Prof. Dr. Sebastian Hack
Julian Rosemann, B. Sc.
Tina Jung, B. Sc.
Maximilian Fickert, M. Sc.

RetroRenderer (15 Punkte)

In diesem Projekt implementieren Sie ein `C` Programm zum Darstellen von dreidimensionalen Umgebungen.

Der Rendering-Algorithmus

Beim Darstellen (Rendering) von dreidimensionalen Umgebungen geht es darum, eine dreidimensionale Szenerie auf einem zweidimensionalen Medium (dem Computerbildschirm) so abzubilden, wie Beobachtende sie wahrnehmen würden. Dies hat viele Anwendungen, zum Beispiel im Computer-Aided-Design (CAD) oder für Unterhaltungssoftware (Videospiele, Animationsfilme). Für diese Zwecke wurden viele Rendering-Algorithmen entwickelt, die sich in ihrer Effizienz und ihrer Darstellungsqualität unterscheiden. Abbildung 1 zeigt eine Szenerie, wie sie mit dem hier verwendeten Algorithmus dargestellt wird.

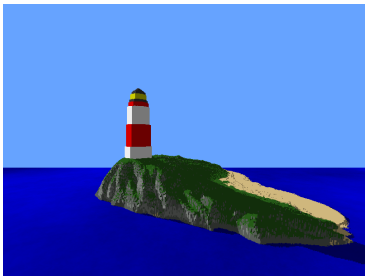
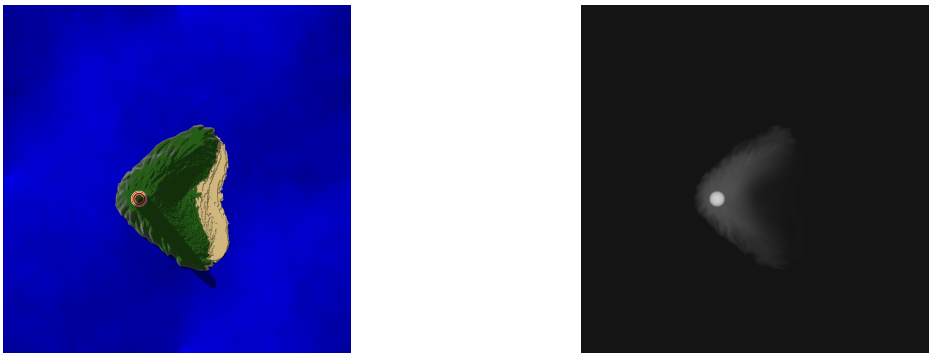


Abbildung 1: Gerenderte Szene

Eingabe

Die Eingabe des Algorithmus ist eine Beschreibung der dreidimensionalen Umgebung in Form einer Höhenkarte, die jedem Punkt auf einer Grundebene eine Höhe zuweist, und einer Farbkarte, die jedem Punkt auf der Grundebene eine Farbe zuweist. Beispiele hierfür sind in Abbildung 2 zu sehen. Zusätzlich wird eine Farbe zur Darstellung des Himmels gewählt.

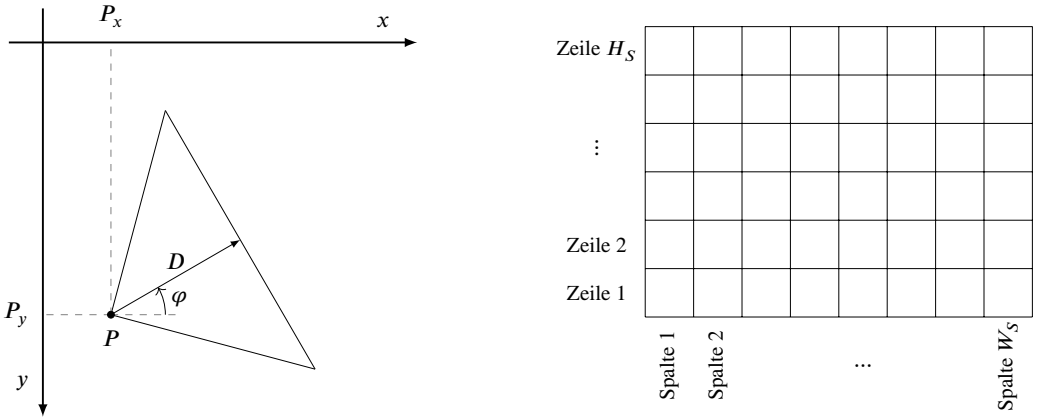


(a) (b)

