

Virtuelle Moleküle sichtbar machen

Ein Angebot des Sonderforschungsbereichs 716 „Simulation mit großen Teilchenzahlen“ der Universität Stuttgart gefördert durch die Deutsche Forschungsgemeinschaft (DFG), Durchgeführt von VISUS (Visualisierungsinstitut der Universität Stuttgart), 25.04.2013

Ein Bild sagt mehr als tausend Worte ... auch an der Uni.

Bunte Bilder sind in der Wissenschaft und Forschung sehr wichtig. Die Ergebnisse aus Experimenten und Simulationen sind oft unübersichtliche Zahlenwüsten. Darum machen wir vom Visualisierungsinstitut aus diesen Zahlen verständliche Bilder und Filme.

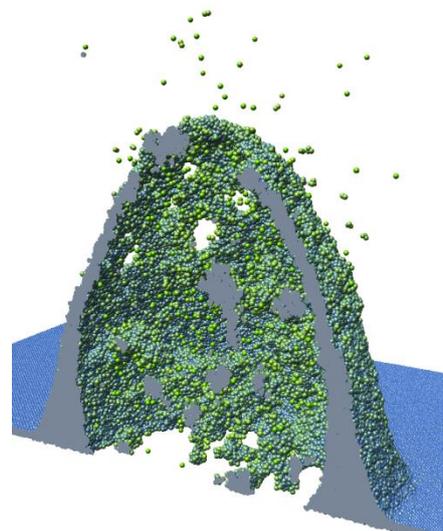
Und das könnt ihr bei uns ausprobieren. In unserem Workshop könnt ihr am Computer die Atome und Moleküle tanzen lassen; sogar in 3D wie im 3D-Kino. Wir zeigen euch wie das geht und beantworten natürlich alle Fragen rund um Informatik. Natürlich dürft ihr euer Programm auch mit nach Hause nehmen und dort weiter machen.

Visualisierung ist gar nicht so schwer und es macht Spaß, weil man direkt etwas sehen kann. Probiert es einfach aus!

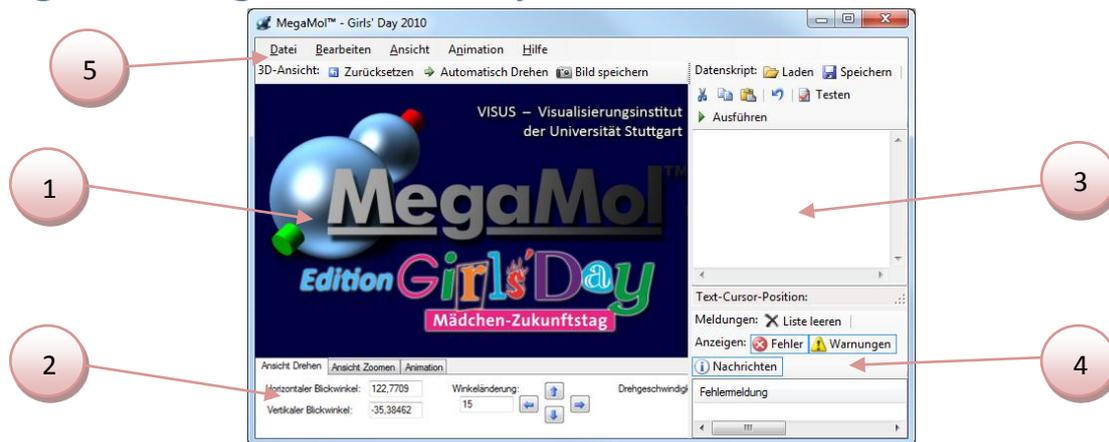
Einführung

Was ist „Visualisierung“ eigentlich? Übersetzt, frei nach dem „Oxford English Dictionary“ bedeutet „Visualisieren“ sich ein (mentales) Bild von etwas nicht Sichtbarem oder Abstraktem zu erzeugen. Im Zusammenhang mit Computer-Graphik bedeutet die Computer-gestützte Visualisierung die bildhafte Darstellung vieler und großer Datenmengen, damit die Daten leichter erfasst und verstanden werden können. Ein weithin bekanntes Beispiel sind Karten (z.B. Wetterkarten) auf denen eine Situation meist besser erfassbar ist, als in einer reinen Textbeschreibung.

An der Universität Stuttgart arbeiten viele unterschiedliche Forscher daran das Verhalten von Atomen bei verschiedenen Prozessen zu verstehen. Schon bei etwas eigentlich ganz einfachen, wie das Verflüssigen eines Gases (z.B. Tröpfchen-bildung bei Wasserdampf), ist das Verhalten einzelner Atome noch nicht wirklich vollständig verstanden. Bei komplizierteren Prozessen, wie beispielsweise das Bohren oder Gravieren von Metallen mit Laserstrahlen, sind natürlich noch viel mehr Fragen offen. Experimente sind meist nicht, oder nur mit extremen Aufwand möglich. Daher benutzt man heutzutage Computersimulationen. Dabei werden tausende oder sogar millionen von Atomen simuliert. So entstehen große Daten, die ohne Visualisierung nur sehr schwer zu überblicken sind.



