UNIVERSITY OF APPLIED SCIENCES
HOCHSCHULE MITTWEIDA

Angewandte Computer- und Biowissenschaften

Professur Medieninformatik

Bachelorarbeit

Konzeption und Umsetzung einer Software zur Systematisierung von im Kontext der Replikation zweideminsionaler Grafikstile in einer 3D-Pipeline notwendigen Techniken

Lars Bönsch

Mittweida, den 21. Mai 2019

Erstprüfer: Prof. Dr. rer. nat. Marc Ritter
Zweitprüfer: Manuel Heinzig M. Sc.

Bönsch, Lars
Konzeption und Umsetzung einer Software zur Systematisierung von im Kontext der Replikation zweideminsionaler Grafikstile in einer 3D-Pipeline notwendigen Techniken
Bachelorarbeit, Angewandte Computer- und Biowissenschaften
Hochschule Mittweida– University of Applied Sciences, Mai 2019

Referat

Diese Arbeit behandelt die Replikation von 2D-Ästhetik Mittels einer 3D-Production Pipeline.

Name: Bönsch, Lars
Studiengang: Medieninformatik und interaktives Entertainment
Seminargruppe: Mi15w1-B
English Title: Design and Implementation of a software to systematizatize of 3D-Pipeline-Tools needed for replication of two dimensional aesthetic

Inhaltsverzeichnis

1. Einführung und Motivation

Kreatives und Technisches ist schwer miteinander zu vereinen. Diese Aussage hat der Autor sinngemäß in vielen persönlichen Gesprächen gehört und teilweise hat sich diese auch in Projekten, bei denen er teilgenommen hat bestätigt:

Im fünften Semester des Studiengangs Medieninformatik und interaktives Entertainment an der Hochschule Mittweida gibt es das Modul „Applied Game Development“. Hier arbeiten alle Studenten des aktuellen Matrikels zusammen, um in 3 Monaten den Prototyp eines Videospiels auf die Beine zu stellen.

Dadurch stellen sich dem Team nicht nur viele kreative Aufgaben, wie die visuelle und erzählerische Konzeption einer Spielwelt, das entwerfen einer Spielmechanik oder die Erstellung von 3D-Modellen; auch werden hohe Anforderungen an technische Fertigkeiten für die Implementierung der Spiellogik in *C#* und das Programmieren von *Shadern*, um das Spiel visuell aufzuwerten oder die Zusammenführung von allen Modellen, Texturen und Code in einer Game Engine.

Das Matrikel Mi15 hat dabei das Spiel „Elemates“ entwickelt und im Modul hat der Autor hin und wieder Konflikte an der Schnittstelle zwischen kreativen und technischen Aufgaben beobachtet: nicht alle Texturen hatten für die *Engine* praktische Zweierpotenzen als Auflösung, Mitglieder des Teams GameDesign wirkten oft zögerlich, entworfene Spielmechaniken schnell in der Engine zu testen und *Coder Art* wurde dem Autor als Begriff für sehr provisorische und ästhetisch unschöne Platzhaltergrafiken gelehrt.

In der Produktion von Elemates gab es jedoch auch viele positive Beispiele für Ergebnisse, die erzielt werden können, wenn diese Schnittstelle besser funktioniert. Eine wichtige Aufgabe in der Finalisierung des Projektes war es, Prototypen der Level zu gestalten. Und die Gestaltung eines Levels benötigt zweifellos kreative Fertigkeiten, es ist allerdings auch Scripting notwendig, damit die Umgebung auf Aktionen des Spielers reagieren kann. Daher hat eine Programmiererin ein Script geschrieben, mit dem ein Level Artist jede öffentlich zugängliche Variable von Objekten in der Szene

verändern konnte, sobald der Spieler einen *Trigger* auslöst.
Ohne das technische Wissen, welches erforderlich ist, um dieses Tool zu schreiben und ohne die vielen kreativen Ideen für Stellen, an denen man dieses Tool einsetzen könnte, hätte es viele Reaktionen der Spielwelt auf den Spieler nicht gegeben.

In dieser Arbeit soll eine solche Schnittstelle entworfen werden: Dem Autor ist aufgefallen, dass viele 3D-Grafiken zwar technisch brilliant und beeindruckend sind, aber oft in stilistischen Aspekten dem Autor weniger zusagen, als vergleichbare Arbeiten von 2D-Grafikern. So steht die Überlegung im Raum, ob es möglich ist, einige Elemente aus dem Repertoire von 2D-Artists auch in der 3D-Grafik Nutzen zu können.

1.1. Aufgabenbeschreibung

Ziel dieser Arbeit ist die Konzeption einer Anwendung, mit deren Hilfe man eine an zweidimensionale Kunst erinnernde Ästhetik in einer modernen *3D-Production Pipeline* replizieren kann. Dabei wird untersucht, wie Arbeitsabläufe für die Erstellung zweidimensionaler Kunstwerke funktionieren und, wie dies auf eine moderne *3D-Produktion Pipeline* übertragen werden kann.

Dies lässt sich in zwei Teilbereiche untergliedern: Zum einen soll eine dynamische Bibliothek aus Methoden und Techniken, die automatisiert in ein Workflow-Wiki hinzugefügt werden können, entworfen werden. Da die Auswahl der hinzuzufügenden Methoden sich nach den stilistischen Vorstellungen des Nutzers richten soll, ist es ebenfalls wichtig, ein Interface zu schaffen, mit dem diese Bibliothek angesprochen werden kann.

Des weiteren soll das entworfene Tool umgesetzt und zur Evaluation mit einem kleinen Datensatz befüllt werden.

1.2. Methodik und Aufbau der Arbeit

Um ein in Kapitel 1.1 beschriebenes System zu konzipieren, ist es notwendig zuerst die Anforderungen zu ermitteln.

Diese Anforderungen werden ermittelt, in dem mehrere Abläufe zur Erstellung von zweidimensionaler und dreidimensionaler Kunst miteinander verglichen werden.

Zunächst wird der Verfasser beispielhaft zwei Kunstwerke im verschiedenen 2D-Stilen anfertigen und den Arbeitsablauf als *Pipeline* dokumentieren. Ebenfalls werden Vergleiche zu anderen Leitfäden gezogen. Die beiden Grafiken werden anschließend mit Hilfe der 3D-Software *Blender* repliziert. Auch hier sind die Arbeitsabläufe als Pipeline zu dokumentieren. Ein Vergleich mit andern Leitfäden zur Erstellung von 3D-Grafiken findet ebenfalls statt.

Nun können die Arbeitsabläufe miteinander verglichen werden. Wesentliche Fragen hierbei sind:

- Wie unterscheiden sich die Arbeitsabläufe bei der Erstellung verschiedener 2D-Grafiken voneinander?
- Wie unterscheiden sich die Arbeitsabläufe bei der Erstellung verschiedener 3D-Grafiken voneinander?
- Wie wirken sich Entscheidungen im traditionell zweidimensionalen kreativen Prozess auf den Arbeitsablauf in einer *3D-Pipeline* aus?

Anschließend soll ein System postuliert und als Programm umgesetzt werden, welches die verschiedenen *Pipelines* abbilden kann und bei Beschreibung einer *2D-Pipeline* bei der Konzipierung einer *3D-Pipeline* behilflich sein kann.

Die Evaluation des Systems passiert durch einen Test, in dem Probanden unter Zuhilfenahme des Systems die in der Bachelorarbeit angefertigten 2D-Grafiken mittels 3D Software replizieren sollen.

Anschließend füllen die Probanden einen kleinen Fragebogen aus und die Werke werden in der Arbeit als Dokument angehängt.

1.2.1. Disclaimer

Diese Arbeit wurde zwar überwiegend in der Reihenfolge der Gliederung konzipiert, jedoch nicht in Reihenfolge der Gliederung geschrieben. So wurde die Implementation aus Planungsgründen durchgeführt, bevor das Kapitel 3.1 vollständig verfasst wurde. Leider ist dadurch ein konzeptioneller Fehler erst bei der Finalisierung von

Kapitel 3.1 und der parallel stattfindenden Vorbereitung der Anwendung zur Evaluation bemerkt wurden. Mehr dazu in Kapitel 3.3.1-2 und 7.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Verfassung dieser Arbeit unterstützt und motiviert haben. Ohne Menschen, wie Til, den ich immer wieder konsultieren konnte, wenn ich mit meinem Code gegen die Wand gefahren bin, den Mitbewohnern meiner WG, mit denen ich meine Überlegungen immer wieder validieren konnte, den Mitarbeitern der HS-Mittweida und der sächsischen Landesunibibliothek, die ich nerven durfte, sowie der Open Source Community, auf deren Errungenschaften ich aufbauen konnte, wäre diese Arbeit in so nicht möglich gewesen.

Abbildung 2.1.: Beispiele von 3D-Grafik (private Arbeiten des Autors)

2. Grundlagen

In diesem Kapitel werden alle für das Verständnis dieser Arbeit erforderlichen Begriffe definiert und Grundlagen erklärt. Dies beinhaltet Definitionen zu dem, was in dieser Arbeit als 3D- und 2D-Grafik verstanden wird, eine Einführung in Webentwicklung unter *Python und Flask* sowie Graphen.

Zunächst ist darauf zu verweisen, dass viele Begriffe nicht einheitlich definiert sind. Es ist daher möglich, dass in dieser Bachelorarbeit verwendeten Begriffe in anderen Kontexten andere Bedeutung haben und, dass es für hier beschriebene Konzepte abweichende Bezeichnungen gibt.

2.1. 3D-Grafik

3D-Grafik sei in dieser Arbeit ein Begriff für einen Teilbereich von *Computer generated*, also der Computergestützen Synthese von Bildern. [Flu08, S.24] [AN11, S.2] Ein wesentliches Merkmal der 3D-Grafik ist, dass die dargestellten Objekte in drei Dimensionen definiert sind und so auch aus allen Blickwinkeln betrachtet werden können. [Flu08, S.51]

Als primäre Quelle für diverse Beispiele von 3D-Grafik sei an dieser Stelle die Kategorie „3D-Art“ der Webseite artstation.com zu nennen.

3D-Grafiken können unter Zuhilfenahme einer *3D-Production Pipeline* erstellt werden.

Im Allgemeinen bezeichnet eine *Production Pipeline* ein Konzept zur Arbeitsteilung bei der Produktion von Gütern. Ein anschauliches Beispiel ist die Montur eines Automobils. Mehrere Schritte werden dabei von verschiedenen Mitarbeitern nacheinander am gleichen Werkstück getätigt. So können sich die einzelnen Mitarbeiter auf ihren jeweiligen Fachbereich besser spezialisieren und so effizienter arbeiten.

Dieses Konzept kann jedoch auch makroskopisch und mikroskopisch auf das Erstellen eines Spiels oder Films übertragen werden. In einem humorvollen Lehrfilm bezeichnet DreamWorks die gesamte Produktion eines Films angefangen bei dem Drehbuch und abgeschlossen mit einem fertigen Film als *Pipeline*. [TJI]

Jedoch können auch Algorithmen, die aus bestimmten Daten (im Falle des klassischen *3D-Renderings* sind das in der Regel Texturen, Lichter, *Shader*, Animationen und 3D-Modelle) Bilddateien errechnen als *Render-Pipeline* bezeichnet werden.

Bei der Erstellung von Computergestützen 3D-Grafiken wird dabei meist eine starre Folge von Arbeitsschritten abgegangen, die hier als *3D Production Pipeline* bezeichnet wird. [Flu08, S. 25]

Dabei sind die wesentlichen, technisch notwendigen Bestandteile der meisten klassischen 3D Production Pipeline folgende: (Übersicht basiert auf [Pel, S. 25])

- **Modelling**
  Auf Basis von Konzepten wird die Form von Objekten im dreidimensionalen Raum konzipiert. Dabei müssen die Objekte ästhetischen Anforderungen genügen, jedoch auch technische Vorraussetzungen für spätere Arbeitsschritte erfüllen.

  Sobald ein Objekt deformierbar sein soll stellen sich in der Regel zusätzliche Anforderungen an das Modell.

- **Surfacing**
  Die Definition von Oberflächeneigenschaften und Texturen der darzustellenden Objekte.

Dabei ist es teilweise auch notwendig, die physikalischen Gegebenheiten der Interaktion von Objekten mit Licht und deren Auswirkung auf die Wahrnehmung durch den Menschen zu verstehen.

- **Rigging**
  Die statischen, im *Modelling* entstandenen Objekte werden mit Systemen versehen, mit denen man die Geometrie manuell bewegen kann.

- **Set Dressing oder Environment Design**
  Verschiedene Objekte werden zu einer virtuellen Umgebung zusammengefügt. Der Begriff *Set Dressing* wird in Kontext von Filmen und vergleichbaren Anwendungen häufiger verwendet, während *Environmnt Art* oder *Environment Design* in interaktiven Anwendungsfällen gängiger ist.

- **Animation**
  Das Definieren von zeitabhängigen Veränderungen der Objekte. Meist werden hierzu nur manuelle Animationsmethoden gezählt, technisch gesehen zählen jedoch auch oft separat betrachteten prozeduralen Methoden, wie zum Beispiel Simulationen dazu.

- **Lighting**
  Mit Hilfe von virtuellen Lichtquellen werden Objekte beleuchtet.

- **Rendering**
  Algorithmen synthetisieren aus den in der *Pipeline* entstandenen Rohdaten zweidimensionale Bilder.

2.2. 2D-Grafik

Der Begriff der *2D-Pipeline* kann prozedural ähnlich abgeleitet werden, ist jedoch im Verlaufe der Literaturrecherche nicht in dieser Form gebraucht worden, weswegen dieser nun wie folgt definiert wird:

2D-Grafik ist ein Begriff für eine Teilmenge an Darstellungsformen der bildenden Kunst. Diese haben dabei folgende Charakteristika:

- Erstellung und Wirkung des Werkes findet wesentlich auf einer zweidimensionalen Ebene statt.

