Masterarbeit im Studiengang Informatik

Virtuelles Korallenriff: Ein Framework zum prozeduralen Wachstum von Korallen auf Basis von L-Systemen und Isoflächen

vorgelegt durch:
Anthea Sander

Gutachter:

Prof. Dr. Gabriel Zachmann
Arbeitsgruppe Computergrafik und Virtuelle Realiät
Universität Bremen

PD. Dr. Hauke Reuter
Theoretische Ökologie und Modellierung
Leibniz-Zentrum für Marine Tropenökologie

8. September 2015

Erklärung zur Masterarbeit

Hiermit erkläre ich, Anthea Sander, dass ich die Masterarbeit mit dem Titel 'Virtuelles Korallenriff: Ein Framework zum prozeduralen Wachstum von Korallen auf Basis von L-Systemen und Isoflächen' selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe.

Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.

Unterschrift: _____

Datum: _____

Danksagung

Jede Masterarbeit trägt die Handschrift des Authors, und doch ist sie niemals die Arbeit eines Einzelnen. An dieser Stelle möchte ich die Gelegenheit nutzen und meinem Professor Dr. Gabriel Zachmann für die Unterstützung und Betreuung während der Arbeit danken. Meinem Betreuer und zweitem Gutachter Dr. Hauke Reuter bin ich für seine bereichernden Ausführungen aus dem Bereich der Zoologie und Botanik dankbar. Ohne unseren regen Austausch hätte die Arbeit nicht den Variantenreichtum erreicht, welche sie heute auszeichnet. Dr. Andreas Kubicek stellte mir das von ihm entwickelte Korallenmodel samt Quellcode zur Verfügung. Ich danke ihm im besonderen Maße für dieses starke Fundament. Ferner danke ich den Studenten des Bachelorprojekts VRCoralReef für die rege Zuarbeit und den tollen Austausch. Auch die Mitarbeiter der Arbeitsgruppe Computergrafik und Virtuelle Realität waren für mich immer ein Anlaufpunkt für Fragen und Diskussionen. Vielen Dank für diese Unterstützung.

Wie in jedem Studium gab es auch bei mir Höhen und Tiefen. Lieben Dank an meine Freunde und Kommilitonen, die mir immer zur Seite standen. In der letzten Phase danke ich vor allen Maria Meister für das Korrekturlesen dieser Arbeit. Außerdem gebührt besonderer Dank meiner Familie, die mich jederzeit mit unendlich viel Geduld und liebevoller Fürsorge unterstützt hat. Meinem Bruder Lennart und meinem Schwäger Arne sei ferner für die mathematische Unterstützung während dieser Arbeit gedankt.

Schließlich und von ganzem Herzen möchte ich auch meinem Mann Sören danken. Einerseits danke ich Dir für die fruchtbaren Diskussionen, Anregungen und Kritiken an dieser Arbeit. Andererseits danke ich für die langjährige Unterstützung und Geduld während meiner gesamten bisherigen beruflichen Laufbahn.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Bei dem Korallenriff handelt es sich um eines der wichtigsten Ökosysteme unserer Erde. Viele Hundert Tier- und Pflanzenarten finden in dem Riff ihr Zuhause (Krupp, 2008). Dieses Ökosystem entwickelt sich sehr langsam. So gibt es heute viele Korallenriffe, die mehrere Tausend Jahre alt sind. Das Korallenriff ist aber keinesfalls unzerstörbar. Vor allem in der heutigen Zeit gibt es viele durch den Menschen verursachte Faktoren, die einen negativen Einfluss auf dieses Ökosystem haben (Schuhmacher, 1988). Lokale Störfaktoren wie Dynamitfischerei oder die minimale Erwärmung des Wassers ($\pm$ 0.5 Grad im Durchschnitt) können für ein Korallenriff verheerende Folgen haben und führen beispielsweise zu einem Massensterben von Korallen und somit zur Zerstörung des gesamten Ökosystems (Brown und Macfadyen, 2007; McClanahan et al., 1999; Hoegh-Guldberg und Smith, 1989). Da die Veränderung im Ökosystem aber ein schleichender und komplexer Prozess ist, der sich über mehrere Monate oder Jahre erstreckt und auf vielerlei Ursachen beruht, ist der Zusammenhang zwischen Ursache und Wirkung für den Menschen nicht direkt ersichtlich.

Um die Auswirkungen einzelner Störfaktoren und das Zusammenspiel der unterschiedlichen Organismen besser verständlich und nachvollziehbar zu machen, hat Kubicek et al. (2012) ein virtuelles zweidimensionales Simulationsmodel eines Korallenriffs entwickelt. Aufbauend auf dieser schematischen Darstellung soll in dieser Arbeit das dreidimensionale Wachstum eines exemplarischen Korallenriffs virtuell animiert werden. Implizite Flächen sind ein verbreitetes Werkzeug um aufwendige Formen in der Computergrafik zu repräsentieren (de Araújo et al., 2015). Hier wird der Metaballs-Ansatz zur Repräsentation des Volumen mit L-Systemen kombiniert und abschließend bewertet.

Kapitel 2 und 3 beschäftigen sich mit den theoretischen Grundlagen von Korallenriffen sowie prozeduralem Pflanzenwachstum. Es wird näher auf das Korallenriffmodel von Kubicek et al. (2012) eingegangen und dargelegt, welche Aspekte in der Generierung einer Virtuellen Realität unter Wasser wichtig sind. Ferner werden etablierte Algorithmen zur prozeduralen Erzeugung von 3D-Modellen von Pflanzen und organischen Strukturen vorgestellt. Die darauf folgenden Kapitel 4 und 5 dokumentieren die Implementation eines virtuellen Korallenriffs. Das Vorgehen gliedert sich in drei Schritte. Zunächst wird eine wasserähnliche Umgebung erzeugt. Dann werden die Positionen aus dem Korallenriffmodel von Kubicek et al. (2012) in die Simulation übertragen. Anschließend werden Algorithmen vorgestellt, die für jede Koralle ein individuelles Gitternetz auf Basis der spezifischen Phänotypen erzeugen.

Im Kapitel 6 werden die vorgestellten Algorithmen hinsichtlich ihrer Performanz analysiert. Der Arbeit liegt ein digitaler Anhang bei, welcher die implementierte Lösung enthält.

Kapitel 2

Das Ökosystems Korallenriff

Dieses Kapitel stellt dem Leser den Organismus Koralle näher vor und geht dabei auf seine spezifischen Merkmale als ortsgebundenes Lebewesen ein. Ferner wird auf die speziellen visuellen Effekte Unterwasser eingegangen. Prof. Dr. Helmut Schuhmacher definiert das Korallenriff aus biologischer und geologischer Sicht folgendermaßen:

> „Ein Riff ist eine maßgeblich von lebenden Organismen aufgebaute, meist bankförmige Struktur, die vom Meeresboden bis zur Wasseroberfläche reicht und so groß ist, dass sie erheblich die physikalischen und damit auch ökologischen Eigenheiten ihrer Umgebung beeinflusst. Ihre Konsistenz ist hinreichend fest, den anbrandenden Wasserkräften zu widerstehen und damit eine vieljährigen charakteristisch gegliederten Raum für spezifisch angepasste Bewohner zu bilden.“ (Schuhmacher, 1988, S. 12)

Er stellt heraus, dass Korallenriffe über spezifische Eigenheiten verfügen, bedingt durch die riffbildenden Organismen. So fungiert ein Riff beispielsweise als Wellenbrecher und schafft somit im sonst bewegten Ozean Brandungs- und Stillwasserbereiche (Schuhmacher, 1988).

2.1 Korallen

Ein Bewohner des Riffs, welcher in dieser Arbeit im Zentrum steht, ist die Koralle. Korallen sind sessile (ortsgebundene) Nesseltiere, welche oft Kolonien bilden. Die für die Riffbildung maßgebliche Gruppe sind die Steinkorallen. Diese Korallen besitzen die Fähigkeit Kalk abzuscheiden und so steinähnliche Strukturen zu bilden (Schuhmacher, 1988, S. 121). Abbildung 2.1 verdeutlicht die phänotypische Formvielfalt von Steinkorallen (Randall und Myers, 1983).

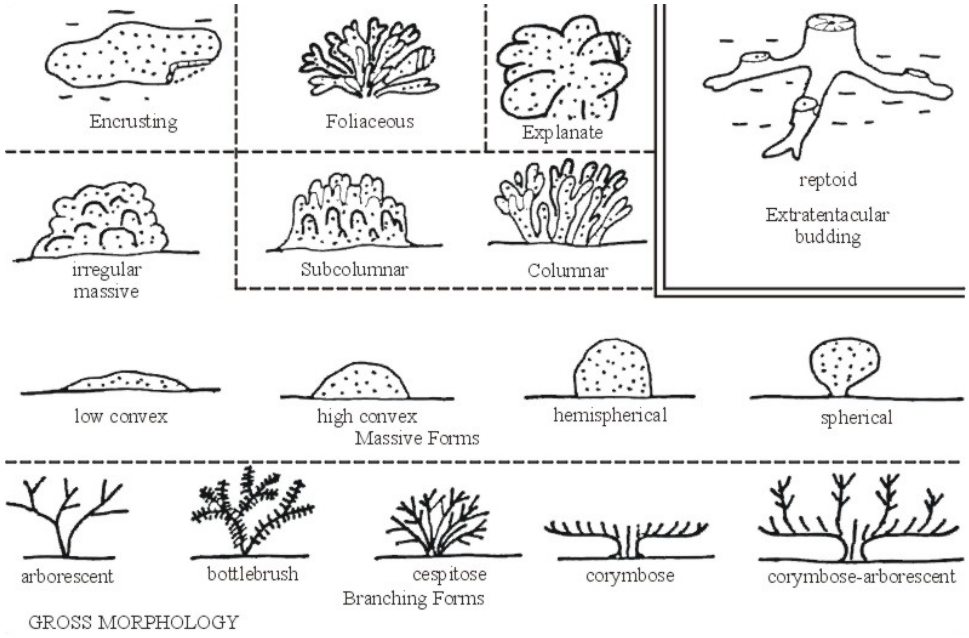


Abbildung 2.1: Formen von Korallen nach Randall und Myers (1983)

In dieser Arbeit stehen zwei Unterarten der Steinkorallen im Vordergrund: die massiven und die verzweigten Korallen. Die massiven Korallen zeichnen sich durch ihr kompaktes kuppelförmiges Aussehen und langsames Wachstum von nur wenigen Zentimetern pro Jahr aus. Beispiele für diese Gruppe sind *Porites Lutea* und *Porites Lobata*. Die verzweigten Korallen bilden feine Verzweigungen aus. Beispiele hier sind *Acropora Muricata* und *Pocillopora Damicornis*.

Steinkorallen leben in enger Symbiose mit einzelligen Algen zusammen, welche ihnen auch zu ihrer einzigartigen Farbe verhelfen. Die Koralle selbst ist eher farblos. Die Symbiose zwischen den Organismen Koralle und Alge ist sehr empfindlich. Wenn sich beispielsweise Einflussfaktoren, wie Temperatur oder Salzgehalt des Wassers ändern, kann es zu einer Stresssituation und letztlich zu der sogenannten Korallenbleiche kommen. Dabei stößt die Koralle die Alge ab und die symbiotische Gemeinschaft zerbricht. Resultierend daraus wird der übrig gebliebene Korallenstock farblich immer blasser. Bei einer länger anhaltenden Stresssituation verlassen alle Algen die Koralle. Diese stirbt daraufhin ab und es bleibt ein weißes Kalkskelett zurück.

2.2 Aspekte einer Unterwasser-Umgebung

Die Umgebung sowie physikalischen Bedingungen unter Wasser unterscheiden sich stark von den Gegebenheiten über Wasser. Zur Simulation einer Unterwasser-Welt, wurde zunächst in einer qualitativen Analyse von Filmmaterial die wichtigsten visuellen Effekte zusammengetragen:

- blau-grüne Umgebungsfarbe und schwebende Partikel
- Kaustiken am Untergrund und volumetrische Lichtbündelung („Godrays“)
- Tiere, welche im Riff leben (Fische, Krebse, ...)

Wasser ist nicht gleich Wasser. Die Zusammensetzung von Partikeln innerhalb des Wassers hat einen großen Effekt auf die Brechung des Lichts und somit auf die visuelle Wahrnehmung von Objekten unter Wasser. Der Farbton von Ozeanwasser kann beispielsweise von einem tiefen Blau bis hin zu einem gelb-braunen Farbton alles abdecken. Mobley (2001) fasst folgende Einflussfaktoren auf die Farbe und die Lichtstärke zusammen:

- Meerwasser hat eine starke Absorption bei einer Wellenlänge unter 250 nm und über 700 nm.
- Aufgelöste organische Verbindungen Überreste von Pflanzengewebe färben das Wasser in ausreichender Konzentration gelb-braun.
- Bakterien absorbieren und streuen vor allem blaues Licht in sauberem Ozeanwasser.
- Phytoplankton produziert Chlorophyll, welches blaues und rotes Licht absorbiert. Phytoplankton ist größer als die Wellenlängen von sichtbaren Licht und hat somit einen erheblichen Effekt auf dessen Streuung.
- Detritus sind tote organische Partikel, z.B. abgestorbener Phytoplankton und absorbiert hauptsächlich blaues Licht. Detritus ist der Hauptgrund für die Lichtstreuung im freien Ozean.
- Anorganische Partikel sind beispielsweise aufgewühlter Sand oder Staub. Die Größe variiert von 0.01 µm bis dutzende Mikrometer. In ausreichender Konzentration haben diese Partikel einen dominanten Einfluss auf die Trübung des Wassers.

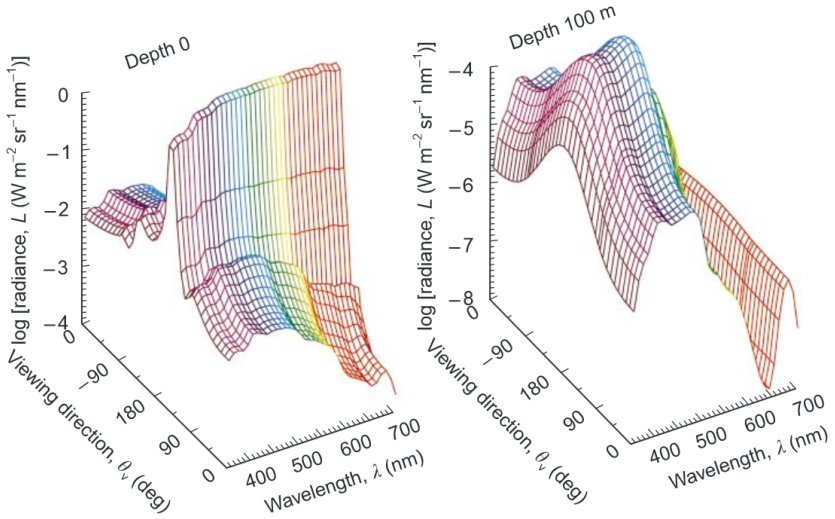


Abbildung 2.2: Wellenlängenzusammensetzung von ozeanischem Wasser bei einer Tiefe von 0 m (gerade unter der Wasseroberfläche) und 100m (Mobley, 2001)

Diagramm 2.2 zeigt die unterschiedliche Wellenlänge von ozeanähnlichem Wasser bei einer Tiefe von 0 m, direkt unter der Wasseroberfläche, und einer Tiefe von 100 m. Es wird deutlich, dass bei einer Tiefe von 100 m der Blauanteil des Lichts deutlich überwiegt. Dadurch werden die Farben von Objekten verfälscht. Beispielsweise erscheint rotes Blut grün. Allerdings muss das Licht sehr lange durch Wasser „gefiltert“ worden sein.

Während ein Lichtstrahl durch Wasser geht, wird er gedämpft und gestreut. Dämpfung führt zu einem Verlust von Photonen und Streuung zu einem Gewinn an Photonen. Beide Effekte haben Einfluss auf die Wellenlänge und somit einen visuellen Einfluss auf die Farbe von Objekten. Basierend darauf entsteht die typisch blau-grüne Umgebungsfarbe unter Wasser.

Ein Sonderfall in den Lichtverhältnissen stellt der Übergang von einem Medium in das nächste dar. In diesem Fall ist es das Licht der Sonne, welches aus dem Medium Luft in das Medium Wasser übergeht. Die Schnittstelle der Medien ist die Wasseroberfläche. Hier spielen vor allem die Brechung und Beugung des Lichts eine große Rolle. Zur Darstellung der Oberfläche selbst werden der Wellengang und die damit verbundene Deformation der Oberfläche mit einbezogen. Durch diese Deformation des transmittierenden Materials kommt es stellenweise zu einer Bündelung des Lichtes. Einerseits wird diese Bündelung als volumetrisches Licht im Wasser in Form von Strahlenbüscheln sichtbar. Andererseits ergeben sich durch Lichtbrechung erzeugte Muster auf dem Meeresboden, die sogenannten Kaustiken.

Die Korallen bieten Lebensraum für Unmengen an Krebstieren, Seeigeln und Fischen. Diese Tiere haben selten Tarnfarben sondern wirken durch den hohen Anteil von blauem und ultra-violettem Licht neon-bunt. Abbildung 2.3 zeigt exemplarisch ein Korallenriff. Deutlich wird die Trübung des Wassers und der Farb- und Variantenreichtum der Tierwelt.

Abbildung 2.3: Typische Szene in einem Korallenriff am Beispiel des Great Barrier Reef Boaden (2015)

Kapitel 3

Theorie der prozeduralen Pflanzen

Wie bereits in Kapitel 2.1 spezifiziert, sind Korallen ortsgebundene Lebewesen mit einem modularen Aufbau. Diese Eigenschaften macht den Phänotyp der Koralle dem von Pflanzen sehr ähnlich. Von daher soll an dieser Stelle ein Überblick über die etablierten Methoden aus der Computergrafik zur prozeduralen Pflanzengenerierung und -wachstum erfolgen. Generell lässt sich die Generierung einer virtuellen Landschaft in drei Abstraktionsstufen unterteilen (Deussen, 2002):

- Die Generierung der Umgebung
- Die Positionierung von Individuen
- Die Modellierung von Einzelpflanzen/-organismen

Je nach den Anforderungen an die jeweilige Abstraktionsebene kommen unterschiedliche Modellierungswerkzeuge zum Einsatz. Bei der Generierung der Umgebung liegt ein Hauptaugenmerk auf der immersiven Wirkung auf den Betrachter. Der Betrachter soll das Gefühl haben, er ist ein Teil der virtuellen Umgebung. Für die Einzelpflanze steht die geometrische Detailtreue im Vordergrund. Bei einer Pflanzenpopulation, also der Ansammlung von Pflanzen einer Art, welche sich zu Pflanzengesellschaften verschiedener Art zusammensetzen, ist vor allen die statistische Zusammensetzung von Bedeutung. Ferner gibt es zwischen den Einzelpflanzen Interaktionen wie Beschattung oder Konkurrenz, welche ein einzigartiges Bild erzeugen. In größeren Arealen ist die Einzigartigkeit eines Individuum weniger wichtig als in kleinen Gruppen, wo dem Betrachter Dopplungen sofort auffallen.

3.1 Spezifikation der Korallenverteilung

Zur Beschreibung von Populationen einzelner Pflanzen sind statistische Modelle notwendig. Generell unterscheidet man reguläre sowie zufallsbasierte Verteilungen. Die regulären Verteilungen kommen meist nur durch menschliche Einflussnahme zustande, beispielsweise in Gärten oder Plantagen. Zufallsverteilungen sind häufiger in der freien Natur zu beobachten. Charakterisiert werden sie ferner durch die Ausbreitungsmechaniken der Pflanze, Interaktion mit anderen Pflanzen, den Kampf um Ressourcen und lokale Begebenheiten (Deussen, 2002, Seite 39).

