Interaktive Erstellung von wissenschaftlichen Illustrationen

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich
Informatik und Mathematik
der Johann Wolfgang Goethe - Universität
in Frankfurt am Main

von
Sebastian Andreas Schäfer
aus
Friedberg

Frankfurt (2015)
(D 30)

vom Fachbereich Informatik und Mathematik der

Johann Wolfgang Goethe - Universität als Dissertation angenommen.

Dekan:

Gutachter:

Datum der Disputation:

Für Emily

Zusammenfassung

Wissenschaftliche Illustrationen werden seit dem 16. Jahrhundert genutzt, um Theorien, Sachverhalte und Erkenntnisse anschaulich zu erklären. Bekannt sind solche Illustrationen vor allem aus den Bereichen Medizin, Biologie und Archäologie. Sie werden heute noch in einem langwierigen Prozess von Hand durch ausgebildete Illustratoren angefertigt. Die Grundlage bildet ein konkreter Gegenstand, der in erster Linie nicht naturalistisch abgebildet wird, sondern abstrahiert und idealisiert dargestellt ist. So kann eine allgemeine Hypothese oder Theorie anschaulich vom Gegenstand losgelöst illustriert werden.

Die Zeichentechniken wissenschaftlicher Illustrationen werden seit Jahrzehnten unter dem Begriff *Non-Photorealistic Rendering* in der Computergraphik erforscht. Mit aktuellen Verfahren lassen sich ästhetisch ansprechende Bilder erzeugen, deren Berechnung allerdings meistens zeitaufwändig ist und die vorwiegend für den nicht-interaktiven Einsatz konzipiert sind.

In dieser Arbeit werden Verfahren vorgestellt, mit dem sich hochaufgelöste wissenschaftliche Illustrationen in einem interaktiven Vorgang erstellen lassen. Die Basis dafür bildet die neu eingeführte GPU-basierte Illustrations-Pipeline, in der auf Grundlage eines 3D-Modells Bildebenen frei angelegt und miteinander kombiniert werden können. In einer Ebene wird ein bestimmter Aspekt der Illustration mit einer auswählbaren Technik gezeigt. Die Parameter der Technik sind interaktiv editierbar. Um Effizienz zu gewährleisten ist das gesamte Verfahren so konzipiert, dass es soweit wie möglich die Berechnungen auf der GPU durchführt. So ist es möglich, dass die Illustrationen mit interaktiven Frameraten gerendert werden.

Ein wesentliches und neues Element sind die Abstraktions- und Korrekturmöglichkeiten, die durch eine Zeichnenfunktion umgesetzt werden. Mit dieser kann nicht nur das Objekt deformiert, sondern auch der Inhalt der Ebenen abgestimmt werden. Der Nutzer kann das Resultat gezielt anpassen, bis es seiner Vorstellung entspricht. Die Zeichnenfunktion operiert komplett auf der GPU und ermöglicht das Zeichnen von Punkten, Linien und parametrischen Kurven auf der Objektoberfläche oder im Bildraum.

Das Framework verarbeitet neben polygonalen Modellen auch Fixfokus-Bilderstapel, wie sie beispielsweise von einem Motorfokus-Mikroskop aufgenommen

werden können. Aus einem solchen Bilderstapel wird mit einem verbesserten Verfahren die 3D-Oberfläche auf der GPU rekonstruiert, das quantitativ hinsichtlich Rekonstruktionsgüte und Geschwindigkeit mit aktuellen Verfahren verglichen ist.

Die den Nutzen des Verfahrens betreffenden Hypothesen wurden mit einer qualitativen Evaluation der prototypischen Implementierung und einer quantitativen Evaluation zu den erzeugten Ergebnissen validiert. Der quantitativen Evaluation liegt eine Online-Umfrage zugrunde.

In der Evaluation zeigte sich, dass Forscher, die nicht zeichnen, von dem Programm am meisten profitieren. Sie können mit dem Programm Skizzen erstellen, auf deren Grundlage ein ausgebildeter Illustrator eine wissenschaftliche Illustration anfertigen kann. Der Forscher kann im Vorfeld Ideen und Techniken interaktiv ausprobieren, ohne (wertvolle) Zeit eines Illustrators zu beanspruchen.

Die Online-Umfrage hat aber auch gezeigt, dass die erstellten Illustrationen den Anspruch erfüllen, im wissenschaftlichen Kontext veröffentlichbar zu sein. Obwohl eine steigende qualitative Anforderung einen höheren Zeitaufwand in der Erstellung erfordert, zeigte sich, dass sich Forscher, die nicht zeichnen, ohne Zugriff auf Illustratoren die Möglichkeit eröffnet, qualitativ hochwertige wissenschaftliche Illustrationen zu erzeugen.

Als Fazit aus den Evaluationen lässt sich formulieren, dass der entwickelte Prototyp erfolgreich im wissenschaftlichen Umfeld eingesetzt werden kann. Allerdings bietet er auch noch Potential zur Weiterentwicklung in verschiedene Richtungen. Neben einem Renderer für volumetrische Daten, wurde vor allem die Integration von weiteren Zeichentechniken gewünscht.

Abstract

Scientific Illustrations have been used since the sixteenth century to depict new scientific theories and findings. Today they are playing an ever more important role in the publication of new research articles, not only in natural sciences like biology or medicine, but also in general sciences like archaeology or palaeontology. They are often created via a tedious and time consuming process by specially trained drafters. An illustration is based on a concrete specimen, but usually shows an abstracted or idealised object to express a theory.

In the field of non-photorealistic-rendering many drawing techniques that are used in the creation of scientific illustrations have been researched for centuries. Current algorithmic methods generate aesthetically pleasing illustrations that are suitable for publication in a scientific context. Unfortunately these methods are predominantly created for an off-line and non-interactive scenario as the calculation of the final images takes a long time. The creation of a desired custom illustration using these algorithms is hence a time-consuming task.

In this work a framework is presented with which high-dpi scientific illustrations and schematics can be created interactively by a non-illustrating researcher. It is based on an extensible GPU-based illustration-pipeline to interactively render characteristics of a 3d-model into image-layers that are combined in an image-editing fashion. The user can choose the techniques used to render each layer and manipulate its key aspects and parameters. The pipeline is constructed to utilise the GPU as much as possible to ensure efficiency.

An important new aspect is the space of abstraction- and correction-possibilities that the pipeline allows, realised using drawing functions. The first allows the user to deform and sculpt the 3d-model and create the abstraction of a concrete specimen. The user can furthermore draw both on the model and into the layers as well as adjust the results of the algorithms to his liking. The drawing functions are realised completely on the GPU and allow points, lines and parametric curves to be drawn.

As well as being able to load 3d-models the framework also supports the processing of fixed-focus image stacks as they are created by motor-focus microscopes. A state-of-the-art algorithm has been extended to reconstruct the 3d-surface on the GPU to create better results more efficiently.

The main hypotheses of this work have been validated by a qualitative evaluation of a prototype implementation and a quantitative evaluation through an online survey. The main use case of such a framework is the creation of scientific illustrations by non-illustrating researchers. They are able to experiment with different viewing and illumination scenarios and parameters interactively and thereby create sketches that can be used to discuss a work order of a scientific illustration drawn by trained drafters.

The online survey has also shown that the scientific illustrations created with the prototype are publishable in a scientific context. Non-illustrating researchers that don't have access to trained drafters are therefore able to create scientific illustrations that are accepted by the community.

The prototype was successfully employed by users in the evaluation, and it also showed that there is interest and potential for further research and extension. Some users asked for the prototype to be extended to also process volumetric data, while others were interested in adding more drawing techniques to extend the scope of the prototype outside just the palaeontological domain at which it was targeted.

Danksagung

Mein Dank gilt …

… meinem Doktorvater Prof. Dr. Krömker, der mir diese Arbeit ermöglicht hat.

… meiner Familie, insbesondere meiner Tochter & meiner Frau, welche mich stets unterstützt und motiviert haben.

… den Mitarbeitern des Senckenberg Forschungsinstituts und Naturmuseums, die als kompetenter, geduldiger und stets ansprechbarer Partner einen maßgeblichen Teil zu dieser Arbeit beigetragen haben: Juliane Eberhardt, Dr. Thomas Lehmann und Dr. Krister Smith.

… Dr. Tobias Isenberg, der mit seinem Fachwissen und Diskussionsbereitschaft mir enorm geholfen hat.

… allen Freunde und Bekannten, die ich hier nicht aufzählen kann.

… meinen Arbeitskollegen bei TomTom.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel

Einleitung

Die Erstellung wissenschaftlicher Illustrationen ist heute immer noch eine Aufgabe, die genauso wie vor Jahrhunderten angegangen wird: Ein ausgebildeter Zeichner fertigt eine Illustration in Absprache mit einem Wissenschaftler in einem mühsamen handwerklichen Prozess an.

Seit Jahrzehnten werden Algorithmen, die das Ziel haben, Zeichentechniken zu imitieren, erforscht. Angefangen bei Arbeiten von Salisbury et al. [Sal94] aus dem Jahr 1994, die die Erstellung von interaktiven *pen-and-ink* Zeichnung behandeln, bis hin zu Arbeiten wie die von Gwosdek et al. [Gwo] aus dem Jahr 2011, die ein Partikelsystem auf der GPU nutzt, um die Punkte einer Punktierungszeichnung zu verteilen.

Auf der weltweit größten Computergrafik-Konferenz *ACM SIGGRAPH* wurden im vergangenem Jahr wieder NPR-Verfahren vorgestellt: z.B. eine Methode zum Berechnung exakter topologischer Konturlinien [BHK14], ein System, das *gemalt*-aussehende Animationen von 3D-Modellen [Bas13] ermöglicht und eine neue Technik, um handgezeichnete 2D-Figuren plastisch zu beleuchten, so dass sie dreidimensional wirken [Sýk14].

Keines der bisherigen Verfahren ist jedoch ausreichend, um wissenschaftliche Illustrationen komplett am Rechner zu erzeugen: Es fehlt ein umfassendes Konzept. Verschiedenste Zeichentechniken müssen frei kombiniert werden können. Die Erstellung sollte zusätzlich interaktiv ablaufen, damit der Benutzer ein direktes Feedback für seine Aktionen bekommt. Und zuletzt muss ein Anwender auch in der Lage sein, das Objekt verändern zu können, um ggf. die von ihm intendierten Abstraktionen erreichen zu können. Oftmals zeigt ein konkretes Fundstück oder ein vorhandenes Objekt nämlich nicht alle Eigenschaften in der Deutlichkeit, wie sie in der wissenschaftlichen Illustration aufgezeigt werden sollen.

Das Ziel dieser Arbeit ist es, ein solches Konzept zu erarbeiten und prototypisch umzusetzen. Vor diesem Hintergrund werden Hypothesen über die Eignung des Konzeptes aufgestellt und anhand des Prototypen durch eine qualitative Evaluation validiert. Da das Programm auf Basis von 3D-Modellen arbeitet, sollen neben polygonalen Modellen auch in der Praxis häufig verwendete Bilderstapel verarbeitet werden können. Hierfür wurde ein bereits bestehendes Verfahren zur Rekonstruktion von 3D-Oberflächen aus

Bilderstapeln verbessert und auf die GPU portiert.

Für ein solches Verfahren sind verschiedene Einsatzszenarien denkbar. Forscher, die nicht als Illustratoren ausgebildet sind, wären damit in der Lage *schnell* qualitativ hochwertige Skizzen zu entwerfen. Diese können z.B. als Grundlage in der Besprechung eines konkreten Zeichnen-Auftrags mit einem Illustrator herangezogen werden. Der Forscher kann so schon bei der Erstellung Techniken ausprobieren und herausfinden, welche Aspekte betont werden sollen.

Ein weiteres denkbares Szenario ist der Einsatz des Programms direkt zur Erstellung von Illustrationen. Dies setzt allerdings voraus, dass die Qualität für eine Veröffentlichung ausreicht. Zuletzt lässt sich das Verfahren, das die Oberfläche eines Objektes aus einem Bilderstapel rekonstruiert, eigenständig nutzen, da das erzeugte 3D-Modell auch mit anderen Programmen verwendet werden kann.

An dieser Stelle soll noch darauf hingewiesen werden, dass die hier vorgestellten Verfahren in Kooperation mit Forschern der Paläontologieabteilung des Naturmuseums und Forschungsinstitutes Senckenberg Frankfurt/Main erstellt wurden. Von ihnen stammen die Illustrationen, die als Vorlage und Muster der Entwicklungsarbeit dienten. Sie haben auch an den Tests und der Evaluation des Prototypen teilgenommen. Die dabei gewonnen Ergebnisse wurden schließlich in einer Onlineumfrage evaluiert.

1.1 Übersicht der Arbeit

In Kapitel 2 werden die nötigen Grundlagen vorgestellt, zu denen einerseits eine genaue Beschreibung wissenschaftlicher Illustrationen und deren Abgrenzung zu wissenschaftlichen Visualisierungen gehören. Gleichzeitig werden OpenGL in der Version 3.2 und verschiedene Techniken der GPU-Programmierung näher vorgestellt.

In dem sich anschließenden Kapitel 3 wird der Stand der Forschung in den verschiedenen für diese Arbeit relevanten Feldern besprochen. Dazu gehören neben Verfahren, die wissenschaftliche Illustrationen oder Visualisierungen erzeugen, auch Techniken der Oberflächenrekonstruktion. Den Hauptteil bildet die Vorstellung von aktuellen Rendertechniken und NPR-Algorithmen. Den Abschluss bildet eine Diskussion von Methoden zur Evaluation von NPR-Verfahren.

Im vierten Kapitel werden die genauen Anforderungen, die an die Pipeline gestellt werden, analysiert. Dazu gehören Anwender, Benutzeroberfläche, Eingabe und die zu erzielende Ausgabe auf Grund von beispielhaften wissenschaftlichen Illustrationen aus der Paläontologie.

Die Hypothesen dieser Arbeit werden im fünften Kapitel vorgestellt.

Im folgenden Kapitel 6 wird dann ein Konzept erarbeitet, mit dem ein Verfahren zur Erfüllung der formulierten Anforderungen sowie der Validierung der Hypothesen geleistet werden kann.

Die wichtigsten und interessantesten Aspekte der Umsetzung werden im siebten Kapitel gezeigt. Hier wird detailliert auf die Probleme und deren Lösung von Rekonstruktion, Rendering, Zeichnenfunktion, Ebenen und GUI eingegangen.

Schließlich werden die Ergebnisse der Arbeit und der Evaluation im achten Kapitel behandelt und bewertet.

Im letzten Kapitel wird ein Fazit gezogen und insbesondere die Frage beantwortet, inwieweit die aufgestellten Hypothesen erfüllt wurden. Am Ende steht ein Ausblick, der auch ungelöste Probleme im Kontext dieser Arbeit aufzeigt.

Kapitel 2

Grundlagen

In diesem Kapitel werden wesentliche Grundlagen erarbeitet, die für diese Arbeit wichtig sind. Vorgestellt werden die wissenschaftliche Illustration, die Techniken ihrer Erzeugung und ihre Einsatzgebiete. Der zweite Teil befasst sich mit modernen Methoden für das hardwaregestützte interaktive Rendering.

2.1 Wissenschaftliche Illustrationen

Das Bild zeigt mir auf einen Blick, wozu es dutzende Seiten eines Buches brauchen würde zu erklären.

Väter und Söhne, Ivan S. Turgenew, 1862 [Tur07]

Der Begriff *Illustration*[1] ist im Duden als *veranschaulichende Bildbeigabe zu einem Text* beschrieben. Er bezeichnet eine Abbildung, deren Ziel die visuelle Darstellung eines Sachverhaltes, Gegenstandes oder Vorgangs ist. Allgemein bekannt sind etwa medizinische Illustrationen der menschlichen Anatomie oder Bilder aus der Botanik und der Zoologie. Als Beispiel soll hier ein Bild der Orchidee *Catasetum tridentatum* von 1862 in Abbildung 2.1 aus [Dar62] dienen.

Die Ausdruckskraft einer Illustration wird meistens durch das Filtern von Informationen und einer abstrahierenden Darstellung verstärkt. So wird durch das Weglassen von unwichtigen Details der Fokus auf die wesentlichen Aspekte gelenkt. Die Abstraktion bzw. Idealisierung erleichtert es dem Betrachter, die dargestellten Merkmale als universelle aufzufassen, die sich auch an anderen Instanzen des selben Objektes auffinden lassen. Dies erleichtert die wissenschaftliche Analyse.

Illustrationen werden aber beispielsweise auch in der Archäologie zu einer möglichst authentischen Dokumention eingesetzt. Da wissenschaftliche Illustrationen überwiegend von Hand erstellt werden, tragen sie die Handschrift ihrer Urheber und gelten somit gleichzeitig auch als ein eigenes 'Kunstwerk'.

Eine wissenschaftliche Illustration ist eine Illustration, die in einem wissenschaftlichen Kontext entstanden oder eingesetzt wird und die in der Regel einen Gegenstand abbildet. Wissenschaftlich werden Illustrationen in den Bereichen Biologie, Medizin,

[1] aus dem Lateinischen *illustrare* - erleuchten, erklären, anschaulich machen, verschönern

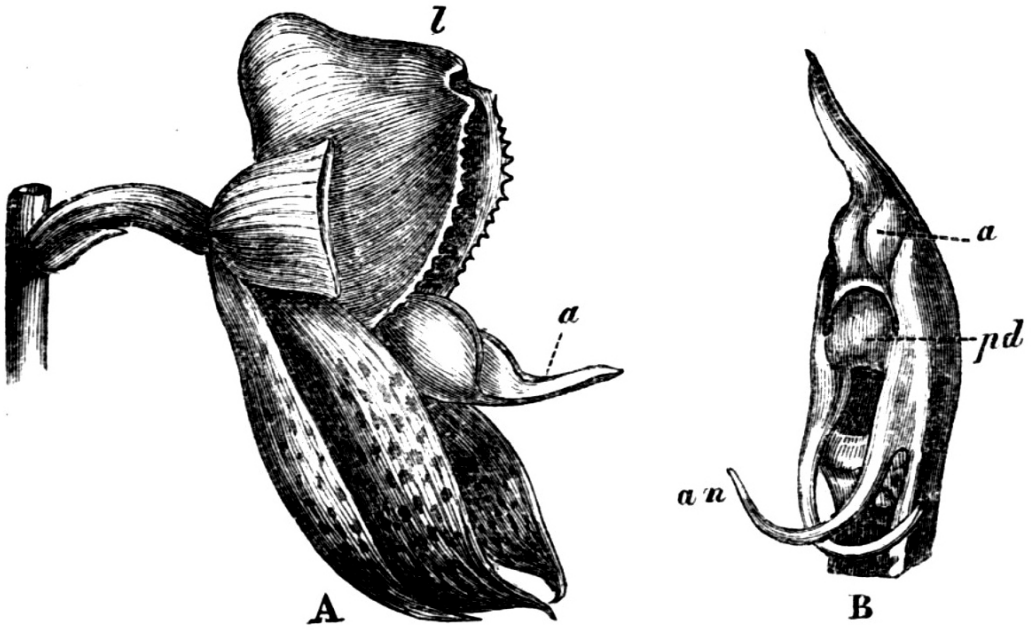


Abbildung 2.1: Zeichnung einer *Catasetum tridentatum* aus [Dar62]

Archäologie und Paläontologie, aber auch in der Geographie und in verschiedenen technischen Anwendungen eingesetzt. Jede Disziplin hat eine bestimmte Sprache und Konventionen entwickelt, die das Aussehen dieser Illustrationen prägen. Es gibt aber darüber hinaus auch allgemeine Richtlinien, die beim Anfertigen einer Illustration zu beachten sind, wie sie etwa in den Lehrbüchern [Fis99] und [Hod03] vermittelt werden.

Die Verwendung der Illustration hat nicht nur expressive, sondern auch historische Gründe. Zu den ersten wichtigen medizinischen Illustrationen zählen die aus Versalius Buch *De Humani Corpus Fabrica libri septem* [Ver73], das im Jahr 1543 veröffentlicht wurde - weit vor der Erfindung der Fotografie (um 1826) [Hoy06]. Eine Illustration aus dieser Publikation wird in Abbildung 2.2 gezeigt. Die zu dieser Zeit gebräuchliche Drucktechnik des Holzschnittes, die auch im Buch des Versalius eingesetzt wurde, benutzt Holzplatten, in die das zu druckende Bild eingeschnitten wird. Da komplexe Bilder sich so nur mit großem Aufwand durch das mehrfache Bedrucken derselben Seite mit unterschiedlich bearbeiteten und eingefärbten Platten realisieren lassen, wurden weitestgehend nur schwarz-weiße Bilder gedruckt. Dies hatte zur Folge, dass Graustufen durch verschiedene andere Methoden, wie z.B. der Schraffurtechnik, approximiert wurden.

Abbildung 2.2: Medizinische Illustration von Versalius (1543) [Ver73]

2.1.1 Farbige Illustrationen

Insbesondere in der Medizin ist es verbreitet, farbige Illustrationen zu veröffentlichen (Abbildung 2.3). Dabei dienen Farben dazu, um mehrere Informationen in einem Bild darstellen zu können.

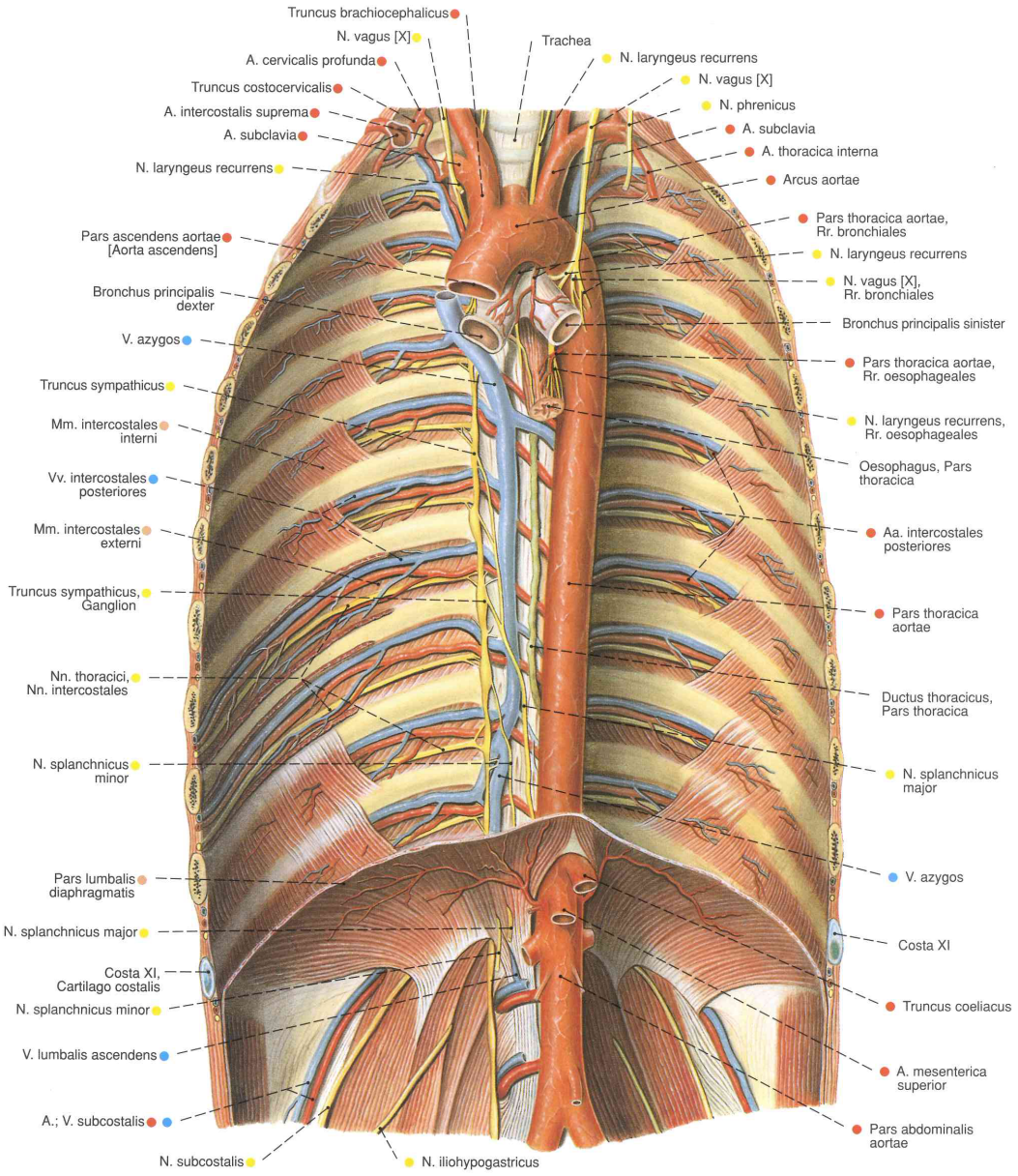


Abbildung 2.3: Farbige medizinische Illustration aus[PP05]

Folgende Färbungen haben sich im Bereich der Anatomie als Konvention herausgebildet (nach [PP05]):

Arterien	rot	*Knochen*	grau/braun
Venen	blau	*Muskeln*	hellrot
Nerven	gelb	*Fett*	hellgelb
Lymphsystem	grünlich	*Haut*	hautfarben

Tabelle 2.1: Übliche Färbungen von Illustrationen der Anatomie [PP05]

2.1.2 Zeichentechniken

Im Folgenden werden Zeichentechniken vorgestellt, die typischerweise in den hier betrachteten wissenschaftlichen Illustrationen eingesetzt werden, wie in den Abbildungen 2.4 und 2.5 gezeigt.

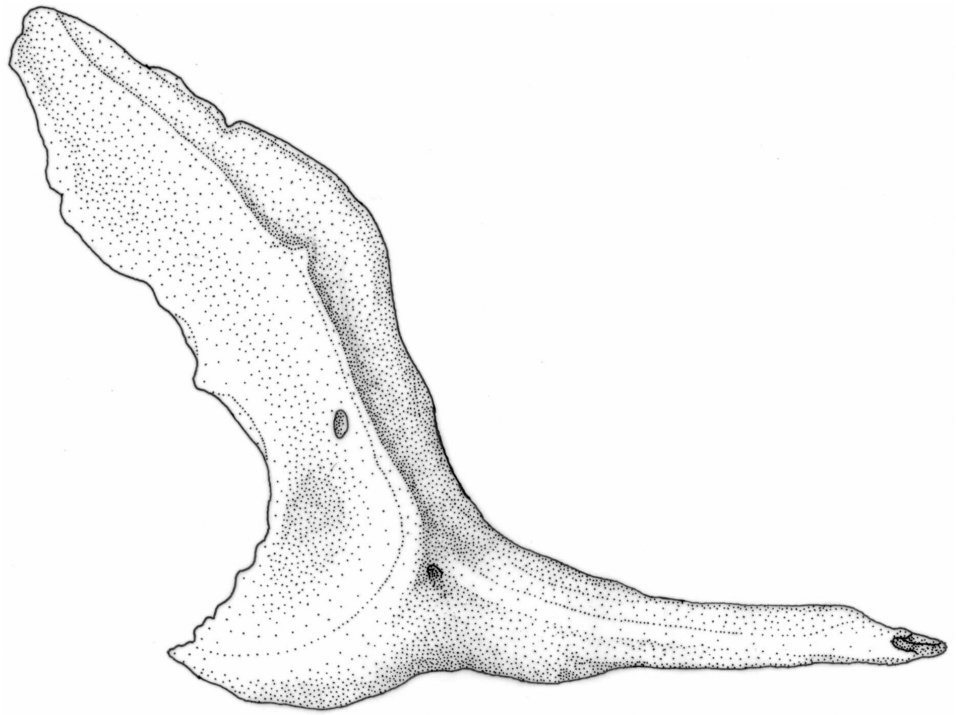


Abbildung 2.4: Illustration eines *Xenosaurus platyceps*-Knochens, Juliane Eberhardt (2012)

Linien

Die Kontur von Objekten wird zumeist durch klare Linien hervorgehoben. Dabei werden überlappende Teile eines Objektes voneinander getrennt. Je nach Detailgrad und Feinheit der gezeichneten Linien bekommt die Illustration einen abstrakten skizzenhaften Charakter. Linien werden auch eingesetzt um bestimmte Merkmale wie z.B. charakteristische Linien zu kennzeichnen.

Pünktelung

Eine häufig eingesetzte Technik der wissenschaftlichen Illustration ist die Pünktelungstechnik, gezeigt in Abbildung 2.4. Der Zeichner/die Zeichnerin benutzt dicht gesetzte Punkte, um entweder Schattierungen zu verdeutlichen oder Oberflächenverläufe und Konturen zu markieren. Entscheidend für die Wirkung einer gepünktelten Illustration ist die scheinbar chaotische aber dennoch regelmäßige Verteilung der Punkte. Je nach gewünschtem Effekt sollen die Punkte dabei entweder keinen Hinweis auf die Oberflächenbeschaffenheit geben oder eine besondere Struktur verdeutlichen.

Da von einer ZeichnerIn jeder Punkt einzeln gesetzt werden muss, ist insbesondere die Pünktelung ein sehr zeitaufwändiges Verfahren. So ist die Illustration in Abbildung 2.5 in einem Zeitraum von einer Woche durch eine technische Zeichnerin erstellt worden.

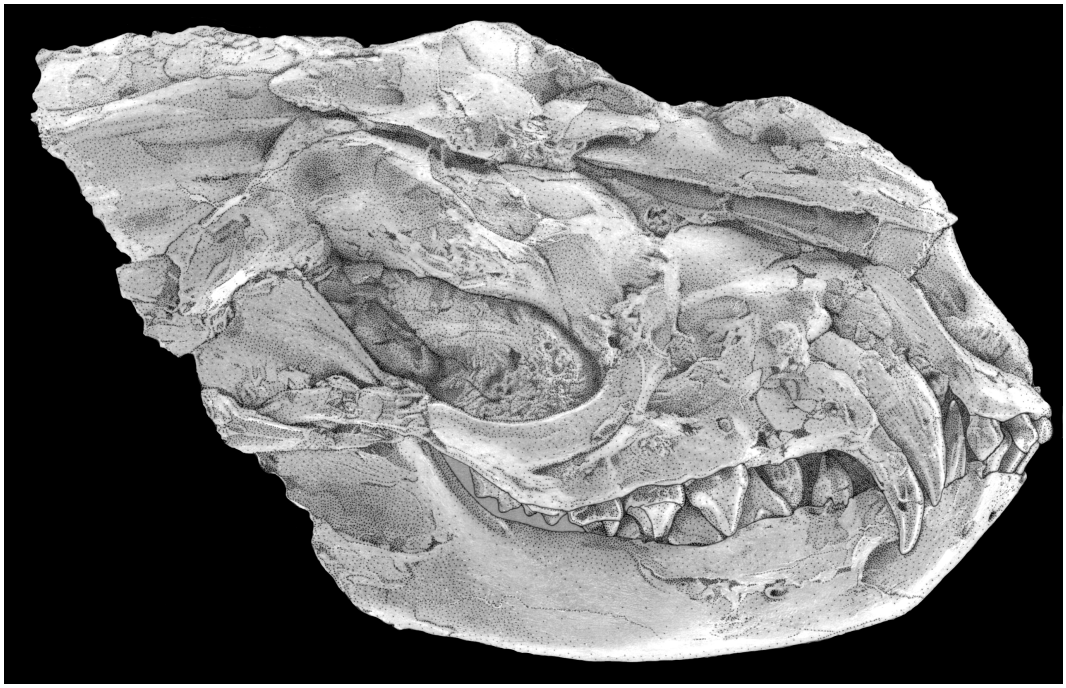


Abbildung 2.5: Illustration eines versteinerten *Kopidodon*-Kopfs, Juliane Eberhardt (2012)

Schraffur

Eine Schraffur[2] besteht aus vielen feinen dicht gezeichneten parallelen Linien. Durch die Dichte der Linien einer Schraffur können verschiedene Helligkeitsstufen dargestellt werden. Als Kreuzschraffur[3] bezeichnet man das Übereinanderlegen von unterschied-

[2] englisch: *hatching*
[3] englisch: *crosshatching*

lich ausgerichteten parallelen Linien. Mit der Kreuzschraffur können zusätzlich intensivere, dunklere Schattierungen dargestellt werden.

Die Form oder Ausrichtung einer Oberfläche lässt sich durch die Form und Ausrichtung einer Schraffur andeuten. Es ist auch möglich, verschiedene Bereiche durch unterschiedlich ausgerichtete und unterschiedlich intensive Schraffierungen oder Kreuzschraffierungen zu kennzeichnen. Die verschiedenen Variationen der Schraffurtechnik sind in Abbildung 2.6 gut zu erkennen. In den Abbildungen 2.1 und 2.2 wird ebenfalls die Schraffurtechnik zur Schattierung eingesetzt.

Abbildung 2.6: *Der Zeichner des liegenden Weibes*, Albrecht Dürer (1515-1525)

2.1.3 Wissenschaftliche Visualisierungen

Beim computer-gestützten wissenschaftlichen Visualisieren geht es nach [Fri09] um die Darstellung von Werten und Daten mit einem Computer. Als Schwerpunkt wird das Rendern von dreidimensionalen Strukturen und Daten aufgefasst. Eine wichtige Komponente ist die Animation der Daten beziehungsweise die Visualisierung von zeitlich veränderlichen Daten. Die Visualisierung kann mit der klassischen Visualisierungspipeline [HM90] erzeugt werden:

Im Gegensatz zu wissenschaftlichen Illustrationen werden bei den Visualisierungen nur selten Verfahren eingesetzt, die Zeichentechniken, wie sie in Abschnitt 2.1.2 vorgestellt wurden, imitieren sollen.

Ein weiteres wichtiges Unterscheidungsmerkmal ist der Anspruch an eine Visualisierung. Eine wissenschaftliche Illustration dient häufig zur Veranschaulichung einer Theorie oder eines abstrakten Konzeptes, in dem ein konkretes Objekt nur ein Puzzlestück ist. Eine Visualisierung auf der anderen Seite handelt vom Konkreten und dient zur Darstellung von vorliegenden Daten. Es gibt auch Fälle, in denen eine Illustration das Ziel hat, das Gesehene möglichst originalgetreu einzufangen. Im Kontext dieser Ar-

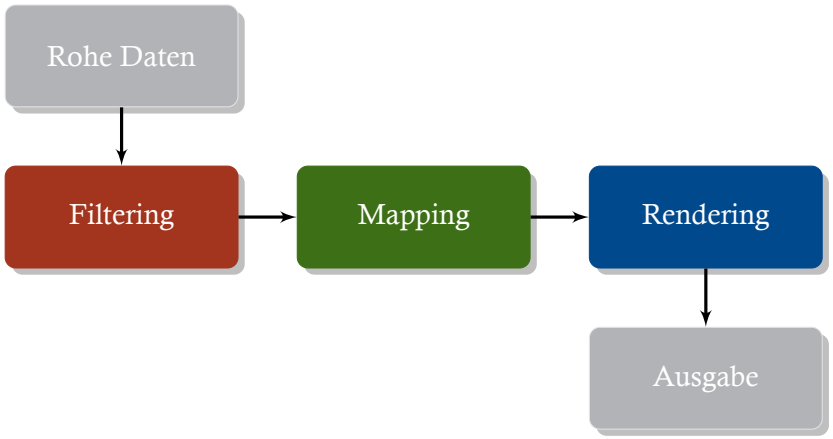


Abbildung 2.7: Visualisierungspipeline, vereinfacht nach [HM90]

beit werden diese als *Untergruppe* der Illustrationen aufgefasst: Der Abstraktionsschritt wurde ausgelassen.

Vor diesem Hintergrund ist es einem Zeichner beim Erstellen einer Illustration auch gestattet, Fakten zur Verdeutlichung übertrieben darzustellen, während bei der Erstellung einer Visualisierung die Daten nur gefiltert werden. Bei diesem Vorgang kann es zwar auch zu *Verfälschungen* des dargestellten Sachverhalts kommen (in [RTB96] werden verschiedene Beispiele genannt), dies ist jedoch bei einer *guten* Visualisierung nicht gewollt. Eine gute Illustration darf also etwas Falsches zum vorhandenen Gegenstand darstellen, eine gute Visualisierung nicht.

2.1.4 Wissenschaftliche Fotografie

Einen Kontrast zu den bisher beschriebenen Möglichkeiten im wissenschaftlichen Kontext Bilder zu erzeugen, bildet die *wissenschaftliche Fotografie*, deren erklärtes Ziel die authentische Reproduktion von Tatsachen, von Geschehenen ist [Ray99]. Zwar wird auch in der wissenschaftlichen Fotografie ein gewisses Maß an Eingriffen geduldet, jedoch dürfen diese keine Tatsachen verfremden und somit nur zum Zwecke der Illustration erfolgen. Ein Beispiel für eine wissenschaftliche Fotografie mit *Manipulation* ist eine Bewegungsstudie, in der Fotos einer Bilderreihe von einer Bewegung zur Verdeutlichung überlagert wurden. Andere Unterarten der wissenschaftlichen Fotografien sind etwa Hochgeschwindigkeitsbilder, Makroaufnahmen oder auch Aufnahmen mit nicht-visuellen Spektren, die mit bloßen Auge nicht sichtbare Bewegungen oder Objekte darstellen.

2.2 Rendering

Die *Renderpipeline*, dargestellt in Abbildung 2.8, wie sie in OpenGL implementiert ist, beschreibt den Weg einer virtuellen Szene hin zu einem Rasterbild. Durch definierte Transformationsschritte der Szenenelemente im Objektraum (in der Abbildung rot markiert) und später dann der gerasterten Bildpunkte im Bildraum (in der Abbildung blau hinterlegt) ist ein universelles Verfahren etabliert, das in den meisten Echtzeit-Renderern umgesetzt ist. Aktuelle Graphikkarten wurden speziell für die Renderpipeline entwickelt, um die darin beschriebenen Operationen schnellstmöglich direkt umzusetzen. Seit der Einführung von Shadern werden Teile der Pipeline flexibel programmierbar gemacht und somit können unterschiedlichste graphische Effekte realisiert werden. Im Folgenden werden die im Prototyp benutzten Techniken näher vorgestellt.

In den ersten Verarbeitungsschritten werden die Eckpunkte der polygonalen Modelle transformiert, so dass sich die gesehenen Eckpunkte im Einheitswürfel befinden. Diese werden zusammen mit den Flächen, die sie definieren, anschließend in Bildschirmkoordinaten gerastert und schließlich wird die resultierende Farbe des korrespondierenden Rasters berechnet. Die Beschreibung ist stark vereinfacht, eine detaillierte Besprechung der Renderpipeline ist etwa in [AHH08], [Shi05] oder [Ros09] zu finden.

2.2.1 Modellformen

Für dreidimensionale Modelle existieren verschiedene Repräsentationsformen. Grundsätzlich unterscheidet man zwischen Oberflächen und Volumen. Aufgrund der Vielzahl an Darstellungsarten wird hier nur auf zwei wesentliche Varianten eingegangen, die auch im Kontext dieser Arbeit eingesetzt wurden: polygonale Oberflächen und Voxel. Eine Sonderform der polygonalen Modelle sind Höhenbilder, die im Kontext dieser Arbeit eine besondere Rolle spielen, weswegen auf sie gesondert eingegangen wird.

Polygonale Modelle

Die Oberfläche des darzustellenden Modells wird mit einem geschlossenen Polygonnetz approximiert. Das Modell kann beispielsweise mit einer Liste von Vertices[5] und Edges[6] beschrieben werden. Die Vertices können auch noch zusätzliche Attribute wie z.B. Farbe oder Oberflächennormale speichern.

Die Renderpipeline, die im vorigen Abschnitt kurz vorgestellt wurde, ist entwickelt worden, um polygonale Modelle (effizient) zu verarbeiten und in ein Rasterbild zu rendern.

[4] Primitive Assembly; deutsch: Primitiven-Konstruktion. Aus Gruppen von Primitiven werden einzelne Primitvie erzeugt.

[5] deutsch: Eckpunkten

[6] deutsch: Kanten

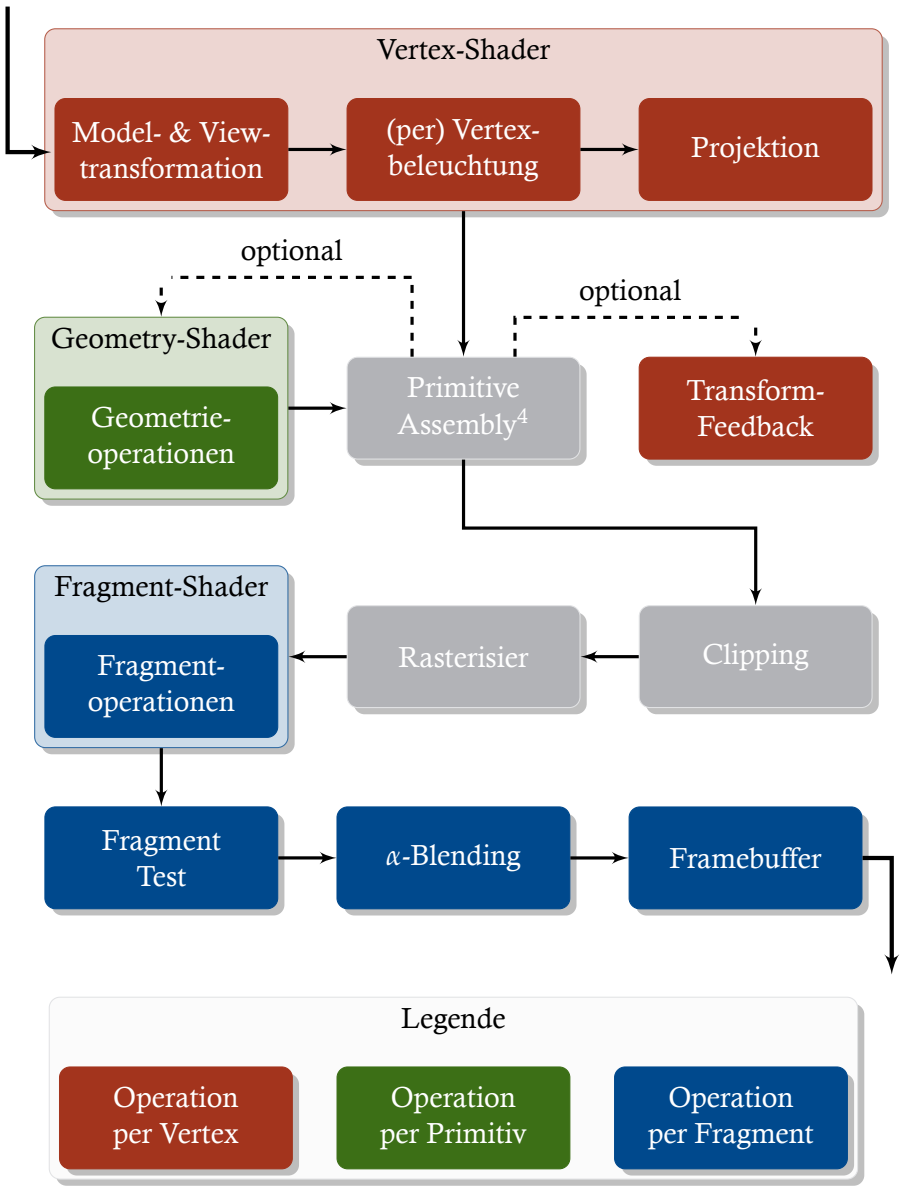


Abbildung 2.8: OpenGL-Pipeline, vereinfacht nach [Ros09]

Voxel

Der Begriff *Voxel* ist ein Kunstwort, das in Anlehnung an das Wort *Pixel*[7] aus der Zusammenziehung der Begriffe Volumetrisches Pixel entstanden ist. Ein Voxel beschreibt den Inhalt eines Raumvolumens. Insbesondere in der Medizin werden häufig Voxel-Daten verarbeitet, die etwa von einem Computertomographie-Scanner (kurz: CT-Scanner), einem Magnetresonanztomographie-Scanner (kurz: MRT-Scanner) oder einem Ultraschallgerät erzeugt werden.

Das Rendern von volumetrischen Daten ist eine besondere Herausforderung für

[7] Pixel - Picture element; deutsch: Bildelement

die Renderpipeline, da die Daten nicht in das klassische Schema der Renderpipeline passen. Anstatt einfacher Schnitttests mit Dreiecken muss das gesamte Volumen mit dem Sichtstrahl geschnitten werden. Eine gute Übersicht der gängigen Verfahren ist im Buch *Real-time Volume Graphics* [Had06] zu finden. Neben Techniken wie dem *Raycasting* existieren verschiedene Ansätze, die eine deutliche Beschleunigung durch Nutzung der GPU bringen. Hervorhebenswert ist das *Gigavoxel* genannte Verfahren von Cyril Crassin [Cra11], mit dem extrem große Volumen sehr effizient gerendert werden können. Kleinere Datensätze können auch durch Punktwolken in Echtzeit verarbeitet werden. Darüber hinaus gibt es auch die Möglichkeit, die Oberfläche des Volumens mit einem polygonalem Modell zu approximieren.

Höhenbild

Ein Höhenbild ist ein digitales Bild, das eine relative Höhe als Grauwert kodiert in jedes Pixel speichert: je heller der Grauwert, desto höher der gespeicherte Höhenwert. Aus einem Höhenbild der Größe $n \times m$ lässt sich direkt ein polygonales Modell der Oberfläche erzeugen, in dem das Höhenbild auf ein $n \times m$ Gitter abgebildet wird. Der Höhenwert des Gitterpunktes (i, j) wird durch die Helligkeit des korrespondierenden Pixels des Höhenbildes bestimmt.

2.2.2 OpenGL 3.2

OpenGL[8] ist ein 3D-API[9] der *Khronos Group*, die plattformübergreifend hardwarebeschleunigtes Rendering ermöglicht. Die Schnittstelle ist im Gegensatz zum parallel entwickelten *DirectX*-API der Firma *Microsoft* unter vielen Betriebssystemen verfügbar, beispielsweise Microsoft Windows, Linux und Apple Mac OS X.

Damit der Prototyp, der im Rahmen dieser Dissertation implementiert wurde, auf verschiedenen Plattformen ausgeführt werden kann, wurde OpenGL als 3D-API ausgewählt. Aufgrund der Beschränkung von Apple in den Betriebssystem OS X 10.7 und OS X 10.8 nur OpenGl 3.2 *core profile* zu unterstützen, wird dieses als Basis für die Entwicklung eines plattformübergreifenden Programms eingesetzt.

Im Schritt von OpenGL 2 auf OpenGL 3 wurde das API stark überarbeitet und ein *deprecated*[10]-Modell eingeführt. Das alte API ist im sogenannten *compatability profile* voll verfügbar, wird allerdings nicht mehr weiterentwickelt. Neue Befehle können ausschließlich im *core profile* genutzt werden, dafür stehen die als *deprecated* markierten nicht mehr zur Verfügung. Hier eine ausgewählte Liste der Features, in denen sich OpenGL 3.2 von den vorherigen Versionen unterscheidet:

[8] **Open G**raphics **L**ibrary; deutsch: Offene Graphik Bibliothek
[9] **A**pplication **P**rogramming **I**nterface; deutsch: Programmierschnittstelle
[10] deutsch: abgelehnt, überholt

- Buffer[11]: Jeder Vertex muss in einem Buffer gespeichert werden - es gibt keinen *immediate mode* mehr.

- Shader: Rendering ausschließlich mit Shaderprogrammen, die mindestens aus Vertex- und Fragment-Shader bestehen.

- Geometry-Shader erlauben es, neue Primitive dynamisch zu erzeugen oder vom Rendering auszuschließen.

- Universelle Buffer können als Vertex- oder Fragment-Daten interpretiert werden.

2.2.3 GLSL 1.5

OpenGL hat mit GLSL[12] eine eigene Shaderhochsprache, die an C/C++ angelehnt ist. Ein Shader ist ein eigenständiges Programm, das auf der GPU ausgeführt wird. Als Besonderheit für die Leistungsfähigkeit der GPU ist es wichtig, dass die Shader parallel verarbeitet werden können. Kommunikation zwischen Shadern gleichen Typs ist nur mit starken Leistungseinbußen möglich; in früheren Versionen war es grundsätzlich verboten. Zusammen mit der Spezifikation von OpenGL 3.2 wurde auch die Sprache GLSL 1.5 veröffentlicht. In dieser werden drei Arten von Shadern definiert: Vertex-, Fragment- und Geometry-Shader, die in den folgenden Abschnitten näher betrachtet werden.

Ein Shaderprogramm oder auch *Effekt* besteht üblicherweise aus einem Vertex- und einem Fragment-Shader. Die Position innerhalb der Renderpipeline ist in Abbildung 2.8 gezeigt. Die Transformation der Vertices wird durch den Vertex-Shader durchgeführt und die Fragmentdaten von einem Fragment-Shader berechnet. Ein Shader hat zwei Arten von Parametern: *uniform* und *varying*. Die *uniform* Parameter sind für jede Shaderinstanz gleich und können mittels der OpenGL-API gesetzt werden. Parameter des Typs *varying* sind entweder Vertexattribute oder werden per Pixel aus den Ausgaben von Vertex- oder Geometry-Shader berechnet.

Vertex-Shader

Ein Vertex-Shader wird auf jeden Vertex angewendet. Zu seinen wichtigsten Aufgaben gehört es, den Vertex an seine *Zielposition* im Einheitswürfel zu bringen und dessen Attribute zu transformieren oder zu erzeugen. Ein Vertex-Shader muss alle Parameter, die der anschließende Shader benötigt, berechnen.

[11] deutsch: Puffer
[12] Open**GL** **S**hading **L**anguage; deutsch: OpenGL Schattierungssprache

Fragment-Shader

Die Aufgabe des Fragment-Shader ist es, die Fragmentdaten pro Pixel zu berechnen. Neben einem oder mehreren *RGBα*-Werten kann auch der Tiefenwert verändert werden. Es ist sogar möglich, den Rendervorgang des Bildfragments abzubrechen.

Geometry-Shader

Der Geometry-Shader kann auf Basis eines Primitives (Punkt, Linie oder Dreieck) dynamisch neue Primitive erzeugen oder bestehende löschen. Durch ihn sind beispielsweise dynamische LODs[13] [LD08] komplett auf der GPU möglich. Mittels *Transform-Feedback* kann der veränderte Vertexbuffer gespeichert werden und im nächsten Rendervorgang als Basis benutzt werden. So ist es z.B. möglich, in jedem Renderschritt neu benötigte Details eines Objektes zu erzeugen und nicht benötigte zu löschen ohne die CPU zu benutzen. Der Vorteil dieser Methode ist, dass die Daten im GPU-Speicher bleiben und nicht zwischen GPU und CPU hin- und zurückkopiert werden müssen.

2.2.4 GPGPU: OpenCL

Der Begriff GPGPU steht für *General-purpose computing on graphics processing units*[14] und bezeichnet die Möglichkeit, die GPU für Berechnungen zu nutzen, die nicht unmittelbar mit dem Rendern zu tun haben. Diese Zweckentfremdung ist in der massiven Berechnungskraft der heutigen Grafikkarten begründet. Forschungsarbeiten wie etwa die Arbeit von Lee et al. [Lee10] belegen, dass ein auf die GPU übertragbares Problem dort meistens deutlich schneller als auf einem Mehrkern-Prozessor berechnet werden kann.

Mit *CUDA*[15] und *OpenCL*[16] gibt es zwei mächtige APIs, die eine direkte allgemeine Verwendung der GPU erlauben. Die Firma NVIDIA hat die API CUDA entworfen und treibt deren Entwicklung voran. Durch die starke Herstellerbindung gibt es auch nur für Grafikkarten mit NVIDIA-GPUs Treiber, die CUDA unterstützen. Aus diesem Grund wird in dieser Arbeit OpenCL eingesetzt, das - wie auch OpenGL - von der *Khronos Group* (weiter)entwickelt wird. Ein weiterer Vorteil von OpenCL gegenüber CUDA ist die Möglichkeit neben der GPU auch andere *Computing Units*[17] zu benutzen. So kann z.B. die CPU benutzt werden, wenn die GPU kein OpenCL kann oder mit anderen Aufgaben ausgelastet ist.

Mittels geteilter Buffer können sowohl OpenGL als auch OpenCL auf den gleichen Speicher zugreifen. Es ist also beispielsweise möglich, ein physikalisches Partikelsys-

[13] level of detail; deutsch: Detailstufe
[14] deutsch: Universelle Berechnungen auf der GPU
[15] **C**ompute **U**nified **D**evice **A**rchitecture;
[16] **Open C**omputing **L**anguage; deutsch: Offene Berechnungssprache
[17] deutsch: Berechnungseinheiten

tem mit OpenCL zu berechnen und die Partikel mit OpenGL als 3D-Punkte (Vertices) anzuzeigen, wobei die Partikel in einem geteilten Buffer gespeichert werden.

Ein OpenCL-Programm, das *Kernel* genannt wird, hat Zugriff auf einen globalen Speicher, auf den alle Kernel schreibend zugreifen können. Ein lokaler Speicher ist für jeden Kernel individuell zugreifbar. Da der globale Speicher gemeinsam genutzt wird, gibt es Synchronisierungsmechanismen in OpenCL. So kann ein Kernel an einem bestimmten Punkt anhalten und warten, bis alle anderen Kernel diesen Punkt erreicht haben und erst dann seine Arbeit fortsetzen. So eine Unterbrechung hat gravierende Auswirkung auf die Performance, ist aber nicht immer vermeidbar, wenn beispielsweise auf das Ergebnis der parallelen Berechnungen zugegriffen werden muss.

In der aktuellen OpenGL Version 4.3 wurden *Compute-Shader* eingeführt, die vergleichbar mit OpenCL Programmen sind. Da OpenGL 4.3 allerdings noch nicht besonders verbreitet ist, wird vom Einsatz abgesehen: Grundlage dieser Arbeit bildet OpenGL 3.2 *core*.

Kapitel 3

Stand der Forschung

Dieses Kapitel befasst sich mit aktuellen Verfahren, die das Problem bzw. Teile des Problems, mit dem sich diese Arbeit befasst, lösen. Die vorgestellten Methoden sind dabei zum Teil auch älter, aber dennoch relevant und verbreitet. Zunächst werden Programme und Methoden vorgestellt, die ein ähnliches Ziel wie diese Arbeit verfolgen. Im zweiten Abschnitt werden Verfahren zur Gewinnung von 3D-Daten und 3D-Modellen vorgestellt. Daran schließt eine Diskussion wichtiger Renderalgorithmen an. Im folgenden vierten Abschnitt werden Methoden für Non-Photorealistic-Rendering vorgestellt. Der letzte Teil widmet sich Möglichkeiten der Evaluation von mit NPR-Methoden erzeugten Bildern und gerenderten Illustrationen.

3.1 Wissenschaftliches Illustrieren

Klassische wissenschaftliche Illustrationen sind auch heute noch von Hand gezeichnet. Jedoch gibt es verschiedene computer-basierte Verfahren und Applikationen, die mittlerweile von Illustratoren genutzt werden, um wissenschaftliche Illustrationen zu erzeugen.

