InfoSphere-Modul: Einstieg in die Computergrafik

Schriftliche Hausarbeit im Rahmen der Ersten Staatsprüfung, dem Landesprüfungsamt für Erste Staatsprüfungen für Lehrämter an Schulen vorgelegt von:

Tim Schellartz

Aachen, den 11. Januar 2012

Gutachter:
Prof. Dr. Schroeder

Betreuerin und Betreuer:
Nadine Bergner
Jan Holz

Lehrstuhl:

LuFGi9: Computer-Supported Learning Research Group

Inhaltsverzeichnis

1 Einleitung **4**
1.1 Motivation: Shaderprogrammierung als abgeschlossenes Thema 5
1.2 Abgrenzung zu anderen Ansätzen und Programmen 8
1.2.1 NeHe 8
1.2.2 Computer-Graphik in Schule 9
1.2.3 Shader-Maker 10

2 Darstellung der fachlichen Inhalte **13**
2.1 Objektdarstellung und Koordinatentransformationen 13
2.2 Farbdarstellung 18
2.3 Shaderprogramme und Phong-Beleuchtung 18
2.4 Stereoskopisches Sehen 21

3 Didaktische Analyse des gesamten Moduls **24**
3.1 Vorwissen 24
3.2 Auswahl der Inhalte 25
3.3 Didaktische Reduktion 27
3.4 Lernschritte und Lernziele 28
3.5 Gewählte Sozial- und Aktionsformen, Methoden sowie Medien 29
3.6 Ergebnissicherung 30

4 Das Modul **31**
4.1 Überblick 31
4.2 Verlaufsplan 32
4.3 Einstieg: Motivation und Zielvorstellung 32
4.4 Phase 1: Objektdarstellung und Koordinatentransformationen 33
4.4.1 Erarbeitungsphase zur Darstellung von Objekten in der Computergrafik 33
4.4.2 Erarbeitungsphase zur Modifikation von Objekten mittels Koordinatentransformationen 36
4.4.3 Sicherungsphase zu Koordinatentransformationen . . . 40
4.4.4 Präsentation Shadermanager und OpenGL Shading Language 41
4.4.5 Übungs- und Erarbeitungsphase durch Animation von Objekten mittels Vertexshaderprogrammierung 45

4.5 Phase 2: Farbdarstellung und Normalenvektoren 47
4.5.1 Erarbeitungsphase zur RGBA-Farbdarstellung 47
4.5.2 Übunsphase durch Simulation eines Comiclooks im Fragmentshader 50
4.6 Phase 3: Anwendungsphase durch Programmierung komplexerer Shaderprogramme 53
4.6.1 Entwicklung der Phase 53
4.6.2 Sicherungsaufgabe zur lokalen Beleuchtung 54
4.6.3 Abschlussaufgabe zum stereoskopischen Sehen 56

5 Zusammenfassung 60

6 Anhang: Arbeitsmaterialien mit Lösungsvorschlägen 61
6.1 Arbeitsblätter 61
6.1.1 Arbeitsblatt Objektdarstellung 62
6.1.2 Sicherungsblatt Koordinatentransformationen 64
6.1.3 Arbeitsblatt Animation 66
6.1.4 Arbeitsblatt Farben 69
6.1.5 Arbeitsblatt Comiclook 71
6.1.6 Arbeitsblatt Phong-Beleuchtungsmodell 74
6.1.7 Arbeitsblatt 3D-Stereo 78
6.2 Infoblätter 81
6.2.1 Verlaufsplan für Schülerinnen und Schüler 82
6.2.2 Infoblatt Mathematik 83
6.2.3 Infoblatt Informatik 84
6.2.4 Infoblatt für Betreuer: Shadermanager und GLSL . . 86
6.3 Materialliste 90

Literaturverzeichnis 91

1 Einleitung

Diese Staatsexamensarbeit dokumentiert die Erstellung eines Moduls für das Schülerlabor Informatik an der RWTH Aachen, genannt InfoSphere[1]. Das Schülerlabor bietet den teilnehmenden Schülerinnen und Schülern aller Schulformen die Möglichkeit in mehrstündigen Einheiten, genannt Modulen, praktische Probleme und Themenstellungen der Informatik zu erarbeiten. Dieses Modul bietet beispielsweise einen Einstieg in die Computergrafik. Die Erstellung eines Moduls zur Computergrafik macht deshalb Sinn, weil sie für viele Menschen im heutigen Alltag eine nicht zu vernachlässigende Rolle einnimmt. So wird man früh morgens beim Blick auf das Handy mit animierten, grafischen Oberflächen konfrontiert oder nachmittags bei Nachrichtensendungen, welche üblicherweise vor einer tristen, einfarbigen Wand gedreht und dann mit grafischen Effekten versehen werden. Das Modul versucht die Schülerinnen und Schüler in ihrer Lebenswirklichkeit abzuholen, wozu aktuelle technische Trends aufgegriffen werden, zum Beispiel Ambilight oder generell die Farbdarstellung bei Flachbildschirmen und die als extrinsische Motivation gedachte Abschlussaufgabe zum stereoskopischen Sehen. Dabei lernen die Schülerinnen und Schüler ganz nebenbei eine algorithmische Herangehensweise an Probleme kennen.

Mein oberstes Ziel bei der Erstellung des Moduls war es nicht, ein tiefgründiges theoretisches Wissen über Computergrafik zu vermitteln. Viel mehr soll den Schülerinnen und Schülern ein in der Schule und im Studium, falls man sich als Student nicht freiwillig dieser Thematik annimmt, vernachlässigtes Thema auf ansprechende Weise näher bringen. Dabei darf man ansprechend nicht mit simpel verwechseln, denn dieses Modul ist rein thematisch eines der schwierigeren Module, da mathematische und informatische Vorkenntnisse benötigt und verwendet werden, welche Schulstoff der Oberstufe sind - also sie werden im Lehrplan beziehungsweise in den Zentralabiturvorgaben gefordert. Gemeint ist, dass das theoretische Wissen in praktischen Arbeiten verpackt und mit schnellen Erfolgserlebnissen vermittelt wird. Insbesondere sind zwei Abschnitte des Moduls geplant, in denen die Schülerinnen und Schüler vom Computer losgelöst arbeiten um durch die realen Erfahrungen ein größeres Erfahrungsspektrum zu erhalten. Das Modul bietet nur einen Einstieg oder Einblick, für eine komplette Behandlung fehlt im Schülerlabor schlicht die Zeit. Aber mit den hier erworbenen Kenntnissen steht einer eigenständigen, vertiefenden Beschäftigung nichts mehr im Wege.

[1] weiter Informationen unter `http://schuelerlabor.informatik.rwth-aachen.de/`

Nun zur Struktur der Ausarbeitung. Die Arbeit beginnt zunächst mit einer genauen Begründung, warum welche Themen behandelt sowie andere weggelassen werden und stellt den Ansatz dieses Moduls im Vergleich zu anderen Projekten in der Computergrafik dar. Um den Leser auf ein fachliches Grundgerüst zu stellen folgt dann bereits eine Darstellung der fachlichen Inhalte, die einen kurzen Einblick in die behandelte Thematik geben soll. Im Anschluss folgt eine didaktische Grobanalyse in der Fragen zur gesamten Modulplanung beantwortet werden sollen:

- Welches Vorwissen wird benötigt?
- Welche Inhalte werden warum behandelt?
- Wie wurde der Stoff didaktisch reduziert und warum?
- Was können die Schülerinnen und Schüler nach dem Modul? Oder anders ausgedrückt: Welche Lernziele werden verfolgt?
- Welche Methodik, welche Medien und welches Material wird verwendet?
- Wie wird das Erlernte vertieft?

Daran schließt sich Kapitel 4 an, welches chronologisch das ganze Modul vorstellt und an entsprechende Stelle Begründungen für getroffene Entscheidungen liefert, nachdem der vorherige Abschnitt bereits die groben Entscheidungen behandelt hat. Dabei werden insbesondere die eigens für das Modul erstellten Arbeitsmaterialien und Programme vorgestellt. Dort finden sich auch Hinweise zur Vorgehensweise für die Betreuer des Moduls, welche diese Staatsexamensarbeit in aufbereiteter Form als Modulhandbuch erhalten werden.

Der Schluss bietet dann einen kompakten Überblick über die erstellten Arbeitsblätter inklusive Lösungvorschläge sowie Infoblätter für die Betreuerinnen und Betreuer des Moduls oder die teilnehmenden Schülerinnen und Schüler.

1.1 Motivation: Shaderprogrammierung als abgeschlossenes Thema

Dieser Abschnitt klärt die Frage, warum die behandelten Themen in einem InfoSphere Modul sinnvoll sind und andere vernachlässigt werden und warum insbesondere das Thema Shaderprogrammierung einen so hohen Stellenwert genießt.

Eine vollständige und abgeschlossene Behandlung der Computergrafik innerhalb eines mehrstündigen Moduls ist nicht machbar und eine Eingrenzung auf wenige Themen ist unumgänglich, was sich bereits in der Wahl des Titels

Einstieg in die Computergrafik andeutet. Als Schwerpunkt wurde die Shaderprogrammierung gewählt, denn diese ist ein

- mächtiges Werkzeug und
- in sich abgeschlossen.

Ein mächtiges Werkzeug ist sie deshalb, da Shaderprogramme für eine Vielzahl aktueller Grafikeffekte notwendig sind, wie die Beispiele in Abschnitt 2.3 zeigen. Neben den Beispielen findet sich dort eine genauere Erläuterung was Shaderprogramme sind. In den vergangenen Jahrzehnten hat sich die Computergrafik stark weiterentwickelt und der Trend geht weg von der konfigurierbaren fixed-function pipeline, das bedeutet dem in der Hardware fest vorgebauten Ablauf auf der Grafikkarte, welcher nur über das Setzen von Parametern zu beeinflussen ist, hin zu programmierbaren Hardwareeinheiten, den Shadern, welche die wichtigste Art darstellen um auf den Grafikprozessor (Graphics Processing Unit - kurz GPU) zugreifen zu können. In Zukunft dürfte dieser Trend weiter gehen und der Zugriff auf die GPU wird immer flexibler, was sich an den relativ neuen Tessellation-Shadern[2] zeigt. Somit lernen die Schülerinnen und Schüler hier ein langlebiges Thema kennen [1].

Aber noch viel wichtiger als die Mächtigkeit erscheint mir die Abgeschlossenheit der Technologie. Damit meine ich, dass es möglich ist Shaderprogramme zu erstellen, ohne sich mit dem zugrunde liegenden, eventuell sehr komplexen OpenGL-Programm zu beschäftigen. Das heißt, es ist prinzipiell egal, ob das Programm, welches das Shaderprogramm ausführt, in Java oder C++ geschrieben wurde, oder wie die Datenstruktur der 3D-Objekte aufgebaut ist. Oder einfacher gesagt, ein Shaderprogramm, welches innerhalb des Moduls erstellt wurde, kann in jedem OpenGL-Programm angewendet werden und es funktioniert, nach eventueller Anpassung der Schnittstelle, welche die Eingaben an das Shaderprogramm enthalten, auf die gleiche Weise. Durch diese Abgeschlossenheit und da Shaderprogramme sehr schnell ausgeführt werden und somit aus wenigen Befehlen bestehen müssen, zeichnet sich die Programmierung dieser als verhältnismäßig einfach ab. Insbesondere für Schülerinnen und Schüler, die über Kenntnisse in C/C++ oder Java verfügen, stellt das Erlernen der OpenGL Shading Language, im folgenden GLSL abgekürzt, vermutlich nur eine kleine Hürde dar, denn die Syntax ist sehr ähnlich zur C-Syntax.

Die weiteren behandelten Themen

- Objektdarstellung als Netze,
- Koordinatentransformationen und

[2] Durch Tessellation kann die Anzahl der Polygone dynamisch bestimmt werden, zum Beispiel werden Objekte nah am Betrachter sehr detailliert und entfernte Objekte sehr grob dargestellt, um Rechenleistung zu sparen. Folgendes Video verdeutlicht den Effekt: `http://www.youtube.com/watch?v=bkKtY2G3FbU`

- Farbcodierung über RGBA-Werte[3]

(siehe Kapitel 2) scheinen sich auf den ersten Blick als Vorbereitung für die Shaderprogrammierung zu erklären, was sicherlich ein Grund für die Behandlung dieser Themen ist, aber als einziger Grund zu kurz greifen würde. Wer tiefer in die Thematik blickt sieht, dass diese Themen bedeutende und langlebige Ideen der Computergrafik sind, denn die *Objektdarstellung als Netze* ist der weit verbreitete Standard, wie sich schon durch die aktuellen 3D-Grafikkarten zeigt, welche spezielle Maschinen zur Verarbeitung und Darstellung von Dreiecksnetzen beziehungsweise Polygonnetzen sind und sich in nahezu jedem Computer finden lassen. *Koordinatentransformationen* sind bedeutend, da sie den Schülerinnen und Schülern einen Verwendungszweck von Matrizen, abseits der Lösung von linearen Gleichungssystemen mit dem Gauß-Algorithmus, zeigen sollen. Koordinatentransformationen haben durch ihre ausgezeichnete Darstellung als Matrizen einige Vorteile für die Computergrafik wie genauer im Abschnitt 2.1 gezeigt wird. Die Farbcodierung über Rot-Grün-Blau-Alpha-Werte zeigt die direkte Verknüpfung der Lebenswelt der Schülerinnen und Schüler mit der Computergrafik, denn in der heutigen Zeit verbringt der Mensch viel Zeit vor Fernsehern und Monitoren, welche über RGBA-Werte angesteuert werden. Die Vielzahl anderer Farbsysteme, wie Hue-Satuaration-Value (HSV) oder Cyan-Mangenta-Yellow-Black (CMYK) macht sichtbar, dass die Idee der RGBA Codierung nicht grundlegend für die Computergrafik ist aber da sich diese Farbsysteme ineinander überführen lassen, sollte zumindest eine Möglichkeit zur Farbdarstellung gelernt werden um Objekte zu färben. Die Wahl fällt dann zugunsten der RGBA-Farbdarstellung aus, da diese weit verbreitet ist und direkt von OpenGL und GLSL unterstützt wird. Diese Darstellung ist zunächst nicht sehr intuitiv, aber genau deshalb wird sie im Modul angesprochen um eine Intuition im Umgang mit RGBA-Werte zu vermitteln.

Es sei noch erwähnt, dass die Wahl der Programmiersprache GLSL nicht willkürlich ist. Es existieren verschiedene Programmiersprachen zur Erstellung von Shaderprogrammen, zum Beispiel HLSL zur Erstellung von Shaderprogrammen für DirectX Anwendungen. Mit der Entscheidung für die Programmiersprache GLSL sollte den Schülerinnen und Schülern die Möglichkeit geben werden, falls das Modul ihr Interesse geweckt hat, sich selber weiter damit zu beschäftigen ohne sich in Unkosten zu stürzen oder über ein spezielles System verfügen zu müssen. Denn OpenGL und GLSL sind beide kostenlos und plattformunabhängig. Obschon die Programmiersprache GLSL an sich nicht allgemeinbildend ist, können die erlernten Konzepte leicht auf andere Programmiersprachen übertragen werden, da diese meist C-ähnlich aufgebaut sind. Somit ist das Lernen der Programmiersprache zumindest im Bereich der Shaderprogrammierung sinnvoll und übertragbar [11].

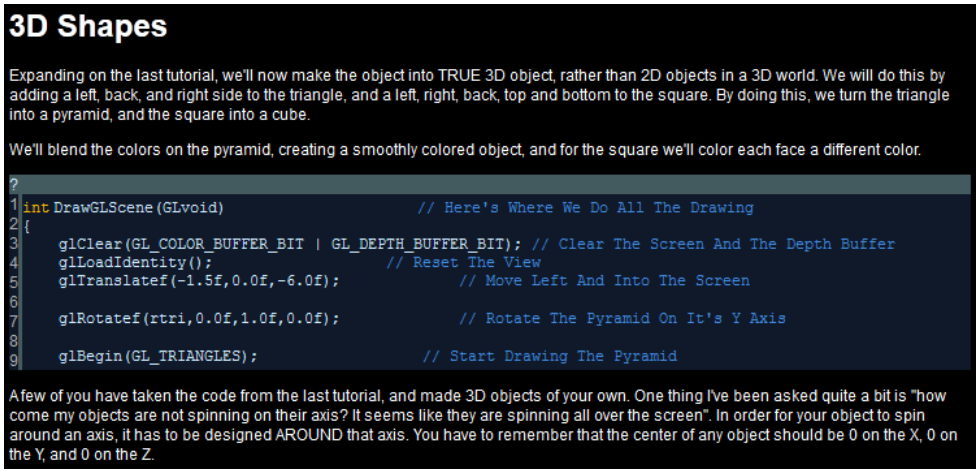
[3] RGBA steht für Rot-Grün-Blau-Alpha, wobei Alpha beschreibt, wie gut durch ein Objekt durchgesehen werden kann.

1.2 Abgrenzung zu anderen Ansätzen und Programmen

Die erstellten Arbeitsblätter und Programme, den Modulablauf sowie die Ideen des Moduls werden in diesem Kapitel mit anderen Arbeiten verglichen und von diesen abgegrenzt. Weiter wird begründet, welche Entscheidungen zur Erstellung eigener Programme geführt haben. Im Folgenden werden die einzelnen Abschnitte jeweils eine andere Arbeit betrachten und Ähnliches hervorheben sowie Unterschiede aufzeigen.

1.2.1 NeHe

Als Erstes sei an dieser Stelle eine Website genannt: http://nehe.gamedev.net/. Auf dieser Internetseite findet der interessierte Leser alles was er für den erfolgreichen Einstieg in die OpenGL-Programmierung bzw. die Grafikprogrammierung braucht. Nach Bearbeitung der NeHe-Tutorials ist man in der Lage, komplette eigene OpenGL-Programme zu bauen und Szenen darzustellen. Die wesentlichen Techniken zur Grafikdarstellung, wie zum Beispiel das Zeichnen einfacher Formen wie einem Würfel, die Texturierung von Oberflächen und die Erstellung und Verwendung von Transformationen werden erklärt. Im Gegensatz dazu ist man nach dem Ende des Moduls „nur“ in der Lage eigene Shaderprogramme zu erstellen und kennt grundlegende Themen der Computergrafik. Diese Diskrepanz erklärt sich durch den Unterschied bei der bloßen Menge der Inhalte. Das Modul ist für knapp 4-5 Stunden ausgelegt, wobei die Thematik ansprechend und motivierend gewählt wurde, hingegen kann man für die Bearbeitung aller Nehe-Tutorials mehrere Wochen einplanen, in denen man die komplette Erstellung eigener OpenGL-Programme erlernt.



3D Shapes

Expanding on the last tutorial, we'll now make the object into TRUE 3D object, rather than 2D objects in a 3D world. We will do this by adding a left, back, and right side to the triangle, and a left, right, back, top and bottom to the square. By doing this, we turn the triangle into a pyramid, and the square into a cube.

We'll blend the colors on the pyramid, creating a smoothly colored object, and for the square we'll color each face a different color.

```
int DrawGLScene(GLvoid)                         // Here's Where We Do All The Drawing
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear The Screen And The Depth Buffer
    glLoadIdentity();                       // Reset The View
    glTranslatef(-1.5f,0.0f,-6.0f);             // Move Left And Into The Screen

    glRotatef(rtri,0.0f,1.0f,0.0f);             // Rotate The Pyramid On It's Y Axis

    glBegin(GL_TRIANGLES);                      // Start Drawing The Pyramid
```

A few of you have taken the code from the last tutorial, and made 3D objects of your own. One thing I've been asked quite a bit is "how come my objects are not spinning on their axis? It seems like they are spinning all over the screen". In order for your object to spin around an axis, it has to be designed AROUND that axis. You have to remember that the center of any object should be 0 on the X, 0 on the Y, and 0 on the Z.

Abbildung 1.1: Auszug eines NeHe-Tutorials [4].

Ein weiterer Unterschied findet sich bei den zugrunde liegenden Konzepten. NeHe stellt häufig eine Vielzahl von Informationen bereit, welche gelesen werden sollen, somit scheint die Webseite auf Einzelarbeit ausgelegt zu sein, wie man in Abbildung 1.1 sehen kann. Ebenso arbeiten die Leserinnen und Leser der Seite keine Aufgaben durch, sondern bekommen die Ergebnisse direkt mitgeliefert. Das Modul hingegen führt die Schülerinnen und Schüler über Aufgaben und Eigenleistungen in Partnerarbeit an den Stoff heran und versucht möglichst wenig frontal zu präsentieren. Dies führt natürlich dazu, dass im Modul nicht entsprechendes Detailwissen erworben wird, zum Beispiel lasse ich die Texturierung bewusst außen vor, die bei NeHe ziemlich früh thematisiert wird.

Dennoch sind auch Parallelen erkennbar. Das Modul wie auch die Webseite versuchen den Inhalt möglichst interessant zu gestalten und den Teilnehmerinnnen und Teilnehmern beziehungsweise Leserinnen und Lesern mit Belohnungen zu motivieren. So gehen beide in relativ kleinen Schritten vor um schnell erste Erfolge zu ermöglichen. Die Thematik wird in beiden Herangehensweisen ansprechend aufbereitet, da man ziemlich schnell visuelle Effekte beobachten kann. Ähnlich ist auch, dass das Modul sowie die Webseite auf Plattformunabhängigkeit ausgelegt sind. Dies ist im Modul leicht realisierbar, da die GLSL an sich schon plattformunabhängig ist und die zugrunde liegende Technik nicht thematisiert wird. NeHe beschäftigt sich aber mit den kompletten Programmen und muss auf verschiedenste Programmieroberflächen und -sprachen reagieren. Dies tut die Community auch, indem sie den Quellcode für alle gängigen Programmiersprachen und Systeme bereitstellt.

Da die Webseite nur auf allgemeinen Programmierkenntnissen aufbaut, würde ich den interessierten Schülerinnen und Schüler diese Webseite als erste Anlaufstelle für die weitere Beschäftigung mit diesem Thema empfehlen.

1.2.2 Computer-Graphik in Schule

„Computer-Graphik in Schule“ ist eine von Gabriel Zachmann entwickelte Ideensammlung, welche seiner Meinung nach in der Oberstufe eingesetzt werden kann. Die Inhalte reichen dabei weit über die des hier beschriebenen Moduls hinaus, es werden Shader, einfache Bildsynthesen und -verarbeitungen, bis hin zu Raytracing vorgestellt und passende Materialien sowie Werkzeuge bereitgestellt. So konnte ich darüber im Laufe meiner Recherchen den Shader-Maker - ein Programm zur Erstellung von Shaderprogrammen - finden, welcher im nächsten Abschnitt durchleuchtet wird. Meiner Meinung nach dringt die Ideensammlung für die Verwendung im Schulbereich sehr tief in die Materie ein, was mich zu dem Schluss kommen lässt, dass man nur bestimmte Themen für den Unterricht wählen sollte und nicht die ganze Sammlung in den Unterricht nehmen kann.

Die Ideensammlung hat den Nachteil, dass sie nur eine Darstellung verschiedener interessanter Inhalte ist und keine didaktischen Konzepte oder Unter-

richtsbeispiele bereit hält. Demnach muss die Lehrkraft, welche die Themen in den Unterricht einbinden möchte, die Unterrichtsreihe zum großen Teil selbst entwerfen und kann auf der Website nur hilfreiche Erklärungen finden, welche offenbar aus einer Vorlesung zur Computergrafik stammen und somit ein gewisses Vorwissen bedingen. Das ist der entscheidende Unterschied zum Modul, welches ein nicht so breites Feld der Computergrafik aufzeigt, aber komplett vorbereitet und an das Vorwissen der Schülerinnen und Schüler angepasst wurde um es an einem Tag im Schülerlabor Informatik erfolgreich abschließen zu können.

Gemeinsamkeiten zwischen dem Modul und der Ideensammlung lassen sich bei der Intention erkennen. Das Thema Computergrafik bereits in der Schule zu thematisieren soll die Schülerinnen und Schüler für diesen Wissenschaftsbereich motivieren. Zachmann macht obendrein eine interessante Aussage zur Begründung für die Beschäftigung mit der Computergrafik, denn diese schult algorithmisches Denken, welches auch im Informatiklehrplan und dem Zentralabitur gefordert wird (vgl. [12, 7, 11]).

1.2.3 Shader-Maker

Der Shader-Maker ist ein Programm zur Erstellung und Testung von Shaderprogrammen. Folgendes Zitat erklärt die Funktionsweise:

> „It provides the basic functionalities of a shader editor, such that students can get started with writing their own shaders as quickly as possible. This includes: editors for vertex, fragment, and geometry shaders; syntax highlighting in the GLSL editors; interactive editing of uniform variables and light source parameters; pre-defined simple shapes (torus et al.) and a simple OBJ loader; and a few more.“ [3]

Durch dieses Programm soll demnach der Einstieg in die Shaderprogrammierung vereinfacht und vom eigentlichen Programm gelöst werden. Die Oberfläche des Shader-Maker wirkt meiner Meinung nach aufgeräumt aber überladen wie man in Abbildung 1.2 sehen kann.

Demnach wäre der Shader-Maker ein gutes Werkzeug für das Modul gewesen. Allerdings besitzt das Programm nicht nur Vorteile sondern auch Nachteile, welche in folgender Tabelle gegenüber gestellt werden.