Ein zufallsbasierte Verteilung für Korallen beschreibt Kubicek et al. (2012) in seinem Korallenriffmodell SICCOM (spatial interaction in coral reef communities). Dieses komplexe Modell bietet die Möglichkeit das Wachstum unterschiedlicher Korallen und Algenarten zu simulieren. Jeder Organismus in diesem Model stellt dabei ein Individuum dar, welches mit anderen Individuen interagiert und durch Einflussfaktoren wie Störungen, Nahrungsverfügbarkeit und Temperatur geprägt ist. Dies wird visualisiert durch Abbildung 3.1. Die Störereignisse, welche in der Simulation auftreten haben mechanische Auswirkungen auf das Riff und töten alle Organismen innerhalb der betroffenen Region. In der Simulation sind zwei unterschiedliche Störereignisse modelliert: kleinere Störungen (2-4 m im Durchmesser) und größere Störereignisse (5-10 m im Durchmesser). Die Frequenz, wann diese Ereignisse auftreten ist parametrierbar.

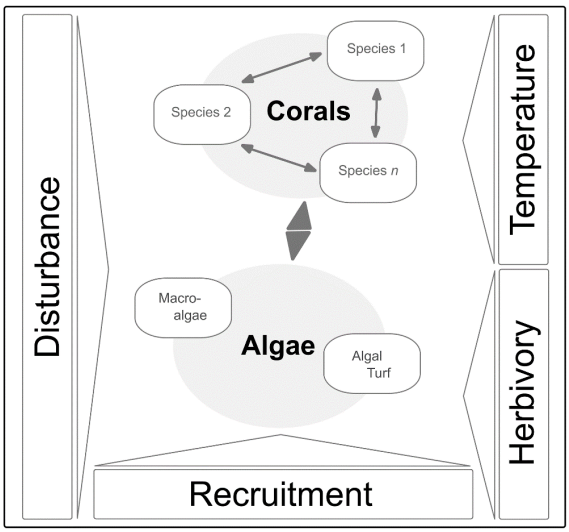


Abbildung 3.1: Schematische Darstellung des Korallenriffmodell von Kubicek et al. (2012). Verschiedene Korallenarten interagieren miteinander und mit zwei unterschiedlichen Algenarten. Von außen wirken Einflussfaktoren wie Störungen, Temperatur und Nahrungsverfügbarkeit.

Zu jedem Zeitschritt wird die aktuelle Population aus Algen und Korallen iterativ berechnet. Korallen werden durch einen detaillierten Lebenszyklus beschrieben (vgl. Abbildung 3.2). Dieser Zyklus unterscheidet sich je nach Korallenart beispielsweise in Wachstumsmuster und -raten. Die virtuelle Koralle wächst mit einer konstanten Rate ausgehend von ihrem Mittelpunkt in 24 Radien, welche die Ausrichtungen in alle Richtungen repräsentieren. Im weiteren werden diese Radien als Zweige bezeichnet. Das Wachstum eines einzelnen Zweiges ist bedingt durch die Interaktion mit benachbarten Organismen, der individuellen Fitness der Koralle sowie dem definierten Wachstumsmuster. Dadurch entsteht für jede Koralle eine individuelle Grundform.

Zunächst wird im Zyklus die Temperatur als einer der wichtigsten Einflussfakturen überprüft. Bei ungünstigen Verhältnissen kommt es zu einer Bleiche, welche im Tod des Tieres enden kann. Allerdings ist es auch möglich, dass sich die Koralle von einer Bleiche erholt. Das lebende Individuum interagiert mit den umgebenden Korallen und Algen. Anschließend setzt die Wachstumsphase und ggf. die Reproduktionsphase ein. Generell kommt es ein oder zwei mal im Jahr zu einer Reproduktionsphase, welche durch den Mondzyklus und kleineren Schwankungen in der Wassertemperatur ausgelöst wird. Der Tod einer Koralle nur bedingt durch Störereignisse, Bleichung oder als Resultat von konkurrierenden Interaktionen ein.

Die in der Simulation verwendete Macroalge ist *Sargassum ilicifolium*. Diese Algenart wächst 30 cm pro Monat und erreicht eine maximale Höhe von 60 cm sowie einen maximalen Durchmesser von 45 cm. Der Lebenszyklus der Makroalge findet in wesentlich kürzeren Zeitspannen statt als der Zyklus einer Koralle. Im Gegensatz zu den Korallen können Algen am Alter sterben und werden nach einer gegeben Zeit aus der Simulation entfernt. Die virtuellen Algen vermehren sich ebenfalls einmal im Jahr. Zur Vermeidung von Seiteneffekten am Riffrand, gibt es in der Simulation Algen außerhalb des virtuellen Riffs. Im Korallenriff treten weiterhin fadenförmiger oder überkrustende Algen auf. Diese werden in der Simulation als Turf zusammengefasst und als Grid mit quadratischen Zellen abstrahiert.

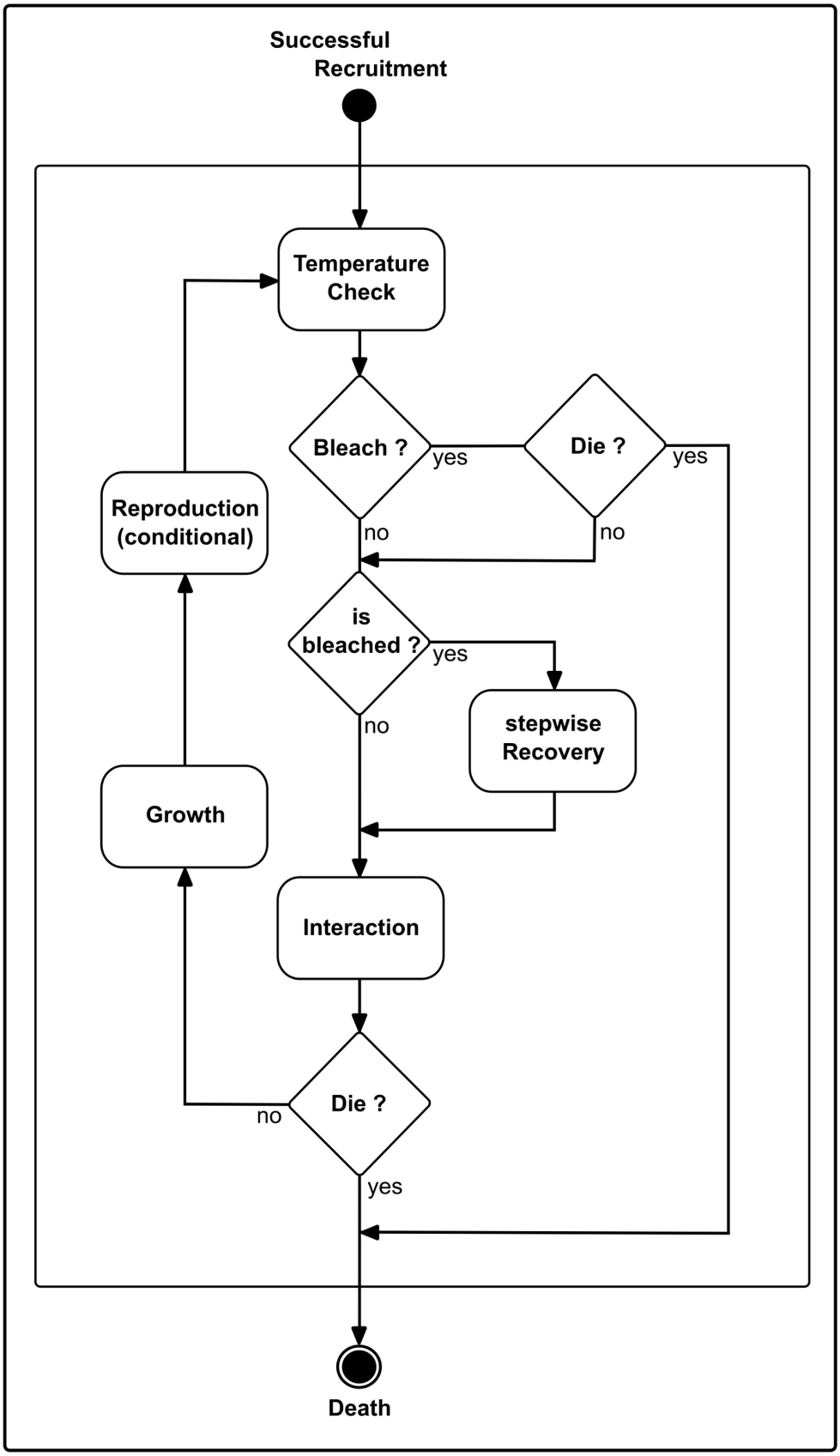


Abbildung 3.2: Lebenszyklus einer Koralle nach Kubicek et al. (2012)

Diese SICCOM-Simulation ist in Java mit dem MASON Framework (Luke et al., 2005) umgesetzt worden und hat ein zwei-dimensionales schematisches Frontend. Abbildung 3.3 zeigt einen Screenshot der graphischen Benutzeroberfläche. Deutlich wird unter anderem, dass die Korallen teilweise ineinander wachsen und die Grundform der Individuen an die umgebenden Korallen anpassen und somit selten kreisförmig sind. Der Screenshot zeigt ferner Flächen an denen Störungen aufgetreten sind. Hier gibt es keine lebenden Korallen mehr und die Fläche ist von Algen bedeckt. In der Realität wäre je nach aufgetretener Störung an diesen Stellen die abgestorbenen Korallenskelette sichtbar. In einem langwierigen Prozess würden diese teilweise zerfallen.

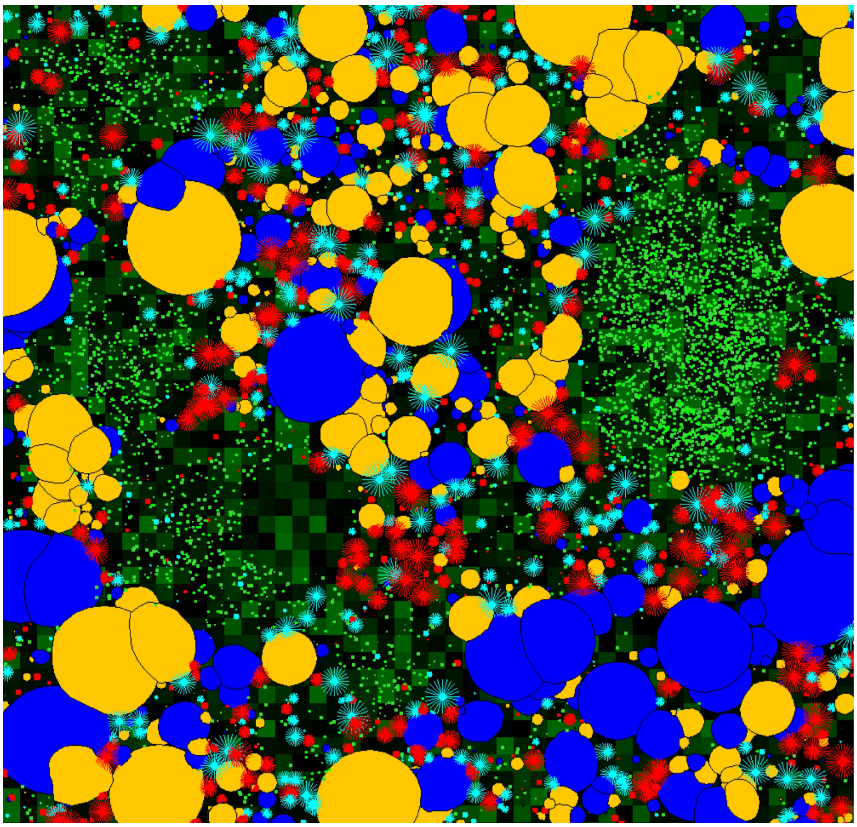


Abbildung 3.3: Screenshot der graphischen Benutzeroberfläche von SICCOM. Die Korallenkolonien der massiven Korallen *Porites Lutea* (orange) und *Porites Lobata* (blau) und der verzweigten Korallen *Acropora Muricata* (rot) und *Pocilopora damicornis* (cyan) sowie der Algen (grün) konkurrieren um Raum miteinander. Der freie Raum, welcher nur durch Makroalgen und Turf bedeckt ist, markiert Flächen an denen Störungen aufgetreten sind (Kubicek et al., 2012).

3.2 Die Modellierung von Einzelorganismen

An dieser Stelle wird die Generierung einer virtuellen Koralle auf einige wesentliche Teilaufgaben reduziert. Zunächst müssen die Position und die Ausdehnung des Organismus im Raum gegeben sein. Diese Aufgabe wurde bereits in Kapitel 3.1 abgehandelt. Auf Basis dieser Information wird ein morphologisches Skelett erzeugt. Damit das Skelett ein Volumen erhält, wird zunächst ein skalares Feld erstellt und anschließend dieses polygonisiert, sprich in ein Polygonnetz überführt.

3.2.1 Beschreibung des morphologischen Skeletts

Eine Variante das morphologische Skelett abzubilden sind Fraktale (Mandelbrot, 1977). Eine wesentlicher Eigenschaft der Fraktale nach Mandelbaum ist die Selbstähnlichkeit eines Objektes, sprich ein kleiner Teil einer Pflanze (Ast oder Zweig) sieht ähnlich aus wie die gesamte Pflanze.
Auf Grund der Selbstähnlichkeit fraktaler Objekte sind deren Implementation rekursiver Natur. Ein populärer Ansatz zur Beschreibung sind die L-Systeme, entwickelt 1968 von dem Botaniker Aristid Lindenmayer (Lindenmayer, 1968a,b, 1971). L-Systeme sind grammatikalische Zeichenketten-Ersetzungssysteme. Das wesentliche Prinzip besteht in der sukzessiven Ersetzung von Einzelteilen mittels Produktionsregeln. In der einfachsten Form beschreibt das L-System eine kontextfreie Grammatik (0L-System). Die Menge der Produktionsregeln ist endlich und bei der Ersetzung eines Literals wird kein Bezug auf umliegende Literale genommen.

Formal beschrieben wird das 0L-System durch ein Tupel (Prusinkiewicz und Lindenmayer, 1996):

$$G = (V, w, P) \tag{3.1}$$

V ... beschreibt ein Alphabet, wobei V^* die Menge aller Wörter aus V ist und V^+ die Menge aller nicht-leeren Wörter aus V ist.

w ... $w \in V^+$ ist ein nicht-leeres Wort, welches das Startwort repräsentiert.

P ... $P \subset V \times V^*$ bezeichnet eine endliche Menge an Produktionsregeln.

Eine Produktionsregel der Form $(a, x) \in P$ wird geschrieben als $a \to x$, wobei a als Vorgänger und x als Nachfolger bezeichnet wird. Für jedes Zeichen $a \in V$ gibt es mindestens ein Wort $x \in V^*$ welches sich daraus ableitet. Wird kein Nachfolger explizit definiert, so gilt $a \to a$. Ein L-System ist deterministisch, wenn für jedes $a \in V$ genau ein $x \in V^*$ existiert (D0L-System). Eine

Ableitung eines Wortes in einem L-System kann durch Anwenden der Produktionsregeln für jedes Literal erzeugt werden. Aus dem generierten Wort kann wiederum eine Ableitung erzeugt werden (Iteration). Die Folge von n Ableitungen wird als Sequenz spezifiziert.

Als Beispiel nennt Prusinkiewicz ein L-System zur Beschreibung der blaugrünen Algenart *Anabaena catenula*, einer Schwingfadenalge. Die Form der Alge lässt sich nur unter dem Mikroskop beobachten und ähnelt einer Reihe unterschiedlich langer Zylinder (vgl. Abb. 3.4 (a)). Die Zylinder sind zwei verschiedene Bakterientypen, die sich in der Größe sowie dem Teilungsverhalten unterscheiden. Beide Bakterientypen sind asymmetrisch aufgebaut, somit können sie in der Kette entweder nach links oder nach rechts orientiert sein. Das L-System zur Beschreibung des Wachstums der Alge beschränkt sich auf nur eine räumliche Dimension. Demnach gibt es vier unterschiedliche Elemente $V = \{\overleftarrow{A}, \overrightarrow{A}, \overleftarrow{B}, \overrightarrow{B}\}$. Graphisch werden diese Elemente wie in Abbildung 3.4 (b) interpretiert.

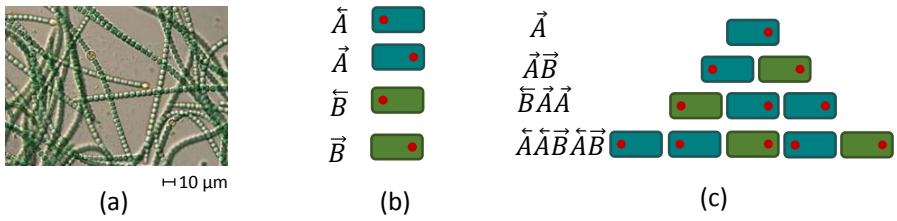


Abbildung 3.4: (a) Mikroskopische Fotographie der Alge *Anabaena catenula* (Freshwater Algae Culture Collection at the Institute of Hydrobiology, 1984), (b) Graphische Interpretation nach Peter und Eyer (2001), (c) Iterationen des D0L-Systems zur Beschreibung des Wachstums

Nach einer gewissen Zeit teilen sich die Zellen nach gegebenen Produktionsregeln:

p_1: $\overrightarrow{A} \rightarrow \overleftarrow{A}\,\overrightarrow{B}$

p_2: $\overleftarrow{A} \rightarrow \overleftarrow{B}\,\overrightarrow{A}$

p_3: $\overrightarrow{B} \rightarrow \overrightarrow{A}$

p_4: $\overleftarrow{B} \rightarrow \overleftarrow{A}$

Bei einem gegebenen Bakterium, dessen Aufbau beschrieben wird mit $w = \overrightarrow{A}$, zeigen sich die unterschiedlichen Iterationen in Abbildung 3.4 (c).

Die graphische Interpretation der Alge *Anabaena catenula* ist recht simpel. Jedem Zeichen wird eine Grafik zugewiesen. Ein aufwendigeres L-System in einem zweidimensionalen Raum verlangt eine komplexere Bildbeschreibungssprache. Für diese Aufgabe hat sich die sogenannte Turtle-Grafik durchgesetzt (Abelson und diSessa, 1986). Ein imaginärer stifttragender Roboter (bildlich eine Schildkröte, engl. „turtle“) bewegt sich auf einer Zeichenebene und erzeugt mit einfachen Kommandos ein Bild. Der aktuelle Status der Turtle ist mit einem Tupel (x, y, α) definiert, wobei x, y die räumliche Position und α die Blickrichtung der Schildkröte beschreibt. Mit einer gegebenen Schrittweite d und einem Winkelinkrement λ reagiert die Schildkröte entsprechend einer vorgegebenen Aktionsmenge A. Dieses Verfahren wird in Abbildung 3.5 visualisiert.

