Rendern von Unterteilungsflächen mittels Hardware Tessellierung*

Matthias Nießner

Friedrich-Alexander-Universität Erlangen-Nürnberg
niessner@cs.stanford.edu

Abstract: Computergenerierte Bilder haben sich längst zu einem festen Bestandteil unseres alltäglichen Lebens entwickelt. Neben Computerspielen und Filmen sind synthetische Bilder auch ein wesentlicher Bestandteil von Illustrationen in Zeitschriften und Plakaten - meist ohne, dass wir es bewusst wahrnehmen. Die Qualität erzeugter Bilder hängt dabei maßgeblich von der verwendeten Geometriebeschreibung der zugrunde liegenden 3D Umgebungen ab. Unterteilungsflächen (*Subdivision Surfaces*) haben sich hierbei aufgrund ihrer besonderen Oberflächeneigenschaften als sehr nützlich erwiesen. Letztendlich haben sich Unterteilungsflächen wegen hochwertiger Resultate - speziell in der Filmbrache - als absoluter Industriestandard durchgesetzt und kommen heutzutage in nahezu allen Produktionen zum Einsatz. Ein gravierender Nachteil ist allerdings die Komplexität zugrunde liegender Berechnungen, was entsprechend lange Rechenzeiten zur Folge hat. In dieser Dissertation befassen wir uns mit Algorithmen, welche die Auswertung dieser Oberflächen um mehrere Größenordnungen beschleunigen. Dadurch können hochwertige Filminhalte binnen weniger Millisekunden auf handelsüblichen Computern dargestellt werden und somit in Echtzeitanwendungen, wie Spielen, eingesetzt werden. Die Ergebnisse dieser Arbeit wurden unter anderem im Rahmen des OpenSource Projekts *OpenSubdiv* in Zusammenarbeit mit Pixar Animation Studios veröffentlicht und finden bereits breite Anwendung in der Industrie, wie zum Beispiel in der Modellierungssoftware von Branchenprimus Autodesk (z.B. Maya, Mudbox) und anderer Hersteller. Darüber hinaus werden unsere Algorithmen in Computerspielen wie *Call of Duty: Ghosts* von Activision verwendet, die dadurch die Oberflächenqualität von Filmen erreichen.

Abbildung 1: Unsere Algorithmen ermöglichen es qualitativ hochwertige Filminhalte auf handelsüblichen Computern in Echtzeit darzustellen. Die Abbildung zeigt wie unser adaptiver Unterteilungsalgorithmus das Automodell aus *Pixar's Cars* in Echtzeit verarbeitet. Die Grundidee dabei ist es, die Oberfläche - falls möglich - analytisch auszuwerten und nur dort wo es auch wirklich nötig ist, aufwändige Unterteilungen zu berechnen. Im linken Bild sehen wir die Eingabedaten, welche die Oberfläche definieren. In der Mitte werden die adaptiven Unterteilungen dargestellt (eine Farbe je Unterteilungslevel), und rechts sehen wir das Ergebnisbild. © Disney/Pixar

*Englischer Titel der Dissertation: "Rendering Subdivision Surfaces using Hardware Tessellation" [Nie13]

1 Einführung

Computergenerierte Bilder werden immer wichtiger im alltäglichen Leben. Mit steigender Relevanz erhöhen sich auch die Anforderungen, die an die Bildqualität gestellt werden. Um eine möglichst wirklichkeitsgetreue Abbildung zu garantieren, benötigen Bilder deshalb zunehmend mehr visuelles Detail. Dieses Detail basiert dabei maßgeblich auf der Oberflächenrepräsentation der zugrunde liegenden Szenengeometrie. Während in der Echtzeitgrafik vor allem Dreiecksnetze etabliert sind, haben sich in der Filmbrache Unterteilungsflächen (*Subdivision Surfaces* [CC78]) zu einem Industriestandard entwickelt (cf. Abbildung 1, 2). Unterteilungsflächen bieten Grafikdesignern im Gegensatz zu polygonbasierten Ansätzen einen weit höheren Grad an Flexibilität beim Modellieren und liefern dank ihrer Oberflächeneigenschaften sehr hochwertige Bildergebnisse.

Ein gravierender Nachteil von Unterteilungsflächen ist die einhergehende Rechenkomplexität beim Auswerten dieser Oberflächendarstellung. Daher werden in Filmproduktionen typischerweise teure Hochleistungsrechner mit entsprechender Rechenleistung zur Bildgenerierung (*Rendering*) eingesetzt. In Echtzeitanwendungen wie Computerspielen oder Modellierungssoftware, müssen allerdings mehrere Bilder pro Sekunde generiert werden (mindestens 30), was die Anwendung von hochwertigen Filminhalten deutlich erschwert.

