HTW-Berlin

Computer Engineering

CE 22 - Grundlagen der Programmierung

Wintersemester 2023

Belegarbeit Seminarprojekt

Quoc Thong Truong

Berlin, 12. August 2025

Inhaltsverzeichnis

1 Mandelbrot

Das Hauptziel dieses Thema ist die Entwicklung eines grafischen Mandelbrotgenerators. Daher sind die Berechnungs- und Zeichenschritte der Mandelbrotmenge sehr wichtig. Darüber hinaus sollte das System so aufgebaut sein, dass es zumindest die genannten Anforderungen erfüllt, d.h. es sollte in der Lage sein, in einer sinnvollen Initialkonfiguration für den Ausschnitt zu starten, die Mandelbrotmenge grafisch darzustellen, in die Mandelbrotmenge hinein- und herauszuzoomen und schließlich eine Konfiguration zu speichern und wiederherzustellen.

1.1 Lösung

In meinem Mandelbrot-Projekt habe ich es in viele kleine separate Dateien aufgeteilt:

Hauptprogramm:	`mandelbrot.c`
Berechnung der komplexen Zahl:	`compute.c`
Zeichnen der Mandelbrotmenge:	`mandelbrot_draw.c`
Zoomen der Mandelbrotmenge und Verschieben:	`zoom.c`
Speichern und Wiederherstellung der Mandelbrotmenge:	`save_and_restore.c`

1.1.1 compute.c

In dieser Datei befindet sich zwei wichtige Funktionen, die eine große Rolle für die Abbildung eines Punktes von einem Bereich auf einen anderen sowie die Berechnung von der komplexen Zahl spielen.

```
double map(double value, double in_min,
double in_max, double out_min, double out_max){
     return (value - in_min) * (out_max - out_min) /
(in_max - in_min) + out_min;
}
```

Listing 1: **Die Funktion** `map()` - Die Funktion gibt einen Wert zurück, der von einem Bereich auf einen gewünschten Bereich abgebildet wird.

Listing 1 zeigt die map-Funktion, die verwendet wird, damit ein Wert von einem Bereich auf einen anderen abgebildet werden kann. Diese Funktion wurde in der Zeichenfunktion `mandelbrot_draw.c` mehrfach genutzt, um die Punkte zu verschieben, sodass die Mandelbrotmenge im gewünschten Ausschnitt dargestellt werden kann.

```
// [...]
result->a_init = current->a;
result->b_init = current->b;

result->a = result->a_init;
result->b = result->b_init;

int n = 0;
for(int i = 0; i < viewport->max_iterations; i++) {
result->a_new = result->a * result->a - result->b * result->b;
result->b_new = 2.0 * result->a * result->b;

result->a = result->a_new + result->a_init;
result->b = result->b_new + result->b_init;

if (sqrt((result->a * result->a) + (result->b * result->b))
> 2) {
break;
}
n++;
}
// [...]
```

Listing 2: **Die Funktion** `multiply_comp()` - Die Funktion berechnet die komplexe Zahl und klassifiziert, welcher Punkt zur Mandelbrotmenge gehört und welcher nicht.

Listing 2 illustriert die Funktion, die Iterationen durchführt, um festzustellen, ob die gegebene komplexe Zahl in der Mandelbrotmenge enthalten ist. Sie aktualisiert die Real- und Imaginärteile des Ergebnisses unter Verwendung der Mandelbrot-Iterationsformel. Wenn der Betrag der komplexen Zahl 2 überschreitet, bricht die Schleife ab und zeigt an, dass der Punkt nicht in der Mandelbrotmenge enthalten ist. Die Variable `n` hält die Anzahl der Iterationen fest. Wenn ein Punkt zur Mandelbrotmenge gehört, wird `n`

einen Wert von 200 bekommen, ansonten wird `n` einen Wert kleiner als 200 haben.

