispiele

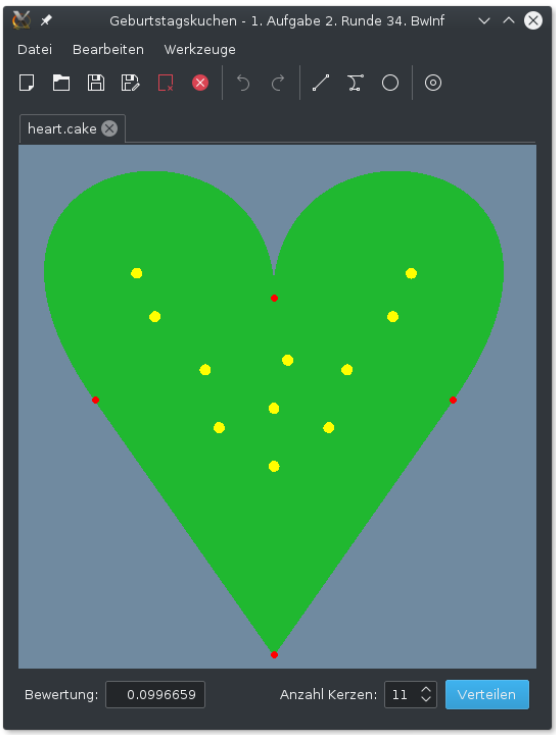
3.1 Herz

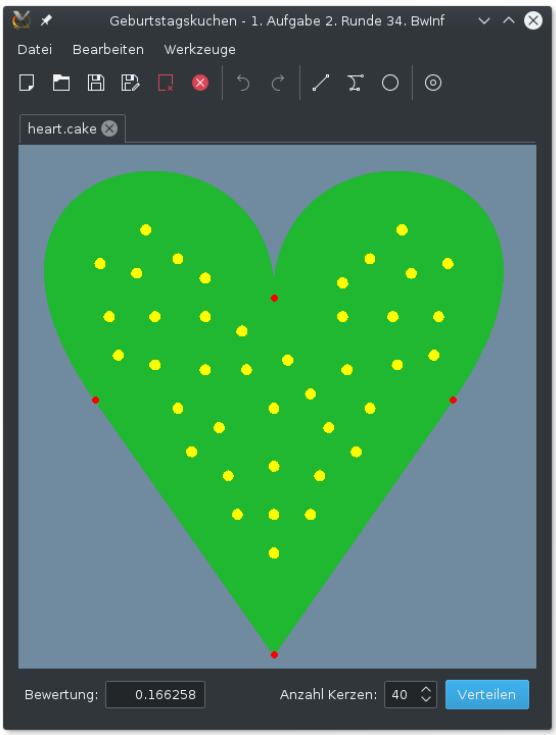
Als erstes natürlich das Beispiel aus der Aufgabenstellung, ein Herz:

Anzahl Kerzen	Rating
10	0.097074
20	0.062750
30	0.168687
40	0.166258
50	0.268295
60	0.268090
70	0.253479
80	0.255688
90	0.258165



Hier das ganze noch als mehr oder weniger schöne Grafik mit 11 und 40 Kerzen:

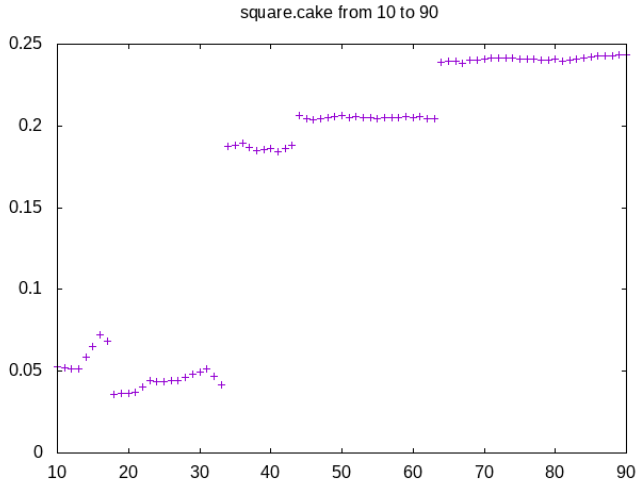




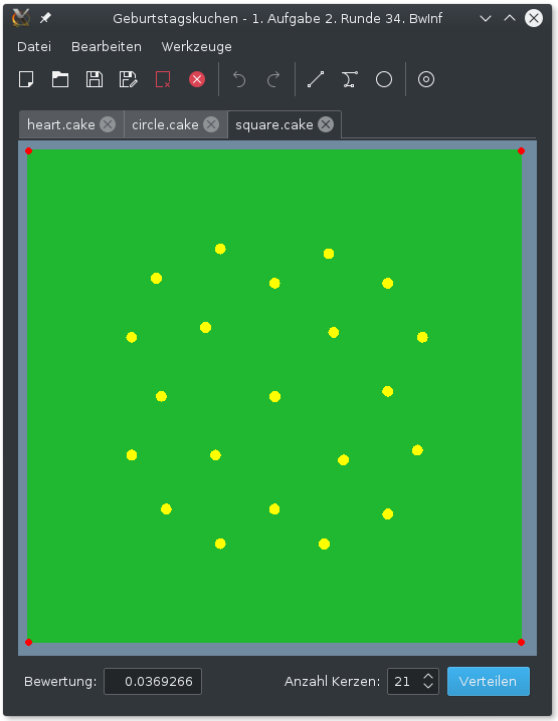
3.2 Quadrat

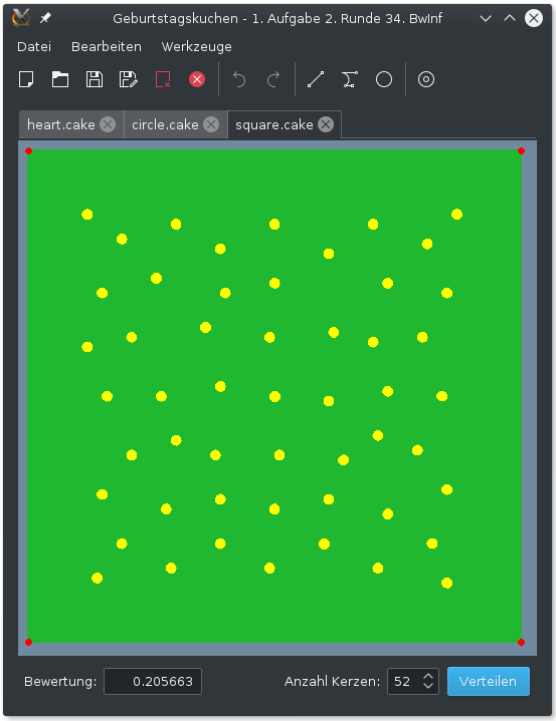
Eine sehr simple Form ist natürlich das Quadrat. Hier einmal das Rating nach der Verteilung einer bestimmten Anzahl an Kerzen:

Anzahl Kerzen	Rating
10	0.052762
20	0.036565
30	0.049570
40	0.185932
50	0.206265
60	0.205176
70	0.240800
80	0.240608
90	0.243805



Hier das ganze noch als mehr oder weniger schöne Grafik mit 21 und 52 Kerzen:

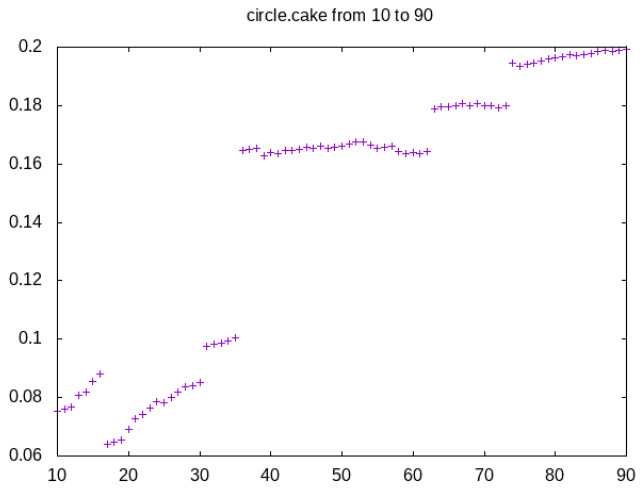




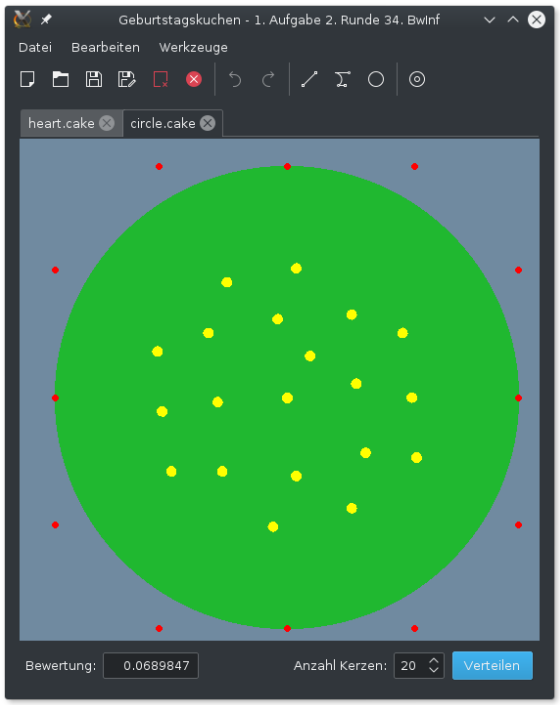
3.3 Kreis

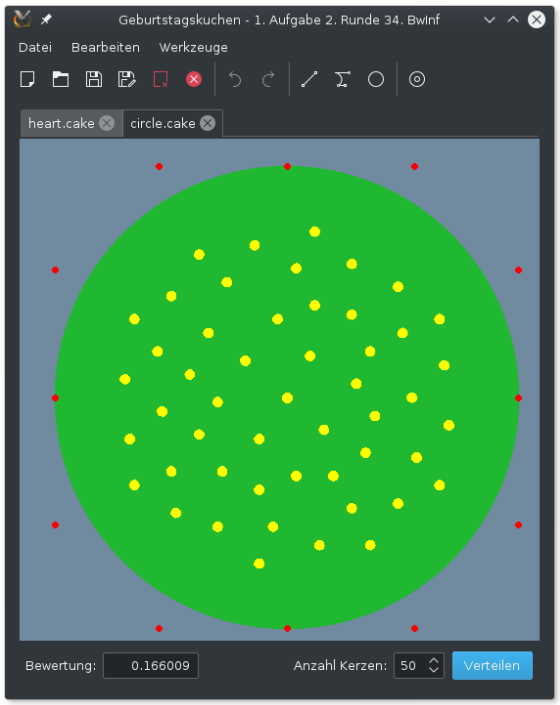
Hier das ganze nochmal mit einem Kreis:

Anzahl Kerzen	Rating
10	0.075190
20	0.068985
30	0.085052
40	0.163995
50	0.166009
60	0.163986
70	0.179887
80	0.196279
90	0.199241



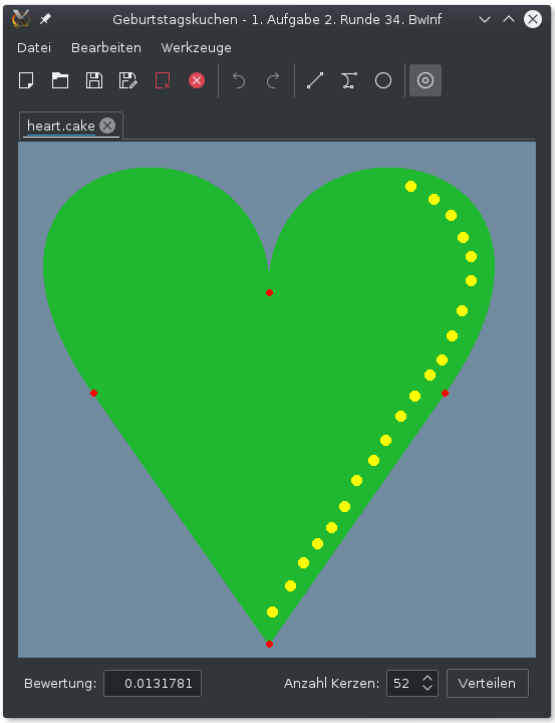
Hier das ganze noch als mehr oder weniger schöne Grafik mit 20 und 50 Kerzen:





3.4 Herz mit Luis Kerzenverteilung

Ich hab mal ein Herz genommen und so wie Luis frei Hand Kerzen darauf verteilt. Das Ergebnis ist:



Da hierbei die Kerzen fast alle denselben Abstand zueinander und zum Rand haben, ist diese Verteilung mit 0.0131781 sehr gut. Ich habe mich jedoch dagegen entschieden, meinen Algorithmus die Kerzen auch nach einem ähnlichen Schema verteilen zu lassen, da dies menschlich betrachtet doch eher hässlich ist. Mir ist jedoch bewusst, dass sich auf einem negativ geladenen Metallkörper alle überschüssigen Elektronen an der Oberfläche platzieren, und an Luis Kerzenverteilung sieht man auch warum.

3.5 Erklärung der Plots

Auf der x-Achse aller Plots ist die Anzahl der Kerzen und auf der y-Achse das Rating, nachdem mein Algorithmus die Kerzen verteilt hat. Was man bei allen Plots sehen kann, ist eine Tendenz, dass die Gleichmäßigkeit mit der Anzahl der Kerzen sinkt. Dies ist damit zu begründen, dass, je mehr Kerzen es gibt, desto größer werden die Unterschiede zwischen den Abständen zwischen den einzelnen Kerzen. Da mein Programm nur die größte Abweichung betrachtet, muss der Wert zwangsweise sinken.

Eine andere Auffälligkeit ist, dass es immer mal wieder mehr oder weniger gerade Linien in den Plots gibt. Auch dies kann man anhand des Algorithmuses erklären. Beim anfänglichen Grid, in dem noch keine Kerzen sind, gibt es meist in der Mitte mehrere Felder mit annähernd demselben Wert. Die Anzahl dieser Felder wird zwar reduziert, sobald ich Kerzen dazusetze, aber im Grunde genommen gibt es solche Felder immernoch. Wenn nun so viele Kerzen vorhanden sind, dass diese Felder einen sehr hohen Wert bekommen haben, gibt es einen solchen Sprung im Graphen.