Feodor-Lynen-Gymnasium
Planegg

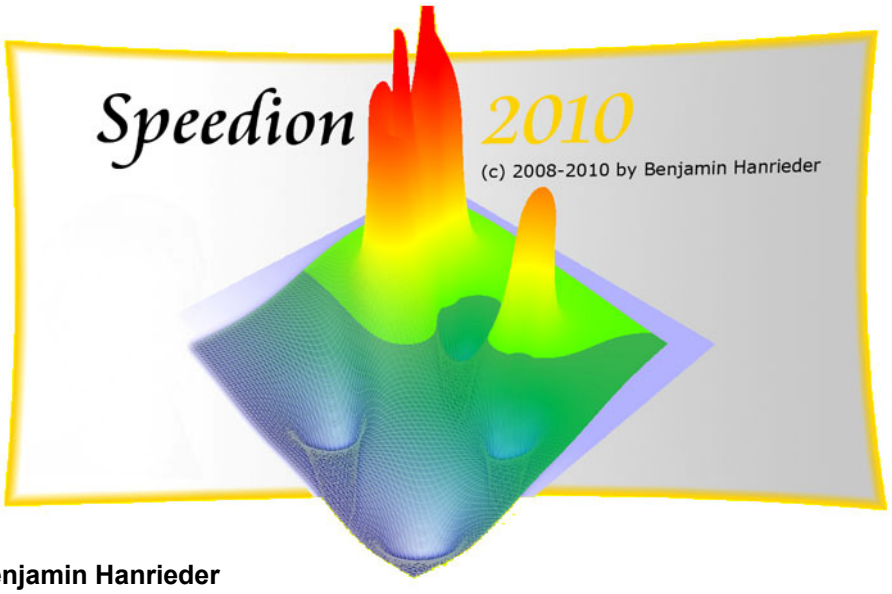
Kollegstufenjahrgang 2008/2010

FACHARBEIT

aus dem Fach

Physik

Thema: **Bewegte Ladungen im elektrischen Feld – Speedion 2010, das Programm zur physikalischen Simulation**



Verfasser: **Benjamin Hanrieder**

Leistungskurs: **Physik 2008/2010**

Kursleiter: **Florian Bell**

Abgabetermin: **29. Januar 2010**

Schriftlicher Teil:

Erzielte Note:..... in Worten:.....

Erzielte Punkte: ..... in Worten:.....

Mündliche Prüfung:

Prüfungstag:.....

Erzielte Note:..... in Worten:.....

Erzielte Punkte: ..... in Worten:.....

Abgabe im Sekretariat am:.....

.....

(Unterschrift des Kursleiters)

Inhaltsverzeichnis

I. Einleitung

Es ist nicht leicht, jemandem einen abstrakten Begriff wie „Potenzial“ oder „Elektrisches Feld“ zu erklären. Am Besten gelingt dies durch grafische Anschauung.
Leider existieren bis jetzt noch keine Programme auf dem Markt, welche es Neulingen ermöglichen, beliebige elektrische Felder anschaulich und benutzerfreundlich darzustellen.
Für die folgende Facharbeit habe ich ein Computerprogramm erstellt, welches einem Benutzer den Feldbegriff und den Potenzialbegriff anschaulich macht und darüber hinaus komplexe Simulationen ermöglicht.
Das Programm berechnet innerhalb von Millisekunden einen Potenzialverlauf in räumlicher Darstellung, nachdem der Benutzer frei definierbare Ladungen auf einer Ebene platziert hat. Doch selbst für Computereinsteiger ist die Bedienung des Programms einfach: Ladungen können zum Beispiel intuitiv per Mausklick auf einer beliebigen Stelle der Ebene platziert werden. Mit der Maus oder der Tastatur kann dann der erzeugte Potenzialgraph so gedreht werden, dass aus jedem Blickwinkel das Potenzialgebirge eingesehen werden kann.
Auch Äquipotentiallinien und Feldlinien werden durch das Programm problemlos dargestellt und können zur besseren Übersicht an- oder abgeschaltet werden.
Nicht nur statische Ladungen, sondern auch eine bewegte Ladung mit definierbarer Position, Richtung und Anfangsgeschwindigkeit, kann auf dem Feld platziert werden.
Die Flugbahn dieser bewegten Ladung wird automatisch unter Berücksichtigung der Feldkräfte, die durch die statischen Ladungen verursacht werden, berechnet und auf dem Bildschirm angezeigt.
Es ist sogar möglich, das Teilchen in Zeitlupe auf dem Monitor zu verfolgen. Ein Diagramm zeigt dann auch die Momentangeschwindigkeit des Teilchens über der Zeit an.
Das Programm dient aber nicht nur zur anschaulichen Erklärung, sondern kann auch noch für komplexe Simulationen benutzt werden. So können Ladungen präzise definiert werden und auch komplexe physikalische Effekte, wie zum Beispiel die Bremsstrahlung berücksichtigt das Programm auf Wunsch.
Schließlich kann an die statisch positionierten Ladungen auch eine Wechselspannung mit einer wählbaren Frequenz angelegt werden, welche es beispielsweise ermöglicht die Bewegung eines Ions in der Paul-Falle zu simulieren:
Die Flugbahn, sowie der zeitliche Geschwindigkeitsverlauf des in der Paul-Falle gefangenen Ions, kann dabei auf dem Bildschirm in Zeitlupe verfolgt werden.
Speicher-, Druck- und Bild-Exportierfunktionen runden das Programm ab.

II. Benutzeroberfläche und Bedienung des Programms

Die Benutzeroberfläche ist von oben nach unten in 5 Teile aufgeteilt (siehe Anhang, S.26):

1.) Die Menüleiste
2.) Die Projektleiste
3.) Der Arbeitsbereich
4.) Die Symbolleiste zum Arbeitsbereich
5.) Simulations-Einstellungen

1.) Die Menüleiste

Mit der Menüleiste können Sie, wie man es auch von anderen Programmen gewohnt ist, auf die meisten Programmfunktionen zugreifen. Sie dient als Ergänzung zu den Symbolleisten, welche ich im Folgenden ausführlich erklären werde.

2.) Die Projektleiste

a) Projekt


Erstellt ein neues Projekt. Ein Projekt kann mehrere Arbeitsbereiche (Tabs) enthalten.


Öffnet ein bestehendes Projekt.


Mit einem Klick auf dieses Symbol laden Sie das Beispielprojekt. Die Datei für das Beispielprojekt ist direkt im Programm enthalten und kann nicht verändert werden. Das Beispielprojekt dient dazu, sich schnell einen Überblick über die Programmfunktionen zu verschaffen.


Speichert das Projekt als Datei auf der Festplatte ab.

b) Tabs

Jedes Tab repräsentiert einen unabhängigen Arbeitsbereich innerhalb eines Projekts. Die Tabs dienen dazu, schnell und einfach verschiedene elektrostatische Feldkonfigurationen zu vergleichen und zwischen verschiedenen Feldern umzuschalten. Dieser Aufbau ist vergleichbar mit dem von MS-Excel. Ein „Projekt“ entspricht einer Excel-Datei, welche

mehrere Spreadsheets enthält. Ein „Arbeitsbereich“ (Tab) entspricht dabei einem Excel-Spreadsheet (Tab). Ein Projekt kann bis zu 20 verschiedene Tabs enthalten.


Erstellt einen neuen Arbeitsbereich.


Mit diesem Schaltknopf können Sie den aktuellen Arbeitsbereich kopieren. Damit können Sie schnell ähnliche elektrische Felder ohne Aufwand erstellen.


Druckt die aktuelle Ansicht der zwei Graphen. Beide Graphen werden auf eine DIN-A4-Seite gedruckt.


Mit einem Klick auf diesen Knopf löschen Sie das aktive Tab. Bitte beachten Sie, dass ein bereits gelöschtes Tab nicht wiederhergestellt werden kann.


Sie können jedem Tab einen aussagekräftigen Namen geben, um bei größeren Projekten den Überblick zu behalten.

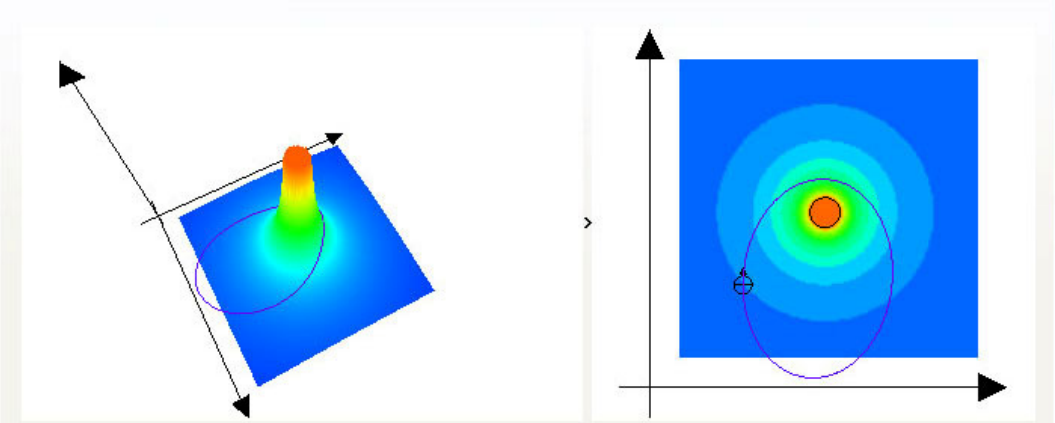

Wenn dieser Knopf aktiviert ist, werden die Vorzeichen aller positiven und negativen Ladungen in ihrer Darstellung vertauscht. Ein anschaulicher Nebeneffekt ist, dass das Elektron nun in das positive „Potenzialloch“ fällt und somit der geläufigen Darstellung von Massen im Gravitationsfeld gleicht.

Aktivieren Sie diese Schaltfläche, um die Einheiten aller Ladungen in Elementarladungseinheiten anzuzeigen. Standardmäßig werden alle Ladungen in Coulomb angezeigt. Programm-intern wird immer mit SI-Einheiten gerechnet.

3.) Der Arbeitsbereich

In einem Projekt kann, wie beschrieben, zwischen verschiedenen Arbeitsbereichen (Tabs) umgeschaltet werden. Hier ist ein Projekt mit 3 Arbeitsbereichen („Paul Falle“, „3 Ladungen“ und „1 Ladung“) dargestellt.

Jedes Tab enthält seine eigene Konfiguration von Ladungen und Probeladungen mit individuellen Eigenschaften.



Auf der linken Seite sehen Sie den Potenzialverlauf des elektrischen Feldes in der 3-dimensionalen Darstellung; auf der rechten Seite in der 2-dimensionalen Darstellung. Mit dem kleinen Pfeil (>) zwischen den beiden Feldern lässt sich die Anzeige zu Gunsten einer Darstellung vergrößern oder verkleinern. Ziehen Sie dazu den Pfeil nach links oder nach rechts.

Um eine neue Ladung hinzuzufügen, führen Sie einen Rechtsklick auf einem beliebigen Graphen an der Stelle aus, an der die Ladung erscheinen soll. Wählen Sie dann aus dem Menü den Betrag der Ladung aus, oder geben Sie einen beliebigen Betrag ein.

Bewegt man die Maus über den 2D-Graphen, dann wird automatisch im 3D-Graphen eine semitransparente blaue Fläche auf der Höhe des Potenzials eingeblendet („Überschwemmung“).

Sie können den 2D-Graphen verschieben, indem Sie auf einen beliebigen Punkt auf dem Graphen klicken und in die gewünschte Richtung ziehen („click & drag“).

Auf gleiche Weise kann der 3D-Graph beliebig gedreht werden. Ziehen Sie dazu mit der Maus einen beliebigen Punkt des 3D-Graphen.

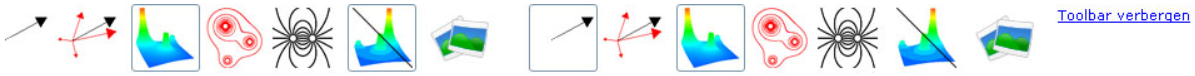
Mit dem Mausrad können Sie in einen Graphen hinein- und herauszoomen. Der Mauszeiger muss sich dazu über dem Graphen befinden, der gezoomt werden soll.

