Comic Generierung - PJ 1

Fenja Harbke

Wintersemester 2014/15

Das charakteristische an einem Comic sind Charaktere im Comic-Stil, welcher die Darstellung von Charakteren und Objekten mit Hilfe von Outlines, also schwarzen Linien entlang von Kanten, beschreibt.

Ziel der Masterarbeit ist das Generieren von Comics auf Grundlage eines User-Inputs in Form einer zu erstellenden DSL.

Die jedem Panel zu Grunde liegende 3D-Szene wird mit zuvor modellierten Charakteren erstellt. Die initiale Modellierung der 3D-Modelle erfolgt mit entsprechenden Tools wie Blender. Um Erweiterbarkeit und Unabhängigkeit vom Modellierungstool zu erhalten, werden die Modelle mit einem entsprechenden, 3D-kompatiblem Framework (LibGDX) geladen und die Szene dort gerendert. Zusätzliche GLSL-Shader erkennen mit Hilfe einer Tiefen- und Normalenmap die äußeren und inneren Kanten der 3D-Objekte und zeichnen sie schwarz nach. Die Ergebnisse der beiden Shader werden addiert und als Ergebnis entsteht ein Bild mit den Comic-typischen Outlines, das in ein beliebiges Bildformat als fertiges Panel exportiert werden kann.

Zugrunde liegende 3D-Modelle und ein Comic-Shader sind Bestandteil dieser Arbeit und ein Schritt auf dem Weg zum Comic-Generator.

1 Einführung

Ziel der Masterarbeit ist die Erstellung einer Comic-DSL, anhand deren Beschreibungen ein Comic generiert wird. Dies geschieht mit Hilfe von vorhandenen 3D-Modellen der verwendeten Charaktere und eines Toon-Shaders, um den Comic-Stil zu erzeugen.

Projekt 1 beschäftigt sich mit der Erstellung von 3D-Modellen, da diese die Grundlage für weitere Bearbeitung darstellen. Zudem wird ein Comic-Shader auf Basis der Arbeit von Decaudin [4] implementiert, um eine 3D-Szene in den typischen Comic-Stil mit schwarzen Outlines der Objekte zu überführen. Das Ergebnis wird als Bilddatei abgespeichert.

Parallel zu PJ1 werden in AW2 (1.1) zugehörige Algorithmen und Vorgehensweisen recherchiert.

Diese Ausarbeitung befasst sich mit dem konkreten Vorgehen in PJ1 und den dort gewonnenen Erkenntnissen und Ergebnissen für die Masterarbeit.

- Einarbeitung in Blender, Erstellung von 3D Charakteren
- 3D Charaktere mit Skelett in Blender. Bewegbarkeit an den Gelenken
- Comic-Shader erarbeiten
- Shader anwenden und Comicpanel erhalten

Diese Ausarbeitung geht zunächst auf die enge Verknüpfung zum parallel verlaufenden AW2 (1.1) ein und beschäftigt sich im Hauptteil mit den Bearbeitungsschritten, um vom *3D-Modell zum Comic* (2.1) zu gelangen.

Erster Schritt ist die Erstellung eines 3D-Modells in *Blender* (2.2) wobei darauf eingegangen wird, welche Anforderungen an ein entstehendes *3D-Modell*(2.3) gestellt werden. Das Modell wird in *LibGDX* (2.4) geladen und ein *Shader* (2.5) für den Comic-Stil angewendet. Mit einem Bild-Export wird das fertige *Comic Panel* (2.6) erhalten.

Zum Schluss wird noch einmal auf die *Ergebnisse* (3.1) von PJ1 eingegangen, auf die *weitere Vorgehensweise* (3.2) und zuletzt auf die damit verbundenen *Risiken* (3.3).

1.1 AW 2

Parallel zu PJ1 werden in AW2 [7] vergleichbare Arbeiten untersucht [11, 4, 15] und verwendbare Algorithmen und Vorgehensweisen extrahiert. Der Toon-Shader [4] findet direkt in PJ1 Anwendung, ein Algorithmus zur Sprechblasenplatzierung [11], sowie die Hierarchie einer fertigen Comic-Seite [15] dienen als Grundlage für die nächsten Schritte.

2 Hauptteil

Der erste Schritt ist es, die Kernelemente des Comic-Generators, also die 3D-Modelle zu erstellen. Hierfür gibt es entsprechende Tools und gewisse Regeln müssen beachtet werden, um ein sinnvolles 3D-Modell zu erhalten, dass von einem Comic-Shader sinnvoll verarbeitet werden kann. Dieser Shader erkennt die Kanten des 3D-Modells, abhängig von der Kameraperspektive, und zeichnet sie schwarz nach. Das als Ergebnis erhaltene Bild kann weitestgehend als ein fertiges Comic-Panel angesehen werden.

Die Auswahl und der Umgang mit benötigten Programmen wurden bereits in AW1 [6] als Risiko eingestuft. Dies betrifft die Erstellung von 3D-Modellen, die Weiterverarbeitung durch einen Comic-Shader und die Zusammensetzung mehrerer fertiger Panel zu einer Comic-Seite.
Die ausgewählten Programme mussten bereits den Anforderungen an das Endergebnis der Masterarbeit entsprechen.
Wichtig ist die Sammlung von 3D-Modellen, um auf dieselben zugreifen zu können. Die durch eine DSL gestellten Anforderungen müssen vom System in komplexere Befehle übersetzt und durchgeführt werden können.

Zudem werden hohe Anforderungen an Erweiterbarkeit gestellt, um möglichst viele Anpassungsmöglichkeiten zu haben. So besonders im Bezug auf die Auswahl eines Bildausschnittes, die Positionierung von 3D-Modellen im Raum und das Anpassen von Gestik und Mimik.

2.1 3D-Modell zu Comic

Im Folgenden noch einmal eine kurze Übersicht über die Arbeitsschritte vom 3D-Modell zum Comic-Panel, welche den in PJ1 durchgeführten Schritten entsprechen.

3D Modell Das 3D-Modell muss einen erkennbaren Charakter aus mehreren Perspektiven darstellen können. Wichtig ist hier eine korrekte, aber nicht zu realistische Modellierung, um den einfacheren Comic-Stil nicht zu verfehlen.

Skelett Die 3D-Modelle erhalten ein Skelett, welches dem einer Zeichenpuppen entspricht, sodass sämtliche Bewegungen, die dargestellt werden sollen, durch ein Anpassen der Gelenke nachvollzogen werden können.

Pose Die mit dem Skelett nachvollzogene Pose sollte möglichst abstrakt auf sämtliche 3D-Modelle mit gleichartigem Skelett übertragbar sein. Das Einnehmen einer Pose sollte den dargestellten Charakter nicht in eine unnatürliche Haltung versetzen.

Bildausschnitt Im Bildausschnitt, welcher durch die Kameraansicht auf die 3D-Szene dargestellt wird, müssen alle für das jeweilige Panel wichtige Elemente sichtbar sein.

