universität wien

DIPLOMARBEIT / DIPLOMA THESIS

Titel der Diplomarbeit / Title of the Diploma Thesis

„Darstellung von Videospielen als Realitätsbezug im Mathematikunterricht der Sekundarstufe II“

verfasst von / submitted by

Oliver Scherbl

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Magister der Naturwissenschaften (Mag. rer. nat.)

Wien, 2018 / Vienna, 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 190 406 412
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Lehramtsstudium UF Mathematik UF Physik
Betreut von / Supervisor:	Doz. Dr. Franz Embacher

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser wissenschaftlichen Arbeit unterstützt haben.

- Zu Beginn möchte ich besonders meinen Eltern danken, die mich mein ganzes Leben und während meines langen Studiums unterstützt haben.
- Weiters möchte ich mich bei meinen Freunden bedanken, deren Hilfe und Unterstützung mein Studium lustiger und weit weniger mühsam gestaltete. Insbesondere gilt mein Dank Angelika Ambrusch, Melanie Strube und Michael Mehsam.
- Ich möchte mich auch bei den Rocket Beans und Funhaus bedanken, deren Videos mir durch stressige Zeiten halfen.
- Und zu guter Letzt gilt mein besonderer Dank meinem Betreuer Franz Embacher, dessen Geduld und Unterstützung diese Diplomarbeit erst möglich gemacht haben.

Inhalt

1. Einleitung

1.1. Absicht und Zielsetzung der Arbeit

Was 1958 mit einer Demonstration für Besucher des Brookhaven National Laboratory begann[1], ist heute eine Milliardenschwere Industrie, die sogar die Filmindustrie Hollywoods übertrifft[2]: Die Rede ist von Videospielen.

In den frühen Exemplaren konnte man lediglich zweidimensionale Figuren durch zweidimensionale Welten bewegen(siehe Abb.1), und Aktionen in diesen Spielen waren sehr limitiert: Laufe von links nach rechts, spring auf Gegner, sammle Münzen.

Abb.1: Super Mario Bros. (1985)

In den 90er Jahren kam es dann zu einem Wandel. 3D fand Einzug in Film und Videospielen. Plötzlich konnte man Figuren frei im Raum bewegen und virtuelle Spielwelten aus allen Blickwinkeln betrachten. Das Konzept einer frei steuerbaren "Kamera" war in den ersten 3D-Spielen so komplex, dass Nintendo im Spiel *Super Mario 64* einen Kameramann einführte, den man durch bewegen des Kamera-Joysticks lenken konnte (siehe Abb.2).

Abb.2: Super Mario 64 (1996)

Hier war aber noch nicht Schluss. Die wachsende Rechenleistung von Computern in den letzten Jahren ermöglicht nun fotorealistische Welten, die in Echtzeit auf einem Bildschirm generiert werden (siehe Abb.3). Zudem werden Gesetzmäßigkeiten der klassischen Mechanik simuliert, um virtuelle Objekte realistisch miteinander interagieren zu lassen.

Abb.3: Final Fantasy 15 (2016)

Diese letzten zwei Punkte weckten während meines Studiums mein Interesse. Rechenstarke Computer sind eine Seite, aber was ist wirklich nötig, um atemberaubende Landschaften mit

[1] Vgl. bnl.gov [Zugriff am 2.7.2017]
[2] Vgl. newzoo.com [Zugriff am 2.7.2017]

Gräsern, die sich realistisch im Wind bewegen und Bäumen, die Schatten werfen darzustellen, und um auch sonst jede Aktion einer Spielfigur oder eines Objekts realitätsnahe wirken zu lassen. Für eine fließende Bewegung müssen mindestens 15 Bilder pro Sekunde wiedergegeben werden[3], moderne Videospiele laufen sogar mit 60 Bildern pro Sekunde. Bei einem Film ist das nur eine fixe Abfolge von festgelegten Bildern. Ein Spiel ist jedoch interaktiv; es gibt keine vorherbestimmte Abfolge, entsprechend muss daher das Spiel jede Sekunde 60 neue Bilder generieren, wenn der Spieler die Figur steuert und Aktionen ausführt. Hinzu kommt noch, dass die Darstellung auf einem Bildschirm so erfolgt, dass jeder Bildpunkt, auch *Pixel* (*pictureelement*) genannt[4], eine bestimmte Farbe wiedergibt und die Summer dieser Bildpunkte aus der Entfernung gesehen ein Bild ergeben. Bei Bildschirmen mit einer HD-Auflösung sind das 1920 × 1080 Pixel[5]. Das bedeutet wiederum, dass jedes Bild bei einem Spiel im Bruchteil einer Sekunde in 2073600 Pixel zerlegt werden muss.

Zudem sollen von der Kamera weiter entfernte Objekte immer kleiner werden, Oberflächen durch Reflexion und Aufhellung auf Lichtquellen reagieren, Gegenstände Schatten werfen und so weiter. Das alles und noch mehr muss so schnell wie möglich und automatisch erfolgen, wobei hier die Mathematik zum Einsatz kommt. Wie in den folgenden Kapiteln zu sehen sein wird, kommt die dabei angewandte Mathematik hauptsächlich aus den Bereichen Geometrie und lineare Algebra. Der Prozess der Bilderzeugung bei interaktiven Medien wie Videospielen wird unter dem Begriff **Echtzeit-Bildsynthese** zusammengefasst. Diese erfordert effiziente Lösungsmethoden für Aufgaben wie Sichtbarkeitsprobleme und Kollisionserkennung. Die Entwicklung von Methoden zur effizienten Lösung geometrischer Probleme macht Videospiele nicht nur für die mathematische Forschung[6], sondern auch für den Oberstufenunterricht interessant. Zum besseren Verständnis wird in naturwissenschaftlichem Unterricht wie Physik der Stoff oft durch Versuche veranschaulicht. Ebenso werden im Mathematikunterricht Beispiele aus dem Alltag herangezogen, um die Anwendung mathematischer Methoden praxisorientierter darzustellen.

Abb.4: Angry Birds (2009)

Da viele SchülerInnen mit Videospielen vertraut sind, könnten Nutzen und Anwendungen dieser mathematischen Methoden in Bezug auf dieses Medium interessanter und besser

[3] Vgl. Akenine-Möller (2008), S. 1
[4] Vgl. Lyon (2006), S. 1
[5] Vgl. Wikipedia.org [Zugriff am 3.7.2017]
[6] Vgl. Akenine-Möller (2008), S. 2

vermittelt werden. Aufgaben aus der Echtzeit-Bildsynthese können hier Abhilfe schaffen. Neben der Echtzeit-Bildsynthese beinhalten Videospiele auch selbstständig ablaufende Vorgänge, welche durch Formeln der klassischen Mechanik beschrieben werden können. Somit lässt sich auch zum Gegenstand Physik eine Verbindung herstellen, beispielsweise die Berechnung der Flugbahn eines abgeschossenen Vogels im Spiel *AngryBirds* (siehe Abb. 4).

Diese Arbeit beschäftigt sich mit den mathematischen Lösungen von Problemen, die in Videospielen auftreten und deren Bezug zum Schulstoff. Das Studium diverser Grundlagenbücher für Spieleentwickler hat gezeigt, dass sich die meisten Probleme auf die Bereiche der Vektorrechnung und der analytischen Geometrie beziehen. Ziel dieser Arbeit ist es, einen grundlegenden Überblick über die Entwicklung von Videospielen zu geben und damit verbunden, Anwendungsmöglichkeiten von Bereichen aus dem Mathematikunterricht der Oberstufe aufzuzeigen.

1.2. Wichtige Begriffe

Fragment: Die notwendigen Daten zur Berechnung eines Pixels. Enthalten Informationen über Position, Tiefe, Farbe, Lichteffekte, usw.
Frame: Das Bild, das vom Computer generiert wird.
Hitbox: Umgrenzungsvolumen um ein virtuelles Objekt.
Paralax Scrolling: Effekt zur Simulierung von Tiefe in 2D-Spielen. Verschiedene Hintergrundebenen bewegen sich unterschiedlich schnell vorbei.
Pixel: kurz für *Picture Element*. Einfarbiger Bildpunkt und kleinstes Element des Bildschirms
Polygon: einfache geometrische Form (meistens ein Dreieck) und Grundbestandteil von 3D-Modellen
Raycasting: Methode zur Bestimmung von Abständen bei der ein Strahl von einem Objekt ausgeht.
Raytracing: Methode zur Bestimmung der Beleuchtung bei der der Verlauf eines gestreuten Lichtstrahls berechnet wird.
Rendern: Das Erstellen einer Grafik, basierend auf 3D-Rohdaten
Scaling: Vergrößern oder Verkleinern von Objekten.
Shader: Anweisungen, die auf jeden Pixel angewandt werden, um deren Farben zu verändern. Kommen bei Lichteffekten zum Einsatz.
Sprite: Grafikelement, das vor dem Hintergrundbild eingeblendet wird.
Szene: Ansammlung von Modellen
Texel: = *Textur Pixel*. Kleinstes Element einer Textur.

Textur: Bilddatei, die auf Oberflächen platziert wird.

Vertices: Eckpunkte der Polygone.

z-Buffer: Zwischenspeicher, der für jeden Pixel den virtuellen Abstand zum Bildschirm speichert.

1.3. Koordinatensysteme in dieser Arbeit

Kartesische Koordinatensysteme in dieser Arbeit sind, wenn nicht ausdrücklich anders beschrieben, rechtshändig mit der x-Achse als Rechtsachse, der y-Achse als Hochachse und der z-Achse als Tiefenachse festgelegt.

2. Allgemeine Grundlagen zur Darstellung von Videospielen

2.1. Entwicklung der 3D-Darstellung

Das erste Videospiel *Pong* wurde auf einem Oszilloskop gespielt (siehe Abb.5). Dabei wurde der Elektronenstrahl so über den Bildschirm gelenkt, dass Linien und Kurven dargestellt wurden, die das Netz, den Boden und den Tennisball repräsentieren sollten [7]. Generell erforderten die ersten Videospiele einiges an Fantasie, da die Welten und Figuren durch die verfügbare Technik meistens nur durch Linien und Ansammlungen an bunten Pixeln angedeutet werden konnten.

Abb.5: Nachbildung der Urversion von Pong

Pong stellte auch die erste der zwei Darstellungsvarianten von Bildern dar: die *Vektorgrafik*. Hierbei bestehen Bilder und Objekte aus einfachen geometrischen Elementen wie Kreisen und Linien, die durch Funktionen beschrieben werden können. Bei frühen Spielen, die Vektorgrafiken einsetzten, wurde der Elektronenstrahl des Monitors direkt manipuliert und so über den Bildschirm gelenkt, dass er die Konturen der Objekte nachzeichnete. Diese Darstellungsart ermöglichte schon früh die Wiedergabe von 3D-Welten. Da jedoch nur Konturen gezeichnet werden konnten, musste auf Details wie Farben und Oberflächen verzichtet werden (siehe Abb. 6)[8].

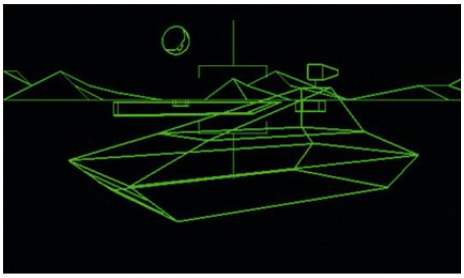

Abb.6: Battlezone (1980)

Die zweite Darstellungsvariante ist die *Rastergrafik*. Hierbei besteht das Bild aus einer rasterförmigen Anordnung von Pixeln, denen jeweils eine Farbe zugeordnet ist. In Summe wird für das Auge ein Bild erzeugt (siehe Abb. 7). Bei Röhrenbildschirmen wurde der Elektronenstrahl Zeile für Zeile über den Bildschirm gelenkt. Bei modernen LED-Bildschirmen werden drei Leuchtdioden (rot, grün, blau) in jedem Pixel nach Wunsch zum Leuchten gebracht. Das Bild wird 50 bis 60 mal in einer Sekunde aktualisiert, und die Farben der Pixel werden bei Bedarf geändert, um für die Augen den Eindruck von Bewegung zu vermitteln. Der Nachteil der Rastergrafik im

Abb.7: The Legend of Zelda (1986)

[7]Vgl. bnl.gov [Zugriff am 2.7.2017]
[8]Vgl. Wolf (2008), S. 67

Vergleich zur Vektorgrafik ist, dass bei einer Vergrößerung einer Rastergrafik auch die Pixel vergrößert werden, wodurch das Bild unscharf wird. Formen, die durch Vektoren beschrieben werden, können beliebig vergrößert werden[9].

2D-Spiele funktionieren prinzipiell der Art, dass auf einer Hintergrundebene Bilder von Objekten und Figuren platziert werden (siehe Abb.8), sogenannte *Sprites*. Bei komplexeren Spielen werden mehrere Hintergrundebenen verwendet. Für jede Figur existieren mehrere Sprites, die den Verlauf einer Bewegung darstellen sollen, vergleichbar mit der Animation eines Cartoons (siehe Abb. 9). Je nachdem wie flüssig eine Bewegung sein soll, werden mehr oder weniger Bilder für die Animation benötigt. Soll sich eine Figur bewegen, oder eine Aktion ausführen, zum Beispiel aufgrund einer Eingabe am Controller, dann werden nacheinander die Sprites der Animationsfolgen abgerufen. Für Interaktionen kontrolliert das Programm ständig die Koordinaten der Sprites. Überschneiden sich die Koordinaten eines Sprites mit einem anderen, dann wird im Programm eine Reaktion ausgelöst. Dies stellt sich beispielsweise wie folgt dar: Die Überschneidung von Schildkröten-Sprite mit Mario-Sprite führt dazu, dass die Sterbeanimation von Mario abgespielt wird. 2D-Spiele waren im Grunde nur eine Datenbank an Bildern, die je nach Bedarf am Bildschirm dargestellt wurden, einzig und allein durch den Speicherumfang beschränkt. Mit steigendem Speicherplatz wurden auch die Sprites immer detaillierter (siehe Abb. 10) und erlaubten sogar den Einsatz von digitalisierten Realbildern[10].

Abb.8: Mario Maker (2016)

Abb.9: Sprites für Mario's Laufanimation

Abb.10: Street Fighter (1987)

Schon früh versuchten Spielentwickler frei erkundbare, dreidimensionale Welten zu kreieren. Das Prinzip dahinter sind *Polygone*, Ansammlungen von Punkten, die durch drei Raumkoordinaten definiert sind und durch Strecken verbunden werden,

Abb.11:I, Robot (1984)

[9]Vgl. foldoc.org[Zugriff am 14.2.2018]
[10]Vgl. prog21.dadgum.com [Zugriff am 14.2.2018]

um ein Objekt durch ein *Drahtgerüst* (*Wireframe*) darzustellen. In Videospielen kommen überwiegend dreiseitige Polygone zum Einsatz. Die Entwickler wurden jedoch durch die damals verfügbare Rechenleistung limitiert, die nur die Darstellung dieser Wireframes erlaubte (siehe Abb. 6). Später war es möglich, die Oberflächen der Polygone darzustellen, jedoch nur in einer Farbe pro Fläche, und auch die Farbpallette war limitiert (*Flatshading*) (siehe Abb. 11). Die meisten Spiele dieser Zeit blieben deshalb zweidimensional. Spielentwickler versuchten jedoch mit Tricks den Eindruck von 3D zu erwecken[11].

Eine frühe Technik dafür ist das *Paralax Scrolling*. Hier werden mehrere Hintergrundebenen verwendet, die sich mit unterschiedlicher Geschwindigkeit vorbeibewegen, die vorderste am schnellsten und die hinterste am langsamsten. So entsteht ein Gefühl von perspektivischer Tiefe. Ein früher Vertreter dieser Technik ist das Spiel *Moon Patrol* (siehe Abb. 12), bei dem der orangene Boden, die grünen Hügel und die blauen Berge drei unterschiedliche Ebenen darstellen, die verschieden schnell vorbeibewegt werden[12].

Abb.12: Moon Patrol (1982)

Eine weitere Methode, um Tiefe zu simulieren, ist das *Sprite Scaling*. Soll sich ein Sprite vom Hintergrund in den Vordergrund bewegen, so wird er in einem entsprechenden Maß vergrößert. Soll er sich in den Hintergrund bewegen, wird er verkleinert. Im Spiel *Turbo* wurden die Sprites der Gebäude gleichzeitig nach außen verschoben und vergrößert, während Rennautos weiter vorne verkleinert wurden (siehe Abb. 13)[13].

Abb.13: Turbo(1981)

Ein Darstellungsmodus von Hintergründen der Super-NES Konsole ermöglichte das Rotieren und Skalieren einer Hintergrundebene. Dieser sogenannte *Mode 7* Modus ermöglichte die Erzeugung eines Perspektiveneffekts einer Hintergrundebene durch geschicktes Skalieren und Rotieren. Der Hintergrund wird in eine horizontale Ebene transformiert und erweckt einen dreidimensionalen Eindruck (siehe Abb. 14)[14].


Abb.14: Final Fantasy 6 (1994)

[11]Vgl. Wolf (2008), S. 247
[12]Vgl. Ebd., S. 11
[13]Vgl. Ebd., S. 11
[14]Vgl.Next Generation.No. 15. (1996), S. 37

Die größte Schwierigkeit beim Darstellen von 3D-Welten war das Zeichnen der Texturen, die frei im Raum rotiert werden. Texturen sind Bilder, die über Polygone gespannt werden, um ihnen ein Aussehen zu geben. In 3D-Spielen muss die Position der Texturen in Abhängigkeit der Polygon-Eckpunkte (*Vertices*) ermittelt und transformiert werden. Dies erfordert viel Rechenarbeit. Das Spiel *Doom* umging dieses Problem durch die *Raycasting*-Methode. Für die Spielwelt wurden auf eine flache Ebene Wände aus einfachen rechteckigen Texturen platziert. Ausgehend von der Entfernung zu einem Objekt, das ein "Sichtstrahl" trifft (Abb. 15), wird die vertikale Länge berechnet, die dargestellt werden soll (siehe Abb. 16). Da nur eine Bildzeile abgetastet wird, war eine Bewegung der Kamera nur horizontal möglich, wodurch alle Wände immer senkrecht orientiert waren. Texturen mussten dadurch, je nach Tiefe, nur skaliert werden. Auch abnehmende Beleuchtung in weiterer Ferne konnte so ermittelt werden. Spielfiguren waren zweidimensionale Sprites, die in diese dreidimensionale Umgebung platziert wurden (siehe Abb. 17)[15].

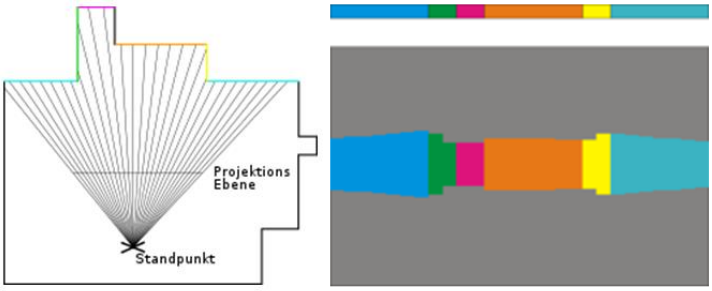

Abb.15 & 16: Raycasting Prinzip

Abb.17: Anwendung in Doom

Mitte der 90er Jahre gelang der erste Vorstoß in *"True-3D"* die Darstellung von frei beweglichen dreidimensionalen Objekten (siehe Abb. 18). Möglich machten das Prozessoren, die zur Entlastung des Hauptprozessors zusätzlich eingesetzt wurden. Diese sogenannten GPU (Graphic Processing Unit) berechnen die Computergrafik[16].

Abb.18: Virtua Fighter (1993)

Der Prozess der Echtzeit-Bildsynthese von 3D-Spielen wird im folgenden Kapitel näher beschrieben.

[15]Vgl. Becker, S. (1996), S. 246
[16]Vgl. Wolf (2008), S. 147

2.2. Grafikpipeline

Die einfachsten Grafikelemente in 3D-Spielen sind Punkte, Linien und Dreiecke (Polygone). Aus der Kombination dieser geometrischen Elemente werden die Oberflächen von Objekten beziehungsweise Modellen gebildet. Da das Resultat einem Drahtgerüst ähnelt, spricht man von *Wireframes*. Dabei gilt: je mehr Punkte man setzt, umso detailierter kann man ein Objekt zeichnen (siehe Abb. 19). Dazu werden Datenbanken erstellt, die alle Punkte und Normalvektoren der Polygone enthalten. Die Punkte sind die Grundlage zum Zeichnen der Polygone. Die Normalvektoren dienen zur Berechnung der Sichtbarkeit und Beleuchtung. Mehr Punkte führen aber zu mehr Rechenaufwand und in der Folge zu einem größeren Speicherbedarf, da jeder Punkt durch Raumkoordinaten festgelegt ist, die bei Bedarf geändert werden müssen[17].

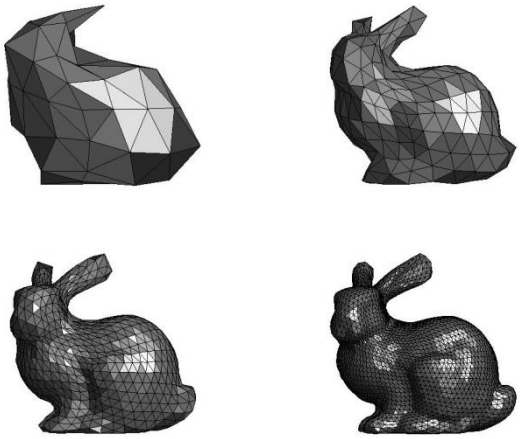

Abb. 19: Wireframe eines Hasen-Modells

Eine Ansammlung von Modellen nennt man Szene. Eine Szene kann auch Materialeigenschaften, Beleuchtungs- und Betrachtungsspezifikationen beinhalten. Kern der Bildsynthese ist es, aus diesen 3D-Daten ein 2D-Bild zu erzeugen. Dazu muss, von einer virtuellen Kamera ausgehend, berechnet werden, welche Objekte sichtbar sind, wie die Oberflächen aussehen und wie das Licht verteilt ist[18].

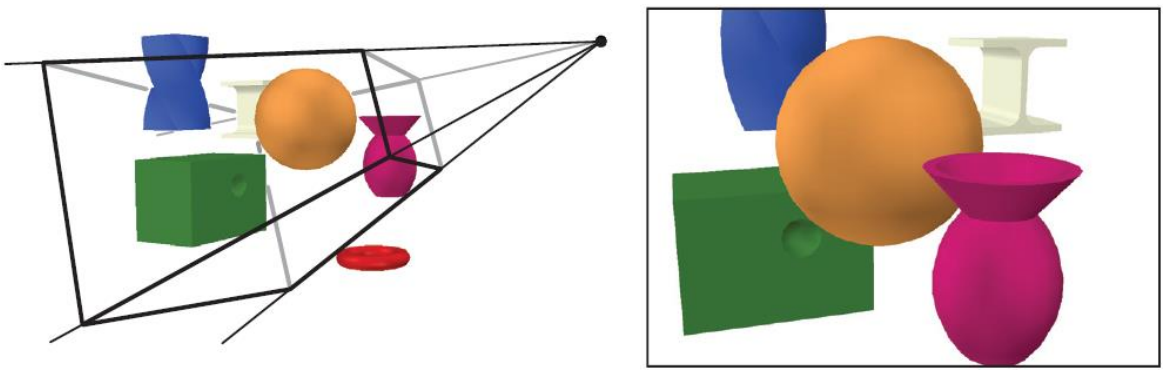

Abb.20: Sichtvolumen und virtuelle Kamera

Das Prinzip der virtuellen Kamera wird in Abb. 20 dargestellt. Die "Kamera" sitzt im linken Bild in der Spitze der Pyramide. Den Pyramidenstumpf nennt man *Sichtvolumen.* Die Deckfläche des Stumpfes entspricht dem Bildschirm, auf sie werden die Objekte später projiziert. Nur die Objekte innerhalb des Sichtvolumens sollen später dargestellt werden. Die

[17] Vgl. Akenine-Möller (2008), S. 8
[18] Vgl. Ebd. S. 11

Pyramidenstumpfform ist außerdem die Grundlage zum Simulieren einer Perspektive. Der Stumpf wird im Zuge der Projektion zu einem Quader transformiert und mit ihm alle enthaltenen Objekte. Objekte weiter hinten werden in Folge stärker gestaucht als Objekte im Vordergrund und erscheinen somit kleiner. Das rechte Bild zeigt was die Kamera "sieht". Man beachte, dass der rote Torus aus dem linken Bild nicht gezeichnet wurde, da er sich außerhalb des Sichtvolumens befand. Außerdem wurde das verdrehte blaue Objekt aus dem linken Bild oben abgeschnitten[19].

Die nötigen Prozesse zur Bildberechnung werden bei der Echtzeit-Bildsynthese durch die *Grafikpipeline* beschrieben. Sie dient als Modellbeschreibung und kann in drei Schritte unterteilt werden: *Anwendung*, *Geometrie* und *Rasterung* (Siehe Abb. 21). Diese Schritte werden bei jedem einzelnen Bild durchlaufen. Jeder Einzelschritt kann aus nur einem Prozess, einer eigenen Pipeline, oder parallelisierten Prozessen bestehen[20].

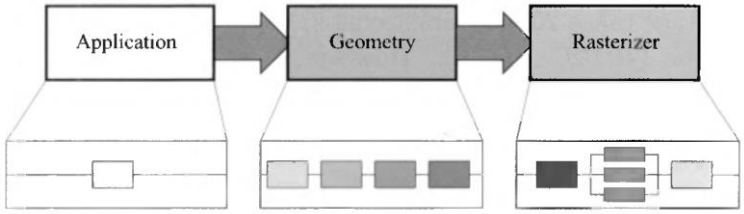

Abb. 21: Ablauf der Grafikpipeline

In den folgenden Kapiteln werden die einzelnen Schritte näher erläutert.

2.2.1 Anwendung

Im Anwendungsschritt werden durch Interaktionen des Anwenders Änderungen an der Szene und in weiterer Folge an den Punkten und Dreiecken der Objekte vorgenommen. Hier kommen Techniken wie Kollisionserkennung, Animation, Morphing, Physik Simulation und Beschleunigungsverfahren zum Einsatz[21].

Kollisionserkennung[22,23]

Damit eine Spielfigur in der virtuellen Welt nicht durch Wände oder Gegenstände gleitet, muss das Spiel erkennen, wann es zu einer Berührung oder Überschneidung der geometrischen Formen kommt, aus denen Objekte bestehen. Das ist der Zweck der Kollisionserkennung. Moderne Charaktere können aus 40000 bis 500000 Dreiecken

[19] Vgl. Akenine-Möller (2008), S. 12
[20]Vgl. Ebd.S. 13
[21]Vgl. Tremblay, C. (2004), S. 14
[22]Vgl. Akenine-Möller (2008), S. 726
[23]Vgl. Tremblay (2004), S. 116

bestehen[24]. Die Überschneidung jedes einzelnen Dreiecks in einem Sekundenbruchteil zu testen ist sehr unpraktisch. Stattdessen werden Objekte mit unsichtbaren, einfachen geometrischen Formen umschlossen, sogenannten *Hitboxen* (siehe Abb. 22). Neben Quadern, werden auch oft Zylinder und Kugeln als Hitboxen verwendet. Ein Überschneidungstest soll feststellen, ob sich zwei Objekte A und B überschneiden, was bedeuten kann, dass sich A komplett in B befindet (oder umgekehrt), dass sich die Grenzflächen von A und B schneiden, oder dass sie voneinander getrennt sind. Manchmal kann auch der Schnittpunkt gesucht werden oder der Abstand zu diesem.

Abb. 22: Hitboxen um ein Charaktermodell

Physik[25]

Um Spielwelten realistisch erscheinen zu lassen, ist es notwendig, die physikalischen Gesetze der Realität in einem gewissen Maß nachzubilden (siehe Abb.23). Hebt man einen Gegenstand auf und lässt ihn wieder los, dann soll er zu Boden fallen und nicht in der Luft hängen bleiben. Fährt man gegen ein Auto, dann soll es aus der Bahn geworfen werden und nicht wie eine Betonwand stehen bleiben. Eine Fahne soll im Wind wehen und nicht wie ein Brett an einer Stange hängen. Da es jedoch nur darauf ankommt, Dinge zu bewegen, sprich die Koordinaten von Objekten zu verändern, beschränkt sich die nötige Physik überwiegend auf Bereiche der klassischen Mechanik. Für ein Objekt, das mit der Horizontalgeschwindigkeit v_x und v_z von einer Klippe fällt, lässt sich die Gravitation einfach durch eine Beschleunigung nach unten simulieren, deren Wert frei bestimmt werden kann. Ein Punkt $\vec{p}(t)$ mit den Koordinaten (x, y, z) kann dann wie folgt durch eine Matrix, in Abhängigkeit der Zeit t, berechnet werden:

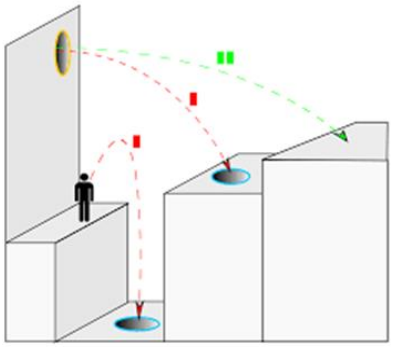

Abb. 23: Plattform-Rätsel aus *Portal* (2007)

$$\vec{p}(t) = \begin{bmatrix} 0 & v_x & x_0 \\ a_0 & v_y & y_0 \\ 0 & v_z & z_0 \end{bmatrix} \begin{bmatrix} t^2/2 \\ t \\ 1 \end{bmatrix} = \begin{pmatrix} v_x t + x_0 \\ a_0 \, t^2/2 + v_y t + y_0 \\ v_z t + z_0 \end{pmatrix}$$

In Kapitel 2.4 wird näher auf dieses Thema eingegangen.