Abbildung 2: Darstellung von Farbkarte (links) und Höhenkarte (rechts, je heller desto größer die Höhe) einer Szenerie

Wir definieren uns ein Koordinatensystem für den so dargestellten dreidimensionalen Raum: Der Koordinatenursprung liegt an der Position oben links in Höhen- und Farbkarte. Von dort aus verläuft die y-Achse nach unten und die x-Achse nach rechts. Die z-Achse steht senkrecht zu x- und y-Achse, wobei größere z-Werte einer größeren Höhe entsprechen.
Wir interpretieren die Karte so, dass am oberen Ende der Karte das untere Ende anschließt und am linken Ende der Karte das rechte Ende anschließt.[1] Somit können Beobachtende, die sich durch die Szenerie bewegen, beliebig weit in jede Richtung gehen.
Damit der Renderer nicht unendlich viele Punkte auf der Karte zur Darstellung erwägen muss, erhält er als Parameter eine maximale Sichtweite D.

[1] Die Karte wird also als Torus interpretiert.



(a) Position und Sichtfeld des Beobachters P

(b) Aufteilung eines Zielbildschirms in Pixel

Abbildung 3: Benennungskonventionen in der Szenerie und im Bildschirm

Abbildung 3a zeigt das so beschränkte Sichtfeld eines Beobachters P in der Draufsicht. Die Position von P wird durch seine Koordinaten P_x, P_y und der hier nicht dargestellten Höhe P_z sowie der Ausrichtung, gemessen als Winkel φ bezüglich der x-Achse, bestimmt. Wie in der Abbildung dargestellt, nehmen wir für dieses Projekt an, dass das Sichtfeld in der Horizontalen auf einen Winkel von 90° beschränkt ist.

Rendering

Das Ziel des Rendering-Algorithmus ist die Darstellung einer Szenerie auf einem Zielbildschirm (in diesem Projekt das Programmfenster). Der Zielbildschirm ist rechteckig und aus quadratischen Pixeln zusammengesetzt, die wir wie in Abbildung 3b abgebildet in Zeilen und Spalten organisieren.
Der Rendering-Algorithmus berechnet für jeden dieser Pixel denjenigen Punkt in der Umgebung, der von einem Standpunkt aus zu sehen ist, und stellt ihn dort in seiner Farbe dar. Er geht dazu wie folgt vor:

initialisiere den Bildschirm mit der Himmelfarbe
für jede Entfernung $d < D$, absteigend:
  berechne die Endpunkte L und R der Senkrechten zur Beobachtungsrichtung mit Abstand d
  wähle für jede Bildschirmspalte u einen Punkt Q_u auf der Strecke zwischen L und R
  für jede Bildschirmspalte u:
    berechne die dargestellte Höhe v von Q_u auf dem Bildschirm
    zeichne eine vertikale Linie in der Farbe von Q_u vom Boden des Bildschirms zu v in Spalte u

