Technische Hochschule Rosenheim

Entwicklung von Computerspielen: Game Engine

Fakultät Informatik
FWPM

Technische Hochschule Rosenheim

Game Engines
Übersicht

1. Was ist eine Game Engine?
2. Architektur
3. Game Loop

Websites zum Thema GameDev:

- https://itch.io/
- https://www.gamasutra.com/
- https://www.gamedev.net/

Technische Hochschule Rosenheim

Game Engines
Sinn einer Game Engine

- **Ursprünglich wurden Spiele ohne gesonderte GameEngine entwickelt**
  Bzw. Engine war spezifisch zu einem Spiel und die verwendete Hardware
- **Jetzt werden Komponenten getrennt und abstrahiert:**
  Software,
  Mechaniken und Regeln der Welt,
  Spielwelt,
  Input...
- **Ziele:**
  Engine wiederverwendbar machen,
  Verschiedene Hardware ansprechen (XBOX, PC, Smartphones uvm),
  Schnellere und günstigere Entwicklung von Spielen,
  Erhöhte Robustheit erzielen durch stetige Verbesserung der Basis...

Technische Hochschule Rosenheim

Game Engines
Sinn einer Game Engine

- **Folgen:**
  - \- Spiele sehen sich immer ähnlicher
  - \+ Entwickler können sich auf den Spielinhalt konzentrieren
  - \+ Vertrieb von Engine kann zusätzliche einnahmen bringen

Game Engines
Sinn einer Game Engine

- Begriff „Game Engine“ stammt aus den 90ern
  - Erste Verwendung in Ego Shootern wie Doom etc.
- Ende der 90:
  - Lizensierung von Quake und Unreal Engine
  - Anpassbar durch Scriptsprachen
- Spektrum von Game Engines:

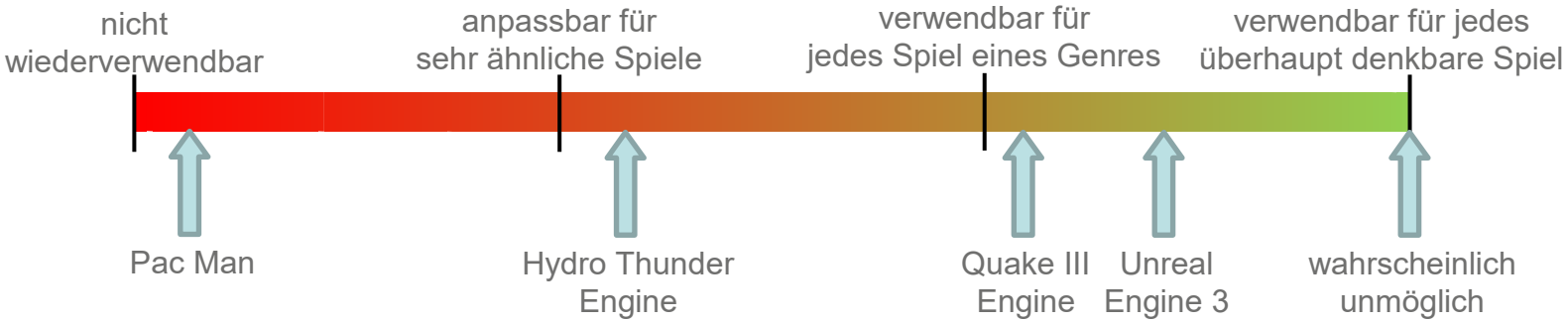


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020
nach J. Gregory: *Game Engine Architecture*, AK Peters, 2009.

Technische Hochschule Rosenheim

Game Engines
Anpassungsfähigkeit einer Game Engine

- Spielegenres haben unterschiedliche Anforderungen.
- Je allgemeiner eine Gameengine ist, desto ineffizienter ist diese in einem spezifischen Anwendungsfall
- Beispiel: *Age of Empires* und *Dirt Rally* haben sehr unterschiedliche Anforderungen:
  - 2D vs 3D,
  - Reduzierte vs Realistische Grafik,
  - Ortographisches vs Perspektifisches Rendering
  - Viele AI Agents vs komplexe Physik

Technische Hochschule Rosenheim

Game Engines
Anforderungen der Genres an Game Engines

- Echtzeitstrategie:
  - „Ameisenhügel“: Simulation der Interaktion von Menschenmassen*
  - Veränderung der Spielwelt während des Spielens
- Plattformer:
  - Dynamische Welt
  - Gute Animation des Hauptcharacters (Spielercharacter)
  - Kamerabewegungen
  - Feinfühliger, (meist) komplexer Character Controller
- Ego Shooter:
  - Effizientes Rendering
  - Physikalisch basierte Animation

Technische Hochschule Rosenheim

Game Engines
Soll man eine Game Engine verwenden?

- Warum?:
  - Aufwand (meist) erheblich reduziert
  - Konzentration auf Spieldesign
  - Eintrittshüde geringer
  - Testaufwand erheblich geringer
  - Viele Tools bereits integriert / verfügbar
- Warum nicht?:
  - Wenig Kontrolle über Implementierung von Features
  - Abhängigkeit von Lizenzierung notwendig
  - Technologien, Tools und andere Module fest vorgegeben und nur schwer / gar nicht zu ändern.

