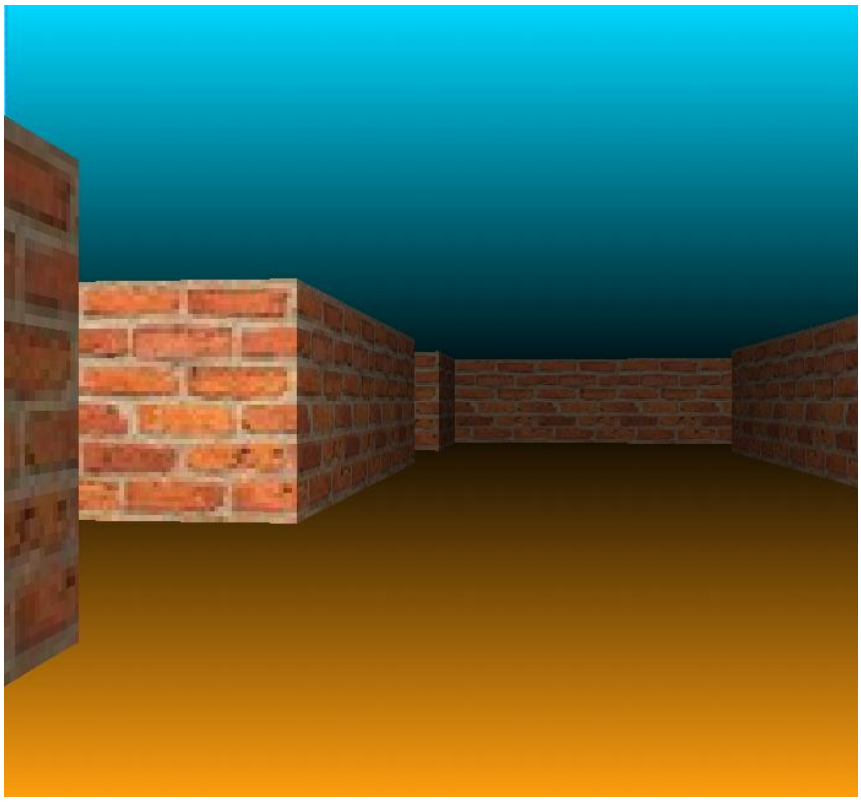
Raycasting zur Berechnung und Darstellung dreidimensionaler Umgebungen



Pascal Krauß & Yannick Weigert

Gymnasium Eversten Oldenburg

Betreuer: Herr Dr. Glade

Inhaltsverzeichnis

1 Kurzfassung

Raycasting ist ein Verfahren, bei dem der Computer eine dreidimensionale Umgebung darstellt, in der sich der Spieler bewegen kann. Diese Methode führte 1992 zu einem Durchbruch in der Videospielgeschichte. Beim Raycasting wird die dreidimensionale Umgebung auf Grundlage einer zweidimensionalen Karte berechnet. Mit Sichtstrahlen wird diese Karte "abgetastet" und die Entfernung zu Wänden berechnet. Dabei spielt die Trigonometrie eine wesentliche Rolle. Ein Raycaster führt Berechnungen für jede Pixel-Spalte auf dem Bildschirm durch.
Wir haben selber mehrere Raycaster programmiert, zu Beginn in der Programmiersprache "Scratch", später in der Sprache "Python". Zusätzlich haben wir ein Programm geschrieben, mit dem der Grundriss eines Hauses dreidimensional dargestellt werden kann.
Der große Vorteil des Raycasting-Verfahrens ist die Geschwindigkeit, mit der die dreidimensionalen Bilder errechnet werden. Allerdings besitzt ein Raycaster auch einige Einschränkungen. So haben beispielsweise alle Objekte im Raum die gleiche Höhe.

2 Einleitung

2.1 Ideenfindung

Schon seit einigen Jahren programmieren wir kleine Computerspiele jeglicher Art. Und so sehr sich die Spiele voneinander unterscheiden, haben sie doch alle eine Gemeinsamkeit: Sie sind zweidimensional.
Wir begannen uns die Frage zu stellen, wie ein Computerprogramm eigentlich dreidimensionale Umgebungen berechnet. Wie ist es möglich, dass auf einem Bildschirm täuschend echte 3D-Effekte erzielt werden?
Schon bald stießen wir auf das Raycasting-Verfahren, welches in einigen bekannten 3D – Computerspielen genutzt wird. Wir waren sofort von der Methode begeistert und beschlossen, unseren eigenen Raycaster zu programmieren.

2.2 Ziel

Unser Ziel ist es mit Hilfe des Raycasting-Verfahrens ein Computerprogramm zu schreiben, das eine dreidimensionale Umgebung berechnet und darstellt. Die Umgebung soll möglichst realistisch aussehen und der Spieler soll sich darin möglichst flüssig bewegen können. Schlussendlich soll unser Programm Grundrisse von Häusern dreidimensional darstellen, so dass sich der Spieler in den Häusern bewegen kann.

3 Methode und Vorgehensweise

3.1 Verwendete Programme

Für die unterschiedlichen Raycaster nutzen wir die Programmiersprachen „Scratch“ und „Python“. Für die grafische Darstellung von Formen und Farben in Python verwenden wir zudem die Bibliothek „PyGame“.

