Technische Universität Berlin
Fakultät III – Prozesswissenschaften
Institut für Technischen Umweltschutz
Umweltverfahrenstechnik

Anhang II zur Diplomarbeit Entwicklung und Anwendung eines rechnergestützten Modells zur Ursachenanalyse großräumiger Luftschadstofftransporte - frequency.c

Diplomarbeit
im Studiengang Technischer Umweltschutz

vorgelegt von
Roman Finkelnburg

Wissenschaftliche Leitung: Prof. Dr.-Ing. Sven-Uwe Geißen
Wissenschaftliche Betreuung: Dr.-Ing. Markus Pesch

Berlin, November 2007

```
/* frequency.c */

/* Copyright (c) 2007 Roman Finkelnburg
 *
 * Permission to use, copy, modify, and distribute this software for ay_field
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 */

/* Dieses Programm errechnet mit Hilfe von Trajektoriendateien
 * Trajektoriendichten und gibt die Verlaeufe der fuer die Berechnung
 * verwendeten Trajektorien und die errechneten Trajektoriendichten in einer
 * KML-Datei aus. Diese KML-Datei ist unter Google-Earth darstellbar.
 *
 * *******
 * *INPUT*
 * *******
 * Als Inputdateien werden Trajektoriendateien verwendet, die dem vom
 * Programm clctrj.c produzierten Format entsprechen. Diese Dateien werden
 * aus dem zum Programmstart angegebenen Verzeichnis eingelesen. Das
 * Verzeichnis darf ausschliesslich Trajektoriendateien enthalten. Das
 * Vorhandensein andere Dateientypen im Verzeichnis fuehrt zum Abbruch des
 * Programms. Es werden alle enthaltenen Trajektoriendateien fuer die
 * Dichteberechnung verwendet. Man muss somit die zur Trajektoriendichte zu
 * verwendenden Trajektoriendatei in ein Verzeichnis kopieren/schieben und
 * den Verzeichnisnamen beim Programmstart uebergeben.
 *
 * ********
 * *OUTPUT*
 * ********
 * Die Ausgabedatei ist eine KML-Datei mit dem zum Programmstart angegebenen
 * Ausgabedateinamen.
```

```
*
* *******
* *START*
* *******
* Zum Programmstart koennen in der Programmumgebung (Shell, MS-DOS)
* spezielle Umgebungsvariablen gesetzt werden. Ueber diese
* Umgebungsvariablen werden dem Programm die gewuenschten Start- und
* Ausfuehrungsparameter uebergeben. Ist eine Umgebungsvariable nicht
* gesetzt, so wird fuer diesen Parameter der programminterne Standardwert
* verwendet.
*
* Dem Programm koennen die folgenden Parameter uebergeben werden:
*
* PARAMETER                              UMGEBUNGSVARIABLE STANDARDWERT
* ---------                              ----------------- ------------
* Name der Ausgabedatei                  FILENAME          freq.kml
*
* Verzeichnisname der Trajektoriendateien  INPUTDIR        traj/
*
* Groesse der Berechnungsnetzmaschen (km)  RES             25
*
* Zu verwendende Wichtung                WEIGHT            0
*  (0: absolute Haeufigkeit,
*   1: Wichtung ueber Abstand zum
*      Startpunkt,
*   2: Wichtung ueber Wurzel des Abstands
*      zum Startpunkt)
*
* Maximum der Skalierung (%)             SCALEMAX          100
*
* Minimum der Skalierung (%)             SCALEMIN          0
*
* Transparenz der Farben (00-ff)         OPACITY           88
*
* Offset zum Kartennullpunkt Laengengrad   OFFLO           0.0
* (Grad)
*
* Offset zum Kartennullpunkt Breitengrad   OFFLA           0.0
* (Grad)
*
```

```
 * Modus der Farbdarstellung                COLOR            0
 *  (0: alle Farben, 1: Dunkelblau,
 *   2: Blau, 3: Hellblau, 4: Minz,
 *   5: Gruen, 6: Hellgruen, 7: Gelb,
 *   8: Orange, 9: Rot, 10: Pink)
 *
 * Groesse des Plotbereichs (Seitenlaenge   SIZE             0
 * als x * RES) (0: alle)
 *
 * Mittelpunkt des Plotbereichs             MIDLO            13.4167
 * Laengengrad (Grad)
 *
 * Mittelpunkt des Plotbereichs             MIDLA            52.5167
 * Breitengrad (Grad)
 */

/*
 * PROGRAMMSTRUKTUR
 *
 * main()
 * .      read_env()
 * .      init_values()
 * .      .      init_colorclasses()
 * .      .      count_files()
 * .      .      get_lo_la_min_max()
 * .      .      .      read_dir()
 * .      .      .      file_lo_la_min_max()
 * .      .      count_squares()
 * .      .      init_field()
 * .      print_header()
 * .      print_colorstyles()
 * .      plot()
 * .      .      init_plot_area()
 * .      .      plot_trajectories()
 * .      .      .      get_trajectory_start_point()
 * .      .      .      .      read_header()
 * .      .      .      print_trajectory_header()
 * .      .      .      reset_counter()
 * .      .      .      init_field()
 * .      .      .      read_trajectory()
```

```
 * .      .      .      .      plot_to_next_point()
 * .      .      .      .      .      get_weight()
 * .      .      .      .      .      .      convert_geo_to_cartesian()
 * .      .      .      .      .      .      distance_in_deg()
 * .      .      .      .      .      rotate_points()
 * .      .      .      .      .      check_plot_area()
 * .      .      plot_frequency()
 * .      .      .      get_w_max()
 * .      .      .      print_freq_header()
 * .      .      .      sort_and_plot()
 * .      .      .      .      plot_element()
 * .      print_end()
 * .      reset_state()
 */

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/types.h>
#include <dirent.h>
#include <string.h>

/***********
 * DEFINES *
 ***********/
#define DEGDISTANCE 111.178 /* Laenge eines Laengengrades in km */
#define MAXLINE     256     /* Zeichenanzahl der verwendeten Linebuffer */
#define MAXCOLORCLASS 10    /* Anzahl Farbklassen in der Ausgabe */
#define HEADERLINES 7       /* Anzahl der Zeilen, die in den Trajektorien-
                             * dateien fuer allgemeine Informationen (Kopf)
                             * vor Beginn den Koordinatenangaben verwendet
                             * wurden
                             */
#define SEPARATOR   ';'     /* In den Trajektoriendateien verwendetes
                             * Trennzeichen zwischen den Koordinaten
                             */

/****************
 * DECLARATIONS *
```

```
 ***************/

/* Startparameter */
struct param {
        char* name;
        enum {
                TYP_INT,
                TYP_FLOAT,
                TYP_STRING
        } type;
        union {
                char* s;
                int i;
                double f;
        } u;
        char* desc;
} param[] = {
        {"FILENAME", TYP_STRING, { "freq.kml" },
         "name of output file"},

        {"INPUTDIR", TYP_STRING, { "traj/" },
         "directory of input data"},

        {"RES",      TYP_INT,    { "25" },
         "resolution [km]"},

        {"SCALEMIN", TYP_INT,    { "0" },
         "minimum of scale [%%]"},

        {"SCALEMAX", TYP_INT,    { "100" },
         "maximum of scale [%%]"},

        {"OPACITY",  TYP_STRING, { "88" },
         "opacity of color [hex]"},

