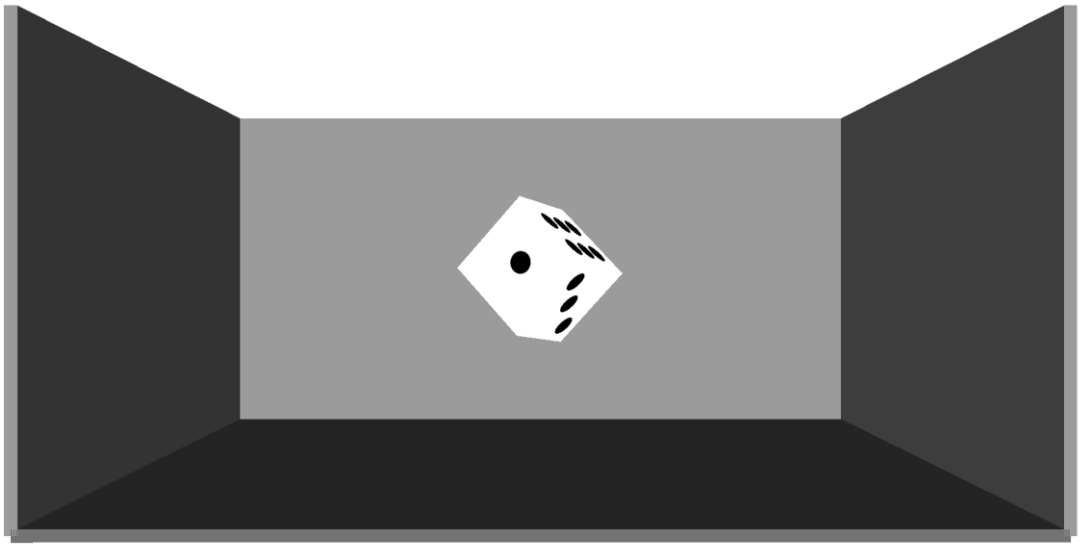
Entwicklung einer gespaltenen Würfelklasse mit 3D-Animation in Java und einer Beispielanwendung

Informatik



Fachlehrkraft: Herr Fassbender

Informatik Leistungskurs IF7

Jahrgangsstufe 11

2018

Julius Klodt

Inhaltsverzeichnis

1. Einleitung

Wenn man nach langem Programmieren ein Projekt endlich fertiggestellt hat, ist man sehr gespannt, sein Endresultat zu sehen. Jedoch kommt es häufig vor, dass man nur eine Zahl in der Konsole sieht und gar keinen richtigen visuellen Output hat. Daher ist mir die Idee gekommen, mich näher mit 3D-Animationen zu beschäftigen. Mit einem visuellen Bild einer 3-Dimensionalen Figur oder Landschaft kann man sein Ergebnis viel besser darstellen. Also habe ich beschlossen, meine Facharbeit über 3-Dimensionale Animationen in Java zu schreiben. Weil ich mein gesammeltes Wissen in anderen Projekten nutzen zu möchte, ist es mein Ziel, meine eigene Bibliothek zu schreiben, sodass ich diese nur noch importieren muss und es möglich wird, durch wenige Zeilen Quelltext, dieses Wissen Abzurufen. Um seine eigenen 3-Dimensionalen Animationen vielseitig und individuell zu gestalten ist es gut, wenn die Bibliothek modular aufgebaut ist. Als Beispiel werde ich bei erfolgreicher Durchführung eine sich rotierenden Würfel zeigen, welcher eine zufällige Zahl zwischen eins und sechs ausgibt. Hier mit soll ein normaler Spielwürfel, welchen man bei vielen Brettspielen findet, simuliert werden. Diesen kann man zur Veranschaulichung einer zufällig generierten Zahl oder eines Würfelspiels nutzen. Die Möglichkeiten, die sich durch eine modular aufgebaute Animations-Klasse ergeben, sind immens. Gerade im Bereich der Spiele und der Simulationen kann diese Bibliothek helfen, neue Türen zu öffnen. Auch Datenstrukturen wie einen Graph oder einen Binärbaum kann man visualisieren und somit veranschaulichen.

2. Grundlegende Informationen

2.1. Die Programmiersprache

Aufgrund der intensiven Behandlung der Programmiersprache Java in der Oberstufe und in meiner Freizeit, habe ich mich dazu entschieden, diese für mein weiteres Vorgehen zu nutzen. Java ist eine objektorientierte Programmiersprache, welche von Sun Microsystems entwickelt und im Jahr 1995 veröffentlicht wurde [11].

2.2. Java3D

Zur Visualisierung von 3D-Animationen habe ich Java3D gewählt. Java3D ist eine Klassenbibliothek für dreidimensionale Darstellungen in Java Anwendungen. Sie wurde von Sun Mircosystems entwickelt und im Dezember 1998 veröffentlicht [10]. Basierend auf einem Szenengraphen werden alle Objekte in einem rechtshändigen Koordinatensystem dargestellt. Ein Szenengraph speichert alle Geometriedaten und Transformationen in einer baumähnlichen Struktur.

2.3. Die Entwicklungsumgebung

Als Entwicklungsumgebung habe ich NetBeans IDE gewählt. NetBeans IDE ist ein Produkt eines Open Source Projektes, welches von Sun Mircosystems im Juni 2000 entwickelt wurde [3]. Man kann mit NetBeans IDE in mehreren Programmiersprachen programmieren, jedoch wird NetBeans IDE hauptsächlich für Java verwendet. Dementsprechend ist es darauf spezialisiert. Für mich ist NetBeans IDE in Frage gekommen, da es eine freundliche Benutzeroberfläche hat und Fehler sowie unnötige Quelltextelemente beim Programmieren erkennt. Deshalb ist es sehr angenehm, in NetBeans IDE zu programmieren

3. Implementation

Das Projekt ist in 4 verschiedene Klassen gegliedert. Die "Window3D" Klasse ist die Hauptklasse. Hier wird die Verbindung zwischen der zweidimensionalen und der dreidimensionalen Welt hergestellt. Es werden alle Objekte in dieser Klasse gespeichert und der graphischen Oberfläche hinzugefügt.
Die optional einstellbare Steuerung ist hier ebenfalls eingebunden.
Die Klasse "Box3D" erstellt einen Quader an einer bestimmten Position mit einer bestimmten Größe und Farbe.
Da ein Würfel jedoch 6 verschiedene Seiten hat, musste ich die Klasse "ComplexBox3D" erstellen. Diese ermöglicht es, einen Quader mit 6 verschiedenen Seiten, Farben und Texturen zu erstellen. Ich habe gewollt diese Klasse nicht als den Würfel bezeichnet, da ich mein Projekt möglichst Modular aufbauen wollte. Somit kann ich in anderen Projekten auch Quader mit verschiedenen Seiten nutzen.
Die letzte Klasse ist der letztendliche Würfel. Die Klasse "Würfel3D" vereinfacht die Erstellung eines Würfels, da die Textuierung nicht manuell durchgeführt werden muss. Des Weiteren sind einige spezielle Funktionen für den Würfel in dieser Klasse vorhanden. Darauf gehe ich später näher ein.