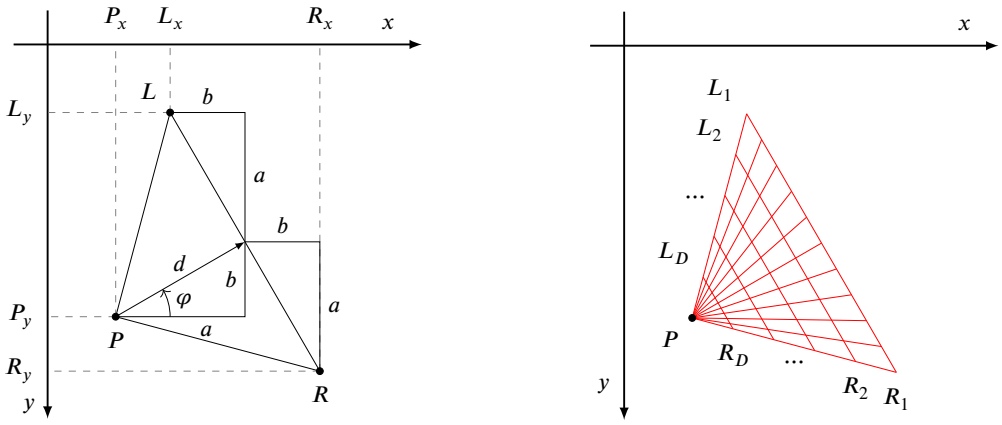
Die einzelnen Teilschritte sind im Folgenden genauer beschrieben.

Berechnen der Endpunkte Um die Senkrechte zur Beobachtungsrichtung mit Abstand d zu rastern, müssen ihre Endpunkte bestimmt werden. Eine Herleitung der Koordinaten dieser Punkte ist in Abbildung 4a dargestellt. Die Seitenlängen a und b der rechtwinkligen Dreiecke in dieser Abbildung können durch Sinus und Kosinus der Ausrichtung φ berechnet werden:

$$a = \cos(\varphi) \cdot d$$
$$b = \sin(\varphi) \cdot d$$

Damit berechnen sich die Koordinaten der Endpunkte L und R wie folgt:

$$L_x = P_x + a - b \qquad R_x = P_x + a + b$$
$$L_y = P_y - b - a \qquad R_y = P_y - b + a$$



(a) Berechnung der Endpunkte L und R

(b) Schnittpunkte der Linien werden beim Rendern abgetastet

Abbildung 4: Darstellungen zum Berechnen der Abtastungspunkte beim Rendern

Rasterung der Senkrechten Der Algorithmus ermittelt für jeden Abstand d innerhalb der Sichtweite für jede Bildschirmspalte, wie hoch die Landschaft in diesem Abstand hier dargestellt werden muss. Dazu müssen entsprechende Punkte auf der Senkrechten zur Beobachtungsrichtung mit Abstand d bestimmt werden, um deren Höhe in der Höhenkarte nachzuschlagen. Stellen Sie sicher, dass diese Punkte auf der jeweiligen Senkrechten gleich verteilt sind und dass für die erste und letzte Bildschirmspalte jeweils der Punkt L und R wie oben bestimmt verwendet werden. Abbildung 4b zeigt schematisch, welche Punkte abgetastet werden sollten.
Hinweis: Sie müssen diese Aufteilung nicht explizit in Datenstrukturen erstellen sondern sollten nach jedem Abtasten den aktuell betrachteten Punkt weiterbewegen.

Berechnen der dargestellten Höhe Die dargestellte Höhe eines Punktes Q auf einem Zielbildschirm kann wie folgt berechnet werden:

$$v_S = \frac{W_S}{2} \cdot \frac{Q_z - P_z}{d} + \frac{H_S}{2}$$

Hierbei sind H_S und W_S die Höhe und Breite des Zielbildschirms, Q_z ist die Höhe des Punktes laut der Höhenkarte, P_z die Höhe des Beobachters und d der Abstand der gerade abgetasteten Senkrechten vom Beobachter. Eine Herleitung für diese Formel können Sie in Anhang A finden.

Implementierung

Ihre Aufgabe in diesem Projekt ist die Implementierung von drei Funktionen in der Datei `src/render.c`. Diese werden im Folgenden beschrieben. Bei allen Aufgaben beachten Sie zusätzliche die Dokumentation, welche im gegebenen Quelltext zu finden ist.