Abbildung 2: Generiertes Bild von *Woody* aus *Pixar's Toy Story* mittels unserem Verfahren. Dank unserer neuartigen Technik können Bilder aus Filmszenen in unter 10 Millisekunden auf handelsüblichen Computern gerendert werden. © Disney/Pixar

Im Rahmen dieser Dissertation haben wir daher Verfahren entwickelt, um Inhalte bestehend aus Unterteilungsflächen, in Echtzeitanwendungen auf handelsüblichen Desktop-Computern einzusetzen. Wir greifen dabei auf die Rechenleistung moderner Grafikkarten zurück und entwickeln unsere Algorithmen speziell für die massiv parallele GPU Prozessorarchitektur. Neben der hohen Prozessoranzahl, besitzen Grafikkarten außerdem eine Hardware Tessellierungseinheit, welche besonders beim Rendern von parametrischen Flächenstücken nützlich ist. Der entscheidende Vorteil der Hardware Tessellierung liegt darin, dass die Oberflächengeometrie auf den jeweiligen Berechnungseinheiten ausgewertet und direkt weiterverarbeitet wird. Dadurch können Oberflächen dynamisch (bzgl. der Kamera) verfeinert und resultierende Dreiecke direkt und ohne zusätzliche Speicherzugriffe rasterisiert werden. Dies ist auf parallelen Plattformen wie Grafikkarten entscheidend, da Speicherbandbreite und Latenz typischerweise die Leistungsfähigkeit limitieren. Mit der Tessellierungseinheit lassen sich Objekte außerdem

sehr einfach animieren, da lediglich die Kontrollpunkte der entsprechenden Flächenstücke angepasst werden müssen.

Ein entscheidender Beitrag der Arbeit ist die (parallele) Formulierung unserer Algorithmen, welche es erlauben die Fähigkeiten der Tessellierungseinheit optimal zu nutzen. Wir zerlegen daher Unterteilungsflächen zunächst in einzelne Flächenstücke, welche anschließend vom Tessellierer verarbeitet werden. Entgegen der naiven rekursiven Definition von Unterteilungsflächen, können wir dadurch die Oberfläche direkt auswerten, was auf Grafikkarten weitaus schneller und speicherfreundlicher ist. Darüber hinaus kann unser Verfahren effizient hierarchisch definiertes Oberflächendetail (*hierarchical edits*) und halbweiche Knicke (*semi-sharp creases*) benutzen und anwenden. Um feine und hochfrequente Oberflächeneigenschaften darzustellen, führen wir außerdem eine analytische Verschiebungsfunktion (*Displacements*) ein, die Oberflächennormalen implizit definiert. Im Gegensatz zu traditionellen Repräsentationen wird dadurch der Speicherbedarf massiv reduziert, was zu einer deutlich verbesserten Renderzeit führt. Unsere Repräsentation erlaubt es zudem Oberflächendetail dynamisch zu verändern ohne zusätzliche Berechnungskosten zu verursachen. Das macht unser Verfahren sowohl schneller als auch deutlich flexibler - und vor allem praktikabel. Bei der Bildgenerierung werden allerdings immer noch viele unnötige Berechnungen durchgeführt. Beispielsweise werden viele Flächenstücke ausgewertet, obwohl sie verdeckt und nicht sichtbar sind. Um dies zu vermeiden, identifizieren wir Flächenstücke die zum einen von der Kamera abgewandt sind und zum anderen durch andere Flächenstücke verdeckt sind [NL12]. Diese werden anschließend aus der Renderliste entfernt und müssen nicht mehr weiter verarbeitet werden. Der Berechnungsaufwand kann dadurch signifikant reduziert werden und Renderzeiten um effektiv die Hälfte gesenkt werden. Wir befassen uns ebenso mit der Kollisionserkennung für die Hardware Tessellierung in Echtzeitanwendungen [NSSL13]. Speziell bei Unterteilungsflächen mit Verschiebungsfunktionen und dynamischer Tessellierung ist dies sehr anspruchsvoll. Im Rahmen dieser Kurzfassung der Dissertation legen wir allerdings den Fokus auf das Auswerten und Anzeigen (siehe Abschnitt 3) von Unterteilungsflächen, und auf die Repräsentation von feinem Oberflächendetail (siehe Abschnitt 4).

Letztlich wurde in der zugrundeliegenden Dissertation eine umfassende Lösung - bestehend aus zahlreichen neuartigen Algorithmen - entwickelt, um Unterteilungsflächen erstmals in Echtzeitanwendungen einzusetzen. Die Forschungsergebnisse bilden die Grundlage von Pixar's OpenSource Projekt *OpenSubdiv* und finden bereits - ein Jahr nach Veröffentlichung - breite Anwendung in der Industrie (z.B. Autodesk Maya & Mudbox, Call of Duty Ghosts, etc.). Siehe Abbildungen 1, 2, 8.