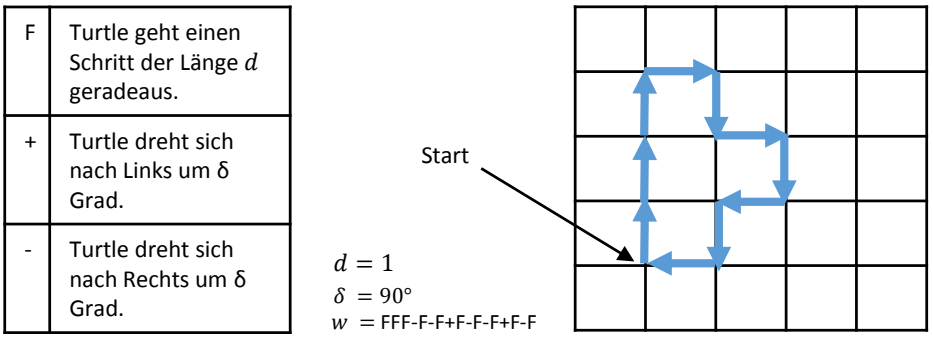


Abbildung 3.5: Turtle-Aktionsmenge und Interpretation eines Strings - initial steht die Turtle auf Start und blickt nach oben

Prusinkiewicz und Lindenmayer (1996) hat das L-System auf unterschiedliche Arten weiterentwickelt. Im Nachfolgenden werden die Weiterentwicklungen zum Erreichen eines geklammerten, stochastischen, parametrischen und umgebungssensitiven L-System erläutert.

Geklammertes L-System: Die D0L-Systeme können nur Linienzüge mit einem Start und einem Ende darstellen. Zur Darstellung von Verzweigungen wird das Alphabet um Klammer-Parameter erweitert. Die geöffnete Klammer definiert, dass die aktuellen Attribute der Turtle auf einem Stapelspeicher abgelegt werden sollen. Die geschlossene Klammer liest die obersten Werte vom Stapelspeicher aus und setzt die Turtle auf diese Position zurück. Die Abbildung 3.6 zeigt verschiedene geklammerte L-Systeme.

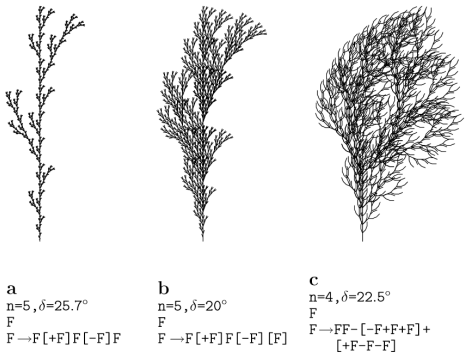


Abbildung 3.6: Beispiele für Bracketed 0L-Systeme (Prusinkiewicz und Lindenmayer, 1996, Seite 25)

Stochastisches L-System: Alle Objekte, die mit demselben D0L-System erzeugt werden sind identisch. Da diese Gleichmäßigkeit bei Pflanzen sehr realitätsfern ist, wird das L-System um stochastische Parameter erweitert. Es wird sichergestellt, dass diese Parameter nur kleinere Abweichungen zulassen und nicht das Aussehen des L-Systems komplett ändern. Das stochastischen L-Systemen wird repräsentiert durch $G_\Pi = (V, w, P, \Pi)$ (Prusinkiewicz und Lindenmayer, 1996, Seite 28). Die Funktion $Pi : P \to (0, 1]$ bildet die Wahrscheinlichkeitsverteilung ab, dass eine Produktionsregel ausgewählt wird. Es wird davon ausgegangen, dass die Summe aller Wahrscheinlichkeiten der Produktion eines Literals a gleich 1 ist. Abbildung 3.7 zeigt mögliche Individuen eines stochastischen L-Systems.

Abbildung 3.7: Verschiedene Individuen eines stochastischen L-Systems (Prusinkiewicz und Lindenmayer, 1996)

Parametrisches L-System: Das bislang definierte D0L-System lässt für alle Rotationsmöglichkeiten immer nur denselben Winkeln λ zu. Auch die Länge d eines Zweig ist innerhalb einer Tiefe gleich. Für eine höhere Varianz nehmen die Winkel und Zweiglängen unterschiedliche Werte an. Prusinkie-

wicz erreicht dieses Ziel mit den parametrischen L-Systemen. Das System wird um einer Menge von formalen Parametern Σ erweitert: $G = (V, \Sigma, w, P)$ mit der formalen Beschreibung der Produktionsregeln $p : a(t) : condition(t) \rightarrow x$. Die möglichen Vorgänger werden als Module bezeichnet. Ein Regel trifft auf ein Modul zu, wenn die Variable a und die Menge von Parametern t übereinstimmt. Außerdem muss die Bedingungsfunktion $condition(t)$ einen wahren Wert zurückgeben.

Umgebungssensitives L-System: Mit Bezug auf die Umgebung ermöglicht das umgebungssensitve L-System eine Struktur vorzugeben, innerhalb welcher der Organismus sich ausdehnen darf. Vor jeder Iteration wird über spezielle Kommunikationsmodule die Umgebung erfasst. Über ein parametrisches L-System werden die Einschränkungen an das System weitergeben und somit die Regeln ausgewählt.
Generell gibt es dabei zwei Herangehensweisen. Einerseits kann der L-System Formalismus erweitert werden und die Umgebung direkt mit einbeziehen. Andererseits ist der Organismus scharf von der Umgebung getrennt und beide werden als eigene parallele, ständig kommunizierende Prozesse implementiert.

3.2.2 Erzeugung des skalaren Felds mit Metaballs

Ein Ansatz zur Generierung eines volumetrischen Objektes zur Laufzeit ist die implizite Oberfläche (Hansen und Johnsen Chris, 2004). Implizite Flächen sind durch folgende mathematische Form definiert:

$$p \in \mathbb{R}^3 : f(p) = 0 \tag{3.2}$$

Diese Gleichung kann wie folgt interpretiert werden: Ist die Funktion f für p gleich Null, so liegt p auf der impliziten Oberfläche. Die Funktion teilt den Raum in zwei Halbräume:

$$H_1 = \{p \in \mathbb{R}^3 : f(p) < 0\} \tag{3.3}$$

$$H_2 = \{p \in \mathbb{R}^3 : f(p) > 0\} \tag{3.4}$$

Definiert man den Halbraum H_1 als Volumen innerhalb des Objektes, so beschreibt der Halbraum H_2 das Volumen außerhalb des Objektes. Eine spezielle Form der impliziten Fläche ist die Isofläche. Jedem Punkt im Raum wird dabei zunächst mittels einer Potentialfunktion $f_p(x, y, z)$ ein skalarer Wert zugeordnet. Die Menge aller skalaren Werte heißt skalares Feld φ. Beispielsweise

lässt sich dafür die Dichte, Temperatur oder elektrische Feldstärke verwenden. Abbildung 3.8 zeigt solch ein zweidimensionales Feld aus dem Bereich der numerischen Strömungsmechanik. Zu sehen ist die Umströmung von drei Zylindern. Die Farben repräsentieren den skalaren Wert an einer Position, hier der Betrag des Geschwindigkeitsvektors U oben und die Ionendichte p_E unten.

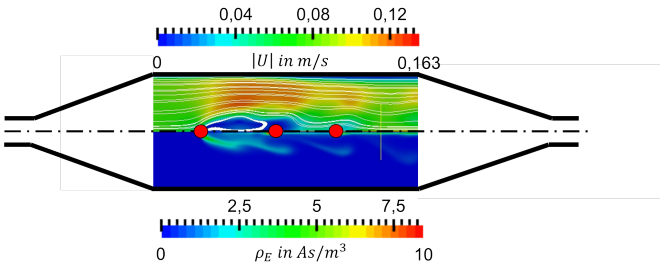


Abbildung 3.8: skalares Feld zur Visualisierung des Betrags des Geschwindigkeitsvektors U und der Ionendichte p_E in der numerischen Strömungsmechanik (Sander, 2014)

Wenn in dem skalaren Wert ein Abstand zum Mittelpunkt oder ein elektrisches Potenzial kodiert ist, dann lassen sich aus dem skalaren Feld auch Oberflächen erzeugen. Dazu wird ein Schwellwert (Isowert $I \in \mathbb{R}$) definiert, welcher die Stelle der Trennung der beiden Halbräume ist. Somit ist eine Isofläche S_c zu diesem Skalarfeld φ durch die Menge $S_c = \{\mathbf{v} \in \mathbb{R}^3 | \varphi(\mathbf{v}) = I\}$ definiert.

Komplexere Objekte können durch Überblendung verschiedener Potentialfelder erreicht werden. Blinn (1982) stellt die Methode Metaballs zur Generierung von Isoflächen vor, welche optisch der Idealvorstellung von Molekülen ähneln: Es gibt mehrere Grundformen, welche ineinander verschmelzen. Jede Grundform ist definiert durch sein Potentialfeld. Abbildung 3.9 zeigt ein Metaball-Objekt, welches aus einer Menge von kreisförmigen Potentialfeldern besteht. Dieses Grundprinzip wurde auch von anderen Autoren umgesetzt und ist nun unter verschiedenen Namen bekannt (vgl. Tabelle 3.1).

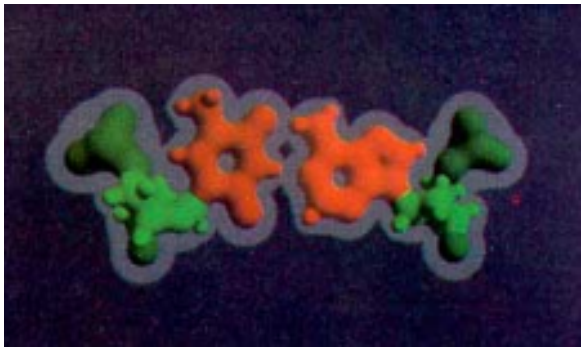


Abbildung 3.9: Metaballs von Blinn (1982)

Zur Beschreibung des skalaren Feldes werden die Potentiale der einzelnen Skelettknoten entsprechend Formel 3.5 anteilig aufaddiert.

$$F = \{P(X) = I | X \in \mathbb{R}^3, P(X) = \sum a_i p(X)\} \tag{3.5}$$

Die Wichtung a_i gibt an, wie stark das Potenzial eines Skelettknotens in die Berechnung des skalaren Feldes mit eingehen soll. Für die Potenzialfunktion $p(X)$ konnten in der Literatur verschiedene Ansätze gefunden werden (vgl. Tabelle3.1, de Araújo et al. (2015)).

Tabelle 3.1: Potentialfunktionen zur Ermittlung des skalaren Feldes aus Skelettknoten

Bezeichnung	Formel
blobby molecules (Blinn, 1982)	$p(X) = I \exp\left(\frac{B_i}{R_i^2} d(X)^2 - B_i\right)$
soft-objects (Wyvill et al., 1986)	$p(X) = -\frac{4}{9}(\frac{r}{R_i})^6 + \frac{17}{9}(\frac{r}{R_i})^4 - \frac{22}{9}(\frac{r}{R_i})^2 + 1$
Metaballs (Nishimura et al., 1985)	$p(X) = \begin{cases} 1 - 3(\frac{r}{R_i})^2 & , 0 \leqslant r \leqslant \frac{R_i}{3} \\ \frac{3}{2}(1 - \frac{r}{R_i})^2 & , \frac{R_i}{3} \leqslant r \leqslant R_i \\ 0 & , r > R_i \end{cases}$
BlobbyModel (Muraki, 1991)	$p(X) = \begin{cases} (1 - (\frac{r}{R_i})^2)^2 & , 0 \leqslant r \leqslant R_i \\ 0 & , r > R_i \end{cases}$

B_i ... „Blobbiness“-Parameter, welcher angibt wie weich die Übergänge sein sollen

R_i ... Effektradius des Potentialfeldes eines Knotens

$d(P)$ ... Entfernung des Mittelpunkt eines Skelettknotens zum Punkt

Das Konzept der Metaballs ist in der Computergrafik zu einem gängigen Modellierungswerkzeug geworden. Es eignet sich unter anderem zur Modellierung von Fluiden (Nakata und Sakamoto, 2015) und organischen Formen wie Menschen (Lewis und Parent, 2000), Adern (Oeltze und Preim, 2004) und Terrain (Yu und Wang, 2014).

3.2.3 Polygonnetz-Erzeugung mit Marching Cubes

Um ein Polygonnetz auf einer impliziten Fläche zu erzeugen gibt es verschiedene Ansätze. Der gängigste Algorithmus zur Polygonnetz-Erzeugung aus der Isofläche ist der Marching Cubes Algorithmus beschrieben durch Lorensen und Cline (1987). Dieser Ansatz findet Anwendung in vielen Bereichen, unter anderem der Biochemie (z.B. Heiden et al. (1993)), der Medizin (z.B. Yim et al. (2003)), Polygonnetzgenerierung aus Tiefenwerten (Rock et al., 2015) sowie Umweltsimulationen wie Wolken (Trembilski, 2001) und Terrain (Lengyel, 2010).

Die Idee von Marching Cubes ist es, den Raum des Objekts zunächst in kleinere Würfel (Voxel/Cubes) zu zerlegen. Lorensen und Cline (1987) teilt den Raum dazu zunächst in Scheiben ein. Auf zwei Scheiben werden jeweils vier Referenzpunkte definiert, welche als Eckpunkte für den Würfel dienen. Im Algorithmus wird dann der Objektraum scheibenweise durchlaufen und jeder Referenzpunkt, dessen skalarer Wert über einen vorher definierten Isowert liegt, wird markiert. Die Kanten an einem markierten und einem nicht-markierten Referenzpunkte sind die Kanten, an der die Oberfläche des Objektes den Würfel durchschneidet. Da jeder der acht Eckpunkt entweder markiert oder unmarkiert sein kann, gibt es 256 (2^8) Möglichkeiten, wie genau ein Würfel polygonisiert wird. Jede dieser Varianten ist als ein spezifisches Schnittmuster definiert und wird vorher einer Tabelle hinterlegt. Während des Algorithmus wird das spezifische Schnittmuster in der Tabelle nachgeschlagen und somit mit nur wenigen Schritten die durchschnittenen Kanten ermittelt. Durch lineare Interpolation zwischen den Skalarwerten zweier Nachbarn wird die genaue Position des Schnittpunkts auf einer Kante berechnet.

Durch Ausnutzen von Rotationssymmetrie und Spiegeln, kann Lorensen und Cline (1987) die Anzahl der Schnittmuster von 256 auf 15 reduzieren (vgl. Abbildung 3.10). Unter anderem beschreibt Gelder und Wilhelms (1994) eine Variante, welche durch Punktsymmetrie mit nur 14 Schnittmuster auskommt. Mit diesen Verfahren geht allerdings ein erhöhter Rechenaufwand beim Durchlaufen der Voxel einher.

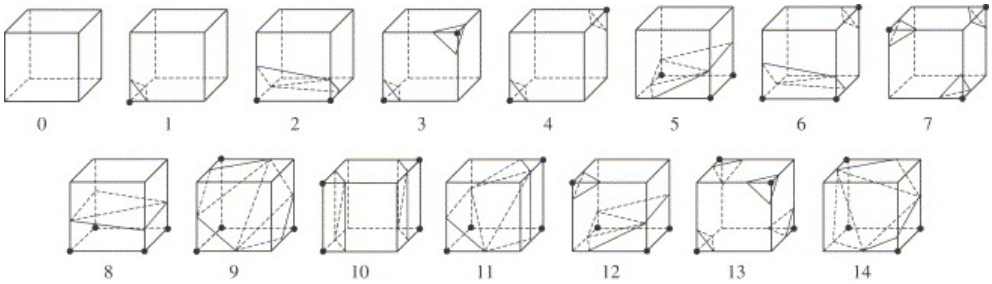


Abbildung 3.10: Die 15 möglichen Schnittmuster eines Voxels im dreidimensionalen Raum nach Lorensen und Cline (1987) durch Ausnutzen von Rotations- und Spiegelsymmetrie, Bildquelle: Newman und Yi (2006)

Ein wesentliches Problem, welches beim Marching Cubes Algorithmus auftritt, ist, dass einige Schnittmuster mehrdeutig sein können. Abbildung 3.11 zeigt so einen Fall im zweidimensionalen Raum. Für ein gegebenes Schnittmuster kommen zwei mögliche Polygonnetze in Frage. Die mehrdeutigen Fälle treten allerdings nur auf, wenn die Seitenlänge des Voxels größer ist, als die dünnste Stelle des Polygonnetzes. Zur Lösung dieses Problems stellt Treece et al. (1999) die Erweiterung Marching Tetrahedron vor. Bei dem Marching Tetrahedron Ansatz wird der einzelne Voxel weiterhin in sechs Tetrahedra unterteilt. Bei der Polygonisierung werden die Vertices, statt auf den zwölf Kanten nun auf den neunzehn Kanten angeordnet. Dadurch kann mit weniger Vertices ein genaueres Polygonnetz erzeugt werden (ca. 70% weniger Vertices). Allerdings wird der Algorithmus auch rechen- und speicherintensiver, da mehr Kanten geprüft werden müssen.

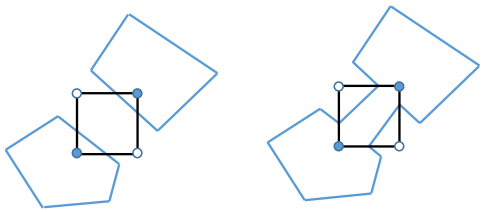
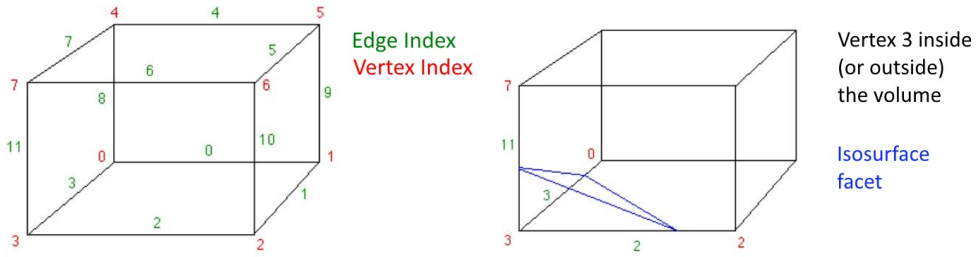


Abbildung 3.11: Mehrdeutigkeit eines Schnittmusters im zweidimensionalen Raum

Durch die Eigenart des Marching Cubes Algorithmus kann es ferner zu überflüssigen Dreiecken kommen, die beispielsweise sehr klein oder degeneriert sind. Generell ist das Polygonnetz gleichbleibend aufgelöst und passt sich nicht adaptiv an die Feinheit der skalaren Daten an. Verschiedene Ansätze verbessern die Qualität des resultierenden Polygonnetz und reduzieren teilweise auch die Polygonzahl ohne Genauigkeit zu verlieren (Moore und Warren, 1991; Raman und Wenger, 2008; Schreiner et al., 2006).

Bourke (1997) stellt eine Implementation des Marching Cubes vor, welches hier als Grundlage für die Implementation dienen soll. Ein Würfel wird dabei lokal durch Indices für die Kanten und Eckpunkte und einem Skalarwert entsprechend Abbildung 3.12a beschrieben. Wenn beim Durchlaufen beispielsweise der Skalarwert des Vertex 3 unterhalb des Isowertes liegt und alle anderen Skalarwerte oberhalb, dann wird ein Dreieck erzeugt, welches die Kanten 2,3 und 11 schneidet. Abbildung 3.12b zeigt das resultierende Polygonnetz für diesen Würfel. Dieses Verfahren wird für alle übrigen Voxel gleichermaßen ausgeführt. Das Verfahren nach Bourke (1997) unterscheidet sich zum Verfahren nach Lorensen und Cline (1987) in wenigen Punkten. Der Objektraum wird nicht in Scheiben unterteilt sondern direkt in Würfeln durchlaufen. Außerdem gibt es 256 Schnittmuster und nicht die reduzierte Variante von 15 Schnittmustern.