Game Engines
Architektur einer Game Engine

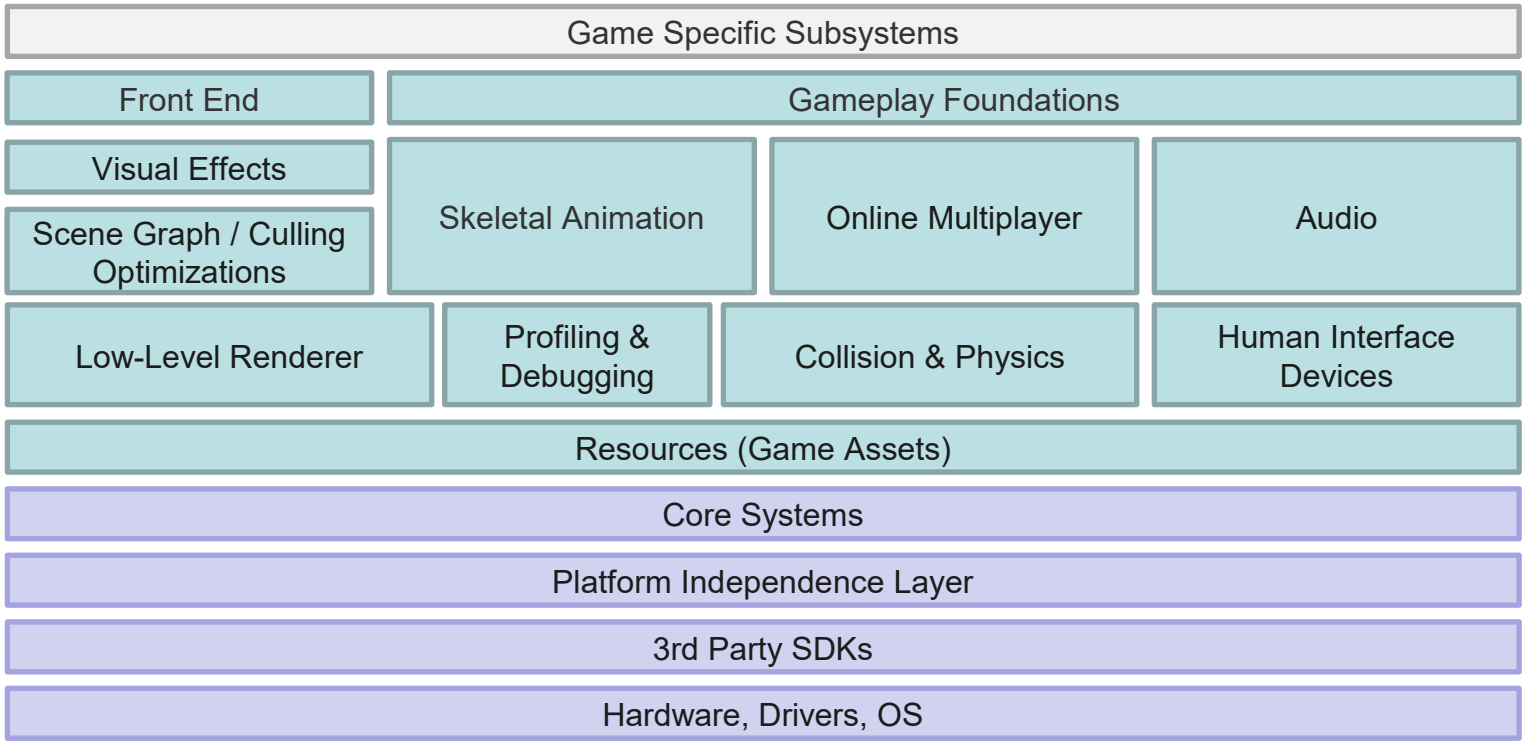


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

Game Engines
Architektur: 3rd Party SDKs

- Physik
  - Havok
  - PhysX (Nvidia)
  - Open Dynamics Engine (ODE, Open Source)
- Grafik
  - OpenGL ,OpenGLES , Vulkan
  - DirectX

Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

Technische Hochschule Rosenheim

Game Engines
Architektur: Plattform Independent Layer

- Spiele sollen häufig auf mehreren Plattformen erscheinen
  => konsistentes verhalten auf mehreren Plattformen gewünscht
- Deshalb:
  Wrapper für C Standardbibliothek, Betriebssystemaufrufe, Input...

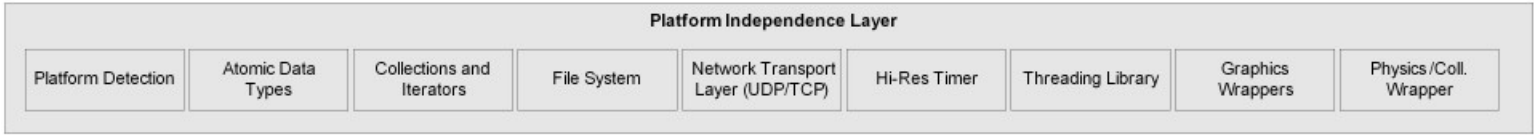


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

Technische Hochschule Rosenheim

Game Engines
Architektur: Core Systems

- Funktionen und Module, die anwendungsunabhängig typischerweise benötigt werden
- Beispiel Mathematikbibliotheken
  Spiele sind **SEHR** Mathematiklastig
  (Vieles von dem man glaubte es nie zu brauchen, findet hier Verwendung (und mehr))

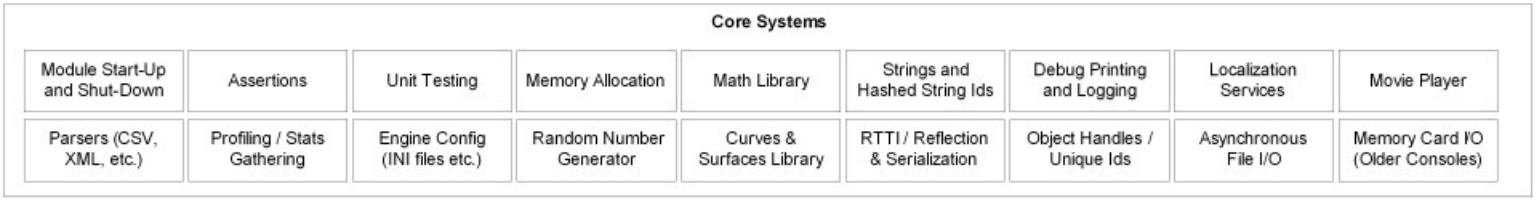


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

Game Engines
Architektur: Game Assets (Grundbausteine)

- Je nach Engine mehr oder weniger
- Verarbeiten von Daten:
  - Bilder
  - Fonts
  - Videos
  - Karten
  - ...

