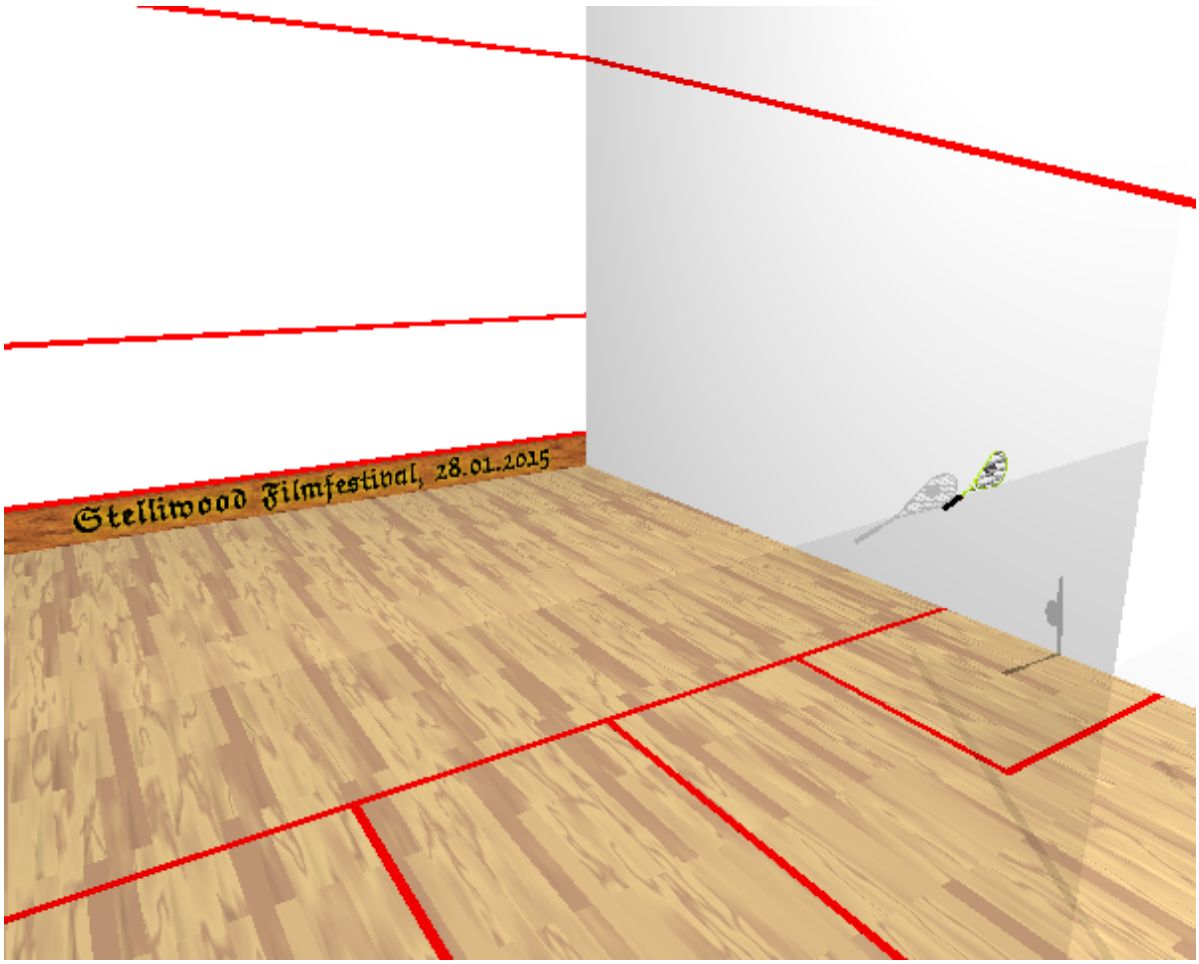
IVC Projekt "InvisiSquash" Dokumentation

Matthis Hauschild und Niels Porsiel

January 26, 2015



1 Anforderungen

- Wenigstens ein Objekt in der Szene sollte bewegte Gliedmaßen haben.
- Wenigstens eine Szene sollte die Kameraeinstellung variieren, z.B. in die Szene hineinfahren, schwenken oder zoomen.
- Setzen Sie ein verarbeitetes Bild ein, z.B. als Höhenprofil, Kulisse oder Textur.
- Erstellen sie einen animierten Titel.
- Setzen Sie Überblendungen ein, um den Schnitt zwischen zwei Kamerapositionen oder anderen Bildwechseln zu betonen oder zu kaschieren.
- Setzen Sie Sound-Effekte zum Vertonen ein.