Shader Der Cartoon-Shader wird auf die 3D-Szene angewendet und soll innere und äußere Kanten mit einer schwarzen Linie nachziehen, sodass der typische Comic-Stil erhalten wird.

Panel Das fertige Comic-Panel entsteht aus einem Export des fertig gerenderten Screenshots. Ein Panel ist schwarz umrandet und hat eine vorher angegebene Größe.

Der erste Schritt beschäftigt sich mit dem eigentlichen Erstellen der 3D-Modelle. Da dies ein für mich noch unbekanntes Gebiet darstellt, nimmt der Aufwand hierfür knapp die Hälfte von PJ1 ein.

2.2 Blender

Als Tool für die Modellierung von 3D-Modellen wurde Blender ausgewählt. In Blender können mit verschiedenen Techniken Meshes aus Edges, Vertices und Flächen erstellt werden. Das Modell kann mit einer "Armature", also einem Skelett versehen und das Ergebnis in ein allgemein gültiges Format exportiert werden. Das erhaltene 3D-Modell lässt sich mit einer Zeichenpuppe vergleichen, deren Gelenke anpassbar sind und auf welche mit der Kamera eine Ansicht gewählt werden kann. Durch den räumlichen Aspekt kommt hier deutlich der Vorteil gegenüber 2D-Ansätzen zur Geltung.

Blender ist eine freie und open-source 3D-Grafiksoftware. Mit ihr können dreidimensionale Körper modelliert, texturiert und animiert werden. Blender ist cross-plattform und nutzt im Hintergrund OpenGL. Das Projekt läuft community-driven unter der GNU General Public License (GPL), wodurch einfach kleine und große Code-Änderungen für neue Funktionen, Bug-Fixes und bessere Benutzbarkeit umgesetzt werden können. [1]

2.2.1 Allgemeines

Die Entscheidung für Blender fiel aus mehreren Gründen. Hauptgrund waren die open-source Verfügbarkeit und die Möglichkeit zur freien Verwendung. Es existieren Alternativen zu Blender, wie z.B. Autodesk *Maya*. Maya ist ebenfalls eine Software zur 3D-Visualisierung und -Animation und ist das bekannteste und meistgenutzte Tool im Bereich 3D-Modellierung, Computeranimation und Rendering. Vor diesem Hintergrund soll das entstehende System die Erweiterbarkeit bieten, auch mit 3D-Modellen, die mit anderen Programmen erstellt wurden zu arbeiten.

Blender wurde zudem auf Grundlage der Empfehlung des Kommilitonen Raimund Wege gewählt. Es ist weit verbreitet, was sich in einer großen Community und somit besserem Support und Rechercheergebnissen bei Problemen zeigt.
Zudem standen das Blender-Buch [17] und gute Tutorials [2] zur Verfügung. Dies vereinfachte den Einstieg in die 3D-Modellierung mit der Software Blender.

2.2.2 Vorgehensweise Modellierung

Die Modellierung beschreibt das Erstellen eines 3D-Objekts mit einem entsprechenden Tool (Blender). Durch Edges verbundene Vertices wird hierbei ein Körper beschrieben, der durch Anpassungen und Erweiterungen in die gewünschte Form gebracht wird. Ein fertiges Modell stellt ein Objekt der realen Welt dreidimensional dar, z.B. einen Charakter, der später in einem Comic auftreten soll.

Zur Modellierung wurde ein Ansatz aus dem Blender-Buch [17] gewählt, um ein 3D-Modell zu erhalten, dass möglichst dicht an eine Comic-Figur heran kommt. Hierfür werden zuvor entworfene Ansichten auf einen Charakter, in Front- und Profilansicht, als Bildvorlagen geladen und das Mesh anhand dieser Rahmenbedingungen modelliert.
Dieses Vorgehen bietet gleichzeitig einen guten *Sketch to 3D* Ansatz, wie er bei [8] beschrieben und schon in AW1 erwähnt wurde [6]. Dies auszuarbeiten ist nicht Teil der Masterarbeit, bietet aber einen Ansatz für weitere Forschung.

Im Folgenden wird etwas genauer auf die Umsetzung dieser Vorgehensweise, sowie die Grundlagen und den Umgang mit dem Programm Blender eingegangen.

Bildvorlagen Die als Vorlage verwendeten Bilder können als Hintergrundbilder geladen, und auf eine bestimmte Ansicht beschränkt werden. Eine Aufteilung des Bildschirms mit einer frontalen und einer Seitenansicht ermöglicht so das Modellieren anhand beider Vorlagen.
Wichtig bei diesem Vorgehen ist, dass die verwendeten Bilder von ihren Abmessungen und Proportionen her überein stimmen, sodass Front- und Profilansicht nicht widersprüchlich sind, was zu einem invaliden Mesh führen würde.
Zudem müssen die geladenen Bilder ggf. noch skaliert werden, um auch in Blender gleich große Proportionen zu beschreiben.

Begonnen wird die Modellierung mit einem Kreis-Mesh, welches mit Hilfe des *extrude* Befehls erweitert wird und durch Stauchungen, Streckungen und Verschiebungen ein Gitter um den Charakter baut. Symmetrische Teile des Modells, wie Arme und Beine, können einmal erstellt und dann gespiegelt werden. Dieses Vorgehen ist in Abbildung 1 zu betrachten.

