3D Simulation

Projektarbeit

über die 5. und 6. Theoriephase
im Studiengang
Informationstechnik
an der
Berufsakademie Ravensburg
Außenstelle Friedrichshafen

vorgelegt von:
Max Schloss
Ansgar Wiechers
(TIT01-NS2, Gruppe 8)

Friedrichshafen, Juni 2004

Inhaltsverzeichnis

1 Aufgabenstellung

1.1 Aufgabenbeschreibung

Im Rahmen der Vorlesung „Grafische Datenverarbeitung“ soll im 5. und 6. Theoriesemester eine 3D-Simulation entwickelt werden. Dabei soll eine 3D-Szene berechnet und auf dem Bildschirm dargestellt werden. Eine wichtige Vorgabe ist, dass die Verwendung von 3D-Bibliotheken, wie z.B. OpenGL, DirectDraw oder Java3D, nicht gestattet ist.

Folgende Anforderungen werden an die 3D-Simulation gestellt:

- Momentaufnahme der Szene (3D-Sicht)
- Draufsicht auf die Szene
- Drei vorgegebene in der Szene platzierte Objekte
  (Kubus, Dreiecksäule und Pyramide)
- Steuerung durch direkte Eingabe von folgenden Parametern:
  - Betrachterposition (x-, y-, z-Koordinate)
  - Blickrichtung (Gierwinkel, Nickwinkel)
  - Öffnungswinkel (Zoom/Bildausschnitt)
- Steuerung ausgehend vom Betrachter
  (vor, zurück, links rechts, hoch, runter, Änderung der Blickrichtung)
- Realisierung zweier automatischer Fahrsequenzen:
  - Kreisbewegung
  - geradlinige Bewegung

1.2 Szene

Die Szene besteht aus einem ebenen, grünen Quadrat mit der Kantenlänge 300. Die Kanten verlaufen parallel zur x- und y-Koordinatenachse und der Mittelpunkt des Quadrats befindet sich bei den Koordinaten (100, 100, 0). Auf der Oberfläche befinden sich drei Objekte, die im nächsten Abschnitt detailliert beschrieben werden.

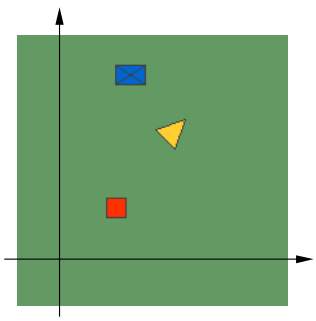


Abb. 1: Draufsicht der 3D-Szene

Die Objekte sind deckend zu zeichnen und verdecken einander, je nach Betrachtungswinkel. Ihre Flächen haben zur Verstärkung des 3D-Effekts unterschiedliche Farben. Alle Objekte sind nicht physisch und können vom Betrachter durchdrungen werden.

Der das Quadrat umgebende Raum ist leer, horizontal unbegrenzt und wird schwarz dargestellt. Nach unten ist die Szene durch die Ebene, in der das Quadrat liegt, begrenzt. Die Höhe 0 darf daher in der Szene niemals unterschritten werden.

Zu Beginn der Simulation steht der Betrachter im Koordinatenmittelpunkt und blickt in Richtung der positiven y-Achse.

1.3 Objekte

Auf der Grundfläche stehen die folgenden drei Objekte:

- Ein Würfel. Koordinaten der Eckpunkte:
  - (40, 40, 0)
  - (40, 60, 0)
  - (60, 60, 0)
  - (60, 40, 0)
  - (40, 40, 20)
  - (40, 60, 20)
  - (60, 60, 20)
  - (60, 40, 20)
- Eine Rechteckpyramide. Koordinaten der Eckpunkte:
  - (50, 175, 0)
  - (80, 175, 0)
  - (80, 195, 0)
  - (50, 195, 0)
  - (65, 185, 50)
- Eine Dreiecksäule. Koordinaten der Eckpunkte:
  - (90, 130, 0)
  - (110, 110, 0)
  - (120, 140, 0)
  - (90, 130, 40)
  - (110, 110, 40)
  - (120, 140, 40)

1.4 Fahrsequenzen

Im Rahmen der Aufgabenstellung sind zusätzlich zur freien Navigation noch zwei automatisch ablaufende Sequenzen zu erstellen: eine Fahrt entlang einer Kreisbahn und eine Fahrt entlang einer Wegpunktstrecke.

Die Kreisbahn hat den Mittelpunkt (50, 100, 10) und den Radius 90. Blickrichtung der Kamera ist immer zum Kreismittelpunkt hin. Die Höhe ist während der gesamten Fahrt konstant.

Die Wegpunktstrecke besteht aus den folgenden Punkten:

1. (80, 0, 5)
2. (70, 20, 5)
3. (96, 80, 5)
4. (82, 120, 5)
5. (20, 180, 5)

Die Punkte werden in der angegebenen Reihenfolge abgefahren, wobei die Blickrichtung der Kamera immer in Fahrtrichtung ist. Die Punkte 2-4 werden bei der Fahrt nicht berührt, sondern jeweils auf einer tangential an die Wegstrecken gelegten Kreisbahn passiert. Die Höhe ist auch hier während der gesamten Fahrt konstant.

2 Grobentwurf

2.1 Konzeption

Die Aufgabe kann grob in drei Teile zerlegt werden: Anzeige der 3D-Szene, Steuerung von Bewegungen in der Szene und Datenmodell. Diese drei Aufgabenteile lassen sich ideal im Model-View-Controller-Konzept (MVC) abbilden. In diesem Konzept stellt der View eine Sicht auf die Daten des Modells dar, entspricht damit also dem Benutzerinterface (UI). Der Controller setzt Eingaben in das Benutzerinterface in Bewegungen in der Szene um und löst die damit verbundenen Änderungen im Modell aus. Das Modell selbst enthält die Repräsentation der dreidimensionalen Szene und die Funktionalität für Manipulationen am Modell, z.B. Bewegung der Kamera in der Szene.

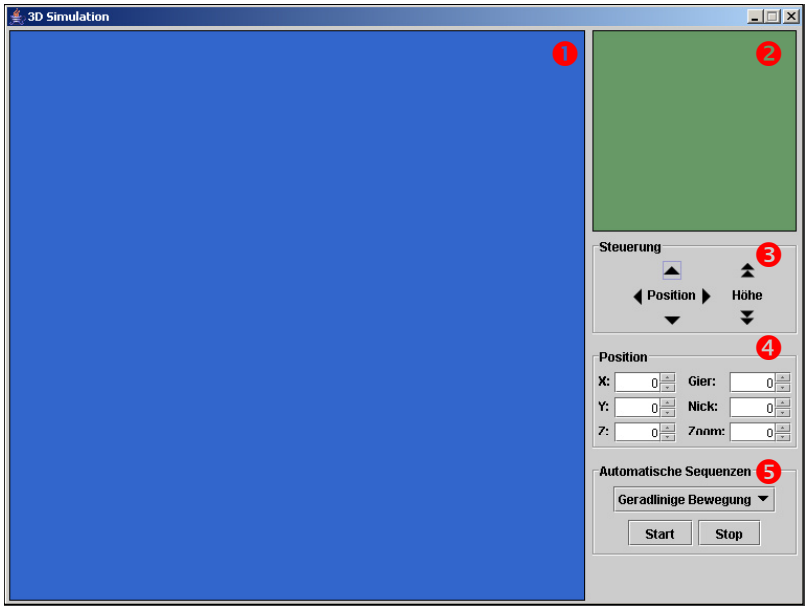
Abb. 2: Model-View-Conroller-Konzept

Um eine saubere Kapselung der Funktionalitäten zu erreichen, wurde zur Realisierung des Konzeptes ein objektorientierter Ansatz gewählt. Die Elemente Model, View und Controller werden als Pakete implementiert. Jedes Paket enthält mehrere Klassen, in denen die jeweilige Funktionalität implementiert wird.

2.2 Koordinatensysteme

Für das Modell wird einerseits ein absolutes Koordinatensystem (Weltkoordinatensystem) benötigt, das die absolute Positionierung von Dreiecken sowie der Kamera erlaubt. Andererseits ist es für Bewegungen der Kamera in der Szene und das spätere Rendering sinnvoller, die Koordinaten von Dreiecken relativ zu Position und Blickrichtung des Beobachters zu haben (Kamerakoordinatensystem), da diese Darstellung die Berechnungen vereinfacht. Im Modell kommen daher beide Koordinatensysteme zum Einsatz.