3.2 Funktionsweise eines Raycasters

3.2.1 Generelle Funktionsweise

Beim Raycasting werden die dreidimensionalen Bilder auf Grundlage einer zweidimensionalen Karte berechnet. Diese Karte gibt die Umgebung an, in der sich der Spieler bewegen kann. Die Karte wird in Form eines zweidimensionalen Arrays dargestellt. Dabei steht eine „0“ für eine freie Fläche und eine „1“ für eine Wand (Abbildung 1).

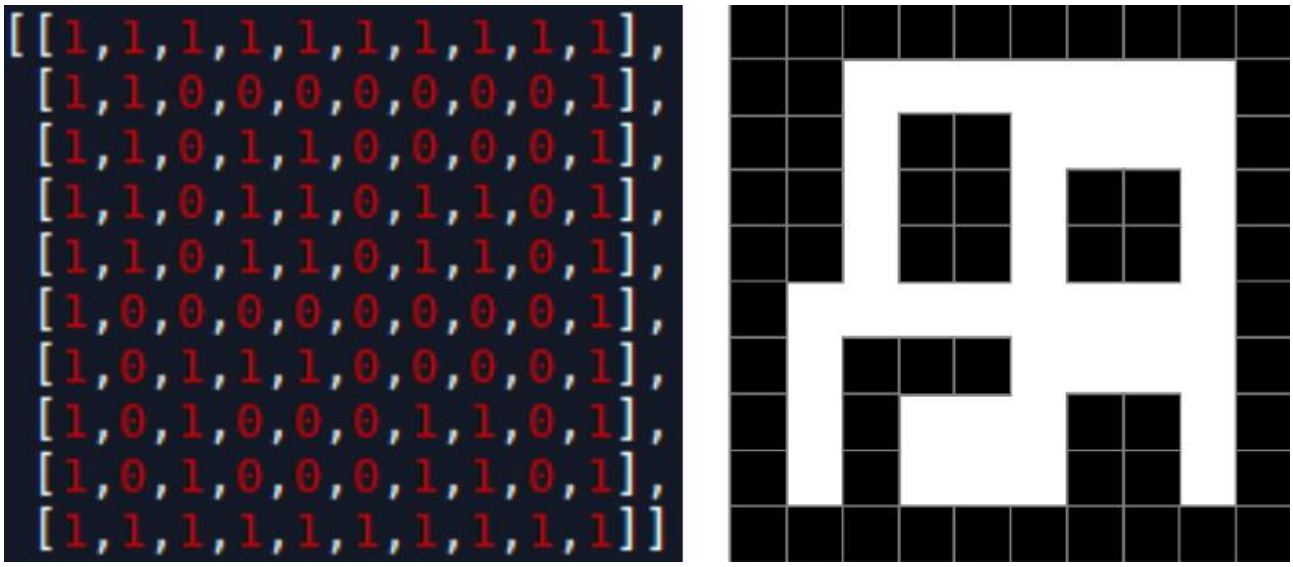


Abbildung 1: Eine beispielhafte Karte als Array und visuell dargestellt

Die Karte wird als Koordinatensystem betrachtet. Dabei haben die Quadrate, aus denen die Karte besteht die Länge 1.
Der Spieler befindet sich auf einer bestimmten Position in dieser Karte. Vom Spieler aus werden Sichtstrahlen ausgesendet, die auf die Wände in der Karte treffen. Die Sichtstrahlen sind dafür zuständig das Sichtfeld des Spielers „abzutasten“ (Abbildung 2).

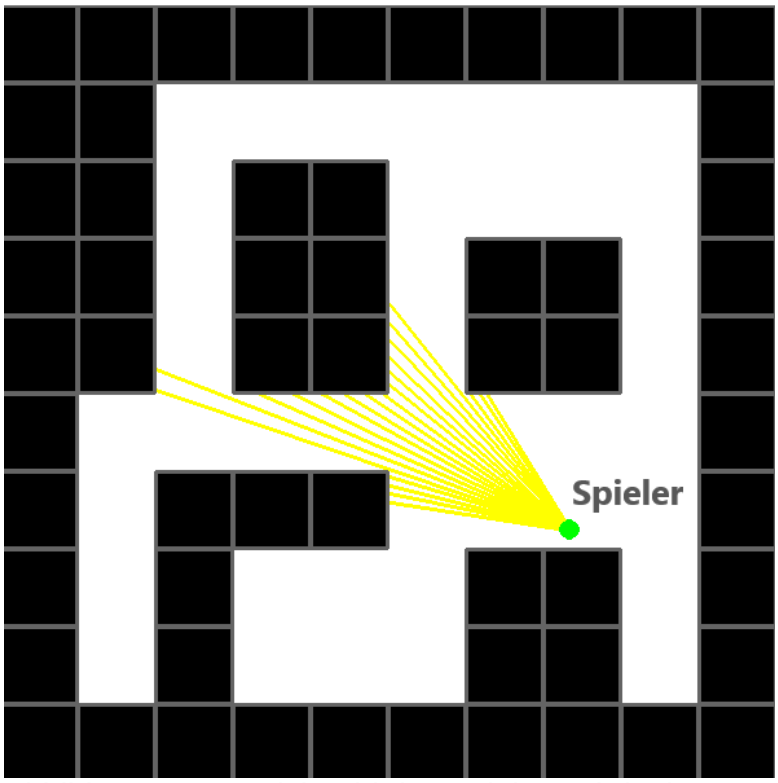


Abbildung 2: Das Sichtfeld des Spielers wird mit Sichtstrahlen abgetastet

Für jeden Sichtstrahl wird ein vertikaler Strich auf den Bildschirm gezeichnet, dessen Größe abhängig von der Distanz des Spielers zur Wand ist. Dadurch entsteht das dreidimensionale Bild (Abbildung 3). Ein Raycaster führt also Berechnungen für jede Pixel-Spalte auf dem Bildschirm aus.