Alternativ können die Graphen auch per Tastatur gesteuert werden.
Die Tasten W-A-S-D, Q-E sind analog zum folgendem Schema für den 3D-Graphen zuständig. Mit den Tasten I-J-K-L, U-O können Sie den 2D-Graphen steuern.

Mit den Tasten, die mit einem „+“ markiert sind, können Sie näher in den Graphen hineinzoomen; „-“ zoomt heraus[1].

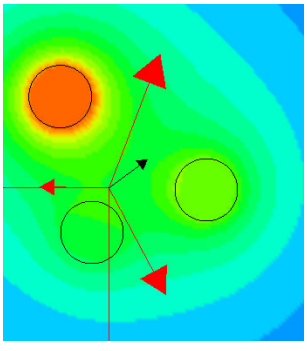
Das Programm erlaubt es, die Länge und Breite des Koordinatensystems von 10^{-10}m bis 100m einzustellen. So kann beispielsweise sowohl das Coulomb-Feld eines Atoms simuliert werden, als auch makroskopische Probleme in Geräten wie Detektoren oder Beschleunigern.

4.) Die Symbolleiste zum Arbeitsbereich



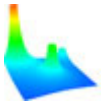
Diese Symbolleiste besteht aus zwei mal 7 Schaltflächen. Mit den ersten 7 Knöpfen lässt sich die Darstellung des 3D-Graphen bearbeiten und mit den zweiten 7 Knöpfen die Darstellung des 2D-Graphen.

Es ist möglich, die Feldstärke an einem beliebigen Punkt auf dem Graphen vektoriell darzustellen. Aktivieren Sie dazu einen der folgenden zwei Knöpfe und fahren Sie dann mit der Maus über einen Punkt auf dem Graphen.

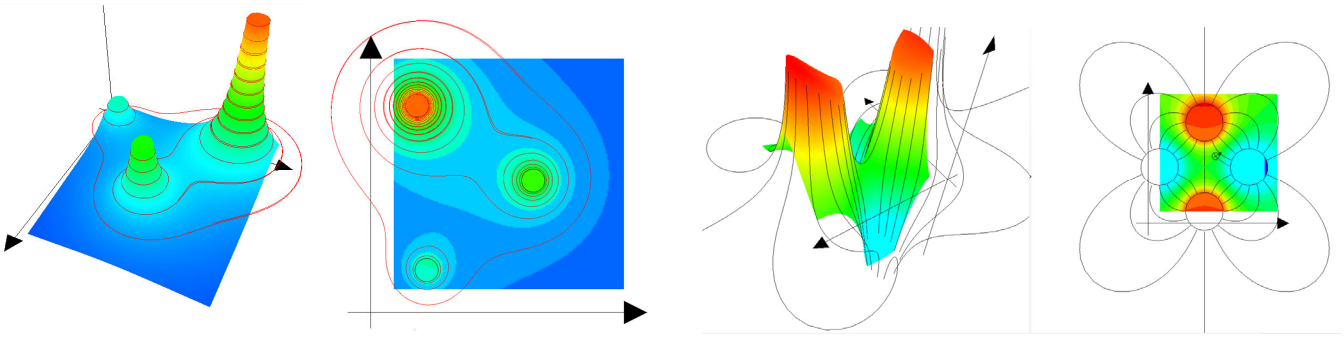



Wenn dieser Knopf aktiviert ist, wird der aus allen Ladungen resultierende Feldstärkevektor als schwarzer Pfeil angezeigt.

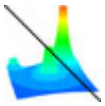

Aktivieren Sie diesen Knopf, um auch die Feldstärken jeder einzelnen Ladung anzuzeigen (rot). Der resultierende Gesamtfeldstärkevektor wird schwarz angezeigt.


Klicken Sie hier, um die Anzeige des Graphen zu deaktivieren. Wird der Graph deaktiviert, verbessert sich die Rechenleistung des Programms. Dies kann sinnvoll sein, um das Wechselfeld (z.B. „Paul-Falle“) schneller zu berechnen.

Wollen Sie auch die Äquipotentiallinien der elektrostatischen Ladungsverteilung anzeigen, klicken Sie hier.



Lassen Sie sich die elektrischen Feldlinien anzeigen.



Jedem Potenzialniveau wird eine Farbe zugeordnet. Schalten Sie zwischen stufenlosem und gestuftem Farbverlauf um.

Exportieren Sie die momentane Anzeige des Graphen als Bilddatei auf Ihre Festplatte (Screenshot-Funktion).

Toolbar verbergen Schaffen Sie mehr Platz für den Arbeitsbereich, indem Sie die Symbolleiste und die Feld-Einstellungen verbergen.

5.) Simulations-Einstellungen

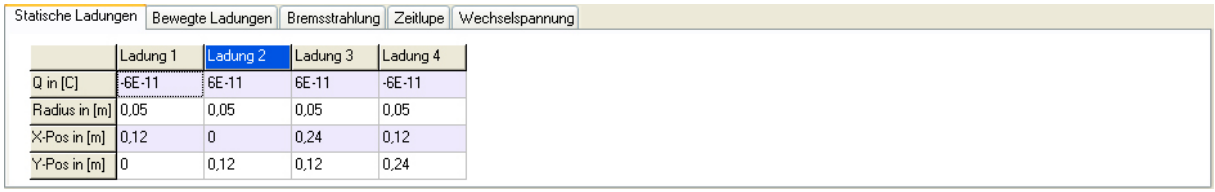
Zunächst werden nur zwei Registrierkarten angezeigt: „Statische Ladungen“ und „Bewegte Ladung“. Sobald man die bewegte Ladung aktiviert, werden weitere drei Registrierkarten eingeblendet.

Mit diesen 5 Registrierkarten lassen sich die physikalischen Parameter der Ladungen einstellen. Alle Einstellungen werden in Echtzeit aktualisiert, vom Programm übernommen, neu berechnet und grafisch dargestellt (siehe Kapitel IV, „Multi-Threading“).

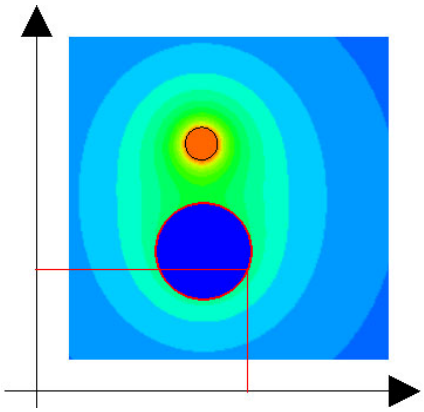
a) Statische Ladungen

Hier kann man den Betrag, die Position und den Radius einer Ladung festlegen. Klicken Sie dazu einfach auf einen Wert in der Tabelle, um den Parameter zu verändern.

Wenn Sie im 2D-Graphen mit der Maus eine Ladung anklicken, wird diese Ladung in der Tabelle blau markiert. Das funktioniert auch umgekehrt. So können Sie immer sicher sein, die richtige Ladung zu bearbeiten und behalten den Überblick.



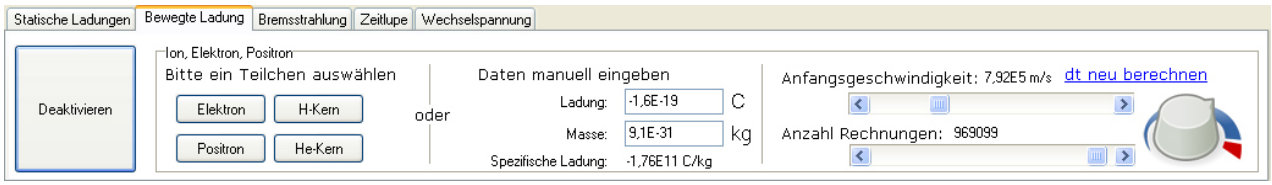
Sie können die meisten Eigenschaften der Ladung aber auch bearbeiten, ohne dass Sie die Tabelle benutzen müssen. Wählen Sie dazu mit der Maus eine Ladung direkt auf dem 2D-Graphen aus.



- Um dann die Ladung zu verschieben, ziehen Sie die Ladung auf eine andere Position.
- Sie können die Ladung löschen, indem Sie auf der Tastatur die DEL-Taste (ENTF-Taste) drücken und mit „J“ bestätigen.
- Den Radius einer Ladung können Sie verändern, indem Sie mit dem Mauszeiger auf den Rand einer Ladung fahren. Der Rand wird dann rot gefärbt. Verändern Sie den Radius der Ladung, indem Sie den Rand bei gedrückter Maustaste größer oder kleiner ziehen.
- Wenn Sie mit dem Bearbeiten der Ladung fertig sind, klicken Sie auf eine beliebige Stelle auf dem Graphen, um den Vorgang abzuschließen.

b) Bewegte Ladung

Es kann auch eine bewegte Ladung (Probeladung) auf dem Feld platziert werden.



Wählen Sie dazu die Registrierkarte „Bewegte Ladung“ aus und klicken Sie auf „Aktivieren“, um die Probeladung auf dem Feld zu platzieren.

Die Parameter der Probeladung können entweder manuell eingegeben werden, oder man wählt ein Teilchen aus der vorgegebenen Gruppe aus.

Rechts lässt sich die Anfangsgeschwindigkeit des Teilchens auswählen. Diese Geschwindigkeit hat das Teilchen an seiner Startposition.

Es empfiehlt sich, die Anzahl der Rechnungen je nach Szenario manuell einzustellen. Je mehr Rechnungen, desto weiter wird die Flugbahn berechnet.

Die Zeitdauer Δt wird vom Programm automatisch so optimiert, dass die Simulation gute Ergebnisse (d.h. möglichst glatte Bahnverläufe) bei möglichst geringer Rechendauer liefert. Sie kann nicht vom Benutzer manuell eingestellt werden. Nachdem Sie die Werte der Probeladung verändert haben, klicken Sie auf dt neu berechnen, damit das Programm Δt optimal einstellt.

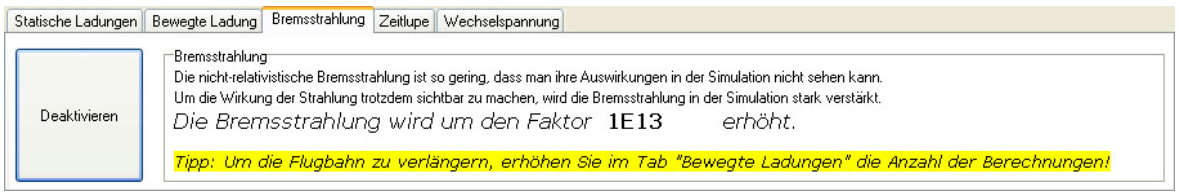
Wenn die bewegte Ladung aktiviert wird, erscheint ein neues Symbol  auf dem 2D-Graphen. Dieses Symbol liegt auf der Startposition der Probeladung und zeigt die Richtung des Vektors der Anfangsgeschwindigkeit an.

Um die Position der Probeladung zu verändern, fahren Sie mit der Maus über dieses Symbol. Es wird dann seine Farbe von schwarz auf rot ändern. Jetzt können Sie die Ladung mit der Maus auf eine beliebige Stelle auf dem Feld ziehen.