2.3 Prototyp



❶ 3D Ansicht

❷ Draufsicht der gesamten Szene mit der Positionsanzeige der Kamera

❸ Panel für die Steuerung

❹ Panel für die Positionseingabe

❺ Panel für die Auswahl automatischer Bewegungssequenzen

2.4 Pakete und Klassen

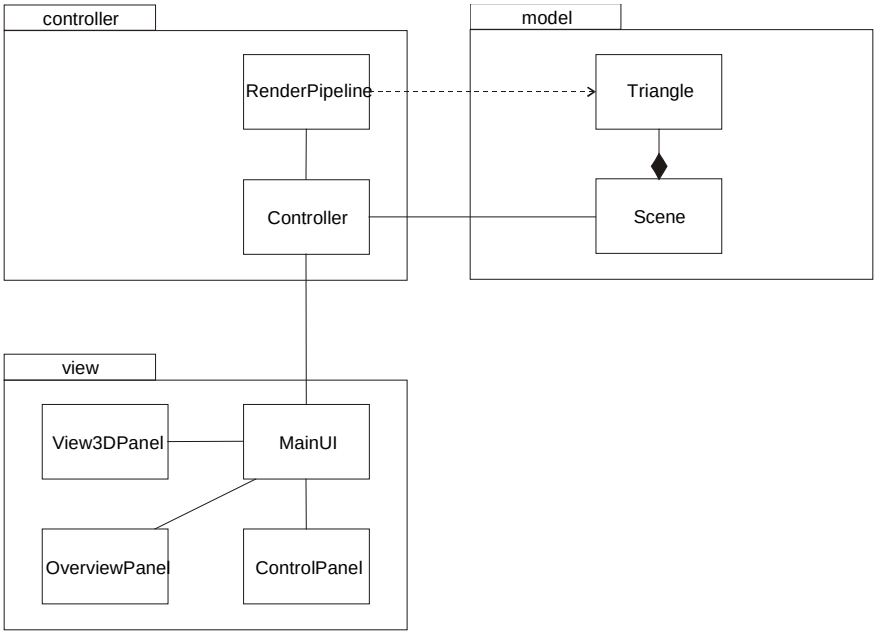


Abb. 3: Struktur der Pakete und Klassen des MVC-Konzeptes

Model: Das Modell enthält zwei Klassen (Scene und Triangle). Instanzen der Klasse Triangle repräsentieren einzelne Dreiecke. Scene enthält die Position der Kamera sowie alle Dreiecke der Szene. Auf die Zusammenfassung von Dreiecken zu Objekten wurde verzichtet, da die Szene nur wenige Dreiecke enthält und das Handling von Objekten unverhältnismäßigen Aufwand und Performance-Einbußen bedeuten würde.

View: Der View enthält die Klasse MainUI als Rahmen für die Anzeige- und Steuerelemente. Die Klasse View3DPanel ist die Hauptansicht der 3D-Szene. OverviewPanel ist eine Draufsicht auf die Szene, in der Position und Blickrichtung des Betrachters dargestellt werden. In der Klasse ControlPanel werden die Steuerelemente für die Navigation in der Szene und den Start der automatischen Sequenzen realisiert.

Controller: Der Controller enthält die Klassen Controller und RenderPipeline. Controller ist die zentrale Kontrollinstanz des Programms, die alle Abläufe steuert. RenderPipeline realisiert die Rendering-Pipeline, in der die Sicht der Kamera auf das 3D-Modell in eine am Bildschirm darstellbare 2D-Ansicht umgerechnet wird.

3 Feinentwurf

3.1 Das Package „Model“

3.1.1 Allgemeines

Das Modell enthält alle Daten, die eine 3D-Szene repräsentieren, einschließlich aller Operationen, die für Manipulationen an diesen Daten erforderlich sind. Diese Daten sind einerseits die Dreiecke, die den Aufbau der Szene ausmachen, und andererseits die Kamera, welche die Sicht auf die Szene definiert.

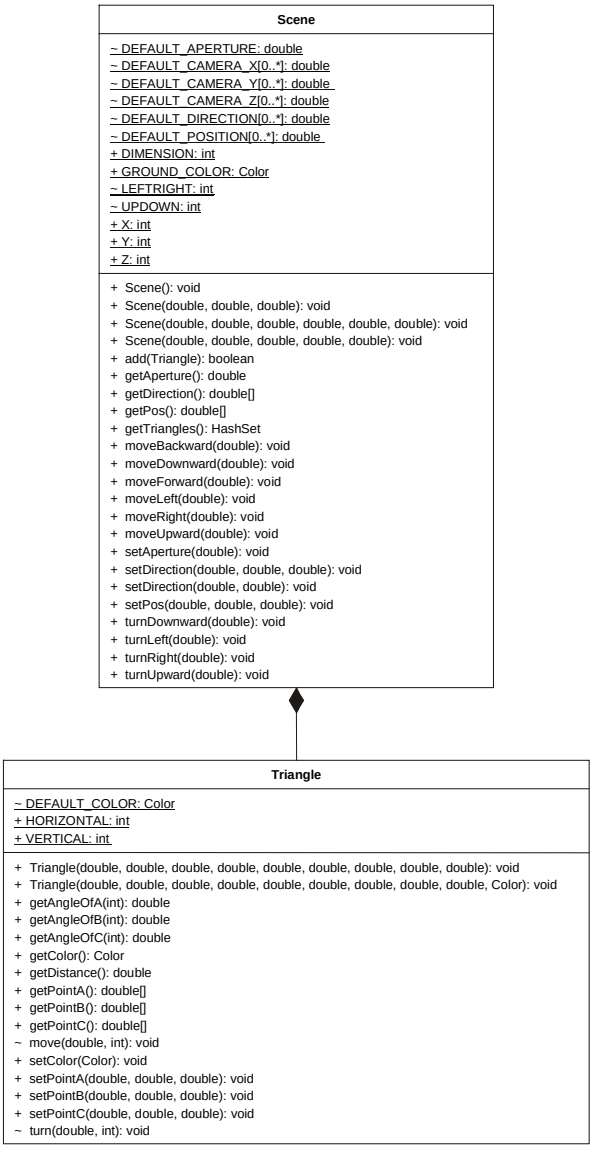


Abb. 4: Klassen des Pakets „Model“

3.1.2 Die Klasse „Scene“

Die Klasse Scene enthält die Kamera sowie die Dreiecke, aus denen sich die Szene zusammensetzt. Die Kamera ist über ihre Position und Lage im Weltkoordinatensystem sowie über den Öffnungswinkel definiert. Um die Kamera in der Szene sowohl bewegen als auch unmittelbar positionieren zu können, werden Funktionen für translatorische (drei Freiheitsgrade) und rotatorische Bewegung (zwei Freiheitsgrade: horizontal und vertikal) sowie für das Setzen von Position und Blickrichtung vorgesehen. Für jede Bewegungsrichtung wird eine separate Methode implementiert, um ein „sprechendes“ Interface zu gewährleisten.

Bei jeder Änderung der Kameraposition werden die Koordinaten der Dreiecke in der Szene neu berechnet. Faktisch wird also nicht die Kamera durch die Szene, sondern die Szene um die Kamera herum bewegt. Dabei ist zu beachten, dass die xy-Ebene im Weltkoordinatensystem „massiv“ ist, die z-Koordinate der Kameraposition darf also niemals negativ werden. Der Öffnungswinkel der Kamera ist auf Werte zwischen 0° und 180° beschränkt.

