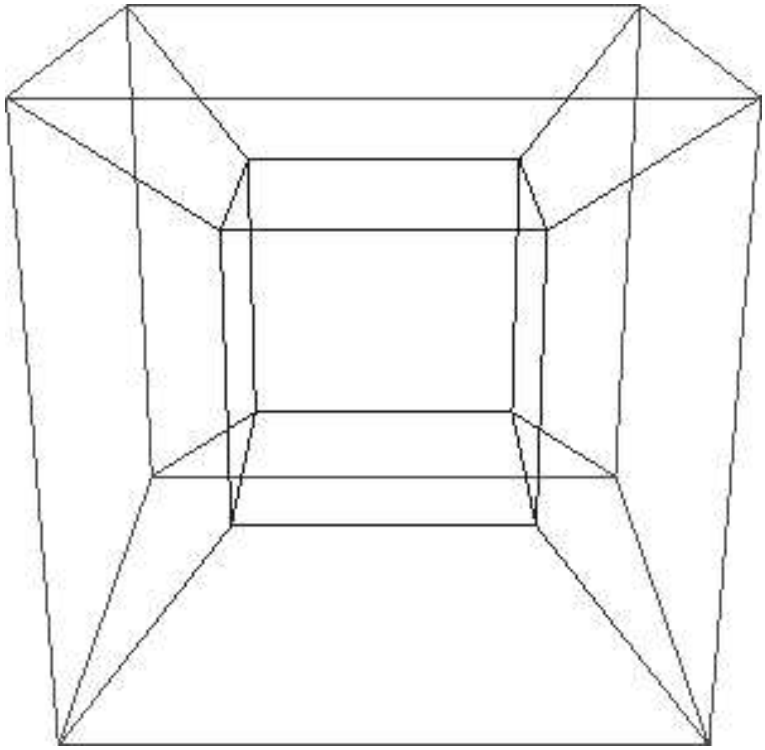
Studienarbeit
Interaktive Visualisierung von 4-dimensionalen Objekten



vorgelegt von

Andreas Mosig
Matrikelnummer 200210251

Betreuer: Prof. Dr. Stefan Müller

Koblenz, im April 2005

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im 19. Jahrhundert begannen sich Menschen Gedanken darüber zu machen, ob auch höhere Dimensionen existieren als die bekannten drei, die in der euklidischen Geometrie einen Raum aufspannen. Das bekannteste Beispiel ist die Science-Fiction Geschichte *Flatland* von Edwin Abbott [1]. Diese Geschichte wird von einem Quadrat erzählt, das von dem Leben zweidimensionaler Wesen, die in einer flachen Ebene leben, berichtet. Diese Wesen haben nur ein zweidimensionales Weltbild und können sich nicht vorstellen, dass es auch eine weitere, räumliche Dimension gibt, bis zu dem Augenblick, als eine Kugel erscheint und sich dem Quadrat vorstellt. Das Quadrat ist natürlich verblüfft, kann es die Kugel doch nur als Kreis erkennen, der seinen Radius ändert. Ihm ist (noch) nicht bewusst, dass der Kreis nur die Schnittfläche der Kugel mit seiner Ebene ist und er die Kugel ganz sehen würde, wenn er nur nach oben und damit entlang der dritten Dimensionsachse schauen würde.
Abbott nutzt in seiner Geschichte den Trick der Analogie, um dem Leser das Verständnis höherer Dimensionen begreifbar zu machen: So wie er 2D-Lebewesen einer 3D-Welt gegenüberstellt, können wir 3D-Menschen uns auch eine 4D-Welt vorstellen. Leider besitzt der Mensch nur räumliche Wahrnehmungsmöglichkeiten, dass heisst, vierdimensionale Objekte werden wir nie vollständig betrachten können. Aber mit den selben Techniken, um Körper zweidimensional zu visualisieren, können wir 4D-Objekte räumlich darstellen und somit einen Teil des Gesamten sichtbar machen.
Die Darstellung von 4D-Objekten beschränkte sich sonst auf Zeichnungen und Diagrammen. Das statische Bild ist aber ein denkbar schlechtes Medium für die Darstellung solcher komplexer Strukturen, da durch die Projektion von 4D auf das 2D-Papier die Information über das wahre Aussehen des 4D-Körpers verloren geht. Solche Bilder erfordern auch eine hohe räumliche Vorstellungskraft beim Betrachter. Mit dem Auftreten moderner Grafikcomputer verbesserte sich diese Darstellung durch die Animation der Bilder. Man konnte so das Objekt durch Drehen auch von der Rückseite sehen. Aber der Betrachter muss in seiner Wahrnehmung immer noch zwei Dimensionsschritte abstrahieren: Das Anschauen eines 2D-Bildes auf dem Monitor, das zu einem räumlichen 3D-Bild in seinem Gehirn interpretiert wird, stellt für den Menschen kein Problem dar. Aber dieses räumliche Bild im Kopf nochmals zu interpretieren und als 4D-Objekt wahrzunehmen, ist die Schwierigkeit.
Das Ziel dieser Studienarbeit ist Darstellung solcher vierdimensionaler Objekte auf einem Stereo-Wiedergabesystem. Der Benutzer soll die Möglichkeit haben, 4D-Objekte dreidimensional zu betrachten und sie interaktiv manipulieren zu können, um das Verständnis der vierten Dimension zu fördern und neue Betrachtungsweisen aufzuzeigen.

Diese Ausarbeitung gliedert sich in folgende Teilbereiche:

1. Die Grundlagen zum Verständnis der Dimensionsproblematik und Möglichkeiten der Visualisierung der vierten Dimension werden in Kapitel 2 erklärt. Zusätzlich werden die Techniken der Transformation und Projektion im vierdimensionalen Raum erläutert.

2. Das Kapitel 3 befasst sich mit der Implementierung der vierdimensionalen Darstellung durch Projektion. Dabei werden die in der Studienarbeit benötigten Klassen vorgestellt und ihre Funktionsweise erläutert.

3. In Kapitel 4 werden die erreichten Ergebnisse des Programms vorgestellt.

4. Zum Schluss wird in Kapitel 5 die Studienarbeit kritisch reflektiert und zukünftige Erweiterung und Verbesserungen angesprochen.

Kapitel 2

Grundlagen

2.1 Was sind Dimensionen?

Der Begriff Dimension existiert in den unterschiedlichsten Fachrichtungen und hat demnach auch unterschiedliche Bedeutungen. Die in der Mathematik und Informatik gebräuchliche Definition ist die *Hamel-Dimension*, besser bekannt als die Dimension eines Vektorraumes. In der euklidischen Geometrie spannen drei zueinander senkrecht stehende Basisvektoren einen Vektorraum auf, so dass jeder Punkt in diesem Koordinatensystem durch drei Zahlen beschrieben werden kann. Die alltäglichen Begriffe *Länge, Breite, Höhe* sind mit den drei Dimensionen unserer räumlichen Wahrnehmung assoziiert.

2.1.1 Zeit als Dimension

Seit Einstein durch seine Forschungen ein neues Bild des Universums formte, spricht man heutzutage von *Raumzeit*. Der Raum wird durch drei Dimensionen beschrieben, die vierte Dimension ist die Zeit. Um einen Punkt innerhalb der Raumzeit zu bestimmen, benötigt man vier Grössen: Drei Raumkoordinaten und einen Zeitpunkt. Die Zeit jedoch als die vierte Dimension zu bezeichnen ist nur teilweise richtig, denn die Zeit ist von der Dimension des Betrachters abhängig. Das Quadrat aus Flatland z.B. sieht die Kugel nur dann, wenn sie seine Ebene schneidet, und dann auch nur als Kreis. Bewegt sich nun die Kugel, ändert sich auch der Durchmesser des Kreises, angefangen vom simplen Punkt bei der Berührung mit der Ebene bis hin zur vollen Ausdehnung, wenn die Kugel bis zur Mitte eingetaucht ist (siehe Abb. 2.1).

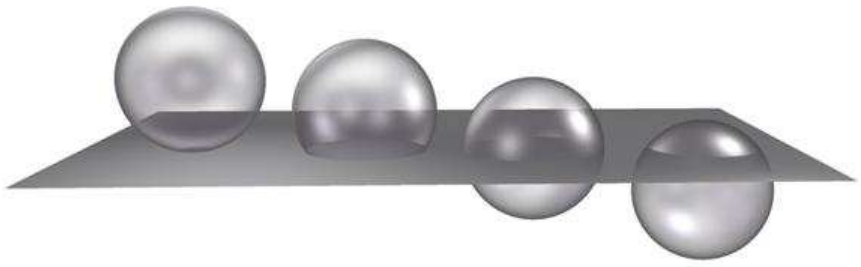


Abbildung 2.1: Beim Schneiden einer Kugel mit einer Ebene werden kreisförmige Schnittmuster erkennbar (Bildquelle: [6]).