2 Grundlagen: Unterteilungsflächen und Moderne Grafikhardware

Unterteilungsflächen In folgendem Abschnitt geben wir eine kurze Einführung zu Unterteilungsflächen, um das Verständnis unserer Algorithmen zu erleichtern. Unterteilungsflächen wurden von Catmull und Clark [CC78] eingeführt und sind eine Verallgemeinerung von Tensorproduktflächen, welche eine glatte Grenzfläche definieren. Wir nehmen dabei hauptsächlich auf Catmull-Clark Oberflächen Bezug, welche bi-kubische B-

Splineflächen auf Topologien beliebiger Konnektivität erweitern und ein reines Quadmesh nach einem Unterteilungsschritt erzeugen. Neben Catmull-Clark Flächen gibt es noch eine Vielfalt anderer Unterteilungsverfahren, die zwar von unseren Algorithmen verarbeitet werden können, allerdings in der Praxis eine untergeordnete Rolle einnehmen. Die Grundidee von Unterteilungsflächen besteht darin, ein grobes Basismesh iterativ mittels Unterteilungsregeln zu verfeinern. Diese Regeln beschreiben Linearkombinationen der Kontrollpunkte des aktuellen Unterteilungslevels i, woraus sich die Kontrollpunkte des nächst feineren Levels $i+1$ ableiten lassen. Die glatte Grenzfläche ergibt sich nach einer unendlichen Anzahl von Unterteilungen und ist C^2, abgesehen von außergewöhnlichen Gitterpunkten (Vertices mit Valenz ungleich 4), wo sie C^1 ist. Die zugrundeliegenden Catmull-Clark Unterteilungsregeln sind definiert durch Flächenpunkte (f_j), Kantenpunkte (e_j) und Vertexpunkte (v_j) mit jeweiliger Valenz n (entsprechend in Abbildung 3 gekennzeichnet). Die zugehörigen Regeln sind gewichtete Mittelwerte des vorherigen Unterteilungslevels:

- Flächenregel: f^{i+1} ist der Schwerpunkt der Vertices der umgebenden Fläche,
- Kantenregel: $e_j^{i+1} = \frac{1}{4}(v^i + e_j^i + f_{j-1}^{i+1} + f_j^{i+1})$,
- Vertexregel: $v^{i+1} = \frac{n-2}{n} v^i + \frac{1}{n^2} \sum_j e_j^i + \frac{1}{n^2} \sum_j f_j^{i+1}$.

Es gibt zudem zahlreiche Erweiterungen zu Catmull-Clark Flächen, wie harte Kanten, halbweiche Knicke und hierarchisch definiertes Oberflächendetail. Diese können problemlos von unseren Algorithmen verarbeitet werden, allerdings gehen wir hier im Rahmen dieser Kurzfassung nicht näher darauf ein.

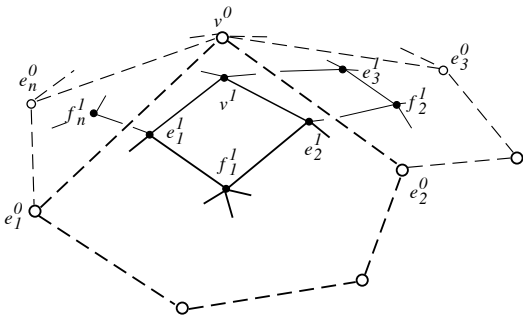


Abbildung 3: Kennzeichnung von Vertices des Basismeshes (Unterteilungslevel 0) um den Punkt v^0 mit Valenz n und des nächsten Unterteilungslevels 1.

Moderne Grafikhardware Moderne Grafikkarten sind massiv parallele Prozessorarchitekturen. Sie bestehen aus mehreren *Streaming Multiprozessoren* (SMs), wobei jeder SM eine eigene Vektoreinheit ist und somit viele Daten mit gleicher Instruktion parallel verarbeiten kann, *Single Instruction, Multiple Data* (SIMD). Im Rahmen dieser Arbeit verwenden wir eine NVIDIA GTX 480, die aus 15 SMs mit einer SIMD-Breite von 32 besteht und folglich 480 Threads parallel ausführen kann. Im Verhältnis zu CPUs, wird bei GPUs weit mehr Chipfläche für Berechnungseinheiten als für Zwischenspeicher verwendet. Dadurch steht zwar sehr viel Rechenleistung zur Verfügung, allerdings ist die Speicherbandbreite typischerweise ein Flaschenhals. Daher ist es sehr wichtig dies im Entwurf unserer Algorithmen zu berücksichtigen. Dies kann speziell mit der Hardware Tessellierungseinheit erreicht werden, welche seit der Einführung der XBox 360 auf moderner Grafikhardware verfügbar ist. Dabei werden einzelne Flächenstücke (*Patches*) jeweils durch ein unabhängiges Set von Kontrollpunkten definiert, parallel ausgewertet und gerendert. Die Patch Kontrollpunkte