(a) Indizierung eines Voxels (b) erzeugtes Polygonnetz

Abbildung 3.12: Repräsentation eines Voxels im Marching Cubes Algorithmus nach Bourke (1997)

Der Marching Cubes Algorithmus wurde in vielerlei Hinsicht erweitert und lässt sich nun in verschiedenen Varianten finden. Eine Möglichkeit den Algorithmus zu beschleunigen ist es, unnötige Operationen in leeren Voxeln zu vermeiden. Laut Gelder und Wilhelms (1994) sind ca. 30 - 70 % der Zellen leer, somit ist das Einsparpotential hier besonders hoch. Generell gibt es drei Ansätze: Hierarchisch, Intervall-basiert oder Propagations-basiert (Newman und Yi, 2006). Ein Beispiel für den hierarchischen Ansatz ist es den Objektraum in einem Octree zu unterteilen (Wilhelms und Van Gelder, 1992). Der Octree ist eine Baumstruktur, dessen Knoten jeweils acht direkte Nachfolger haben oder gar keinen Nachfolger. Der Wurzelknoten des Octrees referenziert das gesamte Volumen. Jeder Kindknoten referenziert ungefähr gleich große Teilvolumen des Volumens des Vaterknotens. Die Blattknoten enthalten die einzelnen nicht-leeren Voxel. Um diese Einteilung vorzunehmen muss im Preprocessing jeder Voxel einmal durchlaufen werden. Anstelle anschließend die Voxel sequentiell alle zu durchlaufen, werden die Blattknoten des Octrees

durchlaufen. Somit werden leere Voxel nicht berechnet. Weiterhin bietet ein Octree die Möglichkeit Teile Objektes nicht zu berechnen, da sie beispielsweise verdeckt sind (Scholz et al., 2015). Je nach Algorithmus und Datengrundlage kann sich dieser zusätzliche Aufwand lohnen. Bei Propagations-basierte Ansätze wird der Objektraum nicht sequentiell abgelaufen. Stattdessen werden beginnend von einem Voxel umliegenden Voxel traversiert. Dabei wird davon ausgegangen, dass wenn der Startvoxel Teil des Objektes ist, so sind umliegende Voxel eher wahrscheinlich Teil des Objektes. Bei der Wahl des Startvoxels wird beispielsweise ein lokales Maximum verwendet (Itoh und Koyamada, 1995) oder die Nachbarschaft von Voxel ausgenutzt (Bajaj et al., 1996). Bei Intervall-basierten Ansätzen werden die Voxel nicht nach räumlichen Aspekten durchlaufen sondern auf Basis des minimalen und maximalen Skalarwertes des Voxels. Diese Werte werden mit einer Referenz auf den Voxel in einer Datenstruktur abgelegt. Ein Ansatz dazu ist das span filtering (Gallagher, 1991). Dabei wird der skalare Raum in gleichmäßige Unterräume unterteilt. Für jeden Unterraum wird eine Liste mit allen referenzierten Voxels verwaltet. Mit dem Bucket-Sort-Algorithmus werden diese Listen gefüllt. Beim Durchlaufen werden nur die Listen beachtet, dessen Intervall oberhalb des Isowertes beginnen.

Durch die Wahl der Größe eines Voxels lässt sich die Auflösung des Polygonnetzes definieren. Dadurch kann auf Basis eines Skalarfeldes das Objekt mit unterschiedlichen Auflösungen generiert werden. Dies kann beispielsweise für LOD-Beschleunigungsverfahren interessant sein. Weber et al. (2001) stellen eine Variante des Algorithmus vor, welche mit einer Hilfe einer Hierarchie verschieden aufgelöste Polygonnetze erzeugt.

Ferner gibt es Ansätze ein Polygonnetz zu generieren, welches auf skalaren Werten basiert, die sich über die Zeit ändern. Typischerweise wird der Algorithmus zu jedem Zeitschritt neu erzeugt (Tory et al., 2001). Andere Ansätze repräsentieren die Daten in einem vierdimensionalen Feld, wobei die Zeit die vierte Dimension darstellt (Weigle und Banks, 1998). Das Isovolumen wird durch die implizite Form $f(x, y, z, t) = \alpha$ dargestellt, wobei t die Zeit ist und α der Isowert. Zu jedem Zeitpunkt kann der aktuelle Isowert interpoliert werden. Allerdings steigt mit der Anzahl der Zeitschritte auch die Speichermenge für das Polygonnetz. Dadurch dass jeder Voxel im Marching Cubes Algorithmus unabhängig seiner Nachbarn berechnet werden kann, eignet sich der Algorithmus gut für eine Parallelisierung auf der CPU (Miguet und Nicod, 1995; Gerstner und Rumpf, 2000) oder Auslagerung auf die GPU (Nguyen, 2007; Ulrich et al., 2014; Dingliana und Ganovelli, 2005).

Kapitel 4

Ein Framework für 3D-Riffsimulationen

Als Grundlage der Simulation dient ein eigens konzeptioniertes Framework. Bei der Wahl der Grafik-Engine kamen zum Startzeitpunkt der Arbeit verschiedene Engines in Frage: Unity3D, CryEngine Free SDK (Stand Dec 2014), Unreal4 und Ogre3D (Version 1.10). Da diese Masterarbeit als Grundlage für andere Projekte dient, sind die Anforderungen dieser Projekte bereits bei der Wahl der Grafik-Engine berücksichtigt worden. Es ergeben sich folgende Anforderungen.

- Quelloffen
- kostenfrei
- Anbindung von Oculus, 3D-TV, 3D-Sound möglich
- Einsteigerfreundlich aufgrund des Einsatzes in der Lehre

Die qualitative Evaluation der Grafik-Engines (Stand März 2015) kann Tabelle 4.1 entnommen werden. Die CryEngine sowie UnrealEngine haben mit einer photorealistischen Grafik die stärkste Realitätsnähe. Allerdings sind beide nicht kostenfrei verfügbar. Unity3D ist zwar kostenfrei, aber nicht quelloffen. Die Grafik-Engine Ogre3D erfüllt alle genannten Anforderungen, bietet allerdings keine photorealistische Grafik. Da diese Anforderung zweitrangig ist, wird Ogre3D für die Umsetzung der Arbeit gewählt.

Tabelle 4.1: Evaluation versch. Grafikengines hinsichtlich ihrer Eignung für diese Arbeit

Engine	Quelloffen	Kostenfrei	Oculus	3D-Sound
CryEngine Free SDK	nein	ja	nein	ja
Unity3D	nein	ja	ja	ja
UnrealEngine 4	nein	nein	ja	ja
Ogre3D	ja	ja	ja	ja

Als Plugins werden weiterhin folgende verwendet:

- Irrklang zur Wiedergabe von 3D Sound.
- Boost für die Implementation von Threads und Nebenläufigkeit.
- LibOVR für die Anbindung des Head-Mounted-Display Oculus.
- ParticleUniverse für erweiterte Partikeleffekte.

Bei dem konzipierten Framework liegen die Hauptaugenmerke auf einer Modularisierung der Komponenten und einer einfachen Erweiterbarkeit durch Dritte. Im Zentrum steht die GameState-Klasse, welche als Haupt-Controller für die Main-Loop, die Initialisierung und Eventlistening dient (Abb. 4.1). Die GameState-Klasse initialisiert die Neben-Controller FishManager, Environment, SiccomManager und Player. Die Events werden bei Bedarf an die jeweiligen Neben-Controller weitergeleitet. Der FishManager kontrolliert die Fische. Beispielhaft wurden einige Fische implementiert, die sich zufällig entlang eines Spline bewegen. Die Klasse Environment koordiniert Licht, Himmel, visuelle Effekte und das Terrain. Die Player-Entität repräsentiert die Position der Kamera, welche über die Tastatur gesteuert werden kann. Die Kamera kann wahlweise eine Stereo- oder Monokamera sein.

Der SiccomManager übernimmt die Kommunikation mit einem Server zur Positionierung der individuellen Korallen und bildet somit die direkte Schnittstelle zum Korallenmodell von Kubicek et al. (2012). Er verwaltet die Korallen und Algenpopulation repräsentiert durch die Klassen Algae, MassiveCoral und BranchingCoral. Für die Korallen wurde das Strategie-Entwurfsmuster verwendet (Gamma et al., 1995). Dieses Entwurfsmuster entkoppelt Objekte von Ihrem Verhalten und unterstützt somit den Austausch von Algorithmen. Im konkreten Fall bedeutet das, dass die Klasse Coral für die Datenhaltung, sowie die generelle Objektinstanzierung und das Rendering zuständig ist. Die Unterklassen MassiveCoral und BranchingCoral verwalten die für die Klasse

spezifische Skeletterzeugung. Bei der verzweigten Koralle ist das die Interpretation des L-Systems. Bei der massiven Koralle entspricht das die Platzierung der Skelettknoten auf Bögen. Die darunter liegenden Unterklassen PoritesLutea, PoritesLombata, AcroporaMuricata und PocilloporaDamicornis repräsentieren je eine spezielle Art und bietet beispielsweise Raum für die Definition des L-Systems oder diverser Parameter. Das eben angesprochene L-System besteht ferner aus den Unterklassen Literals und TurtleState. Literals ist die Repräsentation eines Befehls im L-System. TurtleState repräsentiert die Momentaufnahme einer Schildkröte zu einem Zeitpunkt. Auf die genaue Implementation der Korallen wird im Kapitel 5 weiter eingegangen. Weiterhin erlaubt eine Konfigurationsdatei, generiert durch die Klasse Config, das Ändern von Parametern ohne erneutes Kompilieren.

Entsprechend der ausgearbeiteten Aspekte einer Unterwasserumgebung aus Kapitel 2.2 wurde angestrebt diese in dem Framework nachzubilden. Die blaue Umgebungsfarbe wurde durch mehrere Faktoren erreicht. Einerseits wird als Skybox eine blaue Textur verwendet und ferner der ambiente Lichtanteil von Objekten unter Wasser erhöht. Andererseits simulieren ein Nebel auf alle Objekte, sowie ein Partikelemitter, die Trübheit des Wassers. Die Kaustiken konnten mit einer animierten Textur auf dem Boden erreicht werden.

Für eine höhere Immersion wurde eine Anbindung für einen 3D-TV sowie die Oculus inklusive Headtracking eingebunden. Ferner wurden verschiedene Elemente aus dem Bachelor-Projekt VRCoralReef SS2015 übernommen: Wasseroberfläche, Texturen, Fische und deren Verhalten/Animation, Steine, Umgebungsobjekte, Algenmeshes, Optimierungen des Partikelemitters und Steuerung mit Impulserhaltung.

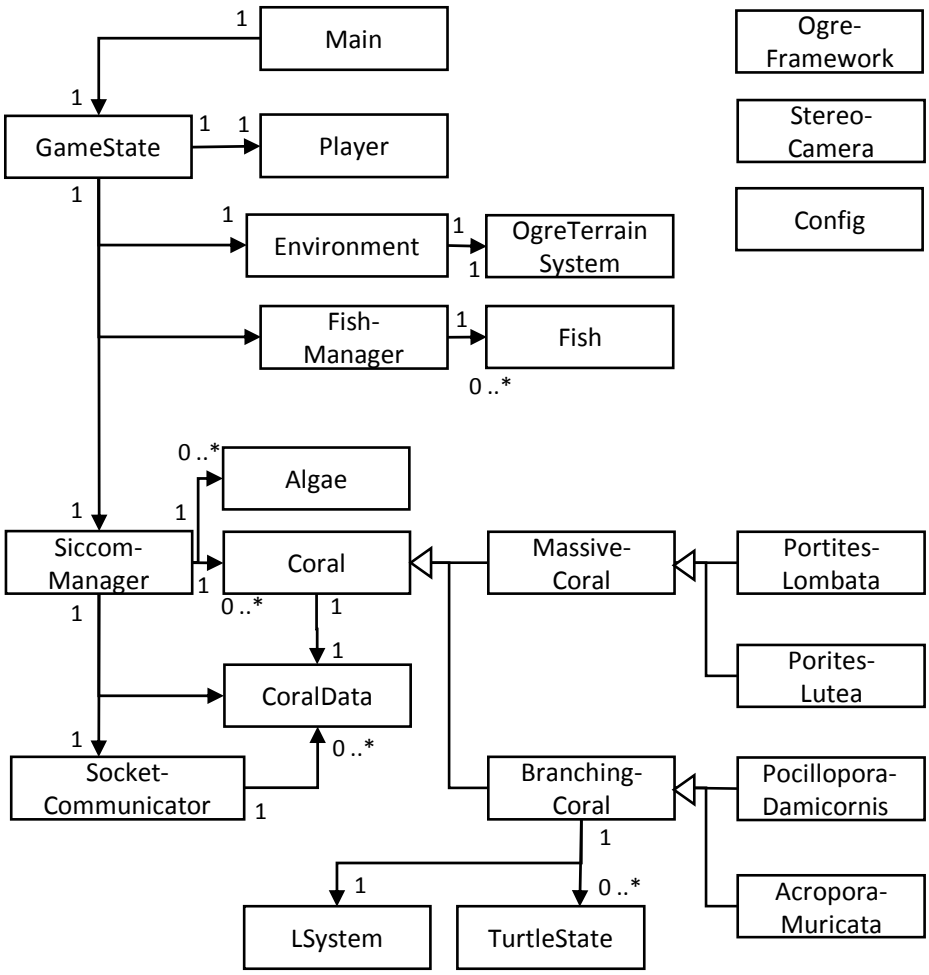


Abbildung 4.1: Klassendiagramm des Frameworks

4.1 Threading-Konzept

Es gibt im Framework vier parallele Prozesse (vgl. Abbildung 4.2). Der Main-Thread koordiniert die Kommunikation mit der Ogre3D-Engine und verwaltet und initiiert alle anderen Threads. Die Interprozess-Kommunikation ist in der Abbildung als Signale visualisiert, wobei es zwei unterschiedliche Signale gibt. Das (B) steht für ein Signal, welches einen blockierten Thread aktiviert. (KP) markiert durch eine Variable repräsentierte Signale. Ist das Signal aktiv, wird die Variable auf wahr gesetzt. Der empfangende Thread prüft stetig, ob das Signal gesetzt ist und blockiert somit nicht.

Neben dem Main-Thread steht der Netzwerk-Thread. Dieser kommuniziert in regelmäßigen Abschnitten mit dem Server und erhält von diesem die Position von Korallen und Algen. Die genaue Kommunikation zum Server wird in Abschnitt 4.2 erläutert. Wurden die Positionen ermittelt, werden die Datensätze in eine Warteschlange geschrieben. Anschließend werden die blockierten Calculator-Threads geweckt. Die Anzahl der Calculator-Threads ist variabel und kann durch den Benutzer über die Config gewählt werden. Jeder dieser Threads nimmt sich stetig Datensätze aus der Warteschlange heraus und bearbeitet diesen indem er das dazugehörige Gitternetz berechnet. Die Calculator-Threads starten ihre Aufgabe durch ein Signal vom Main-Thread. Ist die Aufgabe abgearbeitet, wird ein Signal gesetzt, dass für diesen Datensatz ein Polygonnetz vorhanden ist. Daraufhin übernimmt der Main-Thread wieder und übergibt die berechneten Vertex-Daten zum Rendern an die Grafikkarte.

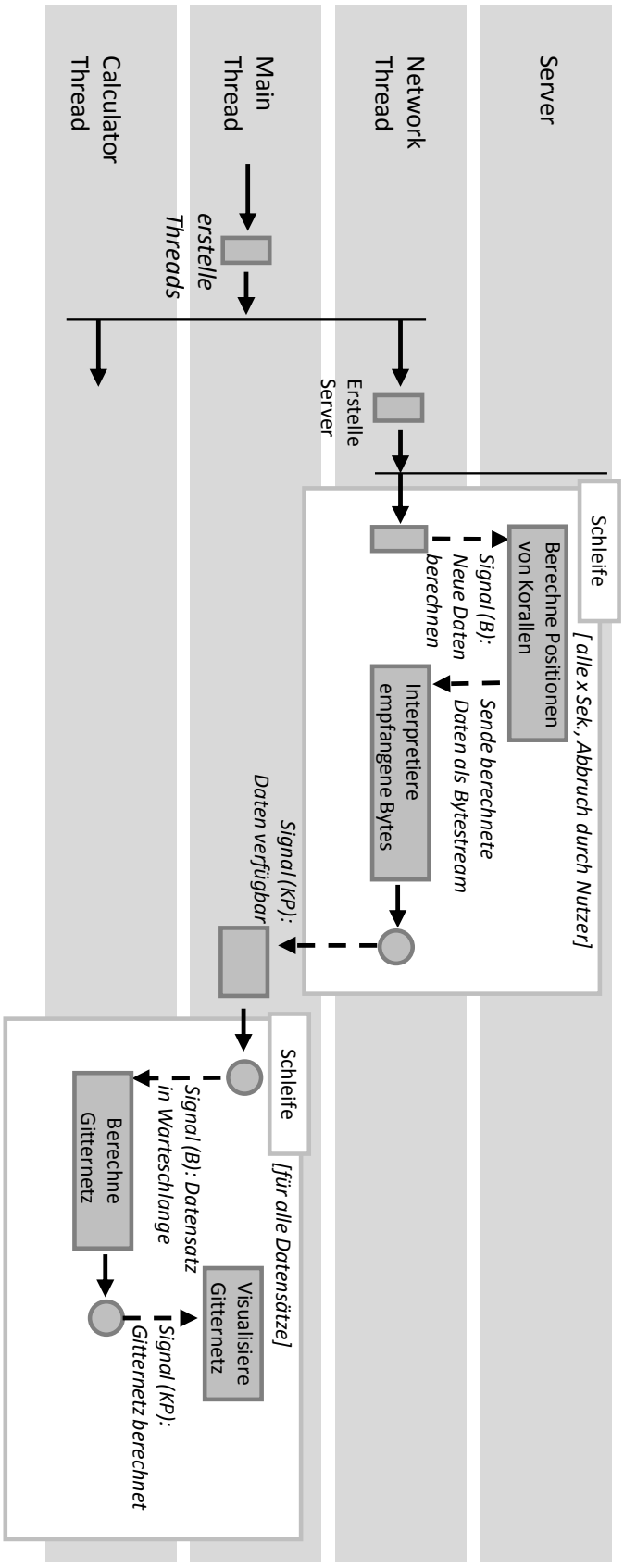


Abbildung 4.2: Threading-Konzept: Aufteilung des Programmablaufs in die einzelnen Threads inklusive Netzwerkkommunikation

4.2 Schnittstelle zum Korallenriffmodell

Die Interprozesskommunikation zwischen der Java-Simulation und dem C++-Frontend hat als besondere Anforderung, dass eine große Datenmenge schnell übertragen werden muss. Dafür bietet sich eine bidirektionale Socketlösung an. Der Vorteil gegenüber beispielsweise SharedMemory-Lösungen ist, dass Sockets vergleichsweise einfach zu implementieren und zu erweitern sind. Weiterhin lässt diese Variante offen, die Simulation auf einen leistungsstarken Server auszulagern. Im Vergleich zu anderen Netzwerk-Technologien, wie beispielsweise Webservices, sind Sockets für den lokalen Einsatz mit einer großen Datenübertragung die bessere Wahl, da man bei Webservices in den meisten Fällen an das HTTP-Protokoll gebunden ist. Somit sind die einzelnen Nachrichten zwar lesbarer aber auch größer als bei einem selbst definiertem Protokoll.