Für einen zweidimensionalen Betrachter ist die Zeit die dritte Dimension, da sich 3D-Objekte nur innerhalb eines Zeitintervalls vollständig betrachten

lassen. Analog verhält es sich für uns dreidimensionale Wesen: Die Zeit tritt als Effekt sich verändernder Räume auf. So kann z.B. ein Raum mit einer Uhr in verschiedenen Ausprägungen existieren. Einmal, in dem das Pendel der Uhr nach links schwingt, das andere Mal der selbe Raum, nur mit dem Unterschied, dass das Pendel nach rechts ausschlägt. Man kann es so formulieren, dass für einen Betrachter der Dimension n die Zeit die Dimension $n+1$ besitzt. Ein vierdimensionales Wesen besitzt also auch einen Zeitbegriff, für dieses wäre dann die Zeit die fünfte Dimension.

2.1.2 Punkt, Gerade, Ebene, ...

Will man die vierte Dimension als Raum betrachten, hilft dabei folgendes Gedankenexperiment: Man stelle sich einen Punkt vor. Ein Punkt ist ein Objekt der Dimension 0, da er keinerlei Ausdehnung in irgendeine Richtung besitzt. Verlängert man diesen Punkt in eine Richtung, entsteht eine Gerade, welche ein Objekt der Dimension 1 ist. Wenn man nun die Gerade entlang einer Achse, die senkrecht zu der Geraden steht verlängert, wird eine Ebene aufgespannt. Die Ebene ist somit ein Objekt der Dimension 2, da sie eine Länge und Breite hat. Die Ebene kann man nun wiederum entlang einer dritten Achse verlängern. Die dritte Achse ist die Normale zu der Ebene oder einfach gesprochen die Höhe. Es entsteht ein Quader, ein räumliches Objekt der Dimension 3. Nun folgt der entscheidende Schritt: Man nehme an, es existiere eine vierte Achse, die orthogonal zu allen drei Raumachsen steht. Dies ist zwar ein Widerspruch zu der euklidischen Geometrie, aber als Gedankenkonstrukt durchaus denkbar. Verschiebt man den Quader entlang dieser vierten Achse, entsteht ein neues, vierdimensionales Objekt: Der Hyperkubus. Man kann sich dieses 4D-Objekt als zwei Würfel vorstellen, deren Eckpunkte mit Kanten verbunden sind (siehe Abb. 2.2).
Es gibt also durchaus einen vierdimensionalen Raum, auch Hyperraum genannt. Das Präfix *Hyper* bedeutet so viel wie darüber oder über dem Raum liegend. Somit gewinnt auch der Begriff Hyperebene an Bedeutung: Über der Ebene liegend, was nichts anderes heisst, dass die Hyperebene den normalen 3D-Raum darstellt, sowie die Hyperlinie eine Ebene bedeutet.

2.2 Visualisierungsmöglichkeiten

Da der Mensch visuell nur räumlich wahrnehmen kann, bleibt ihm die Gesamtheit des Hyperraums vor seinen Sinnen verborgen. Es gibt aber Möglichkeiten,

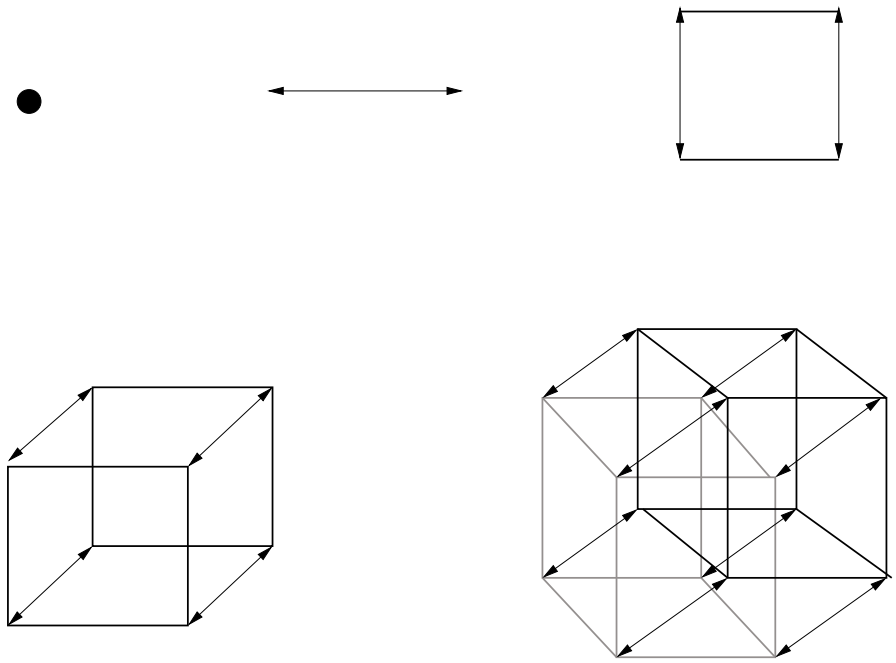


Abbildung 2.2: Punkt, Linie, Quadrat, Würfel und Hyperkubus als Analogie zur Dimensionserweiterung.

gewisse Aspekte der vierten Dimension graphisch aufzubereiten und sie teilweise darzustellen. Dabei ist die Analogie ein wichtiger Verbündeter, denn die vierte Dimension verhält sich zum dreidimensionalen Raum wie die dritte Dimension zur zweidimensionalen Fläche.

2.2.1 Projektion als Schattenwurf

Die einfachste Möglichkeit höherdimensionale Gebilde darzustellen ist die Projektion. Ein Objekt eines bestimmten Dimensionsgrades wird dabei auf eine niedrigere Dimension abgebildet. Projektionen begegnen einen im Alltag als Schattenwurf. Wird ein Körper durch Licht angestrahlt, wird auf der Wand dahinter sein Schatten sichtbar. Auch Zeichnungen oder 3D-Bilder auf dem Computermonitor sind Projektionen, da räumliche Körper um eine Dimension reduziert dargestellt werden. Wenn nun ein 4D-Objekt vom Hyperraum auf eine Hyperebene projeziert wird, sieht man also einen dreidimensionalen Schatten des Objektes.
Der Vorteil bei Projektionen ist, dass das gesamte Objekt abgebildet wird, vorausgesetzt es liegt als Drahtgittermodell vor, so dass keine Verdeckung durch solide Flächen auftreten. Die Nachteile sind auftretende Verzerrungen

und Überschneidungen von Kanten und Flächen.

2.2.2 Betrachtung der Schnittflächen

Eine andere Möglichkeit bildliche Informationen über Körper zu erhalten besteht darin, sie mit niedrigdimensionalen Ebenen zu schneiden und die Schnittfläche zu betrachten. Als Beispiel sei eine Glaskugel erwähnt, die in Wasser getaucht wird. In dem Moment, wo die Kugel die Wasseroberfäche berührt, besteht die Schnittfläche aus einem einzigen Punkt. Taucht man die Kugel weiter in das Wasser ein, entsteht aus dem Punkt ein Kreis, der seinen Durchmesser vergrössert und wieder verkleinert, wenn die Kugel vollständig im Wasser untergeht. Die Schnittfläche einer Kugel sieht immer gleich aus, egal wie herum sie eingetaucht wird. Anders sieht es bei Polyedern aus: Je nachdem ob der Polyeder mit einer Ecke, Kante oder Fläche zuerst eingetaucht wird, ändert sich auch das Schnittmuster. Wenn ein 4D-Objekt den Raum schneidet, macht sich das für den Betrachter als das Erscheinen eines 3D-Körpers bemerkbar, der seine Form und Grösse ändert, je nachdem ob er mit einer Ecke, Kante, Fläche oder einem Raum zuerst unsere Hyperebene schneidet (siehe Abb. 2.3, 2.4, 2.5).
Der Vorteil bei der Betrachtung von Schnittflächen ist, dass man einen Eindruck über die äussere Form des Körpers bekommt. Um aber die gesamte Form zu sehen, muss jede einzelne Schnittfläche aneinandergereiht betrachtet werden. Nachteilig ist auch, dass die Schnittflächenmuster variieren, je nachdem welche Seite des Körpers zuerst geschnitten wird.