werden zunächst von der programmierbaren Hullshadereinheit (je 1 Thread pro Kontrollpunkt) verwendet um sogenannte Tessellierungsfaktoren zu bestimmen (je Patchkante und Patchinnenfläche), welche die Tessellierungsdichte bestimmen. Die Hardware generiert anschließend für jeden Patch Abtastpunkte mit uv-Patchparameterwerten, an welchen die Oberfläche ausgewertet werden muss - dies geschieht in der programmierbaren Domainshadereinheit. Die Herausforderung dabei ist es die Oberfläche *direkt* auszuwerten, was vor allem bei Unterteilungsflächen durch die rekursive Definition sehr anspruchsvoll ist (speziell an außergewöhnlichen Gitterpunkten).

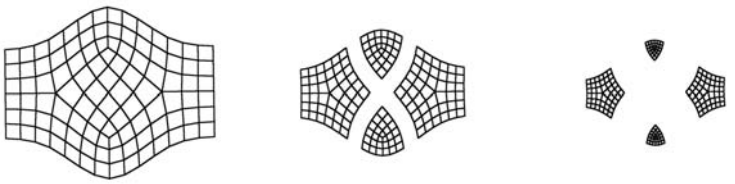


Abbildung 4: Unser adaptives Unterteilungsverfahren angewandt auf einem Kontrollgitter mit vier außergewöhnlichen Gitterpunkten. Wir unterteilen nur in Bereichen in welchen direkte Auswertung nicht möglich ist; i.e., an außergewöhnlichen Punkten.

3 Adaptive Unterteilung von Unterteilungsflächen auf Moderner Grafikhardware

Die Grundidee adaptiver Unterteilung ist es nur dort zu unterteilen, wo es auch wirklich nötig ist. Bei Catmull-Clark Flächen kann daher ausgenutzt werden, dass alle regulären Patches als bi-kubische B-Splines definiert sind. Diese können an beliebigen Parameterstellen ausgewertet und somit direkt vom Hardware Tessellierer verarbeitet werden. Alle anderen Flächen konvertieren wir durch sukzessives Unterteilen in (großteils) reguläre Patches, um sie direkt auswerten zu können. Unser adaptives Unterteilungsverfahren, unterteilt daher zunächst nicht-reguläre Regionen mittels GPGPU-Kernel. Anschließend werden alle Teilflächen so zusammengefügt und mit dem Hardware Tessellierer gerendert, dass dabei keine Löcher entstehen. Das Verfahren ist in Abbildung 1 visualisiert.

3.1 Datenparallele Unterteilung basierend auf Tabellen

Unterteilungsflächen können anhand ihrer rekursiven Definition sehr einfach auf der CPU ausgewertet werden. Eine effiziente GPU Auswertung ist dagegen schwieriger, da Berechnungen parallelisiert werden müssen und zugehörige Nachbarschaftsinformationen benötigt werden. Um dies zu vereinfachen, nehmen wir an, dass die Konnektivität der Unterteilungsfläche statisch ist. Dadurch kann unser Algorithmus vorberechnete Tabellen verwenden, in welchen die Unterteilungsregeln der jeweiligen Kind-Vertices bezüglich ihrer Eltern kodiert sind. Zur Laufzeit müssen nur noch die entsprechenden Linearkombinationen mit den Vertexdaten des vorausgehenden Unterteilungslevels angewandt werden. Dazu verwenden wir drei GPGPU-Kernel je Unterteilungslevel - jeweils für Flächen, Kanten und Vertices - die auf einem gemeinsamen Vertexpuffer arbeiten und einen Thread pro Ausga-

bevertex ausführen. Die Kernel werden ausgehend vom Basismesh sukzessive ausgeführt. Dadurch werden Vertexupdates entsprechend vom gröbsten zum feinsten Unterteilungslevel propagiert (z.B. unter Animation). In den Unterteilungstabellen sind außerdem Features wie zum Beispiel Meshgrenzen und halbweiche Knicke, kodiert. Für weiter Details bezüglich des Tabellenaufbaus verweisen wir auf die entsprechende Veröffentlichung [NLMD12]. Unsere kompakte Darstellung der Unterteilungsregeln ist das derzeit schnellste GPU Unterteilungsverfahren.