3.1.1 Arbeitsfluss und Software

Der aktuelle Arbeitsfluss beim Erstellen von wissenschaftlichen Illustrationen kann folgendermaßen zusammengefasst werden[1]:

1. Auftragsbesprechung
2. Erstellen eines Bildes totaler Schärfe
3. Anfertigen der Zeichnung von Hand auf (mehrere) transparente Blätter
4. Einscannen der Blätter
5. Nachkorrekturen im Bildbearbeitungsprogramm
6. Einzeichnen von Konturlinien in einem Vektorzeichnenprogramm

[1] Quelle: Gespräch mit Juliane Eberhardt, ausgebildete technische Zeichnerin

Als Basis wird neben dem Objekt selbst auch ein Ausdruck oder Foto verwendet, das unter die transparenten Blätter gelegt werden kann. So können die wesentlichen Merkmale des Objektes direkt *durchgepaust* werden.

Im zweiten Schritt wird oft ein *Bild totaler Schärfe* mit einer Software erzeugt. In diesem Bild sind alle Bereiche des Objektes scharf abgebildet. Dies ist insbesondere bei reinen Fotografien kleiner Objekte nicht immer realisierbar. Die Erzeugung eines solchen Bildes ist aber kein *kreativer Vorgang*, sondern eine maschinelle Anwendung.

Erst im letzten Schritt wird Software eingesetzt, die sich mit der eigentlichen wissenschaftlichen Illustration befasst. Hier kommen vor allem Produkte der Firma *Adobe* zum Einsatz: *Photoshop* zur Bildbearbeitung und *Illustrator* für Vektorzeichnungen. Der Umgang mit den Programmen ist bekannt und beschränkt sich im Grund auf Ebenenkompositionen, einfache Malwerkzeuge und Gradationskorrekturen.

3.1.2 Software zur wissenschaftlichen Visualisierung

Dedizierte Software, mit der sich wissenschaftliche Illustrationen auf Basis von 3D-Modellen erstellen lassen, wie sie in Kapitel 2.1 gezeigt wurden, gibt es gegenwärtig nicht. Software zur Darstellung von 3D-Modellen im wissenschaftlichen Kontext gibt es nur im Kontext der Visualisierung. Hier sind insbesondere medizinische Visualisierungen verbreitet, da hier die Datenlage *günstig* ist und sie auch zur Interpretation der anfallenden Daten benötigt werden. Oft geht es um die Darstellung von CT- oder MRT-Daten. Aus der großen Menge von Programmen wurden drei exemplarisch ausgewählt, die nur kurz vorgestellt werden, da der Fokus eindeutig auf dem Aspekt der Visualisierung, nicht dem Anfertigen einer Illustration liegt. Daran anschließend wird ein Verfahren vorgestellt, mit dem sich illustrative Volumen-Visualisierungen erzeugen lassen.

Keines der Programme und Verfahren, die in diesem Abschnitt vorgestellt und innerhalb der Recherche gefundenen wurden, ist in der Lage, die geforderten wissenschaftlichen Illustrationen zu erstellen. Es fehlt ihnen die Kombinationsmöglichkeit von gezielter künstlerischer Anpassung des Objektes mit der Anwendung von individuell anpassbaren Zeichentechniken.

3D-Slicer

Das unter BSD-Lizenz stehende OpenSource-Programm *3D-Slicer*[2] ist spezialisiert auf medizinische Visualisierungen und Bildanalysen. Es wird im Kern seit 1998 entwickelt [Hir98] und dient als Basis für viele wissenschaftliche Ergebnisse im Bereich der Medizin. Schwerpunkte bilden die verschiedene Module zur Segmentierung und Registrierung sowie zur Analyse der Daten. Die Darstellung kann angepasst werden, allerdings

[2] www.slicer.org, besucht 12.11.2014

durch rein visualisierende Methoden. Es sind keine Module oder Ansätze vorhanden, die gezielt Zeichentechniken imitieren.

Volume Flies on GPU

Das Projekt *Volume Flies on GPU* [PVW10] dient als Programm zur illustrativen Visualisierung von volumetrischen Daten. Das Rendering erfolgt auf der GPU. Das Programm verfügt über verschiedene NPR-Verfahren, die alle durch Partikel auf der GPU gerendert werden. Man hat Schraffur, Pünktelung und das Zeichnen der Kontur umgesetzt. Die Verfahren können auch miteinander kombiniert werden, der Fokus liegt jedoch auf der grundsätzlichen Realisierung der Methoden für medizinische Volumen. Ansätze, die Ergebnisse auch interaktiv zu editieren oder sonstige Formen der Manipulation, sind nicht vorgesehen. Auch werden polygonale Modelle nicht unterstützt.

Amira

Die kommerzielle Software *Amira*[3] entstand ursprünglich als Forschungsprojekt an der FU Berlin. Sie wird in der paläontologischen Abteilung des Naturmuseums und Forschungsinstitutes Senckenberg Frankfurt/Main hauptsächlich zur Bearbeitung von 3D-Daten und zur Visualisierung eingesetzt. Die Software unterstützt alle gebräuchlichen Dateiformate und hat einen großen Funktionsumfang. Die gezielte Manipulation von 3D-Modellen ist mit dem Programm möglich, allerdings sind die Visualisierungsmöglichkeiten auf traditionelle wissenschaftlichen Visualisierungen beschränkt. Es gibt keine Möglichkeit Verfahren zu integrieren, die Zeichentechniken imitieren.

Interactive Illustrative Volume Visualization

Eine Dissertation aus dem Jahr 2008 mit dem Titel *Interactive Illustrative Volume Visualization* [Bru08] hat sich mit dem interaktiven Erstellen von Volumen-Visualisierungen befasst. Es werden ausschließlich volumetrische Daten verarbeitet. Da illustrative Visualisierungen das Kernthema der Arbeit sind, werden Zeichentechniken, wie sie in Illustrationen eingesetzt werden, nicht betrachtet.

Die Interaktivität bezieht sich auch vordergründig auf die Darstellung und die Möglichkeit *Ghosted Views* und *Exploded Views* kontrolliert einzusetzen, wie in Abbildung 3.1 gezeigt. Ansätze, die aus der Visualisierung eine Illustration machen, sei es durch Abstraktion, Korrektur oder Manipulation, sind nicht vorgesehen.

[3] www.vsg3d.com/amira/amira, besucht am 12.12.2014

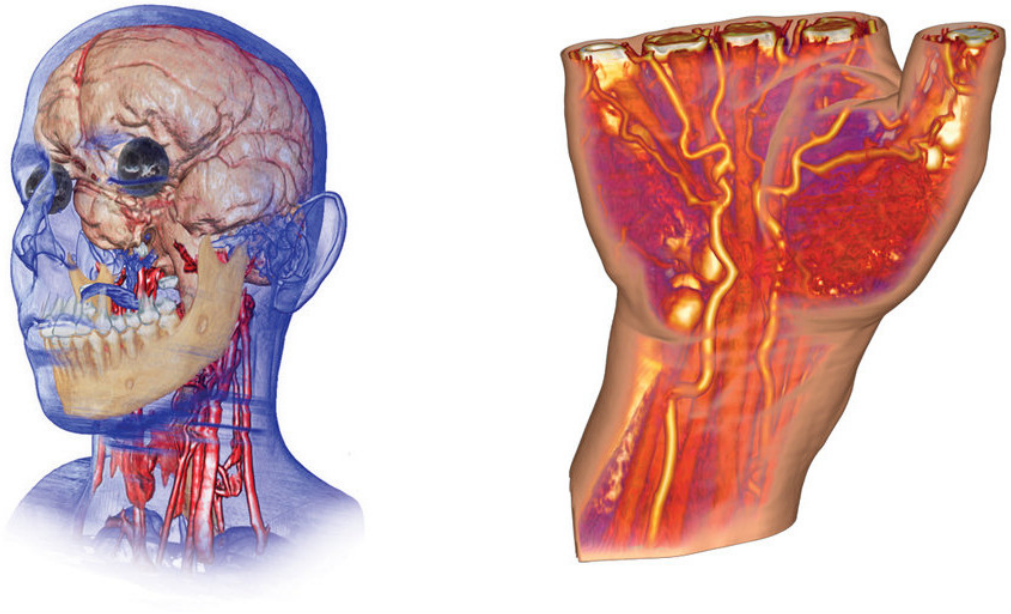


Abbildung 3.1: Bilder des *Interactive Illustrative Volume Visualization*-Verfahrens, [Bru08]

3.2 Erfassen von 3D-Modellen

Es gibt verschiedenste Verfahren, die aus einem realem Objekt entweder ein volumetrisches Modell oder ein Oberflächenmodell erzeugen, das beim Rendern eingesetzt werden kann. Da es sich bei den zu scannenden Objekten ggf. um wertvolle Fundstücke handelt, dürfen nur nichtinvasive Scanmethoden angewendet werden.

3.2.1 Übersicht

- CT-Scanner & (μ-)CT-Scanner

  Ein CT-Scanner berechnet für jedes Voxel einen Absorptionsgrad der Hounsfield-Skala. Diese gibt an, wie stark die Röntgenstrahlung in diesem Volumen abgeschwächt wird. Voxeldaten sind üblicherweise noch nachzubearbeiten, bevor sie verwendet werden. Ein μ-CT-Scanner hat eine besonders hohe Auflösung von etwa 0.4 Mikrometer[4].

- Streifenlicht-Scanner

  Ein Streifenlicht-Scanner projiziert unterschiedliche, definierte Streifenmuster auf das zu scannende Objekt. Durch einen kalibrierten Projektoraufbau können Oberflächenpunkte durch eine Triangulation mit einer Auflösung von weniger als 3

[4]Quelle: www.bruker-microct.com/products/1272.htm, besucht: 07.12.2014

Mikrometer rekonstruiert werden [FCH01]. Aus der Punktwolke kann mit erprobten Methoden, wie beispielsweise der *Poisson surface reconstruction* [KBH06], ein polygonales Modell rekonstruiert werden.

- Konfokaler Laserscanner

  Der konfokale Laserscanner tastet das Objekt mit einem Laserstrahl ab. Durch den Einsatz einer Lochblende kann gezielt der reflektierte Laserstrahl einer Schärfeebene eingefangen werden. Verschiebt man nun die Schärfeebene über das Objekt, kann man die 3D-Oberfläche des Objektes aus allen als fokussiert erkannten Punkten zusammensetzen. Je nach Wellenlänge des benutzten Laserlichts kann z.B. bei einer Wellenlänge $\lambda = 405nm$ eine Höhenmessung von $> 12nm$ durchgeführt werden[5]. Als Ergebnis erhält man ein Höhenbild, aus dem die Oberfläche rekonstruiert werden kann (vgl. Abschnitt 2.2.1).

- Multifokus Mikroskopie

  Die Multifokus Mikroskopie ist einem konfokalen Laserscanner ähnlich, da auch hier das Objekt aus verschiedenen Schärfeebenen heraus betrachtet wird und man als Ergebnis ein Höhenbild erhält. Verändert man die Abbildungsparameter nicht und bewegt das Objektiv auf das Objekt zu, erhält man einen *Multifokus Bilderstapel*, aus dem sich die Oberfläche rekonstruieren lässt. Diese ergibt sich aus den scharfen Bereichen aller Bilder. Als Vorteil dieser Technik gilt, dass vom Objekt auch eine Abbildung erstellt werden kann, die keine Tiefenunschärfe aufweist - ein Bild totaler Schärfe. Die Auflösung hängt von der Abbildungsleistung des verwendeten Mikroskops ab.

Im Folgenden wird die Methode der *Multifokus Mikroskopie* näher betrachtet, da mit ihr nicht nur sehr kleine Fundstücke verarbeitet werden können, sondern weil die Methode im Einsatz auch vergleichsweise günstig ist. Entsprechende multifokale Bilderstapel lassen sich leicht mit verschiedenen Mitteln erzeugen und sind nicht auf ein Motorfokus-Mikroskop beschränkt.

3.2.2 Multifokus Bilderstapel

Mit dem vorhandenen Mikroskop und der zugehörigen Software *Leica LAS Montage Module*[6] und *LAS 3D Viewer Module* ist es auf einfache Weise möglich, einen Bilderstapel zu erzeugen. Unter Angabe des minimalen und maximalen Abstands zwischen Objekt und Objektiv sowie der Anzahl der zu erzeugenden Bilder berechnet die Software die Abstände und fährt sie selbständig an. Ein Stapel mit 63 Bildern ist so in etwa 3 Minuten

[5] Quelle: www.olympus-ims.com/en/metrology/ols4000/, besucht: 07.12.2014

[6] Quelle: www.leica-microsystems.com/products/microscope-software/life-sciences/las-easy-and-efficient/, besucht: 07.12.2014

erstellt. In Abbildung 3.2 sind Bilder eines Stapels im Original sowie maskiert gezeigt. Die maskierten Bereiche sind vom hier implementierten Verfahren als scharf erkannt worden.

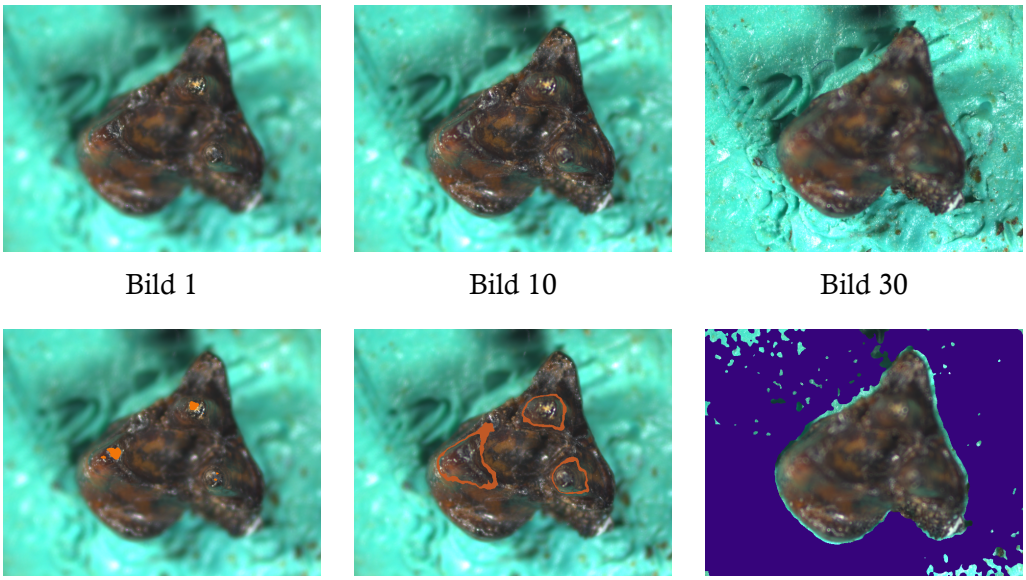


Abbildung 3.2: Ausgewählte Bilder eines Bilderstapels mit 30 Bildern des *Leptictidium*-Zahns: originale Bilder (oben) und markierte scharf erkannte Bereiche (unten)

Es gibt verschiedene kommerzielle und freie Softwareprodukte, die ein *Höhenbild* aus einem solchem Bilderstapel rekonstruieren können. Daneben gibt es auch publizierte Verfahren, die dieses Problem lösen. Es ist nicht bekannt, dass eines der Verfahren aktiv die GPU benutzt. Zur Veranschaulichung der Qualität einer Rekonstruktion sind verschiedene rekonstruierte Höhenbilder zu den Verfahren am Ende des Abschnitts in Abbildung 3.3 gezeigt.

Leica-Software

Die Software *Leica LAS Montage Module* und *LAS 3D Viewer Module* von Leica bieten die Möglichkeit, zusätzlich zu einem Bilderstapel auch die Oberflächen zu rekonstruieren. Das dabei eingesetzte Verfahren ist unbekannt. Wie in Abbildung 3.3 erkannt werden kann, ist die Qualität des rekonstruierten Höhenbildes schlecht. Neben deutlich zu erkennenden fehlerhaften Punkten ist das Ergebnisbild entweder zu stufig oder zu glatt.

Helicon Focus

Die kommerzielle Software *Helicon Focus*[7] bietet dem Nutzer eine Auswahl von drei Verfahren zur Rekonstruktion der Tiefe an: *Weighted average*, *Depthmap* und *Pyramid*. Zusätzlich sind bei allen Verfahren die Parameter *Radius* und *Smooth* einstellbar. Als

[7] www.heliconsoft.com/heliconfocus.html, besucht: 07.12.2014

Besonderheit bietet das Programm eine manuelle Korrekturfunktion für das Bild totaler Schärfe. Bereiche, die fälschlicherweise als scharf erkannt wurden, können markiert und durch Bildbereiche eines Bildes des Stapels ersetzt werden. Zur Erzeugung der Bilder wurde die Evaluationsversion 6.2.2 vom 05.09.2014 genutzt. Die in Abbildung 3.3 gezeigten Ergebnisbilder weisen wenige Fehler auf, haben aber erkennbare diskrete Stufen. Trotz der verschiedenen Rekonstruktionsmethoden und -parameter, von denen sich die *Depthmap* mit den Standardwerten als am geeignetsten erwiesen hat, konnte kein vollständig überzeugendes Höhenbild generiert werden. Es wurde für Helicon Focus das plausibelste Bild der drei Methoden ausgewählt.

Shape From Focus

Das *Shape From Focus*-Verfahren [NN94] rekonstruiert die Tiefe aus einem Bilderstapel. Dazu werden zunächst die fokussierten Bereiche durch Gradienten in jedem Bild bestimmt. Dabei korreliert eine hohe Energie mit einer scharfen Abbildung. Über die gesamte Bilderserie hinweg werden die Bereiche mit der höchsten Energie zusammengesetzt und durch Abspeichern der relativen Höhe des Quellbildes erhält man ein Höhenbild. Das Verfahren wurde gemäß [NN94] umgesetzt und in *C#* implementiert.

Eine Weiterentwicklung des Verfahrens mit dem Namen *Improving Shape From Focus Using Defocus Information* [PR06] benutzt Unschärfeinformationen um die Ergebnisse zu verbessern. Im Originalverfahren lassen sich nur diskrete Höhenstufen erkennen, die durch die Aufnahmehöhe vorgegeben sind. Hat ein Pixel in keinem Bild eine ausreichende Energie, so kann aufgrund der Energie innerhalb der Bildfolge die korrekte Höhe interpoliert werden. Ferner ist es auch möglich, durch eine polynomiale Interpolation die Oberfläche zu rekonstruieren [CAY99]. Bei diesem Verfahren wird ein Suchfenster über das rekonstruierte Höhenbild geschoben, in dem die Höhe des Fenstermittelpunkts durch eine polynomiale Interpolation aus dem Suchfenster neu berechnet wird. In internen Tests hat diese Verbesserung allerdings nur in wenigen ausgewählten Fällen zu einer sichtbaren qualitativen Zugewinn geführt: Die falsch erkannten Bereiche sind so groß, dass das Suchfenster zu klein ist.

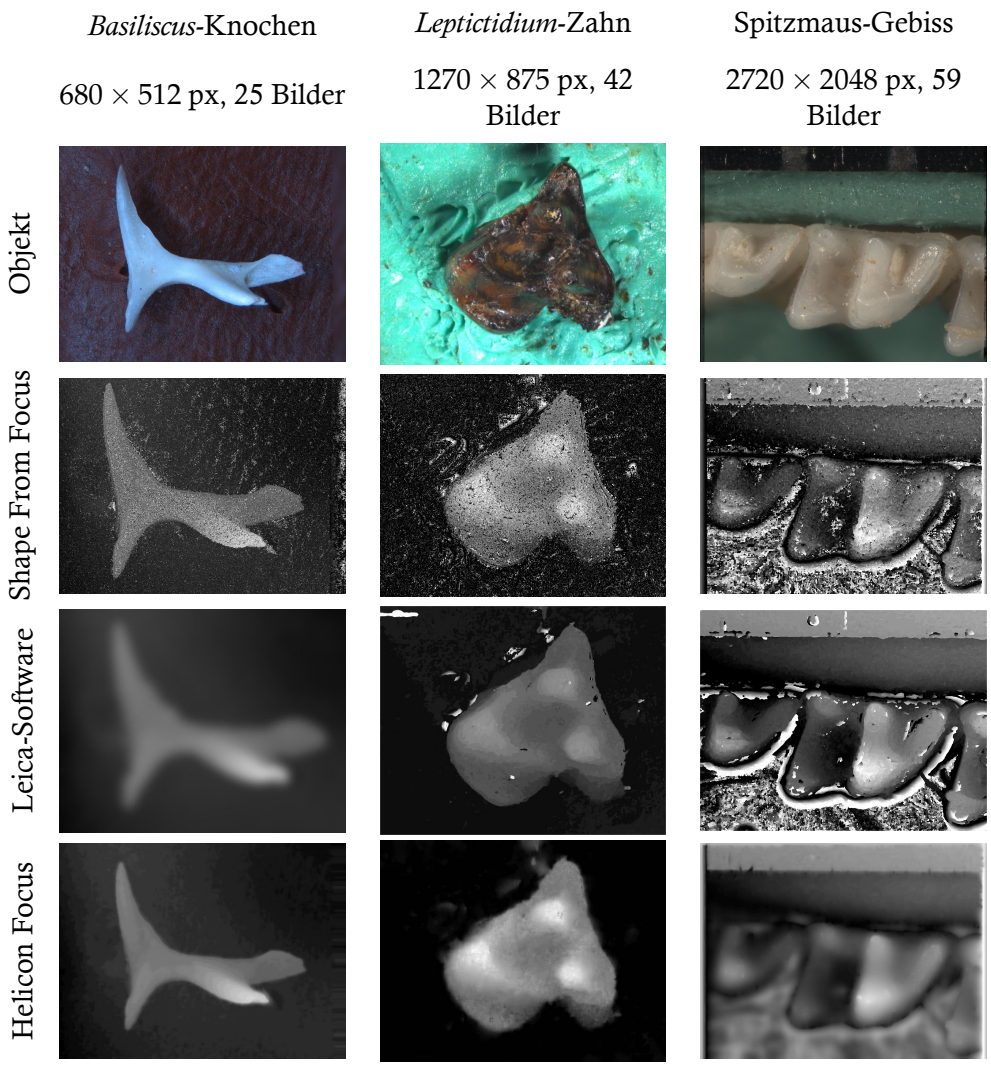


Abbildung 3.3: Mit den aktuellen Verfahren rekonstruierte Höhenbilder

3.3 Rendering

3.3.1 G-Buffer

Die Verwendung von G-Buffer[8] geht zurück auf Saito und Takahashi [ST90]. In G-Buffer werden verschiedene geometrische Eigenschaften des gerenderten Objektes für jeden Pixel geschrieben. Sie werden unter dem Begriff *Deferred Shading*[9] zunehmend in anderen Anwendungsbereichen (beispielsweise beim Non-Photorealistic-Rendering oder in Spielen) eingesetzt.

Anstatt ein Modell direkt mit dem gewünschten Material in den Framebuffer zu rendern, erfolgt die Schattierung erst in einem späteren Schritt. Stattdessen werden im ersten Schritt alle für das Shading benötigten Informationen in (mehrere) G-Buffer geschrieben. Üblicherweise [HH04] sind dies folgende Daten (siehe Abbildung 3.4):

- Oberflächennormale,
- Albedofarbe,
- Beleuchtungsparameter und
- Weltposition oder Tiefe, aus der die Weltposition rekonstruiert werden kann.

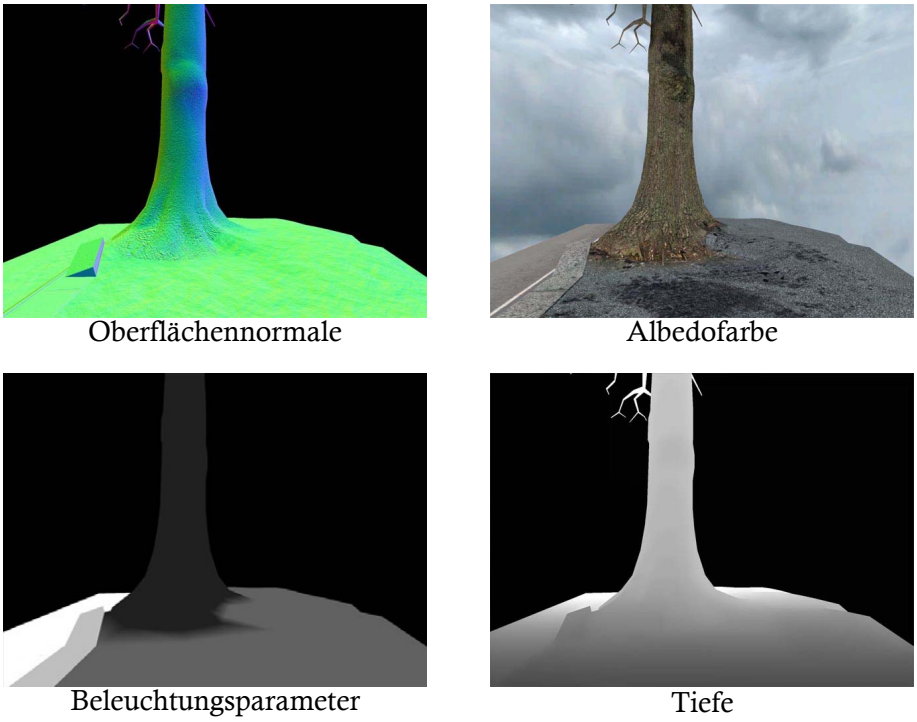


Abbildung 3.4: Vier G-Buffer als Bild dargestellt, aus [HH04]

[8] Kurzform von geometric buffers, deutsch: geometrische Puffer
[9] deutsch: zurückgestelltes/verschobenes Schattieren

Mit diesen Informationen kann eine komplexe Beleuchtungsrechnung erfolgen. Werden dieselben Daten für weitere Effekte und zusätzliche Materialien benötigt, liegen diese bereits vor, ohne dass das Modell erneut gerendert werden muss. So können beispielsweise Szenen mit vielen Lichtern sehr effizient gerendert werden. Mit traditionellem Shading ist der dafür notwendige Shader entweder sehr komplex (da er alle Lichter berücksichtigen muss) oder die Geometrie muss mehrfach (einmal pro Licht) gerendert werden. Mit *Deferred Shading* reicht es aus, die Geometrie einmal in die G-Buffer zu rendern. Durch den Einsatz von Multiple-Render-Targets und Render-To-Texture können in einem einzigen Renderschritt alle Daten des Models in mehrere Texturen geschrieben werden, die dann in den folgenden Schritten eingesetzt werden können. Jedes Licht kann nun auf die Werte des G-Buffer zugreifen und seine Beleuchtung additiv in das Ergebnisbild einzeichnen. Das Shading kann so letztendlich als ein reiner Bildfilter realisiert werden.

Der Einsatz von G-Buffer ist allerdings auch mit Nachteilen verbunden. Neben dem großem Speicherbedarf für die notwendigen Texturen ist auch der direkte Einsatz von Hardware-Antialiasing-Verfahren nicht direkt umsetzbar. Dafür sind andere Methoden anzuwenden, wie sie im Abschnitt 3.3.4 vorgestellt werden.

G-Buffer sind im Kontext von NPR-Algorithmen meistens ein guter Ausgangspunkt [SS02], da sie meistens im Bildraum arbeiten und auf Daten des G-Buffers zugreifen: Tiefe, Normale und Albedofarbe. In der Arbeit von Saito und Takahashi [ST90], in der G-Buffer vorgestellt wurden, ist sogar explizit ein Beispiel gezeigt, in der ein Torus im NPR-Stil aus G-Buffer erzeugt wird. Die Schattierung ist durch eine Schraffur dargestellt und eine Konturlinie ist extrahiert worden.

3.3.2 Beleuchtung

In der Computergrafik wird zwischen lokaler und globaler Beleuchtungsrechnung unterschieden. Erstere wird in der Renderpipeline eingesetzt, da sie schnell zu berechnen ist. Sie betrachtet ausschließlich die Daten der aktuell zu rendernden Oberfläche und die der Lichtquellen. Andere Geometrien oder Oberflächen werden nicht berücksichtigt, so dass sich Schatten oder Spiegelungen nicht unmittelbar berechnen lassen.

Die globale Beleuchtungsrechnung (Ray-Tracing oder Radiosity) berücksichtigt hingegen alle Geometrien einer Szene und ist daher auch entsprechend aufwändiger zu berechnen. Aktuelle Forschungsergebnisse benutzen High-End GPUs um die Verfahren so zu beschleunigen, dass entsprechende Verfahren in Echtzeit ausgeführt werden können.

Im Kontext dieser Arbeit wird ausschließlich die lokale Beleuchtungsrechnung betrachtet. Mit speziellen *Ambient Occlusion*-Techniken, die in Abschnitt 3.3.3 vorgestellt werden, sind verschiedene Effekte der globalen Beleuchtungsrechnung für die Anforderung im Rahmen der Arbeit hinreichend genau simuliert.

Für die lokale Beleuchtungsrechnung wird hauptsächlich das Blinn-Phong-Modell eingesetzt [Bli77], das die Beleuchtung in ambientes, diffuses und spiegelnd-diffuses Licht aufteilt. Wissenschaftliche Illustrationen - auch von versteinerten Objekten - geben die Objekte meistens als diffus-reflektierende Objekte wieder, spiegelnd-diffuse Materialeigenschaften sind nicht dargestellt. Aus diesem Grund werden nun zwei diffuse Beleuchtungsformen vorgestellt.

Lambert-Beleuchtung

Die Lambert'sche diffuse Beleuchtungsformel ist eine physikalische Formel, die die ideale diffuse Reflexion beschreibt. Die Intensität des abgestrahlten Lichts I_{Diff} ist unabhängig von der Kameraposition und hängt ausschließlich vom einfallenden Winkel θ_l ab, der sich aus der Oberflächennormale $\vec{N}$ und dem Vektor zum Licht $\vec{L}$ ergibt:

$$I_{\text{Diff}} = \vec{N} \cdot \frac{\vec{L}}{|\vec{L}|} = \cos\theta_l$$

Die Beleuchtung nach Oren-Nayar

Ein Beleuchtungsmodell für raue Oberflächen wird in der Arbeit von Oren und Nayar [ON94] präsentiert, in der die Oberfläche durch Microfacetten modelliert wird. Jede Facette wird als ideal diffus reflektierendes Element betrachtet, das mittels der Lambert'schen Beleuchtungsformel zur Gesamtintensität des reflektierten Lichtes I_{ON} additiv beiträgt. Im Vergleich zu Lambert-Beleuchtung wird zusätzlich zum Vektor zum Licht (θ_l, ϕ_l) noch der Vektor zum Betrachter (θ_c, ϕ_c) als Hemisphärenwinkel benötigt. Die Rauheit kann über den Parameter σ gesteuert werden, der die Streuung der Microfacetten reguliert: Ist der Parameter $\sigma = 0$, sind die Microfacetten *plan* zur Oberfläche und die Formel vereinfacht sich zur Lambert'schen Beleuchtungsformel.

$$
\begin{aligned}
I_{\mathrm{ON}} &= \cos\theta_l \cdot (A + (B \cdot \max[0, \cos(\phi_l - \phi_c)] \cdot \sin\alpha \cdot \tan\beta)) \\
&\text{mit} \\
A &= 1 - 0.5\frac{\sigma^2}{\sigma^2 + 0.33}, \\
B &= 0.45\frac{\sigma^2}{\sigma^2 + 0.09}, \\
\alpha &= \max(\theta_l, \theta_c), \\
\beta &= \min(\theta_l, \theta_c)
\end{aligned}
$$

$$
\begin{aligned}
&\text{für } \sigma = 0: \quad A = 1,\ B = 0 \\
I_{\mathrm{ON}} &= \cos\theta_l \cdot (1 + (0 \cdot \max[0, \cos(\phi_l - \phi_c)] \cdot \sin\alpha \cdot \tan\beta)) \\
&= \cos\theta_l \cdot (1 + 0) \\
&= \cos\theta_l
\end{aligned}
$$

In Abbildung 3.5 werden beide Formeln auf eine Tonvase angewendet und mit einem Foto verglichen. Es ist gut zu erkennen, dass die Lambert-Beleuchtung im Vergleich zur Seite hin zu schnell dunkel wird. Am Rand ist die Beleuchtungsintensität nahezu 0, da der Vektor zum Licht und die Oberflächennormale einen 90-Grad Winkel bilden. Dieser Fall ist für ein glattes Material realistisch, jedoch nicht für die hier zum Vergleich benutzte Tonvase. Der raue Ton hat feine Mikrostrukturen, die dafür sorgen, dass trotz der seitlichen 90-Grad Beleuchtung noch Licht zur Kamera zurückgeworfen wird. Dieser Fall kann mit der Oren-Nayar Beleuchtungsformel modelliert werden, wie im vergleichenden Bild zu erkennen ist.

Lambert Foto einer Tonvase Oren-Nayar

Abbildung 3.5: Lambert'sche (links) und Oren-Nayar Beleuchtungsformel (rechts) im Vergleich zu original Foto einer Tonvase (Mitte), aus [ON94]

3.3.3 Ambient-Occlusion im Bildraum

Unter dem Begriff *Ambient-Occlusion*[10] versteht man eine zusätzliche Schattierung, die durch indirekte Beleuchtung erzeugt wird, wie in Abbildung 3.6 dargestellt. Diese Schattierung kann beispielsweise an Objekten beobachtet werden, die teilweise verdeckt sind und auf die daher nur wenig direktes Licht einfällt. Der visuelle Effekt erhöht die Plastizität des Objektes, es wirkt dreidimensionaler und weniger flach. Da der Effekt in der Natur vorkommt, wirken Bilder mit *Ambient-Occlusion* natürlicher. Eine grobe Approximation dieses Phänomens liefern Verfahren im Bildraum, die die Abdunklung lokal berechnen und die sich als Bildfilter implementieren lassen. Als Eingabe benötigt das Verfahren im einfachsten Fall Tiefenwerte, im komplexeren zusätzlich noch die Oberflächennormalen. Es ist daher sehr gut geeignet, in einem G-Buffer-Renderer eingesetzt zu werden.

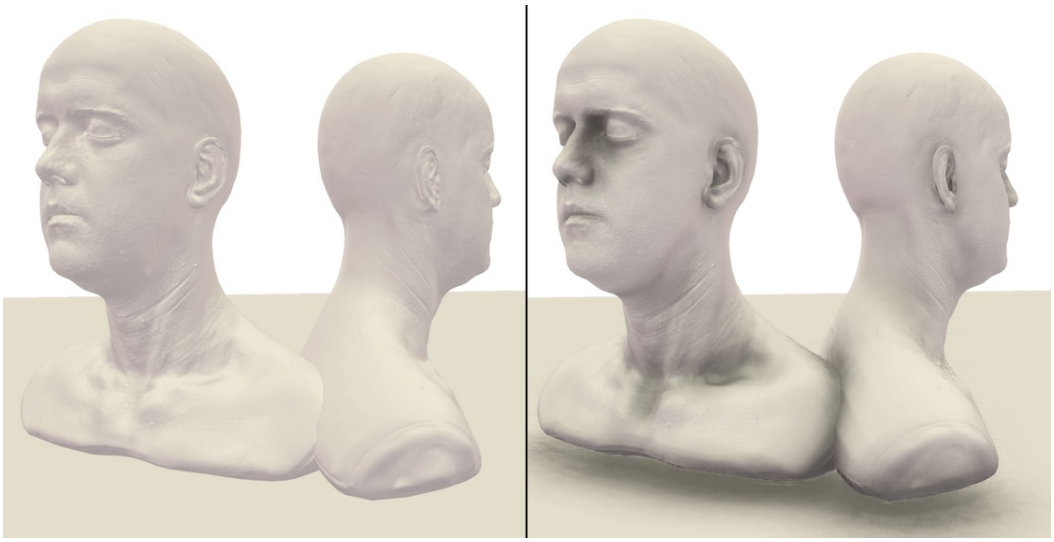


Abbildung 3.6: Szene ohne SAO (links) und mit SAO (rechts), aus [MML12]

Image Enhancement

Die Arbeit *Image enhancement by unsharp masking the depth buffer* [LCD06] ist im engeren Sinn kein *Ambient-Occlusion*-Verfahren, der erreichte Effekt ähnelt diesem aber stark. Unterschiede im Tiefenbild führen zu einer Abdunklung der Szene, zu sehen in Abbildung 3.7. Entfernt man den aufhellenden Term, erhält man genau den beschriebenen Effekt. Das Tiefenbild wird dazu zunächst geglättet und dann vom ungeglätteten abgezogen: Ist das Resultat negativ, ist das Pixel abgedunkelt. Das Ergebnis überzeugt, tritt allerdings nicht an allen Stellen in der notwendigen Intensität auf.

[10] deutsch: Umgebungsverdeckung

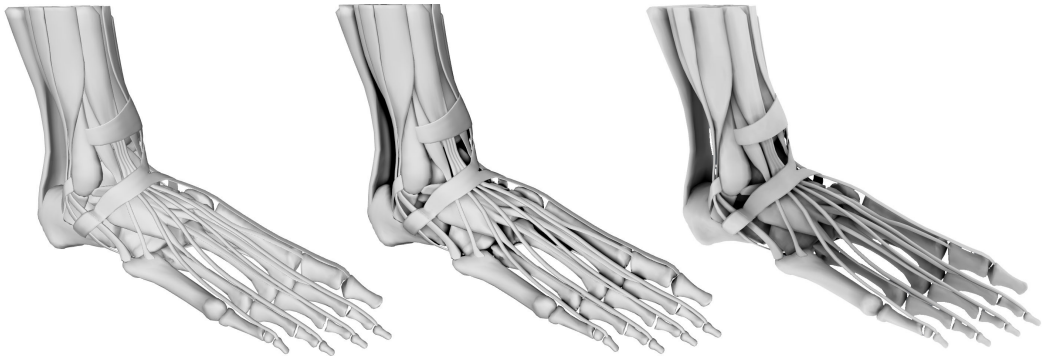


Abbildung 3.7: Szene ohne AO (links), mit *Image Enhancement* (Mitte) und *echten* Ambient-Occlusion (rechts), aus [LCD06]

Horizon-based Ambient Occlusion

Ein Verfahren, das auch die Normalen im Bild berücksichtigt, wurde unter dem Titel *Image-space horizon-based ambient occlusion* (HBAO) von Bavoil, Sainz und Dimitrov [BSD08] vorgestellt (vergleiche Abbildung 3.8). Aus der Umgebung eines Pixels werden *zufällig* Pixel abgetastet. Durch die Position, die aus der Tiefe rekonstruiert wird, und der Normalen kann bestimmt werden, wie viel Licht von den Samples zum betrachteten Pixel geworfen wird. Das Ergebnis wirkt überzeugender als das zuvor vorgestellte Verfahren. Das Bild wirkt allerdings durch das Sampling *körnig* und muss noch geglättet werden.

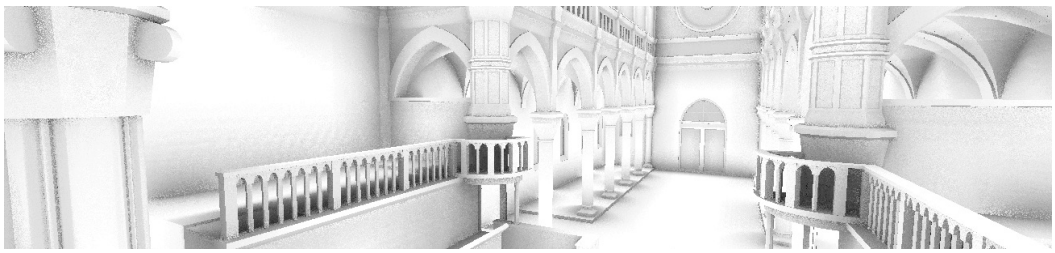


Abbildung 3.8: Horizon-based Ambient Occlusion, aus [BSD08]

Scalable Ambient Obscurance

Ein aktuelles Verfahren mit dem Titel *Scalable Ambient Obscurance* (SAO) [MML12], zu sehen in Abbildung 3.9, benötigt nur das Tiefenbild als Eingabe. Es arbeitet ähnlich wie HBAO, mit dem Unterschied, dass die Normale aus dem Tiefenbild herausgerechnet wird. Durch verschiedene Verbesserungen, unter anderem am Abtastgitter der Samples, wurden Qualität und Geschwindigkeit weiter gesteigert. Das Ergebnis mit integrierter Glättung wirkt sehr natürlich.

Abbildung 3.9: Scalable Ambient Obscurance, aus [MML12]

3.3.4 Antialiasing

Ein visuell auffälliges Problem des Renderings und digitaler Bilder ist das *Aliasing*[11]. Traditionelle Antialiasing-Verfahren sind das *Supersampling Antialiasing* (SSAA) und das *Multi/-sampling Antialiasing* (MSAA), die für die klassische Renderpipeline konzipiert sind. Neuere Verfahren, wie das *Fast Approximate Antialiasing* (FXAA), das *Morphological Antialiasing* (MLAA) oder das *Enhanced Subpixel Morphological Antialiasing* (SMAA) arbeiten rein im Bildraum als Filter und können daher gut mit einem G-Buffer-Renderer verwendet werden.

SSAA & MSAA

Beim SSAA wird die Auflösung erhöht, so dass ein Pixel aus mehreren Subpixeln besteht. Der Farbwert des Pixels ergibt sich aus einem speziellen Abtastmuster mit Gewichtungsfunktion. Somit haben auch Objekte, die kleiner als ein Pixel sind, einen Einfluss auf das Pixel. Der Nachteil ist jedoch, dass SSAA nach dem *brute-force*[12]-Prinzip operiert. Je nach Anzahl der Subpixel steigt der Renderaufwand deutlich an. Das verwendete Abtastmuster hat zudem einen deutlichen Einfluss auf den Eindruck des finalen Bildes.

Das MSAA ist ein Spezialfall des SSAA. Beim SSAA muss für jedes Pixel das Shading durchgeführt werden. Beim MSAA wird nur die Geometrie in der hohen Auflösung gerendert. Vor dem Shading werden die Subpixel nach dem gleichen Prinzip abgetastet, jedoch ist das Ergebnis nur eine Eingabe für die Shadingphase. Die Renderzeit verkürzt sich daher, es werden allerdings nicht alle Artefakte behoben.

[11] Auf eine ausführliche Diskussion wird an dieser Stelle verzichtet, siehe dazu [Shi05] oder [AHH08]
[12] deutsch: brachiale Gewalt

FXAA

Ein rein im Bildraum arbeitendes Verfahren ist das FXAA von Lottes [Jim11a], zu sehen in Abbildung 3.10. Es ist kein direktes Antialiasing-Verfahren, sondern ein dedizierter Bildfilter, dessen Idee vom Autor mit *Local Contrast Adaptive Directional Edge Blur*[13] beschrieben wird. Das Verfahren erzeugt gute Ergebnisse, arbeitet sehr schnell und kann leicht in bestehende Systeme integriert werden.

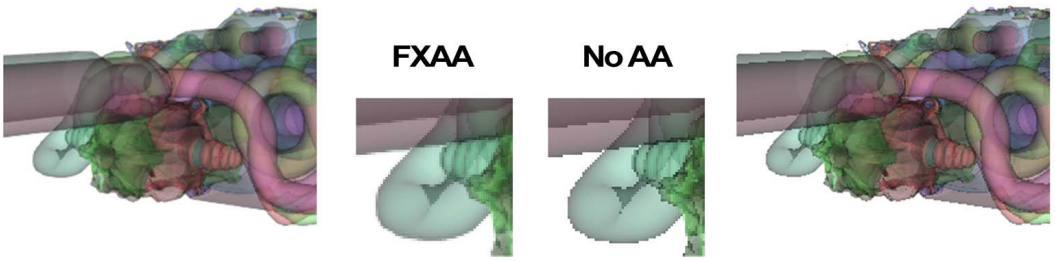


Abbildung 3.10: Antialiasing durch den FXAA-Filter, aus [Jim11a]

MLAA & SMAA

Das Verfahren MLAA [Jim11b] wurde als SMAA [Jim12] mit besserer Qualität weiterentwickelt. Als Kernidee gilt das Auffinden von charakteristischen Kantenmustern, für die eine Lösung hergeleitet werden kann, gezeigt in Abbildung 3.11.

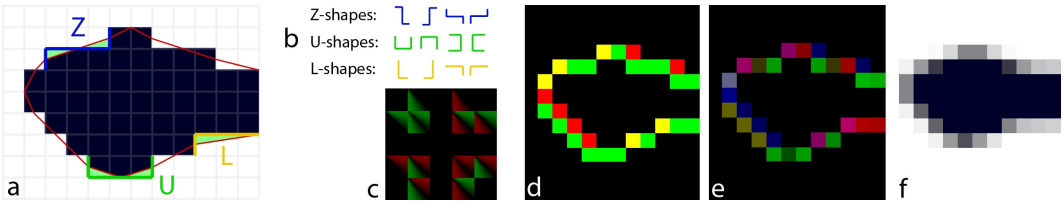


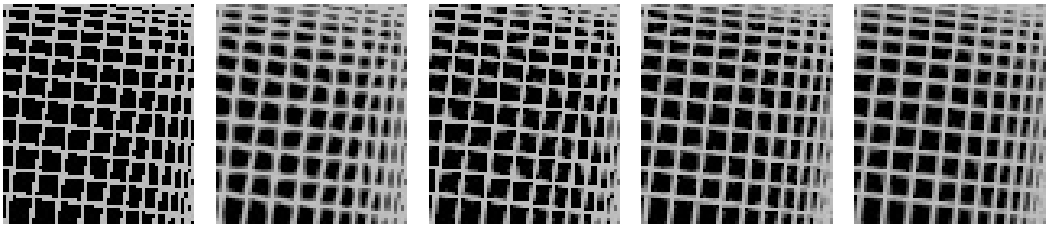
Abbildung 3.11: Rekonstruktion der Kantenverläufe, aus [Jim12]

Je nach Verlauf einer Kante kann diese mit dem Nachbarn vermischt werden und so ein visuell überzeugendes Ergebnis erreichen. Das rechenintensive Verfahren wurde im SMAA beschleunigt und um die Fähigkeit, Subpixel zu verarbeiten, erweitert. Die Ergebnisse sind sehr überzeugend 3.12.

3.3.5 Zusammenfassung

Die in diesem Abschnitt vorgestellten Rendertechniken betreffen neben Kernaspekten des Renderings auch das Ausbessern von systematischen Fehlern der Renderpipeline,

[13] deutsch: lokaler Kontrast und adaptive gerichtete Kantenglättung



Ohne AA MLAA SMAA 1x SMAA 2x SMAA 4x

Abbildung 3.12: Vergleich von Eingabebild, MLAA-Filter und verschiedene SMAA-Filterintensitäten, aus [Jim12]

dem Aliasing. Mit Hilfe der G-Buffer ist es möglich, komplexe 3D-Modelle einmal zu zeichnen und verschiedene Algorithmen im Bildraum anzuwenden. Dies setzt voraus, dass die benötigten Eingabedaten, wie etwa Oberflächennormale, Position oder Texturkoordinaten, in Texturen geschrieben werden.

Für die Beleuchtung wurden zwei Verfahren vorgestellt, die realistische Ergebnisse für diffus-reflektierende und raue Oberflächen berechnen. Insbesondere das Verfahren von *Oren-Nayar*, das eine allgemeinere Form der *Lambert'schen* Beleuchtungsformel darstellt, liefert gute Ergebnisse.

Daran anschließend wurden Methoden zur Approximation von *Ambient Occlusion* präsentiert. Die Verfahren arbeiten im Bildraum und können den dreidimensionalen Eindruck einer Szene deutlich steigern.

Die letzte vorgestellte Verfahrensgruppe mindert das *Aliasing*-Problem durch einen Bildfilter. SMAA, das fortschrittlichste Verfahren, versucht dabei mit Hilfe von charakteristischen Kantenmustern eine subpixelgenaue Abtastung zu liefern.

3.4 NPR-Algorithmen

NPR-Algorithmen werden seit mehreren Dekaden in der Computergrafik untersucht [Sal94]. Im Folgenden werden verschiedene Verfahren kurz vorgestellt, die ausgewählt wurden, weil sie Bausteine für die Anfertigung einer wissenschaftlicher Illustration bilden, wie sie in Abschnitt 2.1.2 vorgestellt wurden. Sie imitieren entweder die benötigten Techniken oder tragen auf andere Art zum Ziel bei, wissenschaftlichen Illustration interaktiv am Computer zu erstellen. Die Auswahl wird dabei beschränkt auf Verfahren, die entweder bereits interaktiv laufen oder mit Hilfe moderner Verfahren (beispielsweise OpenCL) entsprechend beschleunigt werden können.

Betrachtet werden neben Verfahren zum Pünkteln auch Beleuchtungsmethoden, die speziell für den Einsatz im non-photorealistischen Rendern entworfen wurden. Zunächst werden Kontur- und Kantenlinien genauer definiert und Möglichkeiten, sie interaktiv algorithmisch zu erzeugen, besprochen.

3.4.1 Kontur- und Kantenlinien

Kontur- und Kantenlinien sind, wie in Abschnitt 2.1.2 gezeigt wurde, fundamental für wissenschaftliche Illustrationen. Sie heben die Form eines Objektes besser hervor und helfen dabei, dessen Struktur durch Abtrennung von überlappenden Teilen zu erkennen. Zusätzlich werden sie eingesetzt um charakteristische Merkmale hervorzuheben, wie beispielsweise harte Kanten eines Objektes.

Kantendetektion

Die einfachste Methode, Konturen und Kanten in einer 3D-Szene zu detektieren, ist in [ST90] beschrieben. Das Tiefenbild der Kamera wird nach Diskontinuitäten sowohl in der ersten als auch in der zweiten Ableitung gefiltert. Die Filterung entspricht einer Faltung mit einem 3×3 Sobel-Filterkern. Beide Ergebnisse werden miteinander kombiniert, um alle Konturen zu erkennen. Da nach [GG01] die Diskontinuitäten der zweiten Ableitung des Tiefenbildes denen der ersten Ableitung des Normalenbildes entsprechen, kann sowohl das Tiefenbild als auch das Normalenbild mit einem Sobel-Filter gefiltert werden. Dieser berechnet die Diskontinuitäten der ersten Ableitung der Helligkeitswerte. Das Ergebnis, das in Abbildung 3.13 gezeigt wird, ist ein relativer Wert, der mit einem objekt- und szenenabhängigen Schwellenwert kombiniert wird, um die gewünschten Ergebnisse zu erreichen. Die so detektierten Kanten können mit dem Bild überlagert werden, so dass Kontur und Kanten markiert sind.

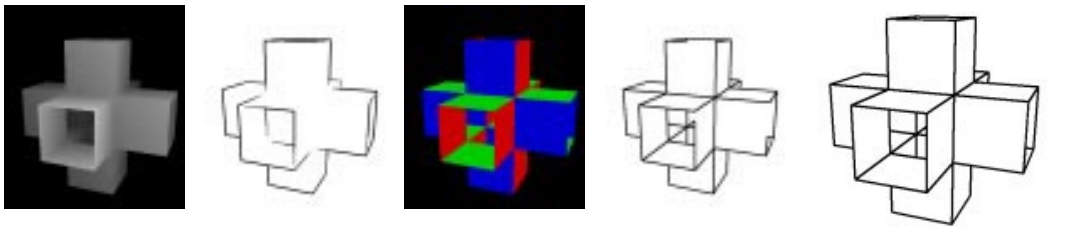


Abbildung 3.13: Konturen durch Kantendetektion, aus [GG01]

Durch unterschiedliche Filterkerne sind unterschiedliche Kantentypen hervorhebbar, wie dies beispielsweise in [Jäh05] beschrieben ist. Zu den verbreitetsten Operatoren zählen der Sobel-, Laplace-, Canny- und Prewitt-Operator.

Image Enhancement

Eine Alternative zur Silhouetten- und Kantenbestimmung durch Kantendetektion ist der Einsatz des schon weiter oben beschriebenen Verfahrens *Image enhancement by un-*

sharp masking the depth buffer [LCD06][14].

Suggestive Konturen

Der Begriff *Suggestive Konturen* bezeichnet Konturlinien, die entlang von Krümmungen und Konturlinien verlaufen und so dem Betrachter einen Hinweis auf die geometrische Struktur geben, wie in Abbildung 3.14 veranschaulicht. Sie sind von grundlegender Bedeutung bei der Erstellung expressiver Linienzeichnungen [Col12]. Sie stellen auch ein wichtiges Mittel dar, um zusätzlich zur Kontur noch weitere Abgrenzungen und Konturinformationen zu liefern.

Konturlinien durch Kantenfilter

Suggestive Konturlinien

Abbildung 3.14: Konturlinien (links) und Suggestive Contures (rechts), aus [Rus08]

3.4.2 Pünktelung

Die Pünktelungstechnik ist eine wichtige Zeichentechnik und daher auch intensiv im Rahmen der NPR-Techniken untersucht. Es gibt verschiedene Ansätze, aus einem Graustufenbild eine Pünktelungszeichnung zu generieren. Kritisch ist bei den Verfahren die Verteilung der Punkte. Bei reinen bildbasierten Methoden ist es dabei schwer, die Punkte *entlang* von Konturen zu verteilen, da diese erst aus zusätzlichen Informationen heraus rekonstruiert werden müssen, wie dies im Artikel *Feature-guided Image Stippling* [Kim08] beschrieben ist.

Ein naiver Algorithmus, unter anderem beschrieben in [SS02], verteilt durch eine Zufallsfunktion die Punkte auf einer *Pünktelungstextur*, speichert den Rang und nutzt ihn als Kriterium für die dargestellte Helligkeit. Je heller der Bildbereich, desto weniger Punkte werden angezeigt: Bei einer Helligkeit von beispielsweise 70% werden nur die ersten 30% der Punkte angezeigt. Durch einfaches Wiederholen der Kacheln entstehen so sichtbare Muster. Eine solche vorberechnete Pünktelungstextur kann auch für Volumen verwendet werden, wie beispielsweise in [Bae07] und [Tie08] beschrieben.

[14] OpenGL-Forum: 'SSAO IS a glorified outlines shader', besucht: 22.06.2015
www.opengl.org/discussion_boards/showthread.php/165133

Andere Verfahren versuchen, die Punkte zielgerichtet auf Basis der Helligkeit und anderen Kriterien zu verteilen. Dies kann wie in [Sec02] oder in [DI13] veröffentlicht auf Basis von Voronoi-Zellen erfolgen. In Abbildung 3.15 ist ein Ergebnis des zuletzt genannten Verfahrens gezeigt. Eine andere Möglichkeit gewichtet die Helligkeitsverteilung der Umgebung in die Punkteverteilung um die Struktur zu erhalten [LM11]. Zuletzt gibt es auch ein Verfahren, das die Punkteverteilung eines vorgegebenen Bildes analysiert und so einen persönlichen Zeichenstil reproduziert [Mar10].