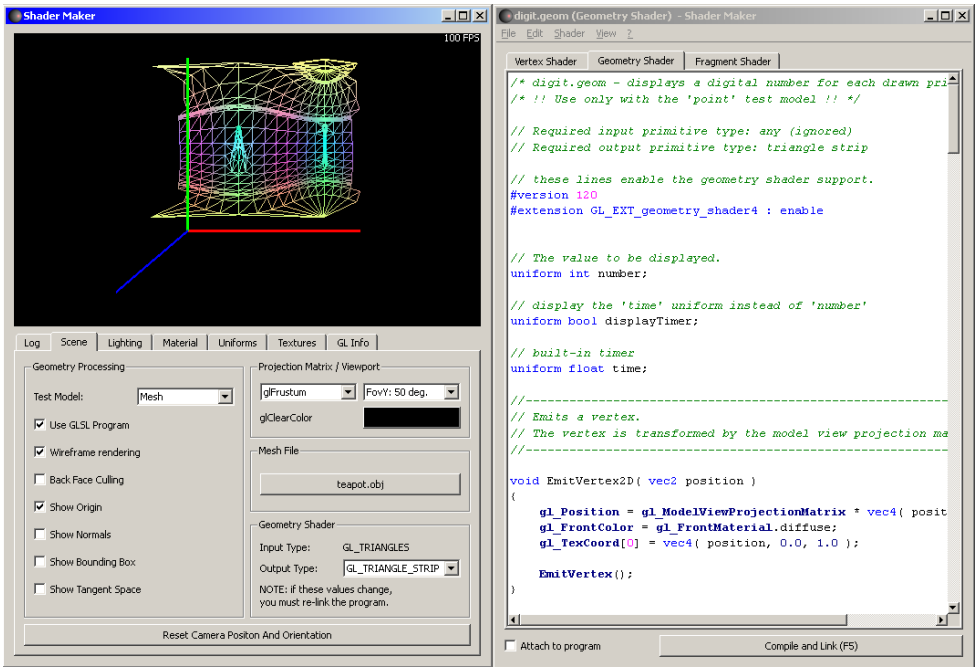


Abbildung 1.2: Oberfläche Shader-Maker [3].

Vorteile	**Nachteile**
• Shaderprogrammierung vom zugrunde liegenden Programm getrennt • Eingebauter Editor • Syntax-Hervorhebung • Direkte Testung der Shader im eingebauten Ausgabefenster • Verwaltung mehrerer Shader • Verschiedene Objekte / Szenen • Variablen aus Benutzeroberfläche setzbar • Plattformunabhängig	• Komplexe Oberfläche • Verschachtelte Menüs • Mehr Informationen und Möglichkeiten als notwendig für das Modul • Erfordert Qt 4.3[4]

[4] Qt ist eine Grundstruktur zur Erstellung von User-Interfaces. Siehe auch `http://qt.nokia.com/products/`

Für den Shader-Maker sprechen demnach viele Vorteile. Aber die komplexen Menüs mit der Vielzahl an Möglichkeiten zur Modifikation sind meiner Meinung nach für den Einstieg mit einer Schulklasse ungeeignet beziehungsweise kosten zu viel Aufmerksamkeit und somit Zeit. Aus diesem Grund habe ich mich für die Erstellung eines eigenen Programms entschieden, welches möglichst wenig Einstellungsmöglichkeiten bereitstellt. Somit büßt das Programm an Funktionalität ein, aber im Modul sollen die Schülerinnen und Schüler schließlich eine Programmiersprache erlernen und die Anwendung nur als Werkzeug kennen lernen. Dieses Programm nennt sich „ShaderManager“. Die wesentlich einfachere Oberfläche lässt sich in Abbildung 1.3 erkennen und das Programm benötigt nur eine installierte Java-JRE, was heute auf nahezu allen Rechnern Standard ist. Leider war es mir nicht möglich alle Vorteile der Shadermakers zu erhalten, insbesondere verfügt der eingebaute Editor nicht über eine Syntax-Hervorhebung. Auch sind keine Variablen aus der Benutzeroberfläche heraus direkt setzbar um die Oberfläche übersichtlicher zu halten.

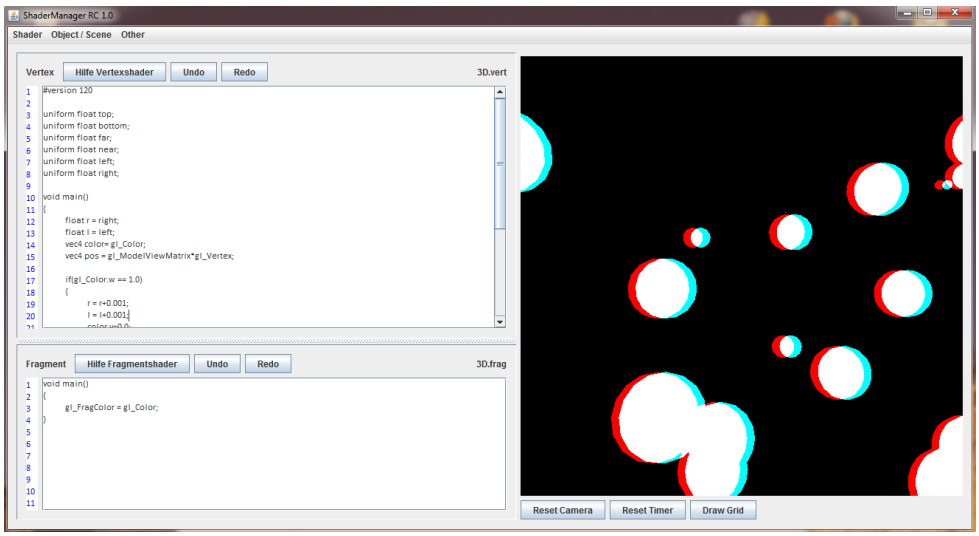


Abbildung 1.3: Grafische Benutzeroberfläche der ShaderMangers.

2 Darstellung der fachlichen Inhalte

An dieser Stelle werden die fachlichen Inhalte kurz dargestellt und erklärt. Zunächst werden die Objektdarstellung und Koordinatentransformationen vorgestellt und die Besonderheiten ausgearbeitet um anschließend über die Farbdarstellung zu sprechen. Was Shader sind und wie mit ihnen auf einfache Weise optische Verbesserungen möglich sind wird anschließend thematisiert. Dabei wird neben den Möglichkeiten auch auf die Grenzen eingegangen werden. Den Schluss bildet eine Einführung in stereoskopisches Sehen. Die restlichen fachlichen Inhalte, wie die mathematischen, werden an den entsprechenden Stellen im weiteren Verlauf des Texts kurz erwähnt, aber nicht gesondert dargestellt.

2.1 Objektdarstellung und Koordinatentransformationen

In der Computergrafik ist es gängig, Objekte nur durch ihre Oberfläche wie in Abbildung 2.1 zu beschreiben. Obendrein arbeitet der Vertexshader auf den Eckpunkten von Objekten, daher ist es notwendig die Darstellung von Objekten über eine Graphstruktur genauer zu betrachten.

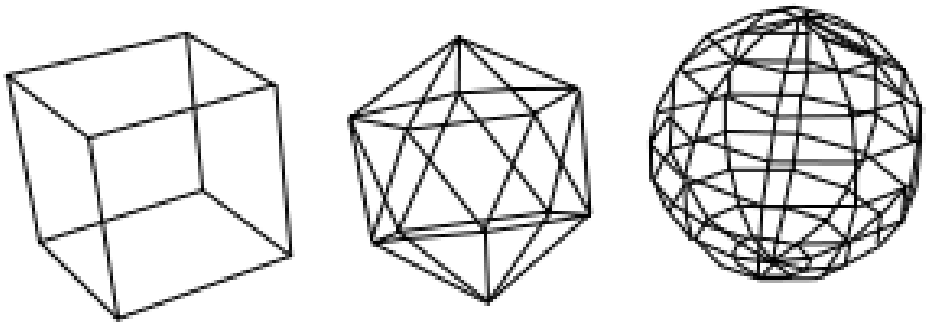


Abbildung 2.1: Objekte dargestellt als Polygonnetze. `http://upload.wikimedia.org/wikipedia/commons/0/06/Wire_frame.svg`

Dazu können beliebige Polygone verwendet werden, Dreiecksnetze haben

gewisse Vorteile bei der Verarbeitung und werden deshalb bevorzugt verwendet. Diese Polygone bestehen aus *Ecken* und *Kanten*, wobei nur die Position der Ecken und welche Ecken verbunden sind festgelegt sein muss, die Lage der Kanten ist dann eindeutig über die Lage ihrer Eckpunkte bestimmt. Das erklärt warum der Vertexshader auf diesen arbeitet und was er modifizieren kann:

- Position der Eckpunkte,
- Größe, Orientierung und Form der Objekte sowie die
- Farbe der Objekte, welche über die Farbe der Eckpunkte definiert und auf die Flächen interpoliert wird.

Um Größe, Orientierung, Ausrichtung und Form der Objekte anhand ihrer Eckpunkte zu ändern, kann man auf Koordinatentransformationen zurück greifen. Koordinatentransformationen sind Matrizen, welche an die Eckpunkte, dargestellt als Vektoren, multipliziert werden um die entsprechenden Transformationen durchzuführen.

Beispiel: Koordinatentransformation

Gegeben sei ein Transformationsmatrix $T = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ und der Punkt $v = \begin{pmatrix} 1 & 2 & 1 \end{pmatrix}^t$ in *affinen Koordinaten*[1]. Dann gilt

$$v' = T \cdot v = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \cdot 1 + 0 \cdot 2 + 3 \cdot 1 \\ 0 \cdot 1 + 2 \cdot 2 + 1 \cdot 1 \\ 0 \cdot 1 + 0 \cdot 2 + 1 \cdot 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \\ 1 \end{pmatrix},$$

also transformiert T den Punkt v auf den neuen Punkt $v' = \begin{pmatrix} 3 & 7 & 1 \end{pmatrix}^t$. Diese Matrix T ist besonders, da sie eine Streckung und Verschiebung des Vektors bewirkt, wie in Abbildung 2.2 im linken Teil visualisiert wird.

Es existieren verschiedene Grundtypen, wobei auf einer Darstellung der Scherung verzichtet wird:

[1] Affine Koordinaten zeichnen n-dimensionale Punkte und Vektoren besonders aus und sind deswegen um eine Dimension erweitert. Die (n+1) Komponente ist dabei entweder gleich 0 wenn es sich um einen Richtungsvektor handelt oder gleich 1 wenn es sich um einen Punktvektor handelt.

Skalierung Größenänderung um a,b und c in x-,y- und z-Richtung.

$$S = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation Drehung um α° im Uhrzeigersinn um x-,y- oder z-Achse. Rotation um beliebige Achse lässt sich durch Kombination von Rotationen um x-,y- oder z-Achse darstellen.

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) & 0 \\ 0 & -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation Verschiebung um a,b und c in x-,y- und z-Richtung.

$$T = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

An dieser Stelle scheint es zunächst nicht intuitiv zu sein, 4x4-Matrizen zu definieren, um auf Punkten im dreidimensionalen Raum verwendet zu werden, aber dies lässt sich leicht über die Mathematik erklären: Als lineare Abbildungen sind Matrizen nicht in der Lage den Nullpunkt zu verschieben. Demnach könnte man ein Objekt nur skalieren und rotieren, aber nicht translieren, da man dies als eine Verschiebung des Nullpunkts vom Objekt weg auffassen kann. Das wäre eine enttäuschende Einschränkung, aber die affine Geometrie beschreibt einen dreidimensionalen Punkt als vierdimensionalen Punkt, wobei die vierte Koordinate immer gleich 1 ist um das Problem zu umgehen. Dann gilt

$$\vec{v} = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ 1 \end{pmatrix} = \begin{pmatrix} v_1 + a \\ v_2 + b \\ v_3 + c \\ 1 \end{pmatrix}$$

was einer Verschiebung des Vektors v entspricht.

Um verschiedene Transformationen zu verknüpfen, genügt es die entsprechenden Matrizen zu multiplizieren, wobei auf die Reihenfolge zu achten ist (Matrixmultiplikation ist nicht kommutativ!). Daran zeigt sich der große Nutzen dieser Darstellung, denn es ist egal, ob man 1, 2 oder 1000 einzelne Transformationen ausführen möchte, wenn man die Matrizen bereits multipliziert hat, ist nur eine einzige Matrix-Vektor-Multiplikation pro Eckpunkt zu erledigen, um das Objekt wie gewünscht zu ändern. Oder anders ausgedrückt, um ein Objekt beliebig im Raum zu positionieren braucht man nur die Eckpunkte und Kanten des Ausgangsobjektes sowie eine Matrix zu speichern. Dies kann zum Beispiel bei 4 sich drehenden Reifen eines Autos sinnvoll sein, da das Objekt nur einmal gespeichert werden muss und für jeden Reifen dann nur die Transformationsmatrix unterschiedlich ist [1, 6].

Beispiel: Verknüpfte Koordinatentransformationen

Dieses Beispiel baut auf „Beispiel: Koordinatentranformation“ auf. Die dort vorgestellte Matrix ist bereits eine verknüpfte Koordinatentransformation, denn es gilt

$$\begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Das Tauschen der Reihenfolge bei der Matrixmultiplikation ergibt

$$\begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix},$$

welche die verschobene Position mit skaliert, denn Koordinatentransformationen arbeiten **um den Ursprung**, wie Abbildung 2.2 und

$$\begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix} = \cdot \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 1 \end{pmatrix}$$

zeigen. Abbildung 2.3 zeigt die Auswirkung auf ein Objekt an einem weiteren Beispiel.

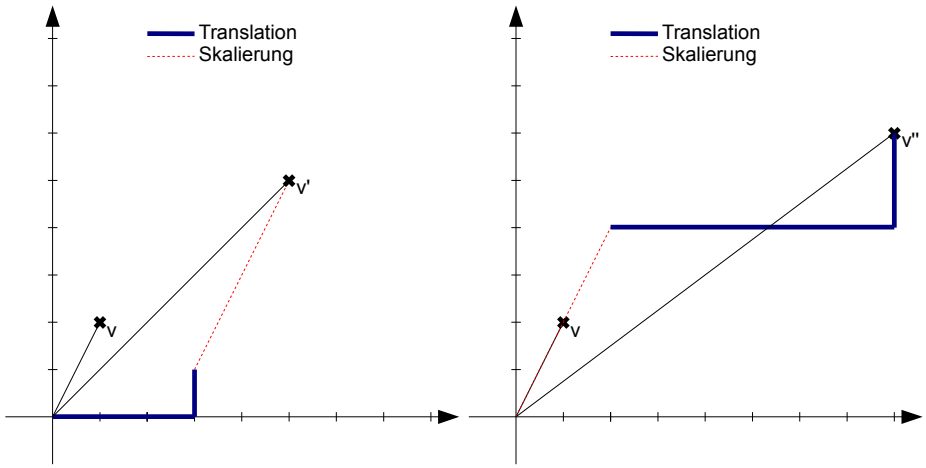


Abbildung 2.2: Visuelle Darstellung der Auswirkungen nach Ändern der Reihenfolge der Transformationen, wobei v der ursprüngliche Punkt, v' der transformierte Punkt aus dem Beispiel „Koordinatentransformation“ und v'' der transformierte Punkt aus dem Beispiel „Verknüpfte Koordinatentransformationen“ ist.

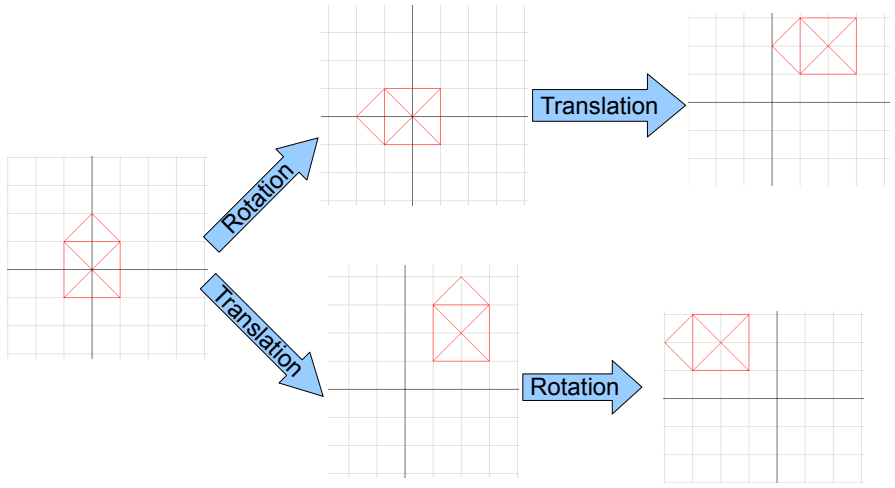


Abbildung 2.3: Oben: Rotation dann Translation. Unten: Translation dann Rotation.

2.2 Farbdarstellung

Es gibt eine Vielzahl von Farbdarstellungen, zum Beispiel in der Kunst wo nach warmen und kalten Farben unterschieden wird oder Farben sogar mit Formen in Verbindung gebracht werden (Scharfe vs. weiche/runde Farben). Diese Farbdarstellungen sind im Sinne der Algorithmisierung, welche eine exakte und *objektive Beschreibung* zur eventuellen Ausführung als Programm Bedarf, ungeeignet. Deshalb verwendet man in der Computergrafik häufig das RGB bzw. RGBA Farbmodell, welches Farben als additive Mischung der *Grundfarben Rot, Grün und Blau* auffasst und eventuell noch die Transparenz *Alpha* beachtet. Dieses Farbsystem hat den Vorteil, dass die Farbmischung des Monitors entsprechend funktioniert und aus diesen 3 Grundfarben das Bild auf dem Monitor erzeugt. Des Weiteren ist eine Umrechnung in andere gängige Farbsysteme einfach möglich um zum Beispiel auf dem Drucker mit der subtraktiven Farbmischung die gleichen Bilder, wie auf dem Monitor zu sehen sind, auszugeben. Ein Nachteil ist, dass in der Praxis nicht alle sichtbaren Farben dargestellt werden können, insbesondere im grünen Spektrum gibt es große Bereiche die fehlen, was aber technisch bedingt ist und dem normalen Verwender nicht auffällt.

Für viele Anwendungen und insbesondere für dieses Modul bietet sich eine Darstellung als Vektor an, wobei man die Farbe als Intensität zwischen 0.0 und 1.0 angibt oder auch zwischen 0 und 255. Aber da GLSL mit 0.0 bis 1.0 arbeitet, bevorzugen die Beispiele auch diese Darstellung, so ist $(1.0, 0.0, 0.0, 0.5)$ eine rote Farbe, die halb durchsichtig ist und $(1.0, 1.0, 0.0, 1.0)$ ist undurchlässiges Gelb.

2.3 Shaderprogramme und Phong-Beleuchtung

Shaderprogramme sind kleine Programme die direkt auf der Grafikkarte kompiliert und ausgeführt werden. Dies geschieht auf speziellen Hardwarekomponenten, den sogenannten „Shading-Units“, weshalb man in diesem Zusammenhang auch von „Programmable-Shading-Units“ spricht. Diese Programme ersetzten Teile der Rendering-Pipeline, also dem Ablauf wie Objekte aus der 3D-Szene auf dem Bildschirm dargestellt werden [1, 6].

Dieses Modul beschäftigt sich mit 2 verschiedenen Arten von programmierbaren Shadern:

Vertexshader arbeiten auf den Eckpunkten der Polygonnetze.

Fragmentshader arbeiten auf Fragmenten. Fragmente sind im Wesentlichen Pixel mit weiteren Informationen über die Szene. Zum Beispiel, wie tief das zum Bildschirmpixel gehörige Fragment der Szene von der Kamera aus gesehen in der Szene liegt.

Shaderprogramme werden in einer eigenen Programmiersprache erstellt, im Modul wird die OpenGL Shading Language verwendet, welche mit GLSL oder glSlang abgekürzt wird. Die GLSL hat den Vorteil, dass ihre Syntax der von C/C++ beziehungsweise Java sehr ähnelt und somit leicht verwendet werden kann. Allerdings ist sie nicht objektorientiert, sondern imperativ. Als bekannte Techniken, welche vollständig oder teilweise im Shader realisierbar sind, seien an dieser Stelle *Bumpmapping*, *Tesselation* oder *Toon-Shading* genannt. Der Effekt des Bumpmappings und des Toon-Shadings werden in Abblidung 2.4 und 2.5 verdeutlicht, die Idee hinter Tessellation wurde bereits in Abschnitt 1.1 angedeutet.

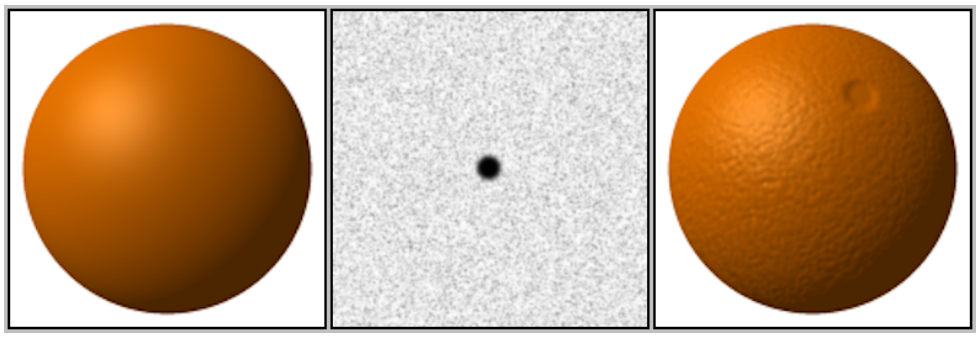


Abbildung 2.4: Links ist das strukturlose Objekt und rechts das Objekt nach Bumpmapping mit der Struktur in der Mitte. `http://upload.wikimedia.org/wikipedia/commons/0/0a/Bump-map-demo-full.png`

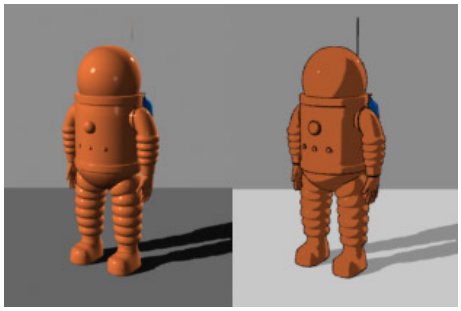


Abbildung 2.5: Das linke Bild wurde mit konventionellen Methoden gerendert, das rechte zum Vergleich mit einem Toon-Shader. Typisch sind die reduzierte Farbpalette und die schwarzen Objektränder. `http://upload.wikimedia.org/wikipedia/commons/1/1a/Toon_Shader.jpg`

Eine weitere Technik, die lokale Beleuchtung nach *Phong*, kann auch im Shader implementiert werden und dies werden die Schülerinnen und Schüler

im Laufe des Moduls durchführen. Dabei bedeutet lokal, dass keine globalen Effekte wie Schatten oder mehrfache Reflektionen simuliert werden. Phong nimmt Licht, physikalisch nicht korrekt, als aus drei Teilen bestehend an, dem ambienten, diffusen und spekularen Anteil. Der *ambiente* Teil imitiert durch einfaches Addieren einer Konstante mehrfach reflektiertes Licht, welches Szenen eine gewisse Grundhelligkeit verleiht, auch wenn an diesen Stellen nie direktes Licht gewesen ist. Der *diffuse* Teil ist das gestreute Licht und der *spekulare* Anteil ist für Highlights verantwortlich, welche dem nahezu perfekt reflektiertem Licht entsprechen (siehe auch Bild 2.6). Durch einfaches Aufaddieren der einzelnen Werte im Fragmentshader erhält man bereits einen überzeugenden Eindruck von Beleuchtung:

$$C_{phong}(p,n,v) = C_{ambient} + \sum_{l \in Lightsources} C_{diffuse}(p,n,l) + C_{specular}(p,n,l,v)$$

wobei p die Position des Fragments, n die Oberflächennormale am Punkt p, v die Position des Betrachters (viewer) und l die Position der Lichtquelle ist. Der ambiente Lichtanteil wird durch

$$C_{ambient} = \alpha_a \cdot C_A$$

bestimmt, wobei α_a die ambiente Materialkonstante ist und C_A die globale ambiente Lichtintensität beschreibt. Diese Konstanten werden durch den Programmierer gesetzt, die Materialkonstante für jedes Objekt, hingegen die Lichtkonstante nur einmal pro Szene. Der diffuse Lichtanteil ist etwas komplizierter, da nun die Fragmentposition, die Lichtposition und die Oberflächennormale einbezogen werden müssen

$$C_{diffuse}(p,n,l) = \alpha_d \cdot C_l \cdot cos(n, l-p) = \alpha_d \cdot C_l \cdot \frac{n^T(l-p)}{\|l-p\|}$$

wobei α_d die diffuse Materialkonstante und C_l die Farbe des Lichts ist. Der Term $cos(n, l-p)$ beschreibt den Kosinus zwischen der Oberflächennormalen und dem Richtungsvektor zwischen der Position der Lichtquelle l und dem Fragment p. Ganz ähnlich gestaltet sich der spekulare Anteil des Lichts, wobei ein anderer Winkel berechnet wird und dieser durch einen sogenannten Shininess-Faktor s beeinflusst wird.

$$r = (2nn^T - I) \cdot (l-p)$$

$$C_{specular}(p,n,l) = \alpha_s \cdot C_l \cdot \cos(r, v-p) = \alpha_s \cdot C_l \cdot \left(\frac{r^T(v-p)}{\|v-p\|}\right)^s$$

dabei ist α_s der spekulare Teil des Lichts, r der perfekt reflektierte Strahl an der Oberfläche und $\cos(r, v-p)$ ist der Kosinus des Winkels zwischen r und dem Richtungsvektor der Fragmentposition p zum Betrachter v (vgl. [6]). Wie

man sieht hält sich die Komplexität in Grenzen, aber die Ergebnisse sehen sehr gut aus, wie beispielsweise der Lösungsvorschlag zur lokalen Beleuchtung in Abschnitt 6.1.6 zeigt. Dort finden sich auch Informationen zu technischen Besonderheiten der Umsetzung.

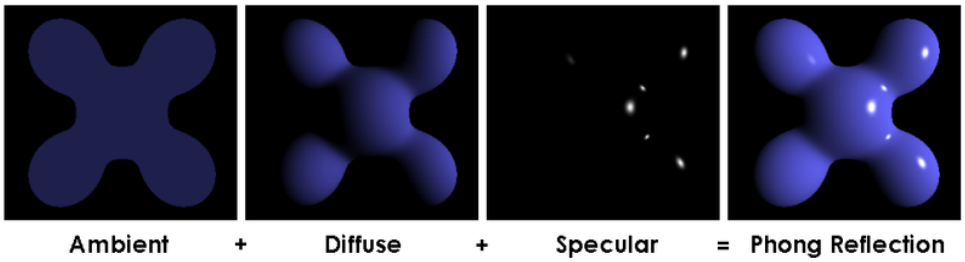


Abbildung 2.6: Phong Lighting. Die einzelnen Komponenten des Lichts werden aufaddiert und ergeben einen überzeugenden Lichteffekt. `http://en.wikipedia.org/wiki/File:Phong_components_version_4.png`

2.4 Stereoskopisches Sehen

Unter stereoskopischem Sehen versteht man für den Betrachter eines Bildes durch technische Tricks einen Tiefeneindruck auf einer planen Fläche zu erzeugen. Dies funktioniert, da Menschen beidäugig sehen und somit im Gehirn durch Vergleich der beiden, leicht verschieden verzerrten Bilder die Tiefe abschätzen können [9]. Falls nun dem linken und rechten Augen ein eigenes Bild dargeboten wird, was aus einem leicht verschobenen Blickwinkel gerendert wird, entsteht ein Tiefeneindruck, obwohl keine Tiefe da ist.