Programm „MegaMol™ Girl's Day Edition“



MegaMol™ ist ein Programm, mit dem Simulationsdaten visualisiert, das heißt dreidimensional angezeigt, werden können. Es wird an der Uni. Stuttgart entwickelt und von vielen Forschern aus den Bereichen Biochemie, Physik und der Materialforschung eingesetzt. Extra für den Girls' Day haben wir eine „Sonderedition“ vorbereitet, mit der wir die Grundlagen der Molekülvisualisierung ausprobieren können.

Das Hauptfenster ① ist die 3D-Ansicht, in der später unsere Daten angezeigt werden. Mit der Maus kann diese Ansicht dann verändert werden (Linke Maustaste zum Drehen, Mittlere Maustaste zum Vergrößern). Die Kontrollelemente an der Unterseite des Fensters ② erlauben uns diese 3D-Ansicht zu verändern oder die Steuerung der Zeit bei bewegten Daten. Auf der rechten Seite ist ein Eingabefenster ③, in dem wir unsere Visualisierung programmieren. Informationen und Fehlermeldungen hierzu werden direkt darunter angezeigt ④. Alle Funktionen des Programms sind über die Schaltflächen im Fenster oder über das Hauptmenü ⑤ erreichbar. Die Schaltflächen im Fenster und die Einträge im Menü, die dieselbe Funktion haben sind auch mit dem gleichen Icon (kleinen Bildchen) gekennzeichnet.

Im Laufe dieser Aufgaben werden wir hier Programme schreiben, mit denen Kugeln im dreidimensionalen Raum gezeichnet werden. Damit werden wir Moleküldatensätze visualisieren, wie es bei der täglichen Arbeit an der Universität oft gemacht wird.

Die wichtigsten Funktionen sind bequem über Schaltflächen über dem Programmierbereich ③ erreichbar. Mit  **Laden** und  **Speichern** kann der Inhalt des Eingabefensters des Programmierbereichs aus einer Datei geladen oder in einer Datei gespeichert werden. Mit  **Testen** wird das Datenskript im Programmierbereich vom Computer in die Maschinensprache übersetzt. Wenn der Computer den Text nicht übersetzen kann, z.B. weil Tippfehler darin enthalten sind, dann wird eine entsprechende Fehlermeldung unter dem Programmierbereich ④ angezeigt. Die vielleicht wichtigste Funktion ist  **Ausführen**. Damit wird das Datenskript im Programmierbereich nicht nur in die Maschinensprache übersetzt (die gleiche Aktion wie bei „Testen“), sondern das Skript wird auch anschließen ausgeführt. Damit erzeugt der Computer die Daten für die Darstellungen in der 3D-Ansicht ①. Wie genau das alles funktioniert, werden wir in den folgenden Aufgaben sehen.

Erste Darstellungen

Eine Kugel wird durch, wie ein Kreis, durch ihren Mittelpunkt und den Radius festgelegt. Zusätzlich können wir jeder Kugel auch noch eine Farbe geben. Im kartesischen, zweidimensionalen Koordinatensystem wird ein Punkt durch seine Koordinaten auf der X- und Y-Achse bestimmt, im dreidimensionalen kommt eine weitere Koordinatenachse dazu, die Z-Achse.

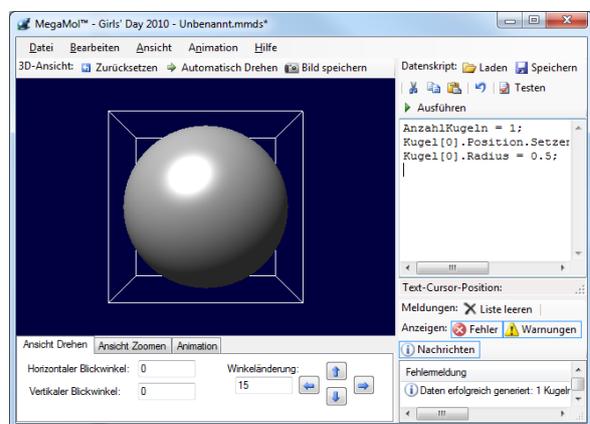
1. Erzeugen einer Kugel in 3D-Koordinaten

Als erste Aufgabe werden wir eine Kugel platzieren. Dies erreichen wir mit dem folgenden Datenskript:

```
AnzahlKugeln = 1;  
Kugel[0].Position.Setzen(0.0, 0.0, 0.0);  
Kugel[0].Radius = 0.5;
```

Zuerst geben wir die Anzahl der Kugel an, die wir erzeugen wollen, in diesem Fall 1. Nun müssen wir die Eigenschaften dieser Kugel festlegen, nämlich den Mittelpunkt (d.h. die Position) und den Radius. Mit `Kugel[0]` greifen wir auf die erste Kugel zu – Achtung: Beim Programmieren wird immer bei 0 angefangen zu zählen! Für diese Kugel setzen wir nun die Position und weisen den Radius zu. Die Nachkommastelle wird dabei nicht wie üblich durch ein Komma abgetrennt, sondern durch einen Punkt (Programmiert wird üblicherweise auf Englisch).