[24]Vgl. polycount.com [Zugriff am 9.7.2017]
[25]Vgl. Tremblay (2004), S. 209-211

Animation[26]

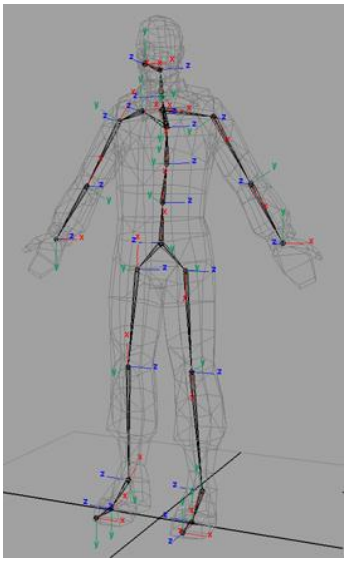

Abb. 24: Skelett eines Modells

In diesem Bereich geht es um das Bewegen von Charakteren und Objekten. Zur Vereinfachung gibt man Modellen ein "Skelett" aus Fixpunkten und Verbindungslinien (siehe Abb. 24). Durch Bewegung eines Fixpunkts lassen sich die Teile des Modells steuern, die mit diesem verbunden sind. Während Charakteranimation in den meisten Spielen in vorherbestimmter Weise abläuft (fixe Animation für Sprünge, Attacken, Laufanimation), verwenden manche moderne Spiele sogenannte *Ragdoll Physics*. Auch hier werden Modellskelette verwendet, diese bewegen sich aber nicht nur in vorbestimmter Weise, sondern wechselwirken mit der Umgebung. Charaktere richten sich zB automatisch auf unebenem Untergrund aus. Im Falle einer Todesanimation werden die Fixpunkte des Skeletts der künstlichen Gravitation ausgesetzt, und das Modell fällt in sich zusammen wie einer Marionette, der man die Schnüre durchgeschnitten hat, daher der Name Ragdoll Physics.

Beschleunigungsverfahren[27]

Um bei Kollisionsberechnungen für jedes Bild nicht ständig jedes einzelne Objekt zu überprüfen, wird die Spielwelt hierarchisch unterteilt. Wird eine Kollisionsberechnung durchgeführt, dann werden nur die Objekte einbezogen, die sich im gleichen Segment wie der Charakter befinden. Zu diesen Verfahren gehören: *Quadtrees*, *Octrees*, *k-D* Bäume und *Binary Space Partitioning* Bäume.

2.2.2 Geometrie

Im Geometrieschritt werden Operationen an den Punkten und Dreiecken durchgeführt, die im Anwendungsschritt berechnet wurden. Dieser Schritt ist weiter unterteilt in die Funktionen: Modell- und Sichtbarkeitstransformation, Eckpunkt-Shading, Projektion, Beschneidung und Bildschirmmapping (siehe Abb. 25). Diese Unterpunkte können eine Pipline bilden, sie können je nach Anwendung aber auch parallel ablaufen. Der Geometrieschritt berechnet was gezeichnet werden soll, wie es gezeichnet werden soll, und wo es gezeichnet werden soll. Zur Vereinfachung werden alle Punkte in der Form $\vec{p} = (x, y, z, 1)$ und Transformationen durch 4x4-Matrizen definiert[28].

[26]Vgl. Watkinson (2009), S. 3-8
[27]Vgl. Tremblay (2004), S. 427
[28]Vgl. Akenine-Möller (2008), S. 15

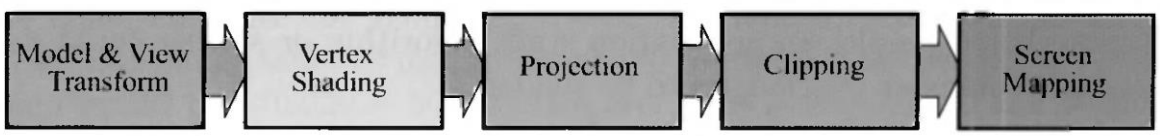

Abb. 25: Operationen im Geometrieschritt

Modell- und Sichtbarkeitstransformation[29]

Auf dem Weg zum Bildschirm wird ein Modell in mehrere verschiedene Räume beziehungsweise Koordinatensysteme transformiert. Ursprünglich existiert ein Modell in seinem eigenen *Object Space*. Dieser ist die Welt aus der Perspektive des Objekts. Der Koordinatenursprung wird innerhalb des Objekts festgelegt. Der Object Space vereinfacht Transformationen, wie beispielsweise eine Rotation des Objekts um sich selbst oder eine beliebige Drehachse. Jedem Modell können mehrere Transformationsmatrizen zugeordnet werden, um es zu positionieren oder zu orientieren. Dadurch kann sich dasselbe Modell an mehreren Orten befinden, unterschiedlich orientiert sein und verschiedene Größen haben, ohne die Originalgeometrie zu verändern. Hierbei werden die Eckpunkte und Normalvektoren des Modells transformiert.

Nach der Transformation befindet sich das Modell im *World Space*. Dieses Koordinatensystem ist einzigartig und nachdem alle Modelle transformiert wurden, befinden sie sich alle in dem selben Raum. Abb.26 zeigt das Modell eines Teekessels, auf den eine Rotationstransformation, gefolgt von einer Translationstransformation angewandt wurde. Transformationen, die wiederum auf den World Space angewandt werden, betreffen alle Objekte darin. Transformationen auf den World Space können eingesetzt werden, um besondere Effekte zu erzielen, beispielsweise einen Welleneffekt für Unterwasserlevel[30].

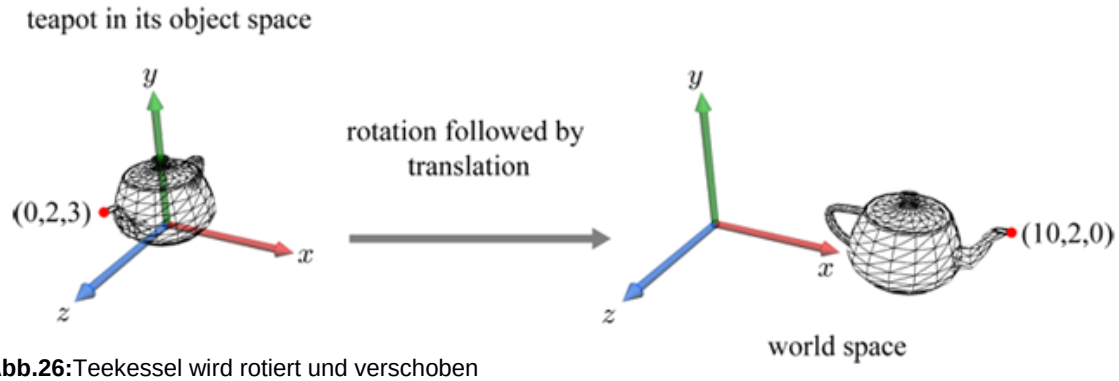

Abb.26: Teekessel wird rotiert und verschoben

Wie schon erwähnt, werden später nur Modelle am Bildschirm gezeichnet, die die Kamera sieht. Die Kamera hat eine Position und eine Ausrichtung im World Space, die genutzt

[29]Vgl. Tremblay (2004), S. 126
[30]Vgl. Ebd., S. 127

werden kann, um die Kamera zu platzieren und zu orientieren. Um die spätere Projektion und Beschneidung zu vereinfachen, wird der World Space so transformiert, dass der neue Koordinatenursprung im Ursprung der Kamera liegt und diese entlang der z-Achse blickt (siehe Abb.27). Dieser neue Raum wird *Camera Space* genannt. Die Koordinaten im Camera Space sind die Koordinaten aus der Sicht der Kamera[31].

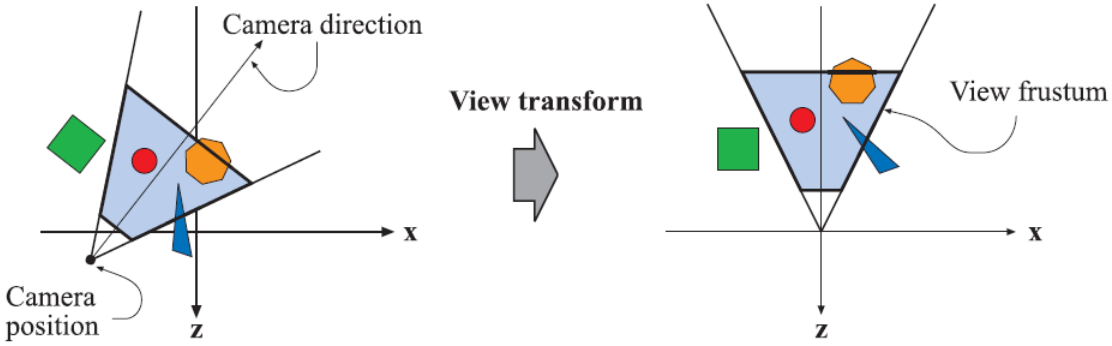

Abb.27: Transformation der Szene in den Camera Space

Eckpunkt-Shading[32]

Um eine realistische Szene zu produzieren, reicht es nicht aus, nur Formen und Position eines Objekts zu zeichnen, auch das Aussehen muss modelliert werden. Diese Beschreibung enthält Materialeigenschaften sowie den Effekt jeder Lichtquelle, die auf das Objekt scheint. Material und Licht können auf viele Arten modelliert werden, von einfacher farblichen Skalierung bis zu aufwendigen physikalischen Verfahren. Beim *Raytracing* werden einzelne Lichtstrahlen und deren Reflexion auf Oberflächen simuliert. Die Operation, die den Lichteffekt eines Materials bestimmt, nennt man *Shading*. Sie involviert die Berechnung der Lichtintensität an mehreren Punkten auf dem Objekt. Üblicherweise werden solche Berechnungen während des Geometrieschritts an den Eckpunkten eines Modells durchgeführt. Die Helligkeitswerte werden dann im Rasterungsschritt zwischen den Punkten über die Dreiecksfläche interpoliert. Weitere Berechnungen können während der Rasterung durchgeführt werden. Eine Vielzahl an Materialdaten kann an jedem Eckpunkt gespeichert werden, so wie die Koordinaten, Normalvektoren, Farben und jeder andere numerische Wert, der nötig ist, um die Schattierung zu berechnen. Um die Berechnungen zu vereinfachen, können Kamera und Lichtquellen in den nötigen Object Space transformiertund später wieder zurücktransformiert werden. Möglich ist das, weil die relativen Beziehungen (Lichtquellen, Kamera und Modell) beibehalten werden, wenn alle Entitäten inklusive Schattierungsberechnung in denselben Raum transformiert werden.

[31]Vgl. Akenine-Möller (2008), S. 16
[32] Vgl. Ebd., S. 17

Projektion[33]

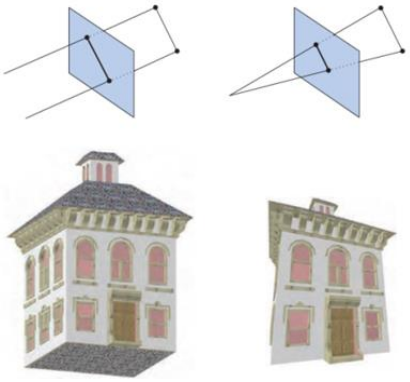

Abb.28: Resultat bei unterschiedlicher Projektion

Nach dem Shading wird die Projektion von 3D auf 2D durchgeführt. Dazu wird das Sichtvolumen in einen Einheitswürfel mit den Extrempunkten (-1, -1, -1) und (1, 1, 1) umgewandelt. Es gibt zwei übliche Projektionsmethoden: orthographische und perspektivische Projektion. Bei der **orthographischen Projektion** ist das Sichtvolumen normalerweise ein Quader. Die Haupteigenschaft dieser Projektion ist es, dass parallele Linien nach der Transformation zum Einheitswürfel parallel bleiben.Die **perspektivische Projektion** imitiert, wie wir Objekte wahrnehmen. Je weiter ein Gegenstand von der Kamera entfernt ist, umso kleiner soll er nach der Projektion erscheinen, außerdem können sich parallele Linien im Horizont treffen (siehe Abb.28).

Zuschneidung[34]

Nur die Elemente, die ganz oder teilweise im Sichtvolumen sind, müssen an den Rasterungsschritt weitergegeben werden, um dann am Bildschirm gezeichnet werden zu können. Ein Element, das sich komplett im Volumen befindet, wird so weitergegeben wie es ist. Elemente außerhalb sind nicht sichtbar und werden daher nicht weitergegeben. Elemente, die nur teilweise im Volumen sind, müssen abgeschnitten werden. Zum Beispiel eine Strecke, die einen Endpunkt im und einen außerhalb des Volumens hat, muss gegen das Volumen geschnitten werden, sodass der Punkt außerhalb durch einen neuen im Schnittpunkt ersetzt wird (siehe Abb.29). Da dieser Schritt nach der Transformierung des Sichtvolumens kommt, müssen Elemente nur gegen den Einheitswürfel geschnitten werden.

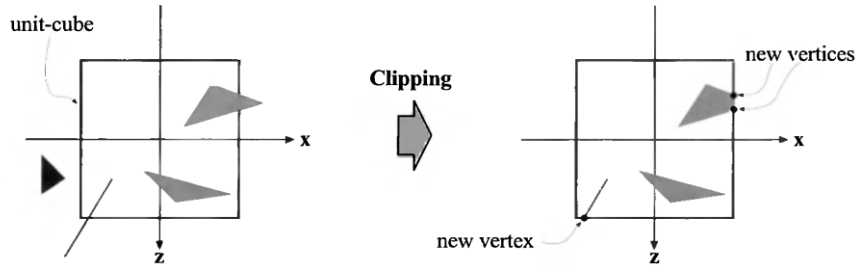

Abb.29: Zuschneidung von Elementen außerhalb des Einheitswürfels

[33]Vgl. Akenine-Möller (2008),S. 18-19
[34]Vgl. Ebd., S. 19

Bildschirm-Mapping[35]

Nur die abgeschnittenen Elemente im Sichtvolumen werden an das Bildschirm-Mapping weitergegeben. Die Koordinaten sind immer noch dreidimensional. Die x- und y-Koordinaten jedes Elements werden zu Bildschirmkoordinaten transformiert. Angenommen die Szene wird in einem Fenster mit den Minimumkoordinaten (x_1, y_1) und den Maximumkoordinaten (x_2, y_2) mit $x_1 < x_2$ und $y_1 < y_2$ gezeichnet: In diesem Fall ist das Bildschirmmapping eine Translation gefolgt von einer Skalierung (siehe Abb.30). Die z-Koordinate wird nicht beeinflusst. Die Bildschirmkoordinaten zusammen mit den z-Koordinaten werden an den Rasterungsschritt weitergegeben.

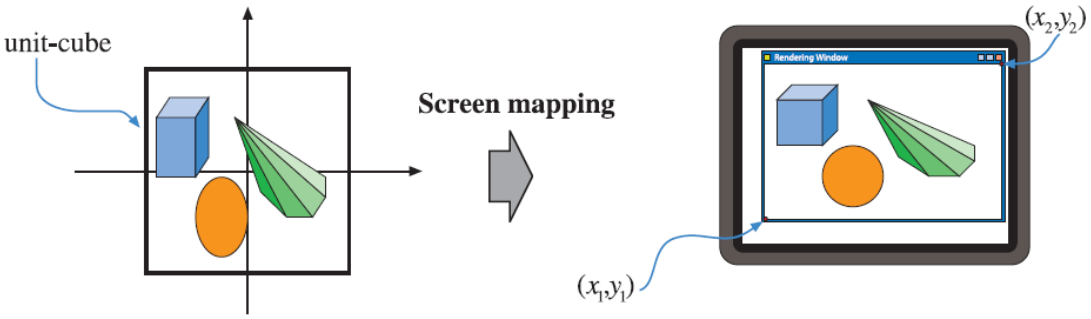

Abb.30: Transformation vom Einheitswürfel zu den Ausmaßen eines Fensters

2.2.3 Rasterung

Im Rasterungsschritt werden aus den projizierten Flächen *Fragmente* erstellt. Jedes Fragment entspricht dabei einem Pixel des Bildschirms. Diese werden dann entsprechend der Textur, Materialdaten und Beleuchtung eingefärbt. Der Rasterungsschritt ist in mehrere funktionelle Schritte zerlegt (siehe Abb.31)[36].

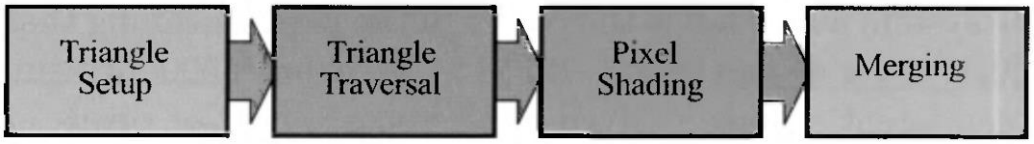

Abb.31: Unterteilung des Rasterungsschritts

Triangle Setup[37]

In diesem Schritt werden die Normalvektoren und andere Daten der Oberflächendreiecke berechnet. Diese Daten werden zur Interpolation verschiedener Schattierungsdaten genutzt, die im Geometrieschritt berechnet wurden.

[35]Vgl. Akenine-Möller (2008),S. 20
[36]Vgl. Ebd.,S. 21
[37]Vgl. Ebd.,S. 22

Triangle Traversal[38]

Hier wird jedes Pixel überprüft, das teilweise verdeckt ist, um ein Fragment für das überlappende Dreieck zu generieren.

Pixel Shading

Hier werden die Berechnungen für jedes Pixel durchgeführt, basierend auf den Schattierungsdaten. Das Ergebnis sind Farben, die an den nächsten Schritt weitergegeben werden. Mehrere Techniken werden in diesem Schritt angewandt, die wichtigste davon ist das *Texturing*. Einfach ausgedrückt wird hier ein Bild auf ein Objekt geklebt. In Abb.32 sieht man links oben das Modell eines Drachen ohne Texturen. Die Bilder rechts werden auf das Modell angebracht und verleihen dem ihm sein Aussehen[39].

Abb.32: Anwendung einer Textur auf ein Modell

Merging[40]

Im letzten Schritt wird die Fragmentfarbe mit den Schattierungsdaten kombiniert. Die Informationen für jedes Pixel wird im Farbpuffer gespeichert (Rot-, Blau- und Grünkomponenten für jede Farbe). Dieser Schritt ist auch für die Sichtbarkeit verantwortlich. Das bedeutet, nachdem die ganze Szene gezeichnet wurde, enthält der Farbpuffer die Farben für die Elemente, die vom Punkt der Kamera aus sichtbar sind. Die Sichtbarkeitsüberprüfung erfolgt durch den *z-Buffer*-Algorithmus. Der z-Buffer speichert für jedes Pixel den z-Wert von der Kamera zur aktuellen Position des aktuell nächsten Elements. Wird ein Element gezeichnet, wird für jedes Pixel des Elements der z-Wert mit dem im Puffer verglichen. Wenn der neue z-Wert kleiner ist, dann ist das gerenderte Element

[38]Vgl. Akenine-Möller (2008),S. 22
[39]Vgl. Ebd., S. 22
[40]Vgl. Ebd., S. 22

näher an der Kamera als das letzte an diesem Pixel und z-Wert und Farbwert werden aktualisiert. Ist der z-Wert größer, bleiben die Werte unberührt.

2.3. Mathematische Grundlagen

Dieses Kapitel behandelt Grundlagen und Methoden, die bei der Darstellung virtueller Objekte auf einem Bildschirm zur Anwendung kommen.

2.3.1 Koordinatensysteme

Jeder (moderne) Bildschirm besteht aus Pixeln, kleinsten Lämpchen, die in verschiedenen Farben leuchten können, um in Summe für unser Auge ein Bild zu erzeugen. Doch wie weiß der Computer, der mit diesem Bildschirm verbunden ist, welches Lämpchen er in einer bestimmten Farbe leuchten lassen soll? Angenommen man will Schach spielen, hat jedoch kein Brett zur Hand. Ohne Felder, an denen man sich orientieren kann, ist es schwer, die Figuren richtig zu bewegen, was schnell zu einem ziemlichen Chaos führt. Um die Position eines Punktes zu beschreiben, benötigt man ein Bezugssystem. Beim Schachbrett wird die Position durch die Nummer der Zeile und den Buchstaben der Spalte beschrieben (siehe Abb.33). Beim Bildschirm sind die Pixel ebenfalls zeilen- und spaltenweise wie in einem Gitter angeordnet, also wird es durch seine Zeilen- und Spaltennummern definiert. Man spricht bei dieser Beschreibung von Koordinaten. In der Mathematik verwendet man das Koordinatensystem mit seinen x- und y-Achsen (und z-Achse) und seinem Ursprung. Ein Punkt hat für jede Achse eine Koordinate, die angibt, wie weit der Punkt in Richtung dieser Achse vom Ursprung entfernt ist (siehe Abb.34). Alle Koordinaten zusammen beschreiben die Position eines Punktes[41].

Abb.33: Schachspiel

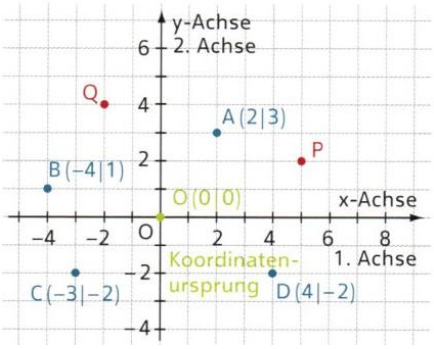

Abb.34: Punkte in einem Koordinatensystem

In besonderen Fällen können auch Kugelkoordinaten verwendet werden. Ein Kugelkoordinatensystem ist festgelegt durch ein Zentrum O, eine *Polachse* durch das Zentrum mit einer *Äquatorebene* orthogonal dazu und einer Bezugsrichtung in der Äquatorebene. Ein Punkt P wird durch die folgenden drei Koordinaten festgelegt: Den Abstand r vom Ursprung, den *Polarwinkel* θ zwischen Polachse und Strecke OP, gezählt von

[41] Vgl: Bleier 5 (2017), S.268

0 bis π und den Horizontalwinkel φ zwischen der Bezugsrichtung und der Orthogonalprojektion der Strecke OP (siehe Abb. 35)[42].

Kugelkoordinaten können mit folgender Formel in kartesische Koordinaten umgewandelt werden:

$$x = r\sin(\theta)\cos(\varphi)$$
$$y = r\sin(\theta)\sin(\varphi)$$
$$z = r\cos(\theta)$$

Alternativ können kartesische Koordinaten mit folgender Formel in Kugelkoordinaten umgewandelt werden[43]:

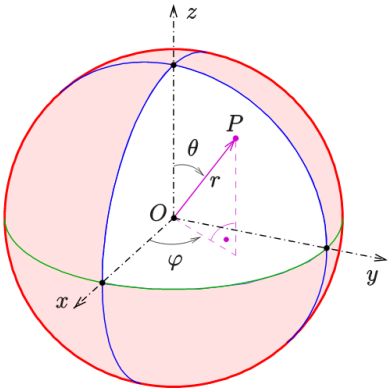

Abb.35: Kugelkoordinatensystem

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = arccos\left(\frac{z}{r}\right)$$
$$\varphi = \begin{cases} arctan\left(\frac{y}{x}\right) & ,für\ x > 0 \\ sign(y)\frac{\pi}{2} & ,für\ x = 0 \\ arctan\left(\frac{y}{x}\right) + \pi & ,für\ x < 0 \wedge y \geq 0 \\ arctan\left(\frac{y}{x}\right) - \pi & ,für\ x < 0 \wedge y < 0 \end{cases}$$

2.3.2 Vektoren

Die Verschiebung von einem Punkt im Koordinatensystem zu einem anderen wird in der Mathematik durch Pfeile beschrieben, die sogenannten *Vektoren*. Ein Vektor ist durch seinen Betrag und seine Richtung gekennzeichnet. Ein Vektor wird durch Koordinaten beschrieben, diese geben jeweils die relative Verschiebung in Richtung der Koordinatenachse an. Der Vektor, der in Abb.36 vom Punkt A zum Punkt B zeigt, hat die Koordinaten (2,6) und wird in Zeilen- oder Spaltenschreibweise angegeben: $\overrightarrow{AB} = (2|6) = \begin{pmatrix} 2 \\ 6 \end{pmatrix}$. Die Koordinaten eines Vektors entsprechen der Differenz von Endpunkt und Anfangspunkt. Für einen Vektor mit dem Anfangspunkt A und einem Endpunkt B gilt: $\overrightarrow{AB} = B - A$. Vektoren können in ihren Koordinaten übereinstimmen, auch wenn sie an verschiedenen Orten platziert werden. "*Die Menge aller Pfeile der Zeichenebene, die gleich lang, parallel und gleich orientiert sind, wird*

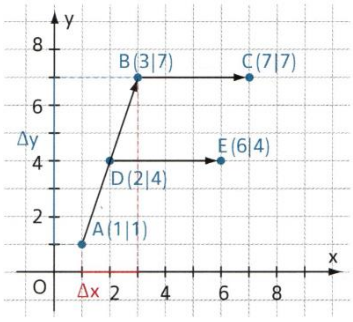

Abb.36: Vektoren

[42]Vgl. Ebd., S. 13
[43] Vgl. mo.mathematik.uni-stuttgart.de

als Vektor bezeichnet" [44] . Im Allgemeinen werden Vektoren durch Kleinbuchstaben beschrieben, zum Beispiel: $\vec{a}$. Der Ort eines Punktes P kann durch den Vektor $\vec{p} = \overrightarrow{OP}$ dargestellt werden, wobei *O* der Koordinatenursprung ist. Die Länge eines Vektors kann im kartesischen Koordinatensystem aus dem pythagoreischen Lehrsatz berechnet werden.[45]

Die Addition von zwei Vektoren entspricht der Hintereinanderausführung der zugehörigen Verschiebungen. "*Vektoren werden addiert, indem komponentenweise die Summe gebildet wird. Vektoren werden subtrahiert, indem komponentenweise die Differenz gebildet wird"*[46]. Geometrisch werden zwei Vektoren addiert, indem man sie so durch Pfeile darstellt, dass der Startpunkt des zweiten mit dem Endpunkt des ersten Pfeils übereinstimmt. Die Vektorsumme wird als Pfeil vom Startpunkt des ersten bis zum Endpunkt des zweiten Pfeils dargestellt. Die Vektorsubtraktion wird geometrisch als Addition mit dem Gegenvektor dargestellt (siehe Abb.37). Vektoren können mit reellen Zahlen multipliziert werden. "*Ein Vektor wird mit einer reellen Zahl multipliziert, indem jede Komponente des Vektors mit der Zahl multipliziert wird. Das Ergebnis ist ein Vielfaches des ursprünglichen Vektors"*[47].[48]

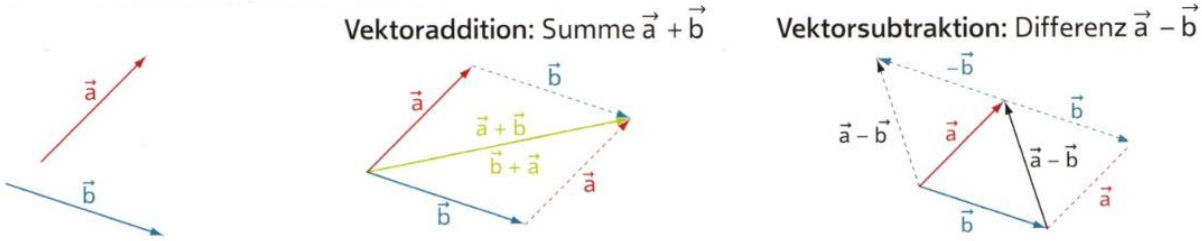

Abb.37: Rechenoperationen von Vektoren

Ein Vektor der Länge 1 heißt Einheitsvektor oder normiert. Man normiert einen von 0 verschiedenen Vektor, indem man ihn durch seine Länge dividiert.[49]

Das **Skalarprodukt** $\vec{a} \bullet \vec{b} = \begin{pmatrix} x_a \\ y_a \end{pmatrix} \bullet \begin{pmatrix} x_b \\ y_b \end{pmatrix} = x_a \cdot x_b + y_a \cdot y_b$ ordnet zwei Vektoren einen Skalar zu. Für das Skalarprodukt ergibt sich aus dem Kosinussatz und dem Satz des Pythagoras die Formel: $\vec{a} \bullet \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos\varphi$. Der Winkel φ wird von den Vektoren $\vec{a}$ und $\vec{b}$ eingeschlossen (siehe Abb.100). Das Skalarprodukt ist also der Kosinus von φ multipliziert mit den Längen der Vektoren. Umgeformt erhält man $\cos\varphi = \frac{\vec{a} \bullet \vec{b}}{|\vec{a}| \cdot |\vec{b}|}$. Sind beide Vektoren Einheitsvektoren, so ist das Skalarprodukt der

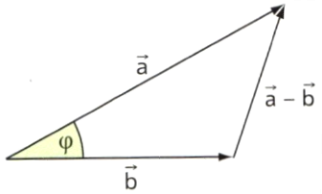

Abb.38: Winkel zwischen zwei Vektoren

[44] Zitat: Bleier 5 (2017), S.271
[45] Vgl: Bleier 5 (2017), S.269ff
[46] Zitat: Bleier 5 (2017), S.276
[47] Zitat: Bleier 5 (2017), S.278
[48] Vgl: Bleier 5 (2017), S.276ff
[49] Vgl: Bleier 5 (2017), S.287ff

Kosinus des eingeschlossenen Winkels. Mit dem Skalarprodukt lässt sich somit der Winkel zwischen zwei Vektoren bestimmen, aber auch mit $\cos\varphi$ alleine können Aussagen über die Lage der Vektoren getroffen werden. Stehen die zwei Vektoren normal aufeinander, so ist das Skalarprodukt null, da $\cos\varphi = \cos 90° = 0$ gilt. Schließen $\vec{a}$ und $\vec{b}$ einen spitzen Winkel ein, dann ist das Skalarprodukt positiv. Schließen $\vec{a}$ und $\vec{b}$ einen stumpfen Winkel ein, dann ist das Skalarprodukt negativ. Aus der Formel $\vec{a} \bullet \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos\varphi$ lässt sich $|\vec{b}| \cdot \cos\varphi = b_a$ herausheben. Die Größe b_a entspricht dabei der orientierten Projektion von $\vec{b}$ auf $\vec{a}$, welche man sich als Schatten von $\vec{b}$ vorstellen kann (siehe Abb.39). Die Projektion von $\vec{a}$ auf $\vec{b}$ funktioniert analog mit $|\vec{a}| \cdot \cos\varphi = a_b$. *"Für zwei Vektoren $\vec{a}$ und $\vec{b}$, die vom Nullvektor verschieden sind, gilt: Das* ***Skalarprodukt*** *der beiden Vektoren entspricht dem* ***Produkt aus der Länge des Vektors a und dem Wert der Normalprojektion b_a:*** *$\vec{a} \bullet \vec{b} = |\vec{a}| \cdot b_a = |\vec{b}| \cdot a_b$."*[50],[51]

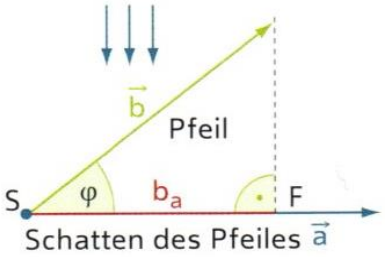

Abb.39: Normalprojektion

"Steht ein Vektor $\boldsymbol{n}$ *zu einem gegebenen Vektor* $\boldsymbol{a}$ *normal, so wird* $\boldsymbol{n}$ *ein* ***Normalvektor*** *von $\vec{a}$ genannt"*[52]*."Zu jedem Vektor $\vec{a} = \begin{pmatrix} x_a \\ y_a \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ gibt es im $\mathbb{R}^2$ genau zwei Normalvektoren mit derselben Länge wie $\vec{a}$:* $\begin{pmatrix} -y_a \\ x_a \end{pmatrix}$ *und* $\begin{pmatrix} y_a \\ -x_a \end{pmatrix}$*. Die beiden Normalvektoren sind entgegengesetzt orientiert. Alle anderen Normalvektoren sind zu diesen parallel und unterscheiden sich durch die Länge"*[53](siehe Abb.40).