Translatorische Bewegungen können leicht über Vektoraddition berechnet werden. Dazu wird der parallel zur Bewegungsrichtung liegende Einheitsvektor des Kamerakoordinatensystems mit der Schrittweite der Bewegung skaliert und zum Ortsvektor der Kameraposition addiert.

$$\vec{p}_{Camera,neu} = \vec{p}_{Camera} + k \cdot \vec{e}_i \tag{1}$$

Die Berechnung von Drehungen ist komplexer. Hierfür wird zunächst eine kartesische Orthonormalbasis erzeugt und um den Drehwinkel in Drehrichtung gedreht. Da die Basisvektoren als Punkte eines Dreiecks aufgefasst werden können, kann das leicht über ein Triangle-Objekt realisiert werden. Das Ergebnis dieser Operation ist das Kamerakoordinatensystem nach der Drehung, aber noch in Kamerakoordinaten. Im nächsten Schritt werden dann die Kamerakoordinaten über eine Matrix-Operation in absolute Koordinaten umgerechnet. Die Transformationsmatrix T ergibt sich als Produkt aus der inversen Basis des Weltkoordinatensystems (W^{-1}) und der aktuellen Basis des Kamerakoordinatensystems (C). Die Spalten der Matritzen sind die Einheitsvektoren der jeweiligen Basis.

$$\mathrm{T} = \mathrm{W}^{-1} \cdot \mathrm{C} \tag{2}$$

$$\mathrm{T} = \begin{bmatrix} \vec{e}_{x,Welt} & \vec{e}_{y,Welt} & \vec{e}_{z,Welt} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \vec{e}_{x,Camera} & \vec{e}_{y,Camera} & \vec{e}_{z,Camera} \end{bmatrix} \tag{3}$$

$$\mathrm{T} = \begin{bmatrix} e_{x,Welt,x} & e_{y,Welt,x} & e_{z,Welt,x} \\ e_{x,Welt,y} & e_{y,Welt,y} & e_{z,Welt,y} \\ e_{x,Welt,z} & e_{y,Welt,z} & e_{z,Welt,z} \end{bmatrix}^{-1} \cdot \begin{bmatrix} e_{x,Camera,x} & e_{y,Camera,x} & e_{z,Camera,x} \\ e_{x,Camera,y} & e_{y,Camera,y} & e_{z,Camera,y} \\ e_{x,Camera,z} & e_{y,Camera,z} & e_{z,Camera,z} \end{bmatrix} \tag{4}$$

Zielkoordinatensystem ist das Weltkoordinatensystem, daher ist W die Einheitsmatrix E. Da die Matritzen quadratisch sind, ist außerdem die inverse Matrix gleich der transponierten Matrix. Da die Transponierte der Einheitsmatrix wieder die Einheitsmatrix und das Produkt einer Matrix C mit der Einheitsmatrix wieder die Matrix C ist, erhält man:

$$T = E^{-1} \cdot C = E' \cdot C = E \cdot C = C \tag{5}$$

Um einen Vektor von Kamera- in Weltkoordinaten umzurechnen, wird die Transformationsmatrix mit dem Vektor in Kamerakoordinaten multipliziert. Das Ergebnis ist der Vektor in Weltkoordinaten.

$$\vec{e} = T \cdot \vec{c} = C \cdot \vec{c} \tag{6}$$

Führt man diese Operation für jeden Basisvektor des gedrehten Koordinatensystems durch, so erhält man die neuen Basisvektoren des Kamerakoordinatensystems in Weltkoordinaten.

Für die Positionierung hat man das gleiche Problem, nur in die andere Richtung: die Weltkoordinaten des Differenzvektors zwischen aktueller Position und Zielposition müssen in Kamerakoordinaten umgerechnet werden, um die in Kamerakoordinaten gespeicherten Dreiecke entsprechend verschieben zu können.

$$T = C^{-1} \cdot E = C' \cdot E = C' \tag{7}$$

$$\vec{e} = T \cdot \Delta\vec{p} = C' \cdot \left(\vec{p}_{neu} - \vec{p}_{aktuell}\right) \tag{8}$$

Gemäß Spezifikation muss es möglich sein, der Kamera durch Eingabe von Nick- und Gierwinkel eine bestimmte Blickrichtung zu geben. Diese Anforderung stellte sich als problematisch heraus, da die Lage der Kamera im Raum durch diese zwei Winkel aber nicht eindeutig definiert ist. Bei Aufruf dieser Funktion wird die Kamera daher zunächst in eine definierte Lage zurückgesetzt, in der die Achsen des Kamerakoordinatensystems parallel zu den Achsen des Weltkoordinatensystems liegen. Danach wird die Kamera dann erst um den Gier- und anschließend um den Nickwinkel gedreht.

3.1.3 Die Klasse „Triangle“

Dreiecke sind die Primitive des Modells. Jedes Dreieck ist über drei Punkte und eine Farbe definiert. Aus Performancegründen wurde darauf verzichtet, Punkte als eigenständige Objekte zu realisieren, auch wenn dies dem objektorientierten Ansatz zuwiderläuft. Stattdessen werden sie als Arrays implementiert.

Da alle Dreiecke in Kamerakoordinaten gespeichert sind, werden außerdem Funktionen vorgesehen, um die Koordinaten der Eckpunkte bei Änderungen der Kameraposition neu zu berechnen. Da diese Funktionen nur paketintern benötigt werden und nicht von Anwendungsprogrammierern genutzt werden sollen, wurde die Funktionalität auf zwei Funktionen (für Translation und Rotation) konzentriert. Die Bewegungsrichtung wird über Parameter der Funktionen gesteuert. Dadurch werden Funktionsaufrufe gespart und so die Performance verbessert.

Bei der Translation finden ausschließlich Verschiebungen entlang der Koordinatenachsen statt. Daher muss lediglich der Verschiebungswert zur entsprechenden Vektorkomponente jedes Dreieckspunktes addiert werden.

Rotationen werden nur um zwei der drei Koordinatenachsen vorgenommen. Die Transformation erfolgt durch Matrix-Multiplikation mit der entsprechenden Drehmatrix. Für Drehungen um die z-Achse (horizontale Drehung) werden x- und y-Komponente jedes Dreieckspunktes wie folgt berechnet:

$$\vec{p}_{neu} = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \vec{p}_{alt} \tag{9}$$

$$p_{x,neu} = p_{x,alt} \cdot \cos\alpha + p_{y,alt} \cdot \sin\alpha \tag{10}$$

$$p_{y,neu} = p_{y,alt} \cdot \cos\alpha - p_{x,alt} \cdot \sin\alpha \tag{11}$$

$$p_{z,neu} = p_{z,alt} \tag{12}$$

Analog erfolgt für Drehungen um die x-Achse (vertikale Drehung) die Berechnung von y- und z-Komponente jedes Dreieckspunktes wie folgt:

$$\vec{p}_{neu} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix} \cdot \vec{p}_{alt} \tag{13}$$

$$p_{x,neu} = p_{x,alt} \tag{14}$$

$$p_{y,neu} = p_{y,alt} \cdot \cos\alpha + p_{z,alt} \cdot \sin\alpha \tag{15}$$

$$p_{z,neu} = p_{z,alt} \cdot \cos\alpha - p_{y,alt} \cdot \sin\alpha \tag{16}$$

In jedem Fall bleibt die Komponente in Richtung der Drehachse konstant und muss daher nicht neu berechnet werden.

3.2 Das Package „View“

3.2.1 Allgemeines

Das Paket „View“ enthält alle Dateien zur Darstellung der Grafischen Oberfläche. Neben den in Abb. 5 dargestellten Klassen sind das sämtliche statischen Grafiken, die als Symbole für das User-Interface benötigt werden. Diese Grafiken werden im Unterpaket „pics“ zusammengefasst.

