HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

Entwicklung eines Programms SubVis zur Visualisierung von Unterteilungsalgorithmen

Tobias Keh, Simon Kessler, Felix Born

Konstanz, 31.08.2015

PROJEKTARBEIT

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CGAL Computational Geometry Algorithms Library

CUDA Compute Unified Device Architecture

Far Feature Adaptive Representation

GPL GNU General Public License

LGPL GNU Lesser General Public License

moc Meta-Object Compiler

MVC Model View Controller

Osd OpenSubdiv cross platform

RWTH Rheinisch-Westfälische Technische Hochschule

Sdc Subdivision Core

Vtr Vectorized Topological Representation

1 Einführung

Mittels eines Polygonnetzes lassen sich Flächen verschiedener Formen beschreiben. Meist wird ein sog. Kontrollnetz verwendet, welches von einem Modellierer erstellt wird. Dieses wird dann durch computergestützte Verarbeitung verfeinert, um eine glatte Fläche zu erzeugen. Hierbei stellt sich die Herausforderung, mit möglichst wenigen Polygonen im Kontrollnetz eine glatte Oberfläche erzeugen zu können. Ein kleines Kontrollnetz spart Speicherplatz und senkt die Komplexität bei Änderungen im Netz. Ein in der Computergrafik häufig verwendeter Ansatz zur Berechnung von glatten Oberflächen aus einem Kontrollnetz, ist die Anwendung von Unterteilungsalgorithmen. Hier soll insbesondere der Algorithmus von Catmull-Clark erwähnt werden. Pixar verwendet diesen für die Entwicklung von animierten Filmen (siehe Abb. 1.1).

Das Projekt *SubVis* setzt sich zum Ziel ein Programm zu entwickeln, das einige Unterteilungsalgorithmen implementiert und visualisiert. Dabei soll einerseits das Kontrollnetz (Polygonnetz) als auch die durch Anwendung von Unterteilungsalgorithmen entstehende Limesfläche dargestellt werden. Dies dient dazu, die Vielzahl an Algorithmen an einem Ort zu bündeln und über eine schlanke, übersichtliche Anwendung zu Verfügung zu stellen. Hiervon sollen insbesondere Studenten und andere interessierte Personen profitieren, die sich mit der Thematik auseinandersetzen möchten. Der modulare Aufbau und eine gute Dokumentation sollen nachfolgenden Projekten (Abschlussarbeiten, Teamprojekte, etc.) die Möglichkeit geben, die Anwendung weiter zu entwickeln bzw. zu erweitern.

Es wird nachfolgend eine kurze Einführung in die Thematik gegeben, gefolgt

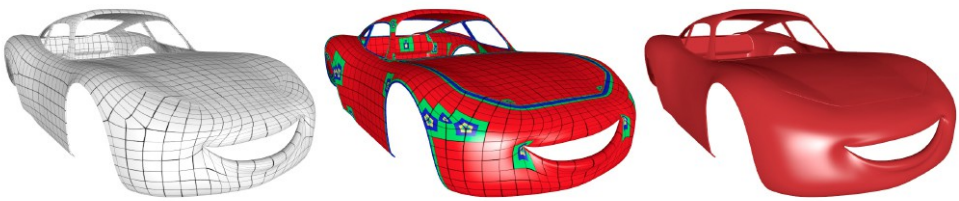

Abbildung 1.1: Anwendung von Catmull-Clark [8]

von der Evaluierung verschiedener Tools und Bibliotheken. Danach folgt eine Beschreibung des geplanten Programms *SubVis*. Abschließend wird der bisherige Projektverlauf und ein Ausblick für das kommende Semester dargelegt.

2 Grundlagen

In diesem Kapitel wird eine kurze Einführung in die Grundlagen von Unterteilungsalgorithmen gegeben.

2.1 Unterteilungsalgorithmen

Unterteilungsalgorithmen erzeugen aus einem Ausgangspolygonnetz eine glatte Fläche. Die glatte Zielfläche ist dabei der Grenzwert eines unendlichen, rekursiven Verfeinerungsschemas. Abb. 2.1 visualisiert die Anwendung eines Unterteilungsalgorithmus auf eine Kurve und auf eine Fläche. Nach mehrfacher Anwendung der Unterteilung konvergiert die Kurve oder Fläche gegen die glatte Zielkurve bzw. Zielfläche.

Unterteilungsalgorithmen kann man anhand ihrer Eigenschaften Kategorisieren. Ein Unterscheidungskriterium betrifft die Art und Weiße, wie unterteilt wird. Man unterscheidet dabei zwischen *Primal* und *Dual*.

Primal Bei dieser Strategie wird die Oberfläche unterteilt („face split“). Abb. 2.2 stellt diese Methode für ein Dreicksnetz und ein Viereckснetz dar.

Dual Auf der anderen Seite ist es möglich Eckpunkte in mehrere Eckpunkte aufzusplitten („vertex split“). Diese Methode ist in Abb. 2.3 abgebildet.

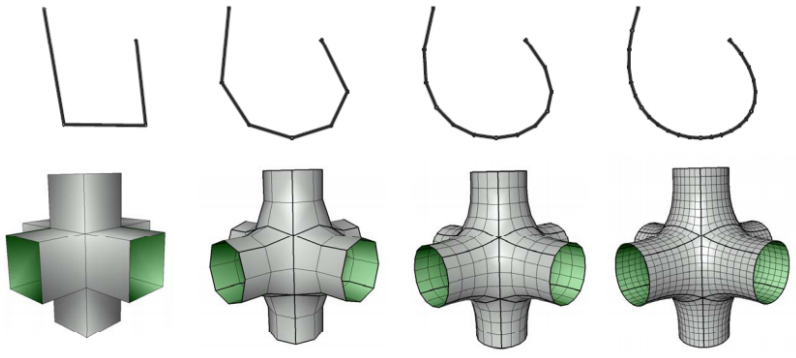

Abbildung 2.1: Unterteilungsalgorithmus - Kurve und Fläche [12]

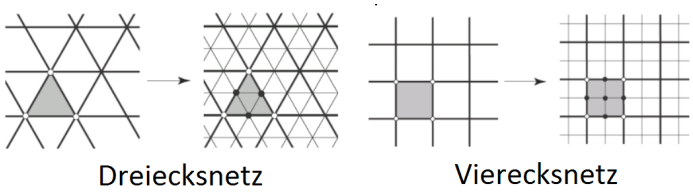

Abbildung 2.2: Primal (face split) [12]

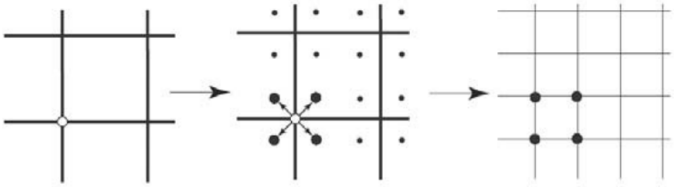

Abbildung 2.3: Dual (vertex split) [12]

Ein weiteres wesentliches Merkmal ist, ob Kontrollpunkte interpoliert werden oder nicht.