        {"OFFLO",    TYP_FLOAT,  { "0.0" },
         "offset of longitude in plot [degree]"},

        {"OFFLA",    TYP_FLOAT,  { "0.0" },
         "offset of latitude in plot [degree]"},
```

```
        {"COLOR",    TYP_INT,    { "0" },
         "modus coloring (1-10, 0: all)"},

        {"WEIGHT",   TYP_INT,    { "0" },
         "weightning modus (0, 1, 2)"},

        {"SIZE",     TYP_INT,    { "0" },
         "side length of plot in RES-elements (0: all)"},

        {"MIDLO",    TYP_FLOAT,  { "13.4167" },
         "midpoint longitude of plot area [degree]"},

        {"MIDLA",    TYP_FLOAT,  { "52.5167" },
         "midpoint latitude of plot area [degree]"},

        {NULL,           0,      { NULL }, NULL }
};

/* NB: selbe Reihenfolge wie im 'param' Array! */
enum {
        FILENAME = 0,
        INPUTDIR = 1,
        RES      = 2,
        SCALEMIN = 3,
        SCALEMAX = 4,
        OPACITY  = 5,
        OFFLO    = 6,
        OFFLA    = 7,
        COLOR    = 8,
        WEIGHT   = 9,
        SIZE     = 10,
        MIDLO    = 11,
        MIDLA    = 12
};

/* Struktur zur Speicherung des momentanen Programmstatus */
struct state {

        int list_max, field_max; /* Elementanzahl der Datenstrukturen list
```

```
                                * und field_grid
                                */
        int x_field, y_field;   /* Anzahl der Netzelemente in x- und
                                * y-Richtung
                                */
        FILE* fh;               /* Filehandle */
        char** list;            /* Liste der Namen von einzulesenen Dateien
                                */
        double* field_grid;     /* Berechnungsfeld zum aufaddieren der
                                * Feldelementwerte
                                */
        double lo_min;          /* kleinster Laengengradwert, der in den
                                * Trajektorien vorkommt
                                */
        double la_min;          /* kleinster Breitengradwert, der in den
                                * Trajektorien vorkommt
                                */
        double lo_max;          /* groesster Laengengradwert, der in den
                                * Trajektorien vorkommt
                                */
        double la_max;          /* groesster Breitengradwert, der in den
                                * Trajektorien vorkommt
                                */
        int colorclass[MAXCOLORCLASS]; /* Farbskala Map */
};

/* Struktur zur Speicherung des momentanen Lese-/Berechnungsstatus
 * (eine einzelne Trajektorie) */
struct trajectory {
        char*   name;
        FILE*   fh;
        double  x_new, y_new;
        double  x_begin, y_begin; /* Startposition der aktuellen
                                  * Trajektorie
                                  */
        double  x_old, y_old; /* vorangegagender Aufpunkt */
        double  x_midpoint, y_midpoint; /* Mittelpunkt zwischen den letzten
                                        * beiden Aufpunkten
                                        */
        int     x_plot_min, y_plot_min, x_plot_max, y_plot_max;
```

```
        int      plot_size;
};

/* zu verwendende Farben */
char* CLASS[MAXCOLORCLASS] = {
        "ff0000", /* dark blue */
        "ff8800", /* blue */
        "ffff00", /* light blue */
        "88ff00", /* mint */
        "00ff00", /* green */
        "00ff88", /* light green */
        "00ffff", /* yellow */
        "0088ff", /* orange */
        "0000ff", /* red */
        "8800ff"  /* pink */
};

/*
 * Makros zum Auslesen der verschiedenen Datentypen (int, float, string) aus
 * der Programmstartparameterstruktur (struct param)
 */
#define get_int(p)     (assert(param[(p)].type == TYP_INT), param[(p)].u.i)
#define get_float(p)   (assert(param[(p)].type == TYP_FLOAT), param[(p)].u.f)
#define get_string(p)  (assert(param[(p)].type == TYP_STRING), param[(p)].u.s)

#define rad2deg(f)        ((f) * 180 / M_PI)
#define deg2rad(f)        ((f) * M_PI / 180)

#define rad2idx(f, dy)        ((unsigned)((f) / (dy))) /* XXX umbennenen oder so */

/**************
 * PROTOTYPES *
 **************/
int    check_plot_area(double, double, struct trajectory *);
void   convert_geo_to_cartesian(double, double, double*);
int    count_files(void);
void   count_squares(struct state*);
double distance_in_deg(double*, double*);
void   file_la_lo_min_max(char*, double*, double*, double*, double*);
void   get_lo_la_min_max(struct state*);
```

```
void   get_trajectory_start_point(struct trajectory*);
double get_w_max(struct state*);
double get_weight(struct trajectory*);
void   init_colorclasses(struct state*);
void   init_field(double*, int);
void   init_plot_area(struct state*, struct trajectory*);
void   init_values(struct state*);
void   plot(struct state*);
void   plot_element(FILE*, double, double, int);
void   plot_frequency(struct state*);
void   plot_to_next_point(double*, struct trajectory*, struct state*);
void   plot_trajectories(struct state*, struct trajectory*);
void   print_colorstyles(struct state*);
void   print_end(struct state*);
void   print_freq_header(struct state*, double, double, double);
void   print_header(struct state*);
void   print_trajectory_header(FILE*, double, double);
void   read_dir(char*, char**, int);
void   read_env(struct param*);
void   read_header(FILE*, int, char*);
void   read_trajectory(struct trajectory*, struct state*, double*);
void   reset_counter(struct trajectory*);
void   reset_state(struct state*);
void   rotate_points(double*, double*, double*, double*);
void   sort_and_plot(struct state*, int, double, double, double);

/****************
 * MAINFUNCTION *
 ****************/
int
main(void) {

        struct state state;

        /* Einlesen der uebergebenen Argumente */
        read_env(param);

        /*
         * Initialisieren der Datenstruktur zum Abbilden des programminternen
         * Berechnungsstatus
```

```
         */
        init_values(&state);

        /* Schreiben des KML-Headers */
        print_header(&state);

        /* Schreiben der zu verwendenden Farbkonfigurationen */
        print_colorstyles(&state);

        /* Plotten der einzelnen Netzelemente (Dichtedarstellung) und
         * verwendeten Trajketorien (unsichtbar)
         */
        plot(&state);

        /* Schliessen der KML-Strukturen */
        print_end(&state);

        /* Reservierte Speicherbereiche wieder freigeben */
        reset_state(&state);

        return 0;
}

/***************
 * SUBROUTINES *
 ***************/

/* Ueberpruefen ob angegebenes Netzelement (x, y) innerhalb des definierten
 * Plotbereits liegt.
