Computergrafik und Visualisierung II

Marilena Froehlich, Mathias Gewissen, Sebastian Mischke

9. Mai 2016

Inhaltsverzeichnis

Teil I
Vektor, Vektor2D und Vektor3D

1 Klassen

Gearbeitet wird mit drei Klassen: der Basisklasse Vektor und den abgeleiteten Klassen Vektor2D und Vektor3D. In Vektor werden sämtliche Methoden implementiert, sodass die Klassen Vektor2D und Vektor3D lediglich aus Konstruktoren bestehen. Außerdem enthält die Klasse Vektor ein double-Array, in dem die Komponenten gespeichert werden, sowie einen int-Wert, in welchem die Anzahl an Komponenten gespeichert wird.

```
1  public class Vektor {
2      protected double[] array;
3      protected int dimension;
4
5      public Vektor(int dimension, double[] array) throws Exception {
6          this.dimension = dimension;
7          this.array = new double[dimension];
8          setPosition(array);
9      }
10 }
```

2 Konstruktoren

2.1 Standard-Konstruktor

Der Standard-Konstruktor der Klasse Vektor ist private, damit diese von außerhalb der Klasse nicht aufgerufen werden kann. Die von Vektor abgeleiteten Klassen hingegen erzeugen beim Aufruf des Standard-Konstruktors einen Nullvektor mit der zugehörigen Dimension.

2.2 Weitere Konstruktoren

Es werden verschiedene Eingabemöglichkeiten zugelassen, mit welchen der Konstruktor aufgerufen werden kann:

- ein double-Array, das die Koordinaten des Vektors enthält
- einzelne double-Werte
- eine Dimension und ein Array bzw. ein Array und eine Dimension (hierbei wird die Dimension des Vektors in einer int-Variable dimension gespeichert)

2.3 Kopierkonstruktor

Der Kopierkonstruktor nimmt einen Vektor als Eingabeparameter und erzeugt einen neuen Vektor mit den gleichen Komponenten. Die clone()-Funktion ruft den Kopierkonstruktor für einen Vektor auf und gibt das neu erzeugte Objekt zurück. Dadurch können die Funktionen der Vektor-Klasse genutzt werden, ohne den Vektor dabei zu verändern.

3 Getter und Setter

Der Zustand eines Vektors (Komponenten, Dimension, Länge) kann abgefragt und teilweise verändert werden

3.1 getDimension()

getDimension() gibt den Wert der Variable dimension, also die Dimension des Vektors zurück.

3.2 getArray()

getArray() gibt das Array, das die Komponenten des Vektors enthält, zurück.

3.3 getElem()

Erhält einen int-Wert und gibt das Element, das sich an dieser Position im Array befindet zurück. Eine einzelne Koordinate des Vektors wird zurückgegeben. Bei einem Übergabewert, der kleiner 0 oder größer als die Dimension-1 ist, wird eine Exception geworfen.

```
1 public double getElem(int dimension) throws Exception {
2     if ((dimension >= this.dimension) || (dimension < 0)) {
3         throw new Exception("Invalid dimension");
4     }
5     return array[dimension];
6 }
```

3.4 length()

Zunächst existiert eine Funktion lengthsquare(), welche das Quadrat der Länge eines Vektors liefert. Dafür werden die Komponenten jeweils quadriert und aufsummiert. length() liefert dann die Länge des Vektors, indem die Wurzel des Returnwertes von lengthsquare() zurückgegeben wird.

3.5 toString()

Es wird ein String angelegt, der aus einer öffnenden Klammer sowie dem ersten Element des Vektors besteht. Durch eine for-Schleife werden alle weiteren Komponente konkateniert, jeweils mit Komma getrennt. Daraufhin wird eine schließenden Klammer angehängt und der String zurückgegeben.

```
1 public String toString() {
2     StringBuilder result = new StringBuilder("( " + array[0]);
3     for (int i = 1; i < dimension; i++) {
4         result.append(", " + array[i]);
5     }
6     result.append(" )");
7     return (result.toString());
8 }
```

3.6 setPosition()

Die Koordinaten des Vektors werden beliebig verändert. Hierbei ist wichtig, dass die Länge des Eingabearrays der Dimension des Vektors entspricht. Andernfalls wird eine Exception geworfen.

```
1 public void setPosition(double... array) throws Exception {
2     if (array.length == dimension) {
3         for (int i = 0; i < dimension; i++) {
4             this.array[i] = array[i];
5         }
6     } else {
7         throw new Exception("Array does not have the right length");
8     }
9 }
```

4 Vergleiche

Die folgenden Funktionen vergleichen den ursprünglichen Vektor mit einem Wert oder einem gegebenen Vektor und liefern als Rückgabewert des Typs boolean. Der Vektor wird hierbei nicht verändert.

