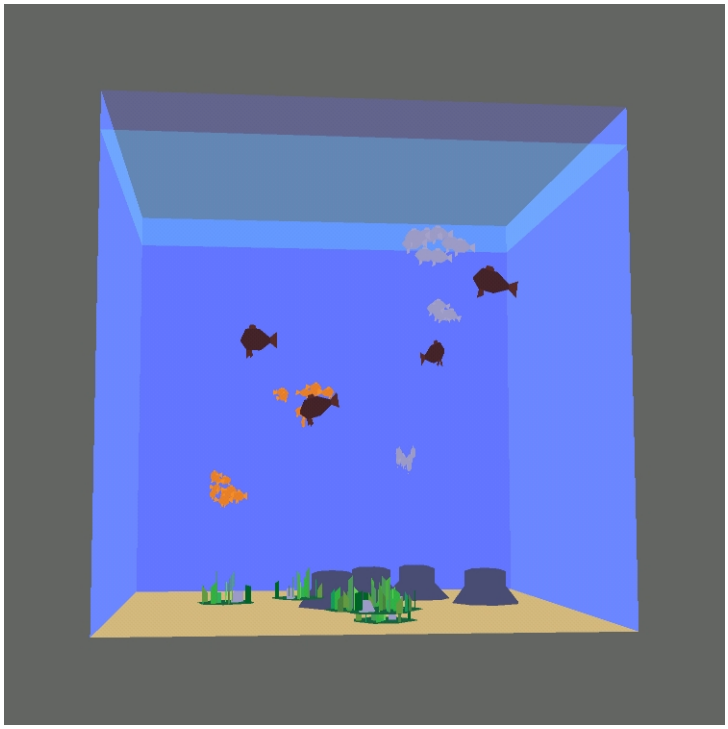
Selbstorganisation von Tierschwärmen

Softwarepraktikum im WS 03/04 bei Dr. Michael Winckler

Julia Ziegler
JuliaZiegler@gmx.de

Marcel Watzlawek
marcel.watzlawek@t-online.de



1 Einleitung

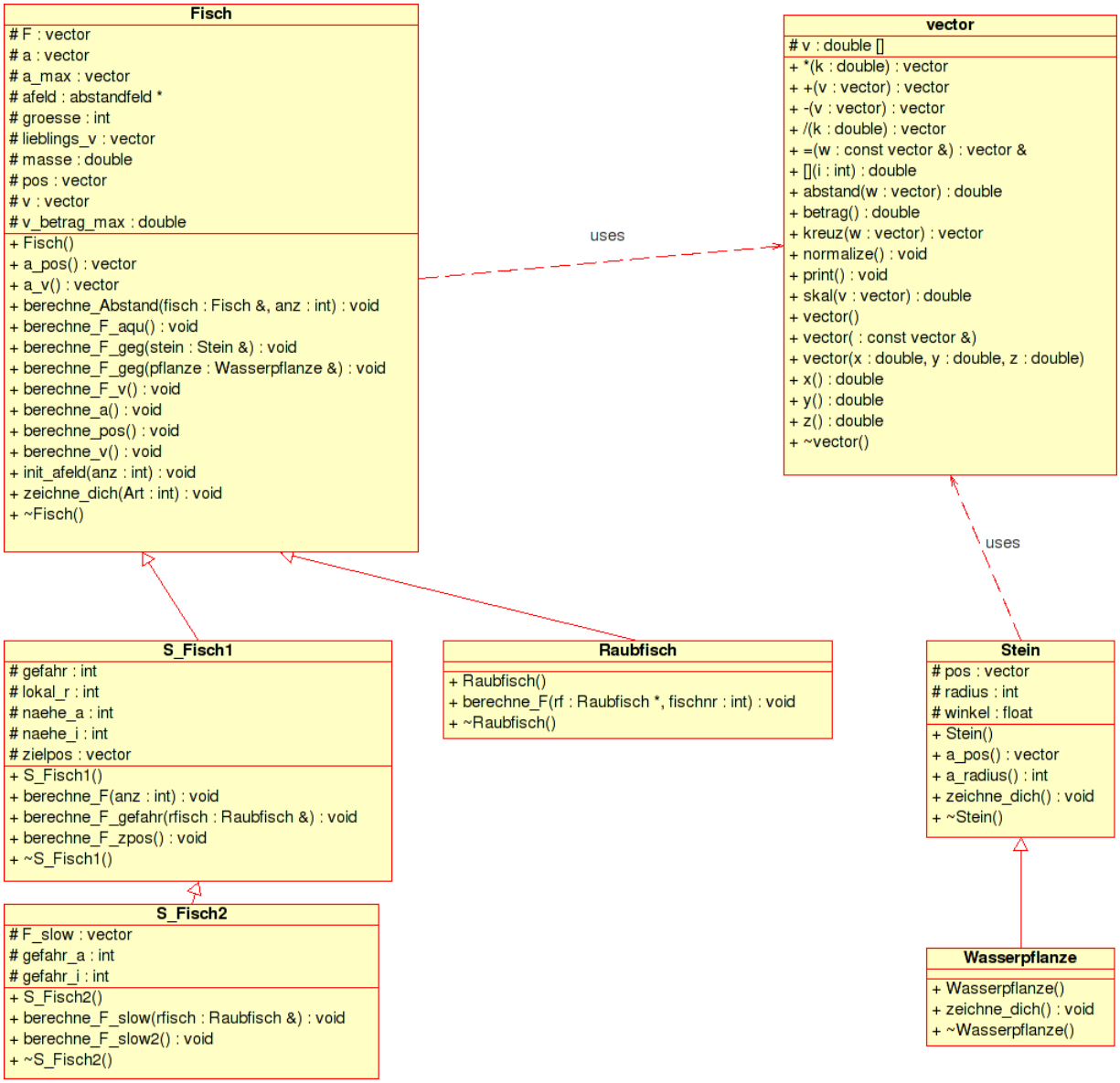
Diese Dokumentation beschreibt die Ergebnisse unserer Arbeit an dem Softwarepraktikum im WS 03/04 an der Ruprecht-Karls-Universität Heidelberg zum Thema „Selbstorganisation von Tierschwärmen“. Hierbei bestand unsere Aufgabe in der Erstellung eines 3D-Computermodells zur Simulation des Verhaltens von verschiedenen Fischschwärmen in ihrem Lebensraum, wobei die Simulation auf Echtzeitberechnungen basiert.
Das Aquariumvolumen sowie die Anzahl der Gegenstände und Fische pro Art ist dabei variabel und wird vom Benutzer ausgewählt.
Zur Realisierung dieser Aufgabe wählten wir einen objektorientierten Ansatz unter Verwendung von C++ und den OpenGL-Bibliotheken `gl`, `glu` und `glut`. Für das Fenster zur Eingabe der oben genannten, variablen Größen wurde zusätzlich die Bibliothek `glui` benutzt.
Diese Dokumentation enthält eine Kuranleitung zur Bedienung des Programms, eine detailierte Beschreibung der Klassen und Informationen zu der physikalischen Kraftberechnung für die Bewegung der Fische. Als nächstes werde ich genauer auf die graphische Darstellung und die Erstellung des Eingabefensters eingehen. Zuletzt werden Screenshots der Simulation für sich selbst sprechen.