 */
int
check_plot_area(double x, double y, struct trajectory *current) {

        return x >= (double)current->x_plot_min &&
            x <= (double)current->x_plot_max &&
            y >= (double)current->y_plot_min &&
            y <= (double)current->y_plot_max;
}

/* Umrechnen der geographischen Positionsangabe in Rad (longitude,
```

```
 * latitude) in einen Ortsvektor in Kugelkoordinaten (X)
 */
void
convert_geo_to_cartesian(double longitude, double latitude, double X[])
{
        X[0] = cos(latitude) * cos(longitude);
        X[1] = cos(latitude) * sin(longitude);
        X[2] = sin(latitude);
}

/* Zaehlen der im Verzeichnis mit dem uebergebenen Verzeichnisnamen
 * enthaltenen Dateien.
 * Rueckgabewert ist die Anzahl der enthaltenen Dateien
 */
int
count_files(void) {

        DIR *dirHandle;
        struct dirent* dirEntry;
        int i = 0;

        /* Oeffnen des Verzeichnisses */
        dirHandle = opendir(get_string(INPUTDIR));

        /* Wenn Verzeichnis existiert */
        if (dirHandle) {
                /* Dateinamen auslesen bis Ende erreicht ist */
                while (0 != (dirEntry = readdir(dirHandle))) {

                        /* Versteckte Dateien nicht mitzaehlen */
                        if (dirEntry->d_name[0] != '.')
                                i++;
                }

                /* Verzeichnis schliessen */
                closedir(dirHandle);
        }
        else {
                printf("Directory %s not found!\n", get_string(INPUTDIR));
                exit (1);
```

```
        }

        return i;
}

/* Errechnen der benoetigten Netzelemente in x- (x_field) und
 * y-Richtung (y_field) fuer die angegebene Aufloesung
 * (Kaestchengroesse (res)) fuer das gesamte Berechnungsgebiet
 * (lo_min, la_min, lo_max, la_max)
 */
void
count_squares(struct state* state) {

        int i;
        double x, y, dx, dy;

        /* Uebrenehmen des Anfangsbreitengrades */
        y = state->la_min;

        /* Abstand eines Netzelements in Breitengradrichtung (konstant)*/
        dy = get_int(RES) / DEGDISTANCE;

        /* Initialiesieren */
        state->x_field = state->y_field = 0;

        /* Solonge, bis Breitengradmaximum erreicht */
        while (y <= state->la_max) {

                /* Umrechnen der Netzelementbreite in km (RES)
                 * in Rad (dx) fuer den momentanen Breitengrad (y)
                 */
                dx = get_int(RES) / (DEGDISTANCE * cos(deg2rad(y)));

                /* Zurueckgehen an den Anfang der neuen Zeile */
                x = state->lo_min;
                i = 0;

                /* Solange, bis Zeilenende erreicht ist*/
                while (x <= state->lo_max) {
```

```
                        x = x + dx;
                        i++;
                }

                /* Wenn Netzelementanzahl in Laengengradrichtung groesser als
                 * vorherige -> speichern der Groesseren Anzahl
                 */
                if (i > state->x_field) {

                        state->x_field = i;
                }

                /* Netzelementanzahl in Breitengradrichtung um 1 Reihe
                 * erhoehen
                 */
                y = y + dy;
                state->y_field += 1;
        }
}

/* Ueber das Skalarprodukts von zwei Ortsvektoren wird der Winkel
 * zwischen ihnen errechnet. Fuer dieses Programm liegen alle Punkte
 * (Ortsvektoren) auf einer Kugeloberflaeche mit dem Radius 1
 * (programminterne Berechnungsystem behandelt die Erdkugel als
 * Einheitskugel) -> Der Betrag aller Ortsvekoren (Kugelmittelpunkt
 * bis Kugeloberflaeche) ist 1
 * Rueckgabewert ist der Winkels (Grad) zwischen den beiden
 * Ortsvektoren
 */
double
distance_in_deg(double X1[], double X2[]) {
        return rad2deg(acos(X1[0] * X2[0] +
                            X1[1] * X2[1] +
                            X1[2] * X2[2]));
}

/* Auslesen der Minimal- und Maximalwerte fuer Laengengrad und
 * Breitengrad aus der aktuellen Trajektoriendatei
 * Rueckgabe der Werte in den Variablen lo_min, la_min,
 * lo_max und la_max
```

```
 */
void
file_la_lo_min_max(char* name, double* lo_min, double* la_min,
              double* lo_max, double* la_max) {

        FILE* fh;
        int i;
        double lo, la;
        int ch;
        char buf[MAXLINE];
        char *rest;

        /* Wenn Datei nicht existiert */
        if (!(fh = fopen(name, "r"))) {
                printf("File %s doesn't exist!\n", name);
                exit(1);
        }

        /* Header weglesen */
        for (i = 0; i < HEADERLINES; i++) {

                while ((ch = fgetc(fh)) != '\n');
        }

        /* Nach Minimal- und Maximalwert der Laengengrade und Breitengrade
         * suchen
         */
        i = 0;

        /* Solange Dateiende nicht erreicht */
        while ((ch = fgetc(fh)) != EOF) {

                /* Wenn Zeile zu lang */
                if (i >= (MAXLINE - 1)) {

                        printf("Error: linebuffer too small!\n");
                        exit(1);
                }

                if ((ch != SEPARATOR) && (ch != '\n')) {
```

```
            buf[i++] = (int)ch;
            buf[i] = '\0';
        }
        else if (ch == SEPARATOR) {

            lo = strtod(buf, &rest);

            /* Wenn Syntaxfehler fuer Laengengradelement in der
             * Datei
             */
            if (strlen(rest) != 0) {

                printf("Syntactic failure in %s!\n", name);
                printf("Value for longitude = %s\n", buf);
                exit(1);
            }

            i = 0;
        }
        else if (ch == '\n') {

            la = strtod(buf, &rest);

            /* Wenn Syntaxfehler fuer Breitengradelement in der
             * Datei
             */
            if (strlen(rest) != 0) {

                printf("Syntactic failure in %s!\n", name);
                printf("Value for latitude = %s\n", buf);
                exit(1);
            }

            i = 0;

            if (lo < *lo_min) { /* Wenn aktueller Laengengrad
                                 * kleiner als bisheriges Minimum
                                 */
                *lo_min = lo;
```

```
                        }
                        if (lo > *lo_max) { /* Wenn aktueller Laengengrad
                                             * groesser als bisheriges
                                             * Maximum
                                             */
                                *lo_max = lo;
                        }