4.1 isNullVector()

isNullVector() liefert true, wenn es sich bei dem Vektor um einen Nullvektor handelt, ansonsten false. Dafür wird die Funktion lenghtsquare() aufgerufen. Ist der Returnwert 0, so handelt es sich um einen Nullvektor. Im Vergleich zur Benutzung der Funktion length() spart man sich das aufwendige Wurzelziehen.

```
1 public boolean isNullVector() {
2     return lengthsquare() == 0;
3 }
```

4.2 isEqual()

isEqual() vergleicht komponentenweise den übergebenen Vektor mit dem ursprünglichen Vektor. Zunächst wird geprüft, ob beide Vektoren die gleiche Dimension besitzen. Trifft dies nicht zu, wird ein false zurückgegeben. Ansonsten werden in einer for-Schleife die Komponenten verglichen. Sollte irgendwo eine Ungleichheit auftreten, wird sofort abgebrochen und ebenfalls ein false zurückgeben. Kommt die for-Schleife an ihr Ende, ohne eine Ungleichheit zu finden, wird ein true zurückgegeben.Bei paarweiser Übereinstimmung aller Werte wird true zurückgeliefert.

```
1  public boolean isEqual(Vektor vec) {
2      if (this.dimension != vec.dimension) {
3          return false;
4      }
5      for (int i = 0; i < dimension; i++) {
6          if (this.array[i] != vec.array[i]) {
7              return false;
8          }
9      }
10     return true;
11 }
```

4.3 isNotEqual()

Die Funktion isNotEqual() gibt den negierten Wert der Funktion isEqual() zurück.

5 Mathematische Operationen

Die mathematischen Operationen verändern den ursprünglichen Vektor, indem sie eine Rechenoperation auf diesen anwenden. Der Rückgabewert ist der Vektor selbst, um eine Hintereinanderausführung mehrerer Operationen zu ermöglichen.

5.1 add()

add() verändert den ursprünglichen Vektor, indem er auf diesen einen zweiten Vektor komponentenweise aufaddiert. Als Returnwert gibt der Vektor sich selbst zurück, um eine Verkettung mehrerer Operationen zu ermöglichen. Vor der Ausführung wird mit Hilfe der Funktion CheckDimension() überprüft, ob die beiden Vektoren die gleiche Länge haben. Außerdem wird durch Aufruf der Funktion CheckAddOverflow auf einen Überlauf getestet.

```
1 public Vektor add(Vektor vec) throws Exception {
2     Function.CheckDimensions(vec.dimension, dimension);
3     for (int i = 0; i < dimension; i++) {
4         Function.CheckAddOverflow(array[i], vec.array[i]);
5         array[i] += vec.array[i];
6     }
7     return this;
8 }
```

5.2 mult()

mult() verändert den ursprünglichen Vektor indem er jede Komponente mit einem double-Wert, dem Skalar, multipliziert. Auch hierbei ist wieder zu beachten, das kein Überlauf entsteht. Dies wird mit Hilfe von CheckMultOverflow gewährleistet.

5.3 negate()

Um einen Vektor zu negieren, wird die mult()-Funktion mit dem Wert -1 aufgerufen.

5.4 div()

div() dividiert komponentenweise durch einen double-Wert d. Die Division erfolgt indirekt durch eine Multiplikation der Komponenten mit dem Reziproken von d. Dafür wird die Funktion mult() aufgerufen. Vor der Ausführung wird überprüft, dass d ungleich 0 ist, also nicht durch 0 geteilt wird. Ist dies der Fall, wird eine Exception geworfen.

```
1 public Vektor div(double d) throws Exception {
2     if (d == 0) {
3         throw new Exception("Cannot divide by zero");
4     } else {
5         return mult(1. / d);
6     }
7 }
```

5.5 sub()

Die Funktion sub() subtrahiert komponentenweise einen zweiten, übergebenen Vektor vom ursprünglichen Vektor. Dazu wird der übergebene Vektor mit Hilfe von negate() negiert und durch den Aufruf von add() komponentenweise auf den ursprünglichen Vektor addiert.

5.6 normalize()

normalize() ändert die Länge des Vektors auf 1. Hierzu wird mit Hilfe von length() die Länge des Vektors berechnet und durch Aufruf der Funktion div() der Vektor durch diese Länge dividiert. Vor der Ausführung wird ausgeschlossen, dass es sich um einen Nullvektor handelt.

```
1 public Vektor normalize() throws Exception {
2     if (!isNullVector())
3         return div(length());
4     else
5         throw new Exception("Nullvector cannot be normalized");
6 }
```

5.7 abs()

In Funktion abs() wird in einer for-Schleife von alle Komponenten eines Vektors der Betrag gebildet.