3.1. Window3D

Grundsätzlich ist die Window3D Klassse die Hauptklasse. Hier werden andere Objekte eingebunden und die optionale Steuerung sowie die graphische Ausgabe befinden sich hier.

Die Window3D Klasse hat zwei verschiedene Konstruktoren, eine Initialisierungsmethode, eine Methode zur Erstellung der Szene, Methoden für die Steuerung, die später näher erläutert werden und eine getter-Methode der graphischen Ausgabe.

Dem ersten Konstruktor (Zeile 67-69) werden die x und die y Größe des Fensters, sowie die Hintergrundfarbe in RGB Werten übergeben. Darauffolgend wird in Zeile 68 die Initialisierungsmethode init(...) mit den übergebenen Parametern aufgerufen. Hierzu später mehr. Im Gegensatz zum ersten Konstruktor wird in dem zweiten Konstruktor

keine Hintergrundfarbe definiert. Hier wird anstelle der RGB-Werte ein String übergeben. In diesem String wird der Dateipfad eines Hintergrundbildes übergeben. Hierbei ist es wichtig, dass die x und die y Größe des angegebenen Bildes immer eine Potenz von zwei ist. Also zum Beispiel 512*1024 oder 2048*2048. Dieser zweite Konstruktor ruft ebenfalls die Initialisierungsmethode init(...) mit den übergebenen Parametern auf. Bei dem ersten Konstruktor wird bei dieser Methode der vierte Parameter als leerer String deklariert. Im Vergleich dazu deklariert der zweite Konstruktor diesen Parameter als den davor übergebenen String, die URL. Dafür übergibt der zweite Konstruktor den zweiten, dritten und vierten Parameter als null. Der erste und die beiden letzten Parameter sind bei beiden gleich. Die letzten beiden übergeben die x und die y Größe des Fensters. Der erste Parameter sagt aus, dass das Programm ein eigenständiges Fenster erstellen soll. Das heißt, dass wenn man ein Objekt der Klasse Window3D erstellt, dass ein eigenes Fenster erstellt werden soll. Wenn dieser Parameter als false deklariert wird, wird kein eigenständiges Fenster erstellt. Somit könnte man das Objekt der Klasse Window3D in einem bereits vorhandenen Fenster mit einbeziehen.

```
67      public Window3D(float r, float g, float b, int x, int y) {
68          init(true, r, g, b, "", x, y);
69      }
78      public Window3D(String url, int x, int y) {
79          init(true, 0, 0, 0, url, x, y);
80      }
```

Abbildung 2, Die Konstruktoren der Klasse Window3D

Um festzulegen was man letztendlich sieht, müssen zwei Punkte immer bestimmt sein. Der eine Punkt umschreibt die Position des Betrachters. Das heißt von wo man auf etwas guckt. Der andere Punkt umschreibt den Punkt wo man hinguckt.

In der Methode init(...) werden zuerst vier Variablen festgelegt. radius legt den Radius des Punktes fest, welchen man als Betrachter anguckt in Relation zu dem Punkt wo man sich momentan befindet. Dieser ist wichtig für die Steuerung, welche später näher erörtert wird. winkel gibt den aktuellen Winkel im Bogenmaß an. Dies ist notwendig, da man den Punkt wo man hinguckt immer neu berechnen muss und dieser sich gleichmäßig bewegen soll. Deswegen wird er anfangs als 3.141, der Kreiszahl pi, festgelegt. Somit ist es gewährleistet, dass man am Anfang geradeaus guckt. Nach

diesen beiden Schritten müssen die x und z Koordinaten für die Blickrichtung festgelegt werden. Danach werden die sogenannten GraphicsConfiguration definiert.

```
121         GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
```

Abbildung 3, Erstellung der GraphicsConfiguration

Dies ist zwingend notwendig, da somit die Eigenschaften des Ausgabegeräts, dem Monitor, festgelegt werden [2]. Nachfolgend werden diese benutzt, um einen Canvas3D zu erschaffen, welcher eine Verbindung zwischen der zwei- und der dreidimensionalen Welt herstellt. Dieses Canvas3D Objekt, hier canvas, kann die später erstellte 3D Animation in einer zweidimensionalen GUI (Graphical User Interface) darstellen [2] und wird somit dem Fenster, welches wir zuvor mit Hilfe der davor übergebenen x und y Größe des Fensters erstellt haben, hinzugefügt. Die vorherige if-Abfrage guckt, ob das Fenster eigenständig läuft oder es in einer anderen GUI mit eingebunden wird.

```
132         univers = new SimpleUniverse(canvas);
```

Abbildung 4, Erstellung des SimpleUniverse

In Zeile 132 wird ein sogenanntes SimpleUniverse mit dem Canvas3D Objekt erstellt. Das SimpleUniverse ist die oberste Struktur Ebene der dreidimensionalen Welt und alle Objekte müssen sich zwangsweise hier drin befinden [2]. Durch die Einbindung des Canvas3D Objekt canvas wird die dreidimensionale Welt, das SimpleUniverse, mit der zweidimensionalen Ausgabe, dem Canvas3D Objekt, miteinander verbunden. Nach der Erstellung eines SimpleUniverse muss dieses auch mit einem Inhalt gefüllt werden. Um dies zu tun, wird ein Objekt der Klasse BranchGroup erstellt. Dies ist notwendig da somit mehrere Unterobjekte zu einer Szene hinzugefügt werden können. Diese Szene wird mit der Methode createBranchGraph(...) erzeugt, welche die fertige Szene als BranchGroup zurückgibt. Dieser Methode werden des Weiteren die RGB-Werte der Hintergrundfarbe, bzw. der Dateipfad des Hintergrundbildes mitgegeben. Der Aufbau der Szene wird im Folgenden näher erörtert.

Nach der Erstellung der Szene wird diese dem universe hinzugefügt.

Damit man diese Szene in dem SimpleUniverse überhaupt betrachten kann, muss ein Objekt der Klasse ViewingPlatform erstellt werden. Dies ist notwendig um ein Objekt der Klasse View, welches die letztendliche dreidimensionale Welt in eine zweidimensionale Ausgabe rendert, zu konfigurieren [2]. Durch die getter-Methode getViewingPlatform() des Objekts der Klasse SimpleUniverse, erhält man die gewünschte ViewingPlatform des SimpleUniverse welches wir vorher erstellt haben.

```
148         view = univers.getViewer().getView();
```

Abbildung 5, Konfiguration des Objekts der Klasse View

Am Ende der Initialisierungsmethode wird in Zeile 141 das eben angesprochene Objekt der Klasse View mit dem View der ViewingPlatform des zu verbindenden SimpleUniverse verbunden.

Nun erörtere ich die createBranchGraph(...) Methode. Diese ist für die Erstellung der Hauptszene und den Hintergrund zuständig. Die Methode wird mit vier Parametern aufgerufen. Den RGB-Werten der Hintergrundfarbe und den Dateipfad des Hintergrundbildes. Um am Ende der Methode eine fertige Szene als BranchGroup zurückzugeben, wird in eine leere BranchGroup erstellt. Hier werden nun vorerst alle anderen Unterobjekte hinein geladen.