2 Kurzanleitung

Die Bedienung unserer Simulation ist nicht weiter schwierig: Einfach in den Listboxen (Drop-Down-Menüs) die gewünschte Aquariumgröße bzw. Anzahl der Gegenstände und Fische auswählen und den Button „Simulation starten“drücken.
Zum Beendender Simulation das Kreuz ganz oben rechts am Fenster anklicken oder die Taste „q“drücken.

3 Klassen

3.1 Klassendiagramm



Legende:
„+“: public: öffentliche Methoden der Klassen
„-“: private: private Methoden und Datenmitglieder der Klassen
„#“: protected: geschützte Methoden und Datenmitglieder der Klassen

3.2 Beschreibung der einzelnen Klassen

3.2.1 Verwendete Funktionen und Variablen

Folgende Funktionen sind definiert:

- `vector zufall(int von, int bis)`: Hier werden einem Vektor zufällige Werte zugewiesen im Bereich von den Variablen `von`, `bis`. Diese Funktion wird verwendet, um den Fischen und Gegenständen zu Beginn der Simulation eine zufällige Position zuzuweisen.

- `vector zufall2(int von1, int bis1, int von2, int bis2, int von3, int bis3)`: Auch hier werden einem Vektor zufällige Werte zugewiesen, wobei die einzelnen Komponenten des Vektors in verschiedenen Bereichen liegen. Diese Funktion wird ebenfalls verwendet, um den Fischen und Gegenständen zu Beginn der Simulation eine zufällige Position zuzuweisen.
- `int vergleiche(const void* a, const void* b)`: Wird für den QuickSort benötigt, um das `afeld` der Fische nach Abstand zum entsprechenden Fisch zu sortieren Die Funktion gibt 1 zurück, falls a¿b, anderenfalls -1.
- `vector normalenvector(vector p1, vector p2, vector p3)`: Berechnung des Normalenvektors einer Fläche, die von den Punkten `p1`, `p2` und `p3` aufgespannt wird.
- `double absbetrag (double wert)`: Berechnet den Absolutbetrag einer Variable vom Typ `double`.