Nachdem das Datenskript im Eingabefenster ③ eingegeben ist, drücke auf **Testen** um zu sehen, ob der Text so richtig ist. Im Meldungsfenster ④ sollte bei Erfolg die Nachricht erscheinen „Skript erfolgreich in Maschinensprache übersetzt“. Dann kann mit **Ausführen** das Skript ausgeführt werden. Als Ergebnis zeigt die 3D-Ansicht ① nun eine Kugel. Das ist natürlich noch etwas langweilig.



2. Hinzufügen von drei weiteren Kugeln

Um das Koordinatensystem zu markieren fügen wir nun drei weitere Kugeln hinzu, die jeweils auf der X-, Y- und Z-Achse liegen. Zuerst fügen wir eine weitere Kugel hinzu, die auf der X-Achse verschoben ist. Dazu müssen wir unseren Quelltext folgendermaßen verändern:

```
AnzahlKugeln = 2;  
Kugel[0].Position.Setzen(0.0, 0.0, 0.0);  
Kugel[0].Radius = 0.5;  
Kugel[1].Position.Setzen(1.0, 0.0, 0.0);  
Kugel[1].Radius = 0.3;
```

Die Anzahl der Kugeln wird auf 2 erhöht, die Eigenschaften der ersten Kugel (also `Kugel[0]`) bleiben gleich, der X-Wert der Position der zweiten Kugel (`Kugel[1]`) wird auf 1.0 gesetzt. Damit wir die beiden Kugeln unterscheiden können ist der Radius der zweiten Kugel kleiner, nämlich 0.3.

Verändere das Programm nun so, dass noch zwei weitere Kugeln hinzugefügt werden, eine auf der Y-Achse (mit Radius 0.2) und eine auf der Z-Achse (mit Radius 0.1). Nach erfolgreichem Ausführen sehen wir jetzt die vier Kugeln. Probier jetzt doch mal mit der 3D-Ansicht rum. Drehe sie, indem Du dir links Maustaste gedrückt hältst und die Maus bewegst. Vergrößere oder verkleinere die Ansicht indem Du die mittlere Maustaste so einsetzt.

3. Einfärben der Kugeln

Als nächstes ändern wir die Farben der Kugeln. Die Farbe ist eine weitere Eigenschaft der Kugeln. Wenn wir die Farbe der ersten Kugel auf schwarz setzen wollen, müssen wir folgende Quellcodezeile zu unserem Programm hinzufügen:

```
Kugel[0].Farbe = Farbe.Schwarz;
```

Weitere Farben, die verwendet werden können sind: Schwarz, Weiß, Grau, Rot, Grün, Blau, Türkis, Magenta, Gelb. Man kann auch andere Farben erzeugen, aber das kommt erst später. Setze nun die Farben der anderen Kugeln auf Rot, Grün und Blau.

4. Animation – Bewegte Kugeln

Als letzten Teil dieser Aufgabe sollen die Kugeln sich bewegen. Das funktioniert ähnlich wie bei einem Zeichentrickfilm, bei dem Einzelbilder gezeichnet und nacheinander angezeigt werden. Wir müssen also die Daten für verschiedene Zeitpunkte festlegen. Für jeden Zeitpunkt setzen wir die Position der Kugeln und der Computer wird diese dann für uns bewegen. Wir definieren nun eine Animation, die aus zwei Zeitpunkten besteht und in der sich die zweite Kugel von Position (0.0, 0.0, 0.0) nach (1.0, 0.0, 0.0) bewegt:

```
AnzahlKugeln = 2;
AnzahlZeitpunkte = 2;

AktuellerZeitpunkt = 0;
Kugel[0].Position.Setzen(0.0 , 0.0 , 0.0);
Kugel[0].Radius = 0.5;
Kugel[0].Farbe = Farbe.Schwarz;
Kugel[1].Position.Setzen(0.0 , 0.0 , 0.0);
Kugel[1].Radius = 0.3;
Kugel[1].Farbe = Farbe.Rot;

AktuellerZeitpunkt = 1;
Kugel[1].Position.Setzen(1.0 , 0.0 , 0.0);
```

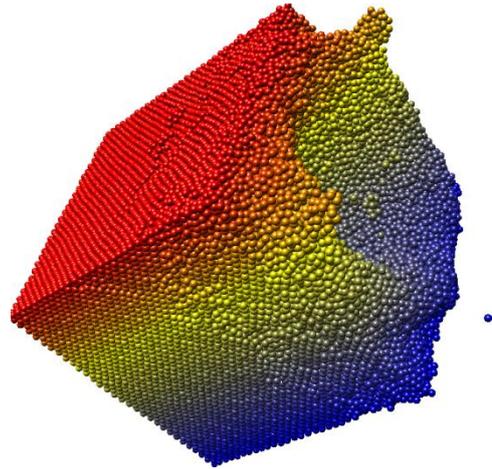
Zuerst setzen wir wie gewohnt die Anzahl der Kugeln, doch nun kommt noch die Anzahl der Zeitpunkte hinzu. Mit `AktuellerZeitpunkt = 0` wählen wir aus, welchen Zeitpunkt wir bearbeiten wollen, in diesem Fall den ersten (auch hier wird wieder bei 0 angefangen zu zählen). Wir setzen hier Positionen, Radien und Farben für beide Kugeln. Anschließend legen wir fest, welche Eigenschaften sich im zweiten Zeitpunkt ändern, in diesem Fall nur die Position der zweiten Kugel. Alle anderen Eigenschaften werden automatisch aus dem letzten Zeitpunkt übernommen.