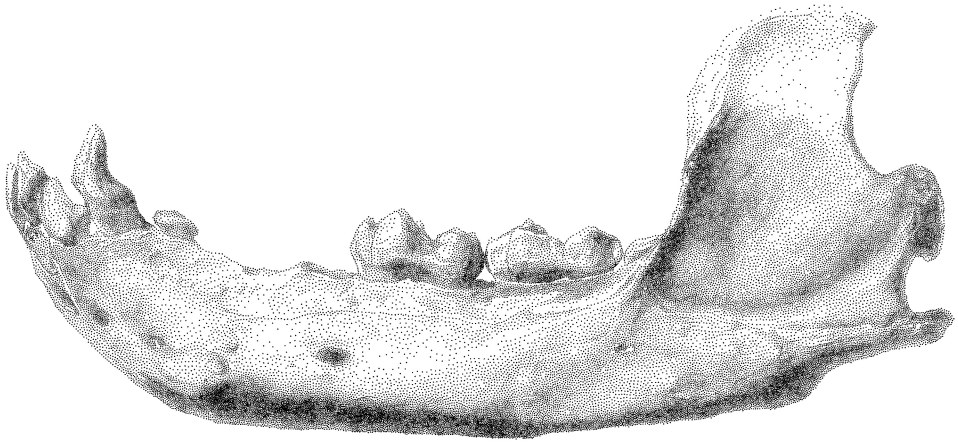


Abbildung 3.15: Pünktelung mit Voronoi-Tesselation, aus [DI13]

Recursive Wang Tiles

Eine Arbeit mit dem Titel *Recursive Wang tiles for real-time blue noise* [Kop06] kombiniert verschiedene Ansätze. Das Ergebnis ist in Abbildung 3.16 zu sehen. Zunächst einmal werden auf Kacheln die Punkte mit der Charakteristik eines blauen Rauschens erzeugt. Der Einsatz erfolgt nach dem oben beschriebenen naiven Vorgehen. Die Kacheln werden allerdings nach dem Prinzip der *Wang-Tiles* auf eine Art miteinander kombiniert, mit der sie eine unendliche Fläche ausfüllen, ohne ein erkennbares Muster zu erzeugen. Als letzter Baustein sind die Kacheln auch rekursiv unterteilbar, so dass sie nach Bedarf vergrößert werden können und die Punkte ihre relative Position im Bild beibehalten.

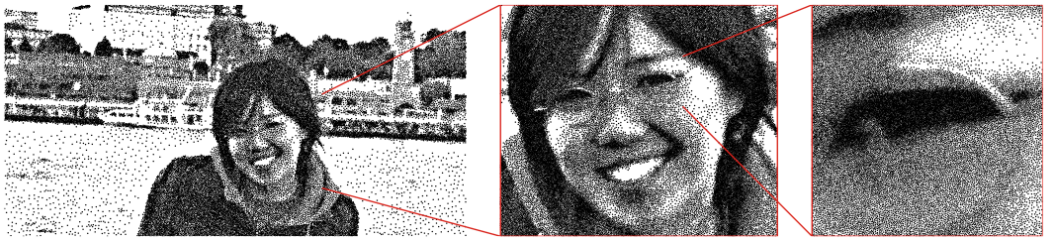


Abbildung 3.16: Pünktelung mit *Recursive Wang Tiles*-Verfahren in 3 Stufen, aus [Kop06]

Electrostatic Halftoning

Beim *Electrostatic Halftoning* [Sch10] werden die Punkte als elektrostatische Partikel aufgefasst und von einem Partikelsystem verteilt, das auf der GPU umgesetzt ist, zu sehen in Abbildung 3.17. Da ein Partikel mit allen anderen interagiert, hat das zu Grunde liegende System eine quadratische Laufzeit. Jedes Pixel des Eingabebildes wird als Ladungsträger aufgefasst, dessen Ladung von seiner Helligkeit abhängt: Je dunkler die Farbe des Pixels ist, desto stärker fällt seine Ladung aus. Die Ladung der Pixel ist umgekehrt zu der der Partikel, so dass die Partikel von den Pixeln angezogen werden: Um so dunkler das Pixel ist, desto größer die Anziehung. Die Summe der Ladung aller Partikel und Helligkeiten muss gleichgroß sein, so dass sich ein Gleichgewicht einstellen kann. In einem iterativen Prozess werden die Partikel verteilt: Sie stoßen sich untereinander ab und werden von den Pixeln angezogen. Dabei entsteht eine regelmäßige Verteilung der Partikel. Wenn eine unregelmäßige Verteilung der Punkte gewünscht ist, können die Partikel mit einem zusätzlichem *Jitter*-Schritt zufällig verschoben werden. Das Verfahren wurde mit einer Approximationsfunktion zur Berechnung der Kräfte, die auf ein Partikel wirken, beschleunigt [Gwo].

Partikel Gitter Originalbild Ergebnis

Abbildung 3.17: Model und Ergebnis des *Electrostatic Halftonings* aus [Sch10]

In [SS02] wird auch eine interaktive Methode vorgestellt, die mehrere Pinsel in einem Malprogramm beschreibt, die zur Erstellung von Pünktelungszeichnungen genutzt werden können. Der Fokus liegt hier auf dem Einsatz für ein *eigenständiges* Malprogramm und nicht unmittelbar für einen Filter zu existierenden Bildern.

Die Verfahren sind alle für Einzelbilder konzipiert, bei einem bewegten Bild erzeugen sie sichtbare Artefakte, bekannt als der *shower-door effect*[15] [Mei96]. Diese Problematik wird zum Beispiel durch Verfahren, wie das *Real-Time Animated Stippling* [PFS03], angegangen.

[15] deutsch: Duschtür-Effekt

3.4.3 Expressives Shading

Während die Beleuchtungsverfahren aus Abschnitt 3.3.2 das Ziel haben, eine realistisch beleuchtete Oberfläche darzustellen, wurden im Rahmen der NPR-Algorithmen Verfahren gesucht, die den Verlauf einer Oberfläche besonders hervorheben und sichtbar machen. Dies erfolgt neben einer angepassten Beleuchtungsformel auch durch den Einsatz von besonderen Farben oder einer Kombination aus beiden Techniken.

Two-Tone-Shading

Ein besonders im technischen Bereich etabliertes Verfahren ist das *Two-Tone-Shading*, das auf Gooch et al. [Goo98] zurückgeht und in Abbildung 3.18 zu sehen ist. Es ist ein lokales Beleuchtungsmodell, das für den diffusen Lichtanteil zwei besondere Farben nutzt: eine warme (rötliche) und eine kalte (bläuliche) Farbe. Dadurch, dass statt Schwarz eine hellere Farbe eingesetzt wird, die dennoch gut mit hellen Farben kontrastiert, werden die Abstufungen vom Auge besser wahrgenommen.

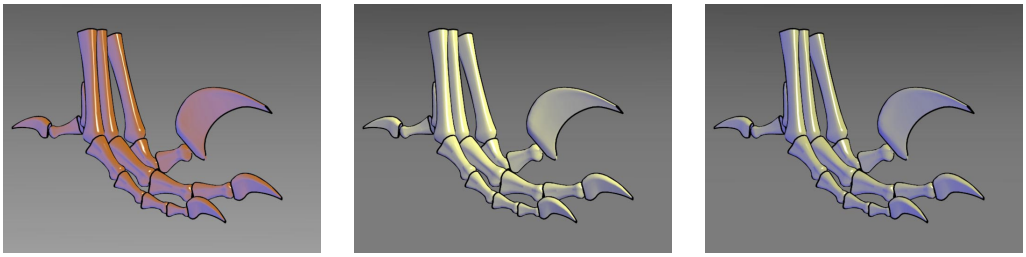


Abbildung 3.18: Verschiedene Farbvarianten des *Two-Tone-Shadings*, aus [Goo98]

Radiance Scaling

Das *Shading with Apparent Relief* -Verfahren [Ver08] und dessen Vorgänger *Radiance Scaling for Versatile Surface Enhancement* [Ver10] berücksichtigen die Form und die Krümmung einer Oberfläche in der Beleuchtungsformel. Dahinter steht die Beobachtung, dass konvexe Objektteile Licht stärker reflektieren und daher heller erscheinen als konkave. Das Verfahren kann nicht nur für diffus-reflektierende Materialien, sondern auch für spiegelnd-diffuse Materialien benutzt werden, wie in Abbildung 3.19 gezeigt.

Exaggerated Shading

Das *Exaggerated shading for depicting shape and detail* [RBD06] genannte Verfahren sorgt für einen deutlich höheren Kontrast der Beleuchtung, wie in Abbildung 3.20 gezeigt ist. In einem Vorverarbeitungsschritt wird die Normale mehrfach hintereinander geglättet. Die Beleuchtung wird durch drei Techniken angepasst. Im einzelnen sind dies:

Abbildung 3.19: Drei Modelle: linke Häfte ohne und rechte mit Radiance Scaling, aus [Ver10]

1. Verzerrung der Lambert'schen Beleuchtungsformel mit einem *soft-toon-shader* mit Parameter τ, der den Kontrast der Beleuchtungsintensität erhöht und die Werte entweder zu reinem Weiss oder reinem Schwarz verschiebt:

$$I_{\text{softToon}} = 0.5 + 0.5 \cdot \min(1, \max(-1, \tau \cdot \vec{N} \cdot \vec{L}))$$

2. Kombination mehrerer Beleuchtungsvorgänge mit unterschiedlich stark geglätteten Normalen.

3. Beleuchtung mit zusätzlichen Lichtquellen, wobei die Position der Lichtquellen vom Vertex und der geglätteten Normalen abhängt.

Abbildung 3.20: Exaggerated Shading (links) und Lambert'sche Beleuchtung (rechts), aus [RBD06]

3.4.4 Universelle Zeichenverfahren

In der Arbeit *RenderBots -- Multi Agent Systems for Direct Image Generation* [SGS05] wird ein Verfahren auf Basis autonomer Agenten vorgestellt, mit dem sich verschiedene Zeichenstile imitieren lassen. Die Agenten agieren dabei nach ähnlichen Prinzipien wie die in Abschnitt 3.4.2 vorgestellten *elektrostatische Partikel*. Mit den Agenten können allerdings auch Schraffuren und anderen Techniken realisiert werden. Jeder Agent agiert auf Basis eines oder mehrerer Eingabebilder und in Relation zu den anderen Agenten seines Schwarms. Nach einer Verteilungsphase führen die Agenten einen Malschritt durch und erzeugen so ein Bild. Im Unterschied zu den Partikeln wird nicht nur die Position, sondern auch die Ausrichtung erfasst und berücksichtigt. So sind auch orientierungsabhängige Schraffuren möglich. Dieses sehr komplex strukturierte System ist bisher nicht auf der GPU umgesetzt worden.

Einen ähnlichen Ansatz verfolgt die Arbeit *Interactive Artistic Rendering* [KGC00] von Kaplan, Gooch und Cohen, deren Verfahren auch eng verwandt mit dem in Abschnitt 3.4.2 vorgestellten Verfahren *Electrostatic Halftoning* [Sch10] ist. Grundlage bildet hier ein Partikelsystem, das komplett im Bildbereich arbeitet. Jeder Partikel ist Repräsentant für eine Markierung oder eine Zeichenoperation (Schraffurstrich, Punkte oder andere Objekte). Kernidee ist, dass das Zeichnen auf einem Blatt Papier imitiert werden soll, bei dem mit einem Stift Markierungen auf einer zweidimensionalen Ebene, dem Papier, gesetzt werden. Das Verfahren ist interaktiv, da der Nutzer sowohl die Art der Markierung als auch indirekt durch Ändern der Beleuchtung die Platzierung der Partikel anpassen kann. Das Partikelsystem wird auf der CPU ausgewertet.

3.5 Evaluation von NPR-Verfahren

Obwohl NPR-Verfahren schon seit mehreren Jahrzehnten erforscht werden, findet eine wissenschaftliche Auseinandersetzung mit den Ergebnissen erst seit relativ kurzer Zeit statt. Im Buchbeitrag *Evaluating and Validating Non-Photorealistic and Illustrative Rendering* [Ise13] werden diese vorgestellt, zusammengefasst und beschrieben.

In der Arbeit mit dem Titel *Non-photorealistic rendering in context: an observational study* [Ise06] werden computer-generierte mit handgezeichneten Illustrationen verglichen. In der Arbeit werden Handzeichnungen von je einem Objekt der Archäologie, der Medizin und der Botanik gezeigt, zu denen ein 3D-Modell existiert. Dazu fertigte man in einer zweiten Stufe verschiedene computer-generierte Bilder an. Die Ergebnisse legte man mehreren Experten aus den eben genannten Fachgebieten zur Bewertung vor. Innerhalb der Studie sollten die Teilnehmer die Bilder in verschiedene Stapel gruppieren und diese Gruppen dann begründen. Die so entstandenen Charakterisierungen waren domänenübergreifend und betreffen hauptsächlich den Detailreichtum der Illustrationen.

Als ein interessantes Ergebnis lässt sich festhalten, dass die Experten die computergenerierten Bilder wie eine Handzeichnung bewerteten. Sie wurden wegen bestimmter Merkmale geschätzt und nicht abgelehnt, nur weil sie erkennbar nicht von Hand gezeichnet wurden. Dieses Ergebnis legt nahe, dass ein Experte computer-generierte Ergebnisse akzeptiert, sofern sie *expressiv* sind und das ausdrücken können, was wichtig ist.

Nach Isenberg [Ise13] ist der reine Vergleich von computer-generierten NPR Bildern mit handgezeichneten prinzipiell nicht einfach. Ein direkter Vergleich, also die Frage, welches Bild *besser* ist, sei zu unspezifisch für einen Erkenntnisgewinn. Die Frage, ob man erkennen kann, ob ein Bild handgezeichnet oder computergeneriert ist - der NPR-Turing Test [Sal02] - ist im Kontext von Illustrationen auch nur von sekundärem Interesse. Speziell in diesem Fall ist es entscheidend, wie die Illustration wirkt, losgelöst vom Vergleich mit einer anderen. Die Frage, die beantwortet werden muss, wenn es sich um eine wissenschaftliche Illustration handelt, lautet: *Stellt die Illustration das dar, was der Urheber ausdrücken möchte?*

3.6 Zusammenfassung

In diesem Kapitel wurde der aktuelle Stand der Technik vorgestellt, soweit er relevant für den Kontext dieser Arbeit ist.

Im ersten Abschnitt wurden die aktuelle Methodik des wissenschaftlichen Illustrierens betrachtet und verschiedene Programme und Verfahren besprochen, die zur Visualisierung eingesetzt werden. Keine der Methoden ist jedoch in der Lage, die interaktiven Erstellung einer wissenschaftlichen Illustration mit der geeigneten Abstraktionsfähigkeit zu kombinieren.

Verfahren, mit denen sich 3D-Modelle erfassen lassen, wurden im zweiten Abschnitt diskutiert. Besonderes Augenmerk liegt auf Algorithmen, die die Oberfläche aus einem Bilderstapel rekonstruieren, da diese Rekonstruktionsmöglichkeit im Folgenden näher betrachtet und im Rahmen der Arbeit verbessert wird. Bisherige Arbeiten liefern nur unzureichende Ergebnisse.

Der dritte Abschnitt hat sich allgemein mit Rendering-Verfahren befasst. Neben der Einführung von G-Buffern wurden mehrere Varianten zur (lokalen) Beleuchtungsrechnung vorgestellt. Im Anschluss sind Techniken erörtert worden, die eine globale Beleuchtung im Bildraum approximieren und sich mit G-Buffern effizient einsetzen lassen. Den Abschluss bildet eine Vorstellung von Methoden zum Antialiasing im Bildraum.

Im vorletzten Abschnitt sind NPR-Algorithmen für ausgewählte Zeichentechniken besprochen worden. Als Kriterium galten hier die in Abschnitt 2.1.2 vorgestellten für wissenschaftliche Illustrationen relevanten Techniken. Neben Verfahren, die offline, al-

so auf Grund der hohen Berechnungsdauer nicht-interaktiv ablaufen, wurden auch real-time fähige NPR-Algorithmen vorgestellt. Zuletzt wurden zwei Techniken präsentiert, die verschiedene Zeichentechniken imitieren können. Im ersten Fall wird ein Partikelsystem modelliert und im anderen Fall werden autonome Agenten eingesetzt.

Zuletzt wurden relevante Arbeiten zur Evaluation von NPR-Verfahren besprochen. Als Ergebnis lässt sich hier festhalten, dass es speziell bei NPR-Verfahren für wissenschaftliche Illustrationen meistens nicht darum geht, ein Bild zu generieren, von dem ein Betrachter nicht erkennen kann, ob es ein Computer oder ein Mensch erstellt hat. Wichtiger ist die Frage, ob ein am Computer algorithmisch erstelltes Bild das transportieren und ausdrücken kann, was der Ersteller aussagen wollte.

Kapitel 4

Anforderungsanalyse

In diesem Kapitel wird die Anforderung an das erarbeitete Konzept analysiert und ein Lastenheft erstellt. Neben Anwendern und Einsatzbereich werden die Eingabe und benötigte Ausgabe diskutiert. Der Fokus liegt hier auf der Erstellung von Illustrationen, wie sie typischerweise in der Paläontologie eingesetzt werden. Dabei soll das erarbeitete Framework so tragfähig sein, dass es in anderen Domänen eingesetzt werden kann.

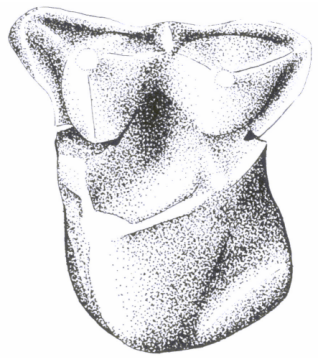
4.1 Ziel und Anwender

Um neue Fundstücke oder neue Erkenntnisse im wissenschaftlichen Bereich der Paläontologie zu publizieren, werden von einem Fundstück die in Abbildung 4.1 gezeigten drei charakteristischen Bilder benötigt:

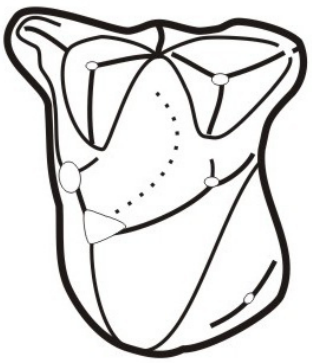
(a) eine wissenschaftliche Illustration,

(b) eine schematische Zeichnung und

(c) ein Bild totaler Schärfe.



Wissenschaftliche Illustration, A. Helfricht



Schematische Zeichnung, Dr. Thomas Lehmann

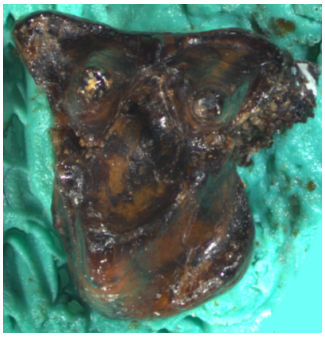


Bild totaler Schärfe, Sebastian Schäfer

Abbildung 4.1: Drei Abbildungen des *Leptictidium*-Zahns

Es werden nicht immer alle Bilder benötigt, und bei manchen Fundstücken sind auch mehrere Bilder des gleichen Typs aus unterschiedlichen Ansichten wichtig, um bestimmte Details besser hervorzuheben.

Domänenexperten und Ansprechpartner sind Mitarbeiter der paläontologischen Abteilung des Naturmuseums und Forschungsinstitutes Senckenberg Frankfurt/Main: Dr. Thomas Lehmann und Dr. Krister Smith, der auch illustriert, sowie Juliane Eberhardt, eine ausgebildete technische Zeichnerin. Das Konzept und der Prototyp soll dabei möglichst optimal in den Arbeitsfluss eingepasst werden.

In meinen Gesprächen mit den Entwicklungspartnern und ihren Kollegen ist vor allem ein Grund deutlich geworden, warum ein Bedarf an einem solchen Werkzeug besteht. Ein Forscher hätte durch dieses Verfahren die Möglichkeit, verschiedene Perspektiven, Beleuchtungen, Zeichentechniken und hervorzuhebende Merkmale auszuprobieren, bevor er einen Zeichnen-Auftrag vergibt. Den Gesprächen zufolge ist hier jede Hilfe willkommen, die einen groben Eindruck von einer finalen Illustration vermitteln kann. So könnten Zeit und Kosten bei der Besprechung eines Auftrags eingespart werden. Von allen Fundstücken eine oder auch mehrere Illustrationen anfertigen zu lassen, ist aufwandbedingt keine Option. Der (nicht zeichnende) Forscher muss eine genauere Vorstellung von den geplanten Illustrationen haben und abwägen, welche er in Auftrag gibt. Die Gespräche verdeutlichten, dass hier vor allem Hilfe willkommen sei, die einen groben Eindruck von einer finalen Illustration vermitteln kann.

Grundlage für die Möglichkeit des Ausprobierens ist ein interaktives Verfahren, das es erlaubt, unterschiedliche Parameterwerte direkt zu vergleichen und unterschiedliche Varianten einer Illustration einfach herzuleiten. Bei einer internen Präsentation des hier erörterten Konzeptes innerhalb der Arbeitsgruppe des Naturmuseums und Forschungsinstitutes Senckenberg Frankfurt/Main wurden diese Punkte als wesentlich von den Kollegen bewertet. Anwesende Experten aus anderen Fachgebieten (Archäologie, Biologie und Zoologie) bestätigten nicht nur das Interesse an einem solchen Werkzeug, sondern auch die grundsätzliche Nähe der gezeigten Illustrationen zur eigenen Domäne, wobei stilistische Variationen in Betracht gezogen werden müssen.

Die Basis für das Verfahren ist zunächst ein 3D-Modell, das in einem ersten Schritt von einem Fundstück erstellt werden muss. Dies setzt, wie in Kapitel 3.2 beschrieben, spezielle Hardware voraus. Die Rekonstruktion soll daher Teil des Konzeptes werden. Hier muss auf die üblicherweise verfügbare Hardware eingegangen werden. Vor Ort vorhanden ist ein Leica Motorfokus-Mikroskop mit Digitalkamera *DFC 500*, mit dem ein dreidimensionales Modell der Oberfläche des Fundstücks rekonstruiert werden kann. Es besteht auch Zugriff auf ein *3D-Scannerlabor*, allerdings sollte nicht allzu oft davon Gebrauch gemacht werden. Am Campus Riedberg der Goethe-Universität verfügt der Fachbereich Physik über einen μ-CT-Scanner, auf den allerdings noch seltener zugegriffen werden kann.

In der Paläontologie-Abteilung wird neben *Microsoft Windows* auch *Apple OS X* eingesetzt. Das Verfahren sollte so konzipiert sein, dass eine Umsetzung *plattformunabhängig* möglich ist. Dies soll durch Verwendung von offenen Bibliotheken und Schnittstellen erreicht werden.

4.2 Eingabe

Das Programm soll die wichtigsten der in dem Bereich etablierten Modellformate verarbeiten können. Auf Basis der Hardware kann folgende Liste erstellt werden, die von Mitarbeitern des Instituts bestätigt wurde:

1. Polygonale Modelle, entweder direkt eingescannt oder Extraktion der Isofläche eines Volumens,

2. Bilderstapel eines Motorfokus-Mikroskops oder

3. Höhenbilder, die aus Bilderstapeln eines Motorfokus-Mikroskops rekonstruiert wurden.

Da das Motorfokus-Mikroskop vor Ort vorhanden ist, bietet es sich besonders als Datenquelle an. Gerade beim Vergleich von existierenden Bildern mit erstellten Bildern des gleichen Fundstücks ist ein 3D-Modell notwendig, das nur mit dieser Hardware erstellt werden kann. Leider ist die Oberflächenrekonstruktion eines Höhenbildes aus einem Bilderstapel heraus nicht immer zufriedenstellend und zeitaufwändig. Hier gilt es nach alternativen Verfahren oder Verbesserungen der bestehenden zu suchen.

4.3 Ausgabe

Je nach Einsatz des Programms müssen unterschiedliche Formate exportiert werden können.

4.3.1 Rekonstruktion

Wird mit dem Verfahren die Modelloberfläche durch ein Höhenbild rekonstruiert, soll diese zusammen mit dem Höhenbild abspeicherbar sein. Um eine Austauschbarkeit mit anderen 3D-Programmen zu gewährleisten, bietet sich hier ein etabliertes, offenes 3D-Modell-Format an.

Auf Basis des Höhenbildes kann auch das Bild totaler Schärfe erzeugt werden, das ebenfalls mit abgespeichert werden soll.

4.3.2 Illustration und Zeichnung

Die Kernfunktionalität des Verfahrens ist die Erstellung einer wissenschaftlichen Illustration und einer schematischen Zeichnung. Dafür müssen diese zunächst im Hinblick auf folgende Fragen näher analysiert werden:

- Welche Zeichentechniken werden eingesetzt?

  Im Kapitel 2.1.2 wurden verschiedene Zeichentechniken vorgestellt. Nicht alle werden immer eingesetzt und jede Domäne hat bestimmte Präferenzen und etablierte Standards. Letztendlich ist es oftmals sogar eine Frage des Geschmacks, welche Techniken in einem Fall eingesetzt werden. Hier ist das Ziel zunächst einmal eine Basisauswahl an Techniken zu bestimmen, die zuerst in den Prototypen implementiert werden. Durch eine offene Schnittstelle ist die Palette vergleichsweise leicht erweiterbar.

- Welche der verschiedenen NPR-Algorithmen einer Technik *gefällt* den Benutzern?

  Im Kapitel 3.4 werden verschiedene Techniken vorgestellt, die alle spezielle Zeichentechniken imitieren. Da die Algorithmen nicht unbedingt zum Erzeugen von Illustrationen erstellt wurden, soll mit dieser Frage geklärt werden, welche Algorithmen gefällige Ergebnisse erzielen. Von diesen werden dann einige in den Prototypen implementiert. Wichtig ist, dass nur solche Techniken integriert werden können, die innerhalb des Szenarios einer interaktiven Anwendung umsetzbar sind.

- An welchen Stellen *lügt* ein Illustrator?

  Diese provokant formulierte Frage zielt auf die Abstraktion ab, die ein Illustrator von einem konkreten Fundstück macht. Um gezielt Aspekte eines Objektes zu illustrieren, die vielleicht nicht so prägnant am konkreten Fundstück vorliegen, muss die Illustration von der Realität abweichen. Aber auch im Rahmen der *künstlerischen Freiheit* und aus *ästhetischen Gründen* kann ein Illustrator eine Zeichnung abweichend von einem konkreten Objekt ausführen. Ein Programm, mit dem ähnliche Illustrationen erstellt werden sollen, muss solche manuellen Eingriffe ermöglichen und anbieten. Es gilt hier also die Reichweite der manuellen Eingriffe auszuloten.

In mehreren Einzelgesprächen und durch Studien von exemplarischen Illustrationen wurden diese Fragen im Detail mit den Entwicklungspartnern besprochen. Die Ergebnisse wurden auf die zwei zu erstellenden Bildertypen aufgeteilt und im Folgenden erörtert. Zunächst werden Anforderungen besprochen, die beide Typen auszeichnen.

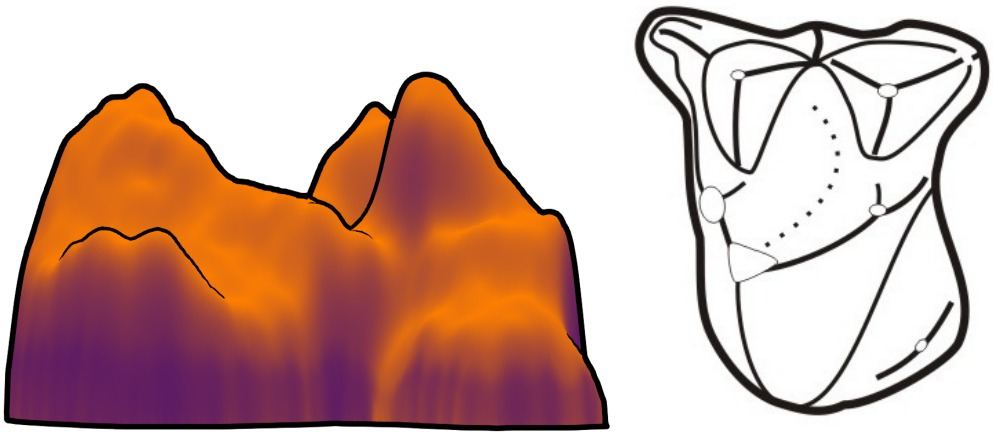


Abbildung 4.2: Rekonstruierter *Leptictidium*-Zahn in Seitenansicht mit Two-Tone-Shading (links) und schematische Zeichnung des Zahns von Dr. Thomas Lehmann (rechts)

Abstraktion der Illustration

Wie in Abschnitt 2.1 dargestellt, ist die Abstraktion und Idealisierung eines konkreten Gegenstandes Teil der Anfertigung einer wissenschaftlichen Illustration. Dies muss sich auch im Verfahren widerspiegeln: Es muss entsprechende Methoden bereitstellen. Die Abstraktion soll an mehreren Stellen erfolgen:

- Der Gegenstand selbst muss verformt werden können.
- Merkmale müssen an der Oberfläche des Gegenstandes editiert werden können.
- Die Ergebnisse der eingesetzten NPR-Verfahren und gesamte Ergebnisbild müssen leicht retuschierbar sein.

Durch diese Manipulationsmethoden wird sichergestellt, dass alle Aspekte einer Illustration nach der Vorstellung des Anwenders angepasst werden können.

Farbige Illustrationen

Das Framework muss in der Lage sein, neben Graustufenbildern auch farbige zu erzeugen. Die Farben sollen auf zwei Arten eingesetzt werden können. Zum einen im Stile der im Abschnitt 2.1.1, Abbildung 2.3 gezeigten medizinischen Illustration. Die geladenen 3D-Objekte bestehen aus verschiedenen Teilen, die jeweils mit einer Basisfarbe versehen werden. Zum anderen sollen aber auch die Schattierungen selbst farbig dargestellt werden, wie in Abbildung 4.2 links gezeigt.

Hochaufgelöste Illustrationen

Eine weitere Anforderung betrifft die *Qualität* des ausgegebenen Bildes, da dies auch im Druck verwendet werden soll. Hier werden Bilder mit einer hohen Pixeldichte von

mindestens 300 Pixel pro Inch benötigt. Dies bedeutet, dass eine 14 cm × 9 cm große Abbildung[1] mit 600 DPI eine Auflösung von 3306 Pixel × 2124 Pixel haben sollte.

Diese Rasterbildgröße ist unabhängig von der verwendeten Monitorauflösung. Der Nutzer muss in der Lage sein, das gesamte Bild unter verschiedenen Vergrößerungsstufen betrachten zu können. Das Programm sollte aber im Hintergrund immer alle Manipulationen am Bild mit der gewünschten Zielgröße durchführen.

4.3.3 Schematische Zeichnung

Die vorgelegten schematischen Zeichnungen sind alle reduziert auf das Wesentliche, beispielhaft zu erkennen in Abbildung 4.2 rechts:

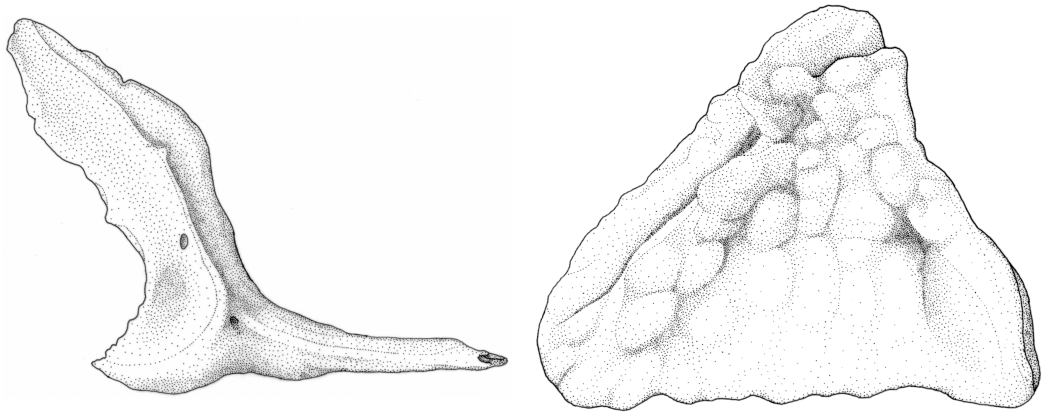
- Die Kontur des Objektes wird durch Linien klar umrissen.
- Markante Oberflächenpunkte sind durch Kreise, Kreuze oder ähnliche geometrische Objekte markiert.[2]
- Markierungslinien werden mit durchgehenden oder gestrichelten Linien verdeutlicht.

Die Zeichnung wird insgesamt von unterschiedlich dicken, abstrakten Linien geprägt. Im Gespräch mit den Mitarbeitern der Paläontologieabteilung stellt sich heraus, dass sie mit dem Vektorzeichnenprogramm *Corel Draw*[3] entstanden ist. Ein Foto wurde in den Hintergrund gelegt und die Kontur wurde durch parametrische Kurven oder Polylinien nachgezeichnet. Dabei ergänzte man auch Teile des Fundstücks *sinngemäß*: Alle Zacken der Zahnkrone wurden gezeichnet, obwohl einer am Fundstück abgebrochen war. Durch das Foto im Hintergrund ist die gesamte Form weitestgehend erhalten, allerdings ist sie durch die Verwendung von parametrischen Kurven geglättet. Diese Zeichnung ist typisch für diese Form der Illustration in diesem Einsatzgebiet und kann als Vorlage herangezogen werden.

4.3.4 Wissenschaftliche Illustrationen

Die hier exemplarisch vorgelegten wissenschaftlichen Illustrationen (Abbildung 4.3) lassen die eingesetzten Techniken gut erkennen: Pünktelung und Linien; Schraffuren werden zur Schattierung überhaupt nicht eingesetzt. Auf Nachfragen teilte man mit, dass sie zwar *schön* aussehen und einen ästhetischen Wert haben, aber im Kontext paläontologischer Illustrationen *unüblich* sind und daher nicht eingesetzt werden. Des Weiteren fällt auf, dass die gesamte Illustration aus verschiedenen Elementen zusammengesetzt ist. Besonders gut ist dies auf Abbildung 4.4 zu erkennen.

[1] Bei einer gebundenen DIN A4 Seite hat der Text eine Breite von 14 cm und eine Höhe von 20 cm.
[2] In der gezeigten Abbildung sind nur Kreise und Ellipsen eingesetzt.
[3] www.coreldraw.com, besucht: 12.11.2014



Linke Jügale Palpebral-Knochen

Abbildung 4.3: Illustration zweier *Xenosaurus platyceps*-Knochen, Juliane Eberhardt (2012)

1. Das Fundstück ist als Graustufenbild leicht im Hintergrund zu sehen. So wirkt die Schattierung des Objektes intensiver und der Oberflächenverlauf wird stärker wahrgenommen. Hier wurde ein Bildbearbeitungsprogramm eingesetzt: Ein Foto wurde mit einem Graustufenfilter bearbeitet und durch α-Blending in den Hintergrund gelegt.

2. Die Kontur wird durch klare Linien gekennzeichnet. Linien werden auch eingesetzt um überlappende oder unterschiedliche Teile voneinander zu trennen.

3. In sehr dunklen Bereichen wird eine unregelmäßige Pünktelung eingesetzt. Durch die *freie* Anordnung der Punkte ist bewusst keine Struktur auf der Oberfläche zu erkennen: Ohne erkennbares Muster wird hier dunkel schattiert.

4. Besondere Markierungslinien werden durch eine regelmäßige Pünktelung hervorgehoben. Diese Art der Markierung ähnelt der schematischen Zeichnung. Auch hier wurden Punkte im gleichen Abstand entlang eines Pfades verteilt.

Allgemein fällt beim Betrachten des Helligkeitsverlaufs und der Pünktelung auf, dass es kaum weiße Flächen gibt, selbst in sehr hellen Bereichen befinden sich einzelne Punkte. An Oberflächenkanten ist der Kontrast stärker hervorgehoben, so dass unterschiedlich intensive Pünktelungen zu erkennen sind. Dies fällt im direkten Vergleich von Objekt und Zeichnung auf.

Im Vergleich zu anderen NPR-Illustrationen wurde relativ wenig manuell eingegriffen. In einer Studie zur Evaluation von NPR-Techniken von Isenberg et al. [Ise06] wird erwähnt, dass auch geometrische Verzerrungen bei der Zeichnung einer Illustration eingesetzt werden, um bestimmte Aspekte zu betonen. Diese Methode konnte nicht beobachtet werden und wurde auf Nachfrage auch ausgeschlossen. Die vorliegenden Illustrationen wurden auf Grundlage eines Bildes beziehungsweise Fotos des Fundstücks

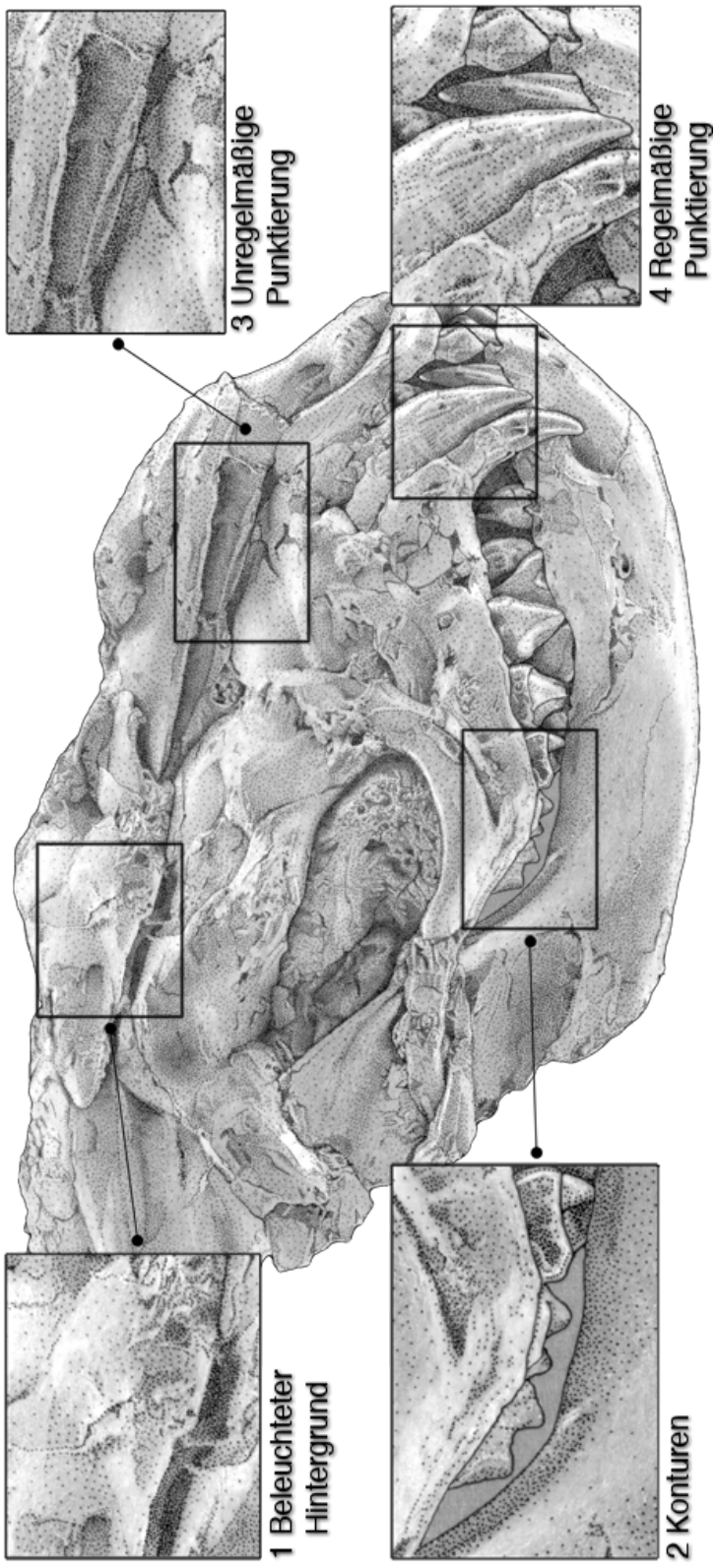


Abbildung 4.4: Illustration eines versteinerten *Kopidodon*-Kopfs von Juliane Eberhardt (2012) mit eingezeichneten identifizierten Rendertechniken aus [SKK12c]

erstellt und die Konturen wurden direkt *durchgepaust*. Im vorliegenden Kontext müssen geometrische und andere Abbildungsverzerrungen als unerwünscht und nicht relevant eingestuft werden. Falls sie dennoch in anderen Domänen gewünscht werden, lassen sie sich über Projektions- und Deformationsmatrizen leicht in die Renderpipeline integrieren.

4.4 Benutzungsoberfläche und Vorwissen

Im folgenden Abschnitt wird das Vorwissen von potentiellen Anwendern auf der Basis von Interviews mit den Entwicklungspartnern besprochen. Die hier gewonnenen Erfahrungen wurden zusätzlich durch Gespräche mit Kollegen und innerhalb der Community bestätigt.

An die Benutzungsoberfläche ist im Kern nur eine Anforderung formuliert worden: eine leichte Benutzbarkeit. Alle an der Entwicklung beteiligten Experten arbeiten täglich mit dem Computer und kennen sich in verschiedenen Bildbearbeitungsprogrammen aus. Dazu zählen die Programme *Adobe Photoshop*[4], *Adobe Illustrator*[5] und *Corel Draw*. Diese Kenntnisse werden aber nicht als Voraussetzung in die Entwicklung mit einbezogen, da dieses Spezialwissen eher die Ausnahme als die Regel ist - zumindest unter den nicht zeichnenden Anwendern, auf die dieses Programm besonders ausgerichtet ist.

Ein spezielles Wissen aus der Computergrafik kann nicht vorausgesetzt werden. Entsprechende Beschriftungen, Parameter und Verfahren sollten daher *allgemeinverständliche Bezeichnungen* tragen. Auch sollte die Oberfläche nicht überfrachtet erscheinen, sondern im besten Fall einen *Einsteigermodus* und einen *Expertenmodus* besitzen, bei dem nur wichtige Schalter zu sehen sind.

Aufgrund des internationalen Anwenderkreises, der allein schon am Naturmuseum und Forschungsinstitut Senckenberg Frankfurt/Main zu finden ist, sollte die Oberfläche, wenn möglich, mehrsprachig sein. Als Ausweichlösung bietet sich zunächst Englisch als Sprache an.

4.5 Interaktivität

Der Begriff der *Interaktivität* bedeutet im allgemeinen Sinn zunächst nur, dass das Programm kein statisches Bild anzeigt, sondern ein vom Nutzer veränderbares. Für die zeitliche Komponente zur Veränderung des Bildes wird auf die Definition von Nielsen [Nie93] zurückgegriffen, der, basierend auf einer Studie von Miller aus dem Jahr 1968 [Mil68], drei psychologische Antwort-Zeitmarken definiert hat:

[4] www.adobe.com/products/photoshop.html, besucht: 12.11.2014
[5] www.adobe.com/products/illustrator.html, besucht: 12.11.2014

- Eine Antwortzeit von 0,1 Sekunden oder weniger nehme ein Anwender als unmittelbare Antwort wahr.

- Eine Antwortzeit von einer Sekunde sei für den Nutzer spürbar, aber sein Gedankenstrom reiße nicht ab.

- Die Aufmerksamkeit eines Nutzer halte ungefähr 10 Sekunden an. Komme es zu einer längeren Verzögerung, bis das System reagiere, sei die Wahrscheinlichkeit hoch, dass der Nutzer sich gedanklich mit anderen Dingen befasse.

Darüber hinaus sagt Miller auch, dass die Antwortzeit *aufgabenangemessen* ist. Sei einem Nutzer bewusst, dass er z.B. eine komplexe Berechnung gestartet hat, so habe er auch mehr Geduld beim Warten auf das Ergebnis.

Ein wesentlicher Aspekt, der in meinen Gesprächen mit verschiedenen Domänenexperten über das Programm deutlich wurde, ist die gewünschte Interaktivität des Programms: die Möglichkeit des Anwenders, das Bild zu verändern und in *angemessener* Zeit ein Ergebnis zu sehen. Insbesondere ein Forscher, der nicht zeichnet, kann so verschiedene Szenarien ausprobieren, ohne vorher eine konkrete Vorstellung der finalen Illustration zu haben. Es können die Kamera, das Licht sowie die ausgewählten Zeichentechniken und deren Parameter interaktiv geändert werden.

Da jede Illustration mit anderen Parametern gerendert wird, müssen sie sogar individuell an das Objekt angepasst werden. Hinzu kommt, dass verschiedene in Kapitel 3.3 und 3.4 erwähnte Algorithmen szenenspezifische Parameter haben, die entsprechend angepasst werden müssen. Mit einem nicht-interaktiven Verfahren ist das nur schwer erreichbar.

4.6 Wiederverwendbarkeit

Von den Entwicklungspartnern des Naturmuseums und Forschungsinstitutes Senckenberg Frankfurt/Main wurde schon früh der Wunsch geäußert, dass die Parameter, mit denen eine bestimmte Zeichnung erzeugt wurde, abspeicherbar sind. Dies hat drei Gründe:

- Zunächst kann eine Zeichnung so zu einem späterem Zeitpunkt leicht verändert werden.

- Weiter kann ein Parameterset zumindest als Startpunkt für die Erstellung anderer Zeichnungen verwendet werden.

- Zuletzt kann eine Parameterdatei zusätzlich zusammen mit einem Modell zwischen Forschern ausgetauscht werden. Dahinter verbirgt sich der Gedanke, dass

in Zukunft eventuell neben den reinen Zeichnungen auch Model- und Parameterdateien mit veröffentlicht werden können, damit sich der Leser selbst ein dreidimensionales Bild machen kann.

4.7 Zusammenfassung

In diesem Kapitel wurden nicht nur wesentliche Eigenschaften des Verfahrens hergeleitet und erörtert. Das Kapitel diente auch dazu, die Lücke zu definieren, die ein Programm ausfüllen soll, das auf Basis des hier erarbeiteten Konzeptes prototypisch erstellt wird. Die Grundlage für die Betrachtung sind zahlreiche Gespräche mit Domänenexperten und die Analyse von ausgewählten vorhandenen Illustrationen, die von den Experten als repräsentativ eingestuft wurden.

Ein Programm, mit dem sich interaktiv wissenschaftliche Illustrationen erstellen lassen, ist eine Bereicherung für die meisten Forschenden. Wann immer Fundstücke veröffentlicht werden, werden spezifische wissenschaftliche Illustrationen benötigt, die von ausgebildeten technischen Zeichnern angefertigt werden müssen. Ein solcher Auftrag kostet nicht nur Geld und Zeit, der Auftrag muss auch konkret formuliert werden. Ein Programm, mit dem man verschiedene Ansichten, Beleuchtungsszenarien und Zeichentechniken ausprobieren kann, hilft hier u.a. Ressourcen zu schonen.

Die benötigte Gesamtfunktion lässt sich in zwei Stufen einteilen. Im ersten Schritt soll das Programm aus Bilderstapeln 3D-Modelle rekonstruieren und exportieren können. Bestehende Verfahren müssen dabei gegebenenfalls hinsichtlich Güte und Geschwindigkeit verbessert werden. Alternativ sollen verschiedene gebräuchliche 3D-Modell-Formate unterstützt werden.

Im zweiten Schritt wird die eigentliche Illustration erstellt. Als typische Unterformen wurden hier die schematische Zeichnung und die wissenschaftliche Illustrationen betrachtet. Die wichtigsten Zeichen- und Illustrationstechniken, die zur Erstellung benötigt werden, sind:

- Konturlinen,
- Pünktelung auf Basis einer Beleuchtung,
- Pünktelung entlang der Konturlinien,
- manuelles Einzeichnen von Linien,
- Markierung bestimmter charakteristischer Modellelemente,
- Integration von Farben für bestimmte Unterobjekte und
- Einsatz von Falschfarben in der Beleuchtungsfunktion,

Die Techniken sollen frei kombinierbar sein.

Zusätzlich muss das Framework erlauben, dass man Modell und Zeichnung manuell manipulieren kann. Neben der Deformation des Objektes muss es möglich sein, die Eingabe und Ausgabe der ausgewählten Techniken manuell anzupassen.

Kapitel 5

Hypothesen

Anhand der vorangegangenen Anforderungsanalyse lassen sich nun die Hypothesen formulieren. Der Kern der folgenden Kapitel bildet die Bestätigung oder Widerlegung dieser Hypothesen.

5.1 Verbesserung der Rekonstruktion

Das Verfahren zur Oberflächenrekonstruktion aus einem Multifokus-Bilderstapel lässt sich als Sequenz von Bildfiltern auf der GPU umsetzen. Weiterhin kann der Algorithmus und damit die Qualität der Rekonstruktion gegenüber aktuellen Verfahren verbessert werden.

Die Betrachtung aktueller Verfahren in Kapitel 3.2.2 hat erwiesen, dass sich klare Defizite in der Rekonstruktion von 3D-Oberflächen aus Bilderstapeln aufzeigen lassen. Insbesondere an Flächen, die in der Mikroskopaufnahme wenig sichtbare Kanten zeigen, versagen die Verfahren.

Auch die Performanz der Implementierungen ist noch verbesserungswürdig. Die GPU ist durch die Programmierbarkeit von Shadern prädestiniert, Bildfilter auszuführen, wie sie in den vorgestellten Algorithmen verwendet werden. Damit sollten sich signifikante Geschwindigkeitssteigerungen ergeben, die mit der eingesetzten Grafikkarte skalieren.

5.2 Illustration als Skizze

Mit dem Framework kann ein nicht zeichnender Forscher eine wissenschaftliche Illustration oder eine schematische Zeichnung erstellen, die sich als Diskussionsgrundlage für eine Handzeichnung eines Illustrators nutzen lässt.

Mit dem Programm kann der Forscher das Fundstück in einer 3D-Ansicht inspizieren und so eine geeignete Zeichenperspektive wählen. Ferner können die implementierten NPR-Algorithmen einen zumindest groben Eindruck von einer fertigen Zeichnung erwecken. Zuletzt bietet das Programm genügend Möglichkeiten der Manipulation am Objekt und an dessen dargestellten Merkmalen.

Ein mit den Entwicklungspartnern besprochenes Einsatzszenario des Frameworks bildet die Erstellung einer Skizze, auf deren Grundlage mit einem Illustrator die geplante Zeichnung besprochen wird. Dies ist besonders dann wichtig, wenn die Zeit der Illustratoren budgetiert ist.

5.3 Illustrationen von Nichtzeichnern

Ist z.B. kein Illustrator verfügbar, kann ein Forscher, der selber nicht zeichnet, mit dem Framework eine wissenschaftliche Illustration oder eine schematische Zeichnung erstellen, die publikationsfähig ist. Er kann dabei mit dem Programm die notwendige Abstraktion von einem konkreten Fundstück erreichen.

Diese Hypothese stellt eine Verschärfung der vorherigen dar. Die erzeugte Zeichnung soll nicht nur die Qualität einer Diskussionsgrundlage haben, sie soll auch den Ansprüchen einer publizierbaren Illustration genügen.

Aus diesen Kriterien ergeben sich auch härtere Anforderungen an alle Elemente, die das finale Bild rendern. Nicht nur die eingesetzten NPR-Algorithmen müssen eine ausreichende Qualität haben, sondern auch die manuellen Anpassungen von Objekt und Renderergebnissen dürfen nicht auffällig und müssen trotzdem in ausreichender Tiefe möglich werden. Die gesamte Pipeline muss darauf ausgerichtet sein, hochauflösende Bilder zu rendern, da in Publikationen verwendete Bilder in den Regel mit mindestens 300 DPI gedruckt werden.

5.4 Interaktivität

Die Pipeline kann so konzipiert und umgesetzt werden, dass sie auf einem aktuellen Rechner mit einem gewöhnlichen Parameterset *interaktiv ist. Dies bedeutet insbesondere, dass geeignete NPR-Algorithmen ausgewählt werden können, die in einem kleinen Zeitbudget berechnet werden können.*

Die Interaktivität der Anwendung ist aus vielen Gründen, die in Kapitel 4.5 besprochen wurden, wichtig für das Zielpublikum. Allerdings müssen durch dieses Kriterium einige NPR-Algorithmen direkt vom Einsatz ausgeschlossen werden.

5.5 Universeller Einsatz

Mit dem Programm können nicht nur Zeichnungen und Illustrationen aus dem Bereich der Paläontologie erstellt werden. Auch für andere wissenschaftliche Fachrichtungen, wie etwa der Biologie, Medizin und Archäologie, lassen sich mit dem Framework veröffentlichbare Illustrationen erzeugen.

Das Programm ist in seinem Einsatz nicht explizit auf die Domäne der Paläontologie beschränkt. Bei Zeichnungen und Illustrationen aus anderen Bereichen werden vergleichbare Zeichentechniken eingesetzt, die im Programm umgesetzt sind. Es stellt sich aber die Frage, ob das Programm die unterschiedlichen Stile und Ansprüche, die sich in den anderen Wissenschaften ausgebildet haben, erfüllen kann.

Kapitel 6

Konzeption

Dieses Kapitel beschäftigt sich mit der Konzeption des Lösungsansatzes, der dazu dient, die im Kapitel 5 formulierten Anforderungen zu erfüllen. Das gesamte Konzept, von der Rekonstruktion hin zu einem illustrativen Rendering, wurde in [SH11] vorgestellt.

Der bisher übliche Arbeitsablauf wird in Abbildung 6.1 in der oberen Hälfte dargestellt. Durch Einsatz des Frameworks kann der hier schematisch gezeigte Arbeitsablauf der unteren Hälfte benutzt werden.

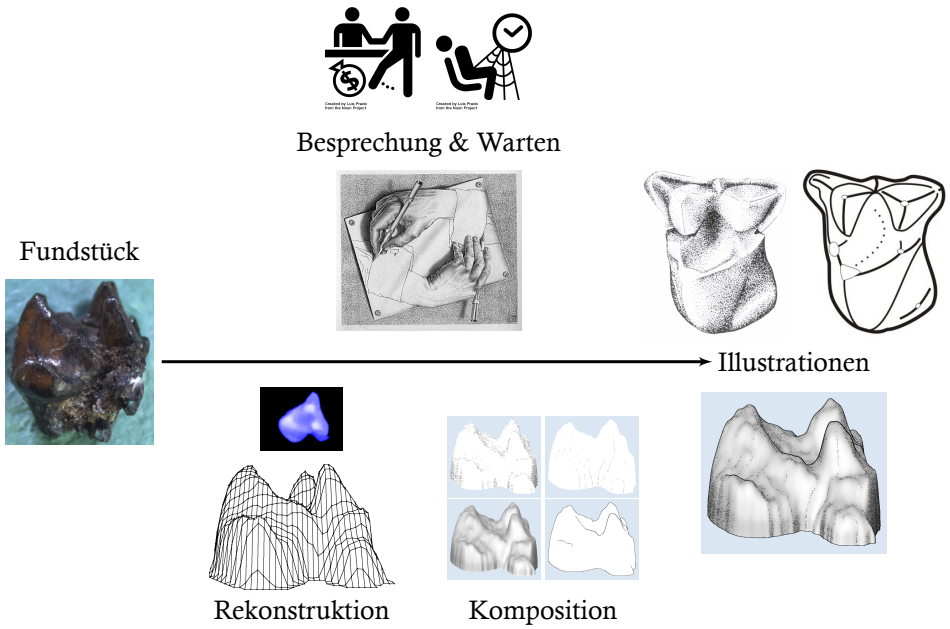


Abbildung 6.1: Arbeitsablauf der Erstellung einer wissenschaftlichen Illustration: traditionell (oben)[1] und mit der Illustrations-Pipeline (unten)

Zunächst wird das Kernelement, die Illustrations-Pipeline, vorgestellt. Im Anschluss daran wird der Ansatz zur Rekonstruktion konzipiert. Danach werden die Elemente der Illustrations-Pipeline, die Ebenen, die eingesetzten NPR-Verfahren sowie die Zeichnen-Funktion und seine Einsatzmöglichkeiten besprochen. Den Abschluss bildet zunächst die Planung der Benutzungsschnittstelle und der Nutzerinteraktion und schließlich das Konzept zur Validierung der Hypothesen.