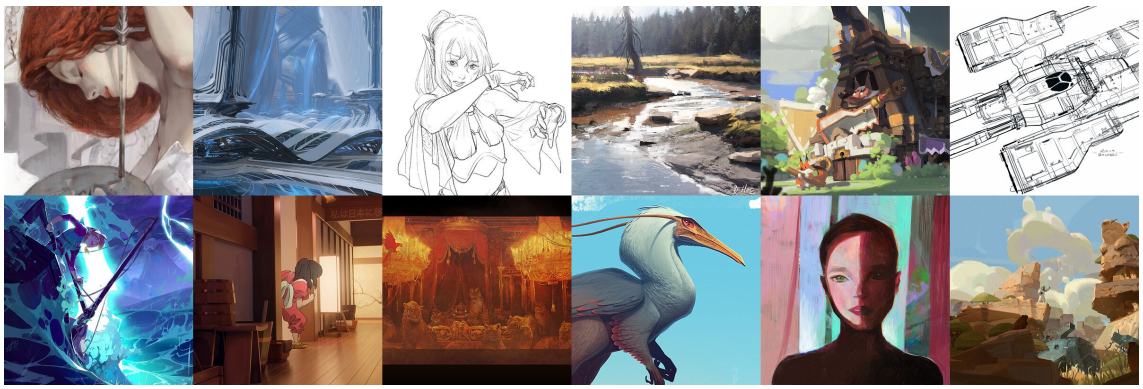

Abbildung 2.2.: Beispiele von 2D-Grafik [MS18, S. 25]

- Die Form der wesentlichen Elemente wird direkt durch menschliche, kreative Entscheidungen hervorgebracht.

So werden in dieser Arbeit beispielsweise Ölgemälde, Aquarelle und Graffiti, jedoch auch 2D-Vektorgrafiken und mit Hilfe von 2D-Grafikprogrammen erstellte Zeichnungen als 2D-Grafik bezeichnet. Fotografien, 2D-Fraktale oder auch Materialbilder werden dabei von dieser Definition ausgenommen.

Als 2D-Pipeline wird hier eine algorithmische Beschreibung des Entstehungsprozesses einer 2D-Grafik bezeichnet.

2.3. Grundlagen Web-Entwicklung

In der Evaluation der verschiedenen Tools ergab sich, dass die hier postulierte Anwendung am besten als Webseite implementiert werden solle. Daher werden hier die für das Verständnis der Arbeit notwendigen Begriffe der Web-Entwicklung erläutert:

Das für die Definitionen herangezogene Grundlagenbuch erklärt die wesentlichen Konzepte der Webentwicklung zwar gut, verwendet jedoch teilweise nicht mehr gebräuchliches Vokabular. So wurden einige Begriffe durch den Autor ersetzt.

- **Client und Server**
  Als *Server* wird in einem Netzwerk ein Computer bezeichnet, der anderen

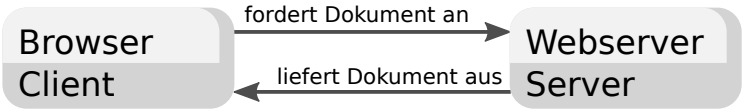

Abbildung 2.3.: Übersicht Client-Server und Webserver-Webbrowser

Rechnern Ressourcen zur Verfügung stellen kann. Nach dem *Client/Server-Paradigma* sind *Clients* die Rechner, die die Initiative ergreifen und von Servern Dienste anfordern. [CM04, S.3]

- **Webbrowser und Webserver**
  Ein Browser ist eine interaktive Anwendung, mit dem Endnutzer auf einem *Client* -PC zugreifen kann. [CM04, S.13]
  Ein *Webserver* hingegen ist ein Programm, dass auf einem Server ausgeführt werden kann, um Anfragen zu verarbeiten. [CM04, S.52]
  Dieses Prinzip ist auch in Abbildung: 2.3 verdeutlicht.

  Prinzipiell kann dabei mit Browsern und Webservern jeder an ein Netzwerk angeschlossener Computer als *Client* und Server fungieren.

- **URL - Uniform Resource Locator**
  Eine *URL* ist wie folgt aufgebaut:

  protokoll://domainname.tld:port/pfad

  – Das erste Segment bestimmt das Protokoll und spezifiziert dadurch die Art der Datenübertragung.

  – Mit Hife von *Domainname* und *Top Level Domain* (Im Beispiel als .tld abgekürzt) kann die Anfrage einem Server zugeordnet werden.

  – Der letzte Teil der *URL* beruft sich ursprünglich auf den Pfad des Dokumenten in der Dateistruktur des *Servers*. Moderne *Weberserver* können diese Informationen jedoch auch anders interpretieren.

  [CM04, S.21]

- **HTML und CSS**
  Die *Hypertext Markup Lanuage* ist eine Sprache, die das Gliedern und Ausgestalten von Dokumenten sowie das Einbinden von multimedialen Inhalten

ermöglicht. [CM04, S.17]
Ein *Cascading Style Sheet* hingegen kann genutzt werden, um die Formatierung der Elemente von HTML zu definieren. [CM04, S.880]

2.4. Python und Flask

Für den in dieser Arbeit implementierten *Webserver* entschied sich der Verfasser für die Verwendung des Frameworks *Flask* unter *Python 3*. Die Gründe dafür werden in Kapitel 5.1.2. ausführlicher erläutert.

Python ist eine Programmiersprache, deren Stärken laut Johannes Ernesti und Peter Kaiser in ihrer Einfachheit, Flexibilität und Eleganz liege. So eigne sie sich gut, um schnell Anwendungen mittels Rapid Prototyping zu entwickeln. [JE12, S. 19, 30]

Python hat zwar viele Ähnlichkeiten mit üblichen Spcriptsprachen, es handelt sich jedoch dennoch um eine interpretierte Programmiersprache: Ein Compiler muss zunächst Byte-Code generieren, der dann vom Python-Interpreter, einer virtuellen Maschine, ausgeführt werden kann. So ist *Python* ähnlich wie Java stark plattformunabhängig, kann jedoch aufgrund der verfügbaren Python-API auch mit performanteren Sprachen, wie C zusammen wirken. [JE12, S. 28f]

Eine weitere Eigenheit von *Python* ist, dass es sich um eine dynamische Programmiersprache handelt: Variablentypen werden nicht explizit durch den Programmierer gesetzt; *Python* nimmt diesem das *Type-Juggeling* ab.

Flask ist ein so genanntes Microframework für *Python*. Dies alle für die Entwicklung eines Webservers fundamental wichtigen Funktionen abnehmen. *Flask* nutzt dabei für routing, debugging und das Netzwerk-Gateway-Interface von *Werkzeug* und unterstützt *Jinjas 2* Templating. [Mac18, Chapter 1]

Doch wie kann *Flask* für Web-Development genutzt werden? Hierfür werden die im Rahmen dieser Bachelorarbeiten benötigten Funktionalitäten von *Flask* dargelegt:

Um *Flask* nutzen zu können, ist es notwendig, dieses Paket zunächst im Kopf des *Python* Files zu importieren. Des weiteren ist es muss, eine Instanz von *Flask* erzeugt werden.

Wird bei dem im Kopf der Klasse erzeugten Flask-Objekt die `run()` Methode aufgerufen, wird über den Port der angegebenen Host-IP ein *Webserver* gestartet. Empfängt nun dieser Port durch einen *Client* eine Anfrage, kann dies in der Python Klasse Methoden aufrufen.

Um eine Methode aufrufbar zu machen, ist die Zeile @app.route() vor den Methodenkopf zu setzen. („app“ ist in diesem Fall der Name des initiierten Flask-Objektes sein.) Eine fundamentale *Flask-Application* sieht dabei wie folgt aus:

Mit dem hier angefügten Code-Beispiel wird durch Flask zunächst ein Webserver gestartet. Ruft ein Client die korrekte URL unter dem Pfad „/hellworld“ auf, sendet der Webserver ein HTML-Dokument mit dem Inhalt „Hello World“

Listing 2.1: Hello World using Flask

```
1  #Flask wird importiert
2  from Flask import Flask
3
4  #Flask-Objekt wird initiiert
5  app = Flask(__name__)
6
7  #wird ausgefuehrt, wenn domainname.tld/helloworld
8  #von Client  aufgerufen wird
9  @app.route("/helloworld")
10 def helloWorld():
11
12     #uebergibt dem Client-Browser ein HTML-Dokument
13     #mit dem Inhalt "Hello World"
14     return "Hello World"
15
16 #Startet Webserver, wenn File direkt ausgefuehrt wird.
17 if __name__ == '__main__':
18     app.run(host='0.0.0.0', port=80)
```

Der auf Zeile 17 verwendeten Variable `__name__` können Informationen entnommen werden, in welchem Kontext ein Python-File ausgeführt wird. Wird es direkt ausgeführt, ist diese Variable auf `'__main__'` gesetzt.

Durch *Jinjas 2* ist es zudem möglich, serverseitig Code in HTML-Dokumenten auszuführen. Um mit *Flask* den Inhalt aus HTML-Dokumenten übergeben und *Jinjas 2* Templates rendern zu können, kann die statische `renderTemplate()` Funktion von *Flask* genutzt werden:

Das hier angefügte Code-Beispiel übergibt dem `renderTemplate()` eine Zählvariable, die mit jeder Anfrage des Pfades „/helloworld“ um eins inkrementiert wird, um dem Nutzer zeigen zu können, wie oft das Dokument bereits angefordert wurde.

Listing 2.2: Hello World using *Flask* including a visitor counter

```
1  #render_template wird importiert
2  from Flask import Flask, render_template
3
4  #Zaehlvariable wird initiiert
5  counter = 0
6  app = Flask(__name__)
7
8  @app.route("/helloworld")
9  def helloWorld():
10     global counter
11     #inkrementiert Zaehlvariable mit jedem Aufruf
12     counter += 1
13
14     return render_template('helloWorld.html',
15                            counter = counter)
16
17 if __name__ == '__main__':
18         app.run(host='0.0.0.0', port=80)
```

Im HTML-Dokument kann nun auf die Variable zugegriffen werden.

Listing 2.3: helloWorld.html

```
1 <html>
2 Hello world! <br>
3 Du bist der {{counter}}. Besucher dieser Webseite.
4 </html>
```

Da *Flask* als return nur einen String erwartet, ist es zudem möglich, aus mehreren Templates eine HTML-Datei zu rendern:

Listing 2.4: *Python* example

```
1
2 @app.route("/")
3 def homepage():
4     return (render_template("Header.html") +
5             render_template("HomeContent.html") +
6             render_template("Footer.html"))
```

2.5. Graphen

Nach der Graphentheorie ist ein *Graph* eine abstrakte Struktur, die aus *Objekten* oder so genannten *Knoten* besteht, die durch Kanten miteinander verbunden sind. Dabei gibt es *gerichtete* und *ungerichtete Graphen*. In *ungerichteten Graphen* sind dabei die *Kanten* richtungsunabhängig definiert. So könnte man zum Beispiel ein Reisedauerplansystem für Binnenschiffahrt im Spreewald zu großen Teilen mit Hilfe eines *ungerichteten Graphen* darstellen, da durch die vernachlässigbare Strömung eine Reise zwischen zwei Anlegestellen richtungsunabhängig immer genau gleich dauern würde.

Kanten können dabei Eigenschaften aufweisen. Für den zur Veranschaulichung genannten Reisedauerkalkulator wäre es beispielsweise nötig, jeder *Kante* die Zeit zuzuweisen, die im Schnitt benötigt wird, um die Kante mit dem Boot abzufahren. Somit handelt es sich hierbei um *gewichtete Kanten*.

Dabei ist es bei diesem Beispiel jedoch auch sinnvoll, verschiedene Typen von Knoten zu verwenden. Ein *Graph*, der nur die Zeiten zwischen Anlegestellen, zwischen denen der direkte Weg nicht über eine weitere Anlegestelle führt, darstellt, ist anders strukturiert, als ein *Graph*, bei dem sowohl Anlegestellen als auch Flusskreuzungen als *Knoten* existieren. Veranschaulicht ist das in Abbildung 2.4 mit neun fiktiven Anlegestellen auf einem Kartenausschnitt im Spreewald.
(Mit 6 zum besseren Verständnis eingezeichneten und ebenfalls fiktiven Zeiten)

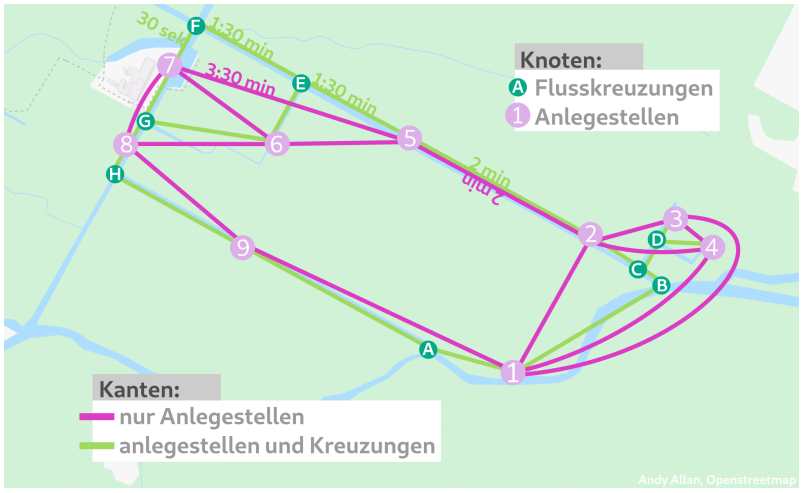

Abbildung 2.4.: Spreewald-Graph [All, lat=51.872, lon=14.012]

Soll das System auf fließende Gewässer ausgeweitet werden, ist hingegen ein *gerichtete Graph* benötigt, um die Gegebenheiten darstellen zu können: nicht nur unterscheiden sich die Reisezeiten durch die Strömung voneinander, manche Stromschnellen sind potentiell auch nur in einer Richtung befahrbar.

Abbildung 2.5 ist eine fiktive Karte, die dies visualisiert. In diesem Beispiel ist des weiteren ein Schwierigkeitsgrad in den Kanten hinterlegt; die Kanten haben eine zweidimensionale Wichtung.

Dependency Graph

In dieser Bachelorarbeit wird auch auf das Konzept eines sogenannten *dependency Graph* zurückgegriffen. Dieses Konzept ist dem Autor als erstes bei dem aufsetzen eines *Linux-Betriebssystems* erklärt wurden. Das Betriebssystem *Linux* basiert zu großen Teilen auf dem *Open Source* Prinzip: jedes Mitglied der Nutzergemeinschaft

Abbildung 2.5.: Fließgewässer-Graph

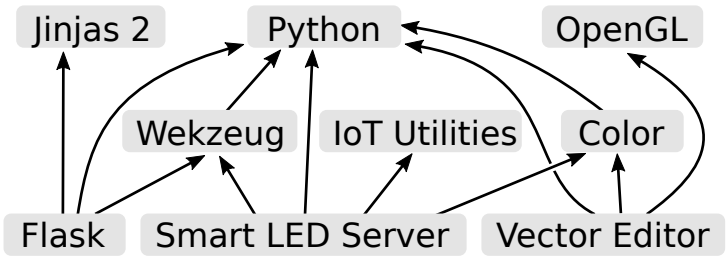

Abbildung 2.6.: fiktiver Dependency-Graph

kann auf der Arbeit anderer profitieren, wenn jeder selbst geschriebenen Code zur Verfügung stellt.