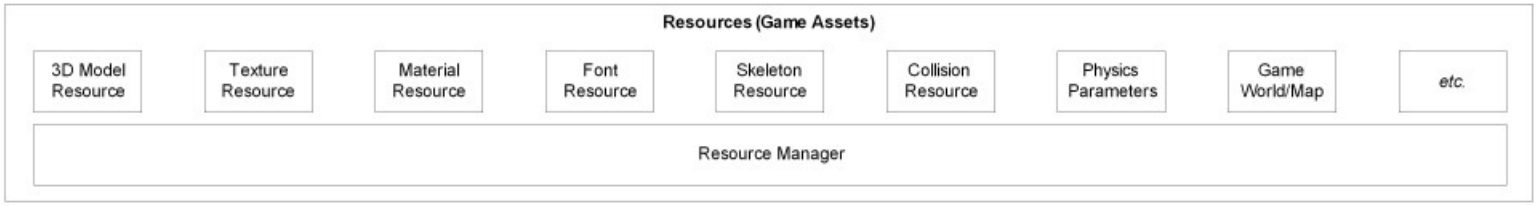


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

Technische Hochschule Rosenheim

Game Engines
Architektur: Rendering Pipeline

- SEHR großer Teil, einige Aspekte davon im weiteren Verlauf genauer
- Viele verschiedene Ansätze möglich
- Orientierung an Grafikhardware
- Typische Architektur:

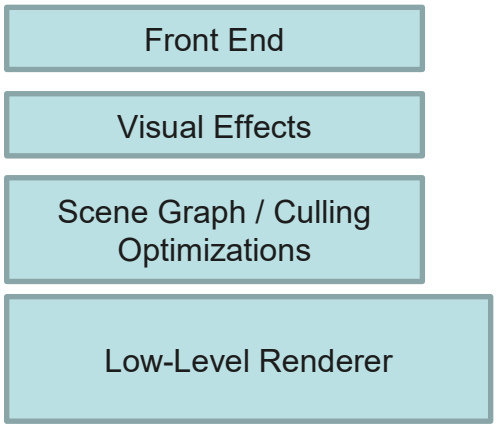


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

Game Engines
Render Pipeline: Low level Renderer

- Schnelles Rendern von Grafikprimitiven (Kugel, Würfel etc.)
- Bildet Schnittstelle zu Grafischen Api's (OpenGl, DirectX etc.)

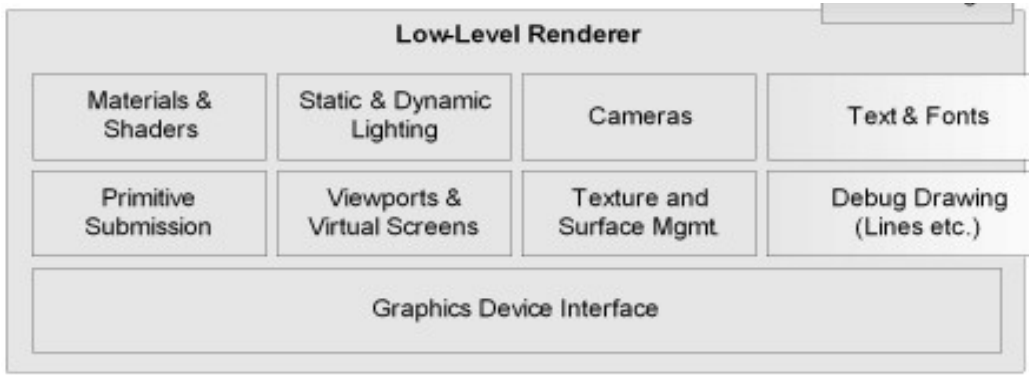
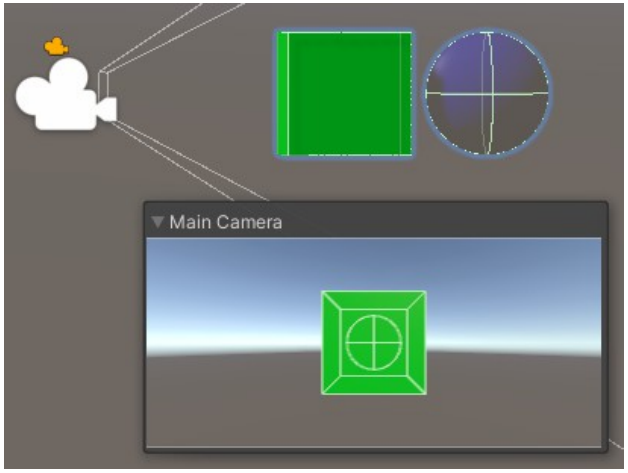


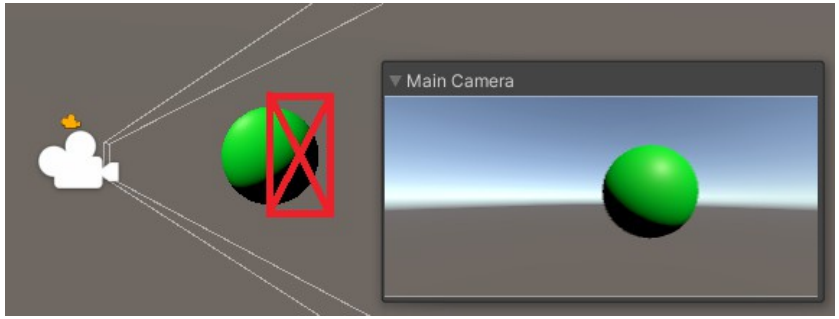
Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