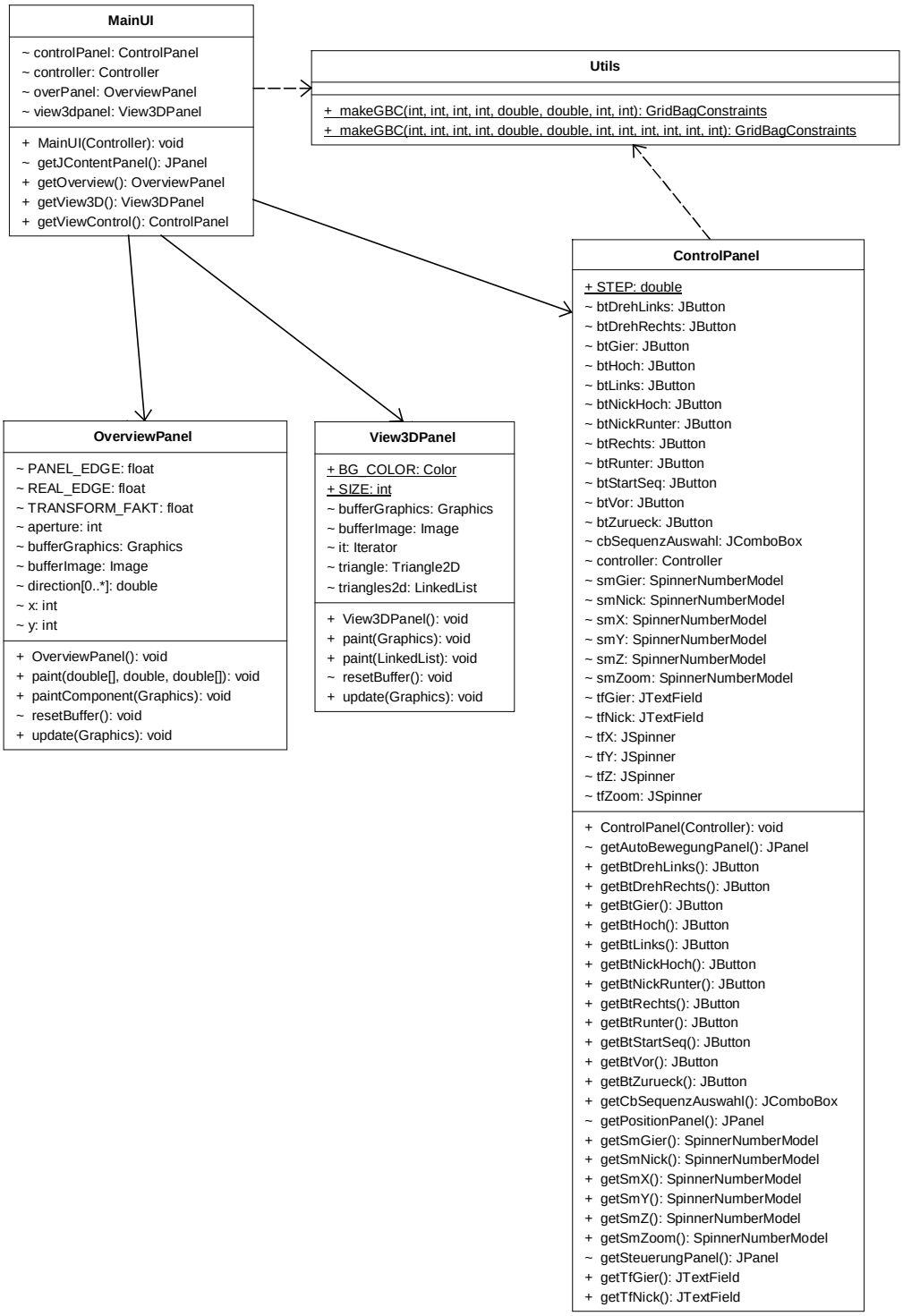


Abb. 5: Klassen des Pakets „View“

3.2.2 Klasse „MainUI“

Die Klasse „MainUI“ stellt das Hauptfenster zu Verfügung, auf dem die einzelnen Ansichten angeordnet sind. In Abb. 6 ist das Layout des Hauptfensters schematisch dargestellt.

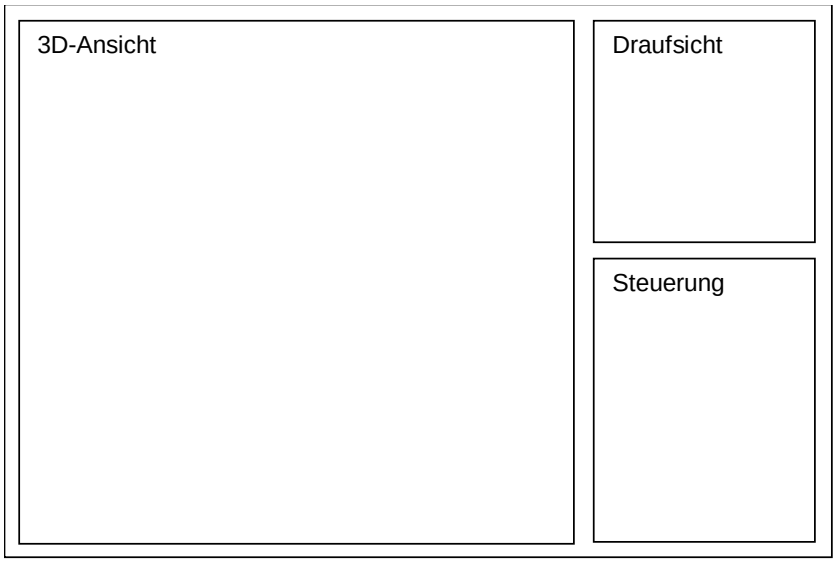


Abb. 6: Aufteilung des MainUI-Fensters

Nach dem Start der 3D-Simulation erscheint das Hauptfenster zentriert auf dem Bildschirm. Die Fenstergröße kann nicht verändert werden.

3.2.3 Klasse „ControlPanel“

Die Klasse „ControlPanel“ beinhaltet die Bedienelemente zum Steuern der Anwendung, z.B. die Eingabefelder für die Position, die Buttons für die Bewegung und Drehung und die Auswahlliste für die Fahrsequenzen.

3.2.4 Klasse „OverviewPanel“

Die Klasse „OverviewPanel“ bietet die Draufsicht auf die Szene und dient als Orientierungshilfe, um festzustellen an welcher Position sich der Betrachter befindet und welche Objekte sich im Sichtbereich des Betrachters befinden. Der Sichtbereich des Betrachtes wird durch ein halbtransparentes gleichschenkliges Dreieck dargestellt. Dazu wird die Position, die Blickrichtung und der Zoomwinkel des Betrachters aus dem Modell ermittelt und auf die Maße des Panels umgerechnet.

„OverviewPanel“ wird bei jeder Änderung des Modells aktualisiert. Um das Flackern des Bildes zu verhindern, wird eine Methode des Double-Buffering angewendet. Da das Zeichnen von mehreren Objekten auf den Bildschirm im Vergleich relativ langsam ist, wird das Bild zunächst in einen Puffer gezeichnet und dann der fertig gezeichnete Pufferinhalt auf dem Bildschirm ausgegeben.

3.2.5 Klasse „View3DPanel“

Die Klasse „View3DPanel“ bietet eine 3D-Sicht auf die Szene. Diese Sicht ist die Hauptansicht der Anwendung und hat deshalb die größte Fläche des Hauptfensters. „View3DPanel“ wird bei jeder Änderung des Modells aktualisiert. Um das Flackern des Bildes zu vermeiden, wird wie beim „OverviewPanel“ die Double-Buffering-Methode angewendet.

3.2.6 Klasse „Utils“

Die Klasse „Utils“ ist eine Hilfsklasse und beinhaltet Methoden für die flexible Anordnung der einzelnen Ansichten und Bedienungselemente. Diese Methoden werden von den Klassen „MainUI“ und „ControlPanel“ verwendet. Deshalb sind die Methoden der Klasse „Utils“ als „statisch“ deklariert, d.h. der Zugriff auf die Methoden erfolgt ohne Instanziierung der Klasse.

3.3 Das Package „Controller“

3.3.1 Allgemeines

Der Controller enthält alle Funktionen, die für die Navigation in der 3D-Szene notwendig sind. Die Funktionen werden über die Steuerelemente im ControlPanel des Views aufgerufen und rufen dann ihrerseits die entsprechenden Funktionen des Modells auf. Auch die automatischen Fahrsequenzen werden im Controller implementiert, sie werden jedoch in einem eigenen Abschnitt behandelt.

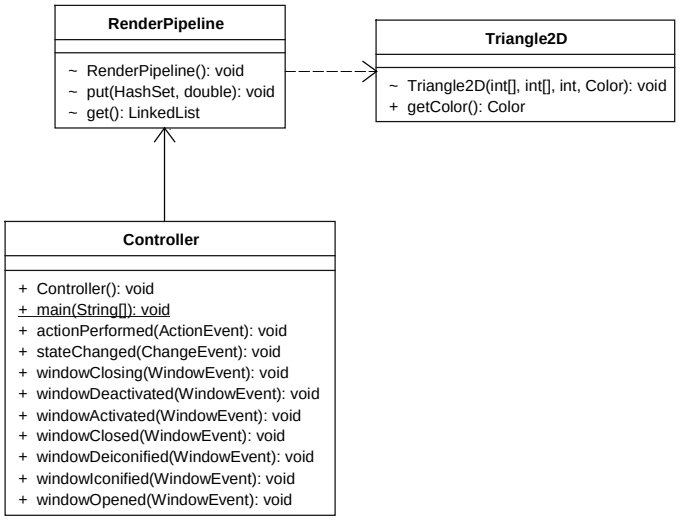


Abb. 7: Klassen des Pakets „Controller“

3.3.2 Die Klasse „Controller“

Die Klasse Contoller ist die zentrale Komponente der Anwendung. Mit der Initialisierung des Contollers wird die 3D-Simulation gestartet. Controller nimmt die Befehle des Anwenders entgegen.

