UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Physikalisch basierte Verwitterung von Früchten

Bachelorarbeit

zur Erlangung des Grades einer Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Julia Chevalier

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Gerrit Lochmann

Koblenz, im April 2016

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Bachelorarbeit

Julia Chevalier

(Mat. Nr. 212100779)

Thema: Simulation physikalisch basierter Verwitterung von Früchten

In vielen Bereichen der Computergraphik soll die reale Welt simuliert werden. Virtuelle Welten sind bereits weit verbreitet und finden zum Beispiel in der Spieleindustrie Anwendung. Allerdings wird die Natur häufig makellos dargestellt. In der realen Welt sind nicht alle Pflanzen saftig und grün und das Obst reif und prall.

In dieser Arbeit wird die Verwitterung von Obst genauer betrachtet. Dabei werden insbesondere die biologischen und physikalischen Formveränderungen beobachtet. Interessant an Obst ist, dass nicht alle Komponenten gleich verwittern. Die Schale verhält sich anders als das Fruchtfleisch oder die Kerne. Jede dieser Komponenten liegen unterschiedliche Eigenschaften zugrunde, welche beim Welken verschiedene Einflüsse auf die Obstform haben.

Ansatz dieser Arbeit ist basierend auf der Materie-Punkt Methode einen Algorithmus zu entwickeln, der die Verwitterung von Obst und den dabei zugrundeliegenden physikalischen Kräften simuliert.

Die inhaltlichen Schwerpunkte der Arbeit sind:

1. Recherche über Simulation von Verwitterung
2. Einarbeitung in die programmiertechnischen und physikalischen Grundlagen
3. Entwicklung/Auswahl eines Verfahrens
4. Implementierung des Verfahrens
5. Demonstration der Ergebnisse
6. Bewertung und Dokumentation der Ergebnisse

Koblenz, 16.10.2015

Chevalier

– Julia Chevalier –

S.Müller

– Prof. Dr. Stefan Müller –

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Zusammenfassung

Der natürliche Prozess der Verwitterung ist ein komplexer Vorgang, der von unterschiedlichsten Parametern beeinflusst wird. Hauptbestandteil dieses Prozesses ist das Zusammenziehen des Fruchtvolumens infolge von Wasserverlust durch Transpiration sowie die Veränderung der Fruchtfarbe und Oberfläche. Es wurden bereits Verfahren entwickelt, die diese Eigenschaften mit Hilfe von Parametrisierung sowie physikalischer Ansätze simulieren. Die in dieser Arbeit erstellte Anwendung simuliert das Fruchtfleisch durch ein Tetraedernetz und die Veränderung der Haut mit Hilfe von dynamischer Texturanpassung. Der entwickelte Algorithmus arbeitet in linearer Laufzeit und seine Ergebnisse werden anhand selbst erstellter Fruchtmodelle präsentiert.

Abstract

The natuaral withering is a complex process and influenced by many different parameters. An essential part is the shrinking fruit flesh initiated by waterloss due to transpiration. The skin changes color and surface structure as well. These properties where already simulated in parameterised and physical based algorithms. This work simulates the flesh with a shrinking tetrahedral mesh and the surface changes through dynamic changing textures. The developed algorithm runs in linear time and the results are presented with self created fruit models.

1 Einleitung

In der Computergrafik werden in vielen Bereichen reale Welten simuliert. Simulationen ermöglichen es physikalische Vorgänge sowie Naturerscheinungen am Computer grafisch darzustellen. Der Vorteil darin besteht, dass Szenarien kostengünstig und zeitsparend erstellt werden können. Außerdem werden dabei bestimmte Eigenschaften wiederholte Male neu ausgetestet. Des Weiteren können Simulationen für die Film- oder Spieleindustrie verwendet werden. In diesen Bereichen ist die naturgetreue Darstellung von realen Vorgängen erstrebenswert. Leider ist es häufig der Fall, dass die Natur makellos modelliert wird. Beispielsweise sind in der realen Welt die Blätter nicht alle saftig und grün und das Obst reif und prall.

Diese Arbeit beschäftigt sich mit der Verwitterung von Früchten, welches ein komplexer biologischer Vorgang ist, der durch viele äußerliche Faktoren beeinflusst wird. Beispielsweise spielen Temperaturen und Luftfeuchtigkeit beim Zerfallsprozess eine wichtige Rolle. Beobachtet man diesen Vorgang, kann man feststellen, dass sich die Frucht deformiert, die Haut Falten bildet und die Farbe verändert. Außerdem werden Schimmelpilze oder andere Verdunklungen auf der Fruchtoberfläche sichtbar.

Die vorliegende Arbeit setzt sich mit der Simulation der Verwitterung von Früchten auseinander. Allerdings werden die biologischen Prozesse auf ihre form- und farbverändernden Eigenschaften reduziert. Als Basis dient die der Verwitterung zugrundeliegende Transpiration. Sie ist dafür verantwortlich, dass die Frucht, die größtenteils aus Wasser besteht, stark an Volumen abnimmt und sich somit deformiert. Die Haut bildet aufgrund ihrer geringen Elastizität bei der Deformation des Fruchtfleisches Falten.

Diese Arbeit simuliert das Fruchtfleisch und deren Formveränderung mithilfe eines Tetraedermodells. Die maßgebliche Idee dabei liegt darin, das Tetraedervolumen nach dem Verlieren einer bestimmten Menge Wasser um genau diese Menge zu verkleinern. Die Veränderung der Haut, also deren Faltenbildung und Farbverdunklung sowie Fleckenbildung, werden mithilfe von Texturen dargestellt. Auf der zu Beginn glatten Fruchthaut bildet sich im Laufe der Simulation eine Oberflächenstruktur, die durch Bump-Mapping implementiert ist.

Im Folgenden wird der genaue Aufbau der vorliegenden Arbeit beschrieben. Kapitel 2 präsentiert andere Arbeiten, die sich mit natürlichem Zerfall und der Verwitterung von Früchten beschäftigen. In diesem Bereich konnten bereits sehr gute Ergebnisse durch physikalische Simulationen, aber auch parameterbasierte Verfahren erreicht werden.

Kapitel 3 führt in die Grundlagen von bisher erarbeiteten Verwitterungsprozessen ein. Außerdem werden die Begriffe Delaunay und Voronoi sowie deren zugrundeliegenden Theorien kurz eingeführt. Im nächsten Kapitel wird der von mir entwickelte Ansatz vorgestellt, der es ermöglicht Tetraeder auf ein bestimmtes Volumen zu skalieren. Des Weiteren setzt sich die-

ses Kapitel mit der Darstellung von Faltenbildung durch Normalen, deren Intensität verstärkt wird, auseinander. Einen Überblick über das von mir entwickelte Programm, die verwendeten Bibliotheken und wichtige Funktionen gibt das Kapitel 5. Das darauf folgende Kapitel präsentiert die Ergebnisse meiner Simulation auf eigens erstellten Modellen. Im letzten und abschließenden Kapitel wird die Arbeit reflektiert und ein Ausblick über mögliche Erweiterungen und Verbesserungen vorgestellt.

2 Verwandte Arbeiten

Unvollkommenheit
Simulation von Verwitterung bedeutet gleichzeitig die Simulation von Unvollkommenheit. In der Computergrafik gibt es bereits mehrere Ansätze dies darzustellen. Dazu gehören beispielsweise Alterungsprozesse, Gebäudezerfall und Vertrocknung. [MG08] zeigt einen allgemeinen Ansatz Alterung und Verwitterung durch grundlegende Methoden wie Textur, Reflektionscharakteristiken und Geometrie darzustellen. Die Unvollkommenheit, die durch natürliche Vorgänge entsteht, wurde beispielsweise an der Simulation von Holzalterung [YFCT08] oder verwitterndem Gras [JHW09] ausgearbeitet.

Arbeiten die sich ähnlich zu dieser Abhandlung auf die Verwitterung von Früchten konzentrieren und dieser Arbeit als Grundlage dienen, sind die von Kider und Liu. Kider und andere entwickelten in [JRB11] eine Methode, um die Seneszenz und das Verfaulen von Früchten zu simulieren. Die Methode benutzt eine verformbare Körper Simulation sowie Kleidungssimulation, um die Früchte unter Einfluss vieler Parameter, wie zum Beispiel Schalendicke, Temperatur und Luftfeuchtigkeit, verwittern zu lassen.

Das Modellieren von verwitternden Früchten wurde in [LCW+12] in mehreren Schritten gelöst. Der dort entwickelte Algorithmus arbeitet in verschiedenen Schichten. Die Veränderung des Fruchtfleisches und die Entwicklung von Schimmel werden durch die Finite Element Method berechnet. In Abschnitt 3.2 wird etwas genauer auf den in [LCW+12] entwickelten Algorithmus eingegangen.

Außerdem wurde ein Verfahren entwickelt, um die Dehydration von Früchten mit Kern und Schale (z.B. Pflaumen) zu simulieren. Dieser Ansatz wurde ebenfalls mit Hilfe eines FEM Frameworks gelöst [LYC+15].

3 Grundlagen

3.1 Biologischer Hintergrund zur Verwitterung von Früchten

Abbildung 1: Einige Früchte vor und nach der Verwitterung.(Quelle [LCW+12])

Das Verfaulen ist ein wichtiger Prozess zur Erhaltung einer Frucht durch Reproduktion. Durch den Zerfall des Fruchtfleisches wird der Samen freigelegt und gleichzeitig genährt, damit daraus eine neue Pflanze mit Früchten entstehen kann. Früchte können zwar jederzeit von Krankheiten befallen werden, allerdings treten sie häufiger erst nach dem Fallen oder Ernten mit Beginn des Zerfalls auf.

Schimmel und Bakterien
Da reife Früchte eine feuchte und nährstoffreiche Schale besitzen, können diese leicht von Schimmelpilzen und Bakterien befallen werden. Dies sind hervorragende Bedingungen auf der die eben genannten Biomassen wachsen und gedeihen können. Dadurch verliert die Frucht Nährstoffe, welche den Wachstumsprozess der Biomassen eindämmen. Trotzdem können auch Faktoren wie Luftfeuchtigkeit oder Temperatur deren Wachstum beeinflussen.