Die Bewegungsrichtung der Probeladung können Sie ändern, indem Sie mit der Maus über den Pfeil des Symbols fahren, bis dieser weiß angezeigt wird. Klicken Sie jetzt mit der Maus auf den Pfeil und ziehen Sie den Pfeil in die gewünschte Richtung. Lassen Sie dann die Maustaste wieder los.

c) Bremsstrahlung

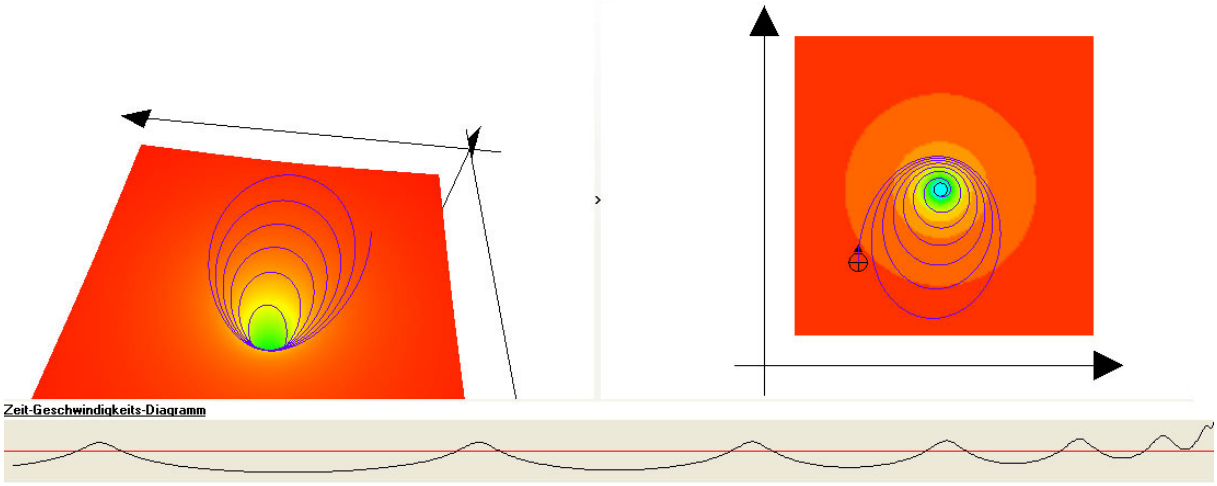
Bei der Beschleunigung von Ladungen strahlen diese Energie in Form von elektromagnetischen Wellen ab.
Durch diesen Energieverlust verändert sich die Flugbahn des Teilchens.



Aktivieren Sie die Bremsstrahlung, um die Flugbahn unter Berücksichtigung der Bremsstrahlung neu zu berechnen.

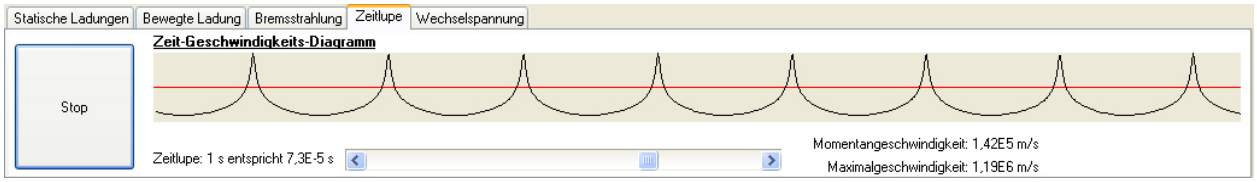
Bitte beachten Sie, dass die Auswirkung der (nicht-relativistischen) Bremsstrahlung in den kurzen Zeiträumen, die betrachtet werden, sehr gering ist und deshalb zur besseren Veranschaulichung in der Simulation der Effekt verstärkt dargestellt wird.

Das Programm schlägt automatisch einen geeigneten Verstärkungsfaktor vor, der aber manuell noch bearbeitet werden kann. Klicken Sie dazu einfach auf die Zahl und geben Sie einen anderen Wert ein.



d) Zeitlupe

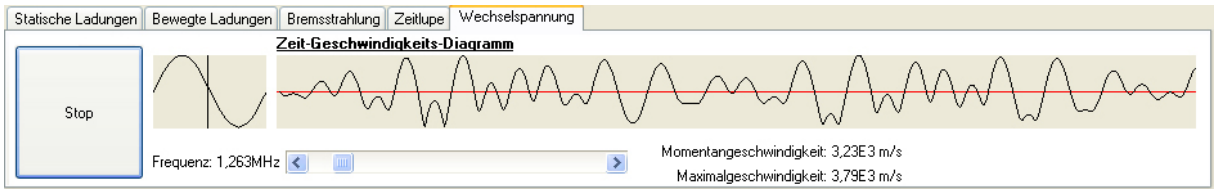
Es ist möglich, den zeitlichen Verlauf der Bewegung der Probeladung zu verfolgen. Starten Sie dazu die Zeitlupe mit einem Klick auf Start.



Dabei wird automatisch auch ein Zeit-Geschwindigkeitsdiagramm erstellt.
Der Wert der Momentangeschwindigkeit und der Maximalgeschwindigkeit wird unterhalb des Diagramms angezeigt. Zur besseren Übersicht wird die halbe Maximalgeschwindigkeit als rote Referenzlinie gekennzeichnet.

e) Wechselspannung

Mit Hilfe der Wechselspannungsfunktion ist es möglich, noch komplexere Simulationen, wie zum Beispiel die Simulation einer Paul-Falle, zu realisieren.



Wenn Sie auf „Start“ drücken, wird ein sinusförmiger Polaritätswechsel an jeder statischen Ladung erzeugt. Die Berechnungen werden in Echtzeit durchgeführt. Es empfiehlt sich daher, bei rechenschwachen Computern, wie beschrieben, die Anzeige von Feldlinien und Äquipotentiallinien zu deaktivieren.

III. Physik und Programmierung – Die wichtigsten Grundlagen

Im Folgenden werden die relevanten physikalischen Gesetze und deren Realisierung im Programm dargestellt.

1.) Berechnung des Coulomb-Potenzials

Das Programm stellt den Potenzialverlauf eines Coulomb-Feldes anhand einer vom Benutzer definierten, ebenen Ladungsverteilung grafisch dar.

Das resultierende Coulomb-Potenzial wird an mehr als 20.000 Orts-Punkten berechnet, indem die Potenziale der einzelnen statischen Ladungen berechnet und aufaddiert werden[2].

$$\varphi = \sum_{i=0}^{n} \frac{1}{4\pi\varepsilon_0} \cdot \frac{Q_i}{r_i}$$

```
GLfloat Potenzial(GLfloat x, GLfloat y) //Potenzialfunktion: [x, y] in m;
[Q] in C; [result] in V;  //x,y sind die Positionen auf dem Feld;
Maximale Werte: 10.0 (Breite des Feldes)
{
    double result = 0;
    for (int i = 0; i<ladungen anzahl[curt];i++) //Jede einzelne
statische Ladung wird nacheinander betrachtet
    {
        if (sqrt(pow(x-ladungen Pos[curt][i][0],2)+pow(y-
ladungen_Pos[curt][i][1],2)) <= ladungen_radius[curt][i]) result +=
D*vorzeichen*ladungen Q[curt][i] /ladungen radius[curt][i]; //Wenn der
Punkt innerhalb des Radius der Ladung liegt, wird das Potenzial des
Radius zurückgegeben
        else
            result += D * vorzeichen*ladungen Q[curt][i] * (1/sqrt(pow(x-
ladungen_Pos[curt][i][0],2)+pow(y-ladungen_Pos[curt][i][1],2)));
    }
    return result;
}
```

2.) Berechnung der elektrischen Feldstärke

Die Feldstärke ist definiert als die Feldkraft auf eine Probeladung geteilt durch den Betrag der Probeladung. Durch vektorielle Addition der einzelnen Coulomb-Feldstärken erhält man die Gesamtfeldstärke am Punkt (x,y) [2].

$$\vec{E} = \sum_{i=0}^{n} \frac{\vec{F_i}}{q}$$

$$\vec{E} = \sum_{i=0}^{n} \frac{1}{4\pi\varepsilon_0} \cdot \frac{Q_i}{|\vec{r_i}|^2} \cdot \vec{n_i}$$

```
GLfloat Feldstaerke(GLfloat x, GLfloat y, double &alpha, int ladung, bool
probeladungnegativ) {
alpha = 0;
GLfloat E;
GLfloat X2 = 0;
GLfloat Y2 = 0;

    for (int i = (ladung>0?ladung-1:0); i <
(ladung>0?ladung:ladungen_anzahl[curt]);i++)
//Folgende Rechnungen werden für jede Ladung wiederholt
    {
        //Wenn innerhalb des Radius: Feldstärke = 0
        if (sqrt(pow(x-ladungen Pos[curt][i][0],2)+pow(y-
ladungen_Pos[curt][i][1],2)) < ladungen_radius[curt][i])
        {
            X2 += 0;
            Y2 += 0;
        }
        else
            //sonst
        {
            // 1. Länge ausrechnen
            E = (probeladungnegativ?-1:1)*D *
vorzeichen*ladungen Q[curt][i] / (pow(x-
ladungen Pos[curt][i][0],2)+pow(y-ladungen Pos[curt][i][1],2));

            // 2. Richtung ausrechnen und zum Vektor (X2, Y2) addieren
            //Grenzfall: wenn x-ladungen Pos[curt][i][0]==0 dann atan =
90 => sin = +/- 1; cos = 0

            X2 += ((x-ladungen Pos[curt][i][0])==0 ? 0 : cos (atan((y-
ladungen_Pos[curt][i][1])/(x-ladungen_Pos[curt][i][0])) ) * E *(x-
ladungen_Pos[curt][i][0] >= 0 ? 1 : -1);
            Y2 += ((x-ladungen Pos[curt][i][0])==0 ? ((y-
ladungen_Pos[curt][i][1])>0?1:-1) : sin (atan((y-
ladungen_Pos[curt][i][1])/(x-ladungen_Pos[curt][i][0])) ) * E *(x-
ladungen Pos[curt][i][0] >= 0 ? 1 : -1);
        }
    }
alpha = X2==0?((Y2)>0?1:-1)*M PI/2:atan(Y2/X2) + ((X2)<0?M PI:0) ;
return sqrt(pow(X2,2)+pow(Y2,2)); //Länge der Feldstärke
}
```

3.) Bewegte Ladung im Coulomb-Feld

In diesem statischen Coulomb-Feld kann eine (Probe-)Ladung platziert und deren Bewegungsverlauf (Ort und Geschwindigkeit) beobachtet werden.

Der Bewegungsverlauf der Probeladung wird berechnet, indem man einzelne (sehr) kurze Zeitschritte Δt betrachtet: Jede Probeladung hat vor und nach jeder dieser Berechnungen einen Ort und einen Geschwindigkeitsvektor. Die folgende Funktion wird in einer Schleife immer wieder ausgeführt. Dabei sind die Resultate der Funktion (Ort und Geschwindigkeitsvektor) immer die Parameter des jeweils nächsten Aufrufs der Funktion. Die Parameter des ersten Aufrufs können durch den Benutzer vorgegeben werden (siehe Punkt II. 5.) b) dieser Facharbeit).

Die folgende (Programm-)Funktion berechnet den neuen Ort und den neuen Geschwindigkeitsvektor der Probeladung nach dem kurzen Zeitintervall Δt. Zum besseren Verständnis wird diese Funktion gegenüber der Funktion im Programm verkürzt dargestellt.

```
void Ionablenkung( double &v1, double &alpha, double q, double m, double
dt, GLfloat &posx, GLfloat &posy)
{
```

Parameter der Funktion:

v1	=	Betrag der Geschwindigkeit vor der Berechnung
alpha	=	Winkel des Geschwindigkeitsvektors vor der Berechnung
q	=	Betrag der Probeladung
m	=	Masse der Probeladung
dt	=	Zeitdauer Δt
posx,posy	=	Koordinaten der Probeladung vor der Berechnung

```
//1. Basics
//Vektor 1 : v1, alpha
         //gegeben (siehe oben)
//Vektor 2 (Ablenkung) : v2, beta
        // F = m * a = E * q; a = (E * q)/m; spezladung = q/m
    GLfloat x = posx;
    GLfloat y = posy;
```

Es wird zuerst der Vektor der Geschwindigkeitsänderung (Δv) berechnet, der durch die Feldkraft verursacht wird. Dieser Vektor besteht aus:

v2 = Betrag der Geschwindigkeit

beta = Winkel

Aus der (durch alle statischen Ladungen resultierenden) Coulomb-Kraft F wird die Beschleunigung a der Ladung berechnet.

$$F = E \cdot q$$

$$F = m \cdot a$$

$$\rightarrow a = \frac{E}{q \cdot m}$$

```
double a = Feldstaerke(x,y,beta,0,false)*(q/m);
// Diese Funktion gibt die Beschleunigung a und den Winkel beta zurück.
```

Die Beschleunigung multipliziert mit der sehr kurzen Zeitspanne Δt ergibt die (vektorielle) Geschwindigkeitsänderung Δv.

```
        double v2 = a * dt;
```

Aus dieser Geschwindigkeitsänderung wird jetzt der neue Geschwindigkeitsvektor der Probeladung berechnet. Dazu wird der Geschwindigkeitsvektor, den die Probeladung bereits vor der Rechnung hatte, und der Geschwindigkeitsvektor Δv addiert.

```
//2. Vektoren addieren
        //Zur einfacheren Berechnung werden die Koordinaten der Vektoren
bestimmt.
        double x1,x2,y1,y2;
        double vres;
        double alphares;

        //Und los

        y1 = sin(alpha)*v1 ; //Vektor 1: (x1,y1)
        x1 = cos(alpha)*v1;
        y2 = sin(beta)*v2; //Vektor 2: (x2,y2)
        x2 = cos(beta)*v2;

//Resultierender Vektor
        vres = sqrt(pow(x1+x2,2)+pow(y1+y2,2)); //Pythagoras
        alphares = atan((y1+y2)/((x1+x2)==0?0.01:(x1+x2)));

//3. Ergebnis:
alpha = alphares;
v1 = vres;
```

Die Ortsänderung ergibt sich aus der neuen Geschwindigkeit multipliziert mit Δt. Diese Ortsänderung wird zur vorherigen (x,y)-Position addiert.

```
posx +=  v1*cos(alpha)* dt;
posy +=  v1*sin(alpha)* dt;
}
```

4.) Bremsstrahlung

Wird eine elektrische Ladung beschleunigt, strahlt sie Energie in Form von elektromagnetischen Wellen ab, was auf Kosten ihrer kinetischen Energie geschieht und somit zum Geschwindigkeitsverlust der Probeladung führt.
Diese Strahlung nennt man auch Bremsstrahlung.
Jede Geschwindigkeitsänderung – sei es in Richtung oder Betrag – einer elektrischen Ladung erzeugt also Strahlung und führt somit zum Verlust an kinetischer Energie.