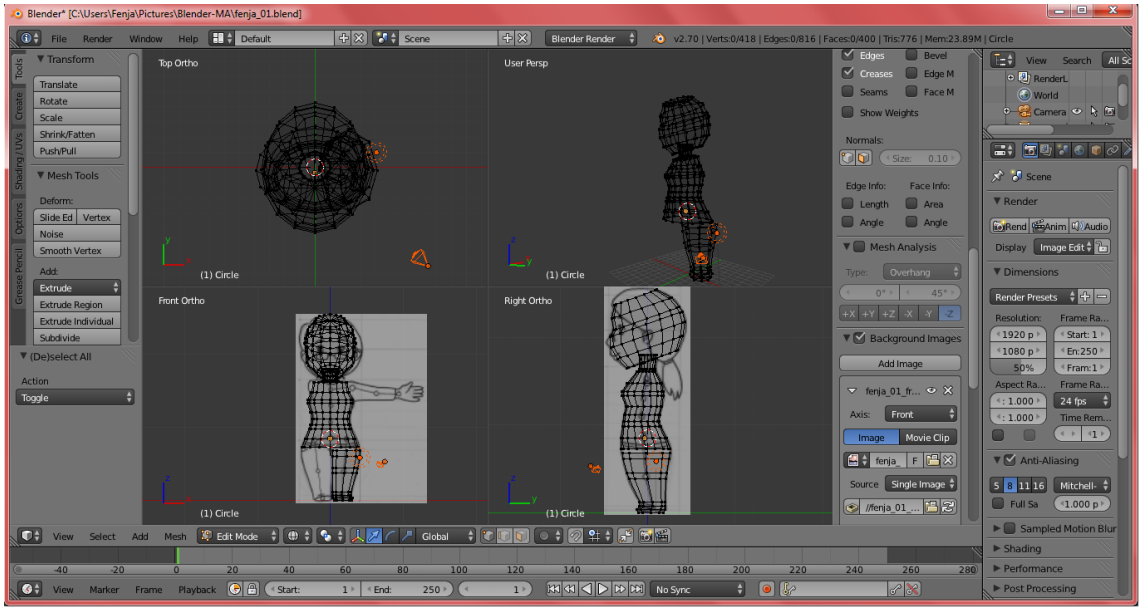


Figure 1: Mesh-Modellierung anhand von Bildvorlagen in Blender 2.6

Beim Bauen des Modells mit dieser Methode sollten möglichst wenig Unterteilungen des Kreis-Meshes verwendet werden, um den Modellierungsprozess einfach zu halten. Um das fertige, relativ kantige Modell zu verfeinern, kann dieses geglättet werden.

Glätten Die Modellierung erfolgt, wie schon erwähnt, mit möglichst wenigen Flächen, was ein vergleichsweise grobes und kantiges Modell zur Folge hat. Um diese unerwünschten Kanten zu glättet bietet Blender eigene Funktionalitäten an.

Einzelne Flächen können ausgewählt und mit *Smooth* (Tool Shelf) geglättet werden. [17, Polygonglättung - Smoothing, S.99]

Für großflächigeres Glätten kann ein *Modifier* angewendet werden. Hierzu wird ein *Edge Split Modifier* hinzugefügt. Mit Hilfe eines Reglers kann die Intensität des Glättens gesteuert werden. Das geglättete Modell ist in Abbildung 3 zu sehen.

In das nun fertige Modell wird nun ein Skelett hinzugefügt, um Bewegungen kontrollieren zu können.

2.2.3 Skelett

Das Skelett sorgt dafür, dass das 3D-Modell später Bewegungen durchführen kann, indem das enthaltene Skelett angepasst wird. Hierbei folgt das Modell den Biegungen von Gelenken, wie es auch beim menschlichen Skelett der Fall ist.

Ein Skelett oder auch *Armature* wird mit Hilfe des Befehls *Add Armature Bone* hinzugefügt. Weitere Bones können wie beim Bau des Meshes mit *extrude* angefügt werden. Es ist hierbei wichtig, dass alle Bones miteinander verbunden sind. Abbildung 2 zeigt das Mesh mit seinem Skelett. Die Bones sollten direkt beim Erstellen benannt werden, damit sie später über sinnvolle Namen angesprochen werden können. Für symmetrische Knochen ist eine links-rechts-spezifische Benennung Standard. Je nach Körperseite wird also z.B. der Armknochen *Arm.l*, bzw. *Arm.r* benannt. [17, 2]

Die weiteren Arbeitsschritte dienen der Verbindung des erstellten Skeletts mit dem Mesh-Objekt, sodass bei einer Bewegung der Bones das Mesh mit bewegt wird.

Das Roll der Bones wird neu berechnet und bei etwaigen Veränderungen die Rotation zurück gesetzt. Im Objekt Mode wird erst das Mesh, dann die "Armature" markieren. Skelett und Mesh werden aneinander gebunden, vorzugsweise mit der Einstellung *With automatic Weights*, sodass beim Bewegen der Bones sich das Mesh mit bewegt.

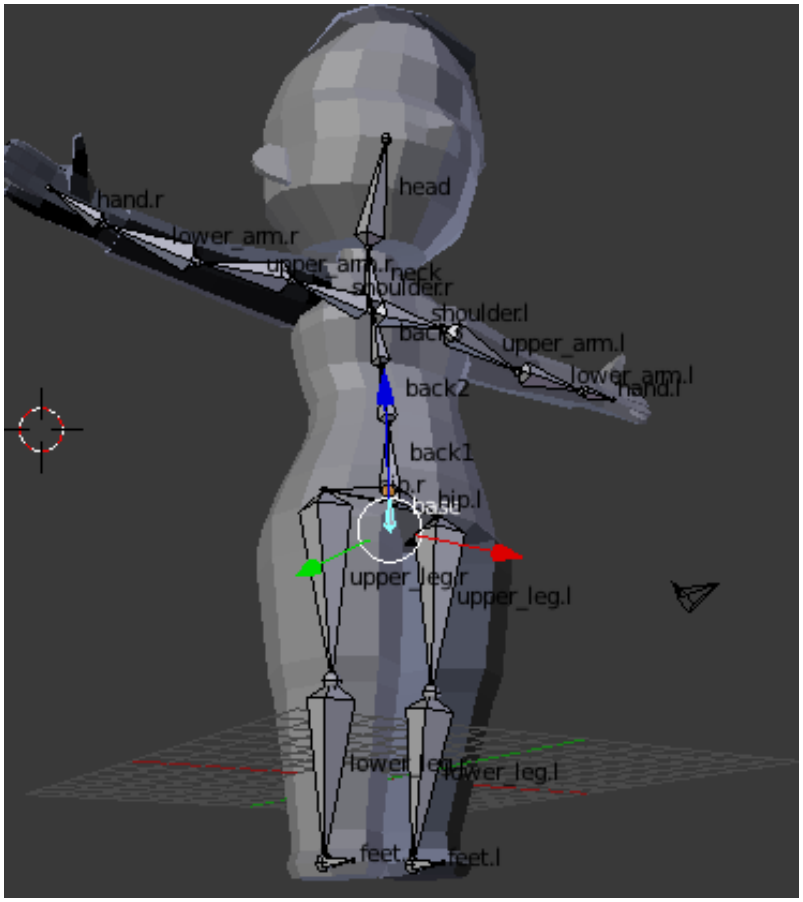


Figure 2: Anzeige der Namen an den Bones in Blender 2.6

Gewichtung Die Gewichtung beschreibt das Skinning des 3D-Modells, also inwiefern eine Bewegung des Skeletts auf das restliche Mesh, genauer auf die Vertices, einwirkt. Es gibt verschiedene Methoden, um die Gewichtung zu erstellen, deren Unterschied darin liegt, wie sie mit dem Übergang von "full weight" (komplette Beeinflussung, 1.0) und keinerlei Einfluss (0.0) verfahren. [10]

With Automatic Weights setzt die Gewichtungen bei der Verbindung von Skelett und Mesh automatisch. Hierbei wird eine lineare Gewichtung eingesetzt. Die Vertices um den Knochen (z.B. der Oberarm) sind gänzlich von einer Verformung betroffen. Zu beiden Seiten hin nimmt der Einfluss progressiv ab, bis eine Verformung keinen Einfluss mehr auf Vertices des Meshes hat (im Beispiel Ellbogen und Schulter). [2]

Da es sich hier aber um das automatisierte Anwenden einer Gewichtungsmethode handelt, kann diese fehlerbehaftet sein. In diesem Falle muss die Gewichtung, also die durch die Transformation eines Knochens hervorgerufene Verformung, händisch angepasst werden.