So teilen sich die viele Programme gemeinsamen Quellcode. Um Speicherplatz zu sparen, werden dabei Programme unter Linux in *Pakete* unterteilt. In jedem *Paket* ist eine Liste an weiteren *Paketen* vermerkt, die notwendig sind, damit das Programm installiert und ausgeführt werden kann. Benötigt eine neu zu installierende Anwendung ein *Paket*, welches bereits installiert wurde, muss so der gleiche Code nicht redundant installiert werden. [Ron08, S. 231]

So entsteht ein *Dependency-Graph* aus *Paketen* (Knoten) und deren Abhängigkeiten mit anderen Paketen (Edges).

Wenn zwei gleichzeitig installierte Pakete ein Problem verursachen würden, ist dies durch eine *Kollision* gekennzeichnet. Durch das Prüfen von *Kollisionen* kann so verhindert, dass die Installation eines *Paketes* andere Funktionalität im Betriebssystem beschädigen könnte.

Ein Beispiel für einen fiktiven *Dependency-Graph* ist in Abbildung 2.5 zu sehen. Es ist zu sehen, dass der *Smart LED-Server* und das *Vektorgrafikprogramm* gemeinsam *Color* nutzen. Da der *LED Server* als *smart* bezeichnet wird, benötigt er zudem noch das Paket *IoT Utilities*. Genau, wie *Flask* nutzt der *smart LED-Server* zudem *Werkzeug*. Die meisten Pakete basieren dabei auf *Python* und setzten dies daher ebenfalls voraus.

3. Beispielhafter Vergleich Arbeitsablauf 2D Kunst und 3D-Pipelines

Sowohl 2D-Artists als auch 3D-Artists folgen in jedem Projekt einem Arbeitsablauf. Dieser wird wie in Kapitel zwei erläutert hier als *Pipeline* bezeichnet. Die Entscheidungen, die 2D-Artists und 3D-Artists im Verlaufe einer *Pipeline* treffen, haben dabei Einfluss auf das resultierende Werk. Soll eine sehr spezifische Ästhetik erzielt werden, ist es dabei auch notwendig, die *Pipeline* anforderungsgerecht vom gewollten Endprodukt ausgehend zu planen.

In den folgenden Kapiteln wird daher behandelt, ob und wie allgemeine *2D-Pipelines* systematisiert und dabei die für die Replikation der Ästhetik relevanten Schritte auf die Planung einer *3D-Production Pipeline* übertragen werden können.

Wichtig ist dabei ebenfalls die Untersuchung der Zusammenhänge von Arbeitsschritten in 2D und den erforderlichen Schritten, die in einer *3D-Production Pipeline* unternommen werden müssen, um die Arbeitsschritte zu replizieren.

3.1. Untersuchungen Arbeitsablauf 2D

Die Grundlage der folgenden Zeilen bildet der in Anhang 1 visualisierte Vergleich von aus verschiedenen Quellen stammenden 2D-Pipelines. [Lev18] [Bro18a]

In der genannten Übersichtsgrafik werden drei verschiedene *2D-Pipelines* aufgelistet. Dabei wurden zum einen die Bücher Creating Stylized Charakters und Creating Charakters sowie eine im Rahmen dieser Bachelorarbeit selbst angefertigte Vektorgrafik als Pipeline dargestellt. Daran ist zu erkennen, dass es möglich ist, die Schritte im Entstehungsprozess einer 3D-Grafik als *Pipelines* darzustellen und dass dies auch von dem untersuchten Büchern genutzt wird. Die Überschriften auf der Übersicht entsprechen Arbeitsschritten beziehungsweise Kapiteln aus den Büchern.

Dabei ist die allgemeine Anwendbarkeit der Schritte wesentliches Thema in den Büchern selbst: Beide Bücher stellen eine allgemeine *2D-Pipeline* vor und nutzen diese dann, um verschiedene Beispielprojekte mit unterschiedlichen Ästhetiken nach dem Schema zu erstellen: Zu jedem Schritt in der *Pipeline* trifft ein Künstler laut den Büchern Entscheidungen, die die resultierende Ästhetik der Grafiken beeinflussen.

Abbildung 3.1.: Vektorgrafik-Blumen

Des weiteren ist aufgrund des Vergleiches der drei *Pipelines* erkennbar, dass starke Ähnlichkeiten auftreten. Dies ist in Anhang 1 kenntlich gemacht: alle ähnlichen Schritte der Pipeline sind farblich gleich gekennzeichnet. Der wesentliche Unterschied der in diesem Vergleich behandelten Büchern ist die Kategorisierung: Bei Creating Characters wurden der Pipeline von Creating Stylized Characters sehr ähnliche Schritte in 4 übergeordnete Kapiteln sortiert.

Ein wesentlicher Unterschied, der jedoch beim Vergleich der beiden Bücher und dem Beispiel des Autors auffällt, ist der Fokus auf Konzeptionelles. Die Bücher haben relativ betrachtet deutlich mehr Fokus auf Recherche, Proportionen, Skizzierung, Iteration und Erzählerisches gelegt. Als möglicher Grund dafür ist allerdings die Zielsetzung der Arbeiten zu nennen.

Die Autoren der Bücher hatten das Ziel, einem breiten Publikum das Erstellen eines Charakters näher zu bringen. Insbesondere Creating Charakters geht dabei auch auf viele mit der Umsetzung eines größeren kreativen Projektes verbunden Aspekte, wie Zielgruppenanalyse, Projektrahmen oder Konzeption von Geschichte und Welt ein. Diese Aspekte haben zwar Einfluss auf die kreativen Entscheidungen, die in der *2D-Pipeline* getroffen werden, können jedoch bei der Replikation einer aufgrund dieser Rahmenbedingungen entstandenen 2D-Ästhetik mit einer 3D-Pipeline vorerst vernachlässigt werden.

Die durch den Autor angefertigte Grafik hingegen hatte vorwiegend den Zweck, einem Fachpublikum Charakteristika von 2D- und 3D-Pipelines zu verdeutlichen.

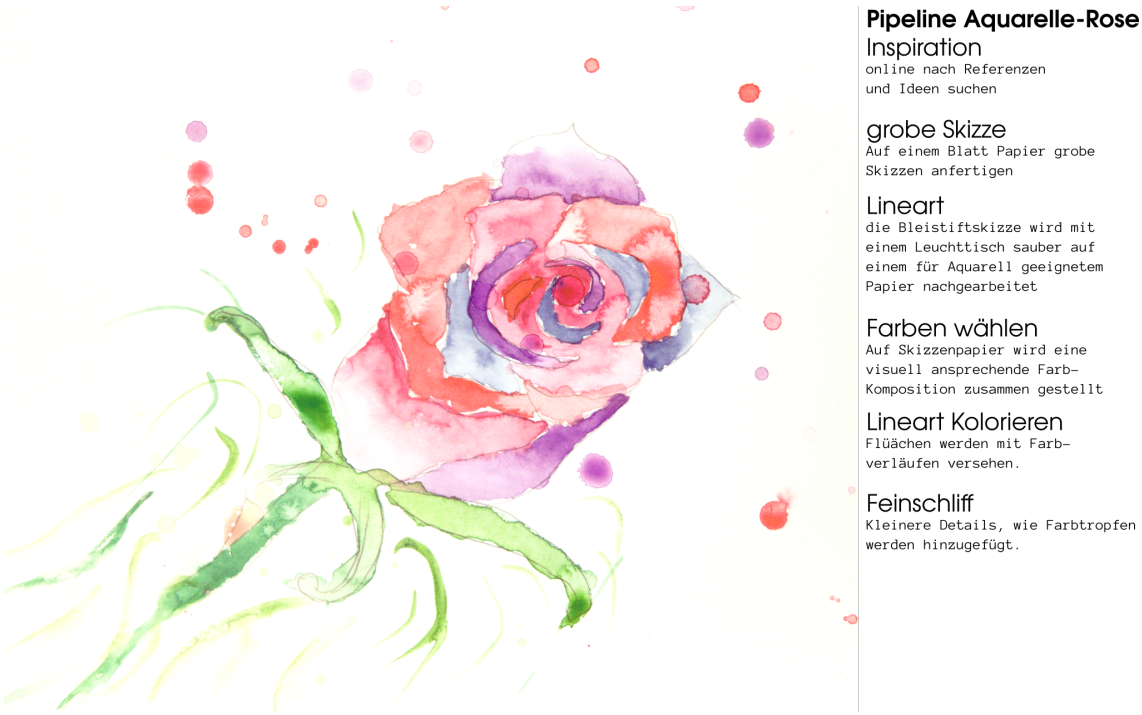

Abbildung 3.2.: Aquarell-Rose und beschriebene Pipeline

So wurde bewusst ein wenig erzählerisches, aber dennoch visuell interessantes Motiv gewählt und daher weniger Zeit in die konzeptionelle Ausarbeitung investieren.

Um einen weiteren Referenzpunkt für 2D-Pipelines zu erlangen, wurde eine zweite 2D-Grafik durch den Autor angefertigt – eine Rose im Aquarell-Stil. Auch die Anfertigung dieser verlief sehr ähnlich zu dem der Vektorgrafik, wie in Abbildung 3.2 zu erkennen ist.

Bei der Anfertigung der Grafiken durch den Autor ist jedoch ein wesentlicher Unterschied im Vergleich zu digitalen Tools aufgefallen und das ist ein Verlust an Kontrolle, der mit dem Arbeiten mit Aquarellfarbe einhergegangen ist. Das Medium Aquarell verhält sich deutlich unvorhersehbarer, als die Programme, die der Autor zur Erstellung von Grafiken genutzt hatte.

3.2. 3D-Production Pipeline

In diesem Kapitel werden Pipelines beschrieben, mit denen der Autor die Resultate der 2D Pipelines in 3D nachzuahmen versuchte.

Wie in Kapitel 2.1. definiert folgen 3D-Pipelines im allgemeinen oft einem ähnlichen Schema. Projektabhängig kann es jedoch im Detail auch starke Unterschiede geben: Je nach Projektrahmen (Größe, Erfahrung, Budget) angestrebter Ästhetik, Zielmedium (Kurzfilm, Kinofilm, Serie, Plakat, Videospiel, Produktvisualisierung), den darzustellenden Objekten und letztendlich auch den persönlichen Erfahrungen der Betiligen 3D-Artists sind bestimmte Reihenfolgen und Methodiken von *Pipelines* unterschiedlich Effizient.

Als Beispiel für diese Aussage sei *Cloth Simulation* angeführt, also die algorithmisch gestützte Erstellung von Kleidungsstücken genannt:

1. Produktionen, in denen keine Charaktere mit Kleidungen vorkommen, können diesen Schritt vollständig ignorieren.

2. Für Produktionen, die eine stark stilisierte Optik erzielen wollen, kann die händische Erstellung von Stoffen ohne algorithmische Unterstützung sinnvoll sein, da dies genauere Kontrolle über das entstandene Erlebnis erlaubt. Dies würde im Schritt des *Modelling* statt finden.

3. Videospiele nutzen so genannte *Normalmaps*, um auf möglichst gering aufgelösten *Polygonmodellen* einen scheinbar höheren Detailgrad zu erreichen.

   Sollen diese *Normalmaps* für die Darstellung von algorithmisch gestützt erstellen Stoffalten produziert werden, sollte zunächst ein nacktes Basismodell des Körpers existieren, der mit Stoff umwickelt werden soll. Für dieses Modell können nun Schnittmuster erstellt werden, die in der Simulation zusammen genähnt werden können und dabei Faltenwurf erzeugen. (3D-Programme verwenden dafür im englischen tatsächlich den Begriff *sewing*) Von dem resultierenden meist hoch aufgelöstem Stoff-Modell kann nun ein niedriger aufgelöstes Modell erstellt und die Details der Stoffsimulation projiziert darauf werden können.

   Hier findet die Stoffsimulation zwischen zwei Schritten der händischen Modellerung statt.

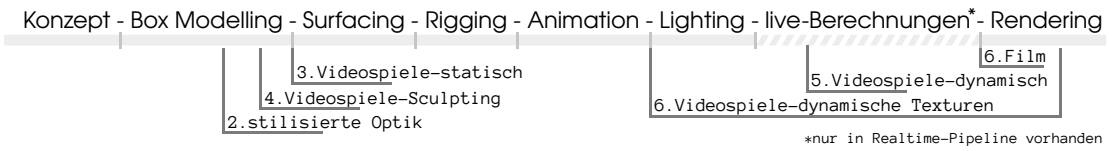

Abbildung 3.3.: Übersicht. An welcher Stelle in einer Pipeline greifen verschiedene Methoden der Cloth Erstellung ein?

4. Abhängig von Erfahrungen der Artists kann es jedoch auch effizienter sein, die Stofffalten mittels händischen Modellierungsmethoden zu erzeugen auch, wenn eine realistische Ästhetik angestrebt wird.

5. Soll die Kleidung in Spielen jedoch dynamisch Flattern, ist es notwendig, in der *Engine* Berechnungen durchzuführen.

6. Viele Filme berechnen Kleidung nach der finalisierten Animation vor dem Rendering. So kann die Stoffsimulation auf die Bewegungen der Figuren reagieren.

7. In manchen Spielen werden jedoch Falten dynamisch bei Animationen durch das Überblenden von verschiedenen *Normalmaps* erzeugt. Dafür ist es notwendig, die Animation anzufertigen, auf Basis dieser ähnlich wie bei Filmen eine Simulation durchzuführen, diese an bestimmten *Frames* in verschiedene *Normalmaps* zu übertragen und dann zur Laufzeit zwischen den einzelnen Texturen zu wechseln.

Abbildung 3.4.: Stoff in Grand Theft Auto: Falten durch Überblenden von Normalmaps [Gam13]

Und das sei nur eine Auflistung von Methoden, die der Autor in eigenen Erfahrungen und durch Berichte von anderen 3D-Artists beziehungsweise Studios gesammelt hat. Es ist anzunehmen, dass eine wissenschaftliche Arbeit, die sich mit diesem Thema im Detail beschäftigt, noch deutlich mehr unterschiedliche Herangehensweisen nennen könnte.