[1] Quelle: Bilder oben: thenounproject.com/Luis/, unten: *Zeichnende Hände* von M.C. Escher (1948)

6.1 Illustrations-Pipeline

Die Illustrations-Pipeline des Rendering-Frameworks wird in Abbildung 6.2 dargestellt, in [SKK12b] erstmalig beschrieben und [SKK12a] vertieft ausgeführt.

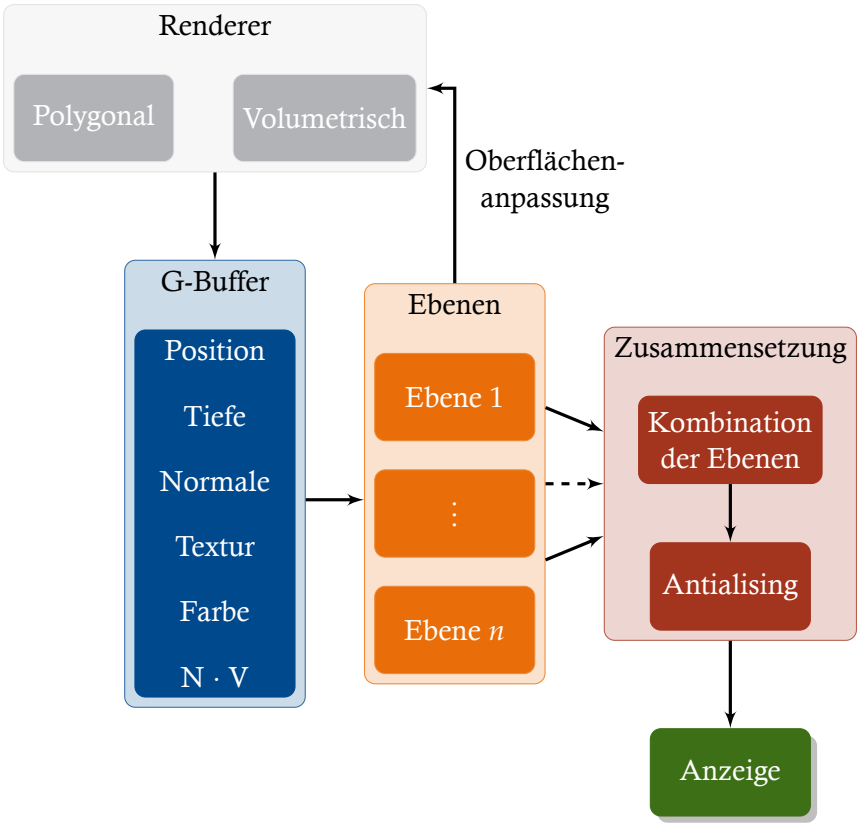


Abbildung 6.2: Die Illustrations-Pipeline im Detail

Im ersten Schritt wird das Modell mit einem passenden Renderer in G-Buffer-Texturen gerendert. Über die anschließend in Texturen gerenderten Ebenen sind sowohl die NPR-Verfahren wie auch die Manipulationsfunktionen umgesetzt. Damit das Bild hochauflösend gerendert wird, haben alle Texturen die gewünschte hohe Auflösung. Die Anzeige lässt sich dabei beliebig skalieren. Durch einen optionalen Antialiasing-Filter lassen sich die visuelle Artefakte klein halten.

Rendering

Durch den Einsatz von G-Buffer muss das Objekt nur einmal gerendert werden. Es können daher beliebig viele Filter bzw. NPR-Verfahren eingesetzt werden, ohne dass das Rendering einer komplexen Geometrie zu einem limitierenden Faktor wird. Auf der anderen Seite bedeutet dies aber auch, dass alle Daten in die G-Buffer geschrieben werden müssen, die die NPR-Verfahren als Eingabe brauchen. Ferner müssen die

NPR-Algorithmen im Bildraum arbeiten, da Daten nur pro Pixel vorliegen. Es sind also geeignete Verfahren zu finden, die dann zum Einsatz kommen.

Ebenen

Mit den Ebenen lassen sich verschiedene Renderalgorithmen und die Abstraktions- und Anpassungsfunktionen umsetzen. Die Ebenen können frei angeordnet werden und eine Ebene kann neben den G-Buffern auch auf vor ihr gerenderte Ebenen zugreifen.

Abstraktion und Anpassung

Wie in Kapitel 2.1 erörtert, sind bei einer wissenschaftlichen Illustration die Aspekte der Abstraktion und Anpassung wichtig. Sie werden innerhalb der Illustrations-Pipeline durch verschiedene Zeichentools implementiert, die alle direkt auf der GPU arbeiten. Vorgesehen sind Freihand-, Polylinien- und Bézierkurvenwerkzeuge.

Die Oberfläche des 3D-Modells wird mit einer *Displacementmap* editiert, mit der Vertices entlang der Normalen verschoben werden. So lassen sich zwar keine radikalen Deformationen erzielen, aber einfache und effektive Anpassungen der Oberfläche sind leicht möglich.

6.2 Rekonstruktion

Die Rekonstruktion ist innerhalb des Frameworks als eigenständiges Modul realisiert, da die Funktionalität losgelöst vom Rendering einsetzbar ist. Basis ist der *Shape From Focus*-Algorithmus, der weiterentwickelt und auf die GPU portiert wurde. Die Erweiterung ist in [SN10] skizziert und in [SH10] veröffentlicht.

6.2.1 Analyse des *Shape From Focus*-Verfahrens

Die Verbesserung setzt an einem inhärenten Problem des originalen *Shape From Focus*-Algorithmus [NN94] an, der in Abschnitt 3.2.2 kurz vorgestellt wurde. Es wird über den gesamten Bilderstapel hinweg die maximale Energie für jedes Pixel berechnet. Die relative Höhe des Bildes aus dem Bilderstapel, in dem die Energie für ein Pixel maximal war, wird in das finale Höhenbild eingetragen.

Die Energie eines Pixels ist definiert durch das Maß der Änderung der Helligkeit in seiner Umgebung [Jäh05]. Man berechnet sie mit einem Gradientenfilter in dem man das Bild mit einem LaPlace-Operator faltet. Der Gedanke dahinter ist, dass man eine hohe Änderung in der Umgebung eines Pixels hat, wenn sie *scharf* ist.

Nicht für jeden Bildpunkt lässt sich eine sinnvolle Energie bestimmen. Gibt es beispielsweise auf einer einfarbigen Fläche bedingt durch die Beleuchtung nur wenig Änderung der Helligkeit, ist die Energie der Pixel, die die Fläche darstellen, niedrig. Das

Problem tritt auch dann auf, wenn die Beleuchtung ungünstig am Material reflektiert wird. Dies kann beispielsweise durch einen schlechten Beleuchtungswinkel oder eine hochglänzende Oberfläche verursacht werden. Ein anderer Fall, in dem die berechnete Energie zu niedrig ist, tritt ein, wenn jedes Bild des Bilderstapels die Oberflächen unscharf abgebildet.

6.2.2 Verbesserung des Verfahrens

Um das Problem zu lösen, wird folgender Weg vorgeschlagen:

1. Elimination von Pixeln mit niedriger Energie durch einen *Schwellwert*.
2. Füllen der eliminierten Pixel (im Folgenden *Lücke*) durch *Interpolation*.

Wenn Pixel nicht genügend Energie haben, liefern sie zunächst keine brauchbaren Informationen für das *Shape From Focus*-Verfahren. Mit einem *geeigneten Schwellwert* können diese Pixel sicher entfernt werden: Würde sich die Oberfläche unter ihnen ändern, so hätte dies auch Auswirkungen auf die Helligkeit und die Energie wäre entsprechend höher. Da dies aber nicht der Fall ist, kann davon ausgegangen werden, dass sich die Oberfläche nicht bedeutsam ändert. Diese Erkenntnis lässt sich beim Interpolieren des Höhenwerts, dem Füllen der Lücke, nutzen.

Die Rekonstruktion kann abhängig von der Qualität der Eingabebilder und Beschaffenheit der zu rekonstruierenden Oberfläche durch zusätzliche Glättung des Höhenbildes verbessert werden. In Tests wurden neben einem Medianfilter [Jäh05] für *Salz- und Pfefferartefakte* vor allem der anisotropische Kuwahara-Filter [Kyp10] eingesetzt. Dieser kann komplett auf der GPU umgesetzt werden. Die Ergebnisse der Operationen werden im Kapitel 8.1.1 dargestellt.

6.2.3 Interpolation der Höhe

Der Höhenwert der durch den Schwellwert eliminierten Pixel, die die Lücken bilden, wird in einem zweistufigen iterativen Verfahren interpoliert, wie in Abbildung 6.3 gezeigt.

In der ersten Stufe werden die Lücken vom Rand aus nach innen gefüllt. Ein Pixel wird dann berechnet, wenn er selbst keinen Höhenwert hat, aber mindestens einer seiner Nachbarn in einer 3×3-Umgebung einen Höhenwert hat. Das Pixel erhält dann den Mittelwert aller Nachbarn als Höhenwert. Das Verfahren ist beendet, sobald alle Pixel einen Höhenwert erhalten haben. Neben der Höhe und der Energie wird noch die Iteration vermerkt, in der ein Pixel in einer Lücke gefüllt wurde. In Abbildung 6.4 ist der gesamte Vorgang an einem Beispiel visuell dargestellt.

Im zweiten Schritt wird die Glättung der Füllung mit einem 3×3 Gauß-Filterkern von innen nach außen vorgenommen. Die Glättung verläuft analog, und zwar von innen

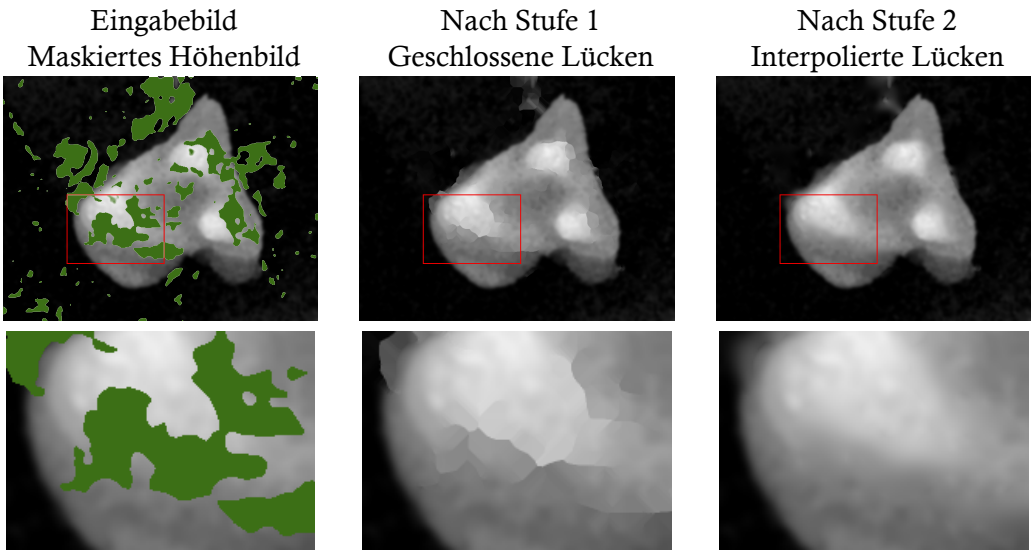


Abbildung 6.3: Schließen der Lücken und Glättung: das gesamte Höhenbild (oben) und ein vergrößerter Ausschnitt (unten). Die Lücken sind grün hinterlegt.

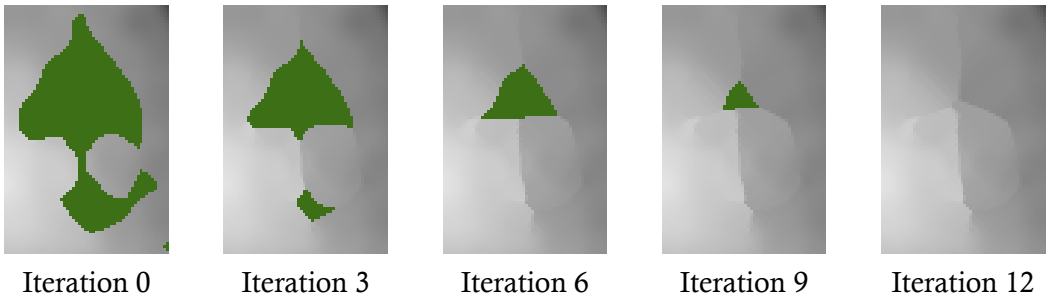


Abbildung 6.4: Auffüllen der maskierten Lücken, die grün hinterlegt sind.

nach außen. In einem Iterationsschritt wird nur das Pixel mit einem adaptiven Gauß-Filter geglättet, dessen Iterationsindex größer als die aktuelle Iteration ist. Es werden in der gesamten Glättung nur solche Nachbarn einbezogen, die in der aktuellen Iteration zu glätten sind. Die Glättung der zuletzt gefüllten Pixel erfolgt sehr häufig, die der zuerst gefüllten nur ein einziges Mal. In Abbildung 6.5 wird die Glättung des vorherigen Beispiels gezeigt.

Die Lücken werden nur über die Interpolation ihrer Höhe gefüllt. Die Oberflächen-Normale könnte an dieser Stelle in das Interpolationsverfahren miteinbezogen werden und so als Stütze für die Interpolationsrichtung der Höhen gedeutet werden. In praktischen Versuchen hat dies zu keiner Verbesserung des Ergebnisses geführt und wurde daher nicht in den Algorithmus integriert.

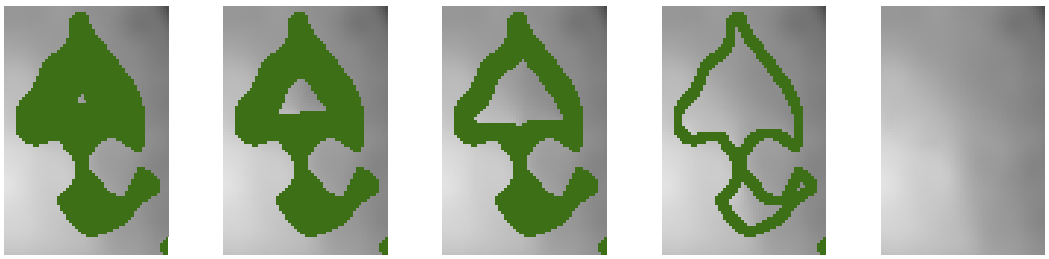


Abbildung 6.5: Glätten der Füllung maskierter Lücken, die grün hinterlegt sind.

6.2.4 Interaktionsmöglichkeiten

Von zentraler Bedeutung ist die Wahl des Schwellwertes. Ist er zu niedrig eingestellt, kommt es zu Fehlern in der Rekonstruktion. Wählt man aber einen zu hohen Wert, so fehlen unter Umständen markante Elemente der Oberfläche und sie wirkt dadurch zu glatt. Da der Parameter von Objekt zu Objekt unterschiedlich gewählt werden muss, sollte dieser interaktiv einstellbar sein. Zu diesem Zweck ist eine *Energy-Height-Map* (EHM) durch das *Shape From Focus*-Verfahren zu erstellen, in der für jedes Pixel die maximale Energie und die zugeordnete Höhe gespeichert ist.

Auf Basis des Höhenbildes wird ein Bild totaler Schärfe aus dem Bilderstapel erzeugt. Auch hier ist es sinnvoll, dass der Nutzer noch manuell in das Ergebnis eingreifen kann. Besonders elegant löst dies die Software *Helicon Focus*, die in Abschnitt 3.2.2 vorgestellt wurde. Mit einem Klonpinsel lassen sich aus den Originalbildern gezielt Bildbereiche markieren, die in das Ergebnisbild kopiert werden. Hier sollte man eine direkte Korrelation zum Höhenbild herstellen können: Sind Bildbereiche im Total-Fokus-Bild aus dem falschen Bild kopiert worden, so wird auch der zugrundeliegende Höhenwert nicht korrekt sein. Die Maskierung lässt sich auch in diesem Programmteil ausnutzen.

6.3 Ebenen

Ein zentrales Konzept des Frameworks bilden die Ebenen, im englischen *Layer*, die in [SKK12c] vorgestellt wurden. Eine Ebene steht im einfachsten Fall für ein bestimmtes Feature, das mit einem bestimmten Zeichenstil dargestellt wird. Eine Ebene kann aber auch andere Informationen, wie etwa handgezeichnete Korrekturen, enthalten. Der genaue Inhalt wird durch den Typ einer Ebene festgelegt.

Das finale Bild wird aus einer oder mehreren überlagerten Ebenen erzeugt. Die Ebenen sind frei arrangierbar und die Sichtbarkeit im finalen Bild kann individuell festgelegt werden. Eine Ebene muss nicht unbedingt visuell *direkt* in das Bild einfließen, sie kann auch *indirekt* als Eingabe einer anderen Ebene oder als neue Displacementmap zum Einsatz kommen.

6.3.1 Ebenentypen

An dieser Stelle werden nun sieben verschiedene Ebenentypen vorgestellt. Die Auswahl erfolgte so, dass damit die zur Analyse vorgelegten wissenschaftlichen Illustrationen, wie sie in Kapitel 4 besprochen wurden, erzeugt werden können. Die Besprechung der eingesetzten Renderingverfahren erfolgt im Abschnitt 6.4:

- Eine *Beleuchtungsebene* hat eine einzelne Lichtquelle. Je nachdem, welches Verfahren ausgewählt wurde, kommen noch zusätzliche Parameter hinzu. Die Beleuchtung ist im Normalfall die Eingabe für die Pünktelung.

- Mit einem *Ambient Occlusion*-Verfahren werden die Effekte einer globalen Beleuchtungsrechnung imitiert. Sie helfen den dreidimensionalen Eindruck zu verstärken. Dazu werden die in Abschnitt 3.3.2 vorgestellten *Postprocessing*-Verfahren eingesetzt.

- Die *regelmäßige Pünktelung* und die *unregelmäßige Pünktelung* sind grundlegende Zeichentechniken in der Erstellung von wissenschaftlichen Illustrationen.

- *Konturlinien* umranden nicht nur die Silhouette des 3D-Modells, sondern trennen auch überlappende Teile voneinander.

- Die *Freihandzeichnenfunktion* und die *Vektorzeichnenfunktion* der Zeichnenebene dienen zum manuellen Eingriff in die Illustration. Mit dem Vektorzeichner lassen sich Linienzüge und parametrische Kurven erstellen. Bei beiden Zeichnenfunktionen ist die Größe, die Farbe und der Verrechnungsmodus mit dem bisher Gezeichneten einstellbar, und zwar additiv und subtraktiv. Einsatz und Nutzen werden detailliert im Abschnitt 6.5 beschrieben.

- Mit einer *Filtergraphebene* ist es möglich, aus vorgegebenen und eigenen Bildfiltern einen Filtergraphen aufzubauen. Die Verwaltung der Filterkette erfolgt mit einem graphischen Editor. Eigene Filter müssen in der Sprache *GLSL* geschrieben sein.

  Eine Kette ist beispielhaft in Abbildung 6.6 gezeigt. Es handelt sich hier um die Filterkette, mit der die in Abschnitt 3.4.1 beschriebene Technik *Image Enhancement by unsharpmasking the depth-buffer* gerendert werden kann.

- *Kombination*: Mit einer Kombinationsebene können mehrere Ebenen zusammengefasst werden. So lassen sich zum Beispiel mehrere Konturlinienebenen zu einer kombinieren, die dann mit Hilfe einer Pünktelungsebene gepünktelt wird. Ohne die Kombination müsste man für jede Konturlinie eine eigene Pünktelungsebene anlegen.

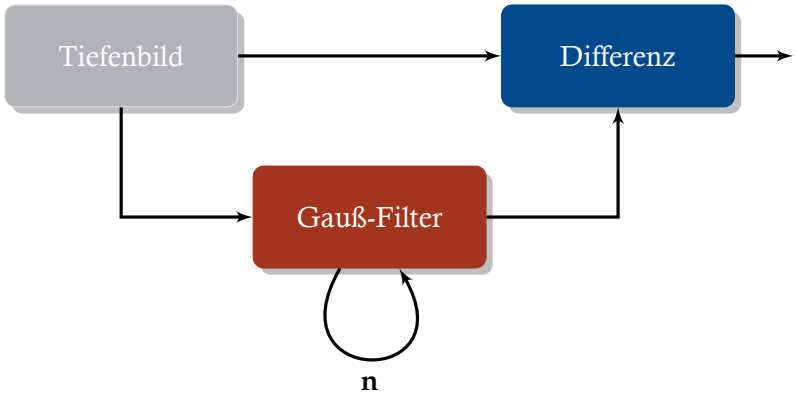


Abbildung 6.6: Filterkette für *Image Enhancement by unsharpmasking the depth-buffer*

- Die *Texturebene* erlaubt es Texturen anzuzeigen. Diese können aus anderen Ebenen heraus kopiert oder extern geladen werden. So ist es beispielsweise möglich eine Illustration auf Basis eines oder mehreren Fotos zu erstellen.

Werden weitere Zeichenstile benötigt, wie beispielsweise Schraffuren, können diese als weiterer Ebenentyp implementiert werden. Über die Mechanismen von dynamischen Bibliotheken können andere Ebenentypen und Renderingverfahren zur Laufzeit nachgeladen werden. Sie müssen nicht Bestandteil des eigentlichen Programms sein, sofern eine definierte Schnittstelle eingehalten wird.

Eine Ebene ist *zeit-veränderlich*, wenn sich ihr Inhalt zwischen den Renderschritten ändert, ohne dass ihre Eingaben oder Parameter geändert werden. Beispielsweise ist das Verfahren zur *regelmäßigen Pünktelung*, *Electrostatic Halftoning* [Sch10], ein *zeitveränderliches* Verfahren. Das Partikelsystem wird zwischen zwei Renderschritten aktualisiert und die Ebene muss neu gerendert werden.

6.3.2 Ausgabe einer Ebene

Die Ausgabe einer Ebene ist eine einzelne Textur, die mit einem α-Wert in den Framebuffer geschrieben werden kann. Diese Textur wird gespeichert und lässt sich in der Illustrations-Pipeline frei einsetzen.

Bevor die Ebene in die Textur gerendert wird, kann sie im letzten Renderschritt noch gefiltert werden. Neben einer Gradationskurvenkorrektur ist eine umfangreiche Einfärbungsmöglichkeit vorgesehen.

Die Gradationskurve wird über einen Gamma- (γ), Kontrast- (k) und Helligkeitsregler (h) eingestellt. Die Formel für die Helligkeit l_{out} basiert nach Poynton [Poy03] auf

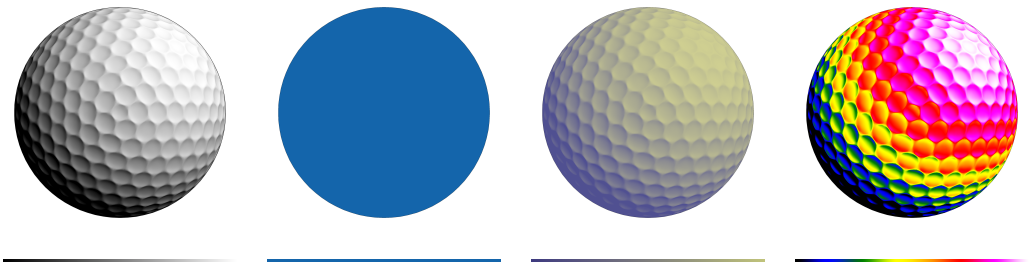
der von *Adobe Photoshop*. Eingabe ist die Helligkeit l_{in}

$$k' = \begin{cases} 1+k & \text{wenn } k < 0 \\ \dfrac{1}{1-k} & \text{sonst} \end{cases}$$
$$l_{\text{out}} = 0.5 + ({l_{\text{in}}}^{\gamma} + h - 0.5) \cdot k'$$
$$\text{mit} \quad l_{\text{in}},\ h,\ k \ \in\ [0;1]; \quad \gamma > 0$$

Für die Einfärbung sind drei verschiedene Möglichkeiten vorgesehen:

1. Die direkte Farbersetzung,
2. die Interpolation zwischen zwei Farben anhand der Helligkeit und
3. der Zugriff auf einen als Textur ladbaren Farbverlauf auf Basis der Helligkeit.

Durch die zweite Möglichkeit kann beispielsweise das *Two-Tone-Shading* [Goo98] realisiert werden. Eine sogenannte *Falschfarbendarstellung*, wie sie von Temperaturbildern bekannt ist, lässt sich über die dritte Möglichkeit erstellen. Alle drei Varianten sind in Abbildung 6.7 dargestellt:



Original | Farbersetzung | *Two-Tone-Shading* | Externer Farbverlauf

Abbildung 6.7: Einfärbungsmöglichkeiten auf Basis der Helligkeit

6.3.3 Abhängigkeiten zwischen Ebenen

Alle Ebenen können zusammengenommen als gerichteter azyklischer Graph dargestellt werden, wie dies die Abbildung 6.8 verdeutlicht. Die Ebenen A - J hängen azyklisch voneinander ab. Das Ergebnisbild entsteht durch Überlagerung der sichtbaren Ebenen A, B, E, J.

6.3.4 Eingabe einer Ebene

Als Eingabe einer Ebene kommen entweder die G-Buffer-Texturen, externe Texturen, die Displacementmap oder *vorher* gerenderte Ebenen in Frage. Diese Einschränkung

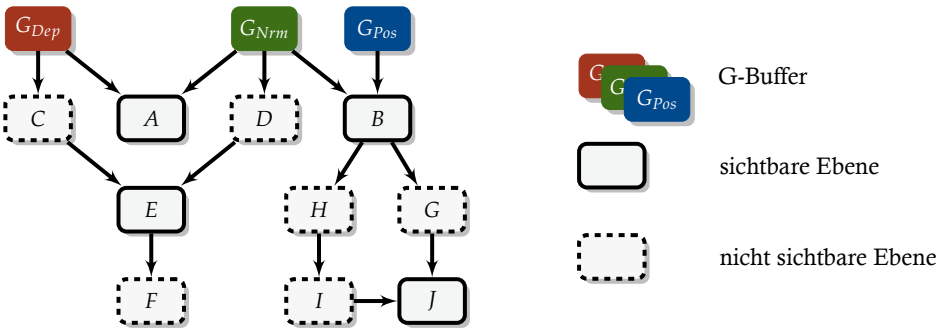


Abbildung 6.8: Ebenengraph, der sich aus den fiktiven Ebenen A - J ergibt

muss an dieser Stelle gemacht werden, da sonst Zyklen im Ebenenrendering entstehen würden: Ebene E_1 hängt von Ebene E_2 ab, die von E_1 abhängt. Solche Abhängigkeitsschleifen sind zum einen semantisch schwierig zu deuten. Nicht nur ist unklar, wie oft die Schleifen ausgeführt werden, sondern auch, wie der so modellierte Effekt aussehen soll. Zum anderen ist auch das Rendern von zyklisch abhängigen Effekten schwierig. Die Ebene E_1 hat E_2 als Eingabe, die vom Ergebnis der Ebene E_1 abhängt. Somit muss eine Ebene ohne alle Eingaben gerendert werden.

Aus den genannten Gründen werden Schleifen verboten und stets nur bereits zuvor gerenderte Ebenen als Eingabe erlaubt.

6.3.5 Ebenen-Manager

Der *Ebenen-Manager* dient dazu, aus den Ebenen das finale Bild performant zu erzeugen. Dazu muss er bestimmen, welche Ebenen gerendert werden müssen und wie sie miteinander komponiert werden. Die erste Aufgabe ist zeitkritisch, da das Rendern der Ebenen die Hauptrenderarbeit des gesamten Frameworks darstellt. Um mit interaktiven Frameraten zu rendern, muss der Ebenen-Manager sicherstellen, dass Ebenen nur dann gerendert werden, wenn sie sich geändert haben und einen Einfluss auf das finale Bild haben. Die Voraussetzungen dafür lauten:

1. Die Ebene wurde noch nicht gerendert.
2. Ein Parameter der Ebene wurde geändert oder
3. die Ebene ist *zeit-veränderlich* und hat sich geändert.
4. Ein Eingabebild der Ebene hat sich verändert.
5. Die Auflösung hat sich geändert.

Zusätzlich ist auch die Sichtbarkeit der Ebenen relevant. Sie muss nur dann gerendert werden, wenn sie angezeigt wird oder sie (Auch indirekt!) die Eingabe einer letztlich sichtbaren Ebenen ist.

Aktualisieren von Ebenen

Es gibt vier verschiedene Typen von Eingabebildern zwischen denen unterschieden werden muss. Jeder Fall ist einzeln zu betrachten, wenn man feststellen will, ob eine Ebene auf Grund eines geänderten Eingabebildes neu gerendert werden muss:

- G-Buffer

  Hat sich das Modell oder die Kamera geändert, liegt ein geändertes Eingabebild vor. Änderungen an Kamera und Modell müssen daher vom Ebenen-Manager vermerkt werden.

- Displacementmap

  Eine Änderung der Displacementmap verändert die Geometrie des Modells. Dieser Fall entspricht einer Modellveränderung und wird somit analog zur Änderung der G-Buffer behandelt.

- Externe Texturen

  Externe Texturen sind nur im ersten Renderdurchlauf wichtig, in dem sie geladen und gesetzt werden.

- Andere Ebenen

  Der schwierigste Fall liegt vor, wenn eine Ebene eine andere als Eingabe benutzt, wie in Abbildung 6.8 dargestellt. Der Ebenen-Manager muss in diesem Fall schnell feststellen, welche Ebenen auf dem Pfad von der geänderter Ebene bis hin zur Senke liegen und alle Ebenen auf diesen Pfad neu rendern. Jede Ebene e führt zu diesem Zweck eine Liste aller Ebenen n_i, die Ebene e als Eingabe ausgewählt haben. Im Ebenengraph speichert eine Ebene alle direkten Nachfolger in der Liste.

Schrittweise Aktualisierung der Ebenen

Ändert man eine zentrale Ebene mit vielen Nachfolgern im Ebenengraphen, müssen viele Ebenen neu gerendert werden. Da dies potentiell lange dauern kann, kann eine zusätzliche Optimierung durchgeführt werden, um den Rendervorgang zu beschleunigen. Die Idee ist, dass man nur eine bestimmte Anzahl von Ebenen in jedem Frame neu rendert. Hier sind zwei mögliche Strategien denkbar:

1. Es werden nicht mehr als n Ebenen in jedem Frame neu gerendert.

2. Es werden so lange Ebenen gerendert, bis eine bestimmte Zeit t vergangen ist. Es muss mindestens eine Ebene pro Renderdurchgang gerendert werden.

Ist eine Ebene auf der Liste der zu rendernden Ebenen, wurde aber im aktuellen Rendervorgang noch nicht aktualisiert, muss sie als *veraltet* markiert werden und darf nicht angezeigt werden. Der visuelle Effekt ist, dass das Bild bei einer Änderung schrittweise aktualisiert wird. Mit jedem sukzessiven Renderdurchgang werden neue Ebenen aktualisiert und können angezeigt werden. Eine erneute Änderung einer zentralen Ebene sorgt dafür, dass der beschriebene Vorgang von vorne beginnt.

Der Vorteil eines solchen Verwaltungssystem ist, dass der Nutzer schrittweises Feedback über das Ergebnis der durchgeführten Operation bekommt. Das Gesamtsystem reagiert direkter und der Nutzer bekommt früher (partielles) Feedback über seine Änderung. Schließlich ist der Verwaltungsaufwand für ein solches System minimal.

6.4 NPR-Verfahren

In diesem Abschnitt werden jene Verfahren besprochen, die sich im interaktiven Kontext auf der Basis von G-Buffern einsetzen lassen. Sie sollen im Prototypen implementiert werden.

6.4.1 Beleuchtung

Die Beleuchtung des Objektes erfolgt nach den in Abschnitt 3.3.2 beschriebenen Verfahren von *Lambert* und *Oren-Nayar*. Für jedes Verfahren soll zwischen zwei Beleuchtungsmodi gewählt werden können:

- Eine *lokale Lichtquelle*, die von einer festlegbaren Position aus das Objekt beleuchtet und
- eine *gerichtete Lichtquelle*, die mit unendlichen Abstand zum Objekt modelliert ist, so dass das Licht auf das gesamte Objekt aus der gleichen Richtung fällt.

Das Verfahren von *Lambert* kommt modifiziert zum Einsatz:

$$\text{intensität} = \frac{(\vec{N} \cdot \vec{L}) - \text{dark}}{1 - \text{dark}}^{\gamma}$$

Der γ-Parameter sorgt für eine allgemeine Betonung oder Abschwächung der Konturen. Der Parameter *dark* transformiert den beleuchteten Bereich in einer Weise, dass auch vom Licht abgewandte Oberflächen noch beleuchtet werden. In wenigen wissenschaftlichen Illustrationen werden komplett schwarze Bereiche gezeichnet, so dass eine Kompensation gerechtfertigt ist.

Der Winkel α zwischen Oberflächennormale $\vec{N}$ und Lichtvektor $\vec{L}$ ist in einem solchen Fall größer als $90°$ und das Skalarprodukt ist kleiner 0. Der *dark*-Parameter verschiebt das Skalarprodukt vom Bereich $[-\text{dark}; 1]$ in den Bereich $[0; 1 + \text{dark}]$. Anschließend wird der Wert in den *üblichen* Bereich $[0; 1]$ skaliert.

Weiter werden zwei der in Abschnitt 3.4.3 beschriebenen Verfahren, *Exaggerated shading* und *Apparent Relief*, eingesetzt. Ihre Betonung der Krümmung der Oberfläche stellt ein wichtiges Merkmal dar, das auch in den Illustrationen zu finden ist. Die Techniken müssen dabei adaptiert werden, so dass sie rein im Bildraum und innerhalb der Illustrations-Pipeline funktionieren.

Mit einer einstellbaren Quantisierung der Helligkeit der Beleuchtungswerte kann die Beleuchtung zusätzlich angepasst werden. Der Helligkeitswert h wird dafür in s Segmente aufgeteilt:

$$h = \frac{\lfloor h \cdot s \rfloor}{s - 1}$$

Das Ergebnis ist in Abbildung 6.9 zu erkennen. Die Quantisierung sorgt insbesondere im Zusammenhang mit Pünktelungsverfahren für ansprechende Ergebnisse, da dadurch ähnliche Helligkeitswerte zusammengefasst werden.

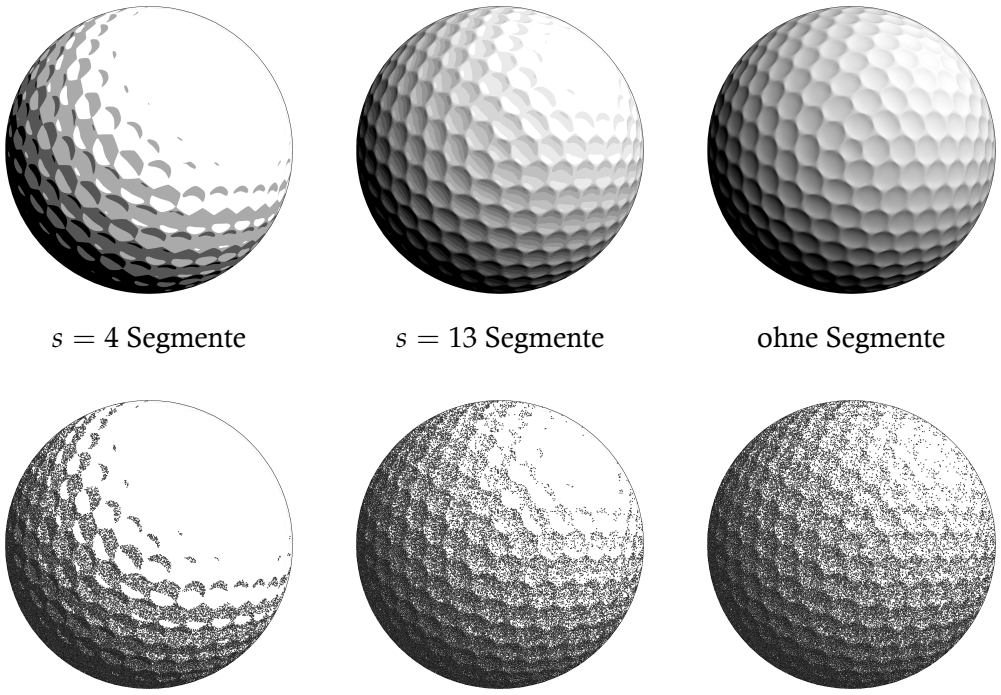


Abbildung 6.9: Beleuchtung mit unterschiedlich vielen Segmenten: vor (oben) und nach Anwendung der unregelmäßiger Pünktelung (unten)

6.4.2 Konturen

Die Konturlinien werden durch die in Abschnitt 3.4.1 vorgestellten Methoden berechnet. Die Verfahren basieren auf Differenzwerten, die entscheiden, wie stark die Änderung in einem Pixel ist. Der Schwellwert, ab wann eine Änderung groß genug ist, um als Kante interpretiert zu werden, ist szenenabhängig und muss daher interaktiv angepasst werden. Drei verschiedene Methoden werden vorgeschlagen, mit denen die Differenz in eine sichtbare Linie umgewandelt werden können:

- eine harte Schwelle,
- eine weiche Kante innerhalb eines Intervalls und
- ein Intervall, in dem die Kante auftritt.

Bei einer weichen Kante wird die Intensität der Kante im α-Kanal gespeichert.

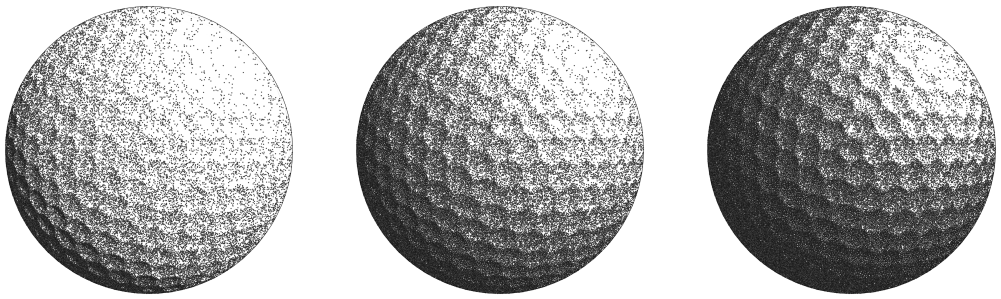
Als Filterkerne zur Kantendetektion werden folgende Kerne nach [Jäh05] eingesetzt: Sobel-Operator, LaPlace-Operator, Canny-Operator und Prewitt-Operator.

6.4.3 Pünktelung

Die Pünktelung erfolgt immer aufgrund der Luminanz der Farbwerte. Je heller, desto weniger Punkte werden gesetzt. Es werden hier die Verfahren *Recursive Wang Tiles for Real-Time Blue Noise* [Kop06] für unregelmäßige und *Electrostatic Halftoning* [Sch10] für eine gleichmäßige, gleichabständige Pünktelung eingesetzt. Die Verfahren wurden im Abschnitt 3.4.2 eingehend beschrieben. Als Quelle sind beliebige Rendertexturen und Ebenen erlaubt. Beide Verfahren arbeiten im Bildraum und erzeugen damit den in Abschnitt 3.4.2 besprochenen *Shower-door effect*.

Das *Electrostatic Halftoning*-Verfahren kommt in abgewandelter Form zum Einsatz. Anstatt das Bild in ein Gitter zu sampeln, werden alle Punkte verarbeitet. Da hauptsächlich ausgewählte Konturlinien als Eingabe dienen, kann davon ausgegangen werden, dass das Eingabebild nur an wenigen Stellen gepünktelt werden soll: Weiße Eingabepixel können direkt verworfen werden. Weiter sind auch die Punkte- bzw. die Partikelanzahl beschränkt. Beide Einschränkungen erlauben den Einsatz des Verfahrens in einem *interaktiven Rahmen*. Das gesamte Verfahren ist mit Hilfe von *OpenCL* als GPGPU-Verfahren auf der GPU implementiert.

Bei beiden Verfahren soll neben einer maximalen Punktzahl auch die Punkteverteilung beeinflusst werden können. Hierfür bietet sich insbesondere ein Gammaregler an, wie in Abbildung 6.10 gezeigt. Gilt $0.0 < \gamma < 1.0$, so wird der mittlere Helligkeitsbereich durch weniger Punkte abgedeckt, bei $\gamma > 1.0$ werden es mehr.



$\gamma = 0.4$ $\gamma = 1.0$ $\gamma = 2.0$

Abbildung 6.10: Pünktelung mit drei Gamma-Werten

6.5 Manuelle Abstraktion und Korrektur der Illustration

Die manuelle Abstraktion eines konkreten Fundstücks gilt als ein elementarer Schritt der Anfertigung einer wissenschaftlichen Illustration. Die Anpassung von algorithmisch gerenderten Ebenen ist eine Eingriffsmöglichkeit, die unter den Begriff der *künstlerischen Freiheit* fällt. Dazu zählt zum einen das gezielte Entfernen oder Hinzufügen von Pünktelungspunkten zur Anpassung des Helligkeitsverlaufs. Zum anderen können so vom Programm nicht extrahierte Features eingezeichnet oder falsch erkannte Features gelöscht oder angepasst werden. Die zweite Methode, ein abstrahiertes Objekt zu erstellen, wird durch die Verformung der Objektoberfläche erzielt.

6.5.1 Zeichnen in Bild- und Objektraum

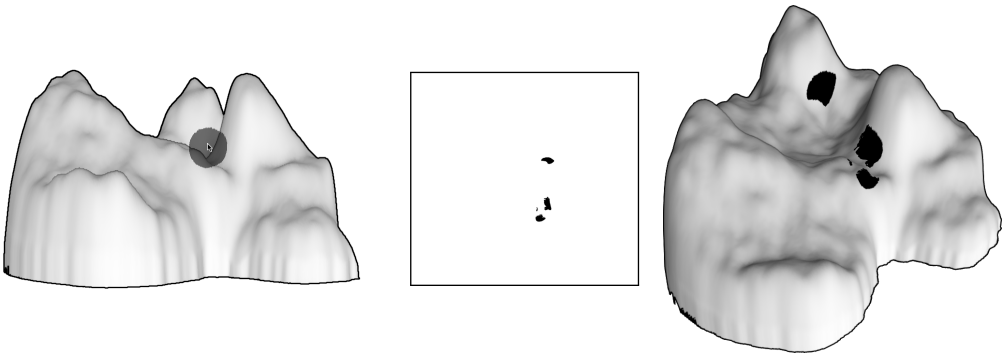
Die beschriebenen Eingriffe erfolgen an zwei verschiedenen Stellen: auf der *Objektoberfläche* und im *Bildraum*. Markierungen auf der Oberfläche erfolgen lokal auf dem Objekt, während sie im Bildraum invariant gegenüber dem dargestellten Objekt bleiben: Sie werden wie eine Folie über das aktuell gezeigte 2D-Bild gelegt. Wichtig für die optische Qualität ist, dass die Textur eine ausreichende Auflösung hat. Dies kann durch Verfahren wie z.B. *Per-face Texture Mapping*[MB11] sichergestellt werden.

Die Funktion des zeichnerischen Eingriffs soll möglichst vollständig auf der GPU umgesetzt werden. Dies bedeutet, dass auf der GPU berechnet wird, an welcher Stelle in einer Textur die Markierung gerendert werden muss, um sie an der gewünschten Position zu erzeugen. Der Vorteil eines reinen GPU-basierten Verfahrens gegenüber einem Verfahren, das auf der CPU die Markierung vornimmt, ist, dass die Texturen nicht zwischen CPU und GPU übertragen werden müssen.

Zeichnen im Objektraum

Das Hinzuzeichnen von nicht extrahierten Features wie auch die Manipulation der Objektform müssen direkt am Objekt selbst, also auf dessen Oberfläche, erfolgen. Hierfür

bietet sich eine Oberflächentextur an. Solche Markierungen sind lokal auf der Oberfläche und damit aus verschiedenen Blickwinkeln sichtbar, wie in Abbildung 6.11 gezeigt wird. Die punktförmige Markierung auf der linken Seite ist fest auf der Objektoberfläche eingezeichnet, wie auf der rechten Seite zu erkennen ist.



Punktförmige Markierung | Oberflächentextur | Rotiertes Modell mit eingezeichneter Oberflächenmarkierung

Abbildung 6.11: Oberflächenmarkierung: Einzeichnung in Ansicht 1 (links), erzeugte Oberflächentextur (Mitte) und alternative Ansicht mit Markierung (rechts)

Zeichnen im Bildraum

Die Korrektur von berechneten Werten, wie beispielsweise der Pünktelung, der Beleuchtung oder der Konturlinien, erfolgt sinnvollerweise im Bildraum. Der Grund dafür ist, dass die Verfahren selbst im Bildraum arbeiten und dort ihr Ergebnis in eine Ebenen-Textur speichern.

6.5.2 Anwendungen der Zeichnenfunktion

Die Zeichnenfunktion wird allgemein als Ebene umgesetzt. In der Ebene wird eine Textur verwaltet, die entweder im Bildraum angezeigt oder auf dem Modell als Oberflächentextur dargestellt wird. In beiden Fällen wird mit der Zeichnenfunktion auf diese Textur gezeichnet. Die Zeichnenfunktion kann auf verschiedenen Arten eingesetzt werden:

1. Die Oberfläche des Objektes wird deformiert.

   Die Oberflächentextur wird als *Displacementmap* [SU08] eingesetzt. In dieser ist die Verschiebung der Vertices entlang der Oberflächennormalen gespeichert. Über die geschriebene Farbe kann kodiert werden, ob die Vertices nach *außen* oder nach *innen* verschoben werden. Neben dem reinen Setzen einer Verschiebung ist auch eine gezielte Glättung und/oder Anrauung der Displacementmap realisierbar.

Die *Displacemap* muss als zusätzliche Textur auf das 3D-Modell angewendet werden. Unter Benutzung von *Transform-Feedback* ist es möglich, das 3D-Modell mit n *Displacementmaps* d_i hintereinander zu verformen:

$$\text{3D-Modell} \xrightarrow{d_0} \text{Modell}_1 \xrightarrow{d_1} \quad \cdots \quad \xrightarrow{d_{n-2}} \text{Modell}_{n-1} \xrightarrow{d_{n-1}} \text{Modell}_n$$

2. Darzustellende Merkmale des Objektes werden editiert.

   Die Merkmale sind durch einen Bildraum-Algorithmus berechnet worden oder noch nicht vorhanden. Die Zeichnenfunktion lässt sich im Bildraum nutzen. Gibt es eine Eingabe, kann diese in die Textur der Ebene kopiert werden, so dass das bisherige Ergebnis editiert wird. Es ist aber auch möglich, nur die Änderungen in der Textur zu halten und durch die Kombination von originaler Ebene und Zeichnenebene das gewünschte Ergebnis zu erzielen. Der Nutzer muss dabei abwägen, welche Methode er wählt, da beide mit Vor- und Nachteilen verbunden sind. Kopiert er beispielsweise den Inhalt in die Zeichentextur, so ist seine Korrektur fest mit der korrigierten Ebene verbunden. Er kann den Inhalt nicht mehr ändern, kann dafür aber immer auf das angepasste Ergebnis zurückgreifen.

3. Eingaben für Ebenen, Filterketten und NPR-Algorithmen werden manipuliert.

   Die Eingabe stellt in diesem Fall entweder eine auf dem Objekt angezeigte Oberflächentextur oder eine Bildraum-Textur dar. Letztere kann entweder aus einer bestehenden Ebene kopiert werden oder mit einer leeren Ebene bei *Null* anfangen.

   In beiden Fällen lässt sich so interaktiv die Eingabe für andere Ebenen beeinflussen. Beispielsweise kann durch die Verknüpfung von *Zeichnenebene* → *Wang-Tile-Pünktelung* ein einfaches Pünktelungs-Malprogramm erzeugt werden. Da die Änderung der Zeichnenebene direkt gepünktelt werden kann, lassen sich so interaktiv unterschiedlich dichte Pünktelungen einzeichnen.

6.6 GUI Konzeption

Das GUI soll mit der plattformübergreifenden Bibliothek *Qt* umgesetzt werden. Sie ist auf allen Zielplattformen mit C++-Anbindung verfügbar. Ein weiterer Vorteil ist das Vorhandensein eines Editors für die Benutzungsschnittstelle. Mit diesem lässt sich ein komplexes GUI grafisch anlegen und verwalten.

Ein zentrales Kriterium für die Erstellung des GUIs ist die Benutzbarkeit (usability). Die elementaren Funktionen sollten zentral erreichbar sein, die Elemente sollten sinnvolle Bezeichnungen tragen und Benutzungsprinzipien sollten wiederverwendet werden.

6.6.1 Projekte

Die Erstellung einer Zeichnung oder einer Illustration ist projektgebunden. Unter einem Projekt sind Modell, verwendete Ebenen und andere ausgabespezifische Einstellungen, wie etwa die Auflösung der wissenschaftlichen Illustration, zusammengefasst. Die Projekte lassen sich abspeichern, laden und austauschen und werden über einen mehrstufigen Assistenten erzeugt:

1. Eingabe von Autor, Titel und weiteren Metainformationen, sowie Wahl der gewünschte Auflösung der wissenschaftlichen Illustration.
2. Laden des 3D-Modells: entweder Starten einer Rekonstruktion oder Auswahl eines polygonalen Modells.
3. Wahl einer Vorlage für die Darstellung: Dies sind globale Einstellungen von Ebenen und Parametern, die mit einem symbolischen Screenshot zur Auswahl gezeigt werden.

6.6.2 Das Hauptfenster

Die Abbildung 6.12 stellt als Skizze das GUI dar. Alle Parameter können aus der Benutzungsschnittstelle heraus über Unterfenster eingegeben werden. Im Anwendungsmenü sind nur die wichtigsten Funktionen zugänglich, um den Fokus des Benutzers ganz auf das Hauptfenster und damit auf die erstellte Illustration zu lenken.

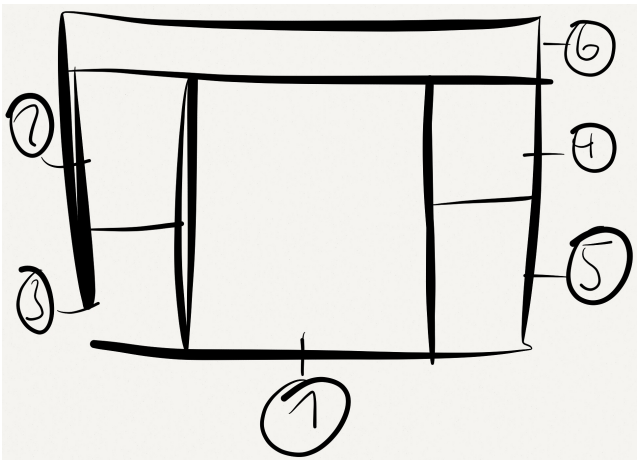


Abbildung 6.12: Hauptfenster des GUIs

Hier eine Auflistung der verschiedenen Komponenten:

(1) Im Hauptfenster ist das Ergebnisbild zu sehen.

(2) Die Parameter der Kamera (Position, Blickfeld usw.) sind im Kamerafenster untergebracht.

(3) Das 3D-Modell kann unabhängig von der Kamera skaliert oder auch um die drei Hauptachsen in der fixen Reihenfolge $X \rightarrow Y \rightarrow Z$ rotiert werden.

(4) Das Ebenenfenster verwaltet die aktuell benutzten Ebenen.

(5) Erweiterte Informationen zum System, wie etwa die Berechnungszeit eines Renderframes, Anzahl der eingesetzte Rendertargets, die Größe des aktuell dargestellten Bildes oder auch Informationen zur OpenGL & GLSL, sind im Informationspanel zu finden.

(6) Über die Werkzeugleiste kann auf wichtige und häufig genutzte Funktionen schnell zugegriffen werden.

Alle Unterfenster sind in das Hauptfenster andockbar. Werden sie zur besseren Übersichtlichkeit deaktiviert, lassen sie sich durch das Anwendungsmenü oder eine Tastaturverknüpfung wieder anzeigen.

6.6.3 Die Unterfenster

Verwaltung der Ebenen

Da das Ebenenkonzept in der bekannten Bildbearbeitung *Adobe Photoshop* ebenfalls vorhanden ist (wenn auch in deutlich anderer Ausprägung) kann sich die grafische Gestaltung des Funktionsblockes an dieser orientieren, wie in Abbildung 6.13 links gezeigt. Ein Nutzer, der mit Photoshop vertraut ist, sollte so durch den Wiedererkennungswert einen leichteren Einstieg in die Bedienung der Software finden.

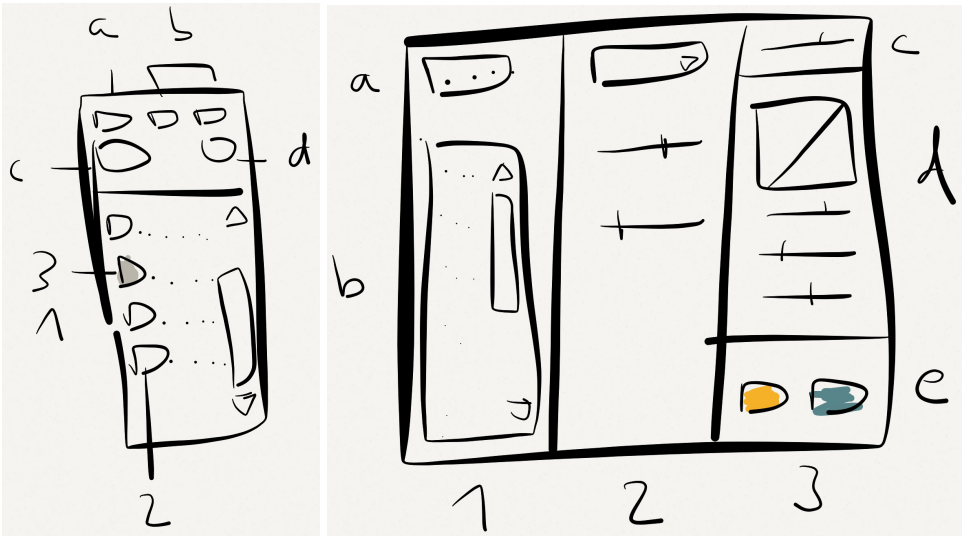


Abbildung 6.13: Fenster zur Ebenenverwaltung (links) und Parameterdialog einer Ebene (rechts)

(1) Die Ebenen stehen in einer frei arrangierbaren Liste übereinander.

(2) Jeder Ebenentyp erhält zur schnellen Identifizierung ein eindeutiges Icon zugewiesen.

(3) Eine nicht angezeigte Ebene verfügt über ein eigenes Symbol.

(4) Über das Menü oberhalb der Ebenenliste können Ebenen angelegt *(a)*, angeordnet *(b)*, entfernt *(c)* oder editiert *(d)* werden.