Nach erfolgreichem Ausführen sehen wir in der 3D-Ansicht nur eine schwarze Kugel, weil die kleine rote Kugel innerhalb der schwarzen versteckt ist. Die wird erst im nächsten Zeitpunkt aus der schwarzen Kugel heraus kommen. Wähle dazu im Kontrollbereich unterhalb der 3D-Ansicht **2** den Punkt „Animation“ aus. Dort siehst Du einen Schieberegler der „die Zeit der Ansicht steuert“. Bewege den Regler hin und her und beobachte, wie die rote Kugel sich bewegt. Mit den Schaltflächen auf der linken Seite des Kontrollbereichs kann diese Bewegung auch als Animation  abgespielt (als Endlosschleife ) , oder gestoppt  werden.

Baue nun die Daten aus der Aufgabe 3 nach (die vier farbigen Kugel) mit drei Zeitpunkten wie hier beschrieben. Die Kugeln sollen alle im Nullpunkt (in der schwarzen Kugel) starten, sich dann entlang der X-, Y-, bzw. Z-Achse bewegen auf die Position wie in Aufgabe 2, und anschließend wieder zum Nullpunkt (in der schwarzen Kugel) zurückkehren.

Molekular-Daten aus der Physik

Da wir jetzt das Programm etwas kennengelernt haben und die ersten bunten, bewegten Kugeln gesehen haben, wird es Zeit für „echte“ Daten.



1. Einen einfachen Datensatz laden

Zunächst wollen wir eine Datei einer kleinen Simulation laden. Hierbei handelt es sich um einen simulierten Block Aluminium, in den mit einem Laser-Strahl ein Krater hineingeschossen wurde. Um diese Datei lesen zu können gibt es in unserem Visualisierungsprogramm Hilfestellungen die wir einsetzen können:

```
IMDDaten dat = new IMDDaten();
if (!dat.DateiLaden()) return;
```

Die zwei Zeilen sind schon etwas komplizierter und wir gehen hier nicht auf die Details ein. Mit der ersten Zeile teilen wir dem Programm mit, dass wir eine IMD-Datei laden wollen (IMD ist das Dateiformat der Simulation). Mit `dat` wird nun diese Daten-Datei verwendet. Die zweite Zeile fordert das Programm auf nun auch wirklich eine Datei zu laden und im Fehlerfall (z.B. wenn die Datei kaputt oder im falschen Dateiformat ist) wird das Datenskript hier abgebrochen.

Wenn Du dieses Skript ausführst wird ein Fenster erscheinen in dem Du die zu ladende Datei auswählen sollst. Geh in das „Datensätze“-Unterverzeichnis im „GirlsDay2013“-Verzeichnis und wähle dort die Datei „laser-klein.chkpt“ aus.

Wenn kein Fehler angezeigt wird, dann wurde die Datei erfolgreich geladen und im Meldungsfenster unter dem Programmierbereich sollten nun auch folgende Meldungen stehen: „Lade Datei ...\laser-klein.chkpt“ und „Datenspalten: number type mass x y z vx vy vz Epot eam_rho“. Diese Aufzählung von „Datenspalten“ zeigt uns welche Daten in der Datei pro Atom gespeichert sind.

In der 3D-Ansicht sehen wir allerdings keine Kugeln, weil wir dem Programm noch nicht gesagt haben, wie er aus den Daten der Datei Kugeln machen soll. Wir werden nun die „Datenspalten“ der Datei benutzen um unsere Kugeln zu erzeugen. Wir benutzen die Datenspalten „x“, „y“ und „z“ als Positionen der Kugeln.

```
IMDDaten dat = new IMDDaten();
if (!dat.DateiLaden()) return;

AnzahlKugeln = dat.AnzahlKugeln;

int x = dat.FindeSpalte("x");
int y = dat.FindeSpalte("y");
int z = dat.FindeSpalte("z");

for (int i = 0; i < AnzahlKugeln; i++) {
    Kugel[i].Position.X = dat.Zeitpunkt[0].AtomDaten[i, x];
    Kugel[i].Position.Y = dat.Zeitpunkt[0].AtomDaten[i, y];
    Kugel[i].Position.Z = dat.Zeitpunkt[0].AtomDaten[i, z];
}
```

Zuerst sagen wir dem Programm, dass wir genau so viele Kugeln haben wollen, wie in der Datei die von `dat` verwaltet wird gespeichert sind. Anschließend müssen wir die

Datenspalten x , y und z „raussuchen“, damit das Programm die Daten finden kann. Das geschieht mit den drei mittleren Zeilen.