2 Handlung

Wir setzen ein Squashspiel mit schwebenden Squashschlägern ohne Spieler in PovRay um. Zwei Zuschauer kommen aus einem Eingang in die Halle und setzen sich auf die Tribüne vorm Squash Court. Die Kamera fliegt durch die Courttür in den Court. Die Tür schließt sich und das Spiel beginnt. Nach ein paar Ballwechseln ist das Spiel vorbei und die Kamera zeigt die jubelnden Zuschauer.

3 Technisches

3.1 Grundlegendes und Konventionen

- Eine PovRay Distanzeinheit von 1.0 entspricht 1.0 Meter in der Realität
- Eine PovRay Clockeinheit von 1.0 entspricht 1.0 Sekunden in der Realität
- Das PovRay Koordinatensystem hat seinen Ursprung in der linken unteren hinteren Ecke des Squash Courts (siehe Grafik 1 und 2).

3.2 Aufbau des Programms

3.2.1 Organisation der Dateien

Jede Szene hat eine eigene `pov` und `ini` Datei, die alle den Präfix `animation` tragen. So kann jede Szene ihre eigenen Einstellungen haben und einzeln gerendered werden. Alle statischen Programmteile, wie die komplette Halle, aber auch das Modell der Zuschauer, des Balles sowie der Schläger befinden sich in einem Unterverzeichnis names `static`. Jede Szene inkludiert die statischen Parts. Dabei muss nur die Datei `main_static.inc`, sowie `figure.inc` inkludiert werden, die alle weiteren Parts enthalten (siehe Grafik 3).