                        if (la < *la_min) { /* Wenn aktueller Breitengrad
                                             * kleiner als bisheriges
                                             * Minimum
                                             */
                                *la_min = la;
                        }
                        if (la > *la_max) { /* Wenn aktueller Breitengrad
                                             * groesser als bisheriges
                                             * Maximum
                                             */
                                *la_max = la;
                        }
                }
        }

        /* Schliessen der Datei */
        fclose(fh);
}

/* Auslesen der Minimal- und Maximalwerte der in den Trajektoriendateien
 * enthaltenen Aufpunkte in Laengen- und Breitengrad (lo_min, la_min,
 * lo_max, la_max)
 */
void
get_lo_la_min_max(struct state* state) {

        int i;

        /* Initialisieren */
        state->lo_min = 2;
        state->la_min = 2;
        state->lo_max = -2;
        state->la_max = -2;
```

```
        /* Auslesen der Dateinamen der im angegebenen Verzeichnis enthaltenen
         * Dateien und Speicherung in einer Dateinamenliste (list)
         */
        read_dir(get_string(INPUTDIR), state->list, state->list_max);

        /* Durchsuchen der in der Liste enthaltenen Trajektoriendateien nach
         * den groessten und kleinsten Koordinaten (lo_min, la_min, lo_max,
         * la_max)
         */
        for (i = 0; i < state->list_max; i++)
                file_la_lo_min_max(state->list[i], &state->lo_min,
                    &state->la_min, &state->lo_max, &state->la_max);

        /* Umrechnen der Minimal- und Maximalkoordinaten von Rad in Grad */
        state->lo_min = rad2deg(state->lo_min);
        state->la_min = rad2deg(state->la_min);
        state->lo_max = rad2deg(state->lo_max);
        state->la_max = rad2deg(state->la_max);
}

/* Weglesen des Trajektoriendatei-Headers und einlesen des ersten
 * Trajektorienpunkts
 */
void
get_trajectory_start_point(struct trajectory* current) {
        char buf[MAXLINE];
        int  ch;
        int  j;

        /* Wenn Trajektoriendatei nicht geoeffjet werden kann */
        if (!(current->fh = fopen(current->name, "r"))) {

                printf("Can't open file %s for reading!\n", current->name);
                exit(1);
        }

        /* Header der Trajektoriendatei weglesen */
        read_header(current->fh, HEADERLINES, current->name);
```

```
/* Einlesen des Startpunktes */
j = 0;

/* Wenn Dateiende erreicht */
if ((ch = fgetc(current->fh)) == EOF) {

        printf("End of file %s!\n", current->name);
        exit(1);
}

/* Bis Zeilenende erreicht ist, bis Startposition (Laengengrad
 * und Breitengrad) eingelesen sind. Die separate Speicherung
 * wird fuer die Wichtung ueber den Abstand zum Startpunkt benoetigt
 */
while (ch != '\n') {

        /* Wenn Trennzeichen zwischen Laengen- und Breitengradangabe
         * noch nicht erreicht */
        if (ch != SEPARATOR) {

                buf[j++] = (char)ch;
                buf[j] = '\0';
        }
        else { /* Wenn Trennzeichen erreicht */

                /* Speichern der Startlongitude */
                current->x_begin = atof(buf);
                j = 0;
        }

        /* Wenn Dateiende erreicht */
        if ((ch = fgetc(current->fh)) == EOF) {

                printf("End of file %s!\n", current->name);
                exit(1);
        }
}

/* Speichern der Startlatitude */
current->y_begin = atof(buf);
```

```
        /* Koordinaten in Grad umrechnen */
        current->x_begin = rad2deg(current->x_begin);
        current->y_begin = rad2deg(current->y_begin);
}

/* Auslesen des maximalen Dichtewertes w_max */
double
get_w_max(struct state* state) {

        double w_max;
        int i;

        w_max = 0;

        for (i = 0; i < state->field_max; i++)
                if (w_max < state->field_grid[i])
                        w_max = state->field_grid[i];
        return w_max;
}

/* Berechnen des Wichtungsfaktors fuer die momentane Position
 * Rueckgabewert ist der Wichtungsfaktor
 */
double
get_weight(struct trajectory* current)
{
        double X1[3];
        double X2[3];
        double distance;

        distance = 0;

        /* Errechne des Abstands (d) von Mittelpunkt zwischen den beiden
         * momentanen Aufpunkten zum Trajektorienstartpunkt
         */
        convert_geo_to_cartesian(deg2rad(current->x_midpoint),
            deg2rad(current->y_midpoint), X1);
        convert_geo_to_cartesian(deg2rad(current->x_begin),
            deg2rad(current->y_begin), X2);
```

```
        distance = distance_in_deg(X1, X2);

        /* Errechnen des gewuenschten Modus (weight) des
         * Wichtungsfaktors (w)
         */
        switch (get_int(WEIGHT)) {
        case 1: /* Wichtung ueber Abstand zum Startpunkt */
                return distance;
        case 2: /* Wichtung ueber Wurzel des Abstandes zum Startpunkt */
                return sqrt(distance);
        default: /* keine Wichtung (absolute Haeufigkeit) */
                return 1.0;
        }
}

/* Initialisieren der fuer die Legende/Netzelemente zu
 * verwendeten Farben
 */
void
init_colorclasses(struct state* state) {

        int i;

        if (get_int(COLOR) == 0) {
                for (i = 0; i < MAXCOLORCLASS; i++)
                        state->colorclass[i] = i;
        } else if (get_int(COLOR) > 0 && get_int(COLOR) <= MAXCOLORCLASS) {
                for (i = 0; i < MAXCOLORCLASS; i++)
                        state->colorclass[i] = get_int(COLOR) - 1;
        } else {
                printf("Invalid color value %d (too large)!\n",
                    get_int(COLOR));
                exit(1);
        }
}

/* Ausnullen der Speicherstruktur (field_grid) fuer die Speicherung der
   Trajektoriendichten */
void
init_field(double* field_grid, int field_max) {
```

```
        int i;

        for (i = 0; i < field_max; i++) {

                field_grid[i] = 0.0;
        }
}

/* Errechnen des zu plottenden Teilbereichs des gesamten Berechnungsgebiets */
void
init_plot_area(struct state* state, struct trajectory* current) {

        double x, y, dx, dy;
        unsigned col;
        unsigned row;

        /* Errechnen der Grenzen des Plotbereichs */
        if (current->plot_size > 0) {
                /* Rangechecking des Plotmittelpunkts */
                if ((get_float(MIDLA) < state->la_min) ||