So ist es für einem Vergleich wichtig, einen validen Referenzwert zu haben. Da es sich bei den in diesem Kapitel untersuchten 3D-Pipelines um Strukturen von durch den Autor allein umgesetzten Projekte handelt, wird als Referenz eine für den Autor übliche Pipeline herangezogen.

Wie bereits erwähnt, weichen Projekte oft auch von dieser Pipeline ab, aber Durchschnitt können die Arbeitsschritte des Autors wie folgt beschreiben werden.

- **Inspiration**
  Jede künstlerische Arbeit des Autors benötigt eine Grundidee, von der Ästhetik und Gegenstand des Bildes sowie die darin erzählte Geschichte abgeleitet werden können.
- **Konzept**
  Eine sehr grobe Bleistiftskizze dient zur Visualisierung und Planung der Bildkomposition.
- **Referenz**
  Der Autor sucht auf Kreativwebseiten nach farblich und thematisch der Vision entsprechenden Arbeiten. Die Skizze wird ggf. an neu erlangte Ideen angepasst.
- **Box-Modelling**
  Mit der Methode des Box-Modelling werden zunächst Objekte in grober Form angefertigt. Entspricht die Grundform den visuellen Anforderungen, wird diese nun mit weiterer Geometrie versehen, bis ein für das Projekt geeigneter Detailgrad erreicht ist.
- **Prozedurales Modelling**
  Erfordert dies das Projekt werden bestimmte Objekte und Formen mit Hilfe von prozeduralen Methoden generiert.
- **Surfacing**
  Nun werden die Oberflächeneigenschaften der erzeugten Objekte definiert. Im

so genannten *Shader Editor* von *Blender* werden Farben, Variablen, *prozedurale Texturen* und *Bildtexturen* in einer Graphenstruktur miteinander kombiniert und mit den Eigenschaften von so genannten *Shadern* verknüpft. Mit Hilfe der *Shader* kann die *Render-Engine* von *Blender* im Verlaufe der Bildsynthese die verschiedenen Oberflächeneigenschaften der Objekte simulieren.

- **Simulation und Spezialeffekte**
  Wenn es in der Szene Objekte mit sehr ungewöhnlichen Anforderungen gibt oder bestimmte Visuelle Effekte für die Bildwirkung erforderlich sind, werden diese meistens in diesem Punkt der Pipeline vorgenommen.

- **Lighting und Bildkomposition**
  Zwar wurden für das *Shading* bereits provisorische Lichtquellen angelegt, jetzt wird jedoch die Beleuchtung der wesentlichen Bildelemente gestaltet. Im gleichen Schritt wird auch das Arrangement der wesentlichen Bildelemente festgelegt.

- **Set Dresing**
  Nun werden Umgebung und Bildhintergrund gestaltet. Oft werden Objekte, die nicht primär für die Wirkung des Bildes verantwortlich sind jetzt erst produziert. (Diese durchlaufen nun wiederum die Schritte Referenz, Box-Modelling und *Surfacing*)

- **Rendering**
  Nun wird das Bild für das *Rendering* vorbereitet. Dabei werden für ein Bild oft mehrere *Renderlayer* angefertigt. Oft ist es im nächsten Schritt, dem *Compositing*, nützlich, einzelne Bildinformationen und -Elemente als separates Bild vorliegen zu haben.

- **Compositing**
  Nun werden die verschiedenen *Renderlayer* ebenfalls in einer Graphenstruktur kombiniert und das Bild wird mit Farb- und Helligkeitskorrekturen sowie simulierten Kameraartefakten und weiteren Effekten verfeinert und zu einem finalisierten Endresultat zusammen gefügt.

3.2.1. Vektor Blumen

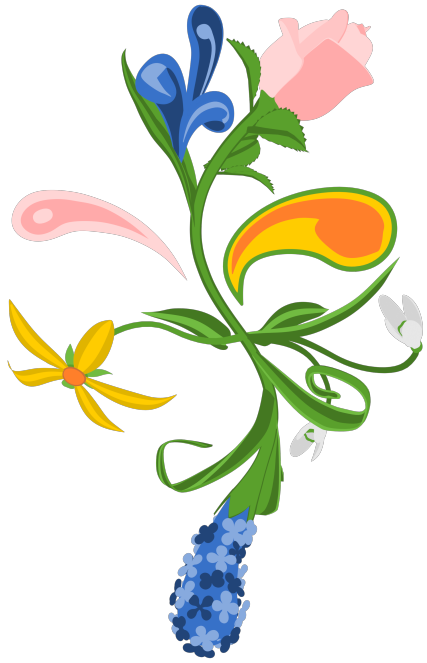

Abbildung 3.5.: Vektor Blumen als 3D-Grafik

Zunächst wird die für die Replikation der Vektorgrafik Blumen genutzte *Pipeline* beschrieben. Dabei werden Unterschiede von der oben beschriebenen *Pipeline* besonders hervorgehoben. Die im letzten Schritt der Pipeline vorgenommenen Änderungen wurden dabei nicht in der 3D-Grafik übernommen.

Inpiration und Konzept
Diese beiden Schritte werden hier nicht behandelt, da durch den Rahmen der Arbeit bereits eine eindeutige Vorlage existierte.

Referenzen
Da bereits eine fertige Illustration existierte, die es genau nachzubilden galt, werden in diesem Schritt vorwiegend technische Referenzen gesucht. Verschiedene Webseiten, auf denen 3D-Artists Arbeiten teilen, wurden nach Beispielen von weiteren ähnlich stilisierten Arbeiten gesucht, um durch genaues Untersuchen Ansätze zu finden, wie man mit den Mitteln der 3D-Grafik die erwünschte Ästhetik erzielen kann.

Modelling
Zunächst wird ein grundlegendes Modell der Geometrie erzeugt. Dafür verwendet der Autor die Methode des Box-Modelling. Nun wird jedoch erneut von einer üblichen Pipeline abgewichen:

Vektorgrafiken sind nicht an realistische Verdeckungen gebunden. Der Baum in Abbildung 3.6 zum Beispiel ist für den Betrachter klar als solcher zu erkennen, verstößt jedoch bei genauerer Betrachtung gegen Logik aus der dreidimensionalen, realen Welt.

In der Grafik verdecken Stamm und Zweige einige wenige dunkelgrüne Blätter, die wiederum die hellgrüne Krone des stilistisch abstrahierten Baums verdecken. Zudem kann dieser Baum auch nur aus einer Perspektive sinnvoll dargestellt werden.

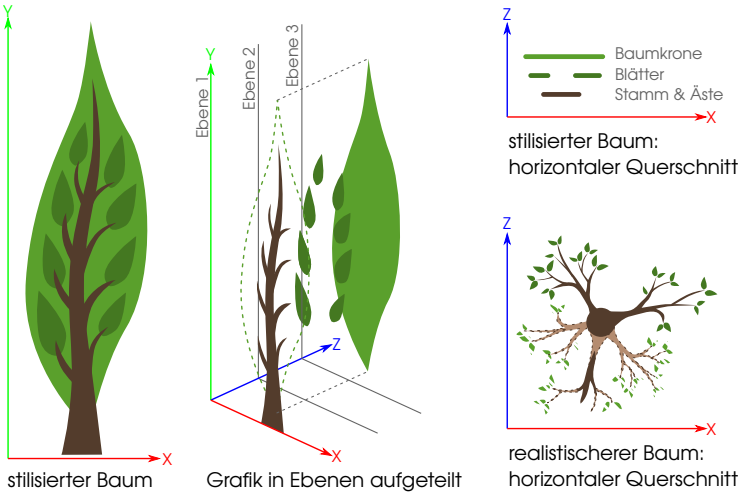

Abbildung 3.6.: Abstratkion in 2D-Art

Bei einer realistischeren Darstellung eines Baums hingegen verdecken aus Sicht des Betrachters zunächst Blätter Teile des Geästs, welches wiederum Sicht auf dem Baumstamm versperrt, welcher auch weiteres Geäst verdeckt, welches wiederum auf der Rückseite des Baums eine weitere Schicht Blätter verdeckt, die die Umgebung des Baums verdecken.

Eine Charakteristik der 3D-Grafik ist jedoch, dass virtuelle Objekte von allen Seiten betrachtet werden können. Das lässt eine Replikation zunächst unmöglich erscheinen.

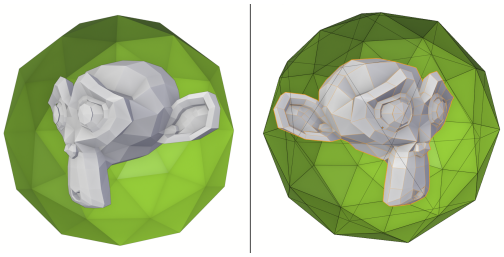

Abbildung 3.7.: Backfacing Trick visualisiert

Zum einen gibt es jedoch trotzdem viele Methoden, mit denen 2D-Grafiken in 3D-Pipelines eingebunden werden können, wie CG Supervisor Chris Browne in einem Video über *Dragon Prince* erklärte. [Bro18b] Es ist auch möglich, unrealistischere Verdeckungen durch das so genannte *Backface Culling* (und andere Methoden) zu replizieren.

Die *Polygone*, aus denen 3D-Modelle bestehen, sind mit einem *Normalenvektor* defi-

niert. So kann beim *Rendering* zwischen Vorder- und Rückseite von *Polygonen* unterschieden werden. Wird nun die Außenseite einer Geometrie als Transparent definiert, kann ein Objekt, welches ein zweites vollständig umschließt dennoch immer im Hintergrund gezeichnet werden.

So wird im Inneren der Objekte zur Replikation der Ästhetik Geometrie erzeugt, die dann durch einseitig transparente Materialien zum Vorschein gebracht werden kann.

Prozedurales Modeling war im Falle dieser Grafik nicht notwendig und wäre vermutlich bei der Anfertigung einer vergleichbaren fotorealistischen Darstellung einer Blumengruppe nicht verwendet wurden.

Surfacing
Das *Surfacing* unterscheidet sich aufgrund der gewünschten Vektorästhetik am stärksten von der Referenzpipeline. Da die Blumen überwiegend unschattiert sind, werden für realistische Blumen übliche *Shader* nicht genutzt. Statt texturierten wird ebenfalls nur auf einfarbige Materealien gesetzt.

Ein weiterer Unterschied besteht darin, dass bei vielen Materialien nun Vorder- und Rückseite der Flächen unterschiedliche Farben und Transparenzen aufweisen. Die dadurch verbundene Gestaltung von wahrgenommenen Oberflächendetails durch im Modell befindlicher Geometrie weicht daher ebenfalls von der Referenzpipeline ab.

Simulation und Spezialeffekte waren in diesem Falle ebenfalls nicht notwendig und wären bei einem ähnlichen Projekt in der Referenzpipeline vermutlich nur bei der Darstellung von Blumen in Verbindung mit Tau, Regen oder Nebel notwendig geworden.

Lighting und Bilskomposition
Durch die nicht schattierten Flächen ist das Aufsetzen einer Beleuchtung nicht notwendig. Die Komposition der wesentlichen Bestandteile wurde zu diesem Schritt jedoch dennoch verfeinert.

Set Dressing
Durch das Fehlen eines Hintergrunds in der Referenzgrafik war zwar eine Dekoration der Umgebung nicht notwendig, es wurden aber in die Grafik zu diesem Schritt dennoch kleine Details zur Aufwertung der Grafik hinzugefügt.

Compositing
war aufgrund der angestrebten Ästhetik nicht notwendig.

3.2.2. Aquarell Rose

Als nächstes wurde beispielhaft eine Rose im Aquarellstil angefertigt. Auch diese Pipeline wird hier erneut mit dem Fokus auf Abweichung von der Beschriebenen Referenz betrachtet.

Inspiration und Konzept
Auch im zweiten Beispiel können diese Schritte aufgrund der klaren Vorlage ignoriert werden.

Referenz
Bei diesem Beispiel hat sich der Autor kaum Arbeiten anderer 3D-Grafiker angesehen, sondern vielmehr das vorliegende Aquarell genau studiert, um Charakteristika dieser Aquarellgrafik ausmachen und replizieren zu können.

Abbildung 3.8.: Aquarell Rose als 3D-Grafik

Box-Modelling
Dieser Schritt unterschied sich kaum von für den in der Referenzpipeline üblichen Arbeitsabläufen. Dabei wurde jedoch von der Referenzgrafik abgewichen: es hat sich bei der Replikation heraus gestellt, dass die angefertigte Aquarellgrafik der Rose perspektivisch keinen Sinn ergibt, da die Öffnung der Blüte wie von oben betrachtet gezeichnet, der Stängel jedoch wie von der Seite betrachtet gezeichnet wurde.

Durch Biegen der Geometrie hätte sich die Form zwar auch in 3D replizieren lassen, der Autor entschied sich jedoch aus ästhetischen Gründen dagegen.

Prozedurales Modelling war auch im zweiten Beispiel nicht notwendig.

Surfacing
Das Surfacing in diesem Fall weicht sehr stark von der Refereznpipeline ab. Die Objekte werden aus technischen Gründen erst im *Compositing geshadet*. Das *gerenderte* Bild dient nur für die Erstellung von Masken im *Compositing*. So sind die *Shader* darauf ausgelegt, dass sie alle für das *Compositing* notwendigen Informationen enthalten und nicht, dass sie visuell ansprechend sind.

Simulation und Spezialeffekte sind auch in dieser Pipeline nicht notwendig gewesen.

Lighting und Bildkomposition
Auch in dieser Arbeit fällt das *Lighting* aufgrund der gewählten Ästhetik weg. Die Bildkomposition beschränkte sich aufgrund der simplen Vorlage und des Projektrahmens auf das Finden einer visuell ansprechenden Perspektive.

Set Dressing entfällt aufgrund der Vorlage.

Rendering
Für dieses Projekt ist es notwendig, die verschiedenen Materialien als Maske dem *Compositor* durch zu reichen. Dies ist in *Blender* durch das verwenden von *MaterialIDs* oder einer *Cryptomatte* möglich. Beide Techniken erlauben es die Einflussbereiche aller Materials als Maske zu speichern.