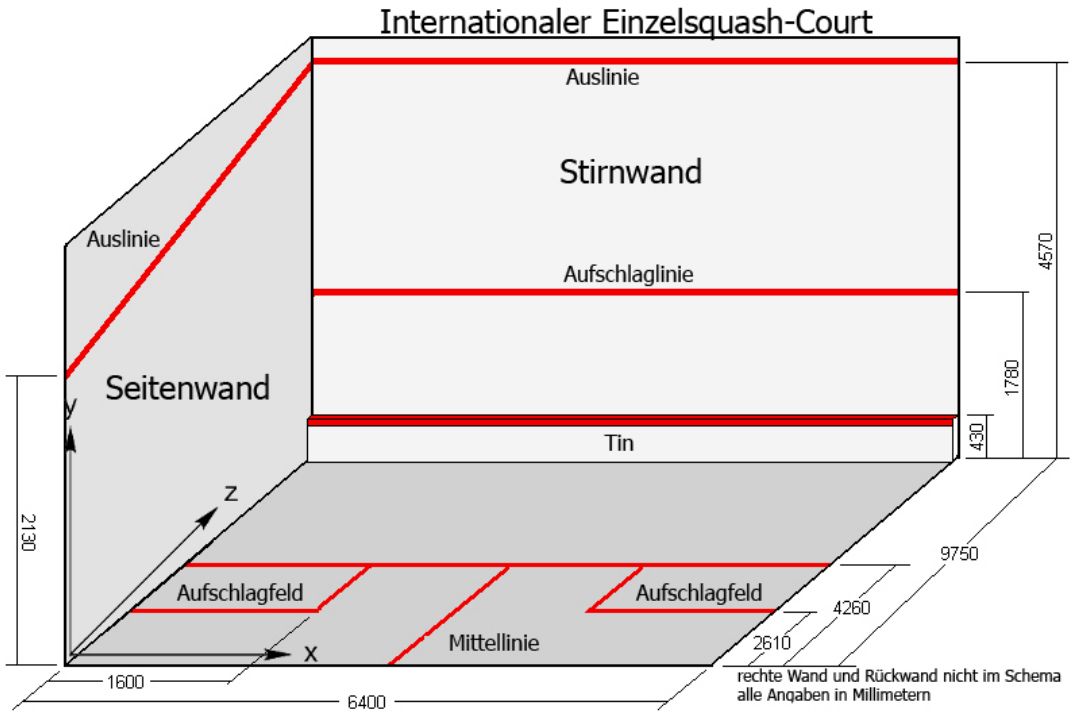


Figure 1: Maße eines Squash Courts + PovRay Koordinatensystem

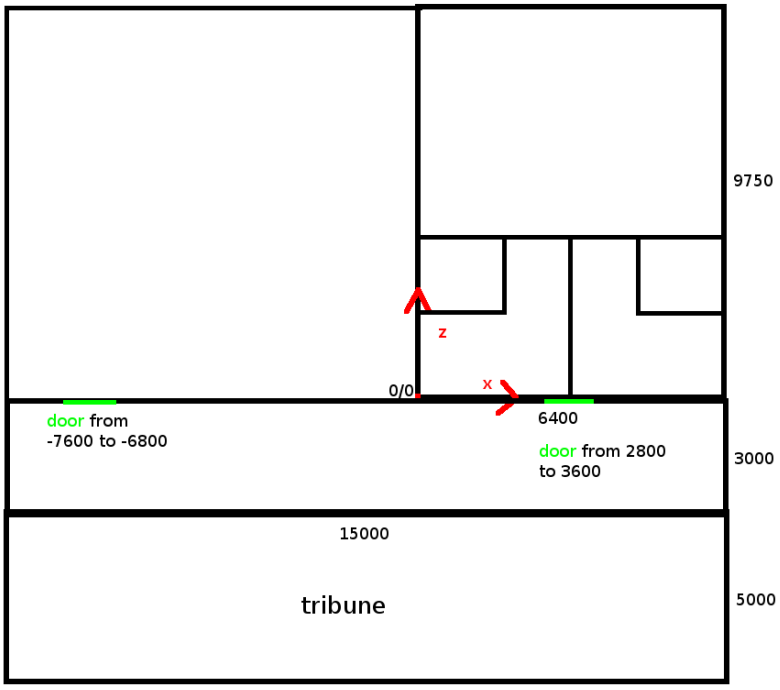


Figure 2: Grundriss der gesamten Karte

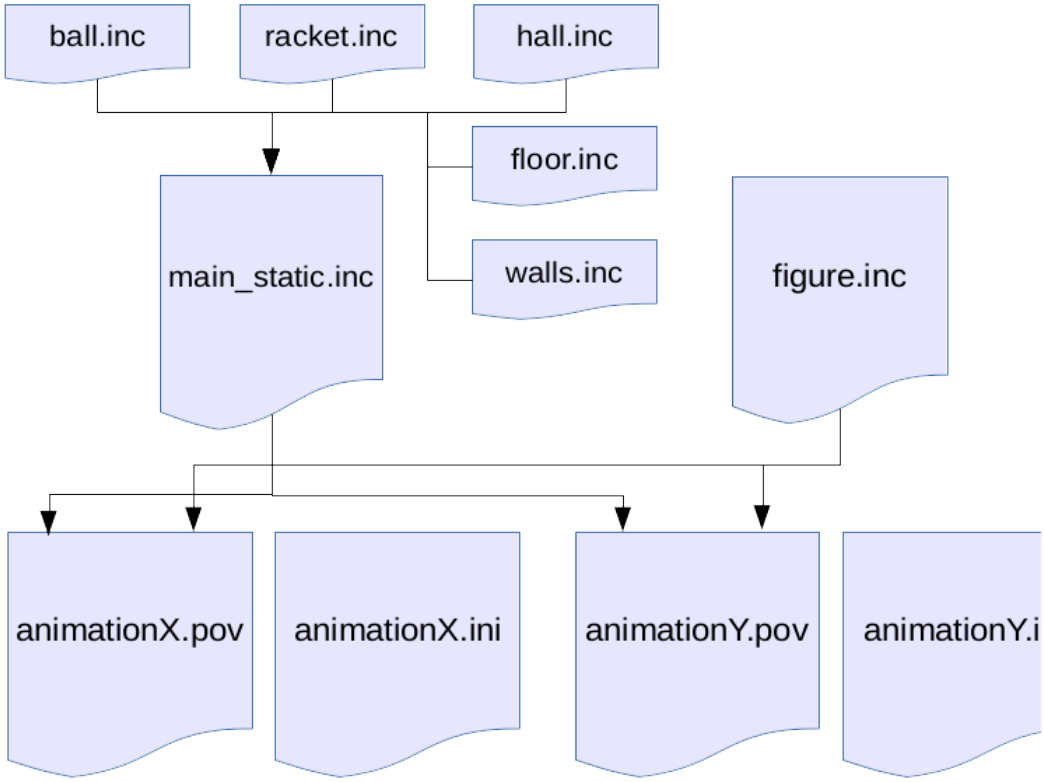


Figure 3: Organisation der Dateien

3.2.2 Schematischer Aufbau einer `animationX.pov` Datei

Die PovRay Clock läuft hier nicht, wie zumeist, von 0.0 bis 1.0 pro Szene, sondern entspricht Sekunden in der Realität. Da wir ein Squashspiel erstellen, ist es von Vorteil, in Sekunden zu rechnen, weil so Animation sowie die Ballgeschwindigkeit realistischer dargestellt werden können.

Um bei Animationen dennoch eine Zahl zwischen 0.0 und 1.0 zu erhalten, wird die Clock in jedem Abschnitt skaliert. Damit bleiben die Vorteile der normalisierten Zeit, nämlich dass Animationen übertragbar sind, sowie leichtere Rechnungen bei der Erstellung der Animationen, erhalten.

Der folgende Quellcode zeigt zusammenfassend die Struktur der Szenen:

Listing 1: Schematischer Aufbau einer animationX.pov Datei

```
#include ...

#switch(clock)
  #range (timeA, timeB)
    #local clk = (clock - timeA) / (timeB - timeA);
    // time dependend animation code like rackets and ball comes
    // here using the locally defined clk variable
  #break
```

```
  #range (timeB, timeC)
    #local clk = (clock - timeB) / (timeC - timeB);
    // time dependend animation code like rackets and ball comes
    // here using the locally defined clk variable
  #break
#end
```

Zunächst kommen die notwendigen includes, anschließlichen wird ein `switch` Statement auf die globale `clock` Variable aufgerufen. PovRay erlaubt im `switch` ein `range` Statement zu benutzen, sodass man Zeitspannen festlegen kann. In jeder dieser Zeitspannen wird die lokale Variable `clk` definiert, die die normalisierte Zeit zwischen 0.0 und 1.0 enthält. So können einfach Animation definiert werden. Falls eine Animation zu langsam ist, kann die Zeitspanne einfach verkürzt werden und die Animation läuft schneller ab.