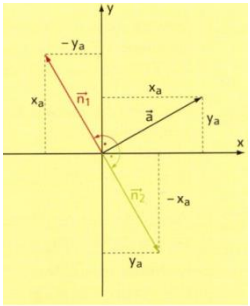

Abb.40: Normalvektoren

"Unter dem Vektorprodukt $\vec{a} \times \vec{b}$ ***(****sprich a kreuz b) zweier Vektoren $\vec{a}$ und $\vec{b}$ versteht man den* ***Vektor*** *$\vec{c}$ mit folgenden Eigenschaften:*
1) $\vec{c} = \vec{a} \times \vec{b}$ steht normal sowohl zu $\vec{a}$ als auch zu $\vec{b}$.
2) $|\vec{c}| = |\vec{a} \times \vec{b}|$ gibt den Flächeninhalt des von den Vektoren $\vec{a}$ und $\vec{b}$ aufgespannten Parallelogramms an.
3) Die Vektoren $\vec{a}$, $\vec{b}$, $\vec{c}$ bilden in dieser Reihenfolge ein ***rechtshändiges System****."*[54] (siehe Abb.41)

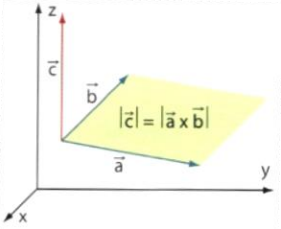

Abb.41: Kreuzprodukt

[50] Zitat: Bleier 5 (2017), S.331
[51] Vgl: Bleier 5 (2017), S.291ff
[52] Zitat: Bleier 5 (2017), S S.297
[53] Zitat: Bleier 5 (2017), S S.299
[54] Zitat: Bleier 6 (2017), S.244

2.3.3 Matrizen[55]

Eine Matrix ist eine rechteckige Anordnung von Elementen. Eine Matrix mit m Zeilen und n Spalten heißt **Matrix vom Typ (m,n)** oder **(m × n)-Matrix**. Eine Matrix mit gleicher Anzahl m von Zeilen und Spalten heißt **quadratische** Matrix der **Ordnung m**. Ein Beispiel für eine quadratische Matrix $\boldsymbol{M}$ der Ordnung 3 ist:

$$\boldsymbol{M} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

*"Eine quadratische Matrix, deren sämtliche Elemente außer der Hauptdiagonalen (Diagonale von links oben nach rechts unten) gleich 0 sind, heißt **Diagonalmatrix**. Eine Diagonalmatrix, deren sämtliche Hauptdiagonalelemente gleich 1 sind, heißt **Einheitsmatrix**. Sie wird mit dem Buchstaben $\boldsymbol{E}$ bezeichnet."*[56]

*"Zwei Matrizen $\boldsymbol{A}$, $\boldsymbol{B}$ vom gleichen Typ werden **addiert** bzw. **subtrahiert**, indem man einander entsprechende Elemente addiert bzw. subtrahiert. Man schreibt: $\boldsymbol{A} + \boldsymbol{B}$ bzw. $\boldsymbol{A} - \boldsymbol{B}$. Eine Matrix $\boldsymbol{A}$ wird **mit einer Zahl** $k \in \mathbb{R}$ **multipliziert**, indem man jedes Element von $\boldsymbol{A}$ mit k multipliziert. Man schreibt: $k \cdot \boldsymbol{A}$"*[57]

Zwei Matrizen $\boldsymbol{A}$ und $\boldsymbol{B}$ können nur dann multipliziert werden, wenn die Anzahl der Spalten von $\boldsymbol{A}$ gleich der Anzahl der Zeilen von $\boldsymbol{B}$ ist. Für das folgende Beispiel ist $\boldsymbol{A}$ eine (3x4)-Matrix und $\boldsymbol{B}$ eine (4x1)-Matrix. Das Ergebnis ist eine Matrix vom Typ (3x1):

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}, \qquad \boldsymbol{B} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$$\boldsymbol{A} \cdot \boldsymbol{B} = \begin{bmatrix} a_{11} * b_1 + a_{12} * b_2 + a_{13} * b_3 + a_{14} * b_4 \\ a_{21} * b_1 + a_{22} * b_2 + a_{23} * b_3 + a_{24} * b_4 \\ a_{31} * b_1 + a_{32} * b_2 + a_{33} * b_3 + a_{34} * b_4 \end{bmatrix}$$

Merkregel für die Matrizenmultiplikation $\boldsymbol{A} \cdot \boldsymbol{B}$: $(m, p) \cdot (p, n) = (m, n)$

*"Ist mit $\boldsymbol{A} \cdot \boldsymbol{B}$ auch das Produkt $\boldsymbol{B} \cdot \boldsymbol{A}$ möglich, so ist im Allgemeinen $\boldsymbol{A} \cdot \boldsymbol{B} \neq \boldsymbol{B} \cdot \boldsymbol{A}$, das heißt das Kommutativgesetz ist nicht erfüllt. Gibt es zu einer Matrix $\boldsymbol{A}$ eine Matrix $\boldsymbol{A^{-1}}$, so dass gilt: $\boldsymbol{A} \cdot \boldsymbol{A^{-1}} = \boldsymbol{A^{-1}} \cdot \boldsymbol{A} = \boldsymbol{E}$, so heißt $\boldsymbol{A^{-1}}$ **inverse Matrix** von $\boldsymbol{A}$."*[58]

[55] Vgl. Timischl (2007), S. 331ff
[56] Zitat: Timischl (2007), S.332
[57] Zitat: Timischl (2007), S.335
[58] Zitat: Timischl (2007), S.337

"Lineare Gleichungssysteme können übersichtlich in der Matrixschreibweise formuliert werden:

zB: Gegeben ist ein lineares Gleichungssystem:

$$I\colon\ 2x + y - z = 2$$
$$II\colon\ x + 3y = 5$$
$$III\colon y + 2z = 7$$

Das Gleichungssystem besitzt die Koeffizientenmatrix $\boldsymbol{A} = \begin{bmatrix} 2 & 1 & -1 \\ 1 & 3 & 0 \\ 0 & 1 & 2 \end{bmatrix}$*. Setzt man noch als Lösungsvektor* $\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ *und als "rechte Seite" des Gleichungssystem* $\vec{b} = \begin{pmatrix} 2 \\ 5 \\ 7 \end{pmatrix}$*, so bestätigt man durch Ausmultiplizieren sofort, dass das Gleichungssystem in der Form* $\begin{bmatrix} 2 & 1 & -1 \\ 1 & 3 & 0 \\ 0 & 1 & 2 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \\ 7 \end{pmatrix}$ *oder kurz* $\boldsymbol{A} \cdot \vec{x} = \vec{b}$ *geschrieben werden kann."*[59]

2.3.4 Geraden und Strecken[60]

"*Durchläuft* t *in* $g\colon X = P + t \cdot \vec{r}$ *alle Zahlen aus* $\mathbb{R}$*, so kann jeder Punkt der Geraden* g *beschrieben werden. Die Vektorgleichung* $g\colon X = P + t \cdot \vec{r}$ *heißt* ***Parameterform einer Geraden g.***"[61] Eine begrenzte Strecke s zwischen den Punkten A und B kann auf folgende Art dargestellt werden:

$$s = \{A + t \cdot (B - A) | t \epsilon [0,1]\}$$

2.3.5 Ebenen

"Eine Ebene ε *im Raum ist durch drei Punkte P, Q und R eindeutig festgelegt, wenn diese drei Punkte nicht auf einer Geraden liegen. Mit diesen Punkten können zwei nicht parallele Richtungsvektoren bestimmt werden. Mit ihrer Hilfe kann die Ebenengleichung* $\varepsilon\colon X = P + u \cdot \overrightarrow{r_1} + v \cdot \overrightarrow{r_2}$ *aufgestellt werden"*[62](siehe Abb.42). "*Jede lineare Gleichung mit drei Variablen x, y und z der Form* $a \cdot x + b \cdot y + c \cdot z = d$ *mit* $(a|b|c) \neq (0|0|0)$ *und* $a, b, c, d \epsilon \mathbb{R}$ *legt genau eine Ebene fest. Diese Darstellungsform heißt* ***allgemeine Ebenengleichung****. Umgekehrt ist jede Ebene durch*

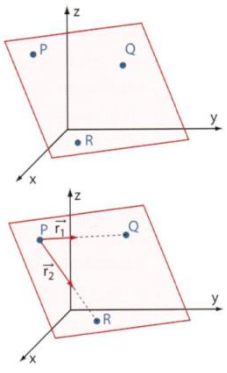

Abb.42: Parameterform

[59] Zitat: Timischl (2007), S.341
[60] Vgl: Bleier 5 (2017), S.311
[61] Zitat: Bleier 5 (2017), S S.312
[62] Zitat: Bleier 6 (2017), S.260

eine lineare Gleichung mit drei Variablen festgelegt."[63]

„Ein Normalvektor $\vec{n}$ einer Ebene steht auf alle Richtungsvektoren dieser Ebene normal. Ist neben dem Normalvektor noch ein Punkt P bekannt, so ist die Ebene eindeutig festgelegt (siehe Abb.43)*. Die Vektorgleichung $\varepsilon: \vec{n} \cdot X = \vec{n} \cdot P$ wird als Normalvektorgleichung einer Ebene im Raum bezeichnet."*[64] *„Die Koeffizienten a, b und c der linearen Glieder der allgemeinen Ebenengleichung $a \cdot x + b \cdot y + c \cdot z = d$ sind die Koordinaten eines Normalvektors der Ebene."*[65] [66]

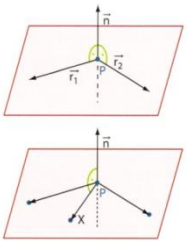

Abb.43: Normal-vektorform

2.3.6 Abstände[67]

Um den Abstand von einem Punkt P zu einer Geraden g im Raum zu berechnen, wird eine Ebene durch den Punkt gelegt, die normal zur Geraden ausgerichtet ist. Der Richtungsvektor der Geraden ist ein Normalvektor der Ebene. Die Gerade schneidet die Ebene im Punkt S. Die Länge des Vektors $\overrightarrow{SP}$ ist der kürzeste Abstand von der Geraden zum Punkt P. Alternativ lässt sich der Abstand über das Kreuzprodukt ermitteln. Sei A ein beliebiger Punkt auf der Geraden g. Der Betrag des Kreuzprodukts des Vektors $\overrightarrow{AP}$ mit dem normierten Richtungsvektor $\overrightarrow{g_0}$ der Geraden ist die Fläche des von ihnen aufgespannten Parallelogramms. Die Strecke PS mit der Länge d und der Einheitsvektor $\overrightarrow{g_0}$ mit der Länge 1 spannen ein Rechteck mit dem gleichen Flächeninhalt auf (siehe Abb.44). Somit gilt für den Normalabstand d: $\left|\overrightarrow{AP} \times \overrightarrow{g_0}\right| = d \cdot 1$. Der Abstand zweier paralleler Geraden g und h ist der Abstand von einem beliebigen Punkt von g zu h (und umgekehrt).

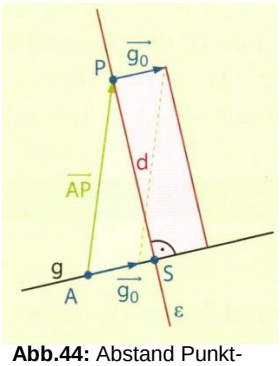

Abb.44: Abstand Punkt-Gerade

Bei windschiefen Geraden verläuft der kürzeste Abstand normal zu beiden Geraden, also entlang des Kreuzprodukts beider Richtungsvektoren. Wir bestimmen einen Vektor wird durch einen Punkt G der Geraden g und einen Punkt H der Geraden h. Die Länge der Projektion des Vektors $\overrightarrow{GH}$ auf das normierte Kreuzprodukt der Richtungsvektoren $\vec{g} \times \vec{h}$ ist der kürzeste Abstand beider Geraden $d(g,h) = \overrightarrow{GH} \cdot \left(\vec{g} \times \vec{h}\right)_0$ (siehe Abb.45).

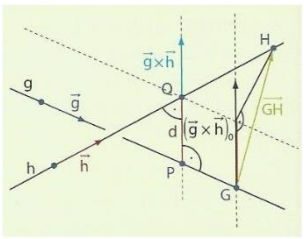

Abb.45: Abstand Gerade-Gerade

[63] Zitat: Bleier 6 (2017), S.262
[64] Zitat: Bleier 6 (2017), S.266
[65] Zitat: Bleier 6 (2017), S.267
[66] Vgl: Bleier 6 (2017), S.266ff
[67] Vgl: Bleier 6 (2017), S.285ff

Um den Abstand von einem Punkt P zu einer Ebene ε zu bestimmen, wird von einem beliebigen Punkt A der Ebene ein Vektor $\overrightarrow{AP}$ zum Punkt festgelegt und auf den normierten Normalvektor $\overrightarrow{n_0}$ der Ebene projiziert. Der Betrag des projizierten Vektors ist der kürzeste Abstand $d(P,\varepsilon) = \left|\overrightarrow{AP} \cdot \overrightarrow{n_0}\right|$ (siehe Abb.46). Der Abstand von einer Ebene zu einer parallelen Geraden ist der Abstand der Ebene zu einem Punkt der Geraden. Der Abstand von einer Ebene zu einer parallelen Ebene ist der Abstand der Ebene zu einem Punkt der anderen Ebene.

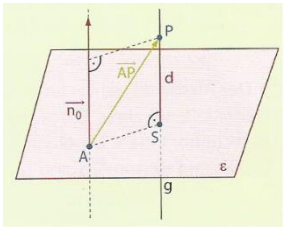

Abb.46: Abstand Punkt-Ebene

Eine weitere Methode, um den Abstand von einem Punkt zu einer Ebene zu berechnen verwendet die hessesche Normalform. In der hesseschen Normalform wird eine Ebene durch einen normierten Normalvektor $\overrightarrow{n_0}$ der Ebene sowie ihren Abstand $d \geq 0$ vom Koordinatenursprung beschrieben. Eine Ebene besteht aus den Punkten, deren Ortsvektoren $\vec{p}$ die Ebenengleichung $\vec{p} \cdot \overrightarrow{n_0} = d$ erfüllen. Mit Hilfe der hesseschen Normalform kann der Abstand eines beliebigen Punktes Q zur Ebene ε berechnet werden, indem sein Ortsvektor $\vec{q}$ in die Ebenengleichung eingesetzt wird: $d(Q,\varepsilon) = \vec{q} \cdot \overrightarrow{n_0} - d$. Der Abstand ist vorzeichenbehaftet. Für $d(Q,\varepsilon) > 0$ liegt der Punkt Q auf der Seite der Ebene, in die der Normalvektor zeigt, ansonsten liegt er auf der anderen Seite.

2.4 Physikalische Grundlagen

Die Physik versucht, die Welt mit Modellen und Mathematik zu beschreiben. Auch wenn diese Beschreibungen nicht zu 100% korrekt sind, genügen sie doch, um Phänomene in Videospielen realistisch erscheinen zu lassen (siehe Abb.47). Was Videospiele besonders macht ist, dass Prozesse in der Spielwelt automatisch ablaufen sollen. Anstatt manuell vorzuschreiben, zu welcher Zeit ein laufender Charakter an welchem Ort sein soll, kann man Bewegungsgleichungen verwenden. Damit ein Charakter nicht Objekte durchdringt, ist es möglich, entweder einen Grenzbereich, den er nicht überschreiten darf, um diese herum zu definieren, oder man bedient sich der Kollisionsberechnung. Um nicht für jeden möglichen Zusammenstoß von zwei Objekten eine Reaktion erfinden zu müssen, kann über die Impulserhaltung eine realistische Reaktion ermittelt werden. Dieses Kapitel beinhaltet mathematische Beschreibungen physikalischer Grundlagen, die verwendet werden, um Spielwelten "realistisch" darzustellen[68].


Abb.47: Brückenbau in *Poly Bridge*

[68]Vgl. Bourg (2013), S. xiii

2.4.1 Geschwindigkeit und Beschleunigung[69]

In einem Raum ohne Reibung und Schwerkraft könnte man einen Körper anstoßen, und er würde sich mit gleichbleibender Geschwindigkeit weiterbewegen. Man spricht von einer gleichförmigen Bewegung. Die Geschwindigkeit lässt sich als Vektor angeben, mit einem Betrag und einer Richtung. Der Betrag des Geschwindigkeitsvektors gibt das Tempo an. Eine Geschwindigkeit ist das Verhältnis von zurückgelegtem Weg zu benötigter Zeit. Mathematisch kann man schreiben:

$$\vec{v} = \Delta\vec{s}/\Delta t$$

wobei $\vec{v}$ für die Geschwindigkeit steht und $\Delta\vec{s}$ der Weg ist, der in der Zeit Δt zurückgelegt wird. Für eindimensionale Bewegungen genügt die Beschreibung durch Skalare:

$$v = \Delta s/\Delta t$$

Dies sei an einem Beispiel veranschaulicht (Abb.48): Ein Auto fährt entlang einer geraden Straße und passiert zur Zeit t_1 den Marker 1 und zur Zeit t_2 den Marker 2. Die gemessene Zeit ist $t_1 = 0s$ und $t_2 = 1{,}136s$. Der Abstand zwischen beiden Markern ist $30m$. Das Auto hat also den Weg $\Delta s = 30m$ zurückgelegt und hat dafür die Zeit $\Delta t = t_2 - t_1 = 1{,}136s$ benötigt. Die Geschwindigkeit des Autos ist somit:

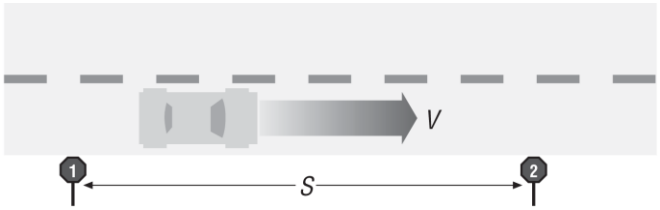

Abb.48: Beispiel Geschwindigkeit eines Autos

$$v = \Delta s/\Delta t = \frac{30m}{1{,}136s} = 26{,}4m/s$$

was ca. 95km/h entspricht. Das war ein einfaches eindimensionales Beispiel, zudem handelt es sich bei der berechneten Geschwindigkeit um die *Durchschnittsgeschwindigkeit*. Wir wissen nicht, ob das Auto zwischen Marker 1 und 2 beschleunigt beziehungsweise abgebremst hat. Um die Bewegung detailreicher beschreiben zu können, benötigt man die *Momentangeschwindigkeit*, also die genaue Geschwindigkeit zu einem bestimmten Zeitpunkt. Es sind also sehr kleine Δt zu betrachten. Die Momentangeschwindigkeit ist der Grenzwert, wenn das Zeitintervall Δt gegen 0 geht, Δt also infinitesimal klein wird. Geschrieben wird dieser Zusammenhang wie folgt:

$$\vec{v} = \lim_{\Delta t \to 0} (\Delta\vec{s}/\Delta t)$$

Die Geschwindigkeit ist somit die Ableitung des Ortes in Abhängigkeit der Zeit:

[69]Vgl. Ebd., S. 36-40

$$\vec{v} = d\vec{s}/dt$$

Diese Beziehung kann man integrieren:

$$d\vec{s} = \vec{v}\, dt$$

$$\overrightarrow{s_2} - \overrightarrow{s_1} = \Delta\vec{s} = \int_{t_1}^{t_2} \vec{v}\, dt$$

Somit lässt sich aus der Geschwindigkeit auf den Weg schließen und umgekehrt.

Im Falle einer konstanten Geschwindigkeit lassen sich die Funktionen für Weg und Geschwindigkeit vereinfachen. Sei $\overrightarrow{v_0}$ die Anfangsgeschwindigkeit und $\overrightarrow{s_0}$ der Anfangsort. Wenn $t_1 = 0$, dann gilt:

$$\vec{v}(t) = \overrightarrow{v_0}$$

$$\vec{s}(t) = \int_0^t \vec{v}(t)dt = \overrightarrow{v_0}t + \overrightarrow{s_0}$$

Eine weitere wichtige kinematische Größe ist die Beschleunigung. Sie beschreibt die Änderungsrate der Geschwindigkeit Δv über die Zeit Δt:

$$\vec{a} = \Delta\vec{v}/\Delta t$$

Der Grenzwert, wenn Δt gegen 0 geht, ist die Momentanbeschleunigung:

$$\vec{a} = \lim_{\Delta t \to 0} (\Delta\vec{v}/\Delta t)$$

$$\vec{a} = d\vec{v}/dt$$

Somit ist die Beschleunigung die Ableitung der Geschwindigkeit in Abhängigkeit der Zeit.

Durch umformen und integrieren erhält man:

$$d\vec{v} = \vec{a}\, dt$$

$$\overrightarrow{v_2} - \overrightarrow{v_1} = \Delta\vec{v} = \int_{t_1}^{t_2} \vec{a}\, dt$$

Dieser Zusammenhang zeigt, dass auch zwischen Beschleunigung und Geschwindigkeit hin- und zurückgerechnet werden kann.

Analog zur konstanten Geschwindigkeit lassen sich die Bewegungsgleichungen auch für eine Bewegung mit konstanter Beschleunigung vereinfachen. Sei $\overrightarrow{a_0}$ die Beschleunigung,$\overrightarrow{v_0}$ die Anfangsgeschwindigkeit und $\overrightarrow{s_0}$ der Anfangsort. Wenn t_1 gleich 0 ist kann man schreiben:

$$\vec{a}(t) = \overrightarrow{a_0}$$

$$\vec{v}(t) = \int_0^t \vec{a}(t)dt = \overrightarrow{v_0} + \overrightarrow{a_0}t$$

$$\vec{s}(t) = \int_0^t \vec{v}(t)dt = \overrightarrow{s_0} + \overrightarrow{v_0}t + \frac{\overrightarrow{a_0}t^2}{2}$$

2.4.2 Zusammengesetzte Bewegung[70]

Will man eine dreidimensionale Bewegung durch die Bewegungsgleichungen beschreiben, kann für jede Raumrichtung die Funktion für die entsprechende Koordinate aufgestellt werden:

$$x(t) = x_0 + v_{0,x}t + \frac{a_{0,x}t^2}{2}$$

$$y(t) = y_0 + v_{0,y}t + \frac{a_{0,y}t^2}{2}$$

$$z(t) = z_0 + v_{0,z}t + \frac{a_{0,z}t^2}{2}$$

Der Ortsvektor $\vec{p}(t)$ eines Punktes enthält nun diese drei Komponenten. Im Kontext der Spielprogrammierung ist es praktischer und platzsparender, dies in Matrixschreibweise anzugeben:

$$\vec{p}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} = \begin{bmatrix} a_{0,x} & v_{0,x} & x_0 \\ a_{0,y} & v_{0,y} & y_0 \\ a_{0,z} & v_{0,z} & z_0 \end{bmatrix} \begin{bmatrix} t^2/2 \\ t \\ 1 \end{bmatrix}$$

Je nach Bedarf können Werte für Ort, Geschwindigkeit und Beschleunigung verändert und gegebenenfalls auch auf null gesetzt werden. Im folgenden Beispiel wirkt die Gravitation in Richtung der negativen y-Achse.

Im Falle des waagrechten Wurfs lässt sich die Gesamtbewegung des Objekts zerlegen in gleichförmige Bewegungen in die waagrechte x- und z-Richtung und eine gleichmäßig beschleunigte Bewegung (freier Fall) in y-Richtung (siehe Abb.49). Man erhält die Bewegungsgleichung für jede Raumkomponente zur Zeit t:

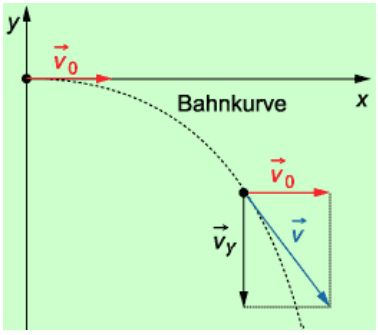

Abb.49: Waagrechter Wurf

[70]Vgl. Bourg (2013), S. 45-48

$$x(t) = x_0 + v_{0,x}t \qquad v_x(t) = v_{0,x}$$

$$y(t) = y_0 + \frac{a_{0,y}t^2}{2} \qquad v_y(t) = a_{0,y}t$$

$$z(t) = z_0 + v_{0,z}t \qquad v_z(t) = v_{0,z}$$

Die Gravitation bewirkt eine Beschleunigung in der y-Richtung. Da die Bewegung nach unten und somit entgegen der positiven y-Richtung verläuft, ist die Beschleunigung negativ. Alternativ könnte man die Orientierung nach unten als positive y-Richtung festlegen. In Matrixschreibweise erhält man für den Ortsvektor $\vec{p}(t)$:

$$\vec{p}(t) = \begin{bmatrix} 0 & v_{0,x} & x_0 \\ a_{0,y} & 0 & y_0 \\ 0 & v_{0,z} & z_0 \end{bmatrix} \begin{bmatrix} t^2/2 \\ t \\ 1 \end{bmatrix}$$

Für den schrägen Wurf ist nur eine zusätzliche Ausgangsgeschwindigkeit nach oben festzulegen:

$$\vec{p}(t) = \begin{bmatrix} 0 & v_{0,x} & x_0 \\ a_{0,y} & v_{0,y} & y_0 \\ 0 & v_{0,z} & z_0 \end{bmatrix} \begin{bmatrix} t^2/2 \\ t \\ 1 \end{bmatrix}$$

2.4.3 Kräfte [71]

Um einen Körper in Bewegung zu setzen, muss eine Kraft auf ihn ausgeübt werden. Die Kraft ist die Ursache einer Bewegungsänderung. Wirkt auf einen Körper zu einem gegebenen Zeitpunkt eine Kraft, so wird er beschleunigt. Diese Beschleunigung zu diesem Zeitpunkt ist proportional zur Kraft. Der Proportionalitätsfaktor ist der Kehrwert seiner Masse. Auf eine Krafteinwirkung $\vec{F}$ reagiert der Körper mit der Beschleunigung

$$\vec{a} = \frac{\vec{F}}{m}$$

Ist die Krafteinwirkung bekannt, so lässt sich daraus etwas über die Bewegung erschließen.

[71] Vgl. Embacher (2010), S. 10

2.4.4 Reibung[72,73]

Ohne Reibung könnte nichts mehr anhalten. Reibung tritt auf, wenn ein Objekt über ein Material gleitet. Je rauer das Material, umso schwerer wird es für das Objekt zu gleiten. Anders ist es, wenn man ein Objekt über Eis gleiten lässt. Reibung hängt von der Oberflächenbeschaffenheit der Objekte ab, die in Kontakt stehen. Die Reibung ist auch abhängig von der Masse des Objekts, der Fallbeschleunigung und der Neigung der Oberfläche (siehe Abb.50). Folgende Formel beschreibt die Reibungskraft für einen Reibungskoeffizienten μ:

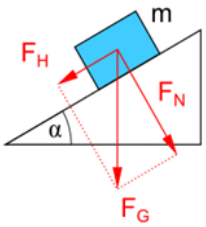

Abb.50: Reibungskraft auf der schiefen Ebene

$$F_R = \mu m g \cdot \cos\alpha$$

Die Reibungskraft wirkt immer entgegen der Bewegungsrichtung. Unterschieden wird zwischen Haftreibung, Gleitreibung und Rollreibung. Haftreibung tritt auf, wenn die Zugkraft auf den Körper kleiner ist als die maximale Haftreibungskraft. Wird die maximale Haftreibungskraft überschritten, so setzt sich der Körper in Bewegung und es kommt zur Gleitreibung. Rollreibung tritt auf, wenn ein Objekt auf einem anderen abrollt. Für jede Reibungsart und Oberflächenbeschaffenheit existiert ein eigener Reibungskoeffizient.

