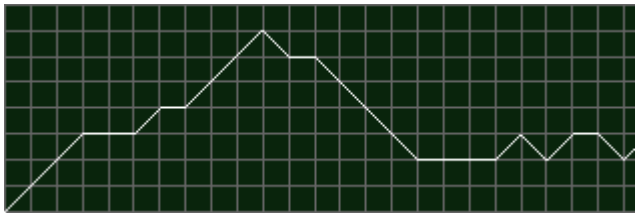
Dokumentation zur Aufgabe 3 – Alle Alpen



Programmdokumentation von

Christian Siewert

für den 27. Bundeswettbewerb Informatik

1 Aufgabenstellung

Eine Schülergruppe möchte ein 2D-Computerspiel entwickeln. Einer der Schüler ist für die dynamische Generierung der Hintergrundbilder zuständig. Die Szenerie soll dabei ein Gebirge sein welches man als Folge von Höhenunterschieden beschreiben kann. Dabei sei ein Gebirgszug der Länge N definiert als h0 = hN = 0 und | h i – h i-1 | < 1 für i = 1,...,N.
Es soll nun ein Programm entwickelt werden welches einen solchen Gebirgszug für die Länge von N = 100 zeichnet. Darüber hinaus sollen alle Gebirgszüge der Länge 6 ausgegeben werden und die Anzahl aller Gebirgszüge der Länge 16 bestimmt werden.
Außerdem sollte eine geeignete Programmiersprache gesucht werden, mit der man einfache Zeichnungen realisieren kann.

2 Lösungsidee

Es gibt viele Möglichkeiten um Zeichnungen auf dem Monitor zu realisieren. Ich habe mich dabei für die plattform- und programmiersprachenunabhängige API OpenGL entschieden. OpenGL unterstützt viele Befehle die die Entwicklung von komplexer 3D-Grafik (oder in diesem Fall einfache 2D-Grafik) erlaubt.
Wie jedes komplexe Problem kann man auch dieses Problem in mehrere Teilprobleme zerlegen. Vorher sollte man jedoch klären wie die mathematische Folge aus der Aufgabenstellung zu verstehen ist. Dabei leuchtet ein das der Höhenunterschied zwischen zwei benachbarten Punkten nur 0 oder 1 sein kann. Der erste und letzte Punkt müssen dabei die Höhe 0 haben.
Als nächstes sollte man sich Gedanken machen wie man diese Gebirgszüge nun zeichnet. Ein Gebirgszug der Länge 100 ist definiert durch 101 Punkte. Diese Punkte könnte man innerhalb einer Schleife festlegen und später verbinden. Dabei ist darauf zu achten das der Gebirgszug bei der Höhe 0 endet. Die einzelnen Höhen der Punkte könnte man mit einem zufallsbasierenden System festlegen. So kann pro Schleifendurchlauf eine Zufallszahl festgelegt werden, welche dann differenziert werden muss um auszuwerten ob die Höhe des nächsten Punkts bei h_{i-1}, $h_{(i-1)+1}$ oder $h_{(i-1)-1}$ liegt. Dabei darf $h_{(i-1)-1}$ natürlich nicht negativ sein sondern maximal 0. Wenn diese Verfahrensweise so durchgeführt wird gibt es allerdings Probleme beim Ende des Gebirges. Es ist durch ein zufallsbasierendes System nicht sichergestellt das das Gebirge bei ($h_{n+1} = 0$) endet.

Dies kann man umgehen indem man prüft ob die Höhe des nächsten Punktes größer als die Differenz von 100 und des aktuellen Werts auf der Abszisse ist. Sollte dies zutreffen so muss die Höhe des nächstes Punkts eine Dekrementierung der Höhe des letzten Punkts sein. Somit sollte das Gebirge am Ende bei der Höhe 0 enden ohne die mathematische Folge aus der Aufgabenstellung zu missachten. Speichern könnte man die Höhe der Punkte in einem Integer Array. Die Punkte könnten dann durch eine OpenGL-spezifische Funktion verbunden werden und es sollte sich dann ein Gebirge ergeben.
Aus der Aufgabenstellung geht hervor das eine Prozedur geschrieben werden soll der man als Parameter ein Gebirgszug der Länge N übergibt. Die Prozedur soll diesen Gebirgszug anschließend zeichnen. Eine Prozedur zu schreiben welche einen Gebirgszug zeichnet ist API-spezifisch und sollte nicht das Problem sein. Allerdings muss man sich Gedanken über den Parameter machen der übergeben werden soll. Eine einfache aber gleichzeitig gute Lösung ist das Benutzen eines Arrays vom Typ Integer. Die Größe dieses Arrays kann angepasst werden sodass Gebirgszüge verschiedener Länge gezeichnet werden können. Beim zeichnen des Gebirgszuges könnte das Array über Indizes abgefragt werden, in welchen die Höhe des aktuellen Punkts gespeichert ist.
Später kann man diese Prozedur benutzen um alle Gebirgszüge der Länge 6 auszugeben. Um dies zu bewerkstelligen könnte man auf Techniken wie Rekursion oder Backtracking zurückgreifen. Eine einfachere, aber laufzeitaufwendigere, Lösung könnte man erreichen indem man einfach alle möglichen Höhen der jeweiligen Punkte durchprobiert und am Ende alle falschen Gebirgszüge ausfiltert. Die gleiche Vorgehensweise könnte man benutzen um alle möglichen Gebirgszüge der Länge 16 zu bestimmen. Hier sollte man sich aber wirklich nur auf das Zählen der Gebirge festlegen und nicht auf das Zeichnen. Eine Auflistung aller möglichen Teilprobleme und deren Lösungsidee finden Sie in Anlage A.