Approximation Kontrollpunkte werden nicht interpoliert.

Interpolation Kontrollpunkte werden interpoliert.

Tabelle 2.1 listet die bekanntesten Unterteilungsalgorithmen auf und ordnet diese den Kategorien zu. Zu jedem Algorithmus ist zusätzlich die „Glattheit“ der Oberfläche angegeben (C-Stetigkeit). Dies kann auch als Maß über die Qualität des Unterteilungsalgorithmus fungieren. Abb. 2.4 Vergleicht die vier unterschiedlichen Unterteilungsalgorithmen Catmull-Clark, Loop, Doo-Sabin und Butterfly. Man erkennt deutlich den interpolierenden Unterteilungsalgorithmus (Butterfly), da dieser durch die harten Interpolationsbedingungen im Vergleich zu den approximierenden Algorithmen viel „welliger“ ist.

2.2 Auswahl von Unterteilungsalgorithmen

Für das Projekt sollen folgende Algorithmen implementiert werden:

- Catmull-Clark

Tabelle 2.1: Unterteilungsalgorithmen Übersicht

	Primal		**Dual**
	Dreiecksnetz	**Viereckснetz**	
Approximation	Loop (C^2)	Catmull-Clark (C^2)	Doo-Sabin (C^1)
Interpolation	Butterfly (C^1)	Kobbelt (C^1)	Biquartic (C^2)

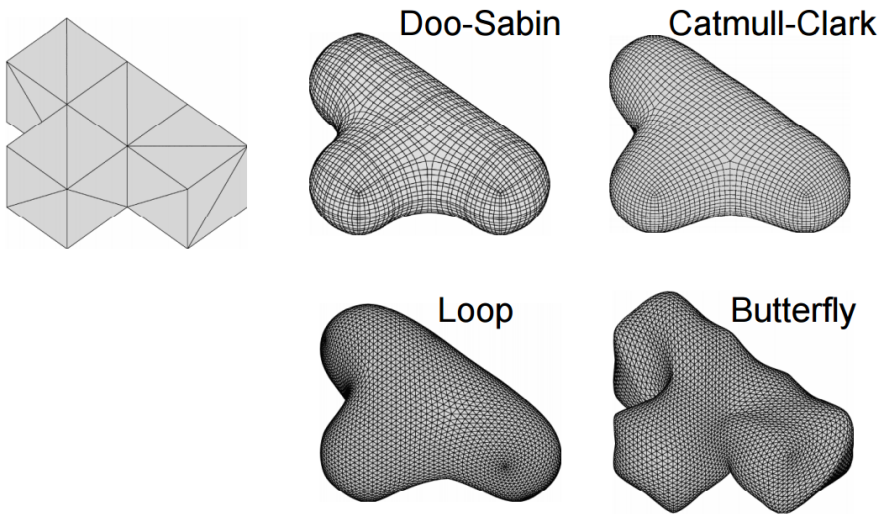

Abbildung 2.4: Vergleich der Unterteilungsalgorithmen [12]

- Loop
- Doo-Sabin
- Butterfly

Dies sind die wichtigsten Vertreter für Dreiecks- und Viereckснetze. Prinzipiell sind jedoch noch weitere Algorithmen denkbar.

3 Tools und Bibliotheken

In diesem Kapitel werden Bibliotheken und Programme untersucht, die für das Projekt verwendet werden können.

3.1 Unterteilungsalgorithmen und Datenstrukturen

Im Bereich Unterteilungsalgorithmen gibt es viele bereits implementierte Datenstrukturen und Algorithmen. Gesucht wird eine einfache Datenstruktur, um Polygonnetze verarbeiten zu können. Diese sollte so wenig Overhead wie möglich mitbringen. Im Allgemeinen besteht solch eine Datenstruktur aus Ecken (Vertices), Kanten (Edges) und Flächen (Faces). Zusätzlich muss noch die Beziehung zwischen den Objekten abgespeichert werden.

3.1.1 OpenMesh

OpenMesh wird von der Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen entwickelt und stellt eine mächtige Datenstruktur für Polygonnetze bereit. Es steht unter der GNU Lesser General Public License (LGPL) v3 Lizenz („with exception“) und kann somit problemlos verwendet werden.

OpenMesh implementiert eine Datenstruktur für Polygonnetze. Darüber hinaus sind bereits einige Unterteilungsalgorithmen implementiert, die auf der OpenMesh Datenstruktur arbeiten können. Zum Funktionsumfang gehören folgende Algorithmen:

1. Uniform subdivision
    - Loop
    - Sqrt3
    - Modified Butterfly
    - Interpolating Sqrt3
    - Composite
    - Catmull Clark
2. Adaptive subdivision

- Adaptive Composite

3. Simple subdivision
    - Longest Edge

OpenMesh implementiert eine *Halfedge* Datenstruktur. Diese *kantenbasierte* Datenstrukturen speichern die Information über die Verbindungen zwischen Eckpunkten in den Kanten, während *flächenbasierte* Datenstrukturen die Verbindungsinformation zwischen den Eckpunkten und Nachbarn in den Flächen speichern.

Jede Kante referenziert also folgende Objekte:

- zwei Eckpunkte
- eine Fläche
- die nächsten zwei Kanten der Fläche

Halfedge bedeutet nun, dass eine Kante in 2 Halbkanten (Halfedge) aufgeteilt wird. Jede Halbkante hat nur eine Richtung. Zwei Ecken A und B sind also über 2 Halbkanten (erste Halkante von A nach B und zweite Halbkante von B nach A) miteinander verbunden. Dies bringt den Vorteil, dass man über die Kanten einer Fläche sehr einfach iterieren kann. Man muss dazu lediglich den Halbkanten folgen.

OpenFlipper

Aufbauend auf OpenMesh wurde von der RWTH Aachen zusätzlich das flexible Plugin-basierte Framework OpenFlipper enwickelt. Damit können geometrische Objekte modelliert und verarbeitet werden. Intern wird auf die Datenstrukur OpenMesh zurückgegriffen. Für die grafische Oberfläche wird QT verwendet. Mit OpenFlipper kann man über die Oberfläche Netze erstellen und die in OpenMesh implementierten Subdivision Algorithmen anwenden. Abb. 3.1 zeigt die Benutzeroberfläche von OpenFlipper.