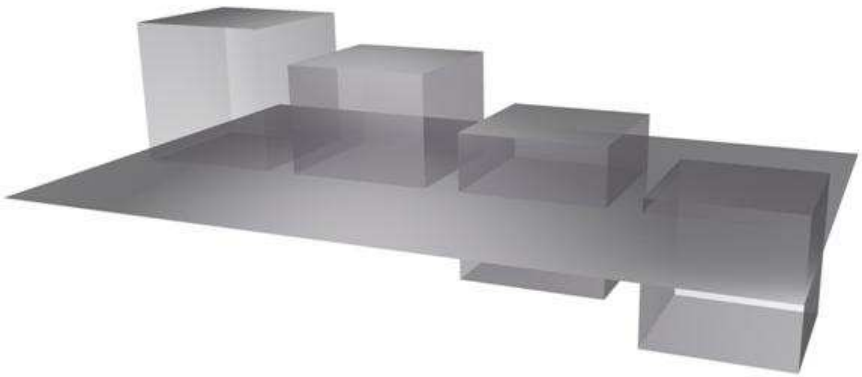


Abbildung 2.3: Ein Würfel schneidet eine Ebene mit einer seiner Flächen (Bildquelle: [6]).

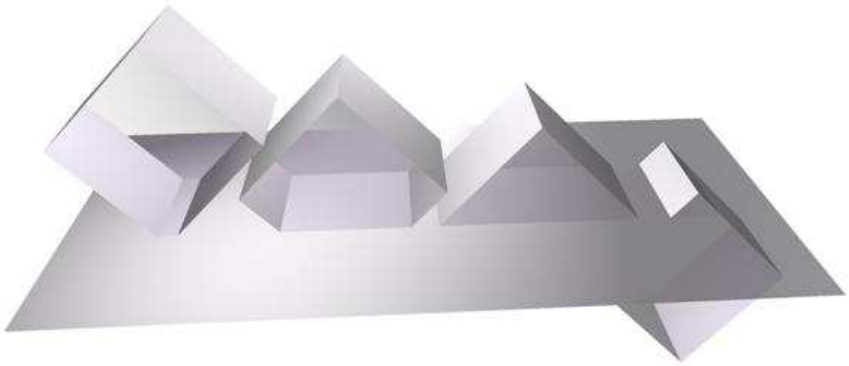


Abbildung 2.4: Ein Würfel schneidet eine Ebene mit einer seiner Kanten (Bildquelle: [6]).

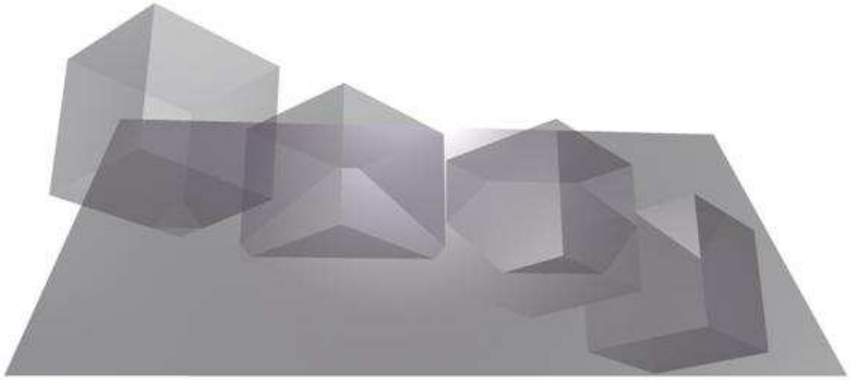


Abbildung 2.5: Ein Würfel schneidet eine Ebene mit einer seiner Eckpunkte (Bildquelle: [6]).

2.2.3 Falten der Objekte

Mit Hilfe von Faltungen lassen sich höherdimensionale Objekte konstruieren, bzw. durch Entfalten in niedrigdimensionale Objekte zerlegen. So wie ein Würfel durch Entfalten zu einer ebenen Aneinanderreihung von quadratischen Segmenten wird, kann ein 4D-Objekt zu seinen räumlichen Segmenten aufgeklappt werden (siehe Abb. 2.6). Das Zusammenfalten von 3D-Körpern

passiert durch das deckungsgleiche Zusammenlegen von freistehenden Kanten an der zweidimensionalen Ausgangsform. Analog faltet man 4D-Objekte so zusammen, dass alle freistehenden Flächen deckungsgleich miteinander verbunden werden. Eine solche 4D-Faltung ist im normalen Raum natürlich nicht zu realisieren, da das Objekt sich unmöglich verbiegen und stauchen würde.
Der Vorteil beim Entfalten ist, dass man alle Raumzellen, aus denen ein 4D-Objekt bestehen kann, unverzerrt und vollständig sehen kann. Der Nachteil ist die nicht immer intuitive Zusammenfaltung.

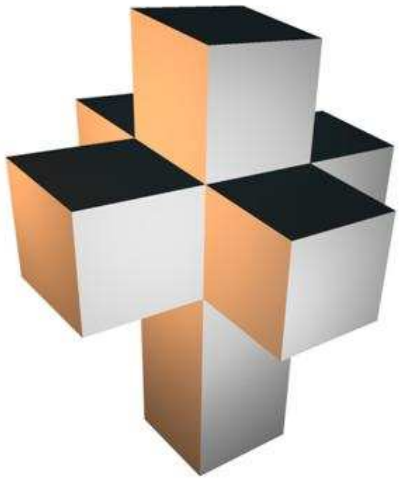


Abbildung 2.6: Der aufgefaltete Hyperkubus (Bildquelle: [6]).

2.3 Vierdimensionale Geometrie

Die vierdimensionale Geometrie ist eine schlichte Erweiterung der euklidischen Geometrie, d.h. neben den bekannten Koordinaten x, y, z wird eine vierte Koordinate w hinzugefügt. Alle Operationen wie Addition, Substraktion und Multiplikation können beibehalten werden, nur mit dem Unterschied, das vierdimensionale Vektoren verwendet werden. Eine Ausnahme bildet dabei das Kreuzprodukt zwischen zwei 4D-Vektoren, das nicht so trivial wie sein 3D-Gegenstück zu berechnen ist [7]. Transformationen wie Translation, Skalierung und Rotation werden mit Hilfe von Matritzen bewerkstelligt, die mit dem Vektor multipliziert werden. Nach den Transformationen wird der Vektor projeziert, d.h. vom vierdimensionalen Hyperraum auf eine dreidimensionale Hyperebene abgebildet. Dieser 3D-Vektor kann nun wie gewohnt

einer Graphikbibliothek wie OpenGL oder DirectX übergeben werden, die ihn dann nochmals von 3D auf eine Ebene projeziert und auf dem Bildschirm darstellt.

2.3.1 Rotation

Eine Rotation ist eine Transformation, die einen Punkt zu einer anderen Stelle des Koordinatensystems verschiebt, dabei aber den Abstand zum Ursprung beibehält und sich nur die Richtung ändert. Der Ursprung wird dabei von der Rotation nicht erfasst, er wird wieder auf sich selbst abgebildet. Bei einer Rotation ändern sich jeweils nur zwei Koordinaten des Vektors, egal in welcher Dimension die Rotation durchgeführt wird.

