

AR Billiard mit OpenCV und OpenGL

Sommersemester 2018

Friedemann Runte, Moritz Ludolph, Diyar Omar, Robin Mertens

Fakultät für Informatik

17. Juli 2018

- Hintergrundfarbe Dunkel-Grün mit der OpenGL ClearColor
- Löcher und Kugeln mit Triangle-Fan

- Struktur aus Dreiecken
- Mittelpunkt c
- Radius r
- Auflösung k ist die Anzahl der Dreiecke, aus denen unser Kreis hinterher besteht

Abbildung: Erstes Dreieck eines Triangle-Fans, $i = 1$, $k = 36$

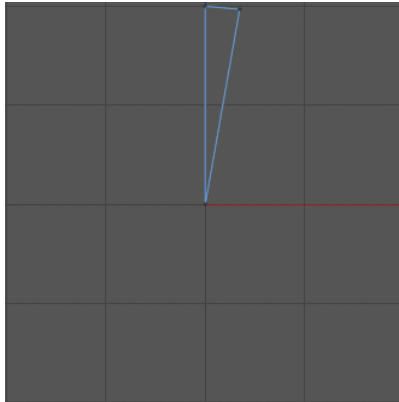


Abbildung: $i = 2$, $k = 36$

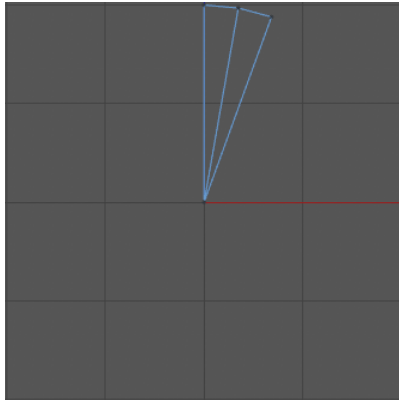


Abbildung: $i = 3$, $k = 36$

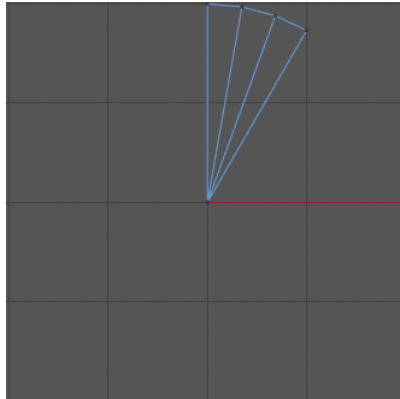
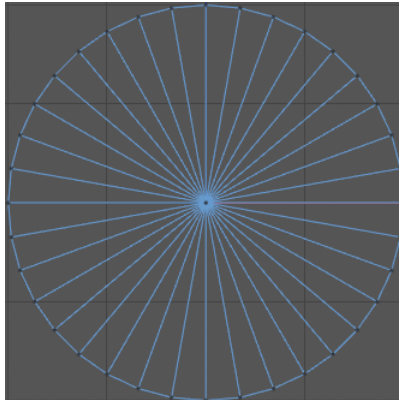
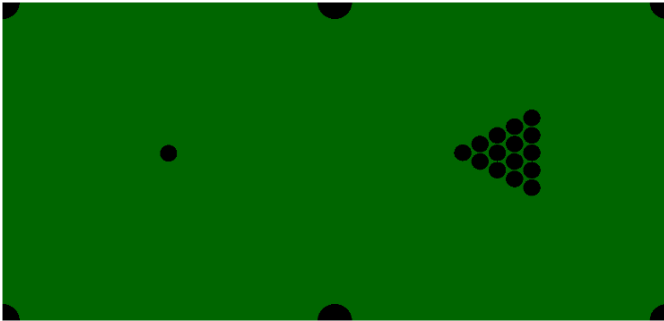


Abbildung: Vollständiger Triangle-Fan, $i = k = 36$



Wir haben nun ein Spielfeld mit Löchern und Kugeln. Jedoch sind die Kugeln noch nicht voneinander zu unterscheiden.