Technisch lässt sich das Darstellen von verschiedenen Bildern für jedes Auge auf verschiedenen Wegen lösen. Shutter-Brillen, welche mit dem Monitor synchronisiert werden, blenden für das menschliche Auge nicht - oder kaum - wahrnehmbar abwechselnd ein Auge aus. Polfilterbrillen hingegen lassen nur einen gewissen Teil des Lichts durch, der falsch „gepolte“ Teil bleibt draußen. Nun werden auf dem Monitor zwei verschieden gepolte Bilder abgestrahlt und im linken oder rechten Auge des Betrachters kommen verschiedene Bilder an. Beide vorgestellten Techniken haben den Nachteil, dass sie sich nur mit speziellen Monitoren realisieren lassen. Ein billigerer Ansatz über verschiedenfarbige Brillengläser, z.B. Rot-Grün oder besser Rot-Cyan, kommt ohne weitere Hardware aus und funktioniert auf ähnliche Weise wie die Polfilterbrillen, nur das hier durch die farbigen Gläser nicht falsch gepoltes Licht sondern Farben gefiltert werden, so dass im linken und rechten Auge ein unterschiedliches Bild entsteht. Dazu wird die 3D-Szene für ein Auge nur mit dem Rotanteil der Farbe und für das andere Auge nur mit der grünen und

blauen Farbe gerendert.

Die Umsetzung in der Computergrafik gestaltet sich dabei denkbar günstig, da die Szenen sowieso schon als 3D-Daten vorliegen. Daher müssen diese nur noch aus verschiedenen Blickwinkeln gerendert und passend gefärbt werden. Die verschiedenen Blickwinkel lassen sich über die Modifikation der Frustum-Matrix erreichen,

$$frustum = \begin{pmatrix} \frac{2.0 \cdot near}{right - left} & 0.0 & \frac{right + left}{right - left} & 0.0 \\ 0.0 & \frac{2.0 \cdot near}{top - bottom} & \frac{top + bottom}{top - bottom} & 0.0 \\ 0.0 & 0.0 & \frac{-far + near}{far - near} & \frac{-2.0 \cdot far \cdot near}{far - near} \\ 0.0 & 0.0 & -1.0 & 0.0 \end{pmatrix}$$

welche die Szene im Sichtbereich von dem Pyramidenstumpf (siehe auch Abbildung 2.7) zu einem Würfel transformiert, was für die perspektivische Verzerrung verantwortlich ist. Die Szene im Würfel kann anschließend einfach auf die Bildebene projiziert werden. Durch das Ändern der Werte $left$ und $right$ können die zwei verschiedenen Blickrichtungen für das linke und rechte Auge simuliert und auf dem Bildschirm ausgegeben werden, wie in Abbildung 2.8 zu sehen ist (vgl. [1]).

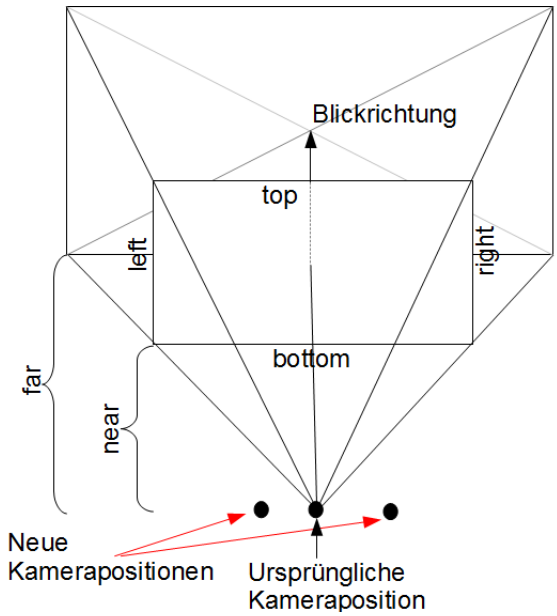


Abbildung 2.7: Der sichtbare Bereich, dargestellt als Pyramidenstumpf (engl. frustum), welcher über die Paramter near, far, top, bottom, left und right definiert werden kann.

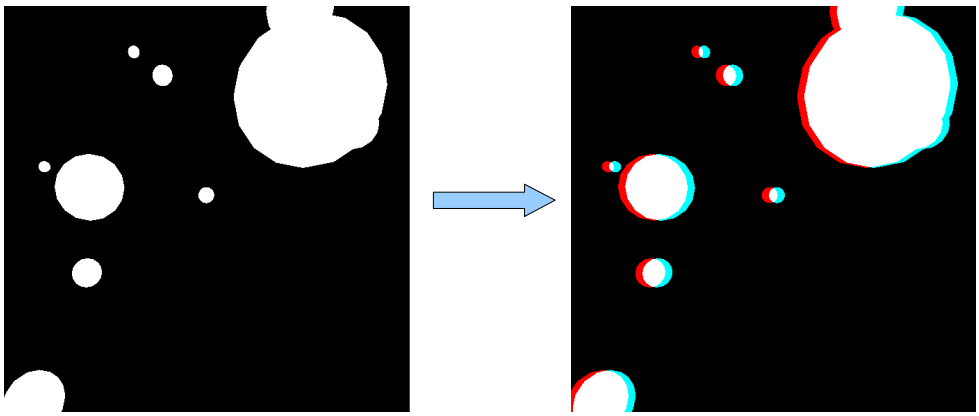


Abbildung 2.8: Darstellung des 3D-Effekts, welcher über Farbfilterbrillen hervorgerufen werden kann.

3 Didaktische Analyse des gesamten Moduls

Dieses Kapitel, „Didaktische Analyse des gesamten Moduls“, thematisiert die groben didaktischen Entscheidungen, die für das gesamte Modul getroffen wurden. Hingegen beschreibt das folgende Kapitel, „Das Modul“, die getroffenen Einzelentscheidungen in jeder Phase. Hier wird dargestellt, was generell an Vorwissen wichtig ist, welcher Stoff ausgewählt und an welchen Stellen und warum dieser didaktisch reduziert wurde, welche Lernschritte vollzogen und welche Lernziele verfolgt werden. Wichtig ist, wie und womit der Inhalt den Schülerinnen und Schülern näher gebracht werden soll, sowie wie die Ergebnisssicherung vonstatten geht, um den behandelten Stoff dauerhaft zu festigen und ein Feedback zum Lernerfolg für die Betreuerinnen und Betreuer zu bieten. Auf eine explizite Einordnung in den Lehrplan wurde verzichtet, da Computergrafik an sich nicht im Lehrplan steht und nur Teilbereiche wie die Algorithmisierung von Problemlösungen direkt relevant im Schulstoff sind. Jedoch ergibt sich die Zielgruppe aus dem nötigem Vorwissen, so dass sich dort auf den Lehrplan bezogen wird.

3.1 Vorwissen

Das für dieses Modul benötigte Vorwissen ist umfangreich und baut auf mathematischen und informatischen Kenntnissen und Fähigkeiten auf:

Matrix-Vektorrechnung Im ganzen Modul stehen Vektoren als Datenstruktur im Mittelpunkt und an einigen Stellen wird die Matrix-Vektorrechnung benötigt. Allerdings sind nur rudimentäre Kenntnisse erforderlich, das heißt man sollte Wissen wie eine Matrix und ein Vektor aussehen und wie die verknüpften Rechenoperationen funktionieren, insbesondere die Matrix-Vektor-Multiplikation, die explizit und implizit verwendet wird. Falls diese den Schülerinnen und Schülern zwar bekannt aber nicht mehr geläufig sein sollten, liegt ein Infoblatt zur Mathematik bereit, auf dem die wesentlichen verwendeten Operationen knapp erklärt werden (siehe Abschnitt 6.2.2).

Normalenvektoren Idealerweise sollten die Schülerinnen und Schüler wissen was ein Normalenvektor ist, das heißt die sinngemäße Definition und

die geometrische Bedeutung sind klar. Falls nicht, finden sie auch dazu Informationen auf dem Matheblatt in Abschnitt 6.2.2.

Programmierkenntnisse Die Programmiersprache GLSL ist von der Syntax und Semantik sehr ähnlich zu C und recht simpel. Ein grundlegendes Vorwissen einer ähnlichen Programmiersprache, wie Java oder C++ ist ausreichend, andere Programmiersprachen wie Visual Basic oder Delphi sind aufgrund der Unterschiede in der Syntax, zum Beispiel bei der Deklaration der Variablen, eher ungeeignet beziehungsweise erschweren den eigentlichen Lerneffekt erheblich. Obendrein hat das Lernen einer komplett neuen Programmiersprache keinen großen Lerneffekt und sie alleine innerhalb eines Moduls zu Lernen ist kaum möglich, sodass diese nur am Rande thematisiert und auf dem Infoblatt zur Shaderprogrammierung (siehe 6.2.3) auszugsweise erläutert und ansonsten als Vorwissen abgestempelt wird.

An den oben genannten Punkten wird wieder die Nähe der Computergrafik zur Mathematik deutlich, welche zwei von drei Bereichen des Vorwissens für sich beansprucht. Obendrein beschreibt dies nur das „harte“ Vorwissen, natürlich sind auch Fähigkeiten in der Problemanalyse und Modellierung, der eigenständigen Beschaffung von Informationen oder den grundlegenden mathematischen und informatorischen Fähigkeiten wie das Eintragen von Punkten in ein Koordinatensystem, Rechenfertigkeiten oder der sichere Umgang mit dem PC erforderlich. Obschon das bereitgestellte Informationsmaterial das komplette Vorwissen abdeckt, ist dies nicht zur „Umgehung“ des Vorwissens geeignet. Dieses Material ist als übersichtliche Darstellung und Auffrischung von bereits Gelerntem gedacht und deswegen eher knapp und sachlich gehalten, wie Abschnitte 6.2.2 und 6.2.3 zeigen.

Zusammenfassend lässt sich mit Blick in den Lehrplan Mathematik sagen, dass dieses Modul nur für Schülerinnen und Schüler der gymnasialen Qualifikationsphase geeignet ist, welche obendrein über das geforderte Vorwissen verfügen, da die lineare Algebra in der Abiturprüfung nicht verpflichtend ist (siehe dazu [8]).

3.2 Auswahl der Inhalte

Die Computergrafik ist noch ein junges wissenschaftliches Forschungsgebiet, aber dennoch ist eine komplette Behandlung der entscheidenden Ideen innerhalb eines Moduls mit eventueller Vor- und Nachbereitungsphase nicht möglich. Demnach bestand der erste Schritt der didaktischen Reduktion in der begründeten Auswahl der zu behandelnden Inhalte zu Ungunsten anderer Inhalte. Die Entscheidung für die *Shaderprogrammierung* wurde bereits in Abschnitt 1.1 ausführlich behandelt und wird hier nicht mehr besprochen. *Objektdarstellung* mittels Polygonnetzen ist ein Kernthema des Moduls. Der

Vorzug gegenüber anderen Themen ist, dass dies grundlegend ist und für viele andere Lerninhalte als Basis benutzt werden kann beziehungsweise muss, wie es auch im Modul geschehen wird. Von einer fundamentalen Idee nach Hubwieser kann man hier leider nicht sprechen, da noch nicht abzusehen ist, ob diese Form der Objektdarstellung das Zeitkriterium erfüllen wird, da andere Verfahren immer wieder in der Literatur auftauchen, die angeblich Vorteile bieten zum Beispiel Voxelgrafik[1] [5]. Polygonnetze sind allerdings momentan der Standard auf dem die Hardware und viele Algorithmen beruhen und somit als Wissen für ein weiteres Vorgehen in der Computergrafik unabdingbar sind.

Die *Transformation dieser Objekte über Matrizen*, den sogenannten Koordinatentransformationen, ist ein Thema, welches mir besonders wichtig ist, da dieses Thema im Allgemeinen negativ behaftet ist. Zur Motivation möchte ich zunächst eine kleine Episode erzählen, die gut dazu passt:

> Zur Zeit der Erstellung dieser Arbeit befand sich in Aachen ein Weihnachtsmarkt, den ich mit Freunden besuchte. Im Laufe der Zeit kamen wir auf meine Staatsexamensarbeit zu sprechen und ich stellte die Themen vor. Beim Thema Matrizen bemerkte einer der Anwesenden, dass dies doch diese Dinger zum Lösen von Gleichungen sind und dass er die nie verstanden und gemocht hat.

Dies ist natürlich nur ein Beispiel von vielen Situationen, die ich im Laufe meiner „Karriere“ in der Mathematik erlebt habe, in denen mit klar wurde dass in den Köpfen der Menschen ein falsches und für die Thematik ungünstiges Bild über Matrizen besteht. Aber es macht gut klar was viele Schülerinnen und Schüler in der Schulzeit mit Matrizen machen: Sie werden als Darstellungsform für lineare Gleichungssysteme genutzt um den Gauß-Algorithmus zu verwenden, wodurch sie meiner Meinung nach einen Werkzeugcharakter erhalten. Aber dies wird ihnen nicht gerecht, denn Matrizen sind Abbildungen, die einige tolle Eigenschaften besitzen und das Lösen von Gleichungssystemen mittels dem Gauß-Algorithmus ist im Prinzip nur ein Spezialfall der Bestimmung von Inversen. Die Koordinatentransformationen sollen dieses Bild ein wenig verbessern und können auf die Schulmathematik übertragen werden, indem man nun spezielle Matrizen sehr leicht invertieren kann, zum Beispiel eine Rotationsmatrix, deren Inverse einfach die Rotation in die andere Richtung darstellt. Ein weiterer Punkt für die Behandlung von Matrizen ist, dass sie in vielen anderen Wissenschaften vorkommen. Explizit natürlich in der Mathematik, wo teils bereits in der Schule Drehmatrizen im Zusammenhang mit komplexen Zahlen oder für sich alleine eingeführt werden, aber auch in

[1] Weitere Informationen zur Voxelgrafik beispielsweise unter `http://www.pcgameshardware.de/aid,681315/Voxel-Rendering-am-Beispiel-id-Tech-6-erlaeutert/Technologie/Wissen/`

BWL, VWL oder in den Erziehungswissenschaften findet man sie als Anwendung wieder, zum Beispiel als Übergangmatrizen. Somit sind vertiefte Kenntnisse absolut lebensvorbereitend in der Oberstufe, die insbesondere auf ein Studium vorbereitet.

Die Vektorrechnung im Allgemeinen mit *Winkelbestimmung über das Skalarprodukt* und *Normalenvektoren* werden in verschiedenen Aufgabenteilen angewendet, da sie grundlegende Kenntnisse in der Mathematik und Informatik darstellen.

Im Laufe des Moduls wird von den Schülerinnen und Schülern der Umgang mit Farben gefordert, wodurch sich die Behandlung eines Farbsystems anbietet. Die Auswahl des *RGBA-Farbsystems* hat folgende Vorteile:

- RGB-Werte sind eine weit verbreitete Darstellungsform, welche in den meisten Programmen und Sprachen wie HTML, aber auch in anderen Bereichen wie Kunst oder Architektur zu finden sind.
- RGB-Werte lassen sich leicht in andere Farbcodierungen überführen, was in diesem Modul jedoch nicht explizit thematisiert wird.
- Die additive Farbmischung, also die physikalische Grundlage der RGB-Farbdarstellung, findet sich in Monitoren wieder und hat somit Bezug zur aktuellen Lebenswelt der Schülerinnen und Schüler.
- RGBA-Werte werden in der GLSL verwendet.
- Die im Modul behandelten Shaderprogramme, wie Phong-Beleuchtung, arbeiten auf einem RGBA-Farbvektor als Datenstruktur.

Den Vorteilen gegenüber steht der Nachteil, dass diese Idee nur in der Informatik und speziell in der Computergrafik eine Anwendung findet. Somit ist sie alles andere als fundamental für die Wissenschaft Informatik, aber als Werkzeug für das weitere Vorgehen unabdingbar.

Die behandelten Algorithmen, *Phong-Beleuchtung* und *3D-Stereo* dienen eher der Sicherung des bisher Gelernten. Für sich gesehen ist die Phong-Beleuchtung ein guter Einstieg in die komplexere Shaderprogrammierung und liefert direkt schöne Ergebnisse und zeigt die Mächtigkeit der Programmiersprache GLSL. 3D-Stereo ist als Ergebnismotivation gedacht, der inhaltliche Gewinn ist dabei eher gering, da eine tiefgründigere Behandlung des Themas eine Einführung in die projektive Geometrie erfordern würde. Dies ist innerhalb des Moduls zeit- und aufwandsbedingt nur schwer machbar und wird darum ausgelassen.

3.3 Didaktische Reduktion

Die didaktische Reduktion ist ein wichtiger Punkt dieser Arbeit, da sie die Grundlage für die Erstellung des Shadermanagers ist und dort noch genauer

thematisiert wird, siehe auch Abschnitt 4.4.4. Bei den anderen thematisierten Inhalten habe ich mich bewusst mit der didaktischen Reduktion zurück gehalten um ein komplettes und korrektes Bild der Grafikprogrammierung zu vermitteln. Somit wird mit der GLSL eine komplette und aus der Praxis kommende Programmiersprache zur Verfügung gestellt, bei den Koordinatentransformationen sind alle Kombinationen möglich und können in einem speziellen Programm oder in den Shaderprogrammen getestet werden (siehe Abschnitt 4.4.2 für weitere Informationen). Eine wesentliche didaktische Reduktion ist, dass während des kompletten Moduls eher mit einfachen Formen gearbeitet wird. Zu Beginn mit dem schematischen „Haus vom Nikolaus“, welches dann in 3D übergeht und auch später werden eher einfache Szenen verwendet, welche die Algorithmen leichter überprüfbar machen als komplexe Szenen mit detaillierten Formen und Texturen.

Die zu erstellenden Programme sind so weit didaktisch reduziert, dass nur noch die Kernideen im Shader zu programmieren sind und die Programmierarbeit wie Übergabe der Uniform-Variablen bereits vorprogrammiert sind. Somit steht nur noch das eigentliche Problem und dessen Lösung im Fokus der Programmierung. Den Schülerinnen und Schülern entgeht hier ein Teil der Programmierarbeit, die für die Erstellung des kompletten Programms notwendig wäre. Da es sich dabei aber um kurzlebige und plattformabhängige Ingenieursfähigkeiten handelt, sollte man dies nicht als Einschränkung, sondern eher als *Fokussierung auf das Wesentliche*, verstehen.

3.4 Lernschritte und Lernziele

Die zu bewältigenden Lernschritte sind im Modulverlauf erkennbar und spiegeln sich in den intendierten Lernzielen wider. Die Schülerinnen und Schüler

- Phase 1
  - erstellen Objekte aus vorgegebenen Netzen.
  - erstellen Netze aus eigenen Objekten.
  - transformieren Objekte mittels Matrizen (Skalierung, Rotation, Translation).
  - transformieren Objekte beliebig über verknüpfte Koordinatentransformationen.
  - erstellen eigene Shaderprogramme.
- Phase 2
  - verwenden das RGBA-System zur Darstellung von Farben.
  - erkennen Farben anhand der gegebenen RGBA-Werte.
  - berechnen Winkel über das Skalarprodukt.

  - arbeiten mit Normalenvektoren.
  - kommunizieren mittels Varying-Variablen zwischen den beiden Shaderprogrammen.

- Phase 3
  - können das Phong-Beleuchtungsmodell erläutern.
  - können die Ideen hinter 3D-Stereo andeuten.
  - erstellen komplexe Shaderprogramme.

Ein thematischer Schwerpunkt ist in der oben stehenden Liste erkennbar, die Shaderprogrammierung. Sie zieht sich durch alle Phasen und wird in Phase 1 eher spielerisch thematisiert, während Phase 2 und 3 schon die Programmierung recht komplexer Shaderprogramme von den Schülerinnen und Schülern verlangen. Durch die Einführung in die Shaderprogrammierung sollen die Schülerinnen und Schüler nach dem Modul eine Vorstellung davon haben, was mit Shadern möglich ist und wie diese programmiert werden können.

In Bezug auf Matrizen sollen die Schülerinnen und Schüler diese auch mal auf eine andere Weise kennen lernen. Es wäre ein Erfolg, wenn wenigstens einige Schülerinnen und Schüler nach dem Modul Matrizen mit anderen Augen sehen und auf die Frage, was man mit Matrizen machen kann, antworten, dass man Vektoren abbilden oder Objekte transformieren kann. So könnten die Schülerinnen und Schüler den Sinn von Matrizen begreifen.

Eine weitere Absicht des Moduls ist, dass die Schülerinnen und Schüler ihre Fähigkeit weiter vertiefen, eigenständig Informationen zu sammeln und daraus ein Programm zu erstellen. Dies geschieht implizit in vielen Aufgaben auf den Blättern, aber explizit beim Phong-Beleuchtungsmodell, siehe dazu Abschnitt 6.1.6. Ein Ziel, was kein Lernziel aber wesentliche Absicht des Moduls ist, ist die Schülerinnen und Schüler für die Informatik allgemein und die Computergrafik im speziellen zu *motivieren* und ein realistisches Bild von dieser zu vermitteln.

3.5 Gewählte Sozial- und Aktionsformen, Methoden sowie Medien

Da es sich um ein Modul für das Schülerlabor Informatik handelt, ist die dominierende Aktionsform, dass die Schülerinnen und Schüler eigenständig tätig sind und Problemstellungen oder Aufgaben bearbeiten. Dazu werden verschiedene Arbeitsblätter zur Verfügung gestellt, die teilweise kleinschrittig ein neues Thema besprechen oder eine Problemsituation aufwerfen und Hinweise zur Lösung bereitstellen. Dies soll ein forschendes Lernen fördern und so die Nachhaltigkeit des erlernten Wissens steigern.

Die verwendete Methodik innerhalb des Moduls ist größtenteils auf Partnerarbeit ausgerichtet. Kurze Präsentationsphasen finden zwar auch statt,

sind aber nicht Kern des Moduls, sondern eher zur Ergebnisssicherung oder der kompakten Präsentation der GLSL gedacht. Allerdings sollen die gewählten Medien den Ablauf auflockern. So sind zwei Phasen geplant in denen die Schülerinnen und Schüler nicht am Computer oder auf Papier arbeiten, sondern mit tatsächlichen Gegenständen. Einmal ist das eine RGB-Led mit einstellbaren Widerständen zur Farblehre und andererseits ein Steckbrett zur Objektdarstellung (siehe Abschnitt 4.5.1 oder 4.4.1). Zum Beispiel werden durch eine eigenständige Informationsbeschaffung im Internet oder durch Aufgaben auf dem Papier die Phasen am Computer ergänzt. Des Weiteren werden in der 3D-Stereo Phase noch 3D-Brillen verwendet um den gewünschten Tiefeneffekt zu erhalten. In den Präsentationsphasen findet das Smartboard Verwendung.

3.6 Ergebnissicherung

Die Ergebnissicherung findet unterschiedlich statt. Die erste Phase setzt dazu eher auf ein direktes Feedback nach jeder Aufgabe, was den individuellen Lerneffekt fördern soll. Bei den Programmierteilen ist das nicht in der Form möglich, so dass sich die Schülerinnen und Schüler selber die Ergebnisse sichern müssen, was allerdings über eine optische Kontrolle gut möglich ist. Abschließend findet eine gemeinsame Präsentation eines Ergebnisses statt, so dass die Schülerinnen und Schüler eventuelle Fehler in der eigenen Lösung erkennen und verbessern können und merken, dass die eigene Leistung anerkannt wird.

Bei den Aufgaben auf den Arbeitsblättern finden keine direkte Ergebniskontrollen statt, da diese eher auf die Thematik hinarbeiten die dann in einem Programm oder Shader erprobt werden. Eine Ausnahme bildet das Sicherungsblatt Koordinatentransformationen, wo die Betreuerinnen und Betreuer sowie die Partnerin beziehungsweise der Partner die Ergebnisse überprüfen und eventuelle Fehler aufzeigen sollen, damit diese berichtigt werden können (siehe Abschnitt 6.1.2).

4 Das Modul

4.1 Überblick

Dieser Abschnitt beschäftigt sich mit den Ideen, Konzepten und Materialien des Moduls. Dazu wird jeder Punkt genau betrachtet und die getroffenen Entscheidungen dargestellt und begründet.

Doch vorab wird der Modulverlauf im Überblick thematisiert. Er ist so gestaltet, dass die Schülerinnen und Schüler möglichst viele Gelegenheiten bekommen, selbst tätig zu werden und somit den Stoff zu verinnerlichen. Des Weiteren werden wechselnde Medien geboten um die langen Computerarbeitsphasen etwas zu lockern. Die vorherrschende Methode ist die Partnerarbeit, dies soll ein verantwortungsvolles Arbeiten und soziale Kompetenzen schulen. Andere, im Laufe der Erstellung angedachte Methoden, wie ein Gruppenpuzzle haben sich als unpassend erwiesen und wurden wieder gestrichen [2].

Dieses Modul ist für einen Zeitrahmen von ungefähr 4 Zeitstunden und 15 Minuten geplant. Thematisch beschäftigen sich die Schülerinnen und Schüler bis zur ersten Pause mit der Darstellung und Modifikation von zwei- oder dreidimensionalen Objekten. Dabei machen sie erste Erfahrungen mit der Programmiersprache OpenGL Shading Language im Vertexshader. Nach der Pause werden sich die Schülerinnen und Schüler anschließend mit der Darstellung von Farben im RGBA-Format beschäftigen und wenden ihr neu erworbenes Wissen in einem Fragmentshader-Programm an. Abschließend werden 2 komplexere Shaderprogramme erstellt, die ein schönes Ergebnis liefern sollen um die Schülerinnen und Schüler für eine eigene, weiterführende Beschäftigung mit dem Thema zu motivieren.

Im folgenden Abschnitt findet sich ein tabellarischer Überblick über den Verlauf des Moduls.

