**Inhaltsverzeichnis**

Abbildungsverzeichnis 02

Tabellenverzeichnis 02

Vorwort 03

Abstract 04

Einleitung 05

Geschichte des Rubik’s Cube 06

Geschichte der Java 3D Bibliothek 06

Theoretischer Teil 07

Rubik’s Cube 07

Begriffe 07

Funktionsweise des Würfels 08

Java 3D Bibliothek 09

Aufbau der Bibliothek 09

Aufbau des SceneGraph Modells 10

Praktischer Teil 13

Vorgehen 13

Planung 13

Informationssuche 13

Informationsverarbeitung 13

Beispielprogramme 13

1. Beispielprogramm 13
2. Beispielprogramm 14
3. Beispielprogramm 14
4. Beispielprogramm 15

Programm-Dokumentation 17

Gesamtheit des Programms 16

Main-Klasse 17

GUI-Klasse 17

Rubiks\_Cube-Klasse 17

Stone-Klasse

Herausforderungen des Programms 18

Erstellung der Steine mit Drehung 18

Begriff Koordinatenausrichtung 20

Dokumentation für Endbenutzerinnen und Endbenutzer 20

Diskussion 21

Quellen 22

Anhang 1 / Projektvertrag Maturaarbeit 23

Anhang 2 / Disposition – Maturaarbeit 24

Anhang 3 / Zeitplan – Maturaarbeit 25

Anhang 4 / Klassenbaum Java 3D 26

Anhang 5 / Eigenes Programm 27

Selbständigkeitserklärung ??

**Abbildungsverzeichnis**

Abb. 1: Der gelöste Würfel mit Begriffen 07

Abb. 2: Querschnitt durch den Würfel auf der Höhe der ersten Ebene 08

Abb. 3: Klassenhierarchie der javax.media.j3d Bibliothek 09

Abb. 4: Symbole für den SceneGraph 10

Abb. 5: Beispiel für einen einfachen SceneGraph 11

Abb. 6: Das SimpleUniverse übernimmt die rechte Seite des SceneGraph’s 11

Abb. 7: Erstes Programm mit der Verwendung der Java 3D Bibliothek 14

Abb. 8: Zweites Java 3D Programm mit dem ColorCube 14

Abb. 9: Drittes Java 3D Programm mit einer eigenen Geometrie 15

Abb. 10: Stone-Objekt, welches aus sechs QuadArrays besteht 16

Abb. 11: Das fertige Programm 17

**Tabellenverzeichnis**

Tab. 1: Beziehung der objekteigenen Koordinatensysteme im Koordinatensystem des *Locale‘s* 20

**Vorwort**

In meiner Maturaarbeit habe ich es geschafft, zwei meiner Interessen in einem Projekt zu vereinen. Das eine ist der Rubik’s Cube – ein faszinierendes farbiges Knobelspiel – und das andere ist die Informatik, die für einen Technikbegeisterten eine grosse Anziehungskraft hat.

Bereits seit vier Jahren fasziniert mich der Rubik’s Cube. Mein jüngerer Bruder erhielt damals zu Weihnachten einen solchen. Seine Begeisterung hielt sich in Grenzen. Als der Würfel erst verdreht war, verlor er das Interesse daran.

Ich versuchte mich ebenfalls mit dem neuen Spielzeug, da es mein Neugier geweckt hatte. Meine ersten Lösungsversuche scheiterten kläglich. Weil mich der Ehrgeiz gepackt hatte, suchte ich auf dem Internet nach Lösungswegen. Nach kurzer Zeit hatte ich eine Dokumentation gefunden, mit der ich den Würfel erfolgreich lösen konnte. Mit der Zeit lernte ich die Algorithmen auswendig.

Die Informatik zog mich bereits in der Volksschule in ihren Bann. Wir hatten in der sechsten Klasse einige Computer in unserem Zimmer, was an unserer Schule eine Ausnahme bildete. Die Sekundarschule bot mir die Möglichkeit, einen Informatikkurs zu absolvieren. Am Gymnasium besuchte ich das Fakultativfach Informatik und ab der Sekunda dasselbe als Ergänzungsfach.

Zu Beginn der Arbeit kannte ich die unterschiedlichen Eigenschaften des Rubik’s Cube gut; entsprechende Literatur besass ich bereits. Die Informatik war dagegen eher unbekannt. Obwohl ich seit zwei Jahren mit Java arbeitete, stellte sich heraus, dass ich eine ganze Bibliothek neu lernen musste.

Mein Projekt wäre niemals so erfolgreich verlaufen, wenn ich nicht von mehreren Leuten in den unterschiedlichen Phasen unterstützt worden wäre:

Mein erster Dank gilt meinem Betreuer, Tobias Bäumlin, der mich durch das ganze Projekt begleitet hat und mir immer wieder mit konstruktiven Lösungen weiterhalf.

Mein zweiter Dank widme ich Werner Randelshofer, Informatikstudent an der ETH Zürich, der mir den Einstig in die dreidimensionale Welt ermöglicht hat. Die beiden Bücher, die er mir auslieh, haben mir geholfen, die Komplexität der virtuellen Welt zu verstehen.

Ebenfalls danke ich Moritz Kobel, BSc BFH in Electrical Engineering, für seine nützlichen Ratschläge. Er half mir aus der Sicht des Softwareentwicklers meine Probleme zu analysieren und mit mir mögliche Lösungsansätze zu besprechen. Zusätzlich hat er mir gezeigt, wie ich mein Projekt mit GitHub [1] organisieren kann.

Nun wünsche ich allen Leserinnen und Leser viel Spass beim Erkunden des Rubik’s Cube, dem Verstehen der Java 3D Bibliothek und dem Entschlüsseln meines Programmcodes.

Rafael Sebastian Müller

1: GitHub ist ein Dienst für der Versionsverwaltung für Software-Entwicklungsprojekte, (Quelle: wikipedia.org/wiki/GitHub Stand 07.10.2012)

**Abstract**

In meiner Maturaarbeit ist es mir gelungen, den Rubik’s Cube mit der Programmiersprache „Java“ zu visualisieren. Die Programmierung ermöglicht, den Würfel um den Ursprung zu drehen und die einzelnen Wände individuell zu rotieren.

Für die Programmierung habe ich mich entschieden die Java 3D Bibliothek (Bibliothek) zu verwenden. Sie stellt einen dreidimensionalen Raum zur Verfügung, in welchem Objekte eingefügt werden können.

Zum Start meiner Arbeit habe ich mich mit der Geschichte des Rubik’s Cube und der Bibliothek befasst.

Der Rubik’s Cube wurde im letzten Drittel des Kalten Krieges im kommunistischen Ungarn durch Erno Rubik erfunden. Der Würfel wurde dank zwei im Westen lebenden Ungaren zum Massenprodukt, das bis heute auf der ganzen Welt zu kaufen ist.

Die Bibliothek ist noch bedeutend jünger. Sie wurde von Sun Microsoft ab dem Jahr 1997 entwickelt. Heute ist die Version 1.5 aktuell.

Als Einstieg in die Programmierung übte ich mit Anleitungen aus dem Internet den Aufbau der Bibliothek. Gleichzeitig lernte ich die Eigenschaften und den Aufbau des Rubik’s Cube kennen.

Um die gelesenen Theorie in meinem Programm anwenden zu können, habe ich sie mit Beispielprogrammen ausgetestet. Mein Programm habe ich in vier Teile zerlegt, welche in der Programmiersprache als „Klassen“ bezeichnet werden. Die erste „Klasse“ startet das Programm, die zweite fertigt das Design an, in der dritten und vierten „Klasse“ wird der Würfel erstellt.

Die grösste Herausforderung waren die Drehbewegungen der einzelnen Wände, da jeder der siebenundzwanzig Steine des Würfels ein objekteigenes Koordinatensystem hat. Um eine Rotation auszuführen muss das Programm immer überprüfen, wie das objekteigene Koordinatensystem zum Koordinatensystem des Würfels ausgerichtet ist. Um dieses Problem zu lösen, habe ich ein eigenes Konzept entwickelt. In diesem Konzept wird überprüft, welche Wand gedreht werden soll und wie das objekteigene Koordinatensystem eines Steins ausgerichtet ist.

Die Programmierung ist erfolgreich. Ich habe ein lauffähiges Programm, welche die oben beschriebenen Rotationen ausführen kann.