2.4.5 Kraftfelder[74,75]

Das beste Beispiel für Kraftfelder ist die gegenseitige Anziehung von Objekten in Folge der Gravitation. Das Newtonsche Gravitationsgesetz besagt, dass die Anziehungskraft von zwei Massenpunkten direkt proportional zum Produkt ihrer Massen und umgekehrt proportional zum Quadrat ihres Abstandes ist. Zudem wirkt diese Kraft entlang ihrer Verbindungslinie. Der Betrag der Kraft zwischen zwei Massepunkten m_1 und m_2 im Abstand r ist :

$$F = G\frac{m_1 m_2}{r^2}$$

wobei G die Newtonsche Gravitationskonstante ist. Für Spiele kann diese Formel angepasst werden, um besondere Effekte zu erzielen. Ein negatives Vorzeichen für G bewirkt eine abstoßende Kraft (siehe Abb.51). Eine Veränderung der Gravitationskonstante beeinflusst die Stärke der Kraft. Aus der Kraft lässt sich für ein Objekt, das in ein Kraftfeld eindringen will, eine Beschleunigung berechnen, die es abbremst und abstößt.

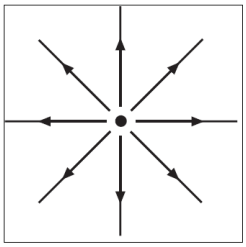
Abb.51: Wirkungslinien eines abstoßenden Kraftfeldes

[72]Vgl. Tremblay (2004),S. 218
[73]Vgl. Bourg (2013), S. 73-74
[74]Vgl. Tremblay (2004), S. 222,
[75]Vgl. Bourg (2013), S. 72

2.4.6 Gedämpfte Bewegung[76,77]

Bei einer Bewegung in einem Medium entstehen Reibungskräfte, die der Bewegung entgegen wirken. Zwei Modelle sind hier von Interesse:

- Ein Körper, der sich in einer zähen Flüssigkeit bewegt, erfährt eine Reibungskraft, die proportional zu seiner Geschwindigkeit ist. Für eine eindimensionale Bewegung wird die Reibungskraft durch einen Ansatz der Form

  $$F(v) = -\alpha v$$

  beschrieben, wobei $\alpha > 0$ die Dämpfungskonstante ist. Das Minuszeichen drückt aus, dass die Reibung der Bewegung entgegen wirkt.

- Der Luftwiderstand ist vom Quadrat der Geschwindigkeit abhängig. Für eine eindimensionale Bewegung wird die Reibungskraft durch einen Ansatz der Form

  $$F(v) = -\beta v|v|$$

  beschrieben, wobei $\beta > 0$ eine Konstante ist, die von der Dichte der Luft und der Geometrie des Körpers abhängig ist.

2.4.7 Druck[78]

Druck ist definiert als Kraft pro Fläche, auf die sie wirkt:

$$\vec{P} = \vec{F}/A$$

Der Druck wirkt immer normal auf die Oberfläche eines Körpers. In Flüssigkeiten ist der Betrag des Drucks gegeben als:

$$P = \rho \cdot g \cdot h$$

wobei ρ die Dichte, g die Fallbeschleunigung und h die Tiefe der Flüssigkeit sind. In Spielen wird der Druck eingesetzt, um Auftrieb zu simulieren.

2.4.8 Auftrieb[79]

Auf einen Körper in einem Fluid (Flüssigkeit oder Gas) wirkt eine Kraft, die sich aus Druckunterschieden über und unter dem Objekt ergeben. Der Druck in einem Fluid steigt mit der Tiefe, somit ist der Druck am Boden eines Objekts größer als auf der Deckfläche.

[76]Vgl. Embacher (2010), S. 17,
[77]Vgl. Bourg (2013), S.89
[78]Vgl. Bourg (2013), S. 76
[79]Vgl. Ebd., S. 77-78

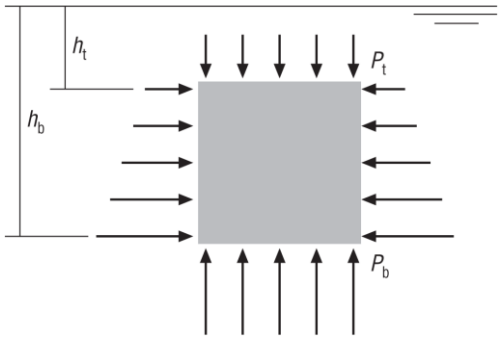

Abb.52: Versenktes Objekt

Am Deckel wirkt der Druck $P_t = \rho g h_t$ nach unten und am Boden wirkt $P_b = \rho g h_b$ nach oben. Man sieht, dass sich die seitlichen Druckkräfte aufheben, dagegen sind die von unten wirkenden Druckkräfte größer als die von oben wirkenden Druckkräfte. (siehe Abb.52).

Die Kräfte, die oben und unten auf das Objekt wirken, entsprechen dem jeweiligen Druck mal der Fläche:

$$F_t = P_t A = \rho g h_t A$$

$$F_b = P_b A = \rho g h_b A$$

Das Ergebnis ihrer Differenz ist die Auftriebskraft:

$$F_A = F_b - F_t = \rho g h_b A - \rho g h_t A = \rho g A (h_b - h_t) = \rho g V$$

Das Produkt aus Dichte des Fluides und des Volumens ergibt die Masse der verdrängten Flüssigkeit. Die Multiplikation mit g ergibt schließlich die Gewichtskraft der verdrängten Flüssigkeit.

2.4.9 Impuls[80,81]

Ein Körper, der sich mit einer Geschwindigkeit $\vec{v}$ fortbewegt, hat einen Impuls $\vec{p}$:

$$\vec{p} = m \cdot \vec{v}$$

Um einen Körper aus der Ruhe auf die Geschwindigkeit $\vec{v}$ zu beschleunigen, muss ein Kraftstoß $\vec{F} \cdot \Delta t$ wirken. Dieser bewirkt eine Impulsänderung:

$$\vec{F} \cdot \Delta t = m \cdot \Delta \vec{v} = \Delta \vec{p}$$

Der Impulserhaltungssatz besagt, dass der Gesamtimpuls in einem abgeschlossenen System vor und nach dem Stoß erhalten bleibt. Für zwei Körper mit den Massen m_1 und m_2

[80]Vgl. Bourg (2013), S. 104-107
[81]Vgl. Tremblay (2004),S. 229-233

und den Geschwindigkeiten $v_{1,vor}$, $v_{2,vor}$, $v_{1,nach}$, $v_{2,nach}$ vor und nach einem eindimensionalen Stoß gilt:

$$m_1 v_{1,vor} + m_2 v_{2,vor} = m_1 v_{1,nach} + m_2 v_{2,nach}$$

In Videospielen ist jedes Objekt ein starrer Körper, das heißt er behält seine Form. In der Realität verformen sich Objekte bei einer Kollision. Dabei wird kinetische Energie in Verformungsenergie umgewandelt. Kinetische Energie ist die Energie, die nötig ist, um einen Körper auf eine bestimmte Geschwindigkeit $\vec{v}$ zu beschleunigen. Die Formel lautet:

$$E_{kin} = \frac{m\vec{v}^2}{2}$$

Stöße, bei denen ein Teil der kinetischen Energie umgewandelt wird, heißen *unelastisch*. Wirft man zwei Kugeln aus weichem Ton gegeneinander, trägt ein Teil der kinetischen Energie zur Verformung bei, und die anschließende Bewegung ist alles andere als spektakulär. Beim idealen unelastischen Stoß bewegen sich beide Kugeln nach dem Stoß zusammen mit der gleichen Geschwindigkeit weiter. Der Impulserhaltungssatz ergibt:

$$m_1 v_{1,vor} + m_2 v_{2,vor} = (m_1 + m_2) v_{nach}$$

Formt man nach v_{nach} um, erhält man das Ergebnis:

$$v_{nach} = \frac{m_1 v_{1,vor} + m_2 v_{2,vor}}{m_1 + m_2}$$

Stöße, bei denen die kinetische Energie erhalten bleibt, heißen *elastisch*. Für die Gesamtenergie vor und nach einem elastischen Stoß gilt:

$$m_1 {v_{1,vor}}^2 + m_2 {v_{2,vor}}^2 = m_1 {v_{1,nach}}^2 + m_2 {v_{2,nach}}^2$$

Aus den Gleichungen für die Impulserhaltung und die Energieerhaltung können die Geschwindigkeiten nach einer elastischen Kollision ermittelt werden:

$$v_{1,nach} = 2\frac{m_1 v_{1,vor} + m_2 v_{2,vor}}{m_1 + m_2} - v_{1,vor}$$

$$v_{2,nach} = 2\frac{m_1 v_{1,vor} + m_2 v_{2,vor}}{m_1 + m_2} - v_{2,vor}$$

In der Realität befinden sich Stöße irgendwo zwischen ideal elastisch und ideal unelastisch. Der reale Stoß wird durch den *Resolutionskoeffizienten* k charakterisiert:

$$k = \frac{v_{2,nach} - v_{1,nach}}{v_{1,vor} - v_{2,vor}}$$

Für einen ideal unelastischen Stoß ist $k = 0$, für einen ideal elastischen Stoß ist $k = 1$. Für einen realen Stoß mit dem Resolutionskoeffizienten k ergeben sich die folgenden Geschwindigkeiten:

$$v_{1,nach} = \frac{m_1 v_{1,vor} + m_2 v_{2,vor} - m_2 (v_{1,vor} - v_{2,vor}) k}{m_1 + m_2}$$

$$v_{2,nach} = \frac{m_1 v_{1,vor} + m_2 v_{2,vor} - m_1 (v_{2,vor} - v_{1,vor}) k}{m_1 + m_2}$$

2.4.10 Drehbewegung[82,83]

Will man die Rotation eines Körpers in einer Ebene beschreiben, kann man angeben, um welchen Winkel φ er sich aus der Ausgangslage verdreht hat (siehe Abb.53). Während der Körper rotiert, ändert sich φ, und die zugehörige Änderungsrate ist die Winkelgeschwindigkeit ω:

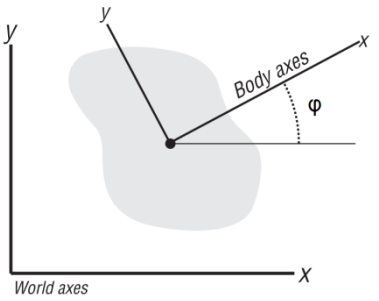

Abb.53: Rotation um φ

$$\omega = \frac{d\varphi}{dt}$$

Die Änderungsrate der Winkelgeschwindigkeit ist die Winkelbeschleunigung α:

$$\alpha = \frac{d\omega}{dt}$$

Analog zur linearen Bewegung lassen sich auch hier Bewegungsgleichungen aufstellen:

$$\alpha(t) = \alpha_0$$

$$\omega(t) = \omega_0 + \alpha_0 t$$

$$\varphi(t) = \varphi_0 + \omega_0 t + \frac{\alpha_0 t^2}{2}$$

Ein Punkt, der um eine Achse rotiert, legt einen Weg zurück, der vom Abstand zur Drehachse und vom überstrichenen Drehwinkel abhängig ist (siehe Abb. 54). Die Formel, mit der die Bogenlänge c berechnet werden kann, lautet:

$$c = r\varphi$$

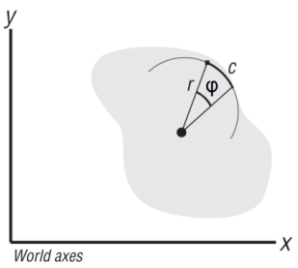

Abb.54: Kreisbewegung von Punkten eines starren Körpers

[82]Vgl. Tremblay (2004),S. 237
[83]Vgl. Bourg (2013), S. 62

wobei φ im Bogenmaß angegeben sein muss. Leitet man beide Seiten der Gleichung nach der Zeit ab,

$$dc/dt = r\, d\varphi/dt$$

erhält man die Umrechnung von linearer in zirkulare Geschwindigkeit.

$$v = r\,\omega$$

Analog lässt sich auch die Umwandlung von linearer Beschleunigung in Winkelbeschleunigung herleiten:

$$a = r\,\alpha$$

2.4.11 Schwingungen[84]

Eine Schwingung stellt eine oszillierende Bewegung dar, bei der eine Masse ausgelenkt wird, zurückkehrt, in die Gegenrichtung ausschlägt und wieder zurückkommt. Eine Beschreibungsart von Schwingungen ist die Pendelschwingung. Sie wird am einfachsten durch ein Fadenpendel beschrieben. Dabei hängt eine Punktmasse an einer masselosen Schnur der Länge L (siehe Abb.55). Für kleine Auslenkungen gilt die Formel:

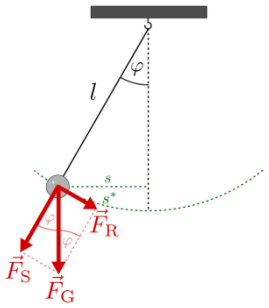

Abb.55: Kräftediagramm am Fadenpendel

$$\varphi(t) = A\sin(\omega_0 t + \varphi_0)$$

Hierbei bezeichnet A die Amplitude und φ_0 die Auslenkung zur Zeit $t = 0$. Die Eigenkreisfrequenz ω_0 und die Periodendauer T_0 einer Pendelschwingung mit kleiner Amplitude sind gegeben durch:

$$\omega_0 = \sqrt{\frac{g}{L}}$$

$$T_0 = 2\pi\sqrt{\frac{L}{g}}$$

2.4.12 Federn[85]

Federn sind Elemente, die, wenn sie mit zwei Objekten verbunden sind, gleich große aber entgegengesetzte Kräfte auf jedes Objekt ausüben. Die Federkraft F_F ist dabei abhängig davon, wie weit die Feder gestreckt beziehungsweise gestaucht wird und von der Federkonstante k_F:

[84]Vgl. Tremblay (2004),S. 242
[85]Vgl. Bourg (2013), S. 79

$$F_F = k_F(L - r)$$

L ist die Länge der gestreckten beziehungsweise gestauchten Feder und r ist die Länge der Feder in Ruhelage.

In Kombination mit Federn werden oft Stoßdämpfer verwendet. Diese verhalten sich wie die Dämpfung einer Flüssigkeit, indem sie der Geschwindigkeit entgegen wirken. Ein Stoßdämpfer, der zwei Objekte verbindet, die sich voneinander wegbewegen, verlangsamt deren relative Geschwindigkeit zueinander. Die Dämpfungskraft F_D ist proportional zur relativen Geschwindigkeit und dem Dämpfungsfaktor k_D:

$$F_D = k_D(v_1 - v_2)$$

Üblicherweise werden Federn und Stoßdämpfer zu einem einzigen Feder-Dämpfungs-Element kombiniert. In der Vektorschreibweise wird die Kraft eines Feder-Dämpfungs-Elements durch folgende Formel beschrieben:

$$\vec{F_1} = -\left\{k_F(L - r) + \frac{k_D(\overrightarrow{v_1} - \overrightarrow{v_2}) \bullet \vec{L}}{\|\vec{L}\|}\right\}\frac{\vec{L}}{\|\vec{L}\|}$$

Der Vektor $\vec{L}$ reicht von der Position des ersten Körpers zur Position des zweiten. Die Kraft $\overrightarrow{F_1}$ wird auf den ersten Körper ausgeübt, während für die Kraft $\overrightarrow{F_2}$ auf den anderen Körper gilt:

$$\overrightarrow{F_2} = -\overrightarrow{F_1}$$

Federn und Stoßdämpfer eignen sich dafür, Ansammlungen verbundener Partikel oder starrer Körper zu simulieren. Die Federn halten die Elemente zusammen, während die Stoßdämpfer die Bewegungen glätten. Um Objekte zu verbinden, muss man nur aus ihren Positionen die Längen der benötigten Federn berechnen. Die Kraft wird dann auf die verbundenen Elemente angewandt. Für starre Körper ist es etwas komplizierter, da der genaue Ort am Körper definiert werden muss, an dem die Feder angreifen soll. Zudem ist für jeden Körper ein Drehmoment zu berechnen.

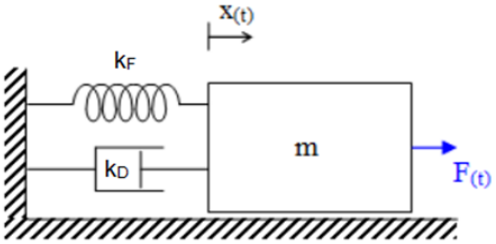

Abb.56: Feder-Dämpfung Kombination

2.4.13 Seile[86]

Seile sind verformbar. Sie sind elastisch und lassen sich zu einem gewissen Grad dehnen. Dieses Verhalten lässt sich durch Partikel simulieren, die durch Federn verbunden sind. Das Seil in Abb.57 besteht aus 10 Partikeln, die durch 9 Federn verbunden sind. Zum Start der Simulation war das Seil horizontal nach rechts ausgerichtet. Infolge der Schwerkraft fällt es nach unten, schwingt hin und her und kommt irgendwann, gerade nach unten hängend, zur Ruhe.

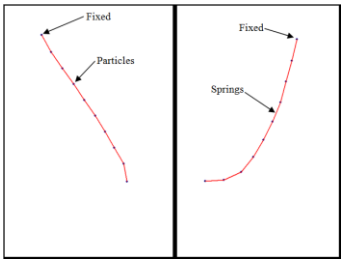

Abb.57: Schwingendes Seil

2.4.14 Tücher[87]

Tücher sind die zweidimensionale Variante von Seilen. Das Modell einer Flagge in Abb.58 ist eine Ansammlung von Partikeln, die in einem Gittermuster positioniert und durch Federn verbunden werden. Jede Linie im Drahtgerüst der Flagge entspricht einem Feder-Dämpfungs-Element, während sich in den Eckpunkten die Partikel befinden. Die horizontalen und vertikalen Federn geben der Flagge Struktur, während die diagonalen Federn Scherkräfte abschwächen und die Flagge weiter stärken sollen. Ein Tuch kann eingesetzt werden, um die Haut eines Charakters zu simulieren. Man platziert sie auf einem Gitter und fixiert ein paar Punkte. Werden diese Fixpunkte bewegt, folgen ihnen die benachbarten Partikel aufgrund der Federn. Auf diese Art kann man Gesichter beispielsweise reden, schmollen oder lachen lassen. Da ein Tuch dehnbar ist, wird sich auch die Haut dehnbar verhalten. Tücher können auch genutzt werden, um Kleidung zu simulieren.

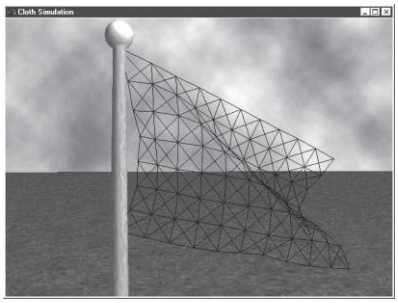

Abb.58: Netz aus Partikeln und Federn

Erweitert man das Gitter noch in die dritte Dimension, lassen sich auch elastische Körper simulieren. Charaktermodelle können auf Strichmännchen reduziert werden, deren Fixpunkte durch Federn verbunden sind und der restliche Körper durch Polygone an diesen Fixpunkten befestigt wird (siehe *Ragdoll* Abb.24).

Bei einem Seil oder Tuch alleine würde nicht viel passieren. Interessanter wird es erst, wenn eine Kraft auf die Partikel ausgeübt wird, wie es bei dem Seil infolge der Schwerkraft der Fall war. Eine weitere interessante

Abb.59: Wellenbewegung in *Sea of Thievs (2018)*

[86]Vgl. Bourg (2013), S. 259
[87]Vgl. Bourg (2013), S. 256

Kraft ist Wind. Betrachtet man die Wasseroberfläche als Tuch, kann die Windkraft angewendet werden, um diese zu bewegen und so Wellenbewegungen zu simulieren. (siehe Abb.59).

2.4.15 Massenmittelpunkt[88]

Der Massenmittelpunkt $\vec{c}$ eines Körpers ist das mit der Masse gewichtete Mittel der Positionen seiner Massepunkte. Mit dem Massenmittelpunkt werden komplexe starre Körper auf einen einzigen Massepunkt reduziert. Allgemein gilt für ein System von n Massepunkten an den Orten $\vec{x}_i$:

$$\vec{c} = \frac{\sum_{i=1}^{n} m_i \vec{x}_i}{\sum_{i=1}^{n} m_i}$$

2.4.16 Drehmoment[89]

Das Drehmoment beschreibt die Drehwirkung einer Kraft auf einen Körper. Es kann die Rotation eines Körpers beschleunigen oder bremsen. Der Vektor des Drehmoments $\vec{M}$ ist das Kreuzprodukt aus Ortsvektor und Kraftvektor (siehe Abb.60):

$$\vec{M} = \vec{F} \times \vec{r}$$

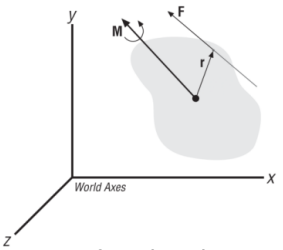

Abb.60: Kraft und Drehmoment

Dabei ist $\vec{r}$ der Ortsvektor vom Bezugspunkt des Drehmoments zum Angriffspunkt der Kraft. Der Bezugspunkt ist frei wählbar. Je nach Orientierung des Drehsinns unterscheidet man linksdrehende und rechtsdrehende Drehmomente.

[88]Vgl. Tremblay (2004),S. 257
[89]Vgl. Ebd., S. 260

3. Einsatz in der Grafikpipeline

Dieses Kapitel beschäftigt sich mit dem Einsatz mathematischer Methoden, um Spielwelten dynamisch und realistisch erscheinen zu lassen.

3.1 Schneiden

Um virtuelle Objekte miteinander interagieren zu lassen, muss getestet werden, ob sie dazu in der Lage sind. Dazu ist zu überprüfen, ob die geometrischen Elemente, aus denen jedes Objekt besteht, einander schneiden. Dadurch können neben der Kollision verschiedene Effekte erzielt werden, beispielsweise das Hinterlassen von Fußspuren, oder Reaktionen von Feinden, wenn man ihnen zu nahe kommt. Im Folgenden werden Methoden beschrieben, mit denen Schnitte verschiedener Elemente getestet werden können.

3.1.1 Schneiden von zwei Ebenen[90]

Je zwei Ebenen nehmen im Raum eine der drei Lagen *schneidend, identisch* oder *parallel* zueinander ein. Die Lagebeziehung von zwei Ebenen kann mit ihren Normalvektoren überprüft werden. Wir bilden also für Ebene E_1 den Normalvektor $\overrightarrow{n_1}$ und für Ebene E_2 den Normalvektor $\overrightarrow{n_2}$**.** Sind die Ebenen parallel oder identisch, dann zeigen ihre Normalvektoren entweder in die gleiche oder in die entgegengesetzte Richtung. In beiden Fällen sind die Normalvektoren parallel zueinander, und das Kreuzprodukt der Normalvektoren ist der Nullvektor. Sind sie nicht parallel, dann kann man die Schnittgerade suchen. Da sie in beiden Ebenen verläuft, muss sie normal auf beide Normalvektoren stehen (siehe Abb.61). Zur Ermittlung des Richtungsvektors der Schnittgerade kann man das Kreuzprodukt der Normalvektoren bilden.

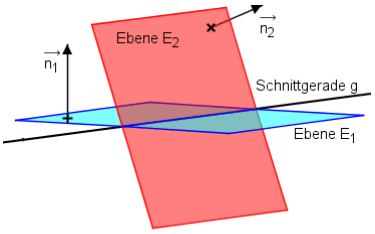

Abb.61: Schnitt zweier Ebenen

$$\vec{r} = \overrightarrow{n_1} \times \overrightarrow{n_2}$$

Anschließend benötigt man einen Punkt P**,** der in der Gerade liegt. Da der Punkt ein Element beider Ebenen ist, muss er für beide Ebenen die Normalvektorgleichungen erfüllen:

$$\overrightarrow{n_1} \bullet P = d_1$$

$$\overrightarrow{n_2} \bullet P = d_2$$

[90]Vgl. Tremblay (2004), S. 104

Viele Punkte erfüllen diese Bedingung. Zur Ermittlung der Koordinaten von P kann man eine davon 0 setzen und nach den anderen auflösen. Das Ergebnis ist die Schnittgerade der Form: $g: X = \boldsymbol{P} + t \cdot \vec{r}$

3.1.2 Schnitt von Ebene und Gerade[91]

Gegeben sei eine Ebene $\vec{n} \cdot \vec{x} + D = 0$ und eine Gerade $g: \vec{x} = \vec{p} + t\vec{v}$**.** Der Schnittpunkt $\vec{x}$ ergibt sich, wenn man $\vec{x} = \vec{p} + t\vec{v}$ in die Ebenengleichung einsetzt, nach t umformt und in die Geradengleichung rückeinsetzt (siehe Abb.62):

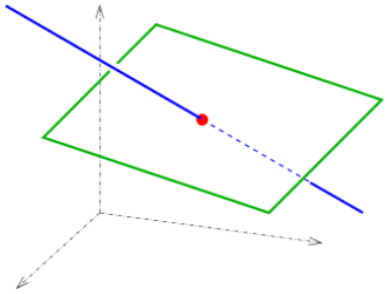

Abb.62: Schnitt von Ebene und Gerade

$$\vec{n} \cdot (\vec{p} + t\vec{v}) + D = 0$$

$$t = -\frac{(\vec{n} \cdot \vec{p} + D)}{\vec{n} \cdot \vec{v}}$$

$$\vec{x} = \vec{p} - \frac{(\vec{n} \cdot \vec{p} + D)}{\vec{n} \cdot \vec{v}} \vec{v}$$

3.1.3 Schnitt von Dreieck und Strecke[92]

Während manche Situationen in Spielen mit endlosen Geraden arbeiten (siehe 3.4.3 Auswahl), betreffen die meisten Schnittprobleme endliche Strecken. Der Schnitt mit einem Dreieck ist von Interesse, da die Oberfläche virtueller Objekte aus Dreiecken modelliert wird (siehe Abb.63).

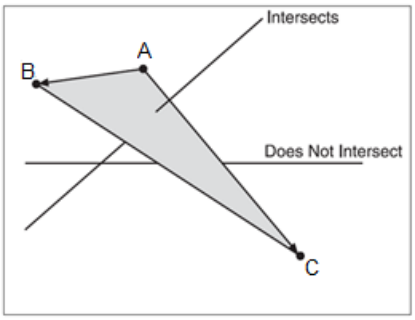

Abb.63: Schnitt und kein Schnitt zwischen Strecke und Dreieck

Zuerst ist festzustellen, ob die Strecke lang genug ist, um durch das Dreieck zu gehen. Zur leichteren Berechnung betrachten wir die Ebene ε, die durch das Dreieck aufgespannt wird. Die Strecke hat die Endpunkte S_1 und S_2. Wenn die Strecke lang genug ist, um die Ebene zu durchdringen, dann muss auf einer Seite der Ebene der Punkt S_1 und auf der gegenüberliegenden Seite der Punkt S_2 liegen. Zur Überprüfung, ob sie auf unterschiedlichen Seiten liegen, berechnen wir die Abstände der Punkte zur Ebene mit der Abstandsformel $d(P, \varepsilon) = \left|\overrightarrow{AP} \cdot \overrightarrow{n_0}\right|$ aus Kapitel 2.6.3, nur lassen wir für unsere Zwecke den Betrag weg. Wir wählen einen Punkt der Ebene, beispielsweise den Eckpunkt A des Dreiecks, und berechnen die vorzeichenbehafteten Abstände zu den Punkten S_1 und S_2:

$$d(S_1, \varepsilon) = \overrightarrow{AS_1} \cdot \overrightarrow{n_0}$$

[91]Vgl. Tremblay (2004), S. 108
[92]Vgl. Ebd., S. 110-114

$$d(S_2, \varepsilon) = \overrightarrow{AS_2} \cdot \overrightarrow{n_0}$$

Betrachten wir die Abstandsberechnung zum Punkt S_1. Das Skalarprodukt $\overrightarrow{AS_1} \cdot \overrightarrow{n_0}$ ist die Projektion des Vektors $\overrightarrow{AS_1}$ auf den normierten Normalvektor $\overrightarrow{n_0}$ der Ebene. Schließen beide Vektoren einen stumpfen Winkel ein, dann ist das Skalarprodukt negativ, bei einem spitzen Winkel ist es positiv. Liegen die Punkte S_1 und S_2 auf unterschiedlichen Seiten der Ebene, und sei zu Demonstrationszwecken der Winkel zwischen $\overrightarrow{AS_1}$ und $\overrightarrow{n_0}$ spitz, dann ist der Winkel zwischen $\overrightarrow{AS_2}$ und $\overrightarrow{n_0}$ stumpf (siehe Abb.64). Liegen die Endpunkte der Strecke also auf unterschiedlichen Seiten der Ebene, dann haben die Skalarprdoukte $\overrightarrow{AS_1} \cdot \overrightarrow{n_0}$ und $\overrightarrow{AS_2} \cdot \overrightarrow{n_0}$ unterschiedliche Vorzeichen, und die Strecke ist lang genug. Haben beide Skalarprodukte das gleiche Vorzeichen, dann liegen beide Endpunkte der Strecke auf der selben Seite der Ebene.

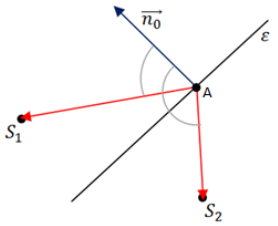

Abb.64: Lage zweier Punkte

Als nächstes ist zu überprüfen, ob die Strecke innerhalb der Grenzen des Dreiecks liegt. Dazu muss man zuerst den Schnittpunkt mit dem Dreieck berechnen. Dieser befindet sich auf der Ebene, die durch das Dreieck definiert wird. Wird nun die Welt so rotiert, dass die Dreiecksebene parallel zu der Ebene ist, die von zwei Koordinatenachsen aufgespannt wird, so wird die dritte Koordinatenachse überflüssig, da alle Punkte der Ebene in dieser neuen Welt den gleichen Wert für die dritte Komponente haben werden. Dargestellt wird der Prozess in Abb.65.