Transpiration
Früchte haben einen sehr hohen Wassergehalt (siehe Tabelle 1). Bei der Verwitterung ist ein deutlicher Zerfall der Strukturen und dem Volumen der Frucht zu erkennen. Abbildung 1 zeigt dazu einige Beispiele. Dieses Verhalten entsteht durch das Transpirieren des Wassers durch die Fruchthaut. Durch das kontinuierliche Verlieren von Wasser beginnt die Frucht zu schrumpfen und zieht sich zusammen. Wie auch im letzten Abschnitt erklärt, kann dieser Prozess durch unterschiedliche Bedingungen beeinflusst werden. Dennoch sind die stärksten Unterschiede durch die Porosität der Fruchthaut zu erkennen. Tabelle 1 zeigt Transpirationsfaktoren unterschiedlicher Früchte.

Frucht	Wassergehalt	K_s
Apfel	84%	16.5
Trauben	83%	122
Orange	87%	121
Pflaume	84%	127

Tabelle 1: Liste von Früchten, deren Wasserhalt und Transpirationskoeffizient K_s (in $mg/kg \times s \times MPa$). Nach[BF96] und [BH97].

3.2 Anregung

Der von [LCW+12] entwickelte Algorithmus diente als Inspiration und Gedankenanstoß für den von mir entwickelten Algorithmus. Aus diesem Grund wird in diesem Unterkapitel ein kurzer Einblick in das genannte Verfahren gegeben, auch wenn mein eigener Ansatz im Kern stark davon abweicht. Wie bereits ausgeführt wird in dem von [LCW+12] entwickel-

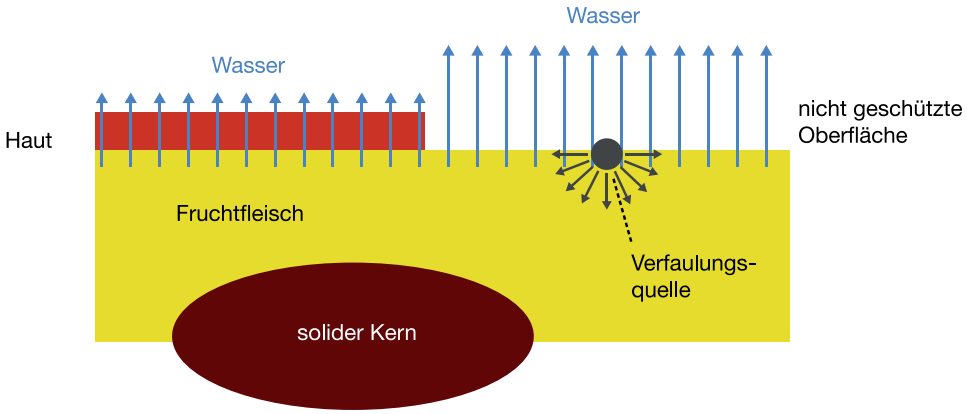


Abbildung 2: Schematische Darstellung der Fruchtzusammensetzung in Bezug auf Verwitterung. Vorlage:[LCW+12]

ten Ansatz in verschiedenen Schichten gearbeitet. Die Idee dabei ist, dass Früchte aus den folgenden drei Komponenten zusammengesetzt sind: dem Fruchtfleisch, der Haut und den Kernen oder Samen (siehe Abbildung 2). Bei der Verwitterung verhält sich jede Komponente unterschiedlich. Während sich der Kern bei der Verwitterung kaum verändert, deformiert sich das Fruchtfleisch aufgrund von Wasserverlust drastisch. Dabei ist zu erkennen, dass Regionen, die nicht von der Fruchthaut geschützt sind, deutlich schneller Wasser verlieren. Die Haut ist im Vergleich zum Fruchtfleisch unelastisch und verhält sich wie eine Membran, ähnlich zu Kleidung. Des Weiteren gehen Liu und andere auf die äußerlichen Veränderungen der Frucht ein. Durch den Wasserverlust wird die Frucht dunkler und durch die Schimmelbildung entstehen weitere dunkle bis schwarze Stellen.

Finite Element Methode
Umgesetzt wird das Ganze durch ein Framework basiert auf der Finite Element Methode (FEM). Dies ist eine Lösungsmethode, die komplexe physikalische Vorgänge, wie z.B. Deformation mit Hilfe von Diskretisierung des zu verformenden Objektes löst. Das Eingabeobjekt wird in *finite Elemente*, also eine endliche Anzahl von Elementen geteilt. Somit können die auf das Objekt wirkenden Kräfte und Funktionen, welche sich meistens aus Differentialgleichungen zusammensetzen, mit Hilfe von numerischen Lösungsverfahren berechnet werden. Auf weitere Details und der dazugehören Kontinuumsmechanik der FEM wird an dieser Stelle nicht eingegangen. Die Arbeit von Zienkiewicz und anderen [ZTZ05] bietet eine ausführliche Erklärung über deren Grundlage. **Algorithmus**

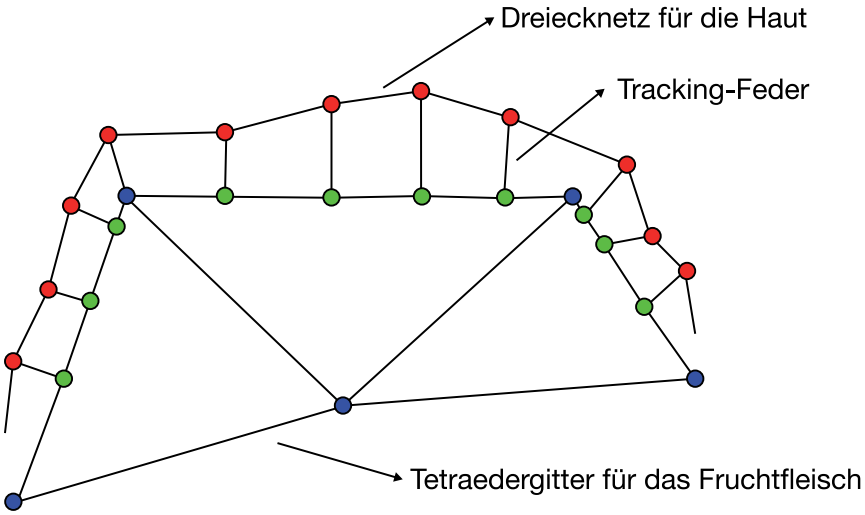


Abbildung 3: Schemadarstellung der Tracking-Punkte (grün) sowie der Tracking-Feder zwischen einem Tetraedernetz (blaue Punkte) und einem Dreiecknetz (rote Punkte).Vorlage: [LCW+12]

In der Arbeit von Liu wird die Eingabefrucht mit Hilfe eines Oberflächennetzes (i.F. Oberflächenmesh) bestehend aus Dreiecken und eines Volumennetz (i.F. Volumenmesh) bestehend aus Tetraedern modelliert. Dabei repräsentiert das Oberflächenmesh die Fruchthaut und das Volumenmesh das Fruchtfleisch und die Kerne. Außerdem wird das Volumenmesh aus dem Oberflächenmesh generiert. Danach wird die Auflösung des Oberflächenmeshes deutlich verfeinert und quasi als Haut auf das Fruchtfleisch gelegt. Darauf hin wird das Dreiecknetz und das Tetraedernetz mit elastischen Federn verbunden, siehe Abbildung 3. Wenn sich nun das Fruchtfleisch zusammenzieht, wird die Bewegung der unelastischen Haut durch Tracking der Federn ermittelt. Die entstehenden Falten in der Fruchthaut werden dadurch simuliert. Diese Idee wurde von der Kleidungssimulation [BFA02], welche ebenfalls mit dem Masse-Feder-System arbeitet, adaptiert.

Im Folgenden wird die Schrittabfolge des resultierenden Algorithmus skizziert:

- Laden des Eingabe-Dreiecksgitter und Erstellen des Tetraedergitters
- Verfeinerung des Dreiecksgitter, welches als Haut dient
- Für jeden Zeitschritt:
  - Berechnung der Veränderung des Fruchtfleisches mit Hilfe des FEM Framework
  - Berechnung der Veränderung der Haut in Bezug auf das Fruchtfleisch
  - Berechnung weiterer Hautveränderungen
  - Aktualisierung des Wassergehaltes und der Schimmelbildung
  - Berechnung der Farbe auf Basis des Wassergehalts und Rendern die Szene

3.3 Delaunay und Voronoi

Die Delaunay-Triangulierung ist ein Verfahren um Graphen zu planarisieren, das heißt sicherzustellen, dass sich keine Kanten im Graphen überschneiden. Die folgende Erklärung bezieht sich auf die Delaunay-Triangulierung im dreidimensionalen Raum.

Gegeben ist eine Menge an dreidimensionalen Punkten V und ein k-Simplex σ, dessen Punkte sich in V befinden. Bildet man nun die Umkugel der Punkte in σ und es befinden sich keine anderen Punkte aus V in dieser Kugel, dann erfüllt σ die Delaunay-Eigenschaft. Eine Delaunay-Tetraedisierung D von V beschreibt alle Punktgruppen aus V, die die Delaunay-Eigenschaft erfüllen.

Durch ein Voronoi Diagramm kann eine Delaunay-Tetraedisierung erreicht werden. Gegeben ist die im letzten Abschnitt beschriebene Punktemenge V. Eine Voronoizelle beschreibt den Bereich um einen Vertex $p \in V$ wobei kein anderer Punkt aus V näher zu den Punkten in der Voronoizelle liegt als p.

