

Bachelorthesis

Erstellung von 3D-Spieleumgebungen
mit Hilfe von *Low-Polygon-Modellierung*
und dem *Camera-Projection-Mapping*

Name: Chris Wodäge

Matrikelnummer: 29784

Studiengang: Digitale Medienproduktion

Hochschule: Hochschule Bremerhaven

Betreuer: Prof. Dr. Holger Rada (Hochschule Bremerhaven)

Andrea Lenkewitz (Hochschule Bremerhaven)

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig und eigenhändig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht.

Bremen den 26.04.2015

Chris Wodäge

Inhaltsverzeichnis

Abbildungsverzeichnis	5 - 6
1. Abstract und Einleitung	7 - 9
2. Echtzeitgrafik, Nichtechtzeitgrafik, sowie <i>Projection-Mapping</i> und dessen Einsatzgebiet	10 - 12
3. Workflow des <i>Projection-Mappings</i>	13 - 31
3.1. Bau und ebenenweise Sortierung der <i>High-Polygon-Geometry</i>	14 - 16
3.2. Bau der <i>Low-Polygon-Geometry</i>	17 - 20
3.3. <i>Rendering</i> von <i>Diffuse- , Alpha- und Light-Maps</i>	21 - 27
3.4. <i>Mapping</i> der <i>Low-Polygon-Geometry</i>	28 - 31
4. Problemstellungen und Lösungsansätze	32 - 40
4.1. Texturverzerrungen	33 - 34
4.2. <i>Nonmanifold Geometry</i>	35 - 36
4.3. Probleme durch <i>Backface Culling</i>	37 - 38
4.4. Das Alphakanten Problem	39 - 40
5. Texturoptimierung	41 - 47
5.1. Vorbereitung der <i>Low-Polygon-Geometry</i>	42
5.2. UV-Entzerrung und Neuorganisation	43 - 45
5.3. <i>Rendering</i> der optimierten Texturen	46 - 47
6. Vor- und Nachteile des <i>Projection-Mappings</i>	48 - 50
7. Fazit	51
Literaturverzeichnis	52
Quellenverzeichnis	53

Abbildungsverzeichnis

2.1: Testlauf auf einem Tabletcomputer.	11
2.2: Blick in eine Spieleumgebung.	11
2.3: Spieleumgebung aus <i>The Book of Unwritten Tales</i> von KING Art Games.	12
3.1: Konzeptzeichnung einer Spieleumgebung.	14
3.2: <i>Rendering</i> der <i>High-Polygon-Geometry</i> .	14
3.3: Doppelprojektion wird durch eine Kamerabewegung sichtbar.	15
3.4: Korrekt sortierte <i>layer</i> wurden ebenenweise gerendert und in Photoshop zusammengesetzt.	16
3.5: Falsch sortierte <i>layer</i> wurden ebenenweise gerendert und in Photoshop zusammengesetzt.	16
3.6: Ein <i>mesh</i> , bestehend aus 4 quadratischen Polygonen.	17
3.7: Darstellung eines <i>High-Polygon-Models</i> (links), bestehend aus 19188 Polygonen und dem <i>Low-Polygon-Pendant</i> (rechts) mit nur 100 Polygonen, das aus einem Würfel modelliert wurde.	18
3.8: Darstellung eines <i>High-Polygon-Models</i> (links), bestehend aus 2770 Polygonen und dem <i>Low-Polygon-Pendant</i> (rechts) mit nur 40 Polygonen, das aus einem Zylinder modelliert wurde.	18
3.9: Händische Reduktion der <i>edge loops</i> , die Anzahl der Polygone beträgt (v.l.n.r.) 1536, 384 und 116.	19
3.10: Automatische Reduktion der Polygone auf (v.l.n.r.) 1536, 382 und 117.	19
3.11: Verwendung einer transparenten Textur auf einer <i>polygon-plane</i> .	20
3.12: Kamerabewegung entblößt die planare Architektur.	20
3.13: Eine bessere Bauweise, bei der Vorder- und Rückseite getrennt gelayert wurden.	20
3.14: Gesetzte Renderkamera mit Sichtkegel, sowie den zu rendernden Bereichen (grün) und den abgeschnittenen Bereichen (rot).	21
3.15: <i>Rendering</i> der ersten Ebene.	22
3.16: <i>Rendering</i> der zweiten Ebene.	22
3.17: <i>Rendering</i> der dritten Ebene.	22
3.18: Anordnung der Ebenen in Photoshop.	22
3.19: Kugel empfängt Schatten des Würfels.	23
3.20: Würfel empfängt Schatten der Kugel.	23
3.21: Boden empfängt Schatten des Würfels und der Kugel.	23
3.22: Anordnung der Ebenen in Photoshop.	23
3.23: Insgesamt 15 gerenderte <i>Diffuse-Maps</i> .	24
3.24: Eine Auswahl an (insgesamt 15) gerenderten <i>Alpha-Maps</i> .	25
3.25: Insgesamt 15 gerenderte <i>Light-Maps</i> .	26
3.26: Darstellung der Texturtypen und deren Funktion (v.o.n.u.): <i>Diffuse-Map</i> , <i>Alpha-Map</i> , <i>Diffuse-Map</i> (freigestellt), <i>Light-Map</i> und die Kombination aus <i>Light-Map</i> und freigestellter <i>Diffuse-Map</i> .	27
3.27: Textur-Raum mit Koordinatensystem und den Texturkoordinaten eines Würfels.	28
3.28: Darstellung einer orthografischen Projektion.	29
3.29: Bildschirmsicht der orthografischen Projektion.	29
3.30: Darstellung einer perspektivischen Projektion.	29
3.31: Bildschirmsicht der perspektivischen Projektion.	29
3.32: Details einer Kamera in Autodesk Maya.	30

Abbildungsverzeichnis

3.33: Quadratische Projektion mit einer Brennweite von 30mm und einem Bildwinkel von 45,89°.	30
3.34: Quadratische Projektion mit einer Brennweite von 50mm und einem Bildwinkel von 28,5°.	30
3.35: Quadratische Projektion mit einer Brennweite von 85mm und einem Bildwinkel von 17°.	31
3.36: Darstellung eines korrekten Mappings (links), eines mit verschobener Kamera (Mitte) und eines mit zu großer Brennweite der Kamera (rechts).	31
4.1: Stark verzerrte Textur auf einer <i>Low-Polygon-Geometry</i> .	33
4.2: <i>Plane</i> bestehend aus einem Polygon.	34
4.3: <i>Plane</i> bestehend aus 4 Polygonen.	34
4.4: <i>Plane</i> bestehend aus 16 Polygonen.	34
4.5: <i>Plane</i> bestehend aus 64 Polygonen.	34
4.6: Feinere Unterteilung der Geometrie beseitigt die Verzerrung.	34
4.7: Ein <i>T-shape</i> .	35
4.8: Ein <i>bow-tie shape</i> .	35
4.9: Vergleich von schlechter (links) und guter Topologie (rechts).	36
4.10: Verbotene (links) und erlaubte Modellierung (rechts) eines konkaven Polygons.	36
4.11: Verbotene (links) und erlaubte Modellierung (rechts) einer nicht-planaren Fläche.	36
4.12: Polygonales Netz mit Normalenvektoren.	37
4.13: Dasselbe polygonale Netz mit einem ausgeblendeten Polygon.	37
4.14: Eine Röhre mit normal ausgerichteten und invertierten Normalenvektoren.	38
4.15: Objekt mit weißer Alphakante und vergrößerter Darstellung.	39
4.16: Freigestelltes Objekt mit weißer Alphakante (links), resultierend aus der Hintergrundfarbe der Textur (rechts).	39
4.17: Freigestelltes Objekt mit schwarzer Alphakante (links), resultierend aus der Hintergrundfarbe der Textur (rechts).	39
4.18: Freigestelltes Objekt ohne störende Alphakante (links) und Textur mit erweiterter Pixelkante (rechts).	40
5.1: Ansicht einer unbeschnittenen Spieleumgebung.	42
5.2: Das Ergebnis ist eine Spielumgebung ohne ungenutzte Flächen.	42
5.3: Originale Textur (links) und verzerrt gerendertes Objekt (rechts).	43
5.4: Originale Textur (links) und entzerrtes, gerendertes Objekt (rechts).	43
5.5: Originale Textur (links) und entzerrtes, sowie skaliertes, gerendertes Objekt (rechts).	44
5.6: Ungeordnete, sich überlagernde Texturkoordinaten im alten <i>UV-Layout</i> .	45
5.7: Geordnetes <i>UV-Layout</i> mit Texturkoordinaten, welche auf 2 Quadrate im Textur-Raum verteilt wurden.	45
5.8: Optimierte <i>Texture-Map</i> der ersten Ebene.	46
5.9: Optimierte <i>Texture-Map</i> der zweiten Ebene.	46

1. Abstract und Einleitung

Abstract

Die Bachelorthesis mit dem Thema der »Erstellung von 3D-Spieleumgebungen mit Hilfe von *Low-Polygon-Modellierung* und dem *Camera-Projection-Mapping*«, beschäftigt sich mit dem Bau von Hintergrundgrafiken für Computerspiele, speziell mit dem von *Point-and-Click-Adventures*. Die Technik wurde im Hause des Bremer Spieleentwicklers KING Art entwickelt und konnte erfolgreich im neuesten Spiel des Studios, *The Book of Unwritten Tales 2* eingesetzt werden. Das *Projection-Mapping* fällt in den Bereich der 3D-Computergrafik und daher werden Grundlagen aus diesem Bereich, welche für den Workflow relevant sind, erläutert. So werden die Unterschiede von Echtzeit- und Nichtechtzeitgrafik und die Anwendungsfelder für das *Projection-Mapping* beleuchtet. Zusätzlich zum Workflow werden häufige Fehlerquellen und die Texturoptimierung thematisiert. Zuletzt sollen Vor- und Nachteile, sowie ein Fazit die Arbeit abschließen.

Einleitung

Die vorliegende Bachelorthesis beschäftigt sich mit dem Thema der Erstellung von 3D-Spieleumgebungen mit Hilfe von *Low-Polygon-Modellierung* und dem *Camera-Projection-Mapping*. Konkret handelt es sich um die Erstellung von Hintergrundgrafiken für Computerspiele, vorrangig für *Point-and-Click-Adventures*, in denen ein Spieler Figuren steuert, Rätsel löst und mit anderen Spielfiguren interagiert. Das Besondere an den betrachteten Spieleumgebungen ist die Bauweise, die sich von bisherigen Spielen im Genre des *Point-and-Click-Adventures* unterscheidet. Während sonst auf flache Hintergründe gesetzt wurde, werden mit Hilfe des betrachteten Workflows komplexe Szenen mit einfachen Polygonnetzen, also der *Low-Polygon-Geometry* komplett nachgebildet. Mit Hilfe einer Kameraprojektion werden hernach hoch aufgelöste Texturen auf die einfachen Polygonnetze projiziert. Das Ergebnis sind sehr detaillierte,

1. Abstract und Einleitung

aber auch sehr performante Hintergrundgrafiken, die im Gegensatz zur flachen Bauweise in älteren Spielen Kamerabewegungen und die damit einhergehende Bewegungsparallaxe erlauben.

Der Workflow für diese Technik wurde im Hause des Bremer Spieleentwicklers KING Art entwickelt und konnte erfolgreich für das im Februar 2015 erschienene *Point-and-Click-Adventure The Book of Unwritten Tales 2* verwendet werden. Alle Arbeitsschritte für diesen Workflow konnten aufgrund des Know-hows im Bereich der 3D-Computergrafik ausgearbeitet und erfolgreich zur Anwendung gebracht werden.

Um eine solch komplexe Arbeitsweise beschreiben zu können, müssen neben dem reinen Workflow auch Grundlagen erläutert werden, die für das Verständnis der Funktion wichtig sind. Zum ersten wäre da eine Einführung in das Themengebiet der 3D-Computergrafik und deren Einteilung in die Echtzeit- und die Nichtechtzeitgrafik, denn während Spieleumgebungen entweder in den Bereich der Echtzeitgrafik oder der Nichtechtzeitgrafik fallen, versucht das *Camera-Projection-Mapping* beide Ansätze zu kombinieren, um möglichst detaillierte und sehr performante Hintergrundgrafiken zu ermöglichen. Ebenfalls werden die Möglichkeiten der Verwendung solcher Hintergründe beleuchtet, denn nicht für jeden Fall ist diese Technik auch geeignet. Für diese Betrachtung werden Spiel-Genres genannt, die sich diese Technik zu Nutze machen können, sowie solche, für die der Workflow des *Projection-Mappings* keinen Sinn macht.

Das erste Kapitel, das sich konkret mit der Erstellung der Hintergrundgrafiken auseinander setzt gliedert sich in vier Unterpunkte und beschreibt ausführlich den gesamten Workflow, mit dem die Spieleumgebungen erstellt wurden. Betrachtet wird hierbei der Bau und die ebenenweise Sortierung der *High-Polygon-Geometry*, welche gewissermaßen die Grundlage zum Bau der Spieleumgebung bilden. Weiterhin wird auf Basis der *High-Polygon-Geometry* der Bau der *Low-Polygon-Geometry* beschrieben, die später auch in das Spiel integriert wird. Zusätzlich muss die *Low-Polygon-Geometry* texturiert werden. Dazu wird beschrieben, wie aus der *High-Polygon-Geometry* Texturen gerendert werden, welche im weiteren Verlauf des Workflows auf die Hintergrund-Geometrie aufgetragen werden. Das korrekte auftragen der Texturen auf die Szene thematisiert der letzte Unterpunkt des Workflows. Das *Mapping* der *Low-Polygon-Geometry* anhand einer perspektivischen Kameraprojektion, die sozusagen Texturen und Geometrie vereint.

Häufig traten bei der Erstellung der Hintergrundgrafiken Probleme und Fehler auf, die behoben werden wollten. Speziell für diese Fälle ist ein ganzes Kapitel vorgesehen, was Fehler thematisiert und Lösungen aufzeigt. So werden unschöne Texturverzerrungen, falsche Modellierungsweisen, unsichtbare Szenenobjekte, wie auch Farbsäume an den Objekten betrachtet und behoben.

1. Abstract und Einleitung

Ferner soll die Optimierung der Texturen erläutert werden. Dieser Punkt könnte in das Kapitel des Workflow des *Projection-Mappings* integriert werden. Da die Optimierung aber keinen Einfluss auf die Funktion der Technik hat und eher zur Steigerung der Performance des Systems gedacht ist, bekommt dieser durchaus wichtige Arbeitsgang ein eigenes Kapitel. Thematisch setzt es sich mit dem Zusammenfassen von diversen Texturen mit viel ungenutztem Platz zu wenigen voll ausgenutzten Texturen auseinander. Anhand eines Beispiels soll erklärt werden, wie wirkungsvoll eine Reduktion der Texturen und die damit einhergehende Einsparung von Grafikspeicher sein kann.

Des Weiteren werden die Vorteile, sowie die Nachteile der Technik erläutert, wobei sich die Vorteile vor allem auf die Möglichkeiten der Inszenierung solcher Szenen beziehen, während die Nachteile sich eher auf finanzieller Ebene niederschlagen. Eine Möglichkeit der Finanzierung von optionalen Features, die nicht relevant für die Funktion eines Spiels sind, aber trotzdem das Gesamtergebnis deutlich aufwerten, soll mit dem Beispiel von Kickstarter.com aufgezeigt werden.

Zuletzt soll ein Fazit die gesamte Zeit der Entwicklung, sowie den Einsatz der Technik in ein reales und bereits erschienenes Produkt einordnen. Ein persönlicher Eindruck der Arbeit am Spiel *The Book of Unwritten Tales 2* soll gegeben werden, um das gesamte Thema abzurunden und damit auch abzuschließen.

2. Echtzeitgrafik, Nichtehtzeitgrafik, sowie *Projection-Mapping* und dessen Einsatzgebiet

Die Computergrafik, die ein Teilgebiet der Informatik ist, hat ihren Ursprung schon zu Beginn der 60er-Jahre. Heute bedient die computergenerierte Grafik ein breites Anwendungsspektrum. Kaum ein Kinofilm und kein einziges Computerspiel kann auf die Computergrafik verzichten, da dieses mächtige Werkzeug es erst möglich macht, jede nur erdenkliche Welt am Rechner zu erschaffen.

Ein Vergleich zwischen einem 3D-Animationsfilm und einem Computerspiel offenbart direkt die offensichtlichste Gemeinsamkeit, beide haben ihren Ursprung in der Computergrafik und wurden mit Hilfe von 3D-Computergrafikprogrammen wie Maya, 3ds Max, Cinema 4D oder Blender erschaffen.

Der größte Unterschied dieser beiden Medien ist die Interaktivität, also die Reaktion des Systems auf die Eingabe des Spielers. So ist es in einem interaktiven Computerspiel möglich die Spieleumgebung mit Hilfe von Eingaben über Maus, Tastatur, aber auch Touchscreens und anderen Eingabegeräten zu manipulieren. Hierbei wird von Echtzeit-3D-Computergrafik gesprochen. Der Betrachter eines Films hat diese Möglichkeit nicht, ein aktiver Eingriff ins Geschehen ist nicht möglich. Es handelt sich um eine Nichtehtzeit-3D-Computergrafik.

Die Trennlinie zwischen Echtzeit- und Nichtehtzeitgrafik verläuft bei einer Bildgenerierrate von *einem* Hertz (Hz). Wenn nun das finale Rendern, also die Erzeugung *eines* Bildes, mit benutzten Materialien, Animationen und Effekten mehr als *eine* Sekunde in Anspruch nimmt, dann wird von Nichtehtzeitgrafik gesprochen. Die Berechnung von Einzelbildern eines 3D-Animationsfilms kann mitunter mehrere Stunden in Anspruch nehmen, da aufwendige Beleuchtungsverfahren vor allem ein realitätsnahes Aussehen zeichnen. In Computerspielen steht dagegen besonders die Spielbarkeit im Vordergrund. Eine Bildfrequenz von 30 Hz (30 Bilder/Sekunde) sind für ein gutes Spielerlebnis notwendig.¹

1. vgl. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*.

3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 24, 25, 27.