Damit man in der Animation etwas sehen kann, muss für Licht gesorgt werden. Als generelles Hauptlicht oder Umgebungslicht wird ein Objekt der Klasse AmbientLight mit einer bestimmten Farbe erstellt. Die Eigenschaft des AmbientLights ist, dass es aus allen Richtungen kommt und somit gleichmäßig erscheint [2]. Da jede Lichtquelle eine bestimmte Reichweite hat, muss diese auch festgelegt werden. Zur Begrenzung des Lichts wird ein Objekt der Klasse BoundingSphere erstellt . Dieses definiert einen bestimmten Bereich in Form einer Kugel und wird daher mit einem Mittelpunkt und einem Radius erstellt [2]. Darauffolgend wird der Bereich, welcher von der Lichtquelle beeinflusst wird, mit der Lichtquelle selbst verknüpft. Nun wird das Umgebungslicht mit Hilfe der Methode addChild(...), der am Anfang erstellten BranchGroup hinzugefügt.

Um ein gezieltes Licht wie die Sonne darzustellen wird ein Objekt der Klasse DirectionalLight erstellt [2]. Dieses wird ebenfalls mit einer Farbe erstellt. Jedoch wird

im Gegensatz zum AmbientLight auch noch ein Vector beim Erstellen hinzugefügt. Dieser bestimmt die Richtung in welche das Licht strahlen soll. Wie beim AmbientLight muss auch beim DirectionalLight ein Bereich, welcher beeinflusst wird, definiert und dem Licht hinzugefügt werden. Zum Schluss wird das Licht der BranchGroup hinzugefügt werden.

Da momentan noch keine sichtbaren Objekte in der dreidimensionalen Welt vorhanden sind, muss die Klasse Window3D gewährleisten, dass Objekte nachträglich hinzugefügt werden können. Diese Aufgabe übernimmt die Methode addObject(TransformGroup tmp). Da einem SimpleUniverse Objekt nur Objekte der Klasse BranchGroup hinzugefügt werden können, muss das übergebene TransformGroup Objekt in ein BranchGroup Objekt eingefügt werden. Dieses wird mit einem Transform3D Objekt in der Größe um das 3,5 Fache herunter skaliert. Dadurch werden alle Objekte im Raum kleiner und somit wird eine größere Welt suggeriert. Schließlich wird das fertige Objekt der Klasse BranchGroup dem SimpleUniverse Objekt hinzugefügt.

3.1.1. Die optionale Steuerung

Zur Bewegung in der dreidimensionalen Welt ist eine Tasten-Steuerung vorhanden. Diese ist optional einschaltbar und grundsätzlich nicht aktiv. Die Hilfsmethode setKeyControl(boolean) kann die Steuerung aktivieren und deaktivieren.

Grundsätzlich wird ein KeyListener für die Steuerung verwendet und am Anfang der Klasse direkt mit integriert. In der Initialisierungsmethode wird dieser dann dem Fenster hinzugefügt. Wenn ein KeyListener verwendet wird, müssen die drei Methoden keyPressed(...), keyReleased(...) und keyTyped(...) überschrieben werden. Die Methoden keyReleased(...) und keyTyped(...) werden jedoch bei meiner Steuerung nicht verwendet und sind damit nicht wichtig. Die Methode keyPressed(...) ist dafür zuständig, keyEvents beim Drücken einer Taste der Tastatur zu verarbeiten. Die dort vorhandene switch-Anweisung übernimmt dies. Das keyEvent wird mit den Identifikatoren der Tasten verglichen und der dazugehörige Befehl wird ausgeführt. Hierbei wird in zwei verschiedene Aufgaben unterschieden. Einmal wird die Position des Betrachters geändert und das andere Mal die Position des zu betrachtenden Punkts. Diese Aufgaben übernehmen die beiden Methoden setFocusPoint(...) und setCamPoint(...).

Bevor ich näher auf diese beiden Methoden eingehe, erkläre ich, wie überhaupt die Betrachtung der Szene eingestellt und geändert wird.

Um richtig navigieren zu können, müssen zwei Punkte gespeichert werden. Die Variablen currentX, currentY und currentZ speichern die Position des Betrachters. currentXView, currentYView und currentZView die des zu betrachtenden Punkts. Um diesen immer wieder neu berechnen zu können muss eine weitere Variable gespeichert werden. Die Variable winkel speichert den Bogenmaß des zweidimensionalen Sehwinkels.

Damit man jedoch die Betrachtung der Szene überhaupt ändern kann, muss in der Initialisierungsmethode der Klasse Window3D, die vorhin angesprochene ViewingPlatform etwas modifiziert werden. Dafür wird diese in ein Objekt der Klasse TransformGroup geladen. Die besondere Eigenschaft der TransformGroup ist es, die Position der darin enthaltenen Objekte während der Laufzeit zu ändern [7]. Um dies jedoch nutzen zu können, wird zusätzlich ein Objekt der Klasse Transform3D benötigt. Dieses führt Translationen, Rotationen und andere Skalierungen durch [6].

Dieses benötigt Eigenschaften des davor erstellten Objekts der Klasse TransformGroup, welche in das Tranform3D Objekt hineinkopiert werden. Nun können die Anfangswerte eingebunden werden. Damit die Anfangsposition für den Betrachter etwas nach vorne rutscht, wird currentZ auf 5 gesetzt. Danach wird die Betrachtung der dreidimensionalen Welt geändert. Dafür wird die Position des Betrachters, die Position des zu betrachtenden Punkts und ein Vektor genutzt. Nach diesem Schritt muss die neue Betrachtung anstelle der alten gezeigt werden. In Zeile 137 wird dieser Vorgang mit der invert() Methode der Klasse Transform3D erledigt. Abschließend wird die neue Transformation in das TransformGroup Objekt hineingeladen (Zeile 138).

Die letzten drei genannten Schritte werden öfters verwendet und nur noch als Aktualisierung der Betrachtung benannt.

```
143         currentZ = 5;
144         t3d.lookAt(new Point3d(currentX, currentY, currentZ),
145                 new Point3d(currentXView, currentYView, currentZView), new Vector3d(0,1,0));
146         t3d.invert();
147         steerTG.setTransform(t3d);
```

Abbildung 6, Aktualisierung der Betrachtung

Nun erkläre ich die Methoden setCamPoint(...) und setFocusPoint(...).

Die Methode setCamPoint(float länge, int bestimmer) wird mit 2 verschiedenen Parametern aufgerufen. Die Variable bestimmer ist dafür da, verschiedene Fälle zu unterscheiden. Die Variable länge gibt an wie groß die spätere Änderung ist. In der Methode wird unterschieden ob man sich hoch, runter oder in der Ebene bewegt. Bei einer Bewegung nach oben oder unten werden die Werte currentY und currentYView mit der Variable länge addiert oder subtrahiert. Bei einer Bewegung in der momentanen Ebene wird die Hilfsmethode berechneRichtung(float länge, int bestimmer) mit den davor übergebenen Parametern aufgerufen. Diese ist dafür da, dass man sich, egal wo man gerade hin guckt, in die richtige Richtung bewegt. Somit läuft man, wenn man geradeaus laufen will, nicht stur an der z-Achse nach vorne, sondern man geht in die Richtung in die man guckt.