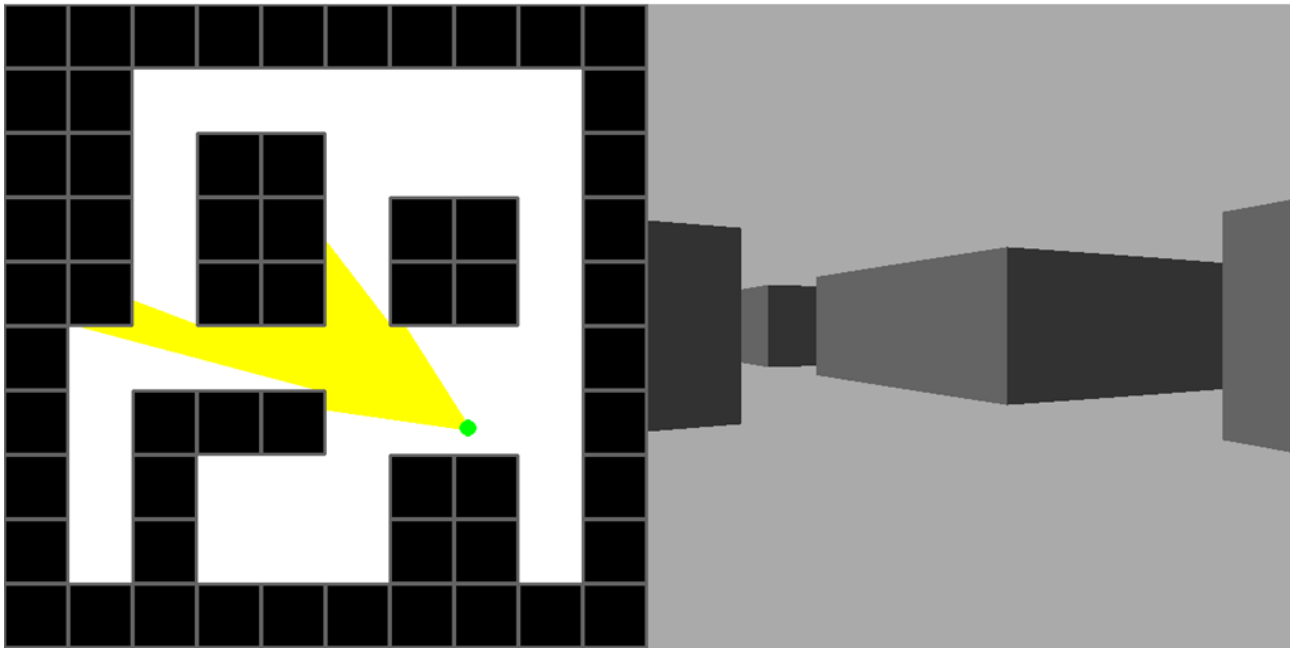


Abbildung 3: Funktionsweise eines Raycasters visuell dargestellt. Links wird die Karte abgetastet. Rechts wird das dreidimensionale Bild erstellt.

3.2.2 Berechnung des Punktes, an dem der Sichtstrahl eine Wand trifft

Für jeden Sichtstrahl muss der Punkt gefunden werden, an dem der Sichtstrahl auf eine Wand trifft. Nur mit diesem Punkt kann die Entfernung des Spielers zur Wand berechnet werden. Um den Punkt zu finden, wird jedes Mal, wenn der Sichtstrahl in ein neues Quadrat gelangt, überprüft, ob sich in dieser Zone eine Wand befindet (Abbildung 4).

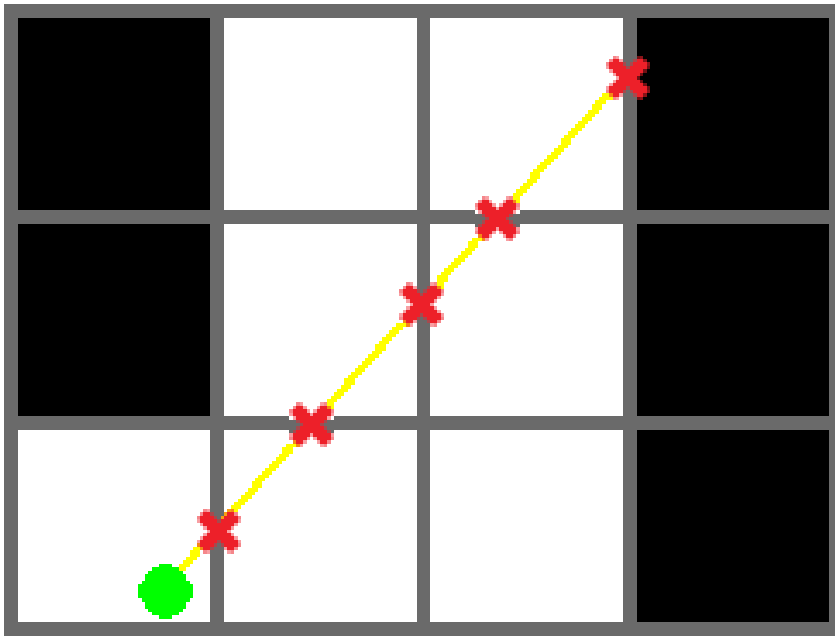


Abbildung 4: An den markierten Stellen trifft der Sichtstrahl auf ein neues Quadrat. An diesen Punkten muss überprüft werden, ob sich im Quadrat eine Wand befindet.