Aufgabe 1: update_player()

Das Render-Programm, das Sie in diesem Projekt schreiben, erlaubt es, die Position des Beobachters zu verändern. Dazu hat das `player` Struct (die Deklaration finden Sie in `include/render.h`) zusätzlich zu Feldern für die aktuelle Position und Ausrichtung auch Felder für Geschwindigkeiten. Diese Geschwindigkeitsfelder werden von dem mitgelieferten Programmrahmen entsprechend der Tastatureingaben gesetzt.
Die `update_player` Funktion wird einmal pro Zeitschritt aufgerufen. Sie soll die folgenden Schritte in dieser Reihenfolge erledigen:

1. die Winkelgeschwindigkeit `v_angular` auf den Winkel addieren,
2. die Position in x und y Richtung um `v` in Beobachtungsrichtung verschieben und
3. die Höhe um `v_height` vergrößern.

Sie müssen dafür sorgen, dass Position und Winkel in ihrem jeweiligen Wertebereich bleiben. Für den Winkel ist dies das Intervall $[0, 360)$, für die x und y Koordinaten ist es jeweils $[0, sz)$ wobei sz die Seitenlänge der quadratischen Szenerie ist. Normalisieren Sie die Werte nach dem Bewegen in diesen Bereich falls nötig. Für die Koordinaten können Sie dazu die mitgelieferte `float_mod()` Funktion verwenden.
Die Höhe des Beobachters darf niemals 20 Punkte über dem Wert der Höhenkarte an der aktuellen Position unterschreiten. Sorgen Sie also dafür, dass bei Unterschreitung dieses Abstands der Beobachter auf die minimale erlaubte Höhe an der aktuellen Position der Karte gesetzt wird.

Aufgabe 2: draw_line()

Diese Funktion soll eine vertikale Linie auf den Bildschirm zeichnen. Der der Funktion übergebene Kontext (die Deklaration finden Sie in `include/render.h`) enthält zu diesem Zweck in dem `out` Feld einen Zeiger auf einen Speicherbereich, der von der bereitgestellten Implementierung über das `sdl2` Framework auf dem Bildschirm dargestellt wird. Beachten Sie die dokumentierenden Kommentare an der Funktion für eine genaue Beschreibung der Argumente.
Sie können mit Hilfe des Testrahmenwerks ein Testbild generieren um Ihre Implementierung zu überprüfen:

```
bin/testrunner testimage
```

Dieser Aufruf generiert ein Bild in der Datei `test_image.ppm`. Wenn Ihre Implmentierung korrekt ist, stimmt dieses vollständig mit dem Bild in `test/ref_images/test_image_ref.ppm` überein.

Aufgabe 3: render()

Implementieren Sie hier den oben beschriebenen Rendering-Algorithmus. Verwenden Sie dazu die vorher implementierte `draw_line()` Funktion. Wandeln Sie Gleitkommazahlen vor der Verwendung als Index in die Höhen- oder Farbkarte in ganze Zahlen um, indem Sie die Nachkommastellen durch einen Cast in den `int` Typ abschneiden.
Sie können das Ergebnis Ihres Algorithmus überprüfen, indem Sie nach dem Übersetzen den Renderer mit einem der folgenden Befehle auf einer Szenerie ausprobieren:

```
bin/render_debug maps/<x>
 bin/render_opt maps/<x>
```

Hierbei sollte `<x>` durch den Namen (ohne Dateiendung) einer Szenerie ersetzt werden.
Der obere Befehl ruft die Debug-Version des Renderers auf. Sie wird so übersetzt, dass einige zur Laufzeit auftretende Fehler leichter erkannt und untersucht werden können, allerdings mit den Kosten einer langsameren Laufzeit.
Die optimierte Version des Renderers im unteren Befehl wird so übersetzt, dass sie schneller läuft, um Ihnen auch bei leistungsschwächeren Rechnern eine akzeptable Bildrate zu ermöglichen. Allerdings können auftretende Prgrammfehler hier zu unerwartetem Verhalten führen und schwieriger zu entdecken sein.
In dem so geöffneten Fenster können Sie die Position des Beobachters in der xy-Ebene durch die Tasten W, A, S und D verändern und die Höhe des Beobachters mit Q und E anpassen.

