Beleg Computergrafik und Visualisierung II - Teil I

Wiebke Rochler, Christopher Praas, Anna Krauß

5. Mai 2017

Inhaltsverzeichnis

1 Einleitung

Dies ist die Dokumentation zum Quellcode für das den Beleg im Fach Computergrafik und Visualisierung II bei Prof. Dr. Marco Block-Berlitz. Im ersten Teil des Belegs war es Aufgabe drei Klassen in Java zu implementieren. Bei der Umsetzung war darauf zu achten, dass Methoden erstellt werden, die in der Lage sind mögliche Fehler aufspüren und entsprechend zu reagieren. Die verschiedenen Klassen samt Methoden wurden in einem Package realisiert. In einem zweiten Package sind alle Tests zu den jeweiligen Methoden implementiert.

2 Erläuterung der verwendeten Klassen

2.1 Vektor2D

Die Klasse Vektor2D repräsentiert 2D-Vektoren. Die Koordinaten werden jeweils durch double-Werte dargestellt. Zunächst besitzt die Klasse drei verschiedene Konstruktoren, durch die es möglich ist auf verschiedene Art und Weisen Objekte der Klasse Vektor2D zu erstellen. Des Weiteren beinhaltet die Klasse die Grundrechenarten Addition, Subtraktion, Multiplikation und Division. Darüber hinaus werden Methoden angeboten, mithilfe derer Vektoren verglichen, normiert und an eine bestimmte Position gesetzt werden können. Außerdem kann die Vektorlänge berechnet und geprüft werden ob es sich um einen Nullvektor handelt. Alle Methoden wurden testgetrieben entwickelt, weshalb sie bestimmte Fehler erkennen und behandeln können. Wenn ein solcher Fehlerfall eintritt, wird eine Exception geworfen um dies kenntlich zu machen. Im Folgenden wird auf die einzelnen Operationen und die eventuell auftretenden Fehler eingegangen.

2.1.1 Methoden

Addition: Bei der Addition bestünde die Möglichkeit, dass es bei zu großen bzw. zu kleinen Zahlen zu einem Überlauf im positiven oder negativen Bereich kommen könnte. Dies lässt sich durch folgende Überlegungen ermitteln:

```
public void add(Vektor2D b) throws Exception
   {
      if((x<0 && b.x<0) || (x>0 && b.x>0) || (y<0 && b.y<0) || (y>0 && b.y>0))
         if((Math.abs(x)+Math.abs(b.x) >= Double.MAX_VALUE) ||
             (Math.abs(y)+Math.abs(b.y) >= Double.MAX_VALUE))
            throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
               beachten!");
```

Es wird also getestet ob die Koordinaten des aufrufenden und des übergebenen Vektors beide größer oder kleiner Null sind. Wenn das der Fall ist, wird aus diesen der Betrag gebildet und dann geprüft ob deren Addition größer/gleich dem maximalen Wert des darstellbaren Bereichs ist. Ist dies der Fall, wird eine Exception geworfen und der Nutzer über den Fehler informiert. (Mit der Bildung des Betrags der Koordinaten genügt der Test auf den Maximalwert von Double.)

```
      if((x>0 && b.x<0) || (x<0 && b.x>0) || (y>0 && b.y<0) || (y<0 && b.y>0))
         if((Math.abs(x)-Math.abs(b.x) >= Double.MAX_VALUE) ||
             (Math.abs(y)-Math.abs(b.y) >= Double.MAX_VALUE))
            throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
               beachten!");
   }
```

Des Weiteren bestünde die Möglichkeit einen sehr großen und einen kleinen Wert zu addieren, was ebenfalls zu einem Überlauf führen könnte. Daher wird getestet ob die einzelnen Koordinaten des aufrufenden und des übergebenen größer und/oder kleiner

Null sind. Wenn das der Fall ist, wird aus diesen der Betrag gebildet und dann geprüft ob deren Subtraktion größer/gleich dem maximalen Wert des darstellbaren Bereichs ist. Ist dies der Fall, wird eine Exception geworfen und der Nutzer über den Fehler informiert.

Subtraktion: Prinzipiell können hier die gleichen Fehler auftreten wie bei der Addition. Es muss lediglich bei der Prüfung auf die Vorzeichen geachtet werden. Aufgrund des Bildens der Beträge in der Überprüfung auf Überlauf von der Addition können wir die gleiche Überprüfung auch für die Subtraktion verwenden.