1.1.2 mandelbrot_draw.c

```
for (int x = 0; x < WIDTH; x++) { // x is a
for (int y = 0; y < HEIGHT; y++) { // y is b

current->a = viewport->center_x + map(x, 0, WIDTH,
viewport->min_x, viewport->max_x);
current->b = viewport->center_y + map(y, 0, HEIGHT,
viewport->min_y, viewport->max_y);

int n = multiply_comp(current, viewport);
int bright = map(n, 0, viewport->max_iterations, 0, 255);

if (n == viewport->max_iterations || bright < 20) {
bright = 0;
}

int green = map(bright * bright, 0, 6502, 0, 255);
int blue = bright;
int red = map(sqrt(bright), 0, 255, 0, 255);
SDL_SetRenderDrawColor(renderer, red, green, blue, 255);
SDL_RenderDrawPoint(renderer, x, y);
  }
}
```

Listing 3: **Die Funktion** `mandelbrot_draw()` - Die Funktion ist verantwortlich für das Zeichnen der Mandelbrotmenge.

Aus Listing 3 kann man sehen, dass diese Funktion verschachtelte Schleifen durchführt, um jedes Pixel im Rendering-Fenster zu durchlaufen. Für jedes Pixel berechnet sie auf der Grundlage der Funktion `map()` den Realteil (`current->a`) und den Imaginärteil (`current->b`) von der komplexen Zahl. Anschließend wird die Funktion `multiply_comp` aufgerufen, um die Anzahl der Iterationen `n` zu berechnen, die benötigt werden, um festzustellen, ob die komplexe Zahl zu der Mandelbrotmenge gehört.

Die Helligkeit (`bright`) des Pixels wird dann auf der Grundlage der Anzahl der Iterationen berechnet. Wenn die maximale Anzahl von Iterationen erreicht ist oder die Helligkeit kleiner als 20 ist, wird die Helligkeit auf 0 gesetzt. Die Werte für Rot, Grün und Blau werden auf der Grundlage der Helligkeit bestimmt und den entsprechenden Bereichen zugeordnet.

Abschließend werden die Farben mit der Funktion `SDL_SetRenderDrawColor()` festgelegt und ein Punkt an der aktuellen Pixelposition `(x, y)` auf dem Renderer gezeichnet. Dieser Prozess wird für jedes Pixel wiederholt, sodass die komplette Mandelbrotmenge sichtbar ist.

1.1.3 zoom.c

Um den dritten und vierten Anforderungen zu genügen, habe ich in einer separaten Datei zwei Funktionen, die es möglich macht, aus der Mandelbrotmenge hinein- und herauszuzoomen . Zusätzlich gibt es noch vier andere Funktionen, die dabei helfen kann, nach oben, unten, links und rechts in der Mandelbrotmenge zu bewegen.

```
void zoom_in(view *viewport){
        viewport->min_x *= viewport->zoom_factor;
        viewport->max_x *= viewport->zoom_factor;
        viewport->min_y *= viewport->zoom_factor;
        viewport->max_y *= viewport->zoom_factor;
}

void zoom_out(view *viewport){
        viewport->min_x /= viewport->zoom_factor;
        viewport->max_x /= viewport->zoom_factor;
        viewport->min_y /= viewport->zoom_factor;
        viewport->max_y /= viewport->zoom_factor;
}
```

Listing 4: **Die Funktionen `zoom_in` und `zoom_out`** - Die Funktion erlaubt, aus der Mandelbrotmenge hinein- und herauszuzoomen.

Wie man in Listing 4 sehen kann, zoomt die Funktion `zoom_in` in die Mandelbrotmenge, indem sie die Minimal-und Maximalwerte von x und y mit dem Zoomfaktor mulipliziert. Dadurch wird der durch das aktuelle Ansichtsfenster definierte Bereich effektiv vergrößert.

Umgekehrt zoomt die Funktion `zoom_out` aus der Mandelbrotmenge heraus, indem sie die Minimal- und Maximalwerte von `x` und `y` durch den Zoomfaktor dividiert.

```
void move_left(view *viewport) {
viewport->center_x -= viewport->move_factor *
(viewport->max_x - viewport->min_x);
}

void move_right(view *viewport) {
viewport->center_x += viewport->move_factor *
(viewport->max_x - viewport->min_x);
}

void move_up(view *viewport) {
viewport->center_y -= viewport->move_factor *
(viewport->max_y - viewport->min_y);
}

void move_down(view *viewport) {
viewport->center_y += viewport->move_factor *
(viewport->max_y - viewport->min_y);
}
```

Listing 5: **Die Funktionen, die die Bewegung behandeln** - Die Funktionen helfen dabei, sich in der Mandelbrotmenge zu bewegen.