                    (get_float(MIDLA) > state->la_max) ||
                    (get_float(MIDLO) < state->lo_min) ||
                    (get_float(MIDLO) > state->lo_max)) {

                        printf("Error: midpoint of plot out of range ");
                        printf("(%5.2f %5.2f | %5.2f %5.2f)\n",
                            state->lo_min, state->la_min, state->lo_max,
                            state->la_max);
                        exit(1);
                }

                /* Wenn Elementanzahl ungerade -> mache gerade */
                if ((current->plot_size % 2) == 1)
                        current->plot_size++;

                y = state->la_min;
                row = 0;

                /* Errechnen der Netzelementbreite in y-Richtung
                 * (konstant) */
```

```
dy = get_int(RES) / DEGDISTANCE;

/* Zeilenindex row und Breitengrad y der
 * aktuellen Mittelpunktposition berechnen
 */
if (get_float(MIDLA) > y) {
        row = rad2idx(get_float(MIDLA) - y, dy);
        y += row * dy;
}

/* Gehe zum ersten Netzelement in x-Richtung */
x = state->lo_min;
col = 0;

/* Errechnen der Netzelementbreite in x-Richtung (abhaengig
 * vom Breitengrad
 */
dx = get_int(RES) / (DEGDISTANCE * cos(deg2rad(y)));

/* Spaltenindex col und Laengengrad x der
 * aktuellen Mittelpunktposition berechnen
 */
if (get_float(MIDLO) > x) {
        col = rad2idx(get_float(MIDLO) - x, dx);
        x += col * dx;
}

/* Da jeweils ein Schritt zu weit gegangen wird -> zurueck
 * auf Netzelement in dem aktueller Mittelpunkt liegt
 */
col--;
row--;

/* Speichern der Grenzen des Plotbereichs */
current->x_plot_min = col - current->plot_size / 2;
current->x_plot_max = col + current->plot_size / 2;
current->y_plot_min = row - current->plot_size / 2;
current->y_plot_max = row + current->plot_size / 2;

/* Rangechecking des Plotbereichs */
```

```
                if (current->x_plot_min < 0) {

                        current->x_plot_min = 0;
                }
                if (current->x_plot_max > state->x_field) {

                        current->x_plot_max = state->x_field;
                }
                if (current->y_plot_min < 0) {

                        current->y_plot_min = 0;
                }
                if (current->y_plot_max > state->y_field) {

                        current->y_plot_max = state->y_field;
                }
                printf("%i %i %i %i | %i %i\n",
                    current->x_plot_min, current->y_plot_min,
                    current->x_plot_max, current->y_plot_max,
                    state->x_field, state->y_field);
        }
        else { /* Wenn keine Plotbereich angegeben, dann plotte alles */

                current->x_plot_min = 0;
                current->x_plot_max = state->x_field;
                current->y_plot_min = 0;
                current->y_plot_max = state->y_field;
        }
}

/*
 * Initialisieren der Datenstruktur zum Abbilden des programminternen
 * Berechnungsstatus
 */
void
init_values(struct state* state)
{
        memset(state, 0, sizeof(struct state));

        /* Initialisieren der Farbskala */
```

```
        init_colorclasses(state);

        /* Anzahl der Trajektorinedateien auslesen */
        state->list_max = count_files();

        /* Speicherplatz reservieren */
        state->list = (char**)calloc(sizeof(char*), state->list_max);

        /* Minimale und maximale Koordinatenwerte der in den fuer die
         * Berechnung zu verwendenden Trajektorienaufpunkte bestimmen
         */
        get_lo_la_min_max(state);

        /* Bestimmen der benoetigten Netzelementeanzahl (abhaengig von
         * Netzelementgroesse (Aufloesung) und der Groesse des
         * Berechnungsgebiets (lo_min, la_min, lo_max, la_max))
         */
        count_squares(state);

        /* Errechnen der Gesamtanzahl der Netzelemente */
        state->field_max = state->x_field * state->y_field;

        /* Reservieren des benoetigten Speicherplatzes fuer die
         * Trajektoriendichte-Speicherstruktur
         */
        state->field_grid = (double*)calloc(sizeof(double), state->field_max);

        /* Berechnungsnetz Initialisieren */
        init_field(state->field_grid, state->field_max);

        /* Kann gewuenschte Ausgabedatei angelegt werden? */
        if (!(state->fh = fopen(get_string(FILENAME), "w"))) {

                printf("Couldn't write file %s!\n", get_string(FILENAME));
                exit(1);
        }
}

/* Auslesen der einzelnen Trajektorienaufpunkte und errechnen der
 * Trajektoriendichte. Schreiben der Trajektorienaufpunkte
```

```
 * (Trajektoriendarstellung) und der Netzelementwerte (Trajektoriendichte-
 * darstellung) in die KML-Ausgabedatei
 */
void
plot(struct state* state) {


        struct trajectory current;

        current.plot_size = get_int(SIZE);

        /* Errechnen des zu plottenden Teilbereichs des gesamten
         * Berechnungsgebiets
         */
        init_plot_area(state, &current);

        /* Einlesen und plotten der Trajektorien */
        plot_trajectories(state, &current);

        /* Plotten der Trajektoriendichten */
        plot_frequency(state);
}

void
plot_element(FILE* fh, double x, double y, int color) {

        /* Schreiben eines Netzelements in die KML-Datei
         * mit der Position x, y, der Seiten Laenge Res, unter
         * Beruecksichtigung der Offsets off_lo und off_la zum eigentlichen
         * Koordinatennullpunkt, mit der Farbeinstellung color der
         * vorkunfigurierten Farben
         */
        double dy;  /* Seitenlaenge in y-Richtung */
        double dx1; /* untere Seitenlaenge in x-Richtung */
        double dx2; /* obere Seitenlaenge in x-Richtung */

        /* Wenn Netzelement sichtbar (Dichte > 0) */
        if (color > 0) {

                dy =  get_int(RES) / DEGDISTANCE;
```

```
                dx1 = get_int(RES) / (DEGDISTANCE * cos(deg2rad(y)));
                dx2 = get_int(RES) / (DEGDISTANCE * cos(deg2rad(y + dy)));

                fprintf(fh, "<Placemark>\n");
                fprintf(fh, "<styleUrl>#%i</styleUrl>\n", color);
                fprintf(fh, "<Polygon>\n");
                fprintf(fh,
                    "<altitudeMode>relativeToGround</altitudeMode>\n");
                fprintf(fh, "<outerBoundaryIs>\n");
                fprintf(fh, "<LinearRing>\n");
                fprintf(fh, "<coordinates>\n");
                fprintf(fh, "%10.6f,%10.6f,0\n", x + get_float(OFFLO), y + get_float(OFFLA))
                fprintf(fh, "%10.6f,%10.6f,0\n",