Editieren einer Ebene

Durch einen Doppelklick auf eine Ebene oder durch den *Editier*-Knopf öffnet sich der in Abbildung 6.13 auf der rechten Seite gezeigte Ebenendialog. Der Dialog verfügt über ein dreispaltiges Layout. In der ersten Spalte *(1)* stehen die allgemeine Eigenschaften, Name *(a)* und Voreinstellungen *(b)*. Die zweite Spalte *(2)* zeigt spezifische Parameter an, die vom Ebenentyp abhängen. In der dritten Spalte *(3)* findet man allgemeine Darstellungsparameter, die jede Ebene hat: Transparenz *(c)*, Tonkurve *(d)* und Einfärbeoptionen *(e)*.

6.6.4 Modelloptionen

Jedes Modell wird beim Laden einheitlich skaliert und um den Ursprung $(0, 0, 0)$ herum zentriert ausgerichtet, so dass das Objekt mit der Standardkameraeinstellung immer zu sehen ist. Da diese Positionierung nicht immer gewünscht ist, soll sowohl Mittelpunkt als auch der Skalierungsfaktor für jede Achse individuell bestimmbar sein. Auch eine Rotation um die drei Hauptachsen kann sinnvoll sein, da das Modell *falsch* rotiert abgespeichert sein könnte.

Ein Modell kann aus mehreren Teilobjekten bestehen. Diese Teilobjekte müssen individuell angezeigt und ausgeblendet werden können. Weiter ist es relevant, wie in Abschnitt 4.3.2 ausgeführt, dass jedem Teilobjekt eine bestimmte Farbe zugewiesen werden kann. Zur Realisierung dieses notwendigen Verfahrensschrittes muss eine geeignete Eingabemöglichkeit in der Benutzungsschnittstelle existieren.

6.6.5 Kameraparameter

Die Kamera beschreibt die Abbildung des Objektes auf die zweidimensionale Bildebene. Durch ihre Modellierung, Positionierung und Orientierung wird die Illustration maßgeblich beeinflusst.

Positionierung und Orientierung

Für die Position und Ausrichtung der Kamera sind zwei Steuerungsmechanismen vorgesehen:

- Die freie Navigation
  Die Positionierung der Kamera erfolgt frei im dreidimensionalen Raum. Mit der Maus kann die Orientierung beeinflusst werden. Über die Tastatur ist eine Bewegung *relativ* zur Blickrichtung möglich.

- Die Kugelkoordinatenkamera
  Die Positionierung der Kamera auf einer Sphäre schränkt die Navigation ein, führt aber im Kontext der Betrachtung eines festgelegten Objektes zu einer Erleichterung der Bedienung.

  Die Kamera c in Abbildung 6.14 blickt stets in Richtung des Sphärenzentrums und der *Up*-Vektor der Kamera zeigt stets zum oberen Pol. Mit der Maus können der Azimuthwinkel ϕ über die x-Achse m_x und der Polarwinkel θ über die y-Achse m_y eingestellt werden. Über eine zusätzliche Taste kann der Abstand r zum betrachteten Objekt positioniert werden.

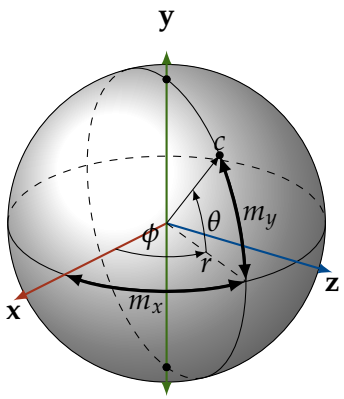


Abbildung 6.14: Kamerasteuerung mit einer Kugelkoordinaten-Kamera

Projektionen

Für die Kamera lässt sich eine aus den folgenden zwei Projektionen auswählen:

a) Die *perspektivische Projektion* wird vom Menschen als *normal* empfunden, da durch sie 3D-Welten so projiziert werden, wie wir sie mit unsern Augen sehen. Dies bedeutet insbesondere, dass Größenverhältnisse nicht erhalten bleiben und Objekte kleiner abgebildet werden, wenn sie weiter von der Kamera entfernt sind.

b) In der *orthogonalen Projektion* wird ein Objekt orthogonal auf die Bildebene projiziert und der Abstand eines Punkte zur Kamera nicht berücksichtigt. Die Projektionsstrahlen sind parallel zueinander. Diese Art der Abbildung ist dann sinnvoll, wenn aus dem Bild Größenverhältnisse erkennbar sein sollen. Sie ist insbesondere in der Architektur und der Kartographie beliebt.

6.6.6 Mausinteraktion

Die Maus wird im Hauptfenster eingesetzt und kontrolliert dort die Position von Kamera und Lichtern, Zoom sowie die Zeichnenfunktion:

- Eine *Combobox* in der Werkzeugleiste bestimmt, an welches Element die Mausbewegung mit gedrückter linker Maustaste weitergegeben wird. In Frage kommen hier Kamera, Licht- oder Zeichnenebene.
- Das Mausrad kann benutzt werden, um auf die Illustration zu zoomen. Diese Funktion ist wichtig, um Details zu betrachten.
- Mit der gedrückten rechten Maustaste lässt sich der in Vergrößerung dargestellte Ausschnitt verschieben. Das Prinzip ist in Abbildung 6.15 illustriert.

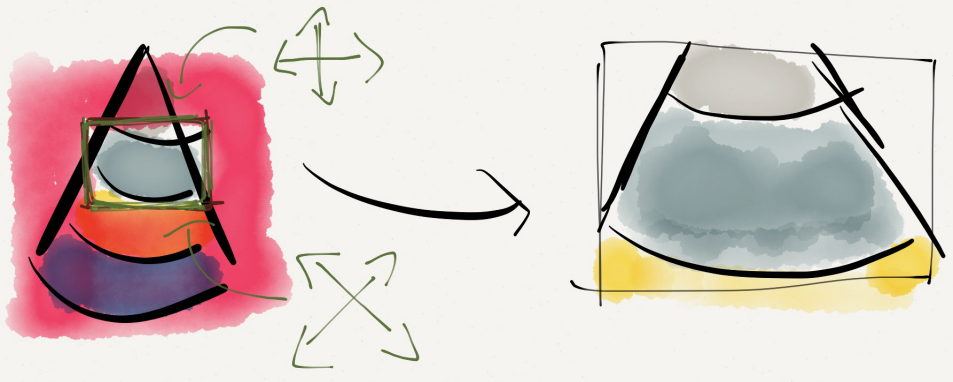


Abbildung 6.15: Die Zoomfunktion im Einsatz: gesamtes Objekt mit ausgewählten Bereich (links) und der vergrößerte Bereich (rechts)

6.6.7 Einstellungen und Personalisierung

Das GUI kann auf verschiedene Arten durch den Benutzer angepasst werden.

Einstellungen

In der Anwendung gibt es zwei Arten von Einstellungen: *globale* und *ebenenspezifische*. In einer globalen Einstellungsdatei sind gespeichert:

- die existierenden Ebenen,
- der Name der Datei, die die Parameter jeder Ebene speichert,
- der Name des Nutzers und das Erstellungsdatum,
- ein Screenshot des aktuellen Hauptfensters, damit die Einstellungsdatei später leichter identifiziert werden kann.

Eine Einstellungsdatei kann noch leicht um weitere Attribute, wie beispielsweise Kategorien (Tags), erweitert werden.

Eine ebenenspezifische Einstellungsdatei besteht aus allen Parametern, die im Ebenendialog ausgewählt sind. Sie trägt einen Namen und kann nur für Ebenen des gleichen Typs verwendet werden.

Personalisierung

Die Anwendung kann auf mehrere Arten personalisiert werden. Die Positionierung und der Anzeigezustand von Haupt- und Unterfenster wird beim Beenden der Anwendung gespeichert. Genauso werden die zuletzt benutzten Pfade und Einstellungen im persönlichen Nutzerprofil gespeichert. Die Anordnung der Unterfenster lässt sich über einen Menüpunkt wieder zurücksetzen.

6.7 Bewertung der aufgestellten Hypothesen

Die im Kapitel 5 aufgestellten Hypothesen werden in dieser Arbeit methodisch in dreifacher Form evaluiert:

- Evaluation des Programms mit Experteninterviews,
- Evaluation der Ergebnisse durch eine Onlineumfrage und
- Beurteilung der Rekonstruktion durch einen Vergleich und Benchmarks.

6.7.1 Anwenderinterviews

Im Rahmen eines Anwendertests zur Benutzbarkeit des Programms durch die Entwicklungspartner und repräsentative Personen der Zielgruppe soll ein Interview zur Verwendbarkeit des Programms stattfinden. In diesem Interview sollen qualitative Aussagen zu folgenden Fragen gefunden werden:

- Reichen die erzielten Ergebnisse aus, damit sie als Diskussionsgrundlage für eine Handzeichnung dienen?
- Können die Ergebnisse in dieser Form veröffentlicht werden?
- Wie aufwändig ist die Erstellung von Zeichnungen und Illustrationen mit dem Programm?

Wie in Abschnitt 3.5 erörtert, sind in der Diskussion über die Eignung von Verfahren von gerenderten NPR-Bildern qualitative Aussagen von großer Relevanz.

6.7.2 Onlineumfrage

In der Onlineumfrage soll eine Auswahl der mit dem Programm erstellten Bilder mit handgezeichneten sowie mit anderen computerbasierten Methoden erzeugten Illustrationen aus [Ise06] verglichen werden. Interessant sind hier weniger die quantitativen Aussagen zu den Bildern. Vielmehr ist es von hohem Interesse, wie die Bilder durch Forscher aus unterschiedlichen Domänen beurteilt werden.

Für die Beurteilung ist besonders die Frage relevant, ob die erzeugten Illustrationen für eine Veröffentlichung als geeignet angesehen werden.

6.7.3 Rekonstruktion

Bei der Rekonstruktion müssen zwei Aspekte evaluiert werden: einerseits die Performanz des Verfahrens, andererseits die Güte der Rekonstruktion. Die Performanz kann durch verschiedene Benchmarks ausgewertet werden. Relevant ist nicht nur der direkte Vergleich von CPU $\leftrightarrow$ GPU, sondern auch die Geschwindigkeitsentwicklung auf verschiedenen GPU-Typen mit unterschiedlich großen Bilderstapeln.

Um die Güte einer Oberflächenrekonstruktion zu bestimmen, muss die Rekonstruktion mit einer *Ground-Truth*, einem bekanntermaßen wahren Wert, verglichen werden. Zu diesem Zweck wird mit einem 3D-Renderprogramm *Blender*[2] die Aufnahmesituation nachgestellt und ein Bilderstapel gerendert sowie das Tiefenbild gespeichert. Die Güte einer Rekonstruktion wird in diesem Kontext als die Differenz zwischen dem rekonstruierten Höhenbild und dem invertierten gerenderten Tiefenbild aufgefasst. Die Inversion ist notwendig, da das Tiefenbild den Abstand zur Kamera speichert, das Höhenbild aber genau den umgekehrten Wert beinhaltet. So kann das in dieser Arbeit vorgestellte Verfahren zur Rekonstruktion quantitativ mit den in Abschnitt 3.2.2 erörterten Verfahren verglichen werden. Im letzten Vergleich wird die Kontur eines rekonstruierten Objektes mit einem Fundstück aus einer orthogonalen Position gegenübergestellt.

6.8 Zusammenfassung

In diesem Kapitel ist die Illustrations-Pipeline, der verbesserte Rekonstruktionsalgorithmus, die Benutzungsschnittstelle für den Prototypen sowie die Evaluation konzipiert worden. Mit diesen Bausteinen sollen die Hypothesen validiert werden.

Die Illustrations-Pipeline beschreibt das Verfahren mit dem sich interaktiv wissenschaftliche Illustrationen erstellen lassen und bildet somit den Schwerpunkt der Arbeit. Das 3D-Modell wird zunächst in G-Buffer gerendert, auf denen dann *Ebenen* aufsetzen. Diese lassen sich beliebig anlegen und miteinander verbinden. Die Kombination der Ebenen ergibt schließlich das finale Bild.

[2] www.blender.org, besucht: 14.12.2014

Grundsätzlich beinhaltet eine Ebene ein charakteristisches Merkmal, das mit einem bestimmten Verfahren erzeugt wurde. Über verschiedene Ebenentypen können unterschiedliche NPR-Algorithmen angewendet werden. Spezielle Ebenen dienen zum Abkapseln der Manipulationsfunktionen. Die Ebenenverwaltung ist so konzipiert, dass sie möglichst effizient erfolgen kann.

Die ausgewählten NPR-Techniken decken die erforderlichen Zeichenstile für paläontologische wissenschaftliche Illustrationen ab und lassen sich alle interaktiv einsetzen. Die Methoden zur Manipulation des Modells und die Implementierung der Zeichnenfunktion erfolgt auf der GPU. Sie sind so gestaltet, dass sie alle in Kapitel 4 gestellten Anforderungen erfüllen.

Die Rekonstruktion der Oberfläche eines Objektes aus einem Bilderstapel basiert auf einem existierenden Algorithmus, der hinsichtlich Güte und Geschwindigkeit verbessert wurde. Dies wurde erreicht in dem ein inhärentes Problem erkannt und konzeptionell eliminiert wurde. Durch das Einführen neuer Parameter, die szenenabhängig angepasst werden müssen, ist es wichtig, dass das Verfahren beschleunigt wird, damit es interaktiv angepasst werden kann. Durch die Portierung auf die GPU soll diese Beschleunigung erreicht werden.

Das GUI ist nach aktuellen HCI-Richtlinien konzipiert und soll mit einem plattformübergreifenden Toolkit realisiert werden, so dass der Prototyp auf verschiedenen Betriebssystemen übersetzt werden kann.

Die Hypothesen werden durch einen Anwendertest und einer Online-Umfrage evaluiert. Hier soll den Fragen nachgegangen werden: „*Ist das Programm geeignet und Skizzen zu erstellen, auf deren Basis ein konkreter Zeichnen-Auftrag besprochen werden kann?*“ und „*Kann man mit dem Prototypen wissenschaftliche Illustrationen erstellen, die veröffentlichbar sind?*“.

Der Rekonstruktionsalgorithmus soll hinsichtlich Qualität und Geschwindigkeit evaluiert werden. Um die Güte einer Rekonstruktion zu bestimmen, soll ein Ground-Truth-Vergleich mit einem synthetischen Bilderstapel durchgeführt werden, da so ein direkter Vergleich von Rekonstruktion und Oberfläche möglich ist.

Kapitel 7

Umsetzung

In Kapitel 7 wird die Umsetzung des beschriebenen Konzepts erläutert. Zunächst werden die vier Elemente des Frameworks vorgestellt:

1. eine Bibliothek mit grundlegenden Renderfunktionen, die *Scilly-Lib*,
2. eine Bibliothek zur Rekonstruktion von Oberflächen aus Bilderstapeln,
3. eine Bibliothek mit Methoden zum Erzeugen der Illustrationen und
4. dem Hauptprogramm mit der Benutzungsschnittstelle.

7.1 Bibliothek Scilly-Lib

Viele der Funktionen aus OpenGL sind vom Charakter eher *LowLevel* und wenig abstrahierend. Aus diesem Grund wird im Rahmen der Arbeit eine eigene Bibliothek mit dem Namen *Scilly-Lib* implementiert, die häufig genutzte, sinnvolle Funktionen und Klassen sowie Ressourcenverwaltungsstrukturen beinhaltet. Der Quellcode der *Scilly-Lib* ist auf *github*[1] unter der MIT/X11-Lizenz[2] veröffentlicht worden. Github ist ein kostenloser web-basierter Dienst zur kollaborativen Softwareentwicklung, der von mehr als 3,4 Millionen Nutzern (Stand: 04/2014) hauptsächlich zur Entwicklung von OpenSource-Software genutzt wird.

7.1.1 Strukturierung

Die Implementierung der Bibliothek Scilly-Lib erfolgt nach objektorientierten Paradigmen. Wenn es möglich ist, wird die Mehrfachimplementierung von Funktionen und Methoden konsequent vermieden. Entsprechende Funktionen rücken entweder in eine gemeinsame Basisklasse auf oder sind in statischen Klassen als statische Methoden von beliebigen Instanzen aus aufrufbar. Durch den Einsatz von abstrakten Klassen und Schnittstellen wird ein Grundgerüst implementiert, dass leicht erweiterbar ist.

Zu den wesentlichen Bereichen, die durch Scilly-Lib abgedeckt sind, gehören:

[1] www.github.com, besucht: 07.12.2014
[2] www.gnu.org/licenses/license-list.de.html, besucht: 07.12.2014

- Texturen laden & speichern
- Rendertargets verwalten & benutzen
- GLSL-Shader & GLSL-Bildfilter
- GLSL-Filtergraphen laden, speichern & rendern
- Modelle laden, speichern & rendern

7.1.2 Shader-Manager

Shader stellen ein zentrales Mittel zur Realisierung der Ziele dieser Arbeit dar. Jeder Filter und jedes gezeichnete Element benötigt einen Shadereffekt. Der *Shader-Manager* erlaubt die mehrfache Nutzung einzelner Shaderinstanzen, so dass der gleiche Shader nicht mehrfach geladen werden muss. Über ein *Dictionary* werden die Parameter einer Instanz verwaltet. Der *Shader-Manager* erlaubt auch das Nachladen eines Shaders, so dass Änderungen des Quellcode zur Laufzeit wirksam werden können.

7.1.3 Rendertarget-Manager

Als *Rendertarget* oder auch *Rendertextur* bezeichnet man eine Textur, in die anstelle des Framebuffers gerendert werden kann. Dies erfolgt bei OpenGL nach folgendem Mechanismus:

1. Rendertarget & Renderbuffer mit der passenden Größe anlegen,
2. Rendertarget als Ziel der Renderoperation festlegen,
3. Rendern und
4. Zurücksetzen des Ziels der Renderoperation auf den Framebuffer.

In dem hier entwickelten Konzept werden Rendertargets häufig eingesetzt: Ebenen, G-Buffer und Filter benötigen alle Rendertargets. Der *Rendertarget-Manager* verwaltet freie Rendertargets und legt bei Bedarf neue in der passenden Größe an. Die Freigabe von nicht mehr gebrauchten Rendertargets kann entweder *direkt*, durch expliziten Aufruf des Rendertarget-Managers, oder *indirekt* erfolgen. Hierzu ist ein spezielles *managed* Rendertarget in der Klasse `CRTManaged` implementiert. Dieses ruft im *Destruktor* den Rendertarget-Manager auf, der so das Rendertarget automatisch freigeben kann.

Wird die Auflösung geändert, können alle verwalteten Rendertargets der alten Auflösung gelöscht werden. Da die meisten Verfahren im Bildraum ablaufen, werden Rendertargets hauptsächlich in der Auflösung des Bildes gebraucht.

Für das Herausfinden von nicht mehr benötigten Rendertargets gibt es kein eigenes System, da dieses in den Tests nicht gebraucht wurde. Hier wäre zum Beispiel eine

Methode mit Zeitstempeln umsetzbar, durch die Rendertargets gelöscht werden, sofern sie für eine bestimmte Anzahl von Frames nicht mehr benötigt wurden. Im Prototyp wurde jedoch ein deutlich einfacheres System implementiert: Nach einer bestimmten Anzahl von gerenderten Frames oder dann, wenn der freie Speicher knapp ist, wird die Methode `CleanUp()` am Ende eines Frames aufgerufen. Sie leert alle gespeicherten Rendertargets, so dass diese beim nächsten Frame wieder neu angelegt werden müssen.

7.1.4 Bildfilter

Ein Bildfilter ist ein reiner Fragment-Shader, der immer auf ein Eingabebild angewendet wird. Des Weiteren sollen Ein- und Ausgabebild die gleichen Dimensionen haben und der *Viewport* muss die gleiche Auflösung aufweisen. Als gerenderte Geometrie wird ein sogenanntes *Screen-Aligned-Quad* benutzt: ein Rechteck, das den kompletten Viewport ausfüllt.

Die statische Klasse *SFilterEngine* erleichtert das Arbeiten mit Bildfiltern durch die Bereitstellung von Methoden zum direkten Filtern. Unter Angabe von Filter bzw. Shader des Eingabebildes und der optionalen Möglichkeit einer Hintergrundfarbe kann mit einem Aufruf ein gefiltertes Ausgabebild erzeugt werden.

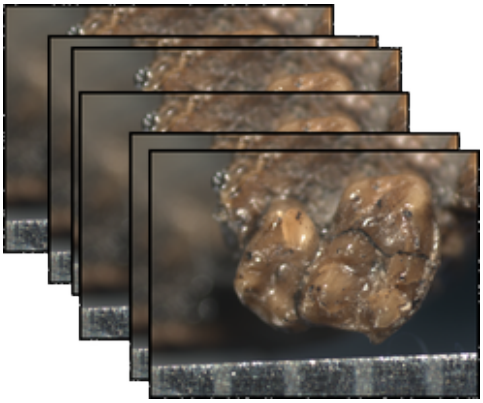
7.2 Rekonstruktion

Die Rekonstruktion der Objektoberfläche aus einem Bilderstapel $\mathcal{I}$ mit n Bildern erfolgt nach dem in [SN10] vorgestellten und in [SH10] beschriebenen Prinzip durchgängig mit Bildfiltern auf der GPU. Die zunächst beschriebene Rektifizierung ist nicht GPU-basiert, sie muss je Bilderstapel aber nur einmal durchgeführt werden. Der Ablauf der gesamten Rekonstruktion ist in Abbildung 7.1 bildlich dargestellt.

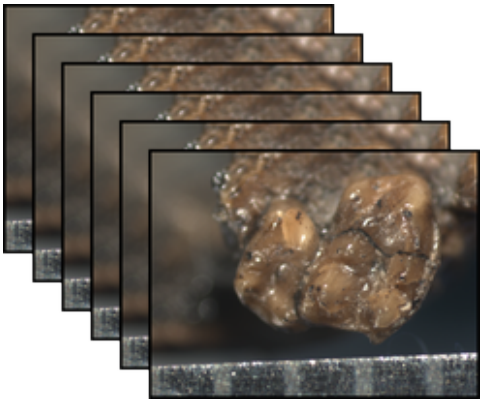
7.2.1 Rektifizierung

Bevor die eigentliche Rekonstruktion durchgeführt werden kann, muss die Bilderserie $\mathcal{I}$ rektifiziert werden. Die Rektifizierung wird aus mehreren Gründen notwendig. Zunächst wird das Objekt umso größer auf einem Bild dargestellt, je näher sich die Kamera auf das Objekt zubewegt. Dieses Verhalten zeigt die Abbildung 7.2, die auch das grundsätzliche Aufnahmesetup illustriert. Weiter wandert ein nicht mittig platziertes Objekt nach außen, je näher die Kamera zum Objekt bewegt wird.

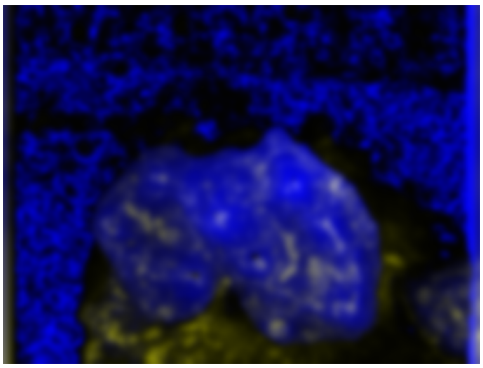
Beide Phänomene lassen sich durch eine *Skalierung* mit Faktor s_i und einer *Translation* t_i für jedes Bild ausgleichen. Diese Parameter müssen nur einmal berechnet werden, da mit ihnen die Bilder rektifiziert gespeichert werden können.



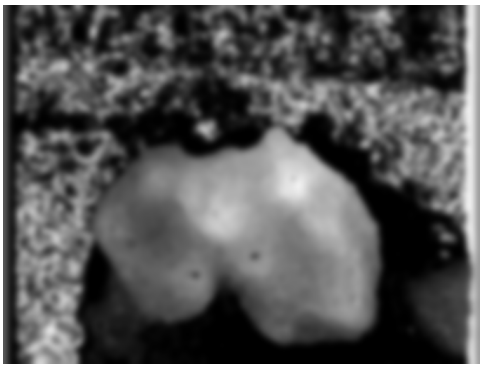
1 - Bilderstapel



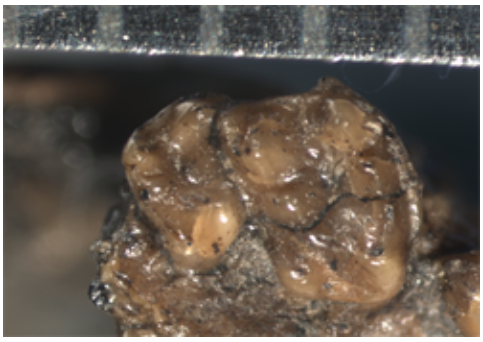
2 - Rektifizierter Bilderstapel



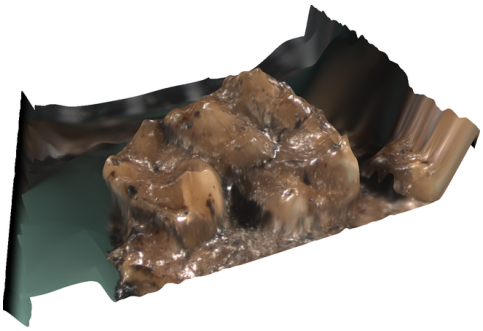
3 - Energy-Height-Map



4 - Höhenbild



5 - Bild totaler Schärfe



6 - Rekonstruktion

Abbildung 7.1: Ablauf der Rekonstruktion aus [SH10]

Skalierung

Die Bilder $\mathcal{I}_i$ mit $0 \leq i < n$ eines Stapels müssen also so transformiert werden, dass das Objekt immer die gleiche Größe hat. Sinnigerweise orientiert man sich am ersten Bild der Reihe, das mit der größten Entfernung zum Objekt aufgenommen wurde. Die anderen Bilder müssen dann mit dem Faktor s_j mit $0 < j < n$ skaliert werden:

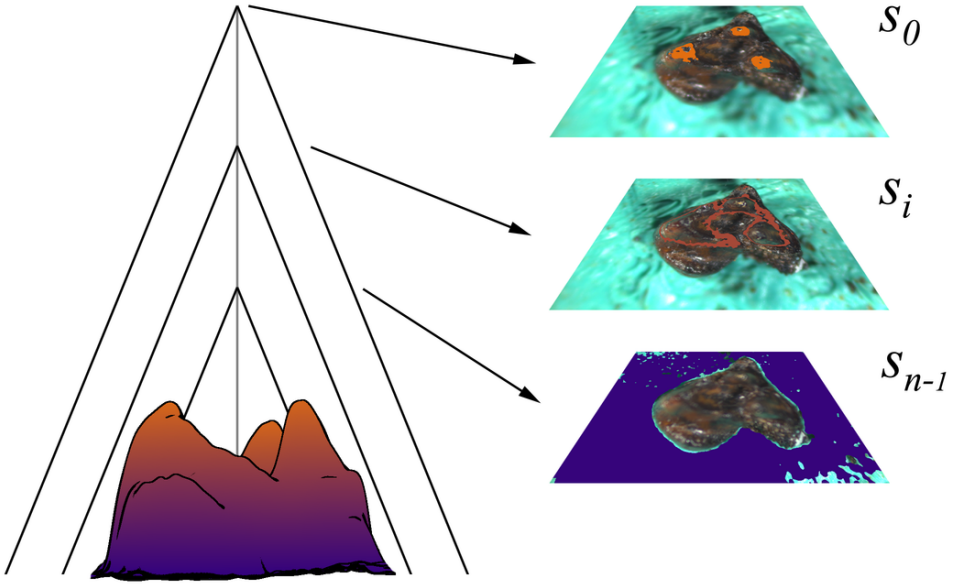


Abbildung 7.2: Darstellung des Setups der Aufnahme eines Bilderstapels. In den drei exemplarisch gezeigten Bildern rechts ist der als *scharf* erkannte Bereich farbig markiert. Aus [SH10].

$$s_i = \frac{|h_{n-1} - h_0|}{|h_{n-1} - h_i|}$$

Die Höhenwerte der j-ten Aufnahme sind durch $h_j := \sum_{i=0}^{j} \Delta_j$ gegeben.

Repositionierung

Ist das Objekt nicht genau in der Mitte des Bildes positioniert, *wandert* das Objekt innerhalb der korrekt skalierten Bilderreihe. Dieses Problem kann mit einer Registrierung gelöst werden, wie beispielsweise der *Methode der kleinsten Quadrate* [Jäh05]. Diese berechnet für das i-te Bild einen Verschiebungsvektor $\vec{t}_i$.

7.2.2 Rekonstruktion der Höhe auf der GPU

Die verbesserte Rekonstruktion, wie in Abschnitt 6.2 vorgestellt, kann vollständig auf der GPU durch Bildfilter implementiert werden. Der gesamte Algorithmus besteht aus vier Teilen:

1. Berechnung der Energy-Height-Map,
2. Maskierung mit einem Energie-Schwellwert,
3. Schließen der Lücken und
4. Glätten der gefüllten Lücken.

Für das gesamte Verfahren benötigt man maximal zwei Rendertargets.

Berechnung der Energy-Height-Map

Im ersten Teil des Algorithmus wird die Energy-Height-Map (kurz: *EHM*) erstellt. Diese speichert für jedes Pixel den maximalen Gradienten der Bilderserie dieses Pixels im roten und grünen Kanal und die relative Höhe des Bildes, in dem die Energie maximal war, im blauen Kanal. Die Berechnung erfolgt in einem iterativen Verfahren, in dem jedes Bild des Bilderstapels mit dem gleichen Fragment-Shader verarbeitet wird. Der Shader braucht neben dem aktuellen Bild des Bilderstapels noch dessen relative Höhe innerhalb des Bilderstapels sowie die bisherige EHM als Eingabe. Das Ausgabe-Bild ist die aktuelle EHM, die im nächsten Schritt wieder als Eingabe dient.

Im Shader wird zunächst die Energie für ein Pixel mit dem LaPlace-Operator bestimmt und anschließend mit der in der EHM gespeicherten Energie verglichen. Ist die Energie des Eingabebildes größer, wird die Energie und Höhe in das Ausgabebild geschrieben, andernfalls werden die Werte der EHM *durchgereicht*. Die beiden Rendertargets speichern Ein- und Ausgabe EHM einer Iteration, da man nicht in die gleiche Textur schreiben kann, aus der man liest.

Je nachdem, wie groß der verfügbare Grafikspeicher ist, wird der Bilderstapel entweder komplett als Stapel von Texturen auf die Grafikkarte geladen oder iterativ einzeln verarbeitet.

Schließen der Lücken

Das Flussdiagramm 7.3 zeigt die Operation des Bildfilters, mit dem sich die Lücken schließen lassen. Die EHM wird so lange mit ihm gefiltert, bis alle Lücken gefüllt sind. Die Lücken werden von außen nach innen geschlossen, wie in Abbildung 6.4 dargestellt. Das hier beschriebene Verfahren benötigt zwei Rendertargets.

Zuerst wird für ein Pixel überprüft, ob es gefüllt werden kann. Dazu muss es in einer Lücke sein und mindestens einen Nachbarn haben, dessen Energie groß genug ist. Die geschriebene Höhe ist der Mittelwert der Höhen aller Nachbarn, deren Energie über dem Schwellwert liegt. So wird die Höhe passend aus der Umgebung berechnet.

Neben der Höhe und der Energie wird noch die Iteration bestimmt, in der ein Pixel in einer Lücke gefüllt wurde. Dieser Schritt wird für das Glätten benötigt.

Glätten der gefüllten Lücken

Die Glättung verläuft in analoger Weise, nur von innen nach außen wobei zusätzlich der Iterationsindex dekrementiert wird. In einem Iterationsschritt werden nur die Pixel mit einem adaptiven Gauß-Filter geglättet, deren Iterationsnummer größer als die aktuelle Iteration ist. Der geglättete Bereich wächst mit jeder Iteration, da in jedem Schritt der

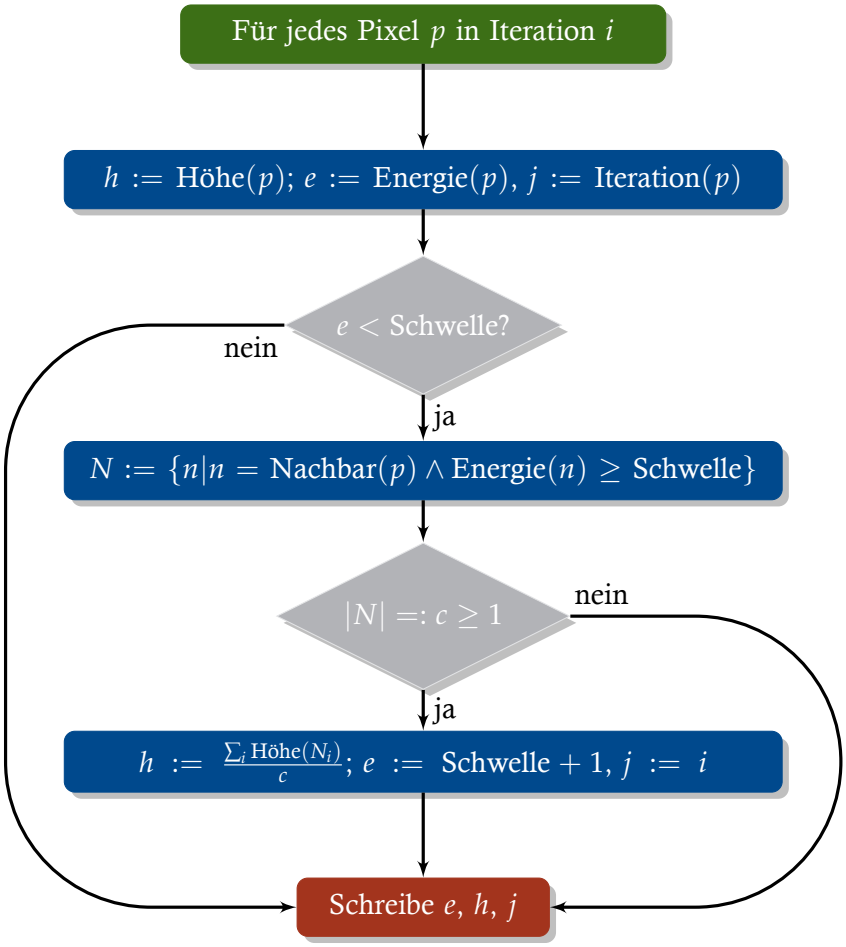


Abbildung 7.3: Flussdiagramm zum Füllen der Lücken

bisherige Bereich um den Rand erweitert wird. So werden die zuletzt gefüllten Pixel sehr oft und die zuerst gefüllten nur einmal geglättet. Die Höhe in der Mitte einer Lücke wird so zum Mittelwert des Randes, gewichtet mit der Entfernung.

Berechnung des Bildes der totalen Schärfe

Ein Bild der totalen Schärfe erhält man, wenn man den Stapel erneut mit einem Shader verarbeitet. Als Eingabe dient die EHM, das bisherige Bild der totalen Schärfe sowie das aktuelle Bild des Stapels und dessen Höhe. Für jedes Pixel wird nun der Höhenwert der EHM mit dem des aktuellen Bildes verglichen. Stimmen sie überein, wird der Farbwert des aktuellen Bildes in das Ergebnisbild geschrieben, ansonsten wird der Farbwert des bisherigen Bildes der totalen Schärfe übernommen. Da für jeden Pixel der EHM genau ein Höhenwert des Stapels gespeichert wird, ist das Ergebnisbild eindeutig und enthält keine Lücken. Auch hier benötigt man zwei Rendertexturen: das bisherige und das neue Bild der totalen Schärfe, die nach jedem Verarbeitungsschritt ihre Rolle tauschen.

7.3 Rendering & G-Buffer

Im Rahmen der Implementierung des Prototypen wird nur ein Renderer für polygonale Modelle und Höhenbilder umgesetzt. Ein dedizierter Renderer für volumetrische Daten lässt sich leicht hinzufügen, wenn dieser alle benötigten Informationen in die G-Buffer rendert.

Das Rendering des Modells erfolgt mit den in OpenGL 3.2 üblichen Methoden des *VAO* (**V**ertex-**A**rray-**O**bject) und *VBO* (**V**ertex-**B**uffer-**O**bject). In einem *VAO* sind verschiedene *VBO*s zusammengefasst. Jedes Attribut, wie Position, Normale, Texturkoordinate oder auch Vertexfarbe, wird in ein eigenes *VBO* gespeichert und dann als Vertexattribute definiert. Über das *VAO* existiert eine OpenGL-ID, unter dem das 3D-Modell mit wenigen Aufrufen gezeichnet werden kann.

Der Rendervorgang erfolgt nicht direkt in den sichtbaren Framebuffer, sondern zunächst wird das Modell in einem einzigen Renderschritt in mehrere Rendertargets geschrieben. Die Technik ist bekannt unter der Bezeichnung *MRT* (**M**ultiple-**R**ender-**T**argets). Folgende Informationen werden in die G-Buffer geschrieben:

- die Oberflächenfarbe,
- die Tiefe (der normalisierte Abstand zur Kamera),
- der Normalenvektor,
- die Texturkoordinate,
- die α-Maske (alle Pixel, an denen das Model zu sehen sind, haben einen α-Wert von $1,0$),
- die Position in Weltkoordinaten,
- das Skalarprodukt von Normale und Vektor zur Kamera ($N \cdot V$) und
- die Unterobjekt-ID (*ID*).

Die benötigten Daten können, wie in Abbildung 7.4 gezeigt, auf insgesamt 4 G-Buffer verteilt werden.

7.3.1 Polygonale 3D-Modelle

Die polygonalen Modelle werden mit der externen Bibliothek *Trimesh2* [Rus12] geladen. Diese kann unter anderem die Formate `.PLY` und `.OBJ` lesen und schreiben. Neben der Berechnung von Vertexnormalen kann die Bibliothek auch *Triangle-Strips*[3] zum effizienten Rendern erzeugen, sowie die *Bounding Sphere* und den Mittelpunkt berechnen. So

[3] deutsch: Dreieckszüge

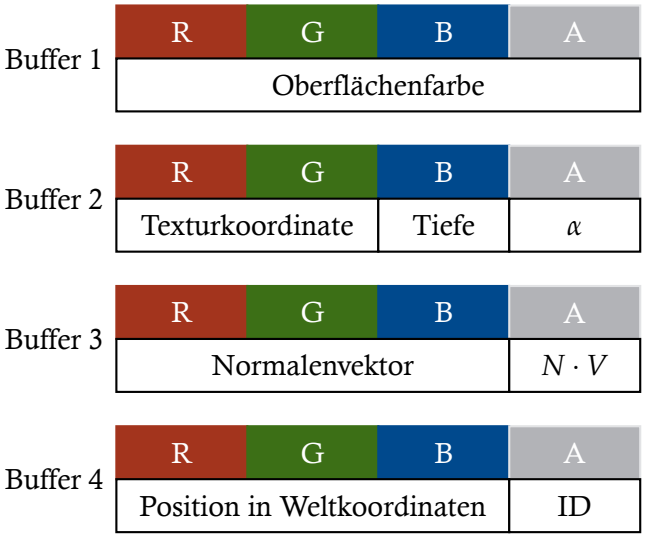


Abbildung 7.4: G-Buffer Organisation

haben alle geladenen Modelle eine Normale, einen Umgebungsradius von 1 und sind um den Ursprung zentriert. Zusätzlich ist noch ein Im- und Exporter in das bekannte und breit genutzte Format *Collada* `.dae` implementiert.

Aus einem Höhenbild wird ein polygonales Modell erzeugt, indem das Gitter mit genauso vielen Knotenpunkten erstellt wird, wie das Höhenbild Pixel hat. Die Höhenwerte der Vertices können direkt aus dem Bild übernommen werden. Die Vertexnormale kann mit einem Verfahren, wie es in der Diplomarbeit *Bumpmapping-Verfahren und deren Weiterentwicklung* [Sch07] vorgestellt wird, als Bildfilter auf der GPU erzeugt werden.

Alternativ kann ein generiertes Modell noch algorithmisch optimiert werden. Verfahren, wie beispielsweise *Surface Simplification Using Quadric Error Metrics* [GH97], benutzen eine Fehler-Metrik um Vertices zu erkennen, die nicht wesentlich zur Form des Modells beitragen. Insbesondere bei aus Höhenbildern rekonstruierten Modellen wird dieser Fall häufiger eintreten, da die erwartete Anzahl von lokal planaren Punkten durch Glättungs- und Fülloperationen hoch ist.

7.3.2 High-DPI Rendering & Zoom

Der Renderer muss zwischen zwei Größen unterscheiden:

- die hohe Auflösung A (im Sinne einer Pixelanzahl) der wissenschaftlichen Illustration und
- die (häufig) kleinere Auflösung B des Ausgabegerätes.

Das Rendering findet grundsätzlich in Texturen mit der hohen Auflösung A statt. Insbesondere benutzen alle Verfahren, die im Bildraum arbeiten, die hohe Auflösung.

Die Darstellung des Ausgabebildes erfolgt in einen Viewport der Auflösung B. Da sich die Illustration sowohl in ihrer Gesamtheit wie auch nur als vergrößerter Ausschnitt anzeigen lässt (siehe dazu Abbildung 7.5), kann es zu sichtbaren Artefakten, *Aliasing*, kommen. Zu diesem Zweck lässt sich optional ein Antialiasing-Filter auf das skalierte Bild anwenden.

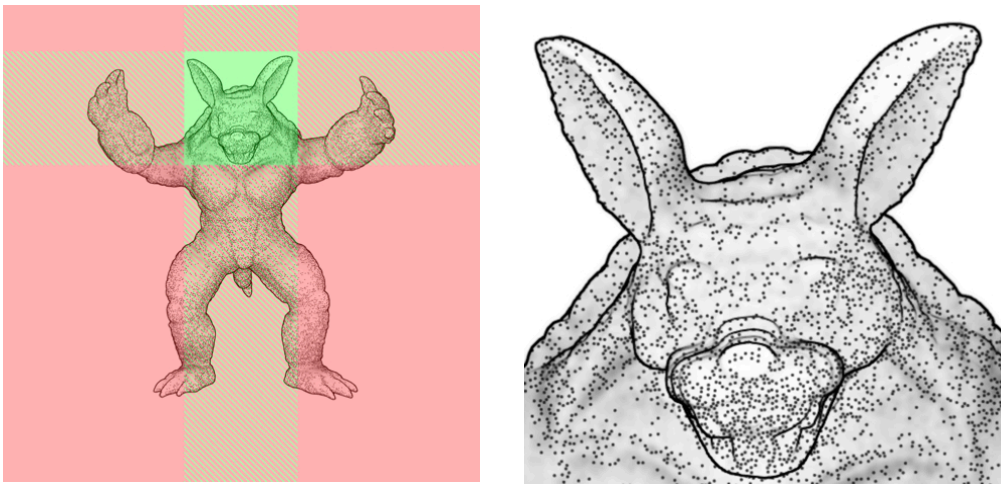


Illustration mit Zoom-Fenster | Vergrößerter Ausschnitt

Abbildung 7.5: Die implementierte Zoomfunktion im Einsatz: gesamtes Objekt mit ausgewählten Bereich (links) und der vergrößerte Bereich (rechts)

Skalierte Anzeige der Illustration

Das Betrachten eines Ausschnitts der Illustration kann alleine mit zwei Parametern beschrieben werden:

- `offset` beschreibt den Abstand der linken oberen Ecke des Ausschnitts zur gleichen Ecke des Bildes und
- `scale` die *Größe* des Ausschnitts.

Beide Parameter werden im Renderer verwaltet und dem Anzeige-Shader übergeben.

Hervorhebung des ausgewählten Bereichs

Um den angezeigten Ausschnitt auszuwählen, lässt sich die gesamte Illustration anzeigen und der ausgewählten Bereich farblich hervorheben, wie in Abbildung 7.5 dargestellt. Die Hervorhebung kann vom gleichen Anzeige-Shader dargestellt werden, der auch den skalierten Ausschnitt anzeigt. Dafür muss als Parameter lediglich bekannt sein, ob diese Ansicht angezeigt werden soll oder direkt der ausgewählte Ausschnitt.

Der relevante Code des Anzeige-Shaders wird im Quellcode 7.1 gezeigt. Die Markierungsfarben wurden im Beispiel genauso wie die Transparenz der Markierung fix eingetragen, sie lassen sich aber leicht von außen durch uniforme Parameter zugänglich machen.

Shader 7.1: Fragment-Shader zum Anzeigen und Färben des vergrößerten Ausschnitts

```
void main()
{
    vec2 tex = texCoord * scale + offset;  // zoomed chunk
    color = texture(image, tex);
    if (showPreview){            // calc overlay color and mix it in
        vec4 blend = vec4(1.0, 0.0, 0.0, 1.0); // default: red
        bool inX = (tex.x > offset.x) && (tex.x < offset.x + size.x);
        bool inY = (tex.y > offset.y) && (tex.y < offset.y + size.y);
        if ((inX && inY){       // center? green!
            blend = vec4(0.0, 1.0, 0.0, 1.0);
        } else if ((inX && !inY) || (!inX && inY)){ //row/col? pattern!
            if (mod(gl_FragCoord.x + gl_FragCoord.y, 16) < 5) {
                blend = vec4(0.0, 1.0, 0.0, 1.0);
            }
        }
        color = mix(color, blend, 0.3); // mix actual image with ↩
            overlay
    }
}
```

7.3.3 Punkte in OpenGL 3.2

Die korrekte und ansprechende Darstellung von Punkten ist grundlegend für die Pünktelungstechnik, die als einer der Eckpfeiler der verwendeten wissenschaftlichen Illustrationen gilt. Mit OpenGL 3.2 ist es allerdings nicht direkt möglich, beliebig in der Größe skalierbare optisch ansprechende runde Punkte zu rendern.

Bewegungsartefakte von Punkten in OpenGL

Verändert man die Punktgröße und wählt eine Breite $b > 1.0$ in OpenGL 3.2, so ist das Ergebnis ein ausgefülltes Quadrat der Pixelgröße $b \times b$. Mit aktivierten OpenGL-interne Antialiasing erhält man einen näherungsweise runden Punkt. Animiert man jetzt die Position des Punktes, so kann man die in Abbildung 7.6 dargestellten Artefakte erhalten. Der Grund dafür liegt darin, dass der Punkt nur innerhalb der Fläche des $b \times b$-Quadrates um den zentralen Pixel des Punktes gerendert wird. Bei einer Positionsveränderung im Subpixelbereich und gleichzeitig aktivierten Antialiasing ändert sich das zentrale Pixel des Quadrates nicht, es verändert seine Position nicht. Gleichzeitig wird

der runde Punkt mit dem Radius $\frac{b}{2}$ um die Subpixelposition gezeichnet. Teile des Kreises, die außerhalb des Quadrates fallen, werden nicht gezeichnet und der Punkt wird abgeschnitten. Sobald die gerundete Mitte auf den nächsten Rasterpunkt fällt, springt das Quadrat und eine andere Ecke des Punktes wird abgeschnitten.

Abbildung 7.6: Vergrößerte Dartellung bewegter Punkte: originale Methode mit sichtbaren Artefakten (oben) und Rendering mit prozeduralen Shader (unten)

Skalierbare Punkte ohne Artefakte

Nun soll ein prozeduraler Shader beschrieben werden, der (frei) skalierbare Punkte ohne Bewegungsartefakte in OpenGL 3.2 rendert. Das Ergebnis ist in Abbildung 7.6 gezeigt. Zu dem Zeitpunkt an dem das hier umgesetzte Verfahren implementiert wurde, war keine solche Methode auffindbar. Mittlerweile wurde das Verfahren von Privatpersonen in privaten Blogs veröffentlicht.

Der Renderer berechnet zu diesem Zweck die *virtuelle* Punktgröße, die bei OpenGL gesetzt wird. Gleichzeitig berechnet man die *relative* Punktgröße, die angibt, wie groß der gewünschte Punkt auf einem Quadrat der *virtuellen* Punktgröße ist. Im Programm wird die quadratische relative Größe angegeben. Dies hat praktische Gründe: Sie müsste in jeder Shaderinstanz ausgerechnet werden und wird durch dieses Vorgehen nur einmal anstatt in jedem Renderdurchgang für jedes Pixel berechnet.

Der Shader, der im Quellcode 7.2 gezeigt wird, benutzt den Alphakanal, um ansprechende Punkte zu rendern. Das Zentrum wird solide gezeichnet, während mit zunehmenden Abstand zum Punktzentrum der α-Wert abnimmt. Die Berechnung erfolgt relativ zum Kreis. Der Mittelpunkt besitzt die Koordinate $(0,0)$ und der Kreis hat immer einen Radius von 1.

In der Konstanten `gl_PointCoord.st` ist die Position eines Pixels relativ zum vom OpenGL-gezeichneten Quadrat im Intervall $[0;1]^2$ angegeben. Durch eine lineare Transformation lässt sich diese Koordinate p in das Intervall $[-1;1]^2$ überführen. Der Abstand zum Mittelpunkt für ein Pixel ergibt sich nun aus der Formel $d = \sqrt{p_x^2 + p_y^2}$. Die teure Quadratwurzeloperation kann allerdings umgangen werden, indem konsequenterweise nur mit den quadratischen Werten gerechnet wird, wie im Quellcode 7.2 gezeigt.

Shader 7.2: Fragment-Shader zum Rendern der Punkte

```
#version 150 core

uniform float relPSS = 1.0; ///< squared relative pointsize
uniform vec4 colPoint = vec4(0.0, 0.0, 0.0, 1.0); ///< point color
uniform float hardness = 1.0; ///< controls size of center

out vec4 fragColor; ///< result: shaded color

void main()
{
   fragColor = colPoint;

    // calculate distance to point-center, calc fall-off
   vec2 p = gl_PointCoord.st * vec2(2.0, -2.0) + vec2(-1.0, 1.0);
   float radius = clamp(dot(p, p) / relPSS, 0.0, 1.0);
   float alpha = 1.0 - clamp(radius - hardness, 0.0, 1.0);

    // antialiasing term (through derivative) modulates alpha
   float aa2 = 2.0 * length(fwidth(p));
   fragColor.a *= (1.0 - smoothstep(1.0 - aa2, 1.0, radius)) * alpha;
}
```

Über den Parameter `hardness` kann die Härte eines Punktes, die Dicke des Zentrums, bestimmt werden. Durch die Shader-Funktion `fwidth` wird der Außenrand jedes Punktes geglättet, so dass kein Aliasing auftritt. Beide Elemente sind in Abbildung 7.7 gezeigt. Zur besseren Hervorhebung des Aliasingfilters sind die Punkte vergrößert abgebildet.

Abbildung 7.7: Vergrößerte gleichgroße Punkte mit unterschiedlicher Härte (aufsteigend nach rechts): mit Antialiasing-Filter (oben) und ohne (unten)

Varianz der Punkte mittels Textur

In Anlehnung an das Verfahren *Stippling By Example*[Kim09] besteht auch die Möglichkeit, eine Textur mit mehreren gezeichneten Punkt-Samples zu benutzen. Die Abbildung 7.8 zeigt eine solche Textur. Ein Illustrator zeichnet nur *sehr* selten exakt gleiche, perfekt runde Punkte. Er variiert Druck und Winkel, so dass kleine Imperfektionen entstehen, die die Zeichnung natürlich und organisch wirken lässt. Durch eine Punkt-

Sample-Textur ist es möglich, auf eine (begrenzte) Anzahl imperfekter Punkte zuzugreifen und sie als Punkte in das Bild einzuzeichnen.

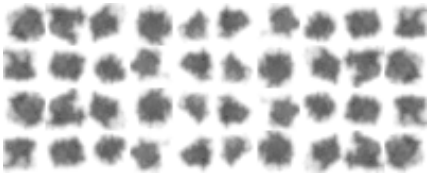


Abbildung 7.8: Textur mit Punkt-Samples aus [Kim09]

Über die Vertex-ID `gl_PrimitiveID` ist es möglich, einen Zeilen- und Spaltenindex zu berechnen. Die Indices adressieren das Punkt-Sample, dass für den zu rendernden Punkt eingesetzt werden soll. In Abbildung 7.9 ist die gleiche Szene links mit uniformen Punkten und rechts mit Punkt-Sample-Textur gerendert.

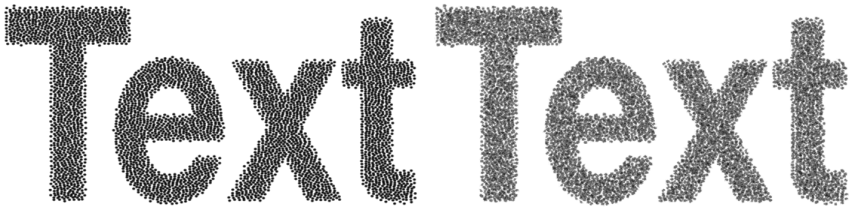


Abbildung 7.9: Uniforme Punkte (links) und Punkt-Samples (rechts)

7.3.4 Linien in OpenGL 3.2

Mit OpenGL 3.2 ist es nicht mehr direkt möglich Linien mit einer anderen Dicke als 1.0 zu rendern. Durch ein in [pau12] erläutertes Verfahren ist es mit dem *Geometry-Shaders* möglich, Linien mit einstellbarer Dicke vollständig auf der GPU zu erzeugen. Zu einem Liniensegment, das aus zwei Punkten besteht, generiert der Geometry-Shader insgesamt vier Punkte, die ein Rechteck bilden, wie in Bild 7.10 zu erkennen ist. Aus der zu zeichnenden Linie $\overline{AB}$ werden die parallelen Linien $\overline{A'B'}$ und $\overline{A''B''}$ erzeugt. Diese vier Punkte werden als *TriangleStrip* gerendert und angezeigt.

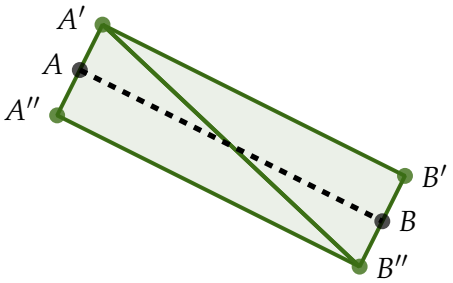


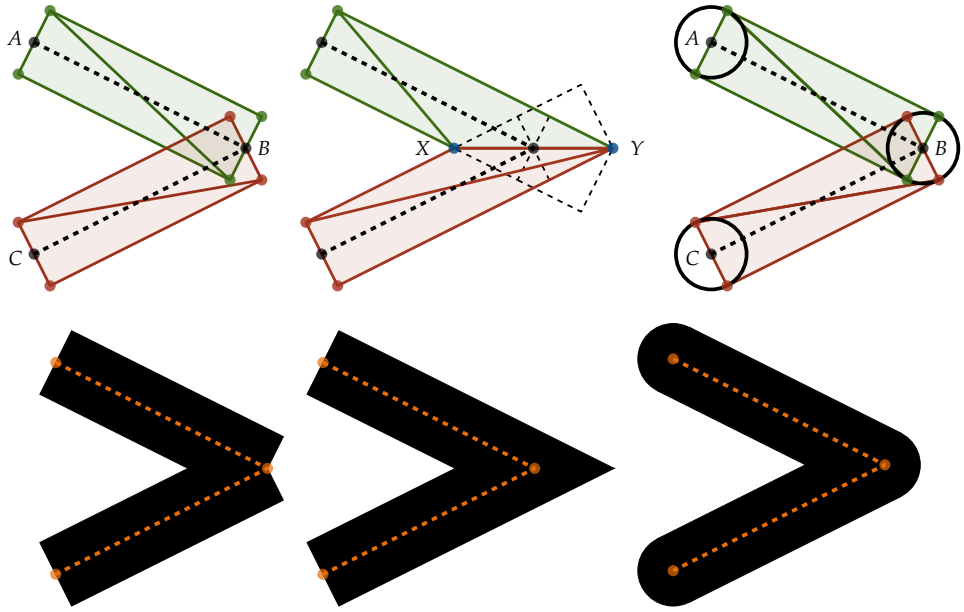
Abbildung 7.10: Dickes Liniensegment erzeugt im Geometry-Shader, nach [pau12]

Problematisch ist das Erzeugen von Linienzügen, da es hier zu ungewollten zusätzlichen Ecken kommen kann, wie in Abbildung 7.11 *(a)* erkennbar wird. Je nachdem, welches visuelle Verbindungselement zwischen zwei aufeinander folgenden Linien eines Linienzugs gewollt ist, ergeben sich drei Möglichkeiten:

(a) Keine Verbindung zwischen den Segmenten, es entstehen sichtbare Kanten.