Die folgenden globalen Variablen werden im Programm verwendet:

```
// Initialisierungsvariablen
int a_hoehe = 500;      //Aquariumhoehe
int a_breite = 500;     //Aquariumbreite
int a_tiefe = 500;      //Aquariumtiefe

int u=30;               //Abstand Wasseroberflaeche - Aquariumrand

int offset = 20;                          //Fensteroffset
int fenster_h = (int)(900 + 2*offset );   //Fensterhoehe
int fenster_b = (int)(900 + 2*offset );   //Fensterbreite
int anz_pos = 2;                          //gibt die Anzahl durlaufender Positionen an.
double delta_t = 0.2;                     //Zeitschritt

//Anzahl der verschiedenen Klassenmitglieder
int anz_sf1;
int anz_umgeb1 = 3;
int anz_sf2;
int anz_umgeb2 = 3;
int anz_rf;
int anz_steine;
int anz_pflanzen;


// Daempfungsfaktoren fuer Kraftberechnung
double d_v = 0.2;       //erreiche Libelingsgeschw. (Betrag)
double d_f = 0.2;       //orientiere dich an anderen Fischen
double d_geg = 0.2;     //weiche Gegenstaenden aus
double d_w = 2.3;       //weiche Waenden aus
double d_wv = 3;        //fuer das Ausweichen an der vorderen Aquaiumwand
double d_z = 0.05;      //erreiche Zielposition
double d_s = 0.2;       //verlangsame Geschwindigkeit falls keine Gefahr
double d_gef = 0.1;     //Gefahr
double d_jage = 0.4;    //fuer Raubfisch; orientiere dich an Position der zu jagenden Fisches
```

```
// fuer das Eingabefenster benoetigte Variablen
bool eingelesen=0;

int AquariumGroesse=0;
int Anzahl_Pflanzen=0;
int Anzahl_Steine=0;
int Anzahl_SF1=0;
int Anzahl_SF2=0;
int Anzahl_RF=0;

int rf_runde=0;

GLUI *glui;
```

Die Funktionen befinden sich in `MyFunktionen.h` und `MyFunktionen.c`. Die globalen Variablen sind in `MyGlobals.h` deklariert und zum Teil auch definiert.

3.2.2 Klasse `vector`

Mit der Klasse `vector` wird ein 3-dimensionaler Vektor und die möglischen Operationen damit implementiert:

- Addition
- Subtraktion
- Multiplikation mit einer Konstanten vom Typ `double`
- Division durch eine Konstante vom Typ `double`
- Skalarprodukt
- Kreuzprodukt
- Normalisierung
- Abstand zwischen zwei Punkten
- Zuweisung
- Zugriff auf eine Komponente (x, y oder z)
- Ausgabe des gesamten Vektors
- Konstruktoren
  - Copy-Konstruktor
  - mit Wert-Zuweisung

Die einzelnen Komponenten des Vektors sind vom Typ `double` und sind in dem protected Feld `v` gespeichert. Instanzen dieser Klasse werden von allen Klassen als grundlegender Datentyp zur Berechnung und Modellierung verwendet.
Die Klasse `vector` ist in `MyVector.h` und `MyVector.c` implementiert.

3.2.3 Klasse `Fisch`

Die Klasse `Fisch` ist Basisklasse für die beiden Schwarmfischarten und die Raubfische, d.h. hier werden die grundlegenden Eigenschaften der Fische implementiert:

```
vector a_pos();          //gibt die Position des Fisches zurueck
vector a_v();            //gibt die Geschwindigkeit des Fisches zurueck
void berechne_a ();      // a= F/m
void berechne_v ();      // v= v+delta_t * a
void berechne_pos ();    // pos= pos + delta_t * v
void berechne_F_v();     //Schwarmbildung und Abstand halten innerhalb des Schwarmes
void berechne_F_aqu();                  //Kraefte, durch die Aquariumwaende verursacht werden
void berechne_F_geg(Stein& stein);      //Kraefte, die durch Steine verursacht werden
void berechne_F_geg(Wasserpflanze& pflanze);

//fuellt afeld mit anz_umgeb vielen Positionen, welche den geringsten Abstand
//zum aktuellen Fisch berechnen
void berechne_Abstand(S_Fisch1& fisch, int anz);

//setzt die Abstaende wieder auf die Maximalwerte. Grund : sonst wird immer nur
//der leicht veraenderte Abstand des gleichen Fisches in das Feld eingetragen
void init_afeld (int anz);

//parametrisierte Zeichenroutine f"ur alle Fischarten
void zeichne_dich(int Art);

vector pos;              // aktuelle Position des Fisches
vector F;                // Kraft, die auf den Fisch wirkt
vector v;                // Geschwindigkeit des Fisches
vector a;                // Beschleunigung
vector lieblings_v ;     // bevorzugte Schwimmgeschwindigkeit und -richtung

//Maximalwerte: Ist v bzw. a groesser als dieser Maximalwert, so wird dieser angenommen
double v_betrag_max;
vector a_max;

//speichert Abstand zu anderen Fischen und deren Position
struct abstandfeld{
double abstand;
vector posi;
S_Fisch1* fisch;
};

//Pointer auf afeld
abstandfeld* afeld;

double masse;            // Masse des Fisches: je groesser der Fisch, desto schwerer ist er
int groesse;             // Ausdehnung des Fisches (Kugel um den Fisch)
```

Die Klasse `Fisch` ist in `MyFisch1.h` und `MyFisch1.c` implementiert.

