3D Racer

Marcus Lehmann, Stefan Voigt und Christoph Knop

20.09.2005

Überblick

1 | Die Spielidee

2 | Geplante Features und ihre Realisierung

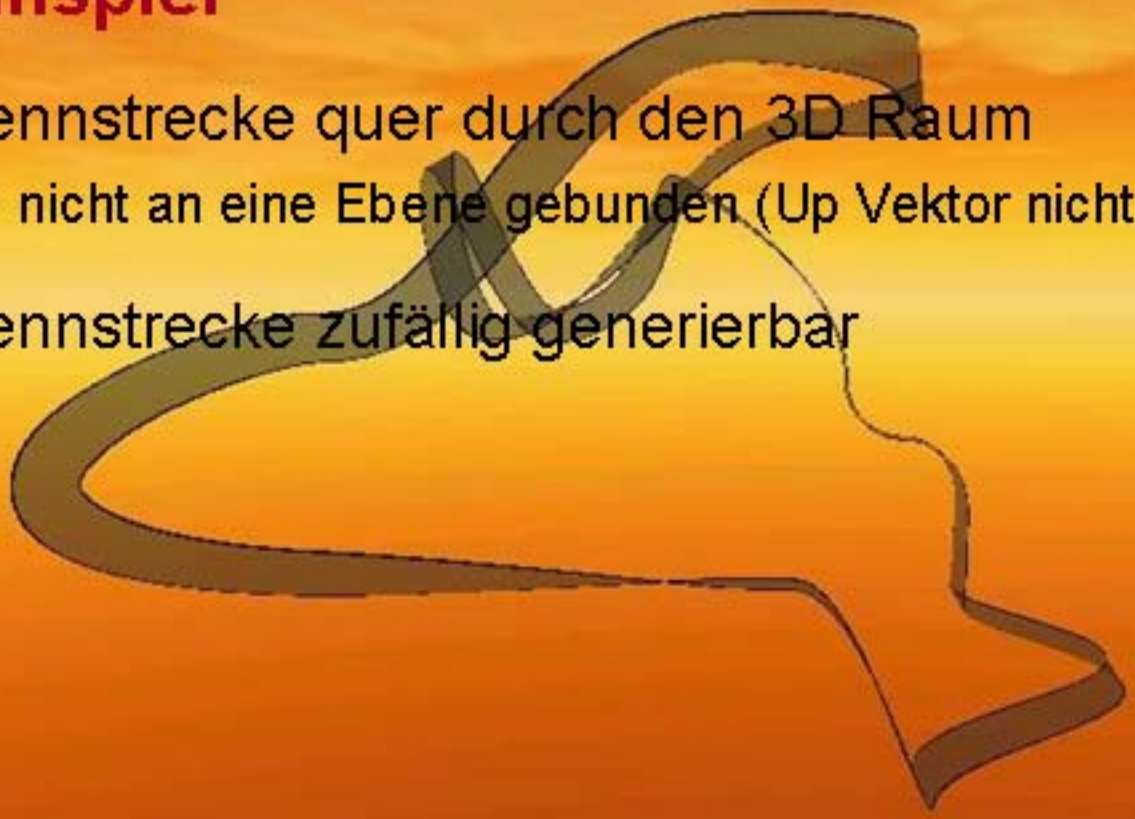
3 | Gesamtsystem und Architektur

4 | Vorführung und Diskussion

1| Spielidee

Rennspiel

- Rennstrecke quer durch den 3D Raum
  - nicht an eine Ebene gebunden (Up Vektor nicht immer gleich)
- Rennstrecke zufällig generierbar



2 Features im Überblick

Hauptfeatures

F1 - Erzeugung einer zufälligen Rennstrecke
F2 - Automatischer Kameraflug
F3 - Steuerung und Integration eines Rennmobils
F4 - Implementierung eines Partikelsystem
F5 - Verschiedene Kameraperspektiven

Zusatzfeatures

Z1 - Realisierung eines einfaches Menü
Z2 - Head Up Display
Z3 - Verschiedene Umgebungsprofile
Z4 - Bonus-Items auf der Rennstrecke
Z5 - Zusatzelemente zur Orientierung des Spielers

2.1 F1 - Erzeugung einer zufälligen Rennstrecke

Anforderung

- Strecke nicht auf 2D Raum beschränkt
- Strecke soll enden wo sie anfängt
- Strecke soll sich nicht durchkreuzen
- „interessanter“ Streckenverlauf