3.1.2 Surface_mesh

Surface_mesh [11] ist eine einfache und effizente Datenstruktur um Polygonnetze beschreiben zu können. Die Datenstruktur wurde als einfachere Alternative zu OpenMesh von der Bielefeld Graphics & Geometry Group entwickelt. Die Datenstruktur soll einfach zu benutzen sein und eine bessere Performance

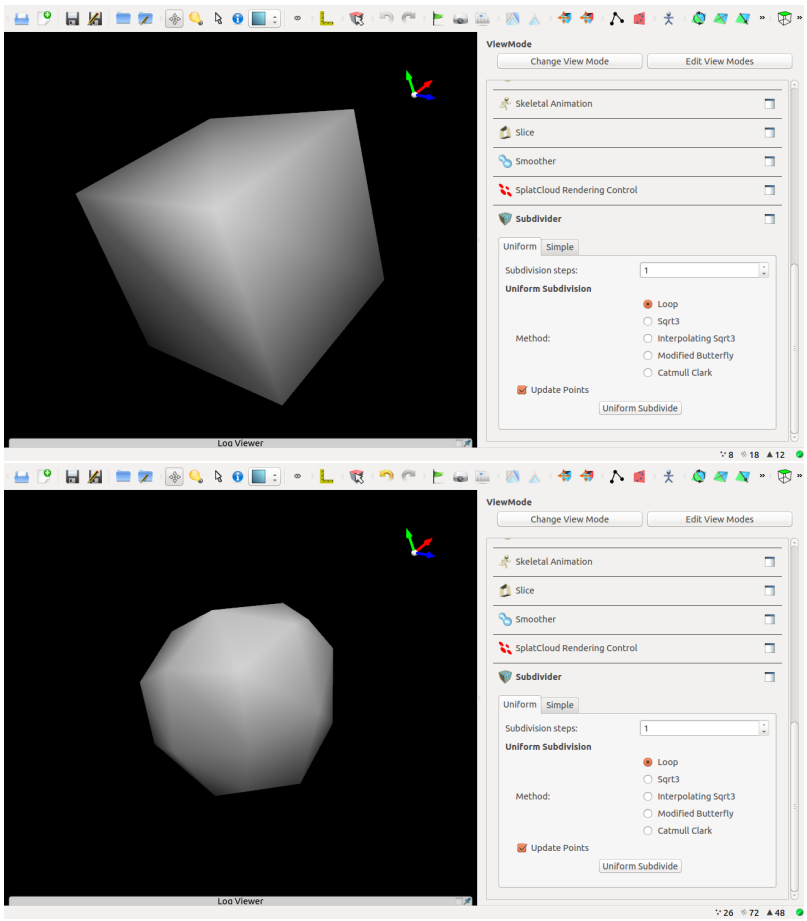

Abbildung 3.1: OpenFlipper

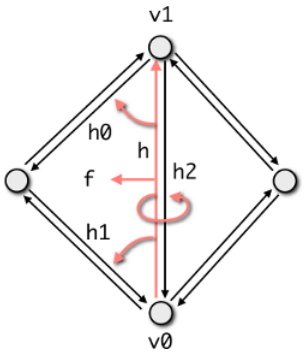

Abbildung 3.2: Surface_mesh - Halfedge Verbindungen [7]

und geringeren Speicherverbrauch mitbringen. Analog zu OpenMesh implementiert Surface_mesh eine Halfedge Datenstruktur. Die Verbindungsinformation der Kanten werden also in einem Paar aus zwei gerichteten Halbkanten gespeichert. Abb. 3.2 visualisiert den Zusammenhang der Halbkanten und Ecken.

Da Surface_mesh auch als Halfedge Datenstruktur implementiert ist, kann ähnlich effizient zu OpenMesh über die Kanten iteriert werden. Listing 3.1 zeigt einige Basisoperationen die möglich sind. Die Operationen aus Listing 3.1 sind zum besseren Verständnis in Abb. 3.2 gekennzeichnet.

Listing 3.1: Surface_mesh - Basisoperationen

```
1 Surface_mesh::Halfedge h;
2 Surface_mesh::Halfedge h0 = mesh.next_halfedge_handle(h);
3 Surface_mesh::Halfedge h1 = mesh.prev_halfedge_handle(h);
4 Surface_mesh::Halfedge h2 = mesh.opposite_halfedge_handle↩
    (h);
5 Surface_mesh::Face     f  = mesh.face_handle(h);
6 Surface_mesh::Vertex   v0 = mesh.from_vertex_handle(h);
7 Surface_mesh::Vertex   v1 = mesh.to_vertex_handle(h);
```

Um das Netz zu verändern oder zu editieren unterstützt die Datenstruktur high-level Operationen zum Verändern der Topologie. Mit *Edge Collapse*, *Edge Split* und *Edge Flip* kann das Netz geändert werden. Die Operationen sind in Abb. 3.3 dargestellt.

Die Bibliothekt steht unter der *GNU Library General Public License*, was eine problemlose Verwendung in diesem Projekt ermöglicht.

3.1.3 OpenSubdiv

OpenSubdiv wird von Pixar entwickelt und ist eine mächtige Bibliothek, die Unterteilungsalgorithmen und Datenstrukturen implementiert. Die Bibliothek

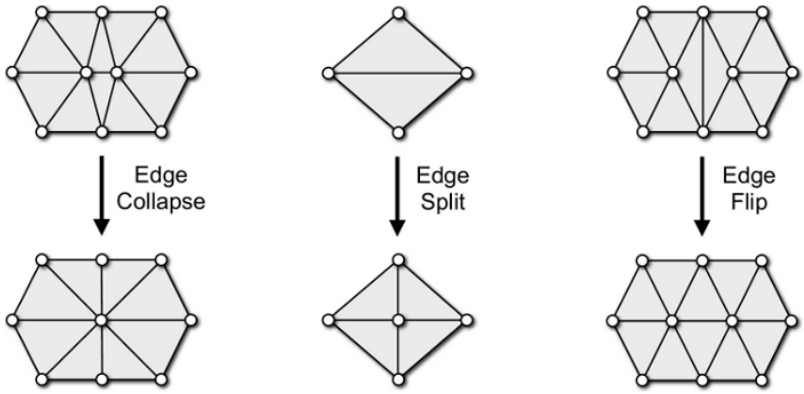

Abbildung 3.3: Surface_mesh - high-level Operationen zum Ändern der Topologie [7]

ist optimiert auf Performance und unterstützt paralleles Rechnen auf CPU und GPU. Primär wird die Bibliothek von Pixar zum erstellen von animierten Filmen verwendet. OpenSubdiv ist lizensiert unter der Apache License und darf somit frei für kommerzielle und nicht kommerzielle Projekt genutzt werden.