3.2.4 Klasse `S_Fisch1`

Diese Klasse implementiert die erste Schwarmfischart, die immer mit einer konstanten Geschwindigkeit schwimmt und in der Simulation silberfarben dargestellt wird. `S_Fisch1` ist public von `Fisch` abgeleitet und erbt somit die grundlegenden Funktionen zur physikalischen Berechnung der Kraft, Geschwindigkeit und Beschleunigung sowie eine Routine zur Berechnung der aktuellen Position. Auerdem wird die Zeichenroutine `zeichne_dich (int Art)` geerbt. Zudem werden die folgenden Funktionen und Variablen definiert:

```
bool get_gejagt();              //gibt gejagt zurueck
void set_gejagt(bool wert);     //weist gejagt wert zu

void berechne_F(int anz);       /*Kraefte, die durch andere Fische auf den Fisch wirken
                                  -> Schwarmbildung
                                  und Abstand (zwischen naehe_i und naehe_a) halten
                                  zu anderen Fischen */
void berechne_F_zpos();         //bringt den Fisch auf seine Zielposition
void berechne_F_gefahr(Raubfisch& rfisch);      //Beschleunigung + Ausweichen falls Raubfisch
                                                //in Radius "gefahr"


vector zielpos;         //aktuelle Zielposition des Fisches
int gefahr;             //Gefahrenradius (falls Raubfisch innerhalb dieses Radius ->Gefahr)
int naehe_i ,naehe_a;   //Toleranzbereich fuer den Abstand zweier Fische
int lokal_r ;           //lokaler Radius
bool gejagt;            //Fisch wird in dieser Runde gejagt
```

Die Klasse `S_Fisch1` ist in `MyFisch1.h` und `MyFisch1.c` implementiert.

3.2.5 Klasse `S_Fisch2`

Hier wird die zweite Schwarmfischart impementiert, die auer dem Schwarmverhalten wie bei der ersten Schwarmfischart zusätzlich noch eine Verlangsamung, wenn kein Raubfisch in der Nähe ist, und eine Beschleunigung bei Gefahr realisiert. Die Fische dieser Art bilden größere und dichtere Schwärme. Diese Fischart wird in der Simulation orangenfarben gezeichnet. Die Klasse `S_Fisch2` ist von `S_Fisch1` abgeleitet und übernimmt somit alle Eigenschaften von `Fisch` und `S_Fisch1`. Zusätzlich wurden die folgenden Methoden und Variablen definiert:

```
//Kraft, die dafuer sorgt, dass der Fisch ohne Gefahr seine Geschwindigkeit verringert
void berechne_F_slow(Raubfisch& rfisch);

//Kraft, die dafuer sorgt, dass der Fisch ohne Gefahr seine Geschwindigkeit verringert
void berechne_F_slow2();

int gefahr_a;           //aeusserer Gefahrenradius.
int gefahr_i;           //innerer       Gefahrenradius
vector F_slow;          //Kraftvektor verlangsamt den Fisch bei Ruhe, Beschleunigung + Ausweichen
                        //bei Gefahr. (abhaengig von den beiden Gefahrenradien)
```

Die Klasse `S_Fisch2` ist in `MyFisch2.h` und `MyFisch2.c` implementiert.

3.2.6 Klasse `Raubfisch`

Diese Klasse ist von `Fisch` abgeleitet und erhält zusätzlich ein Jagdverhalten und weicht anderen Raubfischen sowie den Aquariumwänden und Gegenständen aus. Diese Eigenschaften sind folgendermaßen implementiert:

```
/*berechnet Kraft, um naehesten Fisch zu verfolgen
  (jeder Schwarmfisch darf nur von einem Raubfisch gejagt werden)
  und anderen Raubfischen auszuweichen */
void berechne_F(Raubfisch* rf, int fischnr );
```

Die Klasse `Raubfisch` ist in `MyFisch2.h` und `MyFisch2.c` implementiert.