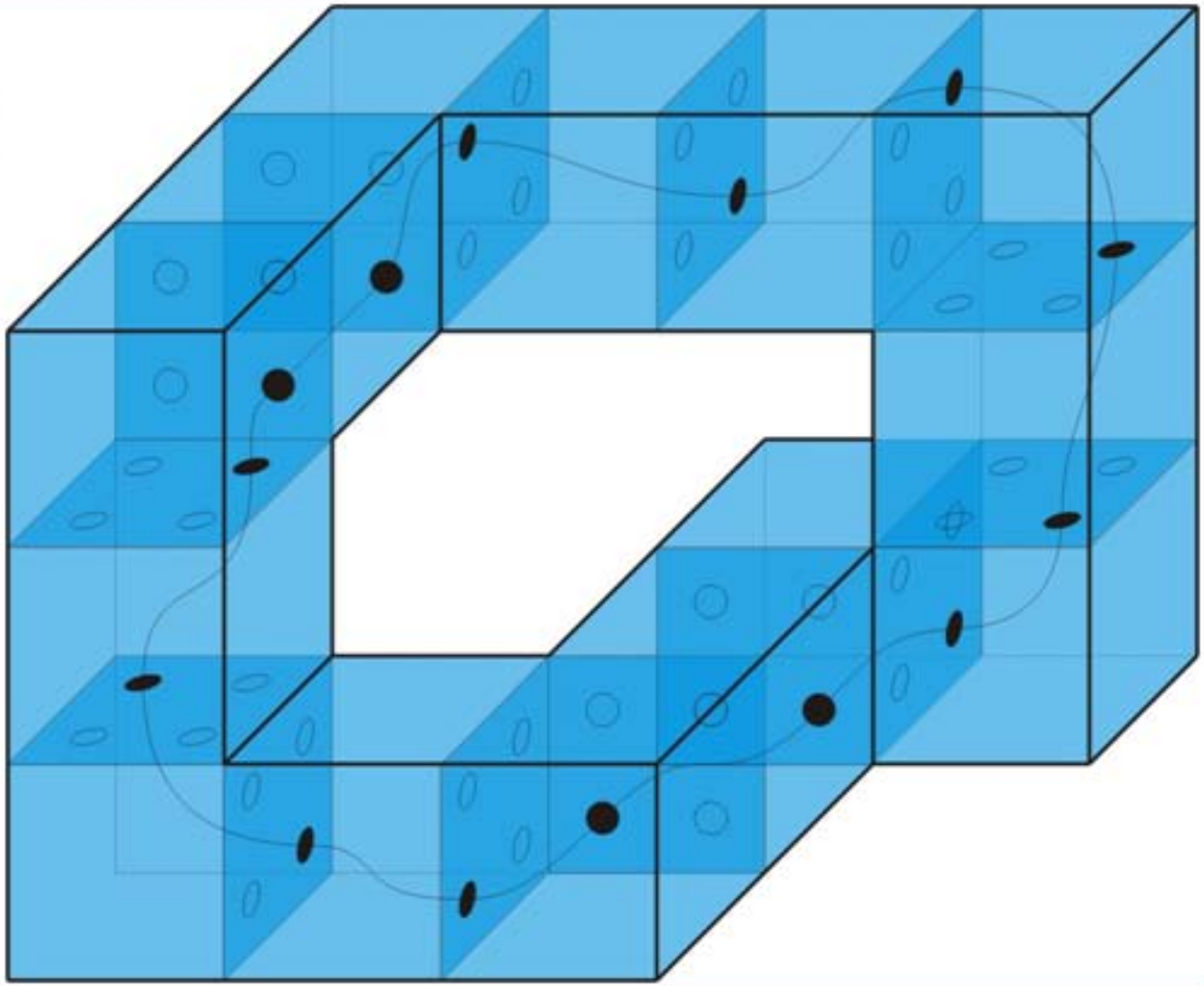
Mehrere Lösungsansätze

- Parametrische Funktion
- Blockkonzept

2.1 F1 - Erzeugung einer zufälligen Rennstrecke

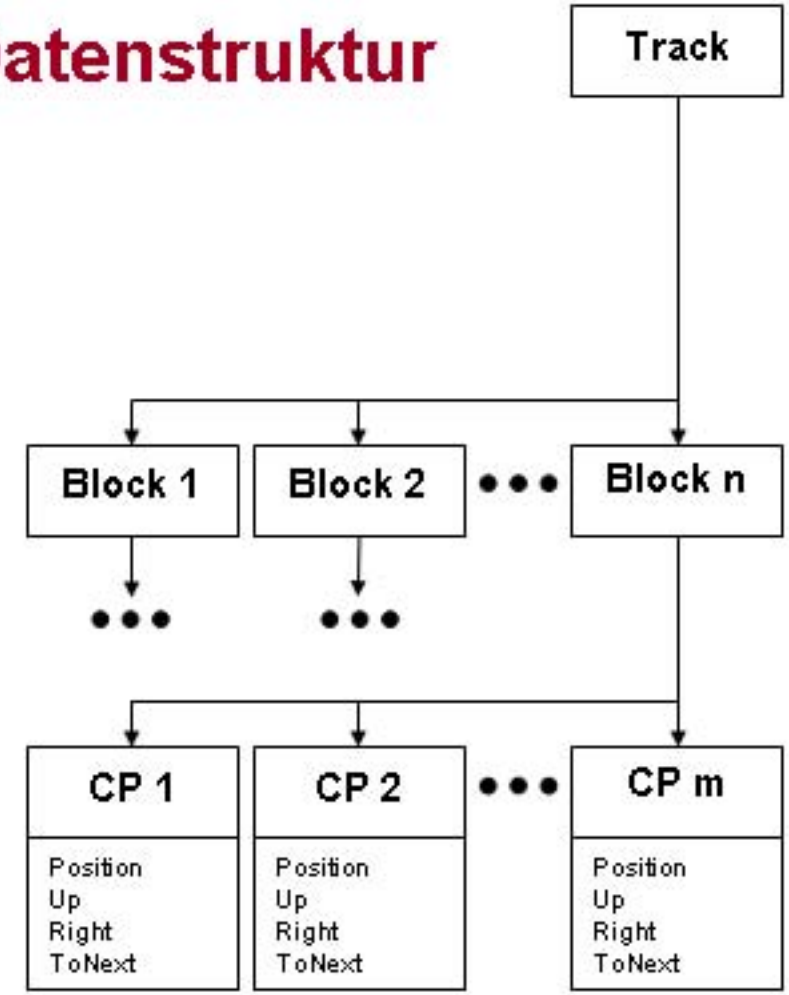
Umsetzung

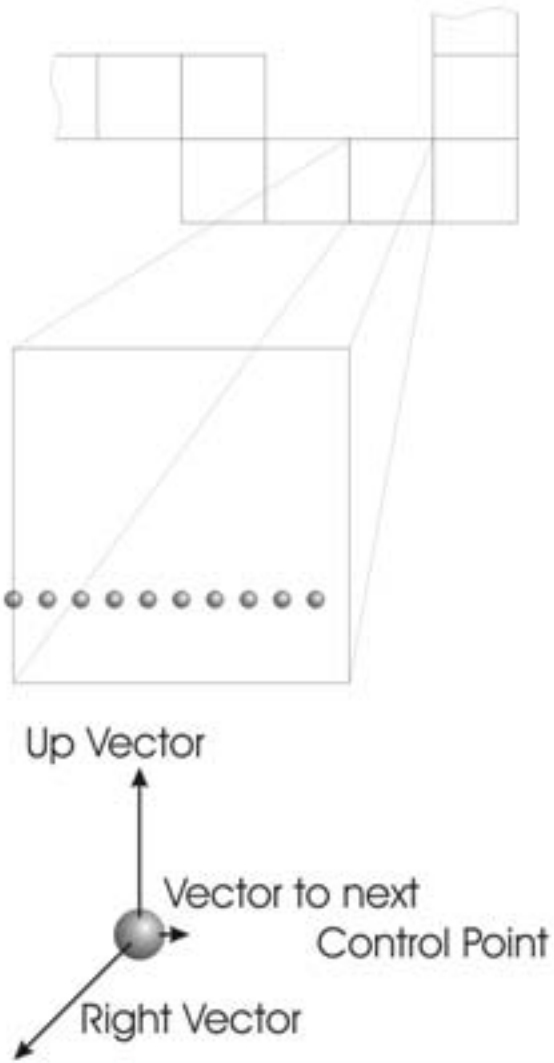
- Blöcke erzeugen
- Durchstoß-punke wählen
- Strecke blockweise erstellen