Nachdem alle Punkte neu berechnet worden sind, wird die Betrachtung aktualisiert.

Die Parameter der Methode setFocusPoint(float x, float y) zeigen an, wie groß eine Änderung der Sicht in der Horizontalen oder in der Vertikalen ist. Wenn x einen Wert hat außer null, wird die Variable winkel mit x addiert und currentXView sowie currentZView werden neu bestimmt. Diese werden als den Sinus, bzw. den Kosinus von der Variable winkel multipliziert mit dem radius definiert. Diese Berechnung gewährleistet eine gleichmäßige Bewegung in der Horizontalen.

Wenn y einen Wert außer null hat, wird die Sicht in der Vertikalen geändert. Also man guckt weiter nach oben oder nach unten. Dafür wir die Variable currentYView als die Summe von currentYView und y gesetzt. Das birgt den großen Nachteil, dass man keine gleichmäßige Bewegung hat und man irgendwann nicht mehr weiter nach oben oder unten gucken kann. Hier könnte man das Projekt erweitern und die Steuerung verbessern, jedoch war mir dies nicht wichtig genug, um mehr Zeit hinein zu investieren.

Am Ende der Methode wird die Betrachtung wieder aktualisiert.

3.2. Box3D

Die Klasse Box3D ist dafür da, einen Quader durch einfache Befehle zu erschaffen und dessen Eigenschaften zu ändern.

Es gibt zwei verschiedene Konstruktoren, welche fast identisch sind. Der einzige Unterschied ist, dass bei dem einen der Quader eine Farbe bekommt und bei dem anderen eine Textur. Beide werden ansonsten mit sechs Parametern aufgerufen. Diese umschreiben wo das Objekt sein soll und welche Größe es hat.

Am Anfang der beiden Konstruktoren werden Objekte der Klassen TransformGroup, Tranform3D, Vector3f und AxisAngle4f erstellt. AxisAngle4f beschreibt, um wie viel Grad sich ein Objekt um einen Vektor, bzw. um eine Rotationsachse gedreht hat. Es wird eine waagerechte Rotationsachse ohne einer Drehung erstellt. Vector3f stellt einen einfachen Vektor im dreidimensionalen Raum dar.

Im nächsten Schritt wird der Vektor festgelegt. Hierzu werden die x, y und z Koordinaten des Quaders genommen. Dann wird die Translation des Transform3D Objekts diesem Vektor gleichgesetzt. Darauffolgend wird dieses Objekt mit dem davor erstellten Objekt der Klasse TransformGroup verknüpft, sodass die TransformGroup immer die gleichen Eigenschaften in Bezug auf Rotation, Geometrie und Position im Raum wie das Transform3D Objekt hat.

```
49          appearance = new Appearance();
50          appearance.setCapability(Appearance.ALLOW_MATERIAL_WRITE);
51          appearance.setCapability(Appearance.ALLOW_TEXTURE_WRITE);
```

Abbildung 7, Appearance und ihre Berechtigungen

In Zeile 49 wird ein Objekt der Klasse Appearance erstellt. Dieses beschreibt die Eigenschaften wie ein Objekt dargestellt und gerendert werden soll. Mit Hilfe dieses Objektes werden Farben, Texturen und Spiegelungen gezeigt [9]. Damit Materialien und Texturen angezeigt werden können, muss zuvor das Objekt der Klasse Appearance die Berechtigung dafür erhalten. Dies geschieht in den Zeilen 50 und 51.

Nun kommt der wesentliche Unterschied. Im ersten Konstruktor wird in den Zeilen 53-55 die Textur generiert und dem Appearance Objekt hinzugefügt. Im zweiten

Konstruktor werden diese drei Zeilen ersetzt. Hier wird eine Farbe und ein Material erstellt und dem Appearance Objekt hinzugefügt (Zeile 87-90).

```
53          TextureLoader texLoader = new TextureLoader(url, null);
54          Texture2D tex = (Texture2D) texLoader.getTexture();
55          appearance.setTexture(tex);
```

Abbildung 8, Erstellung der Textur

Die Textur wird mit Hilfe eines Objekts der Klasse TextureLoader aus einem Bild, welches wir anhand des im Konstruktor übergebenen Dateipfades angeben, geladen [5] (Zeile 53). Die zweidimensionale sichtbare Textur wird mit einem Textur2D Objekt erstellt [4] . Dieses übernimmt die Textur des TextureLoader Objekts (Zeile 54). Dann wird die Textur noch der erstellten Appearance hinzugefügt (Zeile 55).

```
87          ob = new Color3f(r, g, b);
88          Material material = new Material(ob, new Color3f(0.0f, 0.0f, 0.0f),
89                                           ob, new Color3f(1.0f, 1.0f, 1.0f), 64);
90          appearance.setMaterial(material);
```

Abbildung 9, Erstellung der Farbe

Im anderen Konstruktor wird zuerst die Farbe erstellt, welche wir als RGB-Werte dem Konstruktor übergeben haben (Zeile 87). Danach wird ein Objekt der Klasse Material erstellt. Dieses ist dafür da, die Eigenschaften einer Oberfläche in Bezug auf Licht einzustellen [8]. Insgesamt werden bei der Erstellung fünf Parameter übergeben. Am Anfang wird die Umgebungsfarbe des Materials als die davor erstellte Farbe definiert. Die danach erstellte Farbe schwarz beschreibt die Farbe, die von dem Material ausgestrahlt wird. Danach wird die am Anfang erstellte Farbe wieder für die Farbe des Materials bei Beleuchtung einer Lichtquelle verwendet. Die nach diesem Schritt erstellte Farbe gibt nun die Farbe an, welche als Reflektion erscheinen soll. Der letzte ist ein Indikator von 1 bis 128, welcher den Grad des Glanzes wiedergibt, wobei 128 sehr stark glänzend und 1 sehr schwach glänzend ist. Dies waren die einzigen Unterschiede der Konstruktoren.

```
92          box = new Box(xSize, ySize, zSize, Primitive.GENERATE_NORMALS
93                  + Primitive.GENERATE_TEXTURE_COORDS, appearance);
```

Abbildung 10, Erstellung des Quaders

In Zeile 92 wird der eigentliche Quader erstellt. Diesem werden die x-, y- und z-Größe und das modellierte Appearance Objekt übergeben. Des Weiteren wird mit den Identifikatoren "Primitive.GENERATE_NORMALS + Primitive.GENERATE_TEXTURE_COORDS" ein Integer erzeugt, welcher die Textuierung des Objekts erlaubt und zwingend notwendig ist.

Zum Schluss wird die erstellte Box dem anfänglichen TransformGroup Objekt hinzugefügt und die TranformGroup erhält die Berechtigung, dass das Objekt während der Laufzeit transformiert, also gedreht oder bewegt werden kann.

Das Hinzufügen des Quaders in die dreidimensionale Welt wird mit der getter-Methode getObject() ermöglicht. Diese gibt das TransformGroup Objekt zurück, welches den Quader und alle dazugehörigen Eigenschaften speichert.

Für die Veränderung von Eigenschaften während der Laufzeit, gibt es weitere Methoden, welche ich nun näher erläutere.