Rotation in 2D

Rotationen in 2D sind einfach nachzuvollziehen: Ein Punkt in einer Ebene markiert den Ursprung und alle anderen Punkte bewegen sich um ihn herum, ohne ihren Abstand zum Ursprung zu ändern. Es gibt auch nur eine Möglichkeit in der Ebene zu rotieren, nur die Richtung kann variieren.
Bei einem gegebenen Winkel α im Bogenmaß rotiert folgende Matrix einen zweidimensionalen Vektor $\vec{v}$:

$$R_{2D} = \begin{pmatrix} cos\,\alpha & sin\,\alpha \\ -sin\,\alpha & cos\,\alpha \end{pmatrix} \qquad \vec{v} = \begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \vec{v}' = R_{2D} \cdot \vec{v}$$

Rotation in 3D

Im 3D-Raum findet die Rotation nicht um einen Punkt, sondern um eine Achse oder Gerade statt. Man kann sich vorstellen, dass diese Achse die Normale zu vielen Rotationsebenen darstellt, die entlang dieser Geraden aufgestapelt sind. Es existieren aber im euklidischen Raum drei Koordinatenachsen, um die gedreht werden kann und somit auch drei Möglichkeiten Punkte zu rotieren. Eine Verknüpfung dieser drei Rotationen ermöglichen jede Drehung im Raum. Auch hier ändern sich nur zwei von drei Koordinaten: Rotiert man z.B. um die z-Achse, ändern sich die x- und y-Koordinate.
Bei einem gegebenen Winkel α im Bogenmaß rotieren folgende Matritzen einen dreidimensionalen Vektor $\vec{v}$:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\,\alpha & -sin\,\alpha \\ 0 & sin\,\alpha & cos\,\alpha \end{pmatrix} \; R_y = \begin{pmatrix} cos\,\alpha & 0 & sin\,\alpha \\ 0 & 1 & 0 \\ -sin\,\alpha & 0 & cos\,\alpha \end{pmatrix} \; R_z = \begin{pmatrix} cos\,\alpha & -sin\,\alpha & 0 \\ sin\,\alpha & cos\,\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R_{3D} = R_x \cdot R_y \cdot R_z \qquad \vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \Rightarrow \vec{v}' = R_{3D} \cdot \vec{v}$$

Rotation in 4D

Um Rotationen in 4D zu verstehen zu können, halten wir uns nochmals vor Augen, wie in den niedrigeren Dimensionen gedreht wird. In 2D wird um einen Punkt rotiert, in 3D um eine Gerade. Dann liegt die Vermutung nahe, dass in 4D um eine Ebene rotiert wird. Tatsächlich rotieren Punkte parallel zu der Ebene, die von zwei der vier Basisvektoren des Hyperraums aufgespannt wird. Diese Rotationsebene besitzt als Normale eine Koordinatenachse, so entspricht z.B. eine Rotation um die xy-Ebene einer Drehung um die z-Achse. Interessante Effekte treten bei einer Rotationsebene auf, die die Koordinate w enthält. Wird ein herkömmlicher 3D-Körper z.B. um die zw-Ebene gedreht, passiert folgendes: Der Körper wird bei Beginn der Rotation entlang der z-Achse gestaucht, bis er bei einer Drehung von $\frac{\pi}{2}$ zu einer Linie degeneriert. Nach einer weiteren viertel Drehung entfaltet sich der Körper wieder, tritt aber dabei komplett spiegelverkehrt in Erscheinung. Nach einer weiteren Drehung um π ist der normale Zustand wieder hergestellt. Bei vier Koordinatenachsen existieren nun sechs verschiedene Ebenen, um die rotiert werden kann. Es ändern sich dabei die Koordinatenpaare, die deren Basisvektoren die Ebene aufspannen.
Bei einem gegebenen Winkel α im Bogenmaß rotieren folgende Matritzen [7] einen vierdimensionalen Vektor $\vec{v}$:

$$R_{xy} = \begin{pmatrix} cos\,\alpha & sin\,\alpha & 0 & 0 \\ -sin\,\alpha & cos\,\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_{xw} = \begin{pmatrix} cos\,\alpha & 0 & 0 & sin\,\alpha \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -sin\,\alpha & 0 & 0 & cos\,\alpha \end{pmatrix}$$

$$R_{yz} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\,\alpha & sin\,\alpha & 0 \\ 0 & -sin\,\alpha & cos\,\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_{yw} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\,\alpha & 0 & -sin\,\alpha \\ 0 & 0 & 1 & 0 \\ 0 & sin\,\alpha & 0 & cos\,\alpha \end{pmatrix}$$

$$R_{xz} = \begin{pmatrix} cos\,\alpha & 0 & -sin\,\alpha & 0 \\ 0 & 1 & 0 & 0 \\ sin\,\alpha & 0 & cos\,\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad R_{zw} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & cos\,\alpha & -sin\,\alpha \\ 0 & 0 & sin\,\alpha & cos\,\alpha \end{pmatrix}$$

$$R_{4D} = R_{xy} \cdot R_{yz} \cdot R_{xz} \cdot R_{xw} \cdot R_{yw} \cdot R_{zw} \qquad \vec{v} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \Rightarrow \vec{v}\,' = R_{4D} \cdot \vec{v}$$

2.3.2 Projektion

Eine Projektion ähnelt dem Sehvorgang im menschlichen Auge. Dabei werden Lichtstrahlen von dem betrachteten Objekt, dem Projektionszentrum, ausgesandt und auf der Netzhaut abgebildet. Schiebt man jetzt eine semitransparente Fläche zwischen Auge und Projektionszentrum, so schneiden sich die Lichtstrahlen und die Projektionsebene und das Objekt wird auf der Fläche sichtbar. Allgemein gesprochen transformiert eine Projektion Punkte aus einem Raum der Dimension n in einen Raum, der eine Anzahl an Dimensionen kleiner als n enthält [5].
Um 4D-Objekte für das menschliche Auge sichtbar zu machen, müssen sie zunächst vom Hyperraum auf eine dreidimensionale Hyperebene projeziert werden. Anschliessend wird das Objekt auf dem Bildschirm abgebildet, indem es vom 3D-Raum auf eine zweidimensionale Fläche projeziert wird. Die 4D-3D Projektion ähnelt ihrem 3D-2D Gegenstück, mit dem Unterschied, dass vierdimensionale Vektoren verwendet werden. Die wichtigsten Projektionsarten sind die orthographische und die perspektivische Projektion (siehe Abb. 2.7). Beide Arten sind auch in den verschiedenen Projektionsschritten kombinierbar. So kann z.B. von 4D zu 3D perspektivisch und von 3D zu 2D orthographisch projeziert werden oder auch umgekehrt. Insgesamt gibt es vier Möglichkeiten von 4D zu 2D zu projezieren.

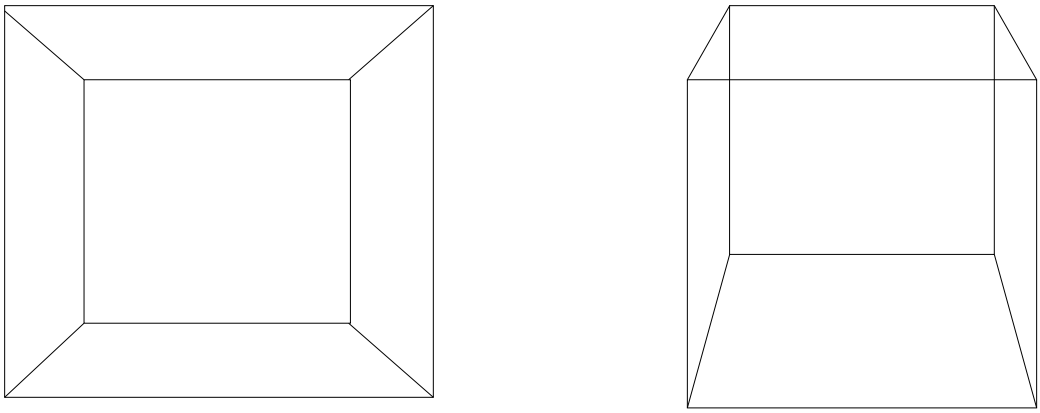


Abbildung 2.7: Der Würfel einmal orthographisch und perspektivisch projeziert.