Multiplikation: Bei der Multiplikation wird ein Vektor mit einer Doublezahl multipliziert. Dabei sollte auch dort darauf geachtet werden, dass kein Überlauf zustande kommt. Das verhindern wir, indem wir auch hier darauf testen ob die Operation einen größeren Wert erzeugen würde, als durch Double darstellbar. Auch hier werfen wir im Fehlerfall eine Exception und geben eine Fehlermeldung zurück.
Programmcode zur Verdeutlichung:

```
public void mult(double b) throws Exception
   {
      if((Math.abs(x)*Math.abs(b) >= Double.MAX_VALUE) ||
          (Math.abs(y)+Math.abs(b) >= Double.MAX_VALUE))
         throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
            beachten!");
      x*=b;
      y*=b;
   }
```

Division: Bei der Division muss abgesehen von dem Test auf positiven und negativen Überlauf auch getestet werden, ob der Divisor Null ist, da Division durch Null nicht zulässig ist. Wurde dies durch den Nutzer nicht beachtet, wird eine Exception geworfen und darauf hingewiesen. Für den Test auf Überlauf muss überprüft werden ob der Divisor zwischen Null und Eins liegt, da das Ergebnis in dem Fall dann größer ist als der Dividend bzw. im Falle b==1 gleich groß. Also wird auch hier der Betrag der Koordinaten gebildet und die Division auf Überlauf getestet. Im Fehlerfall erfolgt auch hier eine Exception.
Programmcode zur Verdeutlichung:

```
   public void div(double b) throws Exception
   {
      if(b==0)
         throw new Exception("Division durch 0 ist nicht zulässig!");

      if((b<1 && b > 0) || b==1)
         if((Math.abs(x)/Math.abs(b) >= Double.MAX_VALUE) ||
             (Math.abs(y)/Math.abs(b) >= Double.MAX_VALUE))
            throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
               beachten!");
```

```
    x/=b;
    y/=b;
}
```

setPosition(): Diese Funktion bietet die Möglichkeit einem Vektor durch Übergabe von x- und y-Koordinate eine neue Position zuzuweisen. Eventuell könnten diese übergebenen Werte größer sein als der double-Maximalwert, was wir durch vorheriges Prüfen ausgeschlossen haben.
Programmcode zur Verdeutlichung:

```
public void setPosition(double x1, double y1) throws Exception
{
    if((Math.abs(x1)>=Double.MAX_VALUE) || (Math.abs(y1)>=Double.MAX_VALUE))
        throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
            beachten!");

    x=x1;
    y=y1;
}
```

isNullVector(): In der Methode wird überprüft, ob der aufrufende Vektor ein Nullvektor ist. Dabei war nur die Korrektheit der Funktion zu prüfen.

isEqual(): In isEqual() wird überprüft, ob der aufrufende und der übergebene Vektor den gleichen Inhalt haben. Dabei war nur die Korrektheit der Funktion zu prüfen. (Vergleich Inhalt und nicht die Objektreferenzen)

isNotEqual(): Hierbei handelt es sich um die Umkehrfunktion von isEqual(), bei welcher wir gleichermaßen vorgegangen sind.

length(): Die Methode length() berechnet die Länge des aufrufenden Vektors.
Mathematische Bildungsvorschrift für die Länge eines Vektors der Dimension Zwei:

$$||\vec{v}|| = \sqrt{x^2 + y^2}$$

Hier war darauf zu achten, dass das Quadrat der x- und y-Koordinaten keinen Überlauf verursacht, was durch eine Abfrage und im Fehlerfall geworfene Exception vermieden wird.

normalize(): Auch bei der Normierung des aufrufenden Vektors wird das Quadrat der x- und y-Koordinaten auf vermeintlichen Überlauf geprüft, was durch eine Abfrage und im Fehlerfall geworfene Exception vermieden wird.
Die Normierung wird erreicht, indem der Vektor durch seine Länge geteilt wird:

$$\frac{1}{||\vec{v}||} \cdot \vec{v}$$

Hier bietet es sich an die bereits vorhandenen Funktionenen mult() und length() aufzurufen.