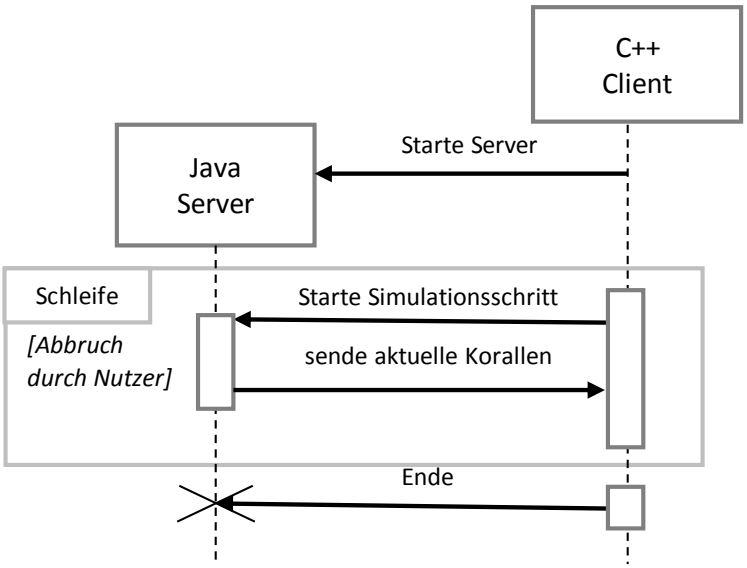


Abbildung 4.3: Sequenzdiagramm zur Client-Server-Kommunikation

Grafik 4.3 zeigt schematisch die Kommunikation zwischen der Java-Simulation und dem C++ - Frontend. Der Client startet und beendet den Server. Das Frontend sendet eine Anfrage an die Siccom-Simulation zur Berechnung des nächsten Simulationsschrittes. Wenn die Berechnung abgeschlossen ist, sendet die Simulation als Antwort die aktuellen Populationsdaten und pausiert anschließend solange bis er eine neue Anfrage erhält. Diese Interaktion findet so lange statt bis der Nutzer den Client abbricht. Die Simulationsgeschwindigkeit richtet sich nach der Geschwindigkeit der Anfragen an den

Server. Die Siccom-Simulation wurde für diese Kommunikation entsprechend angepasst.

Das implementierte Protokoll ist ein strukturierter Bytestream. Beide Kommunikationspartner wissen genau, welche Parameter zu übertragen sind und wie viele Bytes dieses umfassen. Eine Nachricht beginnt mit „!“ und terminiert mit dem „#“-Symbol. Die codierten Bytes können als einfache Datentypen wie int, char, short oder long interpretiert werden. Innerhalb der Nachricht werden die einzelnen Organismen hintereinander aufgezählt. Die Reihenfolge der Datentypen kann Tabelle 4.2 und 4.3 entnommen werden. Durch diese Spezifikation ist die Datengröße einer Koralle 216 Byte und die einer Alge 28 Byte. Die Implementation unterstützt eine Nachrichtenlänge von bis zu 100.000 Bytes und wurde erfolgreich für eine Riffgröße von bis zu 15 x 15 m getestet (vgl. Kapitel 6.2).

Tabelle 4.2: Protokollspezifikation der Korallen

Datengröße (in Bytes)	Bezeichnung	Datentyp	Kommentar
2	Type	short	Organismentyp, massive Koralle (0), verzweigte Koralle (1)
2	SubType	short	Korallenart Porites Lobata (1), Porites Lutea(2), Acropora Muricata (3), Pocillopora Damicornis (4)
8	ID	long	eindeutige ID für dieses Individuum
2	xPos	short	x-Koordinate
2	yPos	short	y-Koordinate
2	age	short	Alter der Koralle in Monaten
2	bleachGrade	short	Bleichungsgrad in %
24 * 2	Branches	short	Liste aller 24 Zweige der Koralle, wobei ein Zweig aus einer x- und einer y-Position besteht

Tabelle 4.3: Protokollspezifikation der Alge

Datengröße (in Bytes)	Bezeichnung	Datentyp	Kommentar
2	Type	short	Organismentyp, aktuell immer Alge (2)
2	SubType	short	Algenart, aktuell immer -1
8	ID	long	eindeutige ID für dieses Individuum
2	xPos	short	x-Koordinate
2	yPos	short	y-Koordinate
2	age	short	Alter der Alge in Monaten
2	height	short	Höhe der Alge %
2	isAlive	char	Status der Alge, A- lebend, D-tot

Kapitel 5

Prozedurale Korallen

Nachdem die Position und das Ausmaß einer Koralle in Kapitel 4.2 definiert wurden, wird nun das Aussehen der unterschiedlichen Korallenarten spezifiziert. Jede Koralle ist individuell in ihren Ausmaßen und ihrer Größe. Demnach wird für jede Koralle ein individuelles Polygonnetz erzeugt. Die Generierung der Koralle ist in zwei Arbeitsschritten geteilt, die Generierung des morphologischen Skeletts und die Polygonnetz-Erzeugung.

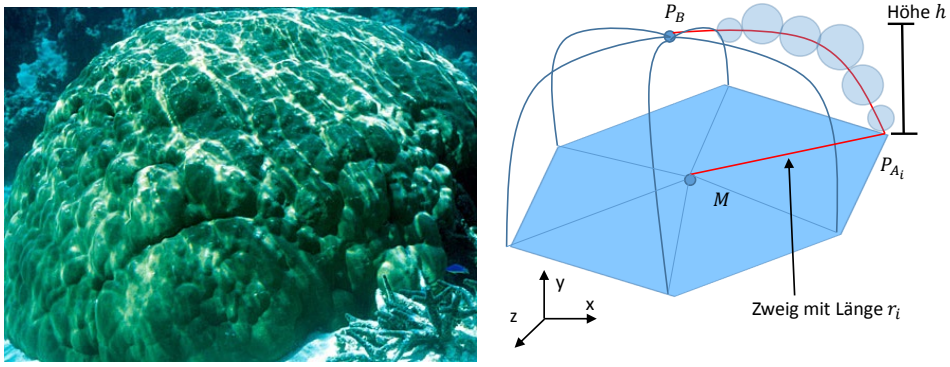
5.1 Generierung des morphologischen Skeletts

Die Struktur der massiven und der verzweigten Koralle unterscheiden sich stark voneinander. Von daher ist für beide je ein separater Algorithmus zur Beschreibung des morphologischen Skeletts vorgesehen. Der Algorithmus für die verzweigte Koralle basiert auf einem L-System (siehe Abschnitt 5.1.2). Für die Beschreibung des morphologischen Skeletts der massiven Koralle wurden Spline-Interpolationen verwendet (siehe Abschnitt 5.1.1).

Ein Skelett besteht aus einer endlichen Menge an Grundformen. Beispielhaft werden die Grundformen Kugel und abgerundeter Zylinder unterstützt. Bei der Erzeugung des Skeletts ist zu beachten, dass im nachfolgenden Schritt aus den Skeletten sogenannte Blobby Objects erzeugt werden und somit die Form der Koralle wesentlich weicher wird als im Skelett definiert. Diese Umformung lässt sich über Parameter spezifizieren (vgl. Kapitel 3.2.2 Erzeugung des skalaren Felds mit Metaballs).

5.1.1 Die massiven Koralle

Für die massive Koralle soll an dieser Stelle beispielhaft die Koralle Porites Lutea betrachtet werden. Sie zeichnet sich durch ihr kompaktes Auftreten mit einer annähernd konkaven Grundform aus (vgl. Abbildung 5.1a). Die Form ähnelt einer verformten Halbkugel mit verschiedenen Ausbuchtungen. Somit lässt diese Korallengattung sich durch eine Menge von Bögen beschreiben, auf welchen kugelförmige Knotenpunkte des Skeletts angeordnet sind. Abbildung 5.1b visualisiert diese Form schematisch.



(a) Typisches Aussehen (b) schematische Interpretation

Abbildung 5.1: Phänotyp der Porites Lutea sowie deren schematische Abstraktion, Quelle für (a): National Park Service U.S. Department of the Interior (2015)

Der Grundriss der Koralle ist in der Siccom-Simulation durch 24 Radien spezifiziert. Für jeden dieser Radien wird ein Bogen festgelegt auf dem die Skelettknoten angeordnet sind. Dieser Bogen ist spezifiziert durch einem individuellen Radius r, dem spezifischen Endpunkt P_{A_i}, sowie die für alle Bögen gemeinsamen Punkten P_B und M. Entgegen der schematischen Abbildung liegen die Punkte P_{A_i} nicht zwangsläufig auf einer Ebene. Der Vektor $\vec{a}$ verbindet die Punkte P_A und P_B. Der Vektor $\vec{x}$ liegt senkrecht auf der Ebene, die von P_{A_i}, P_B und M aufgespannt wird. Berechnet wird er entsprechend Formel 5.1.

$$\vec{x} = ((P_M - P_B) \times a) \times a \tag{5.1}$$

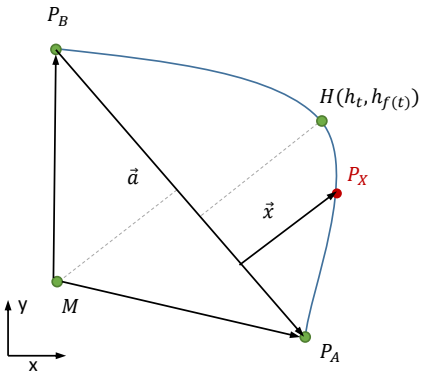


Abbildung 5.2: Interpolation der Punkte entsprechend Formel 5.2

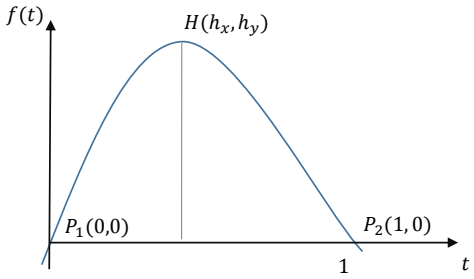


Abbildung 5.3: Abstandsfunktion $f(t)$ zur Beschreibung des Bogens über P_B und P_{A_i}

Der Abstand des Bogens zu $\vec{a}$ ist durch eine Abstandsfunktion $f(t)$ beschrieben. Die Laufvariable t ist im Bereich 0 bis 1 definiert und spiegelt die Position entlang der Geraden $\overline{P_B P_A}$ wider. Somit ist jeder beliebige Punkt P_X auf dem Bogen durch Formel 5.2 spezifiziert. Abbildung 5.2 verdeutlicht diesen Zusammenhang.

$$P_X = P_B + t \cdot \vec{a} + f(t) \cdot \frac{\vec{x}}{|\vec{x}|} \tag{5.2}$$

Für eine bessere Individualisierung der einzelnen Koralle soll die Beugung der Abstandsfunktion in Position und Ausmaße parametrierbar sein. Entsprechend ist der Punkt $H(h_t, h_{f(t)})$ als lokales Maximum definiert. Abbildung 5.3

zeigt die Abstandsfunktion. Die Abstandsfunktion $f(t)$ lässt sich stückweise durch zwei quadratische Funktionen $f_1(t)$ und $f_2(t)$ der Form $f_{1/2} = at^2+bt+c$ beschrieben.

$$f(t) = \begin{cases} \frac{-h_y}{h_x^2} \cdot t^2 + 2 \cdot \frac{h_y}{h_x} \cdot t & t \leqslant h_x \\ -zt^2 + 2h_x zt + z(1-2h_x), z = \frac{h_y}{(h_x-1)^2} & t > h_x \end{cases} \tag{5.3}$$

Der Algorithmus 5.1 zeigt die Verteilung der einzelnen Knotenpunkte. Als Eingabe sind der Mittelpunkt M, die Ausdehnung der einzelnen Zweige branches[ ] und die daraus berechnete durchschnittliche Ausdehnung avgRadius zu setzen. Zunächst werden die eingehenden Zweige um 20 % gekürzt (Algorithmus 5.1, Zeile 4). Zur Simulation des Wachstums, werden die Zufallswerte beim erstmaligen Durchlaufen des Algorithmus gesichert und in der nächsten Wachstumsphase wieder aufgerufen.

,

Algorithmus 5.1: Verteilung der Knotenpunkte für Porites Lutea

```
1  input: branches[ ], avgRadius, M
2
3  if firstRun
4          foreach branch b in branches[ ]
5                  randBranchBending[b] = Random(0.50 to 0.65)
6                  for i = 0 to i < n
7                          randNodeOffset[b *i] = Random(0.6 to 1.0)
8                          randTOffset[b*i] = Random(0.1 to 0.14)
9                          randomNodeSize = Random(0.25 to 0.40)
10
11 foreach branch b in branches[ ]
12         P_A = b.P_A + 0.2 * (M - P_PA)
13         P_B = M + (0, avgRadius / 2, 0)
14         a   = P_A - P_B
15         x   = ((M - P_B) x a) x a
16
17         h_y = randBranchBending[b] * avgRadius
18         h_x = 0.6
19         t = 0
20
21         for i = 0 to i < n
22                 P_X = P_B +t *a + f(t) * x + randNodeOffset[b *i]
23                 radius =  randomNodeSize * avgRadius
24
25                 t = t +  randTOffset[b*i]
```

5.1.2 Die verzweigte Koralle

Das typische Aussehen einer verzweigten Koralle ist eine große Menge an primären Zweigen, welche strahlenförmig vom Ursprung ausgehen. Diese Zweige verästeln sich weiterhin. Als Beispiel sei hier die verzweigte Koralle Acropora Muricata angeführt (vgl. Abbildung 5.4).

Abbildung 5.4: Typisches Aussehen der verzweigten Koralle Acropora Muricata (Lizard Island Research Station, 2015)

Zur Beschreibung des morphologischen Skeletts bietet sich ein L-System an (vgl. Abschnitt 3.2). Das hier beschriebene L-System hat folgende Eigenschaften:

- dreidimensional
- geklammert
- stochastisch
- parametrisch
- umgebungssensitiv

Formal wird das System mit $G = (V, \Sigma, w, P, \Pi)$ beschrieben. Die Iterationstiefe wird auf $n = 3$ festgelegt. Mit jeder Iterationstiefe werden die Äste um 10 % kürzer. Als Variable ist die Menge $V = \{F(d,\delta), x(\lambda,\delta), z(\lambda,\delta), [,]\}$ definiert, welche durch die Turtle wie folgt interpretiert wird:

$F(d,\delta)$ ... Geradeaus mit Schrittlänge $d + r$, $r \in [-\frac{\delta}{2}, \frac{\delta}{2}]$

$x(\lambda,\delta)$ ... Rotation um die x-Achse um $\lambda^\circ + r$, $r \in [-\frac{\delta}{2}, \frac{\delta}{2}]$

$z(\lambda,\delta)$ ... Rotation um die z -Achse um $\lambda^\circ + r$, $r \in [-\frac{\delta}{2}, \frac{\delta}{2}]$

[... speichere den aktuellen Turtle-Status

] ... gehe zurück auf den gespeicherten Turtle-Status

$\frac{\delta}{2}$ stellt jeweils eine Standardabweichung vom vorgegebenen Wert d oder λ dar, welche zufällig generiert wird. Zur Definition der primären Verästelung, repräsentiert durch w, wird ein Fächer um die x- und um die z-Achse aufgespannt. Die einzelnen Zweige haben eine Abweichung von $\delta = 15^\circ$. Der Abstand zwischen zwei Zweigen ist $\lambda = 30^\circ$ Grad. Mit dem Axiom w in Formel 5.4 werden diese primären Verästelungen beschrieben:

$$w = \sum_{i_x=-2}^{2} \left(\sum_{i_z=-2}^{2} [x(i_x\lambda,\delta) z(i_z\lambda,\delta) F] \right) \tag{5.4}$$

Die Regelmenge P in Formel 5.5 zeigt die erlaubten höheren Verästelung. Demnach gibt es entweder zwei Zweige auf der x-Achse, zwei Zweige auf der z-Achse oder nur eine Abbiegung. Die Länge der Zweige variiert jeweils von 20 bis 70 % der eigentlichen Länge des Zweiges in dieser Iteration. Die Regelmenge P und das Startaxiom w wurden durch einen iterativen Prozess aus Testen und Vergleichen mit Anschauungsmaterial (vgl. Abbildung 5.4) ermittelt.

$$\begin{aligned} P = \{ & p_1 : F(d,\delta) \xrightarrow{0.5} [x(30,10)F(0.5,0.3)][x(-30,10)F(0.5,0.3)] \\ & p_2 : F(d,\delta) \xrightarrow{0.4} [z(30,10)F(0.5,0.3)][z(-30,10)F(0.5,0.3)] \\ & p_3 : F(d,\delta) \xrightarrow{0.1} [x(30,10)F(0.5,0.3)]\} \end{aligned} \tag{5.5}$$

Die verzweigte Koralle wächst innerhalb seines vorgegebenen Volumens. Der Ansatz des umgebungssensitiven L-Systems (vgl. 3.2.1) von Prusinkiewicz und Lindenmayer (1996) ist recht rechenintensiv, da zu jedem Zeitschritt das L-System neu erstellt werden muss. Außerdem muss eine Möglichkeit zum Zwischenspeichern der Zufallswerte gefunden werden. Um diese beiden Nachteile zu umgehen, wird an dieser Stelle eine alternative Lösung vorgestellt.

Das L-System wird zunächst für eine komplett ausgewachsene Koralle erzeugt. Beim Ablaufen der Turtle wird geprüft, ob der zu zeichnende Ast innerhalb der Wachstumsvolumen liegt oder außerhalb. Gegebenenfalls wird der Ast komplett weggelassen oder gekürzt. Gegeben sei ein aktueller Turtle Status $ts(T_{start}, \lambda_x, \lambda_z, d)$ mit den Parametern

T_{start} ... Startposition (x,y,z)

λ_x ... Rotation um die x-Achse

λ_z ... Rotation um die z-Achse

d ... Schrittweite.

Zunächst wird die ursprüngliche Endposition T_{end} ermittelt. Der Quaternion r beschreibt die Rotation ermittelt aus den Achsenrotationen r_x und r_z.

$$T_{end} = T_{start} + r \cdot \begin{pmatrix} 0 \\ d \\ 0 \end{pmatrix} \tag{5.6}$$

Es gibt drei Varianten, wie der Zweig gezeichnet werden kann. Liegen Start- und Endpunkt innerhalb des Objekts, wird der Zweig komplett gezeichnet. Wenn der Startpunkt innerhalb des Wachstumsvolumens liegt, nicht aber der Endpunkt, wird der Zweig anteilig gezeichnet. Liegen beide Punkte außerhalb des Wachstumsbereichs, wird der Zweig komplett gezeichnet.

Zur Ermittlung, ob ein Punkt T innerhalb oder außerhalb der Wachstumsfläche liegt, werden die 24 Radien, welche die Ausdehnung einer Koralle beschreiben, zur Hilfe genommen. Die 24 Radien liegen auf der xz-Ebene mit dem gemeinsamen Mittelpunkt M. Radius 0 wird beschrieben durch den Einheitsvektor $E_x = (1, 0, 0)^T$. Die anderen Radien sind im Uhrzeigersinn mit einem Abstand von jeweils 15° angeordnet. Aus dem Winkel α zwischen dem Einheitsvektor E_x und dem Punkt P_{Turtle} lässt sich die Nummer i_{Radius} des zuständigen Radius ermitteln (Formel 5.7).

$$i_{Radius} = \frac{\alpha}{360} \cdot (24 - 1) \tag{5.7}$$

$$\alpha = \begin{cases} \frac{(M - P_{Turtle}) \cdot E_x}{|M - P_{Turtle}|} & 0 \leqslant \alpha < 360 \\ \frac{(M - P_{Turtle}) \cdot E_x}{|M - P_{Turtle}|} + 360 & -360 < \alpha < 0 \end{cases} \tag{5.8}$$

Befindet sich ein Startpunkt innerhalb und ein Endpunkt außerhalb, so muss der Endpunkt neu berechnet werden (Formel 5.10). Dazu wird zunächst der prozentuale Anteil *perc* berechnet. Dieser Anteil beschreibt, wieviel Prozent des eigentlichen Astes gezeichnet werden.

$$T_{end} = T_{start} + perc \cdot r \cdot \begin{pmatrix} 0 \\ d \\ 0 \end{pmatrix} \tag{5.9}$$

$$perc = \frac{br(i_{Radius}) - distProj(M, T_{start})}{distProj(T_{start}, T_{end})} \tag{5.10}$$

, wobei die Hilfsfunktion $distProj(a, b)$ die Vektoren a und b auf die x,z-Ebene projiziert und anschließend den Abstand berechnet. Die Funktion $br(i_{Radius})$ gibt die Länge des zuständigen Radius zurück.