2.1 F1 - Erzeugung einer zufälligen Rennstrecke

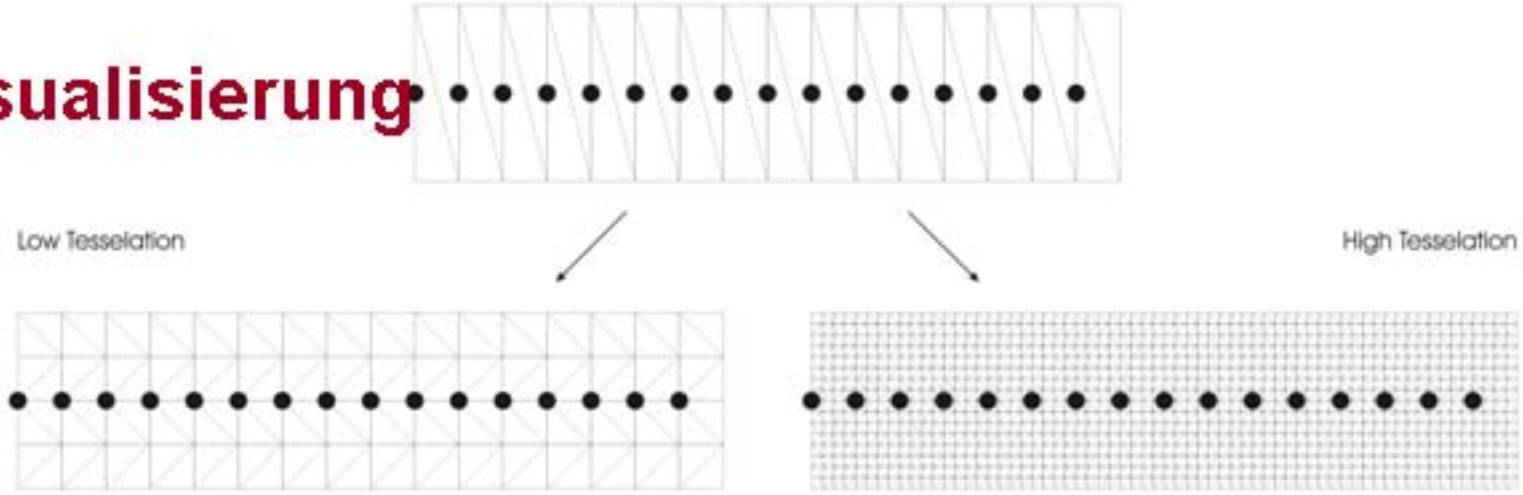
Datenstruktur



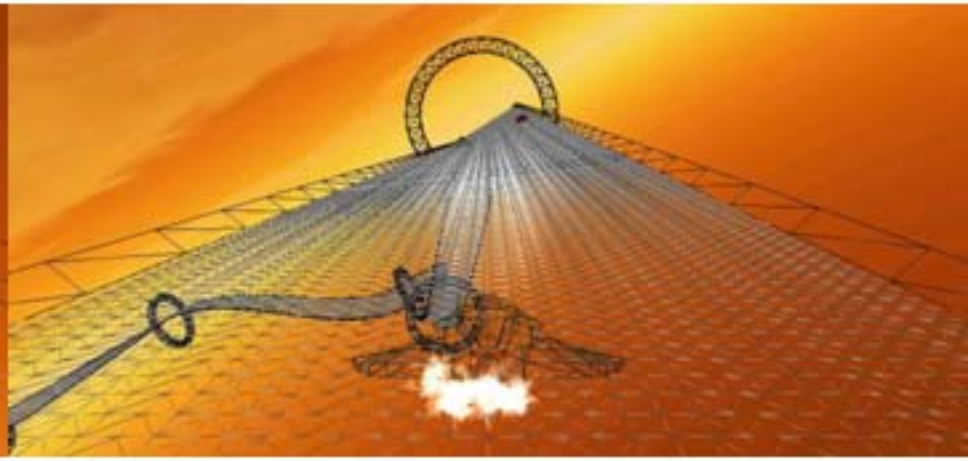


2.1 F1 - Erzeugung einer zufälligen Rennstrecke

Visualisierung



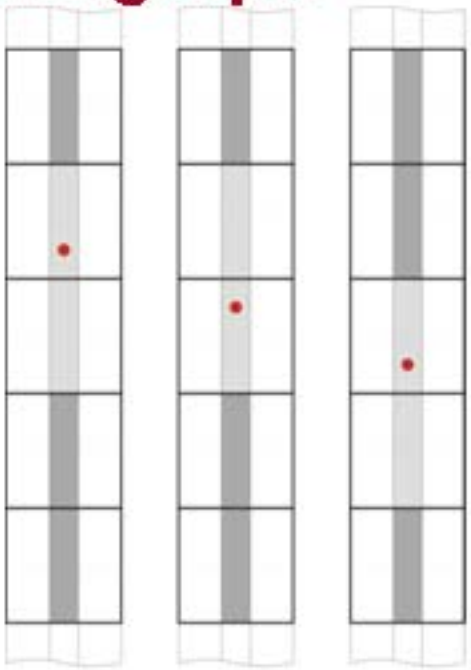
656 Vertices



25680 Vertices

2.1 F1 - Erzeugung einer zufälligen Rennstrecke

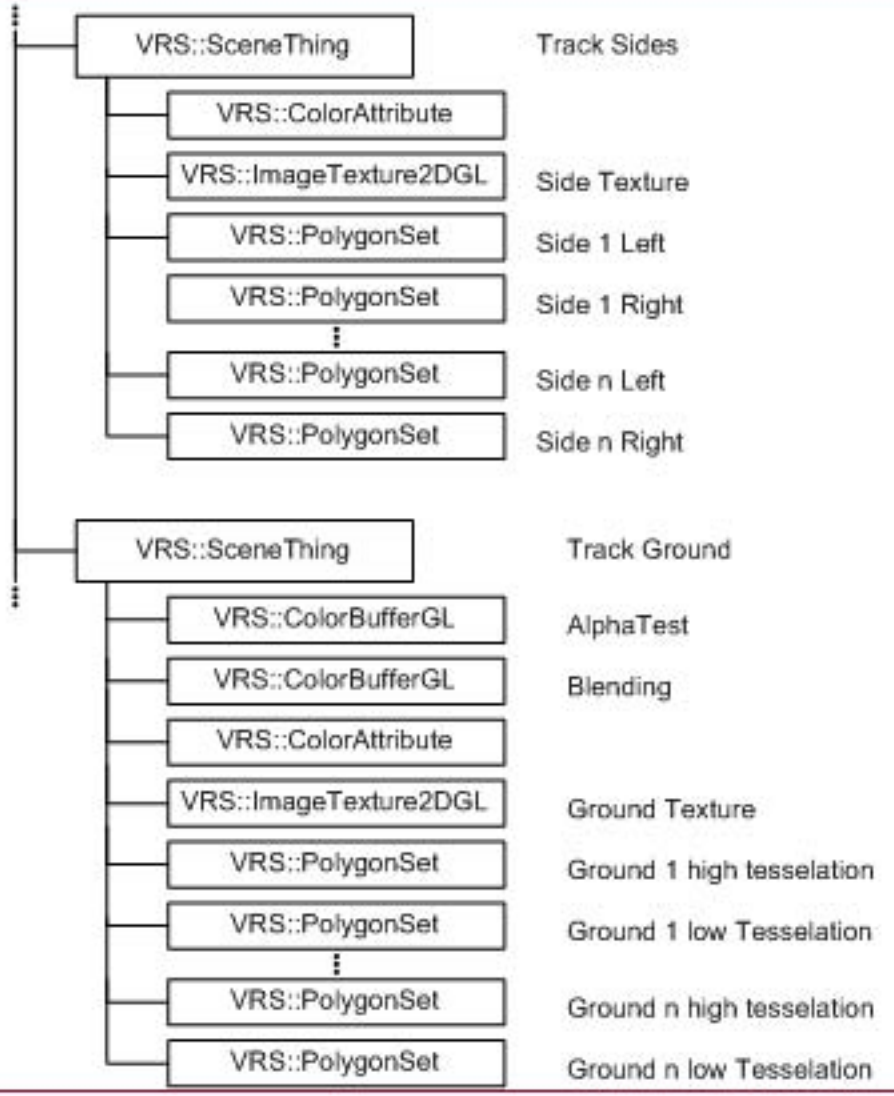
Szenengraph



High Tesselation Block

Low Tesselation Block

Vehicle



2.2 F2 - Automatischer Kameraflug

Anforderung

- Vor der Fahrt soll automatisch generierte Strecke entlanggeflogen werden

Umsetzung

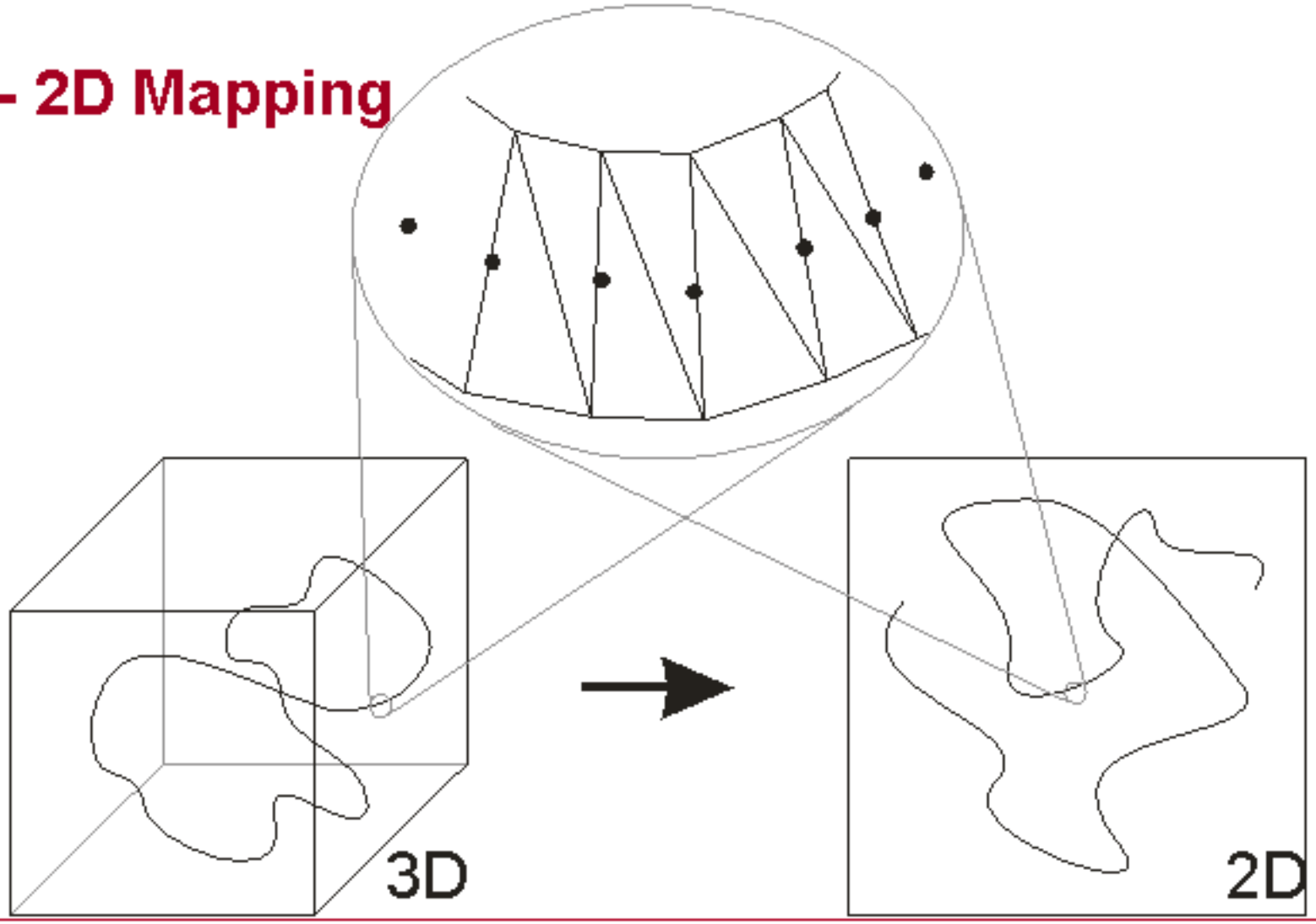
- Kamera orientiert sich an Kontrollpunkten
- Positionsdifferenz zeitabhängig (unabhängig von Framerate)
- Vorführung: später

Umsetzung mittels Backend

- einmalig:
  - Streckentransformation in den 2D Raum
- im Spiel:
  - Bewegungs- und Kollisionsberechnung im 2D Raum
    - → **Lenkung und Beschleunigung**
      - Wendekreisvergrößerung bei Geschwindigkeitszunahme
    - → **Abbremsen bei Kollision mit Bande**
    - → **Schadensmodell**
  - Mapping der Fahrzeugposition und -ausrichtung in den 3D Raum
  - Präsentation im 3D Raum