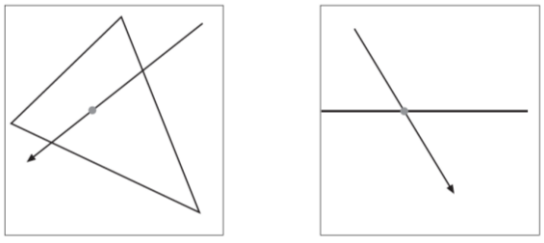
Abb.65: Schnitt vor und nach Rotation

Nachdem der Schnittpunkt von Gerade und Ebene berechnet wurde, ist zu testen, ob dieser im Dreieck liegt. Auch hier wird ausgenutzt, dass das Skalarprodukt von Vektoren mit spitzen Winkeln positiv und mit stumpfen Winkeln negativ ist. Der Vorgang wird in Abb.66 illustriert:

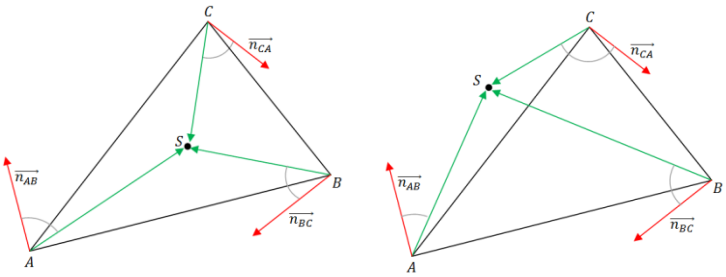

Abb.66:Überprüfung der Lage des Schnittpunkts

Wir müssen nun für jede Seite des Dreiecks ABC überprüfen, ob der Schnittpunkt S auf der Innenseite liegt. Nur wenn das für jede Seite der Fall ist, liegt der Schnittpunkt im Dreieck. Betrachten wir den Fall der Dreieckseite $\overline{AB}$. Ohne Beschränkung der Allgemeinheit konstruieren wir einen Vektor $\overrightarrow{AS}$ vom Eckpunkt A zum Schnittpunkt S, sowie einen Normalvektor $\overrightarrow{n_{AB}}$ auf den Vektor $\overrightarrow{AB}$, der in die Richtung des gegenüberliegenden Eckpunktes C zeigt. Ist das Skalarprodukt $\overrightarrow{AS} \cdot \overrightarrow{n_{AB}}$ positiv, dann schließen beide Vektoren einen spitzen Winkel ein und der Schnittpunkt liegt auf der Innenseite von $\overline{AB}$. Dieser Vorgang wird analog für die Dreieckseiten $\overline{BC}$ und $\overline{CA}$ durchgeführt. Ist jedes der Skalarprodukte $\overrightarrow{AS} \cdot \overrightarrow{n_{AB}}$, $\overrightarrow{BS} \cdot \overrightarrow{n_{BC}}$ und $\overrightarrow{CS} \cdot \overrightarrow{n_{CA}}$ positiv, dann liegt der Schnittpunkt S im Dreieck (siehe Abb.61 links). Unterscheiden sich die Vorzeichen der Skalarprodukte, dann liegt der Schnittpunkt außerhalb des Dreiecks (siehe Abb.61 rechts).

3.1.4 Schnitt von Quader und Strecke[93]

Der Einfachheit halber kann eine Gruppe von Objekten in einem Quader zusammengefasst werden (Hitbox Abb.22). Aus diesem Grund ist es wichtig, den Schnitt einer Strecke mit einem Quader zu behandeln. Wir gehen davon aus, dass der Quader entlang der Koordinatenachsen ausgerichtet ist. Der Quader erstreckt sich in der x-Achse im Intervall $[a, b]$, in der y-Achse im Intervall $[c, d]$ und in der z-Achse im Intervall $[e, f]$. Zuerst testen wir, ob die Strecke lang genug ist, um durch eine der Ebenen zu gehen. Der Test funktioniert analog zum **Schnitt von Dreieck mit Strecke** und wird für alle sechs Begrenzungsflächen des Quaders durchgeführt. Geht die Strecke durch eine oder mehrere Ebenen, wird der Schnittpunkt mit den betroffenen Ebenen berechnet. Zuletzt muss feststellt werden, ob der Schnittpunkt in dem durch den Quader definierten Bereich $[a, b]$, $[c, d]$ und $[e, f]$ liegt.

[93]Vgl. Tremblay (2004), S. 115

3.2 Kollision

Abb.67: Superman gleitet durch Boden

Mit den Methoden aus Kapitel 3.1 kann man feststellen, ob Objekte kollidieren und wo sie kollidieren. Das Spiel ist aber ständig in Bewegung, und Kollisionsabfragen passieren nur Bild für Bild. Feuert man eine Rakete gegen eine Wand, kann es passieren, dass sich die Rakete in einem Frame vor der Wand und im nächsten dahinter befindet. Die Rakete hätte explodieren sollen, aber da nur vorher und nachher auf Kollision getestet wurde, wurde keine festgestellt. In Spielen kann es dann vorkommen, dass Figuren durch Wände gleiten (siehe Abb.67). In komplexeren Fällen muss Kollision für mehrere bewegliche Objekte festgestellt werden[94].

3.2.1 Umweltkollision[95]

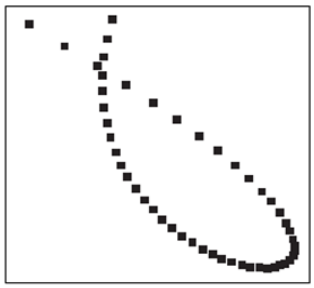

Abb.68: Angenäherte Bahnkurve

Zur Vereinfachung werden die Kollisionen auf Ebene-Objekt-Kollisionen reduziert. Bewegungen können komplex ausfallen, zum Beispiel ein Tuch, das im Wind flattert. Kollision kann für solche Bewegungen schwer ermittelt werden. Anstatt der wahren Bewegung kann eine Annäherung der Bewegung betrachtet werden, indem man die lineare Bahn des Objekts zwischen zwei Bildern betrachtet. Anders ausgedrückt wird die Bewegung von Bild $n-1$ zu Bild n (für $t \in [0,1]$) als gerade Linie beschrieben. Jede Bewegung kann somit als Aneinanderreihung geradliniger Bewegungen betrachtet werden (siehe Abb.68).

Kugel-Ebenen Kollision[96]

Für dieses Problem verwenden wir die hessesche Normalform, um den Abstand d von einem Punkt $\vec{p}$ zu einer Ebene zu berechnen:

$$d = \vec{p} \cdot \overrightarrow{n_0} - D$$

Der Punkt $\vec{p}$ ist der Mittelpunkt einer Kugel mit dem Radius r. Der Abstand von der Ebene zur Kugeloberfläche ist somit $D - r$ (siehe Abb.69). Der Abstand $D - r$ entspricht einer Verschiebung der Ebene um $-r$. Man kann das Problem also

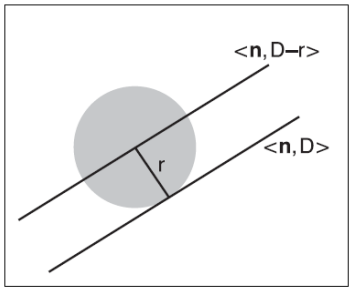

Abb.69: Ebenen-Kugel Kollision

[94]Vgl. Ebd., S. 263
[95]Vgl. Tremblay (2004), S. 264
[96]Vgl. Ebd., S. 265

von einer beweglichen Kugel auf einen beweglichen Punkt reduzieren. Der Punkt bewegt sich entlang einer geraden Linie vom Punkt $\overrightarrow{p_0}$ zum Punkt $\overrightarrow{p_1}$**.** Seine Position $\vec{p}(t)$ ist von der Zeit abhängig. Die Funktion für die Position des Kugelmittelpunktes lautet:

$$\vec{p}(t) = \overrightarrow{p_0} + t(\overrightarrow{p_0} - \overrightarrow{p_1})$$

Werden beide Gleichungen kombiniert, erhält man die Funktion für die Entfernung zwischen einem Punkt und der Ebene in Abhängigkeit der Zeit:

$$d(t) = \left(\overrightarrow{p_0} + t(\overrightarrow{p_0} - \overrightarrow{p_1})\right) \cdot \overrightarrow{n_0} - D + r$$

Eine Kollision erfolgt nur dann, wenn der Abstand zwischen Ebene und Punkt $d(t) = 0$ ist. Damit erhält man den Zeitpunkt, zu dem eine Kollision stattfindet:

$$0 = \left(\overrightarrow{p_0} + t(\overrightarrow{p_0} - \overrightarrow{p_1})\right) \cdot \overrightarrow{n_0} - D + r$$

$$D - r - \overrightarrow{n_0} \cdot \overrightarrow{p_0} = t \cdot (\overrightarrow{p_0} - \overrightarrow{p_1}) \cdot \overrightarrow{n_0}$$

$$t = \frac{D - r - \overrightarrow{n_0} \cdot \overrightarrow{p_0}}{(\overrightarrow{p_0} - \overrightarrow{p_1}) \cdot \overrightarrow{n_0}}$$

Quader – Ebenen Kollision[97]

Bei der Kollision der Kugel mit einer Ebene haben wir das Problem einer beweglichen Kugel auf einen beweglichen Punkt reduziert, indem wir vom Abstand des Kugelmittelpunkts zur Ebene den Kugelradius abgezogen haben. Der Radius ist der Abstand vom Kugelmittelpunkt zu seiner Oberfläche. Auch der bewegliche Quader kann auf seinen beweglichen Mittelpunkt reduziert werden. Dazu berechnen wir den Abstand D des Quaderzentrums M zur Ebene und ziehen davon den Normalabstand von M zum Eckpunkt, der der Ebene am nächsten ist, ab. Ohne Beschränkung der Allgemeinheit sei der Eckpunkt A der Ebene ε am nächsten. Wir konstruieren nun den Vektor $\vec{r} = \overrightarrow{MA}$ und projizieren diesen auf den normierten Normalvektor $\overrightarrow{n_0}$ der Ebene (siehe Abb.70). Der Betrag des Skalarprodukts $|\vec{r} \cdot \overrightarrow{n_0}|$ ist der Normalabstand vom Mittelpunkt M des Quaders zu seinem Eckpunkt A relativ zur Ebene. Wir nennen diesen Abstand den effektiven Radius r_e. Die Verschiebung der Ebene ist analog zur Kugel $D - r_e$. Auch die Zeit, zu der der Quader mit der Ebene kollidiert, errechnet sich analog zur Kugel:

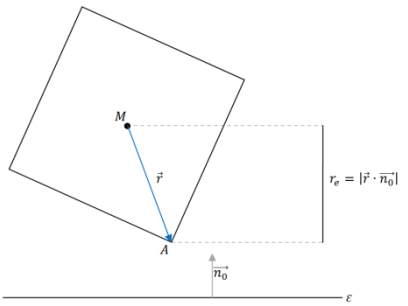

Abb.70: Ermittlung des effektiven Radius

$$t = \frac{D - r_e - \overrightarrow{n_0} \cdot \overrightarrow{p_0}}{(\overrightarrow{p_0} - \overrightarrow{p_1}) \cdot \overrightarrow{n_0}}$$

[97]Vgl. Tremblay (2004), S. 266

Um herauszufinden welcher Eckpunkt des Quaders mit der Ebene kollidiert, werden die Abstände der Eckpunkte zur Ebene verglichen. Der Eckpunkt, der die Ebene trifft, muss einen gleich großen oder kleineren Abstand haben, als die anderen Eckpunkte. Es können maximal vier Eckpunkte gleichzeitig mit der Ebene kollidieren.

3.4.2 Kollision von Objekten untereinander[98]

Auch im Fall von mehreren beweglichen Objekten werden die Bewegungen wieder angenähert. Die Bewegung für jedes Objekt wird durch eine Gleichung beschrieben, und zu jedem Zeitpunkt wird der Abstand ihrer Punkte verglichen. Die Objekte werden dabei entweder durch Quader oder Kugeln angenähert. Im Folgenden wird die Berechnung einer Kugel-Kugel Kollision näher beschrieben.

Kugel-Kugel Kollision[99]

Die Bewegungen zweier Kugeln werden durch die Positionen ihrer Mittelpunkte $\vec{u}$ und $\vec{v}$ beschrieben:

$$\vec{u}(t) = \overrightarrow{u_1} + (\overrightarrow{u_0} - \overrightarrow{u_1})t$$

$$\vec{v}(t) = \overrightarrow{v_1} + (\overrightarrow{v_0} - \overrightarrow{v_1})t$$

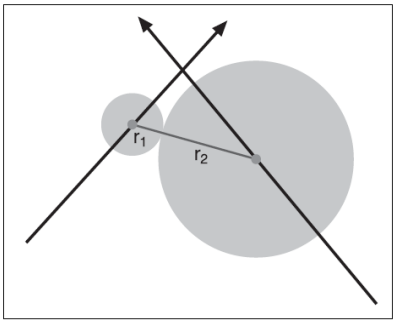

Abb.71: Schnitt zweier beweglicher Kugeln

Der Abstand der Mittelpunkte in Abhängigkeit der Zeit ist gegeben durch:

$$d(t) = |\vec{u}(t) - \vec{v}(t)| = |(\overrightarrow{u_1} - \overrightarrow{v_1}) + (\overrightarrow{u_0} - \vec{u}_1 + \overrightarrow{v_0} - \overrightarrow{v_1})t|$$

Zur Vereinfachung definiert man folgende Vektoren und quadriert die Funktion für den Abstand:

$$\vec{a} = \overrightarrow{u_1} - \overrightarrow{v_1}$$

$$\vec{b} = \overrightarrow{u_0} - \vec{u}_1 + \overrightarrow{v_0} - \overrightarrow{v_1}$$

$$d(t)^2 = \left|\vec{a} + \vec{b}t\right|^2 = \vec{a} \bullet \vec{a} + 2\vec{a} \bullet \vec{b}t + b \bullet \vec{b}t^2$$

Ist d^2 kleiner oder gleich dem Quadrat der summierten Kugelradien, tritt eine Kollision ein (siehe Abb.71). Zur Berechnung der Zeit, zu der die Kugeln kollidieren, wird die quadratische Gleichung gelöst:

$$t = \frac{-\vec{a} \bullet \vec{b} \pm \sqrt{(\vec{a} \bullet \vec{b})^2 - (\vec{b} \bullet \vec{b})(\vec{a} \bullet \vec{a} - d^2)}}{\vec{b} \bullet \vec{b}}$$

[98]Vgl. Ebd., S. 269
[99]Vgl. Tremblay (2004), S. 270

Die Gleichung liefert zwei Lösungen, da sich die Kugeln beim Eintritt und wieder beim Austritt berühren. Da jedoch nur der erste Zeitpunkt von Interesse ist, wird das kleinere Ergebnis gewählt. Da der Abstand d bei der Kollision der Summer der beiden Kugelradien entsprechen muss, lautet die endgültige Formel:

$$t = \frac{-\vec{a} \bullet \vec{b} - \sqrt{(\vec{a} \bullet \vec{b})^2 - (\vec{b} \bullet \vec{b})(\vec{a} \bullet \vec{a} - (r_1 + r_2)^2)}}{\vec{b} \bullet \vec{b}}$$

Die Zeit der Kollision muss im Intervall $[0,1]$ liegen, ist das nicht der Fall, gibt es keine Kollision. Da der Punkt der Berührung auf der Verbindungslinie der Kugelmittelpunkte und im Abstand des Radius vom jeweiligen Zentrum entfernt liegt, erhält man den Berührungspunkt $\overrightarrow{p_u} = \overrightarrow{p_v}$ durch:

$$\overrightarrow{p_u} = \vec{u}(t) + r_u \frac{\vec{v}(t) - \vec{u}(t)}{|\vec{u}(t) - \vec{v}(t)|}$$

$$\overrightarrow{p_v} = \vec{v}(t) + r_v \frac{\vec{u}(t) - \vec{v}(t)}{|\vec{u}(t) - \vec{v}(t)|}$$

3.4.3 Auswahl[100]

Teil diverser Spiele ist die Auswahl von Objekten am Bildschirm über den Mauszeiger, beispielsweise die Auswahl von Truppen in einem Strategiespiel (siehe Abb.72). Dazu muss mit dem Punkt P am Bildschirm mit den Koordinaten $\begin{pmatrix} x \\ y \end{pmatrix}$ eine Gerade konstruiert werden, die von der Kamera ausgehend in die Welt zeigt. Der Bildschirmpunkt wird in einen Punkt aus $\mathbb{R}^3$ umgewandelti indem für die z-Koordinate 0 eingesetzt wird: $P = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}$. Mit dem Normalvektor $\vec{n}$ der Bildschirmebene wird eine Gerade $g: X = P + t \cdot \vec{n}$ konstruiert. Die Auswahl eines Objekts in der Spielwelt wird durch einen Schnitt eines Objekts mit der Gerade g gelöst.

Abb.72: Auswahl von Truppen

[100]Vgl. Tremblay (2004), S. 288

3.3 Geometrische Transformation

In Kapitel 2.2.2 wurde erklärt, dass jedes 3D-Modell eines Spiels zuerst in seinem eigenen Koordinatensystem existiert, dem *Object-Space*. Die Koordinaten der Eckpunkte eines Modells beziehen sich auf den Koordinatenursprung seines Objekt-Space. Meistens liegt der Ursprung innerhalb des 3D-Modells. Platzieren wir nun alle Modelle des Spiels in der Spielwelt, würden sie sich im Koordinatenursprung der Welt überlagern. Wollen wir die Objekte an bestimmten Positionen der Spielwelt platzieren und ausrichten, so müssen wir ihre Koordinaten aus dem Objekt-Space relativ zum Ursprung der Spielwelt transformieren. In der Computergrafik werden zur Transformation der Objektkoordinaten in Weltkoordinaten Abbildungsmatrizen eingesetzt. Dieses Kapitel behandelt verschiedene Transformationen. Objektkoordinaten werden mit $\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$, Weltkoordinaten mit $\overrightarrow{p'} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$ bezeichnet[101].

3.3.1 Translation[102]

Angenommen wir haben ein Objekt, das wir in der Spielwelt platzieren wollen. Das Objekt soll relativ zum Koordinatenursprung der Spielwelt um den Vektor $\vec{r} = \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}$ parallel verschoben werden. Wir erhalten die Weltkoordinaten $\overrightarrow{p'}$ der Punkte des Objekts, indem wir die Objektkoordinaten $\vec{p}$ und den Verschiebungsvektor $\vec{r}$ addieren:

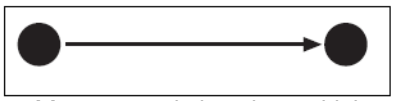

Abb.73: Translation eines Objekts

$$\vec{p} + \vec{r} = \overrightarrow{p'}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} x + \Delta x \\ y + \Delta y \\ z + \Delta z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

Die Weltkoordinaten des Objekts nach einer Verschiebung um $\begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}$ können auch mit einer Matrix berechnet werden:

$$\begin{bmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + \Delta x \\ y + \Delta y \\ z + \Delta z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$

Was hier zuerst auffällt ist, dass die Objekt- und Weltkoordinaten um eine 1 als vierte Koordinate erweitert wurden. Diese vierte Koordinate hat keine Auswirkung auf die Position des Punktes. Sie ist nur dazu da, damit wir die Matrix mit dem Ortsvektor multiplizieren

[101] Vgl. Tremblay (2004), S. 128
[102] Vgl. Ebd., S. 130

können. Um zwei Matrizen zu multiplizieren, muss die Spaltenanzahl der ersten Matrix gleich der Zeilenanzahl der zweiten Matrix sein. (Merkregel: $(m,p)\cdot(p,n)=(m,n)$). Ein Vektor kann als Matrix mit einer Spalte behandelt werden. Da die Matrix vier Spalten hat, braucht unser Vektor vier Zeilen. Nun könnte man jedoch sagen, dass die vierte Zeile der Matrix überflüssig ist, denn auch ohne sie können wir die Weltkoordinaten berechnen:

$$\begin{bmatrix}1 & 0 & 0 & \Delta x\\ 0 & 1 & 0 & \Delta y\\ 0 & 0 & 1 & \Delta z\end{bmatrix}\cdot\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}=\begin{bmatrix}x+\Delta x\\ y+\Delta y\\ z+\Delta z\end{bmatrix}=\begin{bmatrix}x'\\ y'\\ z'\end{bmatrix}$$

Wieso in der Computergrafik (4 x 4)-Abbildungsmatrizen angewandt werden, darauf wird später eingegangen.

3.3.2 Skalieren[103]

Hierbei geht es darum, Objekte zu vergrößern beziehungsweise zu verkleinern (siehe Abb.74). Das erreicht man, indem jede Koordinate eines Punkts mit einem Skalierungsfaktor s multipliziert wird. Die Gleichung, um eine Koordinate x zu skalieren, hat die Form $x'=sx$. Soll jede Koordinate um einen anderen Wert skaliert werden, beispielsweise um ein Objekt in die Länge zu ziehen, muss für jede Koordinate ein eigener Skalierungsfaktor definiert werden. Wir definieren s_x, s_y und s_z als die Skalierungsfaktoren für die entsprechenden Koordinaten. Wir können die Weltkoordinaten nach der Skalierung der Objektkoordinaten mit einer Matrix berechnen:

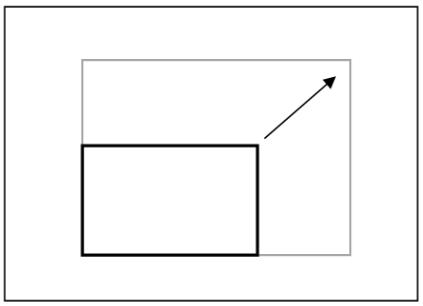

Abb.74: Vergrößern eines Objekts

$$\begin{bmatrix}s_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & s_z\end{bmatrix}\cdot\begin{bmatrix}x\\ y\\ z\end{bmatrix}=\begin{bmatrix}s_x\cdot x\\ s_y\cdot y\\ s_z\cdot z\end{bmatrix}=\begin{bmatrix}x'\\ y'\\ z'\end{bmatrix}$$

Bei der Skalierung würde uns schon eine (3x3)-Matrix reichen. Aus Gründen die später behandelt werden, wird jedoch auch für die Skalierung eine (4x4)-Matrix verwendet:

$$\begin{bmatrix}s_x & 0 & 0 & 0\\ 0 & s_y & 0 & 0\\ 0 & 0 & s_z & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}x\\ y\\ z\\ 1\end{bmatrix}=\begin{bmatrix}s_x\cdot x\\ s_y\cdot y\\ s_z\cdot z\\ 1\end{bmatrix}=\begin{bmatrix}x'\\ y'\\ z'\\ 1\end{bmatrix}$$

Wird für einen Skalierungsfaktor der Wert -1 gewählt, dann wird der Punkt in Richtung der entsprechenden Achse gespiegelt. Skalierung erfüllt demnach zwei Aufgaben: Sie kann die Größe von Objekten verändern und Objekte spiegeln.

[103] Vgl. Tremblay (2004), S. 131

3.3.3 Scherung[104]

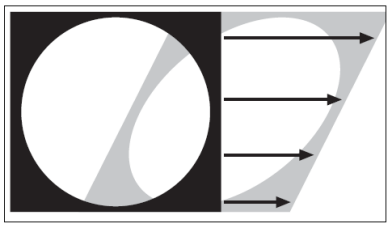

Abb.75: Scherung eines Objekts

Betrachten wird die Scherung des Bildes in Abb.75. Stellen wir uns vor, dass wir das Bild entlang der x-Achse in hauchdünne Streifen schneiden, die wir dann Stück für Stück gegeneinander in x-Richtung verschieben. Die Scherung kann also als stückweise Translation gesehen werden. Je weiter wir in Richtung der y-Achse gehen, umso weiter müssen wir einen Streifen in x-Richtung verschieben. Die Translation der x-Koordinate ist also von der y-Koordinate und einem Faktor k abhängig, mit dem wir festlegen, wie stark der Körper geschert wird: $x' = x + k \cdot y$. Während beim Strecken Flächen vergrößert werden, bleiben sie bei der Scherung gleich groß. In 3D können wir eine Scherung entlang einer Koordinatenachse von den anderen Koordinaten abhängig machen. Die Koordinaten eines Objekts, das in x-Richtung geschert wird kann auf folgende Art berechnet werden:

$$\begin{bmatrix} 1 & k_{xy} & k_{xz} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x + k_{xy}y + k_{xz}z \\ y \\ z \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Auch beim Skalieren werden in der Praxis (4x4)-Matrizen verwendet. Je nachdem, in welche Richtung der Körper geschert wird, können unterschiedliche Transformationsmatrizen eingesetzt werden:

$$\begin{bmatrix} 1 & k_{xy} & k_{xz} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + k_{xy}y + k_{xz}z \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$ Scherung in x-Richtung

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ k_{yx} & 1 & k_{yz} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ k_{yx}x + y + k_{yz}z \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$ Scherung in y-Richtung

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ k_{zx} & k_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ k_{zx}x + k_{zy}y + z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$ Scherung in z-Richtung

[104]Vgl. Tremblay (2004), S. 133

3.3.4 Rotation[105]

Bei der Rotation wird ein Punkt um einen Winkel φ um den Koordinatenursprung gedreht (siehe Abb.76). Ein Punkt $\begin{pmatrix} x \\ y \end{pmatrix}$, der in 2D um den Winkel φ gegen den Uhrzeigersinn gedreht wird, kommt auf $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \cdot \cos\varphi - y \cdot \sin\varphi \\ x \cdot \sin\varphi + y \cdot \cos\varphi \end{pmatrix}$ zu liegen. Bei einer Rotation in 3D wird unterschieden, um welche Achse der Körper rotieren soll:

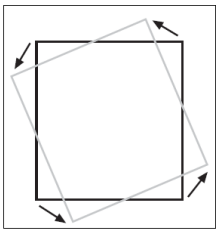

Abb.76: Rotation eines Quadrats

$$\begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ Rotation um die z-Achse

$$\begin{bmatrix} \cos(\varphi) & 0 & \sin(\varphi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\varphi) & 0 & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ Rotation um die y-Achse

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) & 0 \\ 0 & \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$ Rotation um die x-Achse

3.3.5 Kombinationen von Transformationen[106]

Bei der Darstellung von Objekten werden oft mehrere Transformationen hintereinander angewandt, beispielsweise eine Skalierung gefolgt von einer Translation. Auf jeden Punkt des Objekts muss die gleiche Folge von Transformationen angewandt werden. Der Grund, warum wir alle Transformationsmatrizen einheitlich als (4x4)-Matrizen formulieren ist, dass wir quadratische Matrizen in beliebiger Reihenfolge multiplizieren können und wieder eine Matrix vom gleichen Typ erhalten. Dadurch können wir mehrere Transformationen, die auf einen Punkt angewandt werden sollen, multiplizieren und die Gesamtmatrix auf den Punkt anwenden. Nehmen wir zwei Transformationsmatrizen $\boldsymbol{A}$ und $\boldsymbol{B}$. Wir wollen $\boldsymbol{B}$ gefolgt von $\boldsymbol{A}$ auf den Vektor $\vec{p}$ anwenden. Statt $\overrightarrow{p'} = \boldsymbol{B} \cdot (\boldsymbol{A} \cdot \vec{p})$ schreibt man $\overrightarrow{p'} = \boldsymbol{C} \cdot \vec{p}$ (für $\boldsymbol{C} = \boldsymbol{B} \cdot \boldsymbol{A}$). Auch die Inverse einer Transformationsmatrix, also ihre Umkehrung, kann mit anderen Transformationsmatrizen multipliziert werden.

[105]Vgl. Ebd., S. 135
[106]Vgl. Tremblay (2004), S. 138

Als Beispiel für eine komplexe Transformation wollen wir eine Drehung um Winkel φ um eine beliebige Achse im Raum (siehe Abb.77) herleiten. Die Achse sei durch den Ursprung und einen Richtungsvektor $\vec{u}$ gegeben[107]:

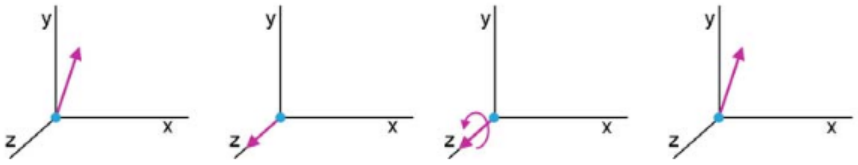

Abb.77: Rotation um eine beliebige Achse

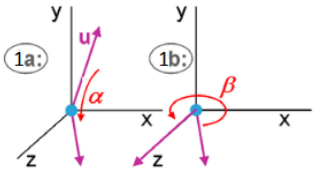

Abb.78: Drehung von u in die z-Achse

1. Schritt: Vektor $\vec{u}$ in die z-Achse drehen (Abb.78)
   1a: Vektor $\vec{u}$ um die x-Achse in die xz-Ebene drehen: $\boldsymbol{R_x}(\alpha)$

   Sei $\vec{u} = (a, b, c)$, dann ist $\overrightarrow{u'} = (0, b, c)$ die Projektion von $\vec{u}$ auf die yz-Ebene.

   Der Drehwinkel α um die x-Achse ergibt sich aus $cos\ \alpha = \frac{c}{\sqrt{b^2+c^2}}$

   1b: Vektor $\vec{u}$ um die y-Achse in die z-Achse drehen: $\boldsymbol{R_y}(\beta)$

   Der Drehwinkel β um die y-Achse ergibt sich aus $cos\ \beta = \frac{c}{\sqrt{a^2+c^2}}$
2. Schritt: Drehung um φ um die z-Achse ausführen: $\boldsymbol{R_z}(\varphi)$
3. Schritt: Vektor $\vec{u}$ in die ursprüngliche Richtung zurückdrehen: zuerst $\boldsymbol{R_y}^{-1}(\beta)$, dann $\boldsymbol{R_x}^{-1}(\alpha)$

Die resultierende Transformationsmatrix ist somit:

$$\boldsymbol{R_u}(\varphi) = \boldsymbol{R_x}^{-1}(\alpha) \cdot \boldsymbol{R_y}^{-1}(\beta) \cdot \boldsymbol{R_z}(\varphi) \cdot \boldsymbol{R_y}(\beta) \cdot \boldsymbol{R_x}(\alpha)$$

Ein Punkt, der um die beliebige Achse rotiert, hat danach die neue Position:

$$\overrightarrow{p'} = \boldsymbol{R_u}(\varphi) \cdot \vec{p}$$

3.4 Projektion

Um die virtuellen dreidimensionalen Objekte der Spielwelt auf einem Bildschirm darzustellen, müssen sie von 3D auf 2D projiziert werden. Dabei wird zwischen *orthographischer* und *perspektivischer* Projektion unterschieden (siehe Kapitel 2.2.2 Projektion)[108].