Die Zeile, die mit „for“ beginnt, ist eine sogenannte Programm-Schleife. Hiermit sagen wir dem Programm, es soll von Null auf `AnzahlKugel` hochzählen. i ist hier ein Platzhalter der während dem Hochzählen die einzelnen Werte annimmt (0, 1, 2, 3, ...). Mit `Kugel[i]` greifen wir also auf die i -te Kugel zu. Wir setzen nun die Positionen aller Kugeln indem wir die X , Y und Z -Werte einzeln setzen (und nicht wie vorher mit dem Setzen-Kommando).

`dat` kann auch Datensätze mit mehreren Zeitpunkten verwalten (das werden wir später machen). Wir suchen uns nun aber erst einmal immer den 0-ten (also den ersten und einzigen) Zeitpunkt heraus und fragen von diesem die `AtomDaten` ab des i -ten Atoms der Datenspalten x , y und z .

Wenn wir das Datenskript so ausführen, dann sieht es allerdings etwas komisch aus, weil die Radien viel zu klein sein. Setze einen besseren Radius für die Kugeln, so dass die Daten so aussehen, wie auf dem Bild auf der vorherigen Seite.

2. Einfärben der Daten

Allerdings sind die Kugeln jetzt noch silbrig grau, was zwar durchaus aluminiumartig aussieht, aber irgendwie auch ziemlich langweilig. Also machen wir das ganze jetzt bunt.

In die „for“-Schleife aus der vorherigen Aufgabe kann man einfach z.B. mit `Kugel[i].Farbe = Farbe.Rot;` den ganzen Datensatz einfärben, indem jede Kugel auf diese Farbe gesetzt wird. Dann ist zwar alles farbig, aber immer noch langweilig. Besser ist es doch, wenn jede Kugel ihre eigene Farbe hat. Hierzu bietet sich der Befehl `Farbe.Mischen` an:

```
...
for (int i = 0; i < AnzahlKugeln; i++) {
    ...
    Kugel[i].Farbe = Farbe.Mischen(Farbe.Gelb, Farbe.Blau, 0.3);
    ...
}
```

Nun erscheint der ganze Datensatz grün, obwohl die einzigen Farben die angegeben sind Gelb und Blau sind. Warum? Die Angabe 0.3 (wieder eine Kommazahl 0,3 in englischer Schreibweise) befiehlt dem Programm die Farben Gelb und Blau im Verhältnis 0,7 zu 0,3 (oder 70% zu 30%) zu mischen. Gebt bei diesem Befehl eine Mischungszahl zwischen Null und Eins an. Ein Wert von Null ergibt genau die erste angegebene Farbe. Ein Wert von Eins die zweite Farbe und ein Wert von 0,5 (geschrieben als 0.5) eine Mischfarbe zu gleichen Teilen. Probiert etwas rum mit den acht Basisfarben die bei Aufgabe 1.3 angegeben sind. Die Mischung kann teilweise etwas andere Ergebnisse liefern, als Ihr es erwarten würdet. Das liegt daran, dass der Computer Lichtfarben mischt (nennt man „additiv Mischen“) was etwas anders funktioniert als flüssige oder feste Farben zu mischen (z.B. Wasserfarben, das ist dann „subtraktives“ Mischen). Wenn Ihr übrigens einen Wert größer als Eins oder kleiner als Null angebt verhält sich der Befehl so, als ob ihr Eins oder Null eingegeben hättet.

Um nun aber endgültig diese Eintönigkeit los zu werden, können wir diese Mischungszahl aber auch pro Kugel ausrechnen. Ein einfaches Beispiel ist:

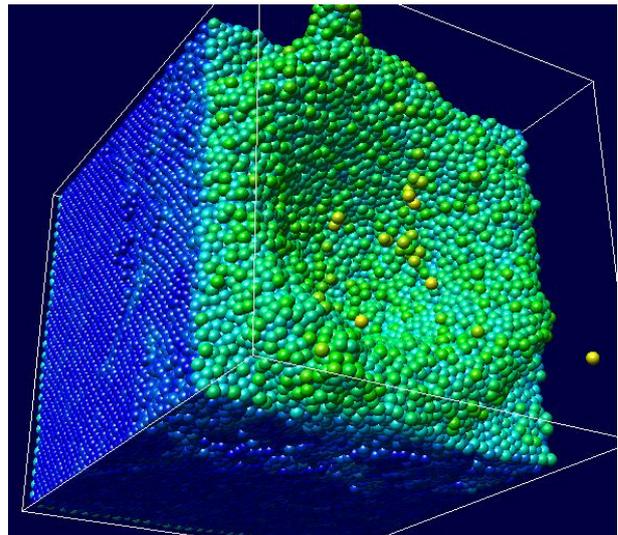
```

...
for (int i = 0; i < AnzahlKugeln; i++) {
    ...
    Kugel[i].Farbe = Farbe.Mischen(Farbe.Gelb, Farbe.Blau,
        (double)i / (double)AnzahlKugeln);
    ...
}

```

Damit wird anhand der Reihenfolge der Kugeln eingefärbt, da *i* ja die Nummer der Kugel ist. Teilen wir diesen Wert durch die Anzahl der Kugeln so zählen wir im Bereich von Null bis Eins (wenn es nicht ganz klar ist warum, dann rechnet euch das Ergebnis einfach mal von Hand aus, für einen Datensatz mit vier Kugeln). So wird's bunt. Jetzt kann man doch auch viel besser erkennen wie tief der Krater in dem Metallblock ist.