Game Engines
Render Pipeline: Screen Graph Culling Optimizations

- Low level Renderer zeichnet alle Primitven, auch welche die durch andere verdeckt und daher nicht sichtbar wären. (Ausnahme: Back Face Culling und clipping)



Occlusion Culling



Backface Culling

Technische Hochschule Rosenheim

Game Engines
Render Pipeline: Screen Graph Culling Optimizations

- Scene Graph enthält Informationen zur räumlichen Struktur der zu zeichnenden Szene
  - BSP Baum (Binary Space Partition)
  - Quadtree / Octree

=> Entfernung von Objekten, welche in der Szene nicht sichtbar sind wird dadurch hier auf dieser Verarbeitungsebene möglich.

Game Engines
Render Pipeline: Visual Effects

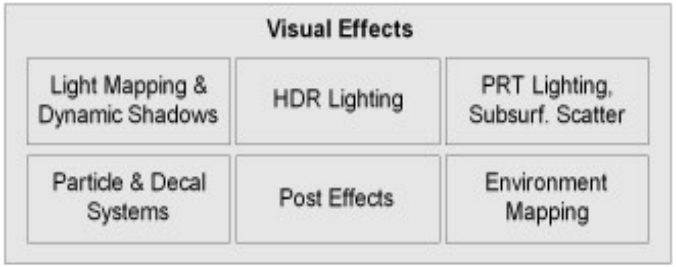
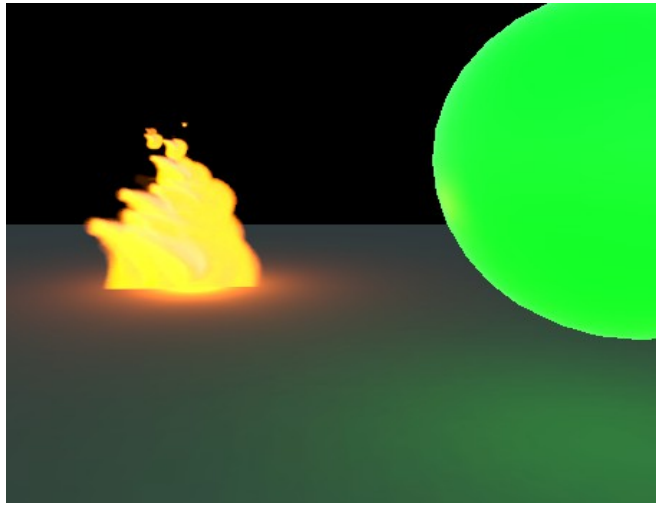


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

- Unterstützt viele Arten von visuellen Effekten:
  - Partikelsysteme (Feuer, Rauch, Zaubersprüche etc...)
  - Farbanpassungen
  - Beleuchtung durch Texturen
  - Schatten



Feuer Partikelsystem
und
HDR Lighting

Technische Hochschule Rosenheim

Game Engines
Render Pipeline: Front End

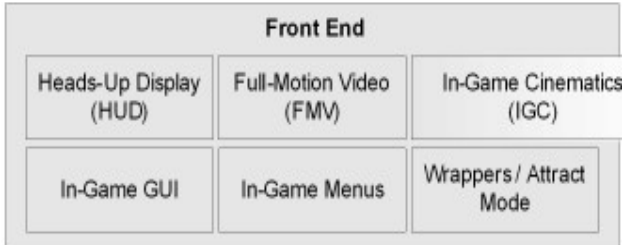


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

- 2D Overlay
  - GUI
  - HUD (Head Up Display)
  - Debugging Informationen
- Zusätzlich:
  - Filme im Vollbild
  - Vorgerenderte Cutscenes, ohne Spielereinfluss

HUD vs GUI:
HUD stellt Informationen beim spielen bereit
GUI empfängt Input von User (interagierbar)

Game Engines
Profiling & Debugging Tools

- Auswertung von Speicherverbrauch in Echtzeit
- Performance Analyse mit Detailinformation zu den Durchlaufzeiten der verschiedenen Bereiche (Rendering, Physics, Mechanics …)
- Abspeichern der Profiling Daten
- Aufzeichnung und Wiedergabe des Spiel Verlaufs

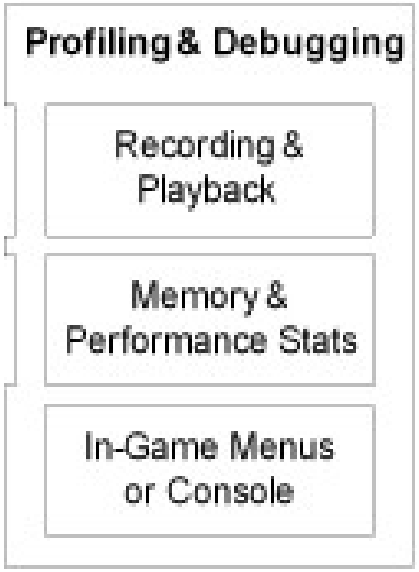


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

Technische Hochschule Rosenheim

Game Engines
Kollisionserkennung & Physik

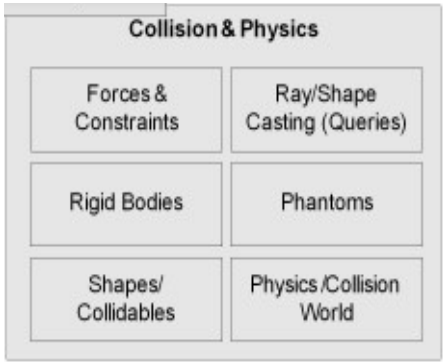


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

- Kollisionserkennung in (fast) allen Spielen sehr wichtig.
  Kollisionen regeln nicht nur, dass Objekte sich nicht durcheinander bewegen, auch das Einsammeln von Münzen oder Waffen wird darüber geregelt.
- Simulation von Beschleunigungen, Impuls und Gewicht von Objekten
  => Annäherung an realistische physikalische Verhältnisse
- Meist in der Engine nur wenig realistisch und grundlegend umgesetzt
  => Verwendung von 3rd Party Tools bei Spielen mit Fokus auf Physik sinnvoll (z.B. Havok)