2.3 F3 - Steuerung und Integration eines Rennmobils

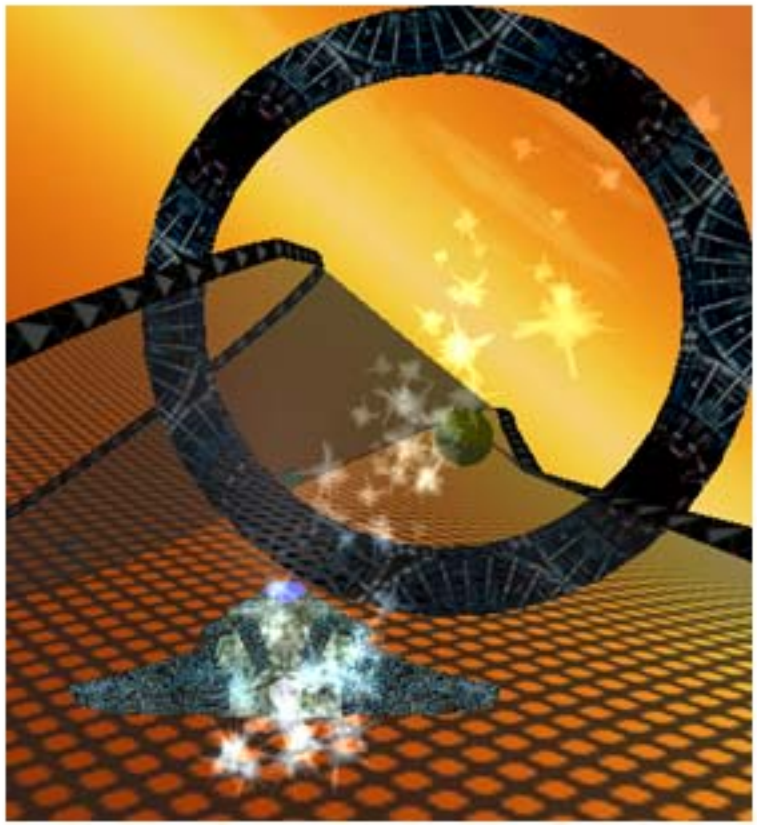
3D- 2D Mapping



2.4 F4 - Implementierung eines Partikelsystems

Anforderung

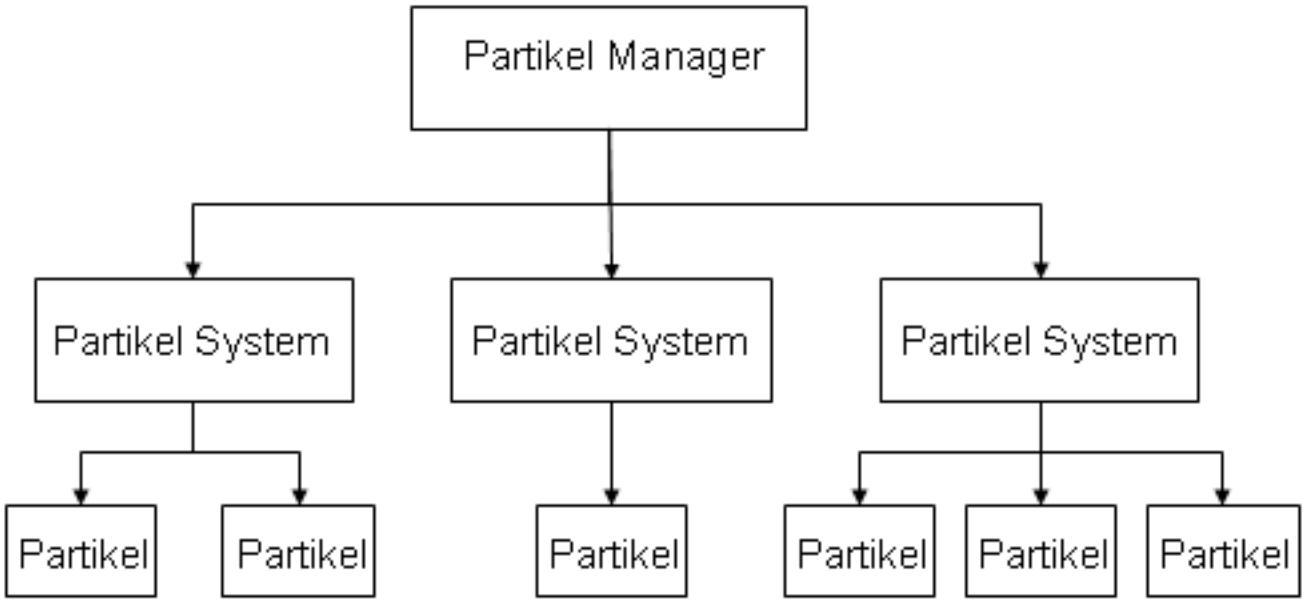
- Spezielle grafische Effekte
  - Feuer, Rauch, Fontänen, Blitze
- Auflockerung der Szene durch dynamische Effekte



Referenz: “Building an Advanced Particle System” by John van der Burg

2.4 F4 - Implementierung eines Partikelsystems

Allgemeiner Überblick



Implementierung

- Partikel Klasse (Particle)
  - Basis Informationen eines Partikels
    → u.a. Position, Größe, Geschwindigkeit, Farbe, Energy
- Partikel Shape Klasse (ParticleShape)
  - Ermöglich bessere Sturkturierung
  - Speichert jeweils 4 Pointer auf Vertices, Color, Normalen und Texturkoordinaten

→ ein Quad pro Partikel

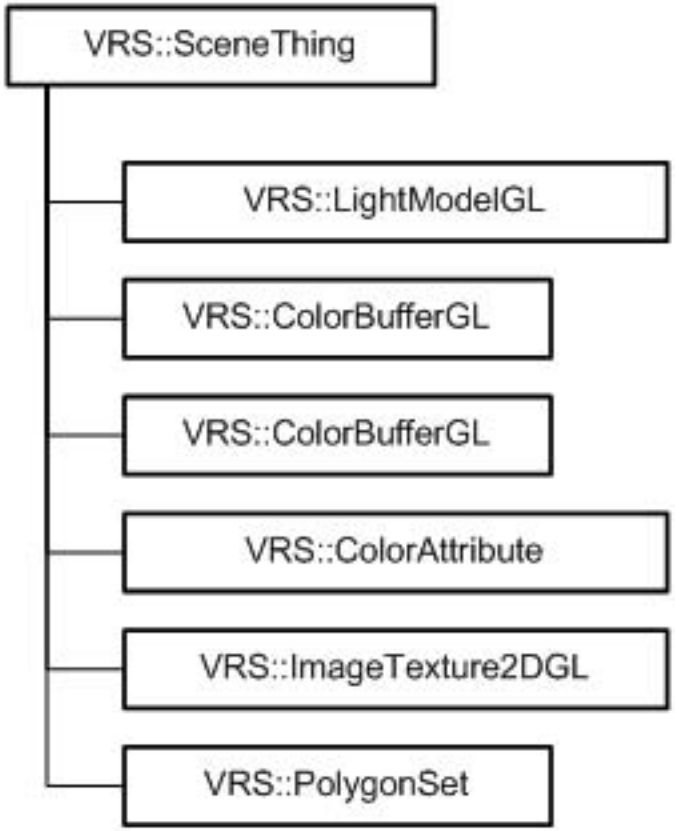
2.4 F4 - Implementierung eines Partikelsystems

Implementierung

- Partikel System Klasse (ParticleSystem)
  - Basisklasse für beliebige Partikelsystemarten
  - Verwaltet ein eigenständiges System aus gleichartigen Partikeln
  - Ein Polygonset für alle Partikel
  - Bestimmt das Verhalten der einzelnen Partikel je nach gewünschtem Effekt
- Partikel System Manager Klasse (ParticleSystemsManager)
  - Verwaltet mehrere Partikelsysteme
  - Kann bei Bedarf tote Partikelsysteme löschen