Abb. 3.4 zeigt den Aufbau der OpenSubdiv Bibliothek. Sie besteht insgesamt aus den 4 Schichten Subdivision Core (Sdc), Vectorized Topological Representation (Vtr), Feature Adaptive Representation (Far) und OpenSubdiv cross platform (Osd) [10].

Sdc ist die unterste Schicht in der Architektur und implementiert die Unterteilungsdetails. Dazu gehören Typen, Optionen und Eigenschaften für die konkreten Unterteilungsalgorithmen.

Vtr beinhaltet Klassen, die das Netz für effiziente Verfeinerung in einer Zwischenrepräsentation darstellen. Diese Schicht ist nur für den internen Gebrauch gedacht.

Far ist die zentrale Schnittstelle, um Polygonnetze mit Unterteilungsalgorithmen zu verarbeiten.

Osd beinhaltet geräteabhängigen Code, um Objekte aus der Schicht Far auch in unterschiedlichen Backends wie Compute Unified Device Architecture (CUDA) oder OpenCL ausführbar zu machen.

Von OpenSubdiv werden die Unterteilungsalgorithmen *Catmull-Clark*, *Loop* und *Bilinear* unterstützt [10].

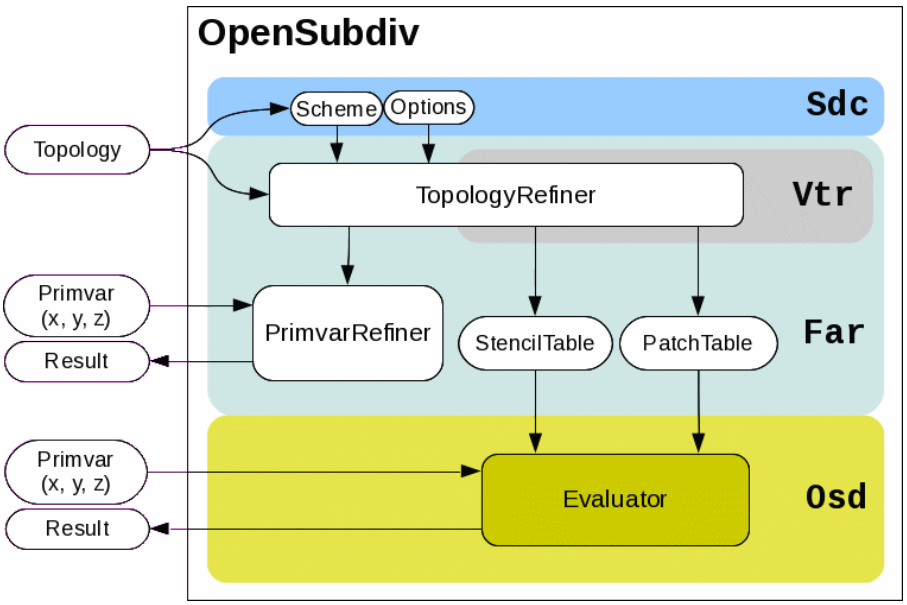

Abbildung 3.4: Pixar OpenSubdiv Architektur [10]

3.1.4 CGoGN

CGoGN ist eine Geometric Modeling C++ Bibliothek und implementiert eine Datenstruktur für n-dimensionale Netze als Combinatorial Maps. Diese Implementierung unterscheidet sich zu den Halfedge Datenstrukteren von OpenMesh und Surface_mesh deutlich. Diese sind zwar alle effizient, haben jedoch Probleme beim Umgang mit Objekten von unterschiedlichen Dimensionen. Für jeden Problemfall muss die spezielle Datenstruktur verwendet werden. All diese Strukturen lassen sich jedoch auf Combinatorial Maps zurückführen. Diesen allgemeineren Ansatz geht CGoGN. CGoGN implementiert bereits den Unterteilungsalgorithmus Catmull-Clark [2].

3.1.5 CGAL

Computational Geometry Algorithms Library (CGAL) ist ein mächtiges Softwareprojekt mit einer Vielzahl an Datenstrukturen und Algorithmen. Neben Unterteilungsalgorithmen werden auch eine Reihe anderer Themengebiete abgedeckt (Voronoi Diagramme, Convex Hull Algorithms, Spatial Searching ...). Für die Repräsentation von Netzen gibt es bei CGAL mehrere Möglichkeiten.

Surface_mesh Zum einen implementiert CGAL die bereits vorgestellte Datenstruktur Surface_mesh.

3D Polyhedral Surface Neben Surface_mesh kann auch die von CGAL entwickelte Halfedge Datenstruktur Polyhedral verwendet werden.

Tabelle 3.1: Vergleich der Unterteilungsalgorithmus Bibliotheken

Bibliothek	**Datenstruktur**	**Unterteilungsalgorithmen**
OpenMesh	Halfedge	Catmull-Clark, Loop, Butterfly
Surface_mesh	Halfedge	keine
OpenSubdiv	Halfedge	Catmull-Clark, Loop
CGoGN	combinatorial maps	Catmull-Clark
CGAL	Halfedge	Catmull-Clark, Loop, Doo-Sabin

Mesquite	1.57
CGAL_list	2.83
CGAL_vector	2.75
OpenMesh	3.37
Surface_mesh	1.13

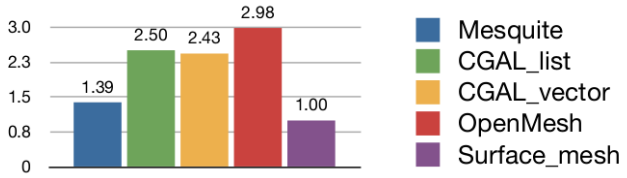

Abbildung 3.5: Benchmarks mit Surface_mesh [11]

CGAL ist sehr mächtig und komplex. Die Bibliothek ist sogar in den meisten Paketquellen der Linux Distributionen enthalten (z. B. Ubuntu) und kann darüber sehr leicht installiert werden. Es sind auch bereits die Unterteilungsalgorithmen *Catmull-Clark*, *Doo-Sabin*, *Loop* und *Sqrt3* implementiert [1].

3.1.6 Vergleich

Tabelle 3.1 gibt einen Überblick über die vorgestellten Bibliotheken und listet auf, welche der ausgewählten Unterteilungsalgorithmen (Catmull-Clark, Loop, Butterfly und Doo-Sabin) bereits implementiert sind.