Die Farben der Kugeln werden aber meist dazu benutzt um Eigenschaften der Atome zu beschreiben. In dem Datensatz den wir hier geladen haben bietet sich die Datenspalte „eam_rho“ an. Diese Datenspalte speichert den Energiewert des Atoms. Ergänzt das Programm so, dass ihr auf diese Datenspalte jedes Atoms zugreifen könnt, genau so, wie ihr es für die Datenspalten x, y und z macht. Benutzt diesen Wert um die Farbe zu mischen. Da die Werte alle etwa zwischen Null ein Eins liegen ergibt sich eine ziemlich gute Darstellung.



Proteine – Daten aus der Biochemie

Moleküle bestehen aus mehreren verschiedenen Atomen, welche durch Atombindungen verbunden sind. Jedes Element, das heißt jede Art von Atom, ist durch eine ganz bestimmte Anzahl von Bindungen mit anderen Atomen verbunden. Kohlenstoff beispielsweise hat vier Verbindungen zu anderen Atomen. Mit einem Molekülbaukasten kann man kleinere Moleküle zusammenbasteln. Die Atome sind meistens kleine Kugeln mit Löchern, welche durch Stäbchen verbunden werden können. Diese Baukasten-Modelle werden deshalb Kugel-Stäbchen-Modelle genannt.

Proteine sind Moleküle, die aus kleineren Molekülen, den Aminosäuren, zusammengesetzt sind. Es gibt 20 Aminosäuren, sie bestehen aus den Elementen Kohlenstoff, Wasserstoff, Stickstoff, Sauerstoff und Schwefel. Ein bestimmter Teil ist bei allen Aminosäuren gleich, das Rückgrat. Jede Aminosäure hat noch einen individuellen Teil, die Seitenkette. In einem Protein sind die Rückgrate vieler Aminosäuren zu einer Kette verbunden, die Seitenketten stehen nach den Seiten ab (daher der Name). Die meisten Proteine bestehen aus sehr vielen Aminosäuren und können deshalb nicht mit einem Molekülbaukasten zusammengesteckt werden, außer man richtig viel Zeit, Langeweile und richtig viele Plastikugeln (eine Aminosäure hat etwa 10 Atome, d.h. ein Protein, welches aus 25 Aminosäuren besteht hat bereits etwa 250 Atome!). Deshalb ist 3D-Visualisierung für Proteine die



bessere Lösung. Am Computer können wir sehr große Kugel-Stäbchen-Modelle anzeigen lassen und so die Form eines Proteins sehen.

In dieser Aufgabe wollen wir Protein-Daten laden und darstellen.

1. Laden und Zeichnen eines Proteins

Dateien, welche Proteine enthalten, können wir aus der Protein-Datenbank (PDB) herunterladen (URL: <http://www.pdb.org>). Für das Laden dieser Dateien gibt es wie in der letzten Aufgabe eine Hilfsfunktion, die wir verwenden:

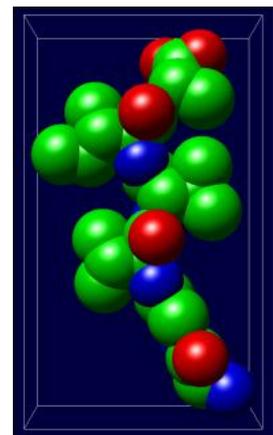
```
PDBDaten dat = new PDBDaten();  
if (!dat.DateiLaden()) return;
```

Zuerst legen wir mit `PDBDaten dat = new PDBDaten()` fest, dass wir eine PDB-Datei laden wollen. Die zweite Zeile lädt die Datei und sagt uns, ob dabei ein Fehler aufgetreten ist (dann wird das Skript durch `return` beendet). In dem Fenster, das auftaucht wenn Du das Skript startest, wird wie in Aufgabe 2 eine Datei ausgewählt. Wähle im „Proteine“-Unterverzeichnis des „Datensätze“-Verzeichnis die Datei „2ONV.pdb“ aus.

Die Protein-Daten, die wir nun gelesen haben, bestehen aus den einzelnen Atomen des Proteins. Jedes Atom hat eine Position und einen Radius und soll als Kugel gezeichnet werden. Dabei gehen wir ganz ähnlich vor wie in den ersten beiden Aufgaben:

```
AnzahlKugeln = dat.AnzahlKugeln;  
  
for (int j = 0; j < dat.AnzahlKugeln; j++) {  
    Kugel[j].Position = dat.AtomDaten[0][j].Position;  
    Kugel[j].Radius = dat.AtomTypRadius(dat.AtomDaten[0][j].Typ);  
    Kugel[j].Farbe = dat.AtomTypFarbe(dat.AtomDaten[0][j].Typ);  
}
```