Orthographische Projektion

Die orthographische Projektion oder auch Parallelprojektion genannt ist die einfachste von allen. Sie entsteht, wenn das Projektionszentrum unendlich weit entfernt ist, so dass die Strahlen parallel in Richtung Projektionsfläche verlaufen. Der Betrachter ist nicht mehr in der Lage, eine räumliche Tiefe in dem Bild zu erkennen. Die höherdimensionale Koordinate wird dabei einfach weggelassen, d.h. von 4D zu 3D wird die w-Koordinate und von 3D zu 2D die z-Koordinate ignoriert.

$$\vec{v} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \quad \Rightarrow \quad \vec{v}\,' = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad \Rightarrow \quad \vec{v}\,'' = \begin{pmatrix} x \\ y \end{pmatrix}$$

Perspektivische Projektion

Wenn das zu betrachtende Objekt in endlicher Entfernung steht, so bündeln sich die Sehstrahlen zu einem Punkt im Auge des Betrachters zusammen. Es entsteht so eine Pyramide an Sehstrahlen, die von der Projektionsfläche geschnitten wird. Dieser Pyramidenstumpf zwischen Projektionsfläche und Projektionszentrum wird auch Frustum genannt. Wenn die Projektionsfläche im Abstand s zum Ursprung steht, dann berechnet sich die perspektivische Projektion folgendermaßen:

$$\vec{v} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \quad \Rightarrow \quad \vec{v}\,' = \begin{pmatrix} sx/w \\ sy/w \\ sz/w \\ s \end{pmatrix} \quad \Rightarrow \quad \vec{v}\,'' = \begin{pmatrix} sx/z \\ sy/z \\ s \\ s \end{pmatrix}$$

Kapitel 3

Implementierung

3.1 Voraussetzungen

Die Grundlage für diese Studienarbeit ist das Programm *Polytopes* [2], das im XScreenSaver-Paket von GNU/Linux enthalten ist. Dieses Programm stellt verschiedene regelmässige vierdimensionale Polytope als 3D-Projektion dar. Das Programm selbst ist in C geschrieben und verwendet OpenGL als Graphikbibliothek. Da die Motivation dieser Studienarbeit bedingt war durch Verwendung von Stereoprojektion zur besseren räumlichen Darstellung, musste eine andere Bibliothek gefunden werden, mit der Stereoprojektion einfacher möglich sind als in OpenGL. Dabei fiel die Wahl auf die Bibliothek OpenSG [4]. Mit ihr ist es recht einfach möglich, 3D-Objekte mittels eines speziellen Wiedergabesystems über zwei Graphikservern hologrammähnlich auf einer Leinwand darzustellen. Dieser räumliche Eindruck soll die Immersion verstärken und es dem Benutzer ermöglichen, die z.T. komplizierten Figuren der 4D-Polytope besser zu erkennen.
Zusätzlich bietet OpenSG einen Szenegraphen zur Verwaltung der 3D-Objekte an. Diese baumähnliche Datenstruktur vereinfacht den Umgang mit den verschiedenen Objekten einer Szene durch eine hierarchische Gliederung in Transformationsknoten, die innerhalb des Graphen stehen und andere Knoten verbinden und in Objektknoten, die als Blätter an dem Szenegraphen hängen. Bei der graphischen Darstellung auf dem Bildschirm wird durch den Szenegraphen traversiert und alle Transformationen, die über einem Blattknoten liegen, auf dieses Objekt angewendet. Der Nachteil besteht darin, dass die 4D-Transformationen nicht über reguläre Transformationsknoten realisiert werden können. Obwohl OpenSG auch mit 4D-Vektoren und 4x4-Matritzen arbeitet und Operationen wie Addition und Multiplikation zur Verfügung stellt, sind sie zur Berechnung von 4D-Geometrie z.T. ungeeignet. Der Grund dafür liegt darin, dass OpenSG einen 4D-Vektor als homogenen 3D-Vektor behandelt und für 4D-Geometrie wie in diesem Fall nicht ausgelegt ist. So lässt sich z.B. keine Multiplikation zwischen einer 4x4-Matrix und einem 4D-Vektor durchführen, da in OpenSG eine 4x4-Matrix eine Transformationsmatrix für homogene 3D-Vektoren darstellt und nur mit 3D-Vektoren multipliziert werden kann.
In diesem Fall müssen die Vektoren des Objektes selbst manipuliert werden. Aber auch hier bietet OpenSG eine Lösung an: Jedes Objekt wird in einem sogenannten Geometrieknoten verwaltet, der neben den Vektoren auch die Kanten, Farben und Texturen speichert. Mittels eines direkten Zugriffs auf das Vektorfeld lassen sich diese verändern und die erwünschten Animationen auch ohne OpenSG-Transformationen erreichen.
Um eine Interaktion mit dem Programm zu gewährleisten, musste ein Eingabegerät gefunden werden, das der Benutzer vor der Leinwand verwenden

kann. Die Wahl fiel auf ein Joypad, mit dessen intuitiver Bedienung jeder Benutzer umgehen kann und es durch die vorhandenen Knöpfe möglich ist, viele Funktionalitäten per Knopfdruck zu implementieren. Als Bibliothek zur Ansteuerung des Joypads wird SDLlib verwendet, eine Sammlung an Funktionen, die neben der Joysticksteuerung auch Sound- und Graphikausgabe unterstützen.
Weiterhin wurden noch folgende Module implementiert:

- Ein unabhängiges Rahmenprogramm, das sich um die Darstellung der Graphiken kümmert und Methoden zur Benutzereingabe wie Tastatur, Maus und Joypad bereitstellt.
- Die Klasse `Levels`, das die einzelnen Module der Anwendung in Levels abkapselt, die von dem Rahmenprogramm eingebunden werden. Zusätzlich beinhaltet jedes Level einen *Event Manager*, der die Benutzereingaben behandelt.
- Die Klasse `Polytop`, die eine Datenstruktur für vierdimensionale Polytope bereitstellt und Methoden zur Rotation und Projektion anbietet.

3.2 Die Klasse Levels

Damit die einzelnen Stationen des Programms, die verschiedene 4D-Körper demonstrieren, nicht alle im Szenegraphen des Hauptprogramm implementiert werden müssen, bietet sich eine Trennung der einzelnen Szenegraphen an, die auf verschiedene Klassen aufgeteilt und vom Hauptprogramm eingebunden werden. Zu diesem Zweck wurde die abstrakte Klasse `Levels` implementiert (siehe Abb. 3.1).

Jede von Levels abgeleitete Klasse stellt eine Anzahl an Methoden bereit:

- `NodePtr createSG( void )`
  Erzeugt einen Knoten, der den Szenegraphen eines Moduls enthält. Diese Methode wird vom Hauptprogramm zu Beginn für jedes Level aufgerufen und die Teilszenegraphen in den des Hauptprogramms eingefügt.
- `void handleKey( UInt8 key )`
  Diese Methode behandelt die Tastatureingaben und löst bei Tastendruck bestimmte Aktionen aus, die für jedes Level verschieden sind.

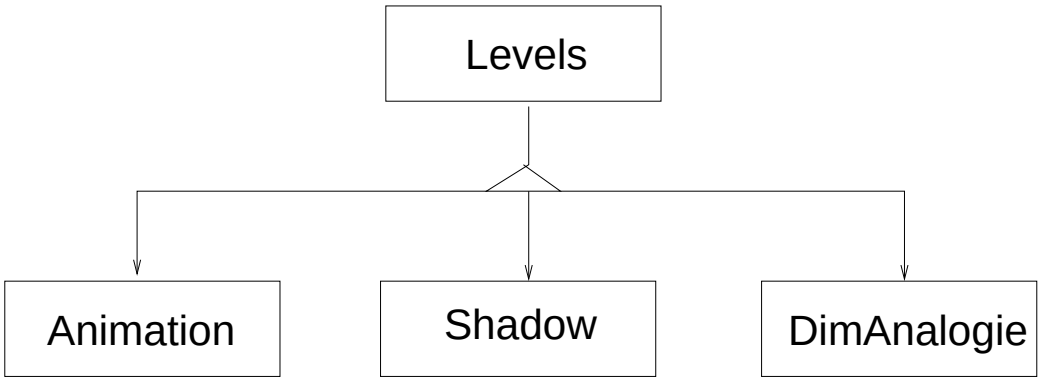


Abbildung 3.1: Die Basisklasse mit den einzelnen Levels für die Animation, Projektion-Schattenwurf und der Dimensionsanalogie

- `void handleJoy( JoyButtons *jbut )`
  Mit dieser Methode werden die Ereignisse vom Joypad, wie Steuerung und Knopfdruck, behandelt und die selben Aktionen wie bei der Tastatureingabe ausgelöst. Die Klasse `JoyButtons` enthält dazu für jede Steuerungsrichtung und jeden Knopf eine boolesche Variable, die anzeigt, ob der Knopf gedrückt ist oder nicht.

- `void redraw( void )`
  Die Graphikschleife im Hauptprogramm ruft diese Methode für jedes Level auf, wenn der Bildschirminhalt neu gezeichnet wird.

3.3 Die Klasse Polytop

Diese Klasse enthält die Geometriedaten der 4D-Polytope, wie Vektoren, Kanten, Flächen und Farben. Für jedes darzustellende Objekt, wie Polytope in wireframe- oder transparenter Darstellung, wird ein Geometrieknoten verwendet. Zusätzlich findet hier die Berechnung der Transformation und Projektion im 4D-Raum statt. Jede Klasse, die sich von `Polytop` ableitet (siehe Abb. 3.2), erbt die Eigenschaften und Methoden dieser Basisklasse. So ist es möglich, verschiedene 4D-Objekte zu implementieren und auf ihnen 4D-Operationen durchzuführen.