5.2 Isoflächen-Implementation

In Abschnitt 5.1 wurde die grobe Struktur der Korallen durch ein Skelett aus Grundformen definiert. Die gegebenen Grundformen sollen nun in ein Polygonnetz übersetzt werden. Als grundlegender Algorithmus wird eine modifizierte Variante des Marching Cubes Algorithmus (vgl. 3.2.3 Polygonnetz-Erzeugung mit Marching Cubes) verwendet. Als Vorlage für die Implementation dient das Code-Beispiel von Paul Bourke (Bourke (1997)). Dieses wurde an einigen Stellen angepasst. Beispielsweise wird das skalare Feld vorberechnet. Im digitalen Anhang befindet sich der gesamte Code in C++-Notation.

Für das Objekt ist ein umgebender rechteckiger Hüllkörper (bounding box) mit den Abmaßen bb_w,bb_h, bb_d definiert. Das Objekt wird in ein Gitter, also eine Menge an Würfeln unterteilt. In x-, y- und z- Richtung hat der lokale Objektraum eine Auflösung von $n_w \times n_h \times n_d$ Würfeln. Ein Würfel hat die Seitenlängen δ. Damit alle Objekte in der Welt die gleiche Auflösung haben ist δ vorgegeben und n_w, n_h und n_d variabel in Formel 5.11 berechnet.

$$n_{w/h/d} = \frac{bb_{w/h/d}}{\delta} + 1 \tag{5.11}$$

Lorensen und Cline (1987) teilt den Objektraum zunächst in Ebenen ein, welche die Höhe eines Würfels haben. Jede Ebene besteht aus einem Gitter von Punkten mit dem Abstand δ. Beim Durchlaufen werden zunächst vier Ebenen ausgewählt und dazu die skalaren Werte ermittelt. Anschließend wer-

den aus zwei Ebenen die Punkte zu einem Würfel extrahiert. Ein alternatives Vorgehen zeigt sich in der hier modifizierten Implementation frei nach Bourke (1997). Zunächst wird für alle Gitterpunkte der skalaren Wert berechnet und in einem Feld zwischen gespeichert. Der Aufwand ist derselbe wie in dem Ansatz von Lorensen und Cline (1987), da für jeden Gitterpunkt maximal einmal der skalare Wert ermittelt werden muss. Allerdings bietet dieser Ansatz die Möglichkeit die skalaren Werte in einem späteren Zeitpunkt, beispielsweise für die Normalenberechnung, wiederzuverwenden.

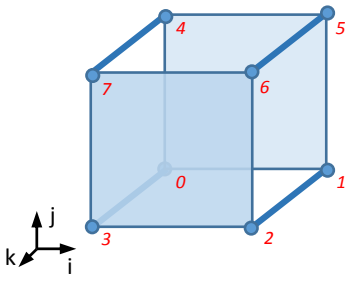
Für die Extraktion des temporären Würfels wird zwischen zwei Koordinatensystemen unterschieden. Das kartesische Koordinatensystem beschreibt die Lage der Punkte in alle drei Dimensionen x,y und z über reelle Zahlen. Ein zweites Koordinatensystem im natürlichen Raum $\mathbb{N}^3$ mit den Dimensionen i,j und k gibt jedem Punkt einen fortlaufenden Index für jede Dimension. Formel 5.12 zeigt die Transformation zwischen den Koordinatensystemen.

$$\vec{P_{x,y,z}} = \begin{pmatrix} P_i \cdot \delta \\ P_j \cdot \delta \\ P_k \cdot \delta \end{pmatrix} \quad ; \quad \vec{P_{i,j,k}} = \begin{pmatrix} \frac{P_x}{\delta} \\ \frac{P_y}{\delta} \\ \frac{P_z}{\delta} \end{pmatrix} \tag{5.12}$$

Aus den achsenabhängigen Indizes P_x, P_y, und P_z eines Punktes lässt sich ein Index $i \in \mathbb{N}$ für den Punkt berechnen, welcher im Objektraum eineindeutig ist:

$$i = P_k \cdot (n_w + 1) \cdot (n_h + 1) + P_j \cdot (n_w + 1) + P_i \tag{5.13}$$

Ein Würfel wird durch einen Referenzeckpunkt spezifiziert. Die Koordinaten sowie Indizes des Würfels lassen sich relativ zu diesem ermitteln. In Abbildung 5.5 ist dieser Zusammenhang verdeutlicht. Die Funktion offset gibt für die Nummer einer Ecke die relative Position zurück.



e	o_i	o_j	o_k
0	0	0	0
1	1	0	0
2	1	0	1
3	0	0	1
4	0	1	0
5	1	1	0
6	1	1	1
7	0	1	1

Abbildung 5.5: Indexierung der Eckpunkte in einem Würfel

Bei der Ermittlung der geschnittenen Kanten wird hier ähnlich vorgegangen wie bei Lorensen und Cline (1987). Für alle Kanten eines Würfels wird geprüft, ob der skalare Wert an einer Stelle den Isowert überschreitet. Ist das der Fall wird mittels Bitshifting-Operation ein Index erzeugt, welcher Rückschluss darauf gibt, welche Eckpunkte Teil des Objekts sind und welche außerhalb liegen (vgl. Algorithmus 5.2).

,

Algorithmus 5.2: Angepasster Marching-Cubes Algorithmus

```
1  for v = 0 to v < 8
2     if (scalarWert <= isoWert)
3           edgeIndex |= 1 << i;
4
5  edgeFlags = edgeTable[edgeIndex]
6
7  if (edgeFlags == 0)
8     return
9  else
10    foreach edge  // 8 edges
11       if intersection on edge
12          offset = linearInterpolation (edge.v1, edge.v2)
13          vertex = edge.v1 + offset * (edge.v2 - edge.v1)
14          if (indexBuffer has vertex)
15             indexBufferID = indexBuffer.getID(vertex)
16          else
17             indexBuffer.addVertex(vertex)
```

In einer Tabelle kann dieses spezifische Muster nachgesehen werden (Zeile 5). Im Falle eines Schnittpunktes auf einer Kante muss die exakte Lage des Vertex bestimmt werden (Zeile 11 - 13). Dazu wird zwischen den beiden Skalarwerten interpoliert. Damit Vertices nicht mehrfach gespeichert werden, wird ein Indexbuffer verwendet (Zeile 12 - 15). Ein Indexbuffer enthält für jeden Punkt im Objekt einen Index und referenziert diesen in den einzelnen Dreiecken. Die errechneten Vertices werden anschließend zu Dreiecken zusammengefasst. Für einen Würfel mit 16 Seiten gibt es maximal 5 Dreiecke. Für einen Würfel gibt es nach den Regeln der Kombinatorik maximal 5 Dreiecke. Die möglichen Interpretationen von Dreiecken eines spezifischen Kantenschnittmusters werden in einer zweiten Tabelle nachgesehen. Das Ergebnis des Algorithmus ist eine indizierte Dreieckliste. Die genaue Implementation kann im digitalen Anhang eingesehen werden.

Um die Individualität jeder Koralle zu erhöhen, erhält sie eine zufällige Farbe. Damit die Zugehörigkeit der Koralle zu ihrer Gattung noch erkenntlich ist, wird die Farbe zufällig zwischen zwei vergebenen Farben c_1 und c_2 entsprechend Formel 5.14 gefällt. Die Formel wird auf alle Farbkanäle äquivalent

angewendet, wobei r eine zufällige Zahl zwischen 0 und 1 ist.

$$c = c_1 \cdot (1 - r) + c_2 \cdot r \tag{5.14}$$

Um ein feineres Bild zu erzeugen wird desweiteren eine einfache Noisetextur per Sphere-Mapping Verfahren auf die Korallen gelegt. Die uv-Koordinaten werden entsprechend Formel 5.15 und Formel 5.16 berechnet. Der Vector $\vec{d} = \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}$ ist der normalisierte Richtungsvektor vom Mittelpunkt vom aktuellen Vertex der Koralle.

$$uv_x = 0.5 + \frac{atan2(d_z, d_x)}{2 \cdot \Pi} \tag{5.15}$$

$$uv_y = 0.5 + \frac{asin(d_y)}{\Pi} \tag{5.16}$$

Damit das Riff optisch größer wirkt, wird es in z- und in x-Richtung jeweils zwei mal gespiegelt. Somit gibt es insgesamt 9 identische Rifffelder, welche mit einer größe von 10 m Seitenlänge so gewählt wurden, dass diese Dopplung optisch nicht auffällt.

Kapitel 6

Analyse

Für eine Bewertung der vorliegenden Lösung werden zunächst die visuellen Ergebnisse präsentiert. Anschließend wird die Performanz der prozeduralen Korallengenerierung, die Auslastung der unterschiedlichen Threads sowie der Netzwerkverkehr dargestellt und analysiert. Das Referenzsystem hat folgende relevante Hardware: Windows 8.1 64 bit, 3.3 GHz Prozessor mit 4 Kernen/-Threads, 8GB RAM, NVIDIA GeForce GTX 760.

6.1 Visuelle Ergebnisse

Abbildung 6.1 zeigt das erzeugte Polygonnetz der massiven Koralle. Bedingt durch den beschriebenen Algorithmus, ist die Koralle innen hohl. Von außen ist dies nicht zu sehen. Das erzeugte Polygonnetz wird nicht auf Löcher oder ähnlich Fehler überprüft. Allerdings sind diese Fehler in mehren Durchläufen nicht aufgetreten. Bruin (2004) beschreibt, dass Polygonnetze mit einer überwiegenden Anzahl von gleichseitigen Dreiecken, als optisch ansprechender empfunden werden. Entsprechend sind langgezogene spitze Dreiecke zu vermeiden. Abbildung 6.1 verdeutlicht, dass diese degenerierten Dreiecke selten auftreten.

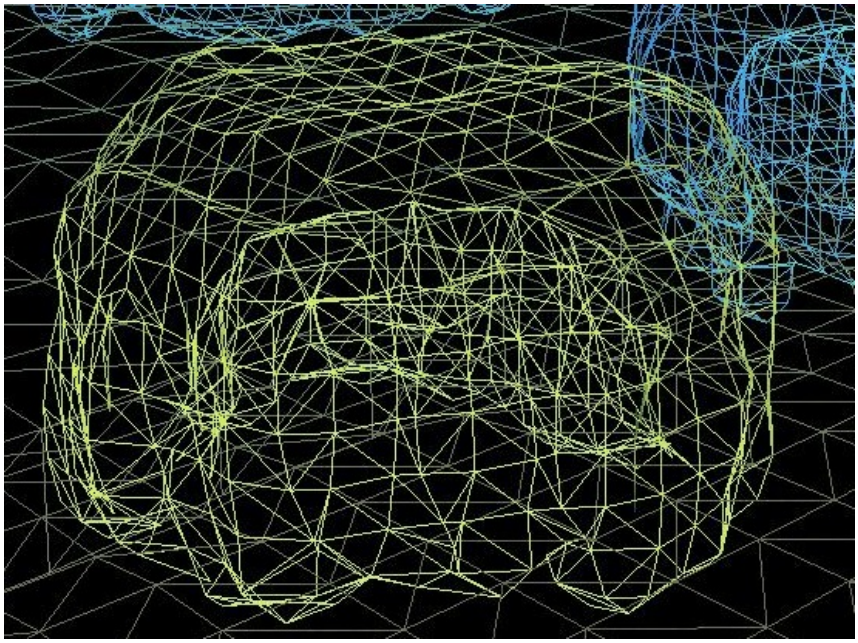


Abbildung 6.1: Polygonnetz einer massiven Koralle erstellt mit Marching Cubes Algorithmus und einer Voxelgröße von 10

In Abbildung 6.2 ist die massive Koralle *Porites Lutea*, gerendert für unterschiedliche Voxelgrößen, zu sehen. Respektive sind in Abbildung 6.3 die unterschiedlichen Körnungen der verzweigten Koralle *Acropora Muricata* abgebildet. Die Farben sind an dieser Stelle explizit kräftig gewählt, damit der Betrachter die Arten besser unterscheiden kann (änderbar über die Konfigurationsdatei).
Fest steht, je kleiner die Voxelgröße ist umso feiner ist das Polygonetz und umso weicher wirkt die Oberfläche. Bei einer zu groben Voxelgröße ist die Skelettstruktur weniger erkennbar. Generell ist die zu wählende Voxelgröße abhängig von der Größe der einzelnen Skelettknoten. Dies ist auch der Grund, warum für die massive Koralle eine größere Voxelgröße verwendet wird als bei der verzweigten Koralle. Generell ist die Auflösung des Polygonnetzes innerhalb einer Koralle an jeder Stelle gleich.
Die abgebildeten Korallen sind jeweils Individuen, welche sich entsprechend in dem morphologischen Skelett unterscheiden. Dem zu Grunde liegen einerseits eine Menge von Zufallsparametern und andererseits die Anpassung des Individuum an die Form des Terrain. Abbildung 6.4 zeigt die gesamte generierte Riffpopulation.

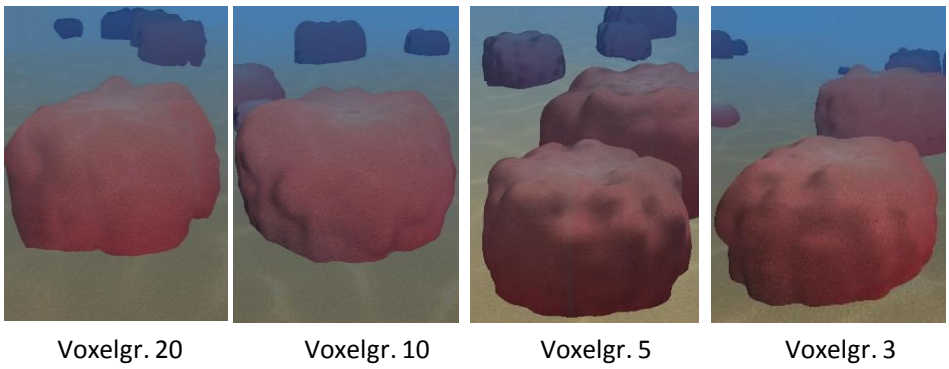


Abbildung 6.2: Porites Lutea gerendert bei unterschiedlichen Voxelgrößen

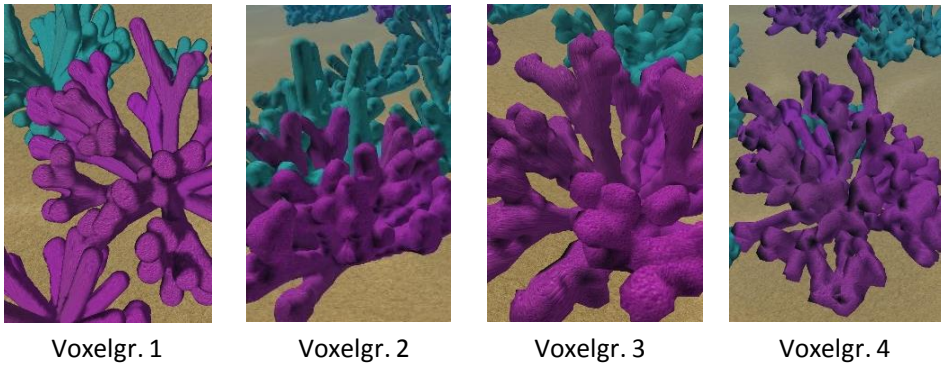


Abbildung 6.3: Acropora Muricata gerendert bei unterschiedlichen Voxelgrößen

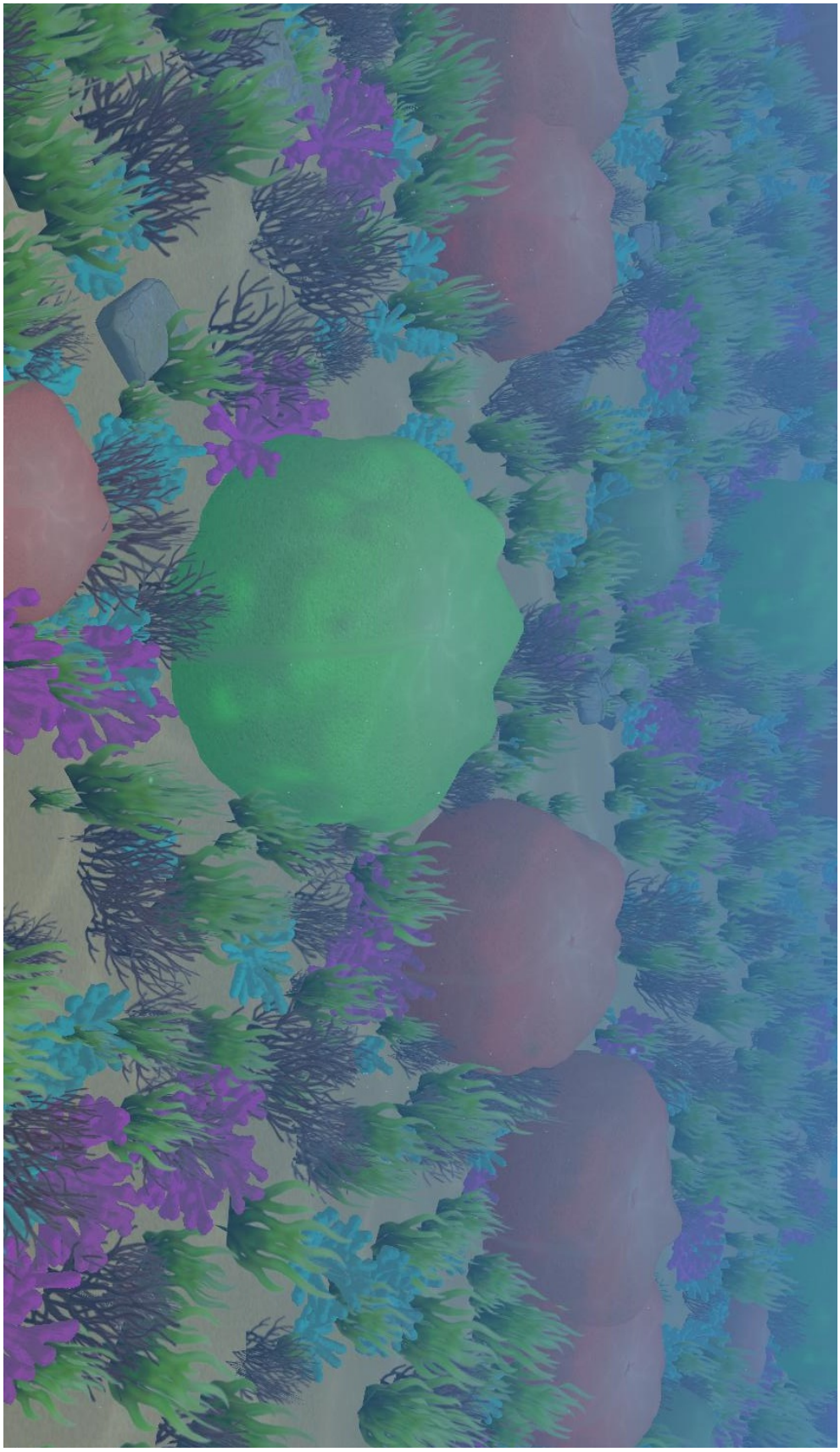


Abbildung 6.4: gesamtes Riff mit 4 farblich unterschiedlichen Korallenarten und zwei unterschiedlichen Wachstumsalgorithmen (Porites Lutea und Acropora Muricata) mit den Voxelgrößen 5 für massive Korallen und 2 für verzweigte Korallen, sowie Algen

6.2 Netzwerkanalyse

Der Socket-Netzwerkverkehr läuft periodisch ab und beide Kommunikationspartner befinden sich auf dem Referenzsystem. Der Visualisierungs-Client sendet in einem gegebenen Intervall Anfragen an den Simulationsserver. Die vom Client gesendeten Daten sind mit wenigen Parametern maximal 10 Byte groß und somit vernachlässigbar gering. Die Daten vom Simulationsserver an den Visualisierungs-Client hingegen sind von der definierten Riffgröße und daraus resultierenden Algen- und Korallenpopulation abhängig und somit nicht konstant. In den nachfolgenden Analysen wird die Zeit für einen Simulationsmonat und somit das Anfrageintervall vom Client an den Server auf 10 Sekunden gesetzt.
Die genaue Zusammensetzung von Korallen und Algen ist abhängig von dem Simulationsalgorithmus und den gesetzen Parametern. Abbildung 6.6 und 6.5 zeigen die durchschnittlichen Algen- und Korallenpopulationen über einen Zeitraum von 100 Simulationsmonaten für unterschiedliche Riffgrößen. Da der Simulationsalgorithmus auch Zufallswerte enthält, sind über mehrere Durchläufe Abweichungen in der Anzahl der Korallen und Algen je Simulationsschritt zu beobachten. Für die Auslegung des Netzwerkverkehrs sind vor allen maximale und minimale Werte von Bedeutung. Diese sind direkt am Graphen aufgetragen. Bei einem kleinen Riff von 5 mal 5 m Größe werden im Durchschnitt ca. 20 Korallen und ca. 26 Algen mit einem Maximum von 37 Korallen und 72 Algen übertragen. Das mittlere Riff hat eine durchschnittliche Übertragungsrate von ca. 124 Korallen und ca. 99 Algen mit einem maximalen Wert von 93 Korallen und 265 Algen. Bei dem großen Riff treten die stärksten Schwankungen in der Population auf. Im Schnitt werden hier ca. 248 Korallen und ca. 220 Algen mit einem Maximum von 430 Korallen und 649 Algen übertragen. Die Diagramme zeigen periodisch alle zwölf Monate eine Spitze. Diese Spitze ist auf den Vermehrungsphase im Frühjahr zurückzuführen. Auch wird sichtbar, dass die Korallenpopulation mit der Zeit nachlässt. Dies ist auf die gesetzten Parameter im Riff zurückzuführen und indiziert, dass die Parameter für die Korallen nicht optimal sind und diese mit der Zeit aussterben werden.