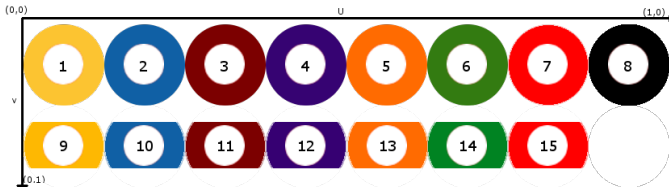
Abbildung: Spielfeld ohne Texturen



Grundkonzepte:

- Eine Textur besteht aus Koordinaten auf der u (waagrecht) und der v (vertikal) Achse
- Die beiden Achsen sind immer im Bereich $[0, 1]$, egal wie groß die Textur ist
- Man gibt beim Erstellen von Objekten für jeden Knoten die Texturkoordinaten in u und v an, um sie auf das Objekt abzubilden

Abbildung: Textur mit u- und v-Achse



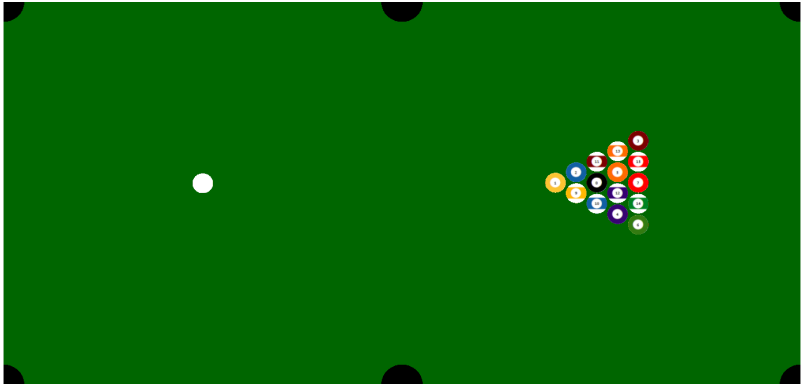
Es fällt auf, dass eine Kugel $\frac{1}{8}$ Durchmesser hat auf u, aber $\frac{1}{2}$ auf v

Wir müssen nun berechnen, welcher Ausschnitt für welche Kugel ist.

- Die Farben sind geordnet auf der Textur und bekommen Werte von 0 bis 7
- Die Vollen Kugeln sind in der ersten Reihe und die Halben in der Zweiten und bekommen damit die Werte 0 (voll) und 1 (halb)
- Wir können jetzt eine Funktion erstellen, die uns anhand von Farbe und Fülle den Textur Mittelpunkt ausgibt

Wir haben nun ein fertiges Spielfeld mit Texturierten Kugeln.

Abbildung: Fertiges Spielfeld



Die Physik-Berechnungen des Spiels lassen sich aufteilen in 3 Bereiche:

- Kollision von Kugel und Wand
- Kollision von Kugeln mit anderen Kugeln
- Kollision vom Queue mit der weißen Kugel

Prinzip sehr einfach:

1. Definiere die Wände als Achsenabschnitte: Linke Wand ist $x = 0$, rechte Wand $x = w$, obere Wand $y = 0$ und untere Wand $y = h$, mit $w = \text{Breite des Spielfeldes}$ und $h = \text{Höhe des Spielfeldes}$
2. Überprüfen ob eine der Koordinaten der Kugel zusammen mit dem Radius eine der Wände schneidet
z.B. für $r = 5$ wäre $x = 5 - r = 0$ und würde somit die Linke Wand schneiden
3. Geschwindigkeit auf der Achse die Geschnitten wurde invertieren, also bei $x = 0$ oder $x = w$ wird $v_x = -v_x$ gesetzt, analog für y

- Die Kollision von 2 Kreisen war uns bereits gegeben durch das Airhockey-Spiel.
- In Airhockey kollidiert ein Puck mit einem der beiden Schläger und bekommt dadurch eine neue Geschwindigkeit
- Der Schläger wird von der Kollision nicht verändert
- Wir brauchen aber, dass sich beide Kugeln bei Kollision verändern

Lösung:

- Wir berechnen für jede Kugel die Kollision mit jeder anderen Kugel
- Wir nehmen die Methode aus dem Airhockey und wählen unsere Kugel die sich bewegen soll als Puck und berechnen die Kollision mit allen anderen Kugeln als Schläger
- Wenn wir das in beide Richtungen ausführen, sodass jede Kugel sozusagen einmal Schläger und einmal Puck ist, wird jede Kugel von einer Kollision getroffen

Die eigentliche Kollision bleibt gleich wie beim Airhockey.