3.2.7 Klasse `Stein`

Diese Klasse definiert den ersten Gegenstand im Aquarium, der von den Fischen umschwommen wird. Zur Darstellung und Berechnung des Ausweichens wurden die folgenden Eigenschaften definiert:

```
vector a_pos(); //gibt die Position des Steines zurueck
int a_radius(); //gibt den Radius zurueck

//Zeichenroutine
void zeichne_dich();

vector pos;     //Position im Aquarium
int radius;     //Steinradius
float winkel;   //Drehwinkel zum Ausgangskoordinatensystem
```

Die Klasse `Stein` ist in `MyGegenstand.h` und `MyGegenstand.c` implementiert.

3.2.8 Klasse `Wasserpflanze`

Die Klasse `Wasserpflanze` ist von `Stein` abgeleitet und stellt den zweiten Gegenstand im Aquarium dar. Die Wasserpflanze kann somit ihre Position und den Radius zu Berechnungszwecken zurückgeben , hat jedoch eine eigene Zeichenroutine und keine Datenmitglieder.

Die Klasse `Wasserpflanze` ist in `MyGegenstand.h` und `MyGegenstand.c` implementiert.

4 Physikalische Kraftberechnung

Das Verhalten der Fische wird durch eine physikalische Kraftberechnung implementiert, dem die folgenden Gesetzmäßigkeiten zu Grunde liegen:

Die Beschleunigung `a` eines Fisches läßt sich aus der Kraft, die auf ihn wirkt, geteilt durch seine Masse, ermitteln:
`a = F / masse`
Für die Beschleunigung gibt es einen Maximalwert, der nicht überschritten werden kann.

Die Geschwindigkeit `v` eines Fisches wird berechnet aus seiner Geschwindigkeit aus dem letzten Zeitschritt addiert mit seiner Beschleunigung mal dem Zeitschritt `delta_t`:
`v = v + a * delta_t`
Auch die Geschwindigkeit kann einen Maximalwert nicht überschreiten, damit zu starke Richtungsänderungen in einem Zeitschritt ausgeschlossen werden können.

Die aktuelle Position eines Fisches wird aus seiner alten Position addiert mit seiner Geschwindigkeit mal dem Zeitintervall `delta_t` errechnet:
`pos = pos + (v * delta_t)`

Folgende Kräfte wirken auf alle Fische:

- `F_v`: Erreichen einer „Lieblingsgeschwindigkeit“
- `F_aqu`: Abstoßung von den Aquariumwänden und der Wasseroberfläche
- `F_geg`: Abstoßung von den Gegenständen im Aquarium

Auf die Schwarmfische wirkt zusätzlich die Kraft `F_f`, die durch eine bestimmte Anzahl von den nähsten Fischen beeinflusst wird und zur Schwarmbildung führt, jedoch auch die Abstoßung zwischen den Fischen realisiert, falls sich diese zu nahe kommen, damit sie einander weder berühren noch überschneiden. Auch wirkt auf die Schwarmfische die Kraft `F_zpos`, die jeden Fisch auf eine zufällig gewählte Zielposition zuschwimmen läßt. Die Kraftkomponente `F_gefahr` führt dazu, dass ein Fisch seine Richtung ändert, wenn ein Raubfisch in seine unmittelbare Nähe kommt.

Auf den zweiten Schwarmfisch wirkt zusätzlich die Kräft `F_slow`. Diese bewirkt eine Verlangsamung der Geschwindigkeit, falls kein Raubfisch in der Nähe ist. Befindet sich der Raubfisch in einem bestimmten Abstand, so wird die Geschwindigkeit beibehalten. Kommt der Raubfisch jedoch in die unmittelbare Nähe des Fisches, so beschleunigt dieser und ändert seine Schwimmrichtung. Enthält die Simulation keinen Raubfisch, so wird der Fisch von der Kraft `F_slow2` beeinflusst, die nur eine Verlangsamung bewirkt.

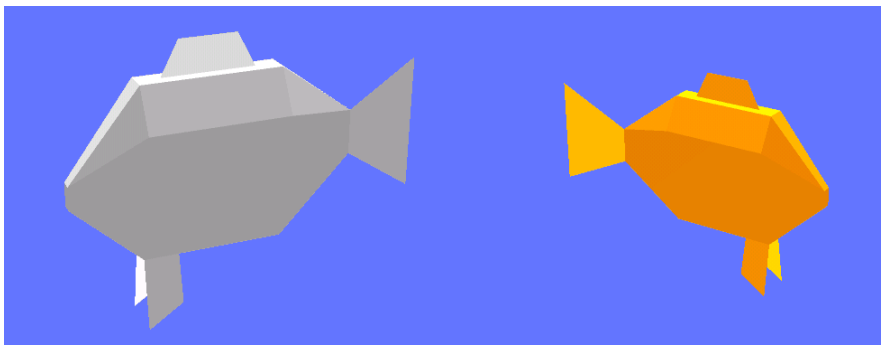
Der Raubfisch hat, zusätzlich zu den oben genannten Kräften, die auf alle Fische wirken, die Kraft `F_f`, die auf ihn wirkt und sein Jagdverhalten bestimmt (ein Raubfisch jagt immer den nähsten Fisch, der nicht bereits von einem anderen Raubfisch gejagt wird) sowie dafür sorgt, dass er Abstand zu anderen Raubfischen hält.