Daraufhin manipuliert Controller das Model und veranlasst die Neudarstellung der Ansichten. Desweiteren beinhaltet Controller die Algorithmen für die automatischen Fahrsequenzen.

3.3.3 Die Klasse „RenderPipeline“

In der Rendering Pipeline werden alle Dreiecke aussortiert, die nicht dargestellt werden müssen. Die verbleibenden Dreiecke werden absteigend nach ihrem Abstand von der Kamera sortiert und für die Darstellung in eine Ebene projiziert (Umwandlung von 3D nach 2D). Dabei kommen folgende Verfahren zum Einsatz:

- *Clipping:*
  Dreiecke, die außerhalb des Sichtbereichs der Kamera liegen, müssen nicht dargestellt werden.

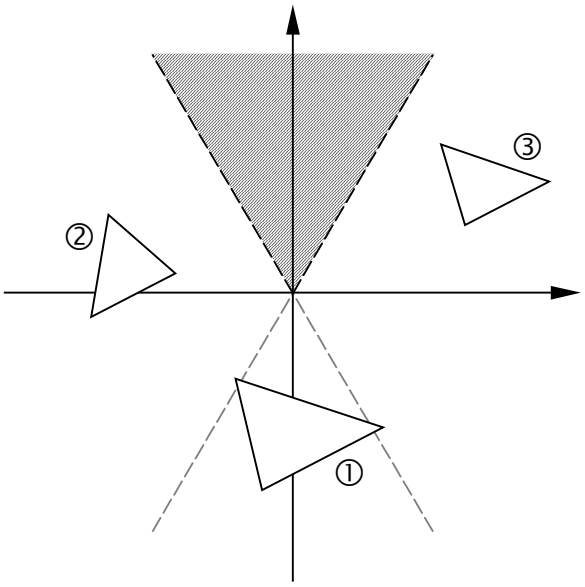


Abb. 8: Clipping von Dreiecken

Außerhalb des Sichtbereichs sind alle Dreiecke, die vollständig
- hinter der Kamera ①
- links von der linken Begrenzungsebene des Sichtbereichs ②
- rechts von der rechten Begrenzungsebene des Sichtbereichs ③
- oberhalb der oberen Begrenzungsebene des Sichtbereichs
- unterhalb der unteren Begrenzungsebene des Sichtbereichs

liegen.

- *Backface Culling:*
  Dreiecke haben eine Vorder- und eine Rückseite, wobei nur die Vorderseite sichtbar ist. Die Rückseite ist durchsichtig und muss daher nicht dargestellt werden.

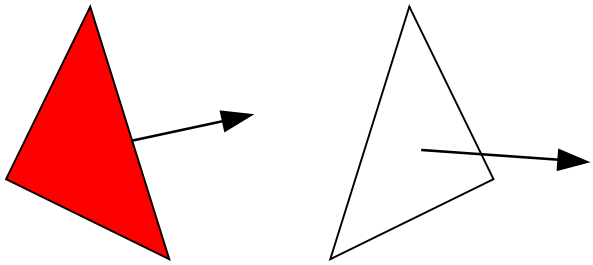


Abb. 9: Flächennormale eines Dreiecks

Ob ein Dreieck seine Vorder- oder Rückseite der Kamera zuwendet, wird durch die Flächennormale des Dreiecks bestimmt, die das äußere Produkt der Differenzvektoren der Dreieckspunkte ist.

$$\vec{n} = \left(\vec{p}_B - \vec{p}_A\right) \times \left(\vec{p}_C - \vec{p}_A\right) \tag{17}$$

Weist der Normalenvektor von der Kamera weg, so ist die Vorderseite des Dreiecks der Kamera zugewendet.

- *Sortierung:*
  Die darzustellenden Dreiecke werden absteigend nach ihrem Abstand von der Kamera sortiert. Als Abstandsmaß dient der am weitesten von der Kamera entfernte Dreieckspunkt. Dieser Schritt ist eine Voraussetzung für die sinnvolle Anwendung des Maler-Prinzips (s.u.).

- *Projektion:*
  Die Punkte jedes Dreiecks werden in eine Ebene projiziert, und das Dreieck so von der 3D- in eine 2D-Darstellung umgewandelt. Dazu wird für jeden Dreieckspunkt die Darstellungsebene parallel zur xz-Ebene durch den Dreieckspunkt gelegt, und die relative Position des Punktes innerhalb der Darstellungsfläche berechnet.

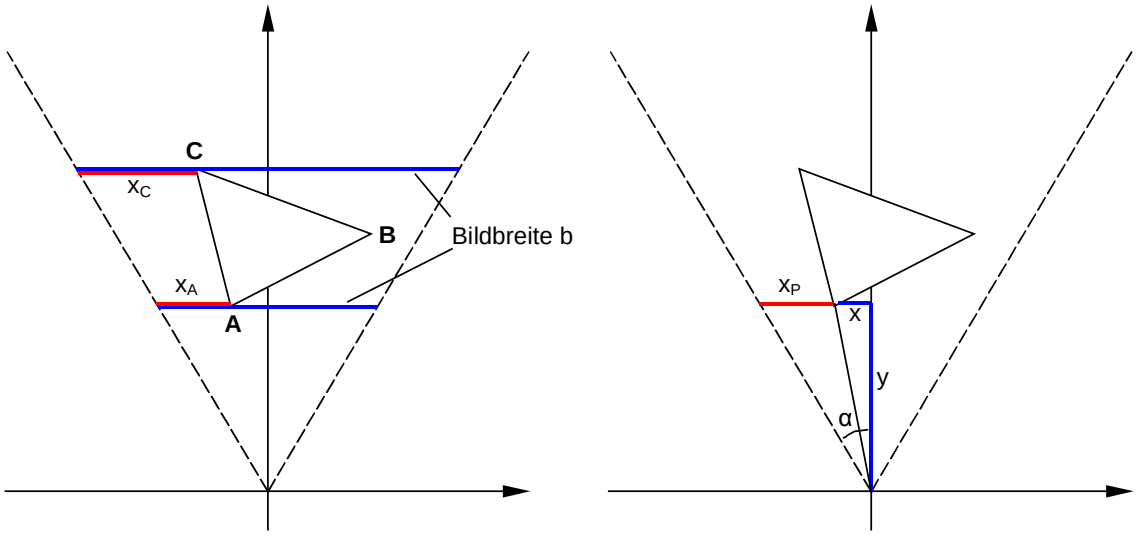


Abb. 10: Projektion von Dreieckspunkten

Die Berechnung erfolgt für alle Punkte nach den Formeln

$$x_P = \left(1 + \frac{x}{y \cdot \tan\alpha}\right) \cdot b \tag{18}$$

$$y_P = \left(1 - \frac{z}{y \cdot \tan\alpha}\right) \cdot b \tag{19}$$

Ein Problem stellen Dreiecke dar, die teilweise hinter der Kamera liegen, da die hinter der Kamera liegenden Dreieckspunkte bei der Projektion zu Darstellungsfehlern führen. Für solche Dreiecke wird daher eine Dreieckszerlegung durchgeführt. Dazu wird eine virtuelle Ebene knapp vor der xz-Ebene des Kamerakoordinatensystems eingeführt und die Schnittpunkte der Dreieckskanten mit dieser Ebene berechnet. Die Schnittpunkte werden dann anstelle der hinter der Kamera liegenden Dreieckspunkte für die Projektion verwendet.