Compositing
Die wesentliche für die Bildwirkung verantwortliche Arbeit erfolgt im *Compositing*. Dafür werden zuerst für die einzelnen Farben Masken erstellt, die anschließend mit

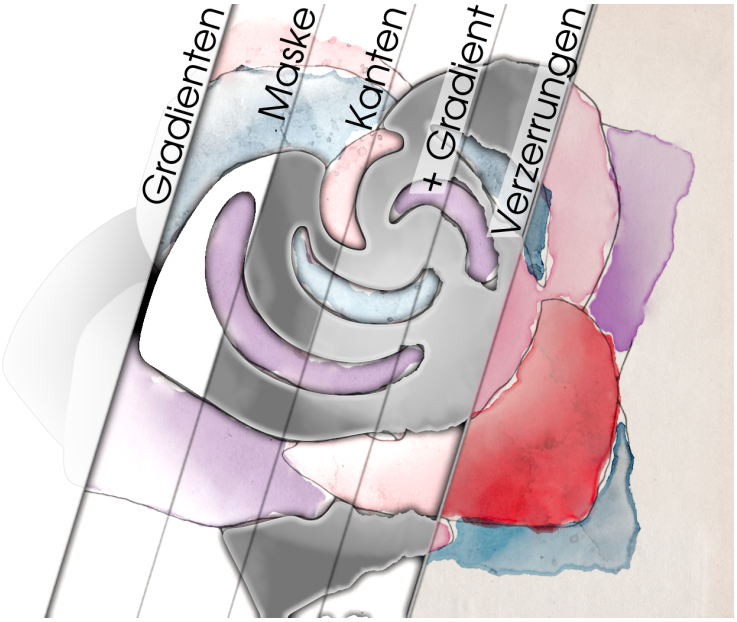

Abbildung 3.9.: Veranschaulichung der Compositing-Schritte

dem *Gaussian Blur-Algorithmus* verwischt werden. Aus dem entstandenen Farbverlauf kann eine neue Kante definiert werden. So wird sich am Rand von Farbflächen sammelnde Farbe simuliert.

Damit die Außenkante nicht zu gleichmäßig wirkt, werden die Farbflächengrenzen mit zwei verschieden eingestellten Gradienten simuliert, die dann nach einer prozedural generierten Textur gemischt werden.

Nun werden im *gerenderten* Bild enthaltene Gradienten additiv hinzugefügt, die Farbverläufe von dünn zu dick aufgetragener Farbe ermöglichen.

Das Ergebnis ist jedoch noch viel zu sauber, um der angestrebten Aquarellästhetik zu entsprechen. Daher wird nun die Maske prozedural verzerrt.

Anschließend wird die Maske genutzt, um eine Textur einer Aquarell-Grafik mit einem Papierhintergrund zu mischen. Auf dieser Papiertextur wurde vorher mit Hilfe der *MaterialIDs* die Grenzen zwischen den Materialien ermittelt und so eine Bleistiftskizze der Rose erzeugt.

3.3. Abbildung traditioneller Arbeitsschritte auf 3D Production Pipeline

Betrachtungen in Kapitel 3.1. haben ergeben, dass verschiedene Arbeitsabläufe eines 2D-Künstlers sich prinzipiell ähneln können und mit einer Pipeline abgebildet werden können. In den jeweiligen Arbeitsschritten trifft ein Künstler nun verschiedene Entscheidungen, die Einfluss auf die Ästhetik der Grafiken hat. In früheren Schritten getroffene Entscheidungen können jedoch auch Einfluss auf spätere Entscheidungen haben. So kann der kreative Prozess eines 2D-Grafikers als Entscheidungsbaum dargestellt werden.

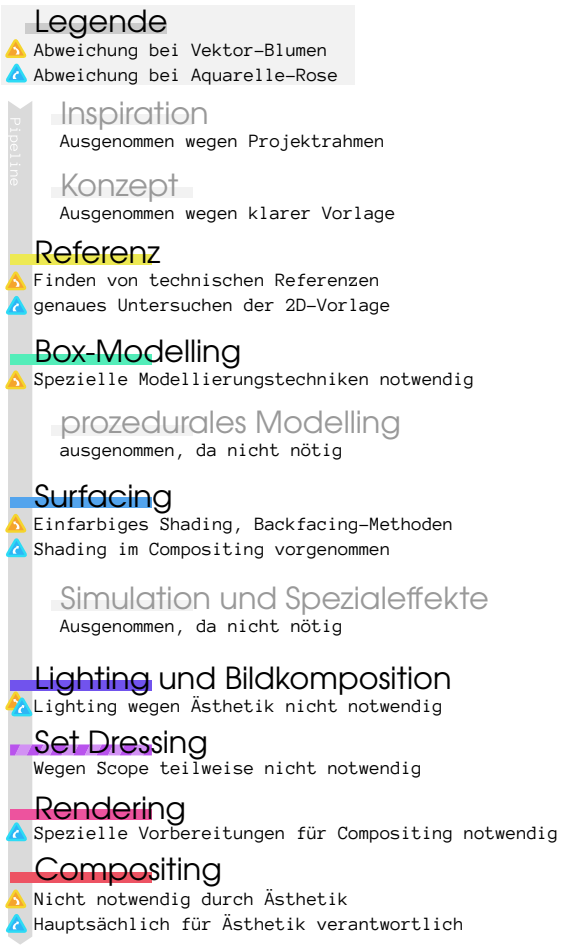

Abbildung 3.10.: Abweichungen von Referenz-Pipeline bei Replikation von 2D-Grafik mittels 3D

Diese kreativen Entscheidungen können dazu führen, dass bei der Replikation mittels 3D-Pipelines Abweichungen vom üblichen Arbeitsablauf auftreten. Dies ist in Abbildung 3.10 zu sehen. Dabei treten vielfältige Zusammenhänge zwischen 2D-Pipeline und 3D-Pipeline auf.

Inspiration, Konzept und die Suche nach Referenzen sind sowohl für 3D-Grafiker als auch für 2D Grafiker prinzipiell sehr ähnliche Schritte. Abweichungen treten dabei vorwiegend auf, weil die 3D-Grafik aufgrund technischer Limitierungen zusätzliche Anforderungen an das Konzept stellen kann. [Bro18a, s.16] Dem Autor ist zudem in der Recherche aufgefallen, dass Leitfäden, die sich der 3D-Grafik widmen relativ gesehen deutlich weniger Fokus auf konzeptionelle Grundlagen, wie Ana-

tomie, Farbenlehre, Charakterkonzeption und ähnliche Aspekte des künstlerischen Prozesses legen. Ein Grund dafür kann sein, dass Bildsynthese am Computer deutlich komplexer ist und signifikant mehr Hintergrundwissen erfordert, als beispielsweise das Arbeiten mit Pinseln oder Markern.

Neben diesem sehr direkten Zusammenhang zwischen 2D- und 3D-Pipelines sind aber auch andere aufgetreten. Im Beispiel der Vektor-Blumen haben Entscheidungen, die der Autor im Schritt des Herausarbeitens von Details, die eher mit dem Surfacing zu vergleichen wären Einflüsse auf den Schritt des 3D-Modelling gehabt.

Die Entscheidung Aquarell für die Kolorierung zu nutzen und die physikalischen Eigenschaften der Farbe hingegen hatten großen Einfluss auf die Gestaltung von Lighting, Surfacing und erforderten einen sehr komplexen Compositing Workflow zum replizieren.

In Abbildung 3.11 ist ein Entscheidungsbaum zu erkennen, mit dem man alle für die Replikation der Vektor- oder Aquarell Ästhetik aus dem hier verwendeten Beispiel abbilden kann. Zudem gibt es noch bestimmte technische Entscheidungen, die beim Aufsetzen einer 3D-Pipeline zu treffen sind. Ebenfalls enthalten sind einige Optionen, die bei für die Erstellung von vergleichbaren Aquarell- und Vektorgrafiken nötig wären. Die dunkelroten Pfeile symbolisieren einen Möglichen Endpunkt des Graphen.

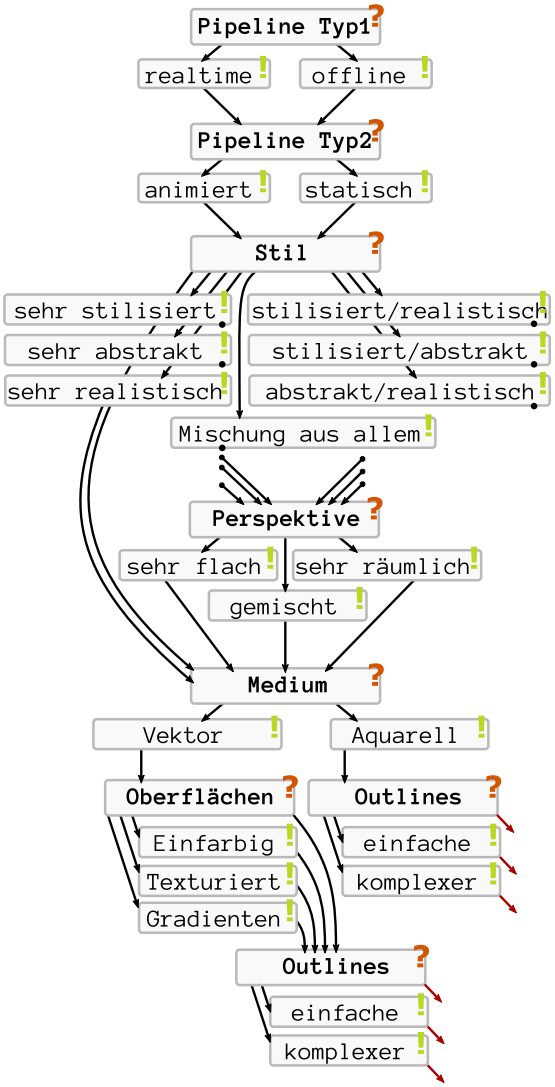

Abbildung 3.11.: Entscheidungsgraph Aquarell- und Vektorgrafik

Dieser Entscheidungsbaum ließe sich
mit Hilfe eines gerichteten Graphen darstellen. So könnte ein Nutzer einer Anwendung durch das Befolgen einer dem Graphen folgenden Umfrage einem System eine Vision vermitteln. Mit bestimmten kreativen Entscheidungen verknüpfte 3D-Methoden könnten so automatisiert zu einem Workflow-Wiki zusammen gefügt werden. In dieser Bachelorarbeit soll nun ein System implementiert werden, welches auf Basis der Entscheidungen des 2D-Workflows dynamisch eine Dokumentation für das aufsetzen einer 3D-Pipeline zusammen stellen kann.

3.3.1. Unerwartete Komplexität

Die Bisherigen Überlegungen haben ergeben, dass es möglich ist, den Entscheidungsprozess eines Künstlers durch eine Baumstruktur abzubilden. Im Implementierungsteil wird auch begründet werden, wie man diesen Entscheidungsgraphen nutzen kann, um modular eine 3D Production Pipeline zu erstellen.

Dabei ist jedoch anzumerken, dass im Allgemeinen jede Abfolge von Entscheidungen durch einen gerichteten Graphen dargestellt werden kann. Zum Beispiel werden Graphen auch in der Planung von Videospielen eingesetzt, um von Spielerentscheidungen abhängige Geschichten darzustellen. Wichtiger ist daher in dieser Bachelorarbeit zu klären, ob dies auch sinnvoll und praktisch für die Mensch-Maschine Schnittstelle einer Anwendung ist

Insbesondere die Komplexität der darzustellenden Entscheidungsgraphen sind für Umsetzung und Nutzbarkeit könnte sich als problematisch heraus stellen:

Im Buch über Concept Art werden in viele verschiedene teils sehr innovative 2D-Pipelines vorgestellt. Bei vielen davon hält der Autor eine Replikation in 3D für plausibel. Allerdings wäre ein Graph, der diese diversen Pipelines abbilden könnte sehr komplex. (siehe angehängte Grafik) So entstehen nicht nur aufgrund der Planung und Erstellung dieses Graphen für den Autor eines solchen Tools große Herausforderungen, für den Nutzer ist ein Abarbeiten des Graphen zudem vermutlich sehr zeitaufwendig. So ist wahrscheinlich, dass die Stilumfrage entweder so komplex werden würde, dass sie kaum benutzbar und erstellbar wäre oder so vereinfacht werden müsste, dass sie ad absurdum geführt würde.

Zu Außerden nüssten Knotenpunkte in ihrer Komplexität teilweise sehr ungleich gewichtet sein. So konnte der Autor nach kurzem Überlegen 17 verschiedene traditionelle Zeichenmittel nennen (Aquarell, Spraydosen, Airbrush, Öl, Bleistifte, Wachsmalfarben, Alkoholmarker, Mosaike, Gouache, Tempera, Tusche, Farbstifte, Pastell, Acryl, Kohle, Tinte und UV-sensitive Farben), die sich in der Replikation durch 3D-Programme voneinander unterscheiden und so gemeinsam mit weiteren traditionellen und digitalen Methoden in einem einzigen Knotenpunkt genannt werden müssten.

Auch lang gezogene Pfade mit nur sehr wenigen Möglichkeiten pro Entscheidung wären laut Überlegungen des Autors nicht zu vermeiden. Beide Szenarien entsprechen nicht den Vorstellungen des Verfassers bezüglich gutem User Experience Design.

Ein weiter wesentlicher Nachteil dieses Systems ist, dass Stile durch die Baumstruktur nur schwer gemischt werden können. In vielen Zeichentrickfilmen und -Serien sind die beweglichen und damit sehr oft neu zu zeichnenden Figuren in deutlich simpleren Stilen gehalten, als die statischen und damit besser von Frame zu Frame wiederverwendbaren Hintergründe und Umgebungen.

Dies sollte demnach bei der Replikation von 2D-Stilen in 3D berücksichtigt werden. Ein anschauliches Beispiel dafür ist die Netflix-Serie Dragon-Prince.

So müsste der Nutzer entweder zwei verschiedene Wikis anfertigen, womit der in Kapitel 4.1.2. beschriebene Dependency Graph den Zweck des Verhinderns von Redundanzen nicht mehr erfüllen könne. Alternativ könnte der Nutzer immer wieder die Option präsentiert bekommen, zu früheren Punkten in der Umfrage zurück zu springen. Dies würde jedoch nicht nur die Komplexität der Graphen erhöhen und gegebenenfalls beim Nutzer für Frust sorgen, dies würde auch mit der derzeitigen Implementation des Graphen wahrscheinlich zu Konflikten führen.

Die Überlegungen des Autos bezüglich eines dynamischen in einem Graph gespeichternen Wikis sind jedoch von diesem Fehler nicht betroffen. Lediglich die Mensch-Maschine-Schnittstelle, mit deren Hilfe die für ein Projekt relevanten Knoten des Graphen zu einem Workflow-Wiki zusammengeführt werden sollten, müsste neu entworfen werden. Daher wird im folgenden Abschnitt ein neues System entworfen.