2.4 F4 - Implementierung eines Partikelsystems

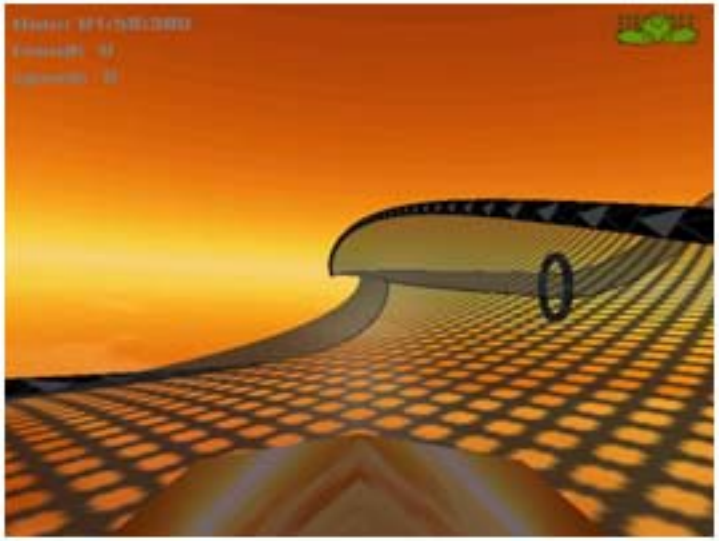
Szenengraph

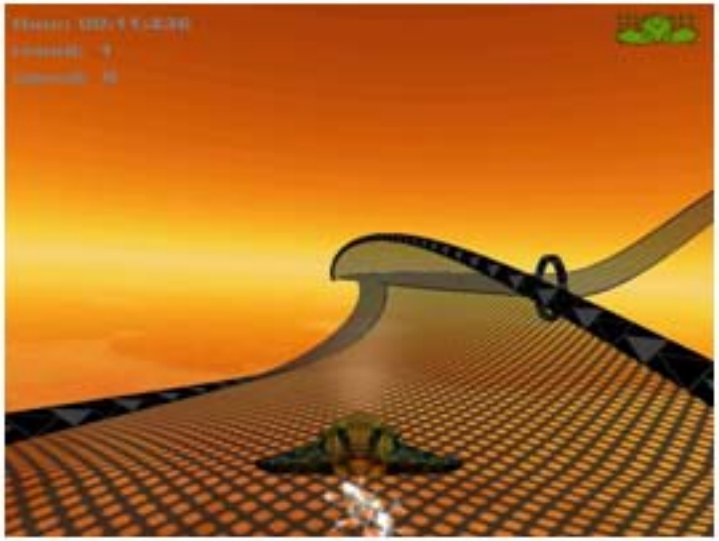


2.5 F5 - Verschiedene Kameraperspektiven

Umsetzung

- Streckenpostenkamera
- Stoßstangenkamera
- Verfolgungskamera





2.6 Z1 – Realisierung eines einfachen Menüs

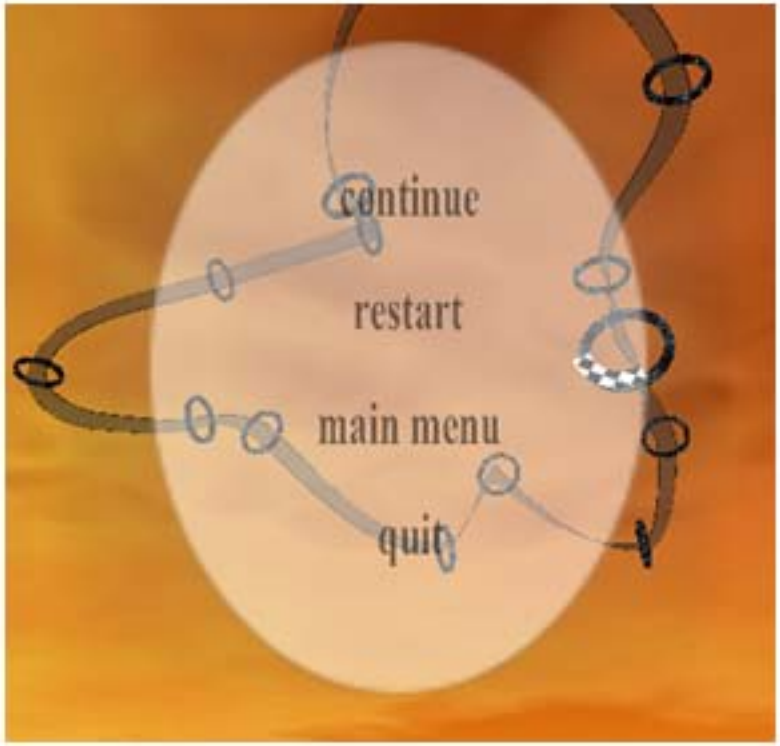
Anforderung

- Benutzerinteraktion mittels Maus
- Transparentes Menu
  - Aktuelle Strecke im Hintergrund sichtbar
- Verschiedene Spielmodi
- Orthogonalprojektion
  - Menu ist immer zur Kamera ausgerichtet mit z=0
  - Position in Bildschirmkoordinaten
  - Eigener Subgraph

2.6 Z1 – Realisierung eines einfachen Menüs

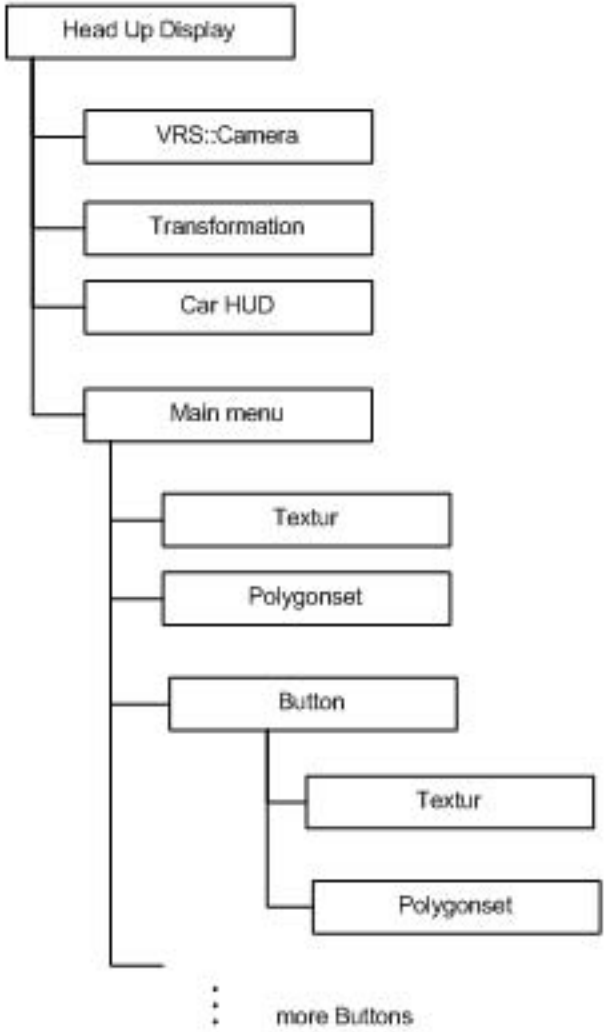
Umsetzung

- Menu Klasse
  - Position, Textur und Instanzen der Button Klasse
  - Darstellung durch Quads
  - durch VRS::Selector Callback wird der Name des Buttons an die Engine weitergeleitet
- Button Klasse
  - pro Button eine Instanz
  - Textur, Position, Name



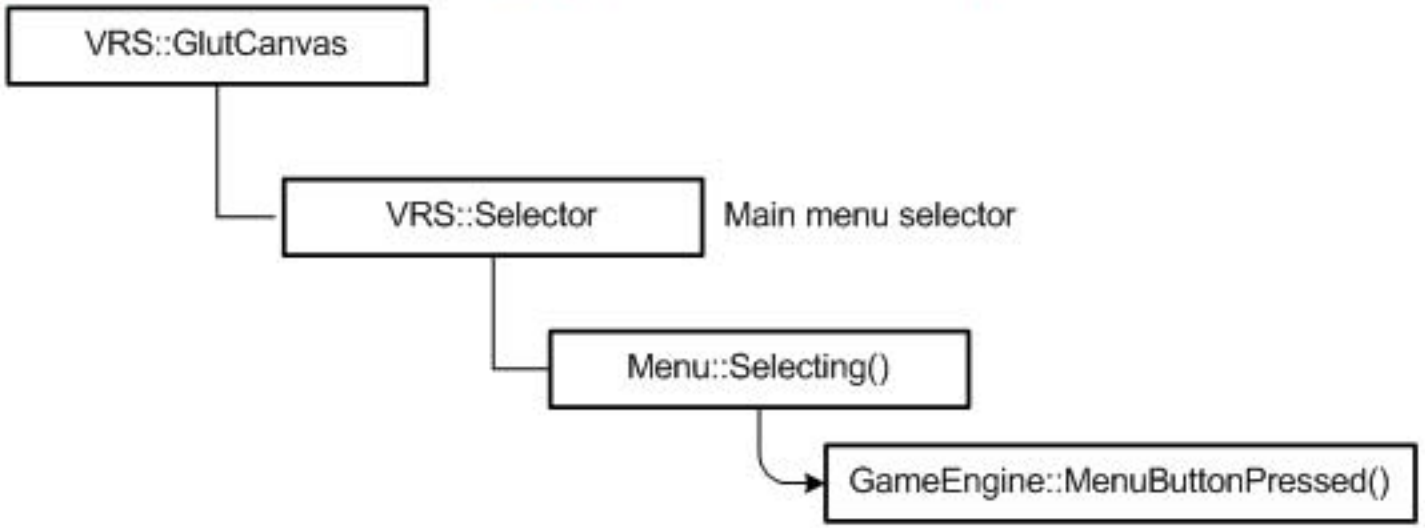
2.6 Z1 – Realisierung eines einfachen Menüs

Szenengraph



2.6 Z1 – Realisierung eines einfachen Menüs

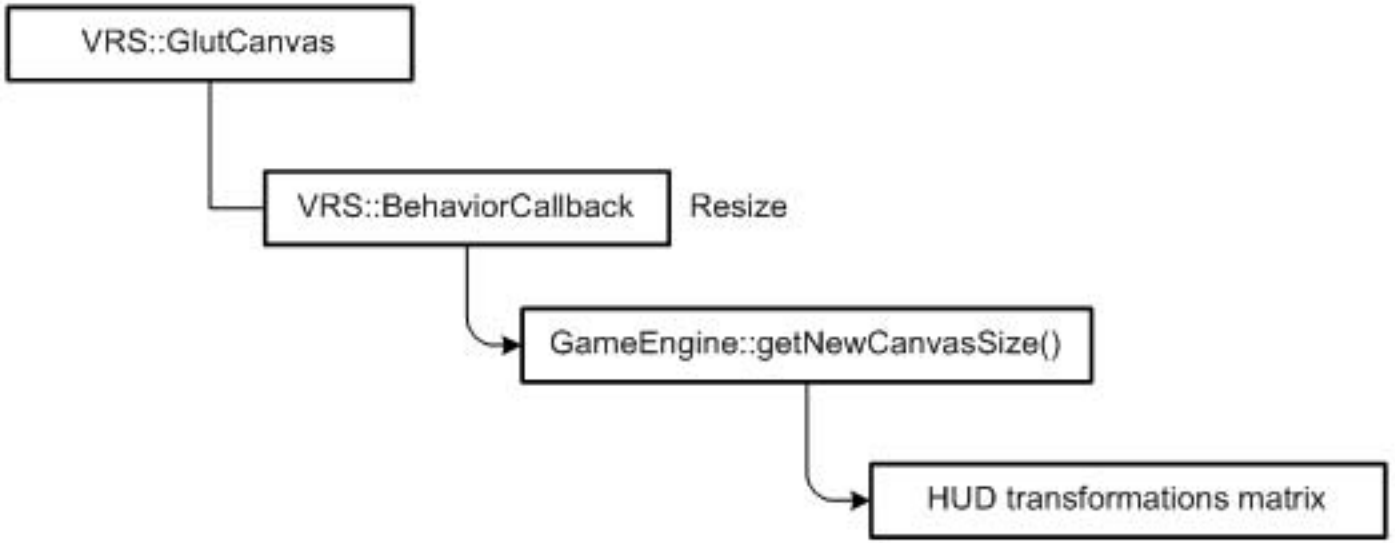
Ereignisbehandlung (Mausklicks)