4.2 Verlaufsplan

Zeit[1]	Inhalt	Methode	Medien
0:00	Begrüßung und Motivation	Präsentation	Smartboard
0:10	Objektdarstellung	Partnerarbeit	Steckbrett + Arbeitsblatt
0:30	Programm Koordinatentransformationen	Partnerarbeit	Computer
1:00	Sicherung Koordinatentransformationen	Einzelarbeit / Partnerarbeit	Arbeitsblatt
1:20	Präsentation des Programms: Shadermanager	Frontal bzw. Unterrichtsgespräch	Smartboard
1:30	Animation durch Koordinatentransformationen	Partnerarbeit	Computer + Arbeitsblatt
2:00	Pause		
2:15	Farben	Partnerarbeit	RGB-LED + Steuerung + Arbeitsblatt
2:30	Comiclook	Partnerarbeit	Computer + Arbeitsblatt
3:00	Pause		
3:15	Lokale Beleuchtung	Partnerarbeit	Computer + Arbeitsblatt
3:45	3D-Stereo	Partnerarbeit	Computer + Arbeitsblatt
4:15	Ende		

4.3 Einstieg: Motivation und Zielvorstellung

Als motivierender Einstieg wird den Schülerinnen und Schülern direkt zu Beginn eine Szene des Stereoskopie-Shaders am Smartboard bzw. Beamer gezeigt. Das frühe Aufzeigen dieses Ergebnisses soll die Schülerinnen und Schüler über eventuell als trocken empfundene Themen innerhalb des Moduls hinweg motiviert halten, da sie dann bereits wissen, wofür diese Grundlagen durchgearbeitet werden. Die Entscheidung, warum hier eine Szene in 3D-Stereo gezeigt wird begründet sich über die Tatsache, dass damit ein Thema aus dem Alltagsleben der Schülerinnnen und Schüler thematisiert wird und somit hoffentlich zu intrinsischer Motivation führt [2, 5].

[1] Zeitangaben sind nur ungefähre Werte.

Gleichzeitig soll diese Szene die Verfügbarkeit des Wissens im Bereich der Informatik aktivieren. Auf andere Ansätze welche den selben Effekt verfolgen, zum Beispiel Brainstorming habe ich verzichtet, da zum Thema Computergrafik im Allgemeinen oder Shaderprogrammierung im Speziellen, bei den meisten Schülerinnen und Schüler nur ein geringes Vorwissen existiert. So werden peinliche Situationen der Stille vermieden.

Neben einer Begrüßung durch die Betreuerinnen und Betreuer und anderer organisatorischer Punkte, wie die Verteilung der Sitzplätze, bleibt noch die Vorstellung des Modulverlaufs wie in Abschnitt 6.2.1 zu sehen, damit die Schülerinnen und Schüler wissen, welche Lerninhalte sie erwarten können und das neu erlernte Wissen in Sinnzusammenhänge setzen können [5]. Kurz gesagt, hier wird die Methode des informierenden Unterrichtseinstiegs verwendet.

4.4 Phase 1: Objektdarstellung und Koordinatentransformationen

4.4.1 Erarbeitungsphase zur Darstellung von Objekten in der Computergrafik

Nach dem motivierenden Einstieg erhalten die Schülerinnen und Schüler die Möglichkeit direkt haptische Lernerfahrungen zur Objektdarstellung zu machen. Die zu beantwortende Kernfrage ist dabei:

Wie werden Objekte in der Computergrafik repräsentiert?

Die Schülerinnen und Schüler sollen sich mit der Darstellung von Objekten als Polygonnetz - hier speziell als Dreiecksnetz - beschäftigen.

Sie arbeiten dabei mit einem Steckbrett, wie in Abbildung 4.1 zu sehen ist. Gleichzeitig erhalten sie ein Arbeitsblatt zur Objektdarstellung, welches

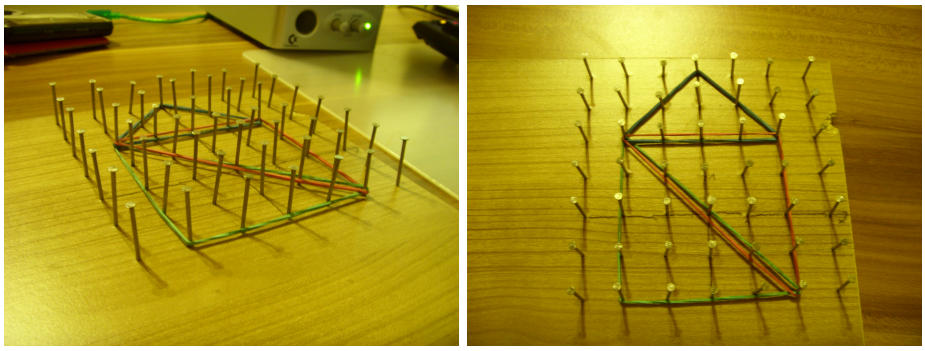


Abbildung 4.1: Bilder des Steckbretts. Zu sehen ist ein Haus, welches mit einem Dreiecksnetz dargestellt wird.

in Abschnitt 6.1.1 zu finden ist und auch mathematische Herausforderungen bereit hält. Dieses Arbeitsblatt bietet eine Infobox mit den wesentlichen Informationen dieser Erarbeitungsphase, welche dann in Aufgabe 1

Aufgabe 1

Steckt auf dem Brett folgende Dreiecksflächen ab. Welches Objekt entsteht?

a) $\begin{pmatrix}-1\\1\end{pmatrix},\begin{pmatrix}-1\\-1\end{pmatrix},\begin{pmatrix}1\\-1\end{pmatrix}$, b) $\begin{pmatrix}1\\1\end{pmatrix},\begin{pmatrix}1\\-1\end{pmatrix},\begin{pmatrix}-1\\1\end{pmatrix}$ c) $\begin{pmatrix}-1\\-1\end{pmatrix},\begin{pmatrix}-1\\1\end{pmatrix},\begin{pmatrix}0\\2\end{pmatrix}$ _____

auf die reale Welt übertragen werden, indem die Dreiecksflächen auf dem Steckbrett abgesteckt werden, das heißt mit den bereitgestellten Kordeln über die Eckpunkte verbunden werden. Als Lösung erhalten die Schülerinnen und Schüler ein Haus, was dem „Haus vom Nikolaus“ bis auf einen fehlenden Strich gleicht. Dies erwähne ich an dieser Stelle explizit, da dies eine weitreichende Entscheidung war. Denn die Struktur des Hauses ist einfach, da es durch wenige Striche dargestellt werden kann, welches als eine reduzierte beziehungsweise schematische Darstellung eines Hauses zu verstehen ist. Wichtig ist dabei, dass dies keine Verfälschung darstellt, denn so kann tatsächlich eine Wand eines Hauses gespeichert und dargestellt werden. Das „Haus vom Nikolaus“ finden die Schülerinnen und Schüler im nächsten Teil des Moduls wieder, was den thematischen Zusammenhang verdeutlicht (siehe Abschnitt 4.4.2). Nachdem die erste Aufgabe die Erstellung von Objekten aus gegebenen Koordinaten erarbeitet, fordert die nächste Aufgabe

Aufgabe 2

Denkt euch nun jeweils ein einfaches Objekt aus und gebt anschließend eurem Partner die Koordinaten des Dreiecksnetzes. Dieser soll das Objekt auf dem Steckbrett abstecken. Erscheint das richtige Objekt?

von den Schülerinnen und Schüler ein umgekehrtes Denken, da nun ein Objekt in Koordinaten überführt werden soll. Die direkte Interaktion mit der Partnerin oder dem Partner soll sozialverantwortliches Handeln fördern, da dieser oder diese von der Korrektheit der eigenen erstellten Aufgabe abhängig ist.

Die dritte und vierte Aufgabe auf dem Arbeitsblatt dienen als Vorbereitung auf die Koordinatentransformationen im nächsten Modulabschnitt. Die dritte Aufgabe

Aufgabe 3

Baut nun das Objekt aus Aufgabe 1 und verlängert die Kanten auf dem Steckbrett um das Doppelte. Lest dann die neuen Koordinaten ab und tragt sie ein:

a) $\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix}$ b) $\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix}$ c) $\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix}$

ist dabei schnell erledigt und lässt wenig Raum für Kreativität. Aber sie ist dennoch wichtig, denn die originale Erfahrung einer Modifikation eines Objekts veranschaulicht, dass sich die Verlängerung der Kanten eines Objekts durch Transformation der Eckpunkte darstellen lässt, was eine wesentliche Einsicht dieser Phase ist [5]. Durch die Erstellung einer Matrix in Aufgabe 4,

Aufgabe 4

a) Wie unterscheiden sich die alten Koordinaten aus Aufgabe 1 von den neuen Koordinaten aus Aufgabe 3? _____

_____

b) Wir suchen eine Matrix M, welche die Vektoren aus Aufgabe 1 auf die Vektoren aus Aufgabe 3 abbildet. Berechnet dazu zunächst

$$M \cdot \begin{pmatrix} 0 \\ 2 \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 2 \end{pmatrix} = \begin{pmatrix} \quad \end{pmatrix}.$$

Dies soll gleich $\begin{pmatrix} 0 \\ 4 \end{pmatrix}$ sein. Was könnt ihr über a, b, c, d sagen? _____

_____

c) Nehmt weitere Vektoren bis ihr die Matrix M komplett bestimmen könnt.

$$M = \begin{pmatrix} \quad \end{pmatrix}$$

welche die Koordinaten auf passende Weise transformiert, soll diese Einsicht noch vertieft und ein Zusammenhang zwischen Schulmathematik und Computergrafik hergestellt werden (Integrationsprinzip [2]). Die Schülerinnen und Schüler sollen mit Objekten gedanklich operieren und aktiv eine Koordinatentransformation ausführen, welche der nächste Modulabschnitt explizit thematisiert. Dazu brauchen die Schülerinnen und Schüler ein gewisses Vorwissen aus der Mathematik, denn ohne Matrizenrechnung geht es nicht. Da es sich aber um eine 2x2-dimensionale Matrix handelt, sollte die Hürde recht gering sein. An dieser Stelle hätte direkt eine 4x4-dimensionale Matrix besprochen werden können, aber:

- 2x2-Matrizen sind übersichtlich und leicht lösbar.
- 2x2-Matrizen sind anschaulich und logisch, da in der Ebene gearbeitet wird.
- 4x4-Matrizen sind korrekter im Hinblick auf die in der Computergrafik beziehungsweise im Laufe des Moduls verwendeten Matrizen.
- 2x2-Matrizen verdeutlichen die Idee und können (später) auf 4x4 **übertragen** werden, da

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{4.1}$$

der Auswirkung nach der 4x4 Matrix

$$\begin{pmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

in der x-y-Ebene entspricht.

Somit war die Entscheidung für den Einstieg über eine 2x2-Matrix leicht. Die Schülerinnen und Schüler werden schon bald auf die „richtigen“ Matrizen treffen.

Nach dieser Arbeitsphase sollten die Schülerinnen und Schüler eine Idee von der Darstellung von Objekten im Computer haben und wissen wie man mit ihnen arbeiten kann. Eine ganz genaue Vorstellung zur Objektdarstellung ist nicht nötig, da dies für die Shaderprogrammierung nur eine geringe Rolle spielt, denn diese arbeiten nur im Anschluss in einem Postprocessing-Step und wie das OpenGL-Programm intern läuft kann man über Shader nicht steuern, man braucht es aber auch nicht (man erinnere sich an den Abschnitt 1.1).

Kurzüberblick Objektdarstellung	
Kernfragen:	• Wie werden Objekte dargestellt? • Wie können Formen von Objekten geändert werden? • Welche Rolle spielen Kanten und Eckpunkte dabei?
Didaktische Kernideen:	• Aktives, selbstständiges Arbeiten. • Haptische Erfahrung.
Medien:	• Steckbrett mit Kordeln. • Arbeitsblatt Objektdarstellung, siehe Abschnitt 6.1.1. • Eventuell Infoblatt Mathematik, siehe Abschnitt 6.2.2.
Methoden:	• Partnerarbeit.
Vorwissen:	• Matrizenrechnung aus der Mathematik.
Mögliche Probleme:	• Probleme mit der Mathematik. Abhilfe: Infoblatt Mathematik. • Was bedeutet um das Doppelte vergrößern? Hier können Hinweise gegeben oder die Lösung durch Beispiele veranschaulicht werden.

4.4.2 Erarbeitungsphase zur Modifikation von Objekten mittels Koordinatentransformationen

Die Beschäftigung mit Koordinatentransformationen ist eine logische Entscheidung, welche sich genetisch aus dem vorangegangen Inhalt ergibt. Denn durch Koordinatentransformationen können die Eckpunkte von Objekten be-

einflusst werden, indem sie in Form von Matrizen an die Eckpunkte multipliziert werden und so die Gestalt, Position und Orientierung des Objekts verändern können. Es werden die Skalierung, die Rotation und die Tranlation thematisiert, die Scherung wird vernachlässigt, da sie nicht so direkt sinnvoll und nachvollziehbar ist wie die anderen genannten.

Dieser Abschnitt des Moduls verwendet zur Vermittlung des Inhalts kein Arbeitsblatt oder ähnliches, sondern die Schülerinnen und Schüler arbeiten mit einem eigens entworfenen Programm zu 2D-3D Koordinatentransformationen. Die Idee und Teile des Interface stammen dabei aus der eLearning Veranstaltung von Prof. Dr. Ulrik Schroeder im Sommersemester 2010, in der dies als Teil eines Tutorials entworfen wurde [11]. Da war dieses Programm aber noch ein Applet, welches nur zweidimensional arbeitete, die Hilfefunktion war viel überladener und viele Details waren anders. In diesem Sinne habe ich das Programm komplett überarbeitet, aber es diente doch als guter Ausgangspunkt.

Durch das Programm sollen die Schülerinnen und Schüler möglichst aktiv, fast schon spielerisch Erfahrungen sammeln indem sie mit dem gegebenen „Haus vom Nikolaus“ geistig operieren und dies in die geforderte Darstellung transformieren. Dabei erhalten sie *direktes visuelles Feedback* in der Ausgabefläche, wie in Abbildung 4.2 zu sehen ist. Die einzelnen Arten von Koordinatentransformationen werden dabei nacheinander besprochen um Verwirrungen und Verwechslungen zu vermeiden, da diese Inhalte für die meisten Schülerinnen und Schüler neu sein dürften und sie somit nach und nach gemeistert werden können. Gemeisterte und noch bevorstehende Aufgaben werden im Programm grafisch dargestellt, so haben die Schülerinnen und Schüler bereits zu Beginn die Ziele und den Ablauf vor Augen. Zu jedem Aufgabenbereich haben die Schülerinnen und Schüler einige Aufgaben, welche die Eingabe einer Matrix zur Transformation des Objekts bedürfen, aber auch der umgekehrte Weg wird angesprochen. Dabei können die Schülerinnen und Schüler das Objekt mittels Maus so ändern, dass am Ende die gewünschte Matrix auf der linken Seite steht. Bei eventuellen Fehlern kann immer wieder die Einheitsmatrix wiederhergestellt werden um neu zu beginnen oder es können die Koordinaten der Eckpunkte eingeblendet werden, so dass die Auswirkung der Matrix auf das Objekt deutlicher wird. Ein Fortschrittsbalken zeigt an, ob man sich auf die Lösung hin bewegt oder auf dem falschen Weg befindet. Der bewusst eingebaute Fertig-Button soll die Schülerinnen und Schüler dazu bewegen, sich zuerst selber Gedanken zu machen ob die Aufgabe richtig ist, bevor sie diese abgeben. Diese Abgabe war einst automatisch, aber dies war für den Lernprozess nicht förderlich, da dann einfaches Ausprobieren reichte um die Aufgabe zu lösen um dann direkt bei der nächsten Aufgabe zu landen ohne Zeit zur Rekapitulation des Getanen zu haben. Der Fortschrittsbalken und die Anzeige beim Drücken des Fertig-Buttons geben den Schülerinnen und Schüler direktes Feedback ob sie sich auf dem richtigen Weg zur Lösung

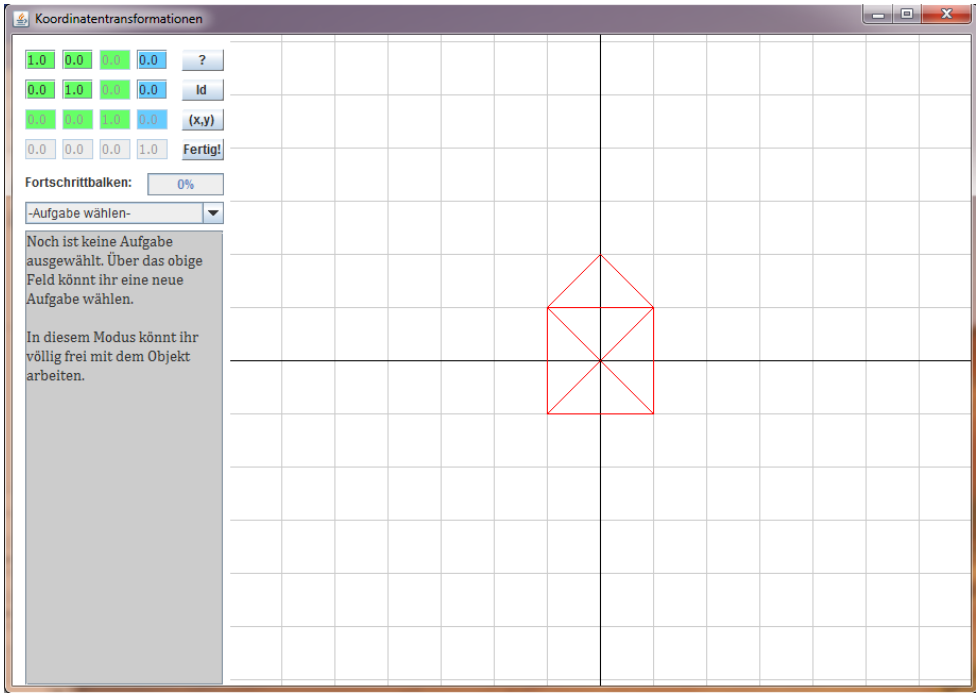


Abbildung 4.2: Programmoberfläche zur spielerischen Erfahrung von 2D-3D Koordinatentransformationen.

der Aufgabe befinden und sollen somit den Lerneffekt steigern [10]. Nach korrektem Abschluss einer Aufgabe erhalten die Schülerinnen und Schüler noch mal die Möglichkeit die End- mit der Ausgangsituation zu vergleichen indem beide gleichzeitig darstellt werden - zumindest in den 2D Aufgaben.

Nachdem die Schülerinnen und Schüler sich im Zweidimensionalen mit dem Stoff vertraut gemacht haben, wechselt das Programm in eine dreidimensionale Ausgabe. Dann wird auch die Matrix so weit zur Bearbeitung frei gegeben, dass die 3D-Kooridnatentransformationen ausgeführt werden können. Diese Bereiche waren vorher von der Bearbeitung ausgeschlossen und ausgegraut, um den Fokus der Schülerinnen und Schüler auf den wichtigen Teilen der Matrix zu lenken. Dieser spezifische Transfer von 2D auf 3D ist wichtig, da mit 3D-Transformationen gearbeitet wird und eine anschauliche Vorstellung in der Ebene meist sehr hilfreich bei der Erstellung von Transformationen im Raum ist. Die 3D-Aufgaben sind problemorientiert gestellt, so will der Nikolaus sein Haus an einer anderen Stelle sehen oder ähnliches. An dieser Stelle wird dann deutlich, warum die 4-dimensionalen Matrizen sinnvoll sind.

Des Weiteren stellt das Programm eine kontextsensitive Hilfefunktion zur Verfügung, welche je nach Aufgabenbereich passende Hinweise gibt, wie in Abbildung 4.3 zu sehen. Anders als in der früheren Programm-Version ist diese Hilfestellung aber keine Komplettlösung mehr, die einem die Lösung

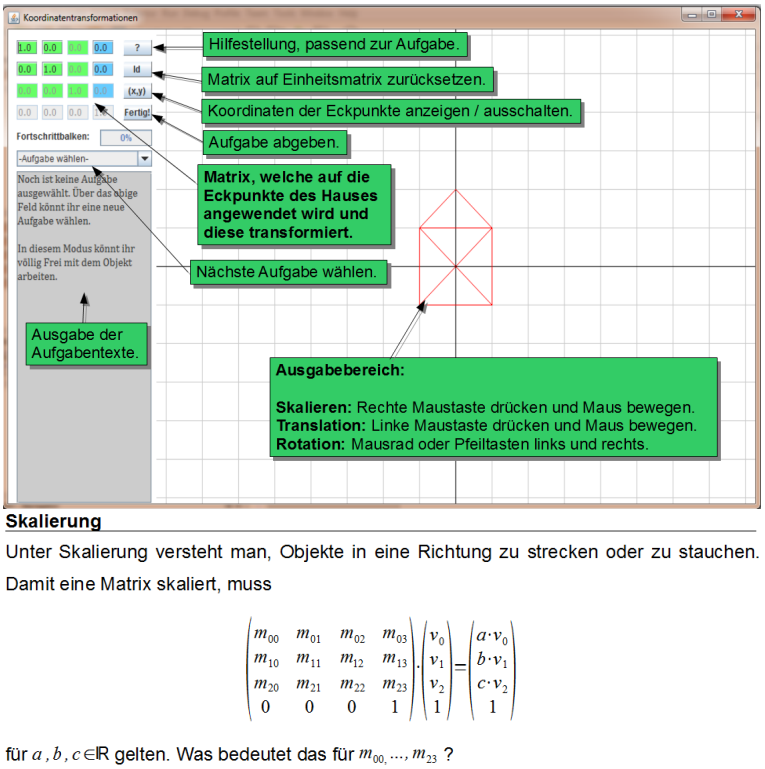


Abbildung 4.3: Kontexabhängige Hilfen. Oben zur Programmoberfläche, unten zur Skalierung.

präsentiert und die einzige Arbeit das Einsetzen der entsprechenden Werte ist. Sondern sie soll Hinweise zur Bearbeitung geben, welche dann eigenständig bis zur Problemlösung verfolgt werden können und eigene Gedankengänge fördern [10]. Ein Sonderfall ist die Rotation, dabei blieb mir aufgrund des komplexen Aufbaus der Matrix keine andere Wahl als die Lösung praktisch vorzugeben und nur noch Einsetzen zur Lösung der Aufgaben zu fordern. Deshalb beschäftigen sich die Aufgaben zur Rotation eher mit der Analyse von Beobachtungen als mit tatsächlich zu lösenden Problemstellungen. Dies ist prinzipiell nichts schlechtes, jedoch vermute ich, dass die meisten Schülerinnen und Schüler eher die Skalierung und Translation verstehen und verinnerlichen werden.

Nach Abschluss der einzelnen Bereiche erarbeiten die Schülerinnen und Schüler noch die Problematik bei verknüpften Transformationen. Einerseits wird der Vorteil von diesen durch die Matrixmultiplikation angedeutet, andererseits wird verdeutlicht, dass die Reihenfolge nicht beliebig ist. Dies üben die Schülerinnen und Schüler an einem Beispiel per Hand durch und überprüfen ihre Ergebnisse im Programm. Hier ist eine Nebenrechnung auf einem Blatt notwendig.

Kurz gesagt sollen die Schülerinnen und Schüler in diesem Abschnitt aktiv Koordinatentransformationen kennen lernen und diese anwenden.

Kurzüberblick Koordinatentransformationen	
Kernfragen:	• Was sind Koordinatentransformationen? • Wie können diese zum Ändern von Objekten genutzt werden? • Was ist Skalierung, Rotation und Translation?
Didaktische Kernideen:	• Aktives, selbstständiges Arbeiten. • Prinzip der minimalen Hilfe. • Spielerisches Lernen. • Direktes, optisches Feedback.
Medien:	• Computer als Medium mit speziellem Programm zu 2D-3D Koordinatentransformationen. • Eventuell Infoblatt Mathematik, siehe Abschnitt 6.2.2.
Methoden:	• Partnerarbeit.
Vorwissen:	• Matrizenrechnung aus der Mathematik.
Mögliche Probleme:	• Probleme mit der Mathematik. Abhilfe: Infoblatt Mathematik. • Probleme mit dem Programm. Abhilfe: Hilfefunktion im Sandbox Modus. • Probleme bei Skalierung, Rotation oder Translation. Erste Abhilfe: Kontextabhängige Hilfefunktion. Zweite Abhilfe: Hinweise aus der Hilfe auf einem Blatt nachrechnen und mit gewünschtem Ergebnis vergleichen lassen.

4.4.3 Sicherungsphase zu Koordinatentransformationen

Diese Phase dient zur Sicherung des Gelernten und stellt somit kein neues Wissen zur Verfügung. Die Schülerinnen und Schüler erhalten zu Beginn das Arbeitsblatt aus Abschnitt 6.1.2 und sollen dieses bearbeiten. Zunächst alleine und bei Aufgabe 3 ist Interaktion mit der Partnerin oder dem Partner gefragt, indem zwei Aufgaben im Stil der bisherigen Aufgaben entworfen werden, welche anschließend gegenseitig bearbeitet werden sollen. Die Schülerinnen und Schüler tragen dabei die Verantwortung für die verständliche und

korrekte Formulierung der Aufgabe. Die letzte Aufgabe dient dann als Dokumentation, damit die Schülerinnen und Schüler auf dieses Wissen später leicht zurückgreifen können, auch ohne diese in den vorangegangen Phasen auswendig gelernt zu haben.

Währenddessen ist die Aufgabe der Betreuerinnen und Betreuer den Schülerinnen und Schülern über die Schulter zu blicken und den Lernerfolg zu evaluieren um eventuell eingreifen zu können. Diese Phase ist ebenso zur Bewertung des Programms interessant, da diese zeigt wie tief die Schülerinnen und Schüler den Stoff begreifen konnten und somit kann es dann eventuell angepasst werden. Da es sich hierbei nur um eine Sicherung handelt, spare ich mir einen Kurzüberblick an dieser Stelle.