                    x + dx1 + get_float(OFFLO), y + get_float(OFFLA));
                fprintf(fh, "%10.6f,%10.6f,0\n",
                    x + dx2 + get_float(OFFLO), y + dy + get_float(OFFLA));
                fprintf(fh, "%10.6f,%10.6f,0\n",
                    x + get_float(OFFLO), y + dy + get_float(OFFLA));
                fprintf(fh, "%10.6f,%10.6f,0\n", x + get_float(OFFLO), y + get_float(OFFLA))
                fprintf(fh, "</coordinates>\n");
                fprintf(fh, "</LinearRing>\n");
                fprintf(fh, "</outerBoundaryIs>\n");
                fprintf(fh, "</Polygon>\n");
                fprintf(fh, "</Placemark>\n\n");
        }
}

/* Ausgeben der Trajektoriendichten in die KML-Datei */
void
plot_frequency(struct state* state) {

        double w_max, min, max, dw;
        int    i, k;

        /* Auslesen des maximalen Dichtewertes w_max */
        w_max = get_w_max(state);

        /* Rangecheching des Skalenminimums */
        if ((get_int(SCALEMIN) < 0) ||
            (get_int(SCALEMIN) >= get_int(SCALEMAX)) ||
```

```
    (get_int(SCALEMAX) > 100)) {
        printf("Scale ranges incorrect!\n");
        exit(1);
}

/* Errechnen der Dichtewerte des Skalenminimums
 * und -maximums
 */
min = w_max * (double)get_int(SCALEMIN) / 100.0;
max = w_max * (double)get_int(SCALEMAX) / 100.0;

/* Errechnen des Dichteunterschiedes dw
 * einer Klasse
 */
dw = (max - min) / 10;

/* Errechnen der Klassezugehoerigkeit
 * jedes einzelnen Elements
 */
for (i = 0; i < state->field_max; i++) {
        state->field_grid[i] =
            (state->field_grid[i] - min) / dw;

        if (state->field_grid[i] > 10)
                state->field_grid[i] = 10.0;
}

/* Header des Dichte-Unterordners in die
 * KML-Datei schreiben
 */
print_freq_header(state, w_max, min, max);

/* Schreiben Trajektoriendichten geordnet nach Klassen
 * in einzelnde Unterornder in der KML-Datei
 */
for (k = 0; k < 10; k++) {
        sort_and_plot(state, k + 1, dw, min, w_max);
}

/* Schliessen des Trajektoriendichteordners */
```

```
        fprintf(state->fh, "</Folder>\n\n");
}

/* Gewichtete Abbildung des Trajektorienverlaufs zwischen den momentan
 * betrachteten Aufpunkten (x_old, y_old und x_new, y_new) in einer
 * temporaeren Matrix 'plot_field'. Im Grunde zeichnen wir eine
 * Gerade zwischen dem alten und dem neuen Punkt im Plotbereich ;)
 */
void
plot_to_next_point(double* plot_field, struct trajectory* current,
    struct state* state) {

        double tmp, x1, y1, x2, y2, dx, dy, w, m, n;

        /* Wichtung des aktuellen Punktes berechnen */
        w = get_weight(current);

        /* Umrechnen der Netzelementhoehe in km (RES)
         * in die Netzelementhoehe in Rad (dy)
         */
        dy = get_int(RES) / DEGDISTANCE;

        /* Umrechnen der Netzelementbreite in km (RES)
         * in die Netzelementbreite in Rad (dx) fuer die
         * Breite des erten Aufpunkts (x_old, y_old)
         */
        dx = get_int(RES) / (DEGDISTANCE * cos(deg2rad(current->y_old)));

        /* Berechnen des Astands in x- und y-Richtung des ersten
         * Aufpunkts (x_old, y_old) zum Ursprung (lo_min, la_min) des
         * Berechnungsgebiets in der Einheit Netzelement
         */
        x1 = (current->x_old - state->lo_min) / dx;
        y1 = (current->y_old - state->la_min) / dy;

        /* Umrechnen der Netzelementbreite in km (RES)
         * in die Netzelementbreite in Rad (dx) fuer die
         * Breite des zweiten Aufpunkts (x_new, y_new)
         */
        dx = get_int(RES) / (DEGDISTANCE * cos(deg2rad(current->y_new)));
```

```
/* Berechnen des Abstands in x- und y-Richtung des zweiten
 * Aufpunkts (x_new, y_new) zum Ursprung (lo_min, la_min) des
 * Berechnungsgebiets in der Einheit Netzelement
 */
x2 = (current->x_new - state->lo_min) / dx;
y2 = (current->y_new - state->la_min) / dy;

if (y1 == y2) { /* Wenn Steigung 0 */

        /* x2 muss der groessere Wert sein */
        if (x2 < x1)
                /* Vertauschen der Punkte (x1,y1) und (x2,y2) */
                rotate_points(&x1, &y1, &x2, &y2);

        /* In x-Richtung den Netzelementebereich zwischen den
         * beiden Aufpunkten abwandern und den Trajektorienverlauf
         * (y=const.) plotten
         */
        while (x1 < x2) {
                /* Checken, ob Element im Plotbereich liegt */
                if (check_plot_area(x1, y1, current)) {
                        unsigned i = (int)y1 *
                            state->x_field + (int)x1;

                        if (plot_field[i] == 0)
                                plot_field[i] += w;
                }

                x1 += dx;
        }
}
else {
        /* y2 muss der groessere Werte sein */
        if (y2 < y1)
                /* Vertauschen der Punkte (x1,y1) und (x2,y2) */
                rotate_points(&x1, &y1, &x2, &y2);

        /* Geradengleichung zwischen den beiden Aufpunkten
         * berechnenen
```

```
                 */
                m = (x2 - x1) / (y2 - y1);
                n = x1 - (m * y1);

                /* In y-Richtung den Netzelementebereich zwischen den
                 * beiden Aufpunkten abwandern und den linear
                 * interpolierten Trajektorienverlauf (Geradengleichung)
                 * zwischen den beiden Aufpunkten plotten
                 */
                while (y1 < y2) {

                        tmp = (m * y1) + n;

                        /* Checken, ob Element im Plotbereich liegt */
                        if (check_plot_area(tmp, y1, current)) {
                                unsigned i = (int)y1 *
                                    state->x_field + (int)tmp;

                                if (plot_field[i] == 0)
                                        plot_field[i] += w;
                        }

                        y1 += dy / get_int(RES);
                }
        }
}

/* Einlesen und plotten der Trajektorien */
void
plot_trajectories(struct state* state, struct trajectory* current) {

        double *plot_field; /* Plotmatrix fuer genau eine Trajektorie */
        int    i, j;

        plot_field = calloc(sizeof(double), state->field_max);
        if (plot_field == NULL) {
                printf("Out of memory!\n");
                exit(1);
        }
```

```
/* Unterordner fuer die Trajektorien in der KML-Ausgabedatei
 * anlegen
 */
fprintf(state->fh, "<Folder>\n");
fprintf(state->fh, "<name>Trajektorien</name>\n");

/* Einlesen aller Trajektorienaufpunkte in Trajektoriendichtenetz