Das Programm berechnet bei jedem inkrementellen Bewegungsschritt über Δt, a) die Beschleunigung der Probeladung, b) die daraus resultierende Bremsstrahlungsleistung, c) den Verlust an kinetischer Energie und d) die daraus resultierende neue Geschwindigkeit der Probeladung.

zu a) Die Beschleunigung der Probeladung resultiert aus der Feldkraft. Die genaue Berechnung wurde bereits unter dem Punkt III. / 3.) dieser Facharbeit erklärt.

zu b) Die abgestrahlte Verlustleistung des beschleunigten (nicht-relativistischen) Teilchens ergibt sich aus der Larmor-Formel[3]:
c: Lichtgeschwindigkeit

$$P_{larmor} = \frac{2}{3} \cdot \frac{q^2}{4\pi\varepsilon_0} \cdot \frac{a^2}{c^3}$$

zu c) Aus der abgestrahlten Leistung lässt sich der resultierende Energieverlust während der Zeitspanne Δt berechnen:

$$E_{larmor} = P_{larmor} \cdot \Delta t$$

```
double Larmor(double a, double q, double dt) //Berechnet E_larmor
{
double P = (2/(3*pow(var_c,3)))*D*pow(a,2)*pow(q,2);
double E = P * dt;
return E;
}
```

zu d) Schließlich bekommt man durch folgende Umformungen die neue, resultierende Geschwindigkeit der Probeladung:

$$E_{neu} = \frac{1}{2} m \cdot v_{neu}^2; \quad E_{alt} = \frac{1}{2} m \cdot v_{alt}^2$$

$$E_{neu} = E_{alt} - E_{larmor}$$

$$\rightarrow \frac{1}{2} m \cdot v_{neu}^2 = \frac{1}{2} m \cdot v_{alt}^2 - E_{larmor}$$

$$\rightarrow v_{neu} = \sqrt{v_{alt}^2 - \frac{2 \cdot E_{larmor}}{m}}$$

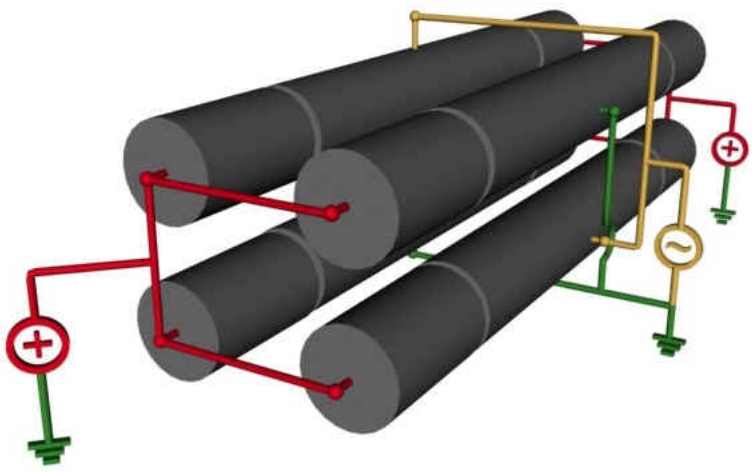
```
double wurzel = pow(v1,2) - Betrag(2*Larmor(a,q,dt)/m)*larmor_faktor;
v1 = (v1<0?-1:1)*sqrt(wurzel<0?0:wurzel);  //neue Geschwindigkeit nach
der Abgabe von Energie durch Bremsstrahlung
```

Das Programm berücksichtigt nun dieses Resultat bei der Berechnung des nächsten Bahnpunktes. Die Bahn des strahlungsgebremsten Teilchens weicht daher von der des ungebremsten Teilchens ab.
Kreist ein ungebremstes Teilchen z.B. in einer Ellipsenbahn um eine statische Ladung, dann führt das Einschalten des Bremseffekts dazu, dass die Probeladung in einer Spiralbahn allmählich in den Potenzialtopf der statischen Ladung fällt (siehe Bild S.11)

Verstärkungsfaktor:
Die Effekte der Bremsstrahlung sind in den betrachteten nicht-relativistischen Fällen extrem klein. Die Abweichungen von der Bahn der ungebremsten Ladung wären deshalb erst nach sehr langen Zeiten sichtbar in der Darstellung.
Zur Verbesserung der Anschaulichkeit wird deshalb ein Verstärkungsfaktor eingeführt. Dieser Faktor bewirkt, dass in der Darstellung schnell etwas vom Bremseffekt sichtbar wird. Das Programm schlägt automatisch einen geeigneten Verstärkungsfaktor vor.

5.) Lineare Paul-Falle

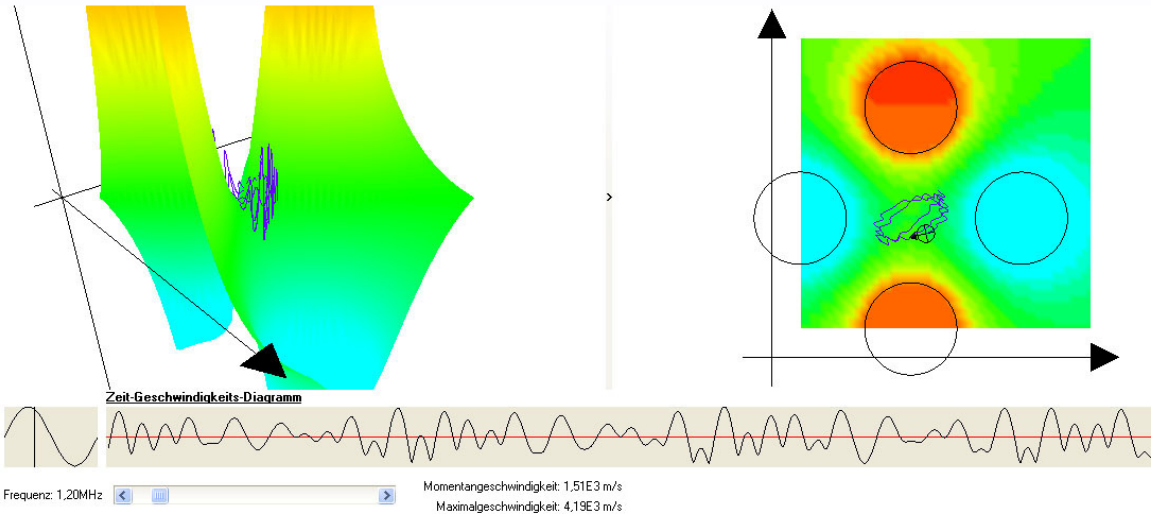
Eine Paul-Falle ermöglicht es, geladene Teilchen (z.B. Ionen) mit Hilfe von elektrischen Wechselfeldern auf begrenztem Raum zu speichern.
Wolfgang Paul erhielt für die Entwicklung der Paul-Falle 1989 den Nobelpreis in Physik[4].
Ein Spezialfall der Paul-Falle ist die lineare Paul-Falle[5]. Diese besteht, analog zu dem nebenstehenden Schema, aus 4 parallelen Stäben.
Jeder der 4 Stäbe ist in 3 Segmente



(Elektroden) aufgeteilt: ein Mittelstück und zwei Endstücke. Da die Endstücke gleichnamig zum Ion gepolt sind, also z.B. positiv gepolt sind bei einem positiven Ion, kann das gefangene Ion die Paul-Falle in Längsrichtung nicht verlassen.
An die 4 Mittelstücke wird eine Wechselspannung angelegt, wobei jeweils die 2 diagonal gegenüberliegenden Elektroden gleich gepolt sind (Elektrodenpaar).
Das Prinzip der linearen Paul-Falle ist somit: Das gefangene Ion wird von einem Elektrodenpaar angezogen und gleichzeitig von dem Anderen abgestoßen. Nach dem Umpolen der Spannung, kehren sich die Verhältnisse um.

Die Frequenz der Wechselspannung wird dabei so gewählt, dass sich die Elektrodenpaare rechtzeitig umpolen, sodass es für das Ion kein Entkommen gibt.

Die lineare Paul-Falle kann mit dem Programm anschaulich simuliert werden:
Dazu platziert man 4 Ladungen, welche einen senkrechten Querschnitt durch die Stäbe der linearen Paul-Falle repräsentieren, auf dem Feld wie folgt:



Durch die Endelektroden der linearen Paul-Falle ist die Bewegung des Ions, im Wesentlichen auf diese dargestellte Simulationsebene reduziert.

Als nächstes platziert man ein Ion nahe der Mitte der Anordnung. Die Anfangsgeschwindigkeit sollte nahe Null gewählt werden (Paul Fallen werden stark gekühlt betrieben!).
Wird jetzt die Wechselspannung angelegt, kann man die Flugbahn und Geschwindigkeit des Ions am Bildschirm verfolgen. Im Idealfall kann das Ion die Paul-Falle nicht mehr verlassen.