**Einleitung**

Das Ziel dieser Arbeit ist die Erstellung eines lauffähigen Programms in der Sprache „Java“, das den Rubik’s Cube in einem dreidimensionalen Raum darstellt. Dabei sollen die Anwenderinnen und Anwender die gleichen Drehbewegungen durchführen können wie am realen Rubik’s Cube.

Mein Einstieg ins Projekt begann mit der Unterzeichnung des Projektauftrages (siehe Angang 1) meiner Disposition (siehe Anhang 2) und dem darauf aufbauenden Zeitplan (siehe Angang 3).

Aus dem oben beschriebenen Ziel entwickelten sich die die Aufgaben; sie bestanden darin,

* den dreidimensionalen Rubik’s Cube auf den Bildschirm –welcher eine zweidimensionale Oberfläche darstellt – zu projizieren.
* die Wände des Würfels um alle Seiten zu drehen.
* den gesamten Würfel um sein Zentrum rotieren zu lassen, damit alle Seiten ersichtlich sind.

Bei der ersten Aufgabe habe ich die mathematischen Berechnungen mit Hilfe der Java 3D Bibliothek, die es ermöglichen einen dreidimensionalen Effekt zu erzeugen, gelöst. Bei der zweiten Aufgabe war ich stark gefordert, da die meisten Steine um mehrere Achsen gedreht werden können. Ich musste mir daher eine spezielle Vorgehensweise ausdenken, die nach einer Drehung jede beliebige Rotation ermöglicht. Bei der dritten Aufgabe habe ich ein Objekt erstellt, das durch die Mausbewegung die Kameraansicht um den Ursprung verändert.

Als Einstieg in meinen Bericht habe ich einen historischen Abriss über den Rubik’s Cube und die Geschichte der Java 3D Bibliothek zusammengestellt.

Im theoretischen Teil gehe ich auf die Eigenschaften des Rubik’s Cube ein und beschreibe die für mein Programm relevanten Themen der Java 3D Bibliothek.

Im praktischen Teil dokumentiere ich die Entstehung meines Programms und die Lösung der aufgetretenen Herausforderrungen.

**Geschichte des Rubik’s Cube**

Die Geschichte des Rubik’s Cube beginnt bei Erno Rubik, einem Dozenten der Fakultät für Innenarchitektur an der Universität für angewandte Künste in Budapest. Er war an Geometrie und dreidimensionalen Formen sehr interessiert.

Er stellte sich selbst die Aufgabe ein dreidimensionales Knobelspiel, das viele Kombinationen zuliess und immer ein Ganzes bildete, zu erfinden. Seine ersten Versuche, einen Würfel mit Gummibändern zusammenzuhalten, scheiterten. Nach längeren Überlegungen kam er auf die Idee, dass die innere Struktur rund sein musste, damit sich die einzelnen Bestandteile des Würfels in alle drei Richtungen verdrehen liessen. Er entschied sich, die sechs verschiedenen Flächen durch Farben voneinander zu unterscheiden. Im Jahre 1974 reichte er die Patentanmeldung für seinen Prototyp ein.

Der ungarische Spielzeughersteller „Politechnika“ begann den Würfel als Massenprodukt herzustellen. Wegen der Komplexität im Inneren des Würfels und der schlechten inneren wirtschaftlichen Lage stellte sich dies als schwierig heraus. Erst 1979 konnten die ersten Würfel in Spielzeugläden gekauft werden.

Die zwei im Westen lebenden Ungaren Dr. Laczi Tibor und Tom Kremer erreichten, dass der Würfel auf der anderen Seite des Eisernen Vorhangs zum Massenprodukt wurde.

Dr. Laczi Tibor trieb die Produktion in Ungarn an und Tom Kremer versuchte weltweit Spielzeughersteller auf das Produkt aufmerksam zu machen. Schliesslich gelang es ihm, mit der Firma „Ideal Toy Corporation“ eine Bestellung von einer Million Exemplaren abzuschliessen.

1980 hatte der Würfel auf mehreren westlichen Spielzeugmessen grossen Erfolg. Nach Produktionsverbesserungen wuchs die Nachfrage global exponentiell an.

Nachdem der Sättigungsgrad erreicht worden war, brach der Handel für den Würfel 1983 zusammen. Tom Kremer gab jedoch den Würfel nicht auf, kaufte alle Recht an „Rubik’s Würfel“ und führte ihn mit einer amerikanischen Vermarktungsgesellschaft in ausgewählten Zielmärkten ein. Damit erreichte er ein Verkaufs-Comeback des Würfels.

Heute kann der Würfel wie andere Erfindungen von Erno Rubik immer noch gekauft werden.

**Geschichte der Java 3D Bibliothek**

Im Jahre 1997 begann die Entwicklung von Java 3D durch Sun Microsystems. Die Version 1.0 erschien im Jahre 1998. Die Entwicklung wurde von Sun zwischenzeitlich eingestellt. Seitdem ist die Bibliothek als Open Source freigegeben. 1.5 ist die aktuelle Version.

**Theoretischer Teil**

**Rubik’s Cube**

**Begriffe**

Um den Würfel zu beschreiben, erkläre ich in diesem Unterkapitel meine verwendeten Begriffe. Die Namen habe ich selbst ausgewählt.

**Würfel**

Rubik’s Cube mit allen seinen Elementen. Wenn es nicht explizit erwähnt wird, handelt es sich um den gelösten Rubik’s Cube. Das heisst, dass jede Wand nur eine Farbe hat.

**Wand**

Eine Seite des Würfels. Jede Seite ist dabei mit einer anderen Farbe koloriert. Normalerweise besitzt er folgende Farben: Weiss, Gelb, Rot, Orange, Blau, Grün.

**Stein**  
Er bezeichnet eine der neun Einzelteile, die jede Wand enthält. Es ist zu beachten, dass die meisten Steine in zwei oder drei Wänden gleichzeitig vorkommen.

**Drehung**

Der Würfel lässt sich insgesamt auf neun Arten drehen. Dabei lassen sich die Rotationen in drei Gruppen einteilen: Die eine Gruppe beinhaltet die Drehung von den drei horizontalen Ebenen um die y-Achse. Die zweite Gruppe beschreibt drei Bewegungen um die x-Achse und die letzte Gruppe enthält die Umdrehung von drei Ebenen um die z-Achse.

**Steintypen**

Jeder Steintyp kennzeichnet sich durch die gleiche Anzahl an farbigen Oberflächen. Sie können (mit Ausnahme der Mittelsteine) ihre Position untereinander austauschen.

Es gibt drei verschiedene Steintypen:

**Mittelstein**

Es handelt sich um einen Steintypen, der in der Mitte jeder Wand vorkommt. Es gibt pro Würfel insgesamt sechs.

**Kantenstein**

Es handelt sich um den Steintypen, der immer zwischen zwei Mittelsteinen liegt und zwei farbige Oberflächen hat. Es gibt insgesamt zwölf solcher Steine pro Würfel.

**Eckstein**

Es handelt sich um den Steintypen, der drei farbige Flächen besitzt. Er befindet sich, wie der Name schon sagt, in den Ecken des Würfels. Es gibt acht solcher Steine pro Würfel.

|  |
| --- |
| Abb. 1: Der gelöste Würfel mit Begriffen |
| Rubik's Cube mit BEschriftung.png |
| Der gelöste Würfel hat drei verschiedene Steintypen. Jeder Steintyp besitzt andere Eigenschaften. Der Begriff Wand bezieht sich auf die komplette Seite des Würfels. |

**Funktionsweise des Würfels**

Der Aufbau des Würfels hat etwas Faszinierendes. Obwohl man jede Seite beliebig oft und in beliebiger Reihenfolge verdrehen kann, fällt er (fast) nie auseinander.

Um das Phänomen zu verstehen, muss man das Innenleben des Würfels kennen.