Bei der *orthographischen Projektion* werden alle Punkte normal auf die xy-Ebene projiziert. Dazu werden die z-Koordinaten aller Punkte auf null gesetzt. Die x- und y-Koordinaten bleiben unverändert. Die Koordinaten des projizierten Punktes $\overrightarrow{p'}$ erhalten wir, indem wir den Punkt $\vec{p}$ mit der Projektionsmatrix $\boldsymbol{P_o}$ multiplizieren. Das Resultat wird in Abb.79 illustriert[109].

[107] Vgl. cg.tuwien.ac.at [Zugriff am 29.4.2018]
[108] Vgl. Akenine-Möller (2008), S. 91
[109] Vgl. Akenine-Möller (2008), S. 90

$$\overrightarrow{p'} = \boldsymbol{P_o} \cdot \vec{p} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

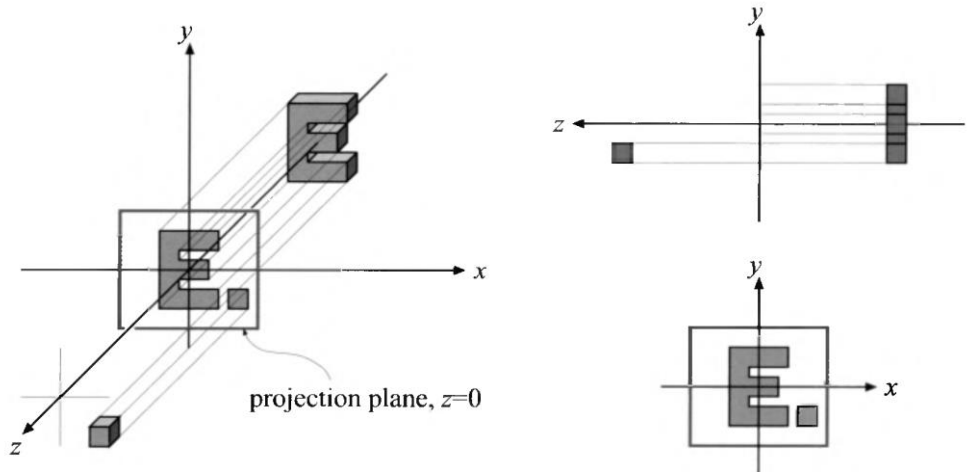

Abb.79: Orthogonale Projektion

Die *perspektivische Projektion* imitiert, wie wir Objekte wahrnehmen. Je weiter ein Gegenstand von der Kamera entfernt ist, umso kleiner soll er nach der Projektion erscheinen, außerdem können sich parallele Linien im Horizont treffen[110].

Die Mathematik dahinter sieht wie folgt aus: Wir gehen davon aus, dass sich die Kamera im Koordinatenursprung befindet. Wir projizieren einen Punkt $\vec{p}$ entlang einer Linie vom Punkt zum Ursprung auf die Ebene $z = -d, d > 0$ und erhalten den neuen Punkt $\vec{q} = (q_x \, |q_y| - d)$ (siehe Abb.80).

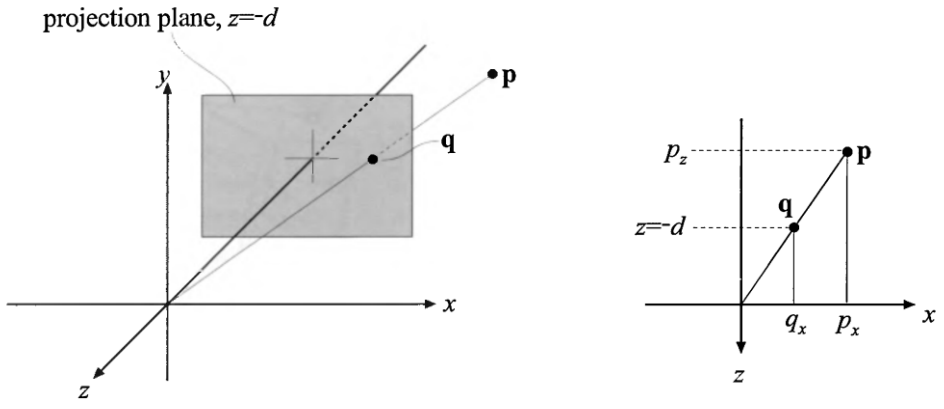

Abb.80: Herleitung der Koordinaten für den projizierten Punkt $\vec{q}$

Aus den ähnlichen Dreiecken in Abb.81 lässt sich die x-Koordinate für $\vec{q}$ wie folgt herleiten:

[110]Vgl. Akenine-Möller (2008), S. 19

$$\frac{q_x}{p_x} = \frac{-d}{p_z} <=> q_x = -d\frac{p_x}{p_z}$$

Die restlichen Koordinaten ergeben sich analog als $q_y = -d\frac{p_y}{p_z}$ und $q_z = -d\frac{p_z}{p_z} = -d$. Die Perspektivenprojektionsmatrix $\boldsymbol{P_p}$ ist:

$$\boldsymbol{P_p} = \begin{bmatrix} -d/p_z & 0 & 0 & 0 \\ 0 & -d/p_z & 0 & 0 \\ 0 & 0 & -d/p_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dass diese Matrix die korrekte Projektion liefert, kann man mit folgendem Beispiel zeigen:

$$\vec{q} = \boldsymbol{P_p} \cdot \vec{p} = \begin{bmatrix} -d/p_z & 0 & 0 & 0 \\ 0 & -d/p_z & 0 & 0 \\ 0 & 0 & -d/p_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} -dp_x/p_z \\ -dp_y/p_z \\ -d \\ 1 \end{bmatrix} = \begin{bmatrix} q_x \\ q_y \\ q_z \\ 1 \end{bmatrix}$$

Die z-Komponente ist immer $-d$ da wir auf diese Ebene projizieren[111].

3.5 Sichtbarkeit

Je mehr Objekte in einer Szene vorhanden sind, desto größer ist die Rechenarbeit für die GPU, die jedes Objekt darstellen will. Zur Entlastung der GPU existieren mehrere Methoden.

3.5.1 z-Buffer[112]

Der z-Buffer ist ein Zwischenspeicher für die Pixel und wird vor der Ausgabe am Bildschirm angewandt. Der z-Buffer merkt sich den z-Wert für jeden einzelnen Pixel. Wenn ein Element gezeichnet werden soll, überprüft die GPU, ob der neue Pixel vor dem alten liegt, indem es die z-Werte vergleicht. Hat das neue Element einen kleineren z-Wert, dann ist es näher an der Kamera,und die vorherige Farbe für den Pixel wird überschrieben (siehe Abb.81). Der z-Buffer arbeitet sehr schnell, hat jedoch Probleme, da er nur endlich viele Gleitkommastellen speichern kann. Bei Objekten, die einander überlappen oder einander sehr nahe sind, kann es zu Überschneidungen kommen (siehe Abb.82).Weit von der Kamera entfernte Objekte sind davon mehr betroffen.

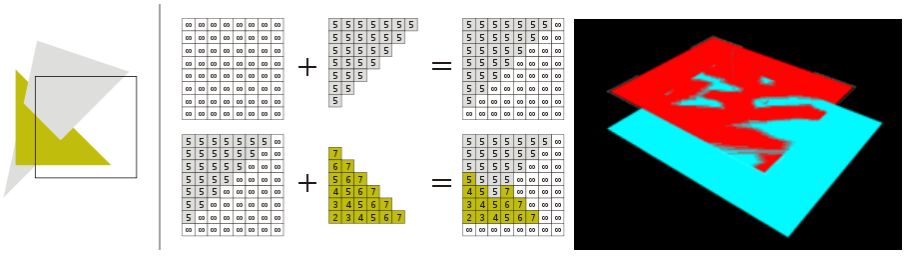

Abb.81: Prinzipdes z-Buffers

Abb.82: Fehler des z-Buffers

[111]Vgl. Ebd., S. 19
[112]Vgl. Tremblay (2004), S. 406

3.5.2 Culling[113]

Der Begriff *Culling* (dt; Auslese, Wegschneiden) ist eine Methode, bei der Polygone, die nicht sichtbar wären, nicht für die Darstellung berechnet werden. Man unterscheidet *Back-Face Culling, Frustrum Culling, Occlucion Culling, Detail Culling* und *Depth Culling*

3.5.3 Back-face Culling[114]

Back-Face Culling ist die einfachste Form der Sichtbarkeitsüberprüfung. Die Idee dahinter ist, dass die meisten Objekte abgeschlossen sind, somit ist nur die Oberfläche sichtbar. Zudem kann man ein Objekt nur von einer Seite betrachten. Im Falle eines Würfels sind maximal 3 Seiten gleichzeitig sichtbar, die restlichen müssten nicht berechnet werden. Dazu wird für jedes Polygon die Vorderseite durch einen Normalvektor $\overrightarrow{n_P}$ definiert. Dieser wird dann mit dem Vektor der Blickrichtung $\boldsymbol{v}$ verglichen, indem man das Skalarprodukt $\overrightarrow{n_P} \bullet \vec{v}$ bildet. Ist das Ergebnis ≤ 0, dann ist die Fläche von vorne zu sehen, ist das Ergebnis > 0, dann wäre die Fläche nur von hinten sichtbar und wird daher nicht dargestellt (siehe Abb.83).

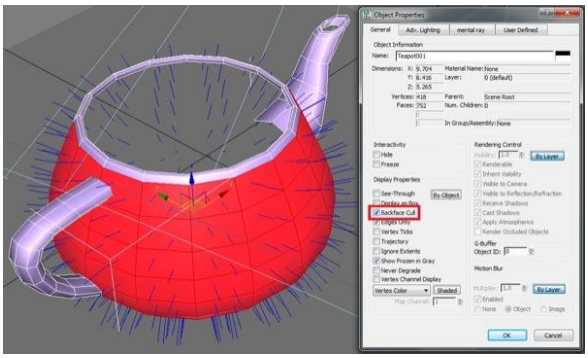

Abb.83: Innenflächen der Teekanne werden nicht gezeichnet

3.5.4 Frustrum Culling[115]

Beim FrustrumCulling sollen nur jene Objekte gezeichnet werden, die sich vollständig oder zumindest teilweise im Sichtvolumen (Frustrum) befinden (siehe Abb.84). Dazu werden die Weltkoordinaten der Objekte mit den Grenzen des Sichtvolumens verglichen. Gegebenenfalls kann der Bereich auch über das Sichtvolumen vergrößert werden, damit Objekte etwas außerhalb auch noch gezeichnet werden und sie bei einem schnellen Kameraschwenk nicht plötzlich ins Bild springen. Nach der Projektion in 2D können noch alle Polygone abgeschnitten werden, die sich außerhalb des Bildschirms befinden (Frustrum Clipping).

Abb.84: FrustrumCulling in *Horizon Zero Dawn* (2017)

[113]Vgl. Ebd., S. 408
[114]Vgl. Ebd., S. 408
[115]Vgl. Tremblay (2004), S. 409

3.5.5 Object Culling[116]

Hierbei sollen Objekte, die von anderen Objekten verdeckt werden, ausgeblendet werden. Aus der Sicht der Kamera wird um das verdeckende Objekt ein einfaches Umgrenzungsvolumen erstellt (sozusagen ein „Schattenvolumen“), und überprüft, ob das Objekt in diesem Volumen liegt (siehe Abb.85). Dazu muss man den Rand des abdeckenden Volumens, aus der Sicht der Kamera, finden. Aus immer 2 Eckpunkten des Randes, zusammen mit dem Ursprung der Kamera, wird eine Ebene gezeichnet. Alle Ebenen zusammen bilden die Mantelfläche des Schattens. Den Rand findet man, indem man für alle Polygone des Objekts eine Liste aller Dreieckskanten anlegt und überprüft, ob die dazugehörigen Dreiecke sichtbar sind. Gehört eine Kante zu zwei sichtbaren Dreiecken, dann kann sie kein Rand sein. Eine Randkante muss zwischen einem sichtbaren und einem nicht sichtbaren Dreieck liegen.

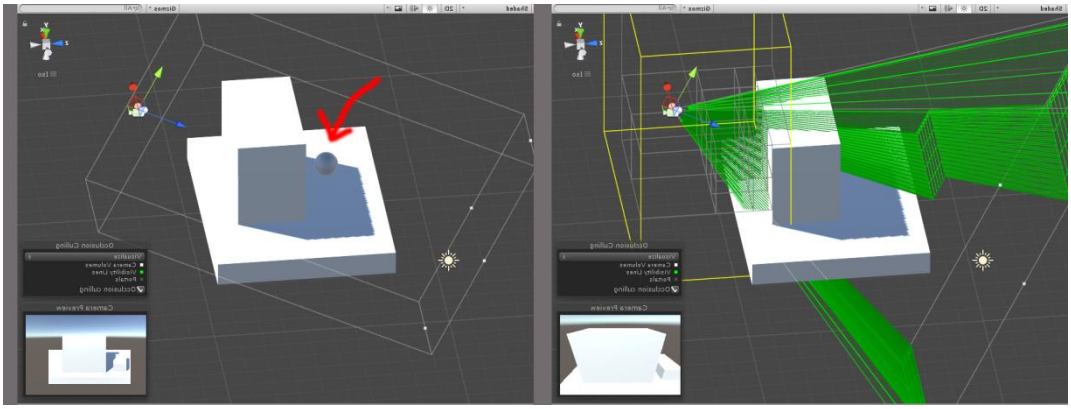

Abb.85: Ball hinter der Wand wird ausgeblendet

3.5.6 Detail Culling[117]

Weit entfernte Objekte müssen nicht so genau dargestellt werden. Daher werden für verschiedene Entfernungen von der Kamera unterschiedliche Versionen eines Objekts erstellt, die zunehmend weniger Polygone haben (siehe Abb.86).

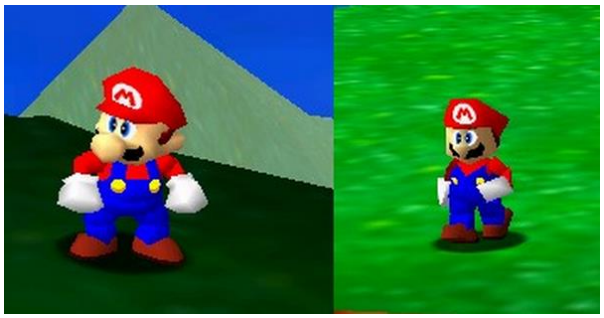

Abb.86: Normale und Niedrig-Polygon Version von Mario

[116]Vgl. Ebd., S. 421
[117]Vgl. Tremblay (2004), S. 425

3.5.7 Depth Culling[118]

Objekte in einer gewissen Entfernung werden nicht mehr gezeichnet. Das kann dazu führen, dass während einer Bewegung Objekte in der Ferne plötzlich auftauchen oder verschwinden. Um diesen Effekt zu verstecken, wird in vielen Spielen eine Nebeltextur in der Ferne dargestellt. Alternativ können Objekte langsam transparent gemacht werden, bevor sie verschwinden.

3.6 Helligkeit und Lichteffekte

Abb.87: Minecraft ohne (links) und mit Shader (rechts)

Wenn Bilder von dreidimensionalen Objekten am Bildschirm gezeichnet werden, sollen sie nicht nur die richtige Form haben, sondern auch das gewünschte Aussehen besitzen. Die grundlegende Optik wird durch die Texturen bestimmt, die auf die Polygone angebracht werden. In vielen Fällen ist das Ziel aber Fotorealismus. Um dieses Ziel zu erreichen, orientiert man sich an der Wirklichkeit. Wie verhaltet sich Licht und welche Auswirkungen hat das Material? Licht geht von einer Lichtquelle aus und interagiert mit Objekten einer Szene. Ein Teil wird absorbiert, und ein Teil wird gestreut und bewegt sich in eine neue Richtung. Zum Schluss trifft es auf einen Sensor (Auge). Zur Erzeugung von Lichteffekten werden *Shader* angewandt. Das sind Anweisungen, die auf jeden Pixel angewandt werden, um deren Farben zu verändern (siehe Abb.87). Für unterschiedliche Effekte gibt es unterschiedliche Shader[119].

3.6.1 Beleuchtungsmodelle[120,121]

Lichteffekte in Spielen werden üblicherweise aus Kombinationen direkter und indirekter Beleuchtung und diffuser und spiegelnder Reflexion erzeugt. Diffuse Reflexion simuliert wie eine raue Oberfläche Licht streut. Spiegelnde Reflexion simuliert den Glanz von glatten Oberflächen. Während raue Oberflächen Licht in alle Richtungen streuen, wird das Licht durch glatte Oberflächen annähernd parallel reflektiert. Direkte Beleuchtung geht von einer Lichtquelle aus. Dieses Licht wird teilweise von Objekten gestreut und bewirkt indirekte Beleuchtung.

[118] Vgl. Ebd., S. 426
[119]Vgl. Akenine-Möller (2008), S.99
[120]Vgl. Tremblay (2004), S. 492
[121]Vgl. Akenine-Möller (2008), S.110

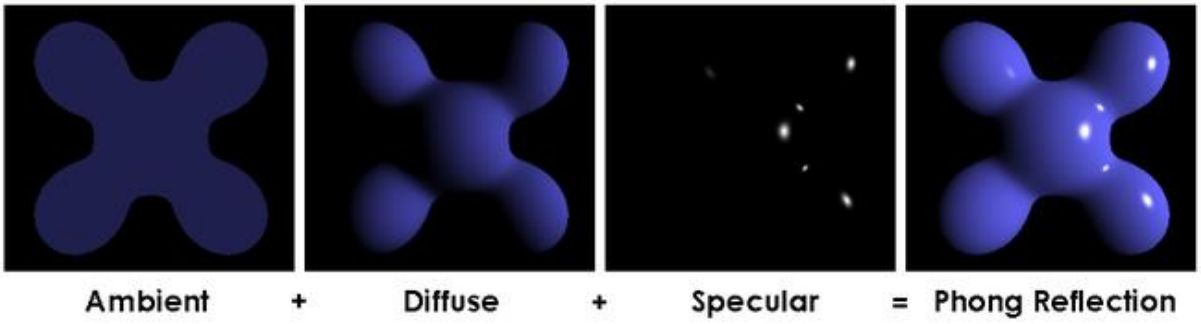

Abb.88: Zusammensetzung des Beleuchtungsmodells

Ein Beleuchtungsmodell beschreibt die Reflexion von Oberflächen als Kombination aus diffuser Reflexion rauer Oberflächen und spiegelnder Reflexion glatter Oberflächen. Es basiert auf der Beobachtung, dass glatte Oberflächen kleine glänzende Highlights besitzen, während die Highlights bei rauen Oberflächen größer sind und nach außen hin schwächer werden. Das Modell beinhaltet auch einen Ambients-Anteil, der die indirekte Beleuchtung der Umgebung einbezieht. Für jeden Pixel wird eine Lichtintensität berechnet, die in Summe aus der Intensität jeder Beleuchtungsart besteht (siehe Abb.88).

Die ambiente Komponente der Intensität ist üblicherweise richtungsunabhängig, da sie nur eine Annäherung des indirekt reflektierten Lichts der Umgebung ist. Sie ist eine Konstante, die den niedrigsten Lichtlevel beschreibt, den eine Fläche haben kann, ob sie nun beleuchtet ist oder nicht. Für die ambiente Lichtintensität eines Pixels auf einer Oberfläche gilt:

$$i_{ambient} = i_a k_{ambient}$$

wobei i_a die konstante Intensität des Umgebungslichts und $k_{ambient}$ eine Materialkonstante ist[122].

Bei diffuser Reflexion wird das Licht unabhängig vom Standpunkt des Betrachters in alle Richtungen gestreut. Die Intensität ist jedoch abhängig vom Einfallswinkel. Eine Fläche, auf die Licht senkrecht strahlt, wird stärker beleuchtet als eine Fläche, auf die Licht schräg einfällt. Beschrieben wird dieser Effekt durch das Lambertsche-Gesetz:

$$i_{diff} = i_{source} k_{diff} \left(\overrightarrow{n_0} \bullet \vec{d}\right)$$

Der Vektor $\overrightarrow{n_0}$ ist der normierte Normalvektor eines Oberflächenpolygons und $\boldsymbol{d}$ ist der normierte Richtungsvektor, der zur Lichtquelle zeigt. Das Skalarprodukt liefert dann den Kosinus des Einfallswinkels. Die Intensität der Lichtquelle ist entfernungsabhängig und wird durch i_{source} beschrieben. Bei direktem Licht zeigt das Licht nur in eine Richtung und hat

[122]Vgl. Tremblay (2004), S. 489

überall die gleiche Intensität. Für punktförmige Lichtquellen kann sie durch folgende Formel berechnet werden:

$$i = \frac{1}{a + b \cdot d + c \cdot d^2}$$

Die Werte a, b und c sind Konstanten, die für jedes Licht definiert werden und d ist der Abstand zur Lichtquelle. Da der Normalvektor über das ganze Polygon gleich ist, wird für jeden Pixel auf dem Polygon die gleiche Farbe berechnet, man spricht von *Flat-Shading*. Um einen gleichmäßigen Verlauf der Farben über das Polygon zu erhalten, wird das Lambert-Shading durch andere Methoden, beispielsweise durch das Gouraud-Shading erweitert(siehe Abb.89). Hierbei werden Normalvektoren an den Eckpunkten der Polygone berechnet, und für jeden Eckpunkt wird die Farbe ermittelt. Diese wird dann über die Fläche des Dreiecks interpoliert[123].

Bei spiegelnder Reflexion wird das Licht in einer gewissen Umgebung der idealen Reflexionsrichtung reflektiert. Die Intensität ist vom Blickwinkel abhängig. Man erhält sie durch:

$$i_{spec} = i_{source} k_{spec} (\vec{r} \bullet \vec{v})^n$$

mit dem Vektor des reflektierten Lichtstrahls $\boldsymbol{r}$ und der Blickrichtung des Betrachters $\vec{v}$. Der Exponent n dient zur Beschreibung der Oberflächenbeschaffenheit. Ein Wert von 1 würde ein sehr großes Highlight ergeben, das im Grunde das ganze Objekt beleuchtet, ein Wert von 50 würde ein scharfes Highlight erzeugen. Da das Gouraud-Shading die Farbe von einem Eckpunkt zum anderen interpoliert, kann ein Highlight in der Mitte des Polygons nicht richtig dargestellt werden. Abhilfe schafft das Phong-Shading. Hierbei werden die Normalvektoren zwischen den Eckpunkten interpoliert, welche dann zur Berechnung der Intensität herangezogen werden. Das Resultat ist ein weicherer Verlauf über die Oberfläche (siehe Abb.91)[124].

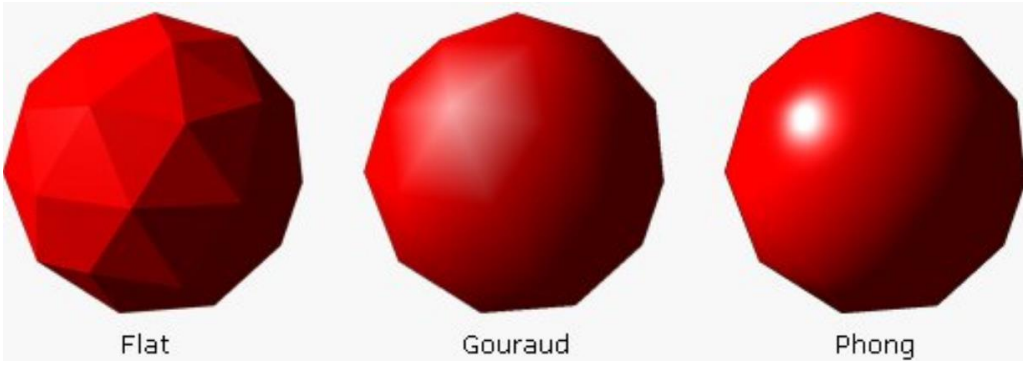

Abb.89: Vergleich von Flat-, Gouraud- und Phong-Shading

[123]Vgl. Ebd., S. 493
[124]Vgl. Tremblay (2004),S.494

3.6.2 Cube Mapping[125]

Um den Effekt von spiegelnden Oberflächen zu simulieren, muss die Umgebung auf die Oberfläche des Objekts projiziert werden. Dazu wäre von jeder Position des spiegelnden Objekts das Aussehen der Umgebung zu berechnen. Da dies zu viel Rechenaufwand bedeuten würde, bedient man sich eines Tricks. Von einem Punkt aus werden 6 Bilder der Umgebung für jede Richtung erzeugt und auf die Innenflächen eines Würfels angebracht. Aus diesem Würfel bezieht die spiegelnde Oberfläche ihre Texel (=Pixel einer Textur). Ausgehend von den Normalvekoren der Objektoberfläche werden die Texel gesucht, die in der Verlängerung des Vektors liegen (siehe Abb.90). Sie werden auf der Oberfläche angebracht, und in Summe wird die Umgebung auf der Oberfläche gespiegelt.

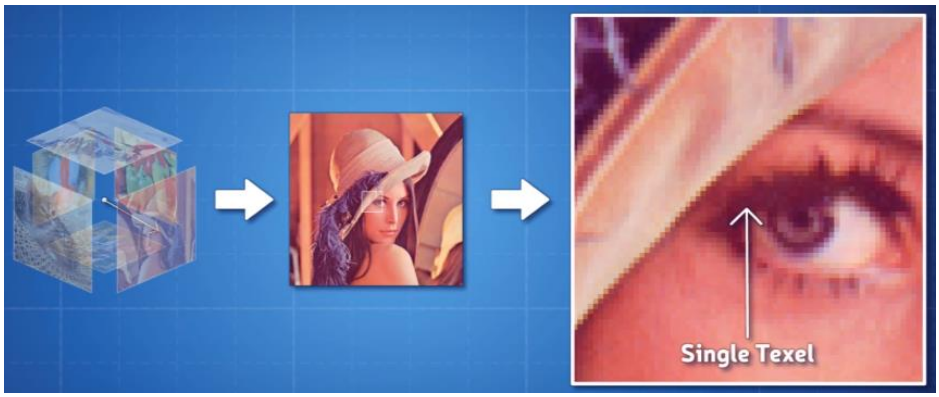

Abb.90: Pixel-Auswahl in der Cube-Map

Auch die indirekte diffuse Beleuchtung kann durch Cube-Maps dargestellt werden, indem die Bilder der Cube-Map unscharf gemacht werden.

3.6.3 Bumpmapping[126]

Da das Beleuchtungsmodell die Oberflächennormalen braucht, um Helligkeit zu berechnen, kann man die Normalvektoren für jeden Pixel der Oberfläche in einer 2D-Textur (Normal-Map) speichern. Wendet man nun Shader auf die Textur an, wird entsprechend der gespeicherten Normalvektoren ein Beleuchtungs-gradient so erzeugt, als wäre das Objekt 3D (siehe Abb.91). Analog kann für Vertiefungen in der Oberfläche eine Hight-Map erstellt werden.

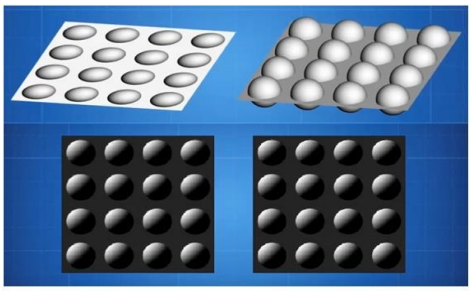

Abb.91: Beleuchtung einer Normalen-Map

[125]Vgl. Ebd., S.463
[126]Vgl. Tremblay (2004),S.496

3.6.4 CelShading[127]

Dieses Verfahren wird eingesetzt, um 3D Objekten die Optik einer Zeichnung zu geben. Anstatt einen linearen Verlauf der Farben zu erzeugen, werden für bestimmte Bereiche des Lichteinfallswinkels nur einheitliche Farben verwendet (siehe Abb.92).

Abb.92: Legend of Zelda: Wind Waker

3.6.5 Schatten[128]

Zur Darstellung von Schatten kommen in modernen Spielen überwiegend zwei Techniken zum Einsatz: *Shadow-Mapping* und *Shadow-Volume*.

Beim Shadow-Mapping wird die Szene zuerst aus der Sicht der Lichtquelle gerendert. Der z-Buffer ermittelt die sichtbaren Oberflächen und speichert sie in der Shadow-Map. Alle anderen Oberflächen erhalten eine Schattentextur.

Beim Shadow-Volume wird um ein Objekt, das einen Schatten wirft, ein Schattenvolumen konstruiert (siehe Kap.3.5.5 Object Culling). Zur Überprüfung, ob ein Polygon im Schattenvolumen liegt, werden Sichtstrahlen verwendet (siehe Abb.93). Ein Sichtstrahl geht vom Beobachter aus und trifft irgendwann auf ein Polygon. Anschließend wird überprüft, wie oft der Sichtstrahl zwischen Beobachter und Polygon eine Grenzfläche des Schattenvolumens geschnitten hat. Gab es keinen Schnitt, dann liegt das Polygon außerhalb des Schattens. Gab es zwei Schnitte, dann ist der Strahl einmal in den Schatten eingetreten und dann wieder ausgetreten, auch hier liegt das Polygon außerhalb des Schattens. Gab es nur einen Schnitt, dann liegt das Polygon im Schatten und erhält beim Rendern eine Schattentextur. Im Allgemeinen gilt: Bei einer geraden Anzahl von Schnitten mit den Grenzflächen des Schattenvolumens liegt das Polygon außerhalb, bei einer ungeraden Anzahl innerhalb des Schattens. Die Ergebnisse dieser Tiefenüberprüfung werden im *Stencil-Buffer* gespeichert. Anschließend wird die gesamte Szene gerendert, als würde sie im Schatten liegen. Mit den Werten im Stencil-Buffer wird eine Lochmaske für die Szene generiert, die an den Stellen Löcher hat, wo Flächen im Schatten liegen. Danach wird mit dieser Maske die Szene ohne Schatten gerendert und beide Bilder werden kombiniert (siehe Abb.94).