(b) Man erzeugt zusätzlich noch ein Dreieck, das man auf die Ecke setzt.

(c) Man erzeugt eine runde Verbindung, indem man einen Kreis über die Ecke legt.



(a) Kantige Verbindung (b) Spitze Verbindung (c) Runde Verbindung

Abbildung 7.11: Verknüpfungs-Varianten dicker Linienzüge mit dem Geometry-Shader: Struktur der Liniensegmente (oben) und ausgefülltes Ergebnis (unten), nach [pau12]

Die Lösung für den dritten Fall besteht darin, den Vertexbuffer mit den Eckpunkten der Linien zusätzlich noch als Punktmenge zu rendern. In [pau12] wird ein Verfahren vorgestellt, mit dem sich Linienzüge mit spitzen Verbindungen zeichnen lassen. Mit diesem Shader lässt sich der zweite Fall abdecken.

Als zusätzliche Variante können die Linien auch als eine nicht-kontinuierliche *gestrichelte* Linie gerendert werden. In [Rou13] ist ein Verfahren beschrieben, mit dem sich auf der GPU komplexe Strich- und Punktmuster auf Modell-basierten Linien prozedural im Shader erzeugen lassen. Das Verfahren kann als zusätzlicher Zeichenmodus dem Nutzer angeboten werden.

7.3.5 Parametrische Kurven in OpenGL 3.2

Mit OpenGL 3.2 ist es nicht direkt möglich, parametrische Kurven zu rendern. Durch den Einsatz von Geometry-Shadern können jedoch Bézierkurven gerendert werden, wie in [Mar10] beschrieben. Die in dem Verfahren erzeugten kubischen Bézierkurven sind vom Grad 3 und haben 4 Kontrollpunkte. Der erste Kontrollpunkt ist der Startpunkt und der letzte Kontrollpunkt der Endpunkt der Kurve. Im folgenden Unterkapitel werden ausschließlich kubische Bézierkurven betrachtet: Sie reichen aus, um parametrische Kurven umzusetzen und sind leicht kontrollierbar.

Das Verfahren funktioniert so, dass man eine Linie vom Start- zum Endpunkt als ein einzelnes Liniensegment rendert. Auf diese Linie wird ein Geometry-Shader angewendet, der dynamisch die Linie in mehrere Liniensegmente aufteilt. Dafür benötigt man den folgenden zur Laufzeit änderbaren *uniform*-Parameter:

- die Position der zwei mittleren Kontrollpunkte als Vektor-Array und
- die Anzahl der gewünschten m Unterteilungen.

Im Geometry-Shader wird nun eine m-segmentige Linie erzeugt. Die Position der neuen $m-1$ Segmentpunkte wird anhand der 4 Kontrollpunkte durch Auswertung der Bézierkurve an den Stellen $\frac{i_j}{m} \mid 1 \leq j \leq (m-1)$ berechnet. Die Stelle 0 entspricht dem Startpunkt und 1 entspricht dem Endpunkt der Kurve.

Glattheit und Kontinuität mehrerer Bézierkurven

Sollen komplexere kontinuierliche Bézierkurven mit mehreren Kontrollpunkten erzeugt werden, muss dies über mehrere miteinander verknüpfte Bézierkurven erfolgen. Durch Überlagerung von Anfangs- und Endpunkt bilden zwei Bézierkurven eine zusammenhängende Kurve, die allerdings nicht notwendigerweise *kontinuierlich* und glatt ist. Die Ableitung der beiden parametrischen Kurven kann am gemeinsamen Punkt springen. Die zusammengesetzte Kurve weist somit nur eine C^0-Kontinuität auf. Die einfachste Methode zum Erreichen der C^1-Kontinuität zwischen zwei Bézierkurven mit gemeinsamen Start- und Endpunkt liegt in der Herstellung der *geometrischen tangentialen Kontinuität* [PT97]. Für zwei beliebige Bézierkurven, die durch die Kontrollpunkte $(k_{i-3}, \ldots, k_i)$ und $(k_{i+1}, \ldots, k_{i+4})$ definiert sind, bedeutet dies:

- Start- und Endpunkt $k_i = k_{i+1}$ sind identisch.
- Die Punkte $k_{i-1},\ k_i = k_{i+1},\ k_{i+2}$ sind kollinear.

Für eine Bézierkurve kann daher bestimmt werden, ob die Kontinuität gewahrt werden soll. Falls ja, wird mit der Bewegung eines Kontrollpunktes auch der entsprechende Kontrollpunkt im anschließenden bzw. vorherigen Segment mitbewegt.

7.4 Ebenen

Eine *Ebene* stellt die zentrale Komponente im Rendern der Illustration dar. In ihr wird ein bestimmter Aspekt der Illustration mit einer auswählbaren Technik gerendert. Sie sind in Abschnitt 6.3 näher beschrieben. Wichtig für diesen Abschnitt sind die folgenden Eigenschaften einer Ebene. Eine Ebene kann …

- …Parameter haben.
- …Eingaben haben, die entweder G-Buffer, andere Ebenen oder statische Bilder sind.
- …ausgeblendet werden.
- …mit einem Transparenzwert α in das finale Bild eingeblendet werden.

Es ist auch möglich, dass eine Ebene nicht angezeigt wird, aber als Eingabe einer anderen Ebene dennoch Einfluss auf das finale Bild hat.

Da es verschiedene Ebenentypen gibt, die alle gemeinsame Eigenschaften haben, sind die Ebenen konsequent objektorientiert durch Vererbung umgesetzt. In der abstrakten Basisklasse `ALayer` werden Member angelegt und Methoden implementiert, die alle abgeleiteten Klassen brauchen. Dazu zählt das Rendertarget *Ebenentextur*, die die final gerenderte Ebene speichert, und ein Ebenen-Shader, der die in Abschnitt 6.3.2 beschriebene Gradationskorrektur und Einfärbungsoptionen enthält. Außerdem werden alle Methoden, die eine Ebene implementieren muss, prototypisch definiert.

Die Verwaltung der Ebenen (Anlegen, Anordnen und Löschen) sowie das Rendern der Illustration übernimmt der Ebenen-Manager.

7.4.1 Ebenen-Manager

Die statische Klasse `SLayerManager` hat verschiedene Aufgaben:

- Verwaltung der Ebenen,
- (Neu-)Rendern der geänderten Ebenen und
- Komposition des finalen Bildes.

Die Ebenen werden in einer Dictionary-Datenstruktur verwaltet: Ihrer ID ist jeweils ein Zeiger auf das Objekt zugeordnet. Da alle Steuerkommandos der Benutzungsschnittstelle, die Ebenen betreffen, immer die ID beinhalten, kann so direkt auf die betroffene Ebene zugegriffen werden. In einem zweiten Dictionary ist jeder ID ein Rendertarget zugeordnet, das den Inhalt der Ebene enthält.

Eine Liste speichert IDs der Ebenen in der Reihenfolge ab, wie sie in dem GUI zu sehen sind. Die Reihenfolgenliste fungiert als Prioritätsliste, da die i-te Ebene nur die $i-1$ Ebenen vor ihr als Eingabe nutzen kann.

Verwalten der Ebenen

Die Hauptverwaltungsaufgabe des Ebenen-Managers dient der Pflege der zwei Abhängigkeitslisten jeder Ebene. Bei Änderung von Eingabeebenen oder der Neuanordnung der Reihenfolge müssen diese aktualisiert werden.

Die erste Liste speichert alle Eingabeebenen einer Ebene: die *direkten Vorgänger* im Ebenengraph. Muss eine Ebene gerendert werden, muss auf diese schnell zugegriffen werden können.

In der zweiten Liste einer Ebene werden alle Ebenen eingetragen, die von dieser abhängen: alle *direkten* und *indirekten Nachfolger* im Ebenengraph. Die Eintragung erfolgt in Form einer Liste, die alle Pfade der Nachfolgerebenen speichert. Diese Liste stellt ein wichtiges Mittel dar, um das Rendern der Ebenen effizient umzusetzen: Ändert sich eine Ebene, müssen alle Nachfolgerebenen neu gerendert werden.

In Tabelle 7.4.1 sind die Listen der Ebenen des in Abschnitts 6.3.3 genannten Ebenengraphen aufgeführt, der in Abbildung 7.12 erneut dargestellt ist.

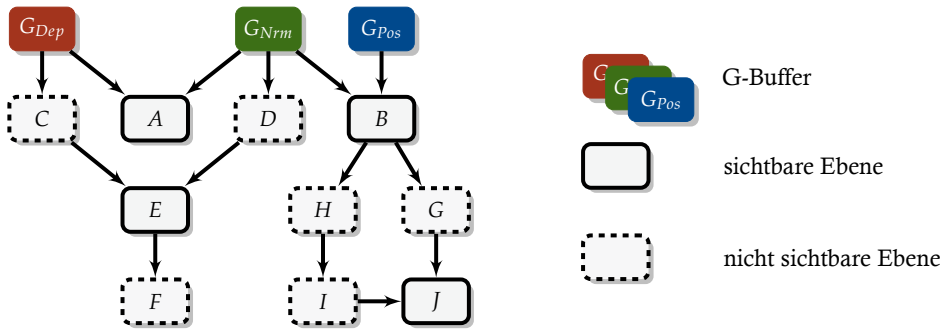


Abbildung 7.12: Graph, der sich aus den fiktiven Ebenen A - J ergibt

Ebene	Vorgänger	Nachfolger
A	[]	[]
B	[]	[[H, I, J], [G, J]]
C	[]	[[E, F]]
D	[]	[[E, F]]
E	[C, D]	[[F]]

Ebene	Vorgänger	Nachfolger
F	[E]	[]
G	[B]	[[J]]
H	[B]	[[I, J]]
I	[H]	[[J]]
J	[G, I]	[]

Tabelle 7.1: Vorgänger- und Nachfolgerlisten des Ebenengraphs aus Beispiel 7.12

Rendern der geänderten Ebenen

Eine Ebene soll nur dann gerendert werden, wenn sich ihr Inhalt ändert und sie selbst oder eine von ihr abhängigen Ebenen sichtbar sind, wie dies in Abschnitt 6.3.5 erörtert wurde.

Mit einem linearen Durchlauf der Reihenfolgenliste können alle notwendigen Ebenen neu gerendert werden. Für jede Ebene wird ermittelt, ob sie neu gerendert werden muss. Es gibt genau zwei Fälle:

- Sie steht in der Liste der geänderten Ebenen.
- Sie wurde in der Vergangenheit geändert, aber nicht gerendert. Zusätzlich wird sie entweder selbst oder einer ihrer Nachfolgerebenen angezeigt.

Da die Pfade in der Nachfolgerliste gespeichert sind, kann über den Anzeigestatus einer Ebene leicht bestimmt werden, ob eine Nachfolgerebene angezeigt wird.

Sind Ebenen aus der Nachfolgerliste einer geänderten Ebene sichtbar, so wird die komplette Nachfolgerliste zur Liste der geänderten Ebenen hinzugefügt. Dadurch ist gewährleistet, dass diese ebenfalls betrachtet werden.

Ein Sonderfall kann auftreten, wenn eine Ebene zwar geändert wurde, aber weder sie noch einer ihrer Nachfolger angezeigt werden soll. In diesem Fall braucht die Ebene nicht unmittelbar gerendert werden. Es muss allerdings vermerkt werden, dass sich der Inhalt geändert haben könnte und sie neu gerendert werden muss, sobald sie oder einer der Nachfolger angezeigt wird. Diese Information wird in der boole'schen Eigenschaft *geändert* einer Ebene festgehalten.

Das Verfahren ist im Quellcode 7.3 in Python-inspirierten Pseudocode gezeigt.

Quellcode 7.3: Bestimmung der neu zu rendernden Ebenen

```
for ebene in ReihenfolgenListe:
   if (ebene in ListeGeänderterEbenen) or (ebene.geaendert):
      neuRendern = ebene.angezeigt
      for nachfolgerPfad in ebene.nachfolgerListe:
         for nachfolger in nachfolgerPfad:
            if nachfolger.angezeigt:
               ListeGeänderterEbenen.extend(nachfolgerPfad)
               neuRendern = True
               break
      if neuRendern:
         ebene.render()
      ebene.geaendert = not neuRendern
```

Durch das Prioritätskriterium der Reihenfolgenliste ist sichergestellt, dass die Ebenen in der korrekten Reihenfolge gerendert werden. Durch die Nachfolgerliste wird ga-

rantiert, dass nur die betroffenen Ebenen gerendert werden. Die Eigenschaften *Sichtbarkeit* und *geändert* helfen zusätzlich die Menge der zu rendernden Ebenen zu minimieren.

Komposition des finalen Bildes

Durch die Überlagerung aller anzuzeigenden Ebenen wird das finale Bild erzeugt. Alle anzuzeigenden Ebenen werden unter Beachtung der Reihenfolge in ein Rendertarget geschrieben. Dieses enthält die aktuelle wissenschaftliche Illustration, die mit dem Anzeige-Shader, vorgestellt in Abschnitt 7.3.2, in den Framebuffer geschrieben wird.

7.4.2 Rendern einer Ebene

Das Rendern einer Ebene wird vom Ebenen-Manager angestoßen. Dieser setzt im ersten Schritt die benötigten Eingabetexturen: geladene Texturen, G-Buffer oder andere Ebenen. Anschließend wird die Ebene nach folgendem Prinzip gerendert:

1. Rendern des Ebeneninhalts in ein temporäres Rendertarget mit dem Verfahren, das dem Ebenentyp zugeordnet ist.

2. Filtern des temporären Rendertargets mit dem Ebenen-Shader in die Ebenentextur mit folgenden Parametern:

   a) Anwendung einer Gradationskorrektur: Gamma $\rightarrow$ Helligkeit $\rightarrow$ Kontrast,

   b) Einfärbung der Ebene - ggf. auf Basis der Helligkeit,

   c) Berechnung der α-Transparenz durch Multiplikation des Ebenentransparenzwertes mit dem α-Wert des Pixels.

Der relevante Teil des Ebenen-Shaders ist im Quellcode 7.4 dargestellt.

Einfärben einer Ebene

Die Einfärbung einer Ebene kann auf drei Arten erfolgen:

1. Farbersetzung: der originale Farbwert wird überschrieben.

2. Interpolation zwischen zwei Farben auf Basis der Helligkeit. Für eine Ebene kann eine Farbe für den Helligkeitswert 0 und 1 festgelegt werden, zwischen denen linear interpoliert wird auf Basis der Helligkeit eines Pixels.

3. Sampeln einer Textur auf Basis der Helligkeit. Aus der Helligkeit des Pixels wird eine Texturkoordinate berechnet, mit der auf eine 1D-Textur zugegriffen wird. Konkret kann die Helligkeit direkt als 1D-Texturkoordinate interpretiert werden, da beide aus dem Intervall $[0;1]$ stammen.

Shader 7.4: Fragment-Shader für Gradationskurve und Färbung

```
float brightness(vec3 col){
    return clamp(dot(col, rgb2brightness), 0.0, 1.0);
}
vec3 tonecurve(vec3 col){
    col = pow(abs(col), vec3(gamma)) * sign(col) + vec3(brightness);
    float k = (contrast < 0) ? 1.0 + contrast : 1.0 / (1.0 - contrast);
    return 0.5 + ((col - 0.5) * vec3(k));
}
void main(){
    vec4 texColor = texture(image, tex); // init from texture
    vec3 col = tonecurve(texColor.rgb);  // apply tonecurve
    if (coloringMethod > 0){
        if (coloringMethod == 1){        // solid color
            col = colSingle;
        }else{
            float grey = brightness(col.rgb);
            if (coloringMethod == 2){    // two-tone-shading
                col = mix(colLow, colHigh, grey);
            }else{                       // external gradient
                col = texture(gradient, grey).rgb;
            }
        }
    }
    color = vec4(col, alpha * texColor.a);
}
```

7.4.3 Ebenentypen

Neben den gemeinsamen Parametern haben die unterschiedlichen Ebenentypen eigene Parameter. Diese und ihre Arbeitsweise sollen nun beschrieben werden. Die eingesetzten Verfahrensweisen werden im Abschnitt 6.4 behandelt, ihre Umsetzung im Abschnitt 7.5 besprochen.

Beleuchtungsebene

Bei der Beleuchtung wird ein Helligkeitswert bestimmt, der in alle drei Farbkanäle geschrieben wird. Als Ergebnis erhält man ein Graustufenbild, das über die Färbungsoptionen einer Ebene, beschrieben in Abschnitt 7.4.2, eingefärbt werden kann.

In der Beleuchtungsebene wird auch die Position und Ausrichtung der Lichtquelle verwaltet. Daher muss sie Maus- und Tastatureingaben verarbeiten können.

Ambient-Occlusion

Da Ambient-Occlusion-Verfahren eine Abdunklung berechnen, wird der berechnete Wert, die Intensität, in den α-Kanal geschrieben und als Farbe Schwarz $(0, 0, 0)$ gesetzt. So wird die Abdunklung mittels *Alpha-Blending* in das Gesamtbild integriert.

Soll nun die Farbe der Abdunklung über die Färbungsoptionen verändern, erfolgt dies nicht auf Basis der Helligkeit, die immer 0 beträgt, sondern mit dem Wert des α-Kanals. Dies erreicht man durch einen speziellen Filter, der die Werte entsprechend umschreibt. Soll eingefärbt werden, wird der α-Wert in der RGB-Kanal geschrieben und $\alpha = 1$ gesetzt.

Konturlinien

In der Konturebene kann eine der in Abschnitt 7.5.2 beschriebenen Verfahren ausgewählt werden. Wurde der Bildfilter gewählt, kann noch die Eingabe sowie Methode der Schwellwertbestimmung ausgewählt werden. Die Ergebnisse sind wie bei einer *Ambient-Occlusion*-Ebene immer schwarz und die Intensität wird im α-Kanal gespeichert. Die Farbersetzung funktioniert nach dem selben Prinzip wie bei der Ambient-Occlusion-Ebene.

Regelmäßige und unregelmäßige Pünktelungen

Die Pünktelungsebenen haben als Eingabe eine Textur, die als Grundlage für die Pünktelung benutzt wird. Die Punkte besitzen eine einstellbare und für jede Ebene einheitliche Größe und Farbe.

Da das *Electrostatic-Halftoning*-Verfahren ein iteratives Verfahren ist, muss eine Start-, eine Stop- und eine Initialisierungsfunktion vorhanden sein.

Freihand- und Vektorzeichnen-Funktion

Die Textur einer Zeichnenebene wird mit schwarzer Farbe und einem α-Wert von $0,0$ initialisiert. Durch das additive Hinzufügen von gezeichneten Elementen werden die vom Nutzer bestimmten $RGB\alpha$-Werte geschrieben. Beim Löschen wird in entsprechender Weise der Initialisierungszustand eingetragen. Für die Zeichnenebenen kann die Farbe und Dicke von Malstift oder Vektorlinie ausgewählt werden.

Alle Freihandzeichnungsoperationen werden sofort ausgeführt und vermischen sich zu einem direkten Ergebnis auf der Textur. In der Vektorzeichnenebene lassen sich Instanzen von *Polylinie* und *Bézierkurve* anlegen oder gezielt löschen. Nur das aktiv ausgewählte Element wird editiert, wobei jedes Element individuell additiv oder subtraktiv mit der Textur kombiniert werden kann.

Details zur Umsetzung der Zeichnenfunktion sind in Abschnitt 6.5 beschrieben.

Filtergraphen

Der mit der Benutzeroberfläche erstellte Graph wird durch einen XML-String an den Renderer übergeben. In diesem stehen nicht nur die Namen, die eingesetzten Shader, sondern auch die Werte der Parameter und die Zusammenhangsliste des Graphen. Das GUI ist dafür verantwortlich, dass der übermittelte Filtergraph korrekt und auch auswertbar ist. Vom Graphen wird eine Prioritätsliste generiert und dabei werden die Filter in korrekter Reihenfolge gerendert.

Kombination

In einer Kombinationsebene lässt sich eine Liste von Ebenen verwalten, die in dieser Ebene additiv zusammengefasst sind. Alle vor dieser Ebene stehende Ebenen können hinzugefügt werden.

Textur

Die Texturebene ist eine reine Datenquelle, da sie keine Eingaben hat. In sie wird entweder aus einer anderen Ebene hineinkopiert oder eine extern geladene Textur angezeigt.

7.5 NPR-Verfahren

In diesem Abschnitt wird auf die Implementierung der NPR-Verfahren eingegangen, die in Abschnitt 6.4 vorgestellt wurden. Die Implementierung der Stipplingverfahren ist detailliert in der Diplomarbeit *Stipplingalgorithmen für paläontologische Fundstücke* [Kno12] beschrieben.

7.5.1 Beleuchtung

Die Beleuchtung mit der modifizierten Lambert'schen Beleuchtungsformel und der Formel von Oren-Nayar kann in einem einzigen Bildfilter umgesetzt werden.

Die Verfahren *Exaggerated Shading* [RBD06] und *Apparent Relief* [Ver08] sind als mehrstufige Bildfilter umgesetzt. Die *Apparent Relief* -Technik ist in Form von GLSL-Shadern auf der Webseite einer der Autoren, Pascal Barla, veröffentlicht[4] und lässt sich so direkt integrieren. Das Verfahren *Exaggerated Shading* benötigt mehrfach geglättete Normalen, die in einem Vorverarbeitungsschritt bei jeder Änderung des G-Buffers neu erzeugt werden müssen.

[4] www.labri.fr/perso/barla/blog/?p=23411, besucht: 07.12.2014

7.5.2 Kontur, Kanten & Outlines

Die Kontur ist durch die *Image-Enhancement*-Technik und durch die Bildfilter, beschrieben in Abschnitt 3.4.1, umgesetzt.

Die *Image-Enhancement*-Technik kommt dabei leicht modifiziert zum Einsatz. Im originalen Verfahren führen negative Differenzwerte zu einer Abdunklung und positive zu einer Aufhellung. Da eine Aufhellung nicht gewünscht ist, führen beide zu einer Abdunklung und der Nutzer kann deren Intensität getrennt steuern. So wird es möglich, die Kontur außerhalb und innerhalb eines Objektes getrennt zu regeln. Weiter lässt sich noch die Stärke der Glättung steuern. Konkret stellt man über diesen Parameter ein, wie oft ein 3×3 Gauß-Filter auf das Tiefenbild angewendet wird.

Die Methode der Kantendetektion in Normalen- und Tiefenbild erfolgt durch verschiedene Filterkerne: LaPlace, Sobel, Prewitt und Frei-Chen. Alle Filterkerne und die Intervallmethoden wurden in einem einzigen Shader zusammengefasst.

7.5.3 Unregelmäßige Pünktelung

Die unregelmäßige Pünktelung erfolgt mit Hilfe des *Wang-Tile-Blue-Noise*-Verfahrens. Auf der Webseite eines der Autoren, Johannes Kopf, ist der Quellcode veröffentlicht[5]. Leider ist der Quellcode speziell für Windows-Systeme ausgelegt, so dass dieser erst noch portiert werden musste, um plattformunabhängig zu kompilieren. Die Anzahl der maximal angezeigten Punkte kann über einen Helligkeitsregler eingestellt werden, da die Punkte hinsichtlich ihres Ranges gleich verteilt sind.

Das Kachelset wird in einem Vorproduktionsschritt erzeugt und bei dem Programmstart geladen. Die Punkte werden in einen *VBO* geladen, der normierte Rang wird in der z-Koordinate des Vertex gespeichert. Die Zielhelligkeit wird mit dem Rang des Vertex verglichen und so lässt sich entscheiden, ob der Punkt gezeigt wird. Die verbleibenden Punkte werden mittels *Transform-Feedback*-Technik in einen zweiten *VBO* geschrieben, der als Punktmenge gerendert werden kann.

Auswahl der Punkte

Für die Berechnung der zu zeichnenden Punktmenge gibt es zwei Möglichkeiten: über Vertex-Shader und CPU oder mit Geometry-Shader und *Transform-Feedback*. In beiden Methoden wird die Textur an der (x, y)-Position des Vertex ausgelesen und die Helligkeit mit dem Rang verglichen.

Wird der Vertex-Shader benutzt, so kann bei auszuschließenden Vertices die w-Komponente auf 0.0 gesetzt werden. Anschließend wird der Buffer zur CPU übertragen, ein entsprechend kleinerer Buffer mit akzeptierten Vertices erzeugt und dann wieder zur GPU hochgeladen.

[5] research.microsoft.com/en-us/um/people/kopf/blue_noise/, besucht: 22.06.2015

Der Geometry-Shader kann selbst Vertices vom Rendern ausschließen, indem für diese nicht die `EmitVertex()`-Methode aufgerufen wird. Aktiviert man *Transform-Feedback* und gibt das Ziel-VBO an, werden die vom Geometry-Shader akzeptierten Vertices direkt in ein neues VBO geschrieben.

In Tests hat sich allerdings herausgestellt, dass die Methode, die den Geometry-Shader benutzt, deutlich mehr Zeit benötigt [Kno12]. Dieses Verhalten wurde unabhängig von der vorhandenen Implementierung auch von anderen Nutzern in verschiedenen OpenGL-Anwenderforen bestätigt.

7.5.4 Regelmäßige Pünktelung

Punkte in gleichen Abständen lassen sich mit dem *Electrostatic Halftoning* rendern. Hier wurde auf das Verfahren auf den speziellen Einsatz angepasst. Sowohl das Sampling des Bildes mit einem Gitter, die Berechnung einer *Forcefield*-Textur als auch der abschließende *Jitter* Schritt entfallen. Dafür besitzt das umgesetzte Verfahren jedoch einen Regler, mit dem sich die Anzahl der angezeigten Punkte anpassen lassen.

Im originalen Verfahren wird die Abtastung mit einem Gitter durchgeführt, um die Zahl der Ladungsträger zu reduzieren. Statt aller Pixel des Bildes wird nur den Gitterpunkten eine Ladung zugewiesen. Die *Forcefield*-Textur speichert die summierte Anziehungskraft auf einen Partikel, die an einem bestimmten Pixel durch alle Gitterpunkte ausgeübt wird. Bei einem statischen Bild ändern sich die Ladungsträger bzw. die Gitterpunkte nicht und können durch eine Vorberechnung ausgeklammert werden.

Im vorliegenden Szenario muss die Abtastung mit einem Gitter entfallen, da alle Details des zu pünktelnden Bildes gezeigt werden müssen. Wie in Abschnitt 6.4.3 erörtert, ist die zu erwartende Zahl der Pixel ohnehin gering. In der Initialisierungsphase oder bei Änderung der Partikelanzahl ist die Ladung des Pixels zu aktualisieren. Die neu berechnete Ladung kann als Parameter dem OpenCL-Programm mitgegeben werden und wird je nach Helligkeit des Pixels entsprechend skaliert.

Ein Wegfall der *Forcefield*-Textur sorgt dafür, dass die Anziehungskräfte, die auf einen Partikel wirken, in jedem Schritt ausgewertet werden. Das Partikelsystem kann so dynamisch ausgewertet werden und die Anzahl der Ladungsträger ist flexibel. Daher können neue hinzugefügt werden, beispielsweise durch Einzeichnen mit der Zeichnenfunktion. Die Laufzeit des Verfahrens lässt sich insgesamt durch das Weglassen der ersten Schritte, die Abtastung und die Berechnung der *Forcefield*-Textur, verbessern.

Im zweiten Schritt werden die Punkte iterativ verteilt: Sie stoßen sich untereinander ab und werden von den Attraktoren, den Pixeln, angezogen. Dieser Schritt wird solange ausgeführt, bis der Benutzer das Verfahren anhält.

Die Änderungen verschlechtern zwar die Laufzeit, bringen dafür aber Details in das Bild und sorgen für eine dynamischere Auswertung, die für den interaktiven Einsatz wichtig ist.

7.6 Die Zeichnenfunktion

Die Zeichnenfunktion stellt ein wichtiges Instrument zur Erstellung einer wissenschaftlichen Illustration dar. Mit ihr kann die Illustration individualisiert werden und sie ermöglicht es dem Nutzer seine Vorstellung besser umzusetzen. Die Zeichnenfunktion gestattet es, von einem konkreten Fundstück oder 3D-Modell zu einem theoretischen Anschauungsobjekt hin zu abstrahieren.

Die Zeichnenfunktion wird vollständig auf der GPU realisiert. Würde man die CPU hinzuziehen, müsste man mehrere Bilder auf die CPU transferieren, dort die Veränderungen berechnen und schließlich die neue Textur wieder zurück zur GPU kopieren. Dieser Schritt wird durch die Verwendung des GPU-basierten Zeichenwerkzeugs obsolet.

Das Ziel der Zeichenoperation ist immer eine Textur, in der an bestimmten Stellen Farben gesetzt oder gelöscht werden. Je nach gewünschtem Modus (Bildraum oder Oberfläche) unterscheidet sich lediglich die Erfassung der Stelle, an der gezeichnet wird. Will man im Bildraum zeichnen, kann die Koordinate der Maus direkt umgewandelt werden. Ist das Zeichenziel die Oberfläche, so berechnet man die korrekte Position auf der Oberfläche. Das dahinter stehende Verfahren ist dabei immer gleich. Im Folgenden wird daher der komplexere Fall, das Zeichnen auf der Oberfläche, erklärt. Auf die Übertragung der Methodik auf den Vektorzeichner wird im folgenden Abschnitt eingegangen.

7.6.1 Zeichnen auf der Oberfläche

Auf der GPU lässt sich die Zeichnenfunktion umsetzen, da OpenGL-Buffer universell sind und sowohl als VBO als auch als Textur interpretiert werden können. Das gesamte Verfahren funktioniert mit zwei Render-Passes und benötigt eine temporäre *Markierungstextur*, die die Texturkoordinaten unter der Markierung enthält. Hier die einzelnen Schritte:

- Bestimmung des Bereichs, in den gezeichnet werden soll,
- Rendern der Texturkoordinaten des Modells in diesem Bereich in eine *Markierungstextur*,
- Interpretieren der Texturkoordinaten als VBO,
- Entfernen der Punkte, die nicht unter der Maus waren,
- Transformation der Punkte in Texturkoordinaten und
- Rendern der Punkte in die Oberflächentextur und Beachtung der gewünschten Kombinationsmethode (Addition, Subtraktion, . . .).

Das Verfahren wird in Abbildung 7.13 beispielhaft dargestellt. Beim gesamten Verfahren muss der aktuell angezeigte Ausschnitt, also Vergrößerung und Offset, beim Setzen der Mausposition berücksichtigt werden.

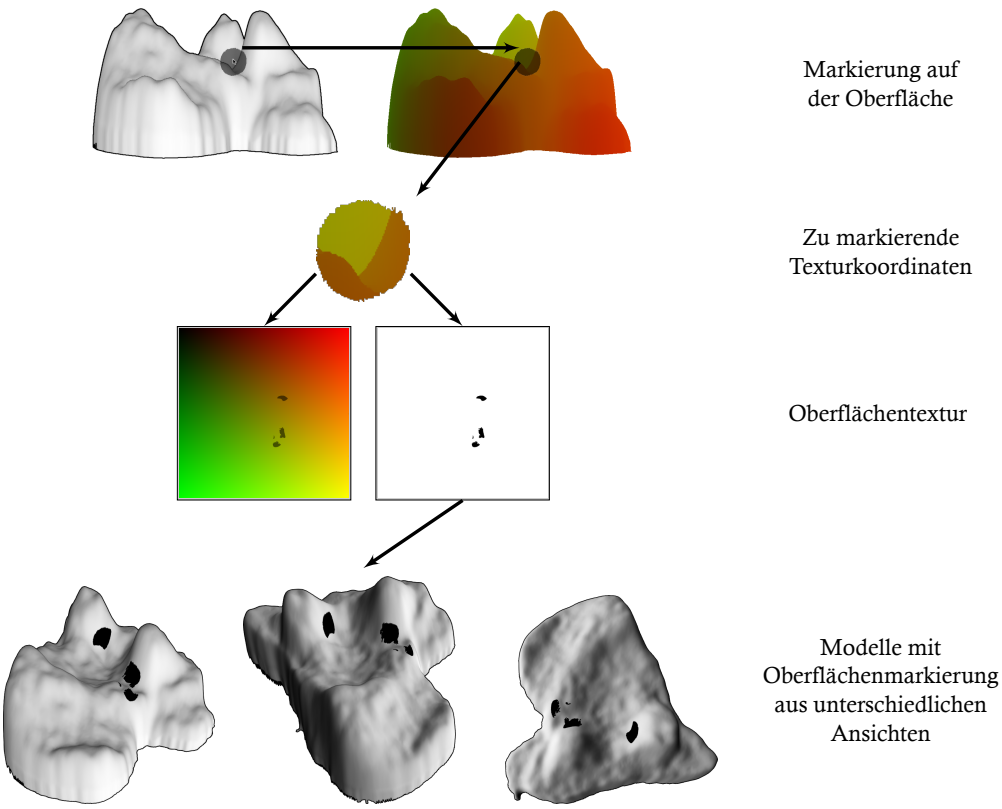


Abbildung 7.13: Ablauf der Oberflächenmarkierung in 3D-Ansicht

Schritt 1: Der Bereich unter der Maus

Mit dem ersten Shader rendert man die *Markierungstextur*, in der die Texturkoordinaten eingetragen sind und an der gezeichnet werden soll. Als Eingabe erhält der Shader die Position der Maus, die aktuellen Zoom-Parameter, die Dicke des Zeichnenwerkzeuges und den G-Buffer mit den Texturkoordinaten des 3D-Modells. Durch GLSL-Methoden kann für jedes Pixel die Bildschirmkoordinate und die aktuelle Auflösung ausgelesen werden.

Texturkoordinaten haben u- und v-Koordinaten, die im G-Buffer im roten und grünen Kanal abgelegt sind. Diese werden unverändert in die Ausgabe geschrieben. Der blaue und der α-Kanal wird auf den Wert 0.0 gesetzt.

Nun bringt man die Mausposition mit den aktuellen Zoom-Parametern in den korrekten Bereich innerhalb der G-Buffer. Befindet sich das aktuelle Pixel unter dem Mauscursor oder noch innerhalb des Zeichenwerkzeugs, wird der α-Wert auf 1.0 gesetzt.

Schritt 2: Transformation der Koordinaten

Im nächsten Schritt interpretiert man die *Markierungstextur* als VBO. Jedes Fragment wird als ein Vertex angenommen und die (r, g, b, α)-Werte als (x, y, z, w)-Wert betrachtet. Da alle Texel[6], die sich nicht unterhalb des Zeichenwerkzeugs befanden, einen α-Wert von 0.0 hatten, haben die entsprechenden Vertices nun einen w-Wert von 0.0 und können *weggeclippt* werden. Ihr w-Wert wird auf -2 gesetzt, so dass sie sich mit Sicherheit nicht im Einheitswürfel befinden.

Die verbleibenden Vertices haben einen (x, y)-Wert, der der Texturkoordinate entspricht, auf der markiert werden soll. Damit sie auch an der korrekten Position angezeigt werden, transformiert man sie im Vertex-Shader aus dem Bereich der Texturkoordinaten $[0; 1]^2$ in den Wertebereich des Einheitswürfels $[-1; 1]^3$.

Schritt 3: Eintragen der Markierung

Die Vertices befinden sich innerhalb der Textur nun an der korrekten Position. Im Fragment-Shader zeichnet man jetzt jeden Vertex nun mit der gewünschten Farbe. Der Shader kann in zwei Texturen schreiben:

1. eine Textur, die nur die aktuelle Markierung enthält und
2. eine neue Oberflächentextur, in der die aktuelle Änderung eingebracht wurde.

Die erste Textur lässt sich für eine Vorschaufunktion nutzen. Weil OpenGL nicht in die Textur schreiben kann, aus der auch gelesen wird, ist die alte Oberflächentextur noch unverändert vorhanden. Nur wenn die Zeichnenfunktion wirklich schreiben soll, beispielsweise wenn die Maustaste gedrückt wird, kann die alte Oberflächentextur verworfen und die neue weiter benutzt werden.

Auflösung der Oberflächentextur

Die Auflösung der Oberflächentextur ist kritisch für die Qualität der eingezeichneten Markierungen und Korrekturen. Sie muss ausreichend hoch sein, damit alle Details wiedergegeben werden können. In der Arbeit *Per-face Texture Mapping for Real-time Rendering* [MB11], kurz: *PTex*, ist ein Verfahren vorgestellt, das ab OpenGL 3.0 in Echtzeit funktioniert und als eine Art dynamisches LOD-SYstem für Modell-Texturen verstanden werden kann. Jedem Dreieck oder Viereck eines polygonalen Modells wird eine eigene Textur zugewiesen, die direkt mit baryzentrischen Koordinaten adressiert werden kann. Das *PTex*-Verfahren garantiert, dass die Texturen nahtlos angezeigt und beliebig bei Bedarf verfeinert werden können. Das Verfahren ist ohne größeren Aufwand in bestehende Rendersysteme integrierbar.

[6] Texel - Kunstwort in Anlehnung an Pixel: **Tex**tur**el**ement

7.6.2 Vektorzeichner

Mit dem Vektorzeichner lassen sich Polylinien und parametrische Kurven zeichnen. Innerhalb des Frameworks sind parametrische Kurven auf Bézierkurve vom Grad 3 mit 4 Kontrollpunkten umgesetzt. Der Vektorzeichner läuft teilweise auf der CPU, da dort die Kontrollpunkte verwaltet werden. Das Zeichnen der Linie und der Bézierkurve erfolgt vollständig auf der GPU, wie in Abschnitt 7.3.4 und Abschnitt 7.3.5 erörtert.

Kontrollpunkte

Beiden Vektortypen gemein sind Kontrollpunkte. Im Fall der Polylinien verläuft die Kurve zwischen den Punkten. Aus n Kontrollpunkten lassen sich $n-1$ Liniensegmente generieren. Bei parametrischen Kurven werden nur solche Kurvensegmente angezeigt, die 4 Kontrollpunkte haben. Die Kurven können allerdings in einer anknüpfenden Weise erzeugt werden: Der vierte Punkt ist der letzte Punkt der erste Kurve und der erste der nächsten Kurve.

Die Kontrollpunkte lassen sich noch nach der Platzierung verschieben und auch löschen. Das Einfügen von Punkten ist zum gegenwärtigen Zeitpunkt nicht implementiert, aber theoretisch leicht umsetzbar. In der Testphase fiel jedoch das Fehlen der Ergänzungsmöglichkeit nicht negativ auf.

Die Kontrollpunkte können mit der Maus verwaltet werden:

- Die Kontrollpunkte werden durch einen Mausklick platziert und gelöscht.
- Ein Kontrollpunkt kann mit der Maus verschoben werden.

Zur einfacheren Benutzung lassen sich die Kontrollpunkte farbig hervorheben. Der Index des ausgewählten Kontrollpunktes und der Index des Kontrollpunktes, der sich unter dem Mauscursor befindet, wird dem Shader mitgegeben. Die Farben für die Hervorhebungen lassen sich ebenfalls einstellen.

7.7 Implementierung der Benutzungsschnittstelle

Das GUI wird komplett mit Qt 4.8 [7] erstellt. Qt ist ein plattformübergreifendes Framework mit C++-Anbindung, das für die Erstellung der Benutzungsschnittstelle einen graphischen Editor enthält.

7.7.1 Render-Widget

Das Qt-Framework stellt ein `QGLWidget` bereit, das einen OpenGL-Kontext erzeugt. Allerdings war es weder unter Windows noch unter Mac OS X möglich, ein OpenGL-

[7] Quelle: http://qt.digia.com/, besucht: 19.01.2013

Kontext mit Profil *3.2 core* zu generieren[8], so dass diese Funktionalität reimplementiert werden musste.

Der OpenGL Renderer wird als eigener Thread implementiert. Da das GUI auch einen eigenen Thread besitzt, ist es wichtig, dass die Kommunikation *kontrolliert* verläuft, um folgende drei Voraussetzungen zu erfüllen:

1. Beim Abarbeiten der GUI-Events ist immer ein OpenGL-Kontext vorhanden.
2. Während der Rendermethode werden die OpenGL-Ressourcen nicht geändert. Dies kann beispielsweise eintreten, wenn der Befehl *'lade neue Textur'* ausgeführt werden soll, während die Textur gerade angezeigt wird.
3. Die GUI-Events interferieren nicht miteinander und werden sequentiell verarbeitet.

Aus diesem Grund wird die Kommunikation zwischen Renderer und GUI über *Message-Queues* realisiert, die mit einem *Mutex-Lock* geschützt sind. Innerhalb der *Mainloop* des Render-Threads wird abwechselnd die Queue *abgearbeitet* und die Rendermethode aufgerufen. Das gesamte Prinzip entspricht der klassischen *Game Main loop*[McS03]: Die Zustandsänderung (sofern durchzuführen) wechselt sich mit dem Rendern ab.

7.7.2 Abspeichern von Ergebnissen

Aus dem Programm heraus lassen sich fünf verschiedene Elemente speichern:

1. Das *finale Bild*.
2. *Alle Ebenen*, inklusive der nicht angezeigten. Im Dateinamen ist der Sichtbarkeitsstatus kodiert.
3. Das *Höhenbild der rekonstruierten Oberfläche* kann zusammen mit dem *Bild der totalen Schärfe* abgespeichert werden.
4. Die *rekonstruierte Oberfläche* lässt sich als polygonales 3D-Modell in allen von der verwendeten *Trimesh2* Bibliothek unterstützten Formaten `.PLY` und `.OBJ` sowie als *Collada*-Datei `.dae` abspeichern.
5. Als *Set* kann eine Gruppe von Dateien exportiert werden: alle G-Buffer, die aktuelle globale Einstellungsdatei und alle Paintertexturen. Mit einem Set lässt sich die gegenwärtige Ansicht auf einem anderen Rechner nachvollziehen. Es können auch noch Ebenen verändert werden, allerdings ist durch die statischen G-Buffer die Ansicht fix.

[8] http://qt-project.org/forums/viewthread/11031

7.8 Evaluation

Die Evaluation wurde im Rahmen der Bachelorarbeit *Evaluation eines Programmes zur Erstellung wissenschaftlicher Illustrationen* von Markus Strobel durchgeführt, deren Ergebnisse im Detail im Anhang A aufgeführt sind. Sie besteht aus zwei Teilen: einer qualitativen Evaluation der prototypischen Implementierung und einer quantitativen Evaluation durch eine Online-Umfrage.

Ziel der qualitativen Evaluation ist eine Betrachtung der Usability und eine Antwort auf die Frage, ob mit dem Programm Illustrationen erstellt werden können, die eine Basis eines konkreten Arbeitsauftrag für einen Illustrator sein können. Diese Frage kann mit einem „*Ja*“ beantwortet werden.

In der Online-Umfrage wurden die erstellten Bilder mit manuell gezeichneten und durch andere NPR-Programme erstellten Illustrationen verglichen, die alle das gleiche Modell aus der gleichen Perspektive darstellen. Die entscheidende Frage lautet „*Kann die Illustration veröffentlicht werden?*“. Von den 129 Teilnehmern aus verschiedensten Fachbereichen haben diese Frage für eine der mit dem Prototypen erstellten Illustrationen mehr als 61% mit einem „*Ja*“ geantwortet. Es ist hervorzuheben, dass 83% der teilnehmenden Doktoranden die Illustration für veröffentlichbar halten.

Eine vollständige Diskussion der Ergebnisse der Evaluation ist in Kapitel 8 zu finden.

7.9 Zusammenfassung

Das siebte Kapitel befasst sich mit der Umsetzung des im Kapitel 6 erarbeiteten Konzepts. Im Rahmen der Implementierung werden häufig genutzte Funktionen in die Bibliothek Scilly-Lib ausgelagert, die auch hilft, eine Abstraktionsebene von der *LowLevel*-API OpenGL herzustellen. Generell wird versucht, alle Methoden so weit als möglich auf die GPU zu portieren, so dass es nicht zu zeitaufwändigen CPU-GPU-Transfers kommt. Ein weiterer Vorteil von GPU-basierten Verfahren ist der zu erwartende Performanzgewinn durch das Nutzen der massiven Berechnungskraft moderner GPUs.

Die Rekonstruktion wird mit Hilfe von Shader-basierten Bildfiltern komplett auf der GPU umgesetzt. Die zusätzlichen Schwellwert-Parameter erlauben eine verbesserte Rekonstruktion der Oberfläche gegenüber aktuellen Verfahren.

In den folgenden Abschnitten wird die Implementierung der einzelnen Stufen der Illustrations-Pipeline besprochen. Es wird die notwendige G-Buffer Konfiguration umrissen und das rein GPU-basierte Rendern von Punkten und Linien mit flexibler Dicke vorgestellt. Hervorhebenswert ist das Verfahren zum Rendern vom parametrischen Kurven mit Hilfe von Geometry-Shadern komplett auf der GPU.

Daran anschließend wird die Verwaltung von Ebenen vorgestellt. Hervorzuheben ist die Komponente *Render-Manager*, die intern einen Abhängigkeitsgraph der Ebenen

erstellt. Durch diesen kann sichergestellt werden, dass Ebenen nur dann neu gerendert werden, wenn sich entweder eine der Eingabeebenen oder einer der Parameter der Ebene geändert hat. Der *Render-Manager* ist somit ein wichtiges Element um die Interaktivität des Gesamtsystems zu erreichen.

Im fünften Abschnitt des Kapitels wird die Implementierung der ausgewählten NPR-Algorithmen erörtert. Der Schwerpunkt liegt auf der Realisierung von Verfahren zur Erzeugung von Konturlinien und Pünktelungsverfahren.

Die Zeichnenfunktion, mit der die Deformation des Objektes und die Anpassungsmöglichkeit der Illustration umgesetzt wird, operiert ebenfalls komplett auf der GPU. Hier wird unterschieden zwischen Methoden, die auf der Oberfläche des Objektes arbeiten, und bildschirmbasierten Methoden. Die einzige Interaktion, die auf der CPU-Seite ausgeführt wird ist das Aktualisieren der Mausposition an die GPU.

In den letzten zwei Abschnitten wird die technische Implementierung der Benutzungsschnittstelle umrissen und zuletzt Ziele und Ergebnisse der Evaluation kurz vorgestellt. Es zeigte sich, dass Illustrationen, die mit dem Prototypen erstellt wurden nicht nur zur Besprechung eines konkreten Zeichnen-Auftrages geeignet sind sondern 61% der 129 Teilnehmer eine solche Illustration für veröffentlichbar im wissenschaftlichen Kontext halten.

Kapitel 8

Ergebnisse und Bewertung

Dieses Kapitel dient dazu, die Ergebnisse vorzustellen und zu bewerten. Zunächst werden Rekonstruktionen vorgestellt und mit Originalobjekten verglichen. Das Verfahren wir dann hinsichtlich Güte und Geschwindigkeit mit aktuellen Methoden verglichen. Im Anschluss daran wird der Prototyp mit Hinblick auf die Bereiche Performanz, GUI, Arbeitsfluss sowie Usability betrachtet. Weiter soll eine der erstellten wissenschaftlichen Illustrationen detailliert besprochen werden, die auch in der Onlineumfrage bewertet wurde. Im letzten Abschnitt werden die Ergebnisse der Onlineumfrage vorgestellt und erörtert.

8.1 Oberflächenrekonstruktion

Die Oberflächenrekonstruktion soll in ihrer Qualität mit den aktuellen Verfahren verglichen werden. Anschließend wird die Güte der Rekonstruktion durch einen Ground-Truth-Test und durch einen visuellen Vergleich mit einem Originalobjekt bewertet. Schließlich wird die Implementierung hinsichtlich der Laufzeit betrachtet.

8.1.1 Vergleich der Verfahren

Die Rekonstruktion wird mit den im Abschnitt 3.2.2 vorgestellten Verfahren verglichen. Aus drei Bilderstapeln wird zu diesem Zweck das erstellte Höhenbild der Verfahren in Abbildung 8.1 gegenübergestellt. Durch eine visuelle Inspektion können Rückschlüsse auf die Qualität der Rekonstruktion gezogen werden. Die Höhenbilder des hier vorgeschlagenen Verfahrens in der untersten Reihe haben das *ruhigste* Bild, das am plausibelsten zum Objekt passt. Insbesondere ist es frei von einem Rauschen, das die anderen Bilder kennzeichnet. Es wurde für Helicon Focus das plausibelste Bild der drei Methoden ausgewählt.

8.1.2 Vergleich mit Original

Um die Güte der Rekonstruktion zu beurteilen, bietet sich ein Vergleich der rekonstruierten Oberfläche mit dem Foto des Originalobjektes an. Dieses sollte aus einem anderen Blickwinkel als der Bilderstapel aufgenommen sein - idealerweise aus einer *orthogo-*

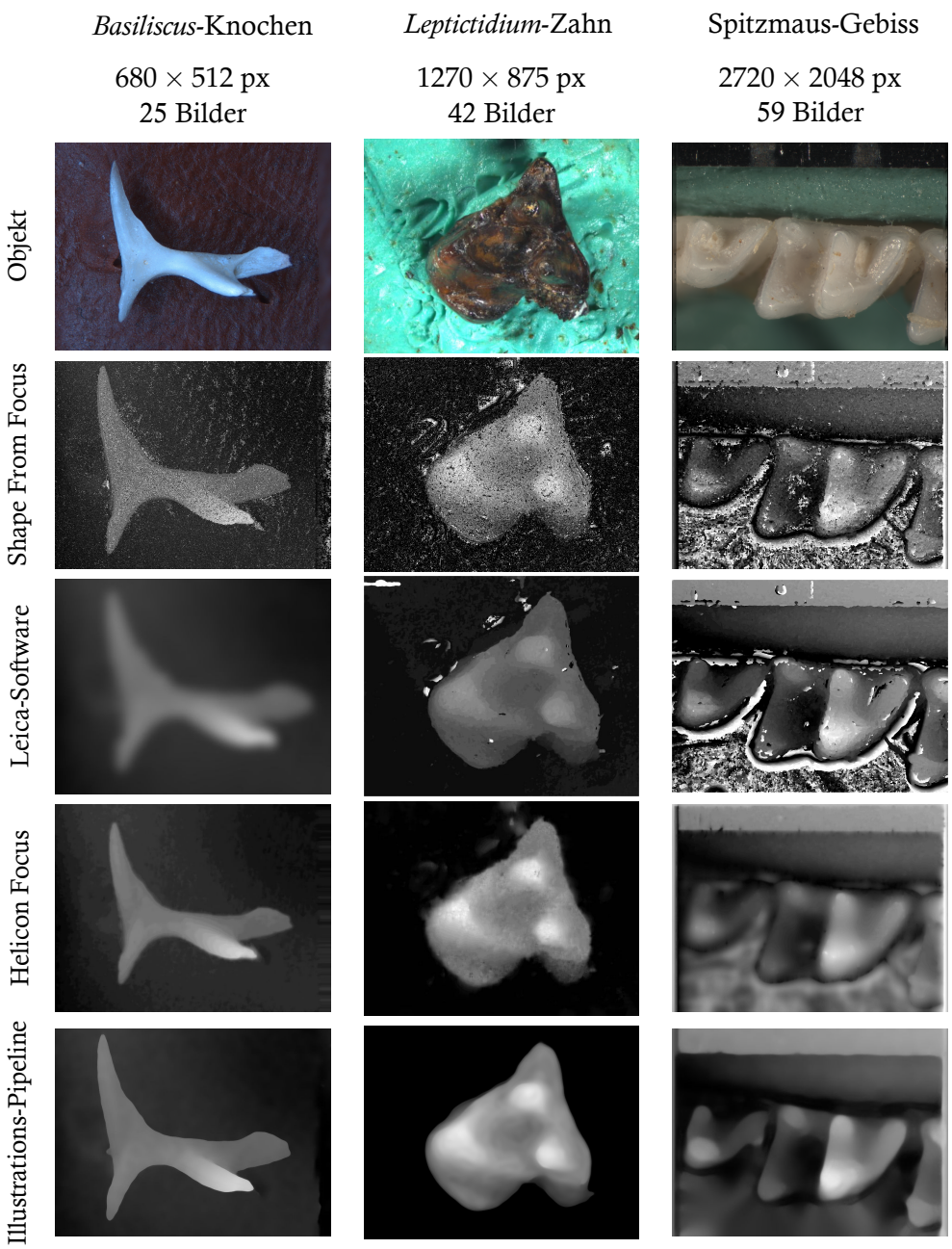


Abbildung 8.1: Mit den aktuellen und der Illustrations-Pipeline rekonstruierte Höhenbilder

nalen Position. Nur so lässt sich beurteilen, wie gut sich die Oberflächenkontur aus dem Stapel rekonstruieren ließ.

In Abbildung 8.2 ist links ein Foto des *Leptictidium*-Zahns zu sehen, das aus (fast) seitlicher Perspektive aufgenommen wurde. Rechts daneben ist die Rekonstruktion gezeigt, die zur besseren Visualisierung mit Hilfe des *Two-Tone-Shading* höhenspezifisch eingefärbt wurde. Anhand des Vergleichs wird deutlich, dass Form und Position der Krone gut rekonstruiert wurden.

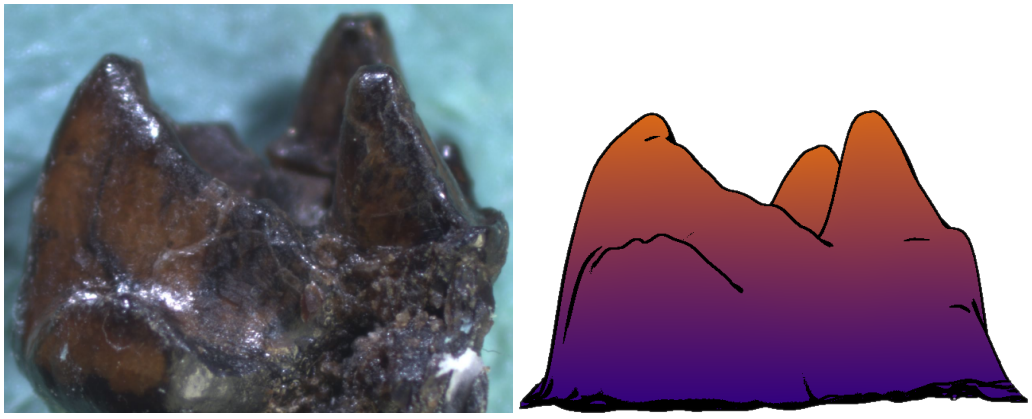


Abbildung 8.2: Aus gleicher Perspektive: Foto des *Leptictidium*-Zahns (links) und Illustration der rekonstruierten Oberfläche (rechts), aus [SH11]

8.1.3 Ground-Truth-Vergleich

Um ein quantitatives Ergebnis für die Rekonstruktionsgüte zu erhalten, wird ein synthetischer Bilderstapel eingesetzt, der mit der Open-Source Rendersoftware *Blender*[1] erzeugt wurde. Mit dieser ist es leicht, das Tiefenbild der Kamera herauszuspeichern, das zu einem perfekten Höhenbild transformiert werden kann. Zunächst wird das Bild invertiert, so dass von der Kamera entferntere Bereiche dunkler werden und anschließend wird das Bild normalisiert.