Leider ist dieser Logikfehler wie in Kapitel 1.2.1. beschrieben erst spät in der Arbeit aufgefallen. So wurde das in diesem Abschnitt als konzeptionell fehlerhaft be-

trachtete System implementiert und im weiteren Verlauf von Kapitel 3 detaillierter beschrieben.

3.3.2. Neue Überlegungen zu Beziehung 2D und 3D

Bei weiteren Überlegungen ist dem Autor aufgefallen, dass die Pilelines 3D und 2D-Grafik sich deutlich mehr ähneln, als bisher in der BA angenommen.

- In der Phase von Inspiration und Konzeption unterscheiden sich 2D und 3D kaum voneinander. Auch Überlegungen bezüglich Umsetzbarkeit im Projektrahmen stellen sich sowohl bei der Planung einer 2D-Pipeline als auch bei der Planung einer 3D-Pipeline. Im Detail sind dabei lediglich verschiedene Aspekte zu betrachten.
- Nachfolgend werden in vielen untersuchten Leitfäden mit verschiedenen Methoden die grundlegenden Formen der darzustellenden Objekte definiert.
- Sind die Grundlegenden Formen fest gelegt, werden feinere Details herausgearbeitet. Im Bereich 3D ist dieser Schritt teilweise von Surfacing jedoch auch teilweise noch durch Modelling abgedeckt.
- Stehen Grundform und Details, werden die Objekte koloriert und schattiert. Hier treten die deutlichsten Unterschiede zwischen 2D und 3D auf. In der 3D-Grafik werden die resultierenden Bilder durch die algorithmische Synthese aus Rohdaten erzeugt. 2D-Artists hingegen fertigen davon deutlich mehr mit manueller Arbeit an.
- Oft werden die daraus resultierenden Arbeiten anschließend mit verschiedenen Methoden nachbearbeitet.

Dabei können sowohl im Bereich 2D als auch im Bereich 3D eine große Menge an Methoden und Werkzeugen zur Verfügung, mit denen das Ziel des jeweiligen Schrittes erreicht werden kann.

Zum Beispiel im Buch Master the Art of Speed Painting werden viele Methoden erklärt, mit denen möglichst schnell konzeptionelle Grafiken erzeugt werden können. So schlagen die Autoren des Buches vor, aus Silhouetten Pinsel für digitale

Grafikapplikationen zu schaffen, um schnell technische Sciense Fiction Konstruktionen zu entwerfen, Teile aus Bildern von technischen Geräten mittels Photobashing zu anderen neuen Formen zu kombinieren, Scans von Acrylfarbe als Texturen für psychiodelische Höhlengemälde zu verwenden oder mit Ölfarbe nachempfundenen digitalen Pinseln impressionistische Illustrationen von Wolken zu kreieren.

So kann ein Zusammenhang zwischen 2D-Pipelines und 3D-Pipelines Auch, beim Beispiel des Aquarell Stils gibt es verschiedene Techniken, mit denen Künstler Arbeiten können: Nass-in-Nass, Trocken-auf-Nass, Nass-auf-Trocken, Lasur-Schicht-Technik sind dabei sehr grundlegende Techniken. Dabei führen verschiedene Techniken zu unterschiedlichen Ästhetiken, die wiederum verschiedene Methoden benötigen, um in 3D repliziert werden zu können.

Im Bereich des 3D-Modelling hat der Autor Erfahrungen mit verschiedenen Methoden gesammelt: Meist greift dieser auf die Technik des Box-Modelliong zurück, für manche Projekte wurden jedoch auch prozedurale Modellierungsmethoden, Photogonometrie und Sculpting eingesetzt, um die 3D-Modelle zu definieren.

Um eine Sofware entwickeln zu können, die bei der Erstellung von 3D-Pipelines und der Replikaiton von 2D Stilen behilflich sein soll, kann jede 2D-Pipeline als Menge an 2D-Techniken betrachtet werden. Soll die 2D-Ästhetik mittels einer 3D-Production-Pipeline repliziert werden, ist jede 2D-Technik mit einer Menge an 3D-Methodiken zu replizieren. Die Summe aller 3D-Methoden chronologisch sortiert ergibt eine 2D-Pipeline.

Die Menge aller 3D-Methoden wird in diesrer Arbeit als Workflow-Wiki bezeichnet.

3.3.3. Neuentwurf Mensch-Maschine-Schnittstelle

Wie könnte dieses komplexe Workflow-Wiki für einen Menschen zugänglich werden? Um diese Frage zu beantworten, wurden die Konzepte von anderen Webseiten, die eine vergleichbare Anforderung umsetzen mussten, analysiert. Webseiten, auf denen Künstler Werke teilen können, stehen ebenfalls vor der Herausforderung, eine große Datenbank an Elementen leicht zugänglich zu machen. Dabei finden sich auf vielen Webseiten zwei gemeinsame Mechanismen, um das Suchen durch Nutzer zu erleichtern:

Kategorien:
Lädt ein Nutzer ein neues Kunstwerk hoch, wird er darum gebeten, dies in eine passende Kategorie einzuordnen. Diese sind in vielen Webseiten auch hierarchisch gegliedert. Auf Abbildung BLA kann man eine Visualisierung von Teilen der Kategorien von Deviantart sehen.

Tags:
Nutzer haben des weiteren die Möglichkeit, ihre Werke mit Tags zu versehen, die der Webseite dabei helfen können n zu Suchanfragen, die diese Schlüsselworte enthalten, passende Werke zu finden. Teilweise sind diese Tags zusätzlich noch hierachisch angeordnet. Übertragen auf das hier gegebene Beispiel würde zum Beispiel der Tag „Ölfarbe“ ebenfalls den Tag „Pinsel“ implizieren.

Darauf aufbauend könnte eine Plattform geschaffen werden, in dem die verschiedenen 3D-Techniken über Kategorisierung und Tagging zugänglich gemacht werden können. Mittels der Kategorien könnten Nutzer unspezifisch nach Inspiration suchen. Wenn ein Nutzer jedoch wissen möchte, wie er Beispielsweise einen Graffiti-Stil replizieren könne, ist es möglich, mit Hilfe des Graffiti-Tags alle mit diesem Tag versehene Techniken angezeigt zu bekommen.

Des weiteren wäre es möglich, die Bilddatenbanken von populären Kreativwebseiten mit den dazugehörigen Tags als Training Data für ein neuronales Netz zu nutzen. Es ist plausibel, dass Nutzer Bild hochladen könnten und vom neuralen Netz verschiedene Tags vorgeschlagen bekommen könnten, die dabei helfen, die für den angestrebten Stil passenden Techniken zu finden.

Hat ein Nutzer die notwendigen Techniken gefunden, könnte er diese in eine Art Warenkorb legen. Wenn der Nutzer alle Tools zusammen gestellt hat, die er für die Realisierung seiner Vision für nötig hält, können die verschiedenen Techniken dynamisch zu einem Wiki zusammen gefügt werden.

4. Implementation

4.1. Konzeption

Das originale Konzept der Anwendung sah vor, dass Nutzer zunächst eine dynamische Umfrage ausfüllen, um dem System eine künstlerische Vision vermitteln zu können. Dafür ist ein System notwendig, dass zum einen Fragen mit unterschiedlichen Antwortmöglichkeiten und zum anderen eine ausführlichere Erklärung zu den Antworten nach dem Anklicken beinhaltet. Beide sollen jeweils Inhalt von Nodes sein, die in der Struktur eines gerichteten Graphen die Stilumfrage abbilden.

Bestätigt ein Nutzer in der Antwortmöglichkeit seine Auswahl, wird durch den Wiki graph temporär eine Node dem Wiki hinzugefügt. Nun kommt das Dependency-System zum tragen: Benötigt eine 3D-Methode weitere Methoden, um durchgeführt werden zu können, sollen diese automatisch mit hinzugefügt werden können. Können bei der gleichzeitigen Nutzung von zwei verschiedenen Methoden technische Probleme auftreten, soll der Nutzer durch einen Konflikt ebenfalls darüber informiert werden können.

Dies erfolgt ebenfalls über einen Graphen. Jede Node speichert einen Wiki-Eintrag, also ein 3D-Methode, und alle Kollisionen und Abhängigkeiten.

Zudem ist ein Backend-Interface notwendig, mit dem Umfrage und Wiki mit dem benötigten Content befüllt werden können. Die Übersichtsgrafik, die zu Beginn der Implementierungsphase dafür angelegt wurde, ist hier angehängt.

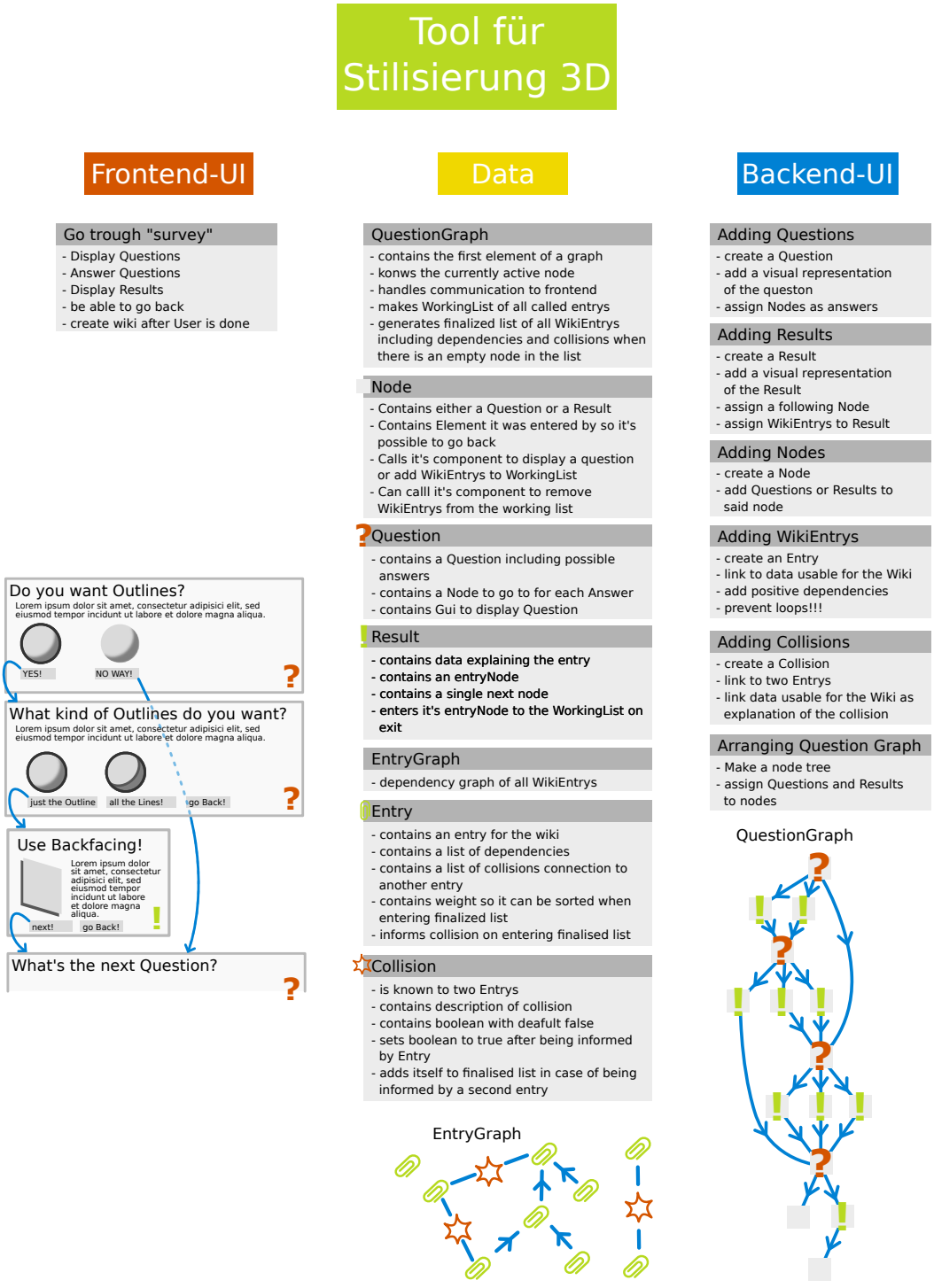

Abbildung 4.1.: Anforderungen Programm

4.2. Umsetzung

4.2.1. Evaluation von Engines, APIs und Programmiersprachen

Zunächst ist die Frage zu klären, mit welcher Sprache und in welchem Framework bzw welcher Engine die Anwendung umgesetzt werden könnte. Dabei wurden folgende Optionen in die engere Betrachtung genommen. (Stichpuntkartige Übersicht über Vor- und Nachteile)

- Unity und C# [Tec18] [Tec19]
  - **Vorteile**
  - weit verbreitete Game Engine, daher viele existierende Frameworks und Tutorials
  - kann teilweise auch im Browser laufen
  - Verfasser ist vertraut mit C#
  - kann Anwendungen für viele Plattformen kompilieren
  - **Nachteile**
  - Verfasser hat wenig Erfahrung mit Entwicklung in Unity
  - Unity läuft nicht immer gut auf Linux (anekddotische Evidenz)
  - Unity hat eine propreitäre Lizenz
  - Unity sammelt Daten über Entwickler
  - Anwender muss Webseite mit Scripten ausführen oder Anwendung auf PC ausführen
- Godot und GDScript (quasi Python) [JL] [JLc]
  - **Vorteile**
  - auf Linux entwickelte Game Engine
  - open Source
  - Verfasser hat Erfahrungen mit GDScript und Godot

  - kann Wendungen für viele gängige Plattformen kompilieren
  - **Nachteile**
  - Godot erfordert ausführen einer Anwendung auf dem Rechner des Benutzers (Open Source: Kontrolle durch den Nutzer möglich)
  - Godot ist weniger bekannt, daher weiter Entwicklung durch Dritte erschwert

- Blender und Python [Fou]
  - **Vorteile**
  - dierekte ansteuerung von Blender durch Wiki möglich (wenn auch nicht im Rahmen der Bachelorarbeit vorgeshehen.)
  - open Source
  - Verfasser ist sehr mit blender vertraut
  - mächtiges Node-UI bereits vorhanden
  - Verfasser mit Python vertraut
  - **Nachteile**
  - unsicher, wie sehr anpassbar Blender ist
  - macht Anwendung unzugänglich für nicht-blender Nutzer
  - macht Weiterentwicklung durch Dritte potentiell extrem kompliziert
  - Ausführung von Python-Scripten auf Seiten des Nutzers notwendig

- Webserver mit Python Backend (siehe Grundlagen)
  - **Vorteile**
  - reine Programmiersprache ohne zusätzliche Strukturen, die verstanden werden müssen
  - open Source
  - zahlreiche existierende Module vorhanden und Nutzbar

- sehr genau auf den Use Case anpassbar
- Möglichkeit HTML und CSS für Frontend zu nutzen
- Möglichkeit der Nutzung von Web/Javascript basierten Frameworks für Node-UI
- keinerlei Installation auf Seiten des Nutzers möglich
- Nutzer muss keinen Programmcode auf eigenem Rechner ausführen
- **Nachteile**
- reine Programmiersprache ohne hilfreiche Zusatzmodule
- Kommunikation zwischen Server und Client sowie Session Management muss beachtet werden

Die Option der Nutzung von *Unity* scheidet dabei bei genauerer Betrachtung kategorisch aus. Der Verfasser hatte aufgrund der im Lizensvertrag vereinbarten umfangreichen Maßnahmen zur Datensammelung durch *Unity* auf Systemen von Endnutzern und Entwicklern zu starke juristische und moralische Bedenken.