Mit Ausnahme von Surface_mesh, das wirklich nur die Polygonnetz-Datenstruktur mit elementaren Algorithmen implementiert, sind bei den anderen Bibliotheken bereits einige Unterteilungsalgorithmen implementiert. Bei Verwendung von OpenMesh oder CGAL könnte man sich viel Arbeit sparen, da dort schon fast alle gewünschten Algorithmen implementiert sind. Ziel dieses Projektes ist jedoch eine einfache und schnelle Implementierung der Algorithmen. Surface_mesh selbst besteht nur aus wenigen Dateien und ist die schlankeste Bibliothek von allen vorgestellten. Abb. 3.5 zeigt die Ergebnisse eines Benchmark-Vergleichs für die unterschiedlichen Datenstrukturen. In dem Diagramm werden die Zeiten relativ zu Surface_mesh angegeben.

Man erkennt deutlich, dass die Datenstrukturen von CGAL und OpenMesh vergleichsweise langsam sind. Da Surface_mesh sehr kompakt ist (es besteht nur aus sehr wenigen C++ und H-Dateien) und in dem Benchmark auch sehr gute Ergebnisse liefert, fällt die Wahl auf Surface_mesh.

3.2 Rendering

Im folgenden Unterkapitel werden Bibliotheken und bereits bestehende Software untersucht, mittels derer das Rendering von Polygonnetzen und Limesflächen realisiert werden kann.

Im Gegensatz zur großen Auswahl an unterschiedlichen Möglichkeiten wie beispielsweise bei den Datenstrukturen, beschränkt sich diese hier auf nur wenige Möglichkeiten. Die Wahl ist auf OpenGL wegen seinem großen Funktionsumfang und seiner weiten Verbreitung und auf BezierView auf Grund sehr guter funktionaler Überschneidungen gefallen.

3.2.1 OpenGL

Bei OpenGL handelt es sich um eine sehr weit verbreitete cross-platform Bibliothek für 2-D und 3-D Rendering. Die API-Spezifikation von OpenGL beinhaltet eine Vielzahl von Befehlen, welche sowohl softwarebasiertes, als auch Hardware beschleunigtes Rendering auf der Grafikkarte ermöglichen. Das ermöglicht eine effiziente Umsetzung des Renderings von Polygonnetzen und Flächen.

Diese Effizienz zeigt sich beispielsweise auch bei Standardoperationen wie dem Draw-Call. Bedingt durch die Verwendung eines internen Zustandsautomaten muss der Befehl lediglich mit den veränderten Parametern aufgerufen werden. OpenGL-Befehle folgen einem konsistenten, sprachunabhängigen Namensschema, welches Informationen zum Aufruf und zur Funktionalität des Befehls enthält. So bezeichnet beispielsweise glVertex3fv() einen OpenGL-Befehl (gl), der einen Eckpunkt definiert (Vertex) und drei (3) float (f) Argumente als Zeiger (v) bekommt.

Ein weiterer Vorteil von OpenGL ist die Lizenzfreiheit für Anwendungsentwickler.

3.2.2 BezierView

BezierView (kurz bview) ist ein einfaches Programm zum Rendern von Bézier Patches, rationalen Bézier Patches und Polygonnetzen
[9].

Das Programm wurde von Saleh Dindar, Xiaobin Wu, Jorg Peters (University of Florida) entwickelt und ist für Forschungs- und Lehrzwecke als Open Source Projekt frei verfügbar. BezierView ist ebenso wie SubVis in C++ implementiert und verwendet OpenGL für das Rendering.

Überschneidungen ergeben sich besonders beim Rendern der Polygonnetze und der Limesflächen. Die Implementierung von BezierView kann als Grundlage

für die konkrete Umsetzung von SubVis verwendet werden.

3.3 Grafische Oberfläche

In diesem Abschnitt werden die Bibliotheken, die für die Darstellung nötig sind, vorgestellt.

3.3.1 Qt

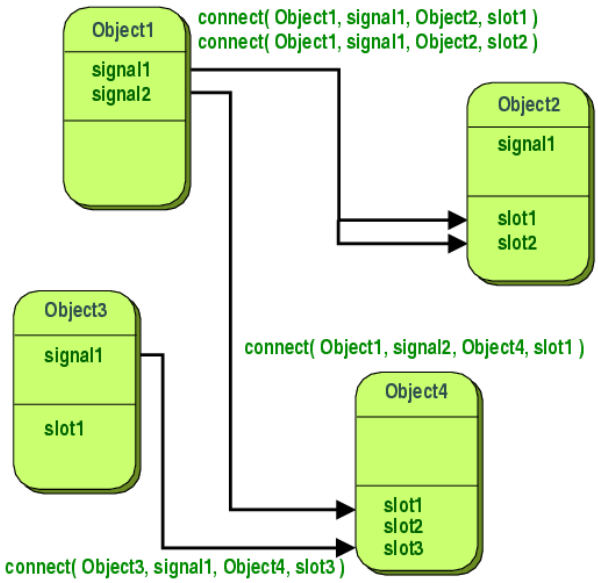

Abbildung 3.6: Signal-Slots Konzept von Qt [3]

Qt ist eine C++ basierte Bibliothek zur Entwicklung von grafischen Anwendungen [3]. Der Vorteil von Qt besteht darin, dass die Anwendungen auf den verschiedenen Betriebssystemen annähernd nativ aussehen, in dem wann immer möglich, die nativen Widgets verwendet werden. Eine Besonderheit von Qt ist das Signals-Slots Konzept (vgl. Abb. 3.6). Dieses kann als eine Art Beobachter-Entwurfsmuster betrachtet werden. Dabei entspricht ein Signal einem *notify* und ein Slot einem *Beobachter*. Mittels einer *connect* Funktion wird die Verbindung zwischen den Komponenten (n:m) hergestellt. Zur Definition von Signalen und Slots werden Makros verwendet, die vom Meta-Object Compiler (moc) in standardkonformen C++ Code umgewandelt werden. Qt wird unter der LGPL verteilt.

3.3.2 libQGLViewer

Die Bibliothek bietet einige grundlegende Funktionen zur Erstellung von 3D OpenGL Betrachtern mit C++ [6]. Sie bietet unter anderem folgende Funktionen bzw. erleichtert deren Erstellung:

- Mit der Maus verschiebbare Kamera
- Weltkoordinatensystem
- Verschiebung von Koordinatensystem und Objekten
- Frame-Animation
- Objektselektion mit der Maus
- Screenshots
- Tastaturkürzel und Maus-Bindings

Die Software ist unter der GNU General Public License (GPL) frei verfügbar.

3.4 IDE

In Bezug auf Qt und C++ bietet sich die IDE *Qt Creator* [4] an. Diese bietet alle gewohnten Funktionen einer IDE wie Syntaxhervorhebung, Autovervollständigung, GUI-Designer, Debugger und Git-Integration. Insbesondere der GUI-Designer nimmt viel Arbeit ab, da so einiges an Boilerplate-Code automatisch erzeugt werden kann.