3 Programm-Dokumentation

Die Umsetzung der gestellten Aufgaben stellten sich als einfacher als gedacht heraus. Mein Programm besteht aus einem Hauptfenster in dem Zeichnungen ausgeführt werden. Darüber hinaus habe ich noch ein Subwindow erstellt in dem bestimmte Texte ausgegeben werden und der User zu Aktionen aufgefordert wird. So kann er sich zum Beispiel entscheiden die Taste F1 zu drücken um sich einen Gebirgszug der Länge 100 ausgeben zu lassen. Beim Druck auf die Taste

F2 werden alle möglichen Gebirgszüge der Länge 6 ausgegeben. Die Auflösung der Anwendung ist voreingestellt auf 1400 Pixel * 800 Pixel.
Die Realisierung dieser Aufgabe wurde genauso durchgeführt wie in der Lösungsidee beschrieben. Ich schrieb zu diesem Zweck die Funktion `draw_mountains_100()`. Es wurde eine Variable (`next_point`) deklariert mit welcher im Verlauf des Programms gerechnet wird. Diese hat am Anfang den Wert 0. Innerhalb der Funktion wird eine Schleife gestartet die 101 mal durchlaufen wird. In dieser Schleife wird eine Zufallszahl bestimmt die zwischen 1 und 3 liegen kann. Sollte der Wert der Zahl 1 betragen so ergibt sich die Höhe des nächsten Punkts aus Höhe des vorherigen Punkts – 1 (`next_point = next_point – 1`). Bei einem Wert von 2 ergibt sich die Höhe des nächsten Punkts aus der Höhe des vorherigen Punkts + 1. Bei einem Wert von 3 passiert nichts, da der Höhenunterschied zum vorherigen Punkt 0 beträgt.
Sollte das Ereignis eintreten das die Höhe des nächsten Punkts negativ ist (der Punkt also unterhalb des Horizonts liegt) so wird die Variable `next_point` auf 0 gesetzt. Somit ist sichergestellt das die Höhe des nächsten Punkts niemals negativ wird.
Nachdem der Wert der Variable geprüft und anschließend darauf reagiert wurde, wird noch geprüft ob die Höhe des nächsten Punkts größer gleich 100 – (aktueller Wert auf der Abszisse) ist. Dieser Wert (im Programm die Variable `akt_x`) wird nach jedem Schleifendurchlauf inkrementiert. Somit ist gewährleistet das der Gebirgszug am Ende sauber bei der Höhe 0 abschließt. Anschließend wird der Wert der aktuellen Höhe in einem Integerarray gespeichert.
Nachdem nun diese Schleife durchlaufen wurde gibt es im Speicher ein Integer Array mit hundert verschiedenen Höhenwerten. Das letzte Feld des Arrays wird mit dem Wert 0 belegt. Nun kann man sich also darum kümmern diese Punkte auf den Bildschirm zu zeichnen und zu einem Gebirge zu verbinden. Dabei wird wieder hundertmal eine Schleife durchlaufen in der jedes Mal eine OpenGL Funktion aufgerufen wird. Diese Funktion (`glVertex2i`) wird mit den aktuellen Werten des Integer-Arrays aufgerufen, welches durchlaufen wird. Bevor dies passiert wird ein Startwert (0;0) festgelegt. Nachdem der letzte Punkt gezeichnet wurde verbindet OpenGL die Punkte zu einer Linie. Dadurch entsteht ein gebirgstypisches Bild. Zur besseren Übersicht wird vor dem eigentlichen Gebirge noch ein Koordinatensystem samt Beschriftung gezeichnet. OpenGL ruft die Funktion `draw_mountains_100` (sofern F1 gedrückt wurde) ununterbrochen auf. Um zu verhindern das laufend neue Gebirge gezeichnet werden, wird nach dem Festlegen der Punkte die Variable `hundert_gezeichnet` auf `true` gesetzt. Dadurch wird beim Aufruf der Funktion verhindert das neue Punkte festgelegt werden. Stattdessen wird eine Schleife durchlaufen in dem das alte Integer Array abgefragt wird und dementsprechend gezeichnet wird. Wenn man die Taste 'n'

drückt wird die Variable `hundert_gezeichnet` auf `false` gesetzt und dadurch ein neuer Gebirgszug gezeichnet. Für eine Ausgabe eines Gebirgszuges der Länge 100 verweise ich auf die letzte Seite dieser Dokumentation. Ein Programmablaufplan über diesen Algorithmus finden Sie in Anlage B. Als nächstes habe ich eine Funktion geschrieben welche mit einem geeigneten Parameter N ein Gebirgszug der Länge N zeichnet. Der Parameter N ist dabei ein Integer Array, dessen Feldgröße man in der Deklarationszeile der Variable ändern kann. Man übergibt dieser Funktion (`void zeichne_gebirgszug(int g_punkt[x])`) dieses Array. In dem Array sollten in den verschiedenen Feldern Höhenwerte stehen. Diese Höhenwerte werden durch eine Schleife innerhalb der Funktion ausgelesen und auf den Monitor gezeichnet. Ein Beispiel für den Fuktionsauruf könnte so aussehen:

```
zeichne_gebirgszug (g_punkt);
```