Teil II
Lineare Algebra

6 Basisoperationen

Die Funktionen add(), sub(), mult(), div(), negate(), normalize(), abs() werden alle auf die gleiche Weise implementiert. Über die clone()-Funktion wird ein neues Objekt der Klasse Vektor erzeugt. Dann wird die zugehörige Funktion aus der Klasse Vektor aufgerufen und das neu instanziierte Objekt zurückgegeben.

```
1 public static Vektor add(Vektor v1, Vektor v2) throws Exception {
2     return (v1.clone()).add(v2);
3 }
```

7 Erweiterte Operationen

Die Funktionen manhattanDistance(), euklDistance(), dotProduct() und crossProduct() werden mit 2 Vektoren als Übergabeparameter aufgerufen. Dabei wird stets am Anfang der Funktion geprüft, ob die übergebenen Vektoren die gleiche Dimension besitzen. Ist dies nicht der Fall, wird eine Exception geworfen.

7.1 Manhattan-Distanz

Die Differenzen der Komponenten werden in einer for-Schleife berechnet und aufaddiert. Die Summe wird anschließend zurückgegeben.

```
1 public static double manhattanDistance(Vektor v1, Vektor v2) throws Exception {
2     Function.CheckDimensions(v1.getDimension(), v2.getDimension());
3     double result = 0;
4     for (int i = 0; i < v1.getDimension(); i++) {
5         result += Math.abs(v1.getElem(i) - v2.getElem(i));
6     }
7     return result;
8 }
```

7.2 Euklidischer Abstand

Die Differenzen der Komponenten werden in einer for-Schleife quadriert und aufaddiert. Die Wurzel der Summe wird anschließend zurückgegeben.

7.3 Determinante

Die Funktion determinante() bekommt ein Array an Vektoren übergeben und überprüft in einer for-Schleife, dass die Anzahl der Vektoren im Array auch mit der Dimension der übergebenen Vektoren übereinstimmt. Eine Switch-Case-Anweisung entscheidet welche Formel zur Berechnung der Determinante ausgeführt wird.

7.4 Skalarprodukt

In der Funktion dotProduct() werden die Produkte der Komponenten in einer for-Schleife berechnet und aufaddiert. Die Summe wird anschließend zurückgegeben.

7.5 Kreuzprodukt

In der Funktion crossProduct() wird in einer Switch-Case-Anweisung entschieden, welche Formel zur Berechnung des Ergebnisvektors ausgeführt wird.

Wenn es sich um zweidimensionale Vektoren handelt, wird die Funktion determinante mit den beiden Vektoren ausgeführt und das Ergebnis als eindimensionaler Vektor zurückgegeben. Wenn es sich um dreidimensionale Vektoren handelt, ergibt sich die Berechnung wie folgt, wobei v1, v2 und v3 um die eingegebenen Vektoren handelt:

$$det \begin{pmatrix} v1_x & v2_x & v3_x \\ v1_y & v2_y & v3_y \\ v1_z & v2_z & v3_z \end{pmatrix} = v1_x v2_y v3_z + v2_x v3_y v1_z + v3_x v1_y v2_z - v3_x v2_y v1_z - v1_x v3_y v2_z - v2_x v1_y v3_z$$

8 Trigonometrische Funktionen

8.1 cosEquation()

Ruft die Funktion dotProduct() mit den eingegebenen Vektoren auf und teilt das Ergebnis durch das Produkt der beiden Längen der Vektoren.

8.2 sinEquation()

Ruft die Funktion crossProduct() mit den eingegebenen Vektoren auf und teilt die Länge des Ergebnisvektors durch die Summe der beiden übergebenen Vektoren.

8.3 Grad- und Bogenmaß

Die Funktionen radToDegree() und degreeToRad() erhalten einen double-Wert der entweder für ein Bogenmaß oder ein Gradmaß steht und geben das jeweilige Pendant zurück. Dabei gilt $\frac{Degree}{180} = \frac{Rad}{\pi}$.

8.4 Arcus-Funktionen

Die Funktionen angleRad und angleDegree bekommen einen double-Wert, sowie einen String, der angibt, ob es sich beim übergebenen Wert um einen sin-, cos- oder tan-Wert handelt. Auf diesen übergebenen Wert wird dann die zugehörige Arcus-Funktion angewendet und das Ergebnis zurückgegeben.

```
1  public static double angleRad(double d, String trig) throws Exception {
2      switch (trig) {
3      case "sin":
4          return Math.asin(d);
5      case "cos":
6          return Math.acos(d);
7      case "tan":
8          return Math.atan(d);
9      default:
10         throw new Exception("Invalid trigonometric function");
11     }
12 }
```

9 Ausgabe

9.1 show()

Die Funktion bekommt einen Vektor übergeben und gibt dessen Ergebnis der toString()-Funktion auf der Konsole aus.

```
1  public static void show(Vektor vec) {
2      System.out.println(vec.toString());
3  }
```