2. Echtzeitgrafik, Nichtechtzeitgrafik, sowie *Projection-Mapping* und dessen Einsatzgebiet

Das *Camera-Projection-Mapping* ist ein Ansatz, welcher den Anspruch hat die Vorteile von Echtzeitgrafik und Nichtechtzeitgrafik zu vereinen. Es wird eine performante Spieleumgebung geschaffen, die Kamerabewegungen erlaubt und die damit einhergehende Parallaxe,² welche durch eine perspektivische Kameraprojektion entsteht, korrekt darstellt. Zusätzlich zur Spielbarkeit steht ebenfalls die Qualität der grafischen Darstellung im Vordergrund, die extrem detaillierte bzw. fotorealistische Hintergründe ermöglicht. Die mit dieser Technik erstellten Spieleumgebungen sind zudem so performant, dass diese selbst auf einem Tabletcomputer flüssig dargestellt werden können.

Der Aufbau dieser Spieleumgebungen ist vergleichbar mit einer Theaterbühne. Während der Spieler quasi aus Sicht des Publikums auf die digitale Bühne blickt, steuert dieser das Spiel, navigiert Charaktere oder löst Rätsel. Trotz des sehr hohen Detailgrades der Spieleumgebung, ist es dem Spieler möglich die Spielfiguren flüssig durch das Level zu steuern, denn die Bühne setzt sich nur aus wenigen tausend Polygonen³ zusammen (*Low-Polygon-Geometry*). Erst hoch aufgelöste *Renderings*, welche auf die *Low-Polygon-Geometry* projiziert werden, verleihen der digitalen Bühne ein sehr detailreiches Äußeres.

Zusammenfassend lässt sich demnach feststellen, dass die *Low-Polygon-Geometry*, welche die Grundlage der Spieleumgebung bildet, in den Bereich der Echtzeitgrafik fällt, da sich diese ohne großen Aufwand mit einer Bildgenerierrate von 30 Hz oder mehr berechnen lässt. Auch texturiert ist dies immer noch der Fall. Einzig die Texturen fallen in die Kategorie der Nichtechtzeitgrafik, da diese auf einem aufwendig modellierten, texturierten und komplex beleuchteten Modell basieren. Diese *High-Polygon-Version* der Spieleumgebung setzt sich zum Vergleich aus mehreren Millionen Polygonen zusammen. Das Rendern einer solchen Szene nimmt je nach Komplexität und zur Verfügung stehender Hardware mehrere Minuten oder sogar wenige Stunden pro Bild in Anspruch. So betrachtet handelt es sich um eine Kombination



Abb. 2.1: Testlauf auf einem Tabletcomputer.



Abb. 2.2: Blick in eine Spieleumgebung.

2. Die Scheinbare Positionsveränderung von Objekten, die bei Eigenbewegung eines Betrachters entstehen.

3. Polygon auch n -gon, n -Eck oder Vieleck ist eine von Strecken begrenzte geometrische Figur (Drei-, Vier-, Fünfeck etc.).

2. Echtzeitgrafik, Nichtechtzeitgrafik, sowie *Projection-Mapping* und dessen Einsatzgebiet

Wie schon erläutert, ist einer der großen Vorteile des *Projection-Mappings* der sehr performante Betrieb, weshalb sich diese Technik besonders für Computerspiele eignet. Jedoch nicht in jedem Computerspiel-Genre kann diese Technik sinnvoll integriert werden. Das hängt zum einen mit der Bauweise der *Low-Polygon-Geometrie* und zum anderen mit der Art des *Texture-Mappings* zusammen, worauf aber im 3. Kapitel, »Workflow des *Projection-Mappings*« noch genauer eingegangen wird. Wichtig ist an dieser Stelle zu wissen, dass die Spieleumgebung, also die digitale Bühne, wie eine echte Theaterbühne nur grob aus einer Richtung betrachtet werden kann. Je weiter sich die Kamera von ihrer Ursprungsposition entfernt, desto eher werden Projektionsfehler oder Löcher im Hintergrund sichtbar. Wer versucht um die Spieleumgebung herumzugehen wird feststellen, dass die Hintergrundobjekte keine Rückseiten besitzen und die Ränder des Hintergrundes schnell erreicht sind. Es gilt also die Kamerabewegungen so anzulegen, dass die Illusion einer glaubwürdigen Spielwelt nicht zerstört wird.

Diese Technik eignet sich besonders für Spiele, in denen der Spieler das Geschehen aus mehr oder weniger einer Perspektive betrachtet. Das können zum Beispiel Point-and-Click-Adventures sein, bei Spiel-Kamera von einem etwas entfernten Standort auf das Geschehen gerichtet ist. Aus dieser Perspektive steuert der Spieler eine der vielen Spielfiguren und folgt dem Spielgeschehen. In einigen Fällen werden auch Schuss-Gegenschuss für Konversationen oder close-up shots als dramaturgische Mittel eingesetzt, was allerdings beim Bau der Spieleumgebung berücksichtigt werden muss.

Ein ebenfalls schon relativ altes Computerspiel-Genre ist das *2D-Jump'n'Run*, welches auf Smartphones und Tablets aber auch anderen Plattformen gerade eine Renaissance erlebt. Titel wie *Trine*, *Limbo* oder *Rayman Origins* zeichnen sich meist durch eine hübsche Grafik und einen interessanten Stil aus und kämen auch für eine Technik, wie das Camera-*Projection-Mapping* in Frage.

Weiterhin wären *Beat 'em ups* mit Vertretern wie der *Tekken-Serie* oder der *Street Fighter-Serie* geeignete Kandidaten, denn auch hier blickt der Spieler aus einiger Distanz auf das Spielgeschehen und die Perspektive ändert sich ebenfalls nur geringfügig. Neben einigen Einsatzmöglichkeiten gibt es auch Genres, in denen es nicht möglich ist, das Camera-*Projection-Mapping* sinnvoll einzusetzen. Hierzu zählen vor allem Spiele, in denen der Spieler seine Spielfigur aus der Ich-Perspektive (Egoperspektive) oder der *Third-Person-Perspektive* steuert. Das gilt für *3D-Shooter*, Rollenspiele, Sportspiele und Simulationen.



Abb. 2.3: Spieleumgebung aus *The Book of Unwritten Tales* von KING Art Games.

3. Workflow des *Projections-Mappings*

Das 4. Kapitel thematisiert die Arbeitsschritte, die zur Erstellung einer *Low-Polygon-Szene* erforderlich sind. Das Themengebiet gliedert sich dabei in vier Unterpunkte, die sich während der Entwicklung des Workflows herauskristallisiert haben und die alle gleichermaßen für das Funktionieren einer Spieleumgebung erforderlich sind.

Im ersten Arbeitsschritt wird die Erstellung einer *High-Polygon-Szene* und dessen Unterteilung in verschiedene Ebenen (*layering*) betrachtet. Die Modellierung, Texturierung und Beleuchtung der Szene wurde im Verlauf der Arbeit von externen 3D-Grafikern ausgeführt. Eine gute Koordinierung zwischen den externen und den betriebsinternen Mitarbeitern war während dessen essentiell, da die *High-Polygon-Szene* unseren Bedürfnissen angepasst werden musste und wir während der Erstellung auch die ebenenweise Sortierung ausführten. Auf Grundlage dessen wurde auch die *Low-Polygon-Geometry* erstellt, welche im zweiten Unterpunkt des Kapitels beschrieben wird. Gezeigt werden Möglichkeiten der Modellierung von *Low-Polygon-Objekten*, die stellvertretend für die *High-Polygon-Geometry* in das Spiel integriert werden.

Weiterhin wird das Rendern der Texturen aus der modellierten, texturierten und beleuchteten, sowie in Ebenen aufgeteilten *High-Polygon-Szene* betrachtet. Dazu wird erklärt, welche Texturtypen für eine, mit Hilfe des *Projection-Mappings* erstellte Spieleumgebung benötigt werden und was beim Rendern zu beachten ist.

Ferner sollen die gerenderten Texturen auf die *Low-Polygon-Objekte* aufgetragen, also *gemappt* werden. Hierfür wird die namensgebende *Mapping-Methode*, das *Projection-Mapping* und die dazu erforderliche Kameraprojektion thematisiert. Mit dessen Hilfe werden die Texturen passgenau auf die *Low-Polygon-Objekte* projiziert und erzeugen somit das detaillierte Erscheinungsbild der Spieleumgebungen.

3. Workflow des *Projection-Mappings*

3.1. Bau und ebenenweise Sortierung der *High-Polygon-Geometry*

Für den Bau einer Spieleumgebung mit Hilfe des *Projection-Mappings*, muss zu aller erst eine wichtige Grundlage geschaffen werden, ohne die der gesamte Workflow nicht funktioniert. Die Rede ist von einer *High-Polygon-Version* der Szene, die in das Spiel eingebaut werden soll. Diese *High-Polygon-Szene* ist nicht nur für die Erstellung der *Low-Polygon-Geometry* essenziell, sondern dient auch als Referenz zur Animation von Charakteren und Objekten, sowie Kamerafahrten. Außerdem werden aus dieser texturierten und beleuchteten Szene die *Maps* gerendert, welche später auf die *Low-Polygon-Geometry* projiziert werden, um dieser Farbe, Licht und Struktur zuzuweisen.

Beim Erstellen der *High-Polygon-Szene* dienen zur Orientierung Konzeptzeichnungen, in denen die gesamte Spieleumgebung abgebildet ist. Es sind alle Objekte zumindest grob eingezeichnet, wichtige Interaktionsobjekte sind zusätzlich gekennzeichnet und namentlich genannt. Wichtig ist auch die angestrebte Lichtstimmung bzw. die grundsätzliche Farbgebung. Auf Grundlage des so erstellten Konzeptes wird die *High-Polygon-Szene* mit Hilfe eines 3D-Computergrafikprogrammes modelliert, texturiert, beleuchtet und gerendert.

Der erste Schritt, der zum eigentlichen Arbeitsablauf des *Projection-Mapping* zählt ist die ebenenweise Sortierung der *High-Polygon-Geometry* auch *layering* genannt. Die *High-Polygon-Szene* muss für diesen Arbeitsschritt final modelliert sein. Eine Texturierung oder gar Beleuchtung der Szene ist dagegen noch nicht nötig. Wenn nun das *Layering* vorher durchgeführt wird, dann muss auch anhand der *gelayerten* Datei weiter texturiert und beleuchtet werden. Es dürfen also keine Arbeiten parallel an zwei Dateien durchgeführt werden, da sonst zwangsläufig ein Arbeitsgang wiederholt werden muss, was unnötig Zeit kostet. Alternativ kann das *Layern* nach Texturierung und Beleuchtung durchgeführt werden.



Abb. 3.1: Konzeptzeichnung einer Spieleumgebung.



Abb. 3.2: Rendering der *High-Polygon-Geometry*.

3. Workflow des *Projection-Mappings*

Mit Durchführung der ebenenweisen Sortierung wird also eine komplette Szene Objekt für Objekt neu gruppiert. Eine final gesetzte Kamera, welche die gesamte Szene überblickt und später auch als Renderkamera eingesetzt werden kann, ist bei diesem Arbeitsschritt sehr hilfreich. Aus Sicht dieser Kamera werden nun also alle Objekte, die von keinem anderen Objekt verdeckt werden und sich auch gegenseitig nicht berühren, in eine gemeinsame Ebene verschoben. Bei den Programmen Maya und 3ds Max von Autodesk bspw. bieten sich dafür die *layer* an, da diese sich einblenden, ausblenden und einfrieren lassen. Das wird bei den folgenden Arbeitsschritten wichtig und erleichtert das weitere Sortieren. Wenn also alle vordersten Objekte in die erste Ebene einsortiert sind, kann diese ausgeblendet werden. Nun werden neue Objekte freigelegt, die vorher ganz oder teilweise verdeckt waren. Die wiederum füllen die zweite Ebene auf. Dieser Schritt wird so lange wiederholt, bis alle Objekte einer Szene auf erfahrungsgemäß 10 bis 15 Ebenen verteilt sind. Aber warum ist eine solch aufwändige Sortierung überhaupt notwendig?

Wenn keine Trennung der Objekte in verschiedene Ebenen stattfindet, kann die Szene nur als *ein* gesamtes Bild gerendert werden. Von Objekten, die durch andere verdeckt werden, können so auch nur sichtbare Teile übertragen werden. Unsichtbare Teile, die aber bei einer Kamerafahrt sichtbar werden können gehen verloren. Auch die Positionen der Objekte im Raum sind so nicht mehr vorhanden und eine durch Kamerabewegungen hervorgerufene Parallaxenverschiebung kann nicht mehr entstehen.

Eine weitere Fragestellung bezieht sich darauf, warum Objekte auf einer Ebene sich untereinander nicht berühren sollten. Wenn das passiert, kann das zu sehr unschönen Dopplungen oder Verzerrungen kommen. Der Grund dafür ist, dass Objekte obwohl auf der gleichen Ebene befindlich, trotzdem eine ganz unterschiedliche Position in der Tiefe haben können, sich also weit von einander entfernt befinden. Kommt es zu einer seitlichen Kamerafahrt, entsteht eine parallaktische Verschiebung der Objekte und der Fehler wird sichtbar. Wie in Abbildung 3.3 zu sehen ist, sind zwei Objekte hintereinander platziert. Trotz der räumlichen Distanz, zu erkennen an dem Größenverhältnis zwischen Krug und Tisch, beziehen beide Objekte ihre Texturinformationen aus ein- und derselben Ebene, wodurch auch beide Objekte die Bildinformationen des jeweils anderen Objektes enthalten.

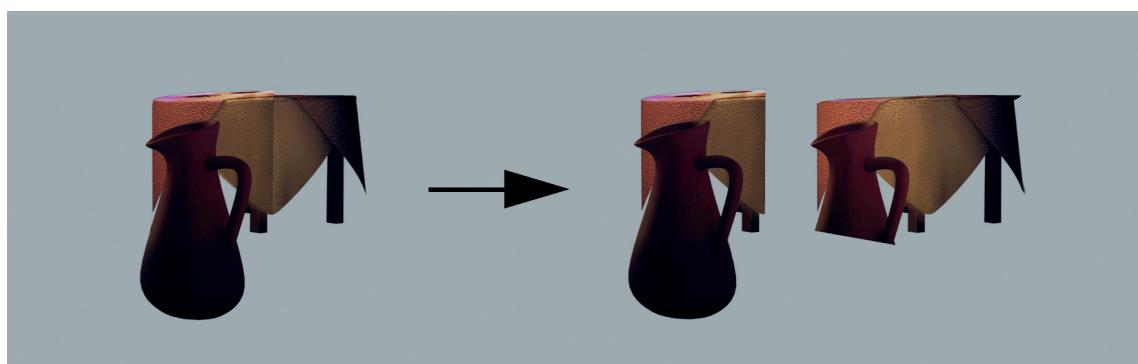


Abb. 3.3: Doppelprojektion wird durch eine Kamerabewegung sichtbar.

3. Workflow des *Projection-Mappings*

Auch ein wichtiger Aspekt, auf den ebenfalls schon eingegangen wurde, ist die Reihenfolge, in der die Objekte sortiert werden, also von vorne nach hinten. Grundsätzlich hat die Reihenfolge auf die Funktion zwar keine Auswirkung, allerdings ist bei Nichtbeachtung eine Retusche der Renderdaten in Photoshop schwierig. Um die Retusche, auch *overpainting* genannt, vorzunehmen, müssen zunächst alle gerenderten Ebenen übereinander gelegt werden. Wenn die Reihenfolge beachtet wurde, sieht das Bild im besten Fall genau so aus, als wäre die gesamte Szene in *einem* Vorgang gerendert worden. Wurde die Reihenfolge wiederum nicht beachtet, finden fehlerhafte Überlagerungen der Objekte statt und eine Nachbearbeitung in Photoshop wird sehr schwierig.



Abb. 3.4: Korrekt sortierte *layer* wurden ebenenweise gerendert und in Photoshop zusammengesetzt.



Abb. 3.5: Falsch sortierte *layer* wurden ebenenweise gerendert und in Photoshop zusammengesetzt.

3. Workflow des *Projection-Mappings*

3.2. Bau der *Low-Polygon-Geometry*

Der nächste Schritt widmet sich der Erstellung der *Low-Polygon-Geometry*, auf welche die gerenderten Ebenen in einem späteren Arbeitsschritt projiziert werden. Ebenfalls handelt es sich um eben diese Geometrien, bestehend aus polygonalen Netzen, die später in eine Spiel-Engine importiert und dort auch mit Items, animierten Objekten und Charakteren »bestückt« werden.

Der Bau der Geometrie geschieht auf Grundlage des in Ebenen unterteilten *High-Polygon-Modells*. Das bedeutet, dass jede Ebene mit wenigen hundert Polygonen nachgebildet wird. Die Umsetzung kann auf verschiedene Weisen geschehen. Entweder durch eine neue Modellierung auf Basis von geometrischen Grundobjekten, den *polygon primitives*. Oder es werden die *High-Polygon-Modelle* dupliziert und automatisch oder von Hand in ein *Low-Polygon-Modell* überführt. Weiterhin können einfache *Planes* erzeugt und verwendet werden.

Wie schon zuvor erwähnt, geschieht die Modellierung der *Low-Polygon-Geometry* mit Hilfe von polygonalen Netzen (*meshes*), welche in der 3D-Computergrafik häufig anzutreffen sind. Polygonnetze, also Netze die sich aus einer Menge von miteinander verbundenen Polygonen zusammensetzen, bestehen zumeist aus Dreiecken und Vierecken. Es lassen somit einfache geometrische Figuren, wie Zylinder, Kugeln, Prismen und Polyeder wie Würfel, Tetraeder, Pyramiden etc. aber auch komplexere Strukturen erschaffen, um bspw. Modelle für Computerspiele oder Animationsfilme zu erstellen.⁴

In Autodesk Maya werden polygonale Flächen auch als *faces* bezeichnet und haben drei oder mehr begrenzende Kanten, die *edges* genannt werden. Die *edges* wiederum sind über Punkte, den sogenannten *vertices* (Plural) miteinander verbunden. Diese drei formgebenden Elemente können transformiert werden, um ein polygonales Netz zu deformieren. Genutzt werden dafür die verschiedenen Transform-Attribute, wie verschieben, rotieren oder skalieren.⁵

Zusätzlich können weitere Unterteilungen eingesetzt werden, um ein Polygonnetz feiner zu strukturieren. Es ist möglich *faces*, *edges* und *vertices* miteinander zu verbinden, aufzutrennen oder um jeweils neue Elemente zu erweitern (extrudieren). Wie so erstellte *Low-Polygon-Objekte* im Vergleich zu den *High-Polygon-Objekten* aussehen können, verdeutlichen die Abbildungen 3.7 und 3.8.