Projekte werden in sog. *.pro* Dateien konfiguriert. Diese Datei ist plattformunabhängig und relativ schlank gehalten. Erst bei Ausführung durch qmake wird ein plattformspezifisches Makefile generiert, welches dann mittels Make ausgeführt werden kann.

3.5 Dokumentation

Im Bereich C++ ist Doxygen [5] eine weit verbreitete Quellcodedokumentationslösung. Prinzipiell ähnelt es JavaDoc, in dem Quellcode direkt mittels spezielle annotierten Kommentaren versehen wird. Diese Annotationen werden dann mittels Transformation in ein Ausgabeformat (z. B. HTML oder PDF) überführt. Der Vorteil liegt darin, dass so die Dokumentation sehr nahe und eng mit dem Quellcode verbunden ist, was die Aktualität der Dokumentation begünstigt. Qt Creator bietet Syntaxhervorhebung für die speziellen Kommentare und erleichtert so die Verwendung. Doxygen steht unter der GPL.

4 SubVis

Nachfolgend wird das Programm *SubVis* spezifiziert und dessen Architektur vorgestellt. Außerdem werden die Entwicklungsorganisation, Dokumentation sowie verwendete Bibliotheken dargelegt.

4.1 Anforderungen

- Architektur
  - Erweiterbarkeit durch andere Algorithmen mittels Plugins.
  - Plugins können eigene GUI-Elemente in dafür vorgesehenen Bereichen zeichnen.
- GUI
  - Darstellung des Kontrollnetzes
  - Darstellung der Limesfläche
  - Rotation des Objektes
  - Translation des Objektes
  - Skalierung des Objektes
  - Beleuchtung der Oberfläche des Netzes, um Glattheit bewerten zu können.
  - Edit-Modus: Verschieben eines Punktes anhand seiner Flächennormalen.
- Dateiformate / IO
  - OFF-Format und NOFF (mit Farben/Normalen)
  - Laden und Speichern von Polygonnetzen
- Unterteilungsalgorithmen
  - Catmull-Clark
  - Loop
  - Doo-Sabin

  - Butterfly

- Funktionen
  - Variable Anzahl von Unterteilungsschritten
  - Beleuchtungsmodus wählbar

4.2 Verwendete Tools und Bibliotheken

SubVis greift auf die Bibliotheken Qt, libQGLViewer, Surface_mesh, OpenGL und Doxygen zurück. Die verwendeten Versionen sind in Tabelle 4.1 ersichtlich.

Die genannten Versionen beziehen sich auf den jetzigen Zustand. Während der laufenden Entwicklung wird unter Umständen auf eine jeweils höhere Version aktualisiert.

Tabelle 4.1: Versionen der Bibliotheken

Bibliothek	Version
Qt	5.4.1
libQGLViewer	2.6.1
Surface_mesh	1.0
OpenGL	10.1.3-0ubuntu0.4[1]
Doxygen	1.8.6

4.3 Architektur

Es wurde bisher die grundlegende Grob-Architektur festgelegt, welche definiert, welche Komponenten bestehen und wie diese miteinander kommunizieren. In Abb. 4.1 ist diese als UML-Komponentendiagramm dargestellt.

Grundsätzlich wird auf eine Model View Controller (MVC) Architektur gesetzt. Die Model-Komponente soll die Datenstruktur mit dem Polygonnetz beinhalten und Import bzw. Persistenzoperationen anbieten. Das Delegieren der Ereignisse aus der View-Komponente wird von der Controller-Komponente übernommen. Dort sollen auch die entsprechende Signale und Slots definiert werden. Insbesondere die Editieroperationen werden an die MeshEdit-Komponente übergeben, welche dann die MeshData-Komponente ansteuert. In der View-Komponente wird die GUI erstellt und gerendert. Deren Schnittstelle muss Operationen zum Erweitern der GUI und zum Beeinflussen des Renderings des Hauptfensters bereitstellen. Auf diese Schnittstelle greifen dann die Plugins zu, die entsprechend eigene GUI-Elemente zeichnen und Plugin-spezifische Renderingmodi anbieten.

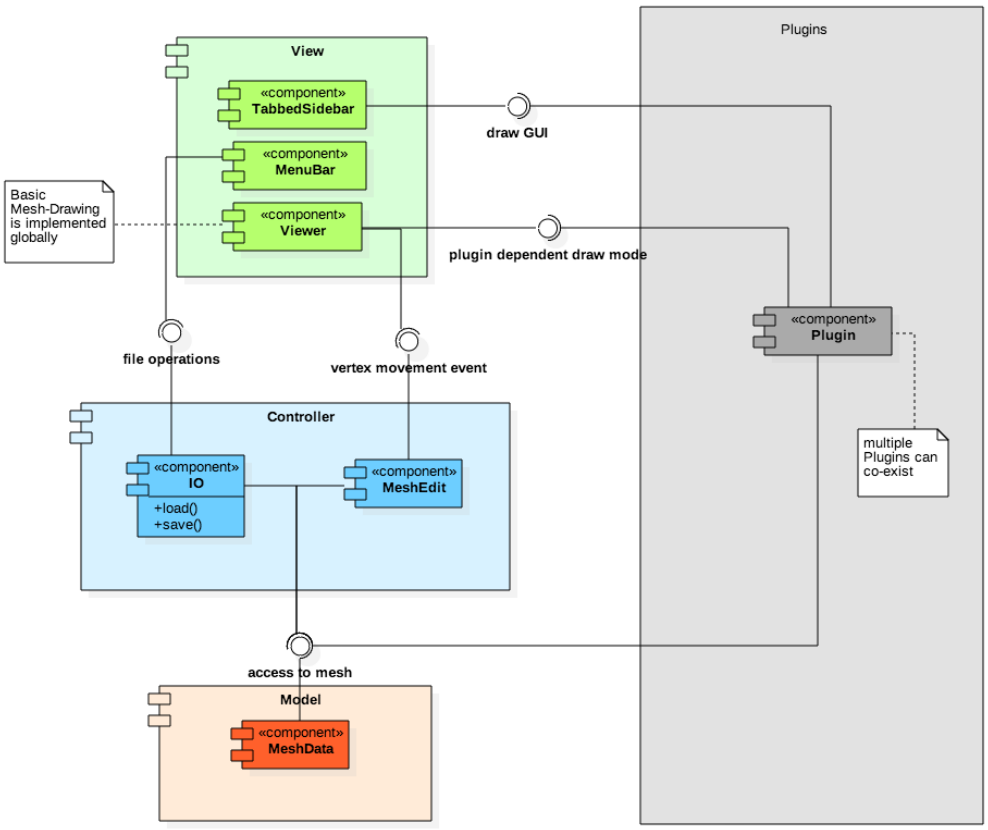

Abbildung 4.1: Geplante Architektur von SubVis

4.4 Grafische Oberfläche

Ein erster Entwurf der grafischen Oberfläche ist in Abb. 4.2 ersichtlich. Während in der oben angeordneten Toolbar die allgemeinen Bedienelemente angeordnet sind, finden sich in der Plugin-spezifischen Seitenleiste die Einstellungen und weitere Optionen. Die tatsächlichen GUI-Elemente in der Seitenleiste sind vom Plugin abhängig.