Game Engines
Animation

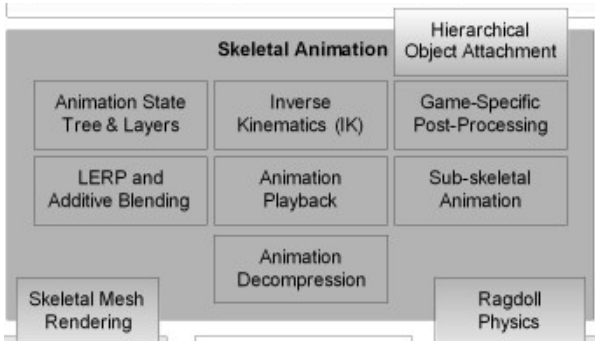


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

- Animation von Zeichentrickfiguren, Menschen, Mobs, Bäumen...
- Die üblichsten Methoden:

2D:

Animation über Abfolge von Sprites / Texturen

3D:

- Kombination Starrer Körper und Bewegungsabfolgen untereinander
- Animation per Meshmanipulation über Skeletsysteme und Manipulationsgewichtung (Menschen z.B.)

Technische Hochschule Rosenheim

Game Engines
HID – Schnittstelle zum Benutzer

Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

- Meist:
  - Gamepad
  - Maus
  - Tastatur
  - Lenkrad
- Selten, aber immer öfter:
  - VR Controller
  - Kinect / Kameratracking Systeme
  - Handtracking

Die Meistverwendeten HIDs werden in aller Regel bereits von der Game Engine unterstützt

Seltenere HIDs benötigen eine Einbindung des SDKs des Herstellers

In (fast) allen Fällen ist die Eingabe => Reaktion Relation nicht fest definiert. Dies ist je nach Spiel selbst zu definieren.

Technische Hochschule Rosenheim

Game Engines
Audio

Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

- Sehr wichtiger Teil, zeichnet bei richtiger Verwendung etwa die Hälfte* des „Bildes“:
  - Hintergrundmusik
  - Effekte
- Aufgaben:
  - Mischen
  - Modulieren
  - Raumklang
- Auch hierfür verwendung von 3rd Party Engines nicht unüblich:
  - Irrklang, OpenAL (beide Open Source)
  - XACT (Microsoft, XBOX, PC)
  - FMOD (Firelight Technologies)

*nicht als Wert zu verstehen, ist nur sinnbildlich gemeint

Game Engines
Netzwerk

Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

- Bis auf wenige Ausnahmen: Verwendung als Technologie für Mehrspieler (Multiplayer) Umsetzungen.
- Aufgaben:
  - Reproduktion der Spielwelt und deren Zustand auf allen beteiligten Systemen.
  - Managment von Objektbesitz (z.B. Wer darf meinen Char steuern?)
  - Gruppierung von Spielern in Lobbys und Sessions (Matchmaking)
- Auch hier gibt es 3rd Party Tools/ Engines:
  Mirror; Photon PUN (Beide für Unity)
  RakNet
  GNet

Die Auslegung eines Spiels(und damit auch Engine) auf Online MP muss von Anfang an berücksichtigt werden.

Online MP => Offline SP/MP ist einfach
Offline SP/MP => Online MP sehr schwierig

Game Engines
Gameplay Foundations

- Gameplay:
  - Regeln der Spielwelt
  - Interaktion zwischen Spieler und Objekten
  - Spielziel
- Implementierung in Sprache der Engine / Scriptsprachen; Manchmal beides
- Gameplay Foundations bildet Schnittstelle zwischen Gameplay Code und Low-Level Schichten der Engine

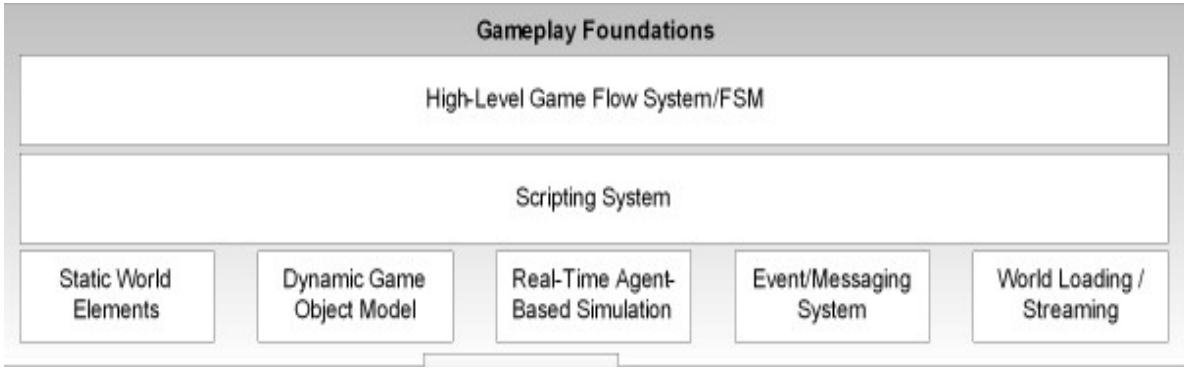


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

Technische Hochschule Rosenheim

Game Engines
Gameplay Core

- Spielwelt und Objektmodelle
- Ereignisverarbeitung
- Scripting System
- Künstliche Intelligenz