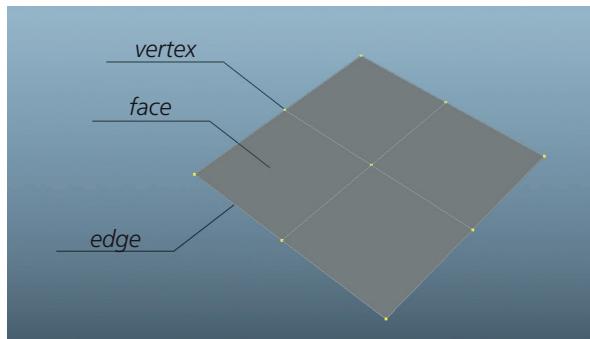


Abb. 3.6: Ein *mesh*, bestehend aus 4 quadratischen Polygonen.

4. vgl. Michael Bender | Manfred Brill (2006): *Computergrafik*. 2.Aufl. München: Carl Hanser Verlag. S. 191.

5. vgl. Todd Palmer (2014): *Mastering Autodesk Maya 2015*. 1.Aufl. Indianapolis: Sybex. S. 107 ff.

3. Workflow des *Projection-Mappings*

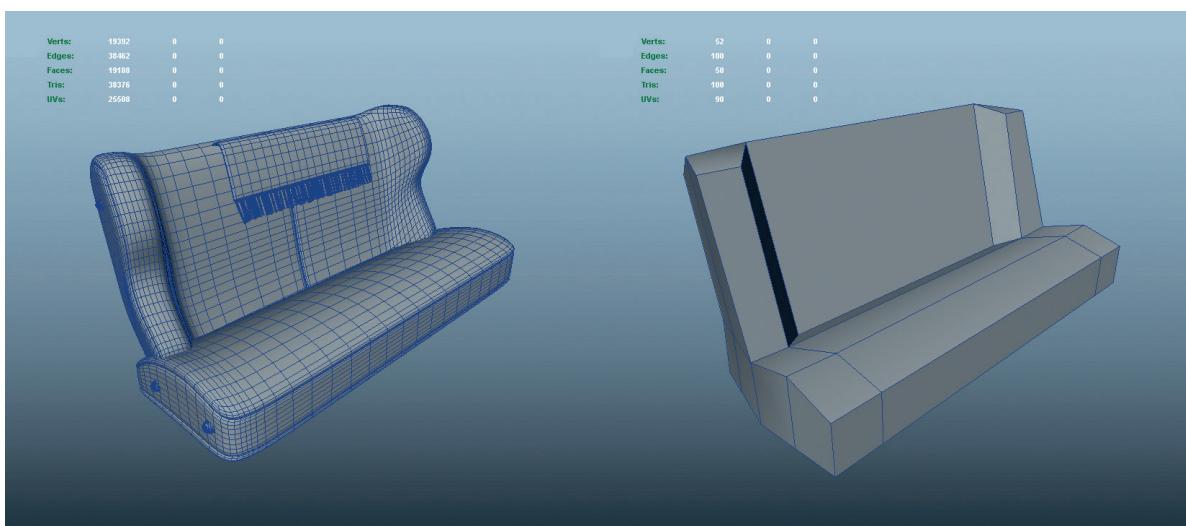


Abb. 3.7: Darstellung eines *High-Polygon-Models* (links), bestehend aus 19188 Polygonen und dem *Low-Polygon-Pendant* (rechts) mit nur 100 Polygonen, das aus einem Würfel modelliert wurde.

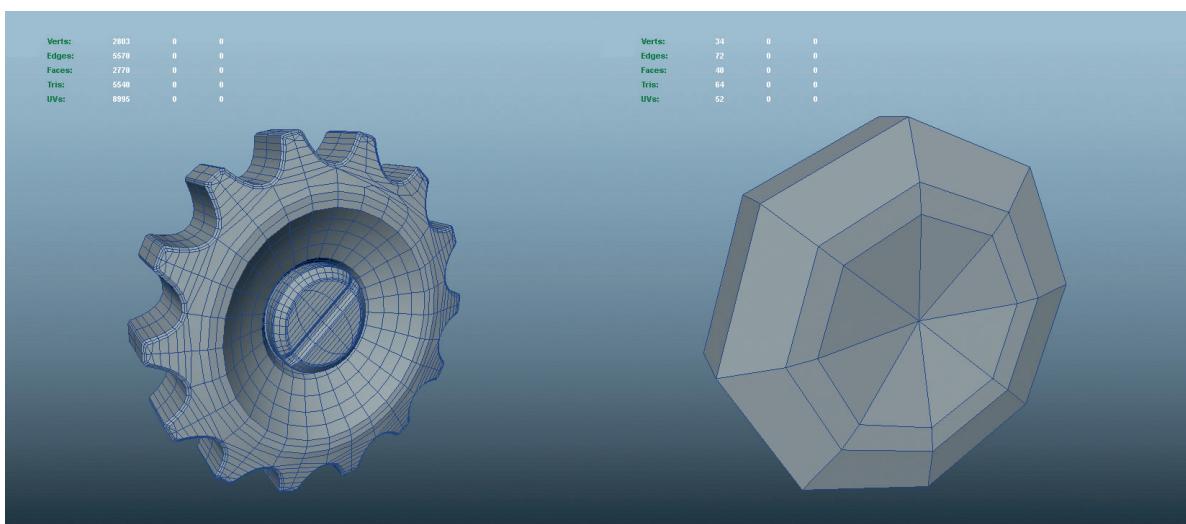


Abb. 3.8: Darstellung eines *High-Polygon-Models* (links), bestehend aus 2770 Polygonen und dem *Low-Polygon-Pendant* (rechts) mit nur 40 Polygonen, das aus einem Zylinder modelliert wurde.

3. Workflow des *Projection-Mappings*

Eine weitere Methode ist das schrittweise Reduzieren von Polygonen durch das händische Entfernen von *edge loops*, bei denen es sich laut dem Autodesk Knowledge Network um folgendes handelt. »An edge loop is a path of polygon edges that are connected in sequence by their shared vertices.«⁶ Bei guter Modellierung eines Objekts, also der Beschränkung auf rechteckige Polygone, lassen sich *edge loops* schnell markieren und löschen. Wie in der Abbildung 3.9 zu erkennen, wird jede zweite *edge loop* horizontal und vertikal gelöscht, um so das *mesh* Schritt für Schritt auszudünnen.

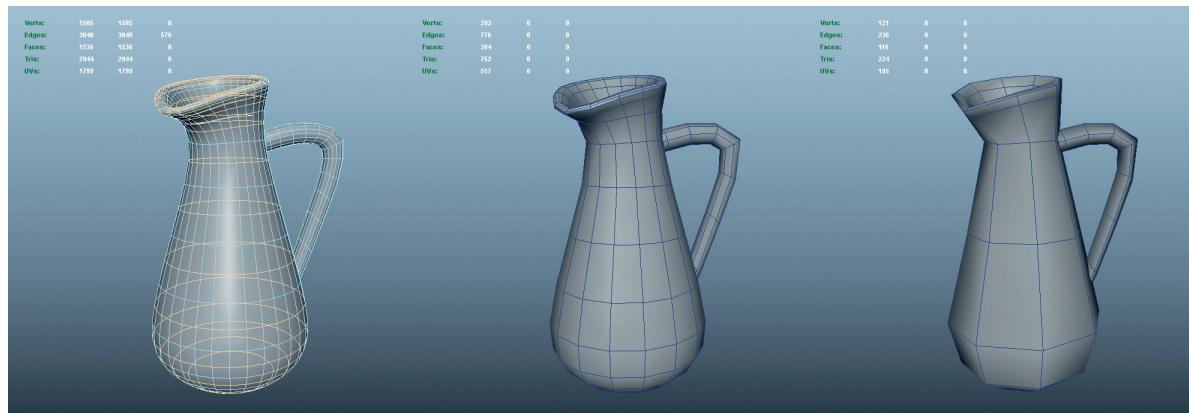


Abb. 3.9: Händische Reduktion der *edge loops*, die Anzahl der Polygone beträgt (v.l.n.r.) 1536, 384 und 116.

Eine extrem schnelle Methode zum Ausdünnen von Geometrien ist die Nutzung eines automatischen Algorithmus, über den moderne 3D-Programme verfügen. Mit den dazugehörigen Einstellungen kann der Vorgang zwar optimiert werden, ein Vergleich der Abbildungen 3.9 und 3.10 verdeutlicht aber die Nachteile der automatischen Routine. Das Polygongitter wird unsauber reduziert. Trotzdem sind die Algorithmen in der Lage die Silhouette eines Objekts im Groben beizubehalten.⁷

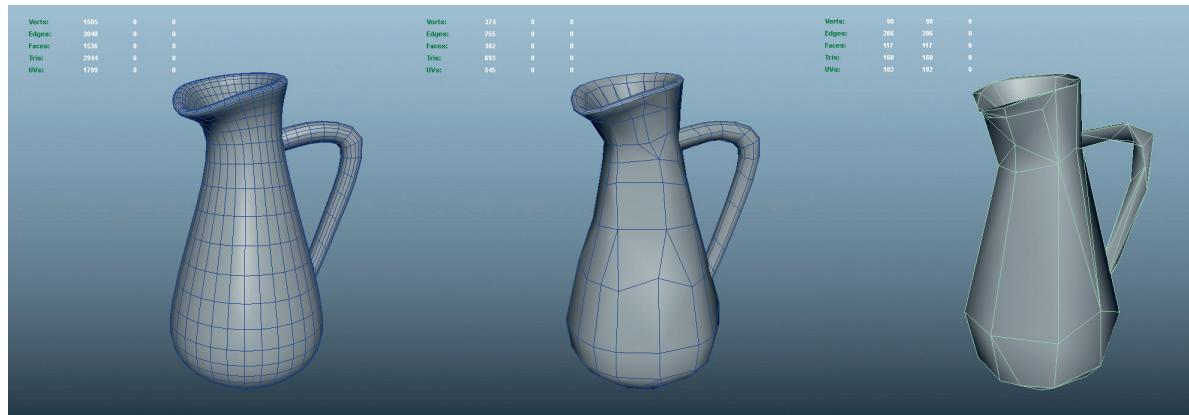


Abb. 3.10: Automatische Reduktion der Polygone auf (v.l.n.r.) 1536, 382 und 117.

6. http://knowledge.autodesk.com/support/maya/learn-explore/caas/mne-help/global/docs/maya2014/en_us/files/Polygon-selection-and-creation-Edge-ring-and-edge-loop-selection-tips-htm.html (abgerufen am 18.02.2015)

7. vgl. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*. 3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 459.

3. Workflow des *Projection-Mappings*

In der Literatur wird die Erstellung mehrerer Detailstufen eines Modells als *Level Of Detail (LOD)* bezeichnet und dient, durch die Einsparung von Polygonen vor allem zur Entlastung der Grafikhardware. Allerdings liegt der Fokus dort hauptsächlich auf den automatischen Reduzierungsverfahren. Trotzdem zielen alle vorher genannten Verfahren darauf ab, eine mit wenigen Polygonen ausgestattete Silhouette des *High-Polygon-Objektes* zu erzeugen.⁸

Die einfachsten Vertreter der *Low-Polygon-Geometry* sind *Planes* die sich nur grob, anhand der Skalierung am *High-Polygon-Objekt* orientieren. Um die Funktion zu erläutern ist ein kleiner Vorgriff nötig, der erst in »3.3. *Rendering* von *Diffuse-, Alpha-und Light-Maps*« vollständig thematisiert wird. Die Rede ist von transparenten Texturen, welche es ermöglichen, ein komplexes Objekt mit *einem einzigen Polygon* darzustellen. Genau genommen wird die, auch *cutout* genannte Technik für alle *Low-Polygon-Objekte* verwendet. Aber besonders bei der Verwendung von *planes* ist die Technik nötig, da das Objekt sonst seine gesamte Form einbüßt.

Ferner bringt die Nutzung von transparenten Planes weitere Nachteile mit sich, die berücksichtigt werden müssen. Ein besonderes Problem ist die Tatsache, dass das Objekt, in diesem Fall der Eimer, nicht mehr für Animationen zur Verfügung steht. Es ist bspw. nicht möglich Items oder kleinere Charaktere darin zu verstecken, da das Objekt keine Tiefe aufweist.

Wie in Abbildung 3.12 dargestellt, können Kamerabewegungen die Planarität des Objektes preisgeben. Oft ist daher eine, dem Objekt angepasste Bauweise ratsam (siehe Abb. 3.13).

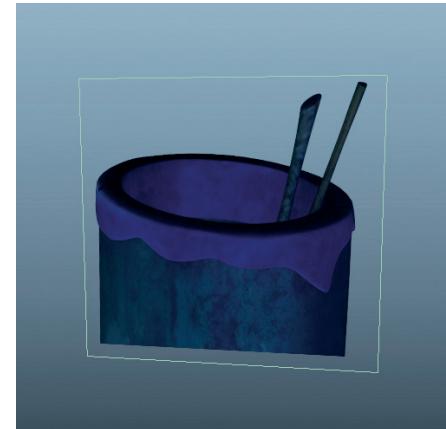


Abb. 3.11: Verwendung einer transparenten Textur auf einer *polygon-plane*.

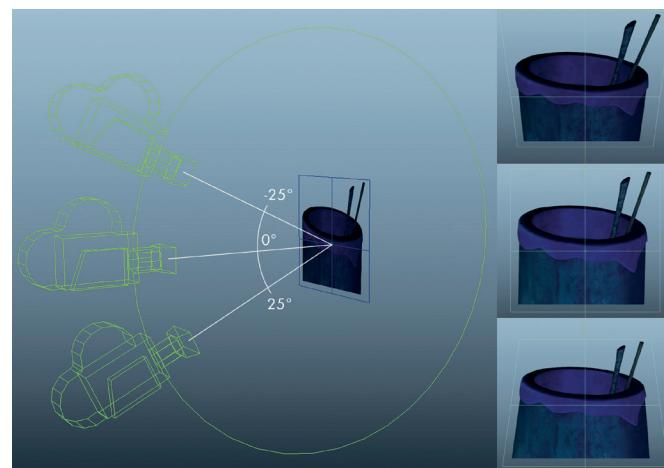


Abb. 3.12: Kamerabewegung entblößt die planare Architektur.

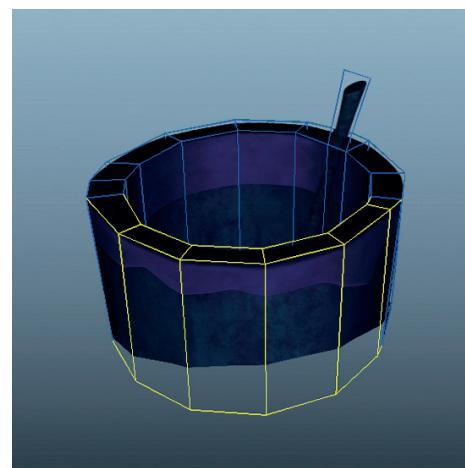


Abb. 3.13: Eine bessere Bauweise, bei der Vorder- und Rückseite getrennt gelagert wurden.

8. vgl. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*.

3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 458 - 459.

3. Workflow des *Projection-Mappings*

3.3. Rendering von *Diffuse-, Alpha- und Light-Maps*

Der nächste Part ist das Rendern der Ebenen. Prinzipiell soll an dieser Stelle nicht der Rendervorgang an sich thematisiert werden, sondern alles was bezüglich des Worksflows wichtig ist. Die Wahl der *renderengine* oder die Einstellungsmöglichkeiten in den *render settings* sind demnach nicht Teil der Betrachtung. Relevant sind dagegen eher die Vorbereitung auf das Rendern und die Ergebnisse, die der Rendervorgang liefert.

Der erste wichtige Punkt ist die Erstellung einer Renderkamera. Diese wird meist schon zu Beginn der Modellierungsphase gesetzt und deckt die gesamte, zu rendernde Szene ab. Alles was darüber hinaus an Objekten platziert wird, kann beim Rendern nicht mehr berücksichtigt werden und ist auch später nicht mehr zu sehen. Alle animierten Kameras, die dann auch im Projekt gesetzt werden, besitzen ein deutlich kleineres Sichtfeld und sind auch näher am Geschehen, um Kamerafahren zu ermöglichen. Die Renderkamera spielt weiterhin in Kapitel »3.4. Mapping der Low-Polygon-Geometry« eine wichtige Rolle und muss deswegen gespeichert und gegen versehentliches Verschieben gesichert werden.

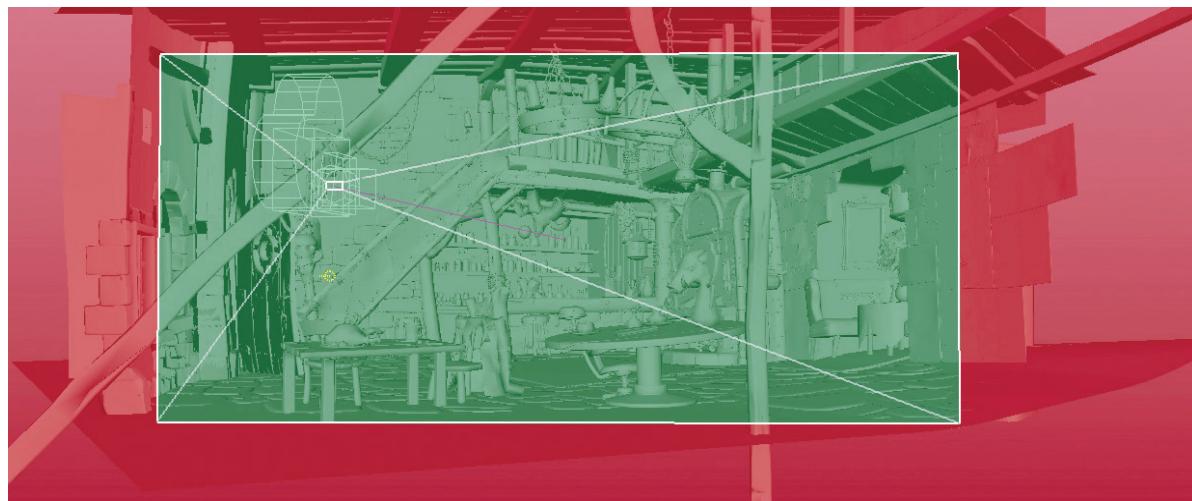


Abb. 3.14: Gesetzte Renderkamera mit Sichtkegel, sowie den zu rendernden Bereichen (grün) und den abgeschnittenen Bereichen (rot).