3.2.3 Die Flugbahn des Balles

Um die Flugbahn des Balles einigermaßen realistisch zu gestalten, wurden die Gleichungen für den schiefen Wurf verwendet. Allerdings wurde aus Gründen der Einfachheit die Reibung nicht berücksichtigt. Ebenso wurde die Abbremsung des Balles an einer Wand oder am Boden lediglich geschätzt.

Die Berechnung der Flugbahn wurde in ein Makro ausgelagert, welches in `ball_trajectory.inc` zu finden ist. Im Laufe der Entwicklung haben wir Mängel festgestellt, sodass auch `ball_trajectory2.inc` und `ball_trajectory3.inc` entstanden sind.

Version2 behebt den Fehler, dass die Richtung des Balles in dem vertikalen Abschlagwinkel durch das Vorzeichen enthalten ist. Dies führt dazu, dass es in Richtung der Wand keine negativen vertikalen Winkel und in Richtung der Glasscheibe keine positiven vertikalen Winkel geben kann. Also wurde die Richtung des Balles in ein extra Parameter ausgelagert.

Weil PovRay sich keine Zustände zwischen dem Generieren von zwei Bildern merken kann, wird für jedes Bild die Flugbahn neuberechnet. Da erschien es am einfachsten, wenn der Ball eine Wand oder den Boden berührt, ab dort eine neue Flugbahnberechnung einzuleiten, indem das Makro erneut mit angepasstem Winkel und Geschwindigkeit aufgerufen wird. Problematisch an dieser Variante ist, dass das Momentum des Ball in Richtung Boden (durch die Erdanziehung), welches ab dem Schlag des Balles immer größer wird, zurück auf 0 gesetzt wird. Dies sorgt für unrealistische Flugbahnen. Version 3 des Makros behebt dies, indem der ursprüngliche Zeitpunkt des Schlages mit angegeben werden kann, woraus die momentane Geschwindigkeit Richtung Boden errechnet wird. Mit dieser Version sind recht realistische Ballwechsel möglich. Das folgende Listing zeigt den Quellcode der 3. Version:

Listing 2: Makro zur Generierung einer Flugbahn des Balles (verkürzt)

```
#macro ball_trajectory3(p_start, v_start, vert_angle, hori_angle,
    ball_direction, local_clock, old_clock, old_v, old_y)
```

```
2   #local x_val =  v_start*local_clock*sin(radians(hori_angle)) +
       p_start.x;
3   #local y_val = old_v*old_clock*sin(radians(vert_angle)) - g/2 *
       pow(old_clock,2) + old_y;
4
5   #if (ball_direction < 0)
6     #local z_val = -v_start*local_clock*cos(radians(vert_angle))
        + p_start.z;
7   #else
8     #local z_val = v_start*local_clock*cos(radians(vert_angle)) +
        p_start.z;
9   #end
10
11  object {
12    ball
13    translate <x_val, y_val, z_val>
14  }
15 #end
```

Die Parameter bedeuten folgendes:

- `p_start`: Die Startposition des Balles (Mittelpunkt des Balles) als 3D-Vektor
- `v_vstart`: Die Geschwindigkeit des Balles
- `vert_angle`: Der vertikale Abschlagwinkel des Balles. Ein Winkel von 0 führt dazu, dass der Ball parallel zum Boden fliegt. Bei einem positiven Winkel fliegt der Ball nach oben, bei einem negativen nach unten.
- `hori_angle`: Der horizontale Abschlagwinkel des Balles. Ein Winkel von 0 führt dazu, dass der Ball parallel zu den Seitenwänden fliegen. Bei einem positiven Winkel fligen der Ball nach rechts, bei einem negativen nach links.
- `ball_direction`: Die Flugrichtung des Balles. Bei 1 fliegt der Ball Richtung Wand, bei -1 fliegt der Ball Richtung Glasscheibe.
- `local_clock`: Zur Berechnung der Flugbahn wird die momentane Zeit benötigt. Wichtig ist, dass die Zeit nicht normalisiert sein darf, weil die Geschwindigkeit auch nicht normalisiert ist und in m/s angegeben wird. Dennoch muss die Zeit zum Zeitpunkt des Schlages 0.0 betragen. Deshalb wird grundsätzlich der Wert $clock - timeHit$ übergeben, also die unnormalisierte, in Sekunden laufende, Uhr abzüglich des Schlagzeitpunktes.
- `old_clock`: Dieser Wert existiert erst ab v3 des Makros. Falls der Ball von einer Wand abprallt, ist `local_clock` der Zeitpunkt des Abpralls.`old_clock` kriegt dann den Wert des Abschlages (also $clock - timeHit$), damit darauf die aktuelle Geschwindigkeit in Richtung Boden berechnet werden kann.
- `old_v`: Dieser Wert existiert erst ab v3 des Makros. Bei einem Abprall von der Wand, verringert sich die Geschwindigkeit des Balles. Für die Berechnung der Geschwindigkeit Richtung Boden wird aber der alte Wert benötigt.

- `old_y`: Dieser Wert existiert erst ab v3 des Makros. Enthält die Höhe des Balles zum Zeitpunkt des Abschlages, weil darauf die aktuelle Höhe auf Basis der Wurfparabel berechnet wird.

Das Makro berechnet den momentanen x, y und z Wert des Balles und verschiebt den Ball anschließend an diesen Punkt.

`x_val` ist der momentane x Wert und hängt von der initialen Position (`p_start.x`), der Geschwindigkeit, des Sinus' des horizontalen Winkels sowie von der aktuellen Zeit ab (siehe Zeile 2).

Der y Wert `y_val` hängt von nur den initialen Werten beim Abschlag ab, egal wie oft der Ball eine Wand berührt außer der vertikale Winkel. Der ändert sich bei einem Zusammenstoß mit einer Wand nach dem Prinzip Einfallswinkel=Ausfallswinkel. Die Schwerkraft ist hier mit g benannt und wird woanders global im Code definiert. Mit g sowie der Geschwindigkeit, dem vertikalen Winkel, der ursprünglichen Höhe, soie der Zeit ab dem Schlag kann die momentane Höhe bestimmt werden.

Der z Wert `z_val` hängt von der Geschwindigkeit, dem vertikalen Winkel, dem Startwert sowie der Zeit ab. Wichtig ist hier die Richtung, der durch `ball_direction` angegeben wird. Dadurch wird der von der Zeit abhängige Wert entweder auf den Startwert addiert, oder vom ihm abgezogen.