Godot als open Source Alternative zu *Unity* ist ebenfalls in Betrachtung gezogen wurden. Die *Game-Engine* verfügt Ebenfalls über *User Interface* Werkzeuge, auf die man zurück greifen kann. Die Möglichkeit, mit *Godot* Anwendungen für *HTML* und *Webassembly* zu kompilieren würde die Anwendung auch gut zugänglich ohne Downloads oder Installationen für Endnutzer gestalten.

Und, wie aus von einer *Godot Web-Application* aus ein Download für den Endnutzer bereit gestellt werden könnte, konnte zu beginn der Entwicklungsphase nicht zufriedenstellend beantwortet werden: Es war nicht klar, wie dieses durch die Engine üblicherweise nicht vorgesehene Feature implementiert werden sollte und es wurden Konflikte mit der *API* befürchtet.

Das Workflow-wiki dierekt in *Blender* zu schreiben hätte einen sehr einfachen Zugriff von der Wiki-Anwendung aus auf diese 3D-Software ermöglicht. Dies sollte jedoch nicht ebenfalls über das *Command Line Interface* und anderer Schnittstellen für *Blender* möglich sein, jede Form von Entwicklung und weiter Entwicklung des Tools würde des weiteren ein starkes Verständnis der *Blender-API* voraus setzen. Dies ist

beim Verfasser nur in geringen Maßen gegeben und kann von anderen, die das Wiki-Tool eventuell weiter entwickeln sollen können nicht erwartet werden. Zudem würde eine Umsetzung in *Blender* alle Anwender ausgeschlossen, die eine andere Software als *Blender* nutzen möchten.

Daher wurde sich dafür entschieden, die Anwendung mit einem klassischen *Webserver* umzusetzen:

Durch *HTML* und *CSS* gibt es einfache Möglichkeiten, die für das *Frontend* notwendigen Texte, Grafiken und Bedienelemente einzubinden. Auch kann durch das Verwenden des sehr gängigen Formates HTML das Tool auch effizient von anderen Entwicklern nach dem *open Source* Prinzip erweitert werden.

Ein Vorteil für den Endnutzer ist dabei ebenfalls, dass eine Webanwendung keinerlei Installation von Software voraus setzt. Da ebenfalls kein *Javascript* genutzt werden muss, ist es noch leichter der Software zu vertrauen.

Aber nun ist zu klären auf welcher Basis kann der *Webserver* entwickelt werden kann. Eines der Bedenken des Autos gegen die Verwendung von *Godot* war, dass es potentiell schwierig ist, die Funktionalität von Downloads zu implementieren, da dies *Game-Engines* typischerweise nicht leisten müssen. So wurde sich für *Flask* entschieden, da dies ein *Framework* ist, bei dem Entwickler aufgrund der konzeptionellen Philosophie erwarten können, relativ vergleichsweise bei der Implementierung von ungewöhnlichen Lösungen gegen die *API* kämpfen zu müssen.

Ein weiterer Grund für Python ist, dass die Sprache den Ruf hat, sich sehr gut für die schnelle Erstellung von Prototypen zu eignen.

Flask kann dabei dabei die wesentlichen Funktionalitäten eines *Webservers* übernehmen.

Dadurch sind im *Backend* große Freiheiten gegeben, da es mit *Flask* möglich ist, eine eigene und auf die Anwendung optimierte Struktur aufzubauen, die beispielsweise nicht den Anforderungen einer *Game-Engine* folgen muss. Auch ist es mit einem auf *Web Development* ausgelegtem *Framework* deutlich leichter, Downloads bereit zu stellen und so das Wiki sich in Dateien zu parsen und dem Nutzer zur Verfügung zu stellen.

So viel die Entscheidung auf die Verwendung von *Flask* und *Python*.

4.2.2. Struktur Wiki-Graph

Die tatsächlich implementierte Struktur unterscheidet sich in manchen Belangen von der Spezifikation, da während der Implementierung neue Möglichektein der Optimisierung des Systems gefunden wurden.

Als Grundlegende Struktur für den Wiki Graph werden dabei zunächst vier Klassen benötigt:

- Der Wiki Graph selbst, als Verwaltungsklasse und Interface für den Rest der Anwendung.
- Die Wiki Node, in denen die jeweiligen Einträge und alle benötigten Informationen gespeichert sind
- Die Collision Edge, in denen eine Erläuterung des Problems und alle weiteren notwendigen Informateionen gespeichert sind.
- eine Dependency-Edge, so dies für die implementierung eines Dependency Syystems notwendig ist.

WikiGraph
Der Wiki-graph Speichert alle Elemente des Wikis in einer Liste und besitzt ebenfalls eine Liste an aktuellen Warnungen. Wenn während des zusammenstellens einer Pipeline Kollisionen auftreten, werden diese an Warnings angehängt.

Der Graph kann des weiteren Nodes hinzufügen. Dazu wird der Klasse ein Eintrag und das Gewicht des Eintrags übergeben. Die Klasse gibt dabei eine ID zurück, damit die Node von extern leicht angesprochen werden kann. Durch die dynamische Natur von Python wird der Dateityp der Einträge nicht in der Klasse definiert, es ist jedoch geplant, mit Dateipfaden zu arbeiten.

Zur Komplexität des Wiki Systems hat zudem beigetragen, dass während Einträge temporär hinzugefügt und durch zurück-gehen des Nutzers wieder aus dem Wiki entfernt werden können.

Hat der Nutzer die Umfrage abgeschlossen, werden alle temporär hinzugefügten Nodes nach Gewicht zu einer finalisierten Liste sortiert.

Node
In den Nodes selbst ist wie oben Bereits erwähnt ein Wiki-Eintrag und eine Liste

an Collision Edges und Dependency Edges gespeichert. Wird eine Node temporär hinzugefügt, informiert sie dabei alle Edges und inkrementiert eine Zählvariable. Jedes mal, wenn die Node erneut aufgerufen wird, inkrementiert diese Zählvariable weiter. Erreicht die Node ein negativer Aufruf, dekrementiert die Zählvariable. Ist die Variable bei Null angekommen, werden CollisionEdges und DependyEdges darüber informiert, dass die Node final entfernt wurde.

Collision Edge
CollisionEdges fügen, sobald alle Bedingungen für die Kollision erfüllt sind, also alle Nodes sich als hinzugefügt gemeldet haben, eine Warnung einer Liste an Warnungen hinzu, die dem Nutzer angezeigt werden können.

DependencyEdge
Sind alle Bedingungen einer Dependency-Edge erfüllt, werden alle in der Edge als benötigt aufgelisteten Nodes aufgerufen, die widerum den eigenen Dependencys und Collisions informieren.

Ist eine Bedingung nicht mehr erfüllt (oder informiert eine Node die Edge über einen negativen Aufruf), bekommen alle benötigten Nodes durch die Edge einen negativen Aufruf.

4.2.3. Implementation SurveyGraph

Auch die Stilumfrage ist durch einen Graphen dargestellt. In diesem Fall haben jedoch sind die Edges selbst keine Klassen, sondern nur Referenzen. In jeder Node ist der Name eines HTML-Dokumentes gespeichert. Bei einer Frage sind zusätzlich noch mehrere Antwort-Nodes hinterlegt. Bei den Antworten je eine nachfolgende Frage und ein- oder mehrere Verweise auf den Wiki Graph. Zusätzlich wird das Element hinterlegt, von dem aus die Node zuletzt aufgerufen wurde. So hat der Nutzer die Möglichkeit, in der Umfrage einen oder auch mehere Schritte zurück zu gehen, wenn er sich anders entschieden hat.

Ruft ein Nutzer die Webseite zum ersten mal auf, wird eine neue Umfrage und eine ID für den Nutzer generiert. Diese Umfrage kann der Nutzer starten, in dem er auf einen Link zum Starten klickt. In dem Link ist eine ID enthalten, mit der dem Nutzer eine Session zugeordnet werden kann. So sind keine Cookies notwendig, um den Nutzer identifizieren zu können.

Ruft der Nutzer nun diesen Link auf, wird das Dokument einer Frage durch Flask gerendert. Mit der Session-ID wird dabei zunächst der korrekte SurveyGraph identifiziert. Anschließend wird das Dokument aus einem Header, der Frage selbst, allen Antwortmöglichkeiten und einem Footer gerendert. Die verschiedenen html-Dokumente werden dabei durch Flask als String gerendert und addiert.

Für die den Dokumentteil der Frage wird der Name aus der aktiven Node des SurveyGraph abgefragt, für die Antwortmöglichkeiten werden die Namen aller in der aktiven Node enthaltetnen Optionen abgefragt und mit dem Preview-Tag versehen. Um Nutzer und die gewählte Antwortmöglichkeit zu erkennen, werden dabei ID und die Nummer der Antwortmöglichkeit in das Preview-Dokument über Jinjas 2 Templating hinzugfügt.

Entscheidet sich der Nutzer für eine Antwortmöglichkeit, wird ihm die Langform des dazugehörigen Dokumentes angezeigt. Dabei wird das aktuelle Objet des SurveyGraph aktualisiert. Der Nutzer kann nun auf einem Link die Auswahl bestätigen oder zur Frage zurück gehen. Bestätigt er die Auswahl, wird die nächste Frage aufgerufen und dargestellt, während im WikiGraph die in der Antwort hinterlegte Node aufgerufen wird.

Der in diesem Kapitel Beschriebene Code ist in der digitalen Version dieser Arbeit als Datei und in der gedruckten Fassung als CD angehängt. Die Beschriebene Code-Struktur ist größtenteils implementiert und ist in der Lage, eine Webseite mit Fragen und Antwortmöglichkeiten gemäß der Struktur darzustellen. Manche der beschriebenen Funktionen (insbesondere das Interface zwischen SurveyGraph und WikiGraph) wurden vor dem Bemerken des In Kapitel 3.3.1 beschriebenen Fehlers noch nicht implementiert. Da sich der Autor davon Entschied, mehr Zeit auf Fehlerbetrachtung und Reevaluation zu investieren, wurde zu diesem Zeitpunkt die Implementierung abgebrochen und wird nach Abschluss der Arbeit fortgeführt.

5. Evaulation

5.1. Methodik

Die Anwendung sollte durch einen Test mit mehreren 3D-Artists und einer Umfrage von zufälligen Menschen durchgeführt werden.

Zunächst bekommen die Künstler die Aufgabenstellung, eine 3D-Grafik entweder in einem Vektor- oder in einem Aquarelle-Stil durchzuführen. Als Referenz sollten dabei die im Rahmen dieser Arbeit angefertigten Grafiken dienen. Dabei sollten die Probanden sich in Ihrer Arbeit durch das hier entworfene Tool unterstützen lassen. Nach dem Anfertigen der Grafik sollten sie einen kleinen Fragebogen ausfüllen. In dem Fragebogen soll ermitelt werden:

- Wie viel Jahre Erfahrung hat der Proband in 3D?
- Sieht sich der Proband eher als Hobbyist oder eher als Professioneller an? (auf einem Spektrum von 1-7)
- Welche Software hat der Proband für das Programm genutzt?
- Wie viel Erfahrung hat der Proband vorher bereits mit der Replikation von 2D-Stilen in 3D gehabt? (auf einem Spektrum von 1-7)
- Wie sehr denkt der Proband hat ihm die Software bei der Erstellung geholfen? (auf einem Spektrum von 1-7)
- Wie angenehm war die Software zu bedienen? (auf einem Spektrum von 1-7)

Als kleine Problematik bei dieser Umfrage ist zu erwähnen, dass die Befragung sehr groß ausgelegt werden müsste, um mit einer zweiten Kontrollgruppe, die Grafiken ohne Zuhilfenahme des Programms Grafiken anfertigen müsste, die Ergebnisse aus der Gruppe, die das Programm nutzt, zu vergleichen.

Im zweiten Teil der Umfrage sollte eine Gruppe aus Menschen mehrere 3D- und 2D-Grafiken in verschiedenen gezeigt werden, darunter auch die im ersten Teil angefertigten Grafiken. Bei jeder Grafik würden die Befragten sagen müssen, wie sehr ihnen die Grafik zusagt und, ob sie die Bilder (auf einem Spektrum von 1-7) eher für 2D-Grafik oder eher für 3D-Grafik halten.

Die für die Evalutation Idealen Testergebnisse würden ergeben, dass ...

- 3D-Artist unabhängig von verwendeter Software und vorhergehender Erfahrung die Software als sehr angenehm zu bedienen und sehr hilfreich wahr nehmen.
- die Gruppe, die sich die Bilder ansehen sollte alle entstandenen 3D-Grafiken als 2D-Grafik wahr nimmt und die entstandenen 3D-Grafiken zudem noch ästhetisch ansprechender findet, als die Referenz-3D-Grafiken.

Realistisch betrachtet ist zu erwarten, dass erfahrenere 3D-Artists, die vorher noch wenige Erfahrungen im Bereich der Stilisierung gesammelt haben am stärksten von der Software profitieren. Die Software benötigt fortgeschrittene Kentnisse der 3D-Grafik und wird weit Fortgeschrittenen 3D-Artists im Stand der Evaluation auch nicht sehr viele Tricks vermitteln können.

Außerdem ist zu erwarten, dass aufgrund der Präferenz des Autos für die Software blender Nutzer des gleichen Tools stärker von den aus dem Tool hervorgehenden Workflows profitieren würden.