- Durch RayRequest prüfen welches Shape selektiert wurde
- Name des angeklickten Buttons an GameEngine weiterleiten
- Engine entscheidet je nach Zustand was passiert

2.6 Z1 – Realisierung eines einfachen Menüs

Ereignisbehandlung (geänderte Fenstergröße)



- Bei ResizeEvent berechne neue Transformationsmatrix
  - Einfache Skalierung mit Werten der Fenstergröße
- Die Größe der HUD Elemente ist relativ zur Fenstergröße definiert

2.7 Z2 – Head Up Display HUD

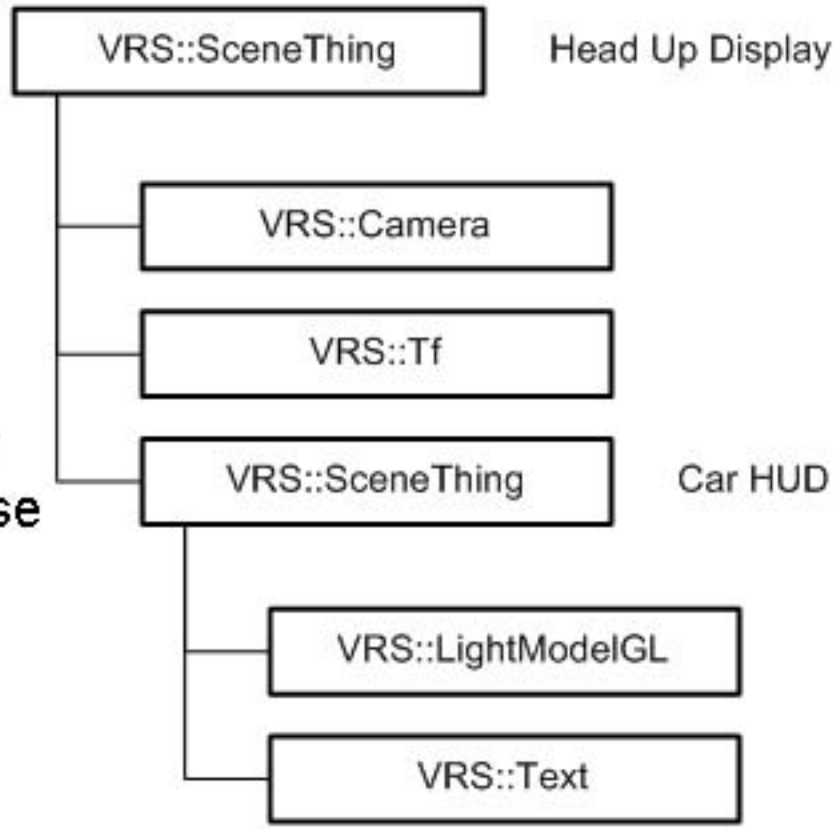
Anforderung

- Anzeige von Statusinformationen für den Spieler
  - z.B. Zeit, Runden, Schaden etc.
- Immer zur Kamera ausgerichtet
  - Orthogonalprojektion

2.7 Z2 – Head Up Display HUD

Implementierung

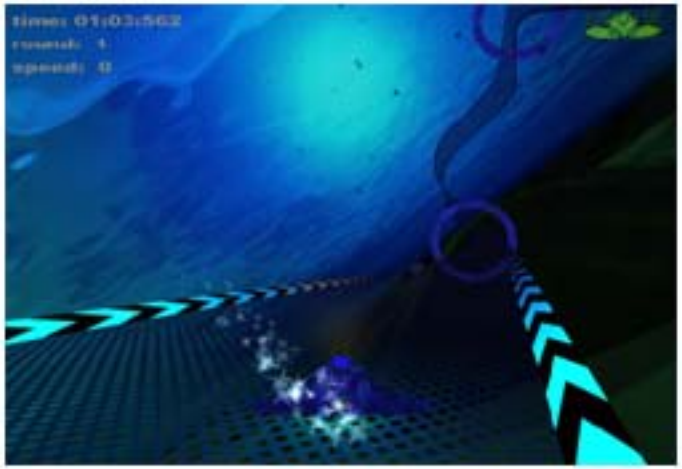
- Orthogonalprojektion
- eigener Subgraph
- Beispiel:
  - Informationen über den Spieler werden von der Player2D Klasse verwaltet und über die Player Klasse mit dem HUD ausgegeben
    - → Player speichert Referenz auf Objekte im Szenengraph



2.8 Z3 - Verschiedene Umgebungsprofile

Anforderung

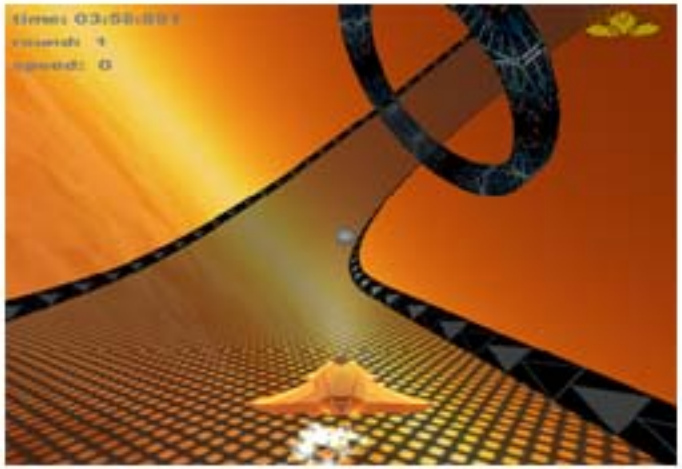
- Implementierung verschiedener Spielumgebungen
- durch Startparameter einstellbar
- 3 verschiedene Profile
  - UnderWater
  - SunWalk
  - Space

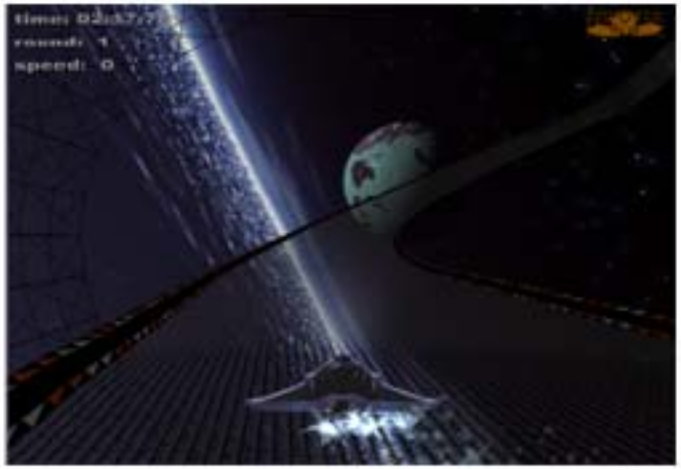


2.8 Z3 - Verschiedene Umgebungsprofile

Umsetzung

- Implementierung einer Klasse ProfileManager zur Abstraktion und Kapselung der Zugriffe auf profilspezifische Objekte
- ein Profil umfasst
  - Beleuchtung
  - Texturen
  - Modelle
  - Partikeleigenschaften