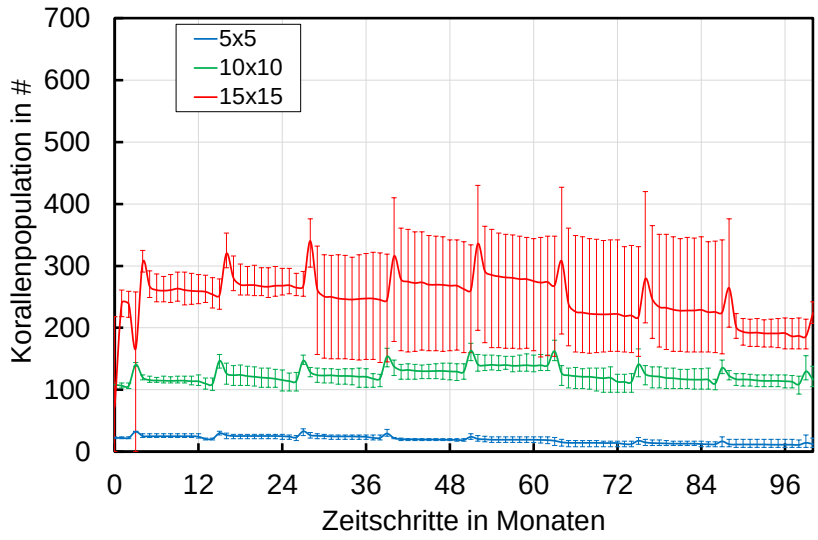


Abbildung 6.5: Anzahl der im Netzwerk übertragenen Korallen bei unterschiedlichen Riffgrößen

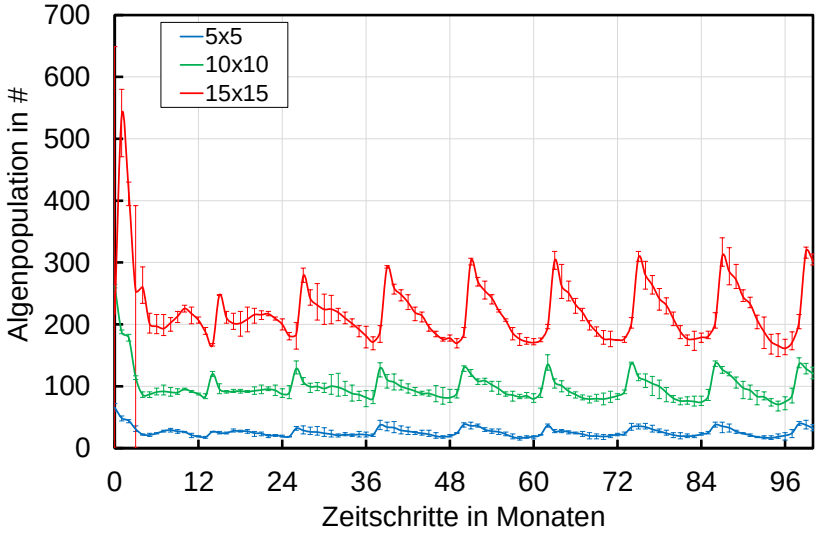


Abbildung 6.6: Anzahl der im Netzwerk übertragenen Algen bei unterschiedlichen Riffgrößen

Aus den übertragenen Riffpopulationsdaten lässt sich die gesamte Datenlast im Netzwerk berechnen. Eine Alge wird durch eine Datengröße von 28 Byte repräsentiert. Eine Koralle lässt sich mit 126 Byte beschreiben (vgl. Kapitel 4.2). In Abbildung 6.7 ist die empfangene Datengröße zur gesamten Riffpopulation bei unterschiedlichen Riffgrößen dargestellt (5 mal 5 m, 10 mal 10 m, 15 mal 15 m). Neben der durchschnittlichen Datengröße zu einem Simulationsschritt wird auch die maximale und minimale Datengröße zu dem Zeitpunkt dargestellt. Das kleine Riff mit einer Seitenlänge von 5 m hat eine gleichmäßige Datenauslastung von durchschnittlich ca. 4,8 kB im gesamten Messraum mit einer maximalen Auslastung von 7,75 kB. Bei einem mittleren Riff ist einen durchschnittliche Datenmenge von 28,87 kB mit einer maximalen Auslastung von 37,66 kB zu beobachten. Die stärkste Abweichung zeigt das große Riff von 15 m Seitenlänge. Hier konnte eine Auslastung von 58,43 kB mit einem Maximum von 78,4 kB beobachtet werden.

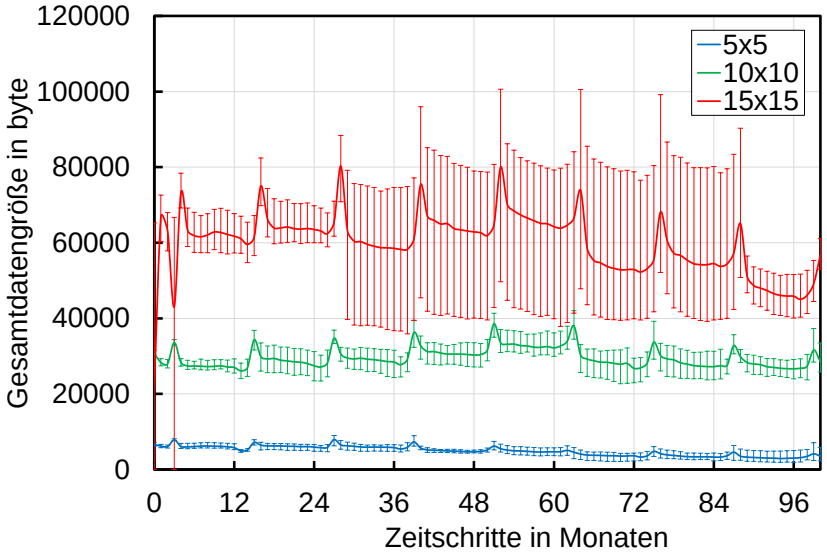


Abbildung 6.7: durchschnittliche übertragene Datenmenge je Simulationsschritt mit maximaler und minimaler Auslastung für unterschiedliche Riffgrößen

Um genauer einzuschätzen, wie sich die Riffgröße auf die zu übertragende Datengröße auswirkt, wurden die mittleren Datenmengen bei weiteren Riffgrößen im Bereich von 5 bis 15 m Seitenlänge gemessen. Abbildung 6.8 zeigt die Datengröße in Abhängigkeit von der Riffläche in m^2. Die übertragenden Datenmengen steigen linear zu der Riffläche. Ferner nehmen die Schwankungen in der übertragenen Datenmenge mit einem größeren Riff deutlich zu.

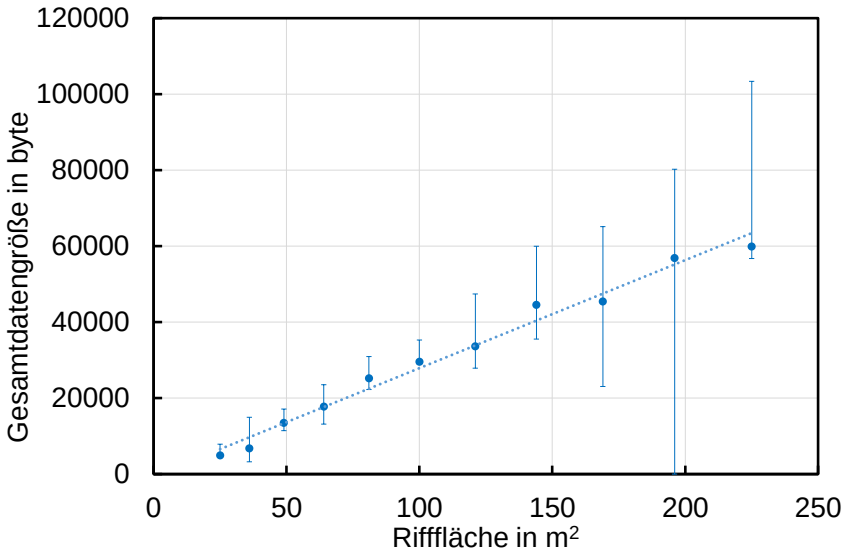


Abbildung 6.8: durchschnittliche übertragene Datenmenge mit maximaler und minimaler Auslastung für unterschiedliche Riffgrößen

Zusammenfassen ist zu sagen, dass die Netzwerklast deutlich von der Größe des Riffs abhängig ist und starken Schwankungen unterliegt. Die Population besteht hauptsächlich aus Algen, wobei der Anteil der Korallen im den größeren Datenanteil ausmacht. In Testläufen konnte mit einer Riffgröße von 10 x 10 m ein zufriedenstellendes Bild mit einer stabilen Netzwerklast von durchschnittlich 28,87 kB erzeugt werden.

6.3 Performanz der prozeduralen Korallen

Generell lässt sich sagen, dass je mehr Vertices in dem darzustellenden Bild enthalten sind, umso schlechter ist die Performanz. Diese Aussage lässt sich treffen, da durch eine höhere Vertexanzahl im Rendering-Prozess mehr Berechnungen durchgeführt werden müssen und somit weniger Bilder pro Sekunde berechnet werden können. Demnach soll die Vertexanzahl pro Koralle an dieser Stelle als Performanzmaß dienen.

Es wurden verschiedene Korallen mit unterschiedlichen Auflösungen gemessen. Insgesamt wurden dazu 42557 Datensätze erfasst. Für die massive Koralle wurde als Auflösung eine Voxelgröße von 5, 7, 10 und 20 erfasst. Für die verzweigte Koralle wurden die Voxelgrößen 2, 3, 4 und 8 gemessen. Abbildung 6.9 und 6.10 zeigen die Anzahl der Vertices in Abhängigkeit der Größe einer Koralle für die unterschiedlichen Voxelgrößen. Es wird ersichtlich, dass mit steigendem Radius die Anzahl der Vertices ebenfalls ansteigt. Die verzweigte Koralle hat bei gleichem Radius deutlich mehr Vertices als die massive Koralle. Auch ist der Anstieg der Vertexanzahl mit größer werdendem Radius bei der verzweigten wesentlich stärker. Die Anzahl der Skelettknoten ist bei der massiven Koralle auf 240 festgelegt. Bei den verzweigten Korallen steigt die Anzahl der Skelettknoten mit dem Radius. Im Schnitt hat die verzweigte Koralle ca. 250 Skelettknoten.

Aufgrund des größeren Radius hat die massive Koralle bei einer Voxelgröße von 5 eine Vertexanzahl von bis zu 15500. Bei der verzweigten Koralle wird dieser Wert bei einer Voxelgröße von 8 nicht erreicht.

Abbildung 6.11 zeigt die Berechnungszeit zur Ermittlung der Vertices für die unterschiedlichen Voxelgrößen. Diese Berechnungzeit schließt die Generierung des morphologischen Skeletts, die Ermittlung des skalaren Felds und den Marching Cubes Algorithmus mit ein. Es ist zu beobachten, dass je geringer die Voxelgröße ist umso länger dauert die Berechnung.

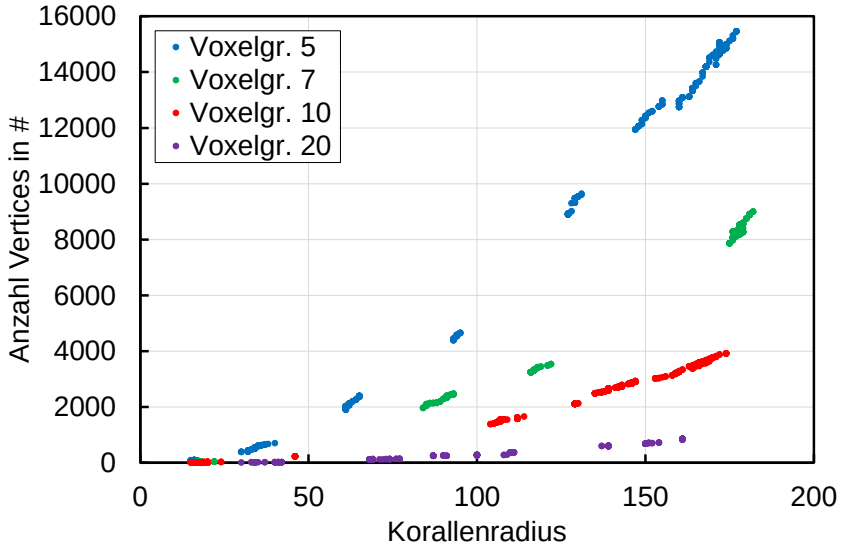


Abbildung 6.9: Anzahl der generierten Vertices für unterschiedliche Radien und Voxelgrößen für die massive Koralle

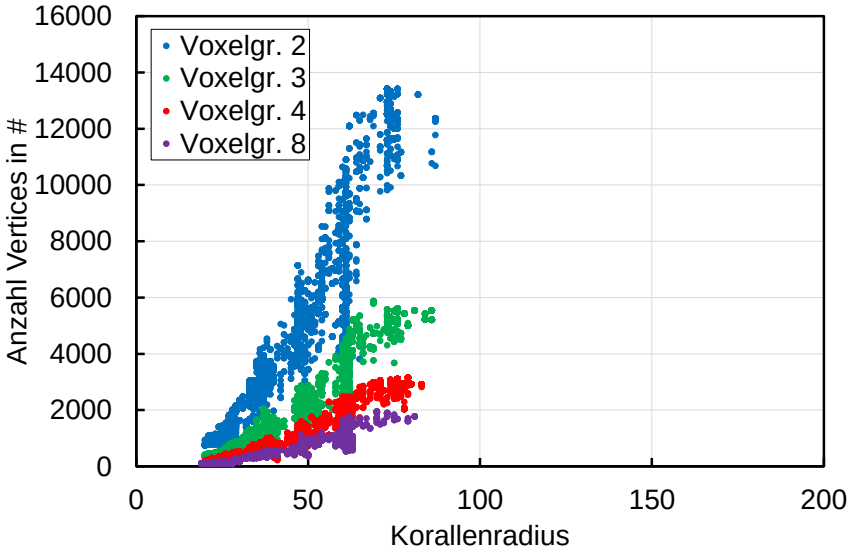


Abbildung 6.10: Anzahl der generierten Vertices für unterschiedliche Radien und Voxelgrößen für die verzweigte Koralle

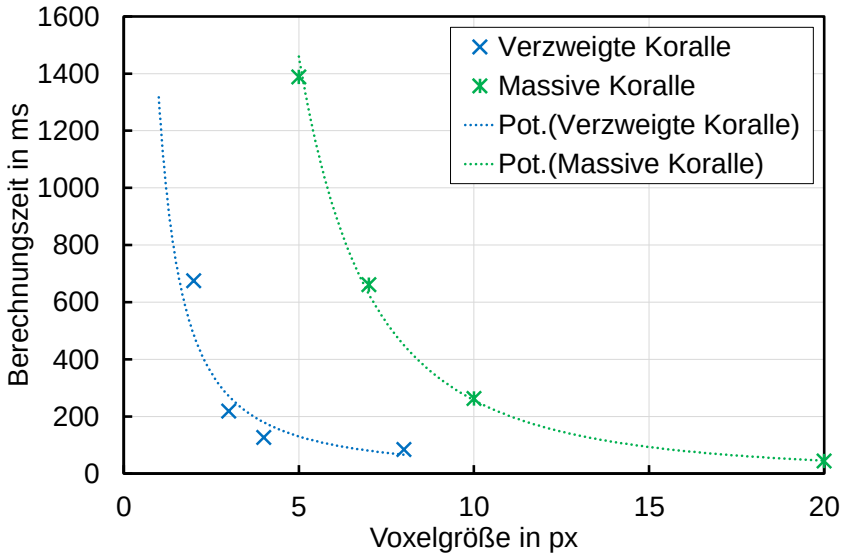


Abbildung 6.11: Berechnungzeit zur Generierung eines Polygonnetzes für massive und verzweigte Korallen bei unterschiedlichen Voxelgrößen

6.4 Auslastung der CPU

Zur Einschätzung der CPU-Auslastung wurde diese mit dem Tool Intel VTune Amplifier XE 2015 gemessen. Abbildung 6.12 zeigt die gemessenen Resultate über einen Zeitraum von 12 Simulationsschritten, wobei ein Simulationsschritt 10 Sekunden Echtzeit entspricht. Für die Messung wurden alle Wassereffekte und Fische ausgeschaltet. Der oberste Thread ist der Main-Thread. Hier werden diverse CPU-seitige Ogre-Berechnungen durchgeführt. Danach folgen die Calculator-Threads, welche das morphologische Skelett und Polygonnetz einer Koralle berechnen. In dem Setup sind es 4 Calculator-Threads. Anschließend ist der Netzwerk-Thread dargestellt. Weiterhin gibt es noch verschiedene Threads für beispielsweise Sound, Fisch-Management und Terrain. Diese werden hier nicht weiter betrachtet. Deutlich zu erkennen ist die periodische Auslastung der Calculator-Threads zu Beginn eines neuen Zeitschrittes. Insgesamt brauchen diese in dem Setup ca. 6 bis 7 Sekunden um die eingehenden Korallen-Polygonnetze zu berechnen. Da die zu berechnenden Aufgaben auf die Threads gleichmäßig aufgeteilt werden, ist die Berechnungszeit von der Anzahl der Threads sowie von den Parametern des Polygonisierungsalgorithmus abhängig. Bei zehn Threads konnte eine Berechnungszeit von 2,5 Sekunden beobachtet werden. Allerdings gingen mit einer höheren Threadzahl deutliche

Lags im Main-Threads einher.