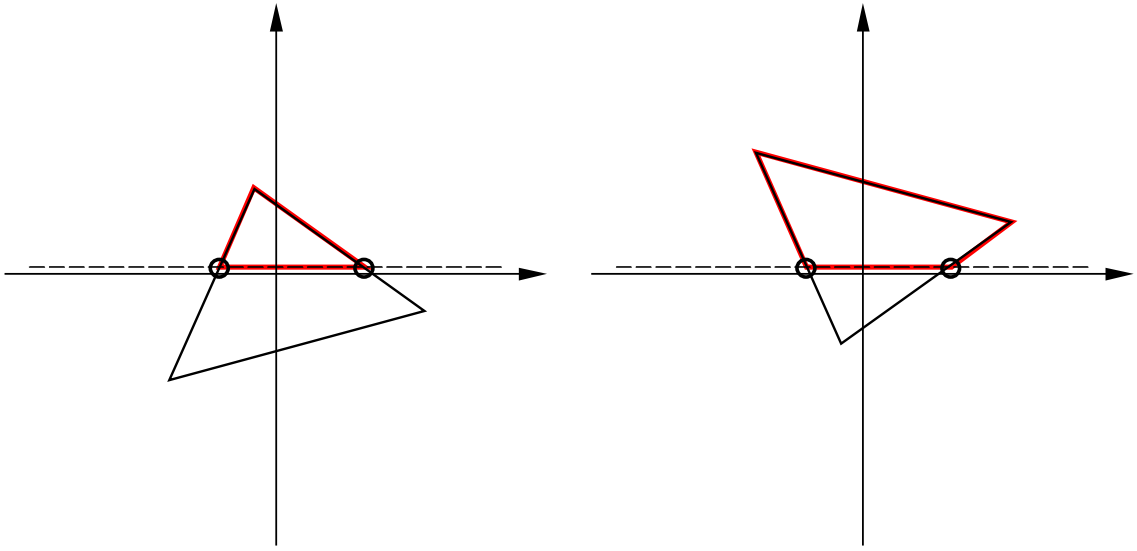


Abb. 11: Dreieckszerlegung

Wie der Abbildung zu entnehmen ist, können dabei „Dreiecke“ mit vier Ecken entstehen. Da jedoch Triangle2D von java.awt.Polygon abgeleitet ist und somit beliebige Polygone repräsentieren kann, ist hier keine weitere Zerlegung in zwei Dreiecke notwendig.

- *Maler-Prinzip:*
  Die darzustellenden Dreiecke werden in der Reihenfolge ihres Abstands von der Kamera von hinten nach vorne gezeichnet. Dadurch überdecken weiter vorne liegende Dreiecke die dahinter liegenden Dreiecke.

- *Double-Buffering:*
  Für die Bildschirmdarstellung werden zwei Puffer verwendet. Die darzustellenden Dreiecke werden zunächst in den ersten Puffer gezeichnet. Dann werden die Puffer umgeschaltet und der Inhalt des ersten Puffers auf dem Bildschirm ausgegeben, während die Dreiecke für das nächste

Bild in den zweiten Puffer gezeichnet werden. Danach werden die Puffer wieder umgeschaltet. Idealerweise erfolgt die Umschaltung beim VSync des Monitors, so dass das Bild beim Umschalten nicht flackert.
Auf diese Weise werden die Bildwechsel deutlich beschleunigt und man kann noch während ein Bild dargestellt wird bereits das nächste Bild berechnen.

3.3.4 Die Klasse „Triangle2D“

Instanzen der Klasse Triangle2D sind zweidimensionale Polygone (meist Dreiecke, in manchen Fällen Vierecke), die direkt am Bildschirm gezeichnet werden können. Da die darzustellenden Polygone eine Farbe haben, musste Triangle2D als eigene Klasse von java.awt.Polygon abgeleitet und um die Property color erweitert werden.

3.4 Fahrsequenzen

3.4.1 Kreisbahn

Die Kreisbahn beginnt und endet am Punkt (140, 100, 10) mit Blickrichtung zum Kreismittelpunkt (50, 100, 10). Der Vollkreis wird abhängig von der in der Klasse Controller definierten Geschwindigkeit in gleich große Winkelschritte zerlegt und die Kamera an den entsprechenden Kreispunkte positioniert.

$$x = x_M + r \cdot \cos\alpha \tag{20}$$

$$y = y_M + r \cdot \sin\alpha \tag{21}$$

Darin ist x_M die x-Koordinate des Kreismittelpunkts, y_M die y-Koordinate des Kreismittelpunkts und r der Kreisradius. α ist der aktuelle Winkel der Kameraposition in Bezug auf den Kreismittelpunkt und wird von 0° bis 360° durchiteriert. Die z-Koordinat e bleibt konstant.

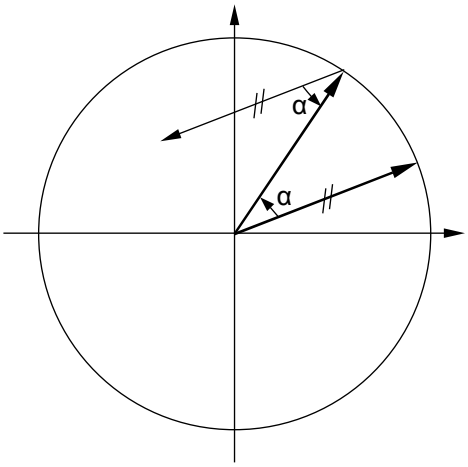


Abb. 12: Drehwinkel der Kreisbahn

Zusätzlich zur Positionierung muss dann noch die Blickrichtung der Kamera Korrigiert werden, da andernfalls die Kamera während der gesamten Fahrt in dieselbe Richtung blicken würde, anstatt zum Kreismittelpunkt. Der Winkel, um den die Kamera gedreht werden muss, ist ebenfalls α (Wechselwinkel an geschnittenen Parallelen), wie Abb. 12 zu entnehmen ist.

3.4.2 Gradlinige Bewegung

Da bei der Wegpunktsequenz die Zwischenpunkte nicht berührt, sondern über einen Kreisbogen angenähert werden sollen, ist es notwendig, für jeden Zwischenpunkt die Berührungspunkte des Kreisbogens mit dem jeweiligen Wegstreckenabschnitt zu ermitteln.

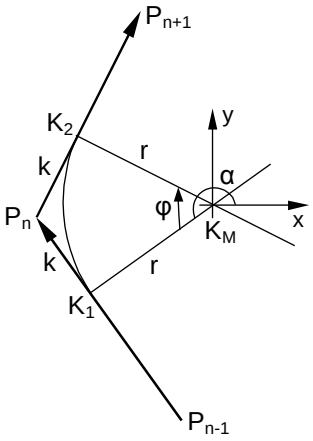


Abb. 13: Berechnung des Tangentialkreisbogens

Die Gesamtsequenz wird dazu in einzelne Iterationsschritte unterteilt. Ein Iterationsschritt umfasst dabei die Bewegung von P_{n-1} bis K_2. Der letzte Bewegungsabschnitt vom Endpunkt des letzten Kreisbogens bis zum Endpunkt der Sequenz muss separat behandelt werden.

Zunächst muss im Iterationsschritt der Parameter k berechnet werden, da er sowohl für die Berechnung des linearen Abschnitts als auch für die Berechnung des Kreisbogens benötigt wird. Da die Bewegung bei konstanter Höhe abläuft, kann man die z-Koordinate vernachlässigen und die Berechnung auf den zweidimensionalen Fall vereinfachen. Man erhält k durch Auflösung des folgenden linearen Gleichungssystems:

$$\vec{p}_n + k \cdot \frac{\vec{b}}{b} + r \cdot \frac{\vec{b}^{\perp}}{b^{\perp}} = \vec{p}_n - k \cdot \frac{\vec{a}}{a} + r \cdot \frac{\vec{a}^{\perp}}{a^{\perp}} \tag{22}$$

Darin sind:

$$\vec{a} = \begin{pmatrix} a_x \\ a_y \end{pmatrix} = \begin{pmatrix} p_{n,x} - p_{n-1,x} \\ p_{n,y} - p_{n-1,y} \end{pmatrix} \tag{23}$$

$$\vec{a}^{\perp} = \begin{pmatrix} a_y \\ -a_x \end{pmatrix} \tag{24}$$

$$a = |\vec{a}| \tag{25}$$

$$a^{\perp} = |\vec{a}^{\perp}| \tag{26}$$

$$\vec{b} = \begin{pmatrix} b_x \\ b_y \end{pmatrix} = \begin{pmatrix} p_{n+1,x} - p_{n,x} \\ p_{n+1,y} - p_{n,y} \end{pmatrix} \tag{27}$$

$$\vec{b}^{\perp} = \begin{pmatrix} b_y \\ -b_x \end{pmatrix} \tag{28}$$

$$b = |\vec{b}| \tag{29}$$

$$b^{\perp} = |\vec{b}^{\perp}| \tag{30}$$

Stellt man das Gleichungssystem nach k und r um, so erhält man:

$$k \cdot \left(\frac{\vec{a}}{a} + \frac{\vec{b}}{b} \right) = r \cdot \left(\frac{\vec{a}^{\perp}}{a^{\perp}} - \frac{\vec{b}^{\perp}}{b^{\perp}} \right) \tag{31}$$