Verformung Um die nicht durch die automatische Gewichtung abgedeckte Verformung nachzubessern gibt es in Blender ein eigenes Tool.

Im *Weight Paint Mode*, erreichbar durch das Selektieren des Meshes im *Pose Mode*, können mit der rechten Maustaste einzelne Bones selektiert werden. Mit gedrückter linker Maustaste kann das Mesh nun eingefärbt werden. Die Einfärbung entspricht der Verformung, welche das Mesh beim Bewegen des Bones ausführt. [17, 8.4.3 Verformung Kontrollieren, S.240]

Sichtbar ist dieser Arbeitsschritt in Abbildung 4. Besonders muss hier die Verformung an den Ellbogen und Knien beachtet werden. Wichtig ist auch, die angepasste Verformung aus mehreren Blickwinkeln zu kontrollieren.

Dieser Schritt würde bei einer Automatisierung der 3D-Modell-Erstellung einen der Schwierigsten darstellen, da die Anpassung der Verformung nach Augenmaß des Benutzers stattfindet.

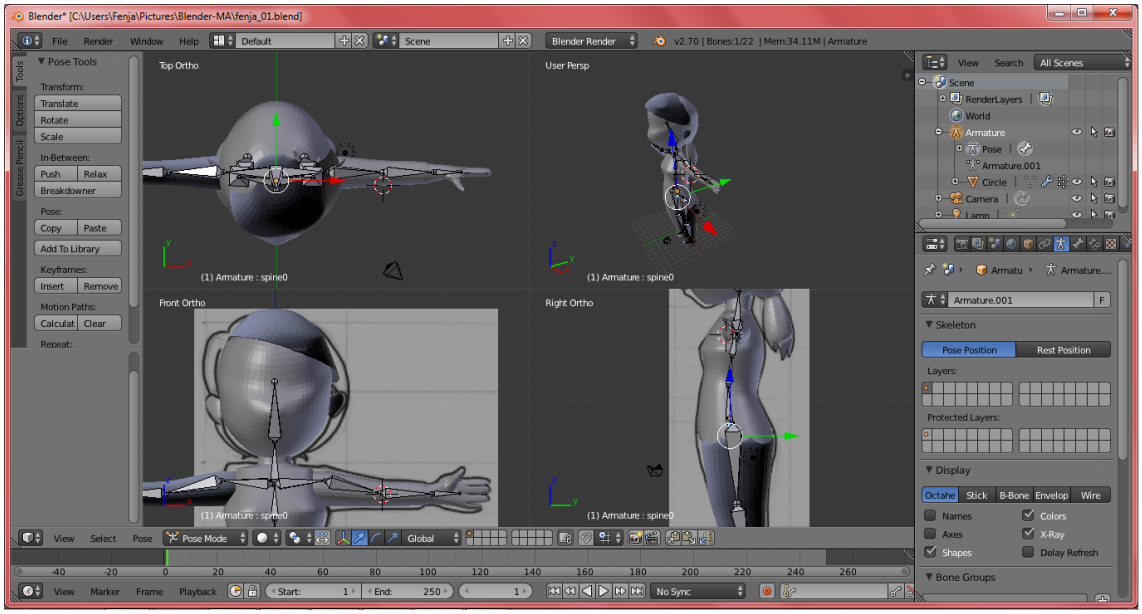


Figure 3: Skelett setzen in einem geglätteten Modell in Blender 2.6

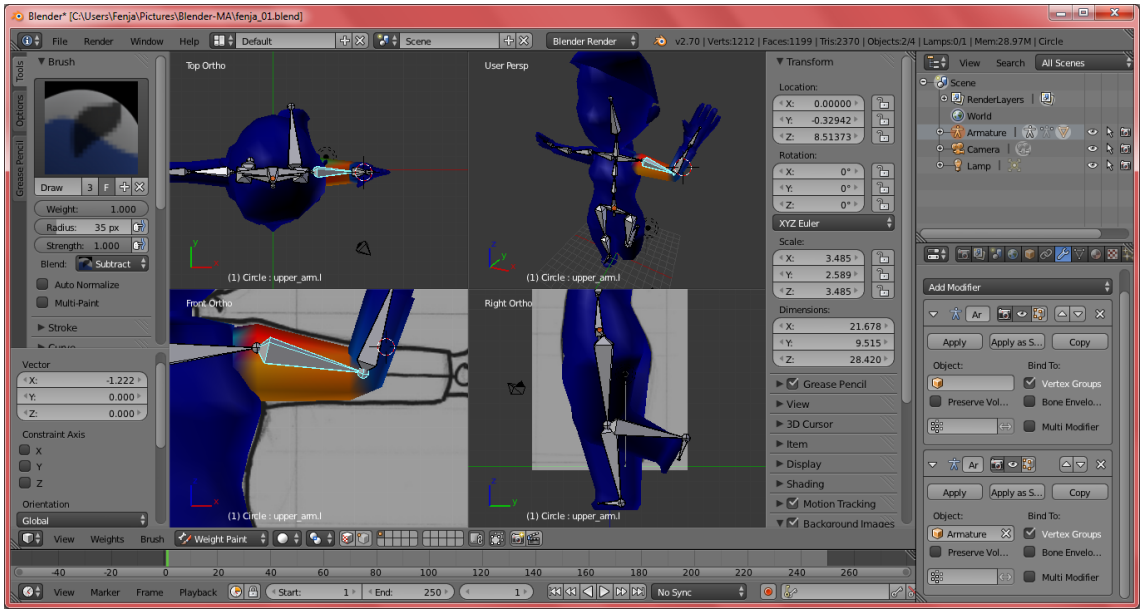


Figure 4: Anpassung der Verformung eines 3D-Modells in Blender 2.6

2.2.4 Was ist zu beachten?

Bei der Modellierung sind einige Besonderheiten über alle Arbeitsschritte hinweg zu beachten, besonders aber bei der initialen Erstellung des Meshes.

Es sollte immer vor Augen gehalten werden, dass die letztliche Bearbeitung auf den ungeglätteten Kanten und der Silhouette aufbaut. Dies bedeutet, dass für die Darstellung von Kleidungsstücken, also Ärmelsaum, Kragen, etc. entsprechende, scharfe Kanten in das Mesh eingebaut werden müssen.

Weiterhin ist es sinnvoll, das Mesh an den Gelenken weiter zu unterteilen, um die Verformung beim Bewegen der Bones zu erleichtern.