- `Polytop( UInt16 numv, UInt16 nume, UInt16 numf, UInt16 numvf, Real32 mined, Real32 maxed, Real32 minfd, Real32 maxfd )`

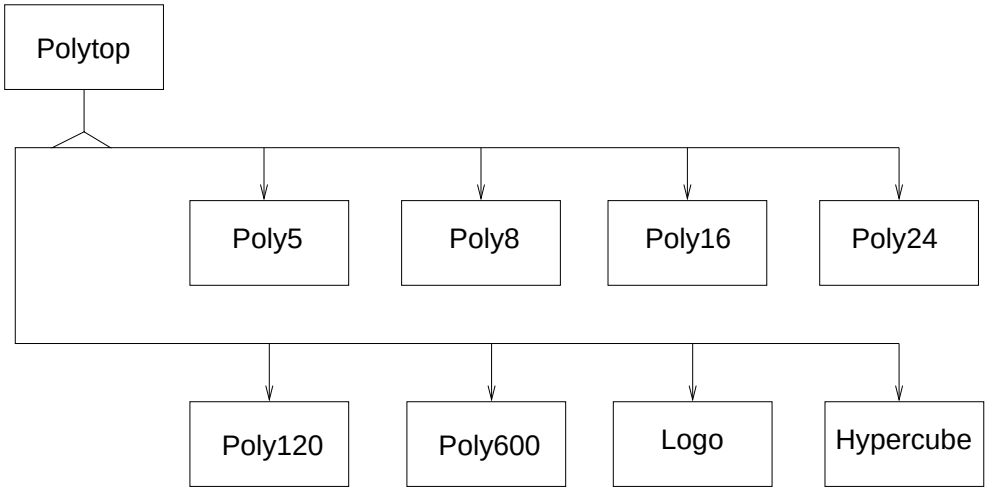


Abbildung 3.2: Die Basisklasse und die abgeleiteten Objekte, wie das 3D-Logo, ein Hyperkubus mit einer transparenten Zelle und alle 4D-Polytope mit der Zellenanzahl 5-600.

Der Konstruktor benötigt als Parameter die Anzahl der Vektoren, Kanten, Flächen und Vektoren pro Fläche. Zusätzlich werden noch die minimale und maximale Tiefen für die Kanten und Flächen übergeben.

- `void init( void )`
  Diese Methode wird einmalig im Konstruktor der abgeleiteten Klasse aufgerufen. Sie initialisiert die Datenfelder und erzeugt den Geometrieknoten des Polytops.

- `Color3f color( Real32 depth, Real32 min, Real32 max )`
  Jeder Vektor im Geometrieknoten bekommt bei der Initialisierung eine Farbe zugewiesen. Diese Farbe wird abhängig von der Tiefe der w-Koordinate und der minimalen und maximalen Tiefe der Kante oder Fläche berechnet.

- `void rotate( Real32 xy, Real32 xz, Real32 yz, Real32 wx, Real32 wy, Real32 wz )`
  Die Rotation des 4D-Objektes wird durch diese Methode erreicht. Übergeben werden die Winkel für jede der sechs Rotationsebenen im Bogenmaß.

- `void translate( Real32 x, Real32 y, Real32 z, Real32 w )`
  Die Translation eines 4D-Objektes benötigt für jede Koordinatenachse einen Wert.

- `void project( bool isOrtho3d, bool isOrtho2d, Real32 f )`
  Die Projektion wird nach allen anderen Transformationen durchgeführt. Als Parameter kann die Art der Projektion eingestellt werden: Entweder orthographisch oder perspektivisch für Projektionen von 4D-3D oder 3D-2D. Zusätzlich lässt sich der Abstand der Projektionsebene zur Kamera übergeben.

- `void drawPoly( UInt8 projection, UInt8 mode )`
  Erst nach Aufruf dieser Methode werden die Transformationen und die Projektionen auf den Geometrieknoten übertragen und auf dem Bildschirm sichtbar. Mittels der Makros `PROJECT_3D` bzw. `PROJECT_2D` als ersten Parameter wird der Grad der Projektion angegeben. Mit `WIREFRAME` oder `TRANSPARENT` als zweiten Parameter werden die beiden Darstellungsformen des Polytops gewählt.

- `NodePtr getPoly( UInt8 projection, UInt8 mode )`
  Mit dieser Methode wird eine Kopie des Geometrieknotens erzeugt, die dann in einen Szenegraph eingefügt werden kann.

3.4 Das Hauptprogramm Dimension

Das Rahmenprogramm beinhaltet sämtliche Funktionen die zur Graphikausgabe auf einem Monitor oder einer Stereoleinwand nötig sind. Ferner werden die Eingabegeräte wie Tastatur, Maus und Joypad abgefragt und die Benuzerinteraktion zu den einzelnen Levels weitergeleitet. Das Hauptprogramm erstellt einen rudimentären Szenegraphen, der die Kamera- und Lichteinstellungen speichert und die weiteren Szenegraphen der einzelnen Levels aufnimmt. Die Methoden sind zum grössten Teil die selben, wie in jedem OpenGL-Programm.

- `int main( int argc, char **argv )`
  Die main-Methode initialisiert das System zur Darstellung auf einen Monitor oder einer Stereoleinwand, instanziiert die Levels und überprüft, ob ein Joypad vorhanden ist.

- `int setupGLUT( int *argc, char *argv[] )`
  Hier wird die GLUT-Bibliothek initialisiert, die für die Fensterverwaltung zuständig ist. Ausserdem werden den Ein-/Ausgabemethoden definiert.

- `void display( void )`
  Diese Methode stellt die Graphikhauptschleife dar, die jedesmal aufgerufen wird, wenn sich der Bildschirminhalt ändert. Vor der Darstellung

wird der Status des Joypads abgefragt und die redraw-Methode des aktuellen Levels aufgerufen.

- `void idle( void )`
  Wenn keine Benutzeraktion erfolgt wird diese Methode angesprungen, die lediglich die Graphikschleife aufruft.

- `void processSpecialKeys( int key, int x, int y )`
  Die Funktions- und Cursortasten werden hier bearbeitet und ihnen einzelne Aktionen zugewiesen.

- `void processNormalKeys( unsigned char key, int x, int y )`
  Jeder normale Tastendruck wird bearbeitet und seiner jeweiligen Aktion zugewiesen.

- `void reshape( int w, int h )`
  Ändert sich die Grösse des Ausgabefensters kümmert sich diese Methode um die Neuzeichnung des Fensterinhaltes.

- `void motion( int x, int y )`
  Übermittelt die Mausbewegung bei gedrücktem Knöpfen.

- `void mouse( int button, int state, int x, int y )`
  Der Status der Maus und ihrer Knöpfe wird hier abgefragt und behandelt.

- `void printDebug( void )`
  Hiermit werden Textmeldungen auf einem Terminal ausgegeben, die den Status des laufenden Programms zeigen.

- `void cleanUp( void )`
  Nach Beenden des Programms löscht diese Methode den reservierten Speicher.

- `void pollJoy(void )`
  Die Abfrage des Joypads über den Zustand seiner Knöpfe wird hier geleistet.

- `float adjustSDLJoystickAxis( int value )`
  Besitzt das Joypad ein analoges Steuerkreuz, so werden diese Werte auf einen Bereich zwischen -1 und 1 abgebildet. Ein Schwellwert verhindert, dass ein leichtes Bewegen des Steuerkreuzes zu einer Aktion führt.

- `NodePtr createSG( void )`
  Mit dieser Methode wird der Szenegraph initialisiert und aufgebaut, um ihn später im Hauptprogramm einzubinden.

Kapitel 4

Ergebnisse

4.1 Animation

Das Animationslevel demonstriert die Erstellung eines Hyperkubus anhand der Analogie zu den Basisobjekten jeder Dimensionsstufe (siehe Abb. 4.1). Der Punkt streckt sich in der ersten Dimension zu einer Linie entlang einer Achse. Im zweiten Schritt verlängert sich die Linie zu einem Quadrat, im dritten das Quadrat zu einem Würfel. Der letzte Schritt zeigt einen Hyperkubus, der sich aus einem Würfel bildet.