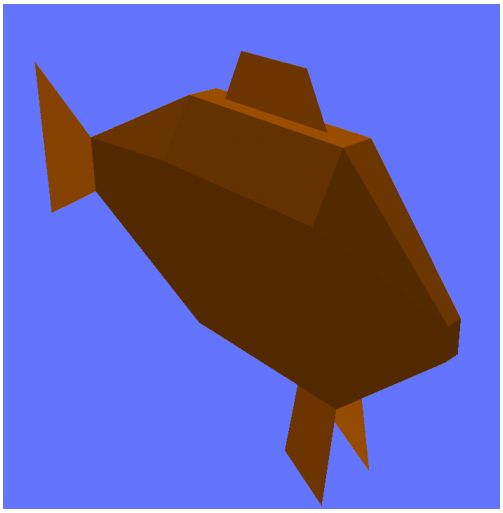
Die Kräfte sind unterschiedlich mit Hilfe von global definierten Dämpfungsfaktoren gewichtet, so dass eine wichtigere Kraftkomponente, z.B. die Abstoßung von den Aquariumwänden, den anderen überwiegt.

5 Graphische Darstellung

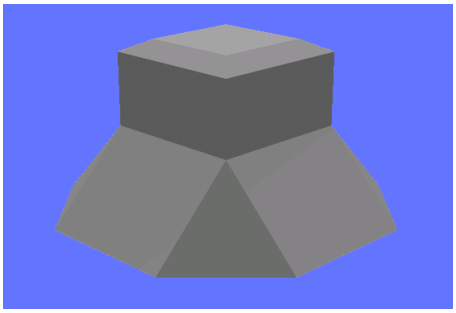
Die Funktion `void Fisch::zeichne_dich(int Art)` ist eine parametrisierte Zeichenroutine, die alle drei Fischarten graphisch darstellt. Sie bekommt die jeweilige Fischart als Parameter übergeben.
In dieser Funktion werden zunächst Felder für die zum Zeichnen benötigten Punkte und Normalenvektoren des Fisches angelegt und weitere Variablen und artspezifische Faktoren deklariert und definiert. Es wird eine Struktur `colour` definiert, die einen Rot-, Grün- und Blauwert enthält und somit die Zusammensetzung eines Farbtons speichert. In einem 2-dimensionalen Feld mit Werten vom Typ `colour` werden die Farben der Fische verwaltet. Auch wird der Mittelpunkt `M` des Fisches (abhängig von der Fischart) definiert, der zur Berechnung der Verschiebung und Rotation benötigt wird. Zusätzlich müssen hierfür die Rotationswinkel unter Beachtung aller Spezialfälle berechnet werden. Nun kann der Fisch an seine Position verschoben und um die berechneten Winkel um die y- und z-Achse rotiert werden. Für eine korrekte Verschiebung muss der Fisch zuletzt noch um `-M` verschoben werden, da unsere Kraftberechnung die Fische als Punkt annimmt und auf dieser Basis die Position berechnet.
Danach wurden blockweise zunächst Punkte und Normalenvektoren definiert und dann mit diesen Flächen erstellt. In einem Block finden sich immer alle Flächen zu einem bestimmten Teil des Fisches bzw. alle Dreiecke, die den Fisch vervollständigen. Es wurden wegen dem Übergang zwischen den Flächen bei Licht bzw. Schatten nur Rechtecke vom Typ `GL_QUADS` verwendet.
Zuletzt müssen die Verschiebungen und Rotationen wieder rückgängig gemacht werden, damit die nächsten Verschiebungen und Rotationen wieder relativ zum Aquarium sind und nicht zur vorherigen Darstellung.

Die folgenden Abbildungen zeigen Screenshots der drei Fischarten:

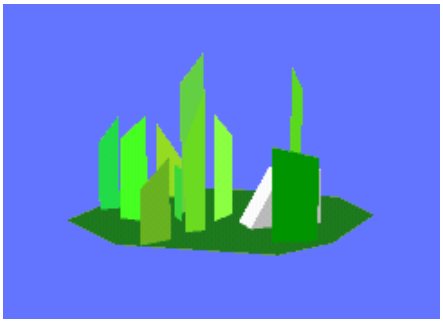




Die Funktion `void Stein::zeichne_dich()` dient zur graphischen Darstellung der Steine und ist der oben beschriebenen Routine `void Fisch::zeichne_dich(int Art)` sehr ähnlich: Zunächst werden auch hier Punkt- und Normalenvektoren deklariert und unter Verwendung von einiger Variablen für die Seitenverhältnisse definiert. Mit diesen Vektoren werden statische Punkt-und Normalenvektoren deklariert: `static GLdouble v1[]`,etc. Auf diese kann leicht zugegriffen werden.
Damit sich die Rotation nur auf das aktuelle Objekt zu einem Zeitpunkt auswirkt, wird die Darstellungsmatrix mit `glPushMatrix()` gesichert. Danach wird der Stein an die richtige Stelle im Aquarium verschoben und um einen zufälligen Winkel um die Bodennormale rotiert. Mit `glPopMatrix()` nach dem Zeichnen restauriert.
Nun folgt die Definition der Flächen mit Hilfe der bereits definierten, statischen Punkt- und Normalenvektoren. Hierbei wurden ausschließlich graue Polygone verwendet.
Der folgende Screenshot zeigt den so dargestellten Stein:



Die Funktion `void Wasserpflanze::zeichne_dich()` übernimmt die graphische Darstellung der Wasserpflanzen. Sie ist analog zu der Zeichenroutine für die Steine. Die Wasserpflanzen werden jedoch skaliert mit einem vom Radius der Pflanze abhängigen Skalierungsfaktor. Punkt- und Normalenvektoren werden hier in Feldern verwaltet, bevor sie wieder zur Definition statischer Punkt- und Normalenvektoren dienen. Auch die Flchen der Wasserpflanzen sind vom Typ `GL_POLYGON`, haben jedoch verschiedene Grüntöne. Die Wasserpflanze enthält einen Stein.
Die folgende Abbildung zeigt einen Screenshot der Wasserpflanze.



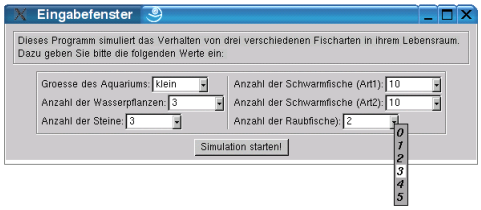
6 Erstellung des Eingabefensters

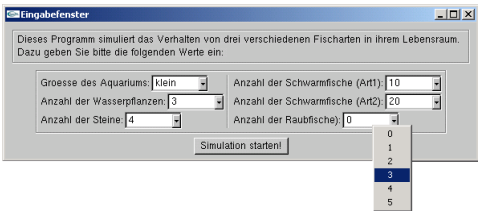
Zur Erstellung des Eingabefensters wurde die `glut`-basierte Benutzerinterface-Bibliothek `glui` verwendet. Das Eingabefenster enthält abgesehen von einem Kasten mit statischem Text einen zweiten, zweispaltigen Kasten mit den Listboxen zur Auswahl der Aquariumgröße sowie Anzahl der Gegenstände und Fische pro Art. Des weiteren gibt es einen Button „Simulation starten!“zur Beendigung der Eingabe und zum Beginnen der Simulation mit den eingelesenen Werten. Sobald der Button gedrückt ist, ruft er die folgende Funktion auf:
`void button_call (int id)`: Hier wird ein Signal gesetzt,welches anzeigt, dass eingelesen wurde und die Simulation beginnen kann. Des weiteren werden die eingelesenen Werte den Variablen `anz_sf1`, `anz_sf2`, `anz_rf`, `anz_pflanzen` und `anz_steine` zugewiesen und die Aquariumgröße initialisiert. Falls das Aquarium nicht „klein“ist, wird das Simulationsfenster als Vollbild angezeigt. Als letztes wird das Eingabefenster geschlossen.

Die Erstellung des Eingabefensters wird in `SimuMain.c` implementiert.

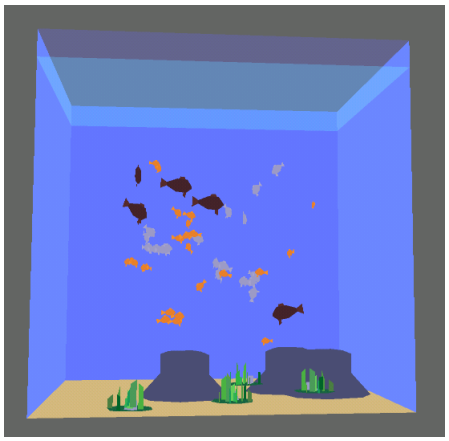
Programmierdetails können im kommentierten Quellcode ersehen werden.