Das Array mit einer Speichergröße von 7 Feldern könnte wie folgt belegt sein:

i	0	1	2	3	4	5	6
g_punkt[i] (Höhe)	0	1	2	3	2	1	0

Durch die Belegung des Arrays mit diesen Werten würde ein symmetrischer Gebirgszug entstehen.
Die nächste Aufgabe bestand darin mit der Prozedur zum zeichnen eines Gebirgszuges alle Gebirgszüge der Länge 6 auszugeben. In meiner Lösung habe ich mich dafür entschieden das ganze mit Schleifen zu lösen. Ein Gebirgszug der Länge 6 besitzt 7 Punkte. Die Höhe des ersten und letzten Punktes ist dabei 0. Die Höhe des zweiten Punkts kann maximal 1 sein, die Höhe des zweiten Punkts maximal 2 und so weiter. In der Mitte des Gebirges geht das ganze dann natürlich wieder „abwärts“.
Um nun alle Gebirgszüge der Länge 6 auszugeben werden in meiner Lösung alle Gebirgszüge durchprobiert und danach alle Gebirgszüge welche die mathematische Folge aus der Aufgabenstellung verletzen herausgefiltert. Da der erste und letzte Punkt 0 ist, benötigt man zur Realisierung statt 7 (für 7 Punkte) nur 5 Schleifen. Diese zählen dabei von 0 bis zur maximalen Höhe des jeweiligen Punkts. Schleife 1 zählt von 0 bis 1, Schleife 2 von 0 bis 2 und so weiter. Innerhalb der letzten Schleife wird dem Array g_punkt die Höhe des aktuellen Punkts zugewiesen. Durch dieses Vorgehen ist sichergestellt das wirklich alle Gebirgszüge berechnet werden. Nachdem ein Array mit Werten belegt ist (die letzte Schleife also abgearbeitet ist),

wird geprüft ob diese Punkte gezeichnet werden dürfen. Dazu wird noch einmal eine Schleife aufgerufen in welcher das Array durchlaufen wird und geprüft wird ob `g_punkt[x+1] - g_punkt[x]` größer als 1 ist oder `g_punkt[x+1] - g_punkt[x]` kleiner als -1 ist. Ist dies der Fall so wird die Variable `zeichne_zug` auf `false` gesetzt und der Gebirgszug nicht gezeichnet. Andernfalls erfolgt eine Ausgabe des Gebirgszuges. Das Spiel geht weiter bis alle Schleifen abgearbeitet sind.Für jeden Gebirgszug wird, aus Gründen der Übersicht, noch ein kleines Koordinatensystem gezeichnet. Es folgt die Ausgabe für alle Gebirgszüge der Länge 6:

Um die Anzahl aller Gebirgszüge der Länge 16 zu bestimmen habe ich den gleichen Weg gewählt. Allerdings musste ich hierbei logischerweise sowohl das Array vergrößern als auch eine

größere Anzahl von Schleifen implementieren.
Durch die Verwendung von Schleifen anstatt von Techniken wie Rekursion oder Backtracking hat das Berechnen aller möglichen Gebirgszüge für N=16 genau 46 Minuten gedauert. Dies führte mich zu dem Schluss das ich keine ideale Lösung besitze.
Allerdings kann ich mit Sicherheit sagen das die Anzahl für alle möglichen Gebirge der Länge 16 genau **853.467** beträgt. Dies ist doch sehr beachtlich. Eine Auflistung aller Funktionen, deren Zweck und aller wichtigen Variablen finden sie in Anlage C und D.

4 Programm-Ablaufprotokolle

Im folgenden möchte ich ein Programm-Ablaufprotokoll für die Ausgabe eines Gebirgszuges der Länge 100 präsentieren. Dabei werde ich mich auf diese Aufgabe beschränken da die anderen Aufgaben in ähnlicher Weise realisiert wurden. Es folgt die Ausgabe nach dem ersten Schleifendurchlauf:

i (Schleifenvariable)	rand_next_point	next_point	akt_hoehe[i+49]
-49	3	0	0

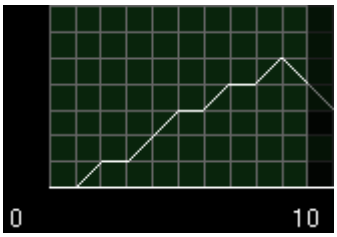
Die Zufallszahl nach dem ersten Schleifendurchlauf betrug 3. Dadurch verändert sich die Höhe nicht. Folglich sollte die Höhe des zweiten Punkts genauso groß sein wie die des ersten Punkts (0).