Die Methode setAxis(float x, float y, float z, float angle) ändert die Rotationsachse. Dafür werden die Werte des AxisAngle4f Objekts mit den übergebenen Parametern überschrieben. Dann wird das TransformGroup Objekt aktualisiert, indem zuerst die momentanen Eigenschaften in das Transform3D Objekt geladen werden (Zeile 141). Darauffolgend wird die Rotation des Objekts dem geänderten AxisAngle4f Objekt gleichgesetzt (Zeile 142) und die neue Transformation wird in das TransformGroup Objekt geladen (Zeile 143). Dieser Vorgang wird häufiger genutzt um die Position und Drehung eines Objekts zu aktualisieren.

```
141          boxGroup.getTransform(tmpTrans);
142          tmpTrans.setRotation(rotationAxis);
143          boxGroup.setTransform(tmpTrans);
```

Abbildung 11, Aktualisierung der Position und der Drehung

Zur Änderung des Drehwinkels, also der Rotation des Quaders entlang der Rotationsachse, gibt es die Methode setRotation(int rotation). Diese ändert den angle-Wert des AxisAngle4f Objekts in den übergebenen rotation Integer. Nun muss wieder die TransformGroup aktualisiert werden.

Die Methoden setTexture(...) und setColor(...) sind dafür da, die Textur und die Farbe während der Laufzeit zu ändern. Beide gehen wie im Konstruktor vor, um die Textur oder die Farbe zu erstellen.

3.3. ComplexBox3D

Bei der Erstellung eines Würfels ist eine Eigenschaft vorhanden, welches nicht mit einem normalen Objekt der Klasse Box3D modelliert werden kann. Ein Würfel hat sechs verschiedene Seiten und dementsprechend muss er auch sechs verschiedene Texturen besitzen. Ein normales Box3D Objekt kann nur eine einzige Textur besitzen. Deswegen bin ich auf die Idee gekommen, eine Klasse zu schreiben welche aus sechs verschiedenen Box3D Objekten einen Quader suggeriert und somit eine individuelle Textuierung gewährleistet.

In dieser Klasse ist nur ein Konstruktor vorhanden. Diesem werden genau die gleichen Werte übergeben wie dem zweiten Konstruktor der Box3D Klasse, welcher dem Quader direkt eine Farbe zuordnet.

Genau wie bei der Box3D Klasse werden anfangs Objekte der Klassen TransformGroup, Transform3D, Vector3f und AxisAngle4f erstellt. Danach wird ein Array von sechs Box3D Objekten angelegt. Nun wird die Position der TransformGroup gesetzt und diese wird durch das Transform3D und dem Vector3f Objekt in die TransformGroup geladen. Diese erhält darauffolgend die Berechtigung während der Laufzeit Transformationen und somit Bewegungen durchzuführen.

Nun werden die sechs verschiedenen Box3D Objekte erstellt, welche zusammen einen Quader darstellen.

Die ersten beiden Box3D Objekte haben die gleichen x- und z-Größen, welche im Konstruktor übergeben worden sind, sowie die y-Größe null. Dadurch sind nun zwei "Platten" vorhanden. Bei der Positionierung werden alle Werte bei null belassen, bis auf die y-Koordinate. Diese wird bei dem Einen auf die positive y-Größe gesetzt und bei dem Anderen auf die negative y-Größe. Dadurch sind diese beiden "Platten" parallel zueinander und genauso positioniert, dass sie bei den anderen Platten nahtlos ansetzten. Des Weiteren werden die RGB-Werte, welche im Konstruktor übergeben worden sind, dem Box3D Objekt übergeben.

Die nächsten beiden Box3D Objekte werden fast genauso erstellt. Nur wird jetzt nicht die y-Größe auf null gesetzt, sondern die x-Größe. Die y-Größe wird gleich der im Konstruktor übergebenen y-Größe definiert. Das gleiche Prozedere passiert bei der Positionierung. Hier bleiben die y- und z-Werte bei null und die x-Koordinate wird einmal gleich der positiven x-Größe gesetzt und einmal gleich der negativen x-Größe.

Die letzten beiden Objekte der Klasse Box3D werden nach dem gleichen Prinzip erstellt, allerdings wird jetzt die z-Größe nur in der Positionierung verwendet.

Am Ende des Konstruktors werden alle Objekte der Klasse Box3D mit Hilfe einer for each-Schleife dem TransformGroup Objekt hinzugefügt.

Die getter-Methode getObject(), ermöglicht es durch die Rückgabe des TransformGroup Objekts mit allen Objekten und Eigenschaften, den erstellten Quader in die dreidimensionale Welt einzubinden.

Genau wie in der Box3D Klasse, sind auch hier die Methoden setAxis(...) und setRotation(...) vorhanden. Diese machen exakt das gleiche wie in der Box3D Klasse.

Der wesentliche Unterschied zur Box3D Klasse ist die Methode severalTextureMapping(...). Mit Hilfe dieser Methode kann jede Seite ihre eigene Textur bekommen. Beim Methodenaufruf werden sechs Parameter übergeben. Dies sind sechs Strings, welche den Dateipfad jeder einzelnen Textur der verschiedenen Seiten speichern. In der Methode selbst, wird jedes Box3D Objekt, mit der Methode setTexture(...) der Klasse Box3D individuell textuiert.

3.4. Würfel3D

Die Klasse Würfel3D ist ein Spezialfall der Klasse ComplexBox3D. Sie ist dafür da, vereinfacht einen Spielwürfel darzustellen.

Es gibt einen Konstruktor mit vier Parametern. Der erste bestimmt die Größe des Würfels. Da alle Seiten gleich groß sind wird nur ein Parameter zur Bestimmung benötigt. Die anderen drei sind zur Positionierung im Raum zuständig. Im Konstruktor wird lediglich ein Objekt der Klasse ComplexBox3D erstellt mit 3 gleich großen Seiten, den Koordinaten im Raum und keiner definierten Farbe. Darauffolgend wird die Textuierung automatisch übernommen und die sechs verschiedenen Seiten eines Würfels werden als Bilder hochgeladen.

Wie auch in den anderen Klassen, muss hier ein Objekt der Klasse TransformGroup in einer getter-Methode zurückgegeben werden. Dies geschieht in der Methode getObject(). Hier wird die TransformGroup des am Anfang erstellten ComplexBox3D zurückgegeben.

Ein Würfel hat einen bestimmten Zweck. Man wirft ihn und erhält somit eine scheinbar zufällige Zahl. Dies soll in der Methode drehen(int v) simuliert werden, wobei v die Geschwindigkeit des Drehens des Würfels definiert.