3.4 Bump-Mapping

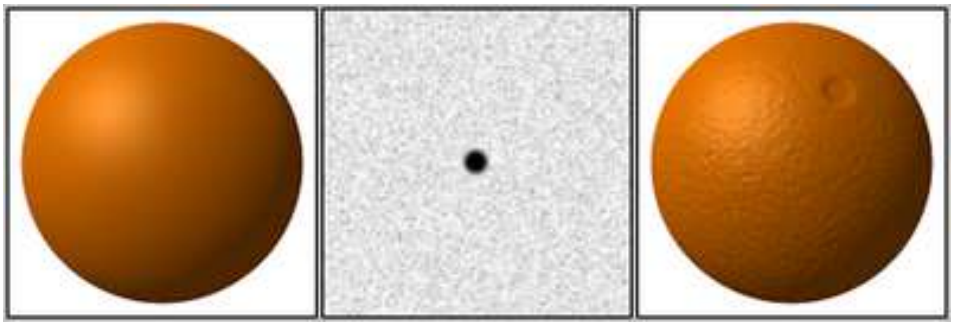


Abbildung 4: Links: Objekt mit glatter Oberfläche, Mitte: Höhentextur, Rechts: Objekt nach Bump-Mapping mit gegebener Höhentextur
Quelle: https://en.wikipedia.org/wiki/File:Bump-map-demo-full.png (Einsicht am 04.04.2016, 13:04)

Bump-Mapping ist eine Technik in der Computergrafik, um glatten Objektoberflächen das Aussehen einer Oberflächenstruktur zu verleihen. Normalerweise müssten feine Oberflächenstrukturen durch viele kleine Polygone dargestellt werden, was sehr rechenaufwendig ist. Bump-Mapping verwendet eine Höhentextur, welche der Bestimmung der Oberflächennormale dient. Anstatt die Normale aus der Geometrie des Objektes zu bestimmen, wird diese aus der Höhentextur berechnet. Dies ist sehr einfach und effizient zu berechnen, da keine neuen Geometrien erzeugt werden müssen. Allerdings besteht der Nachteil beim Bump-Mapping dabei, dass die Silhouette der Objekte dennoch glatt ist. Nur durch die Beleuchtung mit den berechneten Normalen entsteht der Anschein von Struktur. Abbildung 4 zeigt das Beispiel einer Kugel mit und ohne Bump-Mapping, sowie die verwendete Höhentextur.

4 Eigener Ansatz

Der von mir entwickelte Algorithmus basiert auf Grundlage von Wassergehalt und Volumen. Um das Problem der Verwitterung zu vereinfachen, habe ich es auf die zwei grundlegenden Eigenschaften reduziert: die Schrumpfung des Fruchtfleisches sowie die farbliche und strukturelle Veränderung der Haut. Das folgende Kapitel beschreibt, die von mir entwickelten Ideen, um diese zwei Prozesse zu simulieren.

4.1 Das Fruchtfleisch

Die Deformation des Fruchtfleisches ist primär auf den Wasserverlust durch Transpiration zurückzuführen. Dazu wurde in dieser Arbeit – inspiriert durch die in 3.2 erklärte Arbeit – auf einem Tetraedernetz gearbeitet. Das Fruchtfleisch besteht somit aus vielen Tetraeder-Elementen, welche zusammen ein Tetraedernetz aufspannen. Jeder Tetraeder verliert pro Zeitschritt eine prozentuale Menge an Wasser. Demnach muss das Volumen um den Wasserverlust angepasst werden. Da Wasser die Dichte von 1 besitzt, verliert ein Tetraeder prozentual die gleiche Menge an Volumen wie Wasser. Wenn zum Beispiel ein Tetraed-Erelement 1 Prozent Wasser verliert, dann verkleinert der hier entwickelte Algorithmus den Tetraeder so, dass er 99 Prozent des Anfangsvolumen hat.

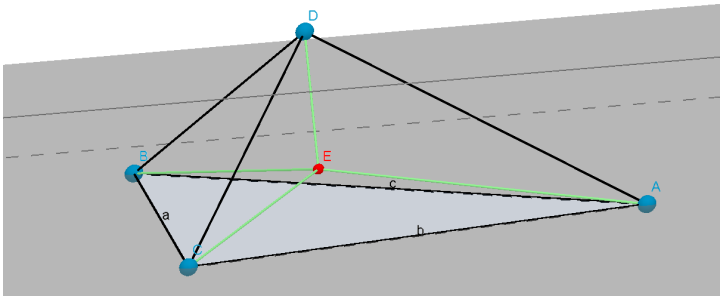


Abbildung 5: Zeichnung des Tetraeders M

Berechnung: Gegeben ist der Tetraeder M mit den Punkten A, B, C, D, dem Mittelpunkt E und dessen Volumen V_m (siehe Abbildung 5). M soll so verändert werden, dass das Volumen des neuen Tetraeders $V_n = V_m - (V_m * (1 - \phi_m))$ mit $V_n < V_m$ entspricht. ϕ_m ist dabei der prozentuale Wassergehalt des Tetraeders M. Für den neuen Tetraeder N müssen nun die Punkte H, I, J, K berechnet werden. Diese ergeben sich aus der Verschiebung von A, B, C, D in Richtung E. Die die Punkte H, I, J, K lassen sich berechnen durch:

$$
\begin{aligned}
H &= t * A + (1-t) * E \\
I &= t * B + (1-t) * E \\
J &= t * C + (1-t) * E \\
K &= t * D + (1-t) * E
\end{aligned}
\tag{1}
$$

Es fehlt der Interpolationswert t, um zu bestimmen wie weit die Punkte A, B, C, D Richtung Mittelpunkt verschoben werden müssen, wobei $0 > t > 1$ gilt. Ein Beispiel für den Wert $t = 0.5$ zeigt Abbildung 6. Der Wert

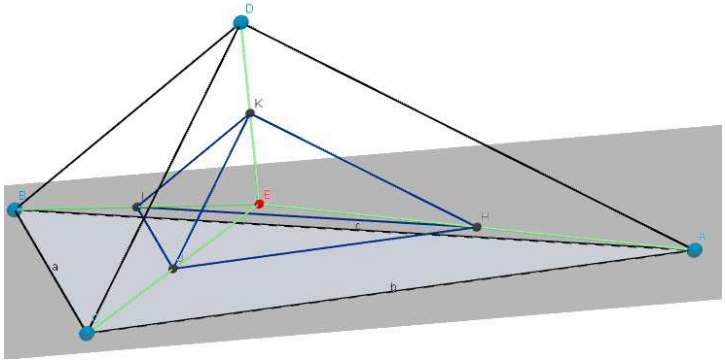


Abbildung 6: Zeichnung des Tetraeders M und des Tetraeders N bei einer Schrumpfung mit $t = 0.5$

für t kann mit Hilfe der bereits gegebenen Volumina berechnet werden. Für die Berechnung des Volumen V_n des Tetraeders N gilt:

$$
V_n = \tfrac{1}{6} * \overrightarrow{HK} * (\overrightarrow{HI} \times \overrightarrow{HJ}) \tag{2}
$$

Es ergibt sich folgende Rechnung, um t mit Hilfe von den Gleichungen aus (1) zu bestimmen:

$$
\begin{aligned}
6 * V_n &= \overrightarrow{HK} * (\overrightarrow{HI} \times \overrightarrow{HJ}) \\
&= ((t * D + (1-t) * E) - (t * A + (1-t) * E)) * \\
&\quad [(t * B + (1-t) * E) - (t * A + (1-t) * E) \times \\
&\quad (t * C + (1-t) * E) - (t * A + (1-t) * E)] \\
&= (t * D - t * A) * [(t * B - t * A) \times (t * C - t * A)] \\
&= t[(D - A) * t^2[(B - A) \times (C - A)]] \\
&= t^3[(D - A) * [(B - A) \times (C - A)]]
\end{aligned}
\tag{3}
$$

Es gilt:

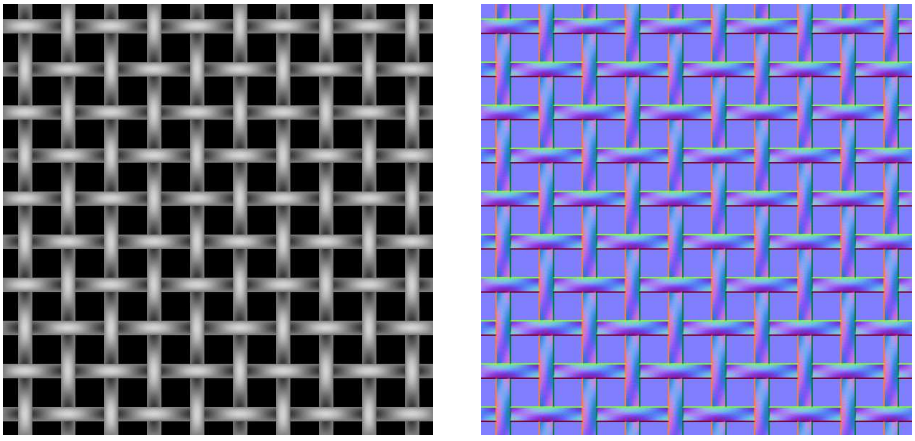
$$6 * V_m = [(D - A) * [(B - A) \times (C - A)]] \tag{4}$$

Deshalb kann t berechnet werden durch:

$$t = \sqrt[3]{V_n / V_m} \tag{5}$$

4.2 Die Haut

Im Gegensatz zur in Kapitel 3.2 entwickelten Methode wird in meinem Ansatz die Haut mit Hilfe von Texturen dargestellt. Die Fruchthaut entspricht dabei der obersten Schicht des Tetraedernetzes. Diese ergibt ein Dreiecknetz auf das eine Textur gelegt wird. Da die Fruchthaut weniger elastisch als das Fruchtfleisch ist, bilden sich beim Zusammenziehen des Fruchtfleisches immer stärkere Falten auf der Haut. Die Falten werden in meinem Ansatz durch Bump-Mapping (siehe Kapitel 3) simuliert. Gegeben ist eine Heightmap, die die Oberflächenstruktur der verschrumpelten Frucht darstellt. Aus dieser Heightmap werden die jeweiligen Normalen pro Fragment berechnet. Dafür wird pro Fragment zunächst der Gradient in vertikaler und horizontaler Richtung ermittelt. Dieser wird mit Hilfe des Sobel-Operators bestimmt. Anschließend wir das Kreuzprodukt der beiden Gradienten berechnet. Abhängig von einem Wert γ werden die Normalen stärker oder schwächer. Abbildung 7 zeigt ein Beispiel einer Höhenkarte und eine mit Hilfe des Sobel-Operators erstellten Normalenkarte.