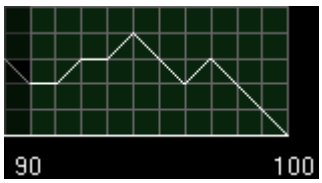
i (Schleifenvariable)	rand_next_point	next_point	akt_hoehe[i+49]
-48	2	1	1
-47	3	1	1
-46	2	2	2
-45	2	3	3
-44	3	3	3
-43	2	4	4
-42	3	4	4
-41	2	5	5
-40	1	4	4

Dies waren die ersten 10 Punkte des Gebirges. Das ganze verläuft genauso weiter. Die letzten 10 Punkte waren die folgenden:

i (Schleifenvariable)	rand_next_point	next_point	akt_hoehe[i+49]
41	3	2	2
42	2	3	3
43	3	3	3
44	2	4	4
45	1	3	3
46	1	2	2
47	2	3	3
48	1	2	2
49	1	1	1
50	1	0	0

Die Ausgaben für die ersten 10 und die letzten 10 Punkte sahen wie folgt aus:





Man sieht das mein Programm die geforderten Aufgaben erfüllt und Ausgaben in einer ansprechenden Form darstellt.

5 Quellcode

```
// ************************ Bibliotheken die implementiert werden müssen *******

#include <GL/glut.h>                                    // GLUT Bibliothek
#include <GL/gl.h>                                      // OpenGL32 Bibliothek
#include <GL/glu.h>                                     // GLu32 Bibliothek
#include <unistd.h>                                     // Header sleeping
#include <stdio.h>                                      // Rudimentäre Dateioperationen, Standard Ein -und ausgabe
#include <stdlib.h>                                     // Wichtig für malloc(), etc.
#include <string.h>                                     // String-Implementation
#include <sys/time.h>                                   // Zeitrechnung (Relevant für FPS)
#include <string>

// ************************ Funktionsprototypen (Void) *************************

void ReSizeGLScene(int Width, int Height);              // Wird aufgerufen wenn die Fenstergröße verändert wird
void InitGL(int Width, int Height);                     // Wird aufgerufen sobald ein OpenGL Fenster erstellt wurde
void DrawGLScene();                                     // Haupt Zeichnen-Funktion
void draw_mountains_100();                              // Zeichnet Gebirgszug der Länge 100
void keyPressed(unsigned char key, int x, int y);       // Wird aufgerufen wenn eine Taste gedrückt wird
void func_key(int key, int x, int y);                   // Wird aufgerufen wenn eine Funktionstaste gedrückt wird
void textausgabe (float x, float y, const char *text, void *font);          // Ausgeben von Text auf Monitor
void idle();
void DrawGLSceneSub();
void subReshape (int w, int h);
void cfps();
void zeichne_ks(int laenge,int hoehe, float pos_x, float pos_y);
void zeichne_gebirgszug(int g_punkt[6]);

// ************************ Integer Variablen **********************************

int window;                                             // Anzahl der Fenster
int subwindow;                                          // Anzahl der SubFenster
int res_x = 1400;                                       // Auflösung - X
int res_y = 800;                                        // Auflösung - Y
int next_point = 0;                                     // Nächster Wert für draw_mountains_100
int akt_hoehe[100];
int akt_x = 0;
int rand_next_point = 0;
int g_punkt[7];
int i[5];                                               // Schleifenvariable für Ausgabe aller G_zuege der Länge 6
int ks_y = 0;

// ************************ Float Variablen ************************************

float beschriftung_x = 0;                               // Beschriftung für KS
float beschriftung_y = 0;                               // Beschriftung für KS
float x_offset = 0;
float y_offset = 0;

// ************************ Char Variablen *************************************

char strFrameRate[4]            = {0};                  // Speicher für Framerate
char strBeschriftung_x[4] = {0};
char strBeschriftung_y[4] = {0};

// ************************ Boolsche Variablen *********************************

bool draw_100 = false;
bool draw_all_6 = false;
bool hundert_gezeichnet = false;
bool zeichne_zug = true;

// ************************ Funktionen (Void) **********************************

int main(int argc, char **argv){                        // Hauptfunktion
```

```
	srand( time(NULL) );				// Starte den Zufallszahlengenerator