[127]Vgl. Ebd,S.466
[128]Vgl. Tremblay (2004),S.468

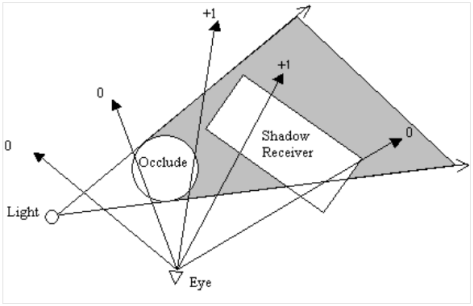

Abb.93: Tiefenpass-Test

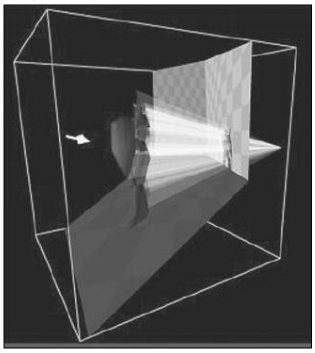

Abb.94: Schattenvolumen und fertiger Schatten

4. Einbindung im Unterricht

Ausgewählte Beispiele aus der Echtzeit-Bildsynthese, im Besonderen aus den Bereichen lineare Algebra und der Geometrie lassen sich im Unterricht der Sekundarstufe II unterbringen.

Lehrstoff[129]:

5. Klasse:

Vektoren und analytische Geometrie der Ebene

- Ermitteln von Einheitsvektoren und Normalvektoren
- Arbeiten mit dem skalaren Produkt, Ermitteln des Winkels zweier Vektoren
- Beschreiben von Geraden durch Parameterdarstellungen und durch Gleichungen, Schneiden von Geraden
- Lösen von geometrischen Aufgaben, gegebenenfalls unter Einbeziehung der Elementargeometrie

6. Klasse:

Analytische Geometrie des Raumes

- Übertragen bekannter Begriffe und Methoden aus der zweidimensionalen analytischen Geometrie, Erkennen der Grenzen dieser Übertragbarkeit

- Ermitteln von Normalvektoren, Definieren des vektoriellen Produkts

- Beschreiben von Geraden und Ebenen durch Parameterdarstellungen bzw. Gleichungen

- Schneiden von Geraden und Ebenen, Untersuchen Lagebeziehungen

- Lösen von geometrischen Aufgaben, gegebenenfalls unter Einbeziehung der Elementargeometrie und der Trigonometrie

Auch im Physiklehrstoff lassen sich Elemente aus der Echtzeit-Bildsynthese finden:

Lehrstoff[130]

5. und 6. Klasse:

- mit Hilfe der Bewegungslehre (Relativität von Ruhe und Bewegung, Bewegungsänderung: Energieumsatz und Kräfte, geradlinige und kreisförmige Bewegung, Impuls und Drehimpuls, Modell der eindimensionalen harmonischen Schwingung) Verständnis für Vorgänge, beispielsweise im Verkehrsgeschehen oder bei der Planetenbewegung, entwickeln

[129]Zitat: https://www.bmb.gv.at/schulen/unterricht/lp/lp_neu_ahs_07_11859.pdf?5te97p S. 4ff [Zugriff am 12.7.2017]

[130] Zitat: https://www.bmb.gv.at/schulen/unterricht/lp/lp_neu_ahs_10_11862.pdf?5te97s [Zugriff am 12.7.2017]

4.1 Mathematikunterricht 5. & 6. Klasse

Im Physikunterricht werden Versuche eingesetzt, um den Lernprozess zu unterstützen. Sie können als Einleitung zu einem Thema eingesetzt werden, um eine Frage aufzuwerfen, mit der sich die SchülerInnen auseinandersetzen sollen, oder sie können am Ende eines Themas eingesetzt werden, um einen Zusammenhang zu demonstrieren. Auch im Mathematikunterricht werden Situationen aus dem Alltag in Beispielen verbaut, um einen Bezug des Stoffs zur Realität aufzustellen. Zu den Themen *Vektorrechnung* und *analytische Geometrie* wird meistens ein Bezug zur Physik aufgestellt. In diesem Kapitel werden GeoGebra-Aufgaben vorgestellt, die vereinfachte Situationen aus Videospielen behandeln. Ziel dieser Aufgaben ist, dass die SchülerInnen Anwendungsmöglichkeiten der analytischen Geometrie und Vektorrechnung kennenlernen. Hierfür sollen sich die SchülerInnen, unter Anwendung von GeoGebra, mit der mathematischen Lösung vorgegebener Probleme auseinandersetzen, wobei die Lehrperson entweder Hilfestellung bietet, wie die Lösung zu erreichen ist, oder der gesamte Lösungsschritt erklärt wird. Anschließend sollen die SchülerInnen eine gleichartige GeoGebra-Aufgabe erstellen oder ein Rechenbeispiel lösen. Um den Unterricht auszuschmücken, gibt es für die SchülerInnen zu jedem Beispiel Kästchen mit wissenswerten Informationen darüber, wie bestimmte Effekte in Videospielen erzeugt werden. Links zu Demonstrationsvideos befinden sich im Anhang.

Geometrie Transformation:

GeoGebra Link: https://ggbm.at/uAAmRQKY (Autor: Oliver Scherbl)

Gegeben ist ein Stern, dessen Position und Größe durch Schieberegler verändert werden kann (siehe Abb.95). Die SchülerInnen sollen die Schieberegler verstellen und die Auswirkung auf die Figur beobachten. Was passiert mit der Figur, wenn die Skalierungsfaktoren negative Werte annehmen?

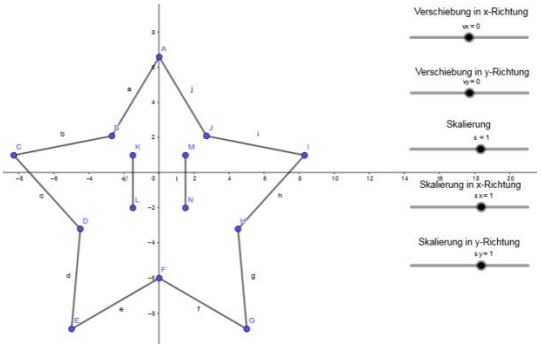

Abb.95: Powerstar aus Super Mario

Erklärung:

Zur Veränderung der Position werden alle Punkte mit dem Vektor $(vx|vy)$ addiert, dessen Koordinaten durch die Schieberegler bestimmt werden. Zur Veränderung der Größe werden alle Punkte mit dem Faktor s multipliziert. Ist $s = -1$, wird die Figur an beiden Achsen gespiegelt. Zur Skalierung in eine bestimmte Richtung werden die entsprechenden Koordinaten der Punkte mit sx beziehungsweise sy multipliziert.

Aufgabe:

Zeichne in GeoGebra ein beliebiges Dreieck durch drei Punkte. Erstelle Schieberegler für die Variablen a und b (Werte von -5 bis 5). Multipliziere die x-Koordinaten deiner Punkte mit a und die y-Koordinaten mit b. Versuche nun mit den Schiebereglern die Figur an der x-Achse zu spiegeln. Welche Variable musstest du verändern und auf welchen Wert? Vergleiche die Koordinaten der Punkte vor und nach der Spiegelung.

Lösung:

Variable b muss -1 sein. Die gespiegelten Punkte müssen gleich weit von der x-Achse entfernt sein, sich aber auf der anderen Seite befinden. Dazu muss das Vorzeichen der y-Koordinaten geändert werden.

Themen, die in diesem Beispiel zum Einsatz kommen: Koordinatensystem, Vektoraddition, Multiplikation eines Punktes mit einem Skalar

Koordinaten

In Videospielen beruht alles auf Koordinaten. Jede Figur und jedes Objekt wird durch Koordinaten in der Spielwelt platziert. In 2D-Spielen besteht alles aus Bildern, die am Bildschirm hin- und her geschoben, gestreckt und gedreht werden. Durch Koordinaten wird die Position dieser Bilder bestimmt und wohin sie sich bewegen sollen. Ein anschauliches Beispiel hierfür ist *UbiArt*, ein Programm der Spielentwickler *Ubisoft* zur einfachen Erstellung von 2D-Spielen (siehe Abb.96). **[Video 1]**

Abb.96:zusammengesetzte Spielfigur

In 3D-Spielen besteht jede Figur aus *Polygonen*, das sind vieleckige Formen, die in einem Stück gezeichnet werden können. Bei wenigen Polygonen bekommt man eine sehr eckige Figur, die aussieht wie ein Papiermodell (siehe Abb.97). Je mehr Polygone verwendet werden, umso glatter und detaillierter kann eine Figur werden. Jedes Polygon wird wiederum durch seine Eckpunkte beschrieben. Verschiebt man einzelne Eckpunkte, verformt man die dazugehörigen Polygone. So lassen sich beispielsweise Gesichtsausdrücke modellieren (siehe Abb 98). **[Video 2]**

Abb.97: Papiemodell Fuchs

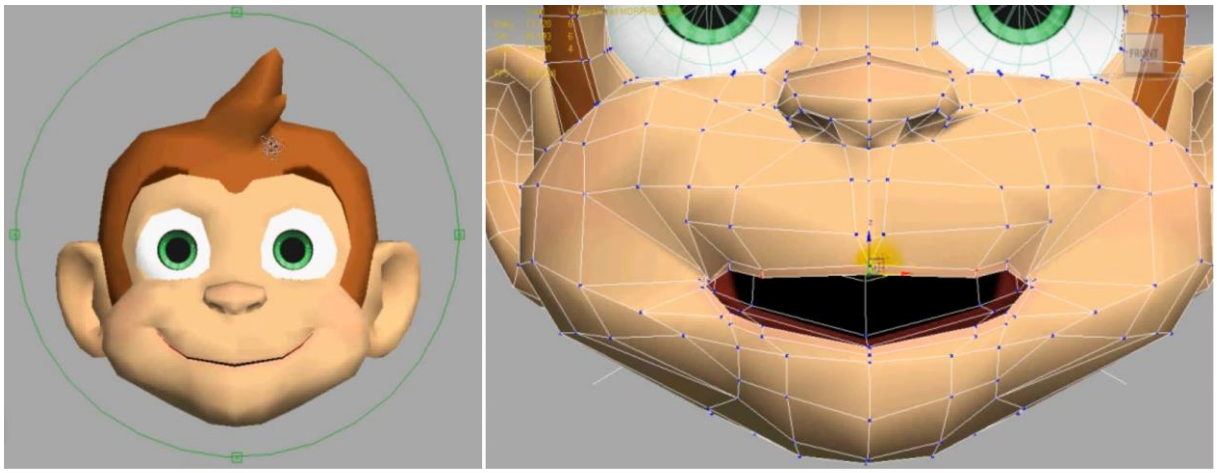

Abb.98: Gesichtsausdruck modellieren

Vektorrechnung

Vektoren können eingesetzt werden, um Objekte in Spielen zu verschieben. Dazu muss nur auf jeden Punkt des Objekts der gleiche Vektor angewandt werden, um die neuen Ortskoordinaten zu erhalten. Vektoren können auch eingesetzt werden, um den Abstand zwischen zwei Punkten beziehungsweise Objekten zu finden. Man nehme ein Rennspiel, bei dem man angeben will, welches Auto an der ersten Position ist. Dazu sind nur die Vektoren vom Ziel zum jeweiligen Auto zu berechnen und die Längen zu vergleichen (siehe Abb.99). **[Video 3]**

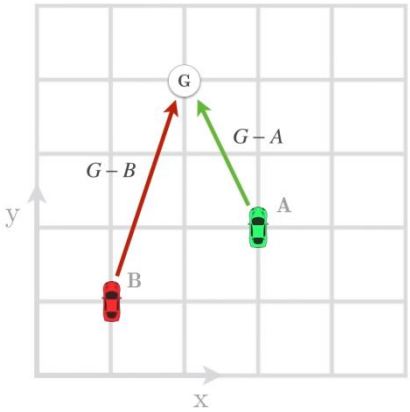

Abb.99: Vektoren im Rennspiel

Sichtbarkeit

GeoGebra Link: https://ggbm.at/NzYp6KPA (Autor: Oliver Scherbl)

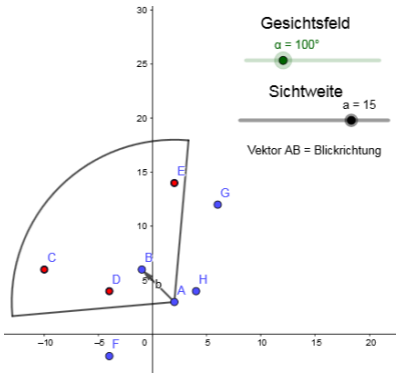

Abb.100: Sichtbarkeit

In vielen Spielen reagieren Feinde erst auf den Spieler, wenn er sich in einem bestimmten Bereich vor diesem befindet. In dieser Aufgabe steht im Punkt A eine Wache. Der Kreissektor ist der Bereich, den die Wache sieht (siehe Abb.100). Der Bereich ist durch die Sichtweite a, die Blickrichtung $\overrightarrow{AB}$ und den Winkel des Gesichtsfeldes α bestimmt. Jeder Punkt in diesem Bereich ist für die Wache sichtbar und wird rot dargestellt. Die SchülerInnen können Sichtweite und Gesichtsfeld, sowie Blickrichtung und Position aller Punkte verändern. Was kann man verändern, damit alle Punkte rot sind? Welche zwei Bedingungen müssen erfüllt sein, damit ein Punkt im sichtbaren Bereich liegt?

Erklärung:

Zur Überprüfung, ob beispielsweise Punkt C sichtbar ist, wird zum einen überprüft, ob sein Abstand zum Punkt A kleiner ist als die Sichtweite. Zum anderen wird der Winkel CAB mit $\frac{\alpha}{2}$ (der Winkel zwischen Blickrichtung und Rand des Kreissektors) verglichen. Ist der Abstand von C zu A kleiner als a und ist CAB kleiner als $\frac{\alpha}{2}$, dann ist der Punkt C sichtbar.

Aufgabe:

Gegeben ist ein Wachmann, der an der Position $A = (-3|8)$ steht und in die Richtung $\overrightarrow{AB}$ blickt mit $B = (-1|10)$. Sein Gesichtsfeld beträgt 110° und die Sichtweite ist 10. Ist Punkt $C = (3|9)$ für den Wachmann sichtbar? Berechne die Länge der Strecke AC und den Winkel CAB.

Lösung:

1) Wir konstruieren den Vektor $\overrightarrow{AC}$, berechnen dessen Betrag und vergleichen ihn mit der Sichtweite 10:

$$\overrightarrow{AC} = (6|1),\ \left|\overrightarrow{AC}\right| \approx \mathbf{6,08} < 10$$

2) Mit dem Skalarprodukt berechnen wir den Winkel zwischen $\overrightarrow{AC}$ und $\overrightarrow{AB}$ und vergleichen ihn mit 55° (das halbe Gesichtsfeld):

$$\overrightarrow{AC} = (6|1),\ \overrightarrow{AB} = (2|2)$$

$$\varphi = \cos\left(\frac{\overrightarrow{AB}\cdot\overrightarrow{AC}}{\left|\overrightarrow{AB}\right|\cdot\left|\overrightarrow{AC}\right|}\right) = \cos\left(\frac{14}{\sqrt{37}\cdot\sqrt{8}}\right) \approx \mathbf{35,5}° < 55°$$

Punkt C ist somit für den Wachmann sichtbar.

Themen, die in diesem Beispiel zum Einsatz kommen: Skalarprodukt, Beträge von Vektoren

Skalarprodukt

In Videospielen müssen Berechnungen sehr schnell ablaufen. Jede Sekunde müssen bis zu 60 Bilder generiert werden. Dabei sind genaue Werte oft nicht wichtig. Bei der Ermittlung von sichtbaren Objekten ist das der Fall. Jedes Objekt in der virtuellen Welt zu zeichnen, selbst wenn es am Bildschirm nicht zu sehen ist, ist unnötige Verschwendung von Rechenleistung und würde das Spiel verlangsamen. Mit dem Skalarprodukt lässt sich aber schnell überprüfen, ob ein Objekt sichtbar ist. Dazu zeichnet man einen Vektor $\overrightarrow{CO}$ von der Kamera zum Objekt, bildet den zugehörigen Einheitsvektor und bildet das Skalaprodukt mit der normierten Blickrichtung $\vec{v}$. Anschließend vergleicht man das Resultat mit dem Kosinus des (halben) Sichtwinkels. Ist das Skalarprodukt kleiner, dann liegt das Objekt innerhalb und ist sichtbar (siehe Abb.102). **[Video 4]**

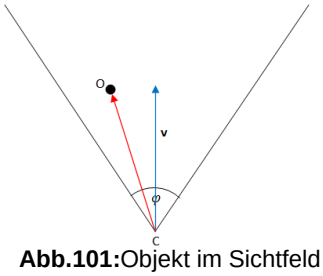

Abb.101: Objekt im Sichtfeld

Eine weitere Anwendung für das Skalarprodukt findet sich im Rennspiel *Mario Kart*. Hier gibt es Plattformen, bei denen die Fahrtgeschwindigkeit kurzzeitig erhöht wird (*Boost*) (siehe Abb.102). Der Winkel, in dem man über die Plattform fährt, gibt an, wie groß der Boost ist. Fährt man parallel über die Plattform, ist die Beschleunigung maximal, normal dazu ist die Beschleunigung null, und dazwischen erhält man nur einen Teilboost. Fährt man entgegen der Fahrtrichtung, erhält man sogar eine negative Beschleunigung und wird abgebremst.

Abb.102: Geschwindigkeits-Boost

Orientierung

GeoGebra Link: https://ggbm.at/uaycjCK9 (Autor: Oliver Scherbl)

Im Spiel *Super Mario Galaxy* kann *Mario* mit einem Weitsprung einmal um einen ganzen Planeten (von wenigen Meter Durchmesser) herumspringen. Dabei ist Mario zu jedem Zeitpunkt senkrecht zu seinem Untergrund ausgerichtet. In dieser GeoGebra Aufgabe können die SchülerInnen Marios Position und den Untergrund, der durch die Strecke AB dargestellt wird, verändern. Marios Ausrichtung wird durch den Vektor $\vec{r}$ beschrieben (siehe Abb.103). Welche Beziehung

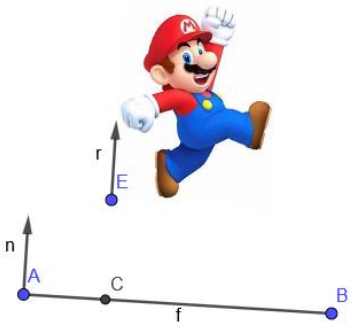

Abb.103: Marios Orientierung

besteht zwischen Vektor $\vec{r}$ und Vektor $\vec{n}$. Was ist $\vec{n}$ zur Strecke AB?

Erklärung:

Vektor $\vec{r}$ ist gleich $\vec{n}$. Der Vektor $\vec{n}$ ist ein Normalvektor der Strecke AB. Indem Marios Ausrichtung durch den Normalvektor des Untergrunds beschrieben wird, ist er immer senkrecht zu diesem ausgerichtet. Sein Ausrichtungsvektor muss nicht gleich dem Normalvektor, aber zumindest parallel zu diesem sein.

Aufgabe:

1) Der Untergrund ist durch eine Gerade g in Parameterdarstellung gegeben. Zeichne in GeoGebra die Gerade und ihre Normalvektoren, die gleich lang sind wie ihr Richtungsvektor. Worauf ist zu achten, wenn Marios Ausrichtung durch einen Normalvektor dieser Gerade beschrieben werden soll?

$$g: X = \begin{pmatrix} -3 \\ 5 \end{pmatrix} + t \cdot \begin{pmatrix} 7 \\ -2 \end{pmatrix}$$

Lösung:

Die Normalvektoren können aus dem Richtungsvektor der Parameterform ermittelt werden: $\overrightarrow{n_1} = \begin{pmatrix} 2 \\ 7 \end{pmatrix}$ und $\overrightarrow{n_2} = \begin{pmatrix} -2 \\ -7 \end{pmatrix}$. Sie zeigen in entgegengesetzte Richtungen. Für Marios Ausrichtung nimmt man am besten den Normalvektor, der in die gewünschte Richtung zeigt.

2) Mario befindet sich an der Position $A = (3|8)$. Berechne ohne GeoGebra seinen Abstand vom Untergrund. Einheiten sind in Meter angegeben.

Lösung:

Hier muss der Abstand von einem Punkt zu einer Geraden berechnet werden. Der Abstand kann durch Projektion des Vektors $\overrightarrow{PA}$ auf den normierten Normalvektor der Gerade berechnet werden. P ist der Punkt $\begin{pmatrix} -3 \\ 5 \end{pmatrix}$ der Geraden g.

$$d(A, g) = \left| \overrightarrow{PA} \cdot \frac{\overrightarrow{n_1}}{|\overrightarrow{n_1}|} \right| \approx \mathbf{4,6}$$

Themen, die in diesem Beispiel zum Einsatz kommen: Parameterdarstellung der Geraden, Normalvektor, Einheitsvektor, Abstand von Punkt und Gerade

Normalvektor

Mit dem Normalvektor kann auch die Neigung eines Untergrunds und die Reaktion einer Figur bestimmt werden. Beschreibt man die Figur durch einen senkrechten Vektor und vergleicht diesen mit dem Normalvektor des Untergrunds, kann durch das Skalarprodukt die Neigung ermittelt werden. Nun kann in einem Spiel festlegt werden, dass die Figur ab einer gewissen Neigung die Schräge hinunterrutschen soll (siehe Abb. 104).

Abb.104: Verschiedene Untergründe

Geraden

Geraden werden eingesetzt, um Bewegungen zu beschreiben und Abstände zu ermitteln. Bei der Raycasting-Methode geht ein Strahl von einem Objekt aus. Trifft dieser auf ein Objekt, beispielsweise eine Wand, wird der Schnittpunkt ermittelt und der Abstand berechnet. In Spielen mit unebenem Untergrund kann durch Überprüfung des Abstands die Position einer Spielfigur immer korrekt angepasst werden. Ein besonderes Beispiel stellt das Spiel *Super Mario Galaxy* dar. Darin bestehen die meisten Level aus "Planeten" mit nur einigen Metern im Durchmesser (siehe Abb.105). Mario kann sich frei über diese Planeten bewegen und bleibt immer senkrecht zum Untergrund ausgerichtet. Das Spiel erreicht das, indem von Mario ein Strahl senkrecht nach unten ausgeht, der mit einem Normalvektor des Untergrunds über das Skalarprodukt verglichen wird. Sind die Vektoren nicht parallel, dann wird Mario neu ausgerichtet. **[Video 5]**

Abb.105: Super Mario Galaxy (2007)

Sichtbarkeit in 3D

GeoGebra Link: https://ggbm.at/dyPNJQqZ (Autor: Oliver Scherbl)

In Videospielen werden Flächen, die für die Kamera nicht sichtbar sind, ausgeblendet, um Rechenleistung zu sparen. Gegeben ist ein Quader und die Blickrichtung der Kamera $\overrightarrow{MN}$. Die sichtbaren Flächen werden grün dargestellt (siehe Abb.106). Die SchülerInnen sollen den Vektor verändern und nachsehen, welche Seiten sichtbar sind. Zur Überprüfung sollen sie den Quader dann durch die Blickrichtung $\overrightarrow{MN}$ betrachten. Dazu müssen sie die Welt so drehen, dass Punkt M Punkt N überdeckt.

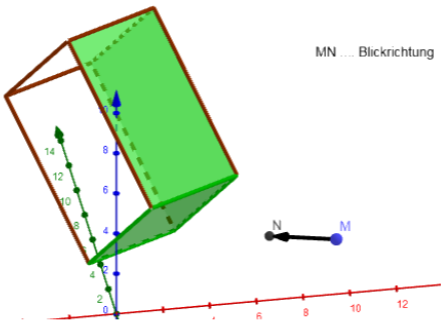

Abb.106: Sichtbarkeit in 3D

Erklärung:

Um festzustellen, welche Flächen sichtbar sind, wird der Winkel zwischen Blickrichtung der Kamera und den Normalvektoren der Oberflächen verglichen. Zeigt ein Normalvektor zur Kamera, dann ist die zugehörige Fläche sichtbar, zeigt er weg von der Kamera, dann ist die Fläche nicht sichtbar.

Aufgabe:

1) Zeichne in GeoGebra 3D einen Würfel ABCDEFGH. Zeichne im Quadrat ABCD die Vektoren $\overrightarrow{AB}$ und $\overrightarrow{AC}$ ein. Berechne die Kreuzprodukte $\overrightarrow{AB} \times \overrightarrow{AC}$ und $\overrightarrow{AC} \times \overrightarrow{AB}$ (Befehl: Kreuzprodukt (<Vektor>, <Vektor>)) Wie sind die Kreuzprodukte im Bezug zum Quadrat ABCD orientiert? Wie unterscheiden sie sich?

Lösung:

Die Kreuzprodukte sind Normalvektoren des Quadrats. Die Reihenfolge, in der man das Kreuzprodukt bildet, beeinflusst die Richtung des Normalvektors. Ein Normalvektor zeigt in den Würfel, der andere zeigt aus dem Würfel hinaus.

2) Gegeben ist die Blickrichtung $\vec{r} = (0|-3|-4)$ einer Kamera. Zeichne den Vektor und überprüfe, ob die Fläche ABCD sichtbar wäre. Berechne den Winkel zwischen $\vec{r}$ und dem Normalvektor von ABCD, der aus dem Würfel hinaus zeigt. Dreh nun den Würfel, damit die Seite ABCD zur Kamera zeigt. In welchem Bereich kann der Winkel liegen?

Lösung:

Ist der Winkel zwischen Normalvektor von ABCD und $\vec{r}$ kleiner als 90° dann zeigt die Seite zur Kamera.

Themen, die in diesem Beispiel zum Einsatz kommen: Kreuzprodukt, Normalvektor der Ebene, Skalarprodukt

Kreuzprodukt

Das Kreuzprodukt findet in allen 3D Spielen Anwendung. Mit ihm wird eine Normale für jedes Polygon und jede Fläche bestimmt und damit ihre Außenseite. Das ist wichtig, wenn festgestellt werden soll, ob eine Fläche für die Kamera sichtbar ist. Ein Polygon, das von der Kamera weg zeigt, gehört meist zu der Rückseite eines Objekts und wird ausgeblendet, um Rechenleistung zu sparen. Zur Überprüfung, in welchem Winkel das Polygon zur Kamera zeigt, wird das Skalarprodukt von Normalvektor und Richtungsvektor der Kamera gebildet. **[Video 6]**

Das Kreuzprodukt kommt auch bei Lichteffekten zum Einsatz. Je steiler Licht auf eine Fläche trifft, umso stärker wird sie beleuchtet. Eine Fläche, deren Normalvektor parallel zum Lichtvektor ist, hat maximale Helligkeit, während eine Fläche im rechten Winkel (und darüber) zur Lichtquelle kein Licht abbekommt. Da nur die Normalvektoren der Oberfläche benötigt werden, um Lichteffekte zu berechnen, werden in Spielen oft an einem genauen Modell die Normalvektoren für jeden Punkt der Oberfläche berechnet und diese auf einer flachen Textur gespeichert. So kann an glatten Oberflächen die Illusion von Rauheit erzeugt werden (siehe Abb.107). **[Video 7], [Video 8]**

Abb.107: Gleiche Kugeln mit unterschiedlichen Texturen

Kollision

GeoGebra Link: https://ggbm.at/sVuQrfwy (Autor: Oliver Scherbl)

Das Spiel *Splatoon* ist ein Shooter, bei dem man Gegner mit Farbe beschießt. In manchen Leveln gibt es Ballons, zu denen man teleportiert wird, wenn man sie trifft. Ein solcher Ballon ist in GeoGebra 3D gegeben (siehe Abb.108). Der Strahl ist die Schusslinie und kann durch die Schieberegler gesteuert werden. Wenn der Strahl die Kugel schneidet, leuchtet sie grün. Ab welchem Abstand zum Kugelmittelpunkt schneidet der Strahl die Kugel?

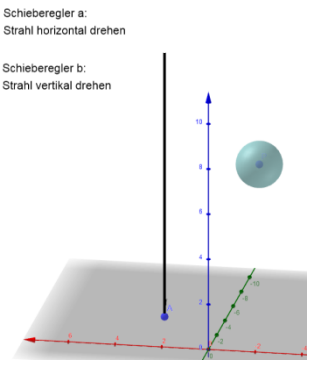

Abb.108: Kollision

Erklärung:

Der Strahl schneidet die Kugel, sobald sein Abstand zum Kugelmittelpunkt kleiner ist, als der Radius der Kugel.

Aufgabe:

Ein Ballon hat seinen Mittelpunkt an der Position $P = (-2|-1|8)$ und hat einen Radius von einem Meter. Die Schusslinie ist gegeben durch die Gerade $g: X = (1|5|3) + t \cdot (-1|-2|2)$. Berechne den Abstand von P zur Geraden g. Wird der Ballon getroffen?

Lösung:

Der Abstand kann mit der Formel $d(P, g) = \left|\overrightarrow{AP} \times \overrightarrow{g_0}\right|$ berechnet werden. Ist der Abstand kleiner als 1, dann wird der Ballon getroffen. Mit dem Punkt $(1|5|3)$ der Geraden berechnen wir $\overrightarrow{AP} = (-3|-6|5)$. Der normierte Richtungsvektor der Geraden ist $\overrightarrow{g_0} = (-1|-2|2)/3$. Der Abstand ist $d(P, g) \approx \mathbf{0,75} < 1$. Der Ballon wird getroffen.