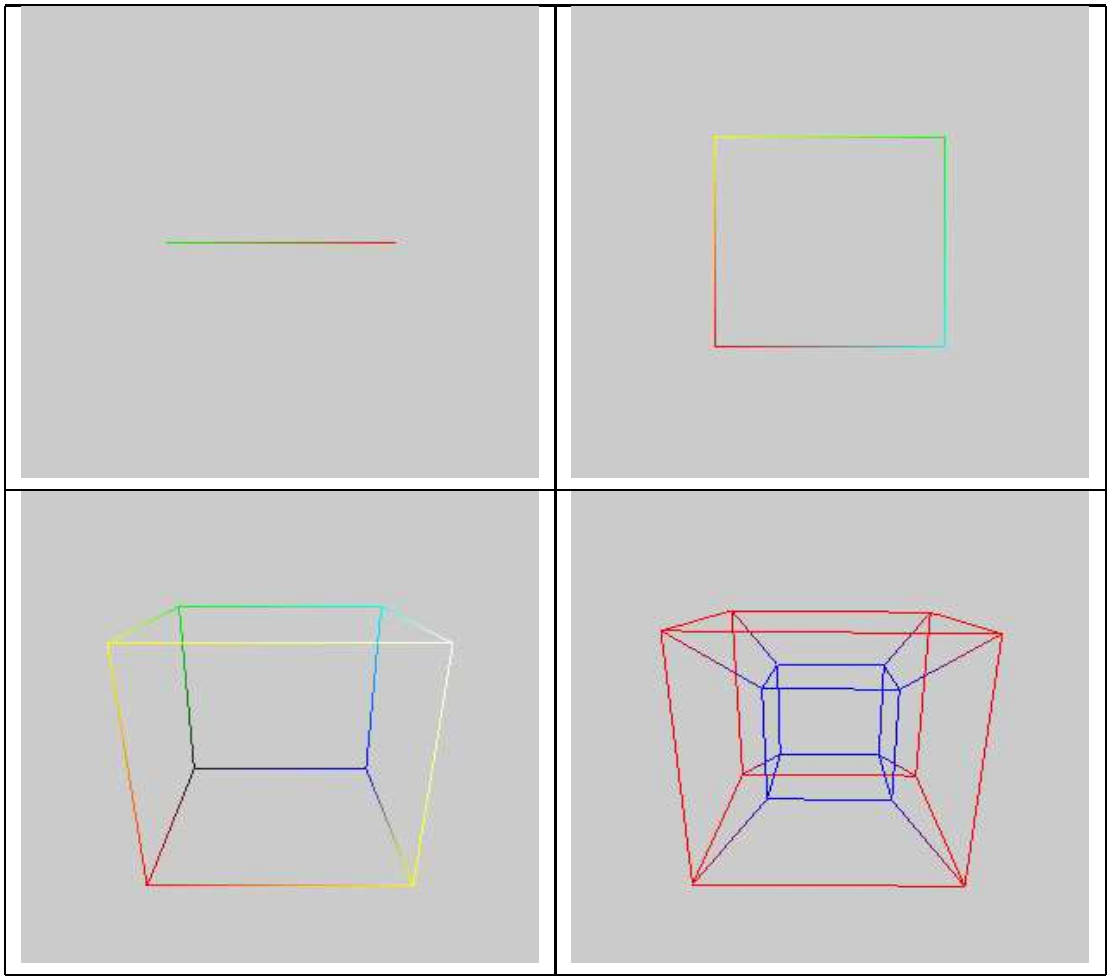


Abbildung 4.1: Der Hyperkubus, generiert aus einer Linie, Quadrat und Würfel

4.2 Projektion

Im zweiten Level wird der Zusammenhang zwischen Schattenwurf und Projektion sichtbar. Ein 4D-Polytop wird in den Raum projeziert, wo es einen dreidimensionalen Körper darstellt. Dieser Körper wird erneut auf eine Ebene projeziert, was einem Schattenwurf des 3D-Körpers gleicht. Der Benutzer kann daraus folgern, dass das 3D-Polytop auch ein räumlicher Schatten eines vierdimensionalen Objektes ist. Dargestellt werden die wichtigsten vierdimensionalen Polytope, die auch n-Zellen benannt werden. Die Anzahl der Zellen entspricht der Anzahl an 3D-Körpern, aus dem das 4D-Polytop zusammengesetzt ist. So heisst ein Hyperkubus auch 8-Zelle, weil er aus acht Würfeln zusammengesetzt ist oder die 5-Zelle, die aus fünf Tetraedern besteht. Die Polytope können abwechselnd im Wireframe- oder Transparenz-Modus betrachtet werden. Auch die orthographische oder perspektivische Projektion von 4D-3D oder 3D-2D ist einstellbar. Neben den Polytopen kann auch ein Wireframe-Hyperkubus mit einer transparenten Zelle (siehe Abb. 4.2) oder ein herkömmliches 3D-Logo angewählt werden. Der Benutzer kann an ihnen die Rotationsmöglichkeiten im Hyperraum austesten und nachvollziehen.

4.3 Dimensionsanalogie

Das dritte Level demonstriert wieder die Analogie zwischen Objekten verschiedener Dimensionen (siehe Abb. 4.3). Die Entstehung eines 4D-Simplex wird anhand der Darstellung einer Linie, Dreieck, Tetraeder und 5-Zelle veranschaulicht. Mit jedem Dimensionsgrad mehr erhöht sich auch die Anzahl der Punkte des Simplex. Auch die Anzahl der Kanten, die von einem Punkt ausgehen, entspricht der Dimensionszahl. Der Benutzer kann die einzelnen Objekten bewegen und zwischen ihnen umschalten. Auch die Entstehung eines Hyperkubus aus einem Würfel und der 16-Zelle aus einem Oktaeder können angezeigt werden.

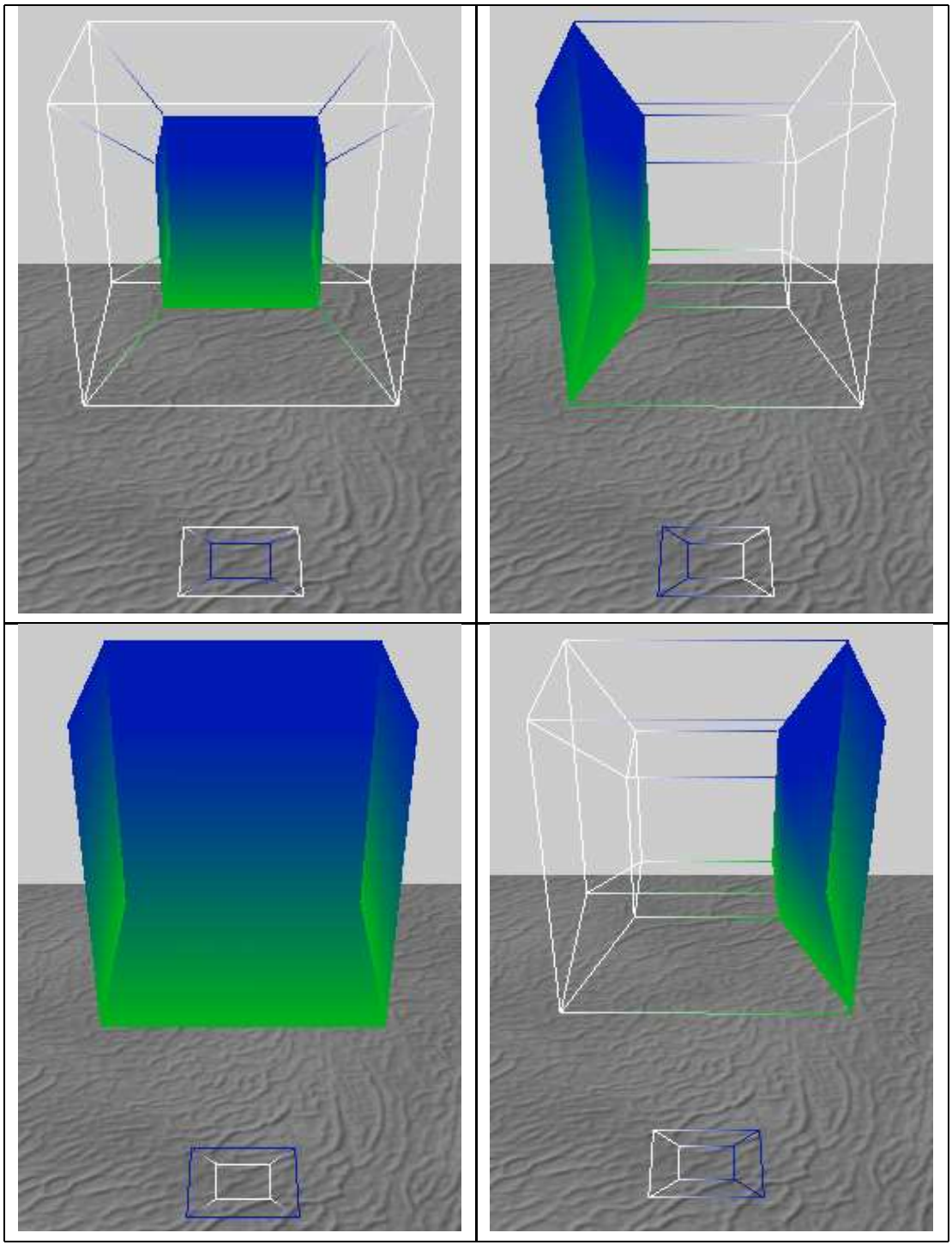


Abbildung 4.2: Hyperkubus (oder 8-Zelle) rotiert an der wx-Ebene um die Winkel 0, $\frac{\pi}{2}$, π und $\frac{3\pi}{4}$