3.2 Feature-adaptive Unterteilung

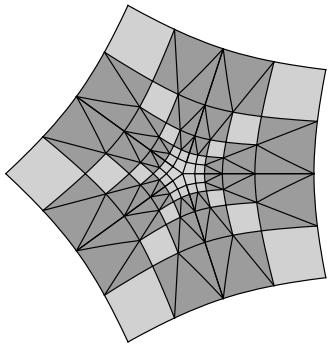


Abbildung 5: Die Patchanordnung um einen außergewöhnlichen Gitterpunkt. Parametergebiete von UPs sind rot, reguläre VPs sind blau, und irregulare VPs sind grün (Zentrum) gekennzeichnet.

Die uniforme Unterteilung einer Catmull-Clark Oberfläche, wie im vorherigen Abschnitt beschrieben, kann zwar in Echtzeit auf Grafikkarten ausgeführt werden, ist jedoch relativ aufwändig. Vor allem der Speicherbedarf wächst exponentiell mit der Anzahl der Unterteilungslevel. Um dies zu vermeiden, unterteilen wir adaptiv; das heißt es werden nur Regionen unterteilt, die wir nicht direkt auswerten können. Im Falle von Catmull-Clark sind dies alle Flächen, die an einen außergewöhnlichen Vertex angrenzen (*irregulärer Patch*). Alle regulären Patches sind definiert als bi-kubische B-Splineflächen und können daher direkt (ohne weitere Unterteilung) ausgewertet werden. Nachdem ein irregulärer Patch unterteilt wurde, entstehen daraus vier Kind-Flächen. Drei davon enthalten keinen außergewöhnlichen Vertex mehr und sind demzufolge regulär und somit direkt auswertbar. Als Ergebnis der adaptiven Unterteilen erhalten wir eine verschachtelte Anordnung von bi-kubischen Flächen und ein sehr kleines irreguläres Rest-Flächenstück auf dem feinsten Unterteilungslevel. Wir können dieses Flächenstück zwar nicht direkt an beliebigen Parameterpositionen auswerten, allerdings ist es möglich die Grenzflächenpositionen exakt an der Stelle des außergewöhnlichen Vertex zu bestimmen. Dafür werden sogenannte Grenzschablonen verwendet, welche eine Linearkombination des 1-Rings um den entsprechenden Punkt definieren.

Für die feature-adaptive Unterteilung verwenden wir Unterteilungstabellen wie in Abschnitt 3.1 beschrieben. Die Tabellen sind allerdings wesentlich kleiner, da im Gegensatz zur uniformen Unterteilung, nur irreguläre Patches unterteilt werden. Bei einem Modell mit v außergewöhnlichen Gitterpunkten, und F Patches, werden ca. $12k \cdot v$ Kind-Flächen nach k Unterteilungsleveln generiert. Bei uniformer Unterteilung sind es ca. $4^k \cdot F$, wobei $F >> k$. Da die regulären bi-kubischen Flächen jeweils aus 16 Kontrollpunkten bestehen, muss zusätzlich zum eigentlichen irregulären Patch auch noch dessen 1-Ring mit unterteilt werden. Ein Beispiel einer drei-Level adaptiven Unterteilung eines Kontrollnetzes mit vier außergewöhnlichen Gitterpunkten ist in Abbildung 4 dargestellt. Da die Konnektivität statisch ist, kann die entsprechende Logik für die Tabellenberechnung in einem Vorverarbeitungsschritt auf der CPU durchgeführt werden, was in unserer unoptimierten Implementierung ca. 50 Millisekunden (je nach Modell) dauert.

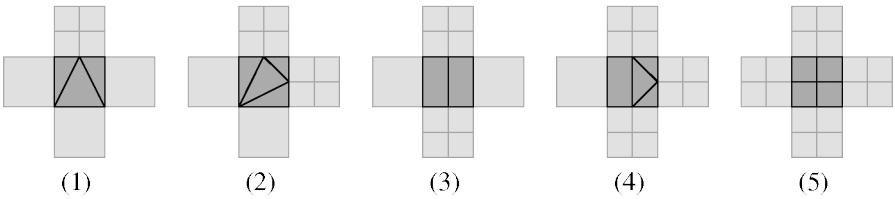


Abbildung 6: Es gibt fünf verschiedene Konstellationen von Übergangs-Patches (**UPs**). UPs sind rot, der aktuelle Unterteilungslevel blau, und der nächste Level grün gekennzeichnet. Das Aufteilen des Parametergebietes eines UPs in mehrere Teil-Patches erzwingt gleiche Länge für gemeinsame Kanten. Dadurch kann die Tessellierungsrate beliebig verändert werden ohne, dass Löcher entstehen.