* Die erste wichtige Erkenntnis ist, dass sämtliche Mittelsteine miteinander verbunden sind. Dadurch behält jeder Mittelstein stets seinen gegenüberliegenden Mittelstein; das heisst der weisse gegenüber dem gelben, der rote gegenüber dem orangen und der blaue gegenüber dem grünen. Durch die Fixierung lassen sich die Mittelsteine nur um ihre eigene Achse drehen.
* Damit die Flexibilität ermöglicht wird – die garantiert, dass nach einer Drehung jede andere Drehung möglich ist – ist die innere Struktur kugelartig geformt (Abb. 2). Die einzelnen Steine halten sich zusammen, da sie sich gegenseitig verkannten. Sobald ein Stein fehlt, beginnt der Würfel auseinanderzufallen.

|  |
| --- |
| Abb. 2: Querschnitt durch den Würfel auf der Höhe der ersten Ebene |
| runde Innenstruktur.png |
| Durch den Querschnitt ist die kugelartige Innenkonstruktion klar zu erkennen. Sie sorgt dafür, dass die Steine sich gegenseitig zusammenhalten. Fehlt ein Stein, beginnt der Würfel auseinanderzufallen. |

**Java 3D Bibliothek**

Beim Erstellen meines Projekts habe ich vorwiegend mit der Bibliothek *javax.media.j3d* gearbeitet. Es handelt sich um eine Java-Bibliothek, die dreidimensionale Objekte erstellen und verändern kann. Zurzeit ist Java 3D noch nicht teil der *Java Fondation Class* (*JFC*).

**Aufbau der Bibliothek**

Die Bibliothek besteht aus mehreren Klassen. Zum grössten Teil gehören diese zum Klassenverband S*ceneGraphObject*. Da die Bibliothek sehr umfassend ist (Abb. 3), habe ich mich entscheiden, nur die selbst verwendeten Klassen zu beschreiben.

|  |
| --- |
| Abb. 3: Klassenhierarchie der *javax.media.j3d* Bibliothek |
| verwendete Klassen in Hierarchie_1.png |
| Die Klassen haben eine klare Hierarchie. Um mit der Bibliothek arbeiten zu können, ist das Verständnis der Hierarchie entscheidend. Die gelb markierten Klassen habe ich in meinem Programm selber verwendet. Es gibt noch eine Menge weiterer Klassen, die auf der Grafik nicht zu sehen sind. (Gesamtansicht im Anhang 4). |

Das *VirtualUniverse* bildet einen Container, in welchen sämtliche eingesetzten *SceneGraphObjects* eingebunden werden müssen. Die meisten Programme enthalten nur ein *VirtualUniverse*, obwohl es denkbar ist, mehrere zu verwenden. Das *VirtualUniverse* stellt im Programm einen virtuellen Raum dar.

Das *Locale* definiert einen Ursprung im *VirtualUniverse* und stellt ein kartesisches Koordinatensystem dar.

Die *SceneGraphObjects* werden in zwei Hauptgruppen unterteilt: Der *NodeComponent* und der *Node*.

* Im *NodeComponent* werden Geometrie, Aussehen und Texturen des *LeafNode* spezifiziert.
  + Der *QuadArray* enthält Arrays, in welchen Koordinaten, Farben und Texturen gespeichert werden können. Dabei bilden immer vier Array-Elemente die Eckpunkte eines Vierecks.
* Der *Node* enthält Objekte, die zum Aufbau eines Programms benötigt werden. Er gliedert sich in *Group* und *Leaf* unterteilt.
  + *Group* beinhaltet *BranchGroup* und *TransformGroup*.
    - *BranchGroup* sind die einzigen Objekte, die direkt hinter einem *Locale* stehen können. Wenn eine *BranchGroup* zu einem *Locale* oder einem *SimpleUniverse* hinzugefügt wird, werden die BranchGroup und alle ihre Child-Objekte aktiv. Durch das Entfernen der *BranchGroup* wird dieser Effekt rückgängig gemacht.
    - Alle Objekte die in der *TransformGroup* enthalten sind, können dadurch transformiert, skaliert und rotiert werden.
  + Der *Leaf* enthält *Shape3D*, welches eine oder mehrere geometrische Komponente(n) enthält und eine einzelne Komponente für das Aussehen.

*Transform3D* nimmt einen Wert auf, welcher der *TransformGroup* übergeben wird. Dadurch werden Manipulationen (Transformation, Skalierung, Rotation) der *TransformGroup* erreicht.

***Capabilities***

Sie kontrollieren den Schreib- und Lesezugriff. Solange ein *Node* nicht aktiv ist, können seine Attribute verändert werden. Die *Capabilities* kontrollieren den Zugriff, sobald ein *Node* aktiv ist.

Capabilities sollten sparsam eingesetzt werden, damit besser optimiert werden kann. Die Capabilities müssen direkt nach dem Erstellen eines Objekts gesetzt werden. Sobald das Objekt aktiv ist, können keine Capabilities mehr gesetzt werden.

**Aufbau des *SceneGraph* Modells**

Um ein Projekt mit der Java 3D Bibliothek zu realisieren, muss man sich an eine klare Struktur halten. Es handelt sich um den *SceneGraph*, eine hierarchische Baumstruktur. Alle Objekte, die verwendet werden, müssen dem *SceneGraph* hinzugefügt werden. Wichtig ist, dass es immer genau einen Weg vom hierarchisch höchsten Objekt bis zu jedem anderen gibt.

|  |
| --- |
| Abb. 4: Symbole für den *SceneGraph* |
| Symbole für SceneGraph.png |
| Um ein Programm grafisch darzustellen, ist es sinnvoll für bestimmte Objekte eigene geometrische Formen zu verwenden. Zu beachten ist, dass *Group* und *Leaf* nicht ein bestimmtes Objekt dartellen, sondern eine Gruppe von Objekten. |

Die Symbole auf der linken Seite stellen Objekte dar, die im *SceneGraph* Verwendung finden. Die ersten beiden bilden Objekte spezifischer Klassen. Das dritte, vierte und fünfte Objekt beinalten die *BranchGroup* / *TransformGroup* bzw. *Shape3D* bzw. *QuadArray*. Das letzte Symbol wird verwendet, um Klassen zu beschreiben, die in keine der obigen Objekte fallen.

Auf der rechten Seite sind die Symbole für die Beziehungen zwischen den Objekten dargestellt. Diese heissen *Arcs*. Die *Parent-Child*- und die *Reference*-Beziehung wird durch eine durchgezogenen bzw. gestrichelten Pfeil abgebildet.

|  |
| --- |
| Abb. 5: Beispiel für einen einfachen *SceneGraph* |
| SceneGraph Beispiel.png |
| An diesem *SceneGraph* sind sämtliche bereits erklärten Symbole vorhanden. Diese Symbole, die eine Gruppe von Objekten darstellen können, sind weiter mit Buchstaben gekennzeichnet, damit keine Verwechslungen entstehen. |

Das *VirtualUniverse* bildet die Wurzel des *SceneGraph*. Unterhalb des *VirtualUniverse‘s* folgt das *Locale*. Danach verzweigt sich der *SceneGraph* zum ersten Mal in zwei *BranchGroups*. Jede der beiden hat eine andere Aufgabe.

* Die *ContentBranchGroup* ist für sämtlichen Inhalt zuständig (Geometrien, Darstellung, Bewegungen, Beleuchtungen).
* Die *ViewBranchGroup* erzeugt die Ansicht (Kameraposition, Kameraausrichtung).

In der *ContentBranchGroup* können nun die Objekte eingefügt werden, die im vorherigen Kapitel beschrieben wurden. Die *Nodes* bilden dabei Knotenpunkte im *SceneGraph*. Zu beachten ist, dass *GroupNode* immer ein *Child*-Objekt hat und *LeafNode* immer am Ende eines Astes steht; mit anderen Worten, kein *Child-*Objekt hat.

|  |
| --- |
| Abb. 6: Das *SimpleUniverse* übernimmt die rechte Seite des *SceneGraph*’s |
| SceneGraph Beispiel.png |
| Das *SimpleUniverse* erleichtert das Erstellen eines Java 3D Programmes, weil es die gesamte rechte Seite selbst erstellt (blau eingerahmt). |

Da das *VirtualUniverse*, das *Locale* und die *ViewBranchGroup* in den meisten Programmen die gleichen Eigenschaften enthalten sollen, gibt es eine Hilfsklasse, die sich *SimpleUniverse* nennt. Sie erstellt die drei oben genannten Objekte und setzt Standartwerte ein. Ich habe diese Hilfsklasse verwendet und gehe deshalb nicht weiter auf ihre Objekte ein.