Mit einer Animation der Kameraposition kann die Situation eines Fixfokus-Mikroskops nachgestellt werden, bei der die Kamera sich dem Objekt annähert und so die Bilder des Bilderstapels rendert. Die Kamera muss dazu entsprechend eingestellt werden, so dass die Schärfeebene der Kamera das Objekt abfährt und eine entsprechende Tiefenunschärfe hat.

Für den Ground-Truth-Vergleich wurde eine Szene ausgewählt, die typisch für den Anwendungsfall ist: ein einzelnes Objekt ist vor einem andersfarbigen Hintergrund zu sehen. Als Objekt wurde das Blender Model *Suzanne* ausgewählt[2], das als Testmodell in Blender genutzt wird und ähnliche Charakteristiken wie die Artefakte aufweist. Das Bild totaler Schärfe, das mit Blender erstellt wurde, ist in Abbildung 8.3 zusammen mit dem Höhenbild und zwei Bildern des Stapels dargestellt. Der gesamte Stapel hat 20 Bilder, die jeweils aus 800×600 Pixeln bestehen.

Die einzelnen Verfahren sollen nun aus den Bilderstapeln die Oberfläche rekonstruieren und das erzeugte Höhenbild kann mit dem exportieren perfekten Höhenbild verglichen werden. Aufgrund der schlechten Ergebnisse wird auf eine Nennung der Leica-Software an dieser Stelle verzichtet. Die Bilder sind in Abbildung 8.4 gezeigt. Die Helicon Focus Software bietet drei verschiedene Methoden an. Weil mit Methode *C* keine

[1] www.blender.org, besucht: 14.12.2014
[2] erstellt von Willem-Paul van Overbruggen

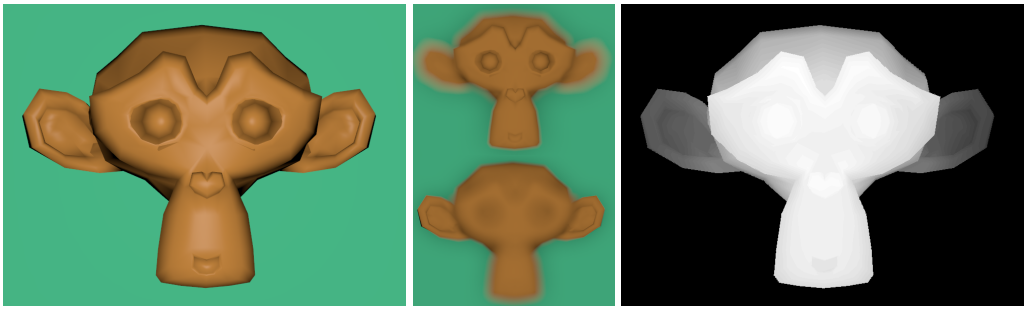


Abbildung 8.3: Gerenderte Bilder für den Ground-Truth-Vergleich: Bild totaler Schärfe (links), Bild 3 & 19 des Stapels (Mitte) und Höhenbild (rechts)

sinnvollen Ergebnisse erzeugt werden konnten, werden nur Bilder der Methoden *A* und *B* gezeigt. Da der Hintergrund aufgrund seiner Farbe leicht erkannt werden kann, wurde er in einem zweiten Testlauf maskiert und vom Vergleich ausgeschlossen, so dass nur die Rekonstruktion des Objektes bewertet wird.

Der Fehlerwert für eine Rekonstruktion wird über die mittlere Summe der Wurzel aus dem quadratischen Fehler per Pixel von idealem und rekonstruierten Höhenbild bestimmt:

$$\text{fehler}(\text{HB}_{\text{ideal}}, \text{HB}_{\text{reco}}) = \frac{1}{|\text{Pixel}|} \sum_{p \in \text{Pixel}} \sqrt{(\text{HB}_{\text{ideal}}(p) - \text{HB}_{\text{reco}}(p))^2}$$

Die Fehler sind in Tabelle 8.1 aufgeführt.

Methode	Fehler unmaskiert	Fehler maskiert
Eigenes Verfahren	$\mathbf{1.961 \times 10^{-4}}$	$\mathbf{4.285 \times 10^{-5}}$
Helicon Focus Methode B	2.064×10^{-4}	9.413×10^{-5}
Shape From Focus	2.832×10^{-4}	1.320×10^{-4}
Helicon Focus Methode A	2.464×10^{-4}	1.335×10^{-4}

Tabelle 8.1: Mittlerer quadratischer Fehler der rekonstruierten Höhenbilder

Es ist zu konstatieren, dass das berechnete Höhenbild des hier vorgestellten Verfahrens den geringsten Fehler und die größte Ähnlichkeit mit dem originalen Höhenbild hat. Insbesondere der Einsatz einer Maske hilft, die Rekonstruktion zielgerichtet auf das Objekt zu beschränken und Interpolationsfehler zu vermeiden. In den hier gezeigten Bildern kann eine Maske sinnvoll gefunden und angewendet werden. Der Hintergrund hat kein Muster und somit keine Energie: Er wird per Schwellwert als Lücke markiert. Das Verfahren füllt somit den Bereich per Interpolation von den zuletzt erkannten Werten auf.

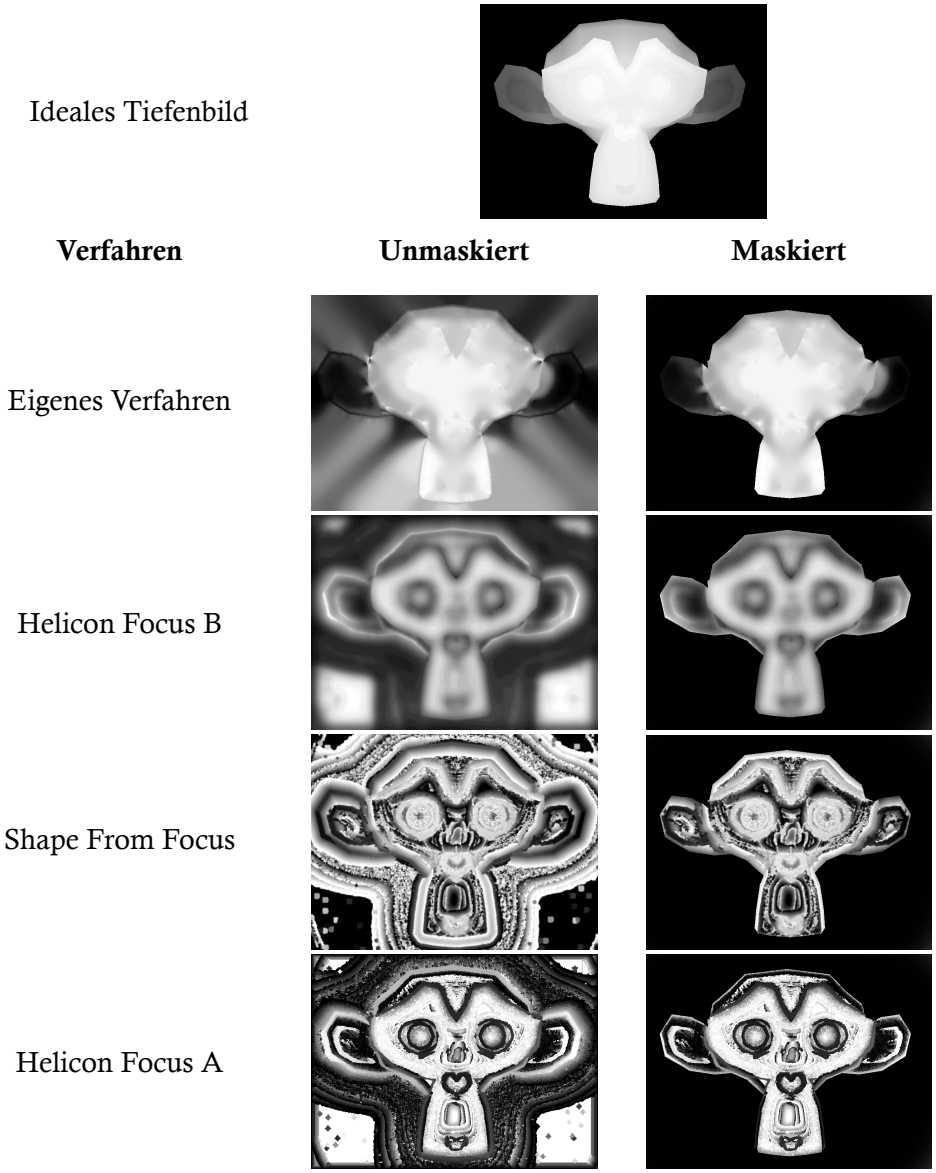


Abbildung 8.4: Ground-Truth-Vergleich: erzeugte Höhenbilder der Verfahren

8.1.4 Benchmarks

Die folgenden Benchmarks stammen aus der Arbeit *3D-Oberflächen-Rekonstruktion und plastisches Rendern aus Bilderserien* [SH10], in der das GPU-basierte Verfahren veröffentlicht wurde. Es wird mit einer auf der CPU arbeitenden Variante verglichen, die aus einer identischen Filterkette aufgebaut ist. Mit beiden Verfahren wurde aus mehreren Bilderstapeln die Oberfläche rekonstruiert, die in der Arbeit gezeigt und verwendet wird. Der Bilderstapel mit 30 Bildern zeigt einen *Ogmophis*-Vertebra und ist in 3 Auflösungen vorhanden: 2720×2048 Pixel, 1360×1024 Pixel und 680×512 Pixel. Die Laufzeiten der Verfahren sind in Tabelle 8.1.4 in Sekunden angegeben. Der mit 59 Bilder fast doppelt so große Bilderstapel eines Spitzmaus-Gebisses lag in zwei Auflösungen vor:

1360 × 1024 Pixel und 680 × 512 Pixel und die Berechnungsdauer der Rekonstruktion ist in Tabelle 8.1.4 gezeigt. Der Testrechner besitzt eine Intel i5 CPU mit 3,47GHz und verfügt über 8GB Arbeitsspeicher. Die Skalierbarkeit des Verfahrens wird durch 3 unterschiedliche Grafikkarten ermittelt: eine 2 Jahre alte NVidia GT 750M, eine 4 Jahre alte NVidia GTX 260 und eine 8 Jahre alte NVidia 8800 GTX. Im Vergleich dazu sind die Laufzeiten der beiden kommerziellen verfahren mit in der Tabelle eingetragen. Helicon Focus berechnet in einem einzigen Schritt sowohl Höhenbild als auch das Bild totaler Schärfe, daher ist in beiden Spalten der gleiche Wert angegeben.

Verarbeitungsdauer 30 Bilder	2720x2048 px		1360x1024px		680x512 px	
	HB	BTS	HB	BTS	HB	BTS
CPU	373.123	12.605	90.855	3.073	22.628	0.756
8800 GTX	32.909	27.051	4.002	4.080	0.734	0.741
GTX 260	14.913	9.813	3.349	3.713	0.593	0.570
GT 750M	8.497	6.228	3.317	2.119	0.408	0.393
Helicon Focus	16	(16)	5	(5)	3	(3)
Leica	48	20	18	7	8	5
HB = Höhenbild; BTS = Bild totaler Schärfe; Angabe in [sec]						

Tabelle 8.2: Verarbeitungsdauer der 30-er Bilderserie eines *Ogmophis*-Vertebra

Verarbeitungsdauer 59 Bilder	1360x1024px		680x512 px	
	HB	BTS	HB	BTS
CPU	176.047	5.897	43.672	1.599
8800 GTX	7.402	8.252	1.935	1.919
GTX 260	5.499	6.193	1.521	1.583
GT 750M	3.989	1.161	1.429	1.161
Helicon Focus	12	(12)	4	(4)
Leica	34	15	11	7
HB = Höhenbild; BTS = Bild totaler Schärfe; Angabe in [sec]				

Tabelle 8.3: Verarbeitungsdauer der 59-er Bilderserie eines Spitzmaus-Gebisses

Die Berechnung des Höhenbildes erfolgt in allen Fällen deutlich schneller auf der GPU. Selbst mit einer 8 Jahre alte GPU der Mittelklasse, die NVidia 8800 GTX GPU, lässt sich ein Geschwindigkeitszuwachs um den Faktor 20 erzielen. Bei der Erstellung des Bildes mit totaler Schärfe liegen CPU und die 3 Jahre alte GPU etwa gleichauf. Da hier einfache Shader eingesetzt werden, ist vermutlich der Bildertransfer von CPU zu GPU der *Flaschenhals* des Prozesses und der Grund, warum die neuere Karte schneller abschneidet. Obwohl bei dem CPU-basierten Verfahren dieser Schritt entfällt, ist es nicht schneller als die GPU, was als weiteres Argument für die Leistungsfähigkeit der GPU als Bildfilter angesehen werden kann. Das hier entworfene und umgesetzte Verfahren ist auch im Vergleich zum schnelleren der beiden kommerziellen Verfahren in

der Rekonstruktion schneller. Allerdings ist der Vergleich nicht ganz fair, da die Helicon Focus Software gleichzeitig Höhenbild und das Bild der totalen Schärfe erstellt. Diese Optimierung ist im hier vorgestellten Verfahren nicht möglich, da der Schwellwert, der das finale Höhenbild stark beeinflusst, szenenabhängig angepasst werden kann. Da sich erst mit diesem das Bild totaler Schärfe berechnen lässt, ist der Rekonstruktionsprozess zweistufig und interaktiv. Hinzu kommt, dass es sich in unseren Tests als vorteilhaft herausgestellt hat, wenn das Höhenbild direkt in einer 3D-Vorschau als 3D-Modell zu sehen ist. Hierfür wirkt das Bild der Totalen Schärfe als Oberflächentextur durch seine Details eher hinderlich. Im Prototypen wird daher ein anpassbares *Two-Tone-Shading* eingesetzt um so die Struktur besser hervorzuheben.

Es ist auch klar ersichtlich, dass das Verfahren gut skaliert. Die neuste Grafikkarte im Test schneidet am besten ab. Hinzu kommt, dass in allen Benchmarks des hier entworfenen Verfahrens die Bilder für jeden Verarbeitungsschritt komplett geladen wurden. Im praktischen Einsatz muss dies nur einmal durchgeführt werden und die Bilder können komplett im GPU-RAM gehalten werden.

8.2 Der Prototyp

In diesem Abschnitt wird zuerst die Performanz des Prototypen besprochen. Dann wird das GUI durch mehrere Screenshots vorgestellt. Anschließend sollen die evaluierten Erfahrungen und Meinungen aus dem Anwendertest des Prototyps analysiert und bewertet werden. Die Ergebnisse werden im folgenden Abschnitt gezeigt.

8.2.1 Performanz der Visualisierungs-Pipeline

Die Analyse der Performanz der Illustrations-Pipeline ist ein wichtiges Thema für ein Verfahren, das den Anspruch erhebt *interaktiv* abzulaufen. Allerdings ist es auf Grund der Flexibilität und der daraus resultierenden Komplexität des Rendervorgangs nicht universell möglich, eine konkrete Laufzeit für einen *Frame* anzugeben. Die erstellte wissenschaftliche Illustration setzt sich aus mehreren (kombinierten) Ebenen zusammen. Die resultierende Renderzeit hängt daher von drei Faktoren ab:

1. Die Auflösung der wissenschaftlichen Illustration.
2. Die Zeit, die benötigt wird, das Objekt in die G-Buffer zu rendern.
3. Die individuelle Renderzeit jeder instanziierten Ebene, die abhängig von dem Ebenentyp und den Parametern ist.

Der erste Faktor ist ein fixer Multiplikator, der sich auf die beiden anderen Faktoren auswirkt. Der zweite Aspekt hängt vom Objekt selbst ab, ist im Kern aber ein einziger Renderschritt (vergleiche 7.3).

Der letzte Punkt wird durch den in Abschnitt 7.4.1 umgesetzten Ebenen-Manager beeinflusst. Er sorgt dafür, dass eine Ebene nur dann gerendert wird, wenn dies notwendig ist. Das bedeutet, dass die Laufzeit stark von den gewählten Kombinationspfaden der Ebenen abhängt. Ändert man einen zentralen Knoten (beispielsweise das 3D-Modell) so müssen entsprechend viele Ebenen neu gerendert werden und die Renderzeit ist entsprechend lang. Die Laufzeit hängt somit maßgeblich von den verwendeten Ebenentypen und deren Parametern ab.

Die Implementierung der im Abschnitt 7.4 vorgestellten Ebenentypen werden in den Abschnitten 7.5 und 7.6 besprochen. Es wurden dabei nur solche Verfahren ausgewählt, die für den Realtime-Rendering Einsatz konzipiert wurden. Weil die meisten Techniken auf Bildfiltern basieren, können die Ebenen sehr schnell von aktuellen GPUs gerendert werden. Eine Ausnahme bilden die Pünktelungsebenen und Zeichnenfunktionen, die im Folgenden detailliert besprochen werden:

- Die unregelmäßige Pünktelung wird durch *Recursive Wang Tiles* als Punktwolke mit $\sim$ 100.000 Punkten gerendert. Der Vertex-Shader sampelt pro Punkt die Eingabetextur und der Fragment-Shader rendert prozedural generierte Punkte. Beide Shader sind wenig komplex und die Vertexmenge nicht groß. Das Verfahren ist Echtzeit-fähig.

- Die regelmäßige Pünktelung ist eine sehr rechenaufwändige Ebene, die sehr stark von der gewählten Punkteanzahl abhängt.

- Die Freihandzeichnen-Funktion im Bildraum basiert auf primitiven Shadern. Das Zeichnen auf der Objektoberfläche benötigt zwar mehrere interne Renderschritte, die allerdings alle wenig komplex sind. Das Verfahren ist Echtzeit-fähig.

- Das Ergebnis der Vektorzeichnen-Funktion sind dicke Linien, deren Segment aus vier Vertices bestehen. Da die eingesetzten Shader simpel sind, ist die gesamte Ebene wenig komplex. Das Verfahren ist Echtzeit-fähig.

Die wissenschaftlichen Illustrationen des Torsos und des Schädels, die für die Online-Umfrage erstellt wurden, sind mit einer Auflösung von 2048 $\times$ 2048 Pixel gerendert worden. Die Framerate lag während der Erstellung auf dem in Abschnitt 8.1.4 vorgestellten Rechner mit *NVidia GTX 260* Grafikkarte mit deaktivierten VSync stets oberhalb von 30 Frames/Sekunde.

8.2.2 GUI

Die Benutzungsschnittstelle des Prototypen, die sich an den Konzeptionsskizzen aus Kapitel 6.6 orientiert, wird durch verschiedene ausgewählte Screenshots vorgestellt.

Haupt- und Unterfenster

Das Hauptfenster wird in Abbildung 8.5 gezeigt. Über die Menüleiste kann auf die wichtigsten Funktionen direkt zugegriffen werden. In das Hauptfenster sind auf der linken Seite auch drei Unterfenster angedockt: die Statusinformationen, die Kameraparameter und der Ebenenmanager.

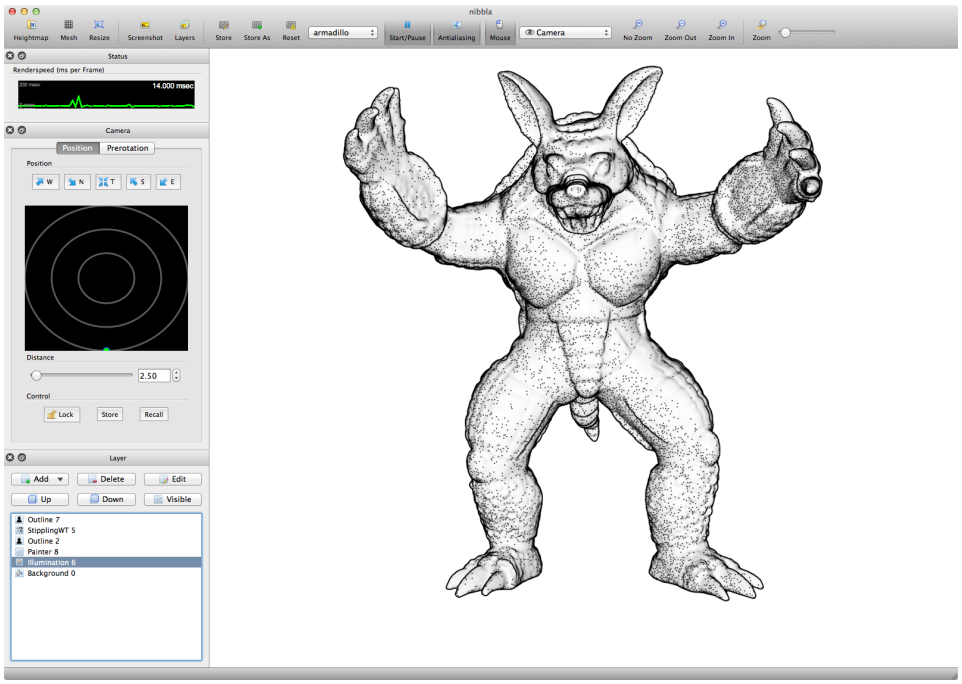


Abbildung 8.5: Das Hauptfenster des Prototyps

Ebenen

Die Eigenschaften einer Ebene lassen sich in Dialogen editieren. Der Dialog für Konturlinien in Abbildung 8.6 links soll an dieser Stelle beispielhaft gezeigt werden. Die Ebenen lassen sich mit dem Ebenenverwaltungsfenster organisieren, in Abbildung 8.6 rechts gezeigt.

8.2.3 Anwendertest

Mit dem Prototypen wurde der Anwendertest von drei geladenen Anwender/innen durchgeführt, die mit Illustrationen in unterschiedlicher Weise beruflich arbeiten müssen: die ausgebildete Zeichnerin *Frau Juliane Eberhardt*, die Dozentin der Medizin *Frau Dr. G. Klauer* mit ihrer Spezialisierung auf Anatomie und der Paläontologe *Herr Dr. Thomas Lehmann*. Die Tester/innen hatten die Aufgabe, mit dem Prototypen und einem vorgegebenen Schädelmodell eine wissenschaftliche Illustration anzufertigen. Die Ergebnisse sind den in Abbildung 8.9 dargestellten Illustrationen zu entnehmen. Da

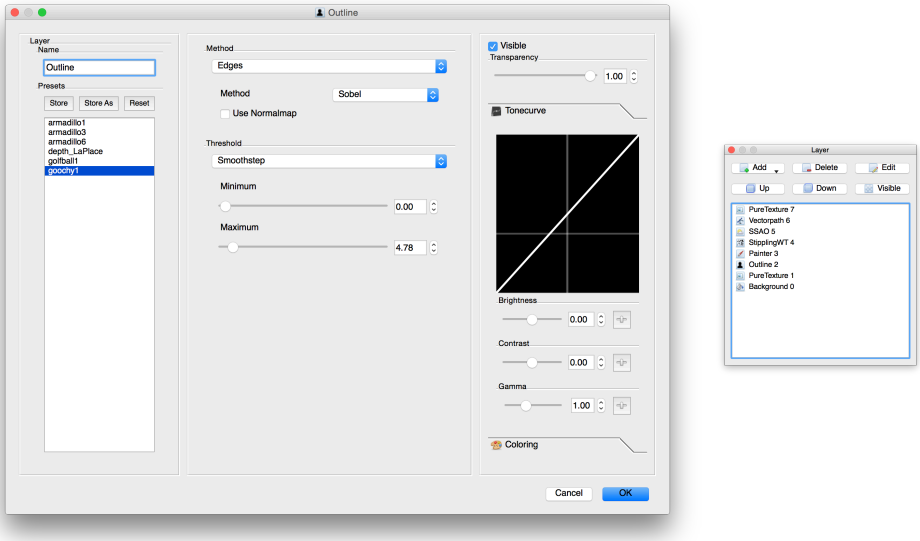


Abbildung 8.6: Der Ebenendialog mit den Eigenschaften einer Konturebene (links) und das Ebenenverwaltungsfenster (rechts)

den Probanden Programm und Bedienkonzept im Vorfeld nicht vorgestellt wurde, fand der Test in der Form des *Coachings* statt. Der Programmautor war beim Test anwesend und unterstützte die Probanden bei der Bedienung des Programms. Die Einweisung ermöglichte die Umsetzung der von den Testpersonen gewünschten Vorstellungen. In dem anschließenden Gespräch zeigte sich, dass hauptsächlich Usability-Probleme erörtert wurden, die Ergebnisse wurden nur kurz qualitativ bewertet. Das Feedback war jedoch sehr positiv.

Eine von der Probandin *Eberhardt* als erste Testperson gewählte Vorgabe ermöglichte es, dass alle Testpersonen ihre Illustration aus der gleichen Perspektive erstellen konnten. Zusätzlich gab der Testleiter die Empfehlung, zunächst das Licht einzustellen, da dies als Basis für die Pünktelung und die Oberflächengestaltung sein soll.

Testerinnen und Tester hielten sich an die Reihenfolge, haben aber unterschiedliche Lichtpositionen und Parameter ausgewählt, um die ihrer Ansicht nach relevanten Teile des Objektes entsprechend exponiert darzustellen. Zur Markierung der Kontur benutzte der Paläontologe in seiner Illustration Linien, während die beiden anderen Probanden Punkte einsetzten. Alle drei Tester/innen machten von der Zeichnenfunktion Gebrauch, wobei sie verschiedene Operationen durchführten:

- Die Zeichnerin hat die Pünktelung verändert, indem sie gezielt Punkte in einem hellem Bereich des Schädels hinzugefügt hat.

- Der Paläontologe passte die Beleuchtung der Augenbrauen an, damit diese markanter erscheinen.

- Die Medizinerin dunkelte neben der Beleuchtung der Augenbrauen auch die Stelle des Schädels ab, durch die der Sehnerv verläuft.

Auswertung

In dem Test des Programms zeigte sich, dass die identifizierten Usability-Probleme hauptsächlich die Anordnung von Bedienelementen betrafen; hier vor allem die Darstellung der Reihenfolge der Ebenen. Weitere Probleme betrafen noch nicht implementierte Komfortfunktionen, wie etwas das Fehlen einer Undo-Funktion im Ebenendialog. Die gefundenen Probleme dieser Art wurden inzwischen behoben und sind in die hier erörterte Umsetzung eingeflossen.

Für die Weiterentwicklung des Programms erwiesen sich die Aussagen zur Benutzung des Programms von großer Relevanz, insbesondere jene, die auf den Einsatz und die Verwertbarkeit der zugrundeliegenden Illustrations-Pipeline schließen lassen. Als wichtiger Aspekt ist folgende Aussage der Biologin zu werten, die direkt eine der Hypothesen adressiert:

> *Das ist jetzt eine gute Grundlage, um eine Illustration mit einem Zeichner zu besprechen. Die wesentlichen Elemente sind erkennbar und die Korrekturen durchgeführt. Das, was gezeichnet werden soll, ist hier zu sehen.*

Die im Testverfahren involvierte Zeichnerin war mit der Platzierung der Punkte unzufrieden. In den hellen Bereichen wurden ihr zu wenige Punkte platziert. Eine Anpassung durch die Justierung der Tonkurve schafft ihr zu wenig Abhilfe. Die Probandin war aus ästhetischen Gründen nicht völlig zufrieden. Die Kritik bezieht sich allerdings nur auf den implementierten Automatismus, da durch die Zeichnenfunktion die Punkteplatzierung auch manuell angepasst werden kann.

Der am Test beteiligte Wissenschaftler der Paläontologie, der selbst beruflich nicht zeichnet, bewertete das vorgestellte Programm grundsätzlich sehr positiv. Allerdings gab er an, dass ihm eine gewisse Einarbeitungszeit in das Programm gefehlt habe, um es noch besser anwenden zu können:

> *Das Ergebnis ist schon sehr gut, allerdings muss ich mal einen ganzen Tag oder eine Woche mit dem Programm verbringen, um sagen zu können, was es wirklich alles kann.*

Für die am Test beteiligte Medizinerin erwies sich das Programm als handhabbar. Sie erstellte zudem noch die Illustration eines Torso-Modells. Alle Bilder wurden für die Onlineevaluation verwendet, die in Abschnitt 8.4 vorgestellt wird.

Gruppe	Funktion	Ebene	Finalbild
A	Kontur	**1** - Image Enhancement	$1 \rightarrow \alpha$
B	Kontur	**2** - Kantenfilter auf Normalenbild	
		3 - Freihandzeichner von **2**	$\alpha + 3 \rightarrow \beta$
C	Oberfläche	**4** - Screen-Space-Ambient-Occolusion	$\beta + 4 \rightarrow \gamma$
D	Pünktelung	**5** - Beleuchtung: Modifizierter Lambert	
		6 - Freihandzeichner von **5**	
		7 - Pünktelung: WangTile von **6**	$\gamma + 7 \rightarrow \delta$
E	Korrektur	**8** - Freihandzeichner	$\delta + 8 \rightarrow \epsilon$
			$\epsilon \rightarrow \omega$

Tabelle 8.4: Ebenen der wissenschaftlichen Illustration des Torsos (Abbildung 8.7)

8.3 Wissenschaftliche Illustrationen

In diesem Abschnitt wird eine mit dem Programm erstellte Illustration vorgestellt und ihre Zusammensetzung, also Wahl der Ebenen und deren Komposition, analysiert.

Das Modell des Torsos wurde aus zwei Gründen ausgewählt. Einerseits wird es in der Studie von Isenberg et al. [Ise06] verwendet. Es existieren somit sowohl handgezeichnete als auch computer-generierte Illustrationen, die in der Bewertung des Verfahrens für einen Vergleich herangezogen werden sollen. Gleichzeitig sind Illustrationen von diesem Modell durch seinen medizinisch-anatomischen Hintergrund von Medizinern gut zu bewerten. Es lässt sich daher auch benutzen, um die Fragestellung zu beantworten, ob das Verfahren auch außerhalb der Paläontologie einsetzbar ist. Das Modell weist auf beiden Beckenknochen einen Bereich mit falschen Normalen auf, die neben einer lokal *falschen* Beleuchtung auch zu *falsch* erkannten Kanten führen.

Die gesamte Illustration besteht aus 8 Ebenen, die sich in 5 Funktionsgruppen einteilen lassen und in Tabelle 8.4 erklärt sowie in Abbildung 8.7 dargestellt werden. Die Abbildung wurde vom Autor erstellt, nachdem dieser beobachtet hatte, auf welche Kriterien die Experten bei der Erstellung Wert gelegt hatten. Die Bilder α bis ϵ zeigen das jeweils aktuelle Kompositionsbild. Es entsteht durch schrittweise Komposition der Gruppenbilder. Auf die finale Komposition ϵ wird noch ein Antialiasing-Filter angewendet ω.

Die erste Zeichnenebene **3** korrigiert die durch die falsche Normalen entstandene Konturkanten, die durch eine Kantendetektion im Normalenbild erzeugt werden. Die Beleuchtung wird mit der Zeichnenebene **6** angepasst. Neben der Korrektur des Beleuchtungsfehlers durch die falschen Normalen lässt sich der Helligkeitsverlauf auf dem Beckenknochen leicht anpassen, so dass die Struktur leicht hervorgehoben ist. Mit der dritten Zeichnenebene **8** werden verschiedene Anpassungen zusammen durchgeführt. Dabei wird die Kontur auf ein gleichmäßiges Niveau gebracht. Zusätzlich wurden verschiedene Punkte auf dem Schultergelenk hinzugefügt. Ein deutlicher Eingriff ist auf

Abbildung 8.7: Alle Ebenen der wissenschaftlichen Illustration des Torsos

dem rechten Beckenknochen unten zu erkennen. Hier wurde ein Loch zusätzlich eingefügt. Dieses Element lässt sich so aus der Perspektive nicht erkennen, aber es sollte bewusst angedeutet werden.

8.4 Evaluation durch Onlineumfrage

Die Onlineumfrage führte Markus Strobel im Rahmen der Bachelorarbeit *Evaluation eines Programms zur Erstellung wissenschaftlicher Illustrationen* durch. Die Umfrage und die Ergebnisse sind detailliert im Anhang A aufgeführt. An dieser Stelle wird die Onlineumfrage zur Evaluation der mit dem Programm erstellten Illustrationen daher nur kurz

vorgestellt. Es werden die Ergebnisse kurz aufgegriffen, analysiert und bewertet.

8.4.1 Ziel, Inhalt und Teilnehmer

Das Ziel der Umfrage galt der Antwort auf die Frage, ob die mit dem Prototypen generierten wissenschaftlichen Illustrationen von der Community akzeptiert werden. Dazu wählte der Autor zwei verschiedene Bilderreihen zur Evaluation aus:

1. Ein Torso-Modell, das auch in der Studie *Non-photorealistic rendering in context* [Ise06] eingesetzt wurde. Von diesem Modell sind sowohl handgezeichnete als auch computergenerierte Illustrationen verfügbar.

2. Ein Schädel-Modell, von dem alle Teilnehmer des Anwendertests eine wissenschaftliche Illustration angefertigt haben.

Der Fragestellung, ob die mit der in dieser Arbeit vorgestellten Programmentwicklung geschaffene Illustrationen wissenschaftlich veröffentlichbar sei, galt das besondere Interesse der Online-Umfrage.

Die Auswertung der 129 Teilnehmer der Umfrage ergab, dass sie vorwiegend in den Fachbereichen Archäologie, Medizin, Biologie und Paläontologie tätig sind. Zu den Teilnehmern gehören Studierende, Dozenten, Wissenschaftliche Mitarbeiter, ein Kurator und eine museumstechnische Assistentin.

8.4.2 Illustration der Evaluation

Torso

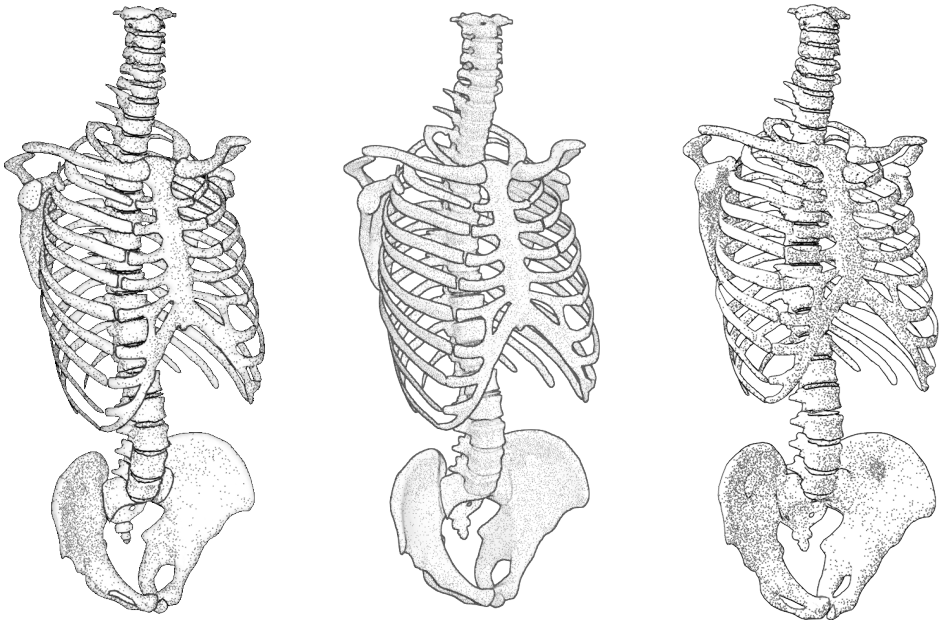
Die drei mit der Illustrations-Pipeline erstellten Illustrationen des Torso-Modells sind in Abbildung 8.8 vorgestellt.

Schädel

Die vier vorliegenden Illustrationen eines Schädelmodells werden in Abbildung 8.9 gezeigt.

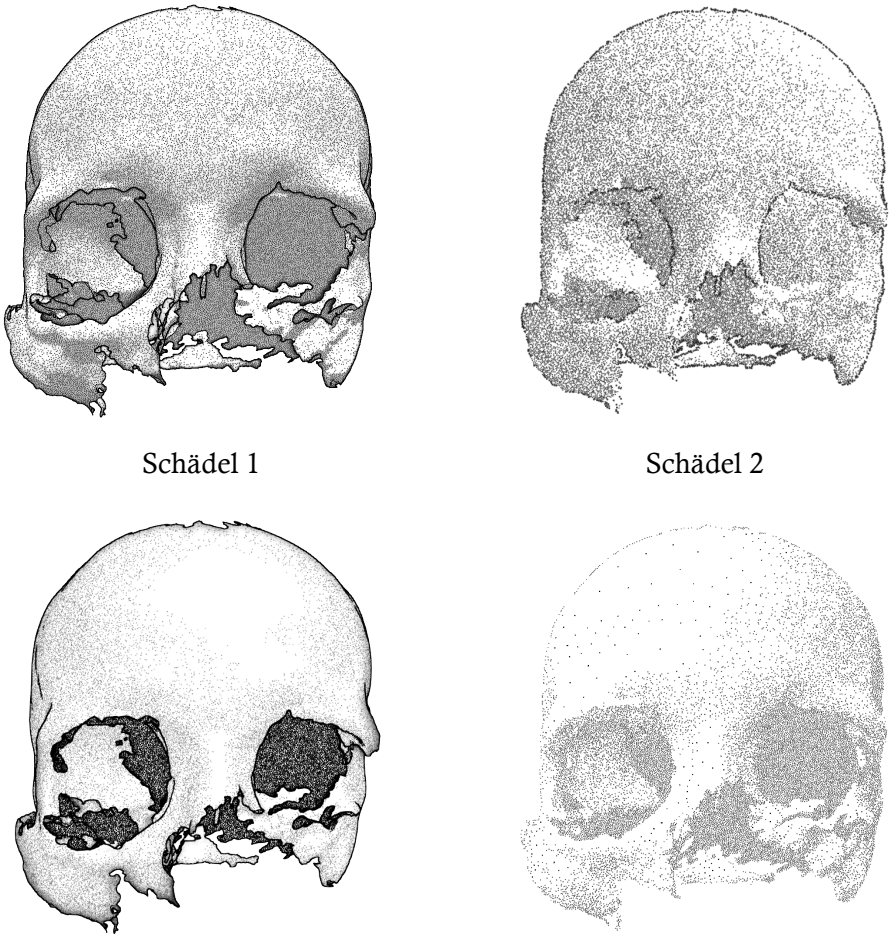
8.4.3 Ergebnisse

Die oben angesprochene Frage, ob das Programm in der Lage ist, im wissenschaftlichen Kontext veröffentlichbare Illustrationen zu erstellen, bejahten die Teilnehmer der Onlineumfrage mehrheitlich. Das Torso-Modell **A** wurde von 61%, das Schädel-Modell **1** von 59% und das Schädel-Modell **3** von 55% aller Teilnehmer als veröffentlichbar eingestuft.



Torso A Torso B Torso C

Abbildung 8.8: Mit der Visualisierungs-Pipeline erstellte Illustrationen des Torsos



Schädel 1 Schädel 2

Schädel 3 Schädel 4

Abbildung 8.9: Mit der Visualisierungs-Pipeline erstellte Illustrationen des Schädels

Dabei zeigte sich bei der Auswertung der Ergebnisse, dass die vom Autor erstellte Illustrationen des Torso-Modells **A** und des Schädel-Modells **1** von allen durch den Prototypen erzeugten Illustrationen am besten bewertet wurden. Erzeugt wurden die Illustrationen nach den Anwendertests; der Autor wusste also, auf welche Aspekte die Experten achteten und welche Merkmale sie hervorheben wollten. Als Autor des Programms kennt er es und er weiß auch genau, *mit welchen Mitteln* sich die *gewünschten Ergebnisse* erreichen und umsetzen lassen.

8.4.4 Zusammenfassung

Im achten Kapitel sind Ergebnisse der Arbeit vorgestellt und betrachtet worden.

Die Oberflächenrekonstruktion aus Bilderstapeln wurde hinsichtlich Güte und Geschwindigkeit mit aktuellen Verfahren verglichen. Im Ground-Truth-Vergleich wies das rekonstruierte Höhenbild weniger Fehler als die anderen Verfahren auf. Dies ist signifikant, da das Höhenbild die Basis für die rekonstruierte Oberfläche ist. Es muss jedoch festgehalten werden, dass die Qualität der Rekonstruktion bei allen Verfahren mit unterschiedlichen Eingabedaten variiert. Diese Tatsache betrifft alle betrachteten Verfahren. Die Geschwindigkeit des Verfahrens konnte gegenüber anderen Verfahren durch Übertragung auf die GPU deutlich gesteigert werden.

Im zweiten Abschnitt wurde der Prototyp vorgestellt und der Anwendertest ausführlich geschildert. Eine zentrale Rolle spielt die Betrachtung der Performanz der Visualisierungs-Pipeline. Auf Grund der Komplexität des Ebenengraphs ist es nicht möglich, universelle Benchmarks aufzuführen. Es konnte aber gezeigt werden, dass die wesentlichen Ebenen auf Bildfiltern basieren, die von aktuellen GPUs sehr schnell angewendet werden. Die verbleibenden komplexeren Ebenen wirken sich nicht negativ auf den interaktiven Eindruck der Pipeline aus.

Daran anschließend wurde das GUI des Prototyps vorgestellt, das hinsichtlich der Usability betrachtet wurde. Es wurden dazu Gespräche mit den Anwendern über die Eignung und Qualität der Ergebnisse geführt. Die Teilnehmer waren sich einig, dass sich mit dem Programm ansprechende Ergebnisse erstellen lassen, die eine solide Grundlage bilden um einen Auftrag zum Anfertigen einer Zeichnung mit einem Illustrator zu besprechen: Die wesentlichen Elemente einer wissenschaftlichen Illustration konnten mit der Visualisierungs-Pipeline wie vom Benutzer gewünscht dargestellt werden. Aufgrund der Komplexität des Programms und der beschränkten Zeit konnten die Teilnehmer nicht alle Möglichkeiten des Prototypen ausprobieren.

Anschließend wurde eine komplexe wissenschaftliche Illustration vorgestellt, die mit dem Programm angefertigt wurde. Es werden die erstellten Ebenen und deren Komposition ins Finalbild besprochen. Die Illustration wurde ausgewählt, weil sie in der im folgenden Abschnitt vorgestellten Onlineumfrage von allen mit dem Prototypen erstell-

ten Illustrationen am besten hinsichtlich der Frage „*Kann diese Illustration veröffentlicht werden?*“ bewertet wurde.

In der Onlineumfrage wurden zwei verschiedene Bildersets von 129 Teilnehmern bewertet. Das erste Set besteht aus neun Bildern eines Torsomodells. Neben den drei mit der Visualisierungs-Pipeline erstellten Bildern wurden drei handgezeichnete und drei mit anderen Algorithmen am Rechner erstellten Bildern verglichen. Die detailliert besprochene mit dem Prototypen erstellte Illustration wurde von 61% der Teilnehmer als veröffentlichbar bewertet.

Das zweite Bilderset setzt sich aus vier Illustrationen eines Schädelmodells zusammen, die alle mit dem Prototypen innerhalb des Anwendertests erstellt wurden. Zwei der Illustrationen wurden von mehr als der Hälfte der Teilnehmer, mit 59% beziehungsweise 55%, als im wissenschaftlichen Kontext veröffentlichbar eingestuft.

Kapitel 9

Fazit

9.1 Zusammenfassung

Die Konzeption und Umsetzung eines Verfahrens, mit dem sich komplexe wissenschaftliche Illustrationen in einem interaktiven Prozess erstellen lassen, ist das Ziel dieser Arbeit. Diese besondere Form der Illustration ist trotz ihrer langen Geschichte auch heute noch von großer wissenschaftlicher Relevanz, da hier eine hervorragende Form der Visualisierung vorliegt, mit der man Theorien über einen abstrahierten Gegenstand veranschaulichen kann. Die Erstellung der abstrakten Zeichnungen erfolgt seit Jahrhunderten manuell von ausgebildeten Illustratoren in Zusammenarbeit mit Wissenschaftlern. Das ist auch heute noch der Normalfall.

Das in dieser Arbeit entwickelte Konzept basiert auf exemplarisch ausgewählten und zur Verfügung gestellten Illustrationen aus der Paläontologieabteilung des Naturmuseums und Forschungsinstitutes Senckenberg Frankfurt/Main. Diese wurden auf ihre verwendeten Techniken hin untersucht, Dabei entwickelte der Autor das Konzept der freien Kombinationen von NPR-Techniken durch Ebenen. Zusätzlich wurden Experten zu verschiedenen *Non-Photorealistic Rendering* Algorithmen für verschiedene Zeichenstile befragt. Danach ließen sich verschiedene Zeichentypen bestimmen und auswählen, die nun im Prototypen implementiert werden konnten. Ziel war es, im wissenschaftlichen Sinne visuell ansprechende Ergebnisse in einem interaktiven Verfahren zu erzeugen.

Die Illustrations-Pipeline

In dieser Arbeit wird die *Illustrations-Pipeline* konzipiert und prototypisch umgesetzt. Diese arbeitet weitgehend auf der GPU und nutzt so die Möglichkeiten und Berechnungskraft moderner GPUs konsequent aus. Die NPR-Verfahren konnten so ausgewählt werden, dass sie ebenfalls auf der GPU arbeiten und sich entweder als Realtime-fähige Bildfilter(-ketten) oder als GPGPU-Programm implementieren lassen. Die Illustrations-Pipeline rendert mit einer frei wählbaren Auflösung, damit sie hochauflösende Illustrationen erzeugen kann. Dabei hat sich in der Evaluation als sinnvoll erwiesen, dass auf dem Anzeigegerät ein frei skalierbarer Ausschnitt angezeigt wird.

Die Ebenen

Kern des Verfahren sind Ebenen, die bestimmte Merkmale eines Objektes mit ausgewählten (NPR-)Techniken darstellen. Die Ebenen sind frei arrangierbar und können auch aufeinander aufbauen. Es wurden Ebenen für die Beleuchtung, verschiedene Pünktelungsarten, Kontur und Bildfiltern konzipiert und umgesetzt. Die Liste der Ebenen lässt sich bei Bedarf leicht um weitere Zeichenstile erweitern.

Durch eine eigene Verwaltungsstruktur werden die Ebenen nur dann gerendert, wenn dies notwendig ist: und zwar genau dann, wenn sich die Ebene selbst oder eine ihrer Eingaben geändert hat. Hier ergab die Auswertung, dass diese Struktur notwendig ist, da einzelne Ebenen durchaus einen hohen Berechnungsaufwand haben können. Sobald die Berechnung abgeschlossen ist, rendert man eine Ebene in ein *Rendertarget*, eine Textur, das dann angezeigt und benutzt werden kann.

Eine GPU-basierte Zeichnenfunktion

Wie schon zuvor hypothetisch angenommen, zeigte sich in der Evaluation, dass es für die Erstellung wissenschaftlicher Illustrationen notwendig ist, dass der Ersteller abstrahierend eingreifen kann. Dies betrifft nicht nur das 3D-Modell, sondern auch alle andere Aspekte von algorithmisch berechneten Ergebnissen.

Realisiert wird dieser manueller Eingriff durch eine GPU-basierte Zeichnenfunktion. Mit dieser lässt sich nicht nur in der Bildebene, sondern auch auf der Oberfläche des dargestellten 3D-Objektes selbst zeichnen. Durch das Zeichnen in eine *Displacementmap* kann sogar das Objekt deformiert und wie gewünscht abstrahiert werden.

Oberflächenrekonstruktion aus Bilderstapeln

Die Evaluation ergab, dass es von Nutzen ist, dass das Framework neben polygonalen 3D-Modellen auch Bilderstapel als Eingabe nutzen kann. Aus diesen wird die 3D-Oberfläche in einem verbesserten und auf die GPU portierten Verfahren rekonstruiert. Das dahinter stehende Konzept ist aber auch in der Lage andere Eingaben, wie beispielsweise Volumendaten, zu verarbeiten, sofern ein entsprechender Renderer Daten in die G-Buffer schreibt.

9.1.1 Bewertung der Hypothesen

Für den Einsatz des Verfahrens wurden verschiedene Hypothesen mit steigendem Qualitätsanspruch formuliert. Auf der untersten Stufe reichen die erzeugten Illustrationen aus, um als Grundlage für die Diskussion mit einem Illustrator zur Erstellung einer Zeichnung zu dienen. Die qualitative Auswertung des Prototypen hat eindeutig bestätigt, dass diese Hypothese zutreffend ist. Dies trifft nicht nur auf die ursprüngliche Zielgruppe, Forscher aus dem Bereich der Paläontologie, zu. Auch eine Medizinerin, die

mit dem Programm arbeitete, sieht in der von ihr erstellten Illustration eine geeignete Rohskizze für einen Auftrag zum Anfertigen einer Zeichnung. Die Auswertung der Onlineumfrage ergab ein ähnliches Bild. Für eine eindeutige Aussage werden jedoch weitere Tests benötigt, die insbesondere die Anwendbarkeit des Konzepts außerhalb der Kerndomäne Paläontologie betrachten.

Die Eignung des Programms als vollwertiger *Ersatz* für die Arbeit von Illustratoren scheint jedoch zum jetzigen Entwicklungsstand noch nicht gegeben zu sein. In der zu dieser Arbeit vorliegenden Onlineumfrage mit 129 Teilnehmern wurden die mit dem Programm erzeugten Illustrationen auf ihre Eignung zur Veröffentlichung hin bewertet. Allerdings haben 61% der Befragten die Eignung der in Abbildung 9.1 gezeigten Illustration, die durch das Programm entstanden ist, für die mit den Hypothesen dieser Arbeit verbundenen Ziele bejaht. Dies zeigt, dass das hier entwickelte Konzept tragfähig ist. Hier muss aber auch konstatiert werden, dass die Illustrationen, die in der Online-Befragung so positiv bewertete wurden, vom Autor dieser Arbeit erstellt worden sind. Er weiß, wie das Programm zu bedienen ist, kennt sich andererseits mit dem Inhalt, der vermittelt werden soll, nur wenig aus. In der Evaluation hat sich auch gezeigt, dass den Probanden zu wenig Zeit eingeräumt wurde, um das Programm kennenzulernen und die geforderten Zeichnungen bzw. Illustrationen anzufertigen. Festgestellt wurde, dass die Probanden mit mehr Hintergrundwissen sicherlich *bessere* Ergebnisse erzielt hätten. Das wird durch die Aussage eines Probanden zu diesem Kontext bestätigt:

> *Das Ergebnis ist schon sehr gut, allerdings muss ich mal einen ganzen Tag oder eine Woche mit dem Programm verbringen, um sagen zu können, was es wirklich alles kann.*

Als Ergebnis der Evaluation (vgl. Kapitel 8) lässt sich konstatieren, dass der Prototyp in der Lage ist, Illustrationen zu erzeugen, die im wissenschaftlichen Kontext veröffentlicht werden können. Die im Rahmen der Evaluation von einem Wissenschaftler der Paläontologie, der beruflich nicht zeichnet, erstellte Illustration (vgl. dazu die Abbildung 9.2) wurde von 55% der 129 Teilnehmer der Online-Befragung als geeignet bewertet, wissenschaftlich angewendet zu werden. Dabei zeigte sich aber auch, dass der Prototyp hinsichtlich der Qualität des Zeichenstils und der Detailungenauigkeit Schwächen aufweist, die eine Weiterentwicklung sinnvoll erscheinen lassen.

9.1.2 Bewertung der Rekonstruktion

Die Evaluation der Rekonstruktion einer 3D-Oberfläche aus einem Bilderstapel wurde auf drei Arten bewertet. Zunächst wurde das Höhenbild, die Grundlage der rekonstruierten Oberfläche, der verschiedenen Verfahren visuell verglichen. Hier ist aufgefallen, dass durch den Schwellwert das Rauschen deutlich gemindert und generell die

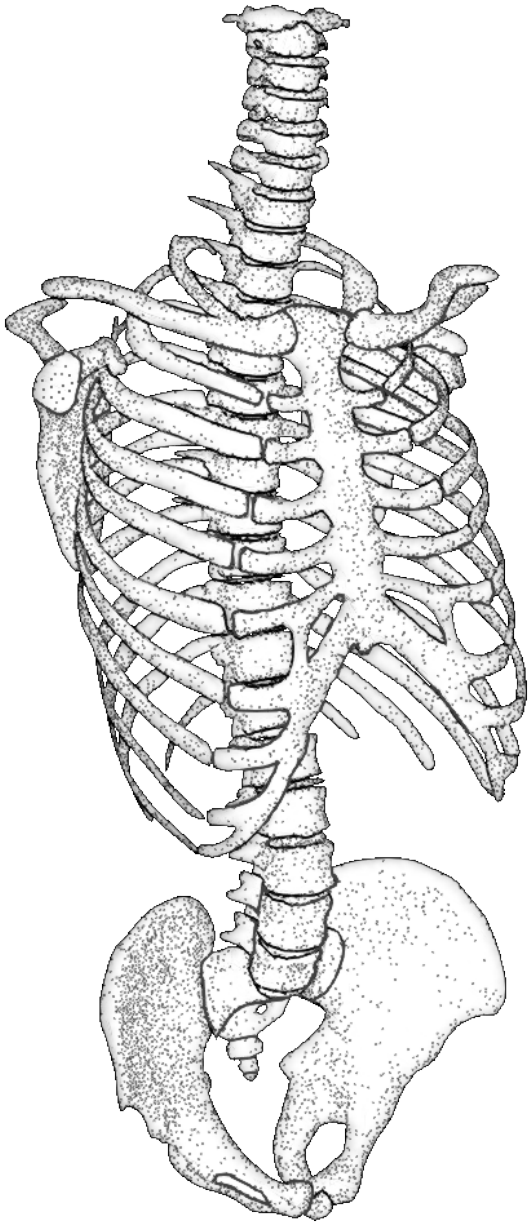


Abbildung 9.1: Mit der Illustrations-Pipeline erstellte wissenschaftliche Illustration eines Torsos

rekonstruierte Oberfläche am plausibelsten wirkt. Anschließend wird in einem Ground-Truth-Vergleich ein synthetischer Bilderstapel und das zugehörige Höhenbild mit der Rendersoftware *Blender* gerendert. Dieses wird mit den rekonstruierten Höhenbildern verglichen und ein Fehlerwert berechnet. Das Höhenbild des in der Arbeit vorgestellten Verfahrens hat von allen Verfahren den geringsten Fehlerwert und somit die kleinste Abweichung. Für den dritten Vergleich wurde die Oberfläche aus einem Stapel heraus rekonstruiert, zu dem auch ein orthogonal aufgenommenes Foto existiert. Dieses wurde mit einem gerenderten Bild der rekonstruierten Oberfläche aus der gleichen Perspektive

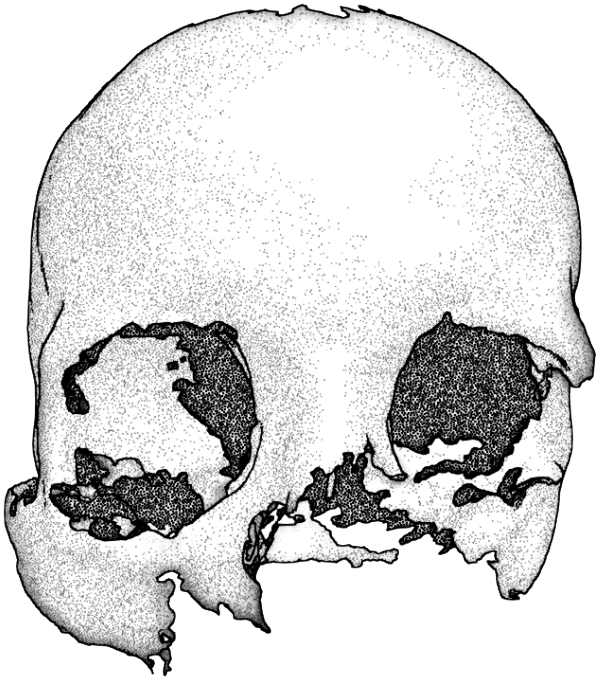


Abbildung 9.2: Mit der Illustrations-Pipeline erstellte wissenschaftliche Illustration eines Schädels

verglichen. Durch diesen Vergleich konnte gezeigt werden, dass die Oberfläche weitestgehend korrekt rekonstruiert wurde. Die Abweichungen der Rekonstruktion ließen sich mit dem Malwerkzeug leicht beheben.