Programmcode zur Verdeutlichung:

```
public void normalize() throws Exception
{
   if(((x*x) >= Double.MAX_VALUE) || ((y*y) >= Double.MAX_VALUE))
   {
      throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
          beachten!");
   }
   mult(1/length());
   // double tmplaenge=Math.sqrt((x*x)+(y*y));
   // div(tmplaenge);
}
```

2.2 Vektor3D

Die Klasse Vektor3D repräsentiert 3D-Vektoren. Der einzige Unterschied zur Klasse Vektor2D liegt bei der Erweiterung um die z-Koordinate. Die Koordinaten werden jeweils durch double-Werte dargestellt. Zunächst besitzt die Klasse drei verschiedene Konstruktoren, durch die es möglich ist auf verschiedene Art und Weisen Objekte der Klasse Vektor3D zu erstellen. Des Weiteren beinhaltet die Klasse die Grundrechenarten Addition, Subtraktion, Multiplikation und Division. Darüber hinaus werden Methoden angeboten, mithilfe derer Vektoren verglichen, normiert und an eine bestimmte Position gesetzt werden können. Außerdem kann die Vektorlänge berechnet und geprüft werden ob es sich um einen Nullvektor handelt. Alle Methoden wurden testgetrieben entwickelt, weshalb sie bestimmte Fehler erkennen und behandeln können. Wenn ein solcher Fehlerfall eintritt, wird eine Exception geworfen um dies kenntlich zu machen. Im Folgenden wird auf die einzelnen Operationen und die eventuell auftretenden Fehler eingegangen.

2.2.1 Methoden

Addition: Bei der Addition bestünde die Möglichkeit, dass es bei zu großen bzw. zu kleinen Zahlen zu einem Überlauf im positiven oder negativen Bereich kommen könnte. Dies lässt sich durch folgende Überlegungen ermitteln:

```
public void add(Vektor3D b) throws Exception
{
   if((x<0 && b.x<0) || (x>0 && b.x>0) || (y<0 && b.y<0) || (y>0 && b.y>0)
       || (z<0 && b.z<0) || (z>0 && b.z>0))
      if((Math.abs(x)+Math.abs(b.x) >= Double.MAX_VALUE) ||
          (Math.abs(y)+Math.abs(b.y) >= Double.MAX_VALUE) ||
          (Math.abs(z)+Math.abs(b.z) >= Double.MAX_VALUE))
```

```
            throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
                beachten!");
```

Es wird also getestet ob die Koordinaten des aufrufenden und des übergebenen Vektors beide größer oder kleiner Null sind. Wenn das der Fall ist, wird aus diesen der Betrag gebildet und dann geprüft ob deren Addition größer/gleich dem maximalen Wert des darstellbaren Bereichs ist. Ist dies der Fall, wird eine Exception geworfen und der Nutzer über den Fehler informiert. (Mit der Bildung des Betrags der Koordinaten genügt der Test auf den Maximalwert von Double.)

```
    if((x>0 && b.x<0) || (x<0 && b.x>0) || (y>0 && b.y<0) || (y<0 && b.y>0)
        || (z>0 && b.z<0) || (z<0 && b.z>0))
       if((Math.abs(x)-Math.abs(b.x) >= Double.MAX_VALUE) ||
           (Math.abs(y)-Math.abs(b.y) >= Double.MAX_VALUE) ||
           (Math.abs(z)-Math.abs(b.z) >= Double.MAX_VALUE))
           throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
               beachten!");
    x+=b.x;
    y+=b.y;
    z+=b.z;
}
```

Des Weiteren bestünde die Möglichkeit einen sehr großen und einen kleinen Wert zu addieren, was ebenfalls zu einem Überlauf führen könnte. Daher wird getestet ob die einzelnen Koordinaten des aufrufenden und des übergebenen größer und/oder kleiner Null sind. Wenn das der Fall ist, wird aus diesen der Betrag gebildet und dann geprüft ob deren Subtraktion größer/gleich dem maximalen Wert des darstellbaren Bereichs ist. Ist dies der Fall, wird eine Exception geworfen und der Nutzer über den Fehler informiert.

Subtraktion: Prinzipiell können hier die gleichen Fehler auftreten wie bei der Addition. Es muss lediglich bei der Prüfung auf die Vorzeichen geachtet werden. Aufgrund des Bildens der Beträge in der Überprüfung auf Überlauf von der Addition können wir die gleiche Überprüfung auch für die Subtraktion verwenden.