In diesem Abschnitt geht es darum, alle Punkte zu berechnen, an denen der Sichtstrahl in ein neues Quadrat gelangt, um schlussendlich den Punkt ausfindig zu machen, an dem der Sichtstrahl die Wand trifft. Es reicht schon aus, wenn man lediglich die x-Positionen der Punkte berechnet.

An einigen Stellen schneidet der Sichtstrahl eine vertikale Linie. An anderen Stellen schneidet er eine horizontale Linie. Zunächst berechnen wir die x-Positionen des ersten horizontalen und des ersten vertikalen Schnittpunkts (Abbildung 5).

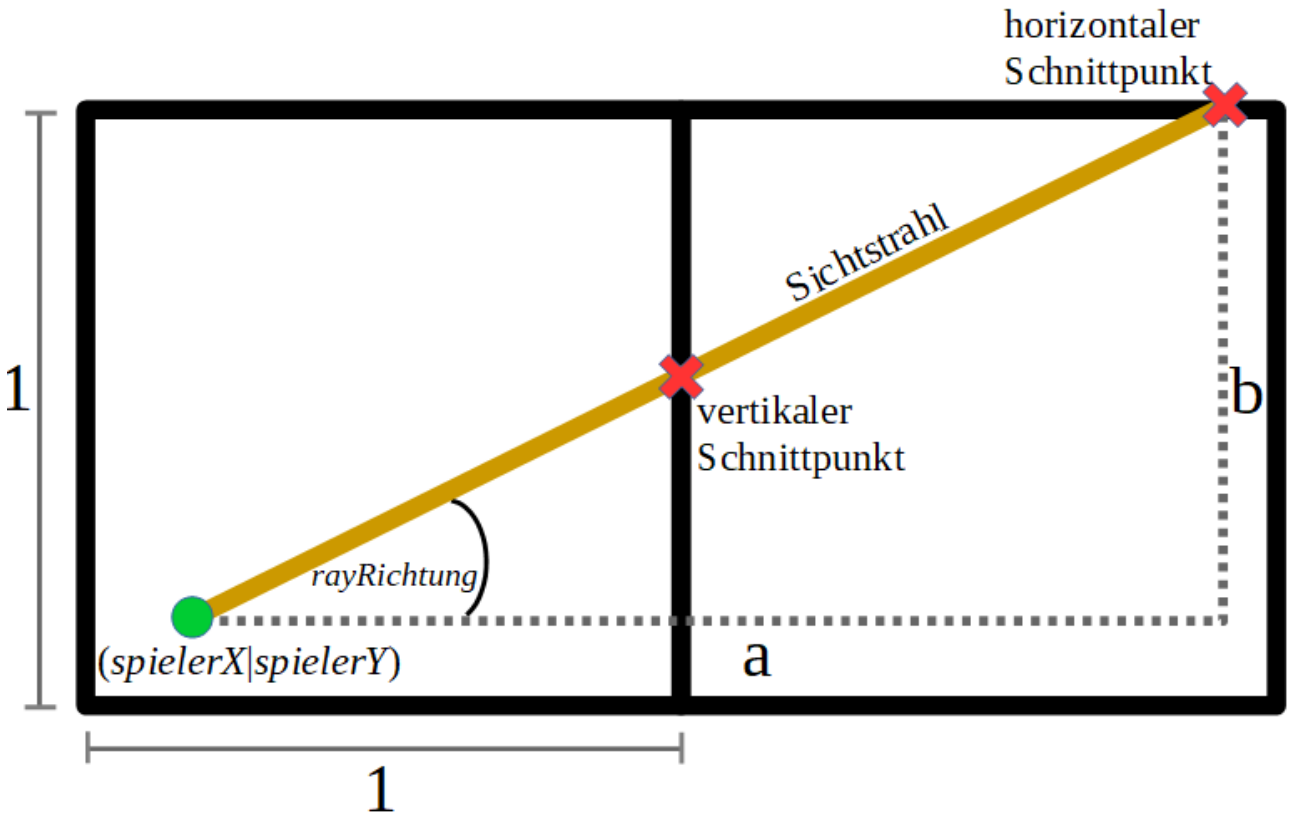


Abbildung 5: Die x-Positionen des ersten vertikalen und horizontalen Schnittpunkts sollen berechnet werden

Variablen:

spielerX:	gibt die x-Position des Spielers an.
spielerY:	gibt die y-Position des Spielers an.
spielerRichtung:	gibt die Richtung, in die der Spieler blickt in Grad an.
rayRichtung:	gibt die Richtung, in die ein Sichtstrahl verläuft in Grad an.

Die gesuchten x-Positionen lassen sich wie folgt berechnen:

vertikaler Schnittpunkt x = ceil(*spielerX*) aufrunden von spielerX

horizontaler Schnittpunkt x = *spielerX* + a

a = b / tan(*rayRichtung*)

b = 1 - (*spielerY* mod 1) 1 - (Nachkommastellen von *spielerY*)

horizontaler Schnittpunkt x = spielerX + (1 - (*spielerY* mod 1)) / tan(*rayRichtung*)

Nun müssen noch die x-Positionen der weiteren Schnittpunkte berechnet werden.
Die weiteren vertikalen Schnittpunkte sind immer 1 voneinander entfernt. Auch der Abstand der horizontalen Schnittpunkte ist immer gleich (Abbildung 6).