(a) Höhenkarte **(b)** Normalenkarte

Abbildung 7: Links: Eine Höhenkarte, Rechts: Die mit Hilfe des Sobel-Operators erstellte Normalenkarte. Die Normalenkarte wurde mit Normal-Map Online erstellt.[1]

[1] `http://cpetry.github.io/NormalMap-Online/`

Es ergibt sich folgende Rechnung: Gegeben ist die Höhentextur F. Um die Normale n an der Stelle i, j zu bestimmen, wird zunächst der Höhenunterschied der benachbarten Pixel einmal in vertikaler (d_a) und horizontaler (d_b) Richtung berechnet:

$$d_a = F_{i+1,j} - F_{i-1,j} d_b = F_{i,j+1} - F_{i,j-1} \tag{6}$$

Danach werden folgende Vektoren erstellt:

$$v\vec{d}_a = \begin{pmatrix} 1 \\ 0 \\ \gamma * d_a \end{pmatrix} v\vec{d}_b = \begin{pmatrix} 0 \\ 1 \\ \gamma * d_b \end{pmatrix} \tag{7}$$

Diese werden anschließend zu den Vektoren $v\tilde{d}_a, v\tilde{d}_b$ normalisiert.

Aus dem Kreuzprodukt der erstellten Vektoren berechnet sich nun die Normale:

$$n_{i,j} = v\tilde{d}_a \times v\tilde{d}_b \tag{8}$$

Durch das Anpassen von γ kann die Oberflächenstruktur intensiviert werden.

5 Implementation

In diesem Kapitel wird das in dieser Arbeit entwickelte Programm erläutert. Zuerst werden die verwendeten Bibliotheken und Programme präsentiert. Des Weiteren wird ein Überblick über die von mir entwickelte Datenstruktur gegeben. Diese ergibt sich aus der Aufteilung der Tetraedermodelle aus Kapitel 5.3, welches sich mit einem Tool zur Erstellung von Tetraedermodellen befasst. Anschließend wird der für den Algorithmus notwendigen Aufbau der Modelle und deren Vorverarbeitungsschritte ausgearbeitet. Danach folgt eine Zusammenfassung des Programmablaufs und der Hauptrenderschleife, in der das Fruchtfleisch deformiert wird. Zum Schluss werden die wichtigsten Berechnungen des zur Laufzeit kompilierten Fragment-Shader erörtert, der für die Veränderung der Haut notwendig ist. Eine schematische Darstellung des entwickelten Programms bietet Abbildung 8.

5.1 Eingebundene Bibliotheken und Programme

Die Anwendung wurde mit OpenGL mit GLSL 3.3 in der Sprache C++ und in der Entwicklungsumgebung Eclipse Mars entwickelt. Dazu verwende ich das Framework der Arbeitsgruppe Müller der Universität Koblenz Landau. Dieses verwendet GLEW[2], um OpenGL in C++ einzubinden und CMake[3], um die ausführbaren Dateien zu erstellen. Darüber hinaus verwende ich die im Framework enthaltene Librarie CVK und die CG1_Tools, welche das Erstellen von Shader-Programmen und Einbinden von Texturen vereinfachen.

Des Weiteren wurde für das Erstellen und Berechnen von Vektoren und Matrizen die OpenGL Mathematics (GLM)[4] Bibliothek verwendet. Die Erzeugung des Fensters, welches die zu rendernde Szene zeigt, sowie das Drehen der Kamera mit Hilfe der Maus wird durch GLFW[5] ermöglicht. Zur Erstellung der Eingabemodelle wurde die Bibliothek Tetgen[6] verwendet, welche in Kapitel 5.3 genauer erläutert wird.

[2] http://glew.sourceforge.net/
[3] https://cmake.org/
[4] http://glm.g-truc.net/0.9.7/index.html
[5] http://www.glfw.org/
[6] http://wias-berlin.de/software/tetgen/

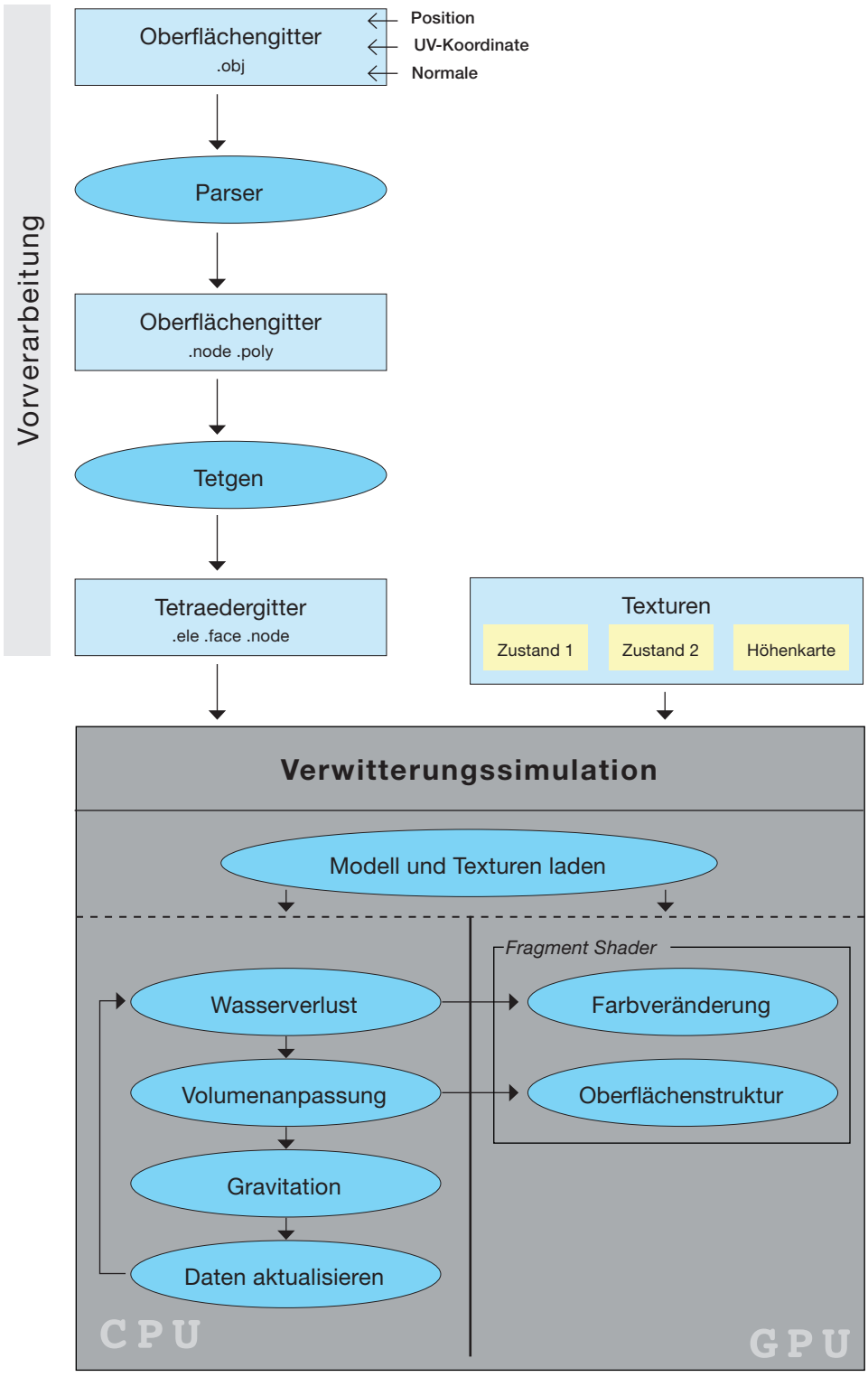


Abbildung 8: Schematische Darstellung des in dieser Arbeit entwickelten Simulationsprogramms

5.2 Datenstruktur

Der Schwerpunkt dieser Arbeit liegt darin, das Schrumpfen des Fruchtfleisches zu simulieren. Dabei wird das Fruchtfleisch durch ein Tetraedernetz dargestellt und besteht dabei aus vielen Tetraedern. Jeder Tetraeder hat vier Punkte (Nodes) und vier Flächen (Faces). Aus diesen Eigenschaften hat sich die folgende Datenstruktur entwickelt.

Die Klasse *TetMesh* repräsentiert das Fruchtfleisch einer Frucht durch ein Tetraedernetz. Alle Punkte, Flächen und Tetraeder werden in dieser Klasse referenziert. Die wichtigsten Methoden der Klasse werden in Unterkapitel 5.5 erläutert. Ein Tetraeder-Element wird durch die Klasse *TetElement* modelliert. Dabei kennt ein *TetElement* seine vier Punkte *TetNode* und Flächen *TetFace*.

Ein *TetFace* wird durch die drei *TetNode*-Objekte aufgespannt. Jedes *TetFace* speichert in einer Ganzzahlvariable die Information, ob es sich an der Oberfläche des *TetMesh* befindet (Wert = -1). Außerdem wird beim Erstellen eines *TetFace* Objektes der initiale Umfang mit der Methode *calc_triangle_length* berechnet und in einer Gleitkommavariable gespeichert.

Ein *TetNode* hat eine Position, welche durch ein *glm::vec3* gespeichert wird. Ebenso wird zu jedem Node die dazugehöre Texturkoordinate, Normale und Beschleunigung gespeichert. Abbildung 9 zeigt ein vereinfachtes UML-Diagramm der in diesem Kapitel aufgeführten Datenkstruktur.