Ein weiterer großer Part war die Sortierung der *High-Polygon-Szene*, siehe »3.1. Bau und ebenenweise Sortierung der High-Polygon-Geometry«. Diese muss nun spätestens an dieser Stelle vollständig modelliert, texturiert, beleuchtet und gelayert sein, denn die Ergebnisse dieses *renderings* dienen als Texturen für die *Low-Polygon-Szene* und haben mehr als alles andere Einfluss auf das spätere Aussehen der Szene. Ebene für Ebene wird die *High-Polygon-Szene* nun gerendert. Dazu werden alle bis auf die aktuelle Ebene ausgeblendet, um eben nur eine zur Zeit zu rendern. Ein Problem ergibt sich aus dieser Aufteilung, das vor dem Rendern berücksichtigt werden muss. Ausgeblendete Objekte werfen keinen Schatten. Das Ergebnis soll in einem Testaufbau verdeutlicht werden (siehe Abbildung 3.15 bis 3.18).

3. Workflow des *Projection-Mappings*

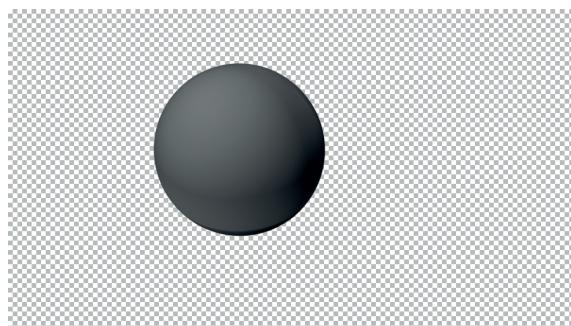


Abb. 3.15: *Rendering* der ersten Ebene.

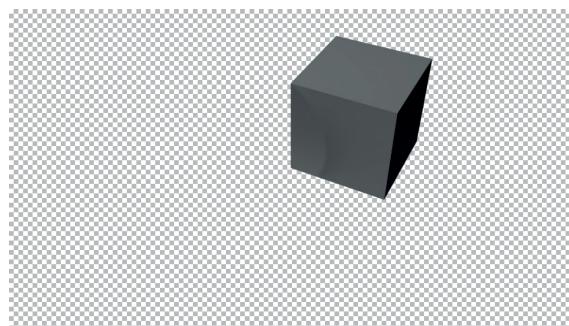


Abb. 3.16: *Rendering* der zweiten Ebene.

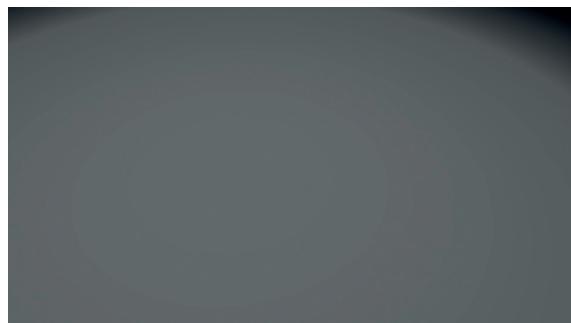


Abb. 3.17: *Rendering* der dritten Ebene.

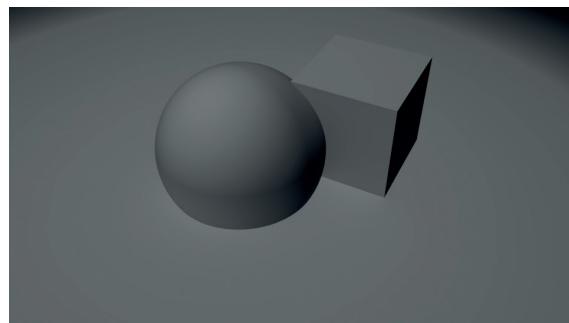


Abb. 3.18: Anordnung der Ebenen in Photoshop.

Das Ergebnis, zu sehen in Abbildung 3.18 wirkt nicht besonders glaubwürdig, da jeglicher Schattenwurf fehlt. Besonders aber für 3D-Szenen »sind Schatten eine wichtige Informationsquelle. So liefern Schatten wichtige Hinweise auf räumliche Anordnung von Objekten in einem Szenario, auf die Form von Schattenspendern (sogenannte „*blocker*“) und Schattenempfänger (sogenannte „*receiver*“) und das zu einem gewissen Grad auch in Bereichen, die vom Augenpunkt nicht direkt sichtbar sind.«⁹ So kann die Position der Kugel und des Würfels in Bezug auf den Boden nicht bestimmt werden.

Um dieses Problem zu lösen, ist es nötig zu verstehen, welches Objekt ein *blocker*, und welches ein *receiver* ist. Die meisten Objekte sind sowohl *blocker* als auch *receiver*. Die Kugel und der Würfel sind bspw. beides, da diese Objekte sich gegenseitig schattieren, also einen Schatten werfen und die Schatten des jeweils anderen empfangen. Der Boden ist in diesem Fall ein *receiver*, da dieser zwar die Schatten des Würfels und der Kugel empfängt, aber selber keinen Schatten auf andere Objekte wirft. In den 3D-Programmen Maya und 3ds Max gibt es die Möglichkeit einzustellen, was von einem Objekt gerendert werden soll und was nicht. Es ist möglich die Sichtbarkeit eines Objektes zu deaktivieren und trotzdem den Schatten zu rendern. Die Optionen *Cast Shadows* und *Receive Shadows* ermöglichen so individuell jedes Objekt als *blocker* oder *receiver* oder beides zu definieren. Deutlich wird das anhand der Abbildungen 3.19 bis 3.22.

9. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*. 3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 363.

3. Workflow des *Projection-Mappings*

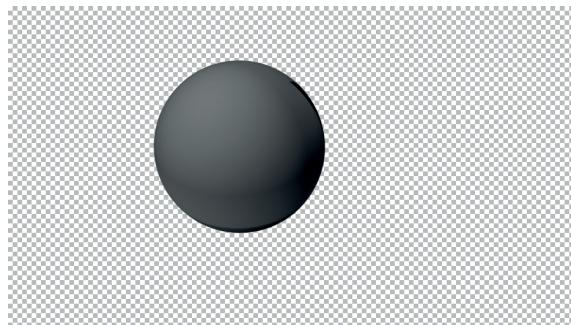


Abb. 3.19: Kugel empfängt Schatten des Würfels.

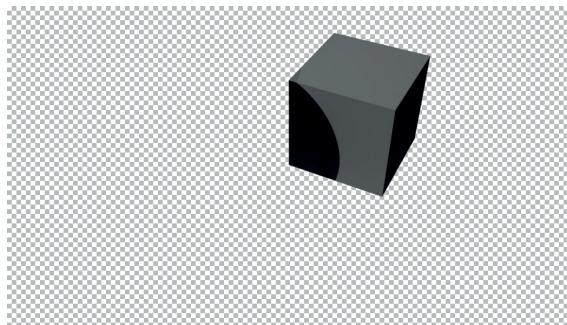


Abb. 3.20: Würfel empfängt Schatten der Kugel.

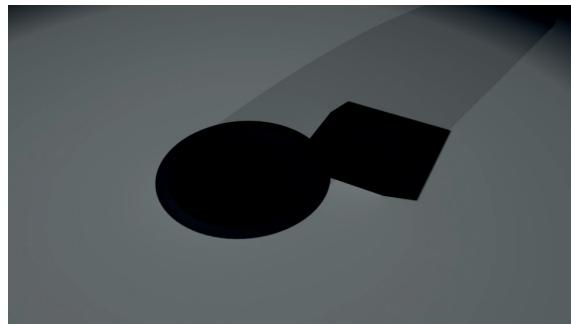


Abb. 3.21: Boden empfängt Schatten des Würfels und der Kugel.

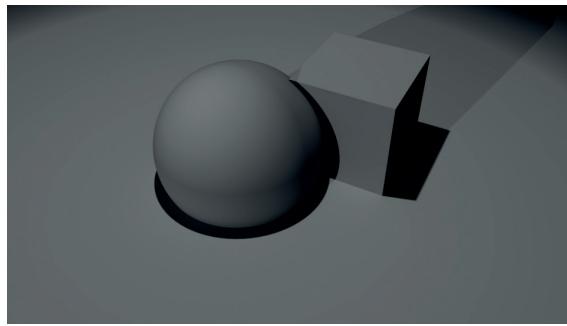


Abb. 3.22: Anordnung der Ebenen in Photoshop.

Nachdem nun die wichtigen Punkte bzgl. der Vorbereitung thematisiert wurden, müssen alle Texturen Ebene für Ebene gerendert werden. Für *The Book of Unwritten Tales 2* wurden insgesamt drei Arten von Texturen pro Ebene gerendert. Diese waren die *Diffuse-Maps*, *Alpha-Maps* und *Light-Maps*. Jede dieser gerenderten Texturtypen hat eine besondere Funktion und wird für verschiedene Aufgaben verwendet. Diese werden auf den folgenden Seiten angesprochen.

3. Workflow des *Projection-Mappings*

Der erste Texturtyp ist die *Diffuse-Map*. Dieser legt die grundlegende Farbgebung eines Objektes fest und ist maßgeblich für deren Erscheinungsbild verantwortlich. So wie hier abgebildet werden die *Texture-Maps* auch auf der *Low-Polygon-Geometry* projiziert aussehen.



Abb. 3.23: Insgesamt 15 gerenderte *Diffuse-Maps*.

3. Workflow des *Projection-Mappings*

Der zweite Texturtyp ist die *Alpha-Map*. Diese, häufig im Alphakanal einer Bilddatei gespeicherte Map legt die Transparenz eines Objektes fest. Weiße Bereiche werden opak, schwarze Bereiche vollständig transparent dargestellt. Mithilfe von Graustufen lassen sich auch halbtransparente Bereiche realisieren. Unter Zuhilfenahme von *Alpha-Maps* lassen sich Konturen pixelgenau freistellen, ohne dass die Polygonnetze einen besonders hohen Detailgrad aufweisen müssen.¹⁰



Abb. 3.24: Eine Auswahl an (insgesamt 15) gerenderten *Alpha-Maps*.

10. vgl. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*. 3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 187.

3. Workflow des *Projection-Mappings*

Bei dem dritten Texturtyp handelt es sich um sogenannte *Light-Maps*. Diese Art von *Texture-Map* wird mit einer anderen Beleuchtung als die *Diffuse-Map* gerendert und dient dazu Licht und Schatten zu verstärken. Die *Light-Maps* werden durch das Setzen von Lichtquellen aktiviert und je nach Intensität der Lichtquellen auch entsprechend stark in die *Diffuse-Map* hinein gemischt. Durch das langsame Ein- und Ausblenden der *Light-Maps* lassen sich auch komplexe Beleuchtungssituationen wie das Flackern eines Feuers simulieren.

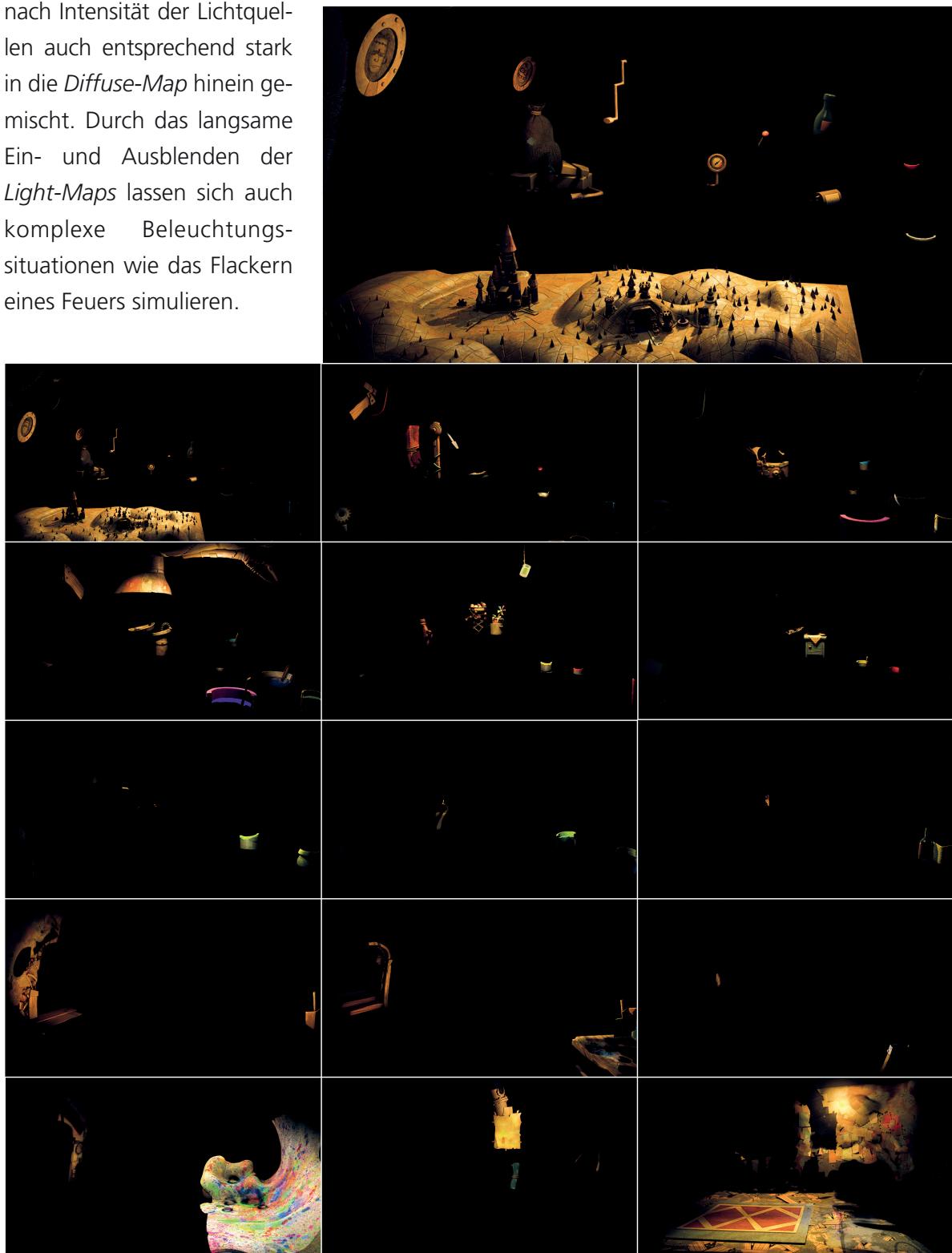


Abb. 3.25: Insgesamt 15 gerenderte *Light-Maps*.

3. Workflow des *Projection-Mappings*

Wie sich die gerenderten Texturtypen gegenseitig beeinflussen, soll die Abbildung 3.26 verdeutlichen. Der oberste Texturtyp, die *Diffuse-Map* (1) wird mit Hilfe der *Alpha-Map* (2) freigestellt und es ergibt sich eine *Diffuse-Map* mit opaken und transparenten Bereichen (3). Die *Light-Map* (4), die eine zweite Art der Beleuchtung ermöglicht, wird bei der Aktivierung von in Echtzeit berechneten Lichtquellen in die freigestellte *Diffuse-Map* gemischt (5).

Der Workflow des *Projection-Mapping* funktioniert genau genommen auch ohne den Einsatz von *Light-Maps*. Allerdings lässt sich mit deren Verwendung eine wesentlich dynamischere Umgebung erzeugen, die auch verschiedene Lichtstimmungen darstellen kann.



Abb. 3.26: Darstellung der Texturtypen und deren Funktion (v.o.n.u.): *Diffuse-Map*, *Alpha-Map*, *Diffuse-Map* (freigestellt), *Light-Map* und die Kombination aus *Light-Map* und freigestellter *Diffuse-Map*.

3. Workflow des *Projection-Mappings*

3.4. Mapping der *Low-Polygon-Geometry*

Nun, da die *Low-Polygon-Szene* erstellt und alle *Texture-Maps* gerendert sind, müssen diese zusammengeführt werden. Das bedeutet, dass die *Texture-Maps* auf die *Low-Polygon-Geometry* projiziert (*gemappt*) werden. Die Technik, die dahinter steht, nennt sich *Texture-Mapping* und geht auf Edwin Catmull und seine Arbeiten aus dem Jahre 1974 zurück.

Das Grundproblem stellt sich folgendermaßen dar. Es muss eine 2D-Bilddatei, die *Texture-Map* bzw. Textur auf die Oberfläche eines 3D-Objektes aufgetragen werden. Michael Bender und Manfred Brill, vergleichen das Auftragen einer *Texture-Map* auf ein Objekt, in ihrem Buch *Computergrafik* mit dem Tapezieren einer Wand, wobei die Tapete der *Texture-Map* entspricht.

Wenn nun also eine Objektoberfläche mit einer Textur überzogen werden soll, so ist es wichtig zu wissen, dass für jeden Objektpunkt (vertex) eine Texturkoordinate angegeben werden kann. Diese Texturkoordinaten werden im Textur-Raum, einem zweidimensionalen Koordinatensystem angeordnet und erstrecken sich üblicherweise über den Wertebereich $[0, 1]^2$. Eine eingebundene Textur wird genau in diesem, in Abbildung 3.27 dunkelgrau dargestellten Bereich aufgezogen und an den Stellen auf das Objekt projiziert, an dem die Texturkoordinaten positioniert wurden. Ein Überschreiten dieses Bereiches führt zur Wiederholung der Textur, was auch als Kachelung bezeichnet wird.¹¹

Die *Mapping-Methode*, die an dieser Stelle wichtig ist, gibt dem gesamten Arbeitsgang seinen Namen. Die Rede ist vom *Camera-Projection-Mapping*, das auch in Kurzform als *Projection-Mapping* bezeichnet wird. Der Name enthält auch schon die wichtigsten Informationen die für diesen Arbeitsschritt unerlässlich sind. Also die Festlegung der Texturkoordinaten, unter Zuhilfenahme einer Kamera-Projektion. Die hier zu wählende Kamera ist eben die Renderkamera, die schon in Kapitel »3.3. Rendering von *Diffuse-, Alpha- und Light-Maps*« genutzt wurde. Warum es nun genau diese und keine andere Kamera sein muss, soll anhand einer genauen Betrachtung geklärt werden.