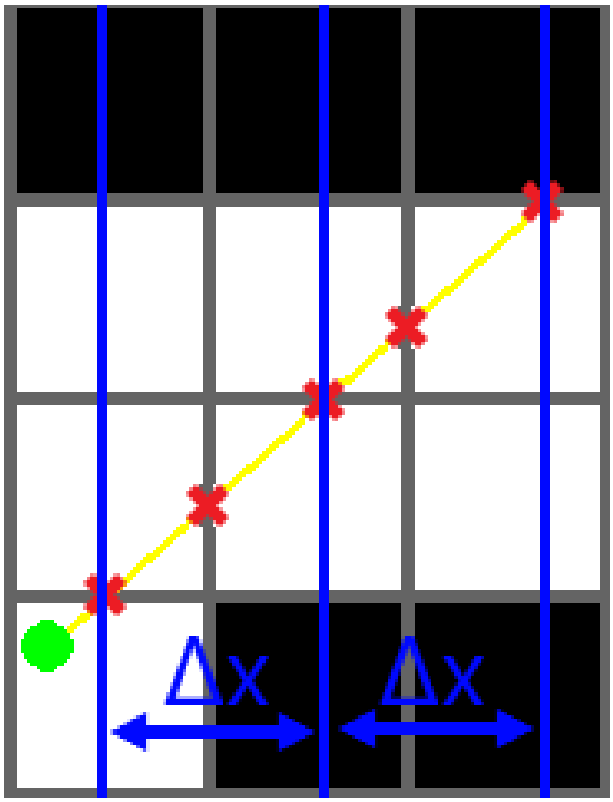


Abbildung 6: Der Abstand zwischen den horizontalen Schnittpunkten ist immer gleich

Δ x gibt den Abstand der horizontalen Schnittpunkte an. Mithilfe der Trigonometrie lässt sich *Δ x berechnen:*

***Δ x* = 1 / tan(*rayRichtung*)**

Mithilfe dieser Formeln lässt sich die x-Position jedes Schnittpunkts eines Sichtstrahls berechnen. Die Schnittpunkte mit der kleineren x-Position werden vom Sichtstrahl zuerst durchlaufen.

Diese Formeln gelten nur für einen Sichtstrahl, der in den Nord-Osten verläuft. Für andere Himmelsrichtung müssen die Formeln zum Teil leicht angepasst werden.

Die Schnittpunkte werden in ihrer korrekten Reihenfolge berechnet und es wird jeweils geprüft, ob der Sichtstrahl an den Punkten in eine „Wandzone“ gelangt. Ist dies der Fall hat man den

gewünschten Punkt gefunden. Die x-Position dieses Punktes wird dann in der Variable *schnittX gespeichert.*

3.2.3 Entfernung des Spielers zur Wand berechnen

schnittX : x-Position des Punktes, an dem der Sichtstrahl die Wand trifft.

Mithilfe des ermittelten Werts *schnittX* können wir nun den Abstand des Spielers zur Wand berechnen. Zuerst muss die Länge des Sichtstrahls berechnet werden (Abbildung 7)