	glutInit(&argc, argv);				// Initialisiert GLUT

	glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE); // Typ des Display-Modus auswählen: RGB-Farbmodell und Doulebuffer

	glutInitWindowSize(res_x, res_y);		// Ein fenster mit 1400 * 800 Pixel

	glutInitWindowPosition(100, 100);		// Das fenster mit abstand von 100*100 (oben, links) anzeigen

	window = glutCreateWindow("Alle Alpen");	// Ein fenster öffnen

		glutDisplayFunc(DrawGLScene);	// DrawGLScene registrieren

		glutIdleFunc(idle);		// Solange nichts passiert, führe "DrawGLScene" aus

		glutReshapeFunc(ReSizeGLScene);	// ReSizeGLScene registrieren

		glutKeyboardFunc(keyPressed);	// KeyPressed registrieren

		glutSpecialFunc(func_key);	// func_key registrieren

		InitGL(res_x, res_y);		// Das Fenster initialisieren

	subwindow = glutCreateSubWindow(window,5,5,res_x-10,res_y / 5);  // Ein SubFenster öffnen

		glutDisplayFunc(DrawGLSceneSub);		// Haupt-Zeichnen (SubWindow) Funktion registrieren

		glutReshapeFunc(subReshape);

	glutMainLoop();					// Den Event-Loop aktivieren

	return 0;
}

void ReSizeGLScene(int Width, int Height){		// Wird aufgerufen wenn die Fenstergröße verändert wird

	if (Height==0)					// Verhindere eine Division durch 0 weil das Fenster zu klein ist
		Height=1;

	glViewport(0, 0, Width, Height);		// Setze Blickwinkel und perspektivische Transformation zurück

	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	gluPerspective(45.0f,(GLfloat)Width/(GLfloat)Height,0.1f,100.0f);
	glMatrixMode(GL_MODELVIEW);
}

void InitGL(int Width, int Height){			// Wird aufgerufen sobald ein OpenGL Fenster erstellt wurde

	glEnable(GL_TEXTURE_2D);			// Aktiviere Texturierung
	glClearColor(0.0f, 0.0f, 0.0f, 0.0f);		// Lösche den Hintergrund zur farbe schwarz

	glClearDepth(1.0);				// Aktiviert löschung des tiefen-puffers

	glMatrixMode(GL_PROJECTION);

	glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

	gluPerspective(0.0f,(GLfloat)Width/(GLfloat)Height,0.1f,100.0f);	// Seitenverhältniss berechnen

	glMatrixMode(GL_MODELVIEW);

	glColor3f(1.0f,1.0f,1.0f);
}

void DrawGLScene(){					// Haupt Zeichnen-Funktion

	glClear(GL_COLOR_BUFFER_BIT);			// Bildschirm löschen
	glLoadIdentity();				// Sicht resetten

	glTranslatef(0.0f,-27.0f,-75.0f);
```

```
	y_offset = 0;
	x_offset = 0;
	ks_y = 0;

	if(draw_100 == true){				// Zeichne Gebirgskette der Länge 100

	// *** Zeichne das Koordinatensystem inkl. Beschriftung ***

		beschriftung_x = -50;
		beschriftung_y = 0;

		glColor3f(1.0f,1.0f,1.0f);
		textausgabe(beschriftung_x-1.5, beschriftung_y-1.5, "0", GLUT_BITMAP_HELVETICA_12);

		glColor3f(0.035f,0.14f,0.047f);

		glBegin(GL_QUADS);			// Zeichne "Backframe"
			glVertex2i(-50,0);
			glVertex2i(50,0);
			glVertex2i(50,40);
			glVertex2i(-50,40);
		glEnd();

		for(int x = -50; x <= 50; x++){		// Zeichne Karos
				glColor3f(0.4f,0.4f,0.4f);
				glBegin(GL_LINES);
					glVertex2i(x,0);
					glVertex2i(x,40);
				glEnd();

				beschriftung_x++;
				if((int) beschriftung_x % 10 == 0){	// Zeichne Beschriftung
					glColor3f(1.0f,1.0f,1.0f);
					sprintf(strBeschriftung_x,"%2.f",(beschriftung_x+50));
					textausgabe(beschriftung_x-0.6, beschriftung_y-1.5, strBeschriftung_x,
GLUT_BITMAP_HELVETICA_12);
				}
		}

		beschriftung_x = -51.5;

		for(int y = 0; y <= 40; y++){		// Zeichne Karos

				glColor3f(0.4f,0.4f,0.4f);
				glBegin(GL_LINES);
					glVertex2i(-50,y);
					glVertex2i(50,y);
				glEnd();

				beschriftung_y++;
				if((int) beschriftung_y % 10 == 0){	// zeichne Beschriftung
					glColor3f(1.0f,1.0f,1.0f);
					sprintf(strBeschriftung_y,"%2.f", beschriftung_y);
					textausgabe(beschriftung_x-0.8, beschriftung_y-0.3, strBeschriftung_y,
GLUT_BITMAP_HELVETICA_12);
				}
		}

	// *** Koordinatensystem komplett ***

	draw_mountains_100();		// Zeichne jetzt die Gebirgskette (n = 100)

	}

	if(draw_all_6 == true){		// Zeichne alle g_zuege der Länge 6

		g_punkt[0] = 0;		// Erster....
		g_punkt[6] = 0;		// ...und letzter Punkt haben Höhe 0
```

```
		for(i[0] = 0; i[0] <= 1; i[0]++)    // Durch diese Schleifen werden alle Gebirgszüge durchprobiert