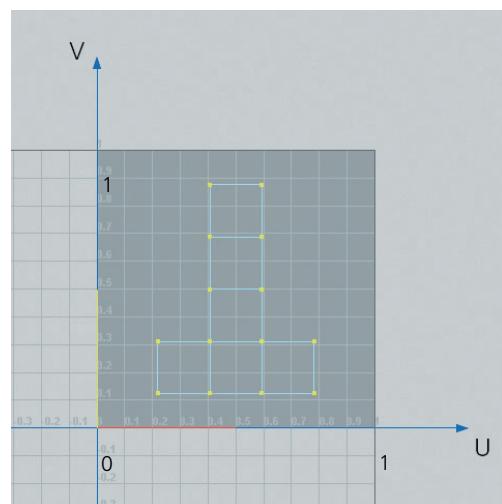


Abb. 3.27: Textur-Raum mit Koordinatensystem und den Texturkoordinaten eines Würfels.

11. vgl. Michael Bender | Manfred Brill (2006): *Computergrafik*. 2.Aufl. München: Carl Hanser Verlag. S. 298.

3. Workflow des *Projection-Mappings*

Generell wird zwischen zwei Arten von Kamera-Projektionen unterschieden. Zum einen wäre dies die orthografische Projektion, zum anderen die perspektivische Projektion.

Die orthografische Projektion, auch Parallel-Projektion genannt, bildet Objekte mit parallel verlaufenden Strahlen ab, weshalb Objekte ihre Größen und Winkel unabhängig von ihrer Entfernung zur Kamera nicht verändern. Diese Art der Projektion wird hauptsächlich in CAD-Programmen verwendet, um technische Zeichnungen in der Vorderansicht, Seitenansicht oder Draufsicht darzustellen.

Die perspektivische Kamera-Projektion, die auch als Zentral-Projektion bezeichnet wird, bildet Objekte mithilfe von konvergierenden Strahlen ab. Das bedeutet, dass die Strahlen pyramidenförmig im Augenpunkt zusammen laufen. Objekte, die sich näher an der Kamera befinden, werden daher größer abgebildet, als solche die weiter von der Kamera entfernt platziert wurden. Diese Art der Projektion entspricht dem des menschlichen Auges.¹² Ein Vergleich soll die Unterschiede veranschaulichen.

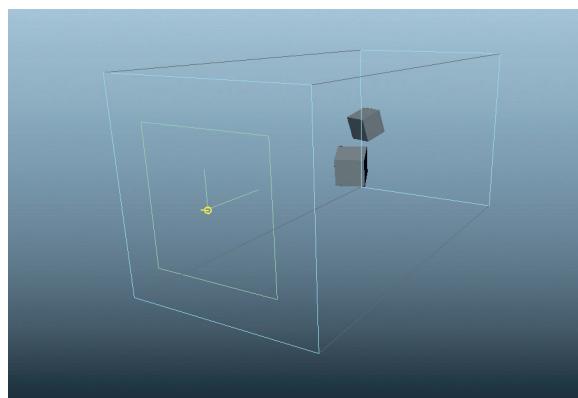


Abb. 3.28: Darstellung einer orthografischen Projektion.

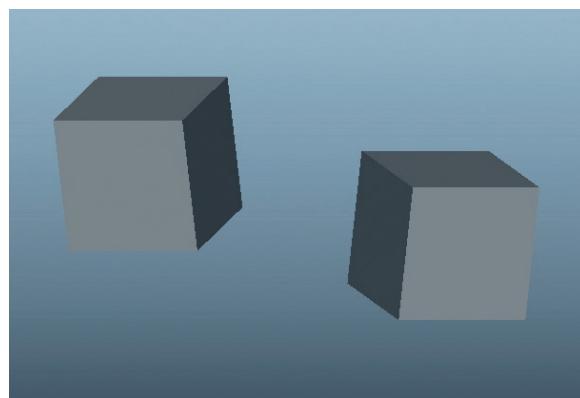


Abb. 3.29: Bildschirmsicht der orthografischen Projektion.

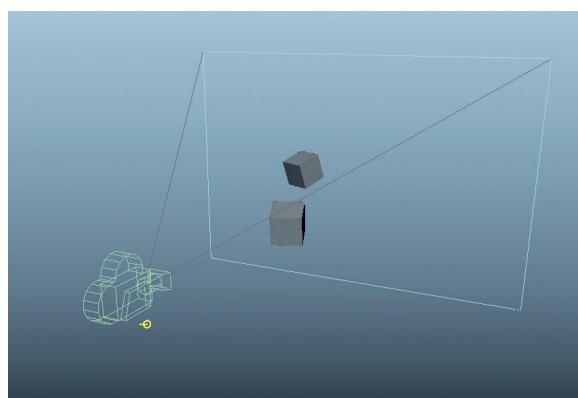


Abb. 3.30: Darstellung einer perspektivischen Projektion.

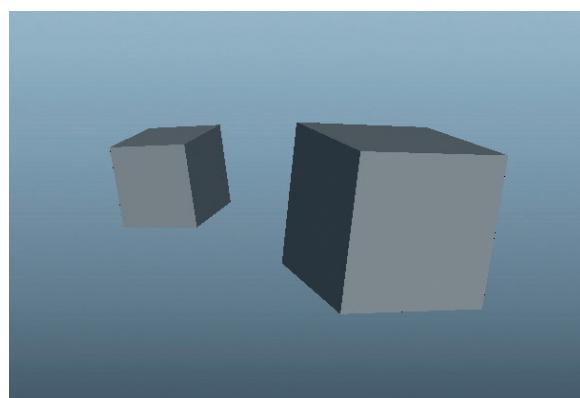


Abb. 3.31: Bildschirmsicht der perspektivischen Projektion.

12. vgl. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*.

3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 142 ff.

3. Workflow des *Projection-Mappings*

Die orthografische Projektion ist an dieser Stelle nicht weiter interessant und sollte nur als Vergleich dienen. Wichtig für das *Projection-Mapping* ist ausschließlich die perspektivische Kamera-Projektion, da physikalisch bedingt auch reale Kameras ein perspektivisches Sichtfeld aufweisen. Neben dem perspektivischen Sichtfeld, das durch den Pyramidenstumpf dargestellt wird, besitzt die künstliche Kamera noch weitere Attribute, die auch von einer realen Kamera bekannt sind.

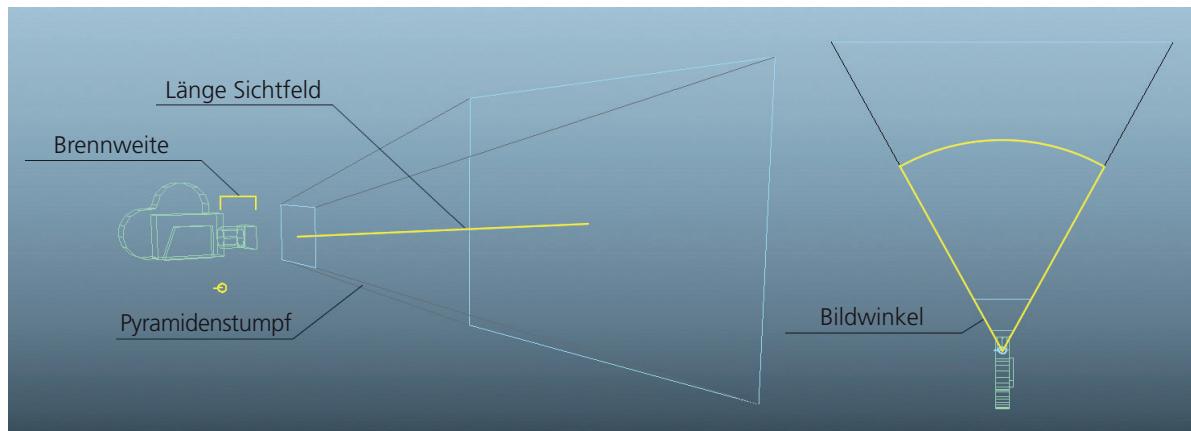


Abb. 3.32: Details einer Kamera in Autodesk Maya.

Wichtig für das *mapping* sind besonders die Brennweite und der daraus resultierende (in Abbildung 3.32 horizontale) Bildwinkel, denn anhand dieser Werte soll festgelegt werden, in welcher Größe die Textur auf die Geometrie projiziert wird. Um zu verdeutlichen wie genau sich das Sichtfeld auf das *mapping* auswirkt, wurde in einem Versuchsaufbau eine *polygon-plane* mit Hilfe einer Kameraprojektion in verschiedenen Brennweiten *gemappt*.

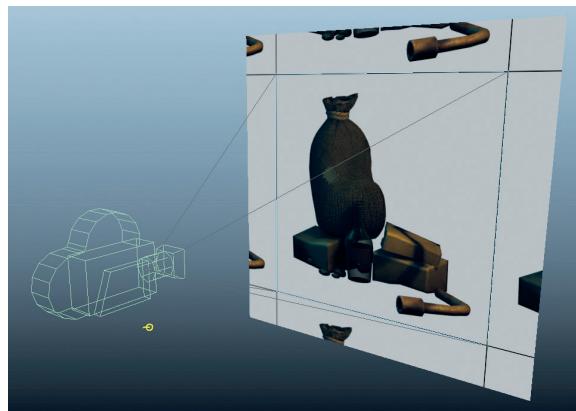


Abb. 3.33: Quadratische Projektion mit einer Brennweite von 30mm und einem Bildwinkel von $45,89^\circ$.

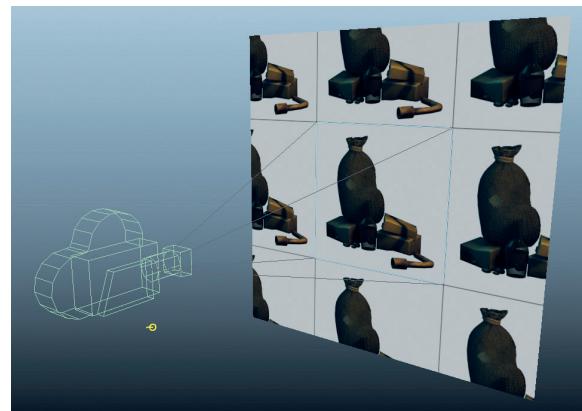


Abb. 3.34: Quadratische Projektion mit einer Brennweite von 50mm und einem Bildwinkel von $28,5^\circ$.

3. Workflow des *Projection-Mappings*

Das Ergebnis zeigt, dass die Textur in der Größe des auftreffenden Sichtfeldes der Kamera auf das Polygonnetz aufgetragen wird. Über das Sichtfeld hinaus wird die Textur gekachelt. Die eingestellte Brennweite und der von Maya errechnete Bildwinkel müssen also bei der Kamera, mit der die *Texture-Maps* gerendert wurden und der mit dem die *Low-Polygon-Geometry* gemappt wird übereinstimmen, da sonst, je nach Einstellung der Parameter, die Textur zu groß oder zu klein auf das Polygonnetz übertragen wird.

Auch muss natürlich die Positionen der Kameras im 3D-Raum exakt die gleiche sein, da sonst die Textur verschoben dargestellt wird.

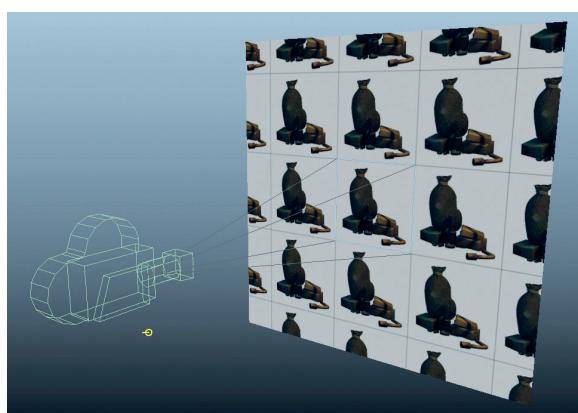


Abb. 3.35: Quadratische Projektion mit einer Brennweite von 85mm und einem Bildwinkel von 17°.

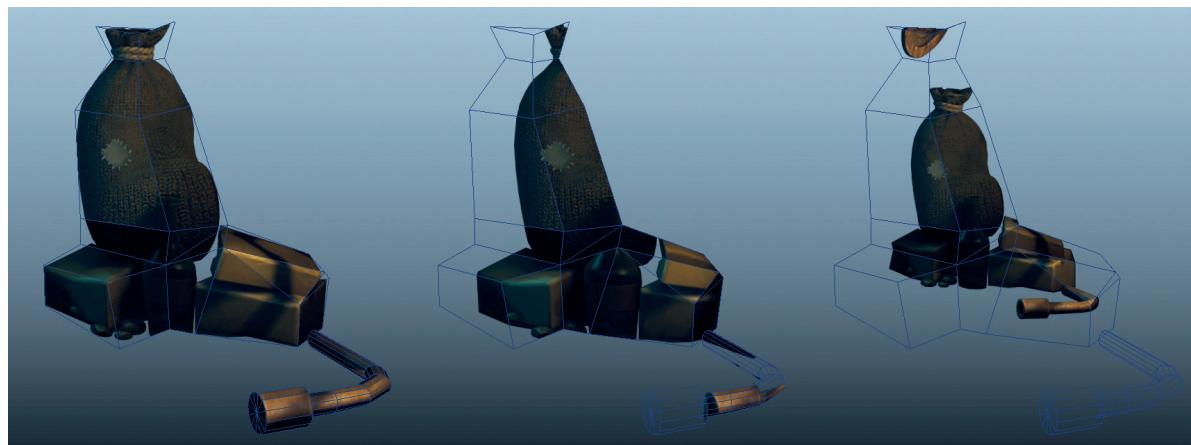


Abb. 3.36: Darstellung eines korrekten Mappings (links), eines mit verschobener Kamera (Mitte) und eines mit zu großer Brennweite der Kamera (rechts).

4. Problemstellungen und Lösungsansätze

Da es sich beim *Projection-Mapping* um einen experimentellen Workflow handelte, traten viele Probleme auf, welche die Entwicklung einer Lösung erforderten. Einige Fehler bezogen sich auf allgemeingültige Regeln beim Umgang mit polygonalen Netzen und Texturen, andere wiederum sind sehr spezifisch für den Workflow des *Projection-Mappings* und treten im Umgang mit der 3D-Computergrafik eher selten auf.

Einen sehr spezifischen Fehler stellten Texturverzerrungen dar, die durch perspektivisch verzerrte Texturkoordinaten hervorgerufen wurden und de facto in jeder der erstellten Spieleumgebungen auftraten. In Extremfällen waren die Verzerrungen so gravierend, dass in der Textur abgebildete Strukturen, Formen und Objekte nicht mehr zu identifizieren waren.

Ein gängiges Problem, dass sehr häufig bei der Modellierung mit polygonalen Netzen auftritt, ist die *Nonmanifold Geometry*. Diese gilt als Zusammenfassung von verbotenen Modellierungsweisen, die in der Weiterverarbeitung von Modellen zu Problemen führen können.

Unsichtbare Objekte, die durch das sogenannte *Backface Culling* ausgeblendet werden, traten ebenfalls hin- und wieder auf. Dieser Fehler wird bei falscher Ausrichtung der Polygone sichtbar und blendet Objekte, die eigentlich in der Szene befindlich sind, aus.

Zuletzt wird auf das Alphakanter Problem eingegangen. Dieses Problem geht auf die Verwendung von Transparenz zurück und erzeugt unschöne Farbränder an allen Objekten.

Für alle diese Problemstellungen wurden während der Entwicklung von *The Book of Unwritten Tales 2* Lösungen erarbeitet und erfolgreich umgesetzt.

4. Problemstellungen und Lösungsansätze

4.1. Texturverzerrungen

Nachdem nun die *Low-Polygon-Geometry* angelegt, anhand der perspektivischen Kameraprojektion *gemappt*, und mit der gerenderten Textur belegt wurde, wird nun folgendes Problem sichtbar. Die Textur wird extrem verzerrt auf das Objekt projiziert (siehe Abbildung 4.1).

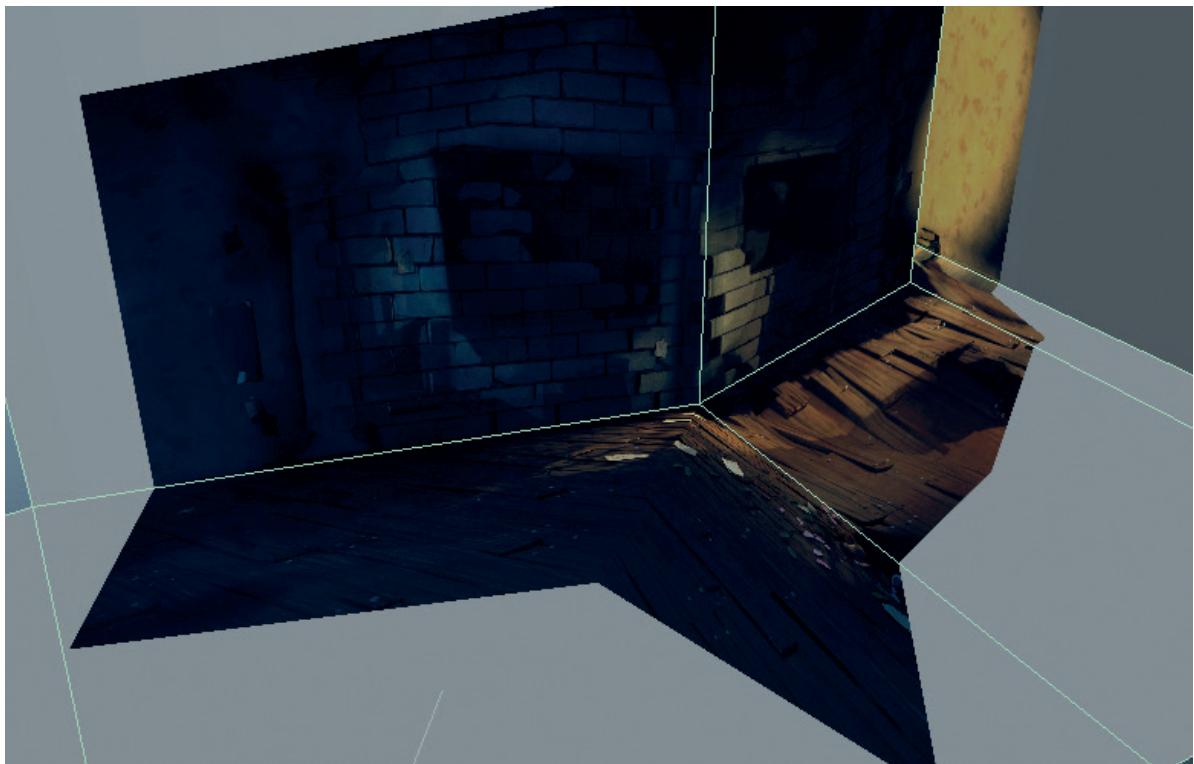


Abb. 4.1: Stark verzerrte Textur auf einer *Low-Polygon-Geometry*.