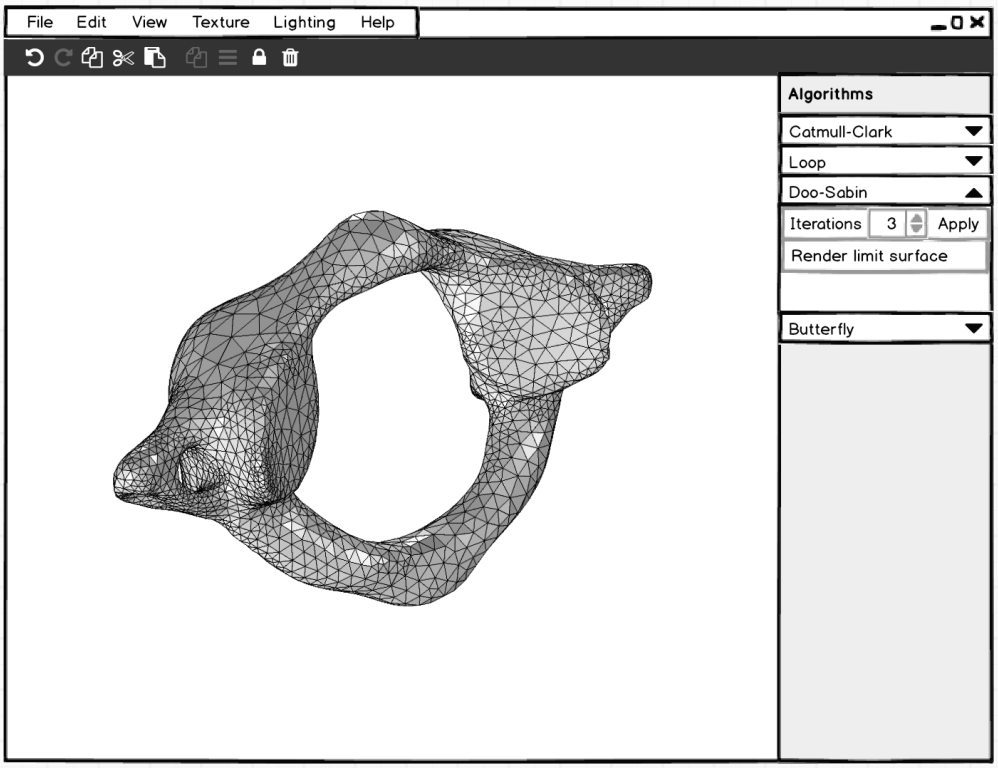

Abbildung 4.2: Mögliche grafische Oberfläche von SubVis

4.5 Dokumentation

Die Dokumentation besteht aus diesem Bericht sowie der finalen schriftlichen Ausarbeitung am Ende des 2. Semesters. Diese Ausarbeitung soll die theoretischen Hintergründe erklären, sowie auf die Entwicklungsprozesse inkl. Vergleich und Entscheidungen für spezifische Technologien eingehen. Ziel ist es, dem Leser eine Top-Down-Ansicht auf das Projekt zu verschaffen. Es wird bewusst auf eine detaillierte Quellcodedokumentation in der Ausarbeitung verzichtet, um die Dokumentation nahe am Quellcode und aktuell zu halten.

Durch Quellcodedokumentierung sollen Entwickler befähigt werden schnell in das Projekt einsteigen zu können und das Programm weiter zu entwickeln

bzw. durch Plugins zu erweitern. Insbesondere bei den Plugins soll ein kurzes Tutorial erstellt werden, welches an die Plugin-Entwicklung heranführt. Des Weiteren soll ein kleines, gut dokumentiertes Plugin entstehen, um den prinzipiellen Aufbau zu veranschaulichen. Wenn zu viel dokumentiert wird, veraltet diese schneller. Deswegen soll *sinnvoll* und *angemessen* dokumentiert werden. Dies bedeutet öffentliche APIs, wenn notwendig, detailliert und ausführlich zu dokumentieren und selbsterklärende Funktionen etc. nicht unnötig zu dokumentieren. Grundsätzlich soll der Quellcode selbst schon als Dokumentation dienen können. Zusätzlich finden sich beim Quellcode README-Dateien, die genaue Details über die verwendeten Funktionen, Build-Anleitungen und die Projektstruktur erläutern.

4.6 Entwicklung

Um eine gemeinsame Entwicklungsumgebung zu schaffen, wurden gewisse Bibliotheken, Tools und Plattformen spezifiziert. Dies betrifft einerseits die Bibliotheken und Tools aus Tabelle 4.1. Andererseits auch das Betriebssystem, Versionsverwaltung, IDE und Programmiersprachenstandard.

Als Betriebssystem wird Ubuntu 14.04 LTS verwendet. Zum Projektabschluss wird jedoch eine plattformunabhängige Anwendung angestrebt. Die Sprachfeatures von C++ sollen maximal dem C++11 Standard entsprechen. Als Versionsverwaltung wird Git in Verbindung mit dem Git-Server des IOS an der HTWG eingesetzt.

Die Entwicklung findet auf dem *develop*-Branch und eventuellen Feature-Branches statt. Dabei sollte ein Branch pro Feature erstellt werden und erst dann in den develop-Branch gemergt werden, wenn das Feature funktioniert. Als IDE wird der vorgestellte Qt Creator verwendet.

Das Projekt teilt sich in zwei Verzeichnisse auf: *SubVis* und *dev-doc*. *SubVis* enthält die gleichlautende Anwendung als Qt-Projekt und unterteilt sich noch in die Ordner *app*, *lib* und *objs*. *app* enthält die Anwendungsteile die selbst entwickelt werden, *lib* die Drittherstellerbibliotheken und *objs* 3D-Modelle zum Testen. *dev-doc* dient der weiterführenden Dokumentation. Das Verzeichnis enthält z. B. Diagramme, diesen Bericht und andere hilfreiche Dokumente zur Entwicklung.

5 Projektverlauf

In den folgenden Abschnitten wird der Projektverlauf beschrieben und ein Rückblick und Ausblick gegeben. Es wird dabei auf die erreichten und geplanten Ergebnisse eingegangen, sowie der bisherige Ablauf des Projekts bewertet.