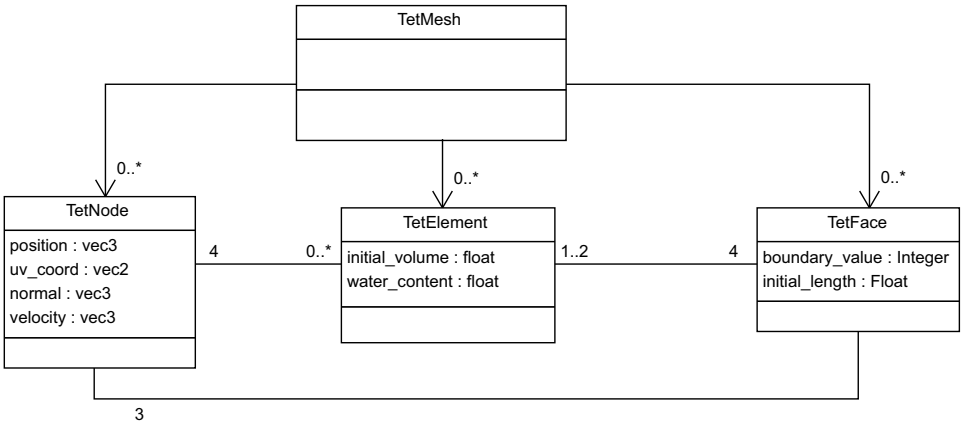


Abbildung 9: UML Diagramm der in dieser Arbeit verwendeten Datenstruktur

5.3 Tetgen

In der Computergrafik übliche Modelle von Objekten bestehen meistens aus Oberflächenmeshes. Diese können sich beispielsweise aus vielen, verbundenen Drei- oder Viereckflächen zusammensetzten. Allerdings arbei-

tet der in dieser Arbeit entwickelte Algorithmus auf einem Volumenmesh, um genau zu sein, einem Tetraedernetz. Da solche Modelle allerdings schwer zu finden sind, wurde die Bibliothek Tetgen verwendet. Diese integriert ein Delaunay basiertes Programm zum Erzeugen von Tetraedergittern aus einem beliebigen dreidimensionalen Polyederkomplex (auch PLC). Dazu werden der Bowyer-Watson [Bow81] und der Flip-Algorithmus [ES92] verwendet. Einen Einblick über die Basis dieser Algorithmen bietet Unterkapitel 3.3. Im Folgenden werden die für meinen Algorithmus relevanten Dateiformate und Funktionen erläutert.

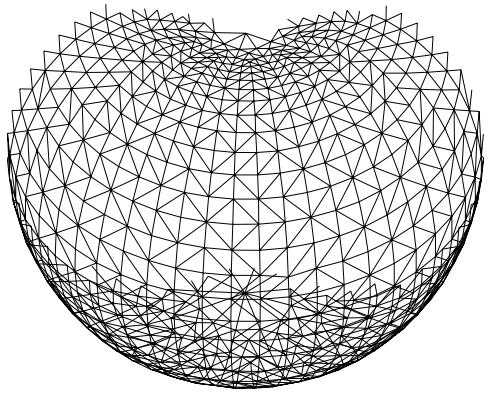


Abbildung 10: Beispielabbildung einer .poly Datei eines Apfels (Ansicht wurde in der Mitte durchgeschnitten). Die Darstellung erfolgt durch Tetview.

Dateiformate

Tetgen arbeitet mit verschiedenen Dateiformaten, denn ein Tetraedernetz wird durch eine Zusammensetzung verschiedener Komponenten repräsentiert. Die für diese Arbeit relevanten Dateiformate werden in der folgenden Tabelle vorgestellt und anschließend erläutert.

.node	eine Liste mit Vertices
.poly	stückweise lineares Komplex
.ele	eine Liste mit Tetraedern
.face	eine Liste mit Dreieckflächen
.edge	eine Liste mit Kanten
.neigh	eine Liste mit Nachbarn

Tabelle 2: Auflistung der wichtigsten Tetgen Dateiformate aus [Si06]

Eine von Tetgen lesbare bzw. erzeugte Nodefile setzt sich aus dem Filekopf und dem Filekörper zusammen. Der Fileheader besteht aus folgender Zeile:

$$< \#ofpoints >< dimension(3) >< \#ofattributes >< boundarymarkers(0or1) > .$$

Abbildung 11: Nodefileheader aus [Si06]

Der Header besteht aus der Anzahl der folgenden Nodes, der Koordinatenanzahl pro Node (für gewöhnlich drei), der Anzahl der Nodeattribute und dem sogenannten Boundary Marker, welcher den Wert 0 oder 1 annehmen kann. Im Header ist der Boundary Marker gesetzt, wenn in der weiteren Datei die optionalen Boundary Marker angegeben werden. Der Rest der Nodefile besteht aus wie folgt aufgebauten Nodes:

$$< point\# >< x >< y >< z > [attributes][boundarymarker].$$

Abbildung 12: Aufbau eines Nodes aus [Si06]

Ein Node besteht folglich aus seiner Nummer (auch Id), den drei Koordinaten gefolgt von den beliebig vielen Attributen und dem Boundary Marker. Dabei muss ein Boundary Marker gegeben sein, wenn im Header auch die Spezifikation (Marker auf 1 gesetzt) gefordert wird. In einer Nodezeile gibt dieser Wert an, ob sich ein Node am Rand des Tetraedergitters befindet. Die übrigen Dateiformate aus 2 sind ähnlich aufgebaut und können in [Si06] eingesehen werden.

Tetgen unterstützt neben den eigenen Dateiformaten noch andere Dateiformate als Eingabe, wie das Geomvie's Polyhedral Datenformat[7] (.off), das Polyhedral Datenformat (.ply) und das Medit's Mesh Datenformat[8] (.mesh).

Tetgen verfügt über unterschiedliche Funktionen und Einstellungen, die die Eigenschaften der erstellten Dateien sowie das davon repräsentierte Tetraedernetz beeinflussen. Im Allgemeinen ist dies durch Kommandozeilen Attribute möglich.

Im Folgenden sind die wichtigsten Kommandozeilen Attribute aufgelistet.

- -p die Eingabe PLC wird in Tetraeder geteilt
- -q verfeinert das Mesh (verbessert deren Qualität)
- -f alle Faces werden in eine .face Datei geschrieben

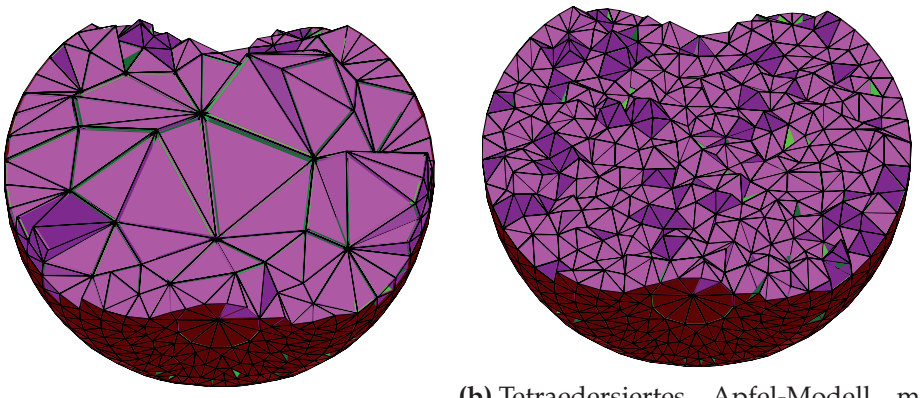
[7] `http://www.geomview.org/`
[8] `http://www-rocq1.inria.fr/gamma/medit/medit.html`

- -nn zu jedem Face in der .face Datei werden die dazugehörigen Tetraeder zugeordnet
- -Y verhindert die Veränderung der Eingabehülle
- -a beschränkt die Volumina der entstehenden Tetraeder auf die dazu angegebene Größe

5.4 Modelle und Vorverarbeitung

Um die Früchte zu modellieren benötigt mein Programm eine OBJ-Datei in der die Vertices, die Texturkoordinaten und die Normalen gegeben sind. Als Vorverarbeitungsschritt liest das Programm die .obj Datei ein und wandelt es in ein für Tetgen lesbares Format um. Es wird eine .node und eine .poly File erzeugt. Neben der Koordinaten eines Vertex werden in der Nodefile noch fünf weitere Attribute gespeichert. Dazu gehören die UV-Texturkoordinaten und die Normale des Vertex. In der Polyfile werden die Faces gespeichert, aus denen sich das Eingabenetz zusammensetzt. Dabei setzt sich ein Face aus den Nummern der drei Nodes zusammen, die es Aufspannen.

Vorverarbeitung



(a) Tetraedersiertes Apfel-Modell

(b) Tetraedersiertes Apfel-Modell mit Volumeneinschränkung

Abbildung 13: Vergleich von zwei tetraedersierten Äpfel Modellen ohne (a) und mit (b) Verwendung des Volumen einschränkenden Kommandozeilen Attributes $-a$. Ansicht wurde in der Mitte durchgeschnitten. Die Darstellung erfolgt durch Tetview.

Die erzeugte .poly File wird nun als Eingabe für Tetgen verwendet. Dabei zu beachten ist, dass Tetgen unbedingt mit den Kommandozeilen Attributen $-pqnnfY$ siehe Unterkapitel 5.3 aufgerufen werden sollte. Außerdem ist zu empfehlen $-a$ verwenden, damit das Tetraedernetz aus relativ gleichgroßen Tetraedern besteht. Dies verbessert die Qualität des hier vorgestellten Algorithmus. Ein Beispiel dazu zeigt Abbildung 13. Durch das

Ausführen von Tetgen mit den angegebenen Attributen werden nun weitere Dateien, in denen das erstellte Tetraedergitter gespeichert wird, erstellt. Unter anderem eine weitere .node-Datei (beinhaltet alle Nodes des Tetraedernetzes), eine .face-Datei (listet alle Faces des Tetraedernetzes) und eine .ele-Datei (beinhaltet alle Elemente oder auch Tetraeder des Tetraedernetzes).

5.5 Programmablauf

Initialisierung

Beim Start des in dieser Arbeit entwickelten Algorithmus wird zunächst das GLFW Fenster initialisiert und alle dazugehörigen Komponenten gesetzt. Danach wird die Methode *prepare_scene<scene#>* aufgerufen. Die Variable Scene# kann mit Zahlen belegt werden und entscheidet welche der verschiedenen Szenen vorbereitet werden sollen. In der zuletzt genannten Methode werden die Dateien aus Unterkapitel 5.4 von der Methode *load_mesh_from_files* eingelesen. Dabei werden die Dateien in die in Unterkapitel 5.2 vorgestellte Datenstruktur umgewandelt. Es wird ein *TetMesh-Objekt* erzeugt, welches die Geometrie der Modellfrucht enthält. Anschließend werden die Texturen für das Modell geladen und die Shader sowie die Buffer initialisiert. Außerdem werden die Texturkoordinaten in den Tangentenspace umgewandelt.