4.4.4 Präsentation Shadermanager und OpenGL Shading Language

Die Phase

Diese Phase dient als Einführung in das Programm Shadermanager und in die Programmiersprache GLSL zugleich. Da dies eine wichtige und komplexe Phase ist, findet sich in Abschnitt 6.2.4 ein mehrseitiger Kurzüberblick zur Thematik und eine mögliche Herangehensweise für die Betreuerinnen und Betreuer. Diese Herangehensweise beschreibt einen möglichen Weg wie die Schülerinnen und Schüler einerseits eine Anwendungsschulung im Shadermanager erhalten können aber gleichzeitig auch einen Einblick in die GLSL erhalten sollen, indem gemeinsam ein Programm erstellt und bearbeitet wird. Wichtig ist an dieser Stelle, dass die Betreuerinnen und Betreuer jeden einzelnen Schritt laut mitdenken, so dass diese vom Publikum leicht nachvollzogen werden können.

Diese Phase ist die einzige tatsächliche Lernphase in der die Schülerinnen und Schüler nicht direkt aktiv arbeiten, sondern der Präsentation folgen sollen und währenddessen Vorschläge machen beziehungsweise Fragen stellen können. Je nach Situation wäre daher vor dieser Phase eine Pause möglich um die Konzentration wieder auf ein angemessenes Level zu führen. Die Phase funktioniert nur frontal, da diese den Shadermanager als wesentliches Werkzeug zur Arbeit mit Shadern vorstellt und erste Fehlerquellen beseitigen soll. Obendrein sollen die Schülerinnen und Schüler sehen, dass die GLSL keine neue Programmiersprache oder gar Hexerei ist, sondern einsehen, dass sie sehr ähnlich zu den bereits bekannten ist.

Kurzüberblick Shadermanager und GLSL	
Kernfragen:	• Wie funktioniert der Shadermanager? • Was ist die GLSL?

Kurzüberblick Shadermanager und GLSL (Fortsetzung)	
Didaktische Kernideen:	• Kompakte Präsentation zur Erleichterung des Einstiegs. • Präsentation zur Darstellung der wesentlichen Punkte.
Medien:	• Smartboard oder Beamer.
Methoden:	• Unterrichtsgespräch.
Vorwissen:	• Programmiersprachen ähnlich zur GLSL (C/C++ oder Java).
Mögliche Probleme:	• Vermutlich keine, falls doch, werden diese während der Präsentation geklärt.

Der Shadermanager

Wie bereits im vorherigen Abschnitt angedeutet, handelt es sich bei dem Shadermanager um ein eigens für dieses Modul entwickeltes Programm. Es erfüllt den hoch spezialisierten Zweck, dass Shaderprogramme für die Vertex- und Fragmentshader erstellt und in derselben Oberfläche getestet werden können. Dies ist eine starke Vereinfachung bei der Shaderprogrammierung, da sich die Schülerinnen und Schüler sonst noch um das Laden der Textdateien der Shaderprogramme, deren entsprechende Formatierung und Übersendung an die Grafikkarte, sowie die Kompilierung und Fehlerabfrage kümmern müssten. Auch diese Schritte zu programmieren macht Sinn, da die Schülerinnen und Schüler dadurch ein gesamtes Bild der Shaderprogrammierung erhalten würden und es besser vernetzen und später wiederverwenden könnten. Dennoch habe ich mich dagegen entschieden und den Shadermanager entwickelt um den Fokus der Arbeiten auf den Shaderprogrammen zu halten und die programmiertechnischen Tücken außerhalb der Shaderprogramme zu umgehen. Aus eigener Erfahrung kann ich sagen, dass die Erstellung eines Programms, welches eine 3D-Szene darstellt und beliebige Shader lädt kein einfaches Problem ist, sondern viel Anstrengung kostet und durch produktbedingte Eigenheiten der verwendeten LWJGL (Lightweight Java Game Library - OpenGL für Java) zu einer echten Geduldsprobe werden kann.

Im Shadermanager können die Schülerinnen und Schüler nun in einer aufgeräumten Oberfläche einige Funktionen aufrufen und sich ansonsten komplett auf die wesentlichen Arbeitsschritte konzentrieren. Zur Unterstützung des Lernprozesses beziehungsweise des Arbeitsprozesses wurden verschiedene Funktionen eingebaut:

Passende Hilfe Für den Vertex- und Fragmentshader steht eine eigene Hilfe zur Verfügung, welche als Glossar die benötigten Informationen bereit

hält, wie in Abbildung 4.4 dargestellt.

Undo-Redo Bei fehlerhaften oder ungewollten Änderungen kann durch einen Knopfdruck der letzte Zustand wiederhergestellt werden. Die von Windows gewohnten Tastenkombinationen STRG+Z, sowie STRG+Y funktionieren auch hier.

Automatische Kompilierung Beim Speichern wird der Shaderquellcode automatisch übersetzt und in der Ausgabe angezeigt, sofern keine Fehler vorhanden sind.

Fehlerausgabe Die vom Compiler gemeldeten Fehler werden dem entsprechende Shader zugeordnet und leicht aufbereitet ausgegeben, wie in Abbildung 4.5 zu sehen ist.

Scrollende Zeilennummern Zeilennummern an den Textfeldern helfen beim Debuggen, da der Compiler bei Fehlern eine Zeilennummer ausgibt.

Reset Camera Falls das Objekt durch wilde Kamerafahrten aus dem Sichtfeld verschwinden sollte, kann durch einen Klick wieder die ursprüngliche Situation hergestellt werden.

Reset Timer Dient zum Zurücksetzten der Timer-Variable, die für Animationen verwendet werden kann.

Draw Grid Zeichnet ein Gitternetz unter die Szenen um den Tiefeneindruck zu verstärken beziehungsweise bewusst zu machen und die Größenverhältnisse zu verdeutlichen.

Anpassbare Programmierbereiche Die Programmierbereiche lassen sich in der Höhe anpassen, so dass bei einer Aufgabe welche eher im Vertexshader bearbeitet wird, mehr Platz für diesen Programmierbereich geschaffen werden kann. Dennoch kann man beide Shaderprogramme gleichzeitig im Auge behalten.

Objekt oder Szene wählen Die Schülerinnen und Schüler können sich einige vorgefertigte Objekte über diesen Menüpunkt auswählen und darstellen lassen um dem individuellen Interesse am ehesten zu entsprechen.

Übliche Dateioperationen Speichern, Laden und Erstellen von neuen Shaderdateien geschieht komfortabel über ein Dialogfenster. Dieses erstellt beziehungsweise lädt einen Vertex- und Fragmentshader, bei erstmaliger Erstellung wird ein Grundgerüst vorgegeben.

Das Layout wurde schlicht gehalten um den Fokus auf den Programmierbereichen und der visuellen Ausgabe zu belassen (siehe dazu Abbildung 1.3). Die

Hilfe für den Vertexshader

Hilfe: Vertexshader

In den Kategorien findest du Hilfe zur Programmierung des Vertexshaders

Allgemein | Variablen / Operatoren | Eingebaute Variablen / Funktionen | Uniform / Varying

Eingebaute Variablen (unterschiedlich zum Fragmentshader)

vec4 gl_Position
Muss geschrieben werden und stellt die Position des Vertex, also des Eckpunkts dar.

vec4 gl_Color
Farbe des Eckpunkts (Eingabe).

vec4 gl_FrontColor.
Farbe des Eckpunkts, welche an den Fragmentshader übergeben wird (Ausgabe).

vec3 gl_Normal
Normale.

Eingebaute Funktionen (auch im Fragmentshader vorhanden)

Geometrische Funktionen:
length(vec3 x) liefert Länge eines Vektors,
dot(vec3 x, vec3 y) liefert Skalarprodukt der Vektoren,
cross(vec3 x, vec3 y) liefert Kreuzprodukt der Vektoren,
normalize(vec3 x) liefert den normalisierten Vektor x zurück.

Standardfunktionen:
abs(float x) leifert den Betrag oder Absolutwert von x,
max(float x, float y) liefert Maximum von x und y.
floor(float x) liefert x abgerundet zurück,
ceil(float x) liefert x aufgerundet zurück,

Trigonometrische Funktionen:
sin(float angle),
cos(float angle),
tan(float angle).

Umkehrfunktionen:
asin(float x),
acos(float x).

Exponentielle Funktionen:
pow(float x, float y) berechnet x hoch y,
sqrt(float x) berechnet die Wurzel von x.

Abbildung 4.4: Hilfefunktion des Shadermanagers am Beispiel des Vertexshaders.

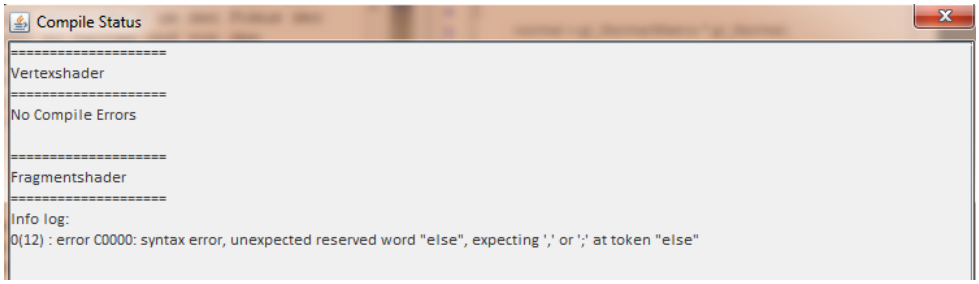


Abbildung 4.5: Bei Fehlern, welche durch den Compiler entdeckt werden, erscheint eine übersichtliche Fehlerausgabe.

dargestellten Objekte sind auch eher schlicht. Dies hat den Grund, dass dadurch die programmierten Effekte schneller sichtbar werden. Des Weiteren ist dies auch technisch bedingt, denn der Shadermanager soll auf möglichst vielen Maschinen einsetzbar und lauffähig sein und kann demnach nicht die komplexesten Objekte darstellen. Zum Beispiel überfordert das Bunny-Objekt schon die Grafikchips der Laptops aus dem Schülerlabor Informatik.

Der Shadermanager hat in der momentanen Version keine mir bekannten Bugs und ist meiner Meinung nach eine gute Hilfe beim Einstieg in die Shaderprogrammierung, da man sich auf diese konzentrieren kann und sich keine Gedanken um die zugrundeliegende Programmierung machen muss.

4.4.5 Übungs- und Erarbeitungsphase durch Animation von Objekten mittels Vertexshaderprogrammierung

Nachdem die vorangegangene Phase den Shadermanager und die GLSL vorgestellt hat, soll dieses Wissen nun durch aktives Arbeiten gefestigt werden. Dazu erstellen die Schülerinnen und Schüler mittels Arbeitsblatt aus Abschnitt 6.1.3 ein einfaches Shaderprogramm um die generelle Struktur kennen lernen zu können und auf einfache Weise bereits interessante Ergebnisse zu erzielen. Dies schult eher das Anwendungswissen bezüglich dieser Programmiersprache, aber deutet auch die Vorteile und die Struktur von Shaderprogrammen an, indem es die Trennung vom Inhalt aufzeigt und somit beliebig wiederverwendet werden kann. Dies können die Schülerinnen und Schüler erfahren, indem sie das gleiche Shaderprogramm auf verschiedene Objekte anwenden. Als Nebeneffekt lernen die Schülerinnen und Schüler Techniken zur algorithmischen Problemlösung.

Im Sinne des Spiralprinzips wird an dieser Stelle bereits die Farbgebung verwendet, ohne sie vorher vertieft und explizit angesprochen zu haben [11]. Für die Lösung der Aufgaben sind tiefe Kenntnisse jedoch nicht nötig. Später im Abschnitt 4.5.1 wird dies explizit aufgegriffen und behandelt. Die grünen Infoboxen sollen Wissenswertes bereitstellen. Der Hinweis auf das Testen des

Programms hat zwei verschiedene Gründe:

- Zum einen ist häufiges Testen eine Modellierungstechnik und sollte insbesondere bei jeder Softwareerstellung beachtet werden. Der Informationsorientierte Ansatz nach Hubwieser verdeutlicht auch den Sinn des Testens um einerseits Syntaxfehler und andererseits semantische Fehler auszumerzen [5].
- Zum anderen hat man bei der Shaderprogrammierung keine andere Möglichkeit die Programme zu debuggen und ist auf kleinschrittiges Testen und optische Kontrolle angewiesen.

Die zweite Aufgabe greift die Koordinatentransformationen als Mittel zur Lösung eines Problems wieder auf. Die Schülerinnen und Schüler sollen ein beliebiges Objekt tanzen lassen. Dies soll die Motivation fördern, insbesondere bei den Schülerinnen. Durch offene Formulierung der Aufgabe kann diese von den eher schlechten Schülerinnen und Schülern in aller Ruhe bearbeitet werden, so dass sich das Objekt vielleicht nur hin und her bewegt. Hingegen haben schnelle Schülerinnen und Schüler die Möglichkeit beliebig komplexe Animationen zu erstellen, zum Beispiel verschiedene verknüpfte Rotationen um das Objekt um den Ursprung und um sich selber drehen zu lassen.

Leider muss bei dieser Aufgabe auf einen anwendungsbedingten Unterschied der GLSL zur Mathematik geachtet werden, da die GLSL die Matrizen transponiert speichert. Dies lässt sich zwar durch einen einfachen Trick umgehen, aber ist dennoch eine meiner Meinung nach unschöne Lösung als eine mit der Mathematik konsistente Darstellung. Eventuell auftretende weitere Probleme bei der Programmierung können mit dem Infoblatt zur Shaderprogrammierung in Abschnitt 6.2.3 begegnet werden.

Zusammenfassend sollen die Schülerinnen und Schüler erste Gehversuche in der Shaderprogrammierung machen und durch interessante Ergebnisse motiviert werden.

Kurzüberblick Animation im Vertexshader	
Kernfragen:	• Wie erstellt man eigene Shaderprogramme? • Was ist und kann der Vertexshader? • Wofür kann man Koordinatentransformationen gebrauchen?
Didaktische Kernideen:	• Spiralprinzip. • Vom Einfachen zum Schweren. • Motivation. • Aktives Lernen. • Wiederholung in anderen Kontexten.

Kurzüberblick Animation im Vertexshader (Fortsetzung)	
Medien:	• Computer + Shadermanager • Arbeitsblatt Animation, siehe Abschnitt 6.1.3. • Eventuell Infoblatt Mathematik, siehe Abschnitt 6.2.2. • Infoblatt GLSL, siehe Abschnitt 6.2.3.
Methoden:	• Partnerarbeit.
Vorwissen:	• Programmiersprachen ähnlich zur GLSL (C/C++ oder Java). • Koordinatentransformationen.
Mögliche Probleme:	• Programmierprobleme. Abhilfe mittels Infoblatt GLSL. • Aufgabe 2 zu offen. Abhilfe schaffen konkrete Vorschläge durch die Betreuerinnen und Betreuer. • Unterschied GLSL vs. Mathematik. Abhilfe durch Beachtung der Hinweise auf dem Arbeitsblatt.

4.5 Phase 2: Farbdarstellung und Normalenvektoren

4.5.1 Erarbeitungsphase zur RGBA-Farbdarstellung

Nach dem Abschluss der ersten Phase und der anschließenden Pause beginnt diese zweite Phase mit der Einführung in die Darstellung von Farben. Dazu erhalten die Schülerinnen und Schüler einerseits das Arbeitsblatt zur Farbdarstellung aus Abschnitt 6.1.4 und eine elektrische Schaltung. Dieses ist selbst entwickelt und hat als Ausgabe eine RGB-LED und als Eingabe drei Drehpotentiometer, welche einen variablen elektrischen Widerstand haben. Der Aufbau ist in Abbildung 4.6 und 4.7 zu sehen. So können die Schülerinnen und Schüler über Drehen der Potentiometer die Farbe der LED beeinflussen, was dem Prinzip des aktuell oft angepriesenen Ambilights oder grob der Monitortechnologie entspricht und so direkt aus der Welt der Schülerinnen und Schüler entstammt (Anm.: In der Praxis werden die Farben nicht über Widerstände gesteuert, aber das Prinzip ist das Gleiche). Die zugrunde liegende Funktionsweise dieser Techniken ist sicherlich für einige Teilnehmerinnen und Teilnehmer interessant. Bautechnisch bedingt dominiert leider die Farbe Blau wie man es von LED-Lichtquellen kennt, das heißt selbst wenn alle Widerstände auf 0 sind, wirkt das Licht weiß-bläulich. Dennoch eignet sich dieses Gerät für den ersten Kontakt mit der Farbdarstellung gut, denn so lernt man dies praktisch kennen und nicht über theoretische Farbwerte [11].

Ziel dieser Einheit ist es nicht nur, dass die Schülerinnen und Schüler nachher wissen, dass (0.2,0.2,0.2) der Farbe Grau entspricht, viel mehr sollen sie ein Gefühl für die additive Farbmischung bekommen, also je mehr und intensivere Farben verwendet werden, desto weißer wird das Licht. Dies verdeutlicht den Unterschied zur subtraktiven Farbmischung aus dem Kunstunterricht oder der Freizeit, welche invers funktioniert. Ein Grundgedanke bei Erstellung der Schaltung war, den Schülerinnen und Schüler auf diese Weise vielfältige Medien bieten zu können und nicht nur am Computer zu arbeiten. Auch wenn das Arbeitsblatt geschlossene Aufgaben bereit hält, bietet diese Einheit Raum für eigene Experimente und Beobachtungen.

Farbdarstellung über RGBA-Werte beziehungsweise RGB-Werte sind sehr weit verbreitet und schon seit längerer Zeit in verschiedensten Anwendungen im Einsatz. Dies ist einerseits ein Vorteil, denn die Schülerinnen und Schüler haben die Möglichkeit etwas zu lernen, was sie häufiger wiedertreffen werden und somit gefestigt wird und auf zukünftige Lebenssitutionen vorbereitet. Andererseits könnte dies zu einer starken Binnendifferenzierung führen, denn manche Schülerinnen und Schüler werden noch nie davon gehört haben, während andere schon ausgiebig mit RGB-Werten gearbeitet haben. Durch den interessanten Aufbau hoffe ich dennoch die Kenner zum Experimentieren ermutigen zu können. Falls dies dennoch nicht reichen sollte, können bereits fertige Schülerinnen und Schüler mit dem nächsten Abschnitt weiter machen und dort direkt mit Farben im Shader arbeiten.

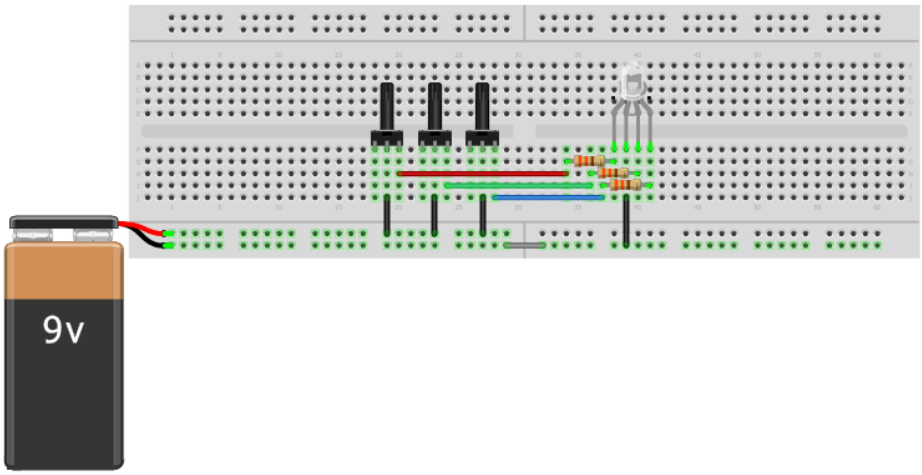


Abbildung 4.6: Darstellung der Schaltung.

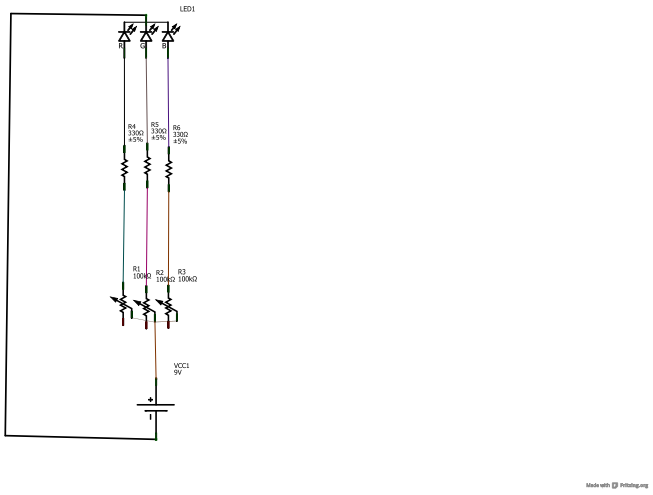


Abbildung 4.7: Schematische Darstellung der Schaltung.

Kurzüberblick Farbdarstellung	
Kernfragen:	• Wie werden Farben technisch repräsentiert? • Was bedeutet RGBA? • Was ist der Unterschied der additiven zur subtraktiven Farbmischung.
Didaktische Kernideen:	• Motivation. • Aktives Lernen. • Prinzip der Lebensnähe und Aktualität.
Medien:	• RGB-LED Schaltung mit Drehpotentiometer. • Arbeitsblatt Farben, siehe Abschnitt 6.1.4.
Methoden:	• Partnerarbeit.
Vorwissen:	• Kein besonderes Vorwissen erforderlich.

Kurzüberblick Farbdarstellung (Fortsetzung)	
Mögliche Probleme:	• Farbdarstellung schon bekannt. Abhilfe: Trotzdem machen und experimentieren lassen. Falls dennoch keine Motivation erkennbar, kann die nächste Aufgabe vorgezogen werden. • Welche Werte sollen eingetragen werden? Abhilfe: Hinweis, dass ungefähre Werte reichen, da die Verhältnisse zwischen den Farben wichtiger sind (0=Aus, 1=Voll und dazwischen schätzen). • Gerät funktioniert nicht. Abhilfe: Batterie tauschen. Falls immer noch nichts zu sehen ist, dann Gerät tauschen und anschließend durch Betreuerin oder Betreuer Verkabelung prüfen (siehe Abbildung 4.7).

4.5.2 Übunsphase durch Simulation eines Comiclooks im Fragmentshader

Diese Phase dient zur Einübung der RGBA-Farbdarstellung und zur weiteren Vertiefung im Bereich der Shadertechnologie, insbesondere wird zum ersten Mal der Fragmentshader verwendet, also programmiert. Dazu startet diese Aufgabe von einem „realen“ Problem aus: Die Schülerinnen und Schüler wollen einen Comiclook für Objekte im Shader erstellen, was durch eine Betreuerin oder einen Betreuer entsprechend motiviert werden sollte. Entweder über direkte Visualisierung des Ergebnisses wie in Abbildung 4.8) oder durch das Präsentieren aktueller Computerspiele und Animationsfilme, welche diese Technik verwenden (z.B. XIII). Comiclook bedeutet dabei, dass mit einfachen Mitteln versucht wird den Objekten einen schwarzen Rand zu verpassen. Dies funktioniert in der Einfachheit leider nur für gekrümmte Objekte wie den Kugeln beim Schneemann oder der Vase. Als Hilfestellung erhalten die Schülerinnen und Schüler das Arbeitsblatt aus Abschnitt 6.1.5, welches auf aktives Arbeiten ausgelegt ist und erstmal noch nicht den Computer verwendet. Zunächst werden die mathematischen Grundlagen zur Winkelberechnung thematisiert und durch optische sowie haptische Darstellung auf dem Steckbrett (siehe Abbildung 4.1) vertieft. Anschließend wird das Programm dann selber konstruiert und als Shaderprogramm in GLSL umgesetzt und getestet. Hierbei besteht wieder nur die Möglichkeit des optischen Debuggens.

In dieser Phase musste das Problem didaktisch reduziert werden, da ein tatsächlicher Toon-Shader recht kompliziert ist. Somit reduziert sich das Hinzufügen der schwarzen Ränder auf einfache Winkelberechnungen und Fallunterscheidungen mittels bedingter Anweisungen im Shader, was für die Schülerinnen und Schüler nach der Einführung über Winkelberechnung eine leichte Aufgabe sein sollte. Dies zeigt wozu diese Technologie mit einfachen Mitteln

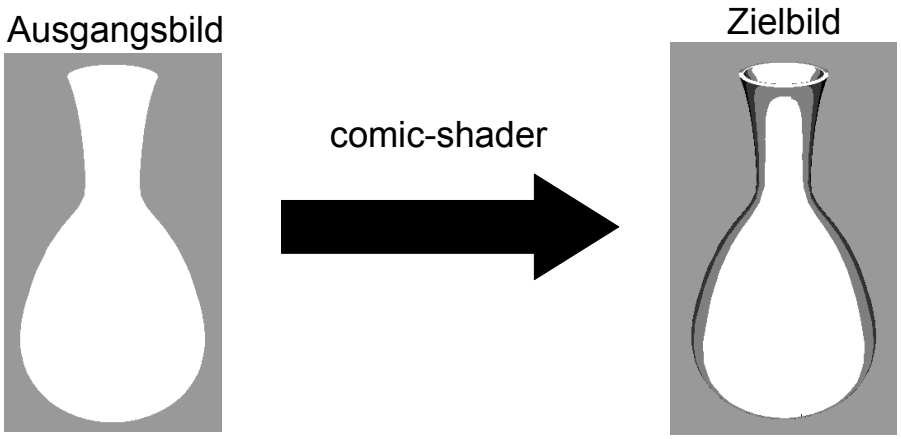


Abbildung 4.8: Mögliche Visualisierung der Zielvorstellung.

in der Lage ist. Der Nachteil dieser Reduktion ist der Verlust der korrekten Darstellung, denn

- der schwarze Rand ist nur abhängig vom Winkel und kann somit an merkwürdigen Positionen erscheinen.
- dies funktioniert nur ordentlich auf gekrümmten Oberflächen, denn bei ebenen Flächen wie bei dem Haus sieht man die Kreisränder.
- der schwarze Rand färbt Teile des Objekts ein und ist kein zusätzlicher schwarzer Rand.