 * und Ausgabe der Trajektorienverlaeufe in der KML-Datei
 */
for (i = 0; i < state->list_max; i++) {
        current->name = state->list[i];

        /* Name der aktuellen Trajektorie schreiben und Unterordner
         * anlegen
         */
        printf("%s\n", current->name);
        fprintf(state->fh, "<Folder>\n");
        fprintf(state->fh, "<name>%s</name>\n", current->name);

        /* Weglesen des Trajektoriendatei-Headers und einlesen des ersten
         * Trajektorienpunkts
         */
        get_trajectory_start_point(current);

        /* Uebernehmen der Startkoordinaten in die weitere
         * Berechnung
         */
        current->x_old = current->x_begin;
        current->y_old = current->y_begin;

        /* Oeffnen der Ordnerstruktur zum Speichern der momentanen Trajektorie
         * in die KML-Datei und schreiben des Trajektorienstartpunkts
         */
        print_trajectory_header(state->fh, current->x_begin,
            current->y_begin);

        /* Zuruecksetzen der Zaehlvariablen fuer das Einlesen der naechsten
         * Trajektorie
         */
        reset_counter(current);
```

```
                /* Initialisieren der lokalen Abbildungsmatrix der Trajektorie */
                init_field(plot_field, state->field_max);

                /* Einlesen und abbilden der Trajektorie auf der lokalen
                 * Berechnungsnetzmatrix
                 */
                read_trajectory(current, state, plot_field);

                /* Addieren der Trajektorienabbildung der lokalen Abbildungsmatrix
                 * zur Gesamtdarstellungsmatrix
                 */
                for (j = 0; j < state->field_max; j++)
                        state->field_grid[j] += plot_field[j];

                /* Aktuelle Trajektoriendatei schliessen */
                fclose(current->fh);

                /* Struktur der aktuellen Trajektorie in der KML-Datei
                 * schliessen
                 */
                fprintf(state->fh, "</coordinates>\n");
                fprintf(state->fh, "</LineString>\n");
                fprintf(state->fh, "</Placemark>\n");
                fprintf(state->fh, "</Folder>\n\n");
        }

        /* Unterordner fuer Trajektorien schliessen */
        fprintf(state->fh, "</Folder>\n\n");

        free(plot_field);
}

/* Schreiben der zu verwendenden Farbkonfigurationen */
void
print_colorstyles(struct state* state) {

        int i;

        for (i = 0; i < 10; i++) {
```

```
                fprintf(state->fh, "<Style id=\"%i\">\n", i+1);
                fprintf(state->fh, "<PolyStyle>\n");
                fprintf(state->fh, "<color>%s%s</color>\n",
                    get_string(OPACITY), CLASS[state->colorclass[i]]);
                fprintf(state->fh, "<colorMode>normal</colorMode>\n");
                fprintf(state->fh, "<fill>1</fill>\n");
                fprintf(state->fh, "<outline>0</outline>\n");
                fprintf(state->fh, "</PolyStyle>\n");
                fprintf(state->fh, "</Style>\n\n");
        }
}

/* Schliessen des KML-Dokuments */
void
print_end(struct state* state) {

        fprintf(state->fh, "</Document>\n");
        fprintf(state->fh, "</kml>");
}

void
print_freq_header(struct state* state, double w_max, double min, double max) {
        /* Ausgabe der Trajektorinedichten in die KML-Datei
         * nach Modi (Farben) in Unterordner geordnet
         */
        fprintf(state->fh, "<Folder>\n"); /* Unterordner fuer
                                           * Trajektoriendichte anlegen
                                           */

        fprintf(state->fh, "<name>Trajektoriendichte/</name>\n");
        fprintf(state->fh, "<description> Hoechstwert: %5.2f/\n", w_max);
        fprintf(state->fh, "Skalenmaximum: %5.2f\n", max);
        fprintf(state->fh, "Skalenminimum: %5.2f\n", min);
        fprintf(state->fh, "Plotmittelpunkt: %5.2f %5.2f\n", get_float(MIDLO),
            get_float(MIDLA));
        fprintf(state->fh, "Plotgroesse: %ix%i Elemente </description>\n",
            get_int(SIZE), get_int(SIZE));
}
```

```
/* Schreiben des KML-Headers */
void
print_header(struct state* state) {

        /* KML-Header schreibe */
        fprintf(state->fh, "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
        fprintf(state->fh,
            "<kml xmlns=\"http://earth.google.com/kml/2.1\">\n");
        fprintf(state->fh, "<Document>\n\n");
        fprintf(state->fh, "<description>Trajektoriendichte/\n");
        fprintf(state->fh, "Resolution: %ix%i km/\n",
            get_int(RES), get_int(RES));

        /* Info ueber verwendeten Wichtungsmodus schreiben */
        switch (get_int(WEIGHT)) {
        case 1:
                fprintf(state->fh, "Wichtung ueber Abstand zum Starpunkt\n");
                break;
        case 2:
                fprintf(state->fh,
                    "Wichtung ueber Wurzel des Abstandes zum Startpunkt\n");
                break;
        default:
                fprintf(state->fh,
                    "keine Wichtung (absolute Haeufigkeit)\n");
                break;
        }

        fprintf(state->fh, "</description>\n\n");
        fprintf(state->fh, "<name>%s</name>\n\n", get_string(FILENAME));
}

/* Oeffnen der Ordnerstruktur zum Speichern der momentanen Trajektorie
 * in die KML-Datei und schreiben des Trajektorienstartpunkts
 */
void
print_trajectory_header(FILE* fh, double x_begin, double y_begin){

        fprintf(fh, "<Placemark>\n");
        fprintf(fh, "<visibility>0</visibility>\n"); /* unsichtbar */
```

```
        fprintf(fh, "<LineString>\n");
        fprintf(fh, "<coordinates>\n");
        fprintf(fh, "%10.6f, %9.6f, 0\n", x_begin + get_float(OFFLO),
            y_begin + get_float(OFFLA));
}

/* Auslesen der im Verzeichnis mit dem uebergebenen Verzeichnisnamen
 * enthaltenen Dateiename und speichern in eine Dateinamenliste
 * (list)
 * Rueckgabewert sind die Dateinamen in der Datenstruktur list
 */
void
read_dir(char* name, char** list, int list_max) {

        DIR *dirHandle;
        struct dirent * dirEntry;
        int i, j;
        char buf[MAXLINE];

        i = 0;
        /* Oeffnen des Verzeichnisses */
        dirHandle = opendir(name);

        /* Wenn Verzeichnis existiert */
        if (dirHandle) {

                /* Dateinamen auslesen bis Ende erreicht ist */
                while (0 != (dirEntry = readdir(dirHandle))) {
                        /* Versteckte Dateien ignorieren */
                        if (dirEntry->d_name[0] == '.')