**Praktischer Teil**

**Vorgehen**

**Planung**

Um den Ablauf für meine Arbeit grob zu strukturieren, habe ich mich entschieden einen Zeitplan (siehe Anhang 4) zu erstellen. Die Teilarbeiten, bei denen ich den grössten Zeitaufwand vermutete, setzte ich in meine Frühlings-, Sommer- und Herbstferien 2012, da ich dort am wenigsten unvorhergesehene Ablenkungen vermutete. Die Rücksprachen mit meiner Betreuungsperson terminierte ich nach längeren Arbeiten, damit sie mir bei Schwierigkeiten aufzeigen konnte, wie ich mein Programm weiterentwickeln kann.

Die Einhaltung des Planes stellte sich als herausfordernd heraus, da ich die zeitaufwendigen Etappen im Voraus nicht genau abschätzen konnte. Auch mit der Verschiebung einzelner Termine konnte ich das Projekt termingerecht abschliessen.

**Informationssuche**

Meine erste Informationssuche startete ich auf dem Internet. Gleichzeitig halfen mehrere Lehrpersonen allen Schülerinnen und Schüler meiner Klasse Informationen für unsere Maturaarbeiten zu suchen, indem sie uns in die Schulhaus Mediothek und die UNI-Bibliothek führten, um uns Hinweise für die Informationsbeschaffung zu geben. Ich fand weder auf dem Internet noch in den Bibliotheken geeignete Dokumente, da ich vermutlich zu diesem Zeitpunkt noch nicht nach den richtigen Quellen suchte.

Einen ersten Fortschritt erlebte ich am Besuchstag der ETH Zürich im Gymnasium Neufeld. Ich erhielt von einem Informatikstudenten den Hinweis, dass ein Studienkollege von ihm, Werner Randelshofer, sich im Rahmen seiner Masterarbeit intensiv mit dem Rubik’s Cube auseinandersetze. Nach der ersten Kontaktaufnahme lieh er mir zwei Bücher aus, welche sich mit Java und der Herausforderungen des Raumes in der Informatik beschäftigen.

Der Durchbruch gelang mir dank meiner Betreuungsperson, die mir riet, die Java 3D Bibliothek für mein Projekt zu verwenden. Um die Java 3D Bibliothek zu verstehen, beschaffte ich mir im Internet eine Tutorial-Reihe und die dazugehörige Dokumentation.

**Informationsverarbeitung**

Während meiner Arbeit habe ich mich intensiv mit dem Rubik’s Cube und der Java 3D Bibliothek auseinandergesetzt. Meine wichtigsten Erkenntnisse dazu sind im Kapitel „Theoretischer Teil“ dargestellt.

In die Programmierung habe ich mich mit Beispielprogrammen eingearbeitet, danach habe ich das Hauptprogramm entwickelt. Den „langen“ Abschluss bildete die Implementierung des Rotations-Algorithmus.

**Beispielprogramme**

Bevor ich mich meinem Programm gewidmet habe, versuchte ich mich mit Beispielprogrammen einzuarbeiten. Ich habe eine Menge solcher Beispiele erstellt. In diesem Kapitel erkläre ich die Beispiele, die für meine Arbeit am Wichtigsten waren.

**1. Beispielprogramm**

Als ich mit dem ersten Programm begann, wusste ich nicht genau, wie ich starten sollte. Also habe ich mit dem Teil begonnen, den ich bereits von Java kannte: Ich erstellte ein *JFrame*, in welchem mein Programm später laufen sollte. Ich habe mich von Beginn an dazu entschieden immer *JFrame* als Container zu wählen. In das *JFrame* fügte ich ein *Canvas3D*- und ein *SimpleUniverse*-Objekt ein, welche die Grundlage darstellten. Das *SimpleUniverse* war jedoch noch leer. Nachdem ich eine *BranchGroup* mit einem *ColorCube*[1] hinzufügte, war ich bereits zufrieden mit meinem ersten Programm.

Zu erkennen ist, dass bereits dieses einfache Programm alle wesentlichen Teile eines *SceneGraph’s* enthält.

[1]: Der ColorCube ist aus der Hilfsklasse *com.sun.j3d.utils.geometry.* Das Objekt stellt einen Würfel dar, mit sechs unterschiedlich eingefärbten Seiten

|  |
| --- |
| Abb. 7: Erstes Programm mit der Verwendung der Java 3D Bibliothek |
| Rubik's Cube Version 1.png |
| Das Programm, welches zu sehen ist, war mein erstes lauffähiges Programm in Java 3D. Es zeigt einen Würfel aus der Richtung der z-Achse. Die anderen Seiten sind nicht zu sehen, weil der Würfel parallel zum Betrachter steht. |

**2. Beispielprogramm**

Im zweiten Programm habe ich zwischen den *ColorCube* und der *BranchGroup* einen weiteren *GroupNode* eingefügt. Es handelte sich um eine TransformGroup. Zusätzlich sind zwei Objekte der Klasse *Transform3D* hinzugefügt worden. Mit diesen zwei Änderungen waren bereits zwei Seiten des *ColorCube* sichtbar. Zuerst hatte ich Probleme mit der Transformation, da ich den Winkel in Grad, statt im Bogenmass angegeben habe. Seither verwende ich immer die Math.toRadiant().

Um zwei *Transform3D*-Objekte zusammen zu rechnen, wird auf das *Transform3D*-Objekt die mul(Transform3D td) angewendet.

|  |
| --- |
| Abb. 8: Zweites Java 3D Programm mit dem ColorCube |
| Rubik's CUbe Vefrsion 2.0.png |
| Nach dem Hinzufügen eines *TransformGroup*- und *Transform3D*-Objekts ist der Würfel bereits als solcher zu erkennen. Wichtig ist, dass der Winkel im Bogenmass angegeben wird. |

**3. Beispielprogramm**

In diesem Programm habe ich noch tiefer in das *SceneGraph*-Modell eingegriffen. Dieser Schritt war einer der wichtigsten, da er mir erlaubte, eigene Geometrien anzuwenden. Für mich war es schwierig herauszufinden, welche der vielen Subklassen der *GeometryArray*-Klasse für mein Programm am besten anwendbar ist. Ich entschied mich für die *QuadArray*-Klasse, weil ich in meinem Programm quadratische Formen benötige.

|  |
| --- |
| Abb. 9: Drittes Java 3D Programm mit einer eigenen Geometrie |
| Rubik's CUbe Vefrsion 3.0.png |
| Die *QuadArray*-Klasse hat sich als beste Klasse herausgestellt, da sie unabhängige quadratische Formen zeichnet, welche ich zur Erstellung eines *Stone*-Objektes benötige. |

**4. Beispielprogramm**

Im vierten Programm habe ich aus mehreren *QuadArrays* einen eigenen Würfel zusammengestellt. Hierbei ist zu beachten, dass Java 3D immer nur die Vorderseite darstellt. Als Vorderseite wird diese Seite betrachtet, von der die Reihenfolge der Punkte im Gegenuhrzeigersinn verläuft.

|  |
| --- |
| Abb. 10: *Stone*-Objekt, welches aus sechs *QuadArrays* besteht |
| Rubik's CUbe Vefrsion 4.0.png |
| Der Würfel wurde aus acht Punkten erstellt. Diese Punkte werden zu sechs Quadraten verbunden. Dabei ist zu beachten, dass die Reihenfolge der Punkte im Gegenuhrzeigersinn verläuft. |

**Programm-Dokumentation**

Nach diesen Beispielprogrammen habe ich mit der eigentlichen Programmierung begonnen. Bei der Beschreibung meines Programms gehe ich auf die einzelnen Klassen ein und erkläre die wichtigsten Objekte und Methoden. Ich kennzeichne Methoden, indem ich hinter ihren Namen ein rundes Klammerpaar anfüge.

**Gesamtheit des Programms**

Mein Projekt besteht aus vier Klassen (Main, GUI, Rubiks\_Cube, Stone). Der Hauptteil der Arbeit befindet sich in der Rubiks\_Cube- und Stone-Klasse.

**Main-Klasse**

Die Main-Klasse bildet die Hauptklasse und damit den Programmstart (Entry Point). In der main() wird ein neues Objekt der GUI-Klasse erstellt. Ansonsten beinhaltet die Klasse keine weiteren Methoden.