Die Einführung der Winkelberechnung über das Skalarprodukt hat mehrere Hintergedanken. Zum einen spielen Winkel in der Computergrafik und Mathematik eine wichtige Rolle und sind somit einerseits allgemeingültig und andererseits für das Modul grundlegend. Zum anderen stellt dies eine einfache und in der späteren Umsetzung schnelle Möglichkeit der Berechnung dar. Denn die Berechnung eines Skalarprodukts mit eventueller Normalisierung geht auf der Grafikkarte sehr schnell und eignet sich somit hervorragend für die Verwendung im Fragmentshader. Falls die Schülerinnen und Schüler den Zusammenhang zwischen diesem Aufgabenteil und der Shadererstellung nicht erkennen und verwenden ist dies nicht tragisch. Andere Techniken, wie die Verwendung von trigonometrischen Funktionen können auch angewendet werden und somit haben sie die Freiheit eine eigene Lösung für das Problem zu entwickeln.

Die Verwendung von Farben in diesem Abschnitt wird als Vorwissen angesehen und die passenden Farbwerte können über freies Probieren erforscht werden. Deswegen werden hier keine Angaben zur entsprechenden Codierung

der Farben gemacht. Somit festigt sich das Wissen und der spezifische Transfer von Farbdarstellung bei den RGB-LEDs über Potentiometer auf RGBA-Werte in den Shaderprogrammen sollte gelingen.

Am Ende des Abschnitts sollte jede Gruppe ein lauffähiges Programm haben um in der folgenden Phase auf den erlernten Techniken aufbauen zu können.

Kurzüberblick Comiclook im Fragmentshader	
Kernfragen:	• Was ist ein Fragment? • Wie können Winkel zwischen Vektoren bestimmt werden? • Wie simuliert man einen Comiclook? • Wie funktioniert der Fragmentshader?
Didaktische Kernideen:	• Motivation durch Lebensnähe, Aktualität und dem interessanten Ergebnis. • Aktives Lernen. • Problemorientiertes Arbeiten.
Medien:	• Computer+Shadermanager. • Arbeitsblatt Comiclook, siehe Abschnitt 6.1.5. • Infoblatt GLSL, siehe Abschnitt 6.2.3. • Infoblatt Mathematik, siehe Abschnitt 6.2.2.
Methoden:	• Partnerarbeit.
Vorwissen:	• Skalarprodukt. • Vektorrechnung. • Kenntnisse über Winkelberechnung (hilfreich, aber nicht zwingend).

Kurzüberblick Comiclook im Fragmentshader (Fortsetzung)	
Mögliche Probleme:	• Programmierschwierigkeiten. Abhilfe: Eventuelle Fehler gemeinsam lesen und Tipps geben sowie auf das Infoblatt zur GLSL hinweisen. • Mathematische Schwierigkeiten. Abhilfe: Matheblatt. • Kein Ansatz. Abhilfe: Tipps geben anhand des Lösungsvorschlags in Abschnitt 6.1.5. Mögliche Fragen zur Anleitung: In welchem Shader sollt ihr arbeiten? Fragmentshader. Welcher Winkel könnte von Interesse sein? Winkel zwischen Blickrichtung und Normale. Wie hängt der schwarze Rand mit der Winkelberechnung zusammen? Winkel bestimmt die Farbe des Fragments. • Problematische Winkelbestimmung im Shaderprogramm. Dies kann an allerhand Fehlerquellen liegen, am Besten hilft ein geschulter Blick und der Vergleich mit dem Lösungsvorschlag.

4.6 Phase 3: Anwendungsphase durch Programmierung komplexerer Shaderprogramme

4.6.1 Entwicklung der Phase

Diese dritte und letzte Phase hat die größten Entwicklungen bei der Modulerstellung hinter sich. Anfangs war diese Phase als eine abschließende Gruppenarbeit geplant, wo verschiedene, didaktisch aufbereitete Themen der Computergrafik erarbeitet und erstellt würden. Dabei sollten einige Schülerinnen und Schüler sich mit lokaler Beleuchtung, andere mit Bumpmapping und die anderen Gruppen mit weiteren Shadertechniken beschäftigen. Zum Ende des Moduls würden die einzelnen Shader in einem großen Programm gesammelt und gezeigt werden, so dass sich für die Schülerinnen und Schüler der Sinn ihrer Arbeiten im großen Ganzen zeigen und die arbeitsteilige Herangehensweise an ein Projekt schulen würde.

Auf meiner Suche nach weiteren Themen kam die Idee, dass eine Gruppe einen Shader zum stereoskopischen Sehen entwickeln sollte. Diese Idee stieß auf große Begeisterung und es kam der Gedanke auf, dass dieses Thema für alle Schülerinnen und Schüler ein motivierendes Ziel wäre. Ganz im Gegensatz zum Bumpmapping, was eher langweilig nur Struktur auf einer ansonsten flachen Wand hervorruft.

Somit ergab sich nach Besprechung mit der Betreuerin und dem Betreuer eine Umstrukturierung der letzten Phase, so dass für alle Teilnehmerinnen und Teilnehmer die Erstellung des stereoskopischen Shaders möglich ist. Um

ihnen eine möglichst vielseitige Sichtweise auf die Shaderprogrammierung zu bieten, wurde trotzdem noch eine Unterrichtseinheit zur lokalen Beleuchtung eingebaut. Somit arbeiten die Schülerinnen und Schüler im Fragment und im Vertexshader und lernen die verschiedenen Verwendungsmöglichkeiten von Shadern direkt kennen.

4.6.2 Sicherungsaufgabe zur lokalen Beleuchtung

Die dritte Phase unterscheidet sich von den anderen beiden, denn nun wird das erlernte Wissen durch das Bearbeiten von Aufgaben vertieft. Dazu erhalten die Schülerinnen und Schüler zunächst ein Arbeitsblatt zur lokalen Beleuchtung nach dem Modell von Phong, welches in Abschnitt 6.1.6 zu finden ist. Dieses Arbeitsblatt ist offener gehalten und kann auf beliebigem Wege gelöst werden. Im Fokus steht nur das Problem, also wie man eine Szene lokal beleuchten kann. Die Schülerinnen und Schüler sollen eigenständig nach Informationen suchen und diese soweit selbst verstehen, dass ihnen eine Umsetzung in ein lauffähiges Programm gelingt. Jedoch bietet das Blatt einige Hinweise, da dieses Thema nicht trivial ist:

- Leitfragen verdeutlichen die Kernpunkte des Modells.
- Die vorgeschlagene mögliche Herangehensweise lässt Rückschlüsse auf die Programmstruktur zu.
- Die wichtigsten Variablen und Parameter werden aufgelistet.
- Der bisher noch nicht thematisierte Unterschied zwischen Welt- und Objektkoordinaten ist nun wichtig, aber wird nur als programmiertechnische Besonderheit dargestellt, da eine Behandlung den Rahmen des Moduls sprengen würde. Zur Vertiefung kann dies nach dem Ende des Moduls in der Schule oder aus Eigeninteresse wieder aufgegriffen werden.

Dass sich die Schülerinnen und Schüler mit dem Phong-Beleuchtungsmodell beschäftigen ist keine willkürliche Entscheidung. Denn man kann dies als eine didaktisch reduzierte Form der realistischen Beleuchtungsberechnung sehen, welche auf das Wesentliche reduziert wurde, um auf einfache und schnelle Weise schöne Beleuchtungseffekte darzustellen. Ein weiterer Punkt für das Phong-Modell ist die Tatsache, dass dies eine real existierende und in der Praxis genutzte Technik ist, welche sich im Internet leicht finden lässt. Dies ermöglicht zum einen die spezielle Aufgabenstellung, also die Internetrecherche, und zum anderen wird den Schülerinnen und Schülern damit das Gefühl gegeben etwas entwickelt und verstanden zu haben was tatsächlich in der realen Welt verwendet wird. Meiner Meinung nach motiviert dies die Schülerinnen und Schüler zur Bearbeitung und kann etwa Gefühle wie Stolz auf die

eigene Leistung hervorbringen und so zur Persönlichkeits- und Interessenentwicklung im Sinne der intrinsischen Motivation beitragen [11].

Jedoch ist das Phong-Modell an sich nicht der Lerninhalt des Moduls, viel mehr steht im Fokus, dass die Schülerinnen und Schüler

- den Umgang mit der RGBA-Farbdarstellung trainieren,
- sicher die Vektorrechnung in Shaderprogrammen vollziehen und
- ihre Fähigkeiten zur Shaderprogrammierung weiter vertiefen.

Also werden im wesentlichen die vorangegangen Inhalte wieder aufgegriffen, denn das Phong-Modell ist als Thema zu speziell um eine Behandlung im Modul zu begründen.

Durch das selbständige Suchen von Informationen und der aktiven Konstruktion des Programms werden inhaltliche, methodische und soziale Kompetenzen geschult. Beim anschließenden Testen und Debuggen ist die Ausdauer der Schüler gefragt. Um diese zu vergrößern ist ein erfolgreicher Abschluss der Aufgabe wichtig, so dass hier von den Betreuerinnen und Betreuern erwartet wird, jedem so weit zu helfen bis es funktioniert, aber nach Möglichkeit durch minimalste Hilfe um dem eigenen Schaffen der Schülerinnen und Schüler nicht im Wege zu stehen. Hier bietet es sich an, dass die Schnellen diesmal nicht zur nächsten Aufgabe fortschreiten falls sie schon fertig sind, sondern ihren Mitschülerinnen und -schülern helfend unter die Arme greifen, im Sinne des kooperativen Lernens.

Das Debuggen gestaltet sich gewohnt schwierig und kann, falls keine Fehlermeldungen des Compilers ausgeworfen werden, nur über optische Kontrolle geschehen. Zu den einzelnen Farbkomponenten können die Schülerinnen und Schüler leicht ein Vergleichsbild, zum Beispiel auf Wikipedia, finden, siehe dazu Abbildung 2.6. Falls dies nicht gefunden wird, ist auf dem Arbeitsblatt die Lösung bereits als Bild dargestellt. Diese zeigt zwar nicht die einzelnen Komponenten, aber dient dennoch als Vergleichsmöglichkeit.

Kurzüberblick Phong-Beleuchtung	
Kernfragen:	• Was ist das Phong-Beleuchtungsmodell? • Welche Möglichkeiten stecken in der Shaderprogrammierung? • Wie lässt sich ein Problem angehen?
Didaktische Kernideen:	• Ergebnismotivation. • Aktives Lernen. • Eigene Informationsbeschaffung und Umsetzung. • Sicherung durch wiederholtes Üben in einem anderen Kontext.

Kurzüberblick Phong-Beleuchtung (Fortsetzung)	
Medien:	• Computer + Shadermanager. • Arbeitsblatt Phong-Beleuchtungsmodell, siehe Abschnitt 6.1.6. • Infoblatt GLSL, siehe Abschnitt 6.2.3. • Infoblatt Mathematik, siehe Abschnitt 6.2.2.
Methoden:	• Partnerarbeit.
Vorwissen:	• Gelerntes aus den vorherigen Aufgaben (mathematisches, GLSL)
Mögliche Probleme:	• Lichtquellen scheinen von anderen Position berechnet zu werden. Wurde der Hinweis unten auf dem Arbeitsblatt beachtet? • Komplett schwarz. Wird die Farbe gesetzt? • Nur gleichmäßige Färbung. Anzeichen, dass der ambiente Teil richtig ist, aber der Rest nicht. • Fehlende Highlights. Anzeichen für fehlerhaften spekularen Lichtanteil. • Ergebnis sieht nicht aus wie erwartet. Abhilfe: Code mit Lösung vergleichen.

4.6.3 Abschlussaufgabe zum stereoskopischen Sehen

Als letzten Teil des Moduls erwartet die Schülerinnen und Schüler eine Aufgabe zum stereoskopischen Sehen. Dazu erhalten sie das Arbeitsblatt aus Abschnitt 6.1.7 und arbeiten aktiv in Partnerarbeit an den Laptops. Die Aufgabe dient zur Motivation über das Modul hinweg und zeigt die Interdisziplinarität der Informatik durch Verknüpfung von informatischen und biologischen Kenntnissen und Methoden.

Dieses Arbeitsblatt musste stark didaktisch reduziert werden, da bei der Programmierung der verschiedenen Blickrichtungen auf die projektive Geometrie zurückgegriffen wird. Diese kann von den Schülerinnen und Schülern nicht als Vorwissen erwartet werden und in dem Modul wurde sie bis dato nicht behandelt, so dass entweder die Erkenntnisse der projektiven Geometrie und die darauf aufbauenden Inhalte der Computergrafik (die Frustum-Matrix) einfach ohne tiefgreifende Erklärung verwendet werden und die Schülerinnen und Schüler müssen glauben, dass die vorgegeben Formeln stimmen oder die projektive Geometrie wird an dieser Stelle des Moduls eingeführt um

den Weg vom dreidimensionalen Raum auf den flachen Monitor zu verdeutlichen um dann selbstständig die Frustum-Matrix herzuleiten. Beide Vorgehensweisen haben ihre Vor- und Nachteile und nach reichlicher Überlegung habe ich mich für Möglichkeit 1 entschieden, denn

- der Weg von der dreidimensionalen Darstellung auf den Monitor selbst herzuleiten würde ein eigenes Modul füllen können, das heißt die Thematik ist sehr umfangreich und komplex,
- diese letzte Aufgabe dient zur Motivation durch Erstellung eines Produkts der aktuellen Lebenswelt der Schülerinnen und Schüler und soll diese nicht überfordern und
- das Ergebnis der interessanten Herleitung der Frustum-Matrix kann durch die Umsetzung der Formeln in ein lauffähiges Programm veranschaulicht werden und braucht nicht in voller Tiefe theoretisch thematisiert werden.

Ein weiteres Problem des behandelten 3D-Stereo-Shaders stellen die technischen Herausforderungen dar, welche keiner tieferen Logik folgen und somit kaum Einsichten für die Schülerinnen und Schüler bringen. So muss zum Beispiel die Matrix transponiert werden um korrekt zu arbeiten, aber der *transpose*-Befehl ist erst ab Version 1.20 in GLSL verfügbar, was durch das Einfügen von *#version 120* am Anfang des Quellcodes zur Verfügung gestellt wird (warum das nicht ohne weitere Auszeichnung funktioniert wenn die Grafikkarte GLSL 1.20 kann ist mir ein Rätsel, aber es ist nun leider so). Es bleibt demnach keine Wahl als diese Besonderheiten bei der Shaderprogrammierung auf dem Blatt explizit zu erwähnen und als Gesetz ohne weitere Begründung fest zu legen. Problematisch ist auch, dass dies in der Praxis nicht im Shader gemacht werden würde da der Vertexshader keine neuen Eckpunkte erstellen, sondern nur die vorhanden modifizieren kann und somit kein Bild doppelt rendern kann. Somit musste ein Trick verwendet werden um die Objekte einmal aus jeder Blickrichtung darstellen zu können. Dazu werden diese doppelt gezeichnet und mit einem speziellen Wert im Alpha-Farbkanal versehen um die zwei Objekte im Shader voneinander unterscheiden zu können. Das heißt der Alphakanal wird für einen anderen Zweck verwendet als er gedacht ist, was eventuell zur Verwirrung führen kann. Allerdings sollte dann durch die Betreuerinnnen und Betreuer klar gemacht werden, dass dies nur ein Trick zur Unterscheidung der Objekte ist, welcher auch häufig in der Praxis verwendet wird [1].

Wie sich schon bisher andeutet, ist dieser Arbeitsabschnitt weit weniger offen gestaltet als die vorherigen. Dennoch soll durch aktives Arbeiten und dem interessanten Ziel die Verlaufsmotivation hoch gehalten werden. Die Ziele dieser Phase sind eine weitere Vertiefung der Fähigkeiten zur Shaderprogrammierung und eine Schulung der Sach- und Selbstkompetenz indem die

Schülerinnen und Schüler ihre bisher erlernten Informatikkenntnisse in einer völlig neuen Situation anwenden und diese zur Bestätigung der Formeln in ein lauffähiges Programm überführen. Des Weiteren kann man dieses Vorgehen, sich auf die Ergebnisse anderer verlassen zu müssen, als typisch für die Informatik ansehen und somit als Vorbereitung auf zukünftige Lebensituationen, innerhalb sowie außerhalb der Infomatik, verstehen.

Mit dem Abschluss dieser Aufgabe endet das Modul und die Schülerinnen und Schüler können ihre erstellten Programme mitnehmen. Zur Wiederverwendung der Shaderprogramme können sie auch den Shadermanager und das Programm zu den Koordinatentransformationen mitnehmen. Die Shaderprogramme sind jedoch auch selbständig, das heißt in anderen Programmen lauffähig, bis auf das 3D-Stereo Programm und Unterschiede bei der Bezeichnungen der *uniform*-Variablen. Als erste Anlaufstelle für interessierte Schülerinnen und Schüler bietet sich das NeHe Tutorial an, welches dann von den Betreuerinnen und Betreuern bei Bedarf erwähnt werden sollte.

Kurzüberblick 3D-Stereo	
Kernfragen:	• Wie funktionert 3D-Stereo? • Wie lässt sich 3D-Stereo im Shader umsetzen? • Was ist die Frustum-Matrix und was bewirkt sie?
Didaktische Kernideen:	• Ergebnismotivation. • Aktives Lernen. • Sicherung durch wiederholtes Üben in anderem Kontext.
Medien:	• Computer+Shadermanager. • Arbeitsblatt 3D-Stereo, siehe Abschnitt 6.1.7. • Infoblatt GLSL, siehe Abschnitt 6.2.3. • Infoblatt Mathematik, siehe Abschnitt 6.2.2.
Methoden:	• Partnerarbeit.
Vorwissen:	• Gelerntes aus den vorherigen Aufgaben (mathematisches, GLSL)

Kurzüberblick 3D-Stereo (Fortsetzung)	
Mögliche Probleme:	• Programmiertechnische Probleme. Abhilfe: GLSL Infoblatt oder mit der Lösung in Abschnitt 6.1.7 vergleichen. • Welche Werte sollen geändert werden? Abhilfe: Das Bild auf dem Arbeitsblatt hält diese Information bereit. • Es ist nur die Hintergrundfarbe zu sehen. Abhilfe: Wie wurden die Farben im Vertex- bzw. Fragmentshader gesetzt? • Schneller arbeitende Schülerinnen und Schüler. Abhilfe: Sonderaufgaben wie „Bringt doch mal die gezeichneten Kugeln in Bewegung so wie ihr das von einem Sonnensystem kennt.“.

5 Zusammenfassung

Diese Staatsexamensarbeit beschreibt eine für Schülerinnen und Schüler aufbereitete Herangehensweise an die Computergrafik. Das benötigte Vorwissen bedingt die Zielgruppe des Moduls auf Schülerinnen und Schüler der gymnasialen Qualifikationsphase. In den Lernphasen liegt der Fokus auf eigenständiges Lernen in Partnerarbeit, wobei sie die Shaderprogrammierung, die Objektdarstellung als Polygonnetze, die Koordinatentransformationen und die Farbdarstellung über RGBA-Werte durch forschendes Lernen verinnerlichen. Dabei werden sie durch interessant gestaltete und aus ihrer Lebenswelt kommende Themen und Aufgaben extrinsisch motiviert, um auf eine intrinsische Motivation für die Informatik oder speziell für die Computergrafik hin zu arbeiten. Neben den erstellten Arbeitsblättern gibt es noch speziell für dieses Modul entwickelte Infoblätter und Programme, welche alle auf der beigefügten CD zu finden sind. Diese Programme unterstützen den Lerneffekt durch direktes, visuelles Feedback. Damit die Schülerinnen und Schüler für die Behandlung der abstrakten Inhalte gut vorbereitet sind, erhalten sie die Möglichkeit die Farbdarstellung über RGB-LEDs und die Objektdarstellung als Polygonnetze mittels Steckbrett real zu erfahren.

Von den anderen dargestellten Ansätzen zum Thema Computergrafik grenzt sich diese Arbeit besonders durch den komplett vorbereiteten Ablauf ab, wodurch dieses Modul mit wenig zusätzlichem Aufwand im Schülerlabor Informatik der RWTH Aachen umgesetzt werden kann. Die Kurzüberblicke am Ende mancher Abschnitte geben einen Überblick über die darin behandelten Inhalte, die dazu benutzen didaktischen Ideen, die verwendeten Medien und Methoden, das benötigte Vorwissen und bieten Abhilfe für eventuell auftretende Probleme.

Abschließend lässt sich feststellen, dass dieses Modul durch vielfältige Aufgaben und interessante Inhalte das im Studium und in Schule kaum beachtete Thema der Computergrafik ansprechend vermittelt.

6 Anhang: Arbeitsmaterialien mit Lösungsvorschlägen

6.1 Arbeitsblätter

In diesem Bereich befinden sich die erstellen Arbeitsblätter, ein Lösungsvorschlag folgt jedem Aufgabenblatt.

6.1.1 Arbeitsblatt Objektdarstellung

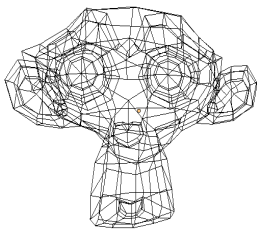
Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Thema: Objektdarstellung

Objektdarstellung

In der Computergrafik werden Objekte häufig als Netze, auch **Mesh** genannt, dargestellt.
Ein Netz besteht aus einer Menge von **Punkten** im Raum, also Vektoren, die durch **Kanten** miteinander verbunden werden. Dies können Dreiecksflächen sein, aber auch andere Vielecke, sprich Polygone, sind möglich.



Aufgabe 1

Steckt auf dem Brett folgende Dreiecksflächen ab. Welches Objekt entsteht?

a) $\begin{pmatrix}-1\\1\end{pmatrix},\begin{pmatrix}-1\\-1\end{pmatrix},\begin{pmatrix}1\\-1\end{pmatrix}$, b) $\begin{pmatrix}1\\1\end{pmatrix},\begin{pmatrix}1\\-1\end{pmatrix},\begin{pmatrix}-1\\1\end{pmatrix}$ c) $\begin{pmatrix}-1\\-1\end{pmatrix},\begin{pmatrix}-1\\1\end{pmatrix},\begin{pmatrix}0\\2\end{pmatrix}$ _____

Aufgabe 2

Denkt euch nun jeweils ein einfaches Objekt aus und gebt anschließend eurem Partner die Koordinaten des Dreiecksnetzes. Dieser soll das Objekt auf dem Steckbrett abstecken. Erscheint das richtige Objekt?

Aufgabe 3

Baut nun das Objekt aus Aufgabe 1 und verlängert die Kanten auf dem Steckbrett um das Doppelte. Lest dann die neuen Koordinaten ab und tragt sie ein:

a) $\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix}$ b) $\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix}$ c) $\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix},\begin{pmatrix}\quad\\\quad\end{pmatrix}$

Aufgabe 4

a) Wie unterscheiden sich die alten Koordinaten aus Aufgabe 1 von den neuen Koordinaten aus Aufgabe 3? _____

_____

b) Wir suchen eine Matrix M, welche die Vektoren aus Aufgabe 1 auf die Vektoren aus Aufgabe 3 abbildet. Berechnet dazu zunächst

$$M\cdot\begin{pmatrix}0\\2\end{pmatrix}=\begin{pmatrix}a & b\\c & d\end{pmatrix}\cdot\begin{pmatrix}0\\2\end{pmatrix}=\begin{pmatrix}\quad\\\quad\end{pmatrix}.$$

Dies soll gleich $\begin{pmatrix}0\\4\end{pmatrix}$ sein. Was könnt ihr über a, b, c, d sagen? _____

_____

c) Nehmt weitere Vektoren bis ihr die Matrix M komplett bestimmen könnt.

$$M=\begin{pmatrix}\quad\\\quad\end{pmatrix}$$

Lösungsvorschlag zum Arbeitsblatt Objektdarstellung

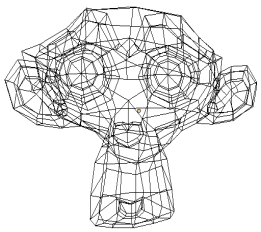
Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Thema: Objektdarstellung

Objektdarstellung

In der Computergrafik werden Objekte häufig als Netze, auch **Mesh** genannt, dargestellt.
Ein Netz besteht aus einer Menge von **Punkten** im Raum, also Vektoren, die durch **Kanten** miteinander verbunden werden. Dies können Dreiecksflächen sein, aber auch andere Vielecke, sprich Polygone, sind möglich.



Aufgabe 1

Steckt auf dem Brett folgende Dreiecksflächen ab. Welches Objekt entsteht?

a) $\begin{pmatrix}-1\\1\end{pmatrix},\begin{pmatrix}-1\\-1\end{pmatrix},\begin{pmatrix}1\\-1\end{pmatrix}$, b) $\begin{pmatrix}1\\1\end{pmatrix},\begin{pmatrix}1\\-1\end{pmatrix},\begin{pmatrix}-1\\1\end{pmatrix}$ c) $\begin{pmatrix}-1\\-1\end{pmatrix},\begin{pmatrix}-1\\1\end{pmatrix},\begin{pmatrix}0\\2\end{pmatrix}$ Es entsteht das „Haus vom Nikolaus“

Aufgabe 2

Denkt euch nun jeweils ein einfaches Objekt aus und gebt anschließend eurem Partner die Koordinaten des Dreiecksnetzes. Dieser soll das Objekt auf dem Steckbrett abstecken. Erscheint das richtige Objekt?