Hierfür wird ein neuer Thread erstellt. Dieser speichert drei zufällig generierte Variablen und zusätzliche eine mit einer zufälligen Zahl zwischen 5000 und 15000 welche anschließend durch v geteilt wird und der Bezeichnung time. Dieser Integer gibt an wie lange sich der Würfel in Millisekunden drehen soll. Man teilt den Wert noch durch v, da v die wiederholende Zeit darstellt, in welcher der Thread pausiert. Einen x-, y- und z-Wert zwischen -25 und 25. Die drei Variablen stellen die Rotationsachse als Vektor da. In einer immer weiter laufenden while-Schleife werden diese drei Werte immer weiter mit dem Sinus oder dem Kosinus des aktuellen Wertes addiert, sodass die Achse sich gleichmäßig dreht und der Würfelwurf simuliert werden kann. Nachdem der Vektor der Rotationsachse geändert wurde, wird mit Hilfe der Methode setAxis(...) der Klasse ComplexBox3D die Achse verändert. Dabei werden die drei Werte x, y und z sowie die Variable angle, welche die Rotation entlang der Rotationsachse umschreibt, übergeben. Darauffolgend wird der der angle um eins erhöht, die Rotation des ComplexBox3D Objekts wird aktualisiert und der Thread wird für v Millisekunden

pausiert. Die letzten drei Schritte werden fünf mal mit Hilfe einer for-Schleife wiederholt.

Bevor die while-Schleife sich wiederholt, wird mit einer if-Abfrage geguckt, ob die am Anfang definierte Zeit, wie lange sich der Würfel drehen soll, schon abgelaufen ist. Dafür wird verglichen, ob der Wert von angle größer gleich dem von time ist. Wenn dies der Fall ist, wird angle mit einer zufälligen Zahl zwischen 0 und 360 addiert. Dann wird angle gleich dem Rest von angle geteilt durch 360 gesetzt. Nun wird in einer if-else-Struktur geguckt, ob angle kleiner als 60, 120, 180, 240, 300 oder 360 ist. Damit wird die gewürfelte Zahl definiert. Beim Eintreten eines solchen Falles, wird der Würfel neu textuiert und die gewürfelte Seite wird mit einer roten Seite versehen. Danach wird der Würfel so gedreht, dass die gewürfelte Seite nach vorne guckt und es wird in der Konsole die gewürfelte Zahl ausgegeben.

Zum Schluss wird die while-Schleife unterbrochen und die Methode drehen(...) ist somit beendet.

4. Beispiel anhand einer Anwendung

Um nun die Bibliothek zu nutzen, muss diese normalerweise zu der Bibliothek der aktuellen Umgebungsoberfläche hinzugefügt werden. Da ich jedoch noch in meinem Projekt selbst implementiere, kann ich diesen Punkt vernachlässigen.

```
6   public class Würfel {
7
8       private Window3D window;
9
10      public Würfel() {
11          window = new Window3D(0.6f,1.0f,1.0f, 1900, 980);
12          window.setKeyControl(true);
13          Box3D b1 = new Box3D(0.1f,4,8,8,0,0,0.5f,0.5f,0.5f);
14          window.addObject(b1.getObject());
15          Box3D b2 = new Box3D(0.1f,4,8,-8,0,0,0.5f,0.5f,0.5f);
16          window.addObject(b2.getObject());
17          Box3D b3 = new Box3D(8,0.1f,8,0,-4,0,0.3f,0.3f,0.3f);
18          window.addObject(b3.getObject());
19          Box3D b4 = new Box3D(8,4,0.1f,0,0,-8,0.5f,0.5f,0.5f);
20          window.addObject(b4.getObject());
21          Würfel3D würfel = new Würfel3D(1,0,0,0);
22          würfel.drehen(3);
23          window.addObject(würfel.getObject());
24      }
25
26      public static void main(String[] args) {
27          new Würfel();
28      }
29  }
```

Abbildung 12, Beispielanwendung

In der Beispielklasse Klasse Würfel, muss man ein Objekt der Klasse Window3D speichern (Zeile 8). Im Konstruktor wird dieses Objekt entsprechend der Display Größe und der gewünschten Hintergrundfarbe definiert (Zeile 11). Danach wird die Tastensteuerung aktiviert und es werden mehrere Quader im Raum und in der Mitte ein Würfel platziert, welcher sich mit der Geschwindigkeit v ist gleich drei dreht (Zeile 12-23).

Um die Klasse aufzurufen, muss nur noch die main-Methode ein Objekt der Klasse Würfel erstellen und schon ist die Animation fertig (siehe Anhang 1 und 2, S. 4)!

5. Fazit

Das Ziel meiner Facharbeit war die Erstellung einer modular aufgebauten Bibliothek zur Erstellung von 3D-Animationen. Dieses ist gut gelungen und man kann mit Hilfe dieser Bibliothek sehr einfach eigene Animationen erstellen. Des Weiteren lässt sich die Bibliothek gut erweitern, wodurch ich in Zukunft mein Projekt ausbauen kann und werde. Anhand der vorgegebenen Beispielanwendung, sind Anforderungen, wie die individuelle Textuierung und kontinuierliche Drehungen um eine sich stetig ändernde Rotationsachse, aufgekommen, welche zur einem besseren Grundverständnis der ganzen Java3D Bibliothek geführt haben.

Alles in allem schließe ich meine Facharbeit als vollen Erfolg ab, da ich nun durch wenige Zeilen Quelltext meine Ergebnisse in anderen Projekten veranschaulichen kann und mir neue Türen für vielseitige Projekte offen stehen.

6. Literaturverzeichnis

[1] Java Archive Downloads -Java Client Technologies. Online verfügbar unter http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-java-client-419417.html#java3d-1.5.1-oth-JPR, zuletzt geprüft am 12.04.2018.

[2] Java3D_1_1. Online verfügbar unter https://www.virtualworlds.de/Java3D/Java3D_1_1.pdf, zuletzt geprüft am 13.04.2018.

[3] NetBeans IDE, die Open-Source Java IDE. Online verfügbar unter https://netbeans.org/index_de.html, zuletzt geprüft am 21.03.2018.

[4] Texture2D (Java 3D 1.3.2) (2005). Online verfügbar unter http://download.java.net/media/java3d/javadoc/1.3.2/javax/media/j3d/Texture2D.html, zuletzt aktualisiert am 15.03.2005, zuletzt geprüft am 11.04.2018.

[5] TextureLoader (Java 3D 1.3.2) (2005). Online verfügbar unter http://download.java.net/media/java3d/javadoc/1.3.2/com/sun/j3d/utils/image/TextureLoader.html, zuletzt aktualisiert am 15.03.2005, zuletzt geprüft am 11.04.2018.

[6] Transform3D (Java 3D 1.3.2) (2005). Online verfügbar unter http://download.java.net/media/java3d/javadoc/1.3.2/javax/media/j3d/Transform3D.html, zuletzt aktualisiert am 15.03.2005, zuletzt geprüft am 08.04.2018.

[7] TransformGroup (Java 3D 1.3.2) (2005). Online verfügbar unter http://download.java.net/media/java3d/javadoc/1.3.2/javax/media/j3d/TransformGroup.html, zuletzt aktualisiert am 15.03.2005, zuletzt geprüft am 08.04.2018.

[8] Material (Java 3D 1.4.0) (2006). Online verfügbar unter http://download.java.net/media/java3d/javadoc/1.4.0/javax/media/j3d/Material.html#Material(javax.vecmath.Color3f,%20javax.vecmath.Color3f,%20javax.vecmath.Color3f,%20javax.vecmath.Color3f,%20float), zuletzt aktualisiert am 15.02.2006, zuletzt geprüft am 11.04.2018.