Doch wo genau liegt das Problem? Im Fachbuch *Computergrafik und Bildverarbeitung* wird das Texturieren eines Objekts verglichen mit einem Foto, dass auf eine beliebig dehbare Gummihaut gedruckt und dann mit Hilfe von Stecknadeln an den Eckpunkten des Polygone befestigt wird. Das bedeutet, dass allen *vertices* eines Objektes, Punkte der Textur als Texturkoordinaten zugewiesen und gleichmäßig verteilt werden. Die Texturkoordinaten werden linear interpoliert.¹³ Nun besteht die, in Abbildung 4.1 dargestellte Geometrie nur aus acht Polygone und beinhaltet daher auch nur sehr wenige *vertices* bzw. die dazugehörigen Texturkoordinaten. Diese Beschränkung der Polygone, wird dann zum Problem, wenn das *mapping* eine perspektivische Verzerrung aufweist. Dadurch, dass zum *mappen* eine perspektivische Projektion genutzt wurde, ist aber genau das der Fall.

13. vgl. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*. 3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 268.

4. Problemstellungen und Lösungsansätze

Die Lösung ist, die Geometrie so lange zu weiter zu unterteilen, bis die Verzerrung nicht mehr stört. Durch das Hinzufügen von neuen Texturkoordinaten können diese besser über die gesamte Fläche verteilt bzw. interpoliert werden.

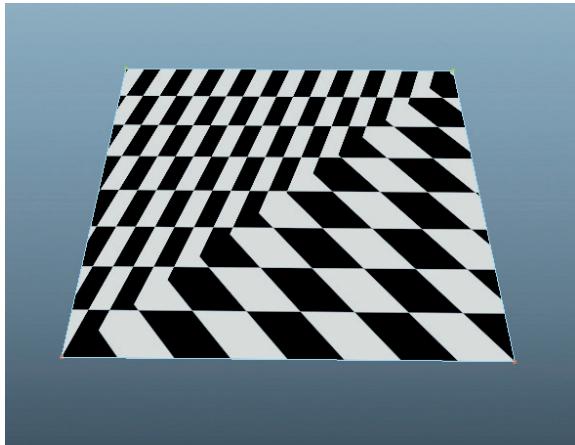


Abb. 4.2: *Plane bestehend aus einem Polygonen.*

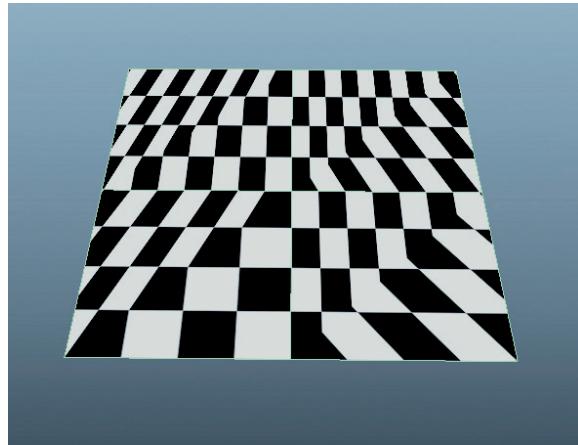


Abb. 4.3: *Plane bestehend aus vier Polygone.*

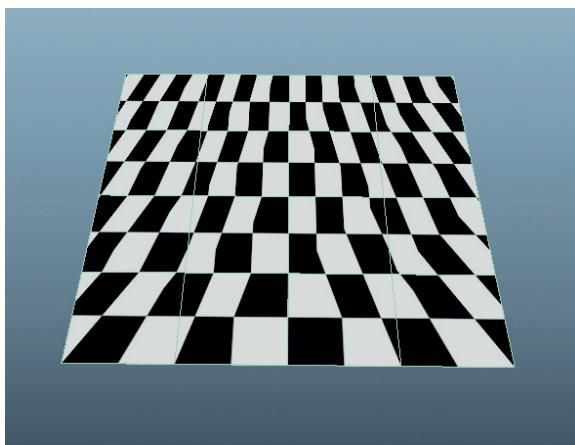


Abb. 4.4: *Plane bestehend aus 16 Polygone.*

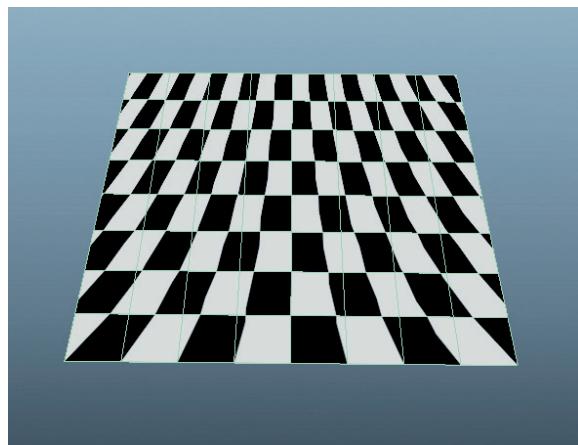


Abb. 4.5: *Plane bestehend aus 64 Polygone.*

Das bedeutet aber auch, dass eine Geometrie nur so oft unterteilt werden sollte, wie eine Verzerrung der Textur erkennbar ist. Alle darüber hinaus erzeugten Polygone sind unnötig und sogar nachteilig für die Performance eines Systems.

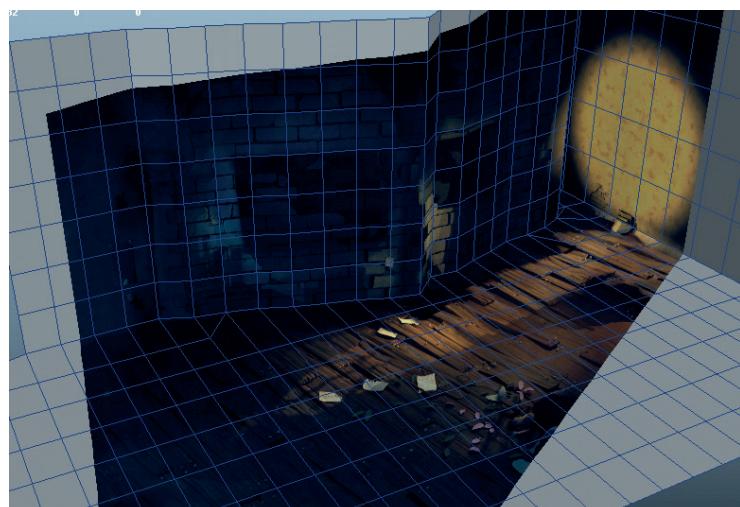


Abb. 4.6: Feinere Unterteilung der *Geometry* beseitigt die Verzerrung.

4. Problemstellungen und Lösungsansätze

4.2. Nonmanifold Geometry

Nonmanifold Geometry ist ein Sammelbegriff für verschiedenste fehlerhafte Modellierungsweisen und kann grundsätzlich immer bei der Modellierung mit polygonalen Netzen auftreten. Es ist also nicht auf den Workflow des *Projection-Mappings* beschränkt. Oft wird erst bei späteren Arbeitsschritten, wie bei der Texturierung oder beim Import in andere Programme festgestellt, dass ein Objekt *Nonmanifold Geometry* enthält. Besser ist es, wenn schon während der Modellierung des Objektes die fehlerhafte Geometrie auffällt, etwa weil Werkzeuge nicht mehr richtig funktionieren oder es zu Problemen beim Animieren oder Rendern kommt. Wichtig ist, dass *Nonmanifold Geometry* immer behoben oder am besten gleich vermieden wird. Anhand von Beispielen soll verdeutlicht werden, wobei es sich um *Nonmanifold Geometry* handelt und wie man diese verbessert bzw. beseitigt.¹⁴

Das *T-shape* ist ein Gebilde, bei dem sich drei oder mehr faces *eine edge* teilen. Besonders bei dieser Verbindung von Polygonen kann es zu Problemen mit einigen Modellierungswerkzeugen kommen. Es ist bspw. nicht mehr möglich *edge loops* vollständig einzusetzen oder die Flächen zu extrudieren, ohne dass kritische Überlappungen entstehen können. Unproblematisch wäre es hingegen, wenn das *T-shape* eine gewisse Flächenstärke aufweisen würde.

Beim *bow-tie shape* handelt es sich um zwei oder mehr faces, die über *einen* einzigen *vertex* miteinander verbunden sind, aber keine gemeinsame *edge* aufweisen. Es handelt sich ebenfalls um ein *bow-tie shape*, wenn mehrere dreidimensionale Körper, z.B. Würfel oder Pyramiden über *einen* Punkt miteinander verbunden sind. Egal in welcher Form, diese Konfiguration sollte vermieden werden.¹⁵

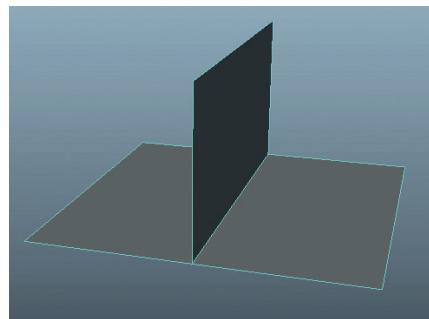


Abb. 4.7: Ein *T-shape*.

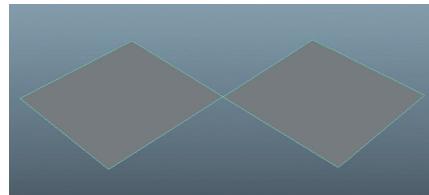


Abb. 4.8: Ein *bow-tie shape*.

13. vgl. Todd Palmer (2014): *Mastering Autodesk Maya 2015*. 1.Aufl. Indianapolis: Sybex. S. 108 f.

14. vgl. http://help.autodesk.com/view/MAYAUL/2015/ENU/?guid=Polygons_overview_Twomanifold_vs
(abgerufen am 27.02.2015)

4. Problemstellungen und Lösungsansätze

Ebenfalls zu vermeiden sind Polygone, die *edges* mit einer Kantenlänge von null beinhalten. Da *edges* am Anfang und am Ende immer einen *vertex* enthalten, bedeutet das, dass sich zwei *vertices* an exakt denselben Koordinaten im Raum befinden. Da dies einer der häufigsten Fehler ist, der beim Modellieren auftritt, sollte das Modell zum Ende der Modellierungsphase auf solche überlagernden *vertices* hin überprüft werden. Eine einfache Lösung für dieses Problem bieten die Funktionen *Merge Vertices* in Maya bzw. *Weld* in 3ds Max, die es ermöglichen, mehrere *vertices* in einer vom Anwender festgelegten Entfernung zueinander zu verschmelzen.

Wie schon angesprochen ist die Architektur von Polygonen eines Objekts (Topologie) für das Funktionieren von Werkzeugen, das Animieren oder das Rendern wichtig. Probleme können vermieden werden, wenn bei der Modellierung optimalerweise nur vier-eckige, in Ausnahmefällen auch dreieckige Polygone verwendet werden. Wenn ein Polygon doch mehr als vier Kanten aufweist, so kann durch Tesselierung, also durch Unterteilung in dreieckige und viereckige Flächen das Polygon strukturiert werden.¹⁵

Nicht zulässig aber möglich sind konkave und nicht-planare Polygone. Konkave Flächen sind nach innen gewölbt. Das bedeutet, dass die *edges* mindestens eines *vertex* einen Innenwinkel von mehr als 180° aufweisen. Erlaubte konvexe Flächen tun das nicht. Eine nicht-planare Fläche enthält *vertices*, die sich nicht exakt auf einer Ebene im Raum befinden. Auf eine planare Fläche trifft das zu. Diese Einschränkungen sind daher vorhanden, da planare und konvexe Polygone sehr viel schneller und korrekt gerendert werden können. Um nicht-planare und konkave Flächen trotzdem darstellen zu können, werden diese trianguliert, also dreieckig unterteilt, denn dreieckige Flächen sind immer planar und konvex.¹⁶

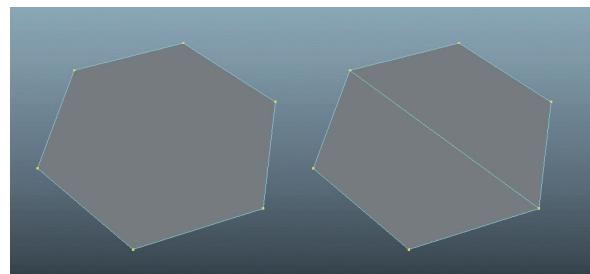


Abb. 4.9: Vergleich von schlechter (links) und guter Topologie (rechts).

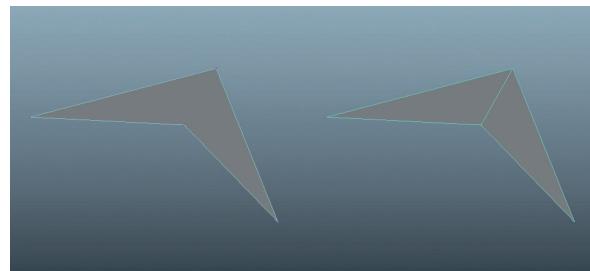


Abb. 4.10: Verbotene (links) und erlaubte Modellierung (rechts) eines konkaven Polygons.

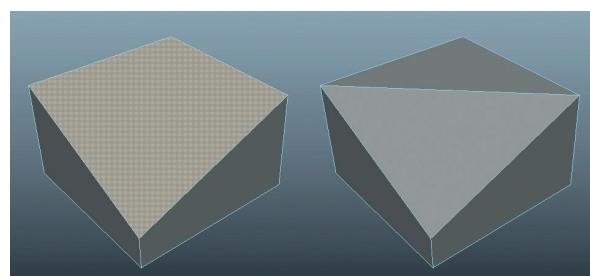


Abb. 4.11: Verbotene (links) und erlaubte Modellierung (rechts) einer nicht-planaren Fläche.

15. vgl. <http://help.autodesk.com/view/MAYAUL/2015/ENU/?guid=GUID-AB60C982-C96E-4947-8CF3-5152406B6A40>
(abgerufen am 06.03.2015)

16. vgl. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 89.

4. Problemstellungen und Lösungsansätze

4.3. Probleme durch *Backface Culling*

Zur performanten Darstellung von Spieleumgebungen gibt es eine ganze Reihe von Beschleunigungsverfahren, welche die Grafikhardware eines Systems entlasten sollen. Eines, das des öfteren zu Problemen geführt hat, ist das sogenannte *Backface Culling*. Das *Backface Culling* zählt zu den *Cull Algorithmen*, die im Prinzip alle die Aufgabe haben, Teile einer Szene, die für den Betrachter nicht einsehbar sind abzutrennen, um zu verhindern, dass diese in die *Rendering Pipeline* geschickt werden. Es wird nur der Teil vom System gerendert, der auch wirklich für den Betrachter sichtbar ist.

Die rückseitigen Polygone von undurchsichtigen Objekten sind vom Betrachter normalerweise nicht einsehbar, es gibt also keinen Grund warum diese vom System gerendert werden sollten. Und dort setzt das *Backface Culling* an. Die Rückseiten aller Objekte in einer Spieleumgebung werden vom Rendervorgang ausgeschlossen, wodurch eine beträchtliche Anzahl an Polygonen eingespart werden kann. Stellt sich nun die Frage, wie der *Cull Algorithmus* feststellen soll, welche Seite eines Polygons die Vorder- und welches die Rückseite ist.

Jedes Polygon im 3D-Raum besitzt eine Vorderseite, sowie eine Rückseite. Die Vorderseite wird durch einen Normalenvektor angezeigt, der orthogonal auf der Fläche steht und die Richtung angibt, in die die Fläche zeigt. Neben der Beleuchtungsberechnung ist der Normalenvektor für die Funktion des *Backface Cullings* wichtig, denn es wird nur die Seite gerendert, in die der Vektor zeigt. Eine Spiegelung des Vektors, wie in Abbildung 4.13 zu sehen, hat das Ausblenden der Fläche zur Folge.¹⁷

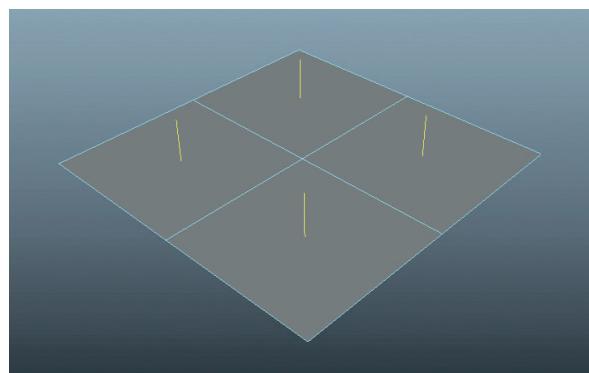


Abb. 4.12: Polygonales Netz mit Normalenvektoren.

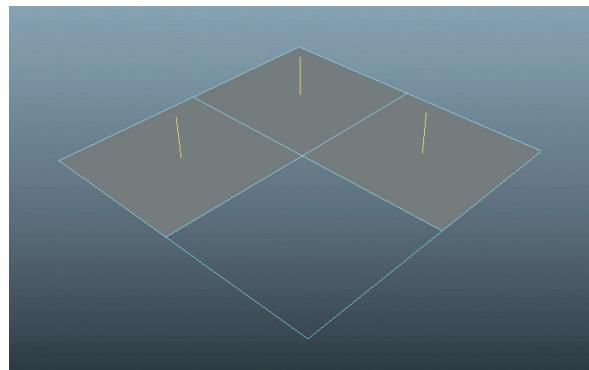


Abb. 4.13: Dasselbe polygonale Netz mit einem ausgeblendeten Polygon.

17. vgl. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*.

3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 446 f, 454 f.