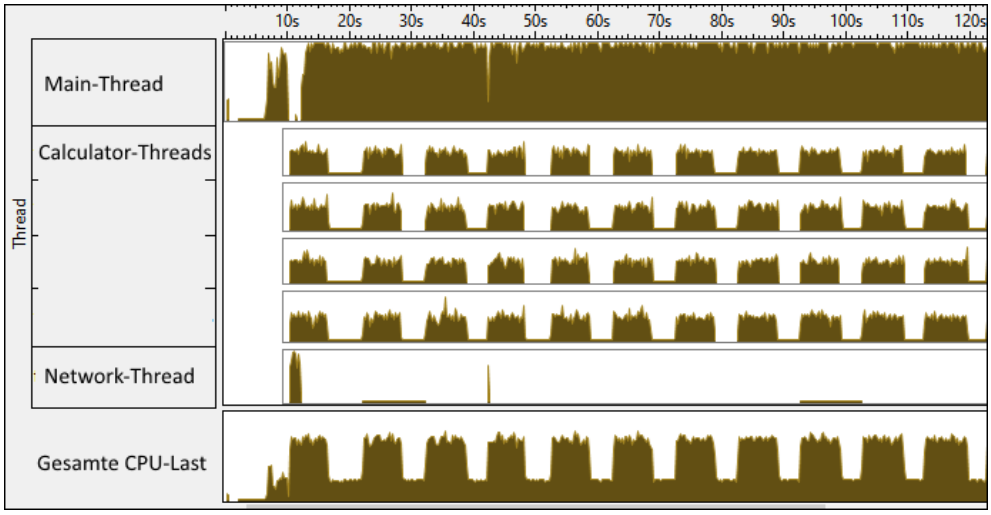


Abbildung 6.12: CPU-Auslastung der unterschiedlichen Threads über ein Zeitraum von 12 Zeitschritten und 4 Calculator-Threads, gemessen mit Intel VTune Amplifier XE 2015

Ferner wurden die Hauptaufgaben der Calculator-Threads analysiert. In einem Messzeitraum von 12 Messschritte über 4 Calculator-Thread kommt eine gesamte Rechenzeit von 120 Sekunden zusammen. Die meiste Zeit verbringen die Threads pausierend und warten auf neue Datensätze. Abbildung 6.13 zeigt die Hauptaufgaben eines Calculator-Threads. Dabei handelt es sich um die Aufgaben: Berechnung des Isowertes, Erstellen des Hashwertes und Entfernungsberechnung. Die Berechnung des Isowertes macht mit 33 % der gesamten Zeit die zeitintensivste Aufgabe aus. Diese Funktion wird verwendet bei der Berechnung des skalaren Feldes und bei der Berechnung der Normale eines Vertex. Die Distanzberechnung und die Ermittlung des Hash-Wertes eines Vertex haben mit 14% die gleiche Dauer. In den drei genannten Aufgaben liegt auch das Hauptpotential zur Optimierung der Polygonnetzerzeugung. Einerseits lassen sich die Funktionen selbst beschleunigen und andererseits die Aufrufanzahl reduzieren. Bezüglich der Aufrufanzahl wurden eingehend geeignete Beschleunigungsverfahren vorgestellt, zum Beispiel die propagationsbasierten Verfahren nach Itoh und Koyamada (1995) und Bajaj et al. (1996). Außerdem wird aktuell zu jedem Zeitpunkt das komplette Polygonnetz neu berechnet. Bei dem massiven Korallen ist dies auch nötig, da sich der obere Mittelpunkt aus den Mittelwert aller Zweige berechnet und somit eine Änderung eines Zweiges unter Umständen auch Einfluss auf die anderen Zweige hat. Bei der verzweigten Koralle wachsen die Zweige unabhängig von einander. Wenn hier nur die betroffenen Teile des Polygonnetzes berechnet werden, ist ein großes

Einsparpotential zu erwartet.

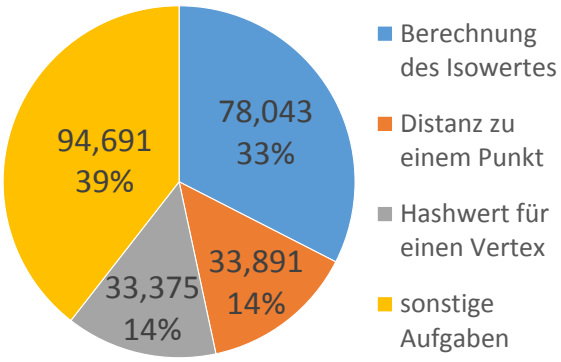


Abbildung 6.13: Hauptaufgaben des Calculator-Threads an einem beispielhaften Thread, gemessen mit Intel VTune Amplifier XE 2015

Kapitel 7

Fazit

Ziel dieser Masterarbeit war es, eine visuelle Repräsentation des Wachstums eines Korallenriffs zu erstellen bei der Korallen als Individuen generiert werden. Zu diesem Zweck wurde ein erweiterbares Framework mit Multi-Threading-Konzept erarbeitet und auf Basis der Grafik-Engine Ogre3D umgesetzt. Für die Verteilung der Korallen im Raum, sowie deren Ausdehnung, wurde als Interprozesskommunikation eine lokale Socket-Anbindung an das Korallenriff-model von Andreas Kubicek Kubicek et al. (2012) implementiert. Das morphologische Skelett der Korallen basiert für verzweigte Korallen auf einem L-System und für massive Korallen auf einer Kurven-Interpolation. Auf Basis dieses Skeletts wird das Polygonnetz mit einem Metaball-Ansatz und Marching Cubes erzeugt.

Durch die Definition weiterer Skelettknoten-Algorithmen lässt sich das System um weitere Korallenarten ergänzen. Das Verfahren zur Ermittlung der Knoten war in dieser Arbeit an Bildvorlagen von verschiedenen Korallenriffen orientiert. Weiterführend wäre es denkbar die Lage der Knoten mit einem empirischen Verfahren zu ermitteln. Dabei werden die Strukturen der Korallen durch Scannen erfasst und daraus ein L-System generiert. Somit kann eine noch höhere Realitätsnähe erreicht werden.

Der Marching Cubes Algorithmus zur Generierung der Polygonnetze aus dem skalaren Feld zeigte sich als sehr geeignet für diesen Anwendungsfall. Die Rechenzeiten vom Übertragen der Position und Ausmaße bis zur Berechnung der einzelnen Vertices einer Korallen sind in einem akzeptablem Maß. Bei vier Threads zur Polygonetzerzeugung und einer mittleren Voxelgröße von 10 für massive und 3 für verzweigte Korallen, lag diese Zeit bei ca. 7 s. Die Hauptaufgabe lag dabei in der Berechnung des Isowertes. Da zu jedem Zeitschritt das gesamte Polygonnetz neu berechnet wird, gibt es hier Potential zur Beschleunigung. Einerseits können beim Marching Cubes Algorithmus leere Vo-

xel von vorn herein umgangen werden. Andererseits können beim Aktualiseren des Polygonnetzes, die Voxel ausgeschlossen werden, welche sich nicht ändern würden. Ferner verspricht eine Auslagerung des Algorithmus auf die Grafikkarte eine Entlastung der CPU und sollte in aufbauenden Arbeiten geprüft werden.

Die Schnittstelle zum Korallenriffmodel wurde bis zu eine Riffgröße von 15 mal 15 m getestet. Dabei lag die Datenübertragungsrate bei ca. 78 kB pro Simulationsschritt. Je größer das Riff ist, umso mehr Polygonnetze müssen auch berechnet werden. Für das Zielsystem hat sich eine Riffgröße von 10 mal 10 m als optimal erwiesen. Durch eine Implementation des Korallenriffmodels nach Kubicek et al. (2012) und des dreidimensionalen C++Frontent innerhalb einer Anwendung, fällt die Netzwerkkommunikation weg. Dadurch kann eine bessere allgemeine Performanz erwartet werden.

Die Eignung von Ogre3D ist für das vorliegende Projekt als bedingt einzuschätzen. Beispielsweise ist die Unterstützung von LOD-Mechaniken und Terraingenerierung nicht zufriedenstellend. Im Vergleich zu anderen Grafik-Engines sind die erzeugten Bilder weniger realitätsnah. Zum Ende dieser Arbeit wurde die Unreal 4 Engine kostenfrei und quelloffen den Nutzern zur Verfügung gestellt. Aufgrund der komplexeren Grafikwerkzeuge empfiehlt sich eine erneute Prüfung der Eignung von Unreal4 als Grafikengine für weiterführende Arbeiten. Die Portierung der implementierten Algorithmen ist denkbar, da beide Engines auf C++ basieren.

Literaturverzeichnis

Abelson, H. und diSessa, A. (1986). Turtle Geometry. MIT-Press, Cambridge.

Bajaj, C. L., Pascucci, V., und Schikore, D. R. (1996). Fast isocontouring for improved interactivity. In Proceedings of the 1996 Symposium on Volume Visualization, VVS '96, pages 39–ff., Piscataway, NJ, USA. IEEE Press.

Blinn, J. F. (1982). A generalization of algebraic surface drawing. ACM Transactions on Graphics, 1(3):235–256.

Boaden, A. (2015). Fishing impacts on the great barrier reef. Onlinezugriff: 09.07.2015, `http://www.coralcoe.org.au/news/fishing-impacts-on-the-great-barrier-reef`.

Bourke, P. (1997). Implicit surfaces. [Online; Zugriff: 07.07.2015].

Brown, J. und Macfadyen, G. (2007). Ghost fishing in european waters: Impacts and management responses. Marine Policy, 31(4):488 – 504.

Bruin, P. d. (2004). Accurate and high-quality surface extraction from medical image data. phdthesis, Delft University of Technology. 90-77595-83-X.

de Araújo, B. R., Lopes, D. S., Jepp, P., Jorge, J. A., und Wyvill, B. (2015). A survey on implicit surface polygonization. ACM Comput. Surv., 47(4):60:1–60:39.

Deussen, O. (2002). Computergenerierte Pflanzen - Technik und Design digitaler Pflanzenwelten. X.media.press. Springer.

Dingliana, J. und Ganovelli, F. (2005). Real-time marching cubes on the vertex shader. EUROGRAPHICS 2005.

Freshwater Algae Culture Collection at the Institute of Hydrobiology (1984). Anabaena catenula - fachb-362. Onlinezugriff: 10.07.2015, `http://algae.ihb.ac.cn/english/algaeDetail.aspx?id=232`.

Gallagher, R. (1991). Span filtering: an optimization scheme for volume visualization of large finite element models. In Visualization, 1991. Visualization '91, Proceedings., IEEE Conference on, pages 68–75, 411.

Gamma, E., Helm, R., Johnson, R., und Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Gelder, A. V. und Wilhelms, J. (1994). Topological considerations in isosurface generation. ACM Transactions on Graphics, 13:337–375.

Gerstner, T. und Rumpf, M. (2000). Multi-resolutional parallel isosurface extraction based on tetrahedral bisection. In Volume Graphics, pages 267–278. Springer.

Hansen, C. D. und Johnsen Chris, R. (2004). The Visualization Handbook. Academic Press.

Heiden, W., Goetze, T., und Brickmann, J. (1993). Fast generation of molecular surfaces from 3d data fields with an enhanced "marching cube" algorithm. Journal of Computational Chemistry, 14(2):246–250.

Hoegh-Guldberg, O. und Smith, G. (1989). The effect of sudden changes in temperature, light and salinity on the population density and export of zooxanthellae from the reef corals stylophora pistillata esper and seriatopora hystrix dana. Journal of Experimental Marine Biology and Ecology, 129(3):279 – 303.

Itoh, T. und Koyamada, K. (1995). Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. Visualization and Computer Graphics, IEEE Transactions on, 1(4):319–327.

Krupp, Friedhelm und Zajonz, U. (2008). Korallenriffe im klimawandel. Natur und Museum, (138):166ff.

Kubicek, A., Muhando, C., und Reuter, H. (2012). Simulations of long-term community dynamics in coral reefs - how perturbations shape trajectories. PLoS Comput Biol, 8(11):e1002791.

Lengyel, E. S. (2010). Voxel-based Terrain for Real-time Virtual Simulations. PhD thesis, Davis, CA, USA. AAI3404919.

Lewis, M. und Parent, R. (2000). An implicit surface prototype for evolving human figure geometry.

Lindenmayer, A. (1968a). Mathematical models for cellular interactions in development i. filaments with one-sided inputs. Journal of Theoretical Biology, 18(3):280 – 299.

Lindenmayer, A. (1968b). Mathematical models for cellular interactions in development ii. simple and branching filaments with two-sided inputs. Journal of Theoretical Biology, 18(3):300 – 315.

Lindenmayer, A. (1971). Developmental systems without cellular interactions, their languages and grammars. Journal of Theoretical Biology, 30(3):455 – 484.

Lizard Island Research Station (2015). A brown colony of acropora muricata from the reef flat, lizard island. Onlinezugriff: 16.07.2015, `http://lifg.australianmuseum.net.au/HotShot.html?resourceId=jhPIEHsN`.

Lorensen, W. E. und Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. SIGGRAPH Comput. Graph., 21(4):163–169.

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., und Balan, G. (2005). Mason: A multiagent simulation environment. Simulation, 81(7):517–527.

Mandelbrot, B. B. (1977). The Fractal Geometry of Nature. W. H. Freeman and Company.

McClanahan, T., Muthiga, N., Kamukuru, A., Machano, H., und Kiambo, R. (1999). The effects of marine parks and fishing on coral reefs of northern tanzania. Biological Conservation, 89(2):161 – 182.

Miguet, S. und Nicod, J.-M. (1995). A load-balanced parallel implementation of the marching-cubes algorithm. In Proceedings of high performance computing symposium, volume 95, pages 229–39.

Mobley, C. D. (2001). Radioactive transfer in the ocean. Academic Press.

Moore, D. und Warren, J. D. (1991). Mesh displacement: An improved contouring method for trivariate data. Citeseer.

Muraki, S. (1991). Volumetric shape description of range data using "blobby model". In ACM SIGGRAPH Computer Graphics, volume 25, pages 227–235. ACM.

Nakata, S. und Sakamoto, Y. (2015). Particle-based parallel fluid simulation in three-dimensional scene with implicit surfaces. The Journal of Supercomputing, 71(5):1766–1775.

National Park Service U.S. Department of the Interior (2015). Porites lutea. mound coral. Onlinezugriff: 01.07.2015, http://www.botany.hawaii.edu/basch/uhnpscesu/htms/NPSAcorl/fish_pops/porites/coral16.htm.

Newman, T. S. und Yi, H. (2006). A survey of the marching cubes algorithm. Computers and Graphics, 30(5):854 – 879.

Nguyen, H. (2007). Gpu Gems 3. Addison-Wesley Professional, first edition.

Nishimura, H., Hirai, M., Kawai, T., Kawata, T., Shirakawa, I., und Omura, K. (1985). Object modelling by distribution function and a method of image generation. Transactions of the IEICE Japan J68-D, (4):718–725.

Oeltze, S. und Preim, B. (2004). Visualization of anatomic tree structures with convolution surfaces. In Proceedings of the Sixth Joint Eurographics - IEEE TCVG Conference on Visualization, VISSYM'04, pages 311–320, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Peter, J. und Eyer, D. (2001). Fraktalwelt: Wachstumssimulation. Onlinezugriff: 10.07.2015, http://www.fraktalwelt.de/lsys/seite16.html.

Prusinkiewicz, P. und Lindenmayer, A. (1996). The Algorithmic Beauty of Plants. Springer-Verlag New York, Inc., New York, NY, USA.

Raman, S. und Wenger, R. (2008). Quality isosurface mesh generation using an extended marching cubes lookup table. In Computer Graphics Forum, volume 27, pages 791–798. Wiley Online Library.

Randall, R. H. und Myers, R. F. (1983). Guide to the Coastal Resources of Guam: Vol. 2 The Corals. University of Guam Marine Laboratory (1983). Onlinezugriff: 10.08.2015, http://www.guammarinelab.com/coral/coral1.html#animals.

Rock, J., Gupta, T., Thorsen, J., Gwak, J., Shin, D., und Hoiem, D. (2015). Completing 3d object shape from one depth image. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

Sander, Sören und Fritsching, U. (Aachen, Germany, 28.9 - 2.10 2014). Analyse der bewegung von partikelkollektiven im elektroabscheider mittels piv. Chem. Ing. Tech. 2014, 86, No. 9, 1588-1589.

Scholz, M., Bender, J., und Dachsbacher, C. (2015). Real-time isosurface extraction with view-dependent level of detail and applications. Computer Graphics Forum, 34(1):103–115.

Schreiner, J., Scheiclegger, C., und Silva, C. T. (2006). High-quality extraction of isosurfaces from regular and irregular grids. Visualization and Computer Graphics, IEEE Transactions on, 12(5):1205–1212.

Schuhmacher, H. (1988). Korallenriffe. BLV Verlagsgesellschaft mbH, München.

Tory, M., Rober, N., Moeller, T., Celler, A., und Atkins, M. (2001). 4d space-time techniques: a medical imaging case study. In Visualization, 2001. VIS '01. Proceedings, pages 473–592.

Treece, G., Prager, R., und Gee, A. (1999). Regularised marching tetrahedra: improved iso-surface extraction. Computers and Graphics, 23(4):583 – 598.

Trembilski, A. (2001). Two methods for cloud visualisation from weather simulation data. The Visual Computer, 17(3):179–184.

Ulrich, C., Grund, N., Derzapf, E., Lobachev, O., und Guthe, M. (2014). Parallel iso-surface extraction and simplification. WSCG Communications proceedings, 2014.

Weber, G., Kreylos, O., Ligocki, T., Shalf, J., Hagen, H., Hamann, B., und Joy, K. (2001). Extraction of crack-free isosurfaces from adaptive mesh refinement data. In Ebert, D., Favre, J., und Peikert, R., editors, Data Visualization 2001, Eurographics, pages 25–34. Springer Vienna.

Weigle, C. und Banks, D. C. (1998). Extracting iso-valued features in 4-dimensional scalar fields. In Proceedings of the 1998 IEEE Symposium on Volume Visualization, VVS '98, pages 103–110, New York, NY, USA. ACM.

Wilhelms, J. und Van Gelder, A. (1992). Octrees for faster isosurface generation. ACM Trans. Graph., 11(3):201–227.

Wyvill, G., McPheeters, C., und Wyvill, B. (1986). Data structure forsoft objects. The Visual Computer, 2(4):227–234.

Yim, P. J., Boudewijn, G., Vasbiner, C., Ho, V. B., und Choyke, P. L. (2003). Isosurfaces as deformable models for magnetic resonance angiography. IEEE Trans. Med. Imaging, 22(7):875–881.

Yu, Y. und Wang, Y. (2014). Study of algorithms for interaction between flowing water and complex terrain in virtual environment. In Wen, Z. und Li, T., editors, Practical Applications of Intelligent Systems, volume 279 of Advances in Intelligent Systems and Computing, pages 337–349. Springer Berlin Heidelberg.