[9] Appearance Is Everything (2017). Online verfügbar unter http://www.java3d.org/appearance.html, zuletzt aktualisiert am 29.04.2017, zuletzt geprüft am 11.04.2018.

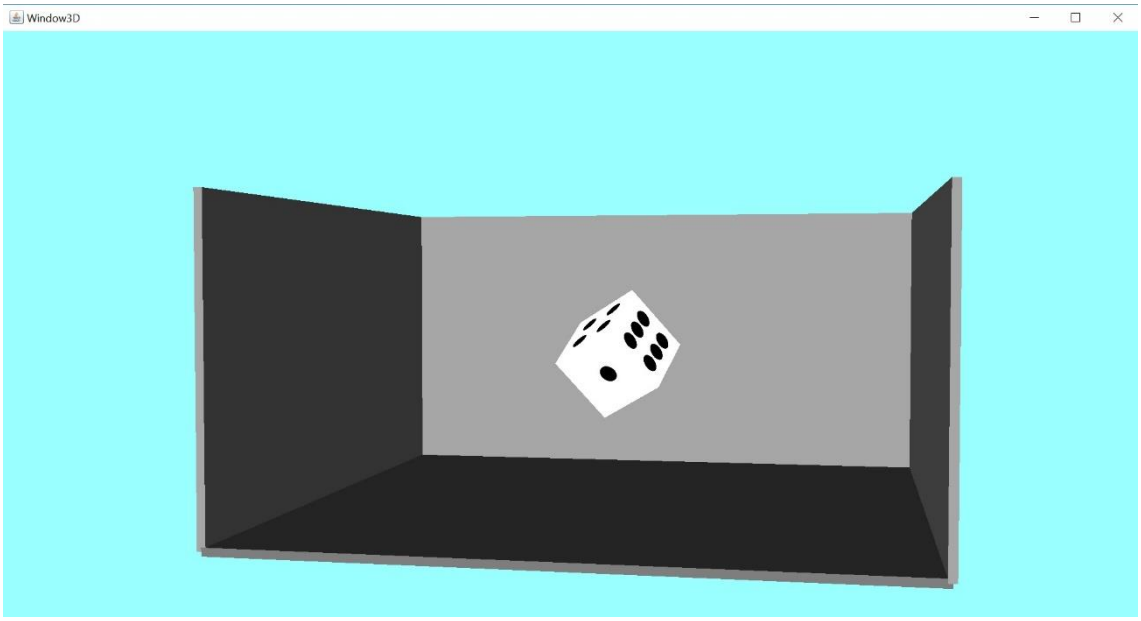
[10] Wikipedia (Hg.) (2018): Java 3D. Online verfügbar unter https://de.wikipedia.org/w/index.php?oldid=145971705, zuletzt aktualisiert am 20.03.2018, zuletzt geprüft am 04.04.2018.

[11] Wikipedia (Hg.) (2018): Java (Programmiersprache). Online verfügbar unter https://de.wikipedia.org/w/index.php?oldid=175233835, zuletzt aktualisiert am 21.03.2018, zuletzt geprüft am 21.03.2018.

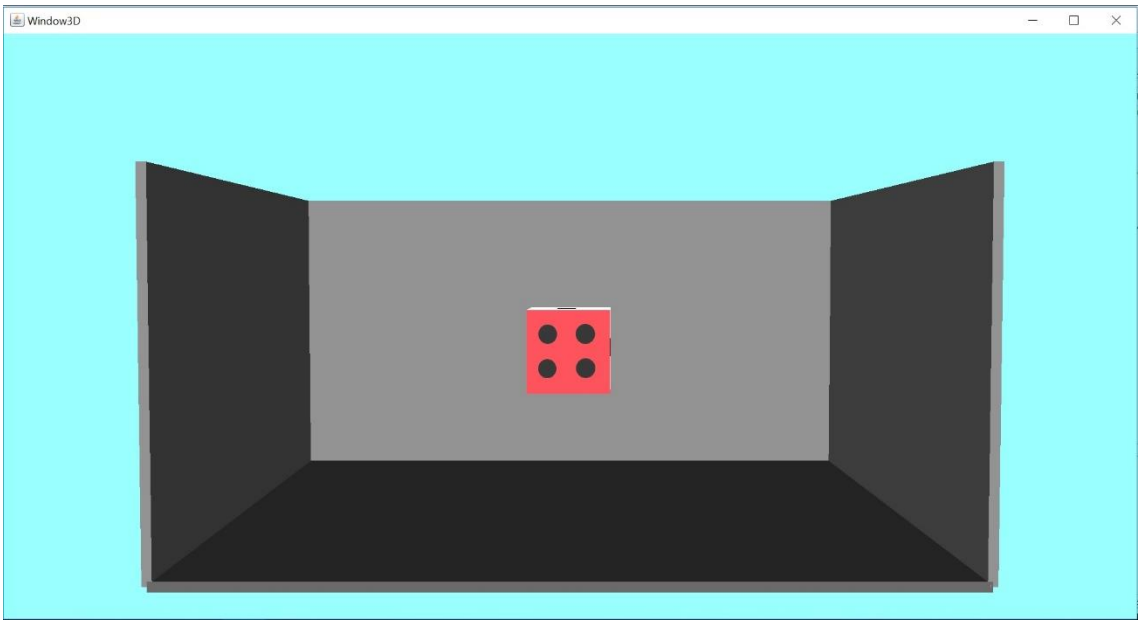
7. Abbildungsverzeichnis

Quelle: Alle genutzten Abbildungen und Texturen habe ich selbst erstellt.