2.3 3D-Modell

Ein 3D-Modell ist das Ergebnis der Arbeitsschritte Modellierung und dem Einfügen eines Skeletts. Es kann mit einem beliebigen Tool erstellt werden und stellt im Beispiel einen Comic-Charakter dar. Das 3D-Modell zeigt den großen Vorteil dieses Ansatzes für den Comic Generator. Dank der dritten Dimension stehen eine Vielzahl von Ansichten auf das Modell zur Verfügung, sodass aus verschiedenen Perspektiven gewählt werden kann. Dadurch, dass diese Wahl für jedes Comic-Panel neu getroffen werden kann, wird mehr Variabilität und somit ein abwechslungsreiches Leseerlebnis ermöglicht.

Das in den vorhergegangenen Arbeitsschritten entstandene 3D-Modell verfügt nun über Detail und Eigenschaften, die für das weitere Vorgehen nützlich, bzw. notwendig sind.

Wie im vorherigen Abschnitt erwähnt müssen Kanten im Modell die Outlines der fertigen Comic-Figur begünstigen. Dies betrifft Haare, Gesicht und Kleidung. Das aktuell in PJ1 verwendete 3D-Modell erfüllt diese Anforderungen noch nicht. Allerdings reicht es aus, um die Anwendung des Toon-Shaders zu demonstrieren.

Eine weitere, wichtige Eigenschaft ist das Skelett des 3D-Modells, auf welches im nächsten Schritt zugegriffen werden soll, um Gestik und Mimik anzupassen. Dieser Arbeitsschritt ist noch nicht in PJ1 enthalten, für die Weiterarbeit allerdings notwendig und somit eine wichtige Eigenschaft, des 3D-Modells.

Technisch gesehen ist der Charakter-Mesh in einer entspannten Haltung zu modellieren, um möglichst wenig Anpassungen an Skelett und Verformung vornehmen zu müssen. Wie zuvor bereits erwähnt ist es zusätzlich wichtig, an den Stellen der Gelenke viele Unterteilungen des Mesh-Gitters zu setzen.

Das fertige, geglättete Modell mit Skelett wird nun weiter verarbeitet.

Export Zur Weiterverarbeitung muss das Blender-Modell in ein möglichst allgemeines Format exportiert werden. Beim Export aus Blender ist es wichtig, das Modell mit zu exportieren, da auf dieses später auch zugegriffen werden soll. Nach einem Export ins FBX-Format sorgt ein Converter für eine Überführung in das Format 3dgb. Dieses kann z.B. von LibGDX importiert werden. [16]

2.4 LibGDX

Zur Weiterverarbeitung des 3D-Modells muss die Datei, die es enthält von einem anderen System geladen und verarbeitet werden können. Dieses muss in der Lage sein, durch Positionierung der Kamera einen Blickwinkel auf das Modell und eine Größe des dargestellten Ausschnitts festzulegen. Weiterhin muss der Comic-Shader angewendet werden können, um den gewünschten Comic-Stil zu erhalten. Das in dieser Arbeit gewählte Framework hierfür ist LibGDX.

LibGDX ist ein Framework zur Spiele-Entwicklung auf Basis von Java. Es ist cross-plattform und unterstützt WebGL.

Mit LibGDX kann die Applikation in der *core* Variante implementiert und mit ggf. kleinen Anpassungen für Android, iOS, Desktop und den Browser ausgespielt werden. [18]

Die Software zur Verarbeitung der 3D-Modelle muss in erster Linie Modelle aus Blender, mitsamt dem Skelett importieren können. Open Source und freie Nutzung sind wie schon bei Blender Hauptkriterien. Innerhalb des Systems sollten Shader anwendbar sein und ein Export des Comic-Panels im einfachen Bildformat möglich sein.

Eine spätere Einbindung in eine Nutzeroberfläche sollte möglich sein, um ein späteres Comic-Programm möglichst benutzerfreundlich umsetzen zu können.

2.4.1 Allgemeines

LibGDX erfüllt vor allem die Anforderungen einer frei verfügbaren Software. Im Hintergrund nutzt LibGDX OpenGL.

Das cross-plattform Prinzip mit der Entwicklung einer Library, die oberflächenunabhängig von *Main Klassen* genutzt werden kann, z.B. durch den *Desktop Launcher*, ermöglicht ein breites Anwendungsfeld.[16]

Zur Einarbeitung und Bekanntmachung mit den Grundlagen stehen zudem mehrere Bücher zur Verfügung [3, 12, 13].

An 3D-Frameworks steht eine ganze Bandbreite zur Verfügung. So wären z.B. Ogre3D, Unity3D und three.js denkbare Alternativen zu LibGDX. three.js wird an dieser Stelle kurz vorgestellt, da mit diesem Framework bereits Erfahrungen gemacht wurden (Kurs *Next Media* im Bachelor und ein Udacity-Kurs zu Webgrafiken).

three.js Eine mögliche Alternative zu LibGDX ist three.js. Es ist ebenfalls open source, der Code ist auf GitHub. three.js basiert auf JavaScript und ist spezialisiert für den Einsatz in WebGL Apps und im Webbrowser. Ein großer Vorteil ist, dass das Kompilieren entfällt und Ergebnisse direkt angezeigt werden können.

Die Wahl fiel auf LibGDX, welches vorrangig für die Implementierung für den Desktop und Android gedacht ist. Eine Empfehlung, vor allem aufgrund der cross-plattform Fähigkeit wurde vom Kommilitonen Raimund Wege ausgesprochen.

Ein kleiner Nachteil von LibGDX ist, dass bei der Recherche im Internet viele Lösungen mit einer alten Version (GL10) gefunden werden, aktuell ist GL20 im Einsatz. Es bleibt also nicht auszuschließen, dass ein Wechsel auf three.js vorgenommen werden muss, wenn LibGDX essentielle Funktionen nicht unterstützt.

Einrichtung Das Einrichten eines LibGDX-Projektes funktioniert sehr komfortabel über eine Executable. In einem Dialog können Details zum Projekt angegeben werden und das Projekt wird für die gewünschte Entwicklungsumgebung und die ausgewählten Plattformen (HTML, Desktop, Android) mit Gradle eingerichtet.

Implementiert wird im Ordner *core*.

2.4.2 Vorgehensweise

LibGDX muss zunächst das Blender-Modell laden. Vom *Model* wird eine *ModelInstance* erzeugt und eine *Environment* angepasst. Am wichtigsten ist die Positionierung einer *orthographischen Kamera*, mit Blick auf das Modell. Hierfür werden aktuell Koordinaten hart gecodet, dieser Teil soll später automatisiert werden.

Für die Comic-Darstellung wird in einem *ModelBatch* mit einem *Shader* die *ModelInstance* gerendert und ausgegeben. Das fertige Bild wird exportiert und stellt ein Comic-Panel dar.

Methoden LibGDX teilt den Code zum Rendern in drei Haupt-Methoden auf. **create** erzeugt alle benötigten Instanzen, wie Modell, Kamera und ShaderProgramme, **render** ist für die Anwendung von Shadern und die Anzeige des Ergebnisses zuständig und **dispose** gibt den beanspruchten Speicherplatz wieder frei, um dies nicht der Garbage-Collection von Java überlassen zu müssen.