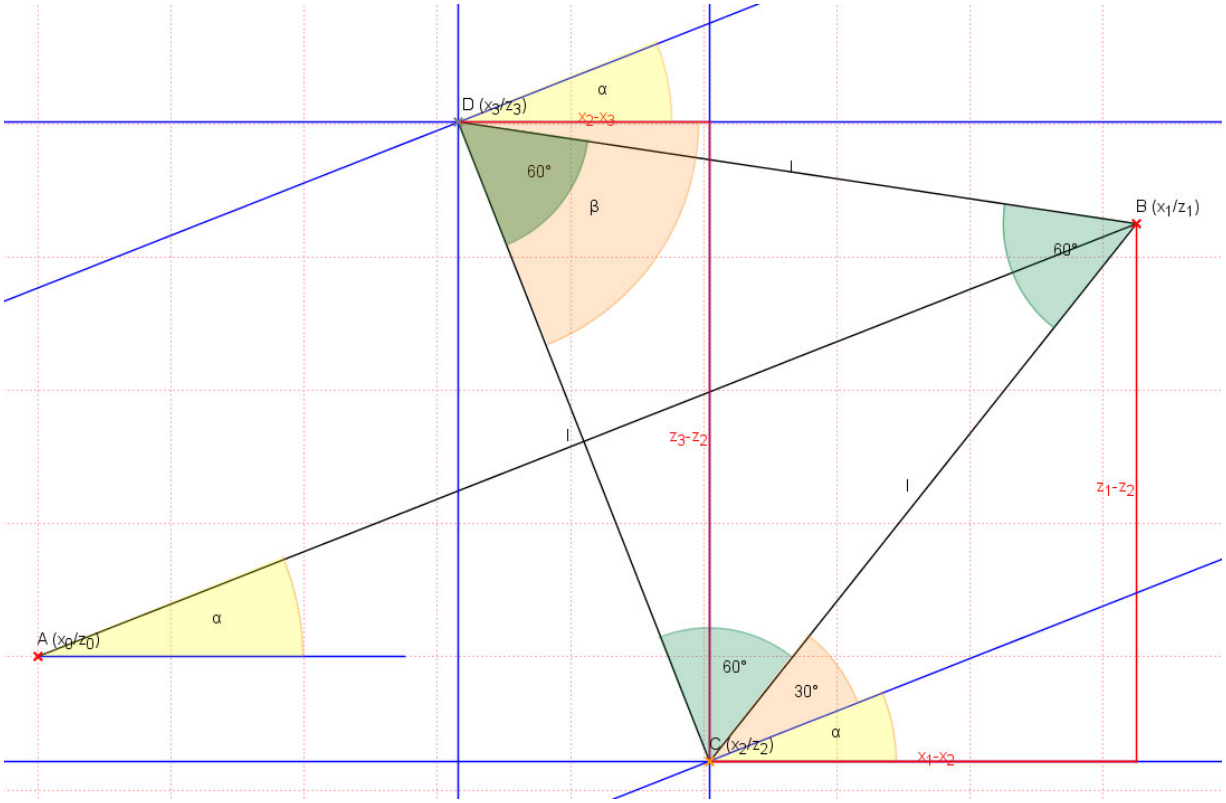
IV. Grundlagen der Programmierung

Um eine größtmögliche Flexibilität beim Programmieren zu erhalten habe ich die Programmiersprache C++ gewählt. Die Vorteile dieser maschinennahen Programmiersprache gegenüber einer modularen Hochsprache wie Java liegen auf der Hand: Da C++-Programme direkt auf den Grafik- und den Rechen-Prozessor zugreifen, sind sie zum einen viel schneller als Java-Programme, zum anderen können sie den vollen Funktionsumfang der Prozessoren benutzen. Der Hauptnachteil liegt darin, dass C++ zwar flexibler, aber dafür wesentlich aufwändiger zu handhaben ist.
Es gibt 2 moderne Schnittstellen zur Grafikkarte: OpenGL und DirectX. Fast alle auf dem Markt verfügbaren 3D-Spiele oder 3D-Programme benutzen eine der beiden Schnittstellen. Ich habe für mein Programm OpenGL gewählt, da diese Schnittstelle quelloffen und auf allen Betriebssystemen verfügbar ist.
Mit OpenGL erhält der Programmierer ein virtuelles 3-dimensionales Koordinatensystem bei dem nur 3 Arten von Objekten zur Verfügung stehen:

1. Punkte
2. Linien
3. Dreiecke

Alle anderen Objekte, wie z.B. Kreise und Kugeln, müssen aus diesen 3 Elementar-Objekten erstellt werden. Die Grafikkarte errechnet dann aus 3-dimensionalen Objekten ein 2D-Bild, welches der Benutzer auf dem Bildschirm zu sehen bekommt.
Im folgenden Programmausschnitt erkläre ich, wie man einen Pfeil mit OpenGL realisieren kann.

Die Skizze veranschaulicht, wie aufwändig es ist, einen einfachen parametrisierten Pfeil zu erzeugen.



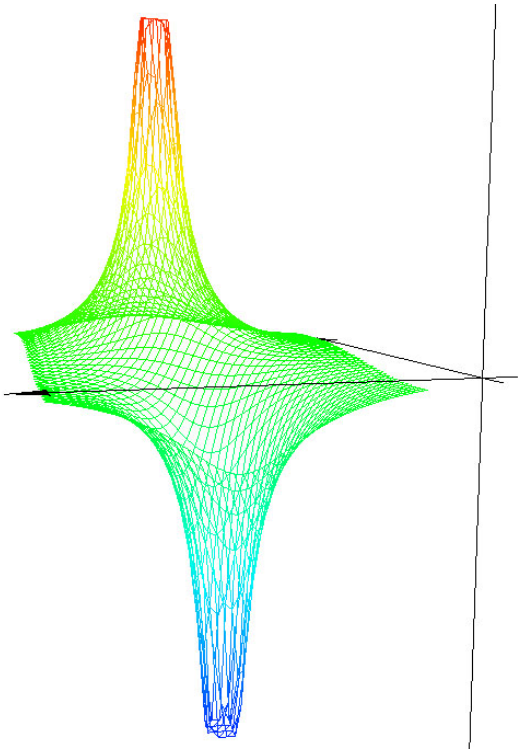
Hinweis: l := laengederspitze

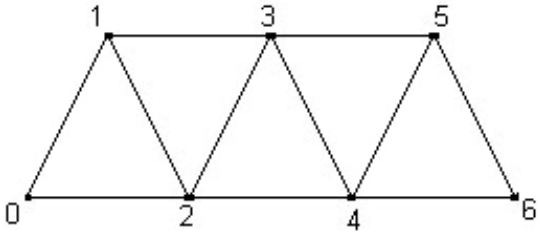
Nach diesem Muster werden mit der folgenden „Pfeilfunktion“ alle Pfeile im Programm erstellt:

```
void BensOpenGL::DrawArrow(GLfloat x0,GLfloat y0,GLfloat z0, GLfloat x1,GLfloat
y1,GLfloat z1, GLfloat laengederspitze)
{

        //Pfeil-Linie zeichnen
        glBegin(GL_LINES); // Mit diesem Befehl teile ich der Graphikkarte mit,
dass jetzt eine Linie gezeichnet werden soll.
                glVertex3f(x0,y0,z0); //Das ist der Anfangspunkt der Linie (A)
                glVertex3f(x1,y1,z1); //Das ist der Endpunkt der Linie (B)
        glEnd();

        //Pfeil-Spitze zeichnen
        glBegin(GL_TRIANGLES); //Jetzt wird ein Dreieck gezeichnet
                glVertex3f(x1,y1,z1); //Punkt B
                  GLfloat alpha = atan((z1-z0)/((x1-x0)==0?0.1:(x1-x0)));
                //Pfeil berechnen

//Der Punkt (x2,z2) wird wie folgt berechnet (analog zur oberen Skizze):
//x2 = x1 - cos(alpha + 30°)*l
//Wenn „x1-x0 < 0“ dann wird das Vorzeichen von x2 umgedreht (x2 = x2 * -1)

       GLfloat x2 = x1 - cos((M_PI/6)+alpha)*laengederspitze * (x1-x0 < 0?-1:1);
       GLfloat z2 = z1 - sin((M_PI/6)+alpha)*laengederspitze * (x1-x0 < 0?-1:1);
             glVertex3f(x2,y1,z2); //Punkt C
       GLfloat x3 = x2 - cos((M_PI/2)-alpha)*laengederspitze * (x1-x0 < 0?-1:1);
       GLfloat z3 = z2 + sin((M_PI/2)-alpha)*laengederspitze * (x1-x0 < 0?-1:1);
```

```
                glVertex3f(x3,y1,z3); //Punkt D
        glEnd();
}
```

Die 2D- und 3D-Graphen der Darstellung bestehen jeweils aus zigtausend aneinander gereihten Dreiecken (siehe die Programmfunktion „BensOpenGL::CalcFunction3D(void)“ im Quelltext).
Um diese Struktur (auch Wireframe genannt) für den interessierten Benutzer sichtbar zu machen habe ich eine weitere Funktion im Programm versteckt. Diese Funktion dient dazu, den Aufbau der Graphen besser zu verstehen und ist deshalb durch eine spezielle Tastenkombination geschützt.
Sie können sie aufrufen, indem Sie die Tasten Strg + Umschalt + R (bzw. Ctrl + Hoch + R) drücken. Es erscheint dann neben der Registrierkarte „Wechselspannung“ eine neue Karte mit dem Namen „Render“. Dort können Sie verschiedene Rendering-Einstellungen verändern.
In dem Programm werden über 100 Spalten gezeichnet. Jede Spalte besteht dabei aus über 100 aneinander gereihten Dreiecken, die nach dem folgenden Schema aufgebaut sind:





(Ausschnitt einer Einzelspalte aus über 100 Dreiecken[6])

Leider kann ich hier nicht alle Funktionen des Programms erklären, da dies über den Umfang des schriftlichen Teils der Facharbeit hinausgehen würde. Ich möchte deshalb nur noch ein paar Funktionen erwähnen, auf die ich bisher nicht näher eingegangen bin. Der Name der entsprechenden Programmfunktionen steht unter jedem Punkt rechts in kursiv. Unter diesem Namen finden Sie die Funktion im Quelltext.

- Farben-Funktion (stufenlose Berechnung von RGB-Farb-Werten je nach Höhe des Potenzials)

BensOpenGL::Topographie

- Algorithmen für die optimale Berechnung von Δt und dem Verstärkungsfaktor der Bremsstrahlung
- Diverse Genauigkeitsfunktionen, welche z.B. ähnliche Werte erkennen und vernachlässigen. Somit steht mehr Rechenleistung zur Verfügung, welche z.B. genutzt werden kann um die Auflösung zu erhöhen

  bool Genauigkeit, BensOpenGL::CalcIonen, BensOpenGL::CalcFeldlinien, BensOpenGL::CalcAequipotenziallinien
- Die Berechnung von Äquipotentiallinien und insbesondere dort der Korrektur-Algorithmus für Werte

  BensOpenGL::CalcAequipotenziallinien
- OpenGL-Funktionen (Initialisierung der Zeichenfläche, richtige Werte für Perspektive, Kamera-Position, Maus-Interaktion und die Berechnung der Maus-Position auf dem Graphen)

  BensOpenGL::DrawGLScene, BensOpenGL::InitGL, BensOpenGL::Get3DMousePos
- Zeichnen des Koordinatensystems

  Init3DGraph, Init2DGraph, BensOpenGL::DrawCoordinateBox, BensOpenGL::DrawCoordinateSystem
- Speichern- und Laden-Funktion

  Load, Save
- Editier-Funktionen (Neue Ladungen, Verschieben von Ladungen, Löschen, etc.)

  EditMode2
- Diverse Resize-Funktionen (sorgen dafür, dass sich beim Verändern der Programm-Fenstergröße automatisch die Elemente des Programms angepasst werden)

  TabControlResize, Resize3DGraph, Resize2DGraph
- Benutzer-Interaktion-Funktionen

  RefreshLadungstabelle, LabelLegendeClick, Panel3DEngineMouseDown, Panel3DEngineMouseMove, Panel2DEngineMouseDown, Panel2DEngineMouseMove, Panel2DEngineMouseUp, StringGridDBSetEditText, WMMouseWheel, FormKeyDown, u.v.m.
- Funktion zum Zeichnen der Zeit-Geschwindigkeits-Diagramme

  PainttvGraph
- Werte-Datenbanken und Variabeln (für Zeitlupe, Potenziale, Eigenschaften)
- Drucken

  Tabdrucken1Click
- Screenshot

  GraphalsBilddateiexportieren1Click, GraphalsBilddateiexportieren2Click
- Tab-Verwaltungsfunktionen
- u.v.m.

Insgesamt besteht das Programm aus 5311 Codezeilen, das entspricht etwa 80 DIN-A4-Seiten (ohne Rand, Schriftgröße 10).

Das Programm wurde mehrmals geschwindigkeitsoptimiert.
Es verwendet die neue Technologie „Multi-Threading“, die es erlaubt mehrere Rechenprozesse parallel laufen zu lassen. Mein Programm benutzt insgesamt 3 Threads. Diese Threads sind selbstständige Programme innerhalb des (Haupt-)Programms, welche unabhängig voneinander ablaufen.
Die Benutzeroberfläche verwendet einen Thread sowie jeder Graph noch je einen Thread. Ein Vorteil ist zum Beispiel, dass der Benutzer das Programm bedienen kann, ohne dass das Programm stockt. Bei normalen Programmen ohne Multi-Threading kann nicht gleichzeitig eine Berechnung durchgeführt und das Programm vom Benutzer bedient werden.

Außerdem wurden große Teile des Programmcodes mehrmals überarbeitet, um bessere Geschwindigkeiten zu erzielen. Gegenüber frühen Versionen des Programms läuft die Endversion bei gleicher Auflösung 10x schneller (siehe Anhang: changelog.txt für mehr Informationen).