4. Problemstellungen und Lösungsansätze

Probleme, die nun entstehen können sind in Abbildung 4.13 schon erkennbar, denn *faces* deren Normalenvektoren nicht zur Kamera hin ausgerichtet sind, werden spätestens beim Import in eine Spiel-Engine ausgeblendet. Dies führt dann zu folgenden Problemen bei der Darstellung.

Zum einen wäre das *Frontface Culling* zu nennen. Anders als beim *Backface Culling*, werden dort nicht die Rückseiten, sondern die Vorderseiten eines Objekts ausgeblendet. Das passiert wenn alle Normalenvektoren eines Obektes invertiert sind und das führt dazu, dass es quasi möglich ist in das Objekt zu schauen.¹⁸

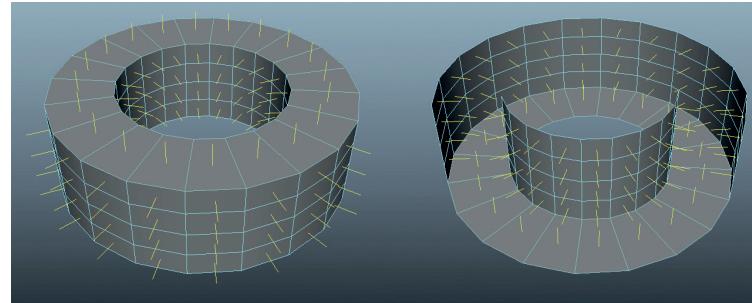


Abb. 4.14: Eine Röhre mit normal ausgerichteten und invertierten Normalenvektoren.

Zum anderen ist ein Fehler zu nennen, der eigentlich zum Bereich der *Nonmanifold Geometry* zählt und exakt in Abbildung 4.13 dargestellt wird. Die Rede ist von polygonalen Netzen, in denen die Normalenvektoren einiger Flächen in eine bestimmte Richtung und die Normalenvektoren der verbleibenden in die entgegengesetzte Richtung weisen.¹⁹ Das führt dazu, dass nie alle Flächen des Polygonnetzes sichtbar sind, egal aus welcher Richtung dieses betrachtet wird. Die Geometrie weist also immer Löcher auf, was vom Betrachter als Fehler wahrgenommen wird.

Um eine fehlerhafte Darstellung bzw. ein Ausblenden von Geometrie zu verhindern, sollten vor dem Import in eine Spiel-Engine alle Polygone auf eine korrekte Ausrichtung hin überprüft werden und im Fehlerfall korrigiert werden.

18. vgl. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*.

3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 92.

19. vgl. Todd Palmer (2014): *Mastering Autodesk Maya 2015*. 1.Aufl. Indianapolis: Sybex. S. 110.

4. Problemstellungen und Lösungsansätze

4.4. Das Alphakanten Problem

Ein Problem, das immer bei der Verwendung von *Alpha-Maps* zur Generierung von Transparenz auftritt, ist das Alphakanten Problem. Hierbei entstehen unschöne, umrahmende Linien am Übergang zwischen opaken und transparenten Teilen der Textur. Diese Ränder werden auch als Alphakanten bezeichnet. Konkret handelt es sich bei diesem Farbsaum um die Hintergrundfarbe der *Texture-Map*. Der in Abbildung 4.15 dargestellte weiße Farbsaum resultiert aus der weißen Hintergrundfarbe der Textur, die noch in das freigestellte Objekt gerendert wird. Ein schwarzer Hintergrund in der Textur hätte eine entsprechend schwarze Alphakante zur Folge. Bei dunkleren Objekten sind weiße Alphakanten auffälliger als schwarze. Umgekehrt sind dunkle Alphakanten an helleren Objekten leichter zu erkennen. Nun könnte für jedes Objekt in der Textur eine separate Hintergrundfarbe festgelegt werden, was allerdings in sehr aufwendige Handarbeit mündet und bei Objekten mit hellen und dunklen Teilen gar unmöglich erscheint.



Abb. 4.15: Objekt mit weißer Alphakante und vergrößerte Darstellung.



Abb. 4.16: Freigestelltes Objekt mit weißer Alphakante (links), resultierend aus der Hintergrundfarbe der Textur (rechts).



Abb. 4.17: Freigestelltes Objekt mit schwarzer Alphakante (links), resultierend aus der Hintergrundfarbe der Textur (rechts).

4. Problemstellungen und Lösungsansätze

Eine schnelle und sehr effektive Methode zur Beseitigung der Alphakanten bietet das Programm xNormal, das bei Installation neue Photoshop-Filter installiert. Einer dieser hinzugefügten Filter nennt sich *dilation* und hat die Aufgabe solche Alphakanten zu eliminieren.

Das Prinzip dahinter ist relativ einfach. Statt einer schwarzen, weißen oder einer sonstigen Hintergrundfarbe sollen die Pixel, welche sich schon im Hintergrund befindet aber noch als Farbsaum erkennbar sind, eine ähnliche Farbe aufweisen wie der letzte Pixel des opaken Bereiches. Anhand der Pflanze in Abbildung 4.18 bedeutet das, dass die Textur an der Blüte rot, am Stengel grün und an der Vase braun auslaufen soll. Da aber auch diese Bereiche verschiedene Helligkeitsstufen aufweisen, muss für jeden Pixel, der sich am Rand des freigestellten Objektes befindet der Farbwert ermittelt und dann über eine einstellbare Entfernung vom Objekt weg mit der ermittelten Farbe gefüllt werden. Es wird gewissermaßen der letzte Pixel extrudiert, um statt der Füllfarbe des Hintergrundes immer die Objektfarbe in die Alphakante zu mischen. So ist ein Farbsaum nicht mehr erkennbar und das Objekt fügt sich besser in den Hintergrund ein.



Abb. 4.18: Freigestelltes Objekt ohne störende Alphakante (links) und Textur mit erweiterter Pixelkante (rechts).

5. Texturoptimierung

Das Kapitel Texturoptimierung beschäftigt sich mit der Steigerung der Performanz durch die Eliminierung von ungenutztem Raum auf den *Texture-Maps* und die damit einhergehende Einsparung von Texturen und Geometrie-Ebenen. Betrachtet werden dafür drei Arbeitsschritte, in denen geklärt wird, wie eine Optimierung mit einem qualitativ hochwertigem Ergebnis ausgeführt werden kann.

Die bisher erstellten Szenen weisen eine selbst festgelegte Anzahl von maximal 15 Ebenen auf. Jede Ebene benötigt die drei Texturtypen (*Diffuse- , Alpha- und Light-Maps*) mit je einer Auflösung von wenigstens 4096 x 2304 Bildpunkten. Hochgerechnet ergeben sich pro Szene eine maximale Anzahl von 45 hoch aufgelösten Texturen. Zur Steigerung der Geschwindigkeit ist eine »Reduktion der verwendeten Texturen, bzw. Verwendung von Texturen mit weniger Quantisierungsstufen (z.B. 16 bit RGBA), so dass alle Texturen in den Texturspeicher passen«²⁰, erforderlich.

Wie schon erwähnt, gliedert sich die Optimierung in drei Schritte. Zum ersten werden überflüssige Polygone von der *Low-Polygon-Szene* abgeschnitten. Gemeint sind die Polygone, dessen Texturkoordinaten im Textur-Raum den Wertebereich $[0,1]^2$ überschreiten und auf dessen Fläche die Textur endet bzw. kachelt.

Zum zweiten werden alle Ebenen in der die *Low-Polygon-Szene* zu einer Ebene zusammengefasst. Hernach werden die durch das *Camera-Mapping* verzerrten Texturkoordinaten entzerrt und im Textur-Raum platzsparend verteilt. Je nach benötigtem Platz im Textur-Raum werden so ein oder mehrere neue Ebenen gebildet.

Zum dritten werden die Texturen von der alten, unoptimierten Geometrie auf die neue, optimierte Geometrie übertragen bzw. in neue Texturen gerendert.

20. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): *Computergrafik und Bildverarbeitung*.

3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 477.

5. Texturoptimierung

5.1. Vorbereitung der *Low-Polygon-Geometry*

Zuallererst wird die noch unoptimierte Geometrie der Spieleumgebung beschnitten und somit alle Polygone, auf denen die Textur endet entfernt. Da sich diese Flächen beim *Camera-Mapping* außerhalb des Sichtkegels der Kamera befanden, überschreiten die Texturkoordinaten an diesen Stellen den Textur-Raum und die Textur endet bzw. kachelt. Da aber alle Texturkoordinaten der Spieleumgebung im Verlauf dieses Workflow neu und platzsparend angeordnet werden und diese Flächen nichts zum späteren Erscheinungsbild der Spieleumgebung beitragen, sondern eher Platz auf den *Texture-Maps* verbrauchen, werden sie entfernt.



Abb. 5.1: Ansicht einer unbeschnittenen Spieleumgebung.

Die Abbildung 5.1 soll verdeutlichen, um welche Flächen es sich konkret handelt. Zu erkennen sind graue Bereiche auf denen keine Textur mehr abgebildet wird. Diese Flächen werden entlang der Texturkante neu unterteilt und anschließend vom Rest der Geometrie abgetrennt und gelöscht.



Abb. 5.2: Das Ergebnis ist eine Spielumgebung ohne ungenutzte Flächen.

5. Texturoptimierung

5.2. UV-Entzerrung und Neuorganisation

Die beschnittene *Low-Polygon-Geometry* wird nun zur Weiterverarbeitung kombiniert. Dazu werden alle (in diesem Fall 15) Ebenen zu einer einzigen Ebene zusammengefasst. Diese Zusammenfassung der Geometrie erleichtert dabei lediglich die Arbeit, da alle Texturkoordinaten mit Auswahl der gesamten Szene im Textur-Raum angezeigt werden und nicht jede einzelne Ebene umständlich markiert werden muss.

Der nun folgende Arbeitsschritt, nämlich das Entzerren und Skalieren der UVs, also der Texturkoordinaten im Textur-Raum ist mit Abstand der fehleranfällige Schritt in der Optimierung, denn bei der Ausführung ist Präzision sehr wichtig.

Warum aber müssen die UVs entzerrt und skaliert werden? Durch das *Camera-Mapping* wurden die UVs mit den spezifischen Einstellungen der Kamera erzeugt. Das bedeutet, dass diese auch das Bildseitenverhältnis der Texturen aufweisen. Die Texturen haben im Fall der betrachteten Szene eine Auflösung von 5000 x 2250 Bildpunkten, was einem Bildseitenverhältnis von 1:2,22 entspricht. Da 3D-Programme aber einzig quadratische Texturen verarbeiten können, wird die Textur mit 5000 x 2250 Bildpunkten verzerrt, um diese in ein Bildseitenverhältnis von 1:1 zu bringen. Wie die Abbildung 5.3 zeigt, ist das Ergebnis eines testweise gerenderten Objektes im Gegensatz zur originalen Textur stark verzerrt, was zu einer Interpolation von Pixeln und damit auch zu Qualitätsverlusten führt. Um diese Verzerrung zu entfernen, müssen die UVs entzerrt bzw. in der Höhe um einen Wert S_y skaliert werden. Um den Wert zu berechnen wurde eine Formel erstellt, mit deren Hilfe der Wert der Skalierung ermittelt werden kann.

$$|S_y| = \left(\frac{W_{uo}}{H_{uo}} \right)^{-1}$$

Für die Variablen W_{uo} und H_{uo} werden die Breite und die Höhe der (unoptimierten) *Texture-Maps*, in diesem Fall 5000 und 2250 eingesetzt.



Abb. 5.3: Orginale Textur (links) und verzerrt gerendertes Objekt (rechts).



Abb. 5.4: Orginale Textur (links) und entzerrtes, gerendertes Objekt (rechts).

5. Texturoptimierung

Mit Hilfe der Formel wird so ein Wert $|S_y| = 0,55$ errechnet. Folglich müssen die UVs also um den Wert 0,55 in y-Richtung kleiner skaliert werden. Das entspricht einer Verkleinerung von 55%. Wie in Abbildung 5.4 zu erkennen, ist das gerenderte Objekt (rechts) nun entzerrt, aber immer noch zu klein. Dieser Größenunterschied resultiert aus dem Fakt, dass für das Rendern eine Texturgröße von 4k angegeben wurde, was einer reellen Größe von 4096 x 4096 Bildpunkten entspricht. Diese Texturgröße wurde daher gewählt, weil die ursprünglichen *Texture-Maps* eine Breite von 5000 Bildpunkten aufweisen und die 4k-Textur dem am nächsten kommt. Um nun zu errechnen, inwieweit die in der Optimierung befindlichen UVs vergrößert werden müssen, dass diese auf der 4k Textur dieselbe Größe wie auf der ursprünglichen *Texture-Map* aufweisen, wurde eine zweite Formel erstellt.

$$|S_{xy}| = \left(\frac{W_{uo}}{W_o} \right)^{-1} - 1$$

Hierbei entspricht die Variable W_{uo} der Breite der unoptimierten Textur also 5000 und W_o der Breite der optimierten Textur, welche im Kapitel 5.3. gerendert werden soll. In diesem Fall wurde, wie schon angedeutet eine 4k Textur angedacht, weshalb ein Wert von 4096 eingesetzt wird. Anhand der Formel wurde ein Wert $|S_{xy}| = 0,18$ ermittelt. Die Texturkoordinaten müssen also um den Wert 0,18 bzw. 18% in x- und y-Richtung vergrößert werden, um die endgültige Größe zu erhalten.

Es gibt allerdings Ausnahmen, in denen nicht alle Arbeitsschritte ausgeführt werden müssen. So gibt es die Möglichkeit, dass die Breite der Ausgangstexturen die selbe ist wie die der geplanten, optimierten Textur. Z.B. wenn beide eine Breite von 4096 Bildpunkten aufweisen. In dem Fall genügt eine Entzerrung der Texturkoordinaten, da diese dann schon die korrekte Skalierung aufweisen.

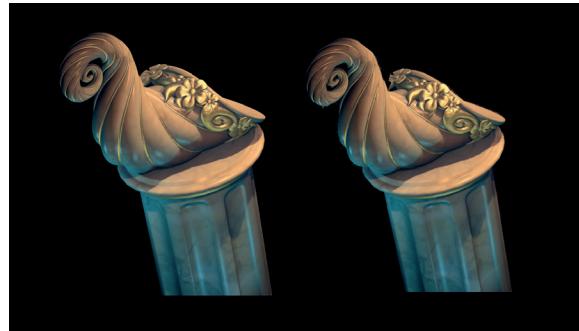


Abb. 5.5: Orginale Textur (links) und entzerrtes, sowie skaliertes, gerendertes Objekt (rechts).

Nun, da die Texturkoordinaten richtig skaliert sind, müssen diese nun platzsparend im Textur-Raum verteilt werden. Zur Orientierung dienen gerasterte Quadrate, in welchem die Texturkoordinaten neu angeordnet werden. Ein Quadrat repräsentiert dabei eine *Texture-Map*. Je weniger Quadrate zur Verteilung aller UVs benötigt werden, desto weniger hoch aufgelöste Texturen müssen später erzeugt werden. Um die Neuverteilung von UVs im Textur-Raum zu veranschaulichen, sind auf Abbildung 5.6 die UVs vor der Umsortierung zu erkennen. Auf Abbildung 5.7 sind wiederum die sortierten UVs zu sehen. Wichtig bei der Neuorganisation der Texturkoordinaten ist die überlappungsfreie Platzierung, denn Überlappungen können zu Fehlern beim Rendern führen.

5. Texturoptimierung

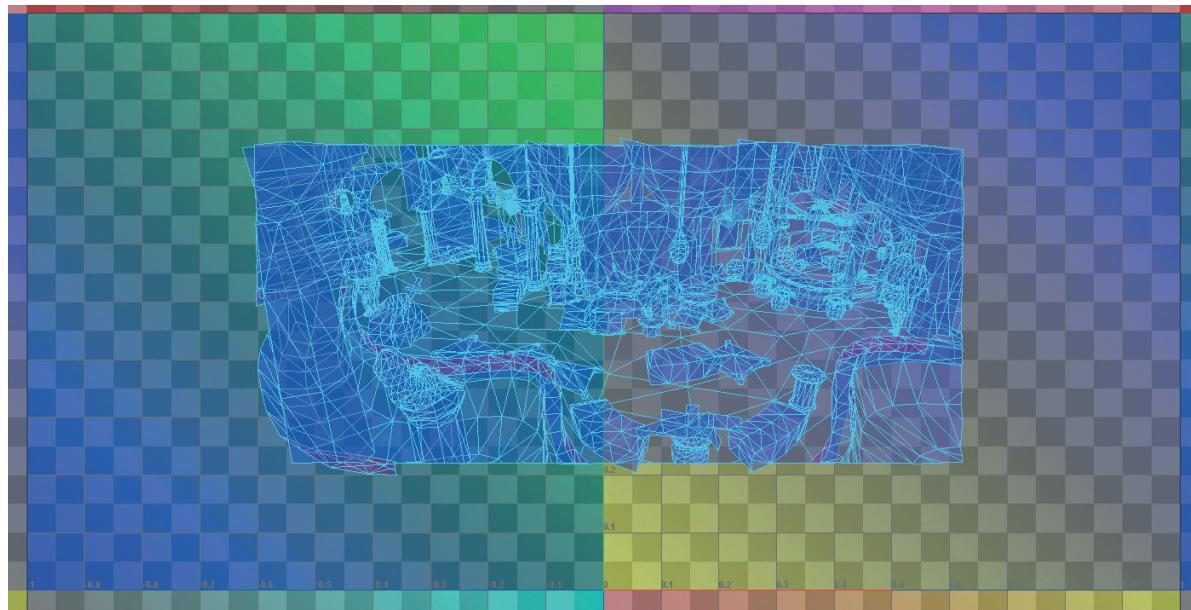


Abb. 5.6: Ungeordnete, sich überlagernde Texturkoordinaten im alten UV-Layout .