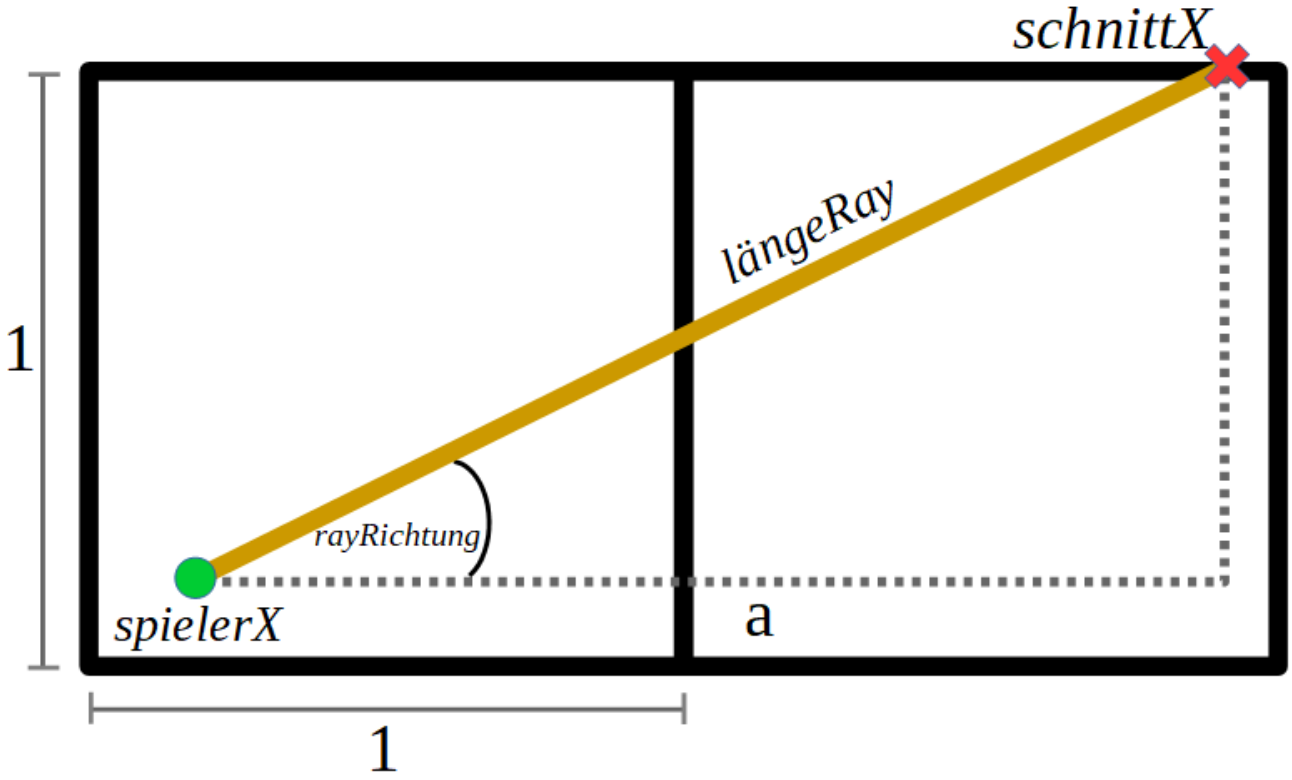


Abbildung 7: längeRay gibt die Länge des Sichtstrahls an und soll berechnet werden

Die Länge des Sichtstrahls lässt sich wie folgt berechnen:

längeRay = *a* / cos(*rayRichtung*)

a = *schnittX – spielerX*

längeRay = (schnittX – spielerX) / cos(rayRichtung)

Die Länge des Sichtstrahls ist nicht, wie man annehmen könnte, die Entfernung des Spielers zur Wand. Würde man die Länge des Sichtstrahls als Entfernung verwenden, würde man eine verzerrte Darstellung der Umgebung erhalten.
Die tatsächliche Entfernung, die wir benötigen wird in Abbildung 9 ersichtlich.

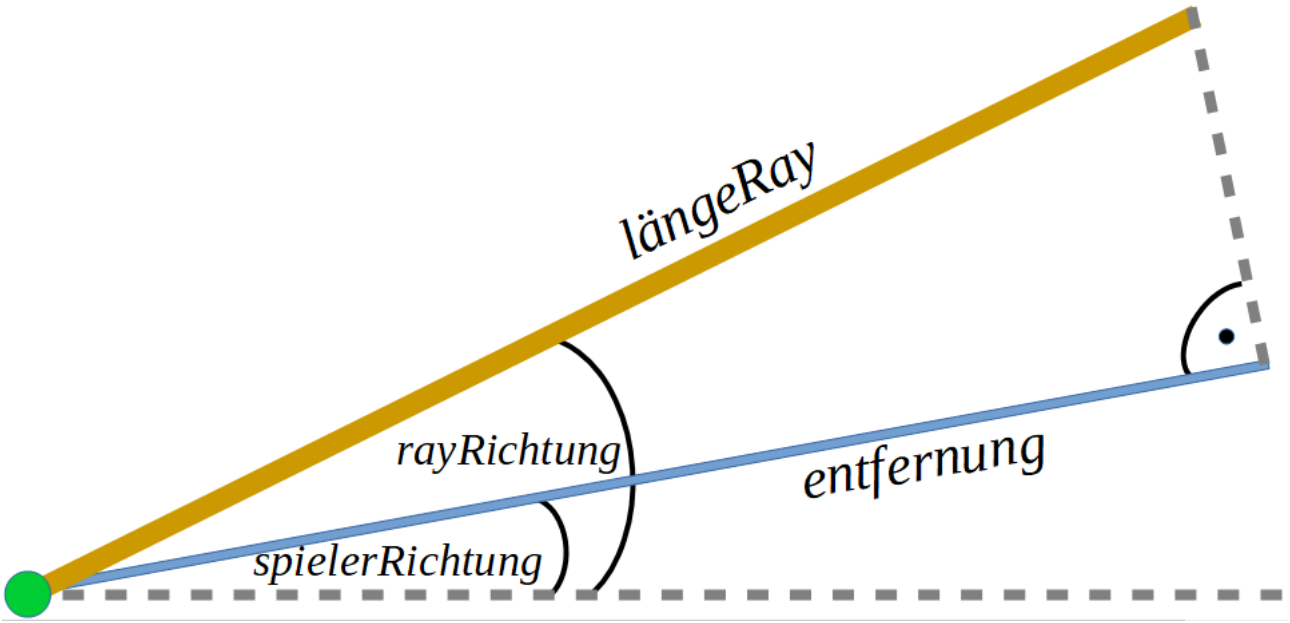


Abbildung 9: Die blaue Linie ist die gesuchte Entfernung. Sie verläuft in die selbe Richtung, in die der Spieler blickt.

Die Entfernung des Spielers zur Wand wird wie folgt berechnet:
***entfernung* = *längeRay* * cos(*rayRichtung* – *spielerRichtung*)**

Mit diesen Formeln kann für jeden Sichtstrahl die Entfernung zur Wand berechnet werden. Somit kann das Sichtfeld des Spielers komplett abgetastet werden.