Multiplikation: Bei der Multiplikation wird ein Vektor mit einer Doublezahl multipliziert. Dabei sollte auch dort darauf geachtet werden, dass kein Überlauf zustande kommt. Das verhindern wir, indem wir auch hier darauf testen ob die Operation einen größeren Wert erzeugen würde, als durch Double darstellbar. Auch hier werfen wir im Fehlerfall eine Exception und geben eine Fehlermeldung zurück.
Programmcode zur Verdeutlichung:

```
public void mult(double b) throws Exception
{
   if((Math.abs(x)*Math.abs(b) >= Double.MAX_VALUE) ||
       (Math.abs(y)+Math.abs(b) >= Double.MAX_VALUE) ||
       (Math.abs(z)+Math.abs(b) >= Double.MAX_VALUE))
```

```
        throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
            beachten!");
    x*=b;
    y*=b;
    z*=b;
}
```

Division: Bei der Division muss abgesehen von dem Test auf positiven und negativen Überlauf auch getestet werden, ob der Divisor Null ist, da Division durch Null nicht zulässig ist. Wurde dies durch den Nutzer nicht beachtet, wird eine Exception geworfen und darauf hingewiesen. Für den Test auf Überlauf muss überprüft werden ob der Divisor zwischen Null und Eins liegt, da das Ergebnis in dem Fall dann größer ist als der Dividend bzw. im Falle b==1 gleich groß. Also wird auch hier der Betrag der Koordinaten gebildet und die Division auf Überlauf getestet. Im Fehlerfall erfolgt auch hier eine Exception.
Programmcode zur Verdeutlichung:

```
public void div(double b) throws Exception
{
    if(b==0)
        throw new Exception("Division durch 0 ist nicht zulässig!");

    if((b<1 && b > 0) || b==1)
        if((Math.abs(x)/Math.abs(b) >= Double.MAX_VALUE) ||
            (Math.abs(y)/Math.abs(b) >= Double.MAX_VALUE) ||
            (Math.abs(z)/Math.abs(b) >= Double.MAX_VALUE))
           throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
               beachten!");

    x/=b;
    y/=b;
    z/=b;
}
```

setPosition(): Diese Funktion bietet die Möglichkeit einem Vektor durch Übergabe von x-, y- und z-Koordinate eine neue Position zuzuweisen. Eventuell könnten diese übergebenen Werte größer sein als der double-Maximalwert, was wir durch vorheriges Prüfen ausgeschlossen haben.
Programmcode zur Verdeutlichung:

```
public void setPosition(double x1, double y1, double z1) throws Exception
{
    if((Math.abs(x1)>=Double.MAX_VALUE) || (Math.abs(y1)>=Double.MAX_VALUE)
        || (Math.abs(z1)>=Double.MAX_VALUE))
        throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
            beachten!");
    x=x1;
```

```
    y=y1;
    z=z1;
}
```

isNullVector(): In der Methode wird überprüft, ob der aufrufende Vektor ein Nullvektor ist. Dabei war nur die Korrektheit der Funktion zu prüfen.

isEqual(): In isEqual() wird überprüft, ob der aufrufende und der übergebene Vektor den gleichen Inhalt haben. Dabei war nur die Korrektheit der Funktion zu prüfen. (Vergleich Inhalt und nicht die Objektreferenzen)

isNotEqual(): Hierbei handelt es sich um die Umkehrfunktion von isEqual(), bei welcher wir gleichermaßen vorgegangen sind.

length(): Die Methode length() berechnet die Länge des aufrufenden Vektors.
Mathematische Bildungsvorschrift für die Länge eines Vektors der Dimension Zwei:

$$||\vec{v}|| = \sqrt{x^2 + y^2 + z^2}$$

Hier war darauf zu achten, dass das Quadrat der x-, y- und z-Koordinaten keinen Überlauf verursacht, was durch eine Abfrage und im Fehlerfall geworfene Exception vermieden wird.