2.9 Z4 - Bonus-Items auf der Rennstrecke

Anforderung

- Elemente, die bei Kollision Effekte hervorrufen

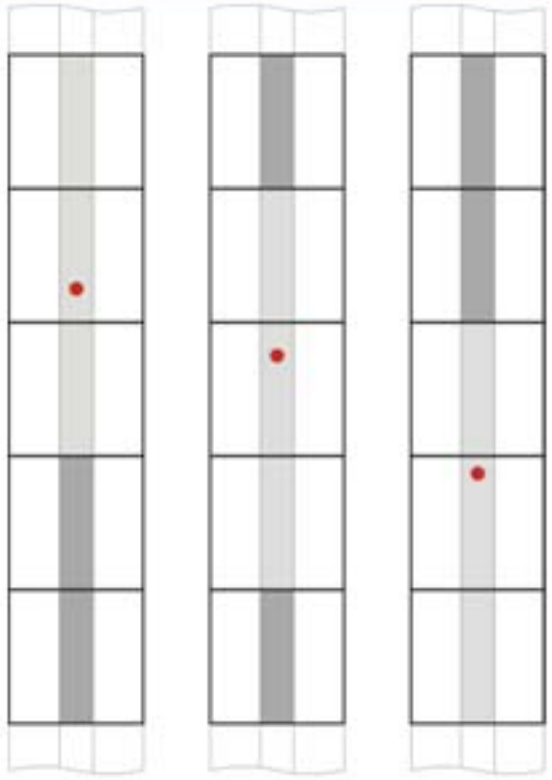
Umsetzung

- 8 verschiedene Bonus-Items
  - Geschwindigkeit erhöhen (+20 Sekunden)
  - Lenkung verbessern (+20 Sekunden)
  - Schaden zufügen
  - Steuerung umkehren (links-rechts)
  - Steuerung umkehren (vorwärts rückwärts)
  - Schaden heilen + Steuerung wiederherstellen
  - Fahrzeug abrupt anhalten
  - Fahrzeugrichtung umdrehen

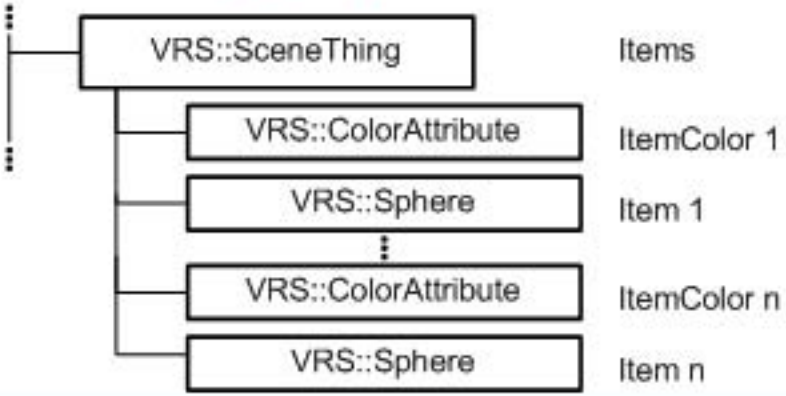
2.9 Z4 - Bonus-Items auf der Rennstrecke

Umsetzung

- Item Manager
  - erstellt und positioniert Items
  - prüft auf Kollision mit Item im Frontend
  - verändert Fahrzeugzustand im Backend



Szenengraph



2.10 Z5 - Zusatzelemente zur Orientierung des Spielers

Anforderung

- Spieler soll sich orientieren können (Fixpunkte etc.)

Andere Grundvorraussetzung

- Strecke 3 dimensional
- Strecke zufällig generiert

→ nur begrenzter Raum um Strecke

2.10 Z5 - Zusatzelemente zur Orientierung des Spielers

Umsetzung

- transparente Strecke
  - es wird nicht viel von der Szene verdeckt
- Ringe um die Strecke
  - Start- / Zielring anderes Aussehen
- schwebende Türme
  - in Blöcken platziert, wo keine Strecke verläuft
  - sind zum Boden hin ausgerichtet
- Skybox
  - erkennt Horizont

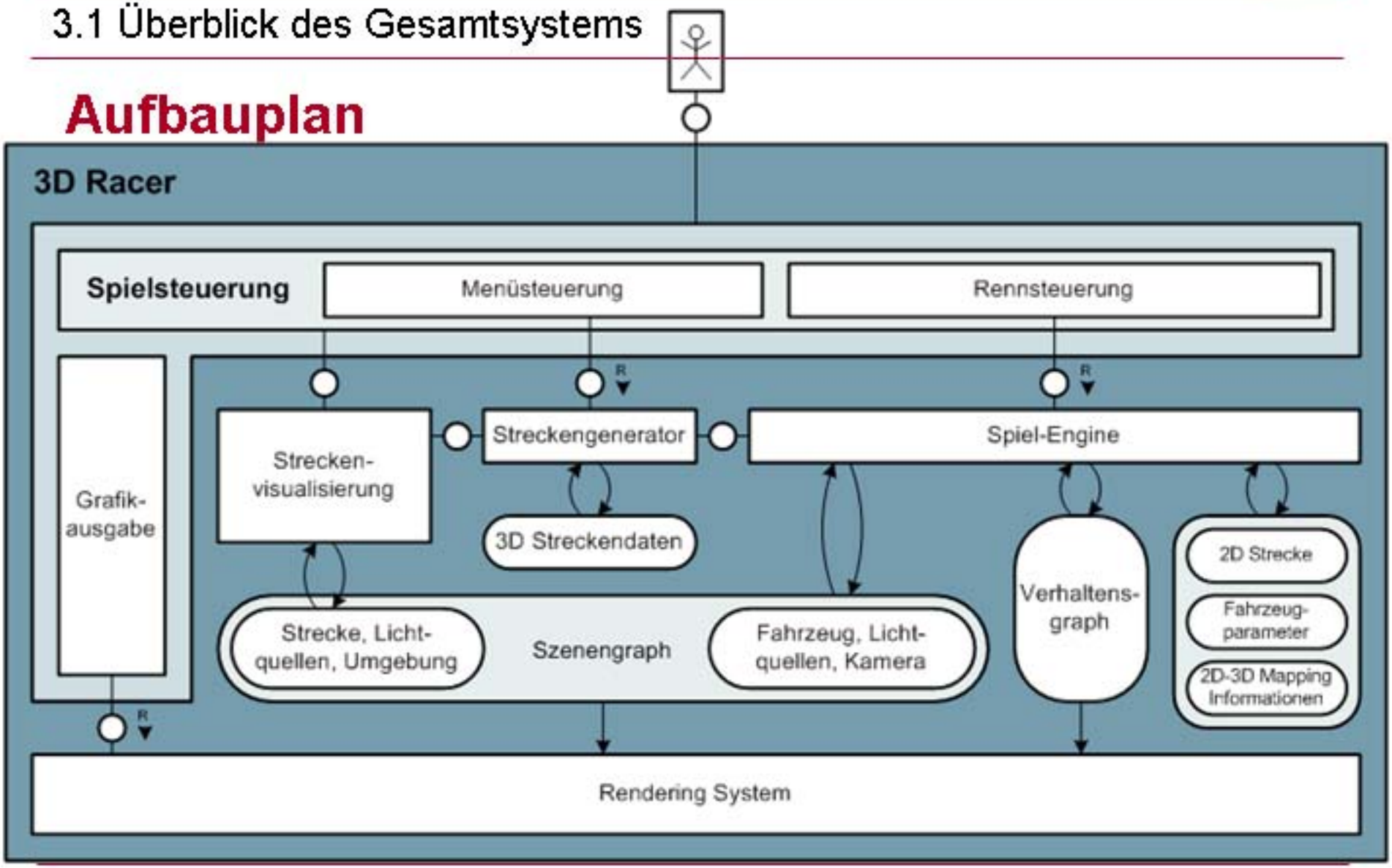
3 Gesamtsystem und Architektur - Überblick