3.2.4 Darstellung des dreidimensionalen Bildes

Wie bereits erwähnt besteht das dreidimensionale Bild beim Raycasting aus vielen vertikalen Linien, die die Wände darstellen (Abbildung 10). Für jeden Sichtstrahl wird eine vertikale Linie gezeichnet. Je größer die Entfernung zur Wand, desto kleiner muss diese Linie gezeichnet werden. Die Größe der Linie berechnet sich wie folgt:

größeLinie = Konstante / *entfernung*

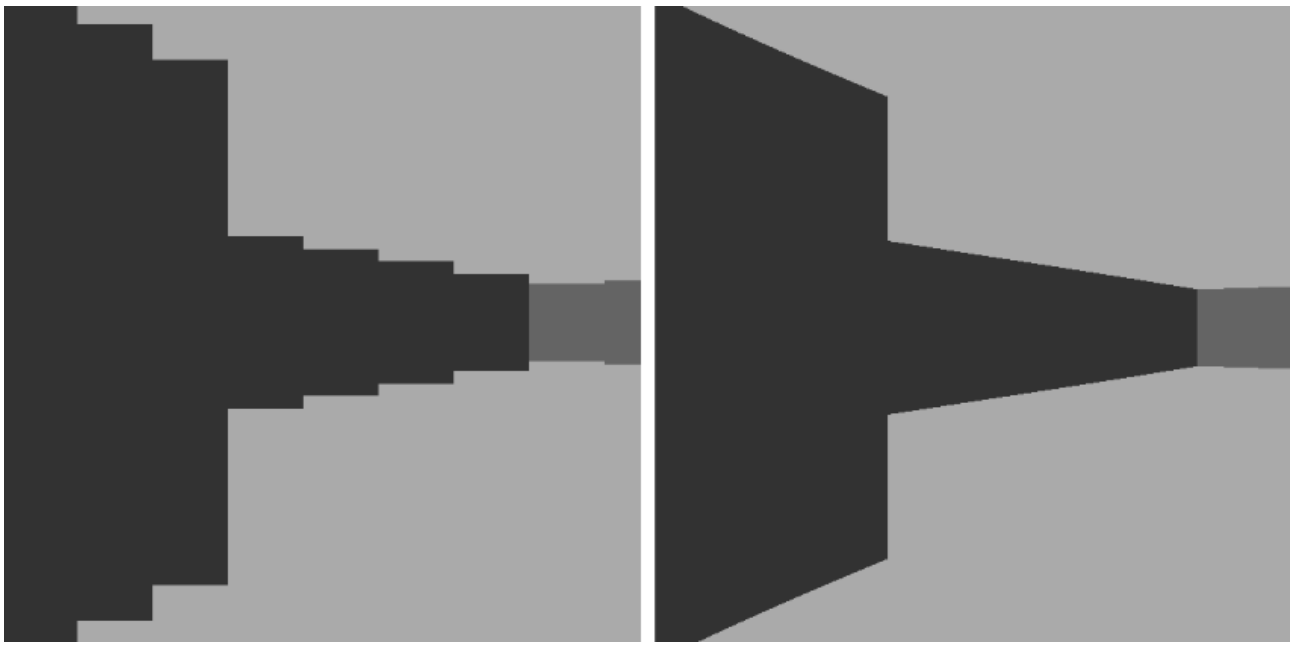


Abbildung 10: Ein Raycaster besteht aus vertikalen Linien. Auf den beiden Bildern ist die gleiche Umgebung abgebildet. Das linke Bild besteht aus wenigen breiten Linien, das rechte Bild besteht aus vielen schmalen Linien

Vertikale Wände werden um einen festgelegten Faktor verdunkelt, um einen „Schatten-Effekt“ zu erzeugen. Zusätzlich kann man Wände, die sich weiter hinten im Raum befinden dunkler darstellen. Als Hintergrund wird außerdem oft ein Boden und ein Himmel abgebildet.

Mit einigen zusätzlichen Berechnungen ist es möglich, Texturen an die Wände zu bringen. Dazu muss berechnet werden, an welcher genauen Stelle der Sichtstrahl die Wand trifft. Dadurch kann die passende „Textur-Pixel-Spalte“ aus einem Array gewählt werden. Die vertikalen Linien bestehen dann jeweils aus unterschiedlichen Farben und ergeben zusammen die entsprechende Textur.

3.3 Die Berechnungen in Python-Code

In Abbildung 11 wurden die theoretischen Überlegungen aus den vorherigen Abschnitten in Python-Code umgesetzt. Der Code ist ein Ausschnitt aus einem unserer Raycaster. Mit Hilfe der eingefügten Kommentare kann man seine Funktionsweise nachvollziehen.

```
#Quadrat, in dem sich der Sichtstrahl befindet
quadratX = int(playerX)
quadratY = t - int(playerY)

# x-Positionen des ersten vertikalen und horizontalen Schnittpunkts
horizontalX = playerX + (1 - playerY % 1) / math.tan(rayRichtung)
vertikalX = math.ceil(playerX)

# Abstand der Schnittpunkte
deltaX = 1/math.tan(rayRichtung)

# Wiederhole bis Sichtstrahl auf Wand trifft
wand = False
while not wand:

    # nächster Schnittpunkt horizontal oder vertikal?
    if horizontalX < vertikalX:
        # aktueller Schnittpunkt
        schnittX = horizontalX
        # nächster horizontaler Schnittpunkt
        horizontalX += deltaX
        # Sichtstrahl gelangt in neues Quadrat
        quadratY -= 1
    else:
        # aktueller Schnittpunkt
        schnittX = vertikalX
        # nächster vertikaler Schnittpunkt
        vertikalX += 1
        # Sichtstrahl gelangt in neues Quadrat
        quadratX += 1

    # Ray trifft Wand?
    if karte[quadratY][quadratX]:
        wand = True

# Länge des Sichtstrahls berechnen
rayLaenge = (schnittX - playerX) / math.cos(rayRichtung)

# Entfernung des Spielers zur Wand
entfernung = rayLaenge * math.cos(rayRichtung - spielerRichtung)

#Größe der vertikalen Linie auf dem Bildschirm
groesseLinie = 800 / entfernung
```