Wie gut Menschen ohne große Fachkenntnis die 3D-Natur der entstandenen Grafiken erkennen werden können, wird zudem sehr von der Arbeit des Künstlers abhängig sein. Der Autor hält es für unwahrscheinlich, dass die Frage nach den ästhetischen Präferenzen statistisch verwertbare Ergebnisse hervorbringen würde.

5.2. Ergebnisse

Leider können zu dieser Evaluation keine Ergebnisse präsentiert werden. Während der finalisierenden Programmierschritte für das Tool ist der in Kapitel 3.3.1 beschriebene Konzeptionelle Fehler auf. Dah die Umstrukturierungen im Rahmen dieser Arbeit nicht mehr durchgeführt werden konnten, wurde die Anwendung nicht finalisiert und somit auch nicht evaluiert.

6. Zusammenfassung und Ausblick

Die Untersuchungen der Arbeit haben ergeben, dass es möglich ist, die Arbeitsabläufe von 2D-Artists zu systematisieren und als 2D-Pipeline darzustellen. Dabei ist aufgefallen, dass eine 2D-Pipeline Parallelen mit 3D-Production Pipelines aufweist. Die wesentlichen Unterschiede liegen dabei in den verwendeten Methoden und Techniken, die genutzt werden, um eine Ästhetik zu erzeugen. Die meisten in einer Pipeline verwendeten Methoden haben dabei Einfluss auf die visuelle Wirkung der resultierenden Grafik. Soll die Ästhetik einer 2D-Grafik mit einer 3D-Production-Pipeline repliziert werden, können die für die Erstellung genutzten Methoden durch Untersuchung der Grafik heraus gearbeitet werden.

Gelingt es Techniken zu finden, die die Wirkung dieser Methoden replizieren können, ist es möglich, die 2D-Ästhetik mit Hilfe von 3D-Anwendungen zu reproduzieren.

Wesentlich für die Wirkung einer Grafik sind auch die kreativen Entscheidungen, die ein Artist während einer Pipeline trifft. Die Abfolge an Entscheidungen kann als gerichtetr Graph dargestellt werden. Da aufgrund der Vielzahl an verschiedenen Stilen ein umfassender Entscheidungsgraph sehr komplex gestaltet sein müsste, ist dies als Grundlage für das Bedienkonzept einer Anwendung, die bei der Replikation von 2D-Stilen in 3D assistieren soll, nicht sinnvoll.

Da die angesprochenen Methoden und Werkzeuge letztendlich als Resultat der Entscheidungen betrachtet werden können, würde eine Anwendung, die eine Ansammlung an verschiedenen 2D-Tools und dazugehöriger Leitfäden zur Replikation dieser in 3D beinhaltet und zugänglich macht der Aufgabenstellung der Arbeit vermutlich genügen.

Das Konzept einer Sammlung von Tools wurde in dieser Arbeit leider erst spät entwickelt. Vorher war bereits viel Zeit ist in die Untersuchung und Implementierung von Entscheidungsbäumen investiert wurden, die sich als kein realistisch benutzbarer Ansatz heraus gestellt haben.

Doch was sind die Konsequenzen für die wissenschaftliche Arbeit?

Da der Fehler erst nach der Implementation aufgefallen ist, wäre ein neuer Entwurf für das User Interface und eine Reimplementierung dessen notwendig. Dies erfordert jedoch zusätzliche Zeit. Da die Prüfungsordnung der Hochschule Mittweida jedoch die Verlängerung einer Bachelorarbeit aufgrund von konzeptionellen Fehlern nicht vorsieht, wird dies nicht diesem Rahmen möglich sein.

Jedoch ist weder der WikiGraph an sich noch die Codestruktur des SurveyGraph hinfällig. Daher wurden in dieser Arbeit die Konzeption und Implementation der Strukturen beschrieben. Zudem ist das Forschungsprojekt mit dem Abschluss dieser Arbeit nicht beendet. Es ist geplant, nach Abschluss der Bachelorarbeit am Projekt weiterhin zu forschen. In den Monaten nach der Veröffentlichung dieser Arbeit wird der Autor die Implementation erneut mit dem beschriebenen verbesserten Konzept angehen und die Webseite projects.betalars.de online stellen, um dort den aktuellen Status dieses Projektes teilen zu können.

6.1. Fehlerbetrachtung

Das Ergebnis der Bachelorarbeit entspricht nur teilweise den Vorstellungen des Autors. Was kann aus den während dieser Arbeit geschehenen Fehlern gelernt werden?

Die Evaluation hat sich im Endeffekt als problematisch für die Implementierung heraus gestellt. Der Grund, weswegen die Langfassung von Kapitel 3 erst nach der Implementierung angefertigt wurde ist, dass die Evaluation selbst Zeit benötigt und nur an einer fertigen Anwendung durchgeführt werden kann. Zwei Wochen auf Resultate einer Evaluation warten und dabei nur an wenigen Kapiteln schreiben können ist im Zeitrahmen einer Bachelorarbeit nur schwer möglich.

Jedoch hätten dem Autor zwei Methoden helfen können, dieses Problem frühzeitiger zu erkennen: Zum einen sind das Referenzen. Welche Anwendungen stehen vor ähnlichen Anforderungen und, wie sind diese damit umgegangen? Des weiteren wäre durch frühzeitiges Erstellen von ausführlicheren Prototypen auf Papier möglicherweise früher aufgefallen können, dass die Komplexität der erdachten Graphen sich als problematisch erwiesen könnte.

Zudem hat Autor vielen Menschen empfohlen, bei der Konzeption immer wieder bildlich beschrieben einen Schritt zurück zu treten und zu überlegen, ob die Lösung für ein Problem, an der gerade gearbeitet wird, tatsächlich das Problem optimal löst oder, ob es nicht andere, bessere Wege gäbe. So hat der Autor bereits mehrere potentielle Aufträge für 3D-Grafiken abgelehnt, weil andere Methoden für den Auftraggeber größere Vorteile gehabt hätten. Die Anwendung dieser Philosophioe während er Konzeption der Arbeit hätte potentiell bei einer frühzeitigeren Diagnose des Komplexitätsproblems helfen können.

6.2. Ausblick

Der Autor hat in dieser Arbeit dennoch viel gelernt: Nicht nur ist das Realisieren und Planen eines dreimonatigen, größtenteils als Einzelperson zu realisierenden Projektes eine lehrreiche Erfahrung gewesen, auch einige während der Recherche aufgetretenen Erkenntnisse werden dem Autor in Zukunft vermutlich nützlich sein. Der Autor plant in Zukunft deutlich mehr Zeit in einer 3D-Production Pipeline auf Konzeption und Gestaltung investieren, um für ein ausgewogeneres Verhältnis von technisch und kreativ anspruchsvollen Aspekten in den eigenen Arbeiten zu Sorgen.

Des Weiteren möchte der Autor in Zukunft auch mehr stilistische Mittel der 2D-Grafik in den eigenen kreativen Arbeiten einbinden, um interessantere Ästhetiken erzielen zu können.

Abschließend ist noch anzumerken, dass der Autor in letzter Zeit mit Filmen, wie *Into the Spiderverse* und insbesondere auch vielen Spielen kleinerer Studios den Eindruck vermittelt bekommt, dass interessanter stilisiertes 3D immer mehr Verbreitung und Anerkennung erlangt und ist zuversichtlich mit seinem Tool – sobald es fertig entwickelt ist – einen positiven Beitrag dafür leisten zu können.

Literaturverzeichnis

[All] Andy Allan: *Spreewald-Kartentile, OpenStreetmap*, URL: https://osm.org/go/0MP6Od_hX-?layers=T, besucht am 15.02.2019.

[AN11] Peter Haberaecker Gudrun Socher Alfred Nischwitz, Max Fischer: *Computergrafik und Bildverarbeitung*, Bd. 1, Vieweg+Teubner Verlag, 3 Aufl., 2011, ISBN 978-3-8348-1304-6.

[Bro18a] Josiah Brooks: *Draw with Jazza: Creating Characters*, Impact, Ohio, 1 Aufl., 2018, ISBN 1-403-4494-9.

[Bro18b] Chris Browne: *The Dragon Prince Behind-The-Scenes | Show Environments*, 12 2018, uRL: https://youtu.be/NNqcqjA9_HY, besucht am 03.03.2087.

[CM04] Harald Sack Cristoph Mainel: *WWW: Kommunikation, Internetworking, Web-Technologien*, Springer Verlag, Berlin Heidelberg, 1st edition Aufl., 2004, ISBN 978-3-642-18963-0.

[Flu08] Barbara Flueckiger: *Visual Effects, Filmbilder aus dem Computer*, Schueren, 2008, ISBN 978-3-8947-2518-1.

[Fou] Blender Foundation: *Blender is Free Software*, uRL: https://www.blender.org/about/license/, besucht am 12.02.2019.

[Gam13] Rockstar Games: *Drand Theft Auto V*, Videospiel, 9 2013, bilder aus kommentiertem Spieldurchlauf entnommen. URL: https://youtu.be/heJ_HEz2U44.

[JE12] Peter Kaiser Johannes Ernesti: *Python 3: Das umfassende Handbuch*, Galileo Computing, Bon, 3., aktualisierte und erweiterte auflage Aufl., 1 2012, ISBN 978-3-8362-1925-9.

[JL] Godot Juan Linietsky, Ariel Manzur: *Godot Licence*, uRL: https://godotengine.org/license, besucht am 12.02.2019.

[JLc] Ariel Manzur Juan Linietsky und Godot contributors: *Godot Features*, uRL: https://godotengine.org/features, besucht am 12.02.2019.

[Lev18] Marisa Levis: *Creating Stilized Characters*, Bd. 1, 3dtotalPublishing, Worchester, 2018, ISBN 978-1-909414-74-7.

[Mac18] Allyson MacDonald: *Flask Web Development*, O'Reilly Media Inc., 1005 Gravenstein Highway, North Sebastopol, CA 95472, second edition Aufl., 2018, ISBN 978-1-491-99173-2.

[MS18] Mingu Choi David Honz sleepypang LI CARLO BALASSU toni infante Quentin Pointillart Francesco Sala PETER KONIG Eugene Korolev hw 6523 MONDEAD STUDIO, Alejandro Burdisio: *Beispiele der 2D-Grafik auf Artstation.com*, Nutzerbeiträge auf Webseite, 3 2018, namen in Ordnung der Bilder in Grafik.

[Pel] Fabio Pellacini: *the 3D production pipeline*, Presentation, uRL http://pellacini.di.uniroma1.it/teaching/projects10/lectures/01_pipeline.pdfZugriff 09.02.2019.

[Ron08] Frank Ronneburg: *Debian GBU/Linux 4: Anwenderbuch für Einsteiger, Umsteiger und Fortgeschrrittene*, Addison-Wesely, München, 2008, ISBN 978-3-8273-2523-5.

[Tec18] Unity Technologies: *Building and running a WebGL project*, 2018, uRL: https://docs.unity3d.com/Manual/webgl-building.html.

[Tec19] Unity Technologies: *Unity Terms of Service*, Jan. 2019, uRL: https://unity3d.com/de/legal/terms-of-service.

[TJI] Kerren Piche Serna Timophy J. Ingersoll: *Script to Sreen: The DreamWorks Animation Pipeline*, Video, originale Quelle unbekannt, verfügbar unter URL: youtu.be/ru0tQRJ4qKs.

Anhang

A. Analysedokumente

A.1. Friendship

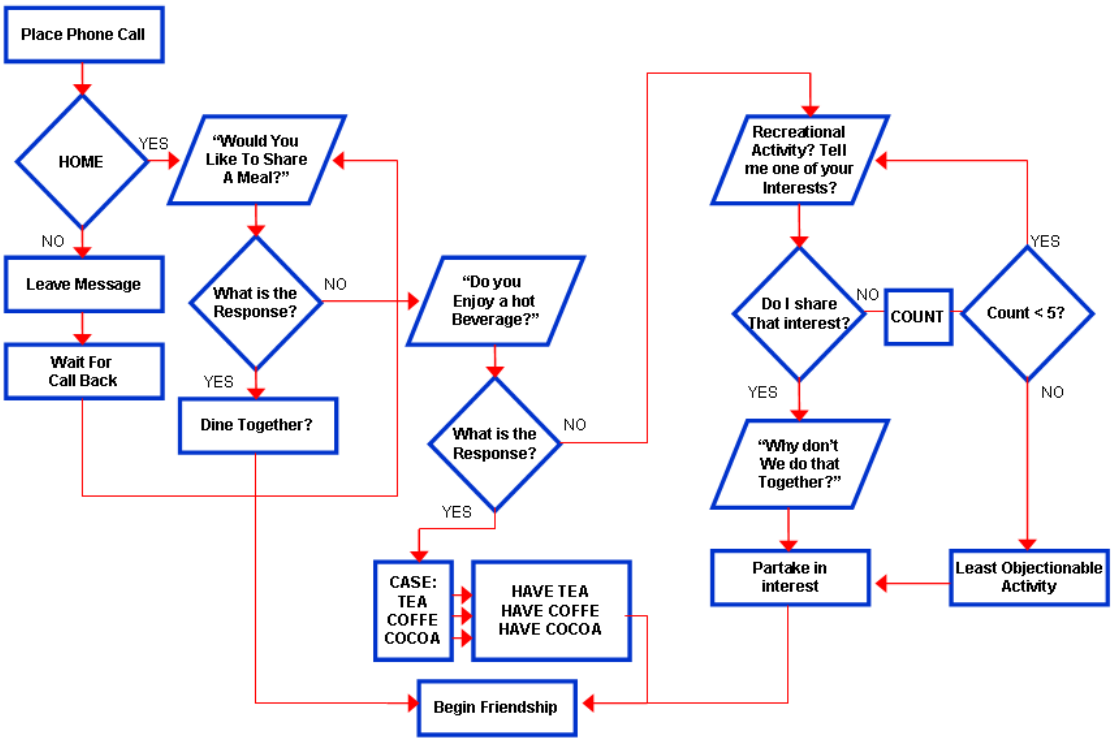

Abbildung A.1.1.: The Friendship Algorithm by Dr. Sheldon Cooper, Ph.D.

A2

B. Konzeptionsdokumente

B.1. Backpacking

Places to Go Backpacking		
Name	Driving Time (hours)	Notes
Big Basin	1.5	Very nice overnight to Berry Creek Falls from either Headquarters or ocean side.
Sunol	1	Technicolor green in the spring. Watch out for the cows.
Henry Coe	1.5	Large wilderness nearby suitable for multi-day treks.

Tabelle B.1.1.: Places to go Backpacking

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Mittweida, den 21. Mai 2019

Lars Bönsch