In der Render-Schleife führt der Algorithmus für jeden Zeitschritt folgende Methoden auf dem *TetMesh* aus:

- *update_water_content* aktualisiert den Wassergehalt der Tetraeder-Elemente

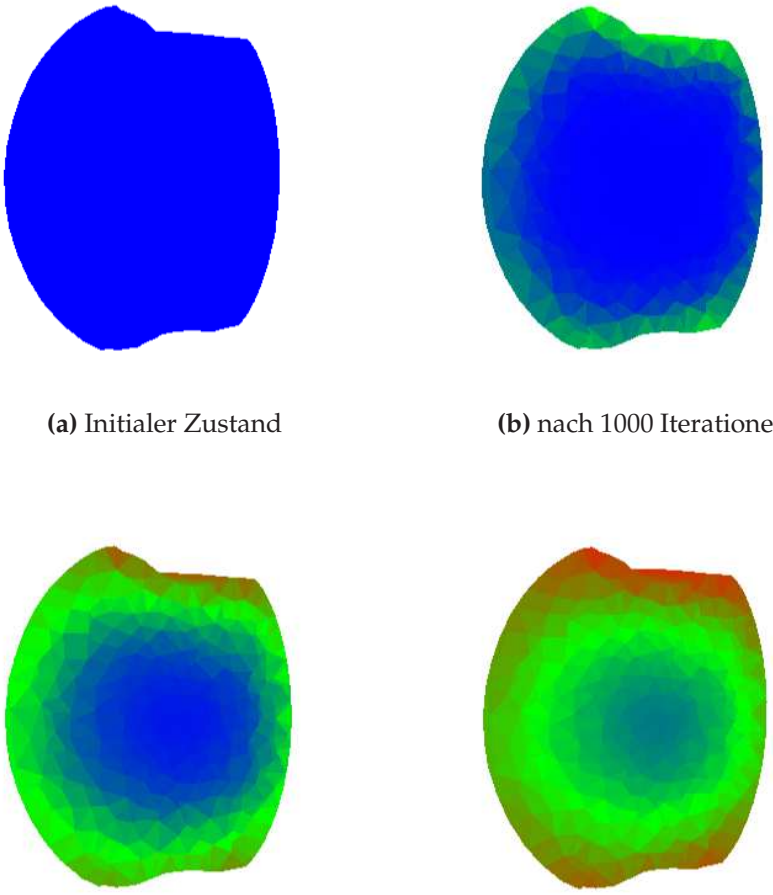
  Die Methode ist so eingebaut, dass nur Tetraeder am Rand des Tetraedernetzes Wasser verlieren und der Feuchtigkeitsverlust anschließend durch die Methode *dispence_water* in das Fruchtinnere verteilt wird. Abbildung 14 veranschaulicht die Funktionsweise dieser Methode am Beispiel eines Apfel Modells.

Quellcode 1: Methode, die alle Randflächen eine bestimmte Menge an Wasser verlieren lässt.

```
// Setzt den Wassergehalt von allen Randpunkten runter
void TetMesh::update_water_content(float delta_time, int water_behave) {
        //Iteriere über jedes Element
for (TetElement* current_element : element_list) {
 //Für jedes aus 4 Flächen des Tetraeders
 for (TetFace* current_face : current_element->faces) {
  float water_loss = 0;
  //MODUS 1
   if ( water_behave == 0 ){
   // Verliere eine konstante Menge Wasser an jedem Dreieck
   water_loss = 0.01;
   }
  //MODUS 2
  if ( water_behave == 1 ){
        // Verliere eine zufällige Menge Wasser an jedem Dreieck
        int water_rand = rand() % 100 + 1;
        water_loss = 1 / (water_rand );
  }

  // Nur Flächen am Rand des Meshes verlieren Wasser
  if (current_face->boundary == -1 ) {
    //Die Frucht verliert an Stellen mit <10% Wassergehalt kein Wasser
         mehr
        if (current_element->water_content > 0.10) {
        //Verliere die durch water_loss bestimmte Prozentzahl an Wasser
        current_element->new_water = current_element->water_content
        - water_loss * delta_time * current_element->water_content;

        current_element->update_to_new_water();
        }
  }
 }
}
//Lässt den Wasserverlust ins Innere des Volumens diffundieren
dispence_water();
}
```



(a) Initialer Zustand

(b) nach 1000 Iterationen

(c) Nach 5000 Iterationen

(d) Nach 10000 Iterationen

Abbildung 14: Veranschaulichung der Wassergehaltsverteilung in einem Apfel nach einer gewissen Anzahl von Zeitschritten. Farbskala von 100% (blau) des Anfangsgehaltes über 50% (grün) nach 0% (rot)

- *calc_forces* berechnet die wirkenden Kräfte pro Node und speichert diese in einer Liste

- *calc_setpoints* berechnet für jeden Tetraeder die neuen Nodekoordinaten nach dem in 4.1 vorgestellten Verfahren

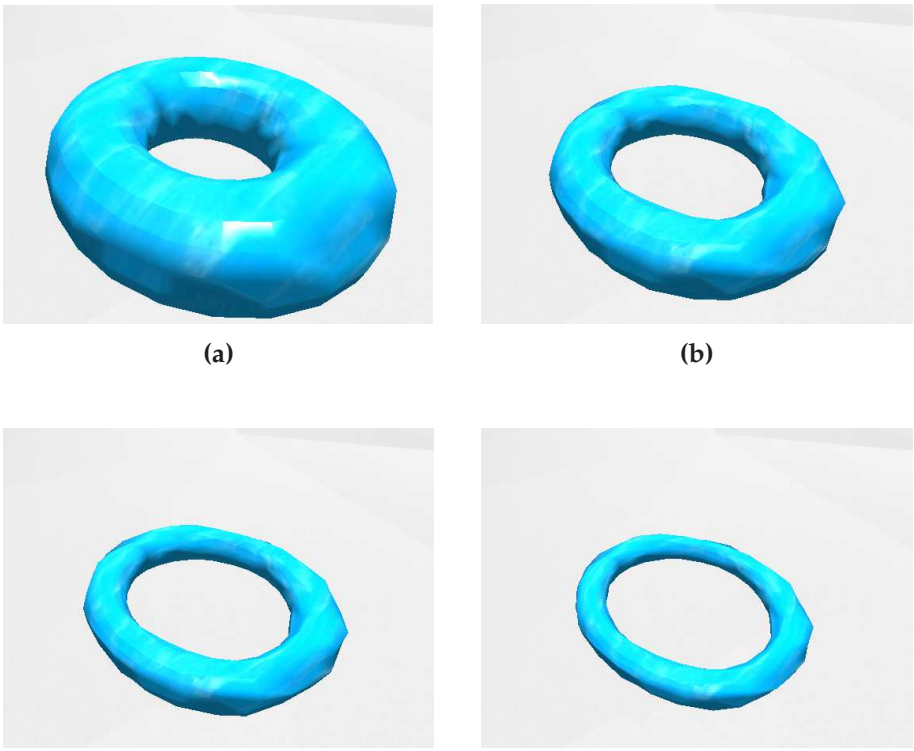
  Die Tretaderecken werden angepasst damit das neue Volumen, welches sich aus dem neuen Wassergehalt ergibt, ungefähr erreicht wird. Ein Node kann Teil mehrere Tetraeder sein und damit nicht auf veränderten Nodes weitergerechnet wird, werden die neuen Koordinaten zunächst in einer Liste pro Node gespeichert.

  Quellcode 2: Methode, welche die Position aller Tetraederpunkte berechnet, die nach einer Verschiebung zur Volumenanpassung, erreicht werden.

```
void TetMesh::calc_setpoints() {
        //Iteriere über jedes Tetraeder-Element
 for (TetElement* current_element : element_list) {
        //Aktuelles Tetraedervolumen
        float old_volume = current_element->get_volume();
        //Tetraedervolumen welches sich aus neuem Wassergehalt ergibt
        float new_volume = current_element->water_content *
            current_element->initial_volume;
        //Stelle sicher, dass sich das Volumen verändert
        if ((old_volume - new_volume) >= new_volume * 0.000001) {
        //Berechne den Interpolationswert t
        float t = pow(new_volume / old_volume, 1 / 3.f);
                //t muss zwichen 1 und 0 liegen
                if (t >= 0 || t <= 1) {
                  vec3 center = current_element->get_center();
                  //Berechne  die Verschiebung alle Punkte des Tetraeders
                       richtung Tetraedermitte
                  for (TetNode* current_node : current_element->nodes) {
                        //Speichert die neue Position für einen Node
                    current_node->setpoints.push_back(
                                (1 - t) * center + t * current_node->
                                    position);
                  }
                }
        }
 }
}
```

- *update_position* berechnet für jeden Node die neue Position

  Diese ergibt sich aus dem Mittelwert, der in Quellcode 2 erstellten Liste. Abbildung 15 zeigt die Wirkung der zuletzt genannten Funktionen am Beispiel eines Torus. Diese Abbildung verdeutlicht insbesondere, dass es sich bei der von mir entwickelten Schrumpfung, nicht um eine Skalierung handelt.



(a) **(b)**

(c) **(d)**

Abbildung 15: Darstellung eines schrumpfenden Torus.

Quellcode 3: Methode, die den akkumulierten Wert der neuen Positionen auf einen Punkt anwendet.

```
//Berechnet die akkumulierte neue Position
void TetMesh::update_position(float delta_time) {
 //Iteriere über alle Nodes
 for (TetNode* current_node : node_list) {
        //Initialiere Werte
        vec3 new_position = vec3(0);
        float setpoint_number = 0;
        //Addiere alle Verschiebungen und merke dir die Anzahl
        for (vec3 setpoint : current_node->setpoints) {
                setpoint_number++;
                new_position += setpoint;
        }
        current_node->setpoints.clear();
        //Falls die Anzahl der Verschiebungen nicht 0 ist, berechne den
            Mittelwert
        //Der Mittelwert ergibt die neue Position
        if (setpoint_number != 0) {
                new_position /= setpoint_number;
                current_node->position = new_position;
        }
 }
}
```

- *update_forces* wendet die Akkumulation der wirkenden Kräfte auf die Nodes an.