Modell laden Das Model in LibGDX wird mit dem Laden der 3dgb-Datei initialisiert. Um Änderungen an dem Modell durchführen zu können wird eine ModelInstance des Models erzeugt. Auf dieser Instanz werden in einem späteren Schritt die Shader angewendet.

Die Kamera-Positionierung passiert im 3D-Raum durch Anpassen der Koordinaten. Eingegeben werden die Position der Kamera und auf welchen Punkt sie gerichtet ist.

2.5 Shader

Ein Shader, wie er für das Vorgehen dieser Arbeit wichtig ist, besteht aus zwei Komponenten. Der Vertex-Shader iteriert über alle Vertices der Szene und wählt ggf. aus, welche davon dargestellt werden. Diese werden an den Fragment-Shader oder auch Pixel-Shader weiter gereicht, der wiederum festlegt, welche und wie die einzelnen Fragmente (Pixel) dargestellt werden. Die Rendering-Pipeline ist in Abbildung 5 dargestellt. Für den Comic-Shader übernimmt der Fragment-Shader die Arbeit der Kantenerkennung und stellt die gefundenen Kanten-Fragmente schwarz dar.

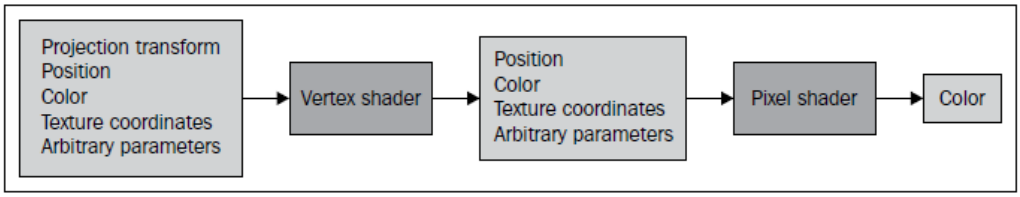


Figure 5: Rendering-Pipeline [12, S.133]

Die in LibGDX verwendeten Shader sind in der *Open Shading Language* verfasst und werden als glsl-Dateien abgespeichert und zur Verwendung im *ShaderProgram* geladen. Ein Shader besteht immer aus einem Vertex- und einem Fragment-Shader.

Der in PJ1 umgesetzte Comic-Shader ist nach dem Vorbild von Decaudin, welches in AW2 [7] vorgestellt wurde, entstanden. [4] Eine Übersicht über die Arbeitsweise der verschiedenen Shader liefert Abbildung 6.

Neben dem so erstellten Comic-Shader kommen in LibGDX noch weitere Shader zum Einsatz, so z.B. der Geometry-Shader, diese werden für das Vorgehen jedoch nicht benötigt.

Eine nützliche Hilfestellung war ein, inzwischen veraltetes, Tutorial [14], welches besonders für die Berechnung der Tiefenmap eine gute Hilfestellung darstellte.

FrameBuffer Die Arbeitsschritte sehen vor, dass die 3D-Szene zunächst als Tiefen- bzw. Normalenmap gerendert wird. Dieses Zwischenergebnis muss gespeichert und dann weiter verarbeitet werden. Das Zwischenspeichern geschieht mit Hilfe eines *FrameBufferObjects* (fbo), in welchem eine *ModelInstance* mit einem entsprechenden Shader als Tiefenmap gerendert wird. Das Ergebnis wird als Textur an einen *ModelBatch* übergeben. Dieser berechnet mit Hilfe der Tiefenwerte und einem Kantenerkennungs-Algortihmus die Silhouette des Objekts.

Zur Erkennung der inneren Kanten war kein zusätzlicher Arbeitsschritt nötig. In der *environment* konnten einfach farbige, gerichtete Lichtquellen platziert, und so eine Normalenmap erstellt werden. Die Kantenerkennung funktioniert äquivalent, anhand der Unterschiede in den Farbwerten.

Addition Die entstandenen Kantendarstellungen werden zuletzt übereinander gelegt und dargestellt. Später kommt hier ein Hintergrund und ggf. Texturen des Objektes hinzu. Das entstandene Bild ist das 3D-Modell aus der von der Kamera gewählten Perspektive, im Comic Stil.

2.6 Comic Panel

Ergebnis der Arbeit ist ein Comic-Panel. Aus der mit Hilfe von 3D-Modellen beschriebenen Szene wird ein Ausschnitt ausgewählt, der durch den Comic-Shader in Comic-Stil dargestellt wird. Erhalten wird also ein Bild, in welchem die 3D-Szene mit Outlines zu sehen ist.

Das fertige Comic-Bild wird im letzten Schritt exportiert. Dies geschieht über das Aufnehmen eines Screenshot und dem Abspeichern als *png* (oder einem beliebigen anderen Bildformat).

Ein Schritt, der noch vor den Bildexport gesetzt werden muss ist das Einfügen von Sprechblasen. Dies wurde nicht im Rahmen von PJ1 umgesetzt, folgt aber später im Zuge der Sprechblasenplatzierung [11].

Mehrere, auf diesem Wege erhaltene Panel werden zuletzt zu einer fertigen Comic-Seite zusammen gesetzt. Dies geschieht hierarchisch nach dem Vorbild von Comic Computing [15]. Ein

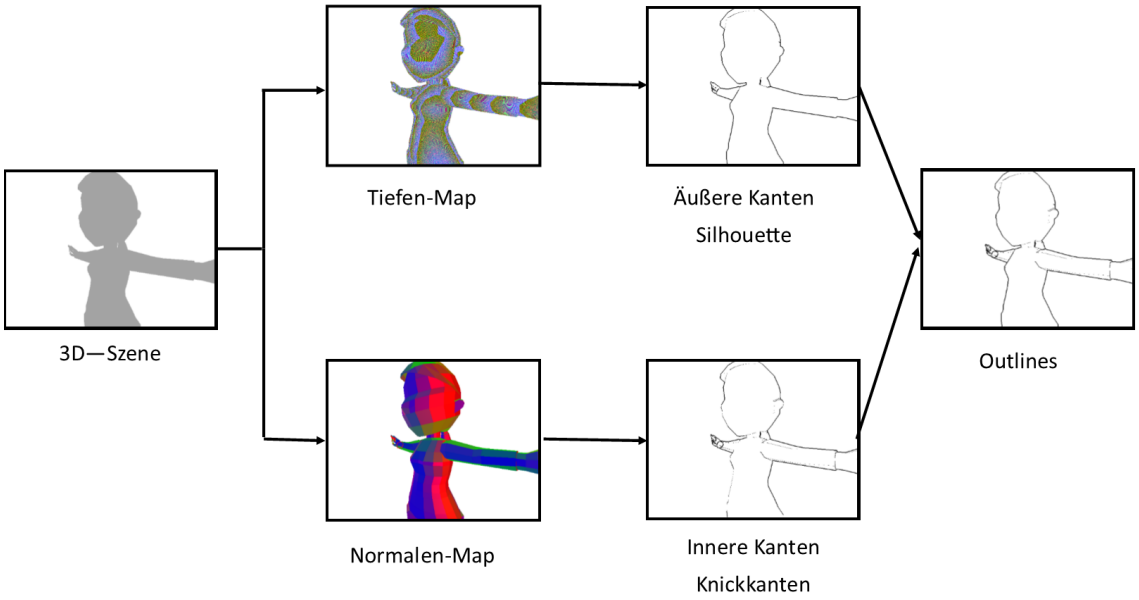


Figure 6: Comic-Shader nach [4]

einzelnes Abspeichern der Bilder ermöglicht zudem eine höhere Flexibilität bei der späteren Darstellung, sodass z.B. eine Anpassung in mobiler Ansicht umgesetzt werden kann.