**GUI-Klasse**

In der GUI() wird das Fenster-Gerüst aufgebaut, in welchem der Rubik’s Cube zu sehen ist. Um eine Interaktion des Benutzers zu ermöglichen, werden sechs Knöpfe implementiert. Sobald ein Knopf gedrückt wird, ermittelt *actionPerformed(ActionEvent e)*, um welchen Knopf es sich handelt und ruft dementsprechend eine andere Methode in der Rubiks\_Cube-Klasse auf.

|  |
| --- |
| Abb. 11: Das fertige Programm |
| Rubik's CUbe Version 5.0.png |
| Auf dem *JFrame* sind die sechs Knöpfe und das *Canvas3D*-Objekt. Durch das Drücken einer der Knöpfe wird eine Seite rotiert. |

**Rubiks\_Cube-Klasse**

Die Klasse hat zwei Aufgaben. Zum einen erstellt sie mit createViewBranchGraph() die Ansicht auf den Rubik’s Cube, zum anderen wird der Inhalt des Programms generiert.

Die erste Aufgabe wird in der createViewBranchGraph() ausgeführt; die Standard-Konfiguration wird eingestellt und die Kameradistanz zum Würfel festgelegt.

Die zweite Aufgabe wird in der createContentBranchGraph() erledigt. Der Würfel wird durch die getStart() gedreht, damit zu Beginn der Applikation drei Seiten sichtbar sind. Danach fügt der Aufruf der createMouseBehavior() dem SceneGraph ein neues Objekt hinzu, das dem Endnutzer erlaubt, die Kameraposition um den Rubik’s Cube zu bewegen. Die nächste Schleifengruppe fügt der Szene die Bestandteile des Rubik’s Cube hinzu. Die einzelnen Steine werden, bevor sie in den SceneGraph eingefügt werden, noch weiter verschachtelt. Die Verschachtelung ermöglicht, dass sie rotieren können. Zum Schluss des Aufbaus werden mit der createCoordSystem() der Szene noch drei Linien hinzugefügt, die das Koordinatensystem visuell darstellen.

Weiter werden die Interaktionen aus der GUI-Klasse verarbeitet. Dies ist der Fall, wenn der Endbenutzer einer der sechs Knöpfe drückt, die sich am unteren Bildschirmrand des Programms befinden. Alle sechs Methoden haben die Aufgabe eine Seite zu rotieren. Je nachdem welcher Knopf gedrückt wird, wird die Methode aufgerufen, welche die entsprechende Seite dreht. Die sechs Methoden führen ähnliche Befehle aus.

Wenn eine Methode aufgerufen wird, wird auf alle *Stonne*-Objekte, die für die Rotation benötigt werden, zurückgegriffen und danach die getTransform() aufgerufen.

Wenn die Methode aufgerufen wird, dreht sie eine Seite um eine Koordinatenachse. Da jedes *Stone*-Objekt sein eigenes Koordinatensystem hat, muss nicht jeder Stein um die gleiche Achse gedreht werden. Deshalb wird jeder Stein einzeln gedreht. Danach wird die changeArrayPosition() aufgerufen, welche die Steine im Array ihrer neuen Position zuweist.

**Stone-Klasse**

In dieser Klasse werden in der *createGeometry()* aus den drei übergebenen Parametern acht Punkte berechnet. Jeder Punkt bezeichnet eine der Ecken, die jeder Stein hat. Sechs *QuadArrays* erstellen anhand der Punkte je ein Quadrat. Jedem Quadrat, das später an der Oberfläche des Würfels zu sehen sein wird, wird in der *createAppearnace()* eine Farbe zugewiesen. Alle Quadrate werden in einem Shape3D-Objekt zu einem dreidimensionalen Stein zusammengefügt und der Rubiks\_Cube-Klasse durch die *getTransformGroup()* zurückgegeben.

Die weiteren Methoden werden für die Interaktion des Endbenutzers benötigt.

**Herausforderungen des Programms**

In diesem Kapitel beschreibe ich einen Lösungsansatz, der von grosser Bedeutung für mein Programmergebnis ist.

**Erstellung der Steine mit Drehung**

Im Kapitel „Theoretischer Teil“ habe ich bereits erklärt, dass der Würfel aus drei verschiedenen Steintypen besteht. Zu Beginn der Programmierung hatte ich die Idee drei Klassen zu erstellen. Jede davon sollte einen Steintypen repräsentieren. Im Verlaufe der Arbeit habe ich mich dagegen entschieden, weil der Aufwand die Vorteile übersteigt. Danach habe ich mich für eine Klasse entschieden, welche für alle *Stone*-Objekte zuständig ist.

1. Um den Würfel zu erstellen, habe ich die *Stone*-Objekte in einen dreidimensionalen Array eingefügt. Der Array hat den Vorteil, dass ich die *Stone*-Objekte direkt mit ihren Koordinaten ansprechen kann. Dies ist für die Rotationen notwendig.

|  |
| --- |
| Abb. 11: Erstellung der *Stone*-Objekte in einem dreidimensionalen Array |
| for(int x = 0; x < stone.length; x++)  for(int y = 0; y < stone.length; y++)  for(int z = 0; z < stone.length; z++)  {  stone[x][y][z] = new Stone (x, y, z);  } |
| Durch die drei Schleifen werden 27 *Stone*-Objekte in das dreidimensionale Array gespeichert. |

1. Im Weiteren ist es nicht möglich, *Stone*-Objekte in mehrere *TransformGroups* zu setzten, da die Java 3D Bibliothek nicht unterstützt, dass ein *Child*-Objekt mehr als ein *Parent*-Objekt hat. Aus diesem Grund habe ich mich entschieden, jedes einzelne *Stone*-Objekt in eine eigene *TransformGroup* zu setzen. Um im Verlauf des Programms die *TransformGroup* und das *Stone*-Objekt ansprechen zu können, habe ich beide in einem Array abgelegt. Ich habe einen dreidimensionalen Array gewählt, weil dadurch die Position eines *Stone*-Objekts schnell ermittelt werden kann.

|  |
| --- |
| Abb. 12: Erster Ansatz zur Erstellung der *Stone*-Objekte |
| for(int x = 0; x < stone.length; x++)  for(int y = 0; y < stone.length; y++)  for(int z = 0; z < stone.length; z++)  {  stone[x][y][z] = new Stone (x - 1, y - 1, z - 1);  stoneTransform[x][y][z] = new TransformGroup();  stoneTransform[x][y][z].addChild(stone[x][y][z].getTransformGroup());  objTransform.addChild(stoneTransform[x][y][z]);  } |
| Bereits bei der Erstellung der *Stone*-Objekte wird der *SceneGraph* so aufgebaut, dass später Rotationen möglich sind. |

1. Nach einer Rotation eines *Stones*-Objekts wird sein aktueller Punkt nicht als neuer Ursprung gespeichert. Daher konnte ich immer nur eine Drehung ausführen. Zuerst löste ich dieses Problem, indem ich den Winkel bei jeder Drehung manuell erhöhte. Später kam ich zum Ergebnis, dass ich eine neue Bewegung mit den bereits ausgeführten Rotationen multiplizieren muss. Mit diesen Erkenntnissen habe ich dem Algorithmus zur Erstellung der *Stone*-Objekte ein *Transform3D*-Objekt hinzugefügt. Dieses wird auch als Array gespeichert, um es im Programm-Verlauf wieder aufzurufen.

|  |
| --- |
| Abb: 13: Überarbeiteter Algorithmus, welcher die *Stone*-Objekte erstellt. |
| for(int x = 0; x < stone.length; x++)  for(int y = 0; y < stone.length; y++)  for(int z = 0; z < stone.length; z++)  {  stone[x][y][z] = new Stone (x, y, z);  stoneTransform[x][y][z] = new TransformGroup();  stoneTransform[x][y][z].addChild(stone[x][y][z].getTransformGroup());  rotate[x][y][z] = new Transform3D();  objTransform.addChild(stoneTransform[x][y][z]);  } |
| Jedes *Stone*-Objekt erhält sein eigenes *Transform3D*-Objekt. Dadurch bleibt jedes *Stone*-Objekt unabhängig. |

1. Da jedes *Stone*-Objekt ein objekteigenes Koordinatensystem besitzt, speichere ich die Koordinatenausrichtung bei der Erstellung der *Stone*-Objekte.

|  |
| --- |
| Abb. 14: Algorithmus, welcher die Koordinatenausrichtung beim Erstellen des *Stone*-Objekts auf den Anfangswert setzt. |
| for(int x = 0; x < stone.length; x++)  for(int y = 0; y < stone.length; y++)  for(int z = 0; z < stone.length; z++)  {  stone[x][y][z] = new Stone (x, y, z);  stone[x][y][z].setLocalCord(1, 5);  stoneTransform[x][y][z] = new TransformGroup();  stoneTransform[x][y][z].addChild(stone[x][y][z].getTransformGroup());  rotate[x][y][z] = new Transform3D();  objTransform.addChild(stoneTransform[x][y][z]);  } |
|  |

1. Wenn der Benutzer eine Rotation ausführen will, drückt er einen der sechs Knöpfe. Dadurch wird in der Rubiks\_Cube-Klasse eine der sechs Rotationsmethode ausgewählt. Jede dieser ruft die getTransform() aus der Stone-Klasse auf, in welcher nacheinander drei Selektionen durchgeführt werden. Bei der ersten Selektion wird ermittelt, um welche Achse rotiert werden soll. Bei den nächsten zwei Selektionen werden die Ausrichtung der x- und y-Koordinate ermittelt. Durch diese drei verschachtelten Selektionen entscheidet das Programm bei jedem einzelnen *Stone*-Objekt, um welche Achse es gedreht werden muss. Zum Schluss werden die Koordinatenausrichtungen angepasst, um bei der nächsten Rotation eine richtige Selektion durchzuführen.