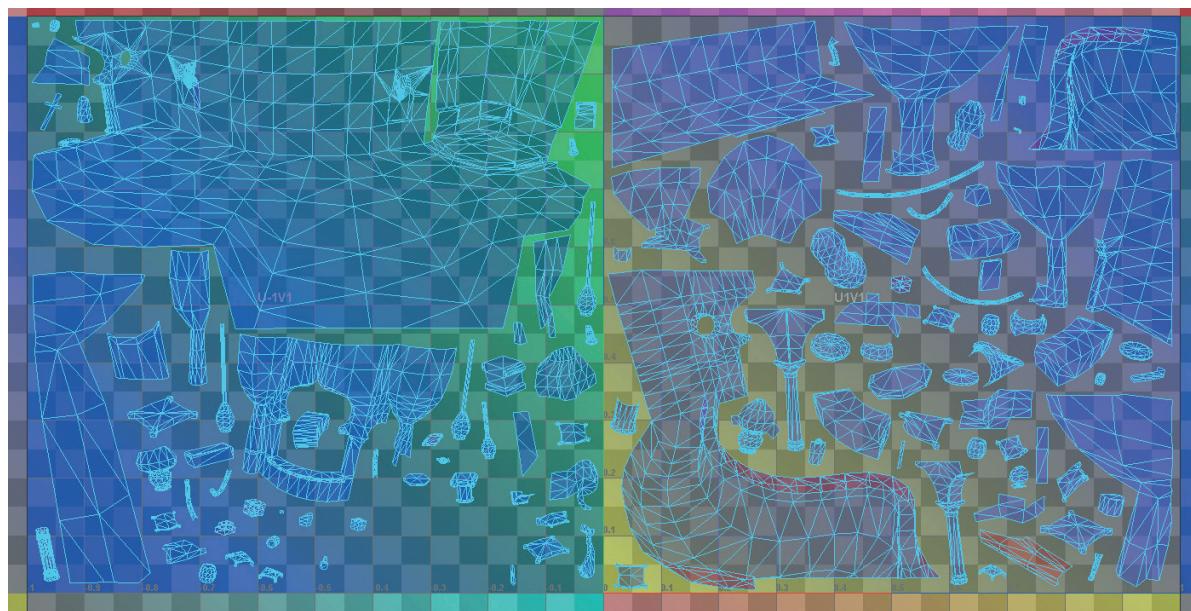


Abb. 5.7: Geordnetes UV-Layout mit Texturkoordinaten, welche auf zwei Quadrate im Textur-Raum verteilt wurden.

Anhand des neu angelegten UV-Layouts werden auch neue Geometrie-Ebenen gebildet. So werden alle Texturkoordinaten in einem Quadrat markiert und die dazugehörige Geometrie zu einer Gruppe zusammengefasst. Im Fall der gewählten Spieleumgebung wurden zwei Gruppen angelegt, da zwei Quadrate im Textur-Raum belegt wurden.

5. Texturoptimierung

5.3. Rendering der optimierten Texturen

Der letzte Schritt im Bereich der Optimierung ist das Rendern der optimierten Texturen, basierend auf dem im vorherigen Arbeitsschritt angelegtem UV-Layout und den daraus entstandenen Gruppen. Für jede Gruppe werden drei Rendervorgänge abgeschlossen, denn es müssen alle *Texture-Maps* übertragen werden, welche auch schon für die unoptimierten Spieleumgebungen verwendet wurden. Die *Diffuse-Maps*, die *Alpha-Maps* und die *Light-Maps*. Mit Hilfe der Tools *Render To Texture* in 3ds Max oder *Transfer Maps* in Maya lassen sich Texturinformationen von einer Geometrie auf die andere übertragen. Es ist also möglich die *Diffuse-, Light-,* und *Alpha-Maps* von der unoptimierten, auf die optimierte Geometrie zu übertragen. Dabei werden die Farbinformationen auf Grundlage der angelegten Texturkoordinaten in neue Texturen gerendert.²¹ Für das Rendern an sich ist es wichtig, alle Filter von den Maps zu entfernen, um ein weichgezeichnetes Ergebnis zu vermeiden.

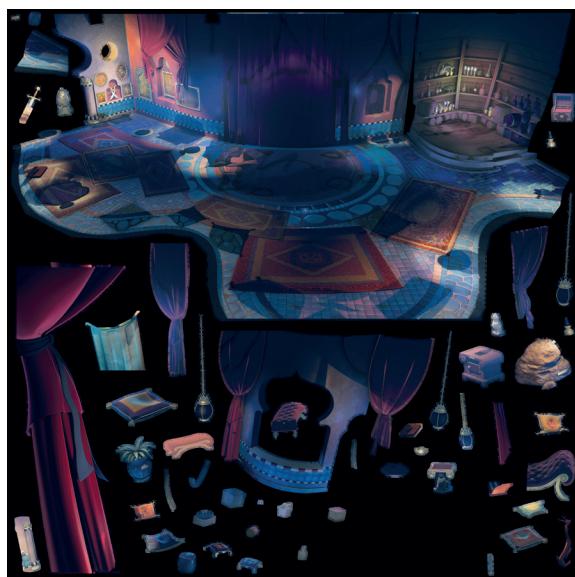


Abb. 5.8: Optimierte *Texture-Map* der ersten Ebene.



Abb. 5.9: Optimierte *Texture-Map* der zweiten Ebene.

In Abbildung 5.8 und 5.9 sind nun die optimierten *Diffuse-Maps* zu sehen, welche mit Hilfe der unoptimierten Geometrie und *Transfer Maps* in Maya gerendert wurden. Insgesamt wurden in der optimierten Variante sechs *Texture-Maps* mit je einer Auflösung von 4096 x 4096 Bildpunkten gerendert. Die *Alpha-Maps* sind im Alphakanal der *Diffuse-Maps* (32 bit RGBA) gespeichert. Die *Light-Maps* enthalten keine *Alpha-Map* und sind als 16 bit RGB gespeichert. Insgesamt benötigen die optimierten *Texture-Maps* 256 MB Speicher.

21. vgl. http://help.autodesk.com/view/MAYAUL/2015/ENU/?guid=Asts_Transfer_Maps (abgerufen am 21.03.2015).

5. Texturoptimierung

Die unoptimierten Texturen belaufen sich auf 15 *Diffuse-Maps* mit *Alpha-Map* im Alphakanal (32 bit RGBA) und 15 *Light-Maps* ohne Alphakanal (16 bit RGB). Insgesamt benötigen die unoptimierten *Texture-Maps* 1070 MB Speicher.

Anhand der zugrunde liegenden Daten lässt sich für diese Spieleumgebung eine Reduktion von 30 auf vier Texturen angeben. Das entspricht einer Ersparnis von ca. 87%. Beim Speicherverbrauch ist die Ersparnis etwas geringer und trotzdem immer noch beachtlich. Im unoptimierten Zustand verbrauchen die Texturen 1070 MB Speicher. Die optimierten benötigen dagegen nur 256 MB. Das entspricht einer Ersparnis ca. 76%.

Aus Erfahrung lässt sich konstatieren, dass die Ersparnis nicht bei allen Spieleumgebungen so hoch ausgefallen ist. Trotzdem konnten besonders die Anzahl der Texturen immer drastisch verringert werden und allein diese Tatsache dient schon der Entlastung der Grafikhardware. Zusätzlich wurden auch Mitarbeiter entlastet, welche die Spieleumgebungen in die Spiel-Engine einpflegten, denn es geht durchaus schneller einer Szene vier, statt 30 Texturen zuzuweisen.

6. Vor- und Nachteile des *Projection-Mappings*

Nachdem nun der gesamte Workflow ausführlich thematisiert wurde, sollen die Vorteile, aber auch die Nachteile des *Projection-Mappings* näher untersucht werden. Als Vorteile werden vor allem die Möglichkeiten der Inszenierung und die Verwendung solch gestalteter Spieleumgebungen genannt. Die Nachteile beziehen sich eher auf die finanziellen Hürden und den Mehraufwand, die ein solch komplexer Workflow mit sich bringt. Zusätzlich soll ein kurzes Resümee, sowie eine Zukunftsprognose einordnen, ob ein solch aufwendiger Workflow sich zum Standard in der Branche entwickeln kann.

Die Vorteile des *Projection-Mappings* liegen klar auf der Hand, es bietet nämlich die Möglichkeit Hintergrundgrafiken besonders dynamisch zu inszenieren und stellt trotzdem den Detailgrad von vorgerenderten Grafiken dar. Zudem läuft es sehr performant. Besonders aber die Möglichkeit dynamische Hintergründe zu entwickeln, die auch realistische Kamerabewegungen erlauben, ist für das traditionsreiche *Point-and-Click-Adventure* eine grafische Revolution, denn bis zur Entwicklung von *The Book of Unwritten Tales 2* waren die Hintergründe in solchen *Adventures* flach und statisch. Kamerabewegungen entblößten, ähnlich wie bei einer Fotografie die Flache Bauweise des Hintergrundes und wirkten aufgrund der fehlenden Bewegungsparallaxe unglaublich. Da aber genau dieses Manöver durch die Technik beseitigt wurde, ergeben sich Möglichkeiten die Kamera mit ins Geschehen zu integrieren.

Es ist möglich die Kamera auf interessante Orte blicken zu lassen. Wenn bspw. ein Charakter eine Weile an einem Ort verharrt, weil er gerade ein Objekt betrachtet, so kann eine kleine Kamerafahrt auf die Spielfigur Details enthüllen, die in der Totalen nicht sichtbar wären. Emotionen können so ebenfalls besser gezeigt werden, da vor allem die Mimik einer Spielfigur in einer halbnahen Einstellung deutlicher in den Fokus rückt. Auf diese Weise können auch kleine Kamerabewegungen sehr gut zum Spielerlebnis beitragen. Natürlich bietet sich diese Art der Kamerabewegung auch bei Interaktionen und Gesprächen zwischen verschiedenen Spielfiguren an.

6. Vor- und Nachteile des *Projection-Mappings*

Eine weitere Möglichkeit, die sich besonders für *Adventures* anbietet, ist die Einbindung der Kamera ins Rätseldesign. Angenommen der Spieler muss ein bestimmtes Objekt finden. So könnte es durchaus möglich sein, das Objekt so in der Spieleumgebung zu platzieren, dass es erst sichtbar wird, wenn die Kamera eine bestimmte Position im Raum einnimmt. Auch könnten durch die Kamerabewegung Vorder- und Hintergrundobjekte zusammentreffen und ein geheimes Bild oder ähnliches sichtbar werden lassen. Es bieten sich also viele neue Möglichkeiten die Kamera intelligent in die Gestaltung von Rätseln mit einzubeziehen.

Ferner bietet sich die Möglichkeit Effekte zu verwenden, welche die grafische Qualität nochmals steigern. Als Beispiel kann hier das Setzen eines Kamerafokus angeführt werden. Dadurch, dass alle Objekte tatsächlich eine Position im Raum inne haben, kann ein Schärfentiefebereich gesetzt werden, der alle Objekte außerhalb weichzeichnet und so einen schönen Filmlook erzeugt und zusätzlich das Auge des Betrachters auf den interessanten Bereich lenkt. Genau diese Technik wurde auch erfolgreich in *The Book of Unwritten Tales 2* eingesetzt.

Die Vorteile, die diese Technik zum Bau von Hintergrundgrafiken bietet, geben den Entwicklern solcher Computerspiele ein mächtiges Werkzeug an die Hand, um Geschichten noch spannender und glaubwürdiger zu inszenieren, als dies im Genre des *Point-and-Click-Adventures* je möglich war und nicht umsonst hat *The Book of Unwritten Tales 2* für die detailreichen Hintergrundgrafiken auch viele gute und sehr gute Bewertungen von namhaften Fachzeitschriften und von den Fans bekommen.

Trotz der vielen Möglichkeiten, die solch schöne und komplex gestaltete Hintergründe auch mit sich bringen, ist der größte Nachteil nicht von der Hand zuweisen. Der Workflow des *Projection-Mappings* ist extrem aufwendig und dementsprechend teuer. Die vielen zusätzlichen Arbeitsschritte des Workflows sind eine finanzielle Belastung für eine Spiele-Firma, vor allem wenn bedacht wird, dass Hintergrundgrafiken auch ohne das *Projection-Mapping* erstellt werden können. Diese bieten dann natürlich nicht die gleiche Dynamik, erfüllen aber trotzdem ihren Zweck.

Für *The Book of Unwritten Tales 2* wurden alle der 57 begehbarer Spieleumgebungen mit Hilfe dieses Workflows erstellt. Über die gesamte anderthalbjährige Entwicklung des Spiels arbeiteten meist mehrere Mitarbeiter gleichzeitig an der Erstellung der Hintergründe, einige in Vollzeit, andere nur wenige Tage die Woche. Dazu kam, dass der Workflow bis zur Entwicklung der Optimierung sehr experimentell war und auch häufig Probleme aufwarf, die aber alle gelöst wurden. Wenn man nun davon ausgeht, dass der gesamte Workflow entwickelt ist, muss bei optimistischer Rechnung wenigstens eine eingearbeitete Person in Vollzeit beschäftigt werden, um alle Hintergründe in einem solch umfangreichen Projekt zu erarbeiten. Das ist zumindest für ein kleines Unternehmen eine nicht zu ignorierende finanzielle Hürde.

6. Vor- und Nachteile des *Projection-Mappings*

KING Art hat zusätzlich zur Finanzierung durch den Publisher eine Kickstarter-Kampagne gestartet, um alle nicht spielrelevanten zusätzlichen Inhalte implementieren zu können. Dazu zählten Kostüme für die Charaktere, optionale Aufgaben und Rätsel, ein durch ein Orchester aufgenommener Soundtrack und eben die Erstellung der Hintergründe mit Hilfe des *Camera-Projection-Mappings*. Mit den durch Kickstarter eingenommenen 171,593 Dollar, konnten alle zusätzlichen Features in das Spiel eingebaut werden.²²

Letztlich sind die begrenzten finanziellen Mittel, wie so oft das größte Hindernis Neues zu wagen. Es lässt sich also nicht unbedingt abschätzen ob diese Technik sich zu einem neuen Standard im Genre des *Point-and-Click-Adventures* entwickelt. Da aber gerade Unternehmen im Bereich der digitalen Medien häufig sehr offen für Innovationen sind und meist auch flexibler als große Konzerne agieren, um auf Kundenwünsche einzugehen, ist es auch nicht ausgeschlossen, dass sich das *Projection-Mapping* als Standard im Genre etabliert.

22. <https://www.kickstarter.com/projects/kingartgames/the-book-of-unwritten-tales-2> (abgerufen am 21.03.2015).

7. Fazit

Abschließend lässt sich konstatieren, dass die Verwendung des *Camera-Projection-Mappings* für die Erstellung von Hintergrundgrafiken einen erfolgreichen Abschluss gefunden hat, denn alle Hintergründe, die so erstellt wurden, haben ihren Weg in das Spiel *The Book of Unwritten 2* von KING Art Games gefunden und die Kritiken über das Spiel im Allgemeinen, aber auch über die Grafik im Besonderen waren ausgezeichnet. Trotz der langen Entwicklung des Workflows und der damit verbunden Probleme sind die Hintergrundgrafiken eine tragende Säule des Spiels und transportieren in jeder Sekunde die besondere Atmosphäre, die das Spiel ausmacht.

Für mich persönlich war die Arbeit nicht immer leicht. Ein langer Lernprozess und viele Fehler, die es auszubügeln galt, wurden von mir begangen, aber schlussendlich spricht das Ergebnis für sich. Natürlich waren neben mir auch noch viele andere Leute an der Arbeit beteiligt, ich kann also nicht die gesamte Entwicklung für mich reklamieren. Nichtsdestotrotz konnte ich als einer der Wenigen von Anfang bis Ende an der Technik arbeiten, diese weiterentwickeln und auch viel Neues dazu lernen.

Sollte sich dieser Workflow zum Standard in der Branche entwickeln, wäre das durchaus zu begrüßen, denn die Entwicklung einer noch so tollen Innovation ist wertlos, wenn sie nicht genutzt werden kann. Nun kann die Entwicklung des Workflows nicht als neue Innovation bezeichnet werden, aber trotzdem ist es befriedigend, wenn die Arbeit wertgeschätzt wird und von anderen aufgegriffen und vlt. sogar noch verbessert wird. So bin ich stolz auf die geleistete Arbeit und hoffe, dass ich auch in Zukunft noch viele Leute mit meiner Arbeit erfreuen kann.

Literaturverzeichnis

help.autodesk.com/

kickstarter.com

knowledge.autodesk.com

Michael Bender | Manfred Brill (2006): Computergrafik. 2.Aufl. München: Carl Hanser Verlag.

Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung. 3.Aufl. Wiesbaden: Vieweg+Teubner Verlag.

Todd Palmer (2014): Mastering Autodesk Maya 2015. 1.Aufl. Indianapolis: Sybex.

Quellenverzeichnis

1. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 24, 25, 27.
4. Michael Bender | Manfred Brill (2006): Computergrafik. 2.Aufl. München: Carl Hanser Verlag. S. 191.
5. Todd Palmer (2014): Mastering Autodesk Maya 2015. 1.Aufl. Indianapolis: Sybex. S. 107 ff.
6. http://knowledge.autodesk.com/support/maya/learn-explore/caas/mne-help/global/docs/maya2014/en_us/files/Polygon-selection-and-creation-Edge-ring-and-edge-loop-selection-tips.htm.html (abgerufen am 18.02.2015)
7. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 459.
8. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 458 - 459.
8. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 458 - 459.
9. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 363.
10. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 187.
11. Michael Bender | Manfred Brill (2006): Computergrafik. 2.Aufl. München: Carl Hanser Verlag. S. 298.
12. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 142 ff.
13. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 268.
13. Todd Palmer (2014): Mastering Autodesk Maya 2015. 1.Aufl. Indianapolis: Sybex. S. 108 f.
14. http://help.autodesk.com/view/MAYAUL/2015/ENU/?guid=Polygons_overview_Twomanifold_vs
(abgerufen am 27.02.2015)
15. <http://help.autodesk.com/view/MAYAUL/2015/ENU/?guid=GUID-AB60C982-C96E-4947-8CF3-5152406B6A40>
(abgerufen am 06.03.2015)
16. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 89.
17. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 446 f, 454 f.
18. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 92.
19. Todd Palmer (2014): Mastering Autodesk Maya 2015. 1.Aufl. Indianapolis: Sybex. S. 110.
20. Nischwitz, Alfred | Fischer, Max | Haberäcker, Peter | Socher, Gudrun (2011): Computergrafik und Bildverarbeitung.
3.Aufl. Wiesbaden: Vieweg+Teubner Verlag. S. 477.
21. http://help.autodesk.com/view/MAYAUL/2015/ENU/?guid=Asts_Transfer_Maps (abgerufen am 21.03.2015).
22. <https://www.kickstarter.com/projects/kingartgames/the-book-of-unwritten-tales-2> (abgerufen am 21.03.2015).