		for(i[1] = 0; i[1] <= 2; i[1]++)
		for(i[2] = 0; i[2] <= 3; i[2]++)
		for(i[3] = 0; i[3] <= 2; i[3]++){
		for(i[4] = 0; i[4] <= 1; i[4]++){

			for(int x = 1; x <= 5; x++)
				g_punkt[x] = i[x-1];  // Schreibe aktuellen Gebirgszug in Array der Länge 7

			for(int x = 1; x <= 5; x++) if((g_punkt[x+1] - g_punkt[x] > 1) || (g_punkt[x+1] - g_punkt[x] < -1)) zeichne_zug
= false;    // Prüft ob aktueller Gebirgszug zulässig ist. Wenn nicht dann wird zeichne_zug auf false gesetzt.

			if(zeichne_zug == true){        // Hier wird der gültige Gebirgszug gezeichnet
				zeichne_ks(6,3,-47.0f + x_offset,10.0f - y_offset);
				ks_y++;
				zeichne_gebirgszug(g_punkt);
				y_offset = y_offset + 5.0f;
				if(ks_y == 8){ x_offset = x_offset + 10.0f; y_offset = 0; ks_y = 0; }
			}

				zeichne_zug = true;
		}
		}
	}

	cfps();                         // Zeige Frames pro Sekunde
	glutSwapBuffers();  // Vertausche Vorder -und Hintergrund-Puffer (DoubleBuffer)
}

void draw_mountains_100(){

	glColor3f(1.0f,1.0f,1.0f);

	next_point = 0;
	akt_x = 0;

	if(hundert_gezeichnet == false){

			for(int i = -49; i <= 50; i++){     // Schleife wird 100mal durchlaufen

				rand_next_point = (rand () % 3) + 1;      // Ermittle Zufallszahl

				if(rand_next_point  == 1){                // Wenn Zahl = 1 dann geht's runter
					next_point = next_point - 1;
					if(next_point < 0) next_point = 0;
				}

				else if(rand_next_point == 2)             // Wenn Zahl = 2 dann geht's rauf
					next_point  = next_point + 1;

				else ;                                    // Ansonsten bleibt Höhe erhalten

				while(next_point >= (100 – akt_x))/       // Prüfe wie weit wir vom Ende weg sind und
dekrementiere evtl.
					next_point--;

				akt_x++;                                  // Inkrementiere Wer auf der Abszisse
				akt_hoehe[i+49] = next_point;             // Schreibe aktuellen Punkt ins Array
			}

		akt_hoehe[99] = 0;

		hundert_gezeichnet = true;              // Verhindert das laufend neue Gebirge gezeichnet werden
	}

	else{
		glBegin(GL_LINE_LOOP);                  // Hier wird der Gebirgszug gezeichnet

			glVertex2i(-50,0);
			for(int i = -49; i <= 50; i++) glVertex2i(i,akt_hoehe[i+49]);

		glEnd();
```

```
	}
}

void keyPressed(unsigned char key, int x, int y){		// Wird aufgerufen wenn eine Taste gedrückt wird

	switch(key){

		case  27: { glutDestroyWindow(window); close(0); break; }
		case 110: { hundert_gezeichnet = false; break; }
	}
}


void func_key(int key, int x, int y){				// Wird aufgerufen wenn eine Funktionstaste gedrückt wird

	switch(key){

		case GLUT_KEY_F1: {				// Zeichnet Gebirge der Länge 100
			if(draw_100 == false) { draw_all_6 = false; draw_100 = true; }
			else draw_100 = false;
			break; }
		case GLUT_KEY_F2: {				// Zeichnet alle Gebirge der Länge 6
			draw_100 = false;
			if(draw_all_6 == false) { draw_100 = false; draw_all_6 = true; }
			else draw_all_6 = false;
			break; }
	};
}

void textausgabe(float x, float y, const char *text, void *font){	// Funktion zum Text ausgeben

	int laenge;

	glRasterPos3f(x,y,0.0f);

	laenge = (int) strlen(text);

	for(int i = 0; i < laenge; i++)
		glutBitmapCharacter(font, text[i]);
}

void idle(){

	glutSetWindow (window);
	glutPostRedisplay ();

	glutSetWindow (subwindow);
	glutPostRedisplay ();
}

void DrawGLSceneSub(){						// Haupt - Zeichnen Funktionen für Sub-Window

	glutSetWindow (subwindow);
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glLoadIdentity();
	glTranslatef(-1.0f,-1.0f,0.0f);

	glColor3f(1.0f, 0.5f, 0.2f);
	textausgabe(0.91f,1.5f, "Alle Alpen",GLUT_BITMAP_HELVETICA_18);

	glColor3f(0.21f, 0.75f, 1.0f);
	textausgabe(0.86f, 1.0f, "Christian Siewert", GLUT_BITMAP_HELVETICA_18);

	if(draw_100 == false) glColor3f(1.0f, 1.0f, 1.0f);
	else{
		glColor3f(0.0f, 1.0f, 0.0f);
		textausgabe(1.72f,1.5f,"n   -   Nochmal zeichnen", GLUT_BITMAP_HELVETICA_12);
		glColor3f(0.0f, 1.0f, 0.0f);
	}

	textausgabe(0.07f, 1.5f, "F1    -   Zeichne Gebirgskette der Laenge 100", GLUT_BITMAP_HELVETICA_12);

	if(draw_all_6 == false) glColor3f(1.0f, 1.0f, 1.0f);
```