**Begriff Koordinatenausrichtung**

Jedes *Stone*-Objekt hat sein objekteigenes Koordinatensystem. Die Beziehung zwischen diesem und dem Koordinatensystem des *Locale* bezeichne ich als Koordinatenausrichtung. Dabei gebe ich an, in welche Richtung die x- und y-Koordinate im Koordinatensystem des *Locale*‘s ausgerichtet sind. Die z-Koordinate wird nicht überprüft, weil von den beiden andreren Koordinaten eindeutig auf die z-Koordinate geschlossen werden kann. Bei der Zuweisung benutze ich Zahlen. In der Tabelle ist zu sehen, welcher Koordinate des *Locale‘s* welche Nummer zugeteilt wird.

|  |  |  |  |
| --- | --- | --- | --- |
| Tab. 1: Beziehung der objekteigenen Koordinatensysteme im Koordinatensystem des *Locale‘s* | | | |
|  | **Zahl** | **Richtung im Koordinatensystem des *Locale‘s*** |  |
| 1 | Nach rechts (positive x-Achse) |
| 2 | Nach hinten (negative z-Achse) |
| 3 | Nach links (negative x-Achse) |
| 4 | Nach vorne (positive z-Achse) |
| 5 | Nach oben (positive y-Achse) |
| 6 | Nach unten (negative y-Achse) |
| Die Zahlen geben an, in welche Richtung die objekteigenen x- und y-Koordinaten zum Koordinatensystem des *Locale’s* stehen. | | | |

**Dokumentation für Endbenutzerinnen und Endbenutzer**

Nach dem Start des Programms haben sie folgende Möglichkeiten:

* Den Würfel können sie mit der Maus um den Mittelpunkt rotieren.
* Sie können einen der sechs Knöpfe drücken. Jeder der Knöpfe ist mit einer Farbe angeschrieben. Die Farbe bezieht sich auf den Mittelstein der Wand. Bei jedem Klick wird die Wand im Gegenuhrzeigersinn um 90° gedreht. Die verschiedenen Drehungen können beliebig oft und in beliebiger Reihenfolge durchgeführt werden.
* Unter dem Menüpunkt „Datei“ können sie den Würfel zurücksetzen. Dadurch wird der Würfel wieder in die Ausgangslage gesetzt.
* Um das Programm zu schliessen, genügt ein Mausklick auf das „Kreuz“ oben rechts.

**Ergebnis**

Die Arbeit hat mir einen Einblick in die Java 3D Bibliothek ermöglicht. Dabei habe ich meine Fertigkeiten in der Programmiersprache „Java“ anwenden und verbessern können.

Durch das Verfassen des Berichts habe ich mich intensiv mit der Erstellung von wissenschaftlichen Arbeiten auseinandergesetzt. Während dieser Zeit habe ich Methoden entwickelt, wie es mir persönlich am Leichtesten fällt eine längere Arbeit zu schreiben.

Neben Lernfortschritten in der Informatik, habe ich ebenfalls neue Menschen kennengelernt. Obwohl keiner dieser Menschen die Java 3D Bibliothek kennt, konnten sie mir immer wieder hilfreiche Hinweise geben.

Wenn ich meine Ziele betrachte, die ich in der Disposition (Anhang 2) notierte, stelle ich erfreut fest, dass ich sie erreicht habe: Die Drehung des Würfels im Raum ist möglich und jede Wand kann individuell gedreht werden.

An meinem Programm gibt es dennoch einige Punkte, die mir als verbesserungswürdig erscheinen: Die Vereinfachung der Benutzersteuerung und eine grafische Aufwertung des Würfels. Ich würde es zum Beispiel als benutzerfreundlicher empfinden, wenn man die Wände mit der Maus rotieren könnte und keine Knöpfe mehr bräuchte. Zusätzlich könnte mit dem Hinzufügen von Lichter und Schatten der Würfel etwas natürlicher aussehen.

Die Arbeit hat mir grossen Spass gemacht. Ich bin stolz, dass ich bis zum Schluss durchgehalten habe und meine Ziele so erfolgreich erfüllt habe. Gut gefallen hat mir ebenfalls, dass ich mehrere hilfsbereite Personen gefunden haben, die ihre wertvolle Zeit für mich aufgebracht haben.

Wenn ich nun auf die Arbeit zurückbliche, sehe viele überwundene Herausforderungen und eine Menge Arbeit, die sich gelohnt hat. Nun freue ich mich auf die Präsentation der Arbeit und die Matura, an der ich dieses Jahr teilnehmen werde.

**Quellen**

**Literatur zum Rubik’s Cube**

Trajber, Josef. 1981. Der Würfel „Rubik’s Cube“. Falken Verlag. Niedernhausen.

**Zwei Bücher von Werner Randelshofer**

Bell, Douglas und Parr, Mike. 2002. Java for Students. Person Education Limited. Harlow.

Plenge Axel. Amiga 3D-Grafik und Animation. 1988. Markt & Tech Verlag. Haar bei München.

**Geschichte des Rubik’s Cube**

Pirzer, Markus. 1998. Kurze Geschichte des Würfels. [rubiks-zauberwuerfel.de/home.html?geschichte.html](http://www.rubiks-zauberwuerfel.de/home.html?geschichte.html). Atari Deutschland GmbH. Frankfurt (Stand 01.10.12)

**Geschichte von Java 3D**

Jachmann, Tobias. Grundlagen der Java 3DProgrammierung. [matse.rz.rwth-aachen.de/dienste/public/show\_document.php?id=7223](https://www.matse.rz.rwth-aachen.de/dienste/public/show_document.php?id=7223). Aachen (Stand 01.10.2012)

**Bilder für theoretischen Teil „Rubik’s Cube“**

selber erstellt mit dem Programm „Google SketchUp Version 8“

**Text und Bilder für theoretischen Teil „Java 3D Bibliothek“**

Java 3D API Version 1.5.0:

[download.java.net/media/java3d/javadoc/1.5.0/index.html](http://download.java.net/media/java3d/javadoc/1.5.0/index.html) (Stand 01.10.2012)

Java Tutorial 1

[cs.stir.ac.uk/courses/ITNP3B/Java3D/Tutorial/j3d\_tutorial\_ch1.pdf](http://www.cs.stir.ac.uk/courses/ITNP3B/Java3D/Tutorial/j3d_tutorial_ch1.pdf) (Stand 01.10.2012)

**Text für Praktischer Teil**

Java 3D API Version 1.5.0:

[download.java.net/media/java3d/javadoc/1.5.0/index.html](http://download.java.net/media/java3d/javadoc/1.5.0/index.html) (Stand 01.10.2012)

Java Tutorial 1-4

[cs.stir.ac.uk/courses/ITNP3B/Java3D/Tutorial/j3d\_tutorial\_ch1.pdf](http://www.cs.stir.ac.uk/courses/ITNP3B/Java3D/Tutorial/j3d_tutorial_ch1.pdf) (Stand 01.10.2012)

[cs.stir.ac.uk/courses/ITNP3B/Java3D/Tutorial/j3d\_tutorial\_ch2.pdf](http://www.cs.stir.ac.uk/courses/ITNP3B/Java3D/Tutorial/j3d_tutorial_ch2.pdf) (Stand 01.10.2012)