Die Anzahl der Atome bzw. Kugeln des Proteins ist in der Datei gespeichert, wir können sie mit `dat.AnzahlKugeln` abfragen. Nun laufen wir mit einer `for`-Schleife über alle Atom-Kugeln und setzen jeweils die Position, den Radius und die Farbe der neuen Kugel, die wir zeichnen wollen. Außer der Position der Kugel ist für jedes Atom noch gespeichert, von welchem Typ es ist. Der Typ ist das Element, also Kohlenstoff, Wasserstoff und so weiter. Für diese Typen sind Farben und Radien in `dat.AtomTypFarbe` und `dat.AtomTypRadius` gespeichert. Mit den letzten beiden Zeilen in der `for`-Schleife weisen wir der Kugel, die für das `j`-te Atom gezeichnet werden soll, den Radius und die Farbe zu, die zu ihrem Atom-Typ gehören. Wie in der letzten Aufgabe mit den IMD-Daten kann `dat` auch für PDB-Dateien Datensätze mit mehreren Zeitpunkten verwalten. Mit `dat.AtomDaten[0][j]` greifen wir im ersten Zeitschritt (also Zeitschritt 0) auf das `j`-te Atom zu.



Wir sehen jetzt alle Atome des Proteins als Kugeln. Allerdings wollen wir ja das Kugel-Stäbchen-Modell nachbilden, deshalb müssen die Kugeln verkleinert werden und die Verbindungen einzeichnen. Als erstes verkleinern wir die Kugeln, indem wir den Radius durch vier teilen. Das Zeichen für „geteilt durch“ ist der Schrägstrich „/“.

Die Verbindungen zwischen den Atomen werden direkt unabhängig gespeichert. Glücklicherweise bereitet uns die Hilfsstruktur `PDBDaten dat` diese Daten bereits vor und wir müssen sie nur übernehmen (ähnlich zu den Atom/Kugel-Positionen):

```
AnzahlVerbindungen = dat.AnzahlVerbindungen;

for (int i = 0; i < dat.AnzahlVerbindungen; i++) {
    Verbindung[i] = dat.Verbindung[i];
}
```

Zuerst setzen wir die Anzahl der Verbindungen und danach laufen wir (wie bei den Kugeln) mit einer `for`-Schleife über alle in `dat` vorhandenen Verbindungen und speichern jeweils die `i`-te Verbindung in `Verbindungen[i]`.

2. Einfärben nach B-Faktor

Für die Atome in einer PDB-Datei können noch zusätzliche Eigenschaften außer der Position gespeichert werden. Einer dieser Werte ist der Temperaturfaktor (auch B-Faktor genannt). Dieser ist für das Verständnis der Funktionsweise eines Proteins wichtig und daher wollen wir in dieser Teilaufgabe diesen B-Faktor auf die Farbe abbilden. Dazu verwenden wir wieder den Befehl `Farbe.Mischen`, den wir in Aufgabe 2.2 kennen gelernt haben. Allerdings liegen die Werte diesmal nicht zwischen Null und Eins, deshalb müssen wir sie umrechnen. Dazu müssen wir zuerst herausfinden, was der kleinste und was der größte Wert für den B-Faktor in diesem Protein ist. Diese beiden Werte speichern wir uns dann, um sie später zum Einfärben der Kugeln zu verwenden. Wir erweitern unser Skript also um folgende Zeilen:

```
double BFaktor, MaximalerBFaktor = 0.0, MinimalerBFaktor = 0.0;
...
for (int j = 0; j < dat.AnzahlKugeln; j++) {
    ...
    BFaktor = dat.AtomDaten[0][j].BFaktor;
    if (j == 0) {
        MaximalerBFaktor = BFaktor;
    }
    if (MaximalerBFaktor < BFaktor) {
        MaximalerBFaktor = BFaktor;
    }
    ...
}
```

In der ersten Zeile legen wir drei „Variablen“ an. „Variablen“ können Werte speichern. Das `double` am Anfang der Zeile legt fest, dass wir in `BFaktor`, `MaximalerBFaktor` und `MinimalerBFaktor` Zahlen mit Nachkommastelle speichern wollen. In der Schleife speichern wir zunächst den B-Faktor des aktuellen Atoms, damit wir mit dem kürzeren Namen komfortabler arbeiten können. Wenn wir beim ersten Atom sind, d.h. wenn `j` gleich Null ist, setzen wir den gelesenen B-Faktor als den maximalen B-Faktor.

Der Befehl `if(...) {...}` ist eine Abfrage: Nur wenn die Bedingung in den runden Klammern erfüllt ist werden die Befehle zwischen den geschwungenen Klammern ausgeführt. Falls die Bedingung also falsch ist (also immer wenn `j` nicht Null ist) werden die Befehle in den geschwungenen Klammern übersprungen (Achtung: In unserer Programmiersprache sind zwei Gleichheitszeichen `==` ein Vergleichstest während ein einzelnes Gleichheitszeichen `=` eine Zuweisung ist). Die zweite `if`-Abfrage überprüft also jedes Mal, wenn ein B-Faktor gelesen wurde, ob der B-Faktor, der gerade als maximaler B-Faktor gespeichert ist, kleiner ist als der neue B-Faktor. Wenn das der Fall ist wird der neue B-Faktor als maximaler B-Faktor gespeichert. Wenn die `for`-Schleife komplett abgearbeitet ist steht also in `MaximalerBFaktor` der maximale B-Faktor aller Atome. Füge nun die nötigen Befehle hinzu, dass auch der minimale B-Faktor gespeichert wird.