Bonus: Effizienterer Algorithmus

Der oben beschriebene Algorithmus verrichtet unnötige Arbeit: Verdeckte Bereiche im gerasterten Areal werden auf den Bildschirm gezeichnet und anschließend beim Abtasten von näheren Bereichen überschrieben. Um Bonuspunkte zu erhalten, implementieren Sie stattdessen eine funktional äquivalente Version des Algorithmus, die jeden Pixel des Bildschirms nach dem Initialisieren mit der Himmelfarbe nur einmal beschreibt. Diese Bonuspunkte werden anhand der Laufzeit Ihrer Implementierung vergeben.
Um Bonuspunkte erhalten zu können, müssen Sie den Rückgabewert der `bonus_implemented` Funktion in der Datei `src/render.c` zu 1 abändern.

Bewertung

Ihr Projekt wird anhand von Tests bewertet. Das Testsystem wird ausschließlich die `render.c` Datei aus dem `src` Unterordner Ihres Projektes verwenden. Insbesondere werden neue Dateien sowie Änderungen an Headern

(`.h` Dateien), anderen `.c` Dateien, Makefile und den Testskripten ignoriert. Stellen Sie also sicher, dass Ihre Abgabe unter diesen Bedingungen lauffähig ist, um das Fehlschlagen aller Tests zu vermeiden.
Ihr Projekt wird mit Sanitizern übersetzt, die unter anderem den korrekten Umgang mit Speicher überprüfen. Diese liefern Ihnen hilfreichere Fehlermeldungen bei Abstürzen durch Programmfehler, führen aber auch zu fehlschlagenden Tests falls Sie Ihren Speicher nicht korrekt verwalten. Achten Sie deshalb immer auf eine saubere Speicherverwaltung.

Wir testen die Funktionen Ihres Programmes einzeln durch Unit-Testing. Stellen Sie also sicher, dass Sie die Signaturen der zu implementierenden Funktionen nicht verändern und dass die Funktionen separat voneinander funktionieren. Sie dürfen zusätzliche Hilfsfunktionen in der `render.c` Datei anlegen und nutzen.

Die einzelnen Aufgabenteile sind separat wie folgt bepunktet:

- `update_player`: 3 Punkte
- `draw_line`: 3 Punkte
- `render`: 9 Punkte + 2 Bonuspunkte

Um Punkte für die ersten beiden Teilaufgaben bekommen zu können, müssen Sie jeweils sämtliche zugeordneten öffentlichen Tests auf unseren Rechnern bestehen. Um für die letzte Teilaufgabe Punkte zu erreichen, müssen Sie **alle** öffentlichen Tests, inklusive der für die anderen Aufgabenteile, auf unseren Rechnern bestehen. Um Bonuspunkte bekommen zu können, müssen Sie ebenfalls alle öffentlichen Tests bestehen.

Projekt aufsetzen

Klonen Sie das Projekt in einen beliebigen Ordner der virtuellen Maschine für die Vorlesung:

```
git clone https://prog2scm.cdl.uni-saarland.de/git/project1/$NAME /home/prog2/project1
```

wobei Sie $NAME durch Ihren CMS-Benutzernamen ersetzen müssen. Dazu benötigen Sie Ihr CMS-Passwort und müssen im Universitätsnetz bzw. dem Universitäts-VPN sein.

Sie können an dem Projekt mit einem Texteditor Ihrer Wahl, beispielsweise dem *Kate* Texteditor, der auf der virtuellen Maschine für die Vorlesung vorinstalliert ist, arbeiten. Das Projekt können Sie von der Befehlszeile aus übersetzen, indem Sie den Befehl

```
make
```

im Hauptverzeichnis Ihres Repos verwenden. Durch den Befehl

```
make check
```

können Sie die öffentlichen Tests ausführen. Dieser Befehl führt ein Skript aus, das Sie nach dem erfolgreichen Übersetzen des Projektes auch direkt ausführen können um einzelne Tests anhand ihres Namens auszuführen:

```
test/run_tests.py -t "public.player.move_01"
```

führt beispielsweise den Test `public.player.move_01` aus.
Zum Untersuchen einzelner öffentlicher Tests können Sie zusätzlich mit dem `-v` Argument die vom Test ausgeführten Befehle anzeigen lassen und die Implementierung der Tests in `src/unit_tests.c` zur Rate ziehen. Nach dem Ausführen eines Render-Tests können Sie Ihr gerendertes Bild als `build/test.ppm` finden und beispielsweise mit dem Gwenview Bildbetrachter auf Ähnlichkeit mit der jeweiligen Referenz in `test/ref_images/` überprüfen.