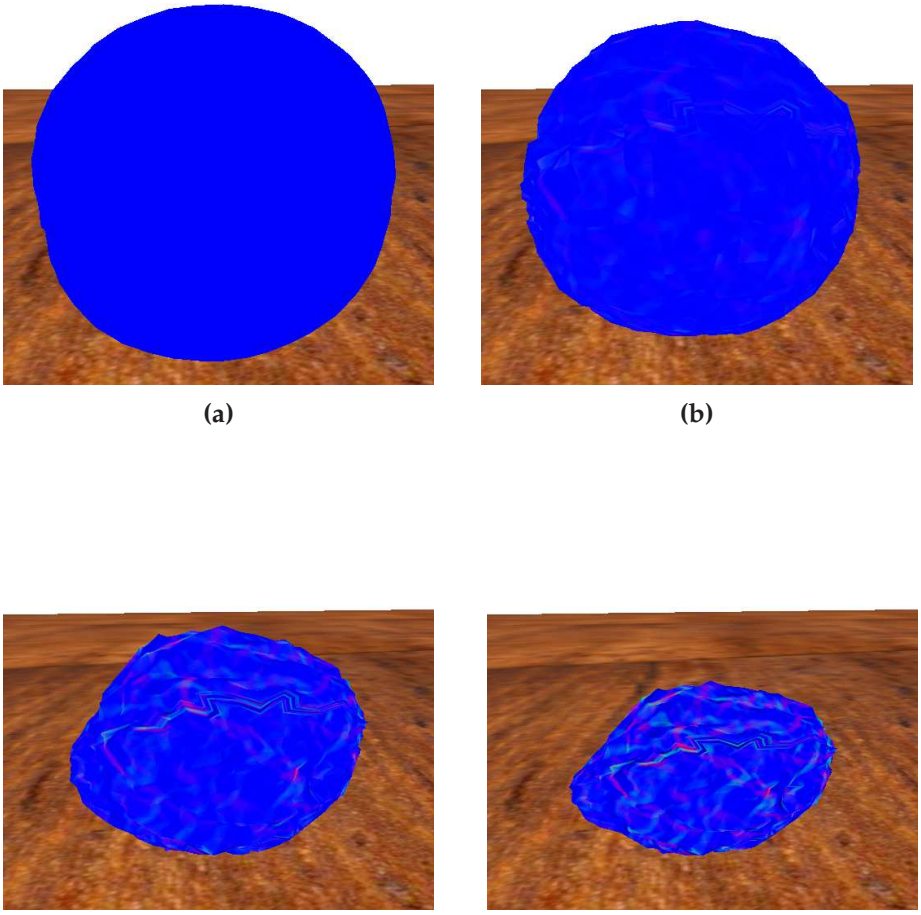
- *create_surface_mesh*: lädt die Daten des zu rendernden Meshes, sodass nur Flächen, die sich am Rand des Tetraedermeshes befinden gerendert werden.

5.6 Textur und Beleuchtung

Beim Verwittern von Früchten verändert sich nicht nur die Geometrie des Objektes. Besonders auffällig sind ebenfalls die farblichen Veränderungen. Im Allgemeinen wird die Frucht dunkler und es entstehen Flecken. Hinzu kommen die Veränderungen der Oberflächenstruktur. Die Frucht wird rauer und bekommt Falten. Diese Eigenschaften wurden in dieser Arbeit durch das Anwenden von Farb- und Höhentexturen im Fragment-Shader gelöst.

Zu jedem Fruchtmodell werden zwei Farbtexturen hinzugefügt. Die erste Textur entspricht der einer frischen und reifen Frucht. Die zweite Textur hingegen stellt die Frucht im verwitterten Zustand dar. Im Fragment-Shader wird für die Fragment-Farbe zwischen den beiden Texturen interpoliert. Als Interpolationsmaß wird der gemittelte Wassergehalt der äußersten Schicht verwendet. Dieser wird als Uniformvariable an den Shader gegeben.

Die Falten werden durch die Verwendung von Bump-Mapping erzeugt, denn zu jedem Fruchtmodell wird eine Höhentextur übergeben. Aus dieser wird mit der in Kapitel 4.2 beschriebenen Methode für jedes Fragment eine Normale berechnet. Die Ausprägung der Normale wird durch den Faktor *crump_value* beeinflusst. Sein Wert wird für jedes Face berechnet und an den Shader weitergegeben. Der Faktor selbst gibt an wie stark sich die Länge eines Faces verändert hat, also wie stark ein Face geschrumpft ist. Somit entstehen dynamisch generierte Normalen, die je nach Grad der Schrumpfung intensiviert werden.



(a) **(b)**

(c) **(d)**

Abbildung 16: Darstellung der berechneten Oberflächennormalen verschiedenen Phasen der Verwitterung.

6 Ergebnis

Die in dieser Arbeit entwickelte Anwendung ermöglicht es aus einem Modell gegebenen Wavefront-Objekt und den dazugehörigen Texturen ein beliebiges Objekt verwittern zu lassen. Der hier entwickelte Verwitterungsprozess lässt das Objekt in sich zusammenschrumpfen und verändert sowohl die Farbe als auch die Oberfläche, je nach Schrumpfungsgrad. Dabei werden allerdings äußere Einflüsse nicht beachtet. Neben dem Algorithmus wurden einige Fruchtmodelle zur Veranschaulichung erstellt. Abbildung 17 zeigt die von mir erstellten Modelle vor und nach der Verwitterung durch meinen Algorithmus. Bereits an diesen Modellen ist zu erkennen, dass die Form welche sich nach der Verwitterung ergibt, stark von dem eingegebenen Modell abhängt. Allgemein ist festzustellen, dass runde Objekte bessere Ergebnisse liefern, als Modelle mit starken Kanten. Durch die vom Modell festgelegte Tetraederstruktur werden Kanten beim Schrumpfen eher verstärkt.

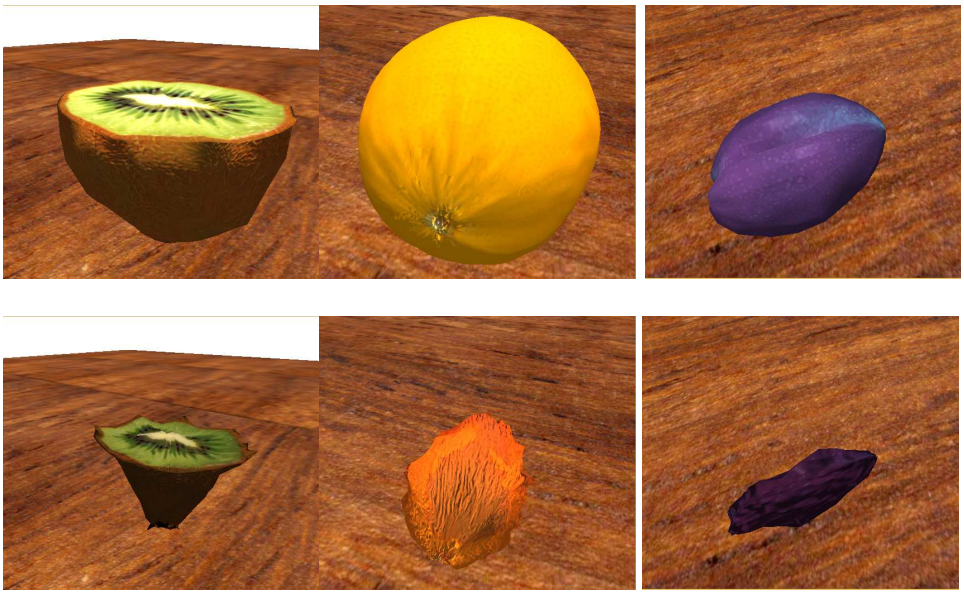


Abbildung 17: Beispiele der Verwitterungssimulation an verschiedenen selbst erstellten Modellen.

6.1 Modelle

Die Ergebnisse des in dieser Arbeit entwickelten Algorithmus ist stark von der Zusammensetzung des eingegebenen Tetraedernetzes abhängig. Dabei bestimmt die Anzahl der Tetraeder die Laufzeit und deren Feinheit beeinflusst die Struktur des Volumens und ebenfalls das Verhalten während

des Schrumpfungsprozesses. Je feiner das eingegebene Tetraedergitter ist, desto weniger Kanten bilden sich nach der Ausführung des Algorithmus. Dies ist auf den prozentualen Wasserverlust zurückzuführen. Besitzt das Eingabemesh sich von der Größe stark unterscheidende Tetraeder, können sich Kanten an den gröberen Tetraedern bilden. Außerdem bestimmen die Tetraeder auch die Formveränderung des Objektes. Ist ein rundes Objekt beispielsweise aus länglichen Tetraedern aufgebaut, wird sich durch das prozentuale Zusammenziehen, das runde Objekt in ein kleineres länglicheres transformieren. Diesen Effekt kann man allerdings ausnutzen, um eine Deformation zu verdeutlichen. Besteht das Eingabeobjekt aus etwa gleich großen Tetraedern, sind die Übergänge bei der Schrumpfung deutlich weicher und es bilden sich weniger Kanten. Der Nachteil bei einem derartig aufgebauten Modell besteht darin, dass sich die allgemeine Form der Frucht wenig ändert und die Schrumpfung einer Skalierung ähnlich sieht. Um dieses Verhalten genauer zu verdeutlichen wurde ein Apfel in hoch aufgelöster Modellierung mit ungefähr gleich großen Tetraedern schrumpfen gelassen (siehe Abbildung 18). Im Vergleich dazu wurde dasselbe Modell in geringerer Auflösung und mit unterschiedlich großen Tetraedern in die Simulation gegeben (siehe Abbildung 19).