3.3 Patchkonstruktion

Nach dem Schritt der feature-adaptive Unterteilung (siehe vorigen Abschnitt), verwenden wir die GPU hardware Tessellierung um die generierten Patches adaptiv zu triangulieren und anzuzeigen. Die Anzahl der Geometrie-Abtastpunkte auf Patchkanten kann dabei beliebig über die Tessellierungsfaktorn gesetzt werden. Für jeden Unterteilungslevel unterscheiden wir zwischen Voll-Patches und Übergangs-Patches.

Voll-Patches Voll-Patches (**VPs**) sind Flächen die nur mit Patches des gleichen Unterteilungslevels gemeinsame Kanten haben. Das können sowohl reguläre, als auch irreguläre Patches sein. Reguläre VPs können direkt von der Hardware Tessellierung verarbeitet und als bi-kubische B-Splineflächen gerendert werden. Durch die feature-adaptive Unterteilung stellen wir sicher, dass irreguläre VPs nur an den Eckpunkten ausgewertet werden müssen (dort kann die Grenzposition berechnet werden). Das bedeutet, für einen gegebenen Tessellierungsfaktor TF sind $\lceil \log_2 TF \rceil$ adaptive Unterteilungsschritte nötig um die selbe Tessellierungsdichte zu erzielen. Da auf aktueller Hardware der Tessellierungsfaktor auf 64 limitiert ist, müssen maximal 6 Schritte durchgeführt werden. Um die Grenzpositionen irregulärer Punkte zu bestimmen, verwenden wir einen separaten Vertexshader, welcher die Grenzschablonen anwendet.

Übergangs-Patches Bei der feature-adaptiven Unterteilung in Abschnitt 3.2 stellen wir sicher, dass generierte bi-kubische Patches nur an Flächen des selben oder um maximal eins verschiedenen Unterteilungslevels angrenzen. Patches, welche an Flächen des nächst feineren Unterteilungslevels angrenzen nennen wir Übergangs-Patches (**UPs**). Wir stellen zudem sicher, dass UPs immer regulär sind. Um Löcher beim Rendern zu vermeiden, müssen gemeinsame Kanten von benachbarten Patches an den selben Parameterpositionen ausgewertet werden. Dazu teilen wir UPs in verschiedene Teil-Patches auf und gewährleisten damit nahtlose Übergänge an T-Abzweigungen. Unter Ausnutzung von Rotationssymmetrie, ergeben sich fünf verschiedene Fälle (siehe Abbildung 6). Jedes Teil-Parametergebiet gehört dabei zu separaten Teil-Patches, welche allerdings alle durch die gleichen 16 Kontrollpunkte definiert sind. Die Unterteilung hat also lediglich auf das Parametergebiet, nicht aber auf die Oberfläche, Einfluss. Durch die geschickte Anordnung

der verschiedenen UPs werden T-Abzweigungen eliminiert und die Tessellierungsrate an gemeinsamen Kanten kann beliebig festgesetzt werden, ohne dass Löcher entstehen. Beim Setzen der Tessellierungsfaktorn sollte allerdings der aktuelle Unterteilungslevel beachtet werden um eine gleichmäßige Oberflächenabtastung zu gewährleisten. Die verschachtelte Anordnung von Patches ist in Abbildung 5 dargestellt. Generierte Bilder mit unserem Verfahren sind in den Abbildungen 1, 2, 8 zu finden.

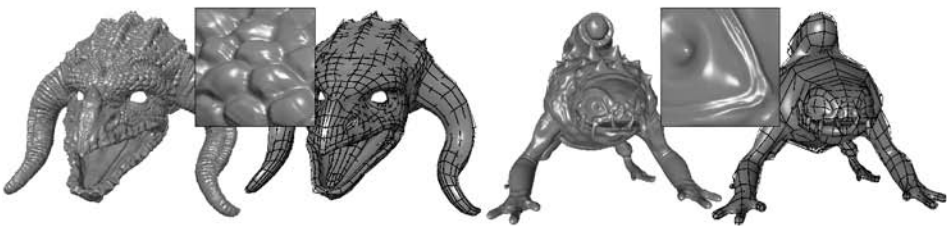


Abbildung 7: Feines Oberflächendetail wird mit unserer analytischen Verschiebungsfunktion sehr effizient auf Unterteilungsbasisflächen (siehe jeweils links) hinzugefügt. Die Renderzeiten von links nach rechts auf einer NVIDIA GTX 480: 1.7 ms, 1.2 ms, 1.3 ms, 0.85 ms. Unter 1 MB Speicher.