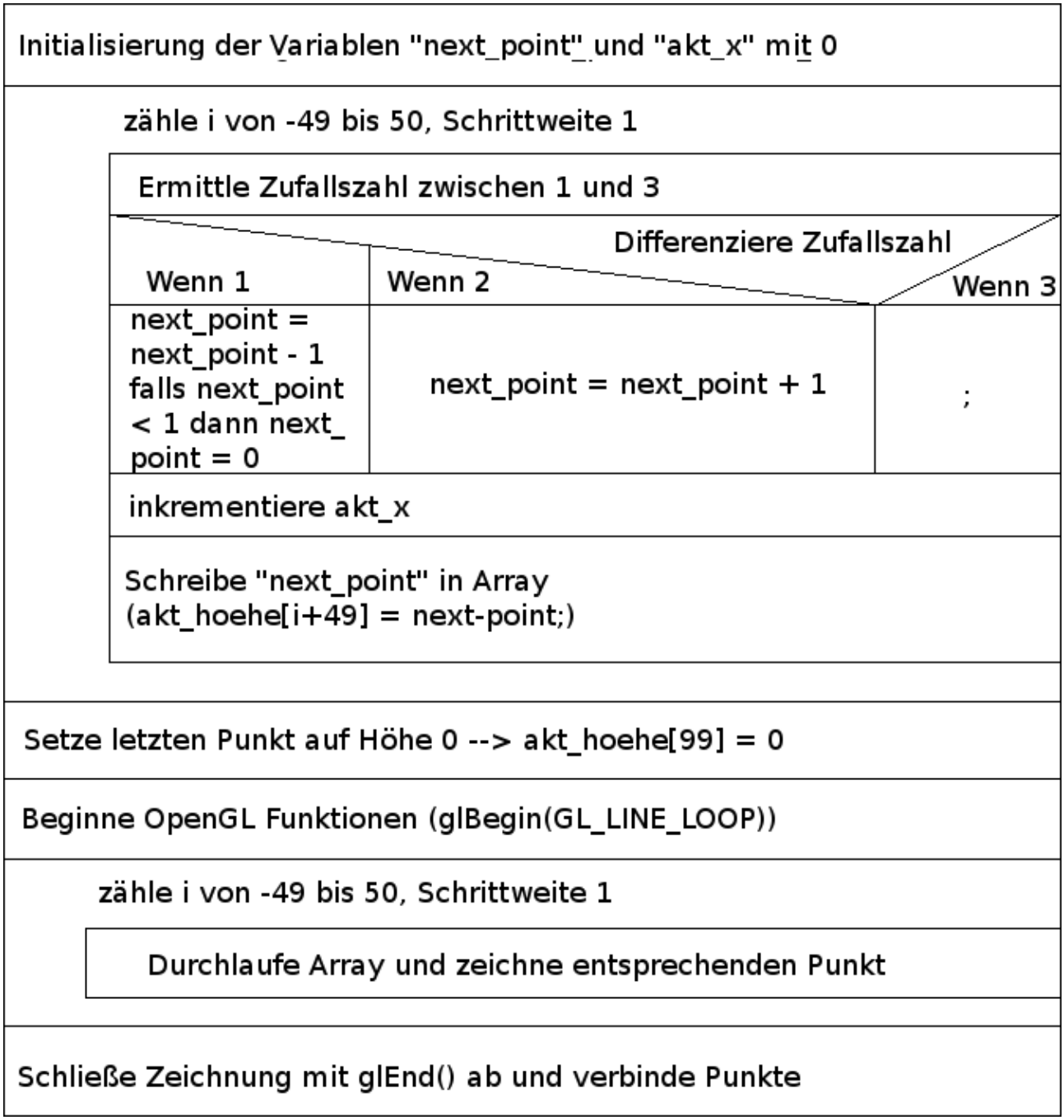
```
        else glColor3f(0.0f,1.0f,0.0f);

        textausgabe(0.07f, 1.0f, "F2    -   Zeige alle Gebirgsketten der Laenge 6", GLUT_BITMAP_HELVETICA_12);

        glColor3f(1.0f, 1.0f, 1.0f);

        textausgabe(0.07f, 0.5f, "ESC  -   Programm beenden", GLUT_BITMAP_HELVETICA_12);

        glColor3f(1.0f, 0.81f, 0.06f);
        textausgabe(0.9f, 0.5f, "FPS:", GLUT_BITMAP_HELVETICA_18);
        textausgabe(1.0f, 0.5f, strFrameRate, GLUT_BITMAP_HELVETICA_18);

        glutSwapBuffers ();
}

void subReshape (int w, int h){                                 // Reshape falls Fenstergröße von  verändert wird

        glViewport (0, 0, w, h);
        glMatrixMode (GL_PROJECTION);
        glLoadIdentity ();
        gluOrtho2D (0.0F, 1.0F, 0.0F, 1.0F);
}

void cfps(){                                                    // Berechnet die FPS

        static float framesPerSecond    = 0.0f;                 // Speichere die FPS
        static long lastTime            = 0;                    // Zeit nach dem letzten Frame
        struct timeval currentTime;                             // Struktur für Zeitberechnung
        currentTime.tv_sec  = 0;
        currentTime.tv_usec = 0;

        gettimeofday(&currentTime, NULL);                       // Millisekunden seit Programmstart

        ++framesPerSecond;                                      // Inkrementiere Frame - Counter

        if( currentTime.tv_sec - lastTime >= 1 ){               // Wenn eine Sekunde vergangen ist...
                lastTime = currentTime.tv_sec;

                 sprintf(strFrameRate, "%d", int(framesPerSecond));         // Kopiere FPS nach strFrameRate

                framesPerSecond = 0;                                        // Setze FPS zurück
        }
}

void zeichne_ks(int laenge,int hoehe, float pos_x, float pos_y){        // Zeichnet ein Koordinatensystem ohne Beschriftung

        glLoadIdentity();
        glTranslatef(pos_x,pos_y,-75.0f);

        glColor3f(0.035f,0.14f,0.047f);

        glBegin(GL_QUADS);                                      // Zeichne "Backframe"
                glVertex2i(-laenge / 2 ,0);
                glVertex2i(laenge / 2,0);
                glVertex2i(laenge / 2, hoehe);
                glVertex2i(-laenge / 2, hoehe);
        glEnd();

                for(int x = -(laenge / 2); x <= (laenge / 2); x++){     // Zeichne Karos
                                glColor3f(0.4f,0.4f,0.4f);
                                glBegin(GL_LINES);
                                        glVertex2i(x,0);
                                        glVertex2i(x,hoehe);
                                glEnd();
                }

                for(int y = 0; y <= hoehe; y++){                        // Zeichne Karos

                                glColor3f(0.4f,0.4f,0.4f);
                                glBegin(GL_LINES);
                                        glVertex2i(-(laenge / 2),y);
                                        glVertex2i(laenge / 2,y);
                                glEnd();
```

```
                }
}

void zeichne_gebirgszug(int g_punkt[7]){                    // Diese Funktion zeichnet einen Gebirgszug

        glLoadIdentity();
        glTranslatef(-50.0f + x_offset,10.0f - y_offset,-75.0f);

        glColor3f(1.0f,1.0f,1.0f);

        glBegin(GL_LINE_LOOP);
                glVertex2i(0,g_punkt[0]);
                for(int i = 0; i <= 6; i++) glVertex2i(i,g_punkt[i]);     // Lese Array aus und zeichne den Punkt
        glEnd();
}
```