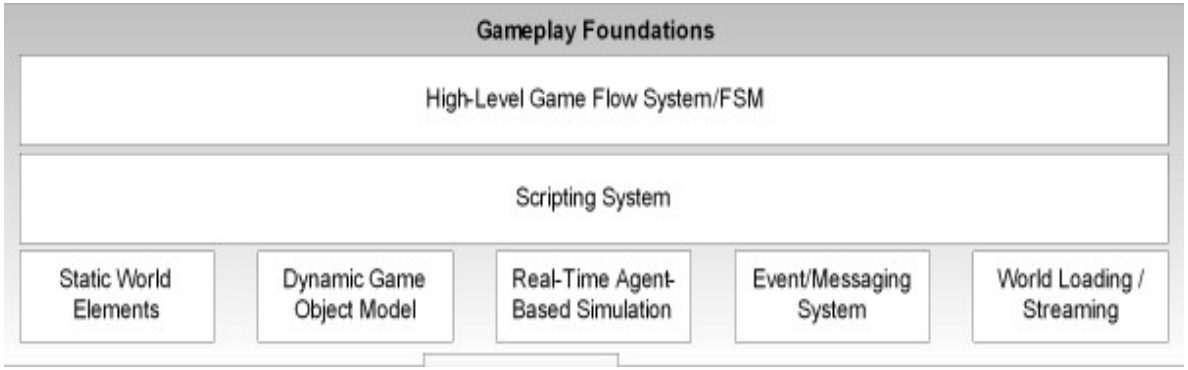


Abbildung aus Folien von Prof. Dr. J. Schmidt, EVC, 2020

Technische Hochschule Rosenheim

Game Engines
Gameplay Spielwelt und Objektmodelle

- Inhalt der Spielwelt meist Objektorientiert Modelliert
- Unterscheidung in der Welt zwischen:
  Dynamischen Objekten (können sich bewegen)
  Statischen Objekten (stehen fest)
- Statische Objekte (für gewöhnlich):
  Haus, Lampe, Straße, Brücken, Bäume...
- Dynamische Objekte (für gewöhnlich):
  Auto, Coladosen, Spielercharactere, Ball...
- Unterscheidung der Objekte ist wichtig, da diese Kategorisierung für diverse (automatische) Optimierungsansätze benötigt wird.
  Beispiele: Occlusion Culling, Licht und Schatten Vorberechnung, Reflection Baking...

Technische Hochschule Rosenheim

Game Engines
Gameplay Ereignisverarbeitung

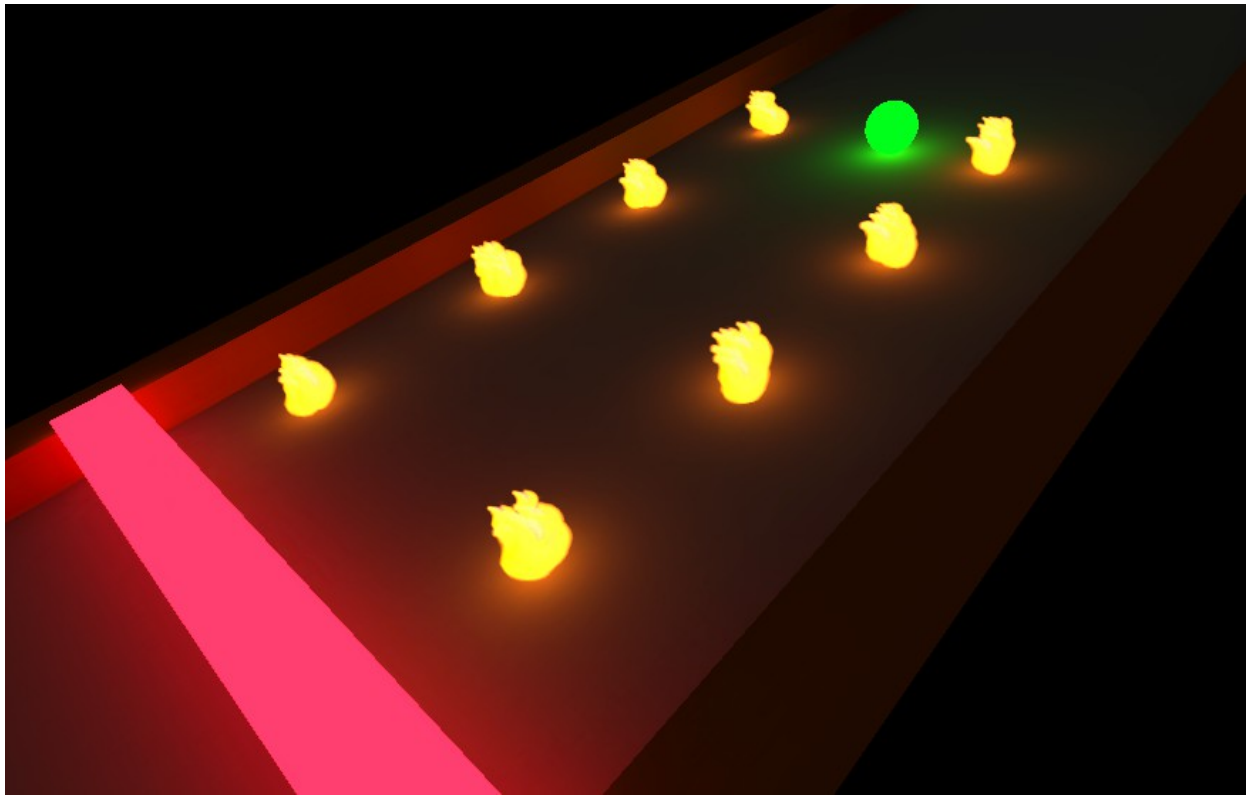
- Zur Kommunikation zwischen Objekten (z.B. Spieler wird von Pfeil getroffen)
- Typische Varianten:

  Sendendes Objekt hat oder erhält Referenz auf das empfangende Objekt und ruft eine Methode/Funktion auf diesem auf.