Oder aufgelöst:

$$\begin{aligned} k \cdot \left(\frac{a_x}{a} + \frac{b_x}{b} \right) &= r \cdot \left(\frac{a_y}{a} - \frac{b_y}{b} \right) \\ k \cdot \left(\frac{a_y}{a} + \frac{b_y}{b} \right) &= r \cdot \left(\frac{b_x}{b} - \frac{a_x}{a} \right) \end{aligned} \tag{32}$$

Da k in diesem Gleichungssystem überbestimmt ist, kann eine beliebige dieser beiden Gleichungen für die Bestimmung von k gewählt werden. Für die Implementierung wurde die obere Gleichung gewählt. Nach k aufgelöst erhält man:

$$k \cdot = r \cdot \frac{\frac{a_y}{a} - \frac{b_y}{b}}{\frac{a_x}{a} + \frac{b_x}{b}} \tag{33}$$

Da der Winkel des Kreisbogens gleich dem Drehwinkel der Kamera zwischen den beiden Bewegungsrichtungen ist, kann φ wie folgt bestimmt werden:

$$\varphi = \arccos \frac{\vec{a} \cdot \vec{b}}{a \cdot b} \tag{34}$$

Ist k negativ, so handelt es sich um eine Linkskurve, andernfalls um eine Rechtskurve. Bei einer Linkskurve muss -φ anstelle von φ verwendet werden, da die Drehung in die entgegengesetzte Richtung erfolgt.

Als nächstes müssen Start- und Endpunkt des Kreisbogens (K_1, K_2) sowie der Mittelpunkt des Kreisbogens (K_M) berechnet werden. K_1 und K_2 erhält man über die folgenden zwei Formeln:

$$\vec{k}_1 = \vec{p}_n - k \cdot \vec{a} \tag{35}$$

$$\vec{k}_2 = \vec{p}_n + k \cdot \vec{b} \tag{36}$$

Die nächsten Berechnungen sind abhängig davon, ob eine Rechts- oder eine Linkskurve vorliegt.

- Rechtskurve: K_M errechnet sich wie folgt:

$$\vec{k}_M = \vec{p}_n - k \cdot \vec{a} + r \cdot \vec{a}^{\perp} \tag{37}$$

Der Startwinkel wird aus $\vec{a}^{\perp}$ errechnet:

$$\alpha_S = \arctan \frac{a_y}{-a_x} \tag{38}$$

- Linkskurve: K_M errechnet sich wie folgt:

$$\vec{k}_M = \vec{p}_n - k \cdot \vec{a} - r \cdot \vec{a}^{\perp} \tag{39}$$

Der Startwinkel wird aus $\vec{a}^{\perp}$ errechnet:

$$\alpha_S = \arctan \frac{-a_y}{a_x} \tag{40}$$

Der Endwinkel ergibt sich durch Subtraktion von φ:

$$\alpha_E = \alpha_S - \varphi \tag{41}$$

In jedem Iterationsschritt wird zunächst die Kamera linear von der aktuellen Position (P_{n-1}) nach K_1 bewegt. Das geschieht über einfache *moveForward()*-Aufrufe, wobei die Schrittweite von der Geschwindigkeit abhängt. Von K_1 aus erfolgt dann die Drehbewegung bis K_2. Die Kamera wird dabei auf dieselbe Art wie bei der Kreisbahn positioniert. Sobald K_2 erreicht ist, beginnt der nächste Iterationsschritt. Die Iteration wird so lange fortgesetzt, bis P_{n+1} gleich dem Endpunkt der Sequenz ist. Am Ende dieses letzten Iterationsschrittes muss die Kamera nur noch linear bis zum Endpunkt bewegt werden. Das geschieht wiederum durch *moveForward()*-Aufrufe.

4 Testfälle

4.1 Durchführung der Tests

Mit den Tests soll die korrekte Funktionsweise der Anwendung überprüft werden. Die Tests sind in mehrere Teilgebiete unterteilt.

- *Prüfung der Programmsteuerung:*
  Dabei ist es wichtig, dass die veranlassten Aktionen des Anwenders seinen Erwartungen entsprechen. Beispielsweise soll der Benutzer bei der Bewegung vorwärts den Eindruck haben, dass sich die Kameraposition vorwärts bewegt.
- *Prüfung der Algorithmen:*
  Dabei ist die korrekte Darstellung der Dreiecke besonders wichtig. Hier werden Entfernung der Rückseiten, Maler-Prinzip, perspektivische Projektion, Zoom und Clipping geprüft.
- *Prüfung der Fahrsequenzen:*
  Besonders entscheidend bei diesen Tests der flüssige Ablauf der Bewegung. Dabei dürfen die einzelnen Dreiecke während des Ablaufs nicht unterwartet verschwinden. Für die flimmerfreie Darstellung wurde DubbleBuffering-Methode eingesetzt, deren Wirksamkeit ebenfalls überprüft werden soll.

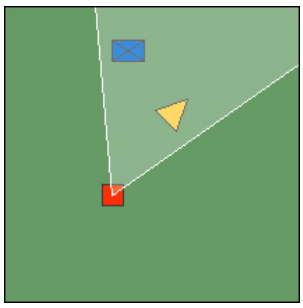
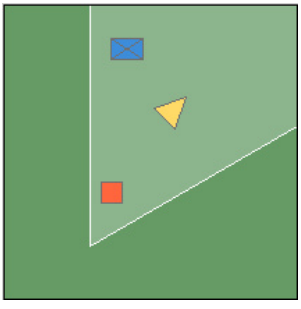
Zusätzlich zu den visuellen Tests werden die Funktionen des Backends (Modell und Rendering Pipeline) durch Unit-Tests auf Korrektheit überprüft.

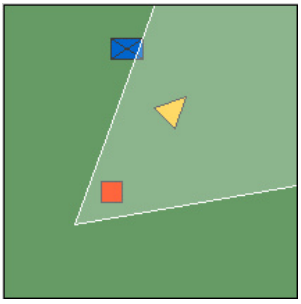
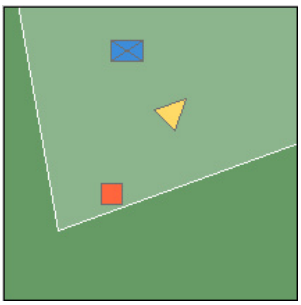
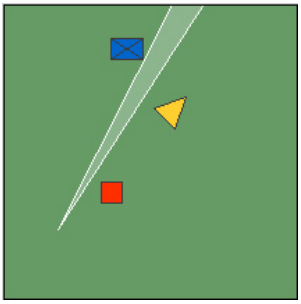
4.2 Steuerung

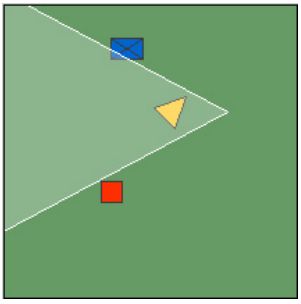
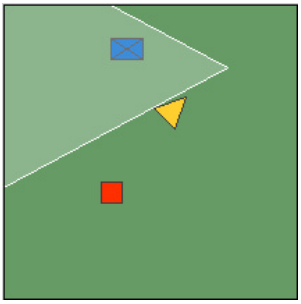
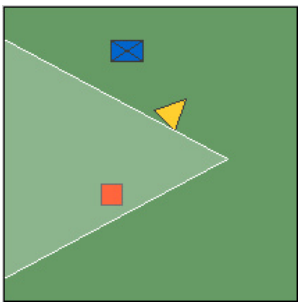
Vorgehen / Durchführung	**Erwartetes Ergebnis**	**Test bestanden**
Die Bewegungsbuttons betätigen	Der Betrachter bewegt sich in die jeweilige Richtung.	Ja
Die Buttons für die Höhe betätigen	Die Höhe des Betrachtes ändert sich.	Ja
Die Buttons für die Drehung betätigen	Die Blickrichtung des Betrachters ändert sich jeweils in die gewünschte Richtung.	Ja
In die Eingabefelder für die Position folgendes eingeben: X = 70; Y = 10; Z = 20 Zoom = 60; Gier = 0; Nick = 0	Der Betrachter bewegt sich zu der absoluten Position. Im Sichtbereich befinden sich alle Objekte.	Ja
In das Eingabefeld „Zoom“ „17“ eingeben, alle anderen Angaben wie im vorigen Test.	Nur die Pyramide ist sichtbar.	Ja