normalize(): Auch bei der Normierung des aufrufenden Vektors wird das Quadrat der x-, y- und z-Koordinaten auf vermeintlichen Überlauf geprüft, was durch eine Abfrage und im Fehlerfall geworfene Exception vermieden wird.
Die Normierung wird erreicht, indem der Vektor durch seine Länge geteilt wird:

$$\frac{1}{||\vec{v}||} \cdot \vec{v}$$

Hier bietet es sich an die bereits vorhandenen Funktionenen mult() und length() aufzurufen.
Programmcode zur Verdeutlichung:

```
public void normalize() throws Exception
{
   if(((x*x) >= Double.MAX_VALUE) || ((y*y) >= Double.MAX_VALUE) || ((z*z)
       >= Double.MAX_VALUE))
   {
      throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
          beachten!");
   }
   mult(1/length());
   // double tmplaenge=Math.sqrt((x*x)+(y*y)+(z*z));
   // div(tmplaenge);
}
```

2.3 LineareAlgebra

Die Klasse LineareAlgebra stellt verschiedene Methoden für 2D- und 3D-Vektoren bereit. Da diese Methoden alle statisch sind, wird ein privater Konstruktor benötigt um sicher zu stellen, dass kein Objekt von LineareAlgebra angelegt wird.

```
private LineareAlgebra(){}
```

Die verschiedenen Funktionen werden im folgenden näher erläutert.

2.3.1 Methoden

Addition: Bei der Addition bestünde die Möglichkeit, dass es bei zu großen bzw. zu kleinen Zahlen zu einem Überlauf im positiven oder negativen Bereich kommen könnte. Dies lässt sich durch folgende Überlegungen ermitteln:

```
  public void add(Vektor3D b) throws Exception
  {
     if((x<0 && b.x<0) || (x>0 && b.x>0) || (y<0 && b.y<0) || (y>0 && b.y>0)
         || (z<0 && b.z<0) || (z>0 && b.z>0))
        if((Math.abs(x)+Math.abs(b.x) >= Double.MAX_VALUE) ||
            (Math.abs(y)+Math.abs(b.y) >= Double.MAX_VALUE) ||
            (Math.abs(z)+Math.abs(b.z) >= Double.MAX_VALUE))
           throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
              beachten!");
```

Es wird also getestet ob die Koordinaten des aufrufenden und des übergebenen Vektors beide größer oder kleiner Null sind. Wenn das der Fall ist, wird aus diesen der Betrag gebildet und dann geprüft ob deren Addition größer/gleich dem maximalen Wert des darstellbaren Bereichs ist. Ist dies der Fall, wird eine Exception geworfen und der Nutzer über den Fehler informiert. (Mit der Bildung des Betrags der Koordinaten genügt der Test auf den Maximalwert von Double.)

```
     if((x>0 && b.x<0) || (x<0 && b.x>0) || (y>0 && b.y<0) || (y<0 && b.y>0)
         || (z>0 && b.z<0) || (z<0 && b.z>0))
        if((Math.abs(x)-Math.abs(b.x) >= Double.MAX_VALUE) ||
            (Math.abs(y)-Math.abs(b.y) >= Double.MAX_VALUE) ||
            (Math.abs(z)-Math.abs(b.z) >= Double.MAX_VALUE))
           throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
              beachten!");
     x+=b.x;
     y+=b.y;
     z+=b.z;
  }
```

Des Weiteren bestünde die Möglichkeit einen sehr großen und einen kleinen Wert zu addieren, was ebenfalls zu einem Überlauf führen könnte. Daher wird getestet ob die einzelnen Koordinaten des aufrufenden und des übergebenen größer und/oder kleiner

Null sind. Wenn das der Fall ist, wird aus diesen der Betrag gebildet und dann geprüft ob deren Subtraktion größer/gleich dem maximalen Wert des darstellbaren Bereichs ist. Ist dies der Fall, wird eine Exception geworfen und der Nutzer über den Fehler informiert.

Subtraktion: Prinzipiell können hier die gleichen Fehler auftreten wie bei der Addition. Es muss lediglich bei der Prüfung auf die Vorzeichen geachtet werden. Aufgrund des Bildens der Beträge in der Überprüfung auf Überlauf von der Addition können wir die gleiche Überprüfung auch für die Subtraktion verwenden.