Abbildung 11: Programmteil eines Raycasters, der alle notwendigen Berechnungen für einen Sichtstrahl ausführt. Dieses Programm ist von uns geschrieben. Der Programmteil ist vereinfacht dargestellt und funktioniert somit nur für einen Sichtstrahl, der in den Nord-Osten verläuft.

4 Ergebnisse

Wir haben unterschiedliche Raycaster programmiert. Unser aktuellster Raycaster besitzt einen sehr realistischen 3d-Effekt und läuft flüssig. An den Wänden befinden sich Texturen. Das Programm ist in Python programmiert und besteht aus ca. 200 Zeilen Code (Abbildung 12).

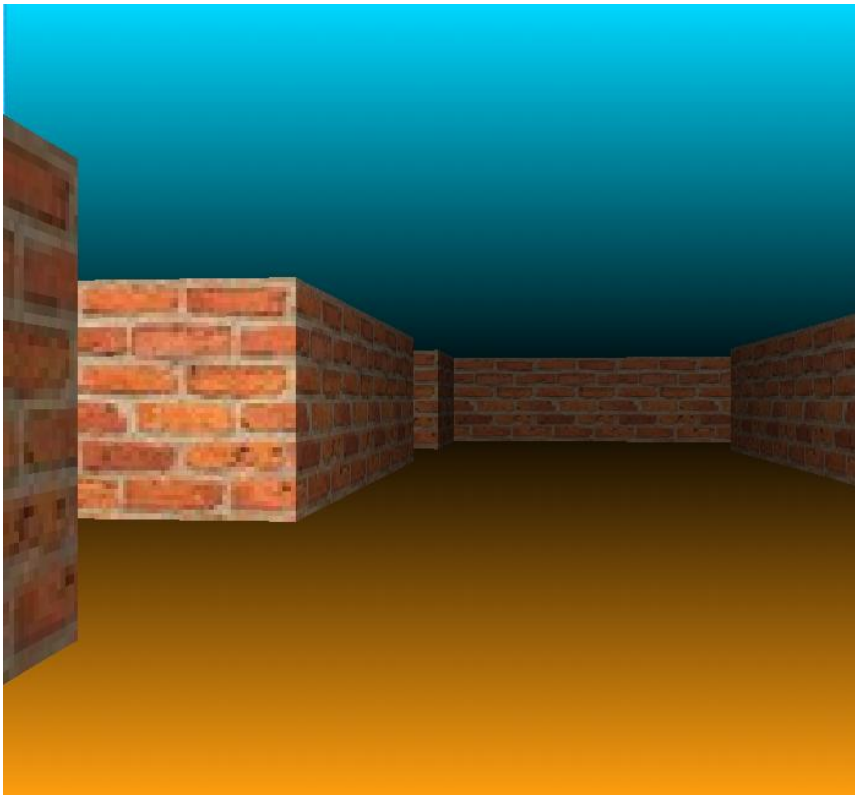


Abbildung 12: Unser aktuellster Raycaster

Zusätzlich haben wir ein Programm geschrieben, das ein Bild von einem Grundriss in ein zweidimensionales Array umwandelt. Dadurch kann sich der Spieler virtuell in dem Haus bewegen und bekommt eine Vorstellung davon, wie es in der Realität aussieht.

5 Diskussion

Wir haben unser Programm so weit optimiert, dass es schwierig ist, es noch weiter zu verbessern. Allerdings weist die Methode des Raycastings selbst einige Probleme auf. Wände können nur im 90 Grad Winkel aneinander angrenzen. Auch ist es nicht möglich runde Formen darzustellen. Außerdem besitzen alle Wände die gleiche Höhe.

Die Vorteile des Raycastings bestehen vor allem in der Geschwindigkeit. Dadurch, dass das Programm nur jede Pixel-Spalte berechnet, verläuft das Verfahren deutlich schneller als Raytracing, welches die Helligkeit für jeden Pixel berechnet. Raycasting wurde oft in frühen Computerspielen eingesetzt, da das Verfahren selbst auf leistungsschwachen Computern funktioniert.

Unser Programm ist in Python geschrieben. Allerdings ist Python eine relativ langsame Programmiersprache und für einen Raycaster eher schlecht geeignet. In einer anderen Sprache wie z.B. „C“ hätten wir vermutlich noch bessere Ergebnisse erzielen können.

6 Literatur

https://www.youtube.com/watch?v=eOCQfxRQ2pY, (09. 2019)
https://en.wikipedia.org/wiki/Ray_casting, (09. 2019)

Unsere Arbeit basiert auf wenigen Quellen. Nachdem wir uns mit den angegebenen Quellen die Funktionsweise eines Raycasters angeeignet hatten, programmierten wir nur noch auf Grundlage eigener Überlegungen.

Alle Bilder, die in diesem Bericht verwendet werden, sind von uns selber erstellt worden.