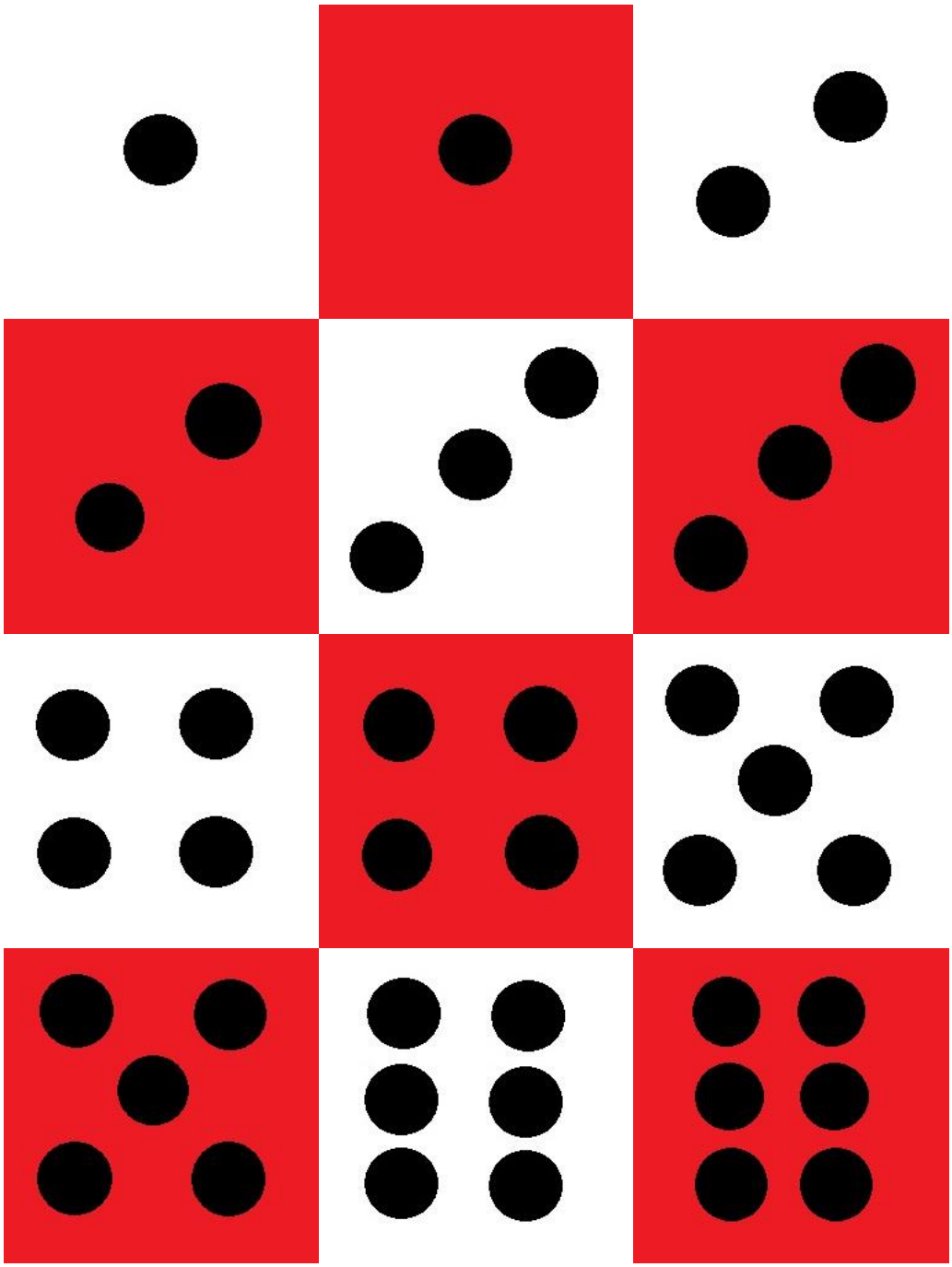
8. Anhang



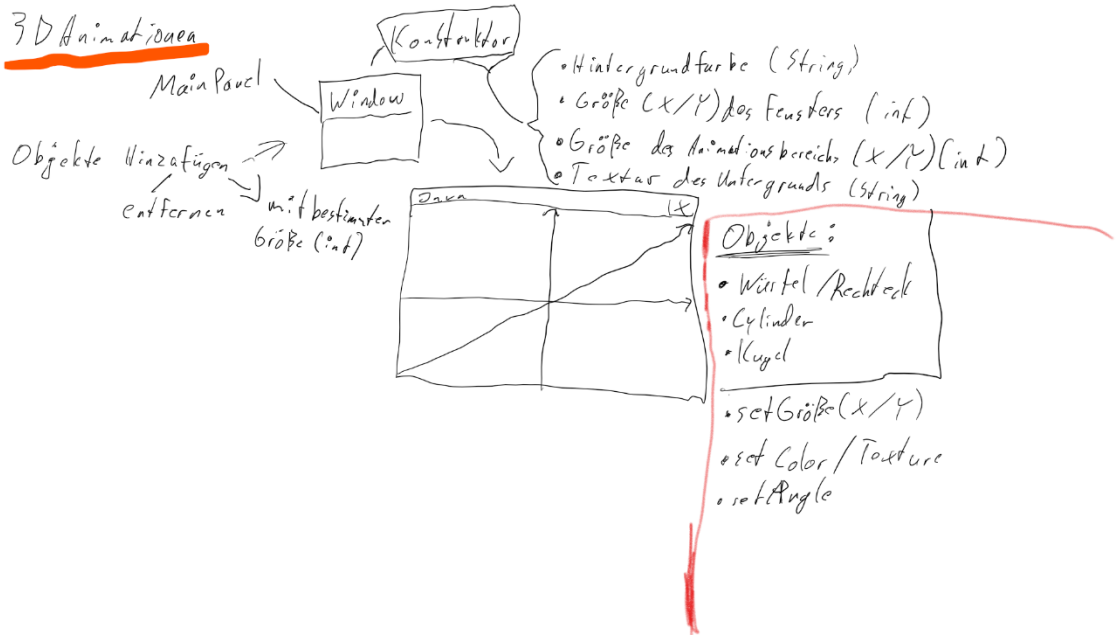
Anhang 1, Würfel in der Drehung aus einer schrägen Perspektive



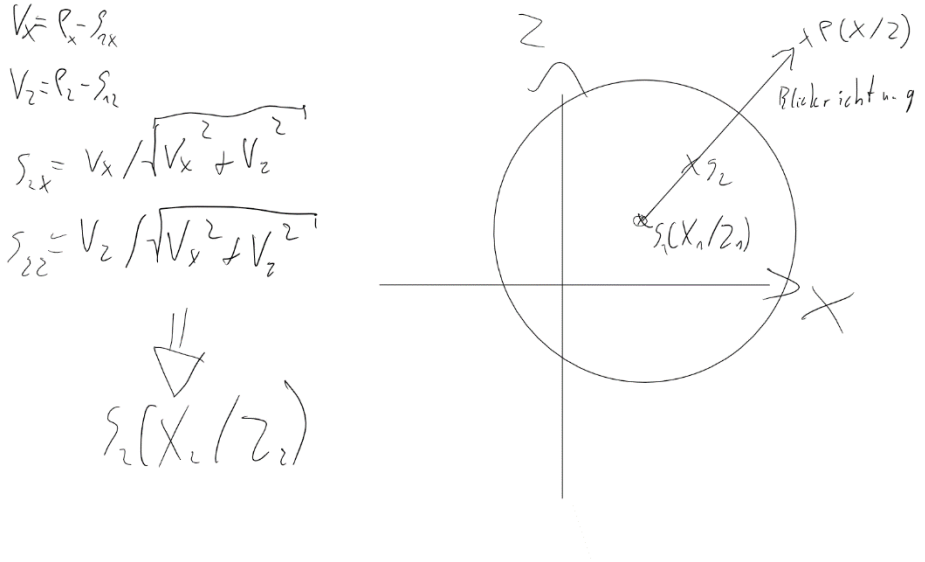
Anhang 2, Fertig gewürfelter Würfel aus einer frontalen Perspektive



Anhang 3, Texturen des Würfels als Gitternetz



Anhang 4, Überlegungen zum generellen Aufbau



Anhang 5, Überlegungen zur Steuerung

Digitaler Anhang:

Die komplette Klassendokumentationen aller Klassen sowie der restliche Anhang befindet sich auf dem beiliegenden USB-Stick.

9. Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die hier vorliegende Facharbeit ohne fremde Hilfe und nur mit den im Literaturverzeichnis angeführten Quellen und Hilfsmitteln erstellt habe.

Buschhoven, 13. April 2018

_____

Julius Klodt