  Ereignisorientiert: (wie in GUIs) Sender erzeugt ein Ereignis (event), Empfänger hat eine Ereignisverarbeitung (Eventhandler)

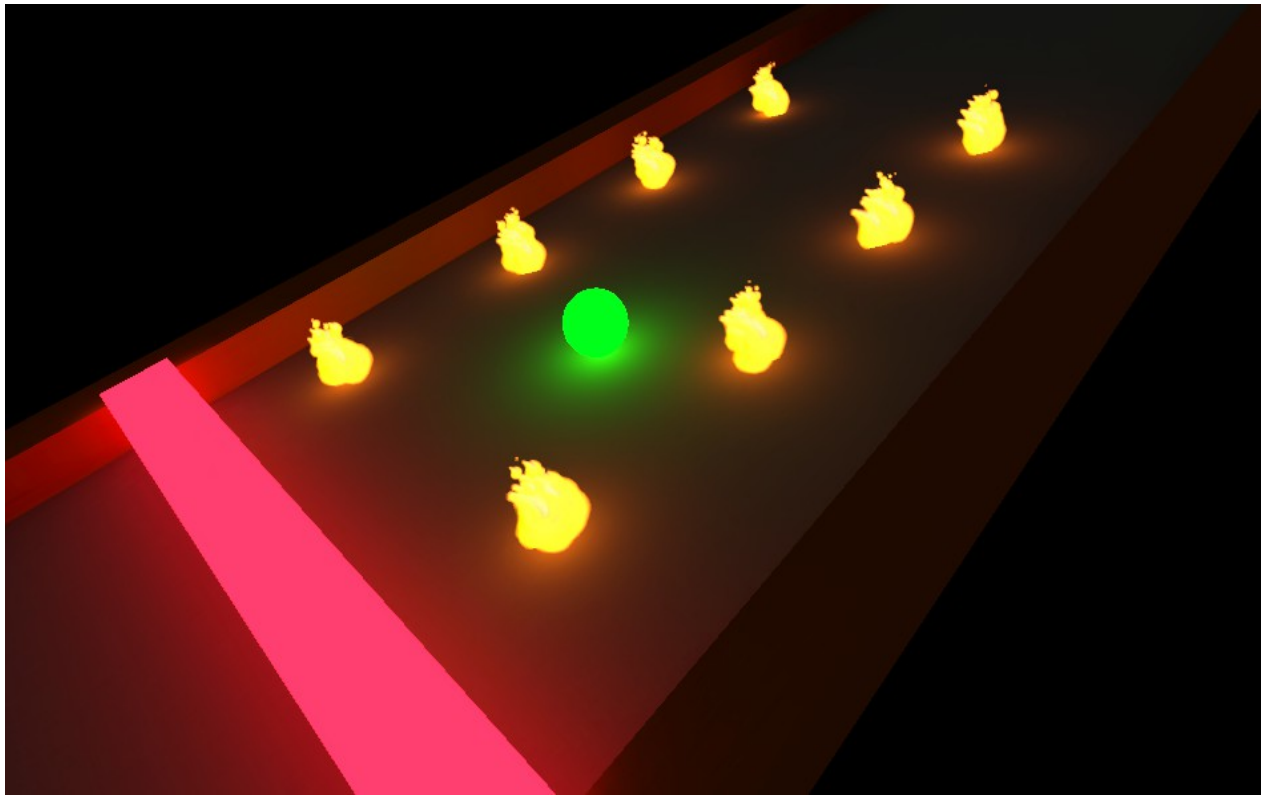
Technische Hochschule Rosenheim

Game Engines
Gameplay Ereignisverarbeitung



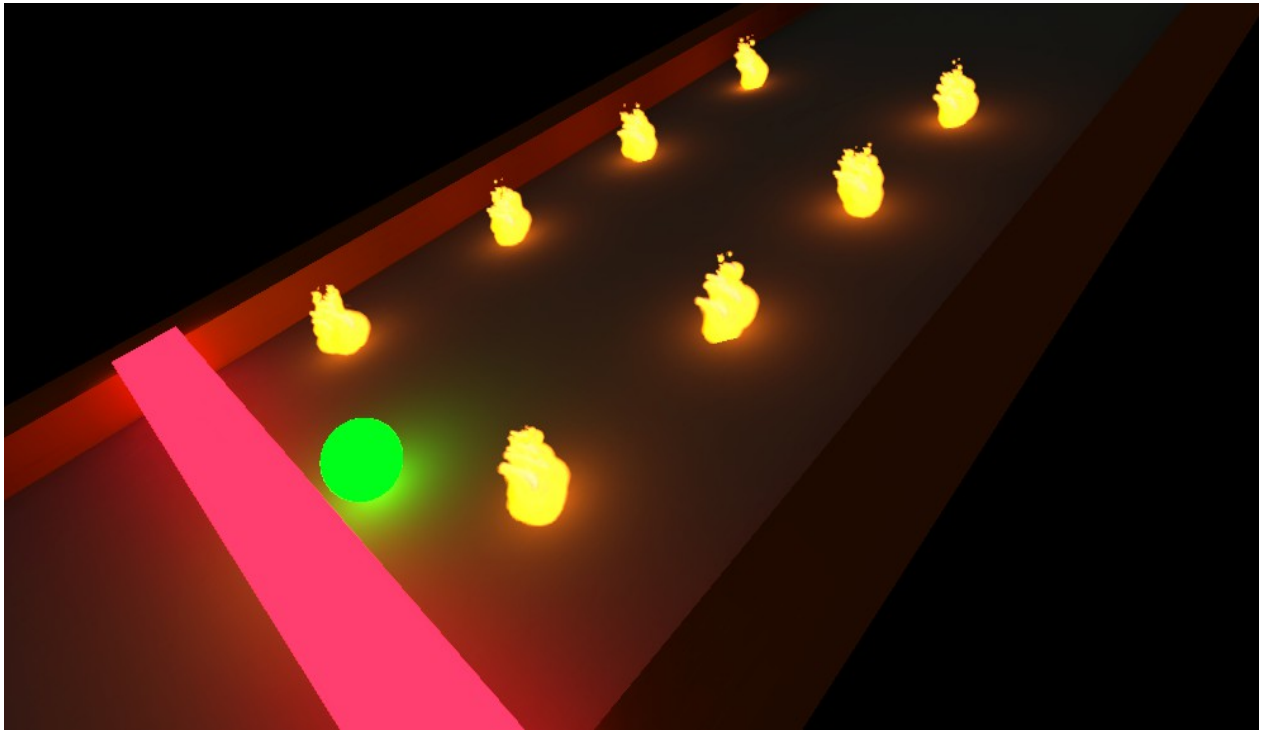
Technische Hochschule Rosenheim

Game Engines
Gameplay Ereignisverarbeitung



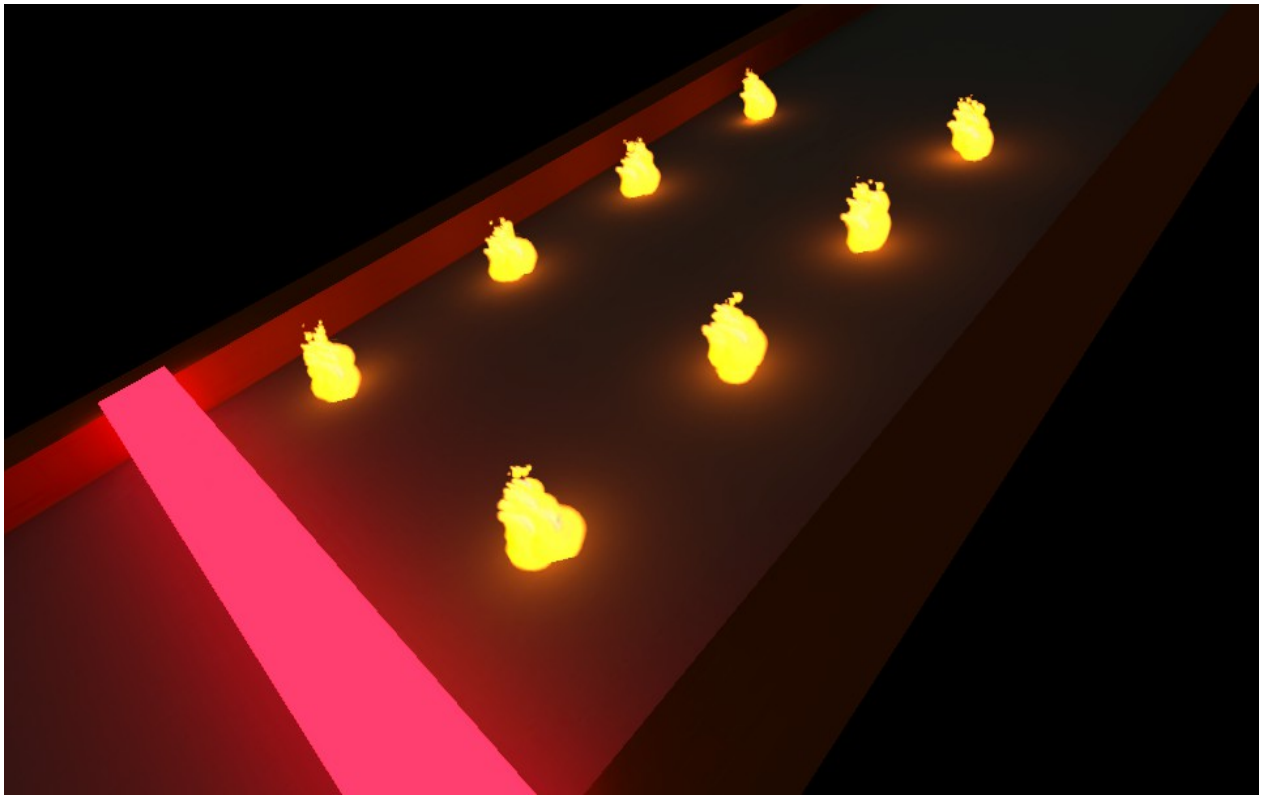
Technische Hochschule Rosenheim

Game Engines
Gameplay Ereignisverarbeitung



Technische Hochschule Rosenheim

Game Engines
Gameplay Ereignisverarbeitung



Game Engines
Gameplay Ereignisverarbeitung

```
public class DeadlyObject : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        Mortal mortal = other.GetComponent<Mortal>();
        if(mortal != null)
        {
            mortal.Die();
        }
    }
}
```

Die grüne Kugel (mit Sphere Collider und Rigidbody) rollt in den Trigger des Roten Quaders. Dadurch wird das Event ausgelöst

Über das Event bekommt der Todesquader die Referenz zur Kugel und kann sie so auffordern sich selbst zu zerstören.

```
public class Mortal : MonoBehaviour
{
    public void Die()
    {
        Destroy(this.gameObject);
    }
}
```