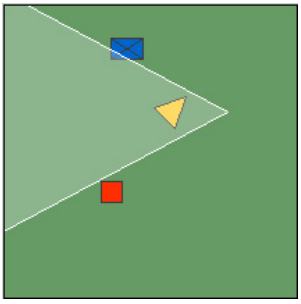
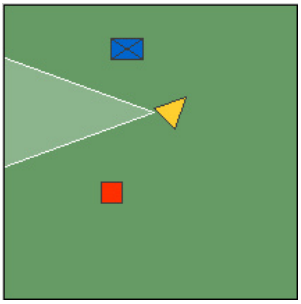
In das Eingabefeld für Gierwinkel einen positiven/negativen Wert eingeben und den Button „OK“ betätigen	Der Betrachter dreht sich nach links/rechts.	Ja
In das Eingabefeld für Nickwinkel einen positiven/negativen Wert eingeben und den Button „OK“ betätigen	Der Betrachter nickt hoch/runter.	Ja
Den Wert für Zoom vergrößern	Der Sichtbereich in der Draufsicht vergrößert sich, die Elemente der Szene erscheinen in der 3D-Ansicht kleiner.	Ja
Den Wert für Zoom verkleinern	Der Sichtbereich in der Draufsicht verkleinert sich, die Elemente der Szene erscheinen in der 3D-Ansicht größer.	Ja

4.3 Algorithmen

Vorgehen / Durchführung	**Erwartetes Ergebnis**	**Test bestanden**
Entfernung der Rückseiten Folgende Position einstellen: X = 50; Y = 50; Z = 20; Zoom = 60; Gier = 10; Nick = 0 	Pyramide und Dreiecksäule sind zu sehen. Die unsichtbaren Seiten der Objekte werden nicht dargestellt, daher sind die Seitenflächen des Kubus nicht sichtbar.	Ja
Maler-Prinzip Folgende Position einstellen: X = 30; Y = 0; Z = 20; Zoom = 60; Gier = -30; Nick = 0 	Alle Objekte sind zu sehen. Der Kubus verdeckt teilweise die anderen Objekte. Die Sortierung der Dreiecke erfolgt nach ihrer Entfernung vom Betrachter. Die Dreiecke, die sich ganz hinten befinden, werden zuerst gezeichnet. Die sichtbaren Dreiecke des Kubus werden als letzte gezeichnet und sind daher vollständig sichtbar.	Ja

Perspektivische Projektion Folgende Position einstellen: X = 15; Y = 20; Z = 70; Zoom = 60; Gier = -45; Nick = -50 	Der Kubus wird schräg betrachtet, so dass man die drei Kanten, die senkrecht zum Boden verlaufen, sieht. In Realität sind die Seiten parallel zu einander. Bei der perspektivischen Darstellung laufen die Seiten aufeinander zu und treffen sich in einem gemeinsamen Fluchpunkt.	Ja
Zoom Folgende Position einstellen: X = 0; Y = 15; Z = 10; Zoom = 80; Gier = -30; Nick = 0 Zoom stufenweise verkleinern. Zoom = 80  Zoom = 5 	Die Objekte verschwinden nacheinander aus dem Sichtbereich. Zusätzlich werden die Draufsicht und die 3D-Ansicht miteinander verglichen. Nur sichtbare Objekte befinden sich im Sichtbereich (siehe Draufsicht).	Ja

Clipping 1 Folgende Position einstellen: X = 160; Y = 125; Z = 20; Zoom = 55; Gier = 90; Nick = 0 	Der Betrachter sieht die Dreieckssäule und im rechten Berech eine Ecke der Pyramide. Der Kubus befindet sich links vom Betrachter und ist nicht sichtbar	Ja
Clipping 2 Folgende Position einstellen: X = 168; Y = 125; Z = 20; Zoom = 55; Gier = 90; Nick = 0 	Der Betrachter sieht die gesamte Pyramide. Dreieckssäule und Kubus befinden sich links vom Betrachter und sind nicht sichtbar.	Ja
Clipping 3 Folgende Position einstellen: X = 160; Y = 83; Z = 20; Zoom = 55; Gier = 90; Nick = 0 	Der Betrachter sieht den Kubus. Dreieckssäule und Pyramide befinden sich links vom Betrachter und sind nicht sichtbar.	Ja

Clipping 4 Folgende Position einstellen: X = 160; Y = 125; Z = 80; Zoom = 55; Gier = 90; Nick = 0 	Der Betrachter befindet sich oberhalb der Dreieckssäule. Diese ist nicht sichtbar.	Ja
Clipping 5 Folgende Position einstellen: X = 90; Y = 125; Z = 20; Zoom = 40; Gier = 90; Nick = 0 	Die Dreieckssäule befindet sich direkt hinter dem Betrachter und ist nicht sichtbar	Ja

4.4 Fahrsequenzen

Vorgehen / Durchführung	Erwartetes Ergebnis	Test bestanden
Kreisbahn In der Auswahlliste „Kreisbewegung“ auswählen und den Button „Start“ betätigen	Die Kamera springt zur Position (X = 140; Y = 100; Z = 10). Danach beginnt die Bewegung mit Radius r = 90 und Mittelpunkt (X = 50; Y = 10; Z = 10) gegen den Uhrzeigersinn. die Blickrichtung bleibt während der gesamten Fahrsequenz in Richtung Kreismittelpunkt. Die Bewegung stoppt nach einem kompletten Umlauf.	Ja
Geradlinige Bewegung In der Auswahlliste „Geradlinige Bewegung“ auswählen und den Button „Start“ betätigen	Die Kamera springt zur Position (X = 80; Y = 0; Z = 5). Danach beginnt die Bewegung. Die Kurvenbewegung wird flüssig animiert. Die Richtungsänderung an den Stützpunkten erfolgt in Kurven an den Stützpunkten vorbei.	Ja
Flimmerfreie Darstellung	Während einer Fahrsequenz ist kein Flimmern zu sehen.	Ja

5 Zusammenfassung der Ergebnisse

Zusammenfassend kann festgestellt werden, dass die 3D-Simulation die gestellten Anforderungen erfüllt. Alle erforderlichen Anzeigen und Steuerelemente wurden erfolgreich umgesetzt. Die implementierten Algorithmen sind sehr robust. Dies macht sich bei der Kreisbewegung besonders positiv bemerkbar.

Sehr vorteilhaft für die Zusammenarbeit erwies sich die Teilung der Anwendung nach der Model-View-Contol-Architektur. Einzelne Komponenten wurden von den Teammitgliedern getrennt voneinander entwickelt. Zum Abgleich der Schnittstellen fanden regelmäßig Reviews statt. Die Model-Klassen wurden über Unit-Tests unabhängig von der Benutzeroberfläche getestet. Dadurch konnte erreicht werden, dass die Anwendung bereits beim Zusammensetzen der Module fast fehlerfrei lief. Die verbleibenden Fehler wurden dann über Tests identifiziert und erfolgreich korrigiert.

Für die Unit-Tests wurde das Java-Framework JUnit eingesetzt, mit dem Tests für einzelne Methoden oder komplette Klassen geschrieben und automatisch abgearbeitet werden können. Die definierten Testfälle können damit wiederholbar und automatisiert überprüft werden.

Wegen der begrenzten Bearbeitungszeit wurde drauf geachtet, ausschließlich die gestellten Anforderungen umzusetzen. Dabei galt die besondere Aufmerksamkeit der Implementierung der Algorithmen. Die Implementierung bietet aber durchaus noch Raum für Verbesserungen und zusätzliche Features. Beispielsweise könnte es wünschenswert sein, die Kamera auch über die Tastatur oder andere Eingabegeräte – beispielsweise Joysticks – steuern zu können. Ebenso könnte es wünschenswert sein, die Geschwindigkeit bei den Fahrsequenzen vom Benutzer manuell angeben zu lassen. Momentan ist sie fest vorgegeben.

Abschließend lässt sich sagen, dass es sehr interessant war, während der Aufgabenbearbeitung verschiedene Methoden der grafischen Datenverarbeitung kennen zu lernen und durch praktische Anwendung nachzuvollziehen.

Abbildungsverzeichnis