3 Schluss

Projekt 1 konnte erfolgreich zu Ende gebracht werden. Es fand eine Einarbeitung in Blender statt, wodurch es möglich war, 3D-Modelle zu erstellen. Diese Modelle konnten in LibGDX geladen und angezeigt werden. Nach einer Einarbeitung in die Funktionsweise von Shadern und die Open Shading Language wurde schließlich der Comic-Shader umgesetzt und somit ein Comic-Panel erhalten.

Das Zurechtkommen mit der zuvor unbekannten Software (Blender, LibGDX, GLSL), welches zuvor als Problematik eingestuft wurde, kann als erfolgreich verbucht werden. Zwar kann vor allem in Blender mehr Übung noch bessere Ergebnisse erzielen, doch die Grundkenntnisse sind nun angeeignet und können als Grundlage für die Weiterverarbeitung der 3D-Modelle dienen.

Bereits während der letzten Arbeitsschritte im PJ1 wurde stetig ein Blick nach vorne und in Richtung weiterführender Arbeitsschritte geworfen. So steht als nächstes der grobe Aufbau einer Architektur an, in welcher der Comic-Shader aus diesem Projekt ein Modul darstellen wird. Viele derzeit manuelle Eingaben werden an Parameter ausgelagert, sodass das fertige Modul eine beliebige Szene zu einem Comic-Panel umwandeln kann. Es gilt als nächstes, besagte Parameterabhängigkeiten einzubauen (z.B. Panelgröße, Szene), sodass lediglich die Funktionalität des Comic-Shaders im Modul verbleibt.
Die Eingabe für den "Comic-Shader" ist eine Szene, die per DSL beschrieben und aufgebaut werden soll. Die Ausgabe, also die einzelnen Comic-Panel, sollen von einem letzten Modul, dem "Page-Builder" zu einer Comic-Seite zusammen gesetzt werden.

Als nächstes wird besonders die Manipulation des Skelettes im Vordergrund stehen, da die Gestik und Mimik vom 3D-Framework (LibGDX) geregelt werden soll. Passende Bezeichnungen, z.B. "Winken", sollen dann von der DSL angefragt werden können.

3.1 Ergebnisse

Während PJ1 ist es erfolgreich gelungen, in Blender 3D-Modelle zu erstellen. Diese wurden mit einem Skelett ausgestattet und Verformungen beim Bewegen der Bones angepasst. Die Schritte sind noch einmal übersichtlich in Abbildung 7 dargestellt.
Über den Zwischenschritt eines Exports kann das Blender-Modell in einer LibGDX-Umgebung geladen und angezeigt werden. Eine Shader-Kombination rendert die 3D-Szene im Comic-Stil und das Ergebnis wird als fertiges Comic-Panel im png-Format exportiert.

(a) Modellierung (b) Skelett und Gewichtung (c) 3D-Szene (d) Comic-Stil (e) Comic-Panel

Figure 7: Zwischenschritte: Vom 3D-Modell zum Comic Panel

3.2 Weitere Vorgehensweise

Die im Rahmen dieses Projekts erarbeiteten Ergebnisse sind der Prototyp eines Bearbeitungsschrittes der endgültigen Masterarbeit. Vor allem um die DSL zu ergänzen gilt es, für die nachfolgenden PJ2 und MS sinnvolle Aufgaben zu finden, um an die Ergebnisse anzuknüpfen. Tatsächlich gilt es, darüber hinaus noch einige Zwischenschritte auszufüllen, wie anschaulich in Abbildung 8 gezeigt wird.

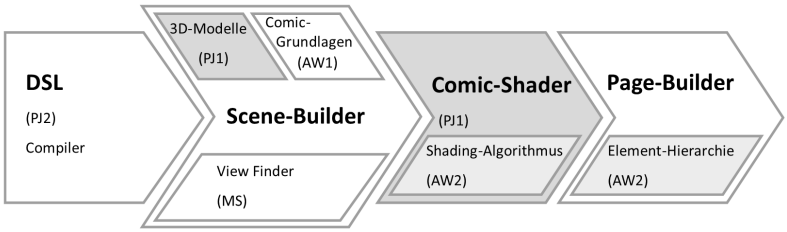


Figure 8: Vorgehensweise Masterarbeit

Die größte Schwierigkeit wird als nächstes die Auswahl des Bildausschnittes sein. Das fertige Panel soll die Gesichter aller sprechenden Charaktere anzeigen und miteinander sprechende Charaktere sollten sich ansehen. Eine Neukonfigurierung der Szene muss in Erwägung gezogen werden, wenn kein passender Bildausschnitt gefunden werden kann.

Viele Ansatzpunkte müssen im Zuge von Automatisierung vorgenommen werden. Vorbereitend auf eine Umsetzung der DSL-Anweisungen müssen Panel-Größe und Sprechblasen, sowie Gestik und Mimik der Charaktere automatisiert angepasst werden.

Wie an den beispielhaften Gegenüberstellungen von 3D-Modell und Comic-Bild in Abbildung 9 und 10 zu sehen ist, hat das bisher verwendete Modell noch einige Fehler. Für weitere Tests müssen den Anforderungen angepasste (2.3) und vor allem mehrere 3D-Modelle erstellt werden.
Nicht zuletzt muss eine DSL entwickelt werden, deren Beschreibungen in sinnvolle Befehle an das System übersetzt werden können.

3.2.1 Projekt 2

PJ2 wird sich voraussichtlich hauptsächlich mit der Implementierung der DSL beschäftigen. Es existieren DSL-Tools, auf die zurückgegriffen werden kann. Zu nennen sind hier Xtext für Eclipse [5] und MPS für IntelliJ [9].
Die spätere Hierarchie des Comics sollte bereits an der DSL ablesbar sein.

3.3 Risiken

Das größte Risiko ist, wie bereits in AW2 erwähnt [7], die Auswahl eines sinnvollen Ausschnittes, vor allem in Hinblick auf die Anforderung, dass wichtige Bestandteile der Szene zu sehen sein müssen.