5.1 Rückblick

Im ersten Semester lag der Schwerpunkt darin, einen Überblick über die theoretischen Grundlagen zu erhalten. Hierzu gehören die Unterteilungsalgorithmen und die Darstellung der Kontrollnetze. Ein weiter wichtiger Punkt war die Evaluierung von Bibliotheken, die bereits Algorithmen und vor allem Datenstrukturen implementieren. Als Ergebnis wurde die Datenstruktur Surface_mesh als am geeignetsten bewertet.

Des Weiteren wurde die Architektur entworfen und in einem UML-Diagramm festgehalten. Außerdem entstand eine einheitliche Entwicklungsumgebung basierend auf Qt und dem Qt Creator unter Ubuntu. Diese enthält die nötigen Build-Dateien, Verzeichnisse, Bibliotheken und Git-Konfigurationen. Jedes Teammitglied besitzt somit eine einheitliche, funktionierende Entwicklungsumgebung. Zusätzlich wurde eine Dokumentationslösung auf Quelltextebene mittels Doxygen implementiert.

Da für alle Beteiligten der Umgang mit C++ und insbesondere Computergrafik ein neues Themengebiet darstellt, war ein ausführlicher theoretischer Einstieg in das Thema notwendig. Hierbei hat sich die Unterstützung durch Herrn Prof. Dr. Georg Umlauf und Pascal Laube als sehr hilfreich erweisen. In den Projektbesprechungen wurden technische Details geklärt und der theoretische Hintergrund erläutert. Erst durch Verstehen der Details bezüglich der Unterteilungsalgorithmen, des Renderings und der Anwendbarkeit der Unterteilungsalgorithmen auf verschiedene Kontrollnetze konnten die Tools und Bibliotheken entsprechend evaluiert werden.

Während des Semesters wurden lediglich für sehr kurze Zeiträume und eng eingegrenzte Themengebiete die Aufgaben zwischen den Teammitgliedern verteilt. Dadurch wurde sichergestellt, dass eine gemeinsame Wissensbasis entsteht, um späteren Missverständnissen vorzubeugen. Dies hat sich als hilfreich

Tabelle 5.1: Aufgabenverteilung unter den Teammitgliedern

Arbeitspaket	Teammitglied
Architektur, Oberfläche, Entwicklungsumgebung	Simon Kessler
Rendering und Darstellung	Tobias Keh
Unterteilungsalgorithmen	Felix Born

erwiesen und ermöglicht es nun an verschiedenen Teilmodulen der Anwendung parallel zu arbeiten.

5.2 Ausblick

Im kommenden Semester liegt der Fokus auf der Implementierung der einzelnen Komponenten des Programms. Das Ergebnis umfasst das spezifizierte Programm *SubVis* und eine schriftliche Ausarbeitung, die den Aufbau und die Umsetzung beschreibt und begründet. Diese enthält zusätzlich eine Dokumentation die eine Top-Down-Ansicht liefert. Implementierungsdetails werden im Quellcode mittels Doxygen dokumentiert.

5.2.1 Aufgabenverteilung

Für die Entwicklung werden die Themen und Aufgaben unter den Teammitgliedern in drei Arbeitspakete aufgeteilt.

Architektur, Oberfläche, Entwicklungsumgebung Hierzu gehört die Oberfläche, Bedienung, Architektur inkl. Plugin-System und die Spezifikation einer Entwicklungsumgebung.

Rendering und Darstellung In diesem Arbeitspaket soll die Berechnung und Darstellung der Kontrollnetze und deren Limesfläche implementiert werden.

Unterteilungsalgorithmen Umfasst die Implementierung der vorgegebenen Unterteilungsalgorithmen.

Tabelle 5.1 zeigt die geplante Aufgabenverteilung für kommendes Semester. Die Zuteilung der Arbeitspakete stellt lediglich einen Startzustand dar. Um jedem Teammitglied Einblicke in alle Aspekte zu ermöglichen und so auch den größten Lerneffekt zu erzielen, werden Teilaufgaben der Arbeitspakete ausgetauscht. Der unterschiedliche Arbeitsaufwand wird kompensiert, in dem nach Fertigstellung eines Arbeitspaketes die entsprechende Person bei den anderen Arbeitspaketen weiter entwickelt.

5.2.2 Zeitplan

In den verbleibenden Semesterferien sollen die folgenden Arbeitsschritte durchgeführt werden. Dabei wird nicht eine endgültige Implementierung angestrebt, sondern eine funktionsfähige prototypische Implementierung, die im kommenden Semester weiter optimiert und ergänzt wird.

- Architektur und Komponenten
- GUI
- IO-Funktionalität
- Plugin-System
- Affine Transformationen
- Unterteilungsalgorithmen
- Rendering des Kontrollnetzes
- Beleuchtungsmodi des Rendering

6 Literatur

[1] CGAL. *The Computational Geometry Algorithms Library*. URL: `http://doc.cgal.org/latest/Manual/index.html` (besucht am 27. 07. 2015).

[2] CGoGN. *n-dimensional meshes with combinatorial mpas*. URL: `http://cgogn.unistra.fr/` (besucht am 27. 07. 2015).

[3] The Qt Company. *Qt - Home*. URL: `http://www.qt.io` (besucht am 10. 08. 2015).

[4] The Qt Company. *Qt - The IDE*. URL: `http://www.qt.io/ide/` (besucht am 10. 08. 2015).

[5] Dimitri van Heesch. *Doxygen: Main Page*. URL: `http://doxygen.org` (besucht am 11. 08. 2015).

[6] *libQGLViewer*. URL: `http://libqglviewer.com` (besucht am 10. 08. 2015).

[7] Open Geometry Processing Library. *Open Geometry Processing Library - Surface Mesh Tutorial*. URL: `http://opengp.github.io/tutorial.html` (besucht am 24. 07. 2015).

[8] M. Nießner u. a. „Feature-adaptive GPU rendering of Catmull-Clark subdivision surfaces“. In: *ACM Transactions on Graphics (TOG)* 31.1 (2012), S. 6.

[9] Jörg Peters. *BezierView*. URL: `http://www.cise.ufl.edu/research/SurfLab/bview/` (besucht am 27. 07. 2015).

[10] Pixar. *OpenSubdiv*. URL: `http://graphics.pixar.com/opensubdiv/docs/intro.html` (besucht am 27. 07. 2015).

[11] D. Sieger und M. Botsch. „Design, Implementation, and Evaluation of the `Surface_mesh` data structure“. In: *Proceedings of the 20th International Meshing Roundtable*. 2011, S. 533–550.

[12] Standford. *Subdivision Surfaces*. URL: `http://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/10_Subdivision.pdf` (besucht am 24. 07. 2015).