In Geschwindigkeitsvergleichen konnte eine deutliche Beschleunigung gegenüber dem Originalverfahren gemessen werden. Es liegt auf Augenhöhe mit dem kommerziellen Produkt *Helicon Focus* und ist schneller als die *Leica-Software*.

Der in dieser Arbeit vorgestellte Prototyp erweist sich den bisherigen Verfahren überlegen. Die Qualität ließ sich nachweislich deutlich verbessern und im Vergleich zum originalen Verfahren drastisch beschleunigen. Als entscheidendes Kriterium für die Bewertung der Rekonstruktion ist erzeugte Modell nach Ansicht der Endanwender ausreichend, um eine wissenschaftliche Illustration anzufertigen, wie in Bild 9.3 gezeigt. Da ein Bilderstapel mit vorhandener Hardware problemlos erzeugt werden kann, wird so die Attraktivität des gesamten Verfahrens gesteigert.

9.2 Ausblick

Die in dieser Arbeit vorgestellte Entwicklung eines Prototyps für wissenschaftliche Illustrationen führt zu einem anwendbaren Produkt. Es zeigte sich aber auch deutlich, dass noch weitere Forschungen auf diesem Gebiet anhand der hier vorliegenden Basis möglich und empfehlenswert sind.

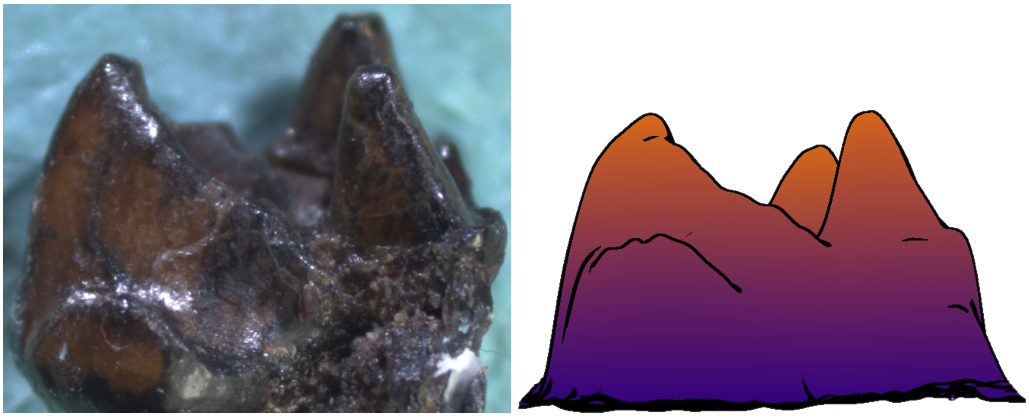


Abbildung 9.3: Aus gleicher Perspektive: Foto des *Leptictidium*-Zahns (links) und Illustration der rekonstruierten Oberfläche (rechts), aus [SH11]

Zunächst wurden bei der Implementierung des Prototypen *bewusste* Abstriche gemacht:

- Ein Voxelrenderer wurde in der ersten Version nicht implementiert. Ein naiver, nicht besonders performanter Renderer ließe sich zwar vergleichsweise leicht hinzufügen, allerdings ist ein effizienter Renderer für große Volumen nicht trivial zu implementieren.
- Es kamen nur einige Zeichentechniken zur Implementierung. Insbesondere wurde auf die Schraffurtechnik komplett verzichtet, die in anderen Fachrichtungen, wie der Biologie oder in der Medizin, gerne eingesetzt wird. Um diese Zielgruppe anzusprechen, ist es unvermeidlich, diese Stile als Ebenentypen umzusetzen. Hinzu kommt, dass jeder Algorithmus eine eigene Ästhetik besitzt, und eine breitere Angebotspalette mehr Geschmacksrichtungen und Vorlieben bedienen könnte.
- Die Zeichnenfunktion wurde in einer minimalen Weise umgesetzt. Komplexere Funktionen, wie beispielsweise besser kontrollierbare, parametrische Kurven (NURBS, …), sind noch nicht implementiert. Sie würden aber dem Nutzer deutlich mehr Kontrolle über das Ergebnis erlauben.
- Die Möglichkeiten, das Objekt zu verformen, sind bisher auch eher rudimentär umgesetzt. Verfahren zum intuitiveren Verformen des Objektes, wie beispielsweise *Volumetrisches Sculpting* [WK95], lassen sich aber leicht mit der existierenden Zeichnenfunktion verbinden. Der Nutzer könnte so einfach auswählen, nach welchem Verfahren er das Objekt bearbeiten möchte.

Die Weiterentwicklung könnte aber auch an anderen Punkten erfolgen. Die meisten Verfahren operieren im Bildraum. Insbesondere Kontur und Kantenlinien werden zwar *gut* erkannt, sind allerdings unter Umständen in ihrer Erscheinung uneinheitlich:

Dem Betrachter fallen augenscheinlich unterschiedlich breite und nicht kontinuierlich ausgeführte Linien auf. Mit speziell auf das Modell angepassten Parametern lassen sich diese Probleme zwar in den Griff bekommen, aber nicht vollständig eliminieren. Hier ist der Einsatz von vektoriellen Verfahren naheliegend. Techniken wie *Snaxels on a plane* [KH11] oder *Computing Smooth Surface Contours with Accurate Topology* [BHK14] gehen das Problem grundsätzlich anders an und könnten besser geeignet sein. Die erkannten Vektoren ließen sich sogar in Verbindung mit dem Vektorzeichnen einsetzen und so noch anpassen.

Vektoren könnten auch für die Pünktelungsverfahren eingesetzt werden. Die gerenderten Punkte sind als Punktmenge in einem VBO ohnehin auf der GPU vektoriell gespeichert. Sie lassen sich auf die CPU übertragen und in einem vektoriellen Bildformat abspeichern. Dies macht nur für die Punkte alleine wenig Sinn, aber wenn auch die anderen Verfahren, wie etwa die Konturlinien, vektoriell vorliegen, ist eine Abspeicherung der gesamten Illustration in einer vektoriellen Datei möglich. Davon würden besonders gedruckte Illustrationen profitieren.

Die Voreinstellungen der Ebenen und die Filtergraphen sind bereits austauschbar. Allerdings sind die Ebenentypen noch fix implementiert. Hier könnten weitere Techniken und NPR-Verfahren über eine geeignete Schnittstelle und der Einsatz von dynamischen Bibliotheken nachgeladen werden. Ein weiterer, bisher nicht betrachteter Aspekt, ist die Animation von 3D-Modellen. Durch bewegte Bilder können zum Teil mehr Informationen über Struktur und die Oberflächenbeschaffenheit als mit einem einzelnen Bild wiedergegeben werden. Gerade heute, in einer Zeit, in der Computer immer tiefer in den Alltag des Menschen eindringen, liegt die Überlegung nahe, Videos anstatt Standbildern auch in wissenschaftlichen Publikationen zu nutzen. In animierten Illustrationen würden sich für die Anwendung neue Herausforderungen stellen, die nach dem jetzigen Publikationsstand nur unvollständig bearbeitet wurden. Pünktelungen können nicht mehr alleine im Bildraum erzeugt werden, es käme sonst zum sogenannten *Showerdoor Effect* [Mei96]. Die Punkte behalten ihre Position auf dem Bildschirm bei, während sich die Oberfläche darunter bewegt. Dieses hier beispielhaft angeführte Problem wurde in der Arbeit *Frame-Coherent Stippling* [PS02] gelöst, jedoch zeigt sich, dass die Lösung nicht auf alle eingesetzten Techniken, die für eine Illustration benötigt werden, übertragbar ist.

Speziell durch die Möglichkeiten der Interaktion mit der Zeichnenfunktion könnte das Verfahren auch vom Einsatz von Touchscreengeräten profitieren. Diese bieten vor allem zeichnenden Forschern oder Illustratoren ein gewohntes Umfeld, so dass sich diese eventuell schneller zurechtfinden könnten. Auch nicht zeichnende Forscher würden sicherlich von einem direkteren Interface profitieren.

Zuletzt sollte das Gesamtverfahren *vollständig* evaluiert werden. Dies betrifft sowohl die Usability der konzipierten Benutzungsschnittstelle als auch die Qualität und Ver-

wertbarkeit der Ergebnisse. Ein Mangel stellt sich dar, dass die bisherige Studie nur im zeitlich knappen Rahmen einer Bachelorarbeit durchgeführt werden konnte.

Das eigentliche Ziel einer wissenschaftlichen Illustration ist es hier, eine Idee bzw. eine Theorie zu einem Objekt wiederzugeben. Die Evaluation des vorgestellten Prototyps hat erbracht, dass eine fachmännische von Hand gezeichnete Illustration vermutlich immer im Vorteil sein wird. Ein Grund dürfte die Expressivität bzw. Subjektivität eines wissenschaftlichen Illustrators sein, die sich als seine Handschrift bemerkbar macht.

Zusätzlich hat sich in der Evaluation des hier vorgestellten Prototyps gezeigt, dass auch wissenschaftliche Illustrationen von den befragten Fachleuten nach ästhetischen Gesichtspunkten beurteilt werden, deren Objektivierbarkeit - Voraussetzung für eine Implementierung - kaum möglich erscheint. Hier zeigen sich deutlich die Grenzen einer computergenerierten Illustration.

Für einen Algorithmus - so kompliziert er auch sein mag - ist es zumindest heute noch nahezu unmöglich, auf der Ebene der *Ästhetik* oder *Attraktivität* mit dem Menschen zu konkurrieren. Ein Grund dafür mag sein, dass wir die der Subjektivität zugeschriebene „*Fehlerhaftigkeit*“, mit der Illustratoren fast zwangsläufig arbeiten, als Produkt von Individualität auch positiv wahrnehmen.

Viele sprechen hier von der persönlichen Handschrift, dem Stil eines wissenschaftlichen Zeichners; also einer persönlichen Bildästhetik, die allgemein wissenschaftlich akzeptiert ist. Auch mit dieser Arbeit zeigt sich, dass computergenerierte wissenschaftliche Illustrationen zwar deutliche Fortschritte machen, dass aber der *NPR-Turing Test* [Sal02] (noch immer) weit davon entfernt ist, bestanden zu werden.

Anhänge

Anhang A

Onlineumfrage

Die Onlineumfrage zur Bewertung von wissenschaftlichen Illustrationen, die mit dem hier vorgestellten Prototypen angefertigt wurde, führte Markus Strobel innerhalb seiner Bachelorarbeit *Evaluation eines Programmes zur Erstellung wissenschaftlicher Illustrationen* an der Johann Wolfgang Goethe-Universität Frankfurt durch. In diesem Anhangskapitel wird die Umfrage kurz vorgestellt und anschließend werden die Ergebnisse ausgewertet. Eine Diskussion der Ergebnisse ist in Kapitel 8.4 zu finden.

A.1 Erstellung der wissenschaftlichen Illustrationen

Im Rahmen der Anwendertests zu dem hier vorgestellten Prototypen wurden zwei verschiedene 3D-Modelle als Basis für die zu erstellenden Illustrationen ausgewählt.

Das Torso-Modell, das auch in der Studie *Non-photorealistic rendering in context* [Ise06], im Folgenden kurz: *NPRIC*, zur Anwendung kam, wurde ausgewählt, da von diesem allgemein bekannteren Modell mehrere Handzeichnungen sowie gleichzeitig computergenerierte Illustrationen vorliegen, die sich gut vergleichen lassen. Sechs ausgewählte Bilder wurden mit Erlaubnis des Autors der Studie, Dr. Tobias Isenberg, in der Onlineumfrage verwendet.

Von dem vorhandenen Schädel-Modell wurden insgesamt vier Illustrationen erstellt und in die Evaluation einbezogen.

A.1.1 Konzeption der Evaluation

Die Illustrationen werden hinsichtlich ihrer wissenschaftlichen Anwendbarkeit untersucht und erprobt. Daher beziehen sich die Fragen auf diese beiden Felder:

- Ist diese Illustration wissenschaftlich veröffentlichbar?
- Halten Sie diese Illustration für computergeneriert?

Angesichts des zeitlich engen Rahmens der Bachelorarbeit wählte der Autor der Onlineumfrage eine ihm geeignet erscheinende existierende Plattform für Evaluationen aus: Q-set[1]. Ein in dem Kontext dieser Arbeit schwerwiegender Nachteil der Plattform

[1] www.q-set.de, besucht: 07.12.2014

ist jedoch, dass die Bilder und Fragen nicht randomisiert werden können. Aus diesem Grund wurden jeweils 3 Bilder des Torso-Modells ausgewählt und nebeneinander zu einer Frage gezeigt: eine Handzeichnung, ein computergeneriertes Bild der Studie *NPRIC* und eine Illustration, die mit dem Prototypen erstellt wurde.

Ein weiteres Problem zeigte sich erst nach Abschluss der Studie und betrifft die Illustrationen, die mit dem hier vorgestellten Programm erstellt wurden. Sie weisen sichtbare Kompressionsartefakte auf, so dass sie nicht nur im direkten Vergleich auffallen, sondern sie sind hier zusätzlich noch nicht in bestmöglicher Qualität wiedergegeben.

A.2 Illustrationen in der Onlineumfrage

Die in Abbildung A.1 dargestellten Paarungen sind genauso wiedergegeben, wie sie in der Onlineumfrage gezeigt wurden. Die vorgelegten Illustrationen, die mit dem entwickelten Prototypen erstellt wurden,wurden von den Teilnehmer/innen des Anwendertests, zusätzlich vom Autor des Prototyps und einer Doktorandin, die sich in ihrer Forschung mit der Computergrafik befasst. Die Quelle der Illustration ist in Tabelle A.1 aufgeführt.

Illustration	Methode		Quelle/AutorIn
1	**C**	Computer	[Ise06]
2	**P**	Prototyp	Probandin
3	**M**	Manuell	[Ise06]
4	**P**	Prototyp	Autor des Prototyps
5	**C**	Computer	[Ise06]
6	**M**	Manuell	[Ise06]
7	**M**	Manuell	[Ise06]
8	**P**	Prototyp	Doktorandin
9	**C**	Computer	[Ise06]

Tabelle A.1: Wissenschaftliche Illustrationen des Torsos der Onlinestudie: Erstellungsmethode und Quelle/AutorIn

Drei der in Abbildung A.2 wiedergegebenen Illustrationen des Schädel-Modells wurden durch den Anwendertest erstellt. Eine zusätzliche vierte Illustration wurde vom Autor des Programms erzeugt.

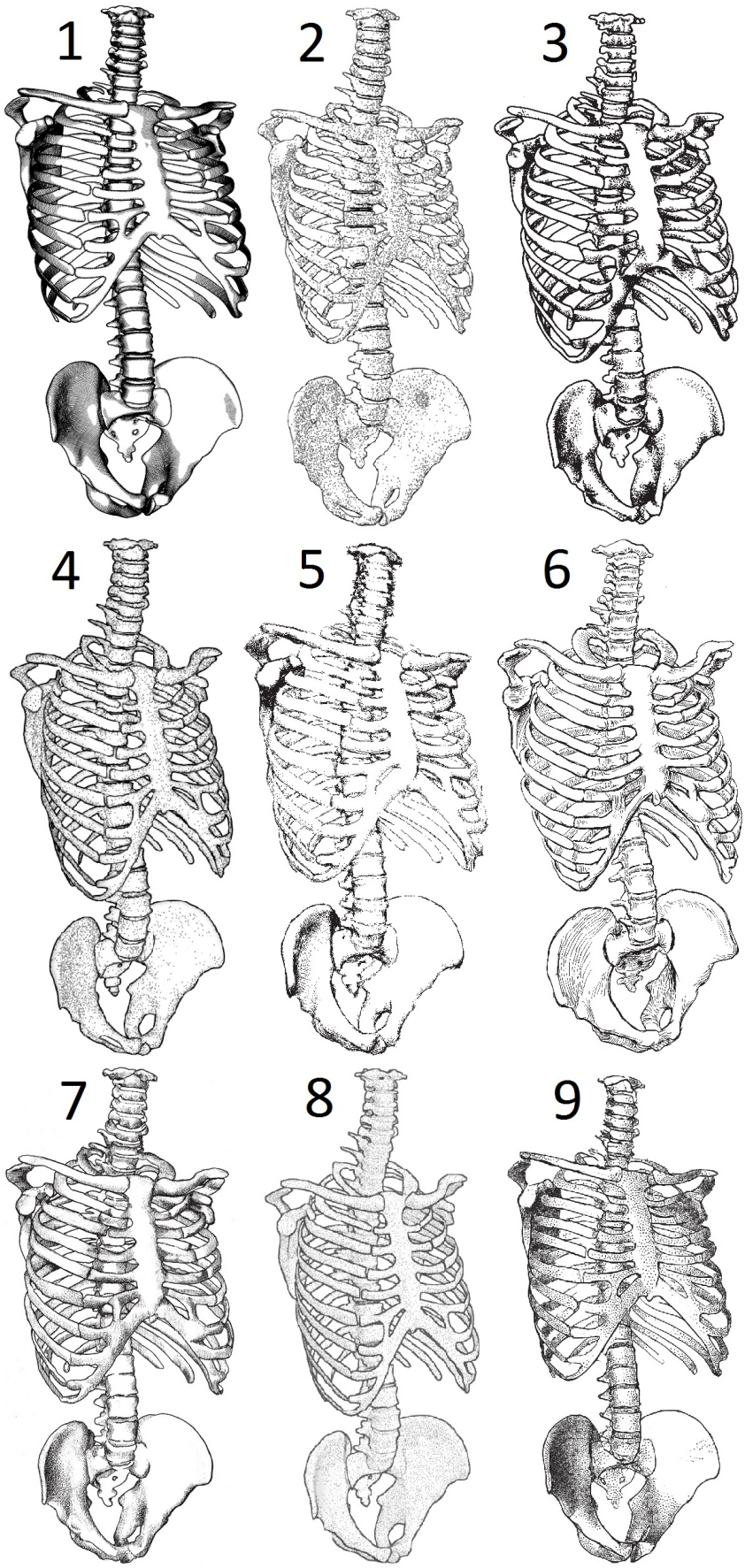
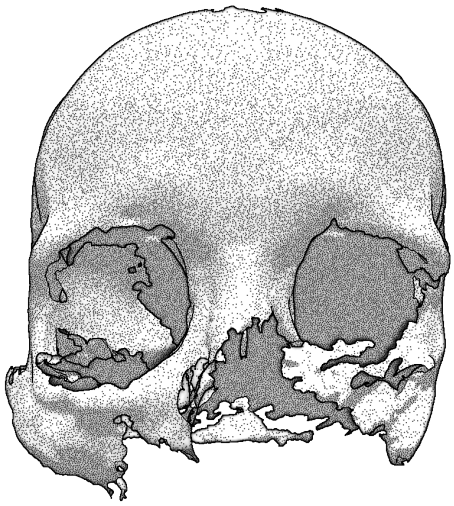
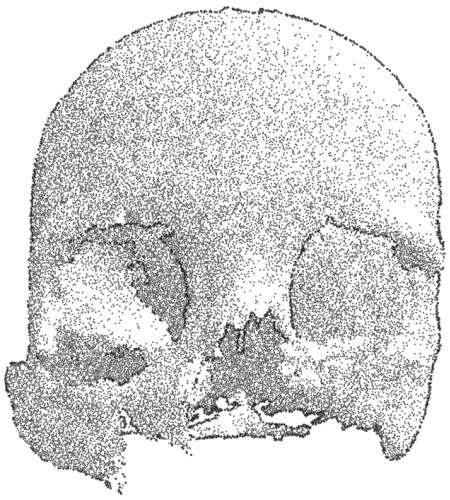


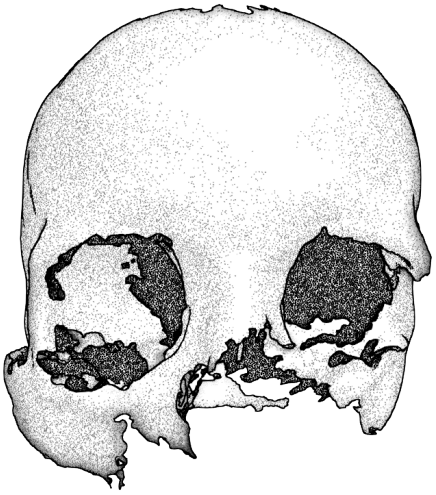
Abbildung A.1: Wissenschaftliche Illustrationen des Torsos



Schädel 1



Schädel 2



Schädel 3

Schädel 4

Abbildung A.2: Wissenschaftliche Illustrationen des Schädels

A.3 Teilnehmer der Evaluation

Eine Einladung zur Teilnahme wurde an Probanden der Fächer Archäologie, Medizin, Biologie und Paläontologie von verschiedenen deutschen Universitäten verschickt. Die 129 Teilnehmer, die die Einladung angenommen und den Fragebogen komplett ausgefüllt haben, sind den in der Tabelle A.2 ersichtlichen Fächern und Berufsfeldern zuzuordnen. Zu den Teilnehmern zählen Studierende, Dozenten, Wissenschaftliche Mitarbeiter, ein Kurator, eine museumstechnische Assistentin sowie Personen die anderweitig *beruflich* mit wissenschaftlichen Illustrationen zu tun haben. Die Ergebnisse werden getrennt nach Studierenden und Nicht-Studierenden aufgeführt. Die Aussagen der Nicht-Studierenden wurden mit besonderen Gewicht gewertet, da sich diese Gruppe hauptsächlich aus Personen zusammensetzt, die mit wissenschaftlichen Arbeiten vertraut sind. Daten zum Geschlecht oder Alter wurden in der Studie nicht erhoben.

Beruf	Paläontologie	Archäologie	Biologie	Medizin	Computergrafik	Andere	Σ
StudentIn	0	58	4	0	0	0	62
DozentIn	1	2	7	0	0	1	11
Wissenschaftliche MitarbeiterIn	3	15	17	0	0	0	35
DoktorandIn	0	4	1	0	0	1	7
Anderer	4	4	2	2	2	1	15
Σ	8	83	31	2	2	3	129

Tabelle A.2: Teilnehmer der Onlineumfrage, getrennt nach Beruf und Fachrichtung

A.4 Ergebnisse der Umfrage

In diesem Anhangskapitel werden die statistisch erfassten Ergebnisse der Onlineumfrage gezeigt. Die Ergebnisse sind anhand der in Anhangskapitel A.1.1 vorgestellten Fragen statistisch erfasst worden. Alle Angaben entsprechen Prozentangaben und stellen die Ja-Antworten dar. Die Auswertung erfolgte getrennt nach Beruf und Fachrichtung. Die hier vorgelegten Ergebnisse liegen der Evaluation zugrunde.

A.4.1 Ist die Illustration computergeneriert?

Ja-Antworten in %		Torso									Schädel			
	Methode	C	**P**	M	**P**	C	M	M	**P**	C	Prototyp			
	Abbildung	1	**2**	3	**4**	5	6	7	**8**	9	**1**	**2**	**3**	**4**
Beruf	StudentIn	94	**60**	26	**87**	44	24	48	**71**	39	**82**	**69**	**58**	**68**
	DozentIn	91	**64**	45	**82**	91	55	55	**91**	55	**100**	**91**	**91**	**82**
	WiMi	89	**51**	31	**86**	57	34	51	**66**	54	**86**	**83**	**74**	**83**
	DoktorandIn	67	**67**	17	**83**	17	33	50	**83**	17	**100**	**83**	**83**	**33**
	Anderer	60	**67**	33	**60**	67	27	40	**67**	60	**73**	**80**	**67**	**67**
	Mittelwert	87	**59**	29	**83**	53	30	49	**71**	46	**84**	**77**	**67**	**71**
	Std-Abw.	15	**6**	11	**11**	28	12	5	**11**	18	**12**	**8**	**13**	**20**

Tabelle A.3: *Nach Beruf*: Ist die Illustration computergeneriert? „*Ja*“ in Prozent

Ja-Antworten in %		Torso									Schädel			
	Methode	C	**P**	M	**P**	C	M	M	**P**	C	Prototyp			
	Abbildung	1	**2**	3	**4**	5	6	7	**8**	9	**1**	**2**	**3**	**4**
Fachrichtung	Paläontologie	63	**38**	50	**63**	75	38	25	**50**	50	**88**	**75**	**63**	**75**
	Archäologie	90	**60**	27	**86**	46	29	48	**69**	41	**83**	**69**	**63**	**69**
	Biologie	87	**58**	35	**81**	68	35	65	**77**	55	**87**	**97**	**84**	**77**
	Medizin	100	**50**	0	**50**	50	0	0	**100**	100	**50**	**100**	**0**	**50**
	Computergrafik	50	**100**	50	**100**	50	50	0	**100**	100	**100**	**50**	**100**	**100**
	Andere	67	**67**	0	**100**	33	0	33	**100**	0	**100**	**100**	**67**	**67**
	Mittelwert	87	**59**	29	**83**	53	30	49	**71**	46	**84**	**77**	**67**	**71**
	Std-Abw.	19	**21**	23	**20**	15	21	26	**21**	38	**18**	**21**	**34**	**16**

Tabelle A.4: *Nach Fachrichtung*: Ist die Illustration computergeneriert? „*Ja*“ in Prozent

A.4.2 Kann die Illustration veröffentlicht werden?

Ja-Antworten in %		Torso									Schädel			
	Methode	C	**P**	M	**P**	C	M	M	**P**	C	Prototyp			
	Abbildung	1	**2**	3	**4**	5	6	7	**8**	9	**1**	**2**	**3**	**4**
Beruf	StudentIn	94	**21**	74	**60**	6	89	95	**32**	66	**65**	**3**	**63**	**21**
	DozentIn	91	**27**	55	**55**	9	73	82	**45**	64	**64**	**0**	**45**	**27**
	WiMi	89	**17**	71	**60**	6	80	94	**23**	40	**57**	**3**	**49**	**14**
	DoktorandIn	100	**33**	50	**83**	17	100	100	**50**	33	**33**	**0**	**33**	**33**
	Anderer	80	**7**	67	**67**	7	60	100	**13**	33	**47**	**7**	**53**	**40**
	Mittelwert	91	**19**	70	**61**	7	82	95	**29**	53	**59**	**3**	**55**	**22**
	Std-Abw.	79	**10**	11	**11**	5	15	7	**15**	16	**13**	**3**	**11**	**10**

Tabelle A.5: *Nach Beruf*: Kann die Illustration veröffentlicht werden? „*Ja*“ in Prozent

Ja-Antworten in %		Torso									Schädel			
	Methode	C	**P**	M	**P**	C	M	M	**P**	C	Prototyp			
	Abbildung	1	**2**	3	**4**	5	6	7	**8**	9	**1**	**2**	**3**	**4**
Fachrichtung	Paläontologie	88	**13**	50	**50**	13	63	100	**13**	38	**38**	**0**	**50**	**25**
	Archäologie	90	**22**	73	**59**	7	86	96	**27**	61	**63**	**2**	**60**	**19**
	Biologie	94	**19**	65	**68**	6	77	87	**42**	42	**65**	**3**	**42**	**26**
	Medizin	50	**0**	100	**50**	0	100	100	**0**	0	**0**	**0**	**50**	**50**
	Computergrafik	50	**0**	100	**100**	0	50	100	**0**	0	**0**	**50**	**100**	**50**
	Andere	100	**0**	67	**67**	0	100	100	**67**	67	**33**	**0**	**33**	**33**
	Mittelwert	90	**19**	71	**61**	7	82	95	**29**	53	**59**	**3**	**55**	**22**
	Std-Abw.	23	**10**	20	**19**	5	20	5	**26**	29	**29**	**20**	**23**	**13**

Tabelle A.6: *Nach Fachrichtung*: Kann die Illustration veröffentlicht werden? „*Ja*“ in Prozent

Literaturverzeichnis

[AHH08] Tomas Akenine-Möller, Eric Haines und Natty Hoffman. *Real-Time Rendering 3rd Edition*. Natick, MA, USA: A. K. Peters, Ltd., 2008, S. 1045. ISBN: 987-1-56881-424-7.

[Bae07] Alexandra Baer u. a. „Hardware-Accelerated Stippling of Surfaces Derived from Medical Volume Data“. In: *IEEE/Eurographics Symposium on Visualization (EuroVis)*. 2007, S. 235–242.

[Bas13] Katie Bassett u. a. „Authoring and Animating Painterly Characters“. In: *ACM Transactions on Graphics* 32.5 (Sep. 2013), 156:1–156:12.

[BHK14] Pierre Bénard, Aaron Hertzmann und Michael Kass. „Computing Smooth Surface Contours with Accurate Topology“. In: *ACM Trans. Graph.* 33.2 (Apr. 2014), 19:1–19:21. ISSN: 0730-0301.

[Bli77] James F. Blinn. „Models of light reflection for computer synthesized pictures“. In: *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '77. San Jose, California: ACM, 1977, S. 192–198.

[Bru08] Stefan Bruckner. „Interactive Illustrative Volume Visualization“. Ph.D. Thesis. Technische Universität Wien, 2008.

[BSD08] Louis Bavoil, Miguel Sainz und Rouslan Dimitrov. „Image-space horizon-based ambient occlusion“. In: *ACM SIGGRAPH 2008 talks*. (Los Angeles, California). SIGGRAPH '08. New York, NY, USA: ACM, 2008, 22:1–22:1. ISBN: 978-1-60558-343-3.

[CAY99] T. S. Choi, M. Asif und Joungil Yun. „Three-dimensional shape recovery from focused image surface“. In: *ICASSP '99: Proceedings of the Acoustics, Speech, and Signal Processing, 1999. on 1999 IEEE International Conference*. Washington, DC, USA: IEEE Computer Society, 1999, S. 3269–3272. ISBN: 0-7803-5041-3.

[Col12] Forrester Cole u. a. „Where Do People Draw Lines?“ In: *Communications of the ACM* 55.1 (Jan. 2012), S. 107–115.

[Cra11] Cyril Crassin. „GigaVoxels: A Voxel-Based Rendering Pipeline For Efficient Exploration Of Large And Detailed Scenes“. English and web-optimized version. Diss. UNIVERSITE DE GRENOBLE, Juli 2011.

[Dar62] Charles Darwin. „On the Three remarkable Sexual Forms of Catasetum tridentatum, an Orchid in the possession of the Linnean Society.“ In: *Journal of the Proceedings of the Linnean Society of London. Botany* 6.24 (1862), S. 151–157. ISSN: 1945-9424.

[DI13] Oliver Deussen und Tobias Isenberg. „Halftoning and Stippling“. In: *Image and Video based Artistic Stylisation*. Hrsg. von Paul Rosin und John Collomosse. Bd. 42. Computational Imaging and Vision. London, Heidelberg: Springer-Verlag, 2013. Kap. 3, S. 45–61. ISBN: 978-1-4471-4518-9.

[FCH01] G. Frankowski, M. Chen und T. Huth. „Optical Measurement of the 3D-Coordinates and the Combustion Chamber Volume of Engine Cylinder Heads“. In: *Fringe 2001*. Elsevier, 2001.

[Fis99] H.W. Fischer. *Naturwissenschaftliches Zeichnen und Illustrieren*. Freunde der Würzburger Geowissenschaften, 1999.

[Fri09] Michael Friendly. *Milestones in the history of thematic cartography, statistical graphics, and data visualization*. 2009.

[GG01] Bruce Gooch und Amy A. Gooch. *Non-Photorealistic Rendering*. Natick: A K Peters, Ltd., 2001.

[GH97] Michael Garland und Paul S. Heckbert. „Surface Simplification Using Quadric Error Metrics“. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, S. 209–216. ISBN: 0-89791-896-7.

[Goo98] Amy Gooch u. a. „A non-photorealistic lighting model for automatic technical illustration“. In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '98. New York, NY, USA: ACM, 1998, S. 447–452. ISBN: 0-89791-999-8.

[Gwo] Pascal Gwosdek u. a. „Fast electrostatic halftoning“. In: *Journal of Real-Time Image Processing* (), S. 1–14. ISSN: 1861-8200.

[Had06] Markus Hadwiger u. a. *Real-time Volume Graphics*. Natick, MA, USA: A. K. Peters, Ltd., 2006. ISBN: 1568812663.

[HH04] Shawn Hargreaves und Mike Harris. *Deferred Shading*. Game Developer Conference. 2004. URL: `http://http.download.nvidia.com/developer/presentations/2004/6800_Leagues/6800_Leagues_Deferred_Shading.pdf` (besucht am 06. 10. 2012).

[Hir98] Y. Hirayasu u. a. „Lower Left Temporal Lobe MRI Volumes in Patients with First-episode Schizophrenia Compared with Psychotic Patients with First-episode Affective Disorder and Normal Subjects“. In: 155.10 (Okt. 1998), S. 1384–1391.

[HM90] R. B. Haber und D. A. Mcnabb. „Visualization Idioms: A Conceptual Model for Scientific Visualization Systems“. In: *Visualization in Scientific Computing*. 1990.

[Hod03] E. Hodges. *The Guild Handbook of Scientific Illustration*. John Wiley und Sons, 2003. ISBN: 0471360112.

[Hoy06] Anne H. Hoy. *Enzyklopädie der Fotografie: Die Geschichte - Die Technik - Die Kunst - Die Zukunft*. NATIONAL GEOGRAPHIC Deutschland, 2006. ISBN: 978-3937606903.

[Ise06] Tobias Isenberg u. a. „Non-photorealistic rendering in context: an observational study“. In: *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*. NPAR '06. New York, NY, USA: ACM, 2006, S. 115–126. ISBN: 1-59593-357-3.

[Ise13] Tobias Isenberg. „Evaluating and Validating Non-Photorealistic and Illustrative Rendering“. In: *Image and Video based Artistic Stylisation*. Hrsg. von Paul Rosin und John Collomosse. Bd. 42. Computational Imaging and Vision. London, Heidelberg: Springer-Verlag, 2013. Kap. 15, S. 311–331. ISBN: 978-1-4471-4518-9.

[Jäh05] Bernd Jähne. *Digitale Bildverarbeitung*. 6. Berlin: Springer, 2005. ISBN: 978-3-540-24999-3.

[Jim11a] Jorge Jimenez u. a. „Filtering Approaches for Real-Time Anti-Aliasing“. In: *ACM SIGGRAPH Courses*. 2011.

[Jim11b] Jorge Jimenez u. a. „GPU Pro 2“. In: Hrsg. von Wolfgang Engel. AK Peters Ltd., 2011. Kap. Practical Morphological Anti-Aliasing.

[Jim12] Jorge Jimenez u. a. „SMAA: Enhanced Morphological Antialiasing“. In: *Computer Graphics Forum (Proc. EUROGRAPHICS 2012)* 31.2 (2012).

[KBH06] Michael Kazhdan, Matthew Bolitho und Hugues Hoppe. „Poisson surface reconstruction“. In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. SGP '06. Cagliari, Sardinia, Italy: Eurographics Association, 2006, S. 61–70. ISBN: 3-905673-36-3.

[KGC00] Matthew Kaplan, Bruce Gooch und Elaine Cohen. „Interactive Artistic Rendering“. In: *Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering*. NPAR '00. Annecy, France: ACM, 2000, S. 67–74. ISBN: 1-58113-277-8.

[KH11] Kevin Karsch und John C. Hart. „Snaxels on a plane“. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*. (Vancouver, British Columbia, Canada). NPAR '11. New York, NY, USA: ACM, 2011, S. 35–42. ISBN: 978-1-4503-0907-3.

[Kim08] Dongyeon Kim u. a. „Feature-guided Image Stippling“. In: *Computer Graphics Forum* 27.4 (2008), S. 1209–1216.

[Kim09] SungYe Kim u. a. „Stippling By Example“. In: *In Proceedings of the 7th international symposium on Non-photorealistic animation and rendering (NPAR)* (2009), to–appear.

[Kno12] Christoph Knopp. „Stipplingalgorithmen für paläontologische Fundstücke“. Diplomarbeit. Goethe-Universität Frankfurt, 2012.

[Kop06] Johannes Kopf u. a. „Recursive Wang tiles for real-time blue noise“. In: *ACM SIGGRAPH 2006 Papers*. (Boston, Massachusetts). SIGGRAPH '06. New York, NY, USA: ACM, 2006, S. 509–518. ISBN: 1-59593-364-6.

[Kyp10] Jan Eric Kyprianidis u. a. „Anisotropic Kuwahara Filtering with Polynomial Weighting Functions“. In: *Proc. EG UK Theory and Practice of Computer Graphics*. 2010, S. 25–30.

[LCD06] Thomas Luft, Carsten Colditz und Oliver Deussen. „Image enhancement by unsharp masking the depth buffer“. In: *ACM Transactions on Graphics* 25 (2006), S. 1206–1213.

[LD08] Haik Lorenz und Jürgen Döllner. „Dynamic Mesh Refinement on GPU using Geometry Shaders“. In: *16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG) - Full Papers*. 2008, S. 97–104.

[Lee10] Victor W. Lee u. a. „Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU“. In: *Proceedings of the 37th annual international symposium on Computer architecture*. ISCA '10. Saint-Malo, France: ACM, 2010, S. 451–460. ISBN: 978-1-4503-0053-7.

[LM11] Hua Li und David Mould. „Structure-preserving Stippling by Priority-based Error Diffusion“. In: *Proceedings of Graphics Interface 2011*. GI '11. St. John's, Newfoundland, Canada: Canadian Human-Computer Communications Society, 2011, S. 127–134. ISBN: 978-1-4503-0693-5.

[Mar10] Domingo Martín u. a. „Example-based stippling using a scale-dependent grayscale process“. In: *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*. (Annecy, France). NPAR '10. New York, NY, USA: ACM, 2010, S. 51–61. ISBN: 978-1-4503-0125-1.

[Mar10] Victor Martins. *Evaluate a Cubic Bézier on GPU*. 11. Mai 2010. URL: http://www.pixelnerve.com/v/2010/05/11/evaluate-a-cubic-bezier-on-gpu/ (besucht am 02. 10. 2012).

[MB11] John McDonald Jr und Brent Burley. „Per-face Texture Mapping for Real-time Rendering“. In: *ACM SIGGRAPH 2011 Studio Talks*. SIGGRAPH '11. Vancouver, British Columbia, Canada: ACM, 2011, 3:1–3:1. ISBN: 978-1-4503-0973-8.

[McS03] Mike McShaffry. *Game Coding Complete*. Paraglyph Publishing, 2003. ISBN: 1932111751.

[Mei96] Barbara J. Meier. „Painterly rendering for animation“. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. SIGGRAPH '96. ACM, 1996, S. 477–484. ISBN: 0-89791-746-4.

[Mil68] Robert B. Miller. „Response time in man-computer conversational transactions“. In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. AFIPS '68 (Fall, part I). San Francisco, California: ACM, 1968, S. 267–277.

[MML12] Morgan McGuire, Michael Mara und David Luebke. „Scalable Ambient Obscurance“. In: *High-Performance Graphics 2012*. Paris, France, Juni 2012.

[Nie93] Jakob Nielsen. *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN: 0125184050.

[NN94] S. K. Nayar und Y. Nakagawa. „Shape from Focus“. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 16.8 (1994), S. 824–831. ISSN: 0162-8828.

[ON94] Michael Oren und Shree K. Nayar. „Generalization of Lambert's reflectance model“. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. SIGGRAPH '94. New York, NY, USA: ACM, 1994, S. 239–246. ISBN: 0-89791-667-0.

[pau12] paul.houx. *Smooth thick lines using geometry shader*. Juni 2012. URL: https://forum.libcinder.org/#Topic/23286000001268015 (besucht am 02. 10. 2012).

[PFS03] Oscar Meruvia Pastor, Bert Freudenberg und Thomas Strothotte. „Real-Time Animated Stippling“. In: *IEEE Comput. Graph. Appl.* 23 (4 Juli 2003), S. 62–68. ISSN: 0272-1716.

[Poy03] Charles Poynton. *Digital Video and HDTV Algorithms and Interfaces*. 1. Aufl. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003. ISBN: 1558607927.

[PP05] Reinhard Putz und Reinhard Pabst. *Sobotta, Atlas der Anatomie des Menschen Band 2*. 22. Aufl. Urban & Fischer Verlag/Elsevier GmbH, 2005. ISBN: 3437444107.

[PR06] K. S. Pradeep und A. N. Rajagopalan. „Improving Shape from Focus Using Defocus Information“. In: *ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition*. Washington, DC, USA: IEEE Computer Society, 2006, S. 731–734. ISBN: 0-7695-2521-0.

[PS02] Oscar E. Meruvia Pastor und Thomas Strothotte. „Frame-Coherent Stippling“. In: *EUROGRAPHICS 2002, Short Presentations*. 2002.

[PT97] Les Piegl und Wayne Tiller. *The NURBS book (2nd ed.)* New York, NY, USA: Springer-Verlag New York, Inc., 1997. ISBN: 3-540-61545-8.

[PVW10] Roy van Pelt, Anna Vilanova i Bartroli und Huub van de Wetering. „Illustrative Volume Visualization Using GPU-Based Particle Systems“. In: *Visualization and Computer Graphics, IEEE Transactions on* 16.4 (Juli 2010), S. 571 –582. ISSN: 1077-2626.

[Ray99] S.F. Ray. *Scientific Photography and Applied Imaging*. Focal Press, 1999. ISBN: 9780240513232.

[RBD06] Szymon Rusinkiewicz, Michael Burns und Doug DeCarlo. „Exaggerated shading for depicting shape and detail“. In: *ACM SIGGRAPH 2006 Papers*. (Boston, Massachusetts). SIGGRAPH '06. New York, NY, USA: ACM, 2006, S. 1199–1205. ISBN: 1-59593-364-6.

[Ros09] Randi J. Rost. *OpenGL(R) Shading Language (3rd Edition)*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2009. ISBN: 0321637631.

[Rou13] Nicolas P. Rougier. „Shader-Based Antialiased, Dashed, Stroked Polylines“. In: *Journal of Computer Graphics Techniques (JCGT)* 2.2 (6. Dez. 2013), S. 105–121. ISSN: 2331-7418.

[RTB96] Bernice E. Rogowitz, Lloyd A. Treinish und Steve Bryson. „How not to lie with visualization“. In: *Comput. Phys.* 10.3 (Juni 1996), S. 268–273. ISSN: 0894-1866.

[Rus08] Szymon Rusinkiewicz u. a. „Line drawings from 3D models“. In: *ACM SIGGRAPH 2008 classes*. SIGGRAPH '08. Los Angeles, California: ACM, 2008, 39:1–39:356.

[Rus12] Szymon Rusinkiewicz. *Trimesh2*. 2012. URL: http://www.cs.princeton.edu/gfx/proj/trimesh2 (besucht am 01. 10. 2012).

[Sal02] David H. Salesin. *Non-Photorealistic Animation & Rendering: 7 Grand Challenges*. Keynote talk at Second International Symposium on Non-Photorealistic Animation and Rendering (NPAR 2002, Annecy, France, June 3--5, 2002). Juni 2002.

[Sal94] Michael P. Salisbury u. a. „Interactive pen-and-ink illustration“. In: *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1994, S. 101–108. ISBN: 0-89791-667-0.

[Sch07] Sebastian Schäfer. „Bumpmapping-Verfahren und deren Weiterentwicklung“. Diplomarbeit. Goethe-Universität Frankfurt, 2007.

[Sch10] Christian Schmaltz u. a. „Electrostatic Halftoning“. In: *Computer Graphics Forum* 29.8 (2010), S. 2313–2327. ISSN: 1467-8659.

[Sec02] Adrian Secord. „Weighted Voronoi stippling“. In: *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*. (Annecy, France). NPAR '02. New York, NY, USA: ACM, 2002, S. 37–43. ISBN: 1-58113-494-0.

[SGS05] Stefan Schlechtweg, Tobias Germer und Thomas Strothotte. „RenderBots -- Multi Agent Systems for Direct Image Generation“. In: *COMPUTER GRAPHICS FORUM* 24.2 (2005), S. 283–290.

[SH10] Sebastian Schäfer und Carsten Heep. „3D-Oberflächen-Rekonstruktion und plastisches Rendern aus Bilderserien“. In: *16. Workshop Farbbildverarbeitung*. Ilmenau, Germany, 2010, S. 193–204. ISBN: 978-3-00-032504-5.

[SH11] Sebastian Schäfer und Carsten Heep. „Surface reconstruction and artistic rendering of small paleontological specimens“. In: *International Symposium on Non-Photorealistic Animation and Rendering 2011*. Poster, 2011.

[Shi05] Peter Shirley u. a. *Fundamentals of Computer Graphics, Second Ed.* Natick, MA, USA: A. K. Peters, Ltd., 2005. ISBN: 1568812698.

[SKK12a] Sebastian Schäfer, Christoph Knopp und Detlef Krömker. „A GPU-based approach for scientific illustrations“. In: *15. Anwendungsbezogener Workshop zur Erfassung, Modellierung, Verarbeitung und Auswertung von 3D-Daten*. Berlin, Germany, 2012, S. 97–105. ISBN: 978-3-942709-07-1.

[SKK12b] Sebastian Schäfer, Christoph Knopp und Detlef Krömker. „Interactive Generation of (Paleontological) Scientific Illustrations from 3D-models“. In: *XXII Spanish Computer Graphics Conference*. Hrsg. von Isabel Navazo und Gustavo Patow. Jaén, Spain: Eurographics Association, 2012, S. 109–112. ISBN: 978-3-905673-92-0.

[SKK12c] Sebastian Schäfer, Christoph Knopp und Detlef Krömker. „Interactive generation of (paleontological) scientific illustrations from 3D-models“. In: *ACM SIGGRAPH 2012 Posters*. (Los Angeles, California). SIGGRAPH '12. New York, NY, USA: ACM, 2012, 106:1–106:1. ISBN: 978-1-4503-1682-8.

[SN10] Sebastian Schäfer und Frank Nagl. „3D-SurReAL - 3D Surface Reconstruction of Arbitrary (Image) Layer“. In: *Visual Computing for Biology and Medicine 2010, Eurographics Workshop*. Poster, 2010.

[SS02] Thomas Strothotte und Stefan Schlechtweg. *Non-photorealistic computer graphics: modeling, rendering, and animation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002. ISBN: 1-55860-787-0.

[ST90] Takafumi Saito und Tokiichiro Takahashi. „Comprehensible rendering of 3-D shapes“. In: *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. (Dallas, TX, USA). SIGGRAPH '90. New York, NY, USA: ACM, 1990, S. 197–206. ISBN: 0-89791-344-2.

[SU08] L. Szirmay-Kalos und T. Umenhoffer. „Displacement Mapping on the GPU - State of the Art“. In: *Computer Graphics Forum* 27.1 (2008).

[Sýk14] Daniel Sýkora u. a. „Ink-and-Ray: Bas-Relief Meshes for Adding Global Illumination Effects to Hand-Drawn Characters“. In: *ACM Transaction on Graphics* 33.2 (2014), S. 16.

[Tie08] Christian Tietjen u. a. „Hardware-Accelerated Illustrative Medical Surface Visualization with Extended Shading Maps“. In: *Proceedings of the 9th international symposium on Smart Graphics*. (Rennes, France). SG '08. Berlin, Heidelberg: Springer-Verlag, 2008, S. 166–177. ISBN: 978-3-540-85410-4.

[Tur07] Ivan S. Turgenew. *Väter und Söhne*. Insel Verlag, 2007. ISBN: 978-3458352129.

[Ver08] Romain Vergne u. a. „Shading with Apparent Relief“. English. In: *SIGGRAPH Talk Program*. ACM. Los Angeles, United States, 2008.

[Ver10] Romain Vergne u. a. „Radiance Scaling for Versatile Surface Enhancement“. English. In: *I3D '10: Proc. symposium on Interactive 3D graphics and games*. Boston, United States: ACM, 2010.

[Ver73] Andreas Versalius. *The Illustrations from the Works of Andreas Vesalius of Brussels*. Hrsg. von Charles O'Malley J. B. Saunders. Dover Publications, 1973. ISBN: 978-0486209685.

[WK95] Sidney W. Wang und Arie E. Kaufman. „Volume sculpting“. In: *Proceedings of the 1995 symposium on Interactive 3D graphics*. I3D '95. Monterey, California, USA: ACM, 1995, 151–ff. ISBN: 0-89791-736-7.

Lebenslauf

Persönliche Daten

Name	Sebastian Andreas Schäfer
Geburtsdatum	12.11.1980
Geburtsort	Friedberg

Beruflicher Werdegang

2013 - dato **Senior Software Engineer**, TomTom Ltd., London, UK
Maprendering & Visualisation

2012 - 2013 **Freiberuflicher Softwareentwickler**

2007 - 2012 **Wissenschaftlicher Mitarbeiter**, Goethe-Universität Frankfurt
Professur für Graphische Datenverarbeitung, Prof. Dr. Krömker
Betreute Veranstaltungen:

- Grundlagen der Computergrafik
- Animation
- Visual Computing Praktikum

2010 - 2011 **Dozent (Lehrauftrag)**, SRH Heidelberg
Veranstaltungen im Studiengang *Game-Development*:

- Programmiervertiefung 2
- Games-Engines & Middleware

2002 - 2006 **Hilfswissenschaftliche Kraft**, Goethe-Universität Frankfurt
Tutor in verschiedenen Informatikveranstaltungen:

- Praktische Informatik 1
- Praktikum Praktische Informatik
- Theorethische Informatik 1 & 2
- Telematikpraktikum

Studium & Schule

2007 - 2012	**Wissenschaftlicher Mitarbeiter**, Goethe-Universität Frankfurt Promotionsstudent, Betreuer: Prof. Dr. Krömker Forschungsschwerpunkte: • Physikalisch-basiertes Realtime-Rendering • GPU-basierte Oberflächenrekonstruktion • Non-Photorealistic-Rendering
2000 - 2007	**Diplom-Studium**, Goethe-Universität Frankfurt Hauptfach: **Informatik** Nebenfach: **Experimentelle Physik** Diplomarbeit: **Bumpmapping-Verfahren und deren Weiterentwicklung**
1998 - 2000	**Abitur**, Otto-Hahn-Schule Hanau Leistungskurse: **Informatik & Physik**

Ehrenamtliche Tätigkeiten

2012 - dato	Reviewer für *Computers & Geosciences*, Elsevier

29. Juni 2015

Eigene Publikationen

[SH10] Sebastian Schäfer und Carsten Heep. „3D-Oberflächen-Rekonstruktion und plastisches Rendern aus Bilderserien“. In: *16. Workshop Farbbildverarbeitung*. Ilmenau, Germany, 2010, S. 193–204. ISBN: 978-3-00-032504-5.

[SH11] Sebastian Schäfer und Carsten Heep. „Surface reconstruction and artistic rendering of small paleontological specimens“. In: *International Symposium on Non-Photorealistic Animation and Rendering 2011*. Poster, 2011.

[SKK12a] Sebastian Schäfer, Christoph Knopp und Detlef Krömker. „A GPU-based approach for scientific illustrations“. In: *15. Anwendungsbezogener Workshop zur Erfassung, Modellierung, Verarbeitung und Auswertung von 3D-Daten*. Berlin, Germany, 2012, S. 97–105. ISBN: 978-3-942709-07-1.

[SKK12b] Sebastian Schäfer, Christoph Knopp und Detlef Krömker. „Interactive Generation of (Paleontological) Scientific Illustrations from 3D-models“. In: *XXII Spanish Computer Graphics Conference*. Hrsg. von Isabel Navazo und Gustavo Patow. Jaén, Spain: Eurographics Association, 2012, S. 109–112. ISBN: 978-3-905673-92-0.

[SKK12c] Sebastian Schäfer, Christoph Knopp und Detlef Krömker. „Interactive generation of (paleontological) scientific illustrations from 3D-models“. In: *ACM SIGGRAPH 2012 Posters*. (Los Angeles, California). SIGGRAPH '12. New York, NY, USA: ACM, 2012, 106:1–106:1. ISBN: 978-1-4503-1682-8.

[SN10] Sebastian Schäfer und Frank Nagl. „3D-SurReAL - 3D Surface Reconstruction of Arbitrary (Image) Layer“. In: *Visual Computing for Biology and Medicine 2010, Eurographics Workshop*. Poster, 2010.

Auszeichnungen

- NPAR 2011: Honorable Mentions - Poster
- SIGGRAPH 2012: Semi-Finalist ACM Student Research Competition

Weitere Veröffentlichungen

GI-Anis Kalender 2011

- Februar: Voxeltracing
- November: Oberflächenrekonstruktion

Anhang E

Betreute Abschlussarbeiten

Im folgenden ein Verzeichnis der vom Autor betreuten Abschlussarbeiten.

E.1 Diplomarbeiten

- Christoph Knopp (2011)
  Stipplingalgorithmen für paleontologische Fundstücke
- Jörg Karpf (2009)
  Visuelle Simulation eines Radiosity Algorithmus und ihre Anwendung in Lernprozessen
- Patrick Guttner (2008)
  3D-unterstützte Bildmanipulation
- Carsten Heep (2008)
  Erweiterungen des Seam Carving
- Daniel Schiffner (2008)
  Wellenlängenbasiertes Beleuchtungsmodell in Shadern

E.2 Bachelorarbeiten

- Markus Strobel (2012)
  Evaluation eines Programmes zur Erstellung wissenschaftlicher Illustrationen
- Thomas Wiegand (2011)
  GUI für GPU Bildfiltergraphen
- Silvia Corinna Walk (2011)
  3D-Oberflächenrekonstruktion aus Bilderstapeln
- Arbin Babazadeh (2010)
  NPR von rekonstruierten Artefakten

- Benjamin Bronder & Christian Hornisch (2010)
  Interactive Gorilla
- Stefan Bechtold & Marc Czerepan (2010)
  Rendering octree managed voxel volumes
- Thorsten Gattinger (2009)
  Shader zur Bildbearbeitung
- Christoph Leineweber (2009)
  Entwicklung einer GUI für einen Shaderviewer
- Ji Han (2008)
  Shader IDE in Eclipse