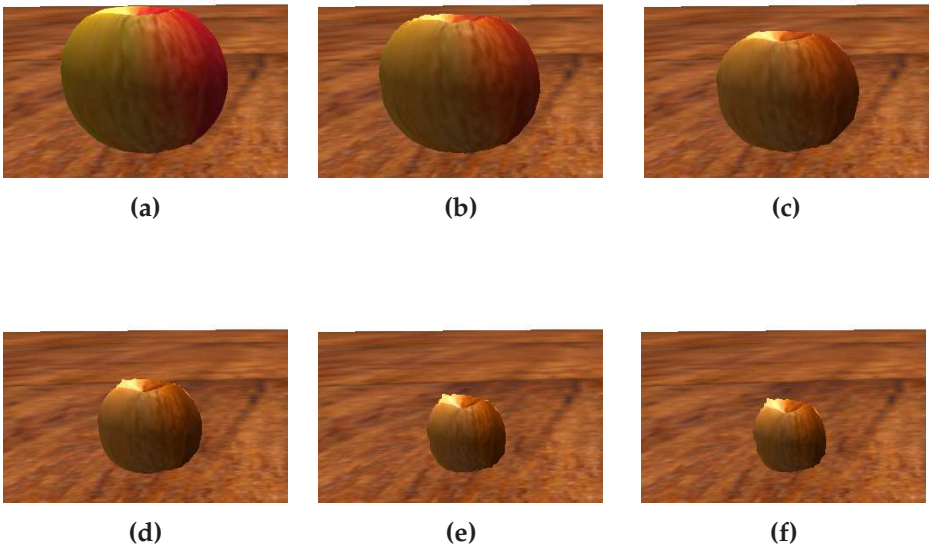
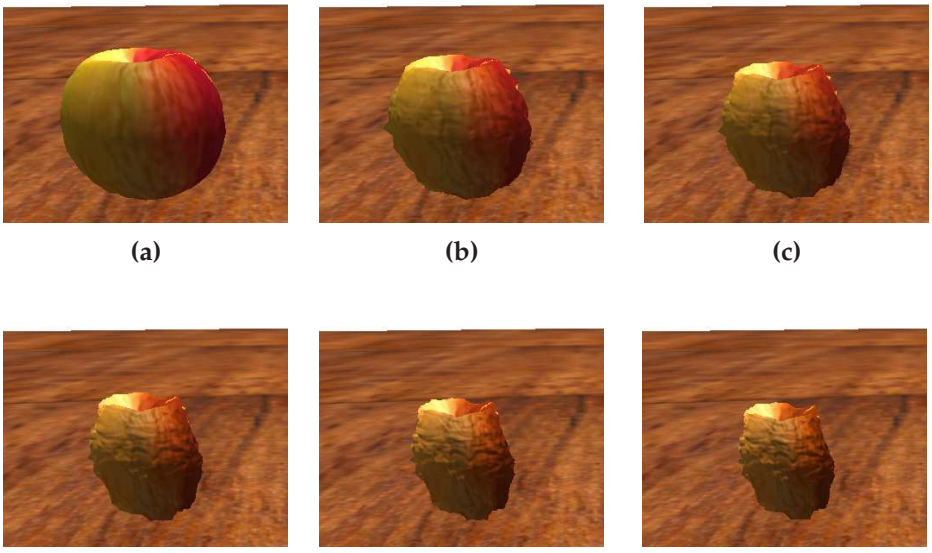


Abbildung 18: Hoch aufgelöster Apfel in verschiedenen Stadien der Verwitterung.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Abbildung 19: Niedrig aufgelöster Apfel in verschiedenen Stadien der Verwitterung.

6.2 Laufzeit

Der von mir entwickelte Algorithmus fokussiert sich auf das Deformieren eines Tetraedervolumen. Dazu wird bei den Rechenvorgängen über alle Tetraeder-Elemente iteriert. Der Aufwand dabei beträgt $\mathcal{O}(n)$ wobei n der Anzahl von Tetraeder-Elementen in dem Tetraedergitter entspricht. Um die Laufzeit zu messen, wurden fünf unterschiedlich aufgelöste Modelle eines Objektes erstellt. Diese besitzen eine steigende Anzahl an Punkten $(400, 800, 1600, 3200, 6400)$ und somit auch an Tetraedern $(1141, 3748, 8869, 18902, 39205)$. Danach wurde gemessen, wie lange der gesamte Algorithmus im Durchschnitt für einen Iterationschritt mit dem gegebenen Modell braucht. Die Messungen (siehe Abbildung 20) haben ergeben, dass sich die benötigte Zeit proportional zu der Anzahl der eingegebenen Tetraeder-Elementen verhält. Durch die Messungen konnte somit die lineare Laufzeit bestätigt werden.

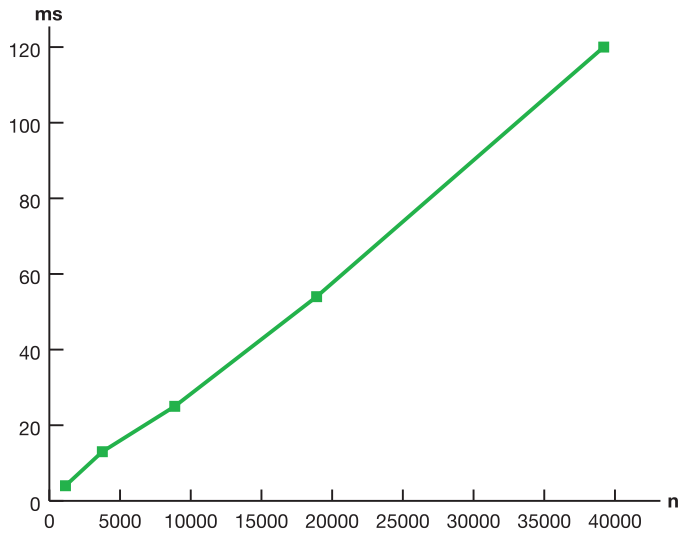


Abbildung 20: Laufzeitmessungen für einen Schleifendurchlauf in Millisekunden mit einem Objekt, welches aus n Tetraeder-Elementen aufgebaut ist.

7 Fazit und Ausblick

In dieser Arbeit wurde ein Verfahren entwickelt, welche die komplexen Vorgänge, die bei der Verwitterung von Früchten beobachtet werden können, auf zwei grundlegende Prozesse und deren grafische Simulation reduziert. Zunächst wird die Schrumpfung des Fruchtfleisches durch Wasserverlust mit Hilfe von Volumenanpassung einer Tetraederstruktur simuliert. Anschließend wird die Veränderung der Haut und deren Oberflächenstruktur anhand von Texturinterpolation und in der Intensität skalierten Normalen dargestellt. Während die Deformation des Fruchtfleisches auf CPU-Ebene stattfindet, werden Berechnungen der Hauteigenschaften hoch parallel auf der GPU ausgeführt. Damit erreiche ich durch das Zusammenspiel von CPU und GPU ein effizientes Verfahren, welches ein beliebiges Modell (gegeben als .obj-Datei) in sich zusammenschrumpfen lässt. Sowohl die Laufzeit als auch die nach der Deformation einstehende Struk-

(a) Echter Apfel **(b)** Simulierter Apfel

Abbildung 21: Vergleich von einem echten und von mir simulierten Apfel.

tur des Objektes hängt dabei von der erstellten Tetraederaufteilung ab. Es ergibt sich eine Laufzeit von $\mathcal{O}(n)$, wobei n der Anzahl von Tetraeder-Elementen aus der Tetraederstruktur entspricht. Die Laufzeit könnte jedoch noch optimiert werden. Beispielsweise könnten einige der in Kapitel 5 vorgestellten Funktionen mit Hilfe von Multi-Threading parallelisiert werden. Eine weitere Verbesserungsmöglichkeit liegt darin, einige Funktionen, wie den Wasserverlust dynamisch an die Frucht anzupassen. In Kapitel 3 wurden die unterschiedlichen Transpirationskoeffizienten von Früchten vorgestellt. Diese könnte man in die Berechnung des Wasserverlustes einbeziehen, um somit für die verschiedenen Früchte eine naturgetreuere Simulation zu erzeugen.

Abschließend ist zu sagen, dass die in dieser Arbeit entwickelte Simulation das Problem der Verwitterung auf das Grundlegende reduziert und dennoch, insbesondere mit realistischen Texturen, gute Ergebnisse (siehe Abbildung 21) mit einer linearen Laufzeit erzielen kann.

Abbildungsverzeichnis

Quellcodeverzeichnis

Literatur

[BF96] BECKER, Bryan R. ; FRICKE, Brian A.: Transpiration and respiration of fruits and vegetables. In: *Science et Technique du Froid (France)* (1996)

[BFA02] BRIDSON, Robert ; FEDKIW, Ronald ; ANDERSON, John: Robust treatment of collisions, contact and friction for cloth animation. In: *ACM Transactions on Graphics (ToG)* Bd. 21 ACM, 2002, S. 594–603

[BH97] BASTIN, Sandra ; HENKEN, K: Water content of fruits and vegetables. In: *Retrieved* 11 (1997), Nr. 20, S. 2010

[Bow81] BOWYER, A.: Computing Dirichlet tessellations. In: *The Computer Journal* 24 (1981)

[ES92] EDELSBRUNNER, H. ; SHAH, N. R.: Incremental Topological Flipping Works for Regular Triangulations. In: *Proceedings of the Eighth Annual Symposium on Computational Geometry*, 1992

[JHW09] JIAO, Shaohui ; HENG, PhengAnn ; WU, Enhua: Realistic Grass Withering Simulation Using Time-varying Texels. In: *ACM SIGGRAPH ASIA 2009 Sketches*, 2009

[JRB11] JR., Joseph T. K. ; RAJA, Samantha ; BADLER, Norman I.: Fruit Senescence and Decay Simulation. In: *Computer Graphics Forum* (2011)

[LCW+12] LIU, Youquan ; CHEN, Yanyun ; WU, Wen ; MAX, Nelson ; WU, Enhua: Physically based object withering simulation. In: *Computer Animation and Virtual Worlds* (2012)

[LYC+15] LIU, Yin ; YANG, Xiaosong ; CAO, Yang ; WANG, Zhao ; CHEN, Biaosong ; ZHANG, Jian-Jun ; ZHANG, Hongwu: Dehydration of core/shell fruits. In: *Computers & Graphics* (2015)

[MG08] MÉRILLOU, Stéphane ; GHAZANFARPOUR, Djamchid: A survey of aging and weathering phenomena in computer graphics. In: *Computers & Graphics* (2008)

[Si06] SI, Hang: TetGen A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator Users's Manual. (2006)

[YFCT08] YIN, Xin ; FUJIMOTO, Tadahiro ; CHIBA, Norishige ; TANAKA, Hiromi T.: Modeling of Wood Aging Caused by Biological Deterioration. In: *JACIII* 12 (2008), Nr. 2, S. 125–131

[ZTZ05] ZHU, J ; TAYLOR, ZRL ; ZIENKIEWICZ, OC: *The finite element method: its basis and fundamentals*. 2005