Die Fehleranalyse ist ein wichtiger Bestandteil bei einer Programmierung. Intern habe ich deshalb für die Fehleranalyse selbst eine Debug-Funktion programmiert, die mir geholfen hat Fehler zu lokalisieren und zu beseitigen. Diese Debug-Funktion ist im fertigen Programm nicht mehr enthalten.
Oft treten Fehler relativ spät in Funktionen auf, die bereits sehr lange Bestandteil des Programms sind. Im Anhang finden Sie das Dokument „fehleranalyse.doc“, das ein Beispiel einer konkreten Fehleranalyse beschreibt.

Generell ist es mir ein Anliegen das Programm flexibel im Code zu halten, sodass der Code später leicht nachvollziehbar bleibt und Code-Erweiterungen keine Probleme bereiten. Portierungen auf andere Systeme sind so unkompliziert möglich. Als Beispiel habe ich das Programm auf das Apple iPhone portiert. (Programmiersprache C++, Objective C++). Einige Screenshots befinden sich im Anhang auf S.32.

Bevor man mit der Programmierung eines Programms anfängt, ist es wichtig sich eine Vorstellung darüber zu machen, wie das Programm aussehen soll und welche Funktionen es haben soll. Ich habe mir deshalb bereits ein paar Tage bevor ich mit dem Programmieren angefangen habe mehrere Skizzen für die Oberfläche und Funktionen des Programms gemacht (siehe Anhang S.33).
Obwohl viele Funktionen dazugekommen sind, die ursprünglich nicht vorgesehen waren, weist die aktuelle Version noch immer große Ähnlichkeit mit den damaligen Skizzen auf.

V. Schluss

Zur Konstruktion von kleinen Geräten wie Prozessoren, bis hin zu großen wie Flugzeugen sind Simulationsprogramme extrem wichtig. Ohne diese Programme würde die Technik heute nicht ihren fortgeschrittenen Stand erreicht haben. Mit ihrer Hilfe ist es möglich, Phänomene, welche entweder bis dahin unentdeckt waren oder aber an die bei der Konstruktion nicht gedacht wurde, vorherzusagen und deren Ursache zu bestimmen. Computerprogramme sind im Vergleich zu Experimenten Kosten sparend und können – einmal programmiert – schnell verschiedenste Szenarien simulieren.
Mit Hilfe eines industriellen Simulationsprogramms („Comsol Multiphysics“), welches es, wie auch mein Programm, ermöglicht die Flugbahnen von Ionen in komplexen elektrischen Feldern zu berechnen, wurde zum Beispiel ein neuer Detektor entwickelt, welcher kleinste Mengen an Sprengstoff und Drogen erkennen kann[7].
Durch Experimentieren mit meinem Programm habe ich z.B. die Apsidendrehung als ein physikalisches Phänomen kennen gelernt: In einem Feld mit einer positiven Ladung kreist ein Elektron um dieses gleichförmig in einer stabilen Ellipsenbahn. Aber bereits durch eine geringe Störung des Feldes (z.B. durch eine kleine zweite Ladung in der Nähe) ändert sich die Orientierung der Ellipsenbahn bei jeder Umdrehung (auch Präzession genannt), die Ellipse fängt an, sich in ihrer Ebene zu drehen (analog zur Ellipsenbahn der Erde um die Sonne, die durch andere Planeten gestört wird, genannt planetare Präzession). Dieses Mehrkörper-Problem kann nicht in mathematischen Formeln geschlossen gelöst werden, sondern nur näherungsweise[8].
Erst durch Computer lassen sich diese Art von Phänomen in Simulationsprogrammen anschaulich und zugleich präzise darstellen (computational physics).

VI. Anhang

Anhang: Inhalt der CD-ROM

Speedion2010.exe – *Das fertige Programm*
Facharbeit.pdf – *Der schriftliche Teil der Facharbeit*
Bilder und Icons – *Dieser Ordner enthält alle Icons, die ich für die Facharbeit entworfen habe*
- gestuft-verlaufen.jpg
- icon_aelinien.jpg
- icon_feldlinien.jpg
- icon_graph.jpg
- ladung-c-e.jpg
- pfeile_all.jpg
- pfeile_one.jpg
- pos-negativ.jpg
- splash_speedion2010.jpg

Facharbeitsbesprechungen
- PhysikFA1.pdf
- PhysikFA2.pdf

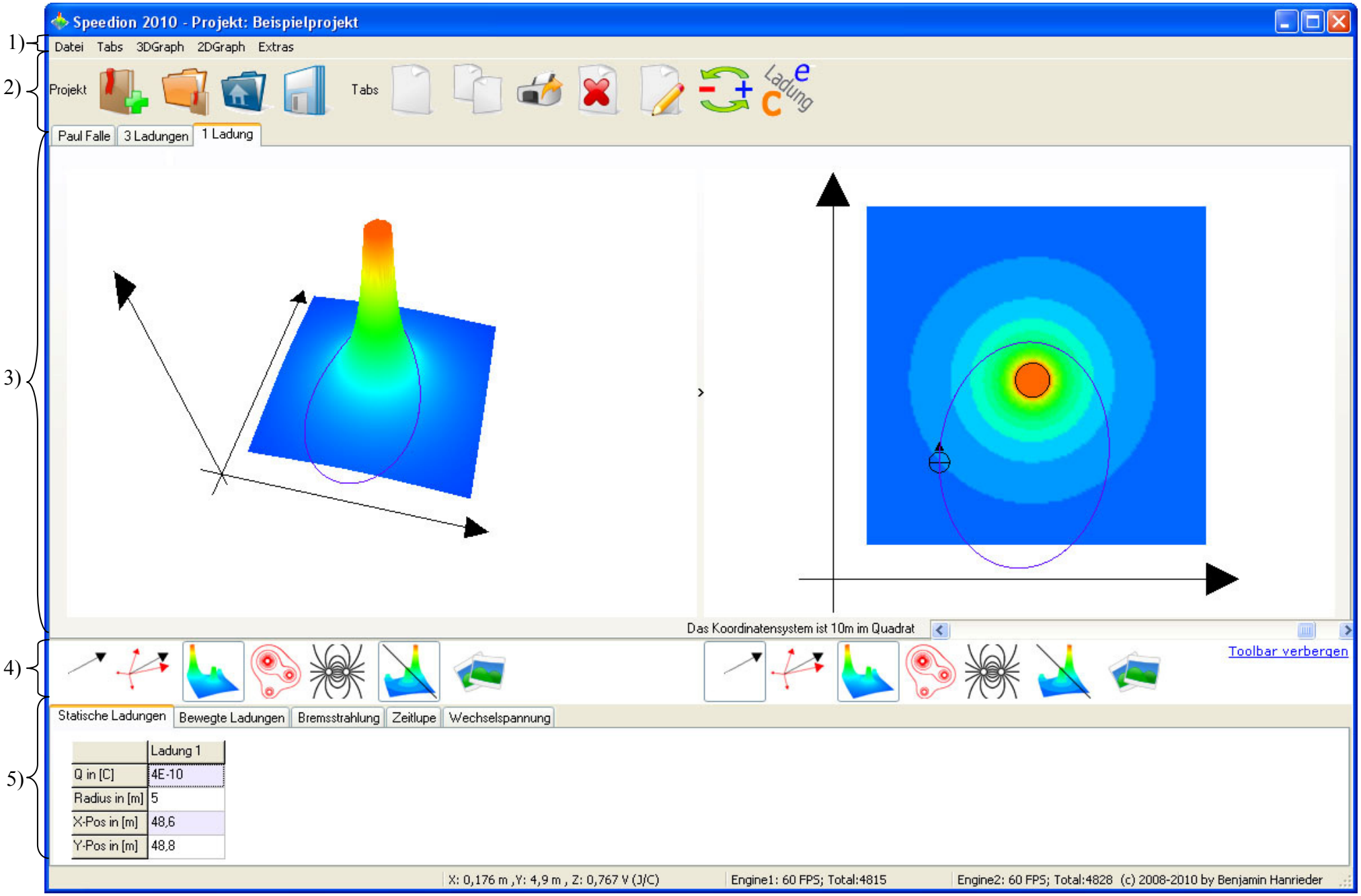
Fehlersuche Praxis
- Beispiel-Fehleranalyse.pdf
- nachher.exe
- vorher.exe

Literatur und Bildquellen – *Dieser Ordner enthält alle Quellen als Screenshots oder gescannte Dokumente*
Quelltext
- BensGraphEngine.cpp.pdf
- BensGraphVariables.cpp.pdf
- BensOpenGl.cpp.pdf
- BensOpenGl.h.pdf
- Unit1.cpp.pdf

Sonstiges
- Entwicklungsumgebung.jpg – *Screenshot meiner Arbeitsumgebung*
- fehler.jpg
- pfeil.svg
- skizze_november_2008.pdf

Verschiedene ältere Versionen
- 112 - 31.10.08
- 118 - 01.11.08
- 139 - 04.11.08
- 201 - 07.11.08
- 205 - 09.11.08 – Schule: 1. Vorführung
- 213 - 01.02.09
- 219 - 28.02.09
- 229 - 23.05.09
- 235 - 18.06.09
- 240 - 26.08.09

Anhang: Screenshot des Programms „Speedion 2010“



Anhang: Ausschnitt der Datei „changelog.txt“

(c) 2008-2010 by Benjamin Hanrieder - All rights reserved

CHANGELOG

Version 1:

101-109: Grundkenntnisse von OpenGL aneignen
110: Engine wird stark verbessert: Sie ist jetzt flexibel, d.h. man kann jetzt 2 Engines in einem Programm haben ohne den Code zu verdoppeln
- Theoretisch können jetzt beliebig viele Engines erzeugt werden, ohne nennenswerten coding-Aufwand
111: Multi-Threading
- Die Engines fungieren jetzt unabhängig voneinander; Berechnungen werden GLEICHZEITIG statt nacheinander ausgeführt
- Außerdem fungiert das Programm ebenfalls unabhängig von den Engines.
- Vorteile: schneller, Bedienung des Programms ist unabhängig von den Engines (=> Programm kann bedient werden obwohl gerade ein Bild berechnet wird)
114: Der Engine-Code wird vom GUI-Code getrennt (für eine bessere Übersicht)
- neue Datei: BensOpenGl.cpp
118: Erste erfolgreiche Darstellung eines Potenzialverlaufs
119: Steuerung der Graphen möglich
121: Graph besteht nicht mehr nur aus Linien (einstellbar in "Button18")
- Erste Farb-Versuche
122: Schöner Farbverlauf (jede Äquipotenzialfläche hat ihre eigene Farbe => je steiler der Graph, desto schneller wechselt die Farbe)
123: Auflösung ist anpassbar
124: Mehrere Potenziale in einem Graphen
127: Versuch Äquipotenzialflächen einzuzeichnen
128: Code-Clean-Up
129: Potenzial-Funktion wird verbessert
130: Erste Maus-Interaktion
132: Koordinaten-Box
133: Code-Clean-Up
137: Animation des Graphen (nett!)
138: Farb-Verlauf kann differenziert werden
- Nochmalige Aufspaltung der Engine in die CoreEngine und GraphEngine
- neue Datei: BensGraphEngine.cpp
139: 2D-Darstellung hinzugefügt
- "Überschwemmung" hinzugefügt

- Unabhängige und Synchronisierte Interaktion zwischen den Engines
- Koordinaten werden angezeigt
140: Hoch und Tiefpunkte können berechnet werden ("Button24")
141: Code-Clean-Up
- Farbverlauf kann während der Programmausführung verändert werden
143: "Überschwemmung" semi-transparent
144: Feldliniendarstellung wird implementiert
145: Editiermodus wird fertig gestellt (Ladungen können während der Laufzeit hinzugefügt werden und deren Radius kann verändert werden)
148: Kleinere Verbesserungen

Version 2:

201: neues Design
203: Anzeige der Ladungen in einer übersichtlichen Tabelle
204: Maus-Interaktion wird verbessert:
- Der Graph lässt sich durch die Maus drehen und mit dem Mausrad vergrößern oder verkleinern
205: Übernahme der Verbesserungen der iPhone-Version (vgl. iPhone-Quelltext AppController.m)
- 3D Funktion stark verbessert: berechnet jetzt bei gleich bleibender Auflösung halb so viele Punkte in einem schnelleren Modus.
- Nochmalige Verbesserung: Die Funktion wird jetzt nicht mehr für jedes Bild neu berechnet, sondern in einer Matrix gespeichert
- Folgerung: Mehr Frames per Second (FPS). Dadurch lässt sich die Funktion besser drehen und Prozessorauslastung und Grafikauslastung sinken stark
206: Kommentierung des Codes
- Reinigung von überflüssigem Code
- "Clean-Up"
208: Hinzufügen der parametrisierten Feldlinien-Vektor-Pfeilspitze (gar nicht so einfach!)
209: Geschwindigkeitsoptimierung und Fehlersuche: Warum nicht konstant bei 60 FPS, was ist so rechenaufwändig?
- Debug-Funktion der Engines wird eingeführt
- Ergebnis: Funktion Get3DMousePos(glReadPixels und glGetIntegerv(GL_VIEWPORT, viewport)) braucht überdurchschnittlich lange (16ms!)
- Nach dem Abschalten der Funktion sinkt die Prozessorauslastung von mindestens 50% auf 0-10%, Außerdem ist die Framerate beider Engines konstant auf 60FPS
- Ein Bild zu berechnen braucht jetzt unter 1ms!