Multiplikation: Bei der Multiplikation wird ein Vektor mit einer Doublezahl multipliziert. Dabei sollte auch dort darauf geachtet werden, dass kein Überlauf zustande kommt. Das verhindern wir, indem wir auch hier darauf testen ob die Operation einen größeren Wert erzeugen würde, als durch Double darstellbar. Auch hier werfen wir im Fehlerfall eine Exception und geben eine Fehlermeldung zurück.
Programmcode zur Verdeutlichung:

```
public void mult(double b) throws Exception
{
   if((Math.abs(x)*Math.abs(b) >= Double.MAX_VALUE) ||
       (Math.abs(y)+Math.abs(b) >= Double.MAX_VALUE) ||
       (Math.abs(z)+Math.abs(b) >= Double.MAX_VALUE))
      throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
         beachten!");
   x*=b;
   y*=b;
   z*=b;
}
```

Division: Bei der Division muss abgesehen von dem Test auf positiven und negativen Überlauf auch getestet werden, ob der Divisor Null ist, da Division durch Null nicht zulässig ist. Wurde dies durch den Nutzer nicht beachtet, wird eine Exception geworfen und darauf hingewiesen. Für den Test auf Überlauf muss überprüft werden ob der Divisor zwischen Null und Eins liegt, da das Ergebnis in dem Fall dann größer ist als der Dividend bzw. im Falle b==1 gleich groß. Also wird auch hier der Betrag der Koordinaten gebildet und die Division auf Überlauf getestet. Im Fehlerfall erfolgt auch hier eine Exception.
Programmcode zur Verdeutlichung:

```
public void div(double b) throws Exception
{
   if(b==0)
      throw new Exception("Division durch 0 ist nicht zulässig!");

   if((b<1 && b > 0) || b==1)
      if((Math.abs(x)/Math.abs(b) >= Double.MAX_VALUE) ||
          (Math.abs(y)/Math.abs(b) >= Double.MAX_VALUE) ||
          (Math.abs(z)/Math.abs(b) >= Double.MAX_VALUE))
```

```
            throw new Exception("Ueberlauf! Bitte den maximalen Wertebereich
                beachten!");

        x/=b;
        y/=b;
        z/=b;
    }
```

---

euklidische Distanz für 2D-Vektoren:

$$euklDistance(p_1, p_2) = \sqrt{(p_2.x - p_1.x)^2 + (p_2.y - p_1.y)^2}$$

euklidische Distanz für 3D-Vektoren:

$$euklDistance(p_1, p_2) = \sqrt{(p_2.x - p_1.x)^2 + (p_2.y - p_1.y)^2 + (p_2.z - p_1.z)^2}$$

Manhattan-Distanz für 2D-Vektoren:

$$manhattanDistance(p_1, p_2) = |p_2.x - p_1.x| + |p_2.y - p_1.y|$$

Manhattan-Distanz für 3D-Vektoren:

$$manhattanDistance(p_1, p_2) = |p_2.x - p_1.x| + |p_2.y - p_1.y| + |p_2.z - p_1.z|$$

Skalarprodukt für 2D-Vektoren:

$$\vec{u} \cdot \vec{v} = u_1 \cdot v_1 + u_2 \cdot v_2$$

Skalarprodukt für 3D-Vektoren:

$$\vec{u} \cdot \vec{v} = u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3$$

Kreuzprodukt für 2D-Vektoren:

$$\vec{u} \times \vec{v} = \begin{pmatrix} u_1 v_2 - u_2 v_1 \end{pmatrix}$$

Kreuzprodukt für 3D-Vektoren:

$$\vec{u} \times \vec{v} = \begin{pmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_3 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{pmatrix}$$

Kosinusformel für 2D-/3D-Vektoren:

$$\cos \alpha = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| \cdot |\vec{v}|}$$

3 Testverfahren

Für die testgetriebene Entwicklung wurde JUnit verwendet. Bei den Tests haben wir vor allem auf Überlauf getestet, da dies bei den meisten der Methoden eine mögliche Fehlerquelle darstellt. Dazu wurden Rechnungen mit dem maximalen und minimalen Zahlenwerten des Typs Double durchgeführt, um sicher zu stellen, dass die Methoden mit Überlauf korrekt umgeht.
Bei der Überprüfung auf Korrektheit der Rechnungen wurde mit verschiedenen Zahlenwerten im Doublebereich gearbeitet um zu gewährleisten, dass die Methoden mit verschiedenen Werten fehlerfrei arbeiten.