Themen, die in diesem Beispiel zum Einsatz kommen: Parameterdarstellung der Geraden in $\mathbb{R}^3$, Kreuzprodukt, Einheitsvektor, Abstand von Punkt und Geraden in $\mathbb{R}^3$

Schatten

GeoGebra Link: https://ggbm.at/UCybgsGY (Autor: Oliver Scherbl)

Gegeben ist ein Quadrat ABCD. Im Punkt S befindet sich eine punktförmige Lichtquelle, die in alle Richtungen abstrahlt. Das Quadrat wirft einen Schatten auf die Ebene ε (siehe Abb.109). Die SchülerInnen können den Punkt S verschieben. Wie werden die Eckpunkte A', B', C' und D' des Schattens auf der Ebene erzeugt? Warum verschwinden sie, wenn der Punkt S zwischen Quadrat und Ebene geschoben oder zu weit nach unten gesetzt wird?

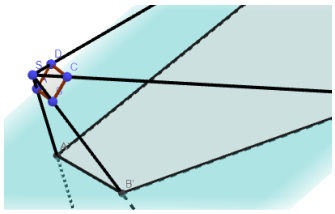

Abb.109: Schatten

Erklärung:

Vom Punkt S zu jedem Eckpunkt wird je ein Strahl gezeichnet. Die Schnittpunkte der Strahlen mit der Ebene sind die Eckpunkte des Schattens. Liegt S zwischen Quadrat und Ebene, dann zeigen die Strahlen von der Ebene weg. Liegt S zu tief, dann gehen die Strahlen an der Ebene vorbei.

Aufgabe:

Gegeben ist $A = (0|15|11)$, die Lichtquelle $S = (14|8|18)$ und die Ebene $\varepsilon: 2x - 3y + 7z = -20$. Berechne die Koordinaten des projizierten Punktes A' auf der Ebene.

Lösung:

Zuerst müssen wir mit den Punkten S und A eine Gerade aufstellen:

$$g: X = S + t \cdot (A - S) = (14|8|18) + t \cdot (-14|7|-7)$$

Die Gerade wird mit der Ebene geschnitten, um den projizierten Punkt zu finden:

$$\varepsilon: 2(14 - 14 \cdot t) - 3(8 + 7 \cdot t) + 7(18 - 7 \cdot t) = -20$$

$$28 - 28t - 24 - 21t + 126 - 49t = -20$$

$$130 - 98t = -20 \Rightarrow t \approx 1{,}531$$

In die Parameterform der Geraden rückeingesetzt erhält man den Punkt:

$$A' = (14|8|18) + 1{,}531 \cdot (-14|7|-7) = (\mathbf{-7{,}43}|\mathbf{18{,}71}|\mathbf{7{,}29})$$

Themen, die in diesem Beispiel zum Einsatz kommen: Parameterdarstellung der Geraden in $\mathbb{R}^3$, Ebene, Schnitt von Geraden und Ebene

Lagebeziehung

Abb.110: Hitbox-Dissonanz

Da virtuelle Objekte nur Daten sind, die auf einem Bildschirm dargestellt werden, muss mathematisch getestet werden, ob Objekte kollidieren, damit eine Reaktion dargestellt werden kann. Bewegungen können durch Geraden beschrieben werden. So kann der Computer durch den Ebene-Gerade-Schnitttest überprüfen, ob ein Objekt mit einer Wand zusammenstößt, und wenn ja, wo das passiert. Kennt man den Berührpunkt, muss der Computer herausfinden, welche Stelle des Objekts die Wand berührt. Da Objekte in modernen Spielen aus mehr als 10000 Polygonen bestehen können, wäre der Test sehr aufwendig. Zur Vereinfachung werden die Objekte mit einfachen Formen wie Quader, Kugeln und Zylinder umgeben, mit denen die Kollision getestet wird. Bei einem Quader als Hitbox müssen nur Ebene-Ebene-Lagebeziehung überprüft werden. Da diese Hitboxen nicht perfekt am Objekt anliegen, kann es passieren, dass eine Figur von Angriffen getroffen werden, die eigentlich knapp vorbeigehen sollten (siehe Abb.110). **[Video 9]**

4.2 Physikunterricht 5. & 6. Klasse

Auch im Physikunterricht können Situationen aus Videospielen für Übungsbeispiele herangezogen werden. Viele Spiele haben beispielsweise eine Fallbeschleunigung einprogrammiert, die sich von der der Erde unterscheidet. So könnten Beispiele gegeben werden, bei denen aus der Fallhöhe und der Fallzeit eines Objekts die Fallbeschleunigung in einem Spiel berechnet werden soll.

Physik in Spielen

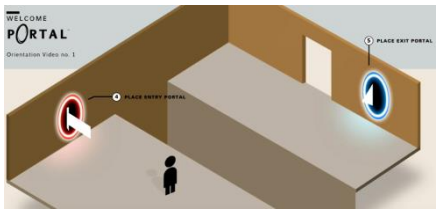

Abb.111: Abkürzung in Portal

Physikalische Formeln kommen in Spielen zum Einsatz, um die Bewegung und Wechselwirkung von Objekten realistisch erscheinen zu lassen. In manchen Spielen ist die Physik sogar der Hauptbestandteil des eigentlichen Spiels. In dem Spiel *Portal* ist das Ziel, Rätsel zu lösen und Hindernisse zu überwinden. Als Hilfsmittel hat man eine Waffe, die Wurmlöcher erzeugt (siehe Abb.111) **[Video10].** In *Legend of Zelda: Breath of the Wild* kann durch eine Fähigkeit die Zeit von Objekten angehalten werden. Während die Zeit gestoppt wird, kann man auf ein Objekt schlagen und eine große Impulsänderung aufbauen, die auf das Objekt wirkt, sobald die Zeit wieder fließt **[Video 11].** Es gibt auch YouTube-Kanäle, die Inhalte von Spielen auf wissenschaftliche Art betrachtet, zum Beispiel *The Game Theorists* **[Video 12].**

5. Anhang

5.1 Ergänzende Videos:

Video 1: How Rayman Legends Is Made!

https://www.youtube.com/watch?v=y-chi097uV4

Video 2: How Did They Do That - Jak& Daxter's Facial Animation

https://www.youtube.com/watch?v=3PUg3ZRUoiE

Video 3: Game Math Theory - VECTORS

https://www.youtube.com/watch?v=wXI9_olSrqo

Video 4: Horizon Zero Dawn – The making of the game (2017)

https://youtu.be/A0eaGRcdwpo?t=1095

Video 5: How Did They Do That - Mario Galaxy's Gravity

https://www.youtube.com/watch?v=vALtyrp87mI

Video 6: Off Camera Secrets | Super Mario Galaxy - Boundary Break
https://www.youtube.com/watch?v=hgnrfpP3J-c

Video 7: Shader Fundamentals - Normal Mapping

https://www.youtube.com/watch?v=6_-NNKc4lrk

Video 8: Shader Fundamentals - Image Based Lighting

https://www.youtube.com/watch?v=xWCZiksqCGA

Video 9: How Collisions Work in Games

https://www.youtube.com/watch?v=z7xMIRzIDpU

Video 10: How Did They Do That - Portal's Portals

https://www.youtube.com/watch?v=yKxMIIqMUAY

Video 11: BotW: Stasis to Temple of Time - Boulder Strat

https://www.youtube.com/watch?v=7Ekbx_tSfS4

Video 12: How Mario Can Fly – SOLVED! | The SCIENCE!... of Mario

https://www.youtube.com/watch?v=Paf4blGVkiU

5.2 Screenshots der GeoGebra Aufgaben:

Geometrie Transformation:

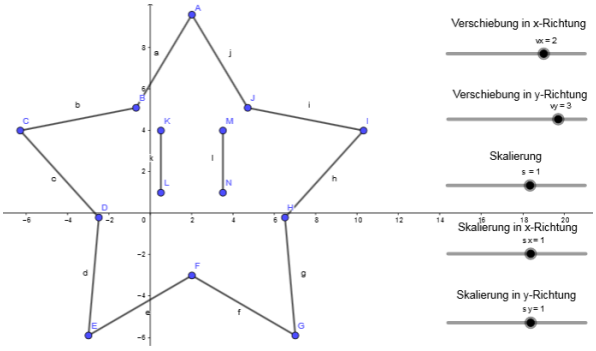

Abb.112: Verschiebung um (2|3)

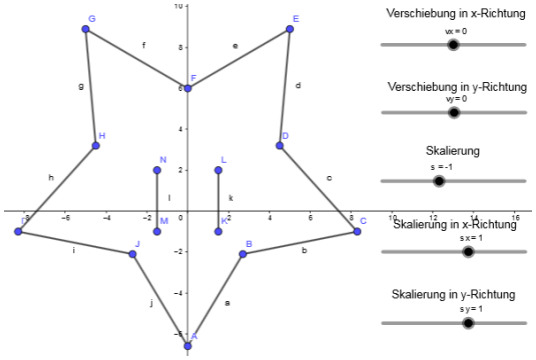

Abb.113: Skalierungsfaktor $s = -1$

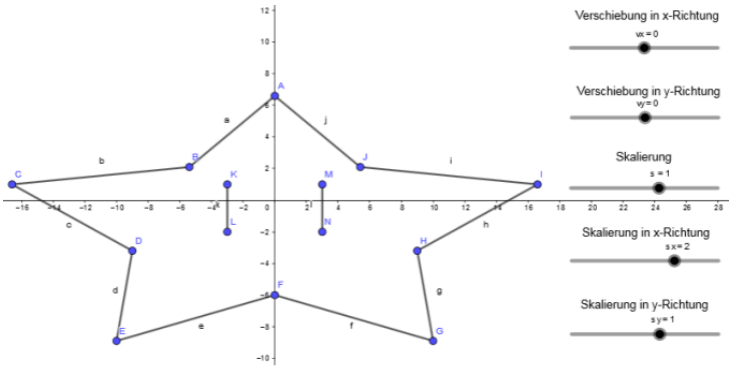

Abb.114: Skalierungsfaktor $sx = 2$

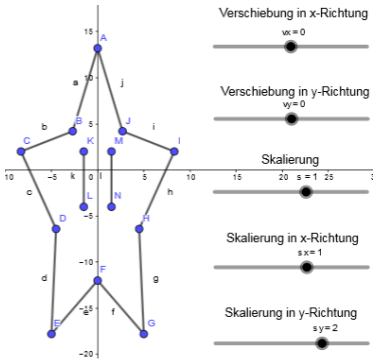

Abb.115: Skalierungsfaktor $sy = 2$

Sichtbarkeit:

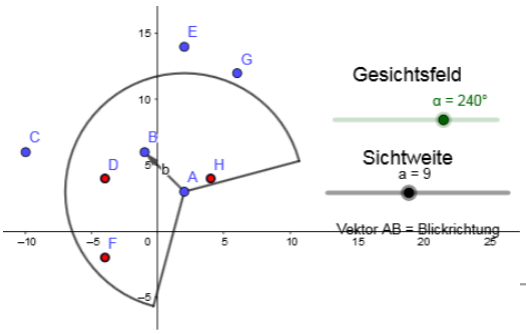

Abb.116: Größeres Gesichtsfeld

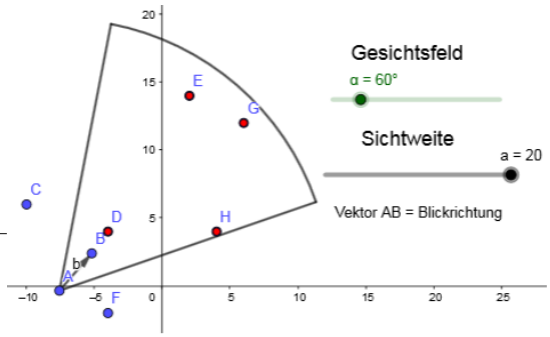

Abb.117: Veränderte Blickrichtung

Sichtbarkeit 3D:

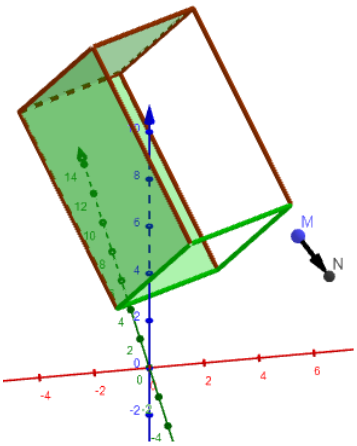

Abb.118: Veränderte Blickrichtung $\overrightarrow{MN}$

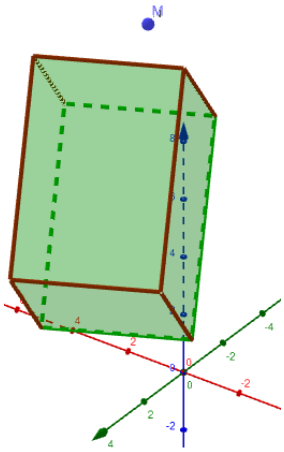

Abb.119: Blickrichtung $\overrightarrow{MN}$ nachverfolgen

Kollision:

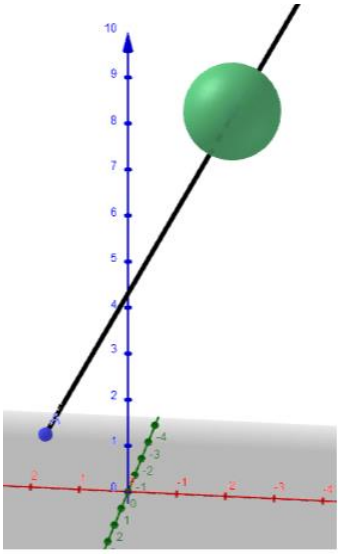

Abb.120: Strahl trifft Kugel

Schatten:

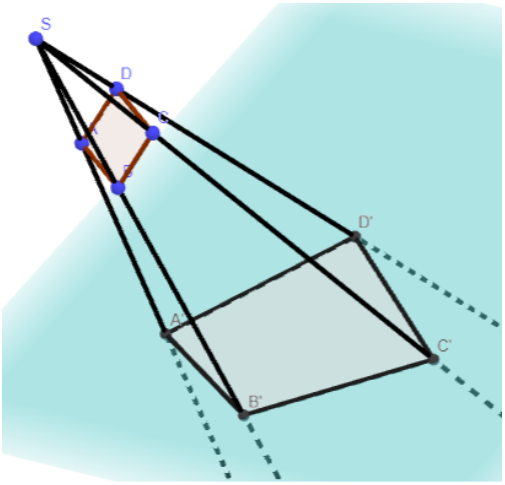

Abb.121: Lichtquelle weiter entfernt

5.3 Zusammenfassung

Diese Diplomarbeit befasst sich mit den mathematischen Methoden zur Darstellung von Videospielen und deren Bezug zum Lehrstoff des Mathematikunterrichts in der Sekundarstufe II.

Im ersten Teil der Arbeit wird ein kurzer Überblick über die Entwicklung der 3D Darstellung gegeben und der Prozess erläutert, durch den virtuelle Objekte auf einem Bildschirm abgebildet werden.

Darauf folgt eine Zusammenfassung mathematischer und physikalischer Grundlagen, die in der Darstellung von Videospielen Verwendung finden.

Anschließend werden Methoden erklärt, durch die verschiedenste Probleme bei der Darstellung virtueller Objekte gelöst werden.

Abschließend werden diese Methoden auf Themen des Lehrstoffes bezogen und in vereinfachter Form mit GeoGebra-Aufgaben dargestellt.

5.4 Abstract

This diploma thesis is dealing with the mathematical methods for displaying video games and their correlation to the curriculum of the secondary level II.

The first part is an overview of the historical development of 3D games und explains the process of displaying virtual objects on a screen.

Next is a summary of basic math and physics necessary for displaying video games.

It follows an explanation of methods, which are used to solve problems of displaying virtual objects.

Lastly, those methods are applied to the curriculum for mathematics and are replicated in GeoGebra.

Literaturverzeichnis

Bücher

- Akenine-Möller, T., Haines, E., & Hoffman, N. (2008). *Real-time rendering*. CRC Press.
- Tremblay, C. (2004). *Mathematics for game developers*. Elsevier.
- Phong, B. T. (1975). *Illumination for computer generated pictures*. *Communications of the ACM*, 18(6), 311-317.
- Bourg, D. M., &Bywalec, B. (2013). *Physics for Game Developers: Science, math, and code for realistic effects*. " O'Reilly Media, Inc.".
- Wolf, M. J. (Ed.). (2008). *The video game explosion: a history from PONG to Playstation and beyond*. ABC-CLIO
- Bleier, Lindenberg, Lindner, Süss-Stepancik (2017). *Dimensionen, Mathematik 5*. E. DORNER
- Bleier, Lindenberg, Lindner, Süss-Stepancik (2017). *Dimensionen, Mathematik 6*. E. DORNER
- Timischl, Kaiser (2007). *Ingenieur-Mathematik 3*. E.DORNER
- Embacher, F (2010). *Elemente der theoretischen Physik, Band 1: Klassische Mechanik und Spezielle Relativitätstheorie. Eine Einführung für das Lehramts- und Bachelorstudium.* VIEWEG+TEUBNER

Zeitschriften

- Lyon, R. F. (2006, January). A brief history of'pixel'. In *Digital Photography* (p. 606901).
- *Watkinson, M. (2009). Real Time Character Animation: A Generic Approach to Ragdoll Physics. MSc. diss., Coventry University.*
- Next Generation. No. 15. (1996, March)*. Imagine Media.*
- Becker, S. (1996, February). Virtuelle Welten mit der Raycasting-Technik darstellen. In *c't*

Onlinequellen

- HompageBrookhaven National Laboratory. Online im Internet: https://www.bnl.gov/about/history/firstvideo.php [Zugriff am 2.7.2017]
- Marktanalytiker für die Videospielbranche. Online im Internet:https://newzoo.com/insights/rankings/top-100-countries-by-game-revenues/ [Zugriff am 2.7.2017]
- Wikipedia Seite zum Thema Full-HD https://de.wikipedia.org/wiki/Full_HD [Zugriff am 3.7.2017]
- Polygon Anzahlen. Online im Internet: http://polycount.com/discussion/141061/polycounts-in-next-gen-games-thread [Zugriff am 9.7.2017]
- Lehrplansammlung des Bundesministeriums für Bildung: https://www.bmb.gv.at/schulen/unterricht/lp/lp_ahs_oberstufe.html [Zugriff am 12.7.2017]
- Bitmap Definition. Online im Internet: http://foldoc.org/bitmap+display [Zugriff am 14.2.2018]
- Anwendung von Sprites. Online im Internet: http://prog21.dadgum.com/181.html [Zugriff am 14.2.2018]
- Textblatt zur geometrischen Transformation in der Computergrafik: https://www.cg.tuwien.ac.at/courses/CG1/textblaetter/02%20Geometrische%20Transformationen.pdf [Zugriff am 29.4.2018]

- Kugelkoordinaten: https://mo.mathematik.uni-stuttgart.de/inhalt/aussage/aussage441/ [Zugriff am 11.5.2018]

Abbildungsverzeichnis

1. http://www.tvovermind.com/wp-content/uploads/2017/04/super_mario_bros.0.png [Zugriff am 2.7.2017]
2. https://i.ytimg.com/vi/lY8hRjqyDR4/hqdefault.jpg [Zugriff am 2.7.2017]
3. http://www.dsogaming.com/wp-content/uploads/2015/08/caem.jpg [Zugriff am 2.7.2017]
4. http://i1-news.softpedia-static.com/images/news2/Angry-Birds-Update-Arrives-with-Brand-New-Levels-Free-2.jpg [Zugriff am 15.7.2017]
5. https://www.bnl.gov/about/history/firstvideo.php [Zugriff am 2.7.2017]
6. http://www.techradar.com/news/gaming/the-evolution-of-3d-games-700995[Zugriff am 27.2.2018]
7. http://www.retronintendoreviews.com/the-legend-of-zelda-nes-review/[Zugriff am 27.2.2018]
8. http://gamona-images.de/639935/20f40d73405bf860c699f6006dae8daf.jpg[Zugriff am 27.2.2018]
9. https://s3.amazonaws.com/media-p.slid.es/uploads/55786/images/1828075/mario-jump.png[Zugriff am 27.2.2018]
10. https://edge.alluremedia.com.au/m/k/2013/10/sf2-s3.jpg[Zugriff am 27.2.2018]
11. https://www.youtube.com/watch?v=qxM9pMEnJQ0[Zugriff am 27.2.2018]
12. https://www.arcade-museum.com/images/118/1181242138160.png[Zugriff am 27.2.2018]
13. https://www.youtube.com/watch?v=dzN2pgL0zeg[Zugriff am 27.2.2018]
14. https://lparchive.org/Final-Fantasy-VI-(by-Blastinus)/Update%2020/1-25062011_104208.png[Zugriff am 27.2.2018]
15. https://de.wikipedia.org/wiki/Raycasting[Zugriff am 27.2.2018]
16. https://de.wikipedia.org/wiki/Raycasting[Zugriff am 27.2.2018]
17. https://www.youtube.com/watch?v=qxM9pMEnJQ0[Zugriff am 27.2.2018]
18. http://gimmegimmegames.com/wp-content/uploads/2013/12/virtuafighter.jpg[Zugriff am 27.2.2018]
19. https://www.cs.auckland.ac.nz/research/groups/GG/images/weeklyimages/sourceimages/20110708-ImageOfWeek_MeshSudivision.jpg [Zugriff am 6.7.2017]
20. Akenine-Möller (2008), S. 13
21. Akenine-Möller (2008), S. 14
22. https://qph.ec.quoracdn.net/main-qimg-91f4b2e89d4b18e26ee364bb377c9a83-c [Zugriff am 9.7.2017]
23. https://commons.wikimedia.org/wiki/File:Portal_physics-3.svg[Zugriff am 27.2.2018]
24. http://www.ant-online.co.uk/Tutorials/RiggingGuide/Rig04.jpg [Zugriff am 19.7.2017]
25. Akenine-Möller (2008), S. 16
26. https://developer.tizen.org/sites/default/files/images/teapot_transformation.png [Zugriff am 15.7.2017]
27. Akenine-Möller (2008), S. 16
28. Akenine-Möller (2008), S. 19
29. Akenine-Möller (2008), S. 19
30. Akenine-Möller (2008), S. 20
31. Akenine-Möller (2008), S. 14

32. Akenine-Möller (2008), S. 21
33. https://derstandard.at/2000076183191/Dramatisches-Schach-Caruana-stoppt-Kramnik [Zugriff am 15.3.2018]
34. Dorner (2017), S. 268
35. https://upload.wikimedia.org/wikipedia/commons/thumb/6/69/Kugelkoord-def.svg/300px-Kugelkoord-def.svg.png [Zugriff am 28.4.2018]
36. Bleier 5 (2017), S.269
37. Bleier 5 (2017), S.277
38. Bleier 5 (2017), S.293
39. Bleier 5 (2017), S.326
40. Bleier 5 (2017), S.299
41. Bleier 6 (2017), S.245
42. Bleier 6 (2017), S.260
43. Bleier 6 (2017), S.266
44. Bleier 6 (2017), S.286
45. Bleier 6 (2017), S.288
46. Bleier 6 (2017), S.289
47. https://www.kotaku.com.au/2015/07/physics-puzzler-has-a-community-of-mad-geniuses/ [Zugriff am 27.2.2018]
48. Bourg (2013), S. 36
49. https://www.lernhelfer.de/schuelerlexikon/physik/artikel/waagerechter-wurf [Zugriff am 27.2.2018]
50. http://www.maschinenbau-wissen.de/skript3/mechanik/kinetik/263-schiefe-ebene [Zugriff am 27.2.2018]
51. Tremblay (2004), S. 222
52. Bourg (2013), S. 77
53. Bourg (2013), S. 62
54. Bourg (2013), S. 64
55. https://www.grund-wissen.de/physik/mechanik/schwingungen-und-wellen/harmonische-schwingungen.html [Zugriff am 27.2.2018]
56. http://me-lrt.de/dampfung-grundlagen [Zugriff am 27.2.2018]
57. Bourg (2013), S. 259
58. Bourg (2013), S. 256
59. https://gamingbolt.com/sea-of-thieves-behind-the-scenes-video-shows-off-npcs-and-lore [Zugriff am 27.2.2018]
60. Bourg (2013), S. 81
61. http://mathenexus.zum.de/html/geometrie/schnitt/SchneideEE.htm [Zugriff am 27.2.2018]
62. https://de.wikipedia.org/wiki/Schnittpunkt [Zugriff am 27.2.2018]
63. Tremblay (2004), S. 109
64. Oliver Scherbl
65. Tremblay (2004), S. 112
66. Oliver Scherbl
67. https://www.youtube.com/watch?v=7Bwj85TKItE [Zugriff am 27.2.2018]
68. Tremblay (2004), S. 263
69. Tremblay (2004), S. 264
70. Tremblay (2004), S. 267
71. Tremblay (2004), S. 272

72. https://www.youtube.com/watch?v=MV2rRcskn4Y [Zugriff am 27.2.2018]
73. Tremblay (2004), S. 130
74. Oliver Scherbl
75. Tremblay (2004), S. 133
76. Tremblay (2004), S. 135
77. https://www.cg.tuwien.ac.at/courses/CG1/textblaetter/02%20Geometrische%20Transformationen.pdf [Zugriff am 28.4.2018]
78. https://www.cg.tuwien.ac.at/courses/CG1/textblaetter/02%20Geometrische%20Transformationen.pdf [Zugriff am 28.4.2018]
79. Akenine-Möller (2008), S. 91
80. Akenine-Möller (2008), S. 93
81. https://de.wikipedia.org/wiki/Z-Buffer [Zugriff am 27.2.2018]
82. https://de.wikipedia.org/wiki/Z-Buffer [Zugriff am 27.2.2018]
83. https://knowledge.autodesk.com/support/3ds-max/troubleshooting/caas/sfdcarticles/sfdcarticles/Backface-Cull-Object-renders-transparent-when-facing-away-from-view.html [Zugriff am 27.2.2018]
84. https://kotaku.com/horizon-zero-dawn-uses-all-sorts-of-clever-tricks-to-lo-1794385026 [Zugriff am 27.2.2018]
85. https://stackoverflow.com/questions/44020521/orthographic-camera-is-not-occlusion-culling [Zugriff am 27.2.2018]
86. https://boards.fireden.net/v/thread/386466842/ [Zugriff am 27.2.2018]
87. https://www.youtube.com/watch?v=Nf-1vPVRY74 [Zugriff am 27.2.2018]
88. https://en.wikipedia.org/wiki/Phong_reflection_model [Zugriff am 27.2.2018]
89. https://i.warosu.org/data/vr/thumb/0035/41/1475633564352s.jpg [Zugriff am 27.2.2018]
90. https://www.youtube.com/watch?v=xWCZiksqCGA [Zugriff am 27.2.2018]
91. https://www.youtube.com/watch?v=6_-NNKc4lrk [Zugriff am 27.2.2018]
92. http://zelda.wikia.com/wiki/History_of_the_Legend_of_Zelda_series [Zugriff am 27.2.2018]
93. https://www.gamedev.net/articles/programming/graphics/the-theory-of-stencil-shadow-volumes-r1873/ [Zugriff am 27.2.2018]
94. Tremblay (2004), S. 470
95. https://ggbm.at/uAAmRQKY [Zugriff am 12.5.2018]
96. https://www.youtube.com/watch?v=y-chi097uV4 [Zugriff am 18.3.2018]
97. https://i.pinimg.com/474x/3b/42/42/3b42423ea0b1060c3511d69c2740e8cf--paper-craft-templates-papercraft.jpg [Zugriff am 15.3.2018]
98. https://www.youtube.com/watch?v=3PUg3ZRUoiE [Zugriff am 9.4.2018]
99. https://www.youtube.com/watch?v=wXI9_olSrqo [Zugriff am 9.4.2018]
100. https://ggbm.at/NzYp6KPA [Zugriff am 12.5.2018]
101. Oliver Scherbl
102. https://betterexplained.com/articles/vector-calculus-understanding-the-dot-product/ [Zugriff am 9.4.2018]
103. https://ggbm.at/uaycjCK9 [Zugriff am 12.5.2018]
104. https://www.youtube.com/watch?v=4PM6Jb3kCvY [Zugriff am 9.4.2018]
105. https://www.gamasutra.com/db_area/images/feature/3593/smga.jpg [Zugriff am 9.4.2018]
106. https://ggbm.at/dyPNJQqZ [Zugriff am 12.5.2018]
107. https://www.youtube.com/watch?v=xWCZiksqCGA [Zugriff am 9.4.2018]

108. https://ggbm.at/sVuQrfwy [Zugriff am 12.5.2018]
109. https://ggbm.at/UCybgsGY [Zugriff am 12.5.2018]
110. http://tvtropes.org/pmwiki/pmwiki.php/Main/HitboxDissonance [Zugriff am 9.4.2018]
111. https://www.youtube.com/watch?v=TluRVBhmf8w [Zugriff am 9.4.2018]
112. https://www.geogebra.org/m/uAAmRQKY [Zugriff am 17.5.2018]
113. https://www.geogebra.org/m/uAAmRQKY [Zugriff am 17.5.2018]
114. https://www.geogebra.org/m/uAAmRQKY [Zugriff am 17.5.2018]
115. https://www.geogebra.org/m/uAAmRQKY [Zugriff am 17.5.2018]
116. https://www.geogebra.org/m/NzYp6KPA [Zugriff am 17.5.2018]
117. https://www.geogebra.org/m/NzYp6KPA [Zugriff am 17.5.2018]
118. https://www.geogebra.org/m/dyPNJQqZ [Zugriff am 17.5.2018]
119. https://www.geogebra.org/m/dyPNJQqZ [Zugriff am 17.5.2018]
120. https://www.geogebra.org/m/sVuQrfwy [Zugriff am 17.5.2018]
121. https://www.geogebra.org/m/UCybgsGY [Zugriff am 17.5.2018]