Problem: Framerate der Kamera

⇒ Unschärfe lässt Kollisionen verloren gehen

Wird Später im Kontext des Queues erläutert.

Problem: Damit der Spieler die Kugeln mit dem Queue spielen kann, muss dieser im Kamerabild erkannt werden

Unsere Lösung:

1. Queue vom Hintergrund des Bildes segmentieren
⇒ zur Vereinfachung ist der Queue schwarz gefärbt
2. Hauptachse des Queues bestimmen
3. Durch die Hauptachse die beiden Endpunkte des Queues bestimmen
4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit **beiden** Endpunkten die übliche Kollision ausführen

Problem: Damit der Spieler die Kugeln mit dem Queue spielen kann, muss dieser im Kamerabild erkannt werden

Unsere Lösung:

1. Queue vom Hintergrund des Bildes segmentieren
⇒ zur Vereinfachung ist der Queue schwarz gefärbt
2. Hauptachse des Queues bestimmen
3. Durch die Hauptachse die beiden Endpunkte des Queues bestimmen
4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit **beiden** Endpunkten die übliche Kollision ausführen

Problem: Damit der Spieler die Kugeln mit dem Queue spielen kann, muss dieser im Kamerabild erkannt werden

Unsere Lösung:

1. Queue vom Hintergrund des Bildes segmentieren
⇒ zur Vereinfachung ist der Queue schwarz gefärbt
2. Hauptachse des Queues bestimmen
3. Durch die Hauptachse die beiden Endpunkte des Queues bestimmen
4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit **beiden** Endpunkten die übliche Kollision ausführen

Problem: Damit der Spieler die Kugeln mit dem Queue spielen kann, muss dieser im Kamerabild erkannt werden

Unsere Lösung:

1. Queue vom Hintergrund des Bildes segmentieren
⇒ zur Vereinfachung ist der Queue schwarz gefärbt
2. Hauptachse des Queues bestimmen
3. Durch die Hauptachse die beiden Endpunkte des Queues bestimmen
4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit **beiden** Endpunkten die übliche Kollision ausführen

Problem: Damit der Spieler die Kugeln mit dem Queue spielen kann, muss dieser im Kamerabild erkannt werden

Unsere Lösung:

1. Queue vom Hintergrund des Bildes segmentieren
⇒ zur Vereinfachung ist der Queue schwarz gefärbt
2. Hauptachse des Queues bestimmen
3. Durch die Hauptachse die beiden Endpunkte des Queues bestimmen
4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit **beiden** Endpunkten die übliche Kollision ausführen

Problem: Damit der Spieler die Kugeln mit dem Queue spielen kann, muss dieser im Kamerabild erkannt werden

Unsere Lösung:

1. Queue vom Hintergrund des Bildes segmentieren
⇒ zur Vereinfachung ist der Queue schwarz gefärbt
2. Hauptachse des Queues bestimmen
3. Durch die Hauptachse die beiden Endpunkte des Queues bestimmen
4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit **beiden** Endpunkten die übliche Kollision ausführen

Problem: Damit der Spieler die Kugeln mit dem Queue spielen kann, muss dieser im Kamerabild erkannt werden

Unsere Lösung:

1. Queue vom Hintergrund des Bildes segmentieren
⇒ zur Vereinfachung ist der Queue schwarz gefärbt
2. Hauptachse des Queues bestimmen
3. Durch die Hauptachse die beiden Endpunkte des Queues bestimmen
4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit **beiden** Endpunkten die übliche Kollision ausführen

1. Queue vom Hintergrund des Bildes segmentieren:

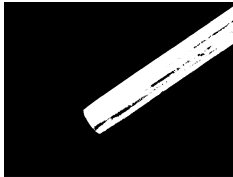


Kamerabild

1. Queue vom Hintergrund des Bildes segmentieren:



Kamerabild

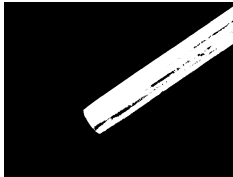


Segmentierung

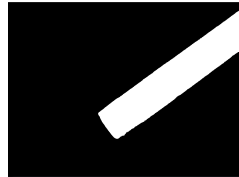
1. Queue vom Hintergrund des Bildes segmentieren:



Kamerabild



Segmentierung



Closing

2. Hauptachse des Queues bestimmen
⇒ durch Hauptkomponentenanalyse (PCA)
3. Durch die Hauptachse die beiden Endpunkte des Queues bestimmen

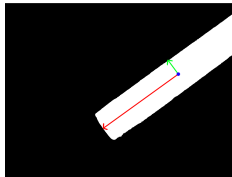


Kamerabild

2. Hauptachse des Queues bestimmen
⇒ durch Hauptkomponentenanalyse (PCA)
3. Durch die Hauptachse die beiden Endpunkte des Queues bestimmen



Kamerabild

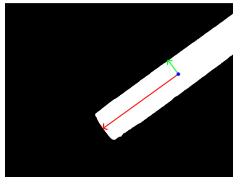


Ergebnis PCA

2. Hauptachse des Queues bestimmen
⇒ durch Hauptkomponentenanalyse (PCA)
3. Durch die Hauptachse die beiden Endpunkte des Queues bestimmen



Kamerabild



Ergebnis PCA



Bestimmte Endpunkte

4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit den Endpunkten übliche Kollision ausführen

Problem: Aufnahmen nur mit 24 Bildern pro Sekunde möglich
⇒ Kollisionen bei schneller Bewegung wird eventuell nicht erkannt!

Lösung: Berechnung des Kollisionspunktes durch Interpolation

4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit den Endpunkten übliche Kollision ausführen

Problem: Aufnahmen nur mit 24 Bildern pro Sekunde möglich
⇒ Kollisionen bei schneller Bewegung wird eventuell nicht erkannt!

Lösung: Berechnung des Kollisionspunktes durch Interpolation

4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit den Endpunkten übliche Kollision ausführen

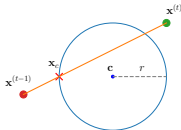
Problem: Aufnahmen nur mit 24 Bildern pro Sekunde möglich
⇒ Kollisionen bei schneller Bewegung wird eventuell nicht erkannt!

Lösung: Berechnung des Kollisionspunktes durch Interpolation

4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit den Endpunkten übliche Kollision ausführen

Problem: Aufnahmen nur mit 24 Bildern pro Sekunde möglich
⇒ Kollisionen bei schneller Bewegung wird eventuell nicht erkannt!

Lösung: Berechnung des Kollisionspunktes durch Interpolation

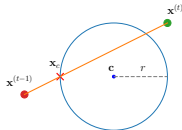


Kollision

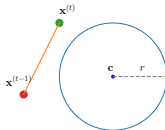
4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit den Endpunkten übliche Kollision ausführen

Problem: Aufnahmen nur mit 24 Bildern pro Sekunde möglich
⇒ Kollisionen bei schneller Bewegung wird eventuell nicht erkannt!

Lösung: Berechnung des Kollisionspunktes durch Interpolation



Kollision

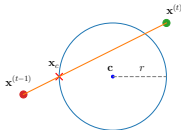


Keine Kollision

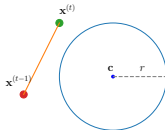
4. Die Endpunkte in Spielkoordinaten transformieren
5. Mit den Endpunkten übliche Kollision ausführen

Problem: Aufnahmen nur mit 24 Bildern pro Sekunde möglich
⇒ Kollisionen bei schneller Bewegung wird eventuell nicht erkannt!

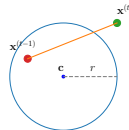
Lösung: Berechnung des Kollisionspunktes durch Interpolation



Kollision



Keine Kollision



Keine Kollision

Die vorgestellte Lösung würde sich noch verbessern lassen:

- Segmentierung (und damit alle folgenden Schritte) stark von der Umgebung abhängig
⇒ Verbesserung z.B. durch markerbasierte Erkennung
- Wiederholffrequenz/Belichtungszeit der Kamera verursacht Bewegungsunschärfe
⇒ Ungenauigkeit der Erkennung

Die vorgestellte Lösung würde sich noch verbessern lassen:

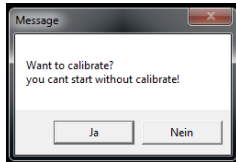
- Segmentierung (und damit alle folgenden Schritte) stark von der Umgebung abhängig
⇒ Verbesserung z.B. durch markerbasierte Erkennung
- Wiederholfrequenz/Belichtungszeit der Kamera verursacht Bewegungsunschärfe
⇒ Ungenauigkeit der Erkennung

Die vorgestellte Lösung würde sich noch verbessern lassen:

- Segmentierung (und damit alle folgenden Schritte) stark von der Umgebung abhängig
⇒ Verbesserung z.B. durch markerbasierte Erkennung
- Wiederholffrequenz/Belichtungszeit der Kamera verursacht Bewegungsunschärfe
⇒ Ungenauigkeit der Erkennung

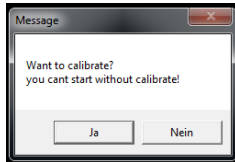
Bei der Benutzerinteraktion wurde der Großteil der Interaktionsmöglichkeiten durch Buttons und Pop-ups realisiert, um dem Nutzer eine möglichst simple Steuerung mit der Anwendung zu gewährleisten.

Bei der Benutzerinteraktion wurde der Großteil der Interaktionsmöglichkeiten durch Buttons und Pop-ups realisiert, um dem Nutzer eine möglichst simple Steuerung mit der Anwendung zu gewährleisten.

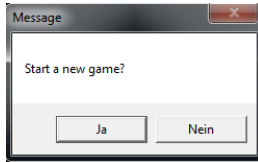


Kalibrierungs-Popup

Bei der Benutzerinteraktion wurde der Großteil der Interaktionsmöglichkeiten durch Buttons und Pop-ups realisiert, um dem Nutzer eine möglichst simple Steuerung mit der Anwendung zu gewährleisten.

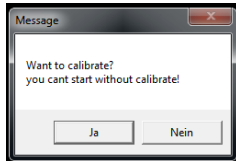


Kalibrierungs-Popup

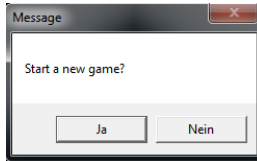


Starte Spiel-Popup

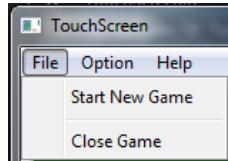
Bei der Benutzerinteraktion wurde der Großteil der Interaktionsmöglichkeiten durch Buttons und Pop-ups realisiert, um dem Nutzer eine möglichst simple Steuerung mit der Anwendung zu gewährleisten.



Kalibrierungs-Popup

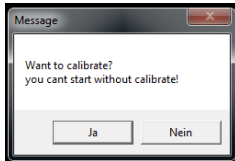


Starte Spiel-Popup

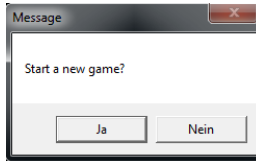


File-Menubar

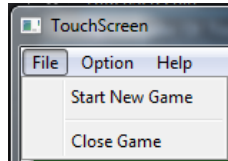
Bei der Benutzerinteraktion wurde der Großteil der Interaktionsmöglichkeiten durch Buttons und Pop-ups realisiert, um dem Nutzer eine möglichst simple Steuerung mit der Anwendung zu gewährleisten.



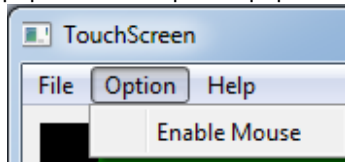
Kalibrierungs-Popup



Starte Spiel-Popup

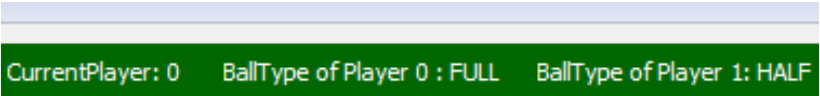


File-Menuubar



Option-Menuubar

Ein Teil dessen war allerdings auch das Anzeigen der Statistiken, worunter u.a. das Anzeigen des aktuellen Spielers oder der Kugeluweisung fällt. Hierbei wurde die Variante gewählt, die Statistiken als Label auf dem Spielfeld anzeigen zu lassen.



CurrentPlayer: 0 BallType of Player 0 : FULL BallType of Player 1: HALF