Die (automatisierte) Anpassung der Gestik und Mimik ist von den Möglichkeiten von LibGDX abhängig. Im schlechtesten Falle ist ein Umschwenken auf ein anderes Framework, wie three.js nötig. Ohnehin stellen die Grenzen von LibGDX mögliche Risiken dar, da hier eine Szene aus

mehreren 3D-Modellen und einem Hintergrund zusammen gebaut werden müssen.

Zuletzt sind die DSL-Tools noch unbekannt, wodurch wiederum das Risiko der mangelnden Erfahrung mit den Tools vorliegt.

References

[1] blender.org - Home of the Blender project - Free and Open 3D Creation Software.

[2] BlenderWiki - 2.6 Manual.

[3] Balakrishnan Nair, S., and Oehlke, A. *Learning Libgdx Game Development*, second edi ed. Packt Publishing Ltd., Birmingham, 2015.

[4] Decaudin, P. Cartoon-looking rendering of 3D-scenes. *Syntim Project Inria*, June (1996), 1–11.

[5] Foundation, E. Xtext - Language Development Made Easy!, 2015.

[6] Harbke, F. Ausarbeitung: Comic generierung.

[7] Harbke, F. Comic generierung - aw 2.

[8] Igarashi, T. Computer Graphics for All. *Commun. ACM 53*, 7 (2010), 71–77.

[9] Jetbrains. Meta Programming System - Language Oriented Programming environment and DSL creation tool, 2015.

[10] Kavan, L. Part I: Direct Skinning Methods and Deformation Primitives. 1–11.

[11] Kurlander, D., Skelly, T., and Salesin, D. Comic Chat. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York and NY and USA, 1996), vol. 96 of *SIGGRAPH '96*, ACM, pp. 225–236.

[12] Márquez, D. S., and Sánchez, A. C. *Libgdx Cross-platform Game Development Cookbook*. Packt Publishing Ltd., Birmingham, 2014.

[13] Oehlke, A. *Learning Libgdx Game Development*. Packt Publishing Ltd., Birmingham, 2013.

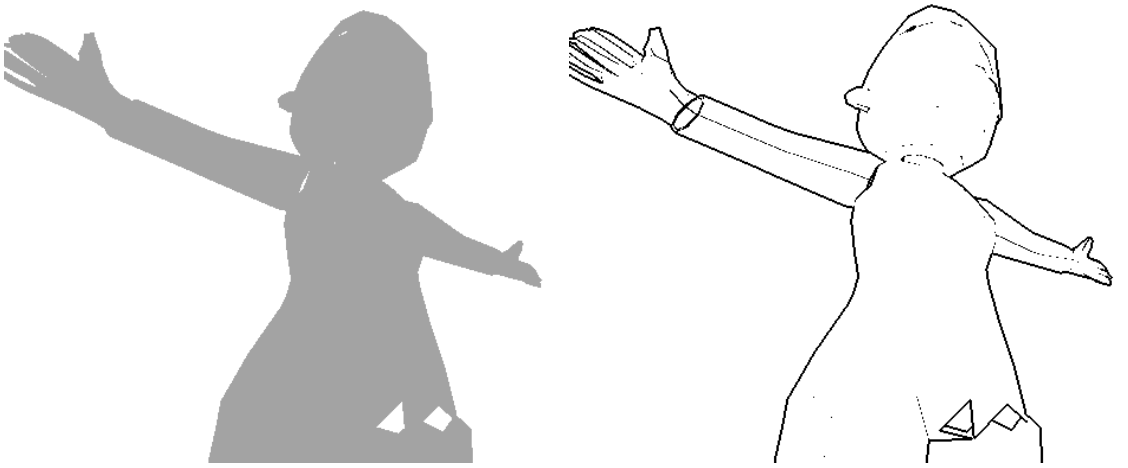
[14] Stärk, M. Tutorial: Cel-Shading With libGDX and OpenGL ES 2.0 using Post-Processing, 2013.

[15] Tobita, H. Comic computing: Creation and communication with comic. In *Proceedings of the 29th ACM International Conference on Design of Communication* (New York and NY and USA, 2011), SIGDOC '11, ACM, pp. 91–98.

[16] van Ham, J. X. 3D models and animation from Blender to LibGDX, 2014.

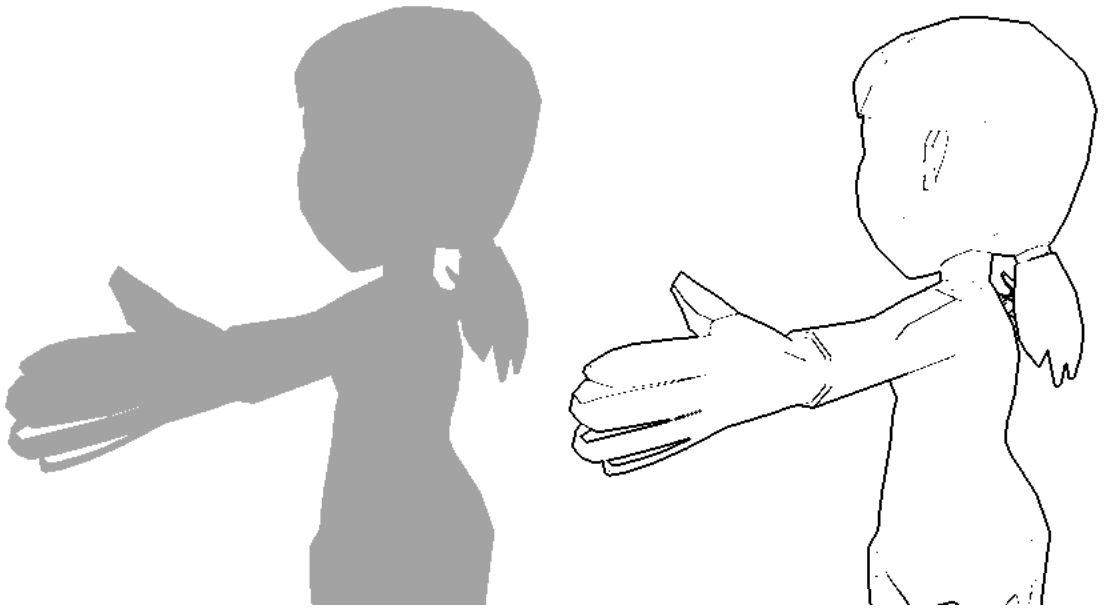
[17] Wartmann, C. *Das Blender-Buch*, 4. auflage ed. dpunkt.verlag, Heidelberg, 2011.

[18] Zechner, M. LibGDX - Homepage, 2013.



(a) Modell aus der Froschperspektive (b) Modell aus der Froschperspektive mit Comic-Shader

Figure 9: Beispielanwendung des Comic-Shaders (1)



(a) Modell in der Ansicht von hinten (b) Modell in der Ansicht von hinten mit Comic-Shader

Figure 10: Beispielanwendung des Comic-Shaders (2)