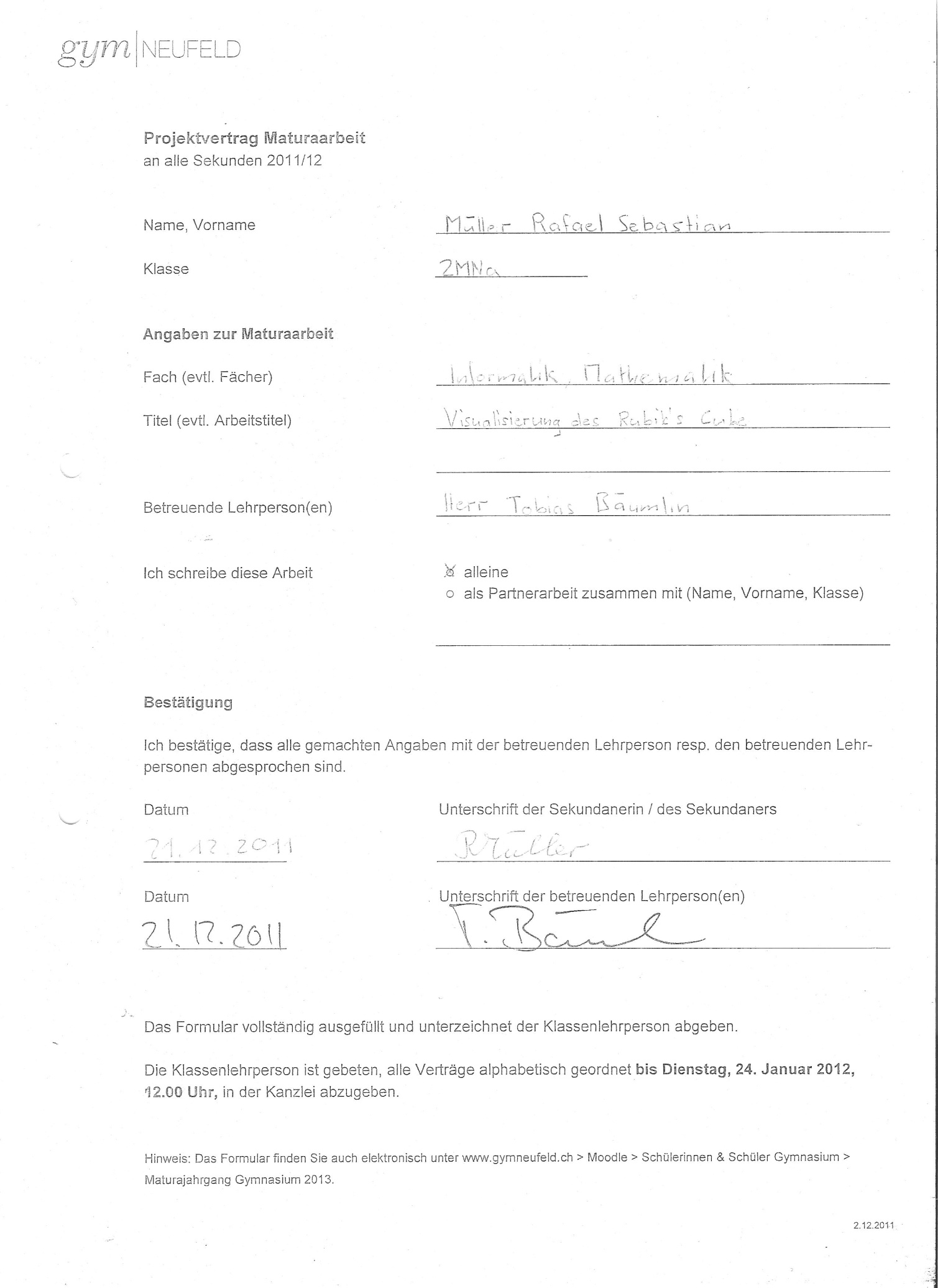
[cs.stir.ac.uk/courses/ITNP3B/Java3D/Tutorial/j3d\_tutorial\_ch3.pdf](http://www.cs.stir.ac.uk/courses/ITNP3B/Java3D/Tutorial/j3d_tutorial_ch2.pdf) (Stand 01.10.2012)

[cs.stir.ac.uk/courses/ITNP3B/Java3D/Tutorial/j3d\_tutorial\_ch4.pdf](http://www.cs.stir.ac.uk/courses/ITNP3B/Java3D/Tutorial/j3d_tutorial_ch2.pdf) (Stand 01.10.2012)

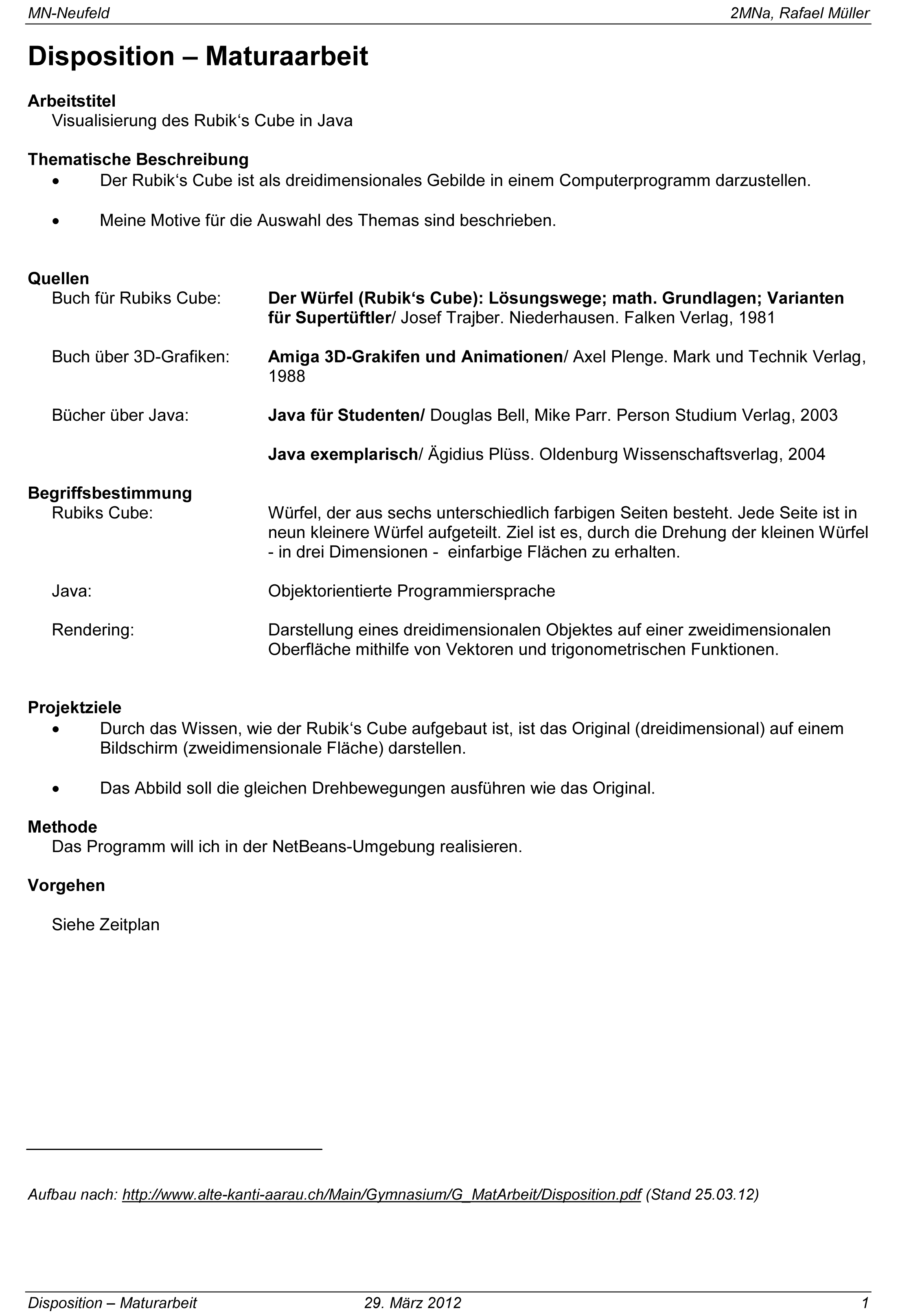
**Bilder für theoretischen Teil**

selber erstellte Screenshots, welche mit ich „Adobe Photoshop Element 9“ bearbeitet habe.