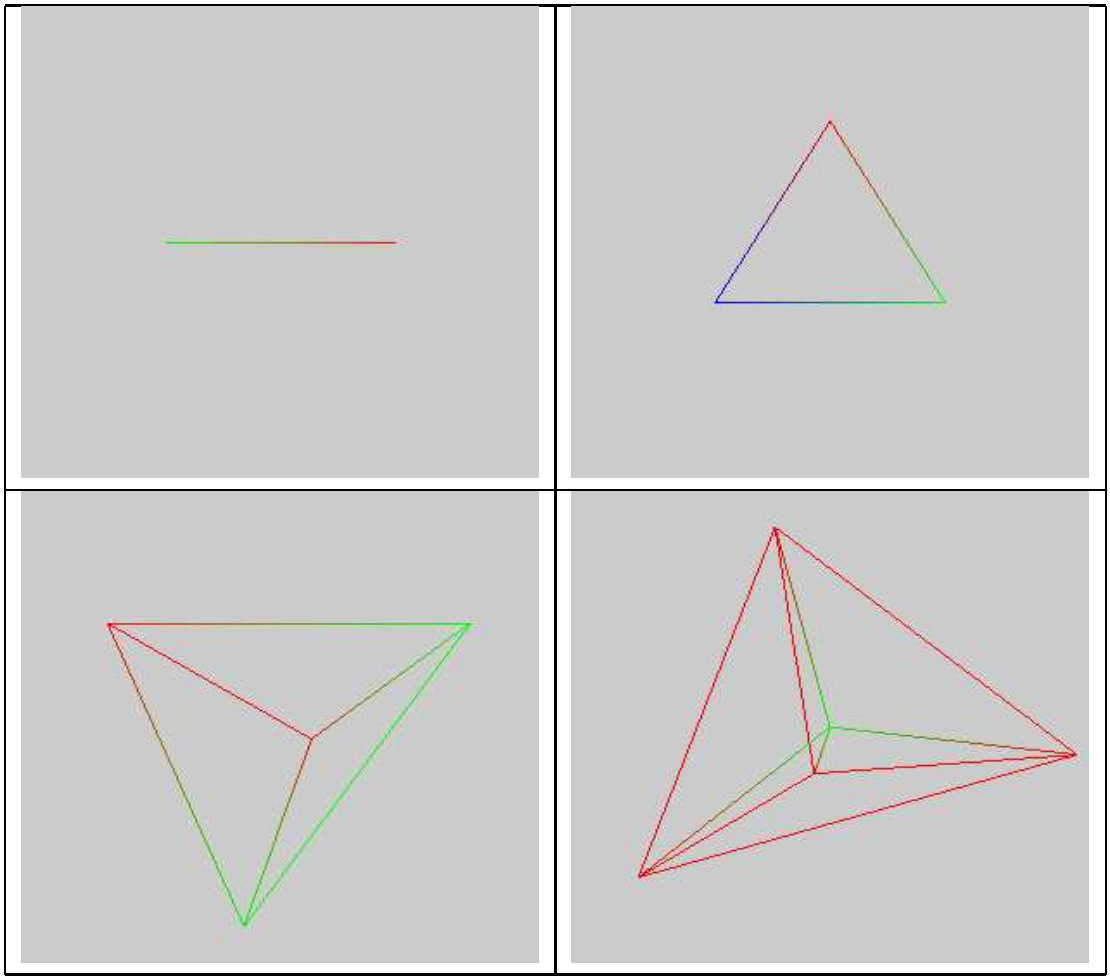


Abbildung 4.3: Die Simplex-Analogie, angefangen von der Linie über Dreieck, Tetraeder bis zur 5-Zelle.

Kapitel 5

Fazit

Die Ziele dieser Studienarbeit sind teilweise erreicht worden. Die Darstellung von vierdimensionalen Polytopen, wie Hyperkubus und 4D-Simplex auf einem Stereowiedergabesystem ist zufriedenstellend. Durch den neu hinzugewonnenen Tiefeneindruck, den der Benutzer vor der Stereoleinwand erfährt, lassen sich die ineinander verwobenen Eckpunkte der Polytope visuell besser räumlich trennen. Gerade die 4D-Rotation der Objekte, die sich als ein Verformen und Biegen in der Darstellung äussert, lassen sich nun besser verfolgen und nachvollziehen. Leider lässt die Darstellung der Objekte mit transparenten Flächen zu wünschen übrig, da die z.T. überschneidenden Flächen nicht korrekt und mit störenden Farbverläufen abgebildet werden. Hier wäre eine tiefere Kenntnis und eine detailiertere Dokumentation von OpenSG nötig, um dieses Problem zu beseitigen.
Die Interaktion mittels Joypad genügt den Anforderungen an die Anwendung. Szenenwechsel können per Knopfdruck oder Rotation über analoge Steuerachsen durchgeführt werden. Leider konnte am Joypad das Steuerkreuz nicht aktiviert werden, was vermutlich ein Problem der SDLlib ist, die die Hardware nicht korrekt erkannt hat. Um noch eine intuitivere Interaktion zu ermöglichen, lohnt sich die Anbindung eines Trackingsystems, das die Position des Joypads in den Händen des Benutzers misst und diese Daten an das Programm weiterleitet. Somit wäre eine Art Drag and Drop möglich, indem der Benutzer das Objekt ergreift und durch Bewegung des Joypads in eine Richtung zieht oder rotiert.
Die didaktische Komponente war schwer zu realisieren, da das Thema sich stark an mathematische Grundlagen anlehnt und recht abstrakt ist, da sich viele unter der vierten Dimension etwas anderes vorstellen. Gerade die Art der Wahrnehmung von Hyperobjekten ist gewöhnungsbedürftig, entspricht sie doch nicht dem alltäglichen Bild und scheint so manches gewohntes Naturgesetz zu brechen. Vergegenwärtigt man aber die Tatsache, dass Beziehungen zwischen 2D und 3D analog zu den Beziehungen zwischen 3D und 4D stehen, erleichtert in einigen Fällen das Umdenken. So wären graphische Umsetzungen der anderen Visualisierungsmöglichkeiten, wie Faltungen und Schnitte durchaus von Vorteil, um sich weitere Techniken anzueignen, die vierdimensionales Verständnis fördern.

Literaturverzeichnis

[1] Edwin Abbott Abbott. *Flatland*. New York: Dover Publications, Inc. 1952.

[2] Carsten Steger. *Polytopes*. Bildschirmschoner im XScreenSaver-Programm. 2003.
`http://www.jwz.org/xscreensaver`

[3] Thomas F. Banchoff. *Dimensionen - Figuren und Körper in geometrischen Räumen*. Spektrum der Wisenschaft. 1991.

[4] Oliver Abert. *OpenSG Starter Guide*. 2004.
`http://www.oliver-abert.de/opensg/StarterGuide.pdf`

[5] James D. Foley. *Computer Graphics - Principles and Practise*. Addison-Wesley. 1990.

[6] Eric Zilli. *Visualization of Four-Dimensional Objects*. Kalamazoo College, Michigan. 2003.
`http://max.cs.kzoo.edu/~ezilli/4dvis.htm`

[7] Steven R. Hollasch. *Four-Space Visualization of 4D Objects*. Arizona State University. 1991.
`http://stevehollasch.com/thesis`