3 | Gesamtsystem und Architektur

- 3.1 Überblick des Gesamtsystems
- 3.2 Szenengraph

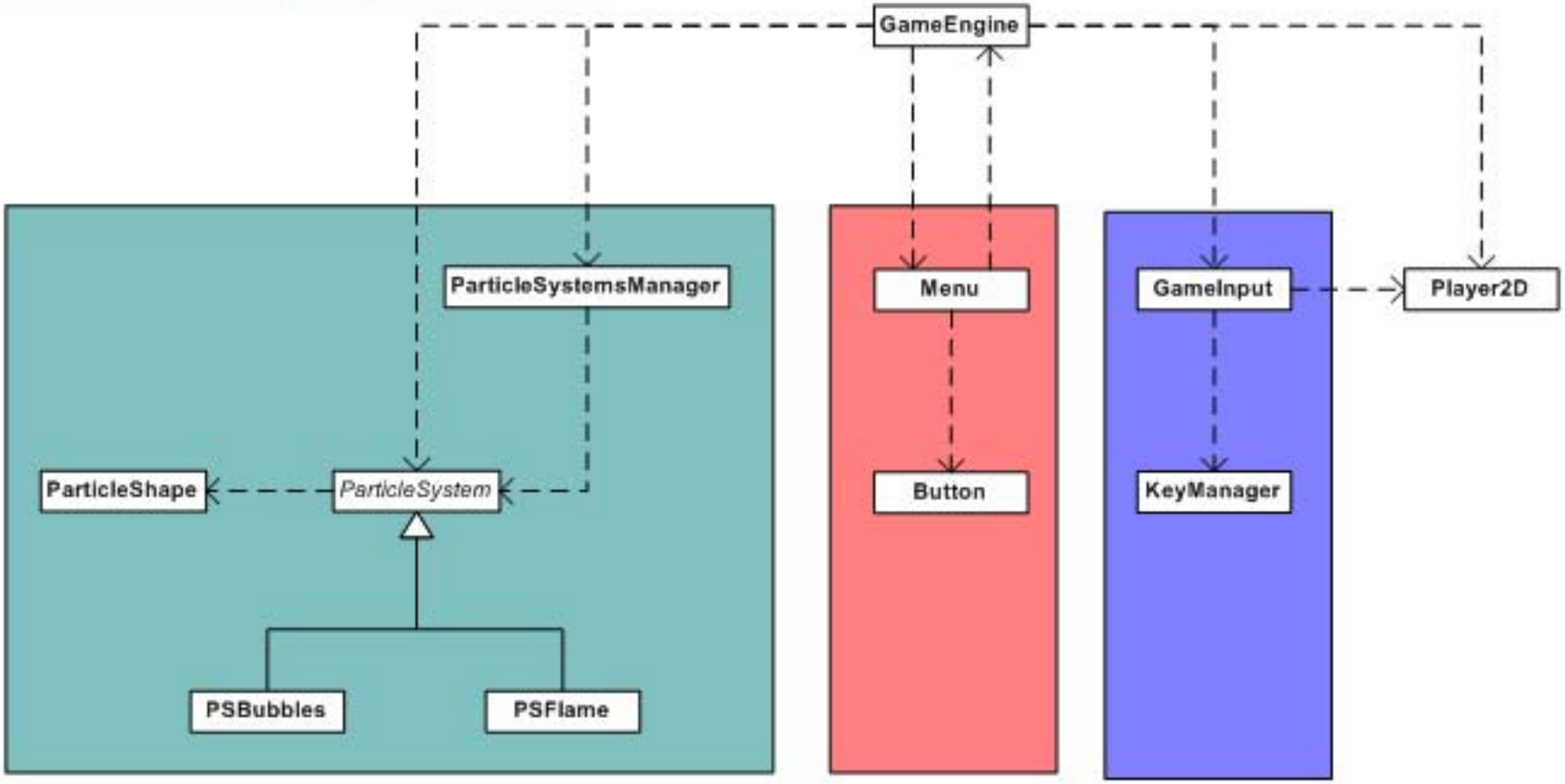
3.1 Überblick des Gesamtsystems

Aufbauplan



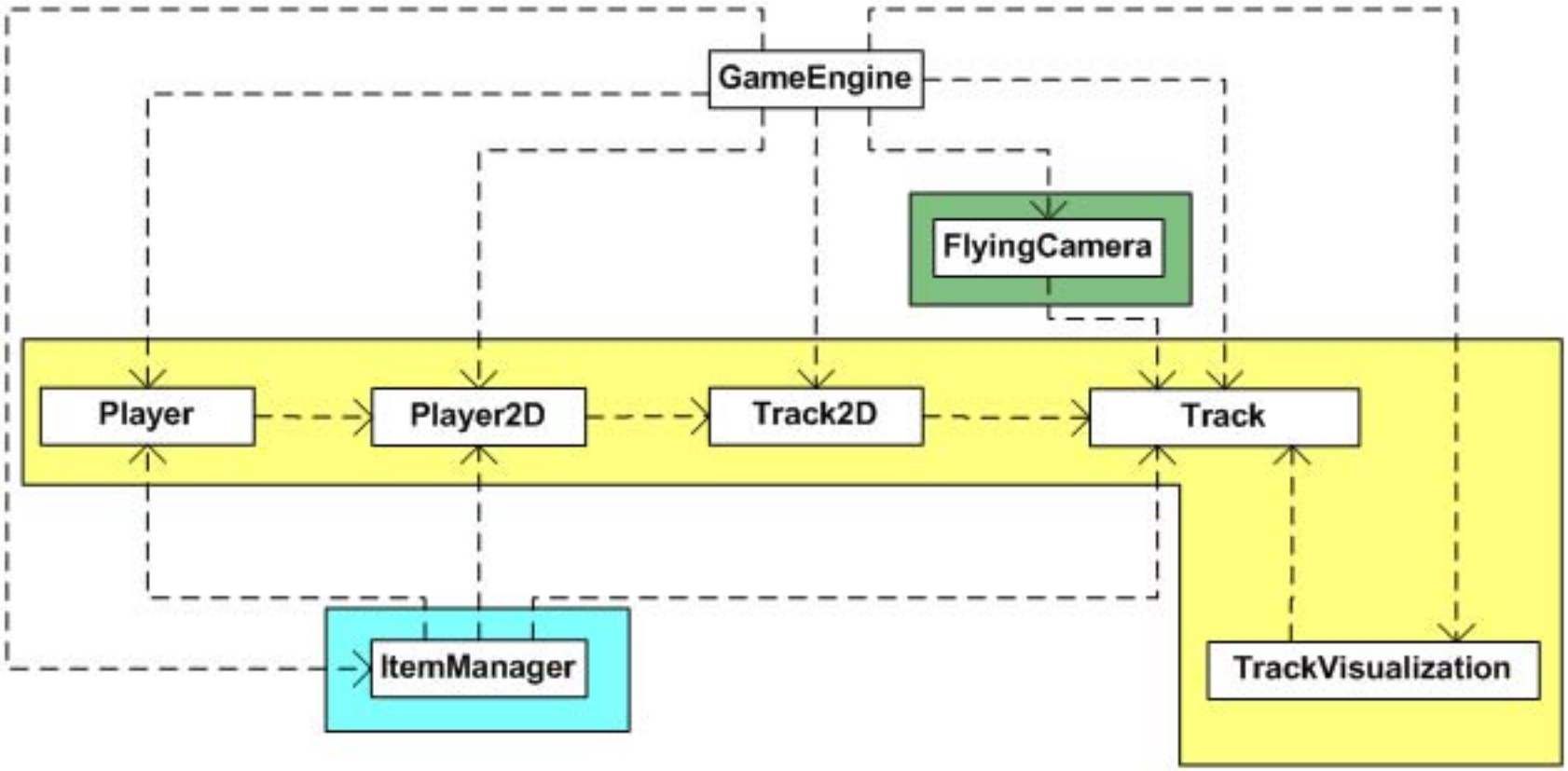
3.1 Überblick des Gesamtsystems

Klassendiagramm



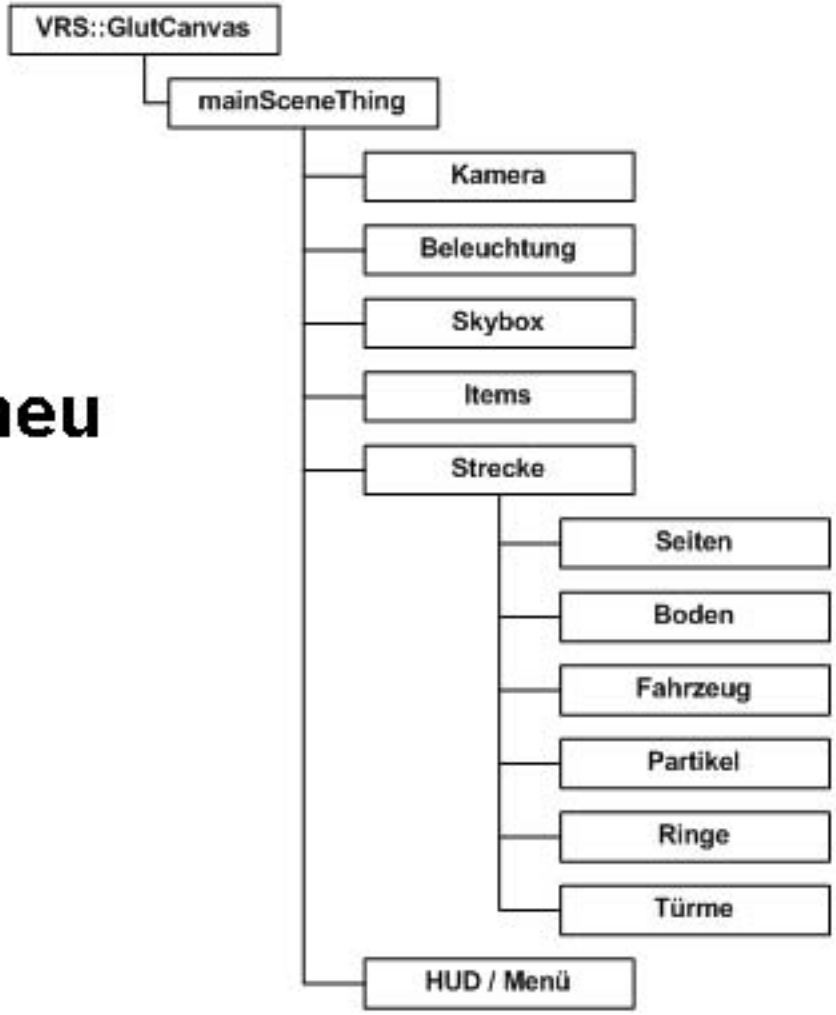
3.1 Überblick des Gesamtsystems

Klassendiagramm



Szenengraphüberblick

- **Streckensubgraph wird bei Neugenerierung gelöscht und komplett neu erstellt**



4 Auswertung und Diskussion – Überblick

4 | Vorführung und Diskussion

- 4.1 Vorführung
- 4.2 Fragen und Diskussion