4 Analytische Verschiebungsfunktionen für Unterteilungsflächen

Unterteilungsflächen definieren (in der Regel) eine glatte Grenzfläche, meist ohne feines Obeflächendetail. Wir führen daher eine Verschiebungsfunktion für hochfrequentes detail auf Unterteilungsflächen ein. Im Gegensatz zu vorherigen Methoden ist unsere Verschiebungsfunktion analytisch und kommt daher ohne zusätzlichen Speicher für Oberflächennormalen aus. Wir stellen unsere verschobene Oberfläche dar als

$$f(u,v) = s(u,v) + N_s(u,v)D(u,v),$$

wobei $s(u,v)$ eine Catmull-Clark Fläche ist, $N_s(u,v)$ die Funktion der zugehörigen Oberflächennormalen, und $D(u,v)$ eine skalarwertige Oberflächenfunktion. Dadurch, dass die Grundfläche C^2 ist, gilt für $N_s(u,v)$ folglich C^1. Wir konstruieren die Verschiebungsfunktion $D(u,v)$ so, dass diese ebenfalls C^1 ist, indem wir skalare bi-quadratische B-Splines verwenden. Folglich ist auch $f(u,v)$ glatt und differenzierbar - d.h. es ist ein kontinuierliches Normalenfeld $N_f(u,v)$ auf $f(u,v)$ definiert. Für Details zur Behandlung außergewöhnlicher Gitterpunkte siehe [NL13].

Oberflächenauswertung Wir werten die Oberfläche inklusive Verschiebungsfunktion anhand der u,v Patchkoordinaten und der zugehörigen Flächenindizes aus. Für die Catmull-Clark Grundfläche $s(u,v)$ und deren Oberflächennormale $N_s(u,v)$ verwenden wir feature-adaptive Unterteilung (siehe Abschnitt 3). Die skalare Verschiebungsfunktion $D(u,v)$ wird ausgewertet indem ein lokales 3×3 Fenster der bi-quadratischen B-Spline Koeffizienten betrachtet wird. Dafür müssen die Patchparameter u,v in das Unterparametergebiet $(\hat{u},\hat{v})$ mittels eine linearen Transformation T gebracht werden:

$$\hat{u} = T(u) = u - \lfloor u \rfloor + \tfrac{1}{2}, \quad \text{and} \quad \hat{v} = T(v) = v - \lfloor v \rfloor + \tfrac{1}{2}.$$

Dadurch kann die skalare Verschiebungsfunktion ausgewertet werden

$$D(u,v) = \sum_{i=0}^{2}\sum_{j=0}^{2} B_i^2(T(u))B_j^2(T(v))d_{i,j},$$

wobei $d_{i,j}$ die ausgewählten Koeffizienten der Verschiebungsfunktion, und $B_i^2(u)$ die quadratischen B-Spline Basisfunktionen sind.

Um die Normale der Verschiebungsfunktion $f(u,v)$, auszuwerten, werden deren partielle Ableitungen benötigt:

$$\frac{\partial}{\partial u} f(u,v) = \frac{\partial}{\partial u} s(u,v) + \frac{\partial}{\partial u} N_s(u,v) D(u,v) + N_s(u,v)\frac{\partial}{\partial u} D(u,v).$$

$\frac{\partial}{\partial v} f(u,v)$ folgt analog. $\frac{\partial}{\partial u} s(u,v)$ ist direkt durch die Definition der Basisfläche gegeben. Um die Ableitungen von $N_s(u,v)$ zu bestimmen, berechnen wir zunächst die Ableitung der un-normalisierten normale $N_s^*(u,v)$ mit der Weingarten Gleichung (E, F, G und e, f, g sind die Koeffizienten der ersten und zweiten Fundamentalform):

$$\frac{\partial}{\partial u} N_s^*(u,v) = \frac{\partial}{\partial u} s(u,v)\frac{fF - eG}{EG - F^2} + \frac{\partial}{\partial v} s(u,v)\frac{eF - fE}{EG - F^2},$$

$\frac{\partial}{\partial v} N_s^*(u,v)$ folgt analog. Durch Nachdifferenzieren bestimmen wir die normalisierten partiellen Ableitungen der Normalen $\frac{\partial}{\partial u} N_s(u,v)$ und $\frac{\partial}{\partial v} N_s(u,v)$, womit sich $N_f(u,v)$ berechnen lässt.

Das Rendern der Oberfläche kann ebenso, wie in Abschnitt 3 beschrieben, mit dem Hardware Tessllierer durchgeführt werden. Oberflächennormalen werden auf Pixelebene ausgewertet, was zu sehr hochwertigen Ergebnissen führt (siehe Abbildung 7).