Aufgabe 3

Baut nun das Objekt aus Aufgabe 1 und verlängert die Kanten auf dem Steckbrett um das Doppelte. Lest dann die neuen Koordinaten ab und tragt sie ein:

a) $\begin{pmatrix}-2\\2\end{pmatrix},\begin{pmatrix}-2\\-2\end{pmatrix},\begin{pmatrix}2\\-2\end{pmatrix}$ b) $\begin{pmatrix}2\\2\end{pmatrix},\begin{pmatrix}2\\-2\end{pmatrix},\begin{pmatrix}-2\\2\end{pmatrix}$ c) $\begin{pmatrix}-2\\-2\end{pmatrix},\begin{pmatrix}-2\\2\end{pmatrix},\begin{pmatrix}0\\4\end{pmatrix}$

Aufgabe 4

a) Wie unterscheiden sich die alten Koordinaten aus Aufgabe 1 von den neuen Koordinaten aus Aufgabe 3? *Verdoppelte Einträge*

b) Wir suchen eine Matrix M, welche die Vektoren aus Aufgabe 1 auf die Vektoren aus Aufgabe 3 abbildet. Berechnet dazu zunächst

$$M\cdot\begin{pmatrix}0\\2\end{pmatrix}=\begin{pmatrix}a & b\\c & d\end{pmatrix}\cdot\begin{pmatrix}0\\2\end{pmatrix}=\begin{pmatrix}2\mathrm{b}\\2\mathrm{d}\end{pmatrix}.$$

Dies soll gleich $\begin{pmatrix}0\\4\end{pmatrix}$ sein. Was könnt ihr über a, b, c, d sagen? a,c noch nichts. b=0, d=2

c) Nehmt weitere Vektoren bis ihr die Matrix M komplett bestimmen könnt.

$$M=\begin{pmatrix}2 & 0\\0 & 2\end{pmatrix}$$

6.1.2 Sicherungsblatt Koordinatentransformationen

Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Thema: Sicherungsblatt

Aufgabe 1

Erstelle eine Matrix, die ein Objekt

a) in x-Richtung verdoppelt und in z Richtung halbiert. $M = \begin{pmatrix} _ & _ & _ & _ \\ _ & _ & _ & _ \\ _ & _ & _ & _ \\ 0 & 0 & 0 & 1 \end{pmatrix}$

b) 2 nach rechts und 1 nach unten verschiebt. $M = \begin{pmatrix} _ & _ & _ & _ \\ _ & _ & _ & _ \\ _ & _ & _ & _ \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Aufgabe 2

Beschreibe in eigenen Worten, was folgende Matrix macht.

a) $M = \begin{pmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ _____

b) $M = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ _____

Aufgabe 3

Bereite nun 2 Aufgaben für deinen Partner vor, analog zu den vorherigen Aufgaben. Tauscht anschließend eure Aufgabenblätter und bearbeitet die Aufgaben.

Aufgabe 1

_____

_____

Aufgabe 2

$M = \begin{pmatrix} _ & _ & _ & _ \\ _ & _ & _ & _ \\ _ & _ & _ & _ \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Aufgabe 4

Notiere die allgemeine Form (d.h. mit Variablen) einer Skalierung, Rotation (in einer Ebene) und Translation.

$S = \begin{pmatrix} _ & _ & _ & _ \\ _ & _ & _ & _ \\ _ & _ & _ & _ \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $R = \begin{pmatrix} _ & _ & _ & _ \\ _ & _ & _ & _ \\ _ & _ & _ & _ \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $T = \begin{pmatrix} _ & _ & _ & _ \\ _ & _ & _ & _ \\ _ & _ & _ & _ \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Lösungsvorschlag zum Sicherungsblatt
Koordinatentransformationen

Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Thema: Sicherungsblatt

Aufgabe 1

Erstelle eine Matrix, die ein Objekt

a) in x-Richtung verdoppelt und in z Richtung halbiert. $M=\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

b) 2 nach rechts und 1 nach unten verschiebt. $M=\begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Aufgabe 2

Beschreibe in eigenen Worten, was folgende Matrix macht.

a) $M=\begin{pmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ Objekt um 2 nach links, 1 nach oben und 2 nach hinten / vorne verschieben.

b) $M=\begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ 90° Drehung gegen USZ.

Aufgabe 3

Bereite nun 2 Aufgaben für deinen Partner vor. Wenn ihr beide fertig seid, tauscht die Aufgaben aus und bearbeitet diese.

Aufgabe 4

Notiere die allgemeine Form (d.h. mit Variablen) einer Skalierung, Rotation (in einer Ebene) und Translation.

$$S=\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R=\begin{pmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad T=\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

6.1.3 Arbeitsblatt Animation

Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Thema: Animation im Vertexshader

Aufgabe 1

Ziel: Erstellung eines ersten Shaderprogramms.

a) Erstellt ein neues Shaderprogramm im Shadermanager.

b) Überschreibt im Vertexshader die Farbe jedes Eckpunkts (*vec4 gl_FrontColor*) mit der Farbe rot (*vec4(1.0,0.0,0.0,1.0)*).

   Testet das Programm. Das Objekt sollte nun rot sein.

c) Setzt nun die Farbe jedes Eckpunkts durch seine Normale (*vec3 gl_Normal*), also soll die Richtung der Normalen die Farbe des Eckpunkts (*vec4 gl_FrontColor)* festlegen.

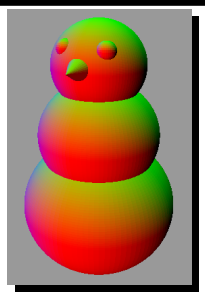
Shaderprogramme

Shaderprogramme sind kleine Programme, welche direkt auf der Grafikkarte kompiliert und ausgeführt werden.

Vertexshader: Der Vertexshader arbeitet auf den Eckpunkten der Polygonnetze.
Fragmentshader: Folgt später.

Zugriff auf Komponenten der Vektoren

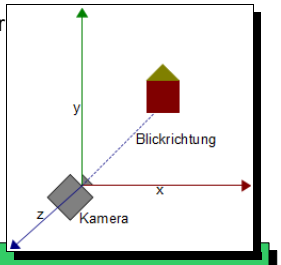
Auf die Komponenten eines Vektors v kann mittels
v.x auf die erste,
v.y auf die zweite,
v.z auf die dritte und
v.w auf die vierte
Komponente zugegriffen werden.



Aufgabe 2

Animiert nun euer Objekt mit den bekannten Koordinatentransformationen. Dazu steht euch eine Variable *uniform float timer* zur Verfügung, um ein Objekt zu bewegen. Diese Variable zählt von 0.0 hoch. Lasst euer Objekt auf beliebige Art **tanzen**. Testet dies auch bei anderen Objekten. Dazu

- *vec4 pos = gl_ModelViewMatrix*gl_Vertex;* liefert die Position der Eckpunkte im Koordinatensystem der Kamera, wie das Bild zeigt.
  - Das Koordinatensystem liegt im Ursprung.
  - Blickrichtung entlang der negativen z-Achse.
  - x- und y-Achse sind parallel zur Bildfläche (das was ihr seht)
- *gl_Position = gl_ProjectionMatrix*pos;* dient nach euren Transformationen dazu, die Punkte auf den Monitor zu projizieren.



Achtung :
Im Vergleich zur Mathematik speichert OpenGL die Matrizen transponiert, d.h. an der Diagonalen gespiegelt. Als Lösung bietet es sich an mittels *transpose(Matrix)* die Matrix zu transponieren. Zur Verwendung des Befehls muss dann *#version 120* an den Anfang des Shaderprogramms geschrieben werden.

Lösungsvorschlag zum Arbeitsblatt Animation

Listing 6.1: Vertexshader rot.vert

```
1 void main()
2 {
3     gl_Position = gl_ModelViewProjectionMatrix *
          gl_Vertex;
4     //Farbe des Eckpunkts auf rot setzen.
5     gl_FrontColor    = vec4(1.0,0.0,0.0,1.0);
6 }
```

Listing 6.2: Fragmentshader rot.frag

```
1 void main()
2 {
3     gl_FragColor = gl_Color;
4 }
```

Listing 6.3: Vertexshader normal.vert

```
1 void main()
2 {
3     gl_Position = gl_ModelViewProjectionMatrix *
          gl_Vertex;
4     // Farbe des Eckpunkts durch die Normale bestimmt
5     gl_FrontColor    = vec4(gl_Normal.x,gl_Normal.y,
          gl_Normal.z,1.0);
6 }
```

Listing 6.4: Fragmentshader normal.frag

```
1 void main()
2 {
3     gl_FragColor = gl_Color;
4 }
```

Listing 6.5: Vertexshader animation.vert

```
#version 120 // Wird für den transpose-Befehl gebraucht

// Variablen von außen
uniform float timer;

void main()
{
    // Eckpunkte in Koordinatensystem der Kamera bringen
    vec4 pos =  gl_ModelViewMatrix*gl_Vertex;

    // Koordinatentransformation anwenden
    pos = transpose(mat4(1.0,0.0,0.0,sin(2.0*timer),
                0.0,1.0,0.0,cos(timer),
                0.0,0.0,1.0,0.0,
                0.0,0.0,0.0,1.0))*pos;

    // Szene auf den Monitor projizieren
    gl_Position =  gl_ProjectionMatrix*pos;
    gl_FrontColor   = gl_Color;
}
```

Listing 6.6: Fragmentshader animation.frag

```
void main()
{
    gl_FragColor = gl_Color;
}
```

6.1.4 Arbeitsblatt Farben

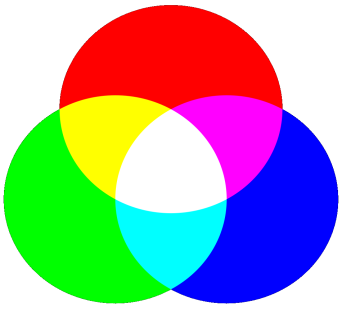
Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Thema: Farbdarstellung

Fardarstellung

Die Darstellung von Farben geschieht häufig über das RGBA-Farbsystem. Das steht für Red-Green-Blue-Alpha, womit man die **Intensität** des jeweiligen **Farbkanals** Rot, Grün oder Blau angibt. Der Alphawert beschreibt, wie durchsichtig ein Material ist, also wie gut dahinter liegende Objekte sichtbar sind.



Aufgabe 1

Für die folgenden Aufgaben verwendet bitte die RGB-Leds mit den angebrachten Reglern. Notiert die ungefähre Intensität in der Tabelle (0=Aus, 1=Voll), wobei der Alphawert ignoriert werden kann.

	Rot	Grün	Blau
Blau	0	0	1
Schwarz			
Rot			
Türkis			
Pink			
Hellgrün			

Aufgabe 2

Erkennt nun die Farben aus den gegebenen Intensitäten und vervollständigt die Tabelle.

	Rot	Grün	Blau
	0	1	0
	1	1	0
	1	0	1
	0.2	0.1	0
	0.5	0.5	0.5
	1	1	1
	1	0.5	0

Lösungsvorschlag zum Arbeitsblatt Farben

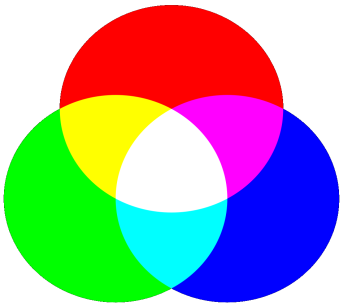
Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Thema: Farbdarstellung

Fardarstellung

Die Darstellung von Farben geschieht häufig über das RGBA-Farbsystem. Das steht für
Red-Green-Blue-Alpha,
womit man die **Intensität** des jeweiligen **Farbkanals** Rot, Grün oder Blau angibt. Der Alphawert beschreibt, wie durchsichtig ein Material ist, also wie gut dahinter liegende Objekte sichtbar sind.



Aufgabe 1

Für die folgenden Aufgaben verwendet bitte die RGB-Leds mit den angebrachten Reglern. Notiert die ungefähre Intensität in der Tabelle (0=Aus, 1=Voll), wobei der Alphawert ignoriert werden kann.

	Rot	Grün	Blau
Blau	0	0	1
Schwarz	0	0	0
Rot	1	0	0
Türkis	0	1	1
Pink	1	0	1
Hellgrün	0.25	1	0.25

Aufgabe 2

Erkennt nun die Farben aus den gegebenen Intensitäten und vervollständigt die Tabelle.

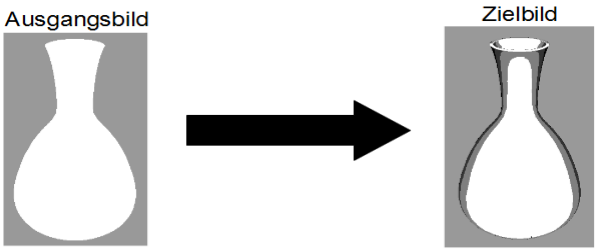
	Rot	Grün	Blau
Grün	0	1	0
Gelb	1	1	0
Pink	1	0	1
Braun	0.2	0.1	0
Grau	0.5	0.5	0.5
Weiß	1	1	1
Orange	1	0.5	0

6.1.5 Arbeitsblatt Comiclook

Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Thema: Comiclook im Fragmentshader



Aufgabe 1

Bestimmt zunächst folgende Winkel und steckt die jeweiligen Vektoren auf dem Steckbrett ab:

$\vec{v}=\begin{pmatrix}1\\0\end{pmatrix}, \vec{w}=\begin{pmatrix}0\\1\end{pmatrix}$ => $\cos\alpha=$_____, $\alpha=$_____

$\vec{v}=\begin{pmatrix}1\\0\end{pmatrix}, \vec{w}=\begin{pmatrix}1\\0\end{pmatrix}$ => $\cos\alpha=$_____, $\alpha=$_____

$\vec{v}=\begin{pmatrix}1\\1\end{pmatrix}, \vec{w}=\begin{pmatrix}0\\1\end{pmatrix}$ => $\cos\alpha=$_____, $\alpha=$_____

$\vec{v}=\begin{pmatrix}1\\0\end{pmatrix}, \vec{w}=\begin{pmatrix}2\\1\end{pmatrix}$ => $\cos\alpha=$_____, $\alpha=$_____

$\vec{v}=\begin{pmatrix}1\\0\end{pmatrix}, \vec{w}=\begin{pmatrix}1\\2\end{pmatrix}$ => $\cos\alpha=$_____, $\alpha=$_____

Aufgabe 2

Erstellt nun ein Shaderprogramm, welches den Comiclook wie oben im Zielbild zu sehen, simuliert.

a) Übergebt die Normale als **varying-Variable** vom Vertex an den Fragmentshader.

   Tipp:
   *normal = gl_NormalMatrix*gl_Normal;* wobei normal die varying-Variable ist.

b) Abhängig vom Winkel zwischen der Blickrichtung $(0.0, 0.0, -1.0)$, und der Normalen sollen verschiedene Farben ausgeben werden, zum Beispiel:

Fragmentshader

Der Fragmentshader arbeitet nach dem Vertexshader und verarbeitet dessen Eingaben zur Bestimmung der **Farbe** eines Fragments.

Er wird für **jedes Fragment** einmal ausgeführt. Ein Fragment ist ein Pixel mit Informationen über die Szene, zum Beispiel Materialwerte oder Tiefeninformation.

Winkel α	Kosinus des Winkels $\cos(\alpha)$	Farbe
Zwischen 90° und ca. 60°	Zwischen 0 und 0.45	Schwarz
Zwischen 60° und ca. 50°	Zwischen 0.45 und 0.6	Grau
Sonst	Sonst	Egal z.B. gl_Color

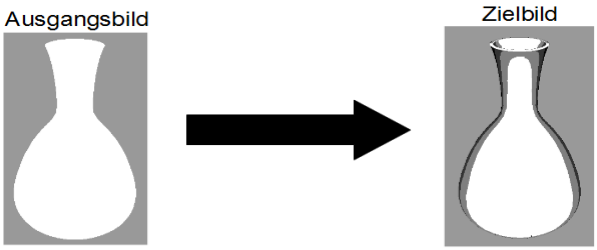
Lösungsvorschlag zum Arbeitsblatt Comiclook

Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik

Thema: Comiclook im Fragmentshader



Aufgabe 1

Bestimmt zunächst folgende Winkel und steckt die jeweiligen Vektoren auf dem Steckbrett ab:

$\vec{v}=\begin{pmatrix}1\\0\end{pmatrix}, \vec{w}=\begin{pmatrix}0\\1\end{pmatrix}$ => $\cos\alpha=0, \alpha=90°$

$\vec{v}=\begin{pmatrix}1\\0\end{pmatrix}, \vec{w}=\begin{pmatrix}1\\0\end{pmatrix}$ => $\cos\alpha=1, \alpha=0°$

$\vec{v}=\begin{pmatrix}1\\1\end{pmatrix}, \vec{w}=\begin{pmatrix}0\\1\end{pmatrix}$ => $\cos\alpha=\frac{1}{\sqrt{(2)}}\approx 0.71, \alpha=45°$

$\vec{v}=\begin{pmatrix}1\\0\end{pmatrix}, \vec{w}=\begin{pmatrix}2\\1\end{pmatrix}$ => $\cos\alpha=\frac{2}{\sqrt{(5)}}\approx 0.89, \alpha\approx 26.57°$

$\vec{v}=\begin{pmatrix}1\\0\end{pmatrix}, \vec{w}=\begin{pmatrix}1\\2\end{pmatrix}$ => $\cos\alpha=\frac{1}{\sqrt{(5)}}\approx 0.45, \alpha\approx 63.43°$

Aufgabe 2

Erstellt nun ein Shaderprogramm, welches den Comiclook wie oben im Zielbild zu sehen, simuliert.

a) Übergebt die Normale als **varying-Variable** vom Vertex an den Fragmentshader.

Tipp:
*normal = gl_NormalMatrix*gl_Normal;* wobei normal die varying-Variable ist.

b) Abhängig vom Winkel zwischen der Blickrichtung $(0.0, 0.0, -1.0)$, und der Normalen sollen verschiedene Farben ausgeben werden, zum Beispiel:

Fragmentshader

Der Fragmentshader arbeitet nach dem Vertexshader und verarbeitet dessen Eingaben zur Bestimmung der **Farbe** eines Fragments.

Er wird für **jedes Fragment** einmal ausgeführt. Ein Fragment ist ein Pixel mit Informationen über die Szene, zum Beispiel Materialwerte oder Tiefeninformation.

Winkel α	Kosinus des Winkels $\cos(\alpha)$	Farbe
Zwischen 90° und ca. 60°	Zwischen 0 und 0.45	Schwarz
Zwischen 60° und ca. 50°	Zwischen 0.45 und 0.6	Grau
Sonst	Sonst	Egal z.B. gl_Color

Listing 6.7: Vertexshader comic.vert

```
1  // Wird an den Fragmentshader interpoliert übergeben
2  varying vec3 normal;
3
4  void main()
5  {
6      gl_Position = gl_ModelViewProjectionMatrix *
           gl_Vertex;
7      gl_FrontColor    = gl_Color;
8
9      // Transformation der Normale in Kamerakoordinaten
10     normal = gl_NormalMatrix*gl_Normal;
11 }
```

Listing 6.8: Fragmentshader comic.frag

```
1  // Übergeben vom Vertexshader
2  varying vec3 normal;
3
4  void main()
5  {
6      vec4 color = gl_Color;
7
8      // Fallunterscheidung: Je nach Winkel zwischen der
           Blickrichtung 0,0,-1 und der Oberflächennormalen
           wird die Farbe gewählt
9      if(abs(dot(normalize(normal),vec3(0.0,0.0,-1.0)))
           <0.45){
10         color = vec4(0.0,0.0,0.0,1.0);
11     } else if(abs(dot(normalize(normal),vec3
           (0.0,0.0,-1.0)))<0.6) {
12         color = vec4(0.5,0.5,0.5,1.0);
13     } else{
14         color= gl_Color;
15     }
16     gl_FragColor = color;
17 }
```

6.1.6 Arbeitsblatt Phong-Beleuchtungsmodell

Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Thema: Lokale Beleuchtung

Die Grundlagen

Eure Aufgabe ist es, euch einen **Überblick über die Thematik** zu verschaffen und zu verstehen wie die einzelnen Komponenten des Programms zusammenwirken.

Phong-Beleuchtung

Die Phong-Beleuchtung ist eine Beleuchtungssimulation die **nicht** physikalisch korrekt ist, aber zu schönen Ergebnissen führt.

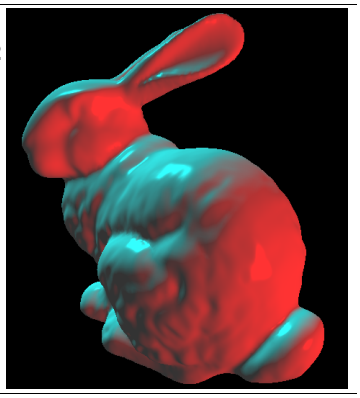
Aufgrund der Einfachheit des Konzepts und der Programmierung eignet es sich besonders für die ersten Schritte in der Computergrafik.

Leitfragen:

- Woraus **besteht** Licht nach diesem Modell und wie hängen die Teile zusammen?
- Was ist mit **ambient** gemeint?
- Was mit **diffuse**?
- Und was ist **specular**?
- Wie könnte ein Programm aussehen, welches das Problem löst?

Das Programm

Erstellt nun ein Shaderprogramm, welches eine Szene lokal nach dem Phong-Modell beleuchtet. Eingebaut sind **2 Lichtquellen** welche über gl_LightSource[0] bzw. gl_LightSource[1] angesprochen werden können.



Mögliche Herangehensweise:

Schritt 1: Erstellt die grundlegende Struktur des Programms.

Schritt 2: Implementiert den ambienten Teil und testet.

Schritt 3: Implementiert den diffusen Teil und testet.

Schritt 4: Implementiert den specularen Teil und testet.

Wichtige eingebaute Variablen für die Programmierung

Kameraposition: uniform vec3 camera

Materialkonstanten:
gl_FrontMaterial.ambient
gl_FrontMaterial.diffuse
gl_FrontMaterial.specular

Lichtparamter der Lichtquelle i:
gl_LightSource[i].position
gl_LightSource[i].ambient
gl_LightSource[i].diffuse
gl_LightSource[i].specular

Wichtig: **gl_Vertex** und **gl_LightSource[i].position** müssen vor Verwendung noch **in Kamerakoordinaten überführt** werden. Dies erledigen die Matrix-Vektorprodukte:

gl_ModelViewMatrix*gl_Vextex bzw. gl_ModelViewMatrix*gl_LightSource[i].position.

Lösungsvorschlag zum Arbeitsblatt Phong-Beleuchtungsmodell

Listing 6.9: Vertexshader Phong.vert

```
1  // Diese Variablen werden interpoliert an den
       Fragmentshader übergeben
2  varying vec3 normal;
3  varying vec4 position;
4
5  void main()
6  {
7      gl_Position = gl_ModelViewProjectionMatrix *
           gl_Vertex;
8
9      // Transformation der Position in Kamerakoordinaten
10     position =   (gl_ModelViewMatrix* gl_Vertex);
11
12     // Transformation der Normale in Kamerkoordinaten
13     normal = normalize(gl_NormalMatrix*gl_Normal);
14
15     gl_FrontColor   = gl_Color;
16 }
```

Listing 6.10: Fragmentshader Phong.frag

```
1  // Übergeben vom Vertexshader
2  varying vec3 normal;
3  varying vec4 position;
4
5  // Kameraposition vom eigentlichen OpenGL-Programm
       übergeben
6  uniform vec3 camera;
7
8  // Ambienter Anteil des Lichts
9  vec4 ambient(int i)
10 {
11     return gl_FrontMaterial.ambient*gl_LightSource[i].
           ambient;
12 }
13
14 // Diffuser Anteil des Lichts
15 vec4 diffuse(int i)
16 {
17     // Vektor vom Fragment zur Lichtquelle
```

```
18     vec3 l = (gl_ModelViewMatrix*gl_LightSource[i].
           position-position).xyz;
19
20     // Winkelbestimmung über Skalarprodukt
21     return gl_FrontMaterial.diffuse*gl_LightSource[i].
           diffuse*max(dot(normal,l),0.0)/length(l);
22 }
23
24 // Bestimmung des perfekt reflektierten Lichtstrahls
25 vec3 r(int i)
26 {
27     mat3 temp = mat3(normal.x*normal.x,normal.x*normal.y
           ,normal.x*normal.z,
28                      normal.x*normal.y,normal.y*normal.y,
                           normal.y*normal.z,
29                      normal.x*normal.z,normal.z*normal.y,
                           normal.z*normal.z);
30     mat3 id = mat3(1.0,0.0,0.0,
31                    0.0,1.0,0.0,
32                    0.0,0.0,1.0);
33     return (2.0*temp-id)* (gl_ModelViewMatrix*
           gl_LightSource[i].position-position).xyz;
34 }
35
36 // Specularer Anteil des Lichts
37 vec4 specular(int i)
38 {
39     // Vektor zwischen Fragment- und Kameraposition
40             vec3 c = ((vec4(camera,1.0))-position).xyz;
41
42     // Reflektionsrichtung
43     vec3 r =r(i);
44
45     // Bestimmung des Winkels zwischen c und r über
           Skalarprodukt. pow(x,y) liefert "x hoch y"
46     return gl_FrontMaterial.specular*gl_LightSource[i].
           specular*pow(max(dot(r,c)/length(r)/length(c)
           ,0.0),gl_FrontMaterial.shininess);
47 }
48
49 void main()
50 {
51     // Startfarbe
```

```
52     vec4 color =  vec4(0.0,0.0,0.0,1.0);
53
54     // Lichtquellen aufsummieren
55     for(int i=0;i<2;i++)
56     {
57         color += ambient(i)+diffuse(i)+specular(i);
58     }
59     gl_FragColor =color;
60 }
```

6.1.7 Arbeitsblatt 3D-Stereo

Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Thema: Stereo sehen

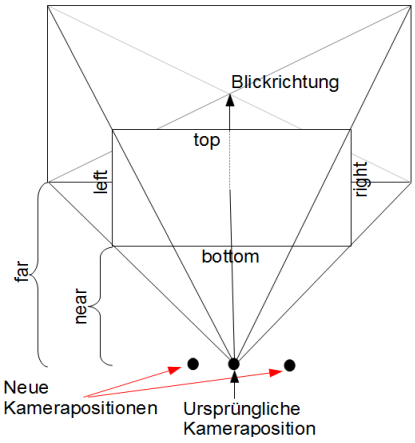
Aufgabe

Ziel der Aufgabe ist es, einen **3D-Tiefeneffekt** zu erzeugen.

a) Fügt *#version 120* und die notwendigen uniform-Variablen oben im Vertexshader ein (alle Typ float):

top, bottom, left, right, near, far

b) Erstellt nun eine Frustum-Matrix, die das Objekt einmal um 0.001 nach links und einmal um 0.001 nach rechts verschoben darstellt. Jedes Objekt wurde doppelt gerendert und kann über den **Alpha-Wert der Farbe unterschieden** werden (1.0 oder 0.99).



Was ist eine Frustum-Matrix?