- Ausbesserung von glGetIntegerv(GL_VIEWPORT, viewport): wird nur noch aufgerufen, wenn die Größe des Fensters verändert wird.
- Funktion Get3DMousePos wird nur noch aufgerufen, falls die Maus über der Engine ist. (=> maximale Prozessorauslastung, wenn die Maus gerade über einer Engine ist: 70%)
- Insgesamt mehr FPS, da die Funktion von maximal 1 (statt 2) Engines aufgerufen wird

210: Feldlinienfunktion wird repariert (Betrag der Gesamtfeldstärke wurde falsch ausgerechnet)

223: Bug bei der Feldstärkeberechnung verbessert: Feldstärke wurde bei den Grenzwerten alpha = +/- 90 grad falsch berechnet (Vorzeichen war falsch)

...

254: Endversion

Anhang: Datei „Beispiel-Fehleranalyse.pdf"

Problembeschreibung:

1. Gleitkommadivision durch 0, wenn man eine Ladung verschiebt. Diese Fehlermeldung bekommt man, wenn man das Programm über den Debugger laufen lässt (siehe vorher.exe)
2. Das Programm stürzt ab

So reproduziert man das Problem:

1. Klicken Sie auf den Edit-Button in der Menüleiste
2. Klicken Sie auf die Ladung in dem 2D-Graphen und verschieben Sie diese so lange bis die Fehlermeldung auftritt

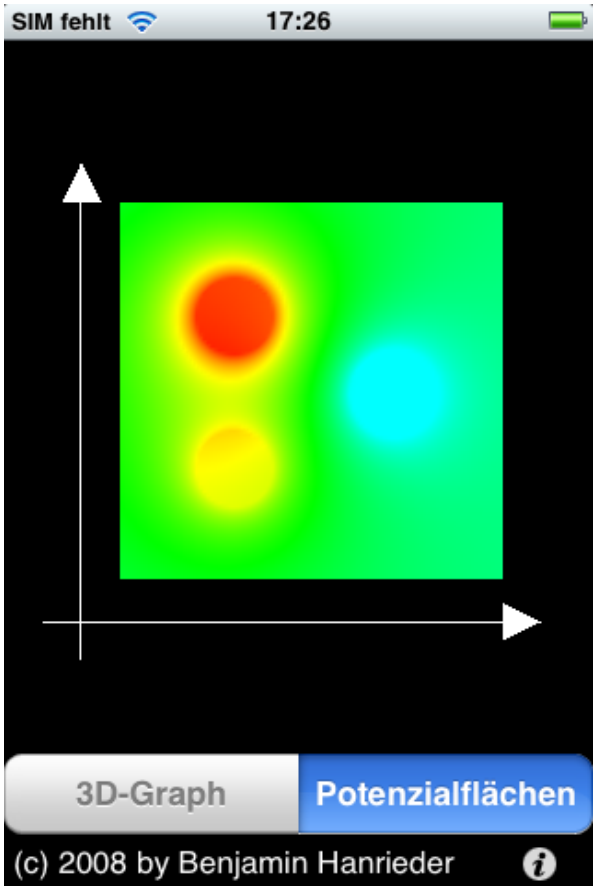
Lösung des Problems:

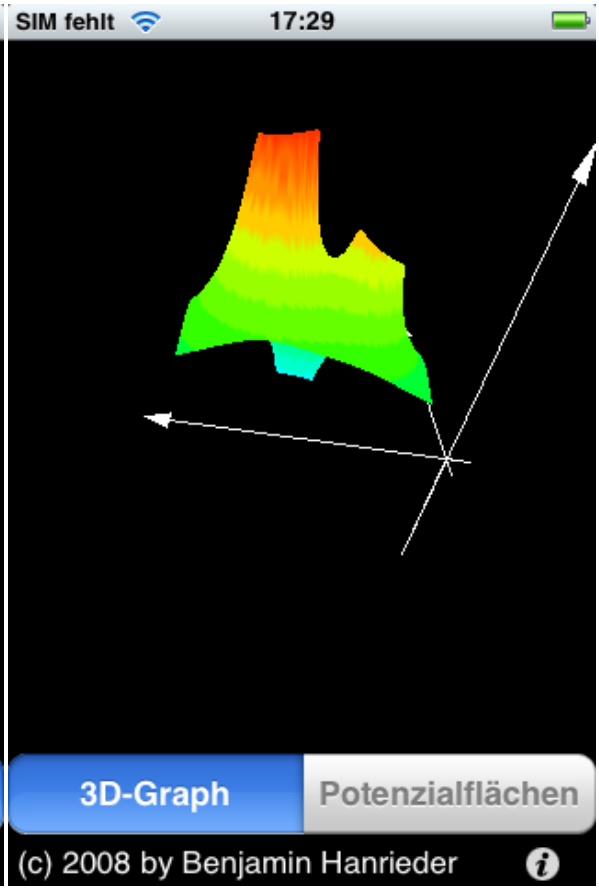
1. Das Problem deutet darauf hin, dass irgendwo eine Division durch 0 ausgeführt wurde, welche automatisch diesen Fehler auslöst.
2. Dadurch, dass dieser Fehler mehrmals hintereinander erzeugt wird, schließe ich darauf, dass der Fehler bei einer Rechnung erzeugt wurde, welche in einer Schleife ausgeführt wird.
3. Ich habe keine Ahnung, welche Rechnung dafür verantwortlich sein könnte, eine Reproduktion des Fehlers lässt mich auch nicht auf einen offensichtlichen Code-Fehler schließen.
4. Deshalb deaktiviere ich nacheinander einzelne Teile des Programms, so lange bis der Fehler nicht mehr auftritt.
5. Als erstes deaktiviere ich auf „gut Glück" die Funktion, die für die Farbenberechnung verantwortlich ist. Eigentlich bin ich davon überzeugt, dass die Funktion fehlerfrei ist und halte diesen Versuch für aussichtslos.
6. Dennoch tritt der Fehler jetzt nicht mehr auf. Normalerweise braucht man für diese Art der Fehlersuche relativ viel Zeit, besonders wenn das Programm groß ist. Ich traue daher meinem Glück nicht sofort und überlege, bevor ich mir die Funktion einmal genauer anschaue eine mögliche Ursache des Fehlers: Beim Verschieben einer Ladung werden sehr viele Rechnungen durchgeführt. In Ausnahmefällen werden so 2 Rechnungen des gleichen Typs, die auf dieselben Variabeln zugreifen, gleichzeitig durchgeführt und es kommt zu einem Fehler. Der Fehler tritt jetzt nicht mehr auf, weil die Farbberechnungsfunktion viel Zeit braucht, sodass es unwahrscheinlicher ist, wenn sie deaktiviert ist, dass 2 Rechnungen desselben Typs gleichzeitig auftreten.
7. Um meine Vermutung zu bestätigen, erhöhe ich die Auflösung der Graphen, was wieder mehr Rechnungen verursacht.
8. Negativ. Ich freue mich und schaue mir die Farbfunktion genauer an.
9. Es kommen tatsächlich einige Division darin vor.
10. Ich glaube den Übeltäter gefunden zu haben: Die Variable trennlinie könnte 0 sein. Das würde den Fehler auslösen, deshalb deaktiviere ich die Variable. Mit Erfolg. Der Fehler tritt nicht mehr auf.
11. Aber warum wird trennlinie 0? Das ist der Fall wenn hochpunkt-tiefpunkt=0, also eigentlich nie. Aber wer berechnet hochpunkt bzw. tiefpunkt?
12. Da stoße ich auf die Funktion CalcTiefHochpunkt, welche die Punkte berechnet, und merke dass dort für eine sehr kurze Zeit hochpunkt=tiefpunkt=0 gesetzt wird. Wenn jetzt gleichzeitig in der kurzen Zeit in der hochpunkt=tiefpunkt die Farben-Funktion aufgerufen wird kommt es zu einem Fehler.
13. Ich behebe den Fehler, indem ich hochpunkt und tiefpunkt nie 0 setze und baue außerdem eine Abfrage ein, die sichergeht, dass sie nie 0 sind.

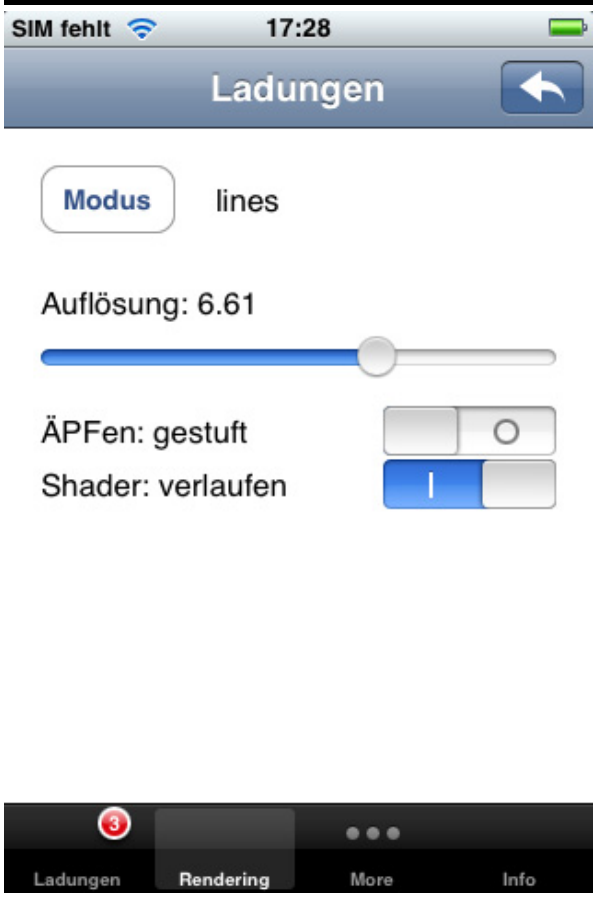
Differenzen der beiden Programme (vorher.exe und nacher.exe)
void CalcTiefHochpunkt(void)