5 Zusammenfassung

Mit unseren Methoden ist es erstmals möglich Unterteilungsflächen in Echtzeitanwendungen einzusetzen. Letztendlich erreichen dadurch Computerspiele und Modellierungssoftware die Oberflächenqualität die bisher nur aufwändigen Filmproduktionen vorbehalten war. Bereits nach kurzer Zeit der Veröffentlichung unserer Algorithmen, wurden die Forschungsergebnisse in kommerziellen Anwendungen integriert und verwendet, wie es zum Beispiel in Abbildung 8 zu sehen ist. Während unsere Arbeit den Fokus speziell auf die geometrischen Aspekte des Echtzeitrenderings legt, erwarten wir in den nächsten Jahren ähnliche Fortschritte im Bereich der Beleuchtungsberechnung und Physiksimulation. Dadurch können visuell noch ansprechendere und von der Realität kaum noch zu unterscheidende Ergebnisse erzielt werden.

In dieser Kurzfassung haben wir einen kleinen Ausschnitt unserer Algorithmen vorgestellt, die während der Promotion entwickelt wurden [Nie13]. Für eine detaillierte Darstellung verweisen wir auf die entsprechenden Publikationen und die Ausarbeitung der Dissertation. Wir hoffen jedoch, dass wir einen kleinen Einblick in die technischen Grundlagen und den akademischen Beitrag unserer Forschung geben konnten.

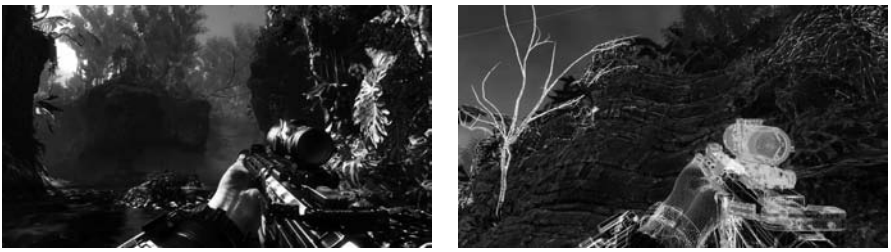


Abbildung 8: Unterteilungsflächen im Bestseller *Call of Duty: Ghosts*, gerendert mit unserem feature-adaptiven Unterteilungsverfahren. Das Drahtgittermodel und zugehörige Unterteilungslevel sind rechts visualisiert. © Activision Blizzard

Literatur

[CC78] Edwin Catmull und James Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-aided design*, 10(6):350–355, 1978.

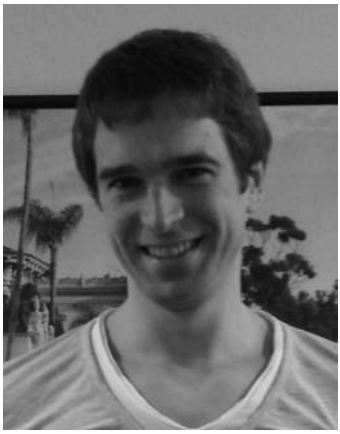
[Nie13] M. Nießner. *Rendering Subdivision Surfaces using Hardware Tessellation*. Dissertation. Dr. Hut, 2013.

[NL12] M. Nießner und C. Loop. Patch-based Occlusion Culling for Hardware Tessellation. In *Computer Graphics International*, Jgg. 2, 2012.

[NL13] M. Nießner und C. Loop. Analytic Displacement Mapping using Hardware Tessellation. *ACM Transactions on Graphics (TOG)*, 32(3):26, 2013.

[NLMD12] M. Nießner, C. Loop, M. Meyer und T. DeRose. Feature-adaptive GPU Rendering of Catmull-Clark Subdivision Surfaces. *ACM Transactions on Graphics (TOG)*, 31(1):6, 2012.

[NSSL13] M. Nießner, C. Siegl, S. Schäfer und C. Loop. Real-time Collision Detection for Dynamic Hardware Tessellated Objects. *Eurographics*, 2013.



Matthias Nießner, geboren 1986, hat an der Universität Erlangen-Nürnberg Informatik studiert und 2009 sein Diplom erhalten (Betreuer: Prof. Dr.-Ing. Marc Stamminger). Wegen exzellenter Studienleistungen (Bestnote), sehr kurzer Studienzeit, und seiner herausragenden Diplomarbeit wurde er mit dem ASQF Förderpreis ausgezeichnet. Anschließend hat er unter der Betreuung von Prof. Dr. Günther Greiner im Bereich der Computergrafik seine Promotion begonnen (ebenfalls Universität Erlangen-Nürnberg). 2013 hat er im Alter von 26 Jahren seine Doktorarbeit abgeben, welche sowohl mit Bestnote als auch mit Auszeichnung bestanden wurde (Externer Gutachter: Prof. Dr. Hans-Peter Seidel). Seit September 2013 ist er Visiting Assistant Professor an der Stanford Universität und arbeitet in der Gruppe von Prof. Pat Hanrahan an aktuellen Forschungsthemen im Bereich der Informatik - *the next big thing*.