Frustum ist englisch und bedeutet Pyramidenstumpf. Ähnlich zur Fotografie, liegen Szenen in der Computergrafik in einem festen Bereich, der die Form eines Pyramidenstumpfs hat und durch die Werte top, bottom, near, far, left und right in Bezug zu der Blickrichtung definiert ist (siehe Bild). Durch die Frustum-Matrix wird die Szene perspektivisch verzerrt und auf dem flachen Monitor ausgegeben. Dies machen wir uns für den 3D-Effekt zu Nutze:
Zeichnet das Bild zweimal. Einmal aus einer nach links und ein anderes Mal aus einer nach rechts verschobenen Kameraposition. Welche Werte sind dafür zuständig?

$$frustum := \begin{pmatrix} \frac{2.0 \cdot near}{(right-left)} & 0.0 & \frac{(right+left)}{(right-left)} & 0.0 \\ 0.0 & \frac{2.0 \cdot near}{(top-bottom)} & \frac{(top+bottom)}{(top-bottom)} & 0.0 \\ 0.0 & 0.0 & \frac{-(far+near)}{(far-near)} & \frac{-2.0 \cdot far \cdot near}{(far-near)} \\ 0.0 & 0.0 & -1.0 & 0.0 \end{pmatrix}$$

c) Setzt die Farbe des Vertex so, dass das

- nach rechts verschobene Objekt nur den **Grün und Blauanteil** enthält und das
- nach links verschobene Objekt nur den **Rotanteil** enthält.

d) Legt die neue Vertexposition durch

```
gl_Position = transpose(frustum)*gl_ModelViewMatrix*gl_Vertex;
```

fest.

Lösungsvorschlag zum Arbeitsblatt 3D-Stereo

Listing 6.11: Vertexshader 3D.vert

```
1  // Wird für den transpose-Befehl benötigt
2  #version 120
3
4  // Definierende Variablen des Sichtbereichs, welche vom
       Programm an den Shader übergeben werden
5  uniform float top;
6  uniform float bottom;
7  uniform float far;
8  uniform float near;
9  uniform float left;
10 uniform float right;
11
12 void main()
13 {
14     float r = right;
15     float l = left;
16     vec4 color= gl_Color;
17     vec4 pos = gl_ModelViewMatrix*gl_Vertex;
18
19     // Objekt einfärben und Blickwinkel einstellen, je
           nachdem welches Auge diesen Teil sieht
20     if(gl_Color.w == 1.0)
21     {
22         r = r+0.001;
23         l = l+0.001;
24         color.y=0.0;
25         color.z=0.0;
26     }else{
27         r = r-0.001;
28         l = l-0.001;
29         color.x = 0.0;
30     }
31
32     // Aufstellen der Frustum-Matrix, welche für
           perspektivische Verzerrung notwendig ist
33     mat4 frustum = mat4(2.0*near/(r-l),  0.0, (r+l)/(r-l
           ),0.0,
34                         0.0,2.0*near/(top-bottom), (top+bottom)
                               /(top-bottom),0.0,
```

```
                    0.0,0.0, -(far+near)/(far-near),-2.0*far
                        *near/(far-near),
                    0.0,0.0, -1.0,0.0);

    // Transformieren der Eckpunkte auf die neue
        Position
    gl_Position = transpose(frustum)*gl_ModelViewMatrix*
        gl_Vertex;

    // Einfärben der Objekte, wobei diesmal die Farben
        additiv gemischt werden (im Programm außerhalb
        des Shaders)
    gl_FrontColor   = color;
}
```

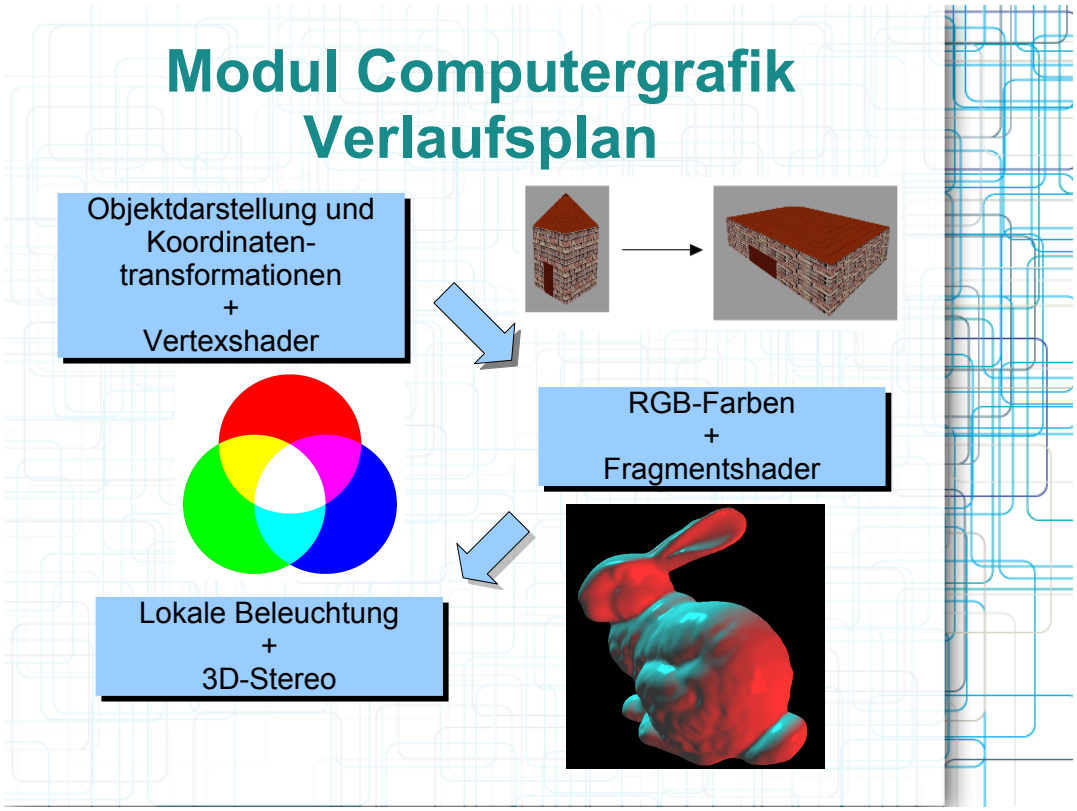
Listing 6.12: Fragmentshader 3D.frag

```
void main()
{
    gl_FragColor = gl_Color;
}
```

6.2 Infoblätter

In diesem Bereich wurden die Infoblätter und Präsentationsmaterialien gesammelt. Die Infoblätter zur Mathematik und GLSL-Programmierung können über das ganze Modul als Hilfestellung für die Schülerinnen und Schüler eingesetzt werden. Das Infoblatt für die Betreuerinnen und Betreuer dient zur Vorbereitung dieser, für die Einführung in die GLSL und den Shadermanager, wie in Abschnitt 4.4.4 beschrieben.

6.2.1 Verlaufsplan für Schülerinnen und Schüler



6.2.2 Infoblatt Mathematik

Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Infoblatt: Mathematik

Vektorrechnung

Seien $\vec{v}=\begin{pmatrix} v_0 \\ v_1 \\ \dots \\ v_n \end{pmatrix}$ und $\vec{w}=\begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_n \end{pmatrix}$ Vektoren.

Vektoroperationen	
Skalarprodukt	$\vec{v}*\vec{w}=v_0\cdot w_0+\dots+v_n\cdot w_n$
Länge von $\vec{v}$	$\lvert\vec{v}\rvert=\sqrt{\vec{v}*\vec{v}}=\sqrt{v_0^2+\dots+v_n^2}$
Winkel zwischen $\vec{v}$ und $\vec{w}$	$\cos\alpha=\frac{\vec{v}*\vec{w}}{\lvert\vec{v}\rvert\cdot\lvert\vec{w}\rvert}$

Normale

Eine Normale ist ein Vektor, welcher **senkrecht** auf einer Ebene steht.

2 Vektoren $\vec{v}$ und $\vec{w}$ sind genau dann senkrecht, wenn der Winkel zwischen ihnen 90° beträgt.

Auf einer ebenen Fläche, wie einem Quadrat, ist die Normale an allen Punkten gleich. Bei einer gekrümmten Fläche zeigt die Normale hingegen in jedem Punkt in eine andere Richtung, wie im Bild rechts zu sehen ist.

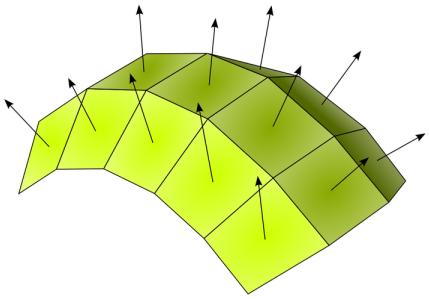


Abbildung 1: http://upload.wikimedia.org/wikipedia/commons/3/3d/Normalenvektor.png

Matrix-Vektorrechung

Seien $\vec{v}=\begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix}$ ein Vektor und $M=\begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix}$ eine Matrix, dann ist

$$M\cdot\vec{v}=\begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix}\cdot\begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix}=\begin{pmatrix} v_0\cdot m_{00}+v_1\cdot m_{01}+v_2\cdot m_{02}+v_3\cdot m_{03} \\ v_0\cdot m_{10}+v_1\cdot m_{11}+v_2\cdot m_{12}+v_3\cdot m_{13} \\ v_0\cdot m_{20}+v_1\cdot m_{21}+v_2\cdot m_{22}+v_3\cdot m_{23} \\ v_0\cdot m_{30}+v_1\cdot m_{31}+v_2\cdot m_{32}+v_3\cdot m_{33} \end{pmatrix}$$

6.2.3 Infoblatt Informatik

Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Infoblatt: OpenGL Shading Language

GLSL – OpenGL Shading Language

Die GLSL ist die Programmiersprache mit der Shaderprogramme erstellt werden. Sie ist von der Syntax her stark an C bzw. C++ und Java orientiert, allerdings ist sie eine imperative Sprache, d.h. es existieren keine Objekte wie in der objektorientieren Programmierung.

Bekannte Programmierkonstrukte sind **bedingte Anweisungen:**

```
if ( Bedingung )
{
        Anweisungsblock1;
}else{
        Anweisungsblock2;
}
```

und **Schleifen**:

```
for( Startwert ; Bedingung ; Aktualisierung )
{
        Anweisungen;
}
```

```
while(Bedingung)
{
        Anweisungen;
}
```

Variablentypen

Die üblichen Verdächtigen	
void	Für Funktionen ohne Rückgabewert
int	Ganze Zahlen
float	Fließkommazahlen, wobei in GLSL nicht 1.2f sondern 1.2 geschrieben wird
Matrizen und Vektoren	
vec2, vec3, vec4	Vektoren mit 2, 3 oder 4 Einträgen
mat2, mat3, mat4	Matrizen mit 2x2, 3x3 oder 4x4 Einträgen

Zum Beispiel erstellt vec4(1.0,1.0,1.0,1.0) einen Vektor mit 4 Einträgen, die alle 1.0 als Wert haben.

Operatoren

+, - , * , /	Übliche Rechenoperationen
%	Rest bei Ganzzahldivision
<=, >=, ==, >, <	Übliche Vergleichsoperationen
&&, ||, !	Und-, Oder- sowie Nicht-Verknüpfung bedingter Anweisungen

Funktionen

Geometrische Funktionen	
length(x)	Länge des Vektors x
dot(x,y)	Skalarprodukt der Vektoren x und y
cross(x,y)	Kreuzprodukt der Vektoren x und y
normalize(x)	Liefert normalisierten Vektor x zurück
Trigonometrische Funktionen:	
sin(alpha), cos(alpha)	Liefert Sinus oder Kosinus von float alpha
asin(x), acos(y)	Umkehrfunktionen des Sinus und Kosinus
Standardfunktionen	
abs(x)	Liefert Betrag von float x
floor(x)	Liefert abgerundetes float x zurück
ceil(x)	Liefert aufgerundetes float x zurück

Die GLSL bietet noch jede Menge weitere eingebaute Funktionen, aber für heute sind die vorgestellten ausreichend. **Eigene Funktionen** können auch geschrieben werden:

Allgemein:

```
Rückgabewert Name(Parameter)
{
        Anweisungen
}
```

Als Beispiel:

```
int multipliziere (int a, int b)
{
        return a*b;
}
```

Uniform-Variablen

Uniform-Variablen werden von **außerhalb**, also dem zugrundeliegenden OpenGL-Programm, an die Shaderprogramme **übergeben**. Zum Beispiel enthält „uniform float timer“ einen Wert der von 0.0 langsam hochgezählt wird.

Varying-Variblen

Eine Variable ist varying zu kennzeichnen, wenn sie vom **Vertex- an den Fragmentshader übergeben** werden soll. Dabei wird der Variablenwert interpoliert.

Deklaration:

varying Typ Name;

am Beginn des Vertex- und Fragmentshader.

Interpolation

Interpolation beschreibt, dass anhand 2 bekannter Werte die dazwischen liegenden Werte bestimmt werden.

Zum Beispiel interpoliert man zwischen 2 Eckpunkten mit den Variablenwerten 0 und 1. Dann bestimmt sich die Variable des Fragments in der Mitte auf 0.5 und je näher das Fragment an einem Eckpunkt liegt desto näher ist der Variablenwert an dem des Eckpunktes. Also nahe bei 0 zum Beispiel 0.1.

6.2.4 Infoblatt für Betreuer: Shadermanager und GLSL

Schülerlabor Informatik

InfoSphere
World of Informatics

Modul Computergrafik
Betreuerblatt: Präsentation Shadermanager & GLSL

Anleitung zur Präsentation des Shadermanagers

Diese ist eine Frontalphase, die nur bedingt interaktiv gestaltet werden kann. Nach der Erläuterung der wesentlichen Funktionen des Shadermanagers gibt es einen Präsentationsvorschlag, welcher versucht die GLSL- und Programmpräsentation gemeinsam zu erledigen.

Generelle Informationen

Der **Shadermanger** ist ein Programm zur

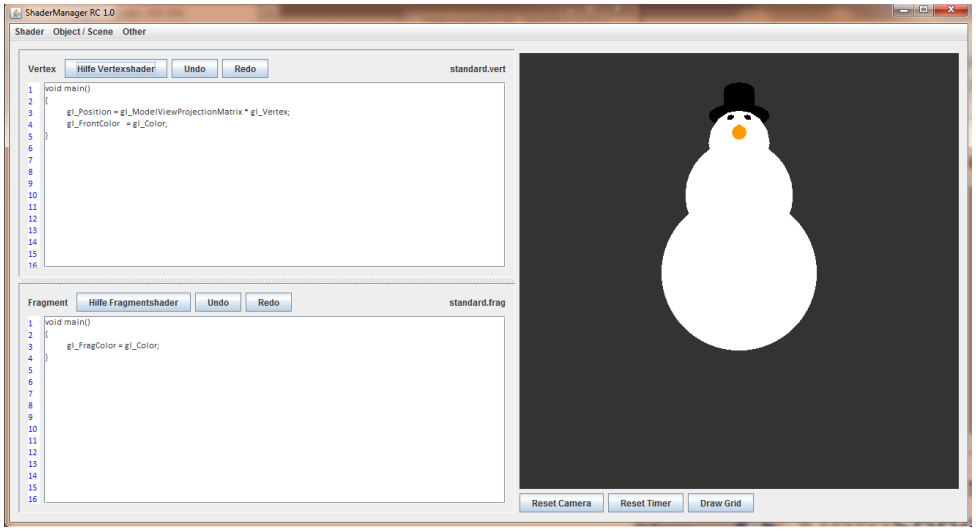
- Erstellung,
- Verwaltung sowie
- Testung

von Shadern.

Shader sind kleine Programme, welche direkt auf der Grafikkarte ausgeführt werden und in einer eigenen Programmiersprache GLSL (OpenGL Shading Language) programmiert werden.

Die Oberfläche

Die GUI ist relativ einfach gehalten, benötigt vor der ersten Verwendung jedoch einige Hinweise, insbesondere wo man arbeiten kann. Als Überblick dient folgendes Bild, welches den Startbildschirm inklusive einem geladenen Vertex- und Fragmentshader zeigt:



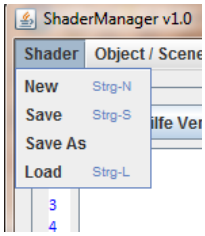
Grob ist der Shadermanager in zwei Bereiche eingeteilt:

- Links befinden sich die **Eingabebereiche** für den Quellcode der Shaderprogramme und
- rechts befindet sich die **Ausgabe**, welche ein Objekt mit den aktuellen Shadern darstellt.

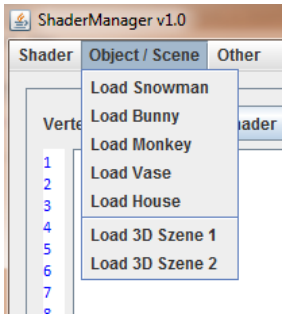
Der Eingabebereich ist wiederum zweigeteilt. Einmal findet sich oben der Eingabebereich für den **Vertexshader**, also dem Shader, welcher auf den Eckpunkten der Netze arbeitet, und darunter für den **Fragmentshader,** welcher auf den Fragmenten arbeitet. Dies sind einfach gesagt Pixel, die noch Informationen der Szene enthalten.

Die obere Menüleiste

Die obere Menüleiste bietet die Möglichkeit über **Shader**



- einen **neuen Shader** zu erstellen, wobei dann ein Grundgerüst der Shaderprogramme bereits vorgegeben wird,
- den aktuellen Quellcode zu **speichern**, bei Bedarf auch an einer neuen Position und
- bereits mit dem Shadermanger erstellte Programme zu **laden**.

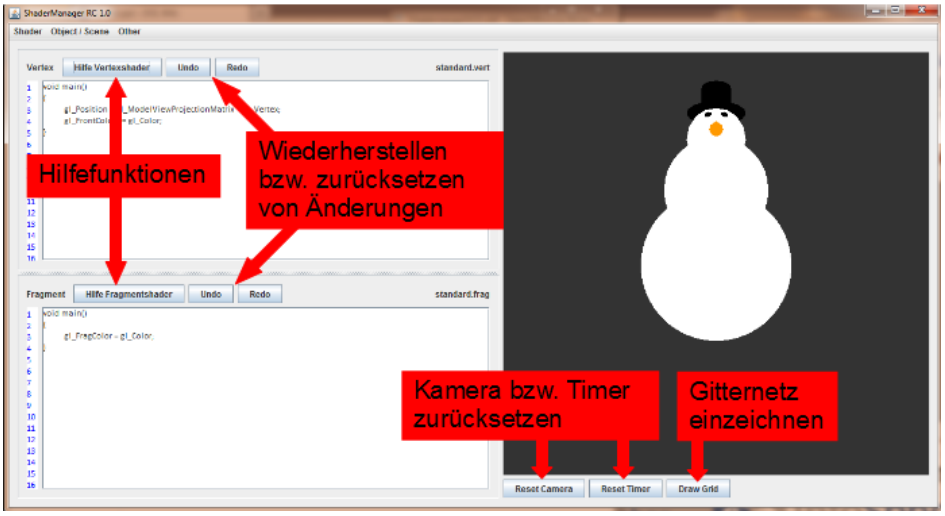


Der Menüpunkt **Object / Scene** erlaubt das Laden anderer Objekte und Szenen im Ausgabebereich.

Das Bunny-Objekt sieht zwar nett aus, ist aber für die gegebenen Laptops etwas zu aufwändig, sodass dies zu Ruckeln führt.

Der Menüpunkt **Other** braucht nicht weiter vorgestellt zu werden, da sich darin nur Informationen über den Ersteller des Programms sowie der OpenGL Version finden lassen.

Die restliche Oberfläche



Auf der Oberfläche sind verschiedene Buttons sichtbar:

- **Hilfe Vertex- bzw. Hilfe Fragmentshader** ruft ein Hilfeprogramm speziell für diesen Shader auf. Diese Informationen finden die Schülerinnen und Schüler auch auf dem Infoblatt zur Shaderprogrammierung.
- **Reset Camera** setzt die Kamera auf die Ursprungsposition zurück, falls man die Orientierung verloren hat.
- **Reset Timer** setzt die *timer*-Variable zurück
- **Draw Grid** zeichnet ein Gitternetz in die Szene

Die **Steuerung der 3D Oberfläche** geschieht mit der Maus, wobei

- Linksklick + Ziehen die Kamera um das Objekt dreht und
- Rechtsklick + Maus nach vorne / hinten ziehen das Objekt ran oder raus zoomt.

Präsentationsvorschlag

Mein Vorschlag zur Präsentation wäre, dass man die oben genannten Funktionen am Beamer direkt vorführt. Dies könnte ungefähr so ablaufen:

1. Programm starten.
2. Neuen Shader erstellen:
   - Auf Grundgerüst hinweisen.
   - Ähnlichkeit von GLSL zu C/C++/Java im Bezug auf Syntax erwähnen. ABER GLSL ist keine objektorientierte Programmiersprache!
3. Speichern.
4. Objekt einmal ändern und wieder zurücksetzen (über Menüpunkt).
5. Programmierung beginnen, dazu zunächst Hilfe für Vertexshader erwähnen.
6. Vertexshaderprogramm so ändern, dass das Objekt mit der Variablen *timer* skaliert wird. Dazu Code im Vertexshader so ändern:

```
uniform float timer;
void main()
{
    vec4 pos = mat4(timer,0.0,0.0,0.0,
                    0.0,1.0,0.0,0.0,
                    0.0,0.0,1.0,0.0,
                    0.0,0.0,0.0,1.0)
    gl_Position = gl_ModelViewProjectionMatrix * pos;
    gl_FrontColor = gl_Color;
}
```

   Dabei können der Typ sowie Befehl vec4 und mat4 kurz erwähnt werden. Hier könnte man die Schülerinnen und Schüler durch die Frage, was das Programm macht aktivieren. Das Programm skaliert das Objekt in x-Richtung.
7. Programm speichern und auf Änderung des Objekts hinweisen, da die Shader erst nach dem Speichern verwendet werden.
8. Anschließend kann man den Timer sowie die Kamera zurücksetzen und man hat in kurzer Zeit das Programm und die Grundstruktur von GLSL präsentiert.

6.3 Materialliste

In diesem Abschnitt findet sich eine Liste mit den benötigten Materialien. Ausgegangen wird bei der Berechnung von einer 20-köpfigen Gruppe, welche sich in 10 Teams zu 2 Personen zusammen findet. Andere Gesamtanzahlen können über die Anzahl der Materialien für jedes Team bestimmt werden.

Material	**Anzahl pro Team**	**Anzahl Gesamt**
Steckbrett bzw. Nagelbrett	1	10
Gummibänder	min. 3	min. 30
Laptops	1	10
RGB-Steuerung	1	10
(rot-grün) 3D-Brille	2	20
Folienstife	min. 1	min. 10
Arbeitsblatt Objektdarstellung	1	10
Arbeitsblatt Animation	1	10
Arbeitsblatt Farben	1	10
Arbeitsblatt Comiclook	1	10
Arbeitsblatt Phong-Beleuchtung	1	10
Arbeitsblatt 3D-Stereo	1	10
Sicherungsblatt Koordinatentransformationen	2	20
Infoblatt Mathematik	1	10
Infoblatt Informatik	1	10
Infoblatt für Betreuerinnen und Betreuer	-	min. 1

Zu den Laptops noch folgende Anmerkung: Da der Shadermanager, als essentieller Bestandteil des Moduls, OpenGL 2.1 benötigt, müssen die Laptops diesen Standard haben. Dies ist bei den Laptops des Schülerlabors Informatik der Fall, allerdings bei dem mir zur Verfügung gestellten Laptop erst nach einem automatischen Windowsupdate, bei dem vermutlich ein Treiberupdate für den Grafikchip enthalten war.

Literaturverzeichnis

[1] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering*. A.K. Peters Ltd., 3. edition, 2008.

[2] Rüdeger Baumann. *Didaktik der Informatik*. Klett, 2. edition, 1996.

[3] TU Clausthal. *Shadermaker*. `http://cg.in.tu-clausthal.de/teaching/shader_maker/index.shtml`, 2011.

[4] GameDev. *NeHe*. `http://nehe.gamedev.net/`, 2011.

[5] Peter Hubwieser. *Didaktik der Informatik*. Springer Berlin Heidelberg, 3. edition, 2007.

[6] Leif Kobbelt, Sandip Sar-Dassi, and Mario Botsch. *Computer Graphics 1*. 2005. Lecture Note.

[7] Wissenschaft und Forschung des Landes Nordrhein-Westfalen Ministerium für Schule und Weiterbildung. *Richtlinien und Lehrpläne für die Sekundarstufe II - Gymnasium/Gesamtschule in Nordrhein-Westfalen: Informatik*. Ritterbach Verlag, Frechen, 1. edition, 1999.

[8] Wissenschaft und Forschung des Landes Nordrhein-Westfalen Ministerium für Schule und Weiterbildung. *Richtlinien und Lehrpläne für die Sekundarstufe II - Gymnasium/Gesamtschule in Nordrhein-Westfalen: Mathematik*. Ritterbach Verlag, Frechen, 1. edition, 1999.

[9] Jochen Müsseler. *Allgemeine Psychologie*. Springer Verlag Berlin Heidelberg, 2008.

[10] Ulrik Schroeder. *eLearning*. Vorlesungsfolien, SS2010; RWTH Aachen, Lehrstuhl LuFGi9.

[11] Ulrik Schroeder. *Fachdidaktik Informatik 1*. Vorlesungsfolien, SS2010; RWTH Aachen, Lehrstuhl LuFGi9.

[12] Gabriel Zachmann. *Computer-Graphik in Schule*. `http://zach.in.tu-clausthal.de/cg_in_schule/`, 2011.

Erklärung

Ich versichere, dass ich die schriftliche Hausarbeit - einschließlich beigefügter Zeichnungen, Kartenskizzen und Darstellungen - selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die im Wortlaut oder dem Sinne nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle deutlich als Entlehnung kenntlich gemacht.

Aachen, den 11. Januar 2012

Tim Schellartz

LANDESPRÜFUNGSAMT

FÜR ERSTE STAATSPRÜFUNGEN FÜR LEHRÄMTER AN SCHULEN
NRW
Geschäftsstelle Aachen

Einverständniserklärung

Name: Tim Schellartz

geboren am 18.03.1987 in Heinsberg

Erklärung:

Ich bin damit einverstanden, dass diese Hausarbeit nach Abschluss meiner Ersten Staatsprüfung wissenschaftlich interessierten Personen oder Institutionen zur Einsichtnahme zur Verfügung gestellt wird und dass zu diesem Zweck Ablichtungen dieser Hausarbeit hergestellt werden, sofern diese keine Korrektur- oder Bewertungsvermerke enthalten und sofern der Benutzer die entstehenden Kosten für die Ablichtung trägt.

(Ort und Datum) (Unterschrift)