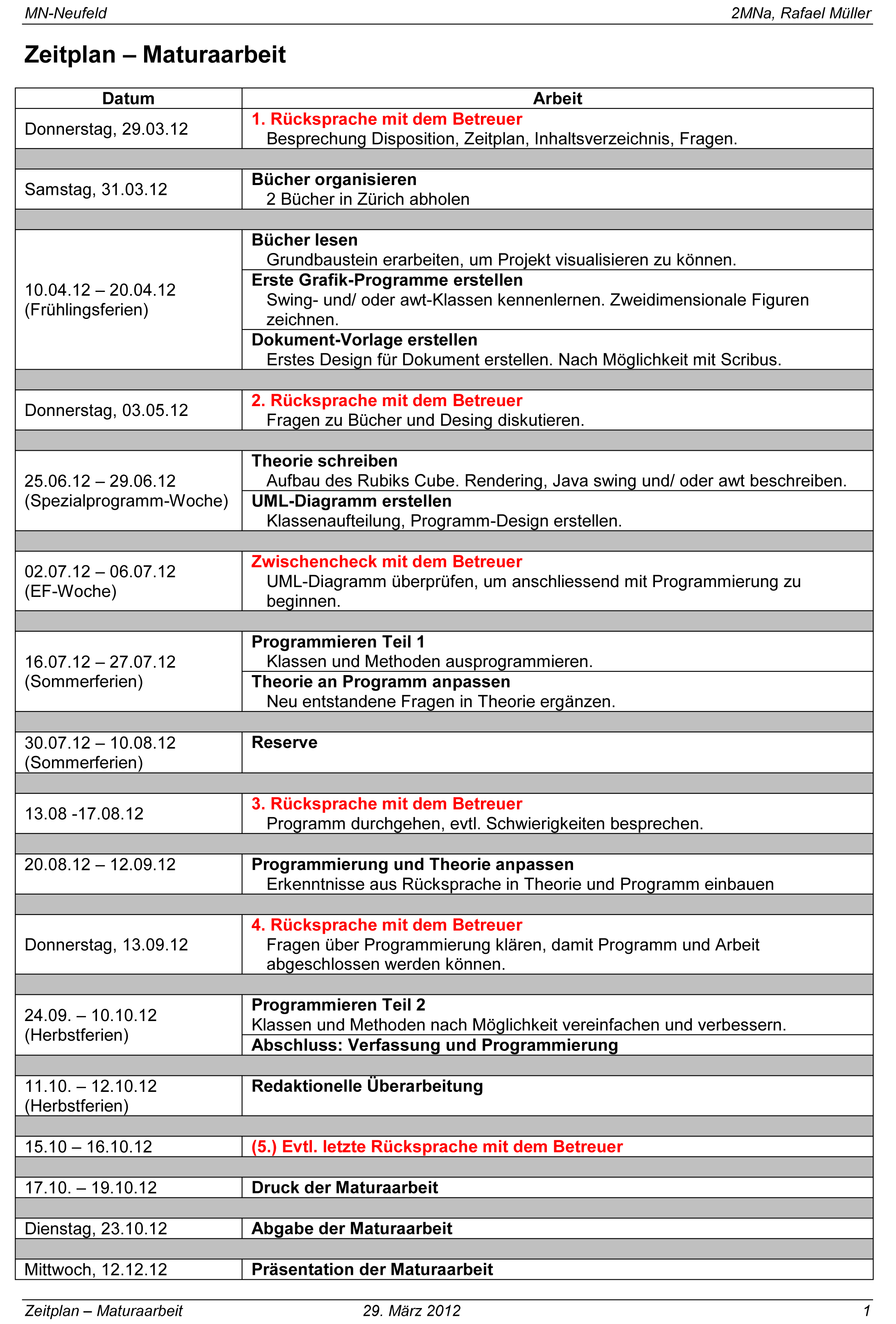
**Angang 1**



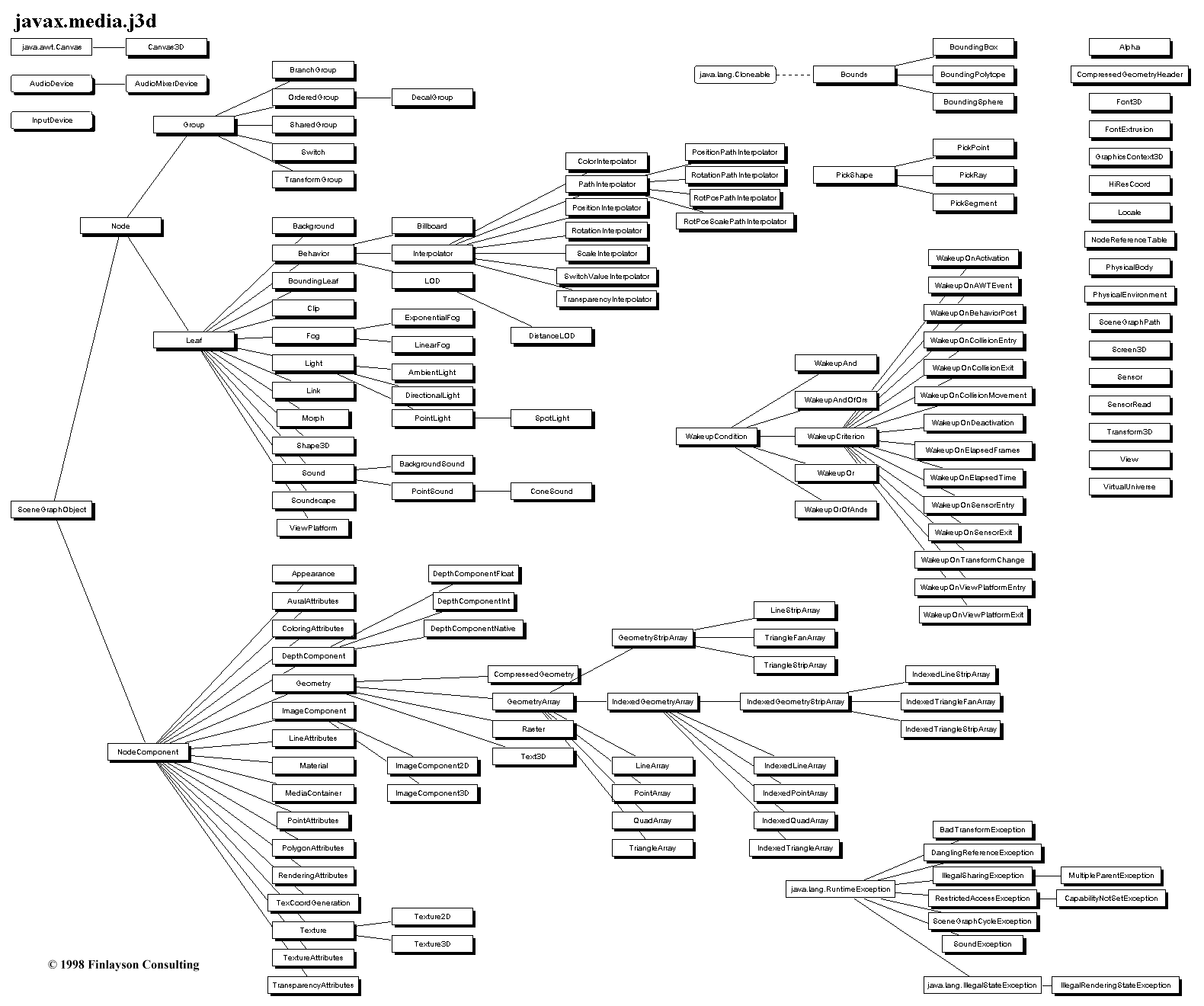
**Angang 2**



**Anhang 3**

****

**Anhang 4 (A3)**



**Anhang 5**

Programm (folgt noch)

**Selbständigkeitserklärung Maturaarbeit**

Ich erkläre hiermit, dass ich diese Abreit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Das Informationsblatt „Plagiatserkennung“ ist mir bekannt und somit die Konsequenzen eines Teil- oder Vollplagiats

Ort und Datum: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Unterschrift des Verfassers: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_