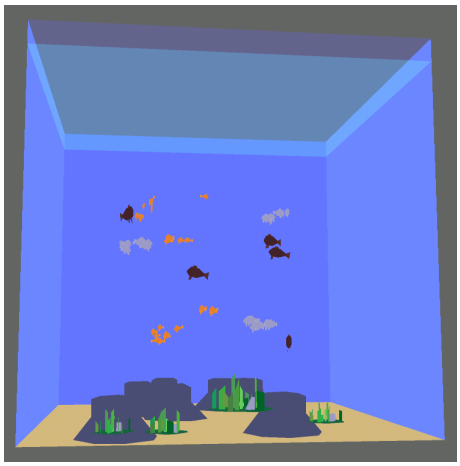
Das so erzeugte Eingabefenster hat folgende Form:

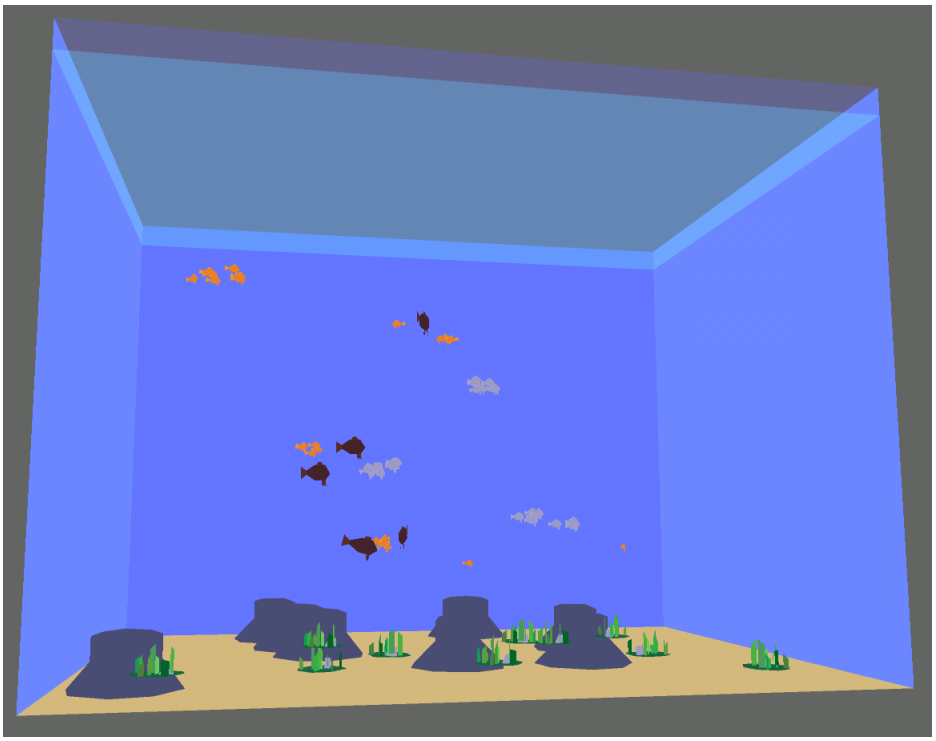


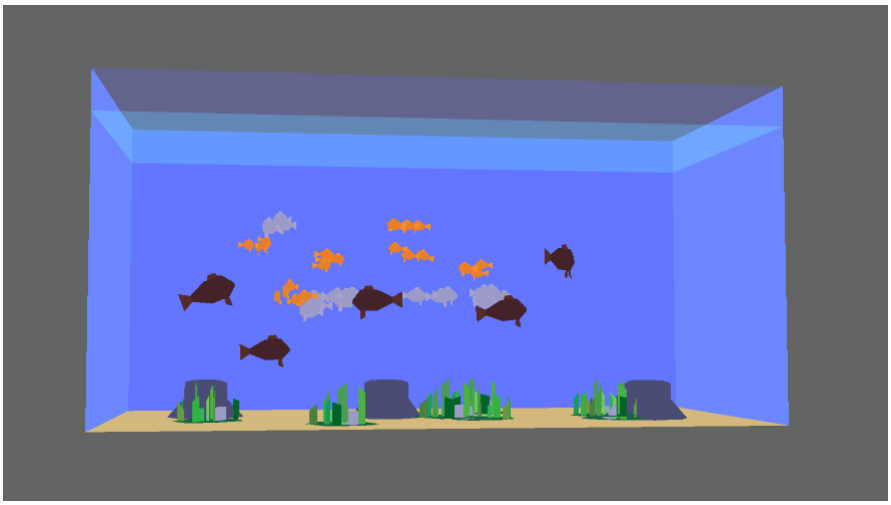


7 Screenshots









8 Literaturverzeichnis

- „Jetzt lerne ich OpenGL“ von Lorenz Burggraf
- „The OpenGL Programming Guide - The Redbook“ von Silicon Graphics, Inc. (`http://www.opengl.org/documentation/red_book_1.0/`)
- „Objektorientiertes Programmieren in C++“ von Nicolai Josuttis
- „Glui“von Paul Rademacher
- `http://pille.iwr.uni-heidelberg.de/`