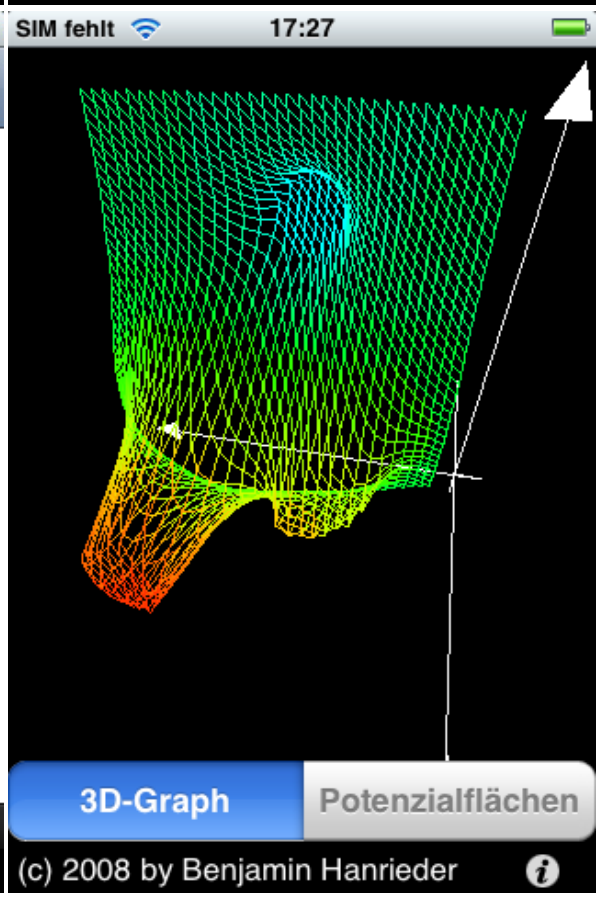
```
{
    tiefpunkt = 0;  //Niedrigster punkt
    hochpunkt = 0;  // Höchsterpunkt;
    for (int i = 0; i<ladungen_anzahl[curt];i++)
    {
        GLfloat extrema =
Potenzial(ladungen_Pos[curt][i][0],ladungen_Pos[curt][i][1]);
        if (extrema < tiefpunkt) tiefpunkt = extrema-1;
        if (extrema > hochpunkt) hochpunkt = extrema+1;
    }
    //  Form1->Label1->Caption = String(tiefpunkt);
    //  Form1->Label3->Caption = String(hochpunkt);
}



   void CalcTiefHochpunkt(void)
   {
       GLfloat temptiefpunkt = 0;  //Niedrigster punkt
       GLfloat temphochpunkt = 0;  // Höchsterpunkt;
       for (int i = 0; i<ladungen_anzahl[curt];i++)
       {
           GLfloat extrema =
   Potenzial(ladungen_Pos[curt][i][0],ladungen_Pos[curt][i][1]);
           if (extrema < temptiefpunkt) temptiefpunkt = extrema-1;
           if (extrema > temphochpunkt) temphochpunkt = extrema+1;
       }
       //  Form1->Label1->Caption = String(tiefpunkt);
       //  Form1->Label3->Caption = String(hochpunkt);
       if (temptiefpunkt != temphochpunkt) {tiefpunkt = temptiefpunkt;
   hochpunkt = temphochpunkt; }
   }

   Versionen: 0.2.30 bzw 0.2.31
```

Anhang: Screenshots „iPhone- Version“

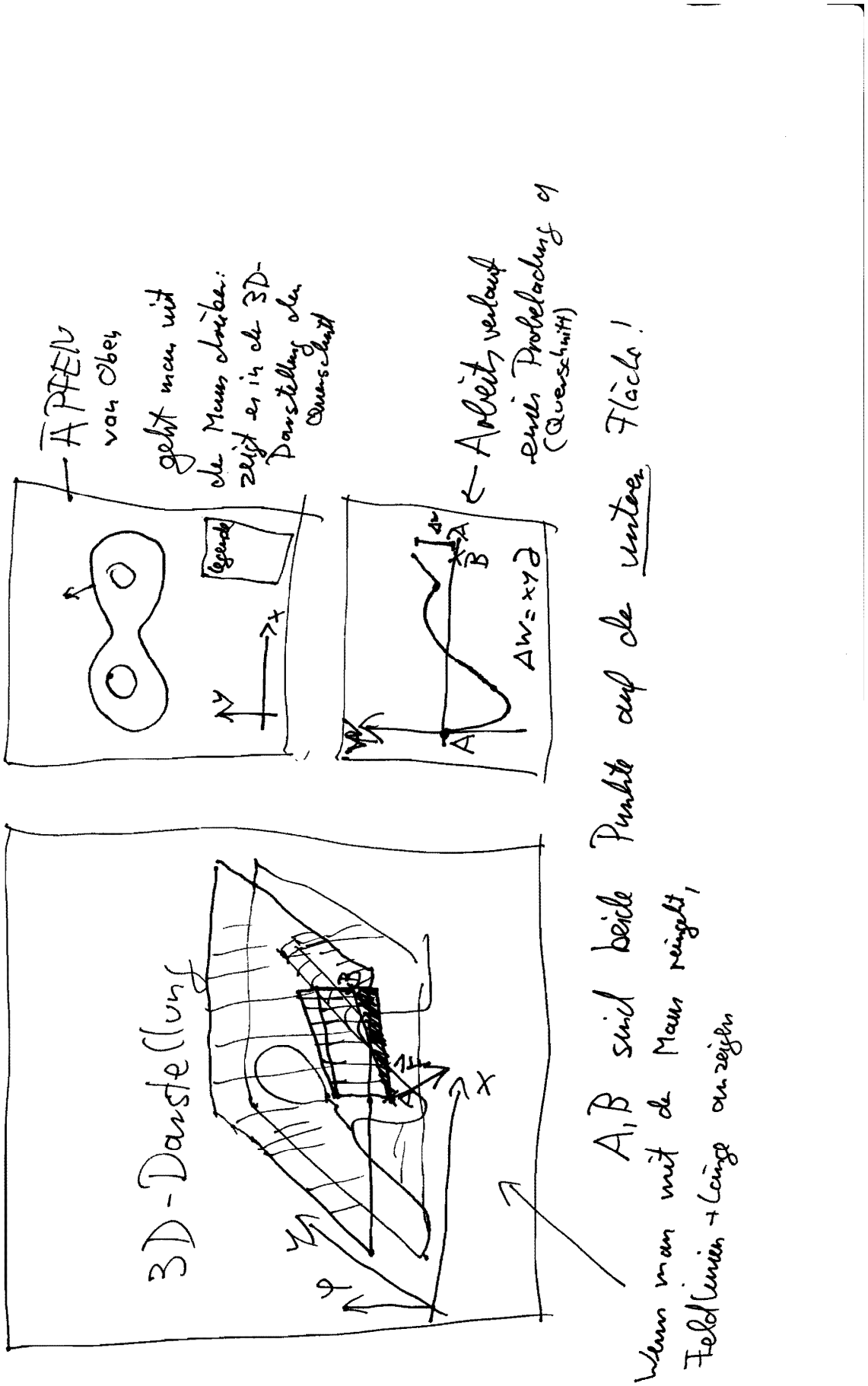


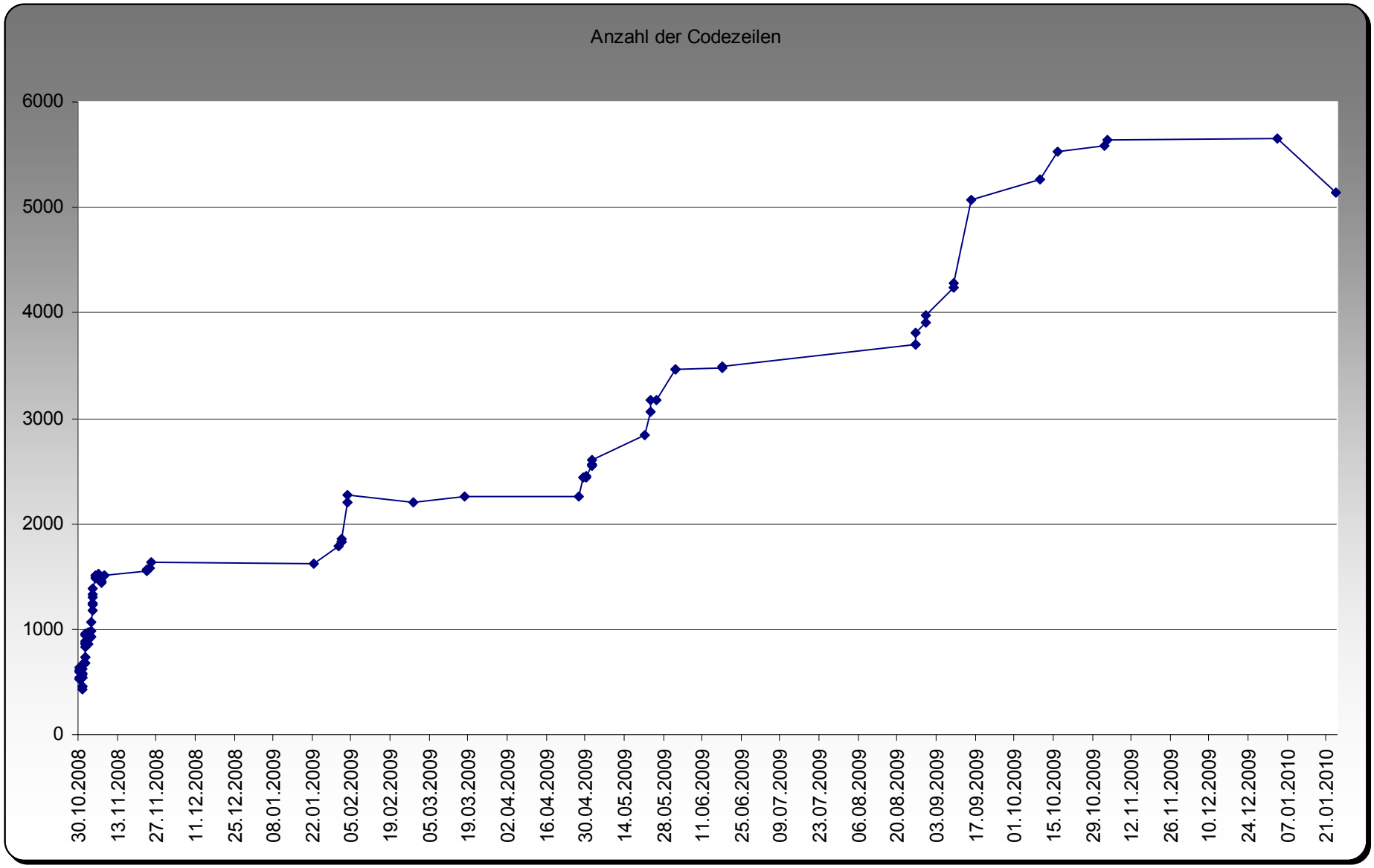






Anhang: Datei „skizze oktober 2008.pdf“




Anzahl der Codezeilen
6000
5000
4000
3000
2000
1000
0
30.10.2008
13.11.2008
27.11.2008
11.12.2008
25.12.2008
08.01.2009
22.01.2009
05.02.2009
19.02.2009
05.03.2009
19.03.2009
02.04.2009
16.04.2009
30.04.2009
14.05.2009
28.05.2009
11.06.2009
25.06.2009
09.07.2009
23.07.2009
06.08.2009
20.08.2009
03.09.2009
17.09.2009
01.10.2009
15.10.2009
29.10.2009
12.11.2009
26.11.2009
10.12.2009
24.12.2009
07.01.2010
21.01.2010

VII. Quellen und Literaturverzeichnis

[1] Bildquelle: http://www.klickdichschlau.at/contentfiles/edv/hardware/Tastatur_de.png abgespeichert am 20.01.2010

[2] Hammer/Hammer, Physikalische Formeln und Tabellen, J. Lindauer Verlag München, 2002[8], S.41, 42

[3] Jackson, Klassische Elektrodynamik, 2.Auflage, Walter de Gruyter Verlag, 1982[2], S. 789 – 790, und
Gebhard von Oppen / Frank Melchert, Physik für Ingenieure, Pearson Studium Verlag, 2005, S. 186 - 187

[4] Internetseite http://www.physik.uni-kassel.de/exp1/de/f-praktikum/f07_paulfalle.pdf abgespeichert am 20.01.2010

[5] A. Ostendorf, B. Roth, C. Lämmerzahl, S. Schiller, Internetseite http://www.exphy.uni-duesseldorf.de/ResearchInst/ultracold_complex_molecules.htm abgespeichert am 20.01.2010

[6] Bildquelle: http://tfc.duke.free.fr/old/models/images/05.gif abgespeichert am 20.01.2010

[7] Physik Journal – Sonderheft: „Best of", Wiley-VCH Verlag, Ausgabe September 2009, S. 31 – 33

[8] Internetseiten: http://www.praxelius.de/astro/stoerung.htm und http://de.wikipedia.org/wiki/Apsidendrehung abgespeichert am 20.01.2010

Für die Programmierung verwendete Webseiten:

- http://www.3dsource.de/faq/selection.htm
- http://www.3dkingdoms.com/selection.html
- http://wiki.delphigl.com/index.php/Tutorial
- http://nehe.gamedev.net/

Für das Programm verwendete Icons:

- kNeu alpha 0.2 – Download: http://linux.softpedia.com/get/Desktop-Environment/Icons/kNeu-5497.shtml

Ich erkläre hiermit, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Planegg, den 27. Januar 2010

Benjamin Hanrieder