                                continue;

                        if (i >= list_max) {
                                printf("Too many files in directory!\n");
                                exit(1);
                        }

                        buf[0] = '\0';
                        j = 0;
```

```
                /* Solange Zeilenende nicht erreicht */
                while ((name[j] != '\0') && (j < MAXLINE))
                        j++;

                /* Wenn Name zu lang */
                if (j >= MAXLINE) {
                        printf("Error: linebuffer too small!\n");
                        exit(1);
                }

                /* Separator des Verzeichnisnamens korrekt umwandeln
                 * und Verzeichnisname vor Dateinamen schreiben
                 */
                if ((name[j-1] == '\\') ||
                    (name[j-1] == '/')) {
                        if (snprintf(buf, MAXLINE, "%s%s",
                                name, dirEntry->d_name)
                            >= MAXLINE) {
                                printf("Linebuffer too small!\n");
                                exit(1);
                        }
                }
                else {
                        if (snprintf(buf, MAXLINE, "%s/%s",
                                name, dirEntry->d_name)
                            >= MAXLINE) {
                                printf("Linebuffer too small!\n");
                                exit(1);
                        }
                }

                /* Name speichern */
                list[i] = strdup(buf);
                if (list[i] == NULL) {
                        printf("Out of memory!\n");
                        exit(1);
                }

                i++;
        }
```

```
                /* Verzeichnis schliessen */
                closedir(dirHandle);

                if (i != list_max) {
                        printf("Directory content has changed!\n");
                        exit(1);
                }
        }
        else {
                printf("Directory %s not found!\n", name);
                exit(1);
        }
}

/*
 * Initialisieren und einlesen der Standardwerte und der gesetzten
 * Programmargumente aus der Programmumgebung
 */
void
read_env(struct param *param)
{
        char *s;

        for (/* */; param->name != NULL; param++) {
                if ((s = getenv(param->name)) == NULL)
                        s = param->u.s;

                switch (param->type) {
                case TYP_INT:
                        param->u.i = atoi(s);
                        printf("%s %d (%s)\n", param->name,
                            param->u.i, param->desc);
                        break;

                case TYP_FLOAT:
                        param->u.f = atof(s);
                        printf("%s %6.2f (%s)\n", param->name,
                            param->u.f, param->desc);
                        break;
```

```
                case TYP_STRING:
                        param->u.s = s;
                        printf("%s %s (%s)\n", param->name,
                            param->u.s, param->desc);
                        break;

                default:
                        printf("Internal error; bad parameter table!\n");
                        exit(1);
                }
        }
}

/* Header der Trajektoriendatei weglesen */
void
read_header(FILE* fh, int headerlines, char* name) {

        int ch;
        int j;

        for (j = 0; j < headerlines; j++) {

                ch = '0';
                while (ch != '\n') {

                        /* Wenn Dateiende erreicht */
                        if ((ch = fgetc(fh)) == EOF) {

                                printf("End of file %s!\n", name);
                                exit(1);
                        }
                }
        }
}

/* Einlesen und abbilden der Trajektorie auf der lokalen Berechnungsnetz-
 * matrix
 */
void
```

```
read_trajectory(struct trajectory* current, struct state* state,
    double* plot_field)
{
        char buf[MAXLINE];
        int  ch;
        int  j = 0;

        /* Einlesen bis Trennzeichen zwischen Laengen-
         * und Breitengradangabe erreicht ist oder neue
         * Zeile beginnt
         */
        while ((ch = fgetc(current->fh)) != EOF) {

                if ((ch != SEPARATOR) && (ch != '\n')) {
                        buf[j++] = (char)ch;
                        buf[j] = '\0';
                }
                else if (ch == SEPARATOR) { /* Wenn Trennzeichen
                                             * erreicht ist ->
                                             * Laengengrad
                                             */

                        /* Laengengrad speichern */
                        current->x_new = atof(buf);
                        j = 0;
                }
                else if (ch == '\n') { /* Wenn Zeilenende erreicht
                                        * ist -> Breitengrad
                                        */

                        /* Breitengrad speichern */
                        current->y_new = atof(buf);
                        j = 0;

                        /* Koordinaten in Grad umrechnen */
                        current->x_new = rad2deg(current->x_new);
                        current->y_new = rad2deg(current->y_new);

                        /* Trajektorienaufpunkt in KML-Datei schreiben (fuer
                         * Trajektoriendarstellung)
```

```
                         */
                        fprintf(state->fh, "%10.6f, %9.6f, 0\n",
                            current->x_new + get_float(OFFLO),
                            current->y_new + get_float(OFFLA));

                        /* Mittelpunkt zwischen vorherigem und aktuellem
                         * Aufpunkt fuer die Trajektoriendichteberechnung
                         * errechnen
                         */
                        current->x_midpoint = (current->x_old +
                            current->x_new) / 2;
                        current->y_midpoint = (current->y_old +
                            current->y_new) / 2;

                        /* Errechenen der momentanen Wichtung und abbilden
                         * aller Trajektorienaufpunkte im Plotbereich
                         * zwischen dem letzten und dem aktuellen Punkt auf
                         * der lokalen Abbildungsmatrix 'plot_field'
                         */
                        plot_to_next_point(plot_field, current, state);

                        /* Speichern des letzten Aufpunkts */
                        current->x_old = current->x_new;
                        current->y_old = current->y_new;
                }
        }
}

/* Zuruecksetzen der Zaehlvariablen fuer das Einlesen der naechsten
 * Trajektorie
 */
void
reset_counter(struct trajectory* current){
    current->x_new = current->y_new = current->x_midpoint =
        current->y_midpoint = 0;
}

/*
 * Freigeben der reservierten Speicherbereiche und schliessen der Sammeldatei
 */
```

```
void
reset_state(struct state* state)
{
        int i;

        fclose(state->fh);
        free(state->field_grid);
        for (i = 0; i < state->list_max; i++)
                free(state->list[i]);
        free(state->list);
}

/* Umkopieren der Werte von Punkt1 (x1, y1) und Punkt2 (x2, y2) */
void
rotate_points(double* x1, double* y1, double* x2, double* y2) {
        double tmp;
                        tmp = *y1;
                        *y1 = *y2;
                        *y2 = tmp;

                        tmp = *x1;
                        *x1 = *x2;
                        *x2 = tmp;

}

/* Plotten aller Netzelemente mit der Trajektoriendichte der aktuell zu
 * plottenden Klasse (k) im Plotbereich
 */
void
sort_and_plot(struct state* state, int k, double dw, double min, double w_max) {

        double p, dy, dx, x, y;
        int    i, val;

        /* Berechnen des Prozentanteils der momentan
         * betrachteten Klasse am Maximalwert
         */
        p = ((double)(k * dw) + min) * 100.0 / w_max;
```

```
        /* Unterordner fuer aktuellen Modus anlegen */
        fprintf(state->fh, "<Folder>\n");
        fprintf(state->fh, "<name>ab %3.0f%%</name>\n", p);

        /* Netzelementhoehe in Grad (konstant) */
        dy = get_int(RES) / DEGDISTANCE;

        for (i = 0; i < state->field_max; i++) {
                val = (int)state->field_grid[i];
                if (k == val) {

                        /* Umrechnen der aktuellen Matrixposition
                         * in Zeile (y) und Spalte (x)
                         */
                        y = (int)(i / state->x_field);
                        x = i - (y * state->x_field);

                        /* Umrechnen der Zeile in Breitengrad */
                        y = state->la_min + y * dy;

                        /* Aktuelle Netzelementbreite in Grad
                         * (Breitengradabhaengig)
                         */
                        dx = get_int(RES) / (DEGDISTANCE *
                            cos(deg2rad(y)));

                        /* Umrechnen der Spalte in Laengenengrad */
                        x = state->lo_min + x * dx;

                        /* Aktuelles Element in die KML-Datei
                         * schreiben
                         */
                        plot_element(state->fh, x, y, k);
                }
        }

        /* Schliessen des aktuellen Modus-Unterordners */
        fprintf(state->fh, "</Folder>\n\n");
}
```