Wie man in Listing 5 sehen kann, passt das Bewegen nach links und rechts die `center_x`-Koordinate des Ansichtsfensters an, und das Bewegen nach oben und unten passt die `center_y`-Koordinate an. Die Größe der Bewegung wird durch den `move_factor` multipliziert mit der Breite oder Höhe des aktuellen Ansichtsfensters bestimmt. Dies macht es möglich, die Ansicht in verschiedene Richtungen zu schwenken.

1.1.4 save_and_restore.c

Speichern und Wiederherstellung der Konfiguration ist auch eine der Anforderungen in diesem Thema.

```
void save_configuration(const view *viewport) {
FILE *file = fopen("config.txt", "w");
if (file != NULL) {
fprintf(file, "min_x: %g\n", viewport->min_x);
fprintf(file, "max_x: %g\n", viewport->max_x);
// [...]
fclose(file);
  }
}
```

Listing 6: **Die Funktion** `save_configuration` - Die Funktion hilft dabei, den aktuellen Zustand der Mandelbrotmenge zu speichern.

Listing 6 zeigt, wie man den aktuellen Zustand von der Mandelbrotmenge speichert. `fprintf()` wurde hier verwendet, um die benögitgen Parameter zu schreiben. Ein bemerkenswerter Punkt ist, dass der Formatbezeichner für Fließkommazahlen `%g` statt `%lf` genutzt wurde, weil er je nach Genauigkeit der Zahl automatisch zwischen `%f` und `%e` wählen kann.

```
void restore_configuration(view *viewport) {
FILE *file = fopen("config.txt", "r");
if (file != NULL) {
int result = fscanf(file, "min_x: %lf\n", &viewport->min_x);
result += fscanf(file, "max_x: %lf\n", &viewport->max_x);
// [...]
if (result != 9) {
fprintf(stderr, "Error!");
return;
  }

fclose(file);
  }
}
```

Listing 7: **Die Funktion** `restore_configuration` - Die Funktion hilft dabei, den aktuellen Zustand der Mandelbrotmenge zu wiederherzustellen.

Listing 7 zeigt, wie man den aktuellen Zustand von der Mandelbrotmenge

wiederherstellt. `fscanf()` wurde hier verwendet, um die benögitgen Parameter zu lesen. Die Variable `result` wurde genutzt, um zu checken, ob die Anzahl der Werte (in diesem Fall ist 9) erfolgreich gelesen wurde.

1.1.5 mandelbrot.c

Diese Datei ist verantwortlich dafür, nicht nur SDL und das Fenster zu initialisieren, sondern auch das Ereignis zu behandeln und die Mandelbrotmenge in der endlosen Schleife zu zeichnen.

```
// [...]
if (event.type == SDL_QUIT) {
return 0;
}
else if (event.type == SDL_KEYDOWN) {
switch (event.key.keysym.sym) {
case SDLK_ESCAPE:
return 0;
case SDLK_q: // Zoom in
viewport.max_iterations += 20;
zoom_in(&viewport);
break;
case SDLK_e: // Zoom out
viewport.max_iterations -= 10;
zoom_out(&viewport);
break;
// [...]
case SDLK_x: // Save configuration
save_configuration(&viewport);
break;
case SDLK_z: // Restore configuration
restore_configuration(&viewport);
break;
        }
}
// [...]
```

Listing 8: **Die Erignisbehandlung innerhalb der inneren While-Schleife** - Es gibt zwei Ereignisse in der inneren While-Schleife, die Beendingungsereignis und Tastendruckereignis sind

Listing 8 demonstriert, worum es in der inneren While-Schleife geht. Außer des Beendigungsereignisses spielt das Tastendruckereignis auch eine große Rolle. Man kann verschiedene Tasten drücken, um verschiedene Aktionen auszulösen, wie z. B. das Zoomen in das Bild hinein und aus dem Bild heraus, das Speichern und die Wiederherstellung der aktuellen Konfiguration. Jedes Mal, wenn man die Vergrößerungstaste und die Bewegungstaste drückt, wird die Anzahl der Iterationen um 20 erhöht, so dass die Mandelbrotmenge fantastischer und detaillierter aussieht. Beim Herauszoomen wird die Anzahl der Iterationen um 10 verringert. Nachdem eine Taste gedrückt wurde, wird die innere Schleife verlassen und zur äußeren Schleife, der endlosen while-Schleife, übergegangen. Hier wird die Zeichenfunktion `mandelbrot_draw.c` aufgerufen, um die Mandelbrotmenge zu zeichnen und zu aktualisieren. Zu Beginn, wenn man das Programm zum ersten Mal startet und noch keine Taste drückt, wird das System automatisch die Mandelbrotmenge mit Hilfe der Zeichenfunktion `mandelbrot_draw.c` in dieser Endlos-While-Schleife auf dem SDL-Fenster anzeigen.

1.2 Abbildungen der Mandelbrotmenge

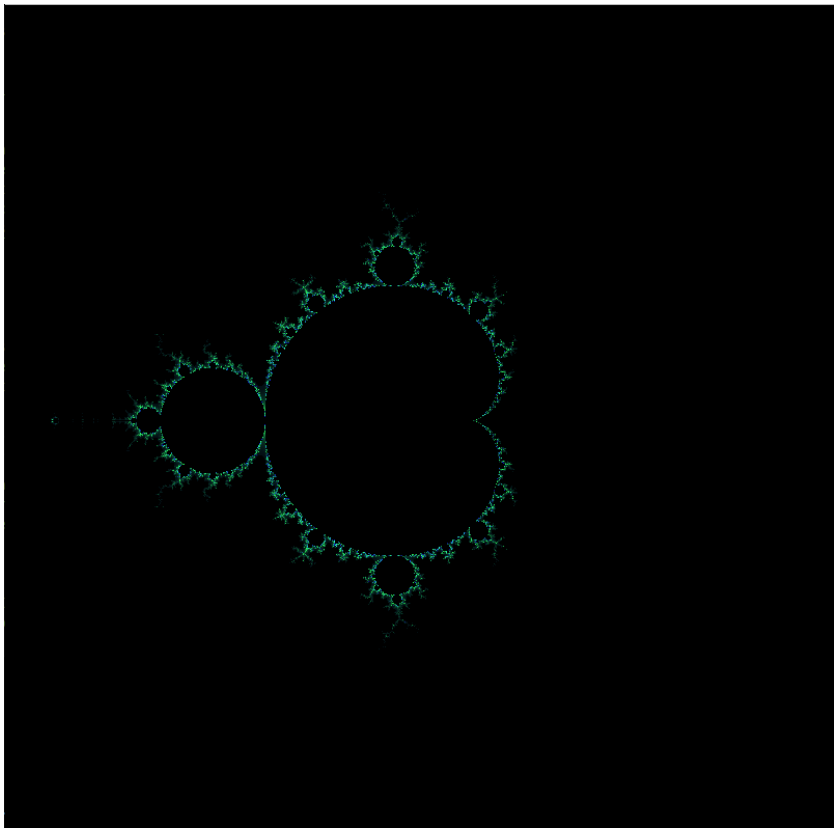

Abbildung 1: Die Mandelbrotmenge für den Ausschnitt mit x- und y-Werten, die von dem Bereich [-2.0, 2.0] gehören

Abbildung 1 zeigt die Konfiguration der Mandelbrotmenge für den vorgegebenen Ausschnitt mit x- und y-Werten, die im Bereich von [-2.0, 2.0] gehören.

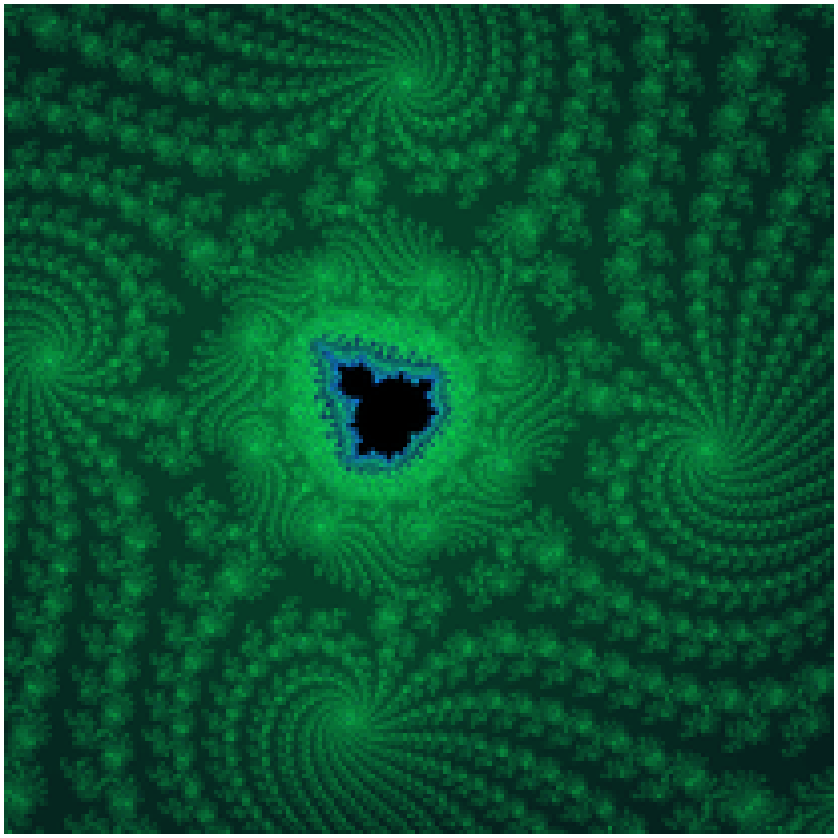

Abbildung 2: Die Mandelbrotmenge beim Hineinzoom mit der Anzahl der Iterationen von 3460

Abbildung 2 demonstriert die aktuelle Konfiguration der Mandelbrot-Menge in einem anderen Ausschnitt mit der Anzahl der Iterationen von 3460.

1.3 Test-coverage

Ich habe den Test-coverage für zwei Dateien geschrieben, `compute.c` und `zoom.c`. Der Grund dafür ist, dass `compute.c` sehr wichtig und verantwortlich für die Berechnung von der komplexen Zahl und die Prüfung, ob ein Punkt zu der komplex ist, und `zoom.c` stellt die Funktionen zur Verfügung, die es möglich machen, die Mandelbrotmenge hinein- und herauszuzoomen, usw.

Für die Dateien `mandelbrot_draw.c`, `save_and_restore.c` und `mandelbrot.c` habe ich den Test-coverage nicht geschrieben, weil es bei `mandelbrot_draw.c` nur um das Zeichnen der Mandelbrotmenge geht, bei `save_and_restore.c` um das Speichern und Wiederherstellung der akutuellen Konfiguration und bei `mandelbrot.c` um die Ereignisschleife.

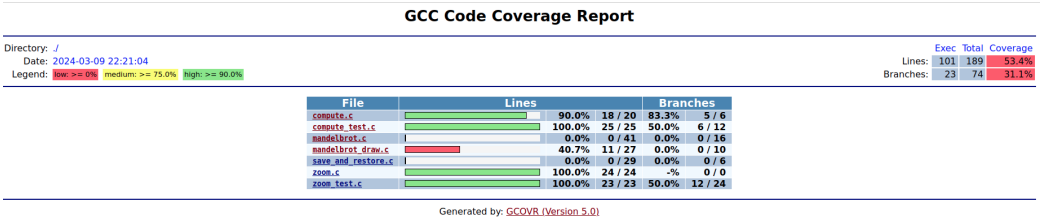

Abbildung 3: Screenshot der HTML-Ausgabe

2 Fazit

Meiner Meinung nach besteht das Hauptziel dieses Projekts darin, die in diesem Semester erworbenen Kenntnisse der Programmiersprache C zu überprüfen. Außerdem ist das Mandelbrot-Projekt ein wunderbares Projekt, bei dem man die Kenntnisse der Mathematik anwenden muss. Dann muss man das ganze System mit den Kenntnissen der Programmierung realisieren. Nun kann das Programm nicht nur eine Konfiguration in einem bestimmten Ausschnitt grafisch darstellen, sondern auch in die Mandelbrotmenge hinein- und herauszoomen, eine Konfiguration speichern und laden, wenn bestimmte Tasten gedrückt werden. Und das Programm erfüllte alle fünf Anforderungen zu diesem Thema. Ein weiterer erwähnenswerter Punkt bei diesem Projekt ist, dass dieses Programm auf verschiedenen Betriebssystemen laufen soll. Daher habe ich mein Makefile bereits so angepasst, dass mein Programm auf beiden Betriebssystemen, nämlich Ubuntu und MacOS, laufen kann.

3 Verzeichnisse der Abbildungen und Listings

Abbildungsverzeichnis

List of Listings