Viel Erfolg!

A Herleitung zur Berechnung der dargestellten Höhe

Wir haben in der Aufgabenstellung die folgende Formel zur Berechnung der dargestellten Höhe eines Punktes Q auf einem Zielbildschirm angenommen:

$$v_S = \frac{W_S}{2} \cdot \frac{Q_z - P_z}{d} + \frac{H_S}{2}$$

Hierbei sind H_S und W_S die Höhe und Breite des Zielbildschirms, Q_z ist die Höhe des Punktes laut der Höhenkarte, P_z die Höhe des Beobachters und d der Abstand der gerade abgetasteten Senkrechten vom Beobachter. Abbildung 5 zeigt die Herleitung für diese Formel.

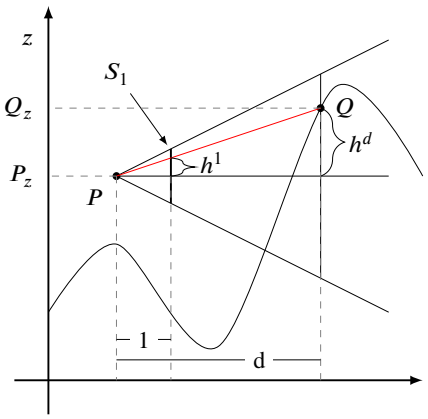


Abbildung 5: Berechnung der dargestellten Höhe eines Punktes in der Seitenansicht

Wir ermitteln v_S, indem wir berechnen, wo die Strecke zwischen Betrachter P und betrachtetem Punkt Q einen Bildschirm S_1 mit Abstand 1 vom Betrachter (und gleichem Seitenverhältnis wie der Zielbildschirm) schneidet. Wir berechnen dazu den vertikalen Abstand h^1 vom Mittelpunkt des Bildschirms S_1. Die beiden Dreiecke in der Abbildung mit Eckpunkt P und Seite h^1 bzw. h^d haben das gleiche Seitenverhältnis, also gilt folgende Gleichung:

$$\frac{h^1}{1} = \frac{h^d}{d} \Leftrightarrow h_1 = \frac{h^d}{d}$$

Weiterhin ist h^d die Differenz der Höhe des Punktes Q und der Höhe des Beobachters:

$$h^d = Q_z - P_z$$

Damit gilt:

$$h^1 = \frac{Q_z - P_z}{d}$$

Um daraus die Höhe v_S auf dem Zielbildschirm zu berechnen, müssen wir von der Höhe H_1 des Bildschirms mit Abstand 1 zu der Höhe H_S des Zielbildschirms skalieren und die Hälfte der Zielbildschirmhöhe addieren. Die Höhe H_1 wird durch das Seitenverhältnis des Bildschirms und der Breite W_1 des Bildschirms mit Abstand 1 bestimmt:

$$\frac{H_1}{W_1} = \frac{H_S}{W_S} \Leftrightarrow H_1 = W_1 \cdot \frac{H_S}{W_S}$$

Wegen des angenommenen horizontalen Sichtwinkels von 90° gilt $W_1 = 2$. Damit ist der nötige Skalierungsfaktor

$$\frac{H_S}{H_1} = \frac{H_S \cdot W_S}{W_1 \cdot H_S} = \frac{W_S}{W_1} = \frac{W_S}{2}$$

Somit ergibt sich die obige Formel:

$$v_S = \frac{W_S}{2} \cdot h^1 + \frac{H_S}{2} = \frac{W_S}{2} \cdot \frac{Q_z - P_z}{d} + \frac{H_S}{2}$$