Anlage A –Teilprobleme und deren Lösungsideen

Teilproblem	Lösungsidee
Welche Programmiersprache / Framework ?	C++ und OpenGL
Gebirgszug der Länge 100 zeichnen	Zufallsbasierendes System.
Prozedur schreiben die einen Gebirgszug zeichnet	Benutzen von OpenGL-spezifischen Funktionen und Benutzen eines Arrays vom Typ Integer
Alle Gebirgszüge der Länge 6 zeichnen	Rekursion, Backtracking, Brute-Force mit anschließender Filterung
Alle Gebirgszüge der Länge 16 zählen	Rekursion, Backtracking, Brute-Force mit anschließender Filterung

Anlage B – Programmablaufplan: Ausgabe eines Gebirges der Länge 100



Anlage C –Funktionen und deren Wirkungsweise

Funktion	Wirkung
void ReSizeGLScene(int Width, int Height)	Wird aufgerufen wenn das Fenster in seiner Größe verändert wird
void InitGL(int Width, int Height)	Initialisiert OpenGL
void DrawGLScene	Die Hauptfunktion für das Zeichnen auf dem Monitor
void draw_mountains_100()	Zeichnet Gebirgskette der Länge 100
void keyPressed(unsigned char key, int x, int y)	Wird aufgerufen wenn eine Taste gedrückt wird
void func_key(int key, int x, int y);	Wird aufgerufen wenn eine Funktionstaste gedrückt wird
void textausgabe (float x, float y, const char *text, void *font)	Selbstgeschriebene Funktion zum Ausgeben von Text
void idle()	Solange nichts passiert führe DrawGLScene aus
void subReshape (int w, int h)	Wird aufgerufen wenn das Subwindow in seiner Größe verändert wird
void cfps()	Berechnet die FPS und gibt sie aus
void zeichne_ks(int laenge,int hoehe, float pos_x, float pos_y)	Zeichnet ein Koordinatensystem
void zeichne_gebirgszug(int g_punkt[x])	Zeichnet einen Gebirgszug der Länge x

Anlage D –Wichtige Variablen und deren Funktion

Variable	Funktion
Int res_x, res_y	Festlegung der Auflösung
Int next_point	Die Höhe des nächsten Punkts
Int akt_hoehe[x]	Integer Array in dessen Feldern Höhenwerte stehen
Int rand_next_point	Zufallswert für den Höhenwert des nächsten Punkts
Bool draw_100	Wenn true wid ein Gebirgszug der Länge 100 gezeichnet
Bool draw_all_6	Wenn true werden alle Gebirgsketten der Länge 6 ausgegeben
Bool zeichne_zug	Wird benutzt um falsche Gebirgsketten auszufiltern