Nun färben wir die Atome nach ihrem B-Faktor ein. Dazu benötigen wir eine zweite for-Schleife, die wieder über alle Atome läuft, diesmal aber nur die Farbe mischt, die dieses Atom bekommen soll:

```
for (int j = 0; j < dat.AnzahlKugeln; j++) {
    BFaktor = dat.AtomDaten[0][j].BFaktor;
    Kugel[j].Farbe = Farbe.Mischen(Farbe.Blau, Farbe.Rot,
        (BFaktor - MinimalerBFaktor) /
        (MaximalerBFaktor - MinimalerBFaktor) );
}
```

Wir mischen hier für jede Kugel eine Farbe zwischen Rot und Blau (oder irgendwelchen anderen zwei Farben, die Dir gefallen). Wie in Aufgabe 2.2 teilen wir durch den Wertebereich, um Werte zwischen Null und Eins zu erhalten. Wir müssen dazu zuerst den minimalen B-Faktor vom B-Faktor abziehen. Überlege Dir mit einfachen Zahlen warum das so funktioniert (z.B.: MinimalerBFaktor sei 2, MaximalerBFaktor sei 10 und BFaktor sei 6).

3. Datensätze mit mehreren Zeitpunkten

Als letzten Teil dieser Aufgabe wollen wir jetzt einen Datensatz mit mehreren Zeitpunkten laden. Die Datei „TEM_simulation30f.pdb“ enthält 30 Zeitpunkte einer Simulation. Du findest diese Datei im „Datensätze\Protein“-Verzeichnis. Im Datenskript fügen wir nun folgende Zeilen hinzu beziehungsweise verändern das Skript:

```
AnzahlZeitpunkte = dat.AnzahlZeitpunkte;
...
for (int i = 0; i < dat.AnzahlZeitpunkte; i++) {
    AktuellerZeitpunkt = i;
    for (int j = 0; j < dat.AnzahlKugeln; j++) {
        Kugel[j].Position = dat.AtomDaten[i][j].Position;
        ...
    }
}
```

Wie in Aufgabe 1.4 wird zuerst die Anzahl der Zeitpunkte festgelegt, diesmal lesen wir jedoch die Anzahl aus `dat`. Um unsere bisherige `for`-Schleife, welche über die Atome eines Zeitpunkts läuft, kommt nun eine zweite `for`-Schleife, welche über alle Zeitpunkte läuft. Die Schleife über alle Atome wird nun für jeden Zeitpunkt ausgeführt, das heißt für jeden Zeitpunkt wird die Kugelposition gesetzt. Da wir nun mehrere Zeitpunkte haben, müssen wir mit `dat.AtomDaten[i][j]` jeweils das `j`-te Atom im `i`-ten Zeitpunkt lesen. Vervollständige das Datenskript so, dass auch die Farbe und der Radius richtig gesetzt werden. Du kannst Dir die Animation wie in Aufgabe 1.4 beschrieben anschauen.

Zeitabhängige Physik-Daten

Genauso wie zeitabhängige Daten für Proteine geladen und dargestellt werden können, kann das natürlich auch mit Daten aus physikalisch motivierten Simulationen gemacht werden.

1. Zeitabhängige Daten aus physikalischen Simulationen

Wie in Aufgabe 1.4 wird zuerst die Anzahl der Zeitpunkte festgelegt, diesmal lesen wir jedoch wieder mit Hilfe von `IMDDaten` unsere Datei ein. Zeitabhängige Daten befinden sich im Verzeichnis „Datensätze\MethanEthan“. Wir wollen jetzt mit „MethanEthan“ experimentieren. Hierbei handelt es sich um eine Simulation zweier Tröpfchen (eines Methan und eines Ethan) die aufeinander prallen.

Lade die Daten wie bei Aufgabe 2.1. beschrieben. Sobald Du eine Datei auswählst wirst Du gefragt, ob Du alle passenden Dateien laden willst, oder nur die eine ausgewählte. Wenn Du „Ja“ wählst, werden alle Dateien geladen. Jede Datei speichert einen Zeitpunkt und zusammen bilden sie den zeitabhängigen Datensatz.

Setze wie in Aufgabe 2.1. die Positionen anhand der Datenspalten x, y und z, jedoch diesmal für jeden Zeitschritt, wie in Aufgabe 3.3. beschrieben.

Wenn Du die Animation abspielst siehst Du wie die zwei Tropfen aufeinander prallen und miteinander verschmelzen. Leider sind so viele Kugeln unterwegs, dass man Methan und Ethan nicht mehr unterscheiden kann. Daher wollen wir nun die beiden Materialien unterschiedlich einfärben.

Aus den Info-Meldungen kannst Du sehen, dass die Dateien eine Datenspalte „type“ haben. Diese speichert als Nummer den Typ des Atoms. Benutze diesen Wert (ähnlich zu „eam_rho“ in Aufgabe 2.4) um die beiden Typen unterschiedlich einzufärben. Jetzt erkennt man doch gleich viel besser was wirklich passiert.

Damit ist das Programm des Girls' Day 2013 abgeschlossen